软件安全与漏洞分析

2.2 缓冲区溢出漏洞 --- 堆溢出

Previously in Software Security

- 栈的构造与动态行为
- 栈溢出原理与实例
- 针对栈溢出的内存完整性保护

缓冲区溢出 --- 堆溢出

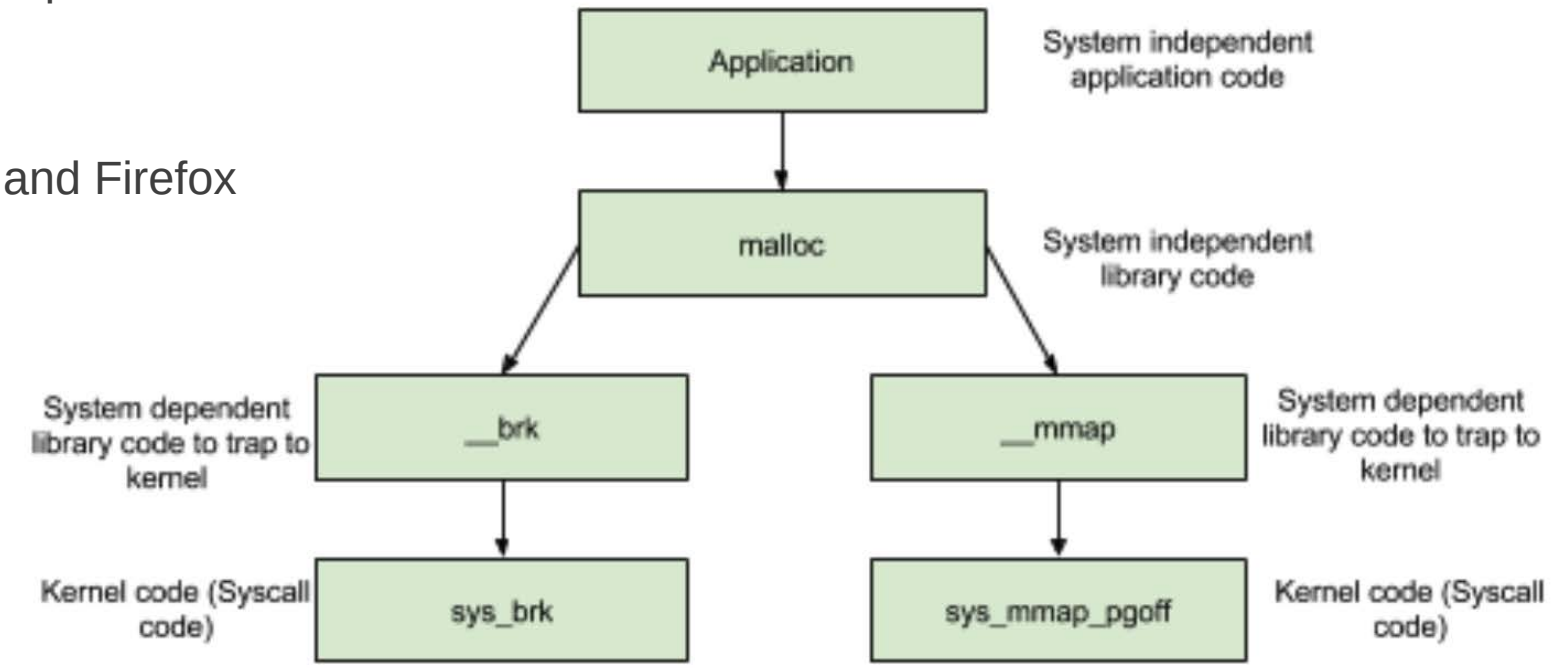
- 本节主题 -- 1. 堆的构造与维护原理
  - 堆的场地（Arena）和块（Chunk）
  - 隐式/显式链表与堆的维护
- 本节主题 -- 2. 堆溢出
  - Linux系统的典型堆溢出原理
  - 现有应对策略与技术

堆的工作原理

- 堆与栈的区别
  - 栈是具有**硬件直接支持**的
  - 栈**存储数据+控制流信息**
  - 堆是由操作系统的**库函数**予以**支持**的
  - 堆**存储数据**

堆的工作原理

- 堆的创建和维护：**通过malloc实现**
  - dlmalloc – General purpose allocator
  - **ptmalloc2 – glibc**
  - jemalloc – FreeBSD and Firefox
  - tcmalloc – Google
  - libumem – Solaris



堆的工作原理

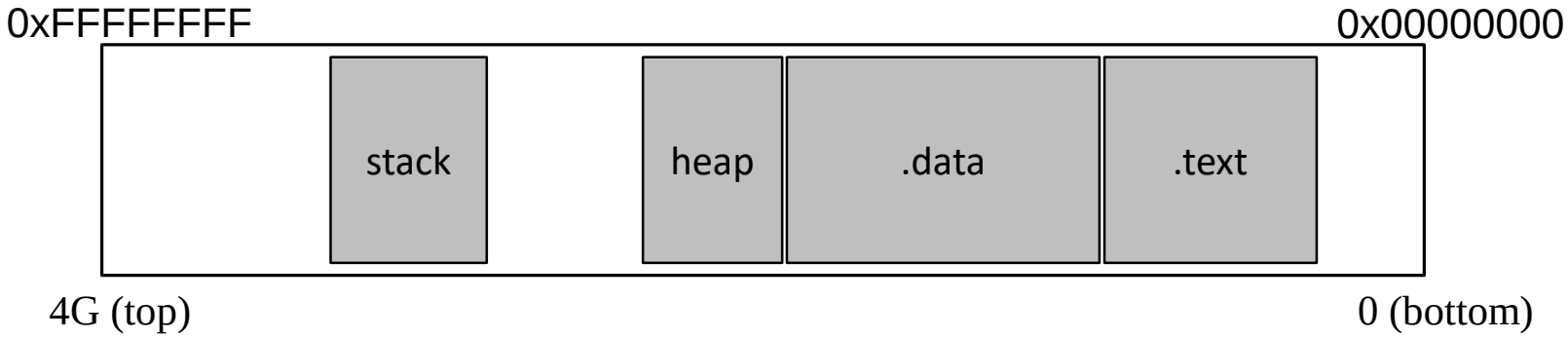
```
int main() {
        pthread_t t1;
        void* s;
        int ret;
        char* addr;

        printf("Welcome to per thread arena example::%d\n",getpid());
        printf("Before malloc in main thread\n");
        getchar();
        addr = (char*) malloc(1000);
        printf("After malloc and before free in main thread\n");
        getchar();
        free(addr);
        printf("After free in main thread\n");
        getchar();
        ret = pthread_create(&t1, NULL, threadFunc, NULL);
        if(ret)
        {
```

```
void* threadFunc(void* arg) {
        printf("Before malloc in thread 1\n");
        getchar();
        char* addr = (char*) malloc(1000);
        printf("After malloc and before free in thread 1\n");
        getchar();
        free(addr);
        printf("After free in thread 1\n");
        getchar();
}
```

堆的工作原理

- 1:1　起初，系统创造进程
- 1:2　数据段之上空虚混沌，并无一物；进程的运行在栈上
- 1:3　malloc说：要有堆。于是就有了堆



堆的工作原理

```
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/mthread$ ./mthread
Welcome to per thread arena example::6501
Before malloc in main thread
After malloc and before free in main thread
...
sploitfun@sploitfun-VirtualBox:~/lsploits/hof/ptmalloc.ppt/mthread$ cat /proc/6501/maps
08048000-08049000 r-xp 00000000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
08049000-0804a000 r--p 00000000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
0804a000-0804b000 rw-p 00001000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
0804b000-0806c000 rw-p 00000000 00:00 0          [heap]
b7e05000-b7e07000 rw-p 00000000 00:00 0
...
```

堆的工作原理

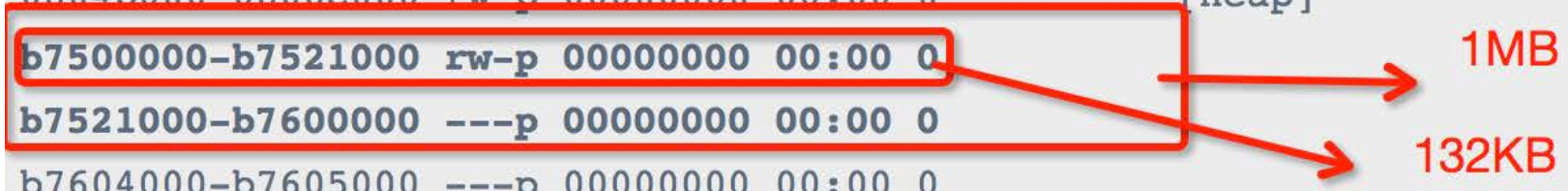
```
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/mthread$ ./mthread
Welcome to per thread arena example::6501
Before malloc in main thread
After malloc and before free in main thread
After free in main thread
...
sploitfun@sploitfun-VirtualBox:~/lsploits/hof/ptmalloc.ppt/mthread$ cat /proc/6501/maps
08048000-08049000 r-xp 00000000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
08049000-0804a000 r--p 00000000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
0804a000-0804b000 rw-p 00001000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
0804b000-0806c000 rw-p 00000000 00:00 0          [heap]
b7e05000-b7e07000 rw-p 00000000 00:00 0
...
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/mthread$
```

堆的工作原理

```
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/mthread$ ./mthread
Welcome to per thread arena example::6501
Before malloc in main thread
After malloc and before free in main thread
After free in main thread
Before malloc in thread 1
...
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/mthread$ cat /proc/6501/maps
08048000-08049000 r-xp 00000000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
08049000-0804a000 r--p 00000000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
0804a000-0804b000 rw-p 00001000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
0804b000-0806c000 rw-p 00000000 00:00 0          [heap]
b7604000-b7605000 ---p 00000000 00:00 0
b7605000-b7e07000 rw-p 00000000 00:00 0          [stack:6594]
...
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/mthread$
```

堆的工作原理

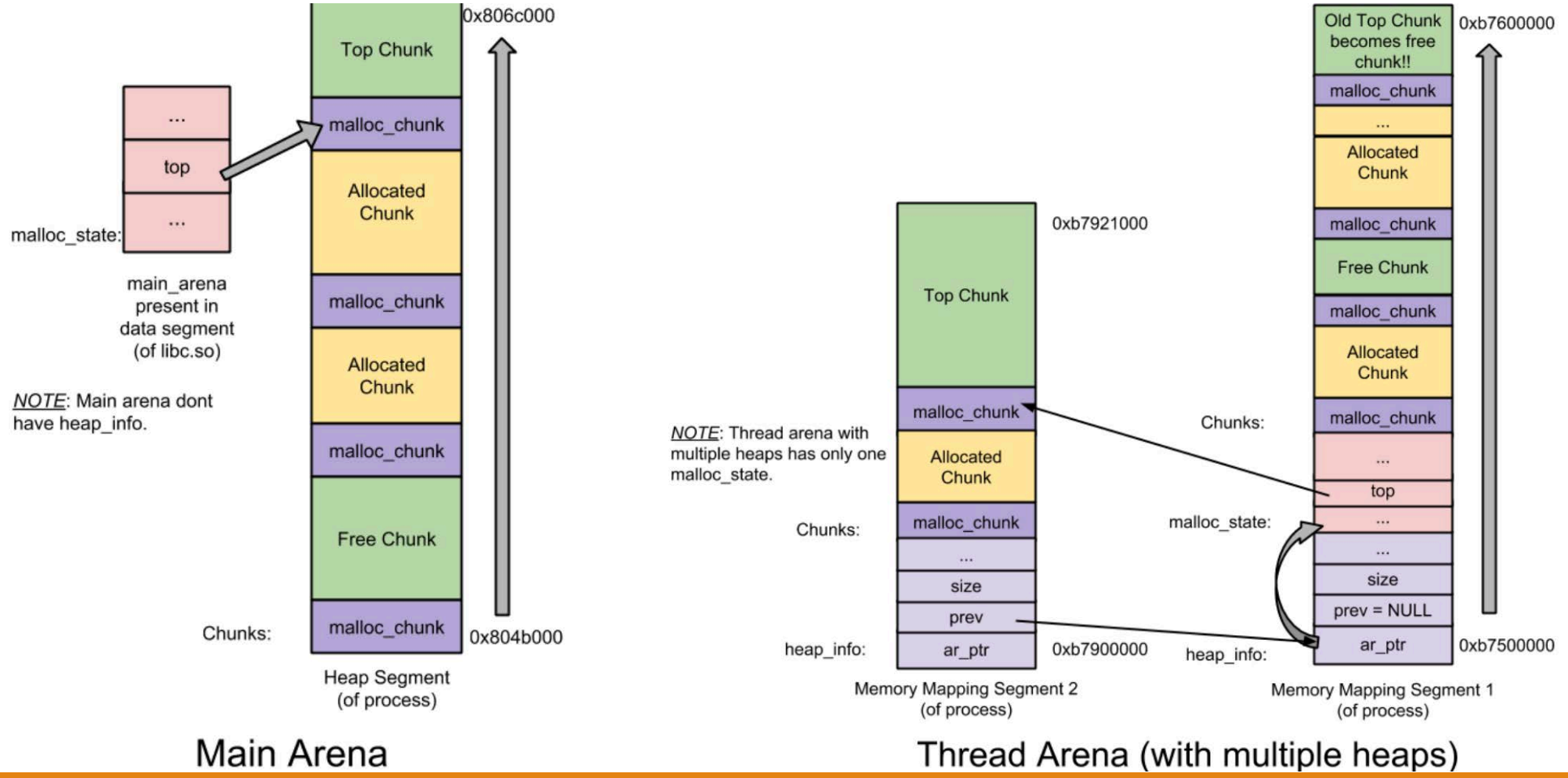
```
After malloc and before free in thread 1
...
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/mthread$ cat /proc/6501/maps
08048000-08049000 r-xp 00000000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
08049000-0804a000 r--p 00000000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
0804a000-0804b000 rw-p 00001000 08:01 539625     /home/sploitfun/ptmalloc.ppt/mthread/mthread
0804b000-0806c000 rw-p 00000000 00:00 0          [heap]
b7500000-b7521000 rw-p 00000000 00:00 0
b7521000-b7600000 ---p 00000000 00:00 0
b7604000-b7605000 ---p 00000000 00:00 0
b7605000-b7e07000 rw-p 00000000 00:00 0          [stack:6594]
```



堆的工作原理

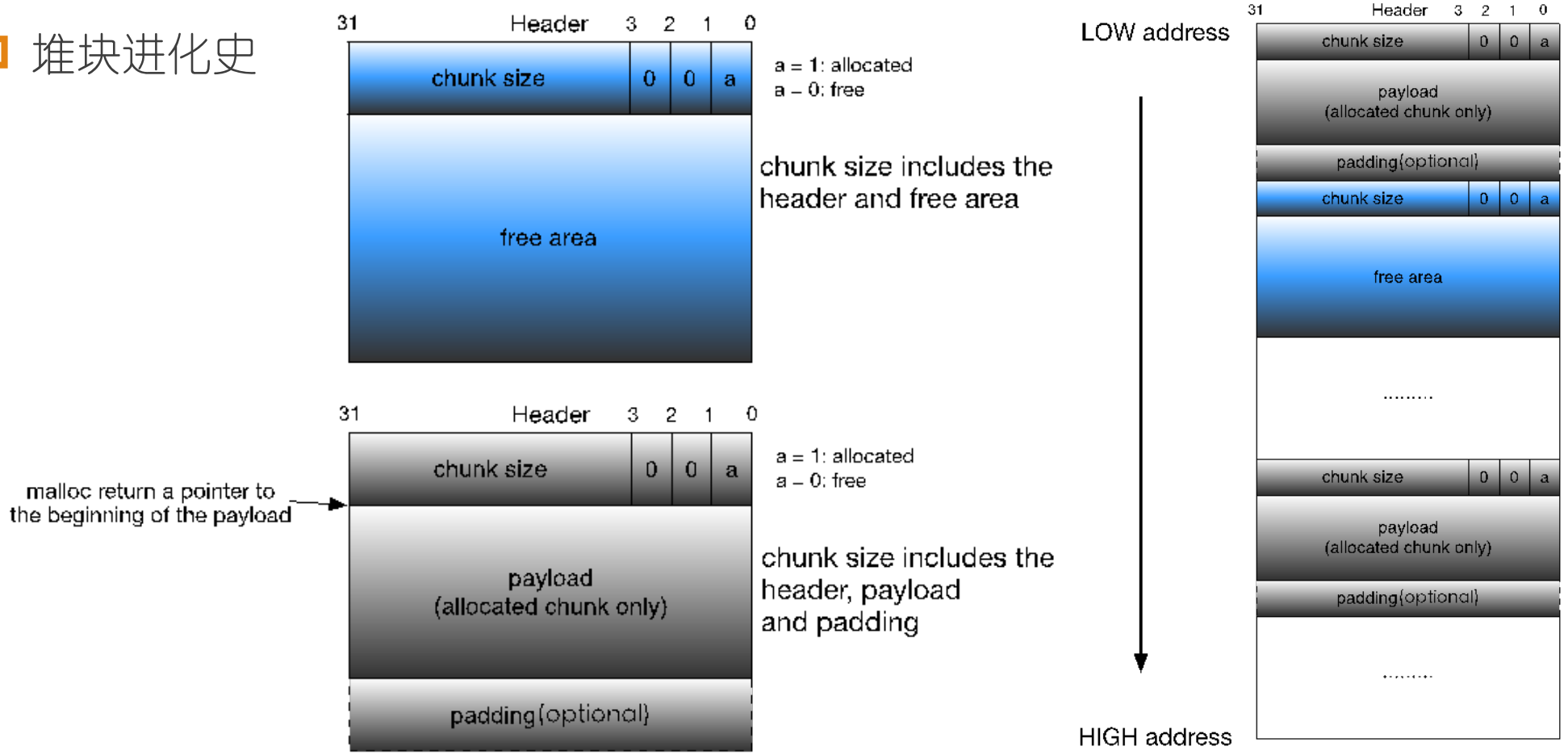
- 上例中，malloc创建的区域首先是**堆的场地（Arena）**
  - 进程申请堆空间时，malloc从场地中划分对应大小的区域供其使用
- **场地的数量限制** -- 与CPU内核数量有关
  - 32位系统：**#_of_Arena = 2 * #_of_cores + 1**
  - 64位系统：**#_of_Arena = 8 * #_of_cores + 1**
- 于是：**同一场地内可能存在多个堆段**

堆的工作原理


0x806c000
Top Chunk
malloc_chunk
Allocated Chunk
malloc_chunk
Allocated Chunk
malloc_chunk
Free Chunk
malloc_chunk
0x804b000
Chunks:
Heap Segment (of process)
...
top
...
malloc_state:
main_arena present in data segment (of libc.so)
NOTE: Main arena dont have heap_info.
Main Arena
0xb7921000
Top Chunk
malloc_chunk
Allocated Chunk
malloc_chunk
...
size
prev
ar_ptr
0xb7900000
Chunks:
heap_info:
NOTE: Thread arena with multiple heaps has only one malloc_state.
Memory Mapping Segment 2 (of process)
0xb7600000
Old Top Chunk becomes free chunk!!
malloc_chunk
...
Allocated Chunk
malloc_chunk
Free Chunk
malloc_chunk
Allocated Chunk
malloc_chunk
...
top
...
...
size
prev = NULL
ar_ptr
0xb7500000
Chunks:
malloc_state:
heap_info:
Memory Mapping Segment 1 (of process)
Thread Arena (with multiple heaps)

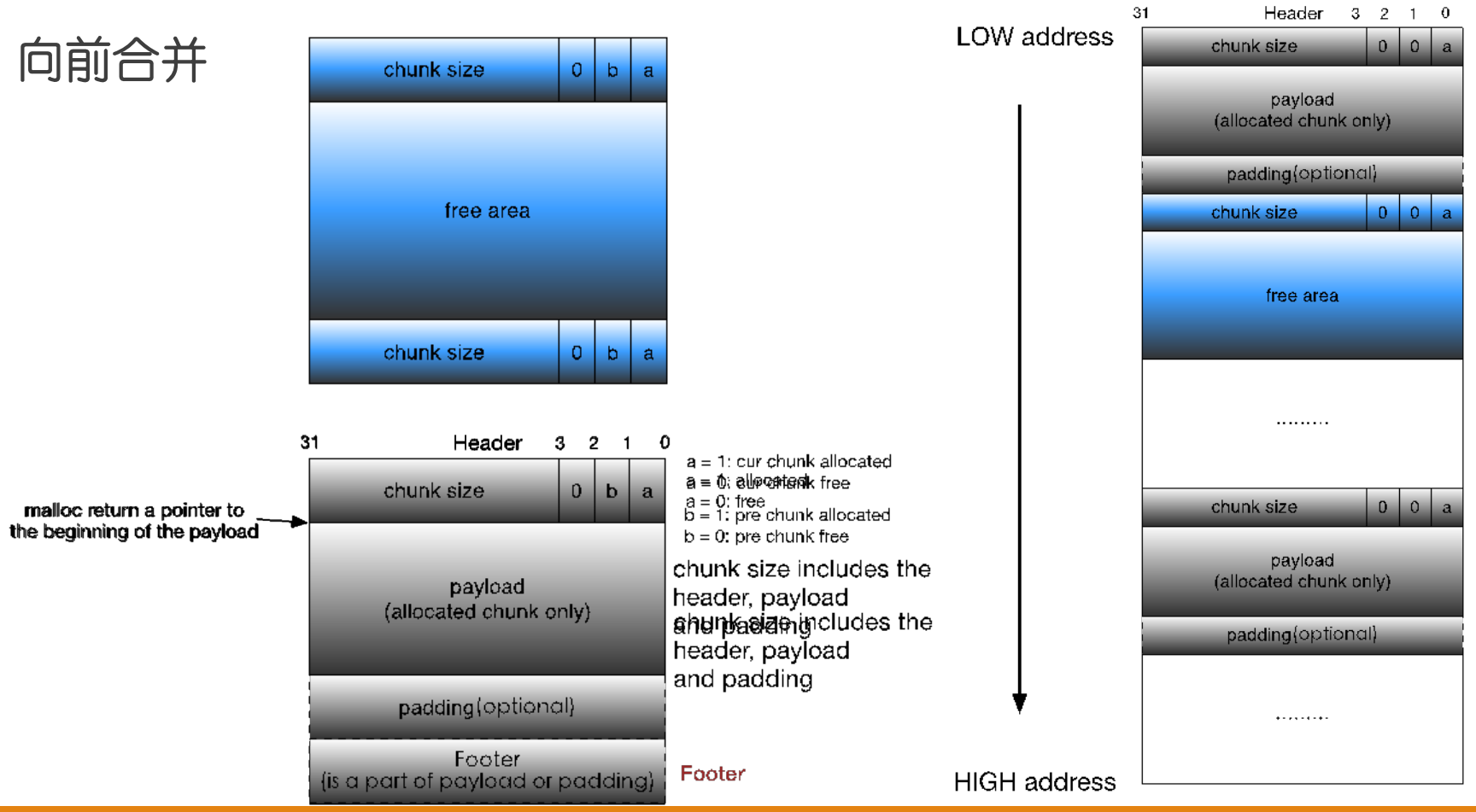
堆的工作原理

- 堆块进化史



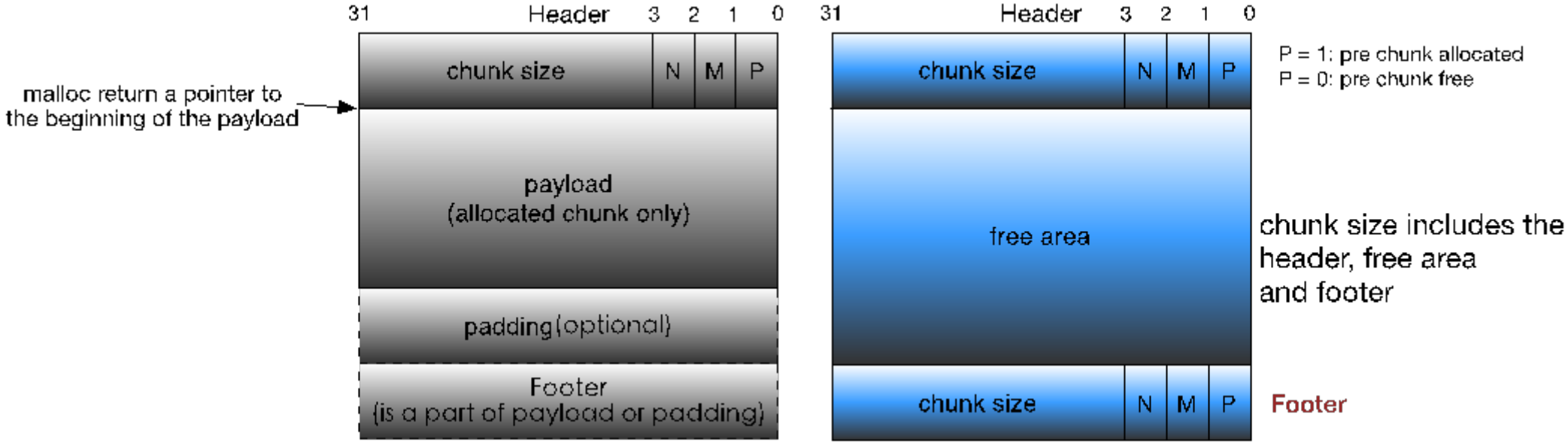
堆的工作原理

- 问题1：向前合并



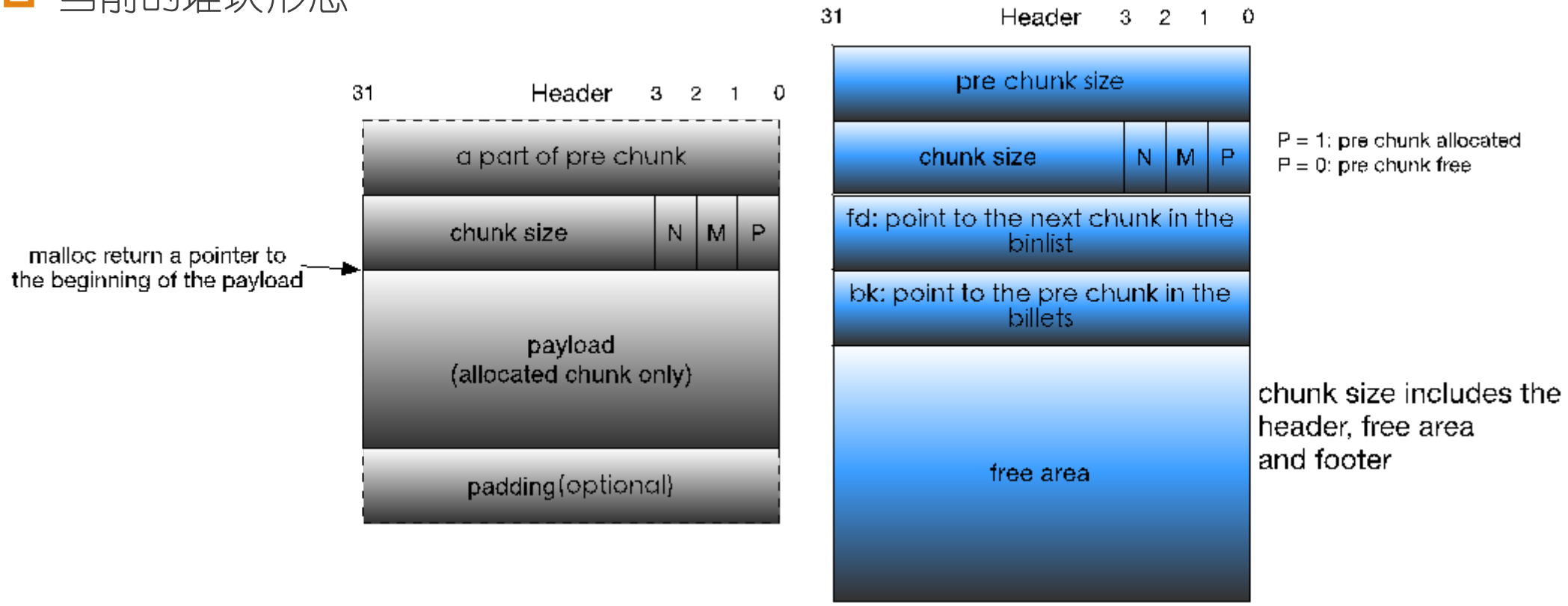
堆的工作原理

- 问题2：多线程时标志位不够！
  - 需要记录：**Main/Thread Arena？ _brk/mmap实现？**
  - 然而，**只有3个标志位**



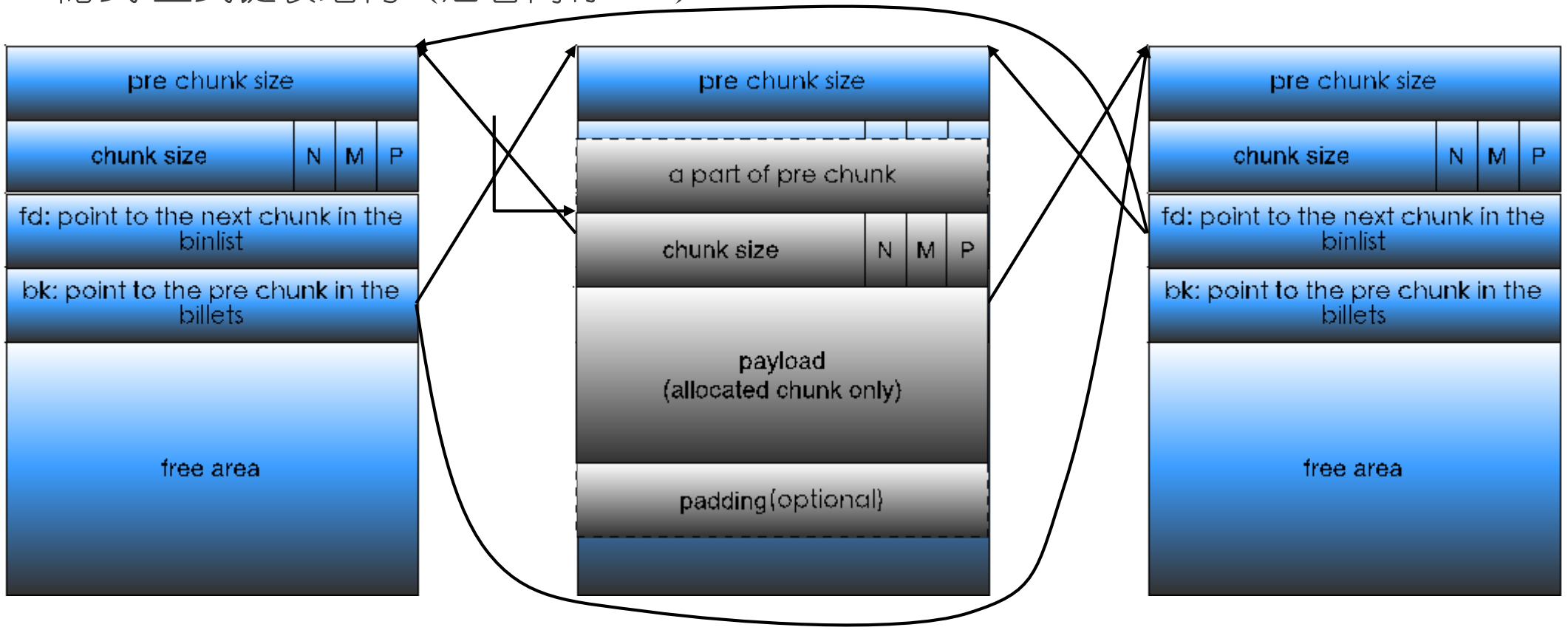
堆的工作原理

- 当前的堆块形态



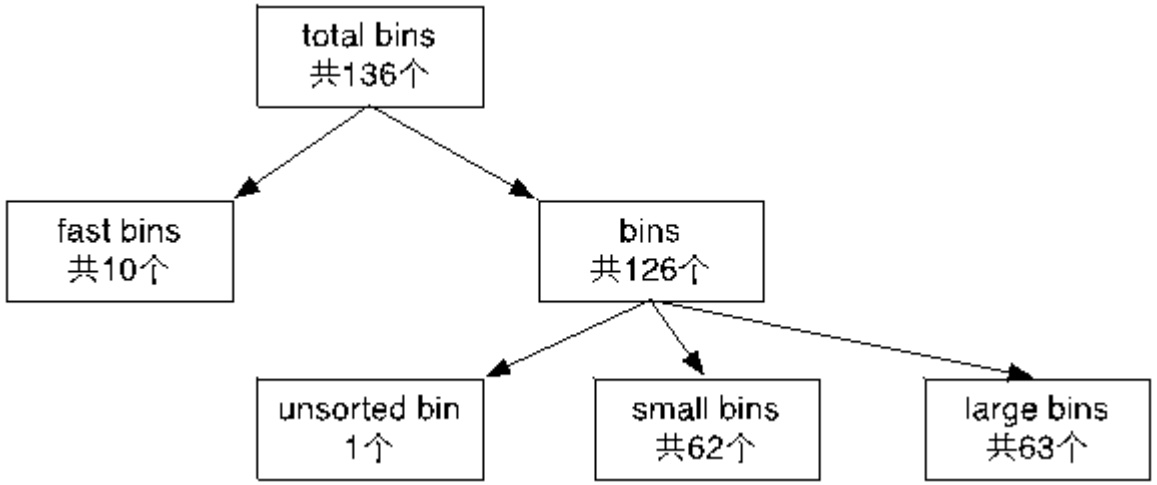
堆的工作原理

- 隐式/显式链表结构（后者简称**bin**）

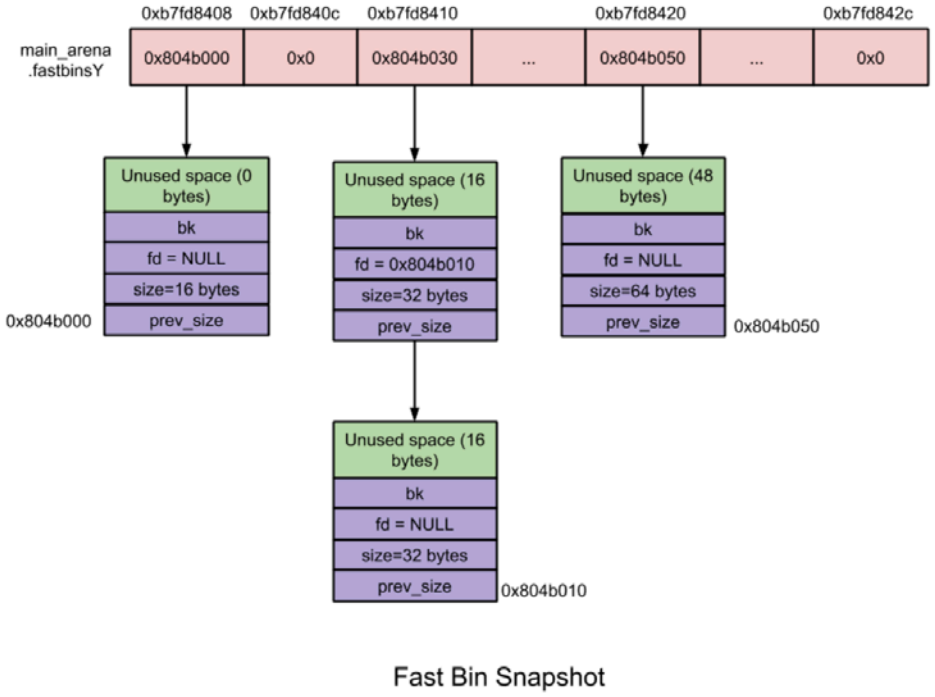


堆的工作原理

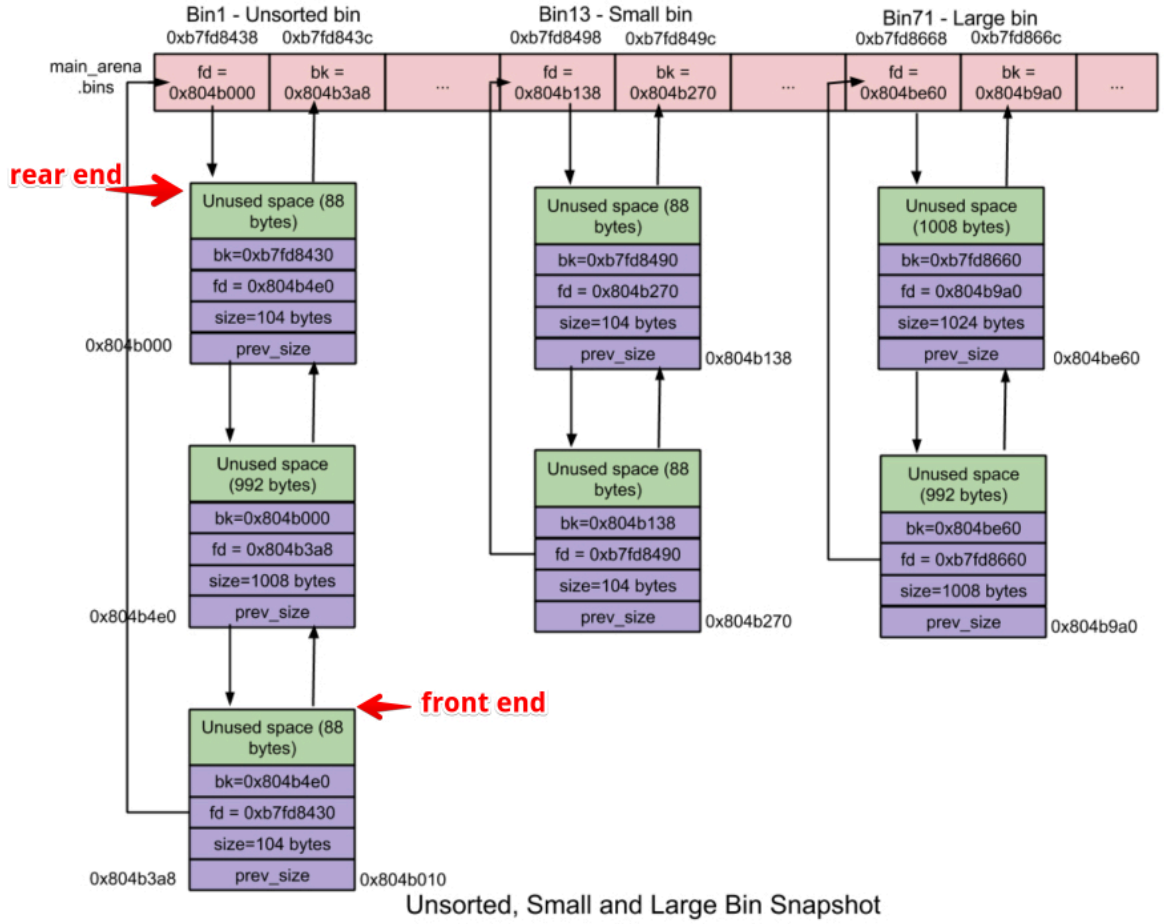
- bin是记录free chunk的链表数据结构，在glibc中可分为：
  - fastbinsY，一个用于记录所有fastbins的数组
  - bins，也是一个数组，记录除fast bins之外的所有bins；其中bin 1 为unsorted bin，bin 2 到63为small bin，bin 64到126为large bin



堆的工作原理

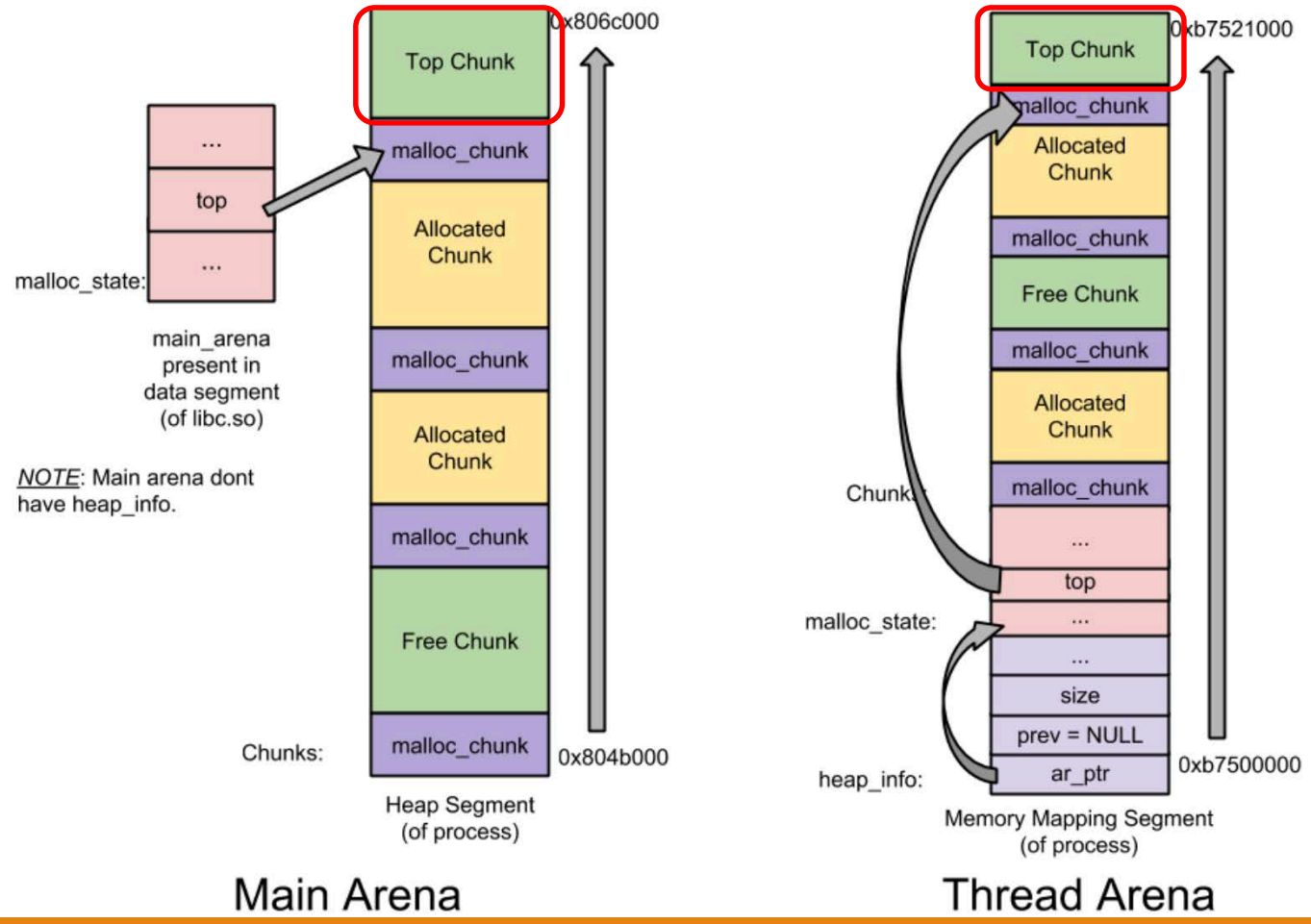


Fast Bin Snapshot



Unsorted, Small and Large Bin Snapshot

堆的工作原理

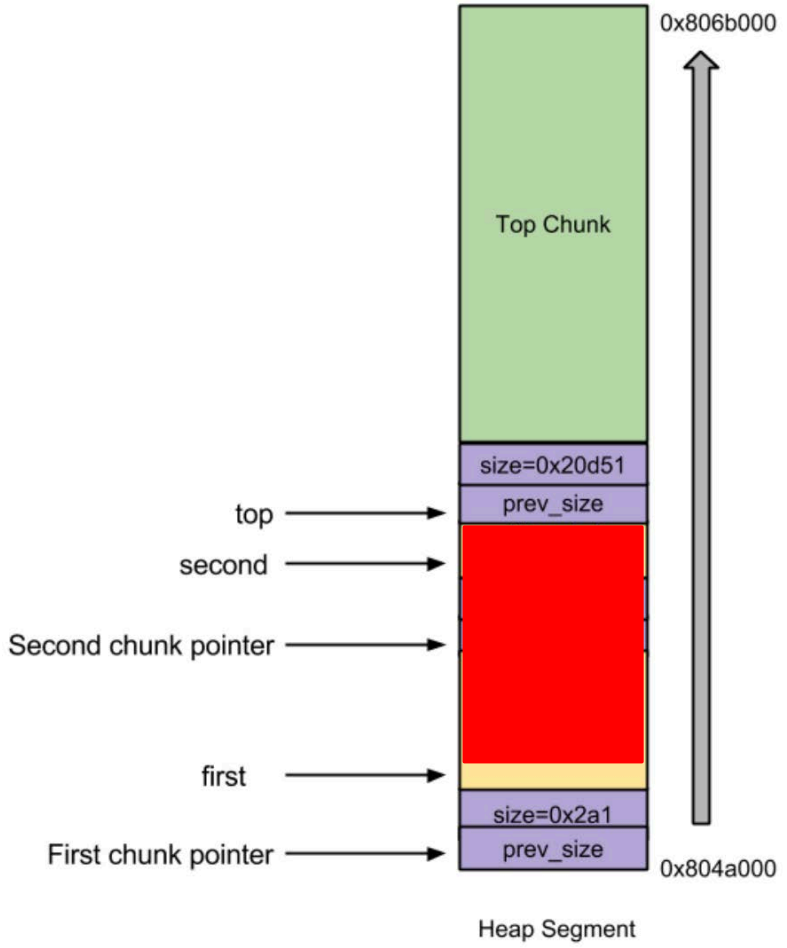

0x806c000
Top Chunk
malloc_chunk
Allocated Chunk
malloc_chunk
Allocated Chunk
malloc_chunk
Free Chunk
malloc_chunk
0x804b000
Chunks:
Heap Segment (of process)
...
top
...
malloc_state:
main_arena present in data segment (of libc.so)
NOTE: Main arena dont have heap_info.
Main Arena
0xb7521000
Top Chunk
malloc_chunk
Allocated Chunk
malloc_chunk
Free Chunk
malloc_chunk
Allocated Chunk
malloc_chunk
...
top
...
size
prev = NULL
ar_ptr
0xb7500000
malloc_state:
heap_info:
Memory Mapping Segment (of process)
Thread Arena

堆的工作原理

- 顶块（top chunk）的作用：**救火队员**
  - 不属于任何bin
  - 当前**所有空闲块**（无论那种bin）全都**尺寸不合**时，由顶块应急
  - 顶块比请求尺寸大 --- **分割**供给使用，**剩余**部分**为新顶块**
  - 顶块比请求尺寸小 --- 全堆无适合块，**扩展堆/分配新堆**

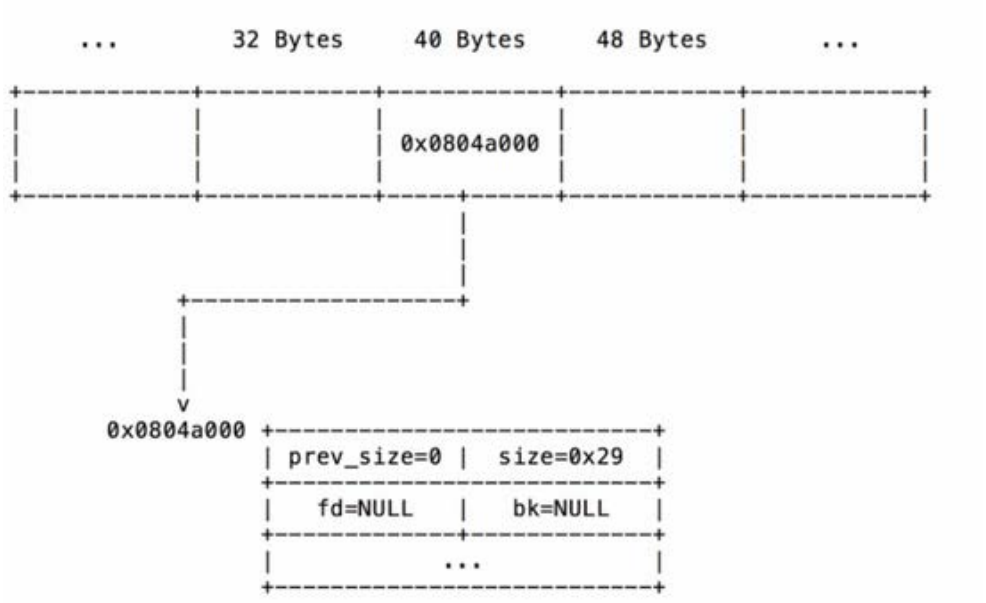
堆溢出

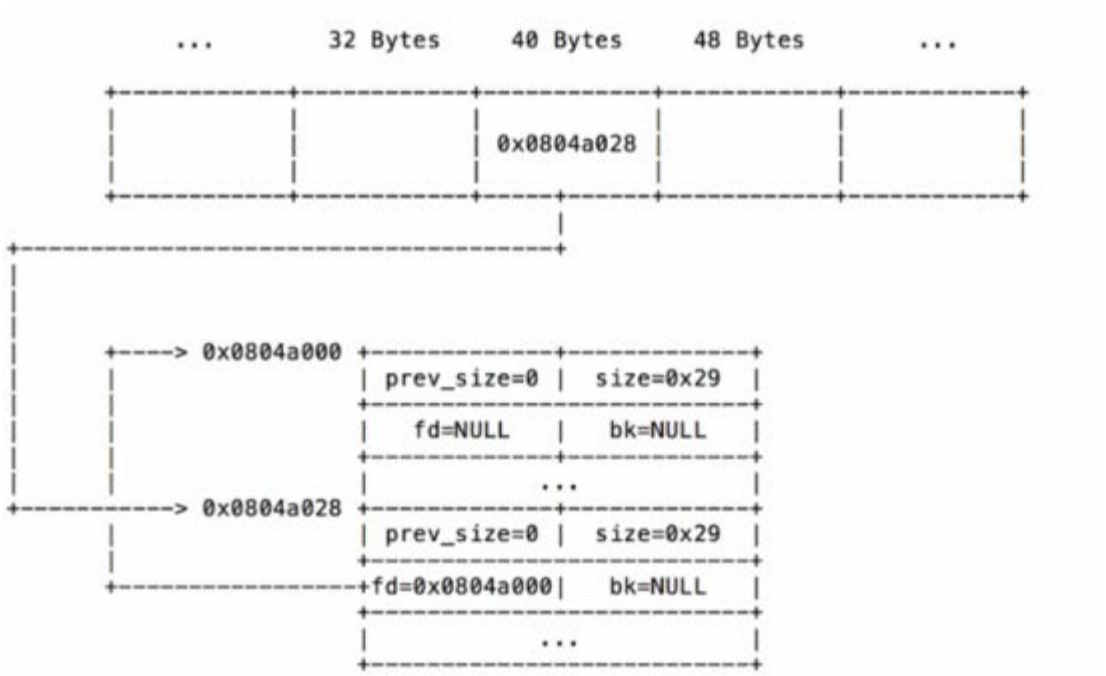
- 与栈溢出的区别：
  - 溢出方向 == 堆增长方向
  - 首先破坏（**虚拟地址意义上的**）下一个堆块的构造
- Linux上的典型堆溢出利用方式：
  - 攻击fastbin
  - 攻击unlink



堆溢出

- 攻击fastbin：





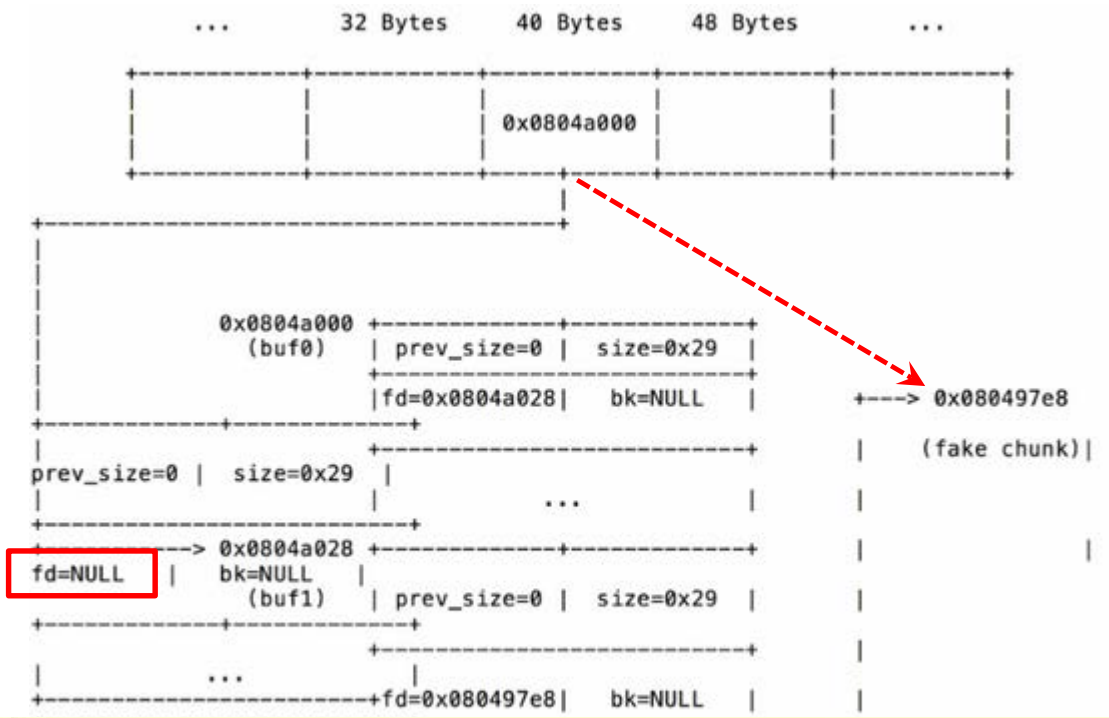
堆溢出

- 攻击fastbin：

```
buf0 = malloc(32);
read(0, buf0, 64);
buf1 = malloc(32);


buf2 = malloc(32);


printf("buf2 is at %p\n", buf2);
```



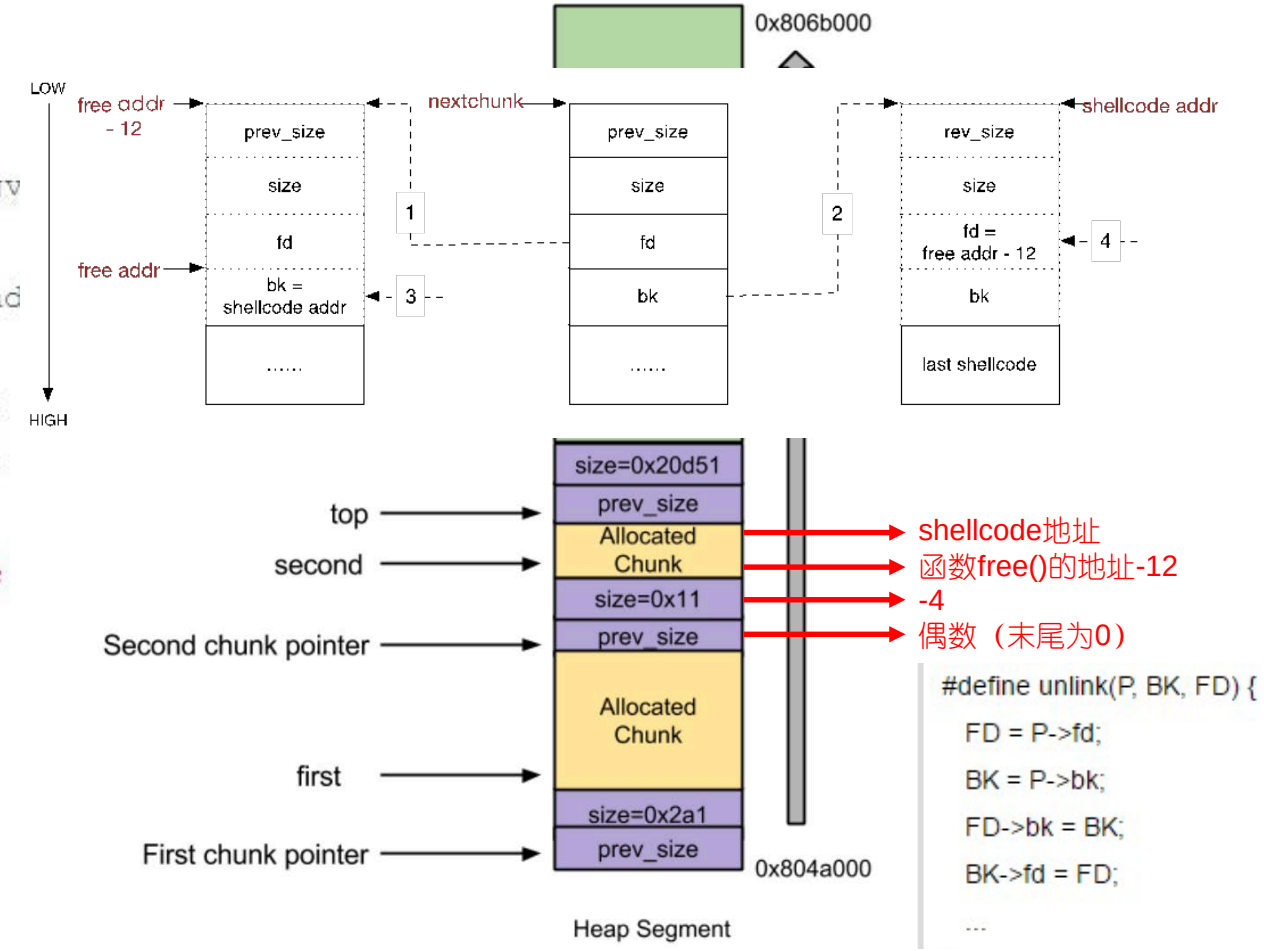

```
root@debian:~/tmp# gcc -m32 fastbin.c -o fastbin.out
root@debian:~/tmp# perl -e 'print "A"x32 . "\x00"x4 . "\x29\x00\x00\x00" . "\xc8\x97\x04\x08"' | ./fastbin.out
buf2 is at 0x80497f0
```

堆溢出

- 攻击unlink：

```
int main( intargc, char * argv
{
        char * first, * second

/*[1]*/ first= malloc( 666 );
/*[2]*/ second= malloc( 12 );
        if(argc!=1)
/*[3]*/     strcpy( first, argv[1] );
/*[4]*/ free(first );
/*[5]*/ free(second );
/*[6]*/ return(0 );
}
```



堆溢出

- Windows堆溢出攻击的主要形式
  - 利用向量化异常处理（VEH）
  - 利用系统默认异常处理函数（UEF）
  - Heap spray
  - Bitmap Flipping攻击
  - Bitmap XOR攻击
  - Heap Cache攻击

堆溢出防御

- 相比栈溢出，针对堆溢出的防御措施更易实用化
  - 堆依靠系统库实现其维护，故堆保护 == 系统库升级
- 针对unlink的保护：
  - Double Free检测
  - next size非法检测
  - 双链表冲突检测

What’s next?

- 格式化字符串溢出漏洞
  - 整数溢出
  - 格式化字符串漏洞