Heap extend.

2019年9月14日 17:06

1.

(1)在攻击者感兴趣的chunk块（C_B）之前分配一块chunk(C_A),假设用户无法直接访问C_B.

(2)通过修改C_A中的size字段，使得系统认为整个C_B是C_A的内容。

(3)释放C_A块

(4)使用malloc重新获得一块大小为sizeof(C_A_user_data) + sizeof(C_B)的C_C块，C_C块user_data中包含有C_B

程序:

```
 7 int main(void)
 8 {
 9     char *C_A, *C_B, *C_C;
10     C_A = (char *)malloc(0x10);
11     C_B = (char *)malloc(0x10);
12     memcpy(C_B, "Hello", 5);
13
14     *(C_A - 0x8) = 0x41;
15     //when call the free, the content in the memory will be set to empty
16     free(C_A);
17     //free(C_B);
18     C_C = (char *)malloc(0x30);
19     //C_C = (char *)malloc(0xa0);
20     return 0;
21 }
```

演示：执行gdb ./a.out

2.

fast bin 最大可提供可用空间为0x70Bytes的chunk.

如果要分配的内存大小超过了0x70Bytes，将不会从fast bin中查找freechunk,

程序中在C_A被释放掉以后由于其size字段为0x41，所以会被放入fast bin 中待用，

修改C_A的size字段为b1,

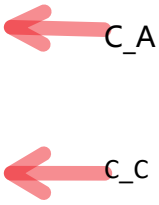
C_C虽然分配user_data是0xa0大小的内存块，需要size字段为0xb1的freechunk，但是由于0xa0 bytes> 0x70Bytes,所以不会从fastbin中查找而会从unsorted bin中查找，这里由于unsorted bin无空闲块 ,所以从top_chunk中重新分配一块。

程序修改:

```
 7 int main(void)
 8 {
 9     char *C_A, *C_B, *C_C;
10     C_A = (char *)malloc(0x10);
11     C_B = (char *)malloc(0x10);
12     memcpy(C_B, "Hello", 5);
13
14     *(C_A - 0x8) = 0x41;
15     //when call the free, the content in the memory will be set to empty
16     free(C_A);
17     //free(C_B);
18     //C_C = (char *)malloc(0x30);
19     *(C_A - 0x8) = 0xb1;
20
21     C_C = (char *)malloc(0xa0);
22     return 0;
```

演示：执行gdb ./a_1.out

```
gdb-peda$ p C_C
$1 = 0x555555767eb0 ""
gdb-peda$ x/32gx 0x555555767e70-0x10
0x555555767e60: 0x0000000000000000      0x00000000000000b1
0x555555767e70: 0x0000000000000000      0x0000000000000000
0x555555767e80: 0x0000000000000000      0x0000000000000021
0x555555767e90: 0x0000006f6c6c6548      0x0000000000000000
0x555555767ea0: 0x0000000000000000      0x00000000000000b1
0x555555767eb0: 0x0000000000000000      0x0000000000000000
0x555555767ec0: 0x0000000000000000      0x0000000000000000
0x555555767ed0: 0x0000000000000000      0x0000000000000000
0x555555767ee0: 0x0000000000000000      0x0000000000000000
0x555555767ef0: 0x0000000000000000      0x0000000000000000
0x555555767f00: 0x0000000000000000      0x0000000000000000
0x555555767f10: 0x0000000000000000      0x0000000000000000
0x555555767f20: 0x0000000000000000      0x0000000000000000
0x555555767f30: 0x0000000000000000      0x0000000000000000
0x555555767f40: 0x0000000000000000      0x0000000000000000
0x555555767f50: 0x0000000000000000      0x000000000000f0b1
```



3.

free的操作会清空chunk块中的内容。

如果在我们获取C_B的内容之前，C_B被free掉了，那么其中的内容会被清空，无法获得需要的信息了。

程序：

```c
int main(void)
{
    char *C_A, *C_B, *C_C;
    C_A = (char *)malloc(0x10);
    C_B = (char *)malloc(0x10);
    memcpy(C_B, "Hello", 5);

    *(C_A - 0x8) = 0x41;
    //when call the free, the content in the memory will be set to empty
    free(C_A);
    free(C_B);
    C_C = (char *)malloc(0x30);
    //*(C_A - 0x8) = 0xb1;
    //C_C = (char *)malloc(0xa0);
    return 0;
}
```

执行 gdb ./a_2.out

4.

(1)在目标块p_2之前分配一大块内存p（超过fastbin 的0x70Bytes上限），修改p的size,使系统误以为p_2也是p的一部分。

(2)释放p(回归unsorted bin注意，p_2其后还有一次malloc，保证在释放p以后，该free_chunk不会被top_chunk合并)

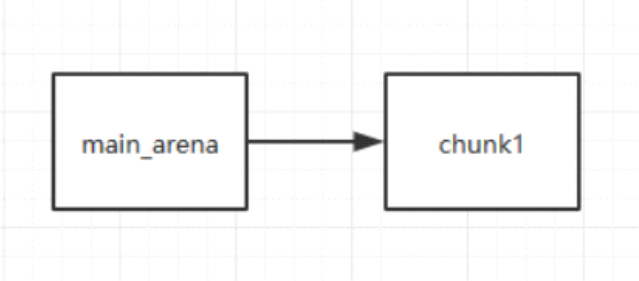
(3)重新分配一块内存p_1,获得目标快p_2的内容。

```
6 int main(void)
7 {
8     char *p, *p_1, *p_2;
9     p = (char *)malloc(0x80);
10    p_2 = (char *)malloc(0x10);
11    memcpy(p_2, "Hello", 5);
12    //incase that the free chunk will be merged to the top chunk.
13    malloc(0x10);
14    *(p - 0x8) = 0xb1;
15    free(p);
16    p_1 = (char *)malloc(0xa0);
17    return 0 ;
```

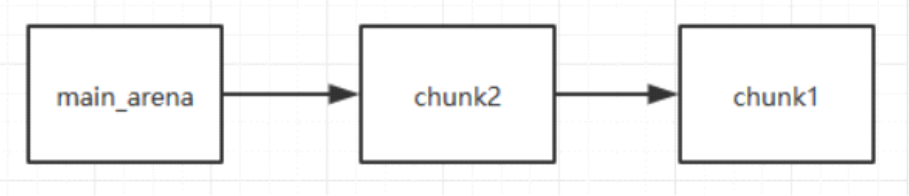
演示：执行gdb ./inuse.out

5.
Double free

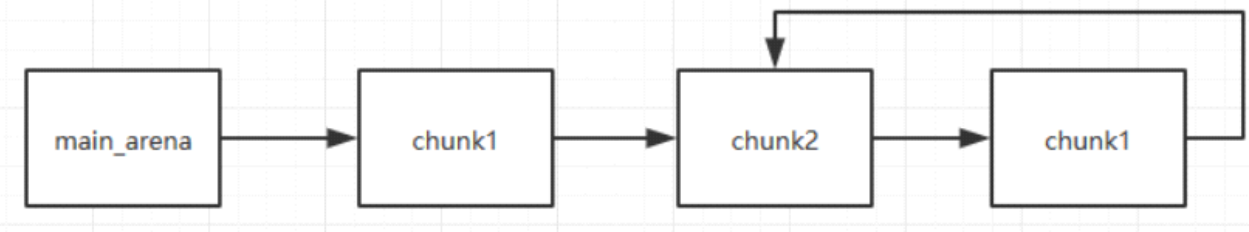
第一次释放 free(chunk1)



第二次释放 free(chunk2)



第三次释放 free(chunk1)



From CTF_WIKI

演示：执行gdb ./double.out

具体按照之前的说明。