CLASS 6

Buffer Overflow Lab 2

吴瑞欣-E41614059

1、名词解释：double free，UAF (Use After Free)，RELRO(Relocation Read Only)，Dangling pointer，Control Flow Guard

Double Free 其实就是同一个指针 free 两次。虽然一般把它叫做 double free。其实只要是 free 一个指向堆内存的指针都有可能产生可以利用的漏洞 double free 的原理其实和堆溢出的原理差不多，都是通过 unlink 这个双向链表删除的宏来利用的。只是 double free 需要由自己来伪造整个 chunk 并且欺骗操作系统。

UAF (Use After Free)应用程序调用 free()释放内存时，如果内存块小于 256kb，dlmalloc 并不马上将内存块释放回内存，而是将内存块标记为空闲状态。这么做的原因有两个：一是内存块不一定能马上释放会内核（比如内存块不是位于堆顶端），二是供应用程序下次申请内存使用（这是主要原因）。当 dlmalloc 中空闲内存量达到一定值时 dlmalloc 才将空闲内存释放会内核。如果应用程序申请的内存大于 256kb，dlmalloc 调用 mmap()向内核申请一块内存，返回返还给应用程序使用。如果应用程序释放的内存大于 256kb，dlmalloc 马上调用 munmap()释放内存。dlmalloc 不会缓存大于 256kb 的内存块，因为这样的内存块太大了，最好不要长期占用这么大的内存资源

RELRO(RELocation Read-Only，只读重定位)让加载器将重定位表中加载时解析的符号标记为只读，这减少了 GOT 覆写攻击的面积。RELRO 可以分为 Partial RELRO(部分 RELRO)和 Full RELRO(完整 RELRO)。开启 Partial RELRO 的话 GOT 表是可写的；开启 FULL RELRO 的话 GOT 表是只读的。从 Fedora 23 开始所有软件包都已启用了 Full RELRO。开启-Wl,-z,relro 选项即可开启 Partial RELRO；开启-Wl,-z,relro,-z,now 选项即可开启 Full RELRO

攻击者可以通过利用内存漏洞来截获控制流。但是研究人员也提出了各种各样的防御手段避免发生这样的问题，例如地址空间随机化（ALSR），异或执行(XOR Execute)，控制流完整性（CFI）等各种各样的防御措施。CFI 通过强制控制流完整性保证程序的执行不会出现问题，目前部署最为广泛的 CFI 是 windows 提供的 control flow guard（CFG）。

Windows 的 CFI 实现称为 Control Flow Guard（CFG），因为实际的性能要求，不可能做到非常精确的 CFI，因此，实际在 windows 上部署的 CFI 是粗粒度的、前向 CFI。首先，粗粒度的 CFI 是：所有的有效跳转地址为一个全局的集合，即不精确的为每一个间接跳转指定一个有效跳转地址；其次，什么叫做前向 CFI：只考略 call，jump 的直接跳转和间接跳转，没有计算 ret 的情况。此外，Windows CFG 实现还依赖于 bitmap 表，该表存储的信息是关于目标地址是否有效的一张表。bitmap 表中的两位与实际地址的 16byte 一一对应，因此有四种情况：

00：该地址范围没有有效的跳转地址
01：地址范围包含导出抑制表目标
10：只有 16 位对其的地址有效（该范围的第一个地址）
11：地址范围的所有地址均有效
https://www.colabug.com/2710390.html

Dangling pointer（悬垂指针、迷途指针）Wild pointer（野指针）

迷途指针经常出现在混杂使用 malloc() 和 free() 库调用: 当指针指向的内存释放了，这时该指针就是迷途的。一个避免这个错误的方法是在释放它的引用后，将该指针的值重置为 NULL。
野指针指的是还没有初始化的指针。严格地说，编程语言中每个指针在初始化前都是野指针。

2、阅读下面这三篇文章：

UAF 学习--原理及利用

https://www.cnblogs.com/alert123/p/4918041.html

操作系统在 free 一块内存后，接着申请大小相同的一块内存，操作系统会将刚刚 free 掉的内存再次分配

Linux 内核提权漏洞的分析和利用（CVE-2016-0728）

http://www.freebuf.com/vuls/93799.html

CVE-2016-0728 这个漏洞本身存在于 Linux 内核密钥管理和保存功能 keyrings 中
keyrings 主要功能是为驱动程序在内核中保留或者缓存安全数据、身份认证秘钥、加密秘钥以及其他数据。由于提供了系统调用接口——即 keyctl 系统调用（此外还有两个处理密钥的系统调用：add_key 和 request_key。不过，keyctl 是本文绝对的关键），因此用户空间的程序可以以管理这些对象，并利用该工具实现各自的目的。

每个进程都会使用 keyctl（KEYCTL_JOIN_SESSION_KEYRING，名称）为当前的会话创建一个密钥环（keyring），然后可以为密钥环（keyring）指定名称，也可以通过传递 NULL 不予指定。密钥环（keyring）对象能够通过引用相同 keyring 名称在不同进程中进行共享。如果进程已经拥有一个会话密钥环(keyring)，keyctl 系统调用便会使用新的密钥环(keyring)取代原来的。如果一个对象被多个进程共享，位于 usage 字段的对象内部引用计数便会递增。当进程替换当前的密钥环（keyring）时，有可能发生泄漏。正如摘自内核 3.18 版本的如下代码片段，代码跳转至 error2 标签，忽略了 key_put 调用，同时泄露由 find_keyring_by_name 增加的引用计数。

通过 UseAfterFree 实现命令执行

https://bbs.pediy.com/thread-221537.htm

通过 UAF 漏洞，实现 GOT 表的覆盖，从而实现命令执行。
在 delete_msg 函数中，我们对节点进行了 free 操作，如果在循环代码中，进行 delete 操作，释放节点后，在选择 2 进入 modify_msg 函数，modify_msg 会根据用户输入的内容，重新分配堆内存。