**一****. 隐藏特定后缀名的文件**

**1.原理（**步骤）

①hook sys\_getdents系统调用的返回值。

int sys\_getdents(unsigned int fd, struct dirent \*dirp,unsigned int count)

其中fd为指向目录文件的文件描述符，该函数根据fd所指向的目录文件读取相应dirent结构，并放入dirp中，其中count为dirp中返回的数据量，正确时该函数返回值为填充到dirp的字节数。

②遍历sys\_getdents的返回值中包含的文件，对比这些文件的文件名是否含有我们需要隐藏的后缀名。

其中，check\_file\_suffix()函数检查文件后缀名

Should\_be\_hidden()函数判断文件是否需要隐藏

③如果需要隐藏，修改返回值中的相应内容。

**功能**：隐藏包含特定后缀的文件，在sys\_getdents中过滤特定后缀的目录项

**2. 隐藏文件的相关函数**

check\_file\_suffix

检查文件后缀名

Should\_be\_hidden()函数

判断遍历到的sys\_getdents系统调用的返回值中的文件是否需要隐藏

new\_sys\_getdents

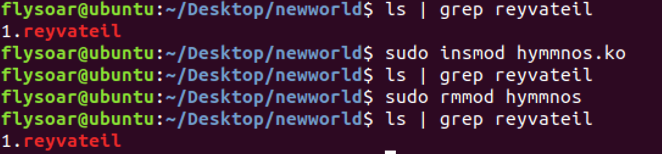
对getdents的hook函数，返回目录文件的相应内容

3.代码注释（在单独的文档里）

4**.测试**

我们隐藏的是后缀名为reyvateil的文件

首先，在/Desktop/newworld 目录下创建一个新的文件1.reyvateil



查看后缀名为reyvateil的文件，返回结果为1.reyvateil；

加载hymmnos模块之后，1.reyvateil文件被隐藏；

卸载hymmnos模块之后，隐藏的文件1.reyvateil被显示

**二.控制(阻止）新的内核模块加载**

**1.原理**

当攻击者获得root权限后，首先应该做的不是提供后门，而是堵上漏洞，以及控制好其他程序，以防其加载其他反rootkit模块与我们在内核态血拼。

控制内核模块的加载，从通知链机制下手。当某个子系统或者模块发生某个事件时，该子系统主动遍历某个链表，而这个链表中记录着其他子系统或者模块注册的事件处理函数，通过传递恰当的参数调用这个处理函数达到事件通知的目的。

注册一个模块通知处理函数，在模块完成加载之后、开始初始化之前， 即模块状态为MODULE\_STATE\_COMING，将其初始函数掉包成一个什么也不做的函数，就会导致模块不能完成初始化。

内核模块的加载过程，我们注册的通知链处理函数是在notifier\_call\_chain函数里被调用的，调用层次为： blocking\_notifier\_call\_chain ->\_\_blocking\_notifier\_call\_chain -> notifier\_call\_chain。

如何注册模块通知处理函数:用于描述通知处理函数的结构体是struct notifier\_block。注册或者注销模块通知处理函数可以使用register\_module\_notifier或者unregister\_module\_notifier，编写一个通知处理函数，然后填充struct notifier\_block结构体，最后使用register\_module\_notifier注册。

功能：阻止新模块的加载

**2.相关函数**

fake\_init，fake\_exit

我们自己写的init，exit函数，替换其他模块的init和exit

module\_notifier

监听新模块regist\_komon，unregist\_komon

注册与解除注册模块notifier

3.代码注释（在单独的文档里）

4**.测试**

module\_notifier函数是用来监听新模块的；测试流程：

①先把当前模块hymmnos加载到内核，此时会将真实的init()和exit()替换为fake\_init()和fake\_exit()；由于将初始化函数替换，导致新模块不能加载。

②加载一个新的模块；加载之后dmesg可以看到fake\_init()的printk内容；

③卸载这个简单的模块，能从dmesg中看到fake\_exit()的printk内容；

如果成功看到这两个printk的信息，则监听成功