# 堆溢出

> “
>
> 各个版本中off by one利用方法

## Glibc2.23

### • 堆溢出

#### - 后向合并

有三个chunk a b c

通过off by one将修改a的size和c的pre_size，使 `size(a') = size(a+b) && size(a') > fast_size`

`free(a')`，此时a和b全部被放入usbin中

申请一个 `size(a)` 的chunk，则此时会对 a‘进行切割 main_arena地址会被写在b的fd和bk指针上，之后 `show(b)` 即可泄露地址

#### - 前向合并

有三个chunk a b c

free(a), 将c的pre_size改为 a+b

free(c)，此时使用中的b就也被裹挟了，之后我们可以利用切割的方式来泄露地址或者任一地址写

### • Off By Null

#### - 堆重叠1

有三个chunk a b c

free b，通过 off by null将b的size改小

add b1 b2，由于b的size被改小，因此c的pre_inuse不会被更新，它仍然认为b是原来的大小

free(b1) free(c), 此时b2就被夹在了中间

- **Off By One**

- **Tips**

  - 堆叠一般都是前向合并，会因为后向合并还要修改下一个堆块的pre_size和pre_inuse

## Glibc2.29

2.29在前向合并时加了一行代码，检测上一个chunk大小是否等于当先chunk的pre_size，即我们不能再在这两个chunk之间 插入准备复用的chunk了 ，之前的方法也就无法使用

```
/* consolidate backward */
if (!prev_inuse(p)) {
    prevsize = prev_size (p);
    size += prevsize;
    p = chunk_at_offset(p, -((long) prevsize));
    // new protecti
    if (__glibc_unlikely (chunksize(p) != prevsize))
        malloc_printerr ("corrupted size vs. prev_size while consolidating");
    unlink_chunk (av, p);
}
```

因此我们需要先伪造上个chunk，让其能通过这两个检测

前向合并检测

```
if (__glibc_unlikely (chunksize(p) != prevsize))
    malloc_printerr ("corrupted size vs. prev_size while consolidating");
```

unlink检测

```
if (__builtin_expect (FD->bk != P || BK->fd != P, 0))
    malloc_printerr (check_action, "corrupted double-linked list", P, AV);
```

## 总结

- **前向合并时** 为什么要把上一个chunk放入usbin中再合并?

- 因为无论是向上还是向下合并都会触发unlink，检查上一个chunk是否存在于一个真实的双链中，因此需要usbin为我们天然的伪造一个双链

- ```
  /* consolidate backward */
  if (!prev_inuse(p)) {
      prevsize = p->prev_size;
      size += prevsize;
      p = chunk_at_offset(p, -((long) prevsize));
      unlink(av, p, bck, fwd);
  }
  ```
  ```