[原创]堆入门攻略-how2heap学习总结

0x2l 4

2020-5-1 21:55 11884

how2heap是一个开源的堆漏洞系列教程,这里简单的总结一下.后续会把一些漏洞详细的利用过程写成博客.

原地址:https://github.com/shellphish/how2heap

0x01 实验环境

- Ubuntu 16.04
- Glibc 2.23

0x02 first_fit

glibc使用一种first-fit算法来选择空闲的chunk.如果分配时存在一个大小满足要求的空闲chunk的话,glibc就会选择这个chunk.

- 步骤
    - a = malloc(512) = 0x2490010
    - b = malloc(256) = 0x2490220
    - free(a)

- c = malloc(500) = 0x2490010

0x02 fastbin_dup

fastbins可以看作是一个栈,使用一个单链表实现,free的时候会对free-list进行检查.所以我们不能free同一个chunk两次.但是可以再两次free之间增加一次对其他chunk的free,从而绕过检查顺利执行,然后malloc三次,就得到了两个指向同一区域的指针.

- 步骤:
  - a = malloc(8) = 0x7a8010
  - b = malloc(8) = 0x7a8030
  - c = malloc(8) = 0x7a8050
  - free(a)
  - free(b)
  - free(a)
  - d = malloc(8) = 0x7a8010
  - e = malloc(8) = 0x7a8030
  - f = malloc(8) = 0x7a8010

0x03 fastbin_dup_into_stack

对于fastbins,我们可以通过double free覆盖fastbins的结构,来获得一个指向任意地址的指针.如果可以对这个指针读或者写的话,我们就有了任意地址读或者任意地址写.

- 步骤:
  - unsigned long long stack_var = 0x20
  - a = malloc(8);b = malloc(8);c = malloc(8)

- free(a);free(b);free(a)
- d = malloc(8);malloc(8)
- *d = (unsigned long long) (((char*)&stack_var) - 8)
- malloc(8)
- malloc(8) == ((char*)&stack_var) + 8

0x04 fastbin_dup_consolidate

fastbins除了在两次free之中加入另外一个free之外,还可以借助large bin中malloc_consolidate来绕过检查达到double free的目的.

- 步骤:
  - a = malloc(0x40) = 0x1b69010
  - b = malloc(0x40) = 0x1b69060
  - free(a)
  - c = malloc(0x400)//申请large bin的时候已经执行了malloc_consolidate
  - free(a)//并不会报错,这个时候a已经被放到了unsorted bin之中
  - malloc(0x40) = 0x1b69010;malloc(0x40) = 0x1b69010

0x05 unsafe_unlink

对全局指针ptr进行内存布局,然后借助unlink操作实现任意地址读/写.

- 步骤:
  - (P->fd->bk != P || P->bk->fd != P) = False
    - fd=ptr-size*3
    - bk=ptr-size*2

- (chunksize(P) != prev_size (next_chunk(P)) == False
  - 修改对应的prev_inuse和prev_size
- unlink执行之后
  - ptr = ptr - size*3
  - ptr[3]可以修改ptr指向的内容

0x06 house_of_spirit

覆盖一个堆指针，使其指向可控的区域，构造好相关数据，释放堆指针时系统会将该区域作为chunk放到fastbin里，再申请这块区域，就可以改写目标区域(一般为返回地址或函数指针,不可控).

```
+------------------+
|     可控区域1     |
+------------------+
| 目标区域（不可控，  |
| 多为返回地址或函数  |
| 指针等）          |
+------------------+
|     可控区域2     |
+------------------+
```

- 利用条件:
  - 想要控制的目标区域前面一段和后面一段都是可控区域.
  - 存在可覆盖的堆指针

- 步骤:
  -
  - 伪造堆块，在可控区域1和2中构造数据，将目标区域伪造成一个fast chunk.
  - 覆盖堆指针指向伪造的fast chunk.
  - 释放伪造的fast chunk到fastbin单链表中.
    申请刚刚释放的chunk，使得可以向目标区域中写入数据.

0x07 poison_null_byte

- 步骤
  - 分配三个稍大的chunk
    -
    - a = (uint8_t*) malloc(0x100)
    - b = (uint8_t*) malloc(0x200)
      c = (uint8_t*) malloc(0x100)
  - 在free(b)之前伪造一个假的下个chunk的prev_size，即*(((uint64_t)*c) - 2 -2) = b.size & (~0xff)，以此绕过再次分配的chunksize(P) != prev_size (next_chunk(P))
    -
    - *(size_t)*(b+0x1f0) = 0x200
      free(b)
  - 触发漏洞
    -
      a[0x108] = 0
  - 分配两个较小的chunk

        - b1 = malloc(0x100)
        - b2 = malloc(0x80)
    - 依次free，因为c的prev_size没有更新，造成合并
        - free(b1)
        - free(c)
    - 再次分配大小覆盖b2的chunk，可对b2任意写
        - d = malloc(0x300)
        - d == b

0x08 house_of_lore

house of lore用来构造一个small bin链,可以实现分配任意指定位置的chunk,从而修改任意地址的内存.

- 利用条件:
    - 需要控制small bin chunk的bk指针
    - 控制指定位置的chunk的fd指针
- 步骤:
    - 修改small bin中chunk的bk指针指向fake chunk
    - 令fake chunk的fd指向small bin中的chunk//绕过检查
    - fake chunk不能是small bin的最后一个chunk//绕过检查
    - free(small bin chunk)

- malloc(size);malloc(size)//最后一次返回的是我们构造的fake chunk

0x09 overlapping_chunks

简单的堆重叠,通过修改size来吞并邻块,然后在下次malloc的时候把邻块也给一起分配出来

- 步骤:
  - p1 = malloc(0x100 - 8);p2 = malloc(0x100 - 8);p3 = malloc(0x100 - 8)
  - free(p2);p2->unsorted bin
  - 通过溢出修改chunk p2的size
  - p4 = malloc(0x180)//p2和p3被一起分配出来了
  - 可以对p4进行操作,顺便写了p3;也可以通过修改p3来修改p4的内容

0x10 overlapping_chunks2

还是堆重叠,但是是在free之前修改size值,使free错误地修改下一个chunk地prev_size值,导致中间的chunk被合并.

- 步骤:
  - p1 = malloc(1000);p2 = malloc(1000);p3 = malloc(1000);p4 = malloc(1000);p5 = malloc(1000)
  - free(p4)//因为p5的存在所以p4在free之后不会被合并入top chunk
  - p2.size = p2.size + p3.size + prev_inuse//修改p2的size大小
  - free(p2)
  - p6 = malloc(2000);p6 == p2

0x11 house_of_force

- 利用条件:

    - 能够以溢出等方式修改top chunk的size域
    - 自由控制堆分配的大小

- 步骤:

    - malloc(100)//随便分配一个chunk
    - 使用溢出修改top chunk的size为一个大数//不会去调用mmap
    - malloc(size)//size=目标地址减去 top chunk 地址，再减去 chunk 头的大小
    - p = malloc(100); p == 目标地址

0x12 unsorted_bin_into_stack

通过改写 unsorted bin 里 chunk 的 bk 指针到任意地址，从而在栈上 malloc 出 chunk.

- 步骤:

    - victim = malloc(0x100);p1 = malloc(100);free(victim)//p1防止victim和top chunk合并
    - 在栈上fake一个chunk,bk指向自身
    - 通过溢出漏洞修改victim的bk指向fake chunk
    - 下一次malloc就会遍历unsortedbin从而找到fake chunk.fake chunk的fd指针被修改成了unsortedbin的地址,可以泄露libc地址.

0x13 unsorted_bin_attack

- 利用条件:

    - 能够控制 unsorted bin chunk 的 bk 指针

- 步骤:

- p1 = malloc(0x100);p2 = malloc(0x100);free(p1)
- 利用溢出修改怕p1的bk为目标地址-2,相当于在目标地址有一个fake free chunk
- 此时malloc系统就会循着bk去找fake chunk

0x14 house_of_einherjar

可以强制使得 malloc 返回一个几乎任意地址的 chunk .

- 利用条件:
  - 要求有一个单字节溢出漏洞，覆盖掉 next chunk 的 size 字段并清除 PREV_IN_USE 标志，然后还需要覆盖prev_size 字段为 fake chunk 的大小
- 步骤:
  - a = malloc(0x38);b = malloc(0xf8)
  - fake一个chunk
  - 使b的PREV_INUSE = 0,prev_size = b - addr(fake_chunk) - sizeof(size_t)*2
  - 使fake_chunk的size=prev_size
  - free(b)//这个时候PREV_IN_USE为0,unlink会根据prev_size去找上一个free chunk并合并.这个时候top chunk->fake_chunk
  - 这个时候malloc的话返回的将是fake chunk的位置.

0x15 house_of_orange

这个涉及的方面有点多,,以后单独发帖写一下.

关于我

博客有我的联系方式,欢迎大家来玩,地址:https://www.0x2l.cn