Calipso

SECURITY

Jordi Mato Parente

Mary Qjuyenum Igbineweka

2n ASIX

Resumen del proyecto

Este proyecto va a ser una herramienta por vía web, en la que los autores se han decidido concentrar en el programa **Suricata**, que es un **analizador de tráfico** de la red, para así poder analizar de una forma más completa los paquetes de la red, y así de esta forma, se podrá recopilar todos los datos posibles para posteriormente añadir los protocolos los cuales detectarán un índice de paquetes, que sean más elevados de lo habitual. Con toda la información recopilada, luego, serán gestionados mediante un **gráfico** que será actualizado en tiempo real y así el usuario pueda tener un seguimiento de su estado de red, también se añadirá un filtro para así saber si están atacando a la red del usuario. Otra función sería la de detectar si el sistema del usuario ha sido infectado e incluso llegar a geo-localizar el atacante. Todo esto está pensado hacerlo mediante una página web y así simplificar el uso del programa. El programa final se podrá utilizar tanto en **Ubuntu** como en **Fedora**, y se utilizará un servidor de Alibaba que ha sido proporcionado por un profesor de los autores y así ellos podrán utilizar el programa en una situación real.

Palabras clave

- ★ Suricata
- ★ Analizador de tráfico
- ★ Gráfico
- ★ Ubuntu
- ★ Fedora

Abstract

This project will be a web-based tool, in which the authors have decided to focus on the **Suricata** program, which is a network **traffic analyzer**, in order to analyze in a more complete way the network packets, and thus in this way, you can collect all possible data and then add the protocols which will detect an index of packets, which are higher than usual. With all the information collected, then, will be managed through a **graph** that will be updated in real time and so the user can keep track of their network status, also a filter will be added to know if they are attacking the user's network. Another function would be to detect if the user's system has been infected and even geo-locate the attacker. All this is intended to be done through a web page and thus simplify the use of the program. The final program can be used in both **Ubuntu** and **Fedora**, and will use an Alibaba server that has been provided by a teacher of the authors so they can use the program in a real situation.

Keywords

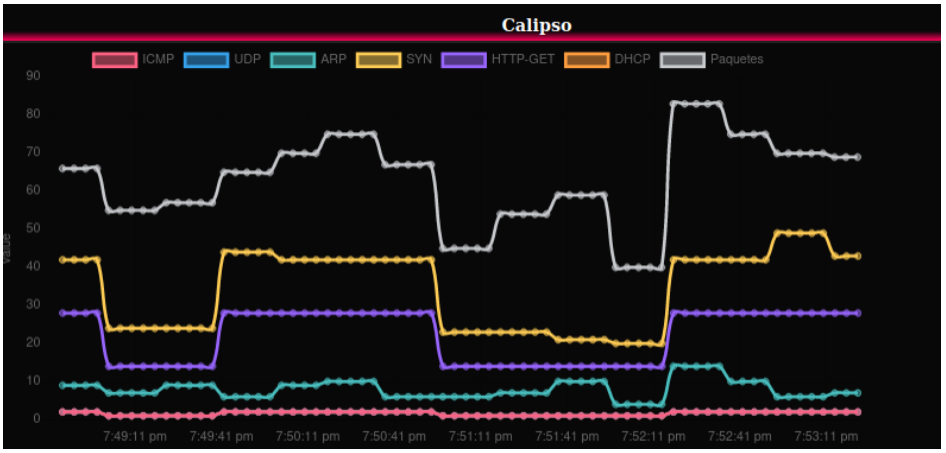
- ★ Suricata
- ★ Trafic analyzer
- ★ Graphic
- ★ Ubuntu
- ★ Fedora

Índice

1. Introducción

1.1 Contexto y justificación

Este proyecto se llama Calipso, la razón por la que los autores de este proyecto se han decidido por este nombre, ha sido por el nombre que tenía la diosa griega del mar (Calypso).
Jordi, que es uno de los creadores del proyecto, hizo en grado medio un proyecto similar a este, pero en ese momento se tenía un conocimiento menor y entonces ahora en grado superior se lo comentó a su otra compañera, Mary y decidieron hacer por así decirlo el hermano mayor del programa que ya había hecho Jordi. Por si hay interes se puede ver la versión que se hizo en grado medio a través de este enlace y una imagen del gráfico creado del programa:
https://github.com/Jmatop/Calipso_Install



Ahora que los dos autores están en grado superior, utilizaran las herramientas que han ido estudiando durante el curso y de esta manera crear herramientas nuevas y personalizadas para la seguridad de la red de los usuarios.

1.2 Objetivos

Objetivo general

El objetivo general de este proyecto es crear una herramienta que garantice un plus de seguridad a las redes de los usuarios, ya que puede detectar ataques de denegación de servicio y bloquearlos. Además de disponer de una herramienta para observar el status de tu servidor.
Objetivos específicos

Los objetivos específicos para este proyecto, es que en las próximas semanas se pueda entender y utilizar la herramienta Suricata la cual digamos que es la base del proyecto puesto que tiene una amplia base de datos de ataques los cuales dicho programa alertará a los usuarios de cuando se produzca alguno de ellos. Una vez entendido todo esto, se querrá poder recopilar la información de la red para así poder implementarla dentro de un gráfico el cual va a estar ubicado en el programa final. También está pensado añadir en el programa gráficos que se encargaran de mostrar la información del status del servidor. Y

cuando se tenga todo esto, se implementara las medidas de protección las cuales básicamente son:

- Si atacan, poder detectarlos y bloquearlos.
- Detectar archivos infectados.
- Geolocalizar a los atacantes

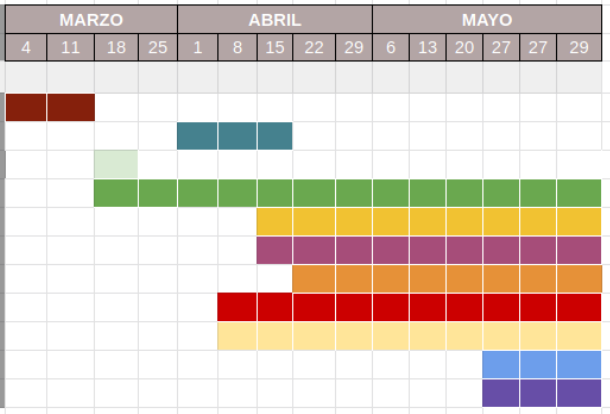
Y poder crear reglas personalizadas desde nuestro navegador web.

1.3 Metodología de trabajo

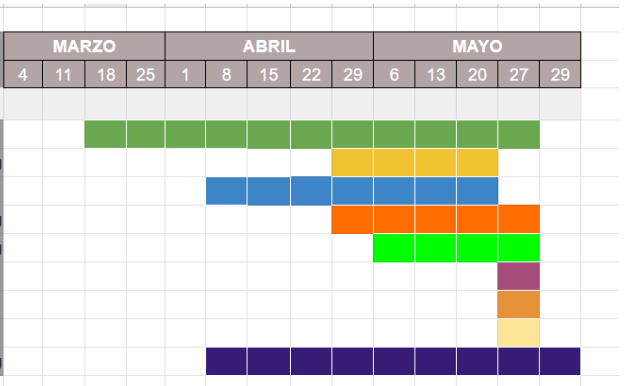
Se ha utilizado el diagrama de Gantt para así planificar mejor, donde ha sido distribuido las tareas entre nosotros, entonces, según que tareas si a uno se le da mejor esa tarea en concreto pues las hacemos por separado y si es una tarea que nos parece complicada pues en conjunto.

Diagrama de Gantt

1r ENTREGA	FINAL: 1/04/2022		
Tarea	Inicio	Final	
Ideas Proyecto	04/03/2022	11/03/2022	M-J
Diseño de la página web	01/04/2022	15/04/2022	M
Diseño Logo	18/03/2022	18/03/2022	J
Recoger datos de suricata	18/03/2022		J
Implementación en gráfico			M-J
Mostrar en web Status del Servidor			J
Detectar ataques y bloquearlos			M
Detectar archivos infectados			M-J
Geolocalizar atacantes			J
Suricata Reglas personalizadas en web			M
Extras			M-J



ENTREGA FINAL	FINAL: 29/05/2022		
Tarea	Inicio	Final	
Recoger datos de suricata	18/03/2022	27/05/2022	M
Crear Instalador	24/04/2022	26/05/2022	M-J
Creación parsers	03/05/2022	23/05/2022	M
Hacer configuracion fluent-bit	29/04/2022	27/05/2022	M-J
Gráfico en tiempo real	01/05/2022	21/05/2022	M-J
Gráficos status del servidor	23/05/2022	25/05/2022	J
Creación reglas suricata	22/05/2022	23/05/2022	M
Geolocalizar atacantes	27/05/2022	27/05/2022	J
Terminar Memoria	08/05/2022	29/05/2022	M-J



2. Tecnologías

Las tecnologías de este proyecto se basan en linux, tiene unos requisitos muy concretos, la más importante es que se utilice la versión 20.04 de Ubuntu la cual es la versión más actualizada en la cual está la herramienta del suricata, si se estuviera un sistema Fedora con la versión más reciente ya estaría bien, es recomendable también que dicha máquina tenga minimo 3-4 GB de ram y 20 GB de almacenamiento y que sea un servidor.

Las tecnologías que han sido utilizadas para desarrollar este proyecto son las siguientes:

2.2 Alibaba Cloud for students

Es una empresa de cloud, los proveedores de servicios de nube son empresas que instalan nubes públicas, gestionan nubes privadas u ofrecen elementos de cloud computing (también llamados servicios de cloud computing). Entonces Alibaba Cloud proporciona servicios de cloud a empresas en línea y al propio ecosistema de comercio electrónico de Alibaba.

La razón por la que será utilizada en el proyecto, es porque estaría bien poder ponerlo en una situación real, es decir probar como funciona nuestro programa sobre un servidor que recibe paquetes de forma constante.

La competencia de este podría ser AWS que viene a ser el Cloud Computing de la empresa Amazon. Pero finalmente nos hemos decidido por el de Alibaba ya que nos dan un servidor de forma gratuita al ser estudiantes, cosa que AWS también lo proporciona, pero AWS cobra según los recursos que sean utilizados, y en cambio no cobra por los recursos, pero eso si, sus recursos son muy limitados.

2.3 Suricata

Es el principal motor independiente de detección de amenazas de código abierto. Al combinar la detección de intrusiones (IDS), la prevención de intrusiones (IPS), la supervisión de la seguridad de la red (NSM) y el procesamiento de PCAP, Suricata puede identificar, detener y evaluar rápidamente incluso los ataques más sofisticados.

La razón por la que se utiliza este programa en el proyecto, es con tal de poder detectar ataques de forma rápida, aplicar reglas personalizadas que veamos

que sean importantes, para así llegar a proteger de una forma más amplia el sistema del usuario contra ataques, y por último llegar a bloquear el ataque.

2.4 Tshark

Es un analizador de tráfico de red de línea de comando que permite capturar datos de paquetes desde una red activa o paquetes de lectura de un archivo de capturas guardado anteriormente. Tshark viene a ser lo mismo que Wireshark, simplemente que Wireshark es por interfaz gráfica y Tshark en terminal.
Aparte de que tenemos Suricata, para estar visualizando la red, también utilizaremos Tshark, ya que nos facilita el poder controlar el número de paquetes de cada protocolo y así tenerlo más fácil a la hora de realizar el gráfico en tiempo real con la función de mostrar los paquetes de cada protocolo.
Una competencia de este, sería TCPDump que es un comando muy conocido y utilizado, pero nos encontramos con el problema de que no podíamos escoger los protocolos de los paquetes de la forma que queríamos.

2.5 Elasticsearch

Es un motor de analítica y análisis distribuido, que se encarga de almacenar en el disco los datos recibidos. Su especialidad es la de realizar búsquedas y también el poder hacer análisis de estos datos. Gracias a Elasticsearch podremos almacenar todos los datos que hayan sido recogidos del sistema del usuario, para luego gestionar los.

La competencia de este sería Opensearch, que vienen a ser un fork de este programa, pero lo diferente de ellos dos es que Opensearch creó el fork a partir de solo la parte gratuita de Elastic. Es decir con Elastic a lo mejor si deseamos descargar algún plugin, puede llegar a valer dinero, en cambio con Opensearch todo es libre.
Aun así,se escogió elastic, porque es más fácil su instalación, ya que viene todo pautado en su documentación y hay más información de ella en internet, porque con Opensearch se llegaba a tener ya problemas de instalación que no se podía a entender y por problemas de su falta de documentación, lo dificulta todo más.

2.6 Kibana

Este programa proporciona capacidades de visualización de datos y de búsqueda para los datos indexados en Elasticsearch.Kibana también actúa

como la interfaz de usuario para monitorear, gestionar y asegurar un cluster del Elasticsearch. Con Kibana estamos haciendo que la gestión de los datos que han sido recogidos por ElasticSearch, sea más fácil y mejor para visualizar.

La competencia de ella, es OpenSearch Dashboards, que viene a ser el programa que viene en conjunto con el OpenSearch que se ha explicado en el apartado anterior. Y como se ha dicho anteriormente dieron problemas de instalación así que se decidió optar por Elastic.

2.7 Fluentbit

Es un programa que se encarga de recoger las líneas que se vayan añadiendo al final de los ficheros o logs que se hayan indicado o también se puede ir recogiendo las líneas directamente del journal. Ya con los datos, se encargará de enviar los datos hacia un servidor HTTP, a servicios como Elasticsearch, o también se pueden llegar a mostrar por el terminal.

Se escogió este programa porque, como necesitábamos un programa que se encargue de enviar los datos hacia ElasticSearch y fue utilizado en la asignatura de Seguridad, pues al tener ya un conocimiento de ella, facilita el trabajo.

3.Desarrollo Proyecto

3.2 Creación Logo

La creación del logo fue bastante compleja, puesto que se buscaba un diseño el cual se identificara con el proyecto, de esta manera se decidió buscar en la página web de Placeit ya que en esa página puedes crear los logos cómo al usuario le guste, una vez se seleccionó el logo se tuvo que hacer una captura de pantalla del mismo puesto que se deja con marca de agua, posteriormente se llevó la foto a krita para de esta manera poder editar pixel a pixel y así eliminar la marca de agua.

3.3 Creación servicio

Dado que el programa tiene que estar con un escaneo constante del tráfico de la red, y así tener un control del número de paquetes que va recibiendo el usuario y verificar que no están siendo más elevados de lo normal, utilizaremos a Tshark para realizar este conteo.
En un principio se tenía pensado utilizar TCPDump, pero nos encontramos con el problema de que el tcpdump al estar escaneando se nos hacía difícil poder reconocer los protocolos, cosa que con el Tshark se ve de una forma más clara.

Con la creación del servicio podremos hacer que cuando se encienda la máquina, Systemd sea el encargado de poner en marcha Tshark y mantenerlo encendido desde su activación.
Para ello se tuvo que crear un fichero en el directorio /etc/systemd/system con el nombre del servicio, que sería tshark.service.
El contenido del servicio vendria ser el siguiente:

```
  GNU nano 5.8                                          tshark.service
[Unit]
Description=Tshark
[Service]
Type=simple
ExecStart= tshark -Y "icmp||arp||tcp||udp"
[Install]
WantedBy=multi-user.target
```

El servicio creado es sencillo:

- La primera sección que vemos es la de [Unit] y es utilizada para definir datos sobre la propia unit. En este caso solo se ha puesto una simple descripción.
- En la segunda sección ya tenemos [Service], como es una unit de tipo servicio pues se hace uso de esta sección para poner las funciones que tendrá el servicio. Dentro de esta sección hemos puesto lo siguiente:
    - Type → Con esto decimos la manera de arrancar que tendrá el servicio, hay varios tipos (Simple,forking,oneshot), y el que más se ajustaba para este servicio era simple. Que viene a ser que el servicio se queda en primer plano de forma indefinida y luego Systemd será el encargado de ponerlo en segundo plano, de esta forma nos estamos asegurando de que tshark se quedara en segundo plano ejecutándose.
    - ExecStart → Aquí se dice el comando que se ejecuta al iniciar nuestro servicio. Entonces se iniciará el comando Tshark, el parámetro -Y es para indicar filtros, en nuestro caso estamos filtrando por los protocolos: icmp,arp,tcp,udp. De esta forma hacemos que sea menos pesado al filtrar solo lo necesario.

- La última sección que es [Install], es utilizada para decir cómo y cuándo podrá ser activada o desactivada la unit.

3.3.1 Poniendo en marcha el servicio

Una vez creado el servicio, se puso en marcha el servicio para ver si daba algún fallo y ver si todo estaba correcto, para iniciarlo se utiliza el siguiente

comando, se pone tshark porque es el nombre que le hemos dado al crear su fichero .service.

```
[root@fedora system]# systemctl start tshark
```

Ya habiendo hecho el comando anterior, se tuvo que verificar su estado, es decir si se ha llegado a activar el servicio de forma correcta. Como se puede ver en la captura de abajo, el servicio acabó fallando.

```
[root@fedora system]# systemctl status tshark.service
× tshark.service - Tshark
     Loaded: loaded (/etc/systemd/system/tshark.service; enabled; vendor preset: disabled)
     Active: failed (Result: exit-code) since Sun 2022-05-22 18:04:28 CEST; 2min 25s ago
    Process: 2125 ExecStart=tshark -Y icmp||arp||tcp||udp (code=exited, status=203/EXEC)
   Main PID: 2125 (code=exited, status=203/EXEC)
        CPU: 1ms

may 22 18:04:28 fedora systemd[1]: Started Tshark.
may 22 18:04:28 fedora systemd[2125]: tshark.service: Failed to execute /usr/bin/tshark: Permission denied
may 22 18:04:28 fedora systemd[2125]: tshark.service: Failed at step EXEC spawning tshark: Permission denied
may 22 18:04:28 fedora systemd[1]: tshark.service: Main process exited, code=exited, status=203/EXEC
may 22 18:04:28 fedora systemd[1]: tshark.service: Failed with result 'exit-code'.
[root@fedora system]#
```

La razón por la que está fallando el servicio es por SELinux. Que en resumen es una arquitectura de seguridad que define los controles de acceso para las aplicaciones.
Debido a esto, nos encontramos con el problema de que Tshark no está definido en las reglas de SELinux, cosa que hace que Tshark no se pueda iniciar.
Para solucionar este problema hay dos opciones:

- Se crea una política personalizada para Tshark y así SELinux permita su acceso al sistema.
- Directamente deshabilitar SELinux en su fichero de configuración.

Se intentó la primera solución que es la de crear una política de acceso para Tshark, pero debido al poco tiempo que había y que no es sencillo de crear, se decidió hacer la segunda solución.

Así que para ello se añade la siguiente línea en el fichero de servicio:

```
ExecStartPre = setenforce 0
```

Con ExecStartPre decimos que antes de que se ejecute el comando que se encuentra en la línea ExecStart se ejecute este. Y el comando que se ejecutara es setenforce 0, que es el comando que se utiliza para desactivar de forma temporal SELinux, es decir si se apagara la máquina volvería a estar activo.

Ya hecho el cambio anterior en el archivo y vuelto a encender el servicio, ya se ve que el servicio ahora si se ha iniciado bien.

```
[root@fedora system]# systemctl status tshark.service
● tshark.service - Tshark
     Loaded: loaded (/etc/systemd/system/tshark.service; enabled; vendor preset: disabled)
     Active: active (running) since Sun 2022-05-22 18:23:50 CEST; 2s ago
    Process: 2262 ExecStartPre=setenforce 0 (code=exited, status=0/SUCCESS)
   Main PID: 2263 (tshark)
      Tasks: 6 (limit: 3819)
     Memory: 88.9M
        CPU: 321ms
     CGroup: /system.slice/tshark.service
             ├─2263 tshark -Y icmp||arp||tcp||udp
             ├─2278 /usr/bin/mmdbresolve -f /usr/share/GeoIP/GeoLite2-Country.mmdb -f /usr/share/GeoIP/GeoLite2-City.mm>
             └─2283 /usr/bin/dumpcap -n -i enp0s3 -Z none

may 22 18:23:50 fedora systemd[1]: Starting Tshark...
may 22 18:23:50 fedora systemd[1]: Started Tshark.
may 22 18:23:50 fedora tshark[2263]: Running as user "root" and group "root". This could be dangerous.
may 22 18:23:50 fedora tshark[2263]: Capturing on 'enp0s3'
```

3.4 Integrando Fluent Bit con Elasticsearch

En este apartado ya empezamos con las configuraciones que tendrá el archivo de configuración de Fluent Bit, que es el encargado de enviar los datos que recoge a ElasticSearch, entonces en este apartado se irá explicando apartado por apartado lo que contiene y lo que se trata de enviar. Si se quiere ver por completo los ficheros utilizados de Fluent Bit, ir al apartado 7.2 de Anexos.

3.4.1 Primera prueba Tshark

Como se ha estado diciendo, se quiere el poder crear unos gráficos en tiempo real sobre el estado de la red.
La primera prueba que se llegó a hacer, fue con el programa tcpdump, pero como se ha dicho anteriormente, hubo problemas con el ya que no se podía recoger los protocolos de una forma correcta, así que fue cuando se decidió cambiar a Tshark.
En este, fue la primera vez que se empezó a mandar datos desde Fluent bit, a Elastic. Este vendría a ser la versión beta del fichero de configuración del fluent bit.

```
[SERVICE]
 Flush 5
 Daemon off
 Log_Level debug
 Parsers_File /usr/local/etc/fluent-bit/parsers.conf

[INPUT]
 Name systemd
 Tag arp
 Systemd_Filter _SYSTEMD_UNIT=tshark.service

[FILTER]
 Name parser
 Match *
 Key_Name *
 Parser json

[OUTPUT]
 Name es
 Match *
 Host 192.168.1.47
 Port 9200
 Index prueba
 HTTP_User elastic
 HTTP_Passwd W5qMUYZbcStH*WkPa=aN
 Suppress_Type_Name On
 Tls on
 Tls.verify off
```

Las partes más importantes de este fichero son INPUT, FILTER y OUTPUT.

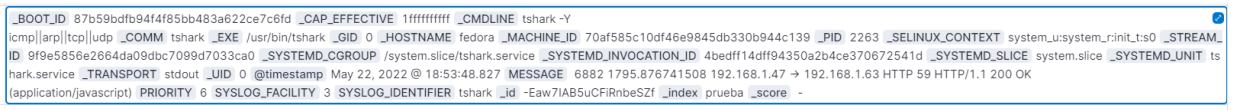
En Input indicamos los ficheros o logs que el fluent bit tendrá que ir leyendo y enviando. Como se ve en el fichero, estamos diciendo que leerá systemd, que viene a ser el journal y le aplicamos un filtro para que solo lea los que vienen del servicio creado anteriormente de tshark.

Luego en Filter, indicamos opciones que pueden llegar a ser parsers o incluso utilizar el grep para buscar información en el fichero o log indicado en la parte de Input. Entonces en este fichero se está utilizando el filtro parser, para parsear la información del journal a formato json, ya que lo que le tiene que llegar a Elastic debe estar en formato json.

Y por último tenemos el apartado Output, que viene a ser a donde queremos enviar los datos del Input. En este caso es hacia elastic y se debe indicar la ip del servidor, y luego el usuario y la contraseña que se utilizara.

Ya enviado los datos mediante el fluent bit, una forma rápida que hay para ver los datos que han sido enviados a ElasticSearch desde Kibana es el apartado de Discover, que sirve para buscar,filtrar, y obtener información sobre la estructura de los campos.

Dicho esto, al ver el Discover, se vio que la estructura de los campos era la siguiente, esta es una captura de uno de los registros que han sido enviados:



Vemos varios campos, pero el más interesante es este, el campo Message y esto es porque este es el campo que contiene lo que ha llegado a recoger Tshark, que es las Ip de Origen y destino, el protocolo y luego más información.

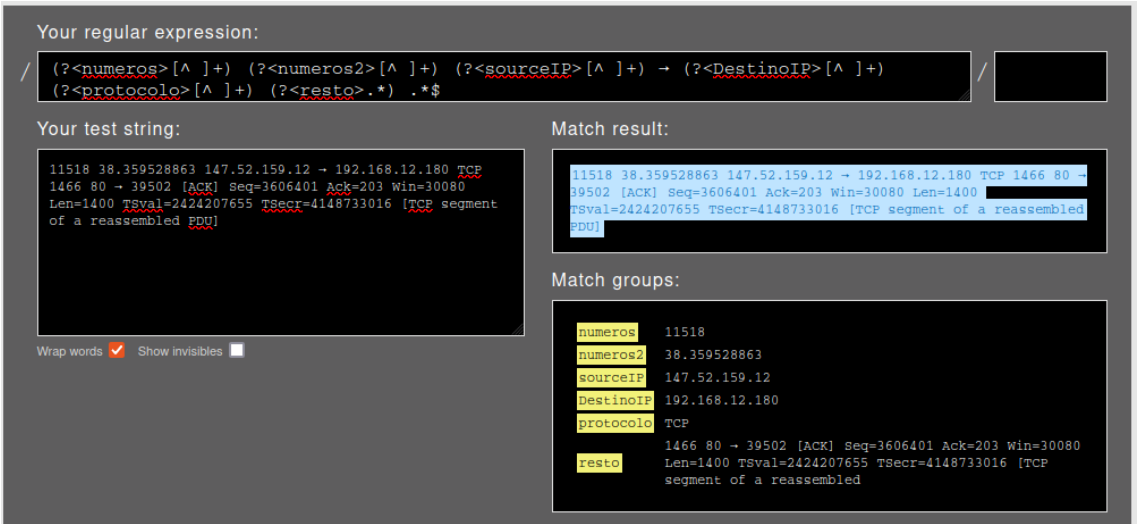
```
    "MESSAGE.keyword": [
      " 6882 1795.876741508 192.168.1.47 → 192.168.1.63 HTTP 59 HTTP/1.1 200
OK  (application/javascript)"
    ],
```

Al ver esto, los autores del proyecto se dieron cuenta de que los valores de este campo no llegaban a estar separados como se esperaba. Así que la siguiente tarea viene a ser crear un parser personalizado para así al enviar la información a elastic ya venga separada por los campos deseados.

3.4.2 Creacion Parser personalizado

Como se ha dicho anteriormente, ahora se debía crear un parser personalizado y de esta forma dividir los campos por los valores deseados y así a la hora crear los gráficos sea fácil el poner los datos deseados.

Para la creación de ellos, se utilizó como ayuda la página https://rubular.com/, que es una página en la que podemos meter el texto que queramos dividir por campos e ir probando para separar los campos de la forma deseada.
Como se ha dicho, Tshark está sacando solo los protocolos que sean ICMP,TCP,UDP y ARP. Y como por cada protocolo Tshark saca una información diferente, habría que crear un parser por cada de los protocolos, empezando por el protocolo TCP el parser se vería de la siguiente manera desde la página Rubular:



Como se puede ver en la imagen nos muestra al final como estarían agrupados los campos.

Entonces, después de haber creado el parser anterior, pues se volvió a probar de enviar a elasticsearch los datos, donde ahora añadimos un nuevo fichero que contenía el parser personalizado y se modifica el fichero anterior con tal de aplicar el parser nuevo.

Las únicas modificaciones que se han realizado en el fichero anterior fueron las siguientes:

En el apartado de [SERVICE], se cambió la ruta de los parsers, por la del nuevo fichero que va contener todos los parsers creados.

```
Parsers_File /home/usuario/Calipso/custom-parsers.conf
```

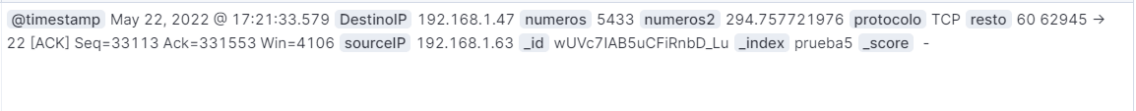
Luego en el apartado de [INPUT], pues se añadió la opción Tag, que sirve para nombrar los datos que le vayan llegando y así nombrar cada input por su protocolo, y así para luego aplicar el parser, por cada uno de sus protocolos.

```
[INPUT]
 Name systemd
 Tag proto_tcp
 Systemd_Filter _SYSTEMD_UNIT=tshark.service
 Read_From_Tail on
```

Y por último la sección [FILTER], donde se ha cambiado el nombre del parser, por el nombre que se le ha dado en el fichero de los parsers, luego en Match, se pone el nombre del tag que se ha puesto en la sección anterior, de esta manera se dice que solo se aplique al parser indicado, por último el key_name, que nos sirve para decir en qué campo se debería poner el parser, que en este caso es MESSAGE, que como se ha dicho anteriormente, es el que contiene la información importante.

```
[FILTER]
 Name Parser
 Parser protocolo_tcp
 Match proto_tcp
 Key_Name MESSAGE
```

Una vez enviados los datos con los nuevos cambios, al volver a mirar el apartado de discover, se ve como ahora si que se encuentra la información necesaria distribuida por campos y así poder crear luego el gráfico en Kibana.



3.4.3 Suricata

Creación Reglas con Suricata

Como se ha dicho, se va a utilizar el programa Suricata con tal de poder detectar ataques que se están realizando hacia la máquina servidor. Así que para ello se tuvo que crear reglas personalizadas, aunque es verdad que el propio Suricata ya te da reglas para utilizar, se ha preferido crear propias, ya que así es más fácil de saber las reglas que hay.
Las reglas creadas solo han sido dos, la primera regla sirve para detectar ataques DOS de tipo TCP, es decir va a servir para detectar cuando una máquina intenta realizar un ataque de inundación de paquetes TCP hacia el servidor, para intentar dejar al servidor inaccesible.
La otra regla sirve para detectar el mismo tipo de ataque DOS, pero ahora mediante el protocolo ICMP.

Un ejemplo de creación de regla en suricata sería la siguiente:
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"DOS SYN packet flood inbound"; flow:to_server; flags: S; threshold: type both, track by_dst, count 100, seconds 1; classtype:misc-activity; sid:5000000;)

alert → viene a ser la acción que debe hacer suricata, si el paquete que evalúa suricata sus características coinciden con la regla indicada, entonces al poner como acción alert, lo que hará suricata sería generar una alerta de la regla.

tcp → Protocolo que debería tener el paquete.

$EXTERNAL_NET → Que viene a ser la dirección de Origen del paquete, y esto es una variable que se encuentra en el fichero de configuración de Suricata y nos sirve para decir cualquier IP que no sea la propia IP de la máquina.

any → Puerto de origen, en este caso puede ser cualquier puerto de origen y el otro any es lo mismo, pero puerto de destino.

$HOME_NET → La IP de destino, este también es una variable que se encuentra en el fichero de configuración de Suricata y con este se puede decir la red a la que pertenece la máquina o simplemente poner la IP de la máquina, en este caso es solo la IP de la máquina.

msg → Mensaje que se mostrará si se cumple la alerta.

threshold → Con este se está haciendo que se limiten las alertas que se generan en el log y que solo que se generan a una cantidad, es decir se generará una alerta, si pasado 1 segundo ya se cuentan 100 paquetes que coinciden con la regla.

Añadir Suricata al fichero FluentBit

Ya creadas las alertas del Suricata, era hora de ya añadir en el fichero de Fluent Bit a suricata y así enviar las alertas que se vayan generando hacia Elastic.

Lo que se añadió fue lo siguiente:

```
[INPUT]
 Name tail
 Tag suricata
 Path /var/log/suricata/fast.log
```

Fluent Bit se encargará de ir mirando las nuevas líneas que aparezcan en el fichero fast.log, que es un fichero de Suricata que sirve para ver alertas que se hayan generado en el servidor.

Para suricata también hizo falta crear un parser que se encargará de cortar los campos de las líneas que muestra suricata. Si se quiere ver este parser, se puede ir al apartado de Anexos 7.2.

Lo siguiente que también se añadió fue otra sección [INPUT] igual a la anterior pero simplemente se cambia el nombre de su Tag, para poder luego utilizar el filtro GeoIP, que viene a ser un filtro que tiene Fluent Bit, para localizar el país del que proviene una dirección IP.

Y se escribió el filtro de la siguiente manera:

```
[FILTER]
 Name geoip2
 Match suricatamap
 Database /usr/share/GeoIP/GeoLite2-Country.mmdb
 Lookup_key DireccionOrigen
 Record country DireccionOrigen %{country.names.en}
 Record isocode DireccionOrigen %{country.iso_code}
```

Para utilizar este filtro, hace falta tener una base de datos de países, que viene a ser simplemente a utilizar un fichero, que es la que utilizará luego fluent bit para localizar las direcciones IP, luego también hay que decirle a fluent bit, cuál es el campo que contiene la dirección IP a mirar.

3.4.4 Monitorización estado sistema

Disco

Para poder monitorizar el espacio libre del propio disco dentro del fluent-bit se tuvo que añadir el comando **df --output=pcent / | tail -n 1,** utilizando en la sección [INPUT] el exec, que es una directiva que nos sirve para ejecutar comandos.
Debido a que fluent bit por defecto envía los datos en tipo string, y que el porcentaje de disco utilizado es necesario que sea de tipo integer, fue necesario el crear un parser y de esta manera hacer que los datos se cambiaran a integer al ser enviados a elastic. Dicho parser sería el siguiente, donde se indica que el tipo del campo que contiene el porcentaje de disco sea integer.

```
[PARSER]
 Name usadodisco
 Format regex
 Regex (?<porcentajeDisco>\d{1,2}).*$
 Types porcentajeDisco:integer
```

CPU

Para poder lograr una monitorización del sistema se utilizó un plugin de fluent-bit el cual nos muestra los usos de la CPU, dicho plugin es el siguiente:

```
[INPUT]
 Name cpu
 Tag  my_cpu
```

Al añadir esa configuración al propio fluent-bit lo que hará es sacarnos algo cómo esto:

```
[2022/05/27 14:20:54] [ info] [sp] stream processor started
[0] my_cpu: [1653661255.129288488, {"cpu_p"=>0.000000, "user_p"=>0.000000, "system_p"=>0.000000, "cpu0.p_cpu"=>1.000000, "cpu0.p_user"=>1.000000, "cpu
0.p_system"=>0.000000, "cpu1.p_cpu"=>0.000000, "cpu1.p_user"=>0.000000, "cpu1.p_system"=>0.000000}]
```

Como se ve en la imagen, saca varios valores de la cpu cómo del propio sistema, pero simplemente se ha de sacar el valor del primer input el cual es la de **cpu_p** dado a que este valor es la propia suma de todos los valores de la cpu y además nos lo mostrará en porcentaje. De está manera se podrá crear un gráfico posteriormente con el uso de la cpu.

Puertos

Para poder envíar los datos de los puertos que están escuchando en el servidor, se utilizó otra vez la directiva exec, para así poder poner el siguiente comando, con el que se verían los puertos de tipo tcp, que estén escuchando, el nombre de sus procesos y solo los que sean de ipv4.

- **ss -tlnp4 | tail -n +2 | awk '{print $4" "$5" "$6}'**

Pero se tuvo que utilizar un parser para limpiar los datos que no necesitábamos, dicho parser contiene la siguiente información:

```
[PARSER]
 Name listenports
 Format regex
 Regex (?<LocalAddress>[^ ]+):(?<puerto>\d{2,})(?<PeerAddress>.*):\*(?<proceso>.*).*$
```

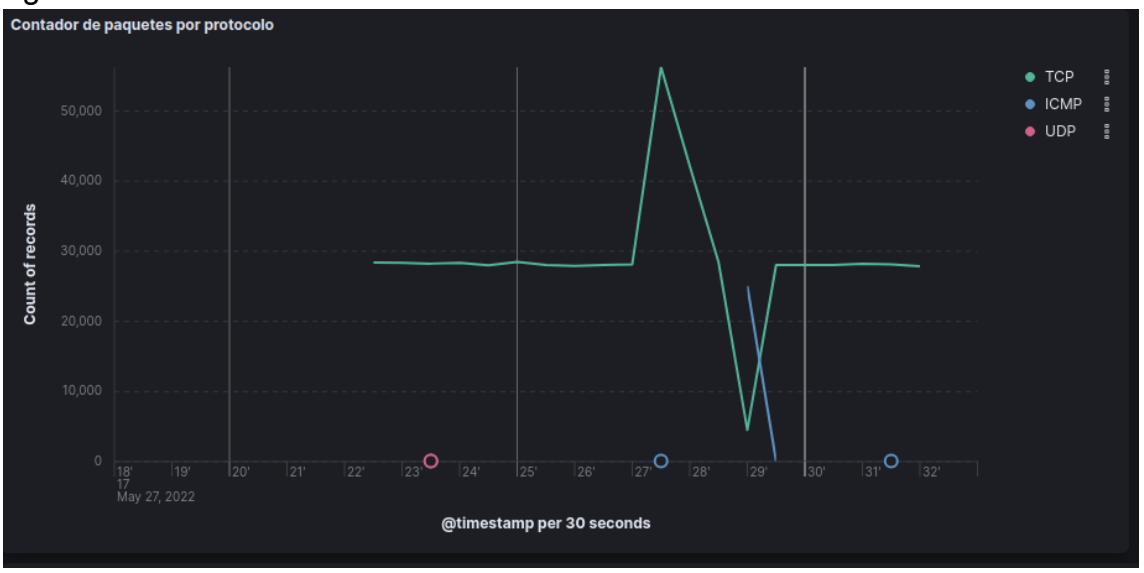
3.5 Gráficos

Se ha decidido implementar una serie de gráficos los cuales van a ir indicando una serie de valores los cuales van a ser útiles a la hora de monitorizar todo lo que esté pasando en nuestro servidor. El primer gráfico que se implementó es un gráfico lineal, cuyo cual vaya indicando todo el tráfico que pasa por nuestra red, dicho tráfico incluye los protocolos:

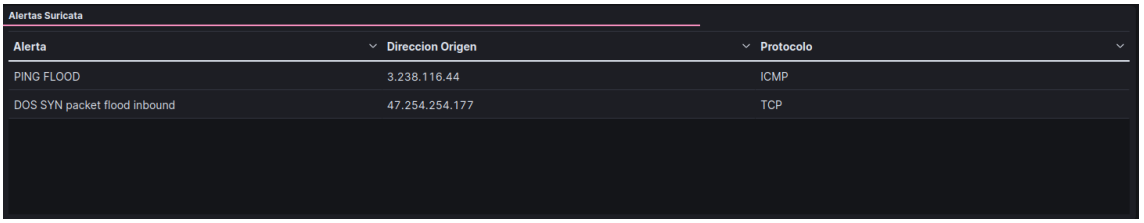
- TCP
- ICMP
- UDP
- ARP

De esta manera plasmará un gráfico con todo el tráfico respecto a los protocolos indicados anteriormente para así poder llevar un control más adecuado de la red.

Para la creación de este gráfico en Kibana se utilizó, el servicio Tshark que ha sido comentado en el apartado anterior 3.4.1 Tshark y el resultado es el siguiente:



El segundo gráfico que se ha realizado, fue el gráfico de tipo tabla y que sirve para ver las alertas de suricata y de esta manera si alguien está atacando al servidor, se verá en la tabla que está siendo atacado. La información a mostrar es el tipo de ataque, la dirección IP del que ha realizado el ataque y su protocolo. Este es el resultado del gráfico, y para crear este gráfico se ha utilizado los datos recibidos del apartado: 3.4.3 Suricata.



El tercer gráfico, es un gráfico que va a servir para ver los puertos que están en modo escucha en el servidor, de esta manera se puede tener un control de los puertos que están escuchando y los procesos que están siendo ejecutados en

esos. El gráfico es de tipo tabla y su resultado es el siguiente, los datos de aquí provienen del apartado 3.4.4 Puertos.

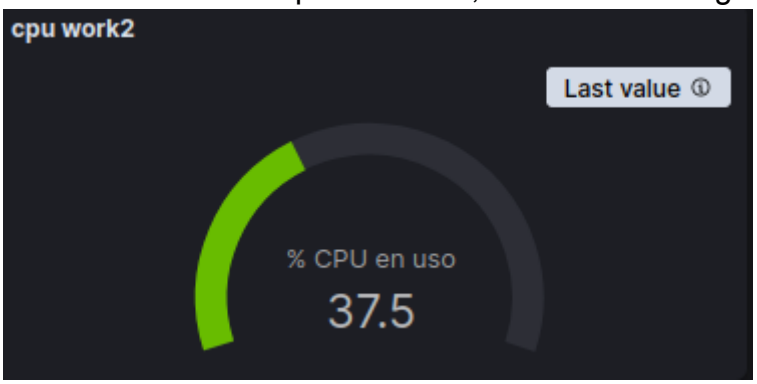
Puertos en escucha servidor

Direccion Local	Puertos	Direccion Remota	Proceso
0.0.0.0	5355	0.0.0.0	users:(("systemd-resolve",pid=595,fd=11))
0.0.0.0	5601	0.0.0.0	users:(("node",pid=785,fd=52))
127.0.0.53%lo	53	0.0.0.0	users:(("systemd-resolve",pid=595,fd=17))
127.0.0.54	53	0.0.0.0	users:(("systemd-resolve",pid=595,fd=19))

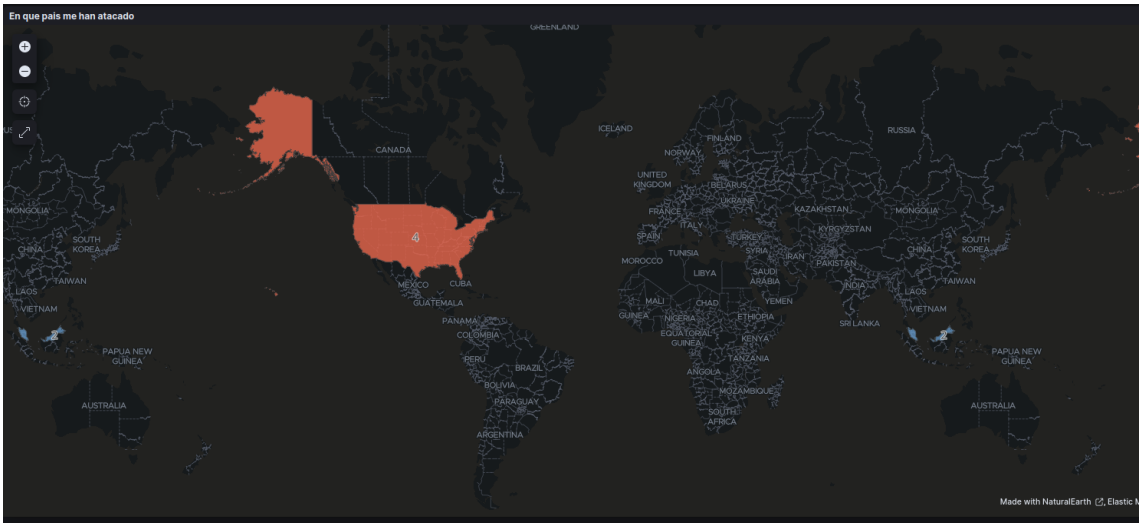
Los siguientes gráficos que se han hecho, son para ver parte del hardware de la máquina, que se considera importante tener un control sobre ellos y saber más o menos su estado, que son el disco y la CPU.

A la hora de añadir el gráfico del disco se recoge la información indicada anteriormente en el punto 3.4.4 Disco. Es un gráfico de tipo Gauge y el resultado es el siguiente, se va mostrando en porcentaje lo que ha llegado a ser utilizado del disco.

Para implementar el gráfico de la cpu, se ha comentado su configuración en el punto 3.4.4 CPU. Es un gráfico de tipo gauge el cual se le indica que recoja la información enviada por fluent bit, el resultado del gráfico es el siguiente.



El último gráfico que se había creado, fue el del tipo mapa, en el que con este tenemos la opción de que si, suricata detecta de que se está realizando un ataque en el servidor, se cogerá su dirección de origen, se localiza de dónde proviene y se pintara en el mapa en el país del que venga. Y esta información la recibe del apartado 3.4.3 de suricata.



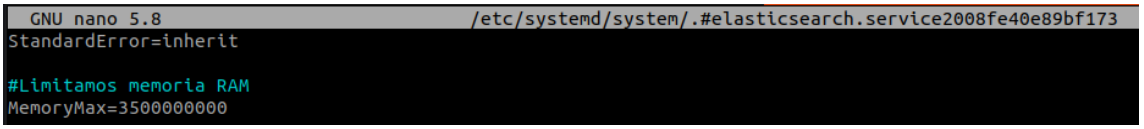
3.6 Poner programa en situación real

Pues ya terminado de desarrollar el proyecto, se quiso el poder probar de poner el proyecto en una situación real, que vendría ser utilizar un servidor utilizando los servicios de las empresas de cloud computing.
En un principio se tenía pensado utilizar los servicios de Alibaba como se dijo en el apartado de tecnologías, pero después de que se hicieron pruebas de la instalación del ElasticSearch y Kibana en los servidores que proporciona Alibaba. Se vio que ElasticSearch no llegaba a funcionar, hasta que se acabó viendo que la RAM que tenía el servidor era de 1GB y la necesaria para que llegue a funcionar el programa era 3-4 GB.

Asique la solución fue cambiar a AWS, ya que allí sí podemos elegir la RAM que debe tener la máquina, aunque es verdad que aquí cobran según los recursos que utilices, pero como esto ya se hizo a pocos días de que se presentará el proyecto y aún se tenía el suficiente dinero como para el dia de la presentación se pudo utilizar el AWS sin problemas.

Ya sabido esto, en el AWS se creó una instancia de tipo t3.large que viene a ser 2 núcleos de CPU y 8 GB de RAM, con un sistema de Fedora 34 Cloud Base.

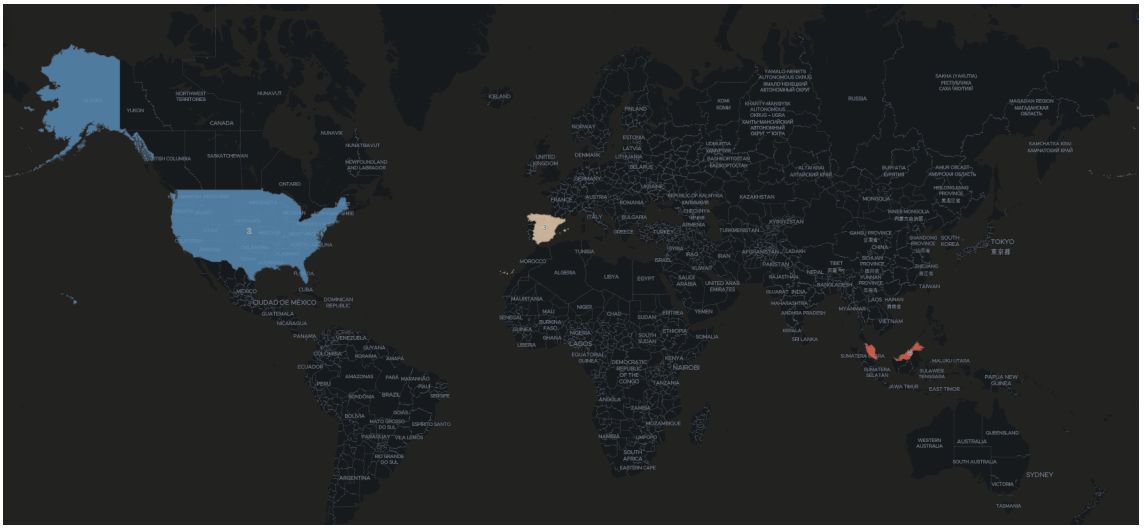
Después, de ya realizar las instalaciones de todos los programas necesarios, se puso en marcha el Fluent Bit para ya empezar a mandar datos al ElasticSearch. Se vio que en un principio funcionaba, pero al cabo del tiempo el servidor se quedaba congelado y empezaba a ir lento, entonces se miró con el programa Top si había algún proceso que estaba consumiendo demasiada memoria del servidor que le hiciera ir lento. Con ello se vio al proceso Elastic como estaba consumiendo demasiada memoria RAM que hiciera que el sistema acabará colapsando debido a que no quedaba suficiente espacio libre para los demás procesos. Así que se decidió editar el servicio de ElasticSearch para así ponerle un límite de memoria RAM y que no consumiera más de la necesaria. En la siguiente captura se ve la línea que se tuvo que añadir para llegar a conseguirlo.



Ya aplicados los cambios, se volvió a probar y verificar que ahora si el proceso de ElasticSearch ya no se supera el límite de memoria RAM que se le puso.

Una vez que se consiguió solucionar este problema, apareció otro, cuando se estuvo inspeccionando con el programa top, también se vio que el servicio Tshark, cada vez empezaba a consumir más memoria RAM, debido a que este almacena los paquetes que va capturando a través de la red.
para ello se implementó una solución, la cual constaba en añadir unas ciertas limitaciones al propio servicio de Tshark, las cuales era añadir el parámetro **-m** para así limpiar el exceso de paquetes cada x cantidad y además añadir dentro del servicio una nueva línea llamada **MaxHight=100M** el cual lo que hace es regular la memoria al valor añadido, si es superado el propio valor lo que hace es ralentizar el propio tshark para así intentar bajar el consumo.

Aun hecho lo anterior, a los pocos minutos la máquina volvió a fallar, así que se tomó la decisión de crear una nueva máquina, con un sistema operativo distinto que fue Fedora 36 y cambiando el tipo de instancia a t2.large. Se volvió a instalar el programa sin hacer ninguna modificación como en la anterior máquina y esta sí resultó funcionar bien.
Este es un resultado final del programa siendo ejecutado en la máquina creada con AWS.



3.7 Instalador

Teniendo en cuenta que esto va a ser un conjunto de programas que luego se van a integrar todos en uno solo para llegar a lograr el objetivo final, se tuvo la decisión de crear un script de bash que se encargaría de completar la instalación de estas y así también querer que la instalación de los programas sea lo más fácil posible para el usuario, el instalador va mostrando paso por paso en el punto de la instalación que se encuentra.
A continuación se dirán los programas que están en el instalador, pero si se quiere tener el instalador y una explicación más detallada de cómo se debe utilizar se puede ir al apartado 7.1 de annexos

3.7.1 Contenido del instalador

El instalador consta de los siguientes programas y configuraciones:

- Suricata
- Java
- ElasticSearch
- Kibana
- Tshark
- Creación de servicio Tshark
- FluentBit

- Creación de servicio Fluent Bit

4.Conclusiones

4.1 Conclusiones generales del proyecto

En este proyecto se ha podido aprender más a fondo el uso correcto de las herramientas opensearch, fluent-bit, suricata, tshark y kibana. Se han tenido varias complicaciones para llevar este proyecto a su propia finalidad dado a que al principio no se configuró correctamente el servidor en el que se va a trabajar y esto provocó que no funcionara correctamente el propio programa pero con perseverancia se logró que funcionará de nuevo correctamente, Calipso actualmente creemos que es una herramienta bastante adecuada para llevar una mejor visión de nuestro propio servidor. Pero sobre todo a pesar de los problemas tenidos a la hora de crear el proyecto Calipso se han aprendido muchas cosas interesantes cómo funciones que no se conocían tanto sobre los propios servicios o simplemente cómo crear un gráfico el cual pueda identificar las direcciones ip y geolocalizarlas. Por desgracia se quería añadir además de todo lo que está hecho se quiso implementar también el propio gráfico en una propia página web con su propio login, se querían añadir más funciones las cuales te permiten bloquear los propios atacantes entre otros, pero debido al poco tiempo que se tuvo para realizar el proyecto se decidieron descartar dado a que se prefería entregar algo que funcionará correctamente y sea interesante de ver, en vez de entregar algo que esté aún por acabar.

4.2 Vision del futuro

Lo que se espera de este proyecto en un futuro es básicamente que se solucionen los problemas que han habido además de añadir los gráficos en una propia página web la cual tenga su propio menú de login además de añadir más pestañas para poder añadir más contenido para la propia seguridad de la red, también se espera poder añadir más protocolos de red para el propio tráfico. A parte se espera que se pueda añadir un método el cual te permita bloquear las conexiones de los atacantes de la propia red, de esta manera se convertiría en una herramienta de seguridad bastante notable para ciertas empresas.

5. Glossario

IDS: Un IDS es un programa de detección de accesos no autorizados a un computador o a una red. El IDS suele tener sensores virtuales con los que el núcleo del IDS puede obtener datos externos.

IPS: Un sistema de prevención de intrusos es un software que ejerce el control de acceso en una red informática para proteger a los sistemas computacionales de ataques y abusos

Systemd: Systemd es un conjunto de demonios o daemons de administración de sistema, bibliotecas y herramientas diseñados como una plataforma de administración y configuración central para interactuar con el núcleo del Sistema operativo GNU/Linux.

NSM: NSM es un proceso automatizado que supervisa los dispositivos de red y el tráfico en busca de vulnerabilidades de seguridad, amenazas y actividades sospechosas.

SELinux: SELinux es un módulo de seguridad para el kernel Linux, implementado utilizando el framework del núcleo Linux Security Modules, que proporciona el mecanismo para implementar una políticas de control de acceso de tipo Control de acceso obligatorio y control de acceso basado en roles

Journal: se encarga de recopilar y manejar todos los mensajes producidos por el kernel, initrd, servicios, etc...

Formato JSON : JSON es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje

Parser: Parser es un programa informático que analiza una cadena de símbolos según las reglas de una gramática formal.

Top: Comando el cual se utiliza para imprimir por pantalla el consumo del propio equipo al igual que los procesos que están utilizando recursos del sistema.

6. Bibliografia

https://www.elastic.co/guide/en/elasticsearch/reference/current/deb.html
http://pevma.blogspot.com/2014/08/suricata-idsips-http-custom-header.html
https://rubular.com/
https://www.elarraydejota.com/caso-practico-de-stack-elk-metricbeat/
https://support.logz.io/hc/en-us/articles/210207225-How-can-I-export-import-Dashboards-Searches-and-Visualizations-from-my-own-Kibana-
https://www.freedesktop.org/software/systemd/man/systemd.resource-control.html
https://github.com/P3TERX/GeoLite.mmdb
https://docs.fluentbit.io/manual/

7.Annexos

7.1 Instalador paso a paso

Primero de todo, se encuentra el instalador y todos los archivos necesarios en el siguiente repositorio de Git: https://github.com/Jmatop/Calipso.

Una vez que ha sido descargado el repositorio, en la carpeta se encontrará varios archivos, el importante ahora es el script InstallFedora.sh, habrá que darle permismos de ejecucion chmod +x InstallFedora.sh y despues se ejecuta el script ./InstallFedora.sh.
Ya hechas las instalaciones de suricata y Java, se empezará con las instalaciones y sus configuraciones de ElasticSearch y Kibana.

Entonces primero se empieza con la creación de un repositorio de Elasticsearch.
En la imagen siguiente se puede ver como se ha realizado esta parte para el script.

```
echo -e "\e[1;32m*************************************************************"
echo -e "*                                                           *"
echo -e "*          Creación de Repositorio ElasticSearch            *"
echo -e "*                                                           *"
echo -e "*************************************************************\e[0m"
sudo rpm --import https://artifacts.elastic.co/GPG-KEY-elasticsearch
echo "[elasticsearch]" | sudo tee -a /etc/yum.repos.d/elasticsearch.repo
echo "name=Elasticsearch repository for 8.x packages" | sudo tee -a /etc/yum.repos.d/elasticsearch.repo
echo "baseurl=https://artifacts.elastic.co/packages/8.x/yum" | sudo tee -a /etc/yum.repos.d/elasticsearch.repo
echo "gpgcheck=1" | sudo tee -a /etc/yum.repos.d/elasticsearch.repo
echo "gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch"| sudo tee -a /etc/yum.repos.d/elasticsearch.repo
echo "enabled=0" | sudo tee -a /etc/yum.repos.d/elasticsearch.repo
echo "autorefresh=1" | sudo tee -a /etc/yum.repos.d/elasticsearch.repo
echo "type=rpm-md" | sudo tee -a /etc/yum.repos.d/elasticsearch.repo
```

Básicamente primero ha sido importada la clave de firma pública de elasticsearch. Y luego ya se ha podido ir creando el fichero para el repositorio de ello.

Entonces ya creado el fichero del repositorio ya se puede proceder a hacer el comando dnf install elasticsearch. Pero como los autores del proyecto quieren que la instalación sea lo más fácil y cómoda posible para el usuario, tuvieron que ir guardando la salida de la instalación en un fichero aparte, todo esto es porque Elasticsearch cuando realiza su instalación imprime por pantalla la contraseña que tendrá el usuario “elastic”, que viene a ser el usuario admin.

Siendo así, con tal de asegurarse que después de la instalación el usuario tenga la contraseña en un sitio visible, se fue guardando en un fichero aparte las credenciales que serían importantes luego.
En la siguiente imagen se puede ver como ha sido realizado.

```
echo -e "\e[1;32m*********************************************"
echo -e "*                                           *"
echo -e "*          Instalando ElasticSearch         *"
echo -e "*                                           *"
echo -e "*********************************************\e[0m"
#Instalamos elasticsearch, pero vamos guardando su salida en el fichero prueba.txt
sudo dnf -y install --enablerepo=elasticsearch elasticsearch > prueba.txt
#Reiniciamos el systemd y habilitamos e iniciamos el servicio de elasticsearch
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable elasticsearch.service
sudo systemctl start elasticsearch.service
#Guardamos en una variable la contraseña, haciendo una busqueda en el fichero prueba.txt que es el que contiene la salida de la instalación
pass=$(cat prueba.txt | grep "generated password for the elastic built-in superuser " | awk '{print $11}')
#Lo guardamos en el fichero que va a contener todas las credenciales necesarias para el usuario
echo "password-elastic:" $pass | sudo tee -a config-elastic.txt
#Y eliminamos el fichero que contenia la salida de la instalación
rm prueba.txt
```

También una vez finalizada la instalación se hace una comprobación de que Elasticsearch esté funcionando correctamente.

```
sleep 20
sudo curl -s --cacert /etc/elasticsearch/certs/http_ca.crt -k -u elastic:$pass https://localhost:9200 | grep "You Know, for Search"  > /dev/null
funciona=$?
if (( $funciona == 0 )); then
        echo -e "\e[1;32m ElasticSearch funciona correctamente !!!\e[0m"
else
        echo "No funciona"
fi
```

Es una comprobación simple, donde según la respuesta del comando curl, sabremos si nos está respondiendo Elasticsearch.

Ahora en la instalación de Kibana se hizo prácticamente lo mismo, en la creación del repositorio es igual que la de Elasticsearch y luego la instalación de ella solo realizamos el comando dnf install kibana, ya que de aquí no había ninguna información a guardar.

```
echo -e "\e[1;32m*****************************************************"
echo -e "*                                                   *"
echo -e "*            Creación de Repositorio Kibana         *"
echo -e "*                                                   *"
echo -e "*****************************************************\e[0m"
echo "[kibana-8.x]" | sudo tee -a /etc/yum.repos.d/kibana.repo
echo "name=Kibana repository for 8.x packages" | sudo tee -a /etc/yum.repos.d/kibana.repo
echo "baseurl=https://artifacts.elastic.co/packages/8.x/yum" | sudo tee -a /etc/yum.repos.d/kibana.repo
echo "gpgcheck=1" | sudo tee -a /etc/yum.repos.d/kibana.repo
echo "gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch"| sudo tee -a /etc/yum.repos.d/kibana.repo
echo "enabled=1" | sudo tee -a /etc/yum.repos.d/kibana.repo
echo "autorefresh=1" | sudo tee -a /etc/yum.repos.d/kibana.repo
echo "type=rpm-md" | sudo tee -a /etc/yum.repos.d/kibana.repo

echo -e "\e[1;32m*************************************"
echo -e "*                                   *"
echo -e "*           Instalando Kibana       *"
echo -e "*                                   *"
echo -e "*************************************\e[0m"
sudo dnf -y install kibana
```

El siguiente paso de la instalación es ya empezar con la configuración de kibana, en la imagen se puede ver como ha sido realizado con algunos comentarios para entender mejor lo que se está haciendo.

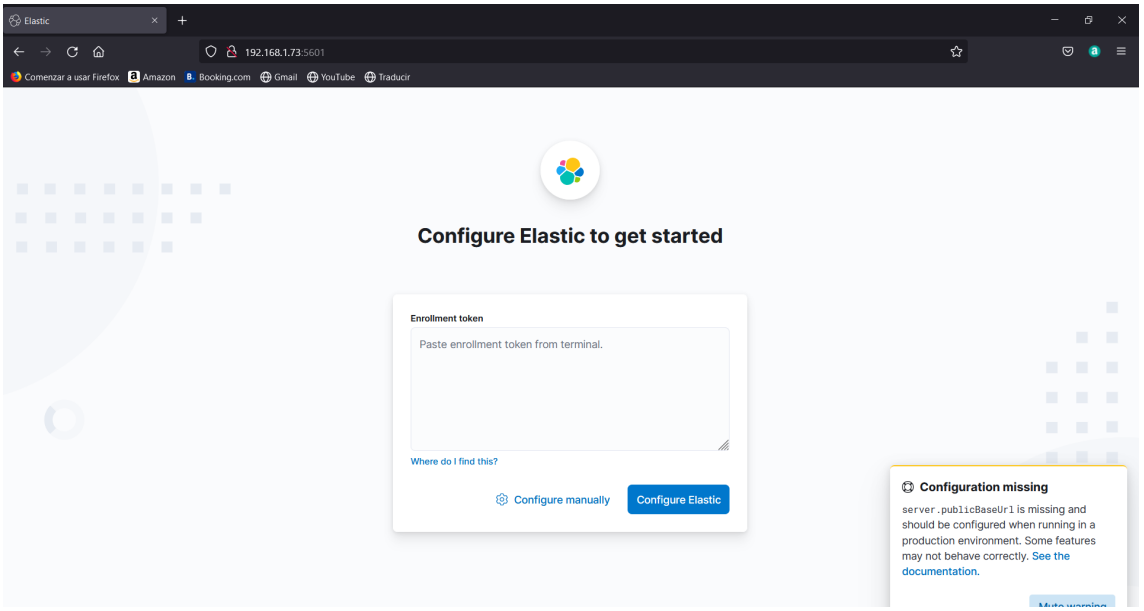
```
echo -e "\e[1;32m*****************************************"
echo -e "*                                       *"
echo -e "*          Configuración Kibana         *"
echo -e "*                                       *"
echo -e "*****************************************\e[0m"
#Remplazamos con el comando sed, la linea server.host del fichero de configuracion de kibana, y lo cambiamos a 0.0.0.0 para que tambien se pueda acceder
#de forma remota en la web a kibana.
sudo sed -i 's/#server.host: "localhost"/server.host: "0.0.0.0"/' /etc/kibana/kibana.yml
#Tenemos la opcion de que Elastic luego se encargue de configurar por si mismo kibana, para ello tenemos que crear un token para kibana con el comando:
#elasticsearch-create-enrollment-token, tambien guardamos la salida de este comando en la variable token
token=$(sudo /usr/share/elasticsearch/bin/elasticsearch-create-enrollment-token -s kibana)
#Ya guardada la variable, lo guardamos en el fichero que es el que contiene lo necesario para luego finalizar la instalación.
echo "Token-kibana:" $token | sudo tee -a config-elastic.txt
#Se reinicia el systemd, e iniciamos y habilitamos kibana.
sudo /bin/systemctl daemon-reload
sudo systemctl start kibana.service
sudo /bin/systemctl enable kibana.service
```

Pero aún no se ha terminado la configuración, como se ha dicho en los comentarios anteriores tenemos la opción de que elastic se encargue de configurar por sí mismo kibana.
Entonces una vez iniciado kibana, él se encargará de generar un código que se necesitará indicar al acceder por primera vez a la web. Este código es enviado al journal así que se encargaron de buscar el código a través del journal y guardarlo en una variable, por último este código es guardado en el fichero que tiene la información.

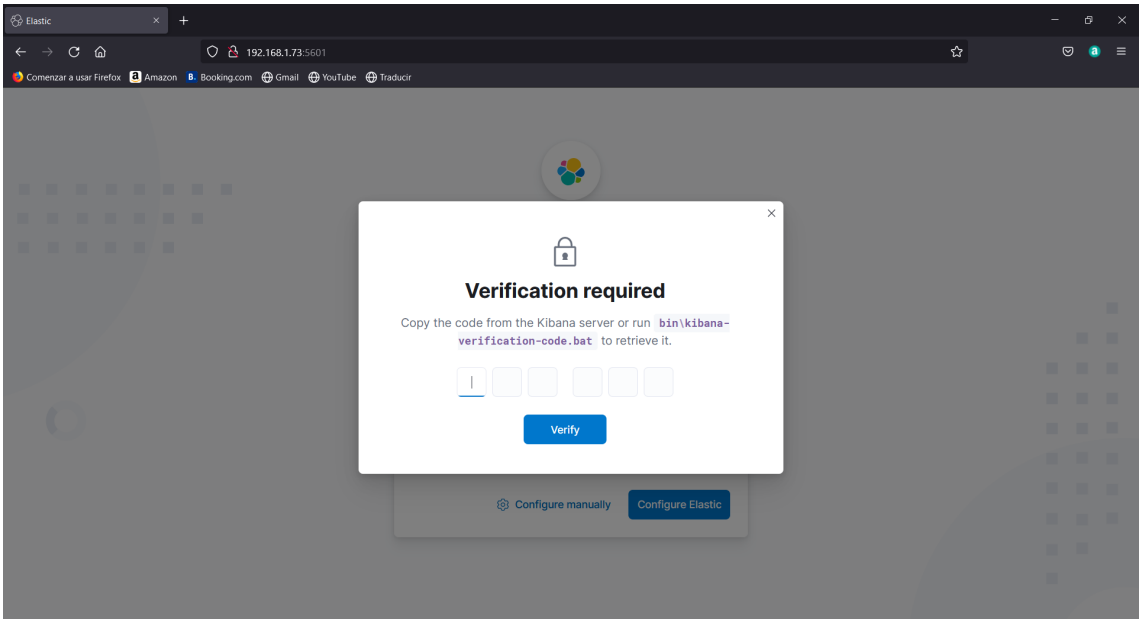
```
echo -e "\e[1;32mEsperando Codigo....\e[0m"
sleep 50
codigo=$(journalctl -u kibana.service | grep "code=" | awk '{print $8}')
echo "Code Verification:" $codigo | sudo tee -a config-elastic.txt
```

Y por último luego se harán las instalaciones de Tshark y fluent bit, y también sus respectivos servicios.

Teniendo todo esto, así se vería al acceder por primera vez a kibana en el navegador web, para acceder simplemente ponemos la IP del servidor y luego :5601 que es el puerto por el que se accede a Kibana.



Donde alli solo tendríamos que ir al fichero creado config-elastic.txt y pegar el token que encontraremos en el fichero y luego nos pedira el codigo, tambien guardado en ese fichero:



Ya instalado y configurado ElasticSearch y Kibana, encontraremos varios archivos en la carpeta del repositorio que ha sido descargado.
El primero a utilizar es el fichero test.rules, que es el fichero de suricata que contiene las reglas personalizadas que han sido creadas, asique debemos mover el fichero test.rules al directorio que se encuentran las reglas de suricata, de la siguiente manera:

```
[root@ip-172-31-85-36 Calipso]# mv test.rules /var/lib/suricata/rules
```

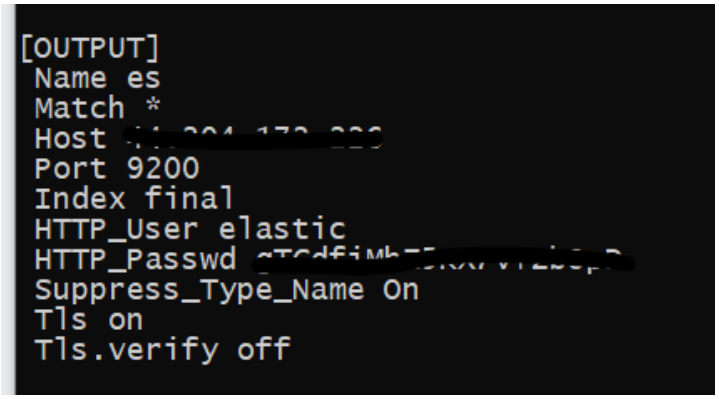
Ya movido, lo siguiente será acceder al fichero de configuración de Suricata, que está ubicado en /etc/suricata/suricata.yaml. Vamos a la sección de rule-files para añadir el fichero test.rules y eliminamos la línea de suricata.rules.

```
 GNU nano 5.8                                    /etc/suricata/suricata.yaml
    #
    hashmode: hash5tuplesorted

##
## Configure Suricata to load Suricata-Update managed rules.
##

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

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

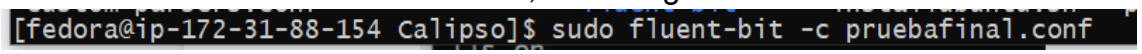
Luego también para que funcione el filtro de GeoIP que nos sirve para localizar a los atacantes, hay que descargarse la base de datos en el siguiente enlace https://github.com/P3TERX/GeoLite.mmdb, y nos descargamos el archivo con nombre, GeoLite2-Country.mmdb.
Seguidamente debemos mover este archivo al directorio /usr/share/GeoIP.

Por ultimo, tambien vamos al fichero de fluent bit que se encuentra en la carpeta, pruebafinal.conf y editamos lo siguiente:

La directiva Host, por la IP de nuestro servidor, y la directiva HTTP_Passwd por la contraseña que nos haya generado elastic, que se encuentra en el fichero config-elastic.txt.



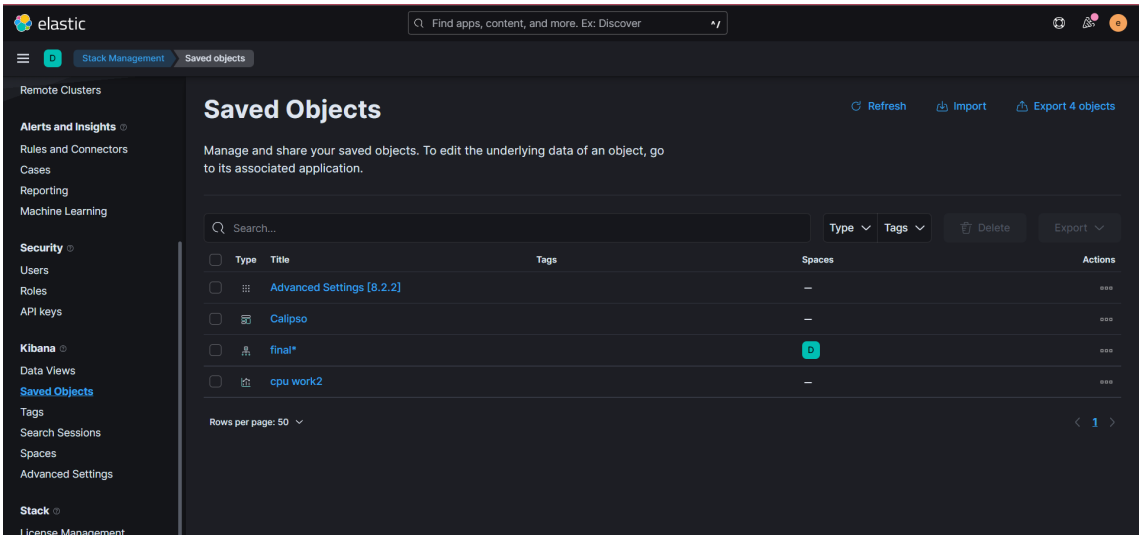
Si se ha hecho todo lo anterior, ya se podrá empezar a enviar los datos mediante fluent-bit los datos a Elastic, con el siguiente comando:



Al hacer el comando vemos que se queda en primer plano ejecutándose, a lo mejor esto no es muy practico, asi que si deseamos que este en segundo plano habría que poner el contenido del fichero pruebafinal.conf en el siguiente archivo:

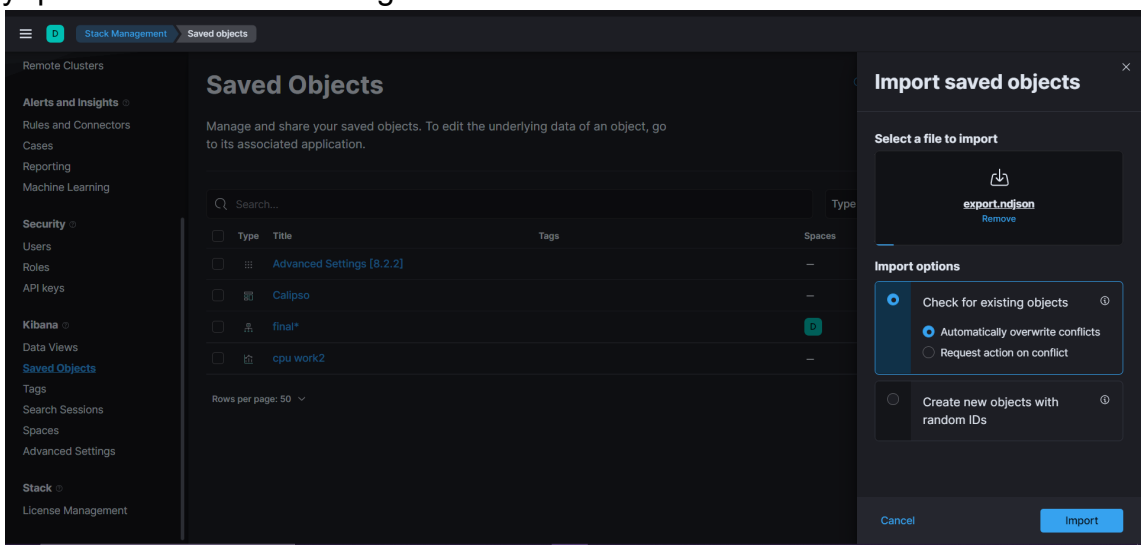
Se debería primero crear una copia del contenido de ese archivo y modificar ese mismo para cambiar el contenido por el de pruebafinal.conf, de esta manera conseguimos que sea Systemd el que se encargue de ir ya ejecutando por sí mismo el comando fluentbit -c …, esto es gracias a que antes se creó el servicio de fluent-bit.

A parte de esto, ahora si queremos tener los graficos, despues de estar enviando los datos, deberíamos acceder a kibana y hacer lo siguiente:
En el menú de kibana vamos al apartado de Stack Management, ya allí vamos al apartado de Kibana y le damos a saved objects.



Ya aquí le damos al botón Import y lo que debemos subir es el fichero export(1).ndjson, este se encuentra en el repositorio anterior, y este fichero

viene a ser un fichero que contiene como han sido creados todos los gráficos y que serán insertados luego de forma automática en el Dashboard.



7.1.3 Instalador para las pruebas de Ataques

Para poder probar que Calipso funcione correctamente se ha creado un script el cual nos permita atacar a dicha máquina y así probar si realmente recoge bien los paquetes o no, dicho script contiene un menú el cual nos indica la dirección ip del servidor de Calipso para así que sea más cómodo ejecutar un ataque dado a que siempre hemos de escribir la dirección ip.

```
echo " ------------------------------------------------------";
echo "|                 (⊃◕‿◕)⊃  Menu                        |";
echo "|------------------------------------------------------|";
echo "|                        OPCIONES                      |";
echo "|------------------------------------------------------|";
echo "| 30 - Salir                                           |";
echo "|------------------------------------------------------|";
echo "|                    54.165.72.80                      |";
echo "|------------------------------------------------------|";
echo "|------------------------------------------------------|";
echo "| 20 - TCP Attack                                      |";
echo "| 19 - UDP Attack                                      |";
echo "| 18 - ICMP Attack                                     |";
echo "| 17 - ARP Attack                                      |";
echo "|------------------------------------------------------|";
```

Cómo hemos visto en el menú, tenemos varias opciones las cuales podemos usar para probar nuestro servidor cómo puede ser el ataque TCP, UDP, ICMP y ARP. Dichos ataques se pueden ejecutar sin ningún tipo de problema hacia el servidor de Calipso excepto el ataque ARP dado a que dicho ataque no es posible ejecutarlo en redes completamente distintas cómo puede ser un servidor de España como uno de Singapur, el ataque ARP se debería utilizar dentro de la propia red del servidor para su uso correcto. Con este menú simplemente tendríamos que escribir la casilla que nosotros queramos utilizar para que así continúe el script. Para las opciones

del TCP, UDP y ARP se utiliza el programa nping el cual nos será útil para dichos ataques. Excepto el ataque ICMP el cual solo se utilizará el típico Ping de la Muerte.

Las opciones que contiene vendrían siendo las siguientes:

```
17)
read -p "Dime la ip de la Víctima: " ip
nping --arp --rate 3000 -c 100000 $ip
;;
18)
read -p "Dime la ip de la Víctima: " ip
sudo ping -f $ip
;;
19)
read -p "Dime la ip de la Víctima: " ip
nping --udp --flags syn --rate 3000 -c 100000 $ip
;;
20)
read -p "Dime la ip de la Víctima: "  ip
nping --tcp --flags syn --rate 3000 -c 100000 $ip
;;
```

Con esto enviamos alrededor de unos 100.000 paquetes hacia nuestro objetivo, menos el icmp que lo paramos cuando nosotros queramos. Este script se puede encontrar en Github . Dentro del propio menú podemos ver una opción 0 la cual instalará todos los paquetes necesarios para todos los ataques que contiene el propio menú .

```
echo "|  0 - Instalar dependencias                              |";
```

7.2 Contenido ficheros Fluent Bit.

Como el fichero de configuración de Fluen Bit, que es donde se han puesto todos los datos a enviar es muy largo, es mejor acceder a él través del siguiente enlace:
https://github.com/Jmatop/Calipso/blob/main/pruebafinal.conf

El otro archivo es el que contiene los parsers que han sido creados y utilizados, también se puede acceder a él través del siguiente enlace:
https://github.com/Jmatop/Calipso/blob/main/custom-parsers.conf

```
[PARSER]
 Name protocolo_arp
 Format regex
 Regex (?<numeros>[^ ]+) (?<numeros2>[^ ]+) (?<detalles>.*) → (?<detalles2>[^ ]+)    (?<protocolo>[^ ]+) (?<resto>.*).*$

[PARSER]
 Name protocolo_icmp
 Format regex
 Regex (?<numeros>[^ ]+) (?<numeros2>[^ ]+) (?<SourceIP>[^ ]+) → (?<DestinoIP>[^ ]+) (?<protocolo>[^ ]+) (?<resto>.*).*$

[PARSER]
 Name protocolo_tcp
 Format regex
 Regex (?<numeros>[^ ]+) (?<numeros2>[^ ]+) (?<sourceIP>[^ ]+) → (?<DestinoIP>[^ ]+) (?<protocolo>[^ ]+) (?<resto>.*) .*$

[PARSER]
 Name protocolo_udp
 Format regex
 Regex (?<numeros>[^ ]+) (?<numeros2>[^ ]+) (?<IPorigen>[^ ]+) → (?<IPdestino>[^ ]+) (?<protocolo>[^ ]+) (?<resto>.*).*$

[PARSER]
 Name rules_suricta
 Format regex
 Regex (?<Fecha>[^ ]*)  \[\**\]\ (?<nosirve>[^ ]+) (?<alerta>.*) \[\**\]\ (?<class>.*) {(?<protocolo>[^ ]*)} (?<DireccionOrigen>[^ ]*):.*$

[PARSER]
 Name listenports
 Format regex
 Regex (?<LocalAddress>[^ ]+):(?<puerto>\d{2,})(?<PeerAddress>.*):\*(?<proceso>.*).*$

[PARSER]
 Name usadodisco
 Format regex
 Regex (?<porcentajeDisco>\d{1,2}).*$
 Types porcentajeDisco:integer
```