华中科技大学

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

网络空间安全学院

释放后使用及双重释放漏洞

慕冬亮

邮箱：dzm91@hust.edu.cn

常见堆管理器

- dlmalloc – General purpose allocator
- ptmalloc2 – Glibc
- jemalloc – FreeBSD and Firefox
- tcmalloc – Google
- libumem – Solaris
- musl-mallocng – Musl

释放后使用漏洞

- 英文： Use After Free
- 释放后使用漏洞一般涉及三个步骤：
  - 一内存区域A被分配，且有指针p指向它
  - 该内存区域A被回收，但该指针p仍然指向这个区域
  - 解引用悬挂指针p从而访问被释放的内存区域A
- 悬挂指针（Dangling pointer）
  - 指向已释放内存区域的指针

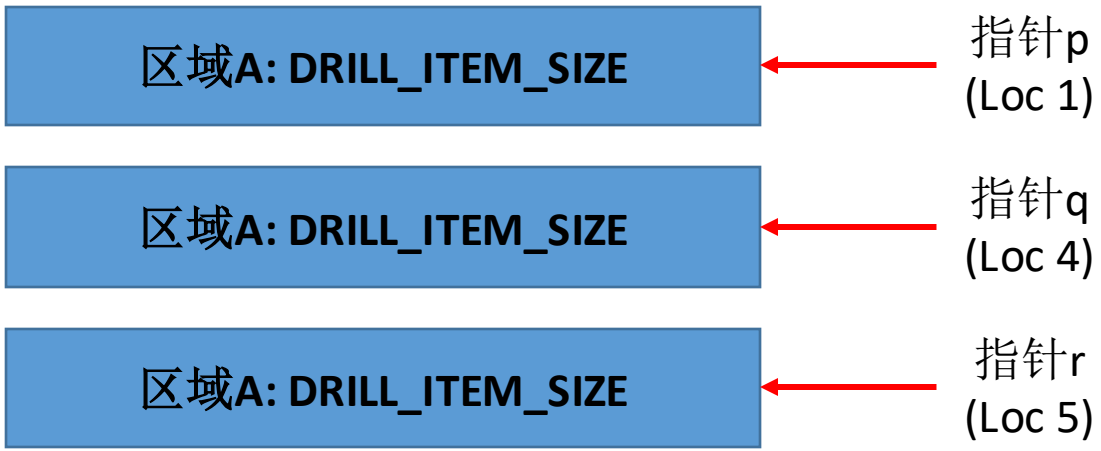

```
1. p = malloc(DRILL_ITEM_SIZE);
2. free(p); // 释放p所指向的对象
// 使用悬挂指针 p 对已回收的数据区域进行操作
3. q = malloc(DRILL_ITEM_SIZE)
// 使用残留指针 p 对新分配的对象进行操作
```

存在两个指针 p, q 同时指向同一内存区域A！

双重释放

- 英文： Double Free
- 双重释放, 特殊的 UAF：
  - 一内存区域A被分配，且有指针p指向它
  - 该内存区域A被回收，但该指针p仍然指向这个区域
  - 调用Free()解引用悬挂指针p再次释放内存区域A

```
1. p = malloc(DRILL_ITEM_SIZE);
2. free(p); // 释放p所指向的对象
3. free(p); // 双重释放p所指向的对象
// 使用悬挂指针 p 对已回收的数据区域进行操作
4. q = malloc(DRILL_ITEM_SIZE)
5. r = malloc(DRILL_ITEM_SIZE)
// 使用残留指针 p 对新分配的对象进行操作
```



存在三个指针 p, q, r 同时指向同一内存区域A！

释放后使用的危害及利用

- 任意地址读写
  - 利用 UAF 读取或修改被释放内存区域的关键数据
  - 利用 UAF 读取或修改新分配对象的内容

```c
char *a = malloc(0x18);
memset(a, 0, 0x10);
// 生成悬挂指针 a
free(a);
printf("Here is a dangling pointer a\n");
char *b = malloc(0x18);
// 读取 flag 至 b 所指向的内存区域
printf("Read key flag into buffer b\n");
read_flag(b);
// UAF 读取新分配对象 b 的内容
printf("UAF leak the content of b\n");
puts(a);
// UAF 修改新分配对象 b 的内容
printf("UAF overwrite the content of b\n");
memset(a, 0x41, 0x8);
puts(b);
```

```
$ ./uaf_demo
Here is a dangling pointer a
Read key flag into buffer b
UAF leak the content of b
flag{local_test}
UAF overwrite the content of b
AAAAAAAAal_test}
```

UAF 读写新分配对象内容

释放后使用的利用思路

- 关键信息泄露
  - UAF读取堆块释放后残留的libc地址数据，绕过ASLR

- 任意地址读写
  - 泄露更多关键数据，如堆地址，Canary(TLS结构体)，栈地址，ELF基地址等
  - 可绕过 Stack Canary 保护，实现控制流劫持（栈迁移等）
  - 可绕过 NX 不可执行保护，实现任意代码执行（ shellcode 等）