heap(Use-After-Free)

堆概述

在程序运行过程中，堆可以提供动态分配的内存，允许程序申请大小未知的内存。堆是程序虚拟地址空间的一块连续的线性区域，它由低地址向高地址方向增长。目前 Linux 标准发行版中使用的堆分配器是glibc中的堆分配器：ptmalloc2。ptmalloc2主要是通过 malloc/free 函数来分配和释放内存块。

堆数据结构

```
/*
  This struct declaration is misleading (but accurate and necessary).
  It declares a "view" into memory allowing access to necessary
  fields at known offsets from a given base. See explanation below.
*/
struct malloc_chunk {

  INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;         /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```

prev_size：当物理内存空间中的上一块堆块被释放后，当前堆块会存储它的size。

size：存储自己堆块大小。

fd, bk：有些大小的堆块在被释放后由双向链表管理，fd, bk存储逻辑上的上一块和下一块被释放的堆块。

fd_nextsize, bk_nextsize：比较大的堆块用这两个区域存储上一块和下一块的大小。

已经分配在使用中的堆块

pre_size	size
user_data	user_data

…	…

释放后的堆块

pre_size	size
fd	bk
…	…

大多数程序经常会申请以及释放一些比较小的内存块。堆管理器中专门设计了一个Fast Bin的数据结构回收释放后的小堆块，Fast Bin的结构是单项链表，采用LIFO的策略。

Use-After-Free

当一个内存块被释放之后再次被使用。

内存块被释放后，其对应的指针没有被设置为 NULL，但是在它下一次使用之前，有代码对这块内存进行了修改，那么当程序再次使用这块内存时，**就很有可能会出现奇怪的问题。称被释放后没有被设置为 NULL 的内存指针为 dangling pointer。**

释放堆块后其对应的指针没有被清除，很有可能会导致存在多个指针指向同一个堆块，就会产生一些非预期的行为，攻击者利用这一点可以完成利用。