how2heap 问题汇总(上)

pic4xiu / 2019-10-01 09:20:59 / 浏览数 5424 安全技术 二进制安全 顶(0) 踩(0)

好多大佬们都对how2heap这个项目进行了汇总,我就不班门弄斧了,但是同时大佬对一些问题一笔带过,这里就记一下本人在学 how2heap 中的一些有疑问的点,应该具有一定的代表性.大佬可以帮忙挑错,希望和大家一起进步

first_fit 疑问和拓展

我一开始就有疑问,为什么明明是 smallbins 和 largebins 范围内的 chunk ,它直接去 unsortedbins 呢,事实上只要不是 fastbins 范围内的,经过 free 后都会先进入 unsorted bin 待命.系统在进行 malloc 分配的时候 unsortedbin 算是起到一个缓冲区的作用,测试程序如下

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
   char* a = malloc(512);
   char* b = malloc(256);
   free(a);
   char* c = malloc(500);
}
```

断在 malloc(c)处(注:以后 `char* c = malloc(500)` 这种直接简写成 malloc(c))

```
4 int main()
   5 {
   6      char* a = malloc(512);
   7      char* b = malloc(256);
   8      free(a);
■  9      char* c = malloc(500);
  10 //   char* d = malloc(512);
  11 }
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ STACK ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
...
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ BACKTRACE ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
...
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
Breakpoint /home/pic/■■/te.c:9
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x602000 —■ 0x7ffff7dd1b78 (main_arena+88) ■— 0x602000
smallbins
empty
largebins
empty
```

之后 malloc(c) 会把 unsortedbin 这个取出(只有这个bin不加s,不是复数,也可以说明只有一个链表,2333),而倘若把程序进行如下的修改

```
5 {
   6      char* a = malloc(512);
   7      char* b = malloc(256);
   8      free(a);
   9      char* c = malloc(10);
■ 10      char* d = malloc(512);
  11 }
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ STACK ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
...
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ BACKTRACE ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

```
...
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
pwndbg> x/32gx 0x602000
0x602000:   0x0000000000000000  0x0000000000000021
0x602010:   0x00007ffff7dd1d78  0x00007ffff7dd1d78
0x602020:   0x0000000000000000  0x00000000000001f1
0x602030:   0x00007ffff7dd1b78  0x00007ffff7dd1b78
0x602040:   0x0000000000000000  0x0000000000000000
0x602050:   0x0000000000000000  0x0000000000000000
0x602060:   0x0000000000000000  0x0000000000000000
0x602070:   0x0000000000000000  0x0000000000000000
0x602080:   0x0000000000000000  0x0000000000000000
0x602090:   0x0000000000000000  0x0000000000000000
0x6020a0:   0x0000000000000000  0x0000000000000000
0x6020b0:   0x0000000000000000  0x0000000000000000
0x6020c0:   0x0000000000000000  0x0000000000000000
0x6020d0:   0x0000000000000000  0x0000000000000000
0x6020e0:   0x0000000000000000  0x0000000000000000
0x6020f0:   0x0000000000000000  0x0000000000000000
```

可以看到程序仍然是从 unsortedbin 取出的,不过进行了切割, c 只拿走了0x20(1是标志位)个字节,剩下了0x1f0字节(剩下的我们称之为 remainder chunk ,仍留在 unsortedbin 中),而 remainder chunk 完全不够 d 的大小,所以猜想 d 会切割 top chunk ,之后我们再 n 单步运行发现果然是这样,但是同时

```
pwndbg> heap
0x602000 FASTBIN {          <------------------c
 prev_size = 0,
 size = 33,
 fd = 0x7ffff7dd1d78 <main_arena+600>,
 bk = 0x7ffff7dd1d78 <main_arena+600>,
 fd_nextsize = 0x0,
 bk_nextsize = 0x1f1
}
0x602020 PREV_INUSE {           <------------------■■■ remainder chunk
 prev_size = 0,
 size = 497,
 fd = 0x7ffff7dd1d58 <main_arena+568>,
 bk = 0x7ffff7dd1d58 <main_arena+568>,
 fd_nextsize = 0x0,
 bk_nextsize = 0x0
}
0x602210 {              <------------------b
 prev_size = 496,
 size = 272,
 fd = 0x0,
 bk = 0x0,
 fd_nextsize = 0x0,
 bk_nextsize = 0x0
}
0x602320 PREV_INUSE {           <--------------------d
 prev_size = 0,
 size = 529,
 fd = 0x0,
 bk = 0x0,
 fd_nextsize = 0x0,
 bk_nextsize = 0x0
}
0x602530 PREV_INUSE {           <--------------------top chunk
 prev_size = 0,
 size = 133841,
 fd = 0x0,
 bk = 0x0,
 fd_nextsize = 0x0,
 bk_nextsize = 0x0
}
pwndbg> p d
$1 = 0x602330 ""
pwndbg> bins
fastbins
0x20: 0x0
```

```
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x0
smallbins
0x1f0: 0x602020 —■ 0x7ffff7dd1d58 (main_arena+568) ■— 0x602020 /* '  `' */
largebins
empty
```

发现我们之前剩下的 chunk 这才被归入了 smallbins .我们可以形象的理解为 unsortedbin
非常强势,试图掌握一切,但是它在能力不足时才会把别人应得的归还,也即我们常说的甩锅,2333

consolidate研究

我在这里直接演示一个比较极端的例子,因为 how2heap 中的程序 fastbin_dup_consolidate 看上去就像是直接整合到了 smallbins 一样

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
int main() {
 void* p1 = malloc(0x70);
 void* p3 = malloc(0x70);
 void* p4 = malloc(0x70);
 void* p5 = malloc(0x70);
 void* p6 = malloc(0x70);
 void* p7 = malloc(0x70);
 void* p8 = malloc(0x70);
 void* p9 = malloc(0x70);
 void* p10 = malloc(0x70);
 void* p2 = malloc(0x70);
 free(p1);
 free(p3);
 free(p4);
 free(p5);
 free(p6);
 free(p7);
 free(p8);
 free(p9);
 free(p10);
 void* p12 = malloc(0x400);
}
```

直接在 malloc(p12) 中下断,跑起来

```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x602400 —■ 0x602380 —■ 0x602300 —■ 0x602280 —■ 0x602200 ■— ...
unsortedbin
all: 0x0
smallbins
empty
largebins
empty
```

之前 free 的都跑到了 fastbins 再次 n 单步,发现

```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
```

```
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x602410 —■ 0x7ffff7dd1b78 (main_arena+88) ■— 0x602410
smallbins
empty
largebins
empty
```

经过 consolidate 后,我们的 fastbins 全部被摘下来了,同时进入了 unsortedbin ,而且也够 p12 要求的大小.所以整合过的 chunk 就直接分配给 p12 了.

```
pwndbg> p p12
$2 = (void *) 0x602010
```

这时候我们再回头看看 fastbin_dup_consolidate ,把之前我们介绍的强势的 unsortedbin 概念拿过来,我们发现,事实上该程序在 `void* p3 = malloc(0x400);` 时, unsortedbin 的大小并不满足 p3 所要求的大小,所以会进行"甩锅",把 unsortedbin 中的 chunk 丢到 smallbins 中

unsafe_unlink

大佬们已经总结的很好了,解决这两个问题就完事

- unlink 过程
- 最后的赋值修改

unlink过程

```
FD = P->fd;
BK = P->bk;
FD->bk = BK
BK->fd = FD
```

我们看一下, FD 和 BK 都是相对 P(fake chunk) 而言的,所以 FD 就是 0x602058 ,同时 BK 是 0x602060 ,所以 FD->bk = 0x602060 , BK->fd = 0x602058 ,由于`FD->bk`和`BK->fd`这两块都是指向一处地址,后边的有效,改成了

```
pwndbg> x/32gx 0x602058
0x602058:   0x0000000000000000  0x00007ffff7dd2540
0x602068 <completed.7594>:  0x0000000000000000  0x0000000000602058
```

要注意 chunk0_ptr 此时地址为 0x602070 ,我们可以修改指针的指向以达到任意写的目的,可以看看如下的小 demo

最后的赋值修改,类似如此

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

uint64_t  *chunk0_ptr;
int main()
{
   char b[8]="aaaa";
   chunk0_ptr=&b;
   chunk0_ptr[0] = 0x4242424242424242LL;
   printf("%s\n",b);
}
```

程序结果

```
{13:40}~/■■ ➪ gcc te.c
te.c: In function ‘main’:
te.c:10:12: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
 chunk0_ptr=&b;
           ^
{13:40}~/■■ ➪ ./a.out
BBBBBBBB
```


上一篇：apk加固工具探究系列——Obfu... 下一篇：某info 6.2.0正则匹配不严...

1. 1 条回复


knigh**** 2019-10-02 05:42:34

大佬带带弟弟好吗

0 回复Ta

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