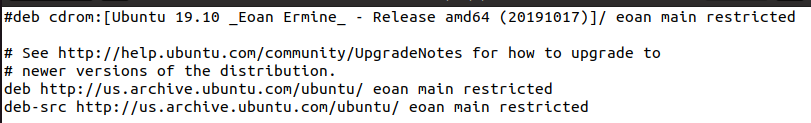
1. 搭建调试ubuntu内核环境

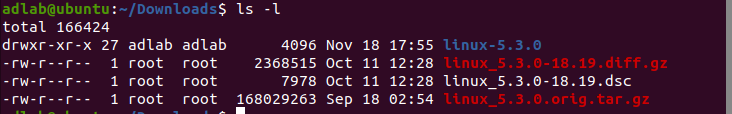
Ubuntu内核源码和内核符号表是开源的，因此都可以下载。

1. 出去source.list中deb-src注释



就打开这个一个注释，这是下载只包含内核的源代码，不下载更新代码。

1. sudo apt-get update 更新，然后sudo apt-get source linux-source-`uname –r`，这里存在漏洞的内核版本是5.3.0-18.19。顺利下载后目录如下：



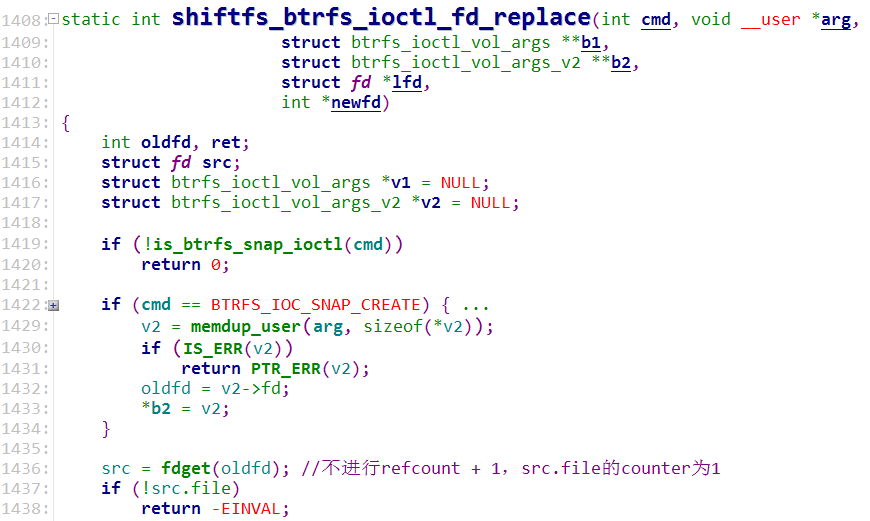
下载过程中apt-get会自动解压缩并将diff文件自动创建到对应目录中。

1. 编译源码并调试
2. 漏洞分析

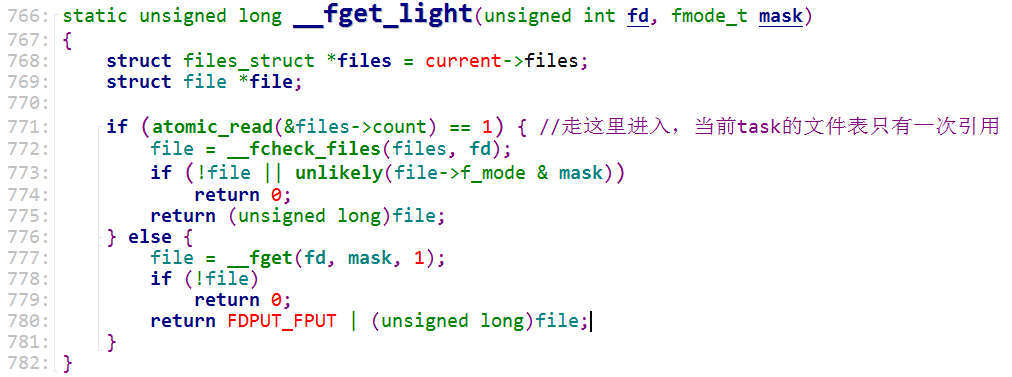
1、CVE-2019-15791是引用计数下溢漏洞，refcount underflow导致的拒绝服务漏洞，即引用计数少于实际引用个数，被引用对象被释放后，依然有指针指向该对象地址，一旦解引用该指针，发生内存访问错误。

该漏洞发生在ubuntu中特定的shiftfs驱动模块中，该驱动模块不存在linux主线中。Shiftfs为ubuntu独有的文件系统。该文件系统可以从用户空间挂载，表明这是一个不需要高权限的攻击面。

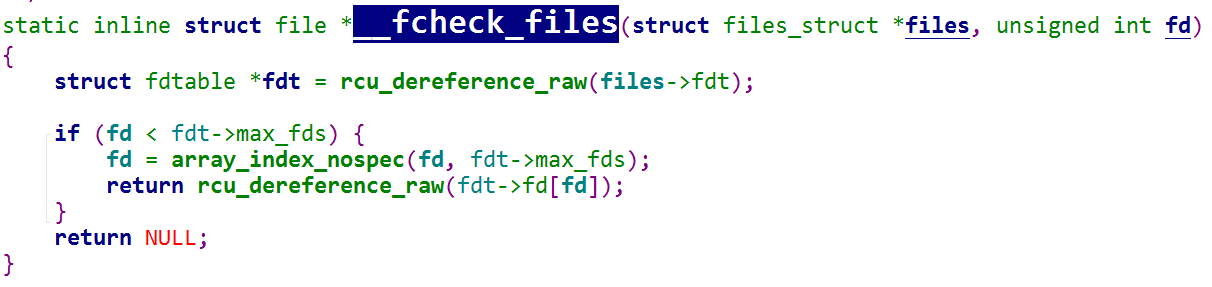
漏洞出现在shiftfs\_btrfs\_ioctl\_fd\_replace()函数中。



通过fdget获取oldfd的fd，也就得到了file。Fdget会调用\_\_fget\_lighth函数



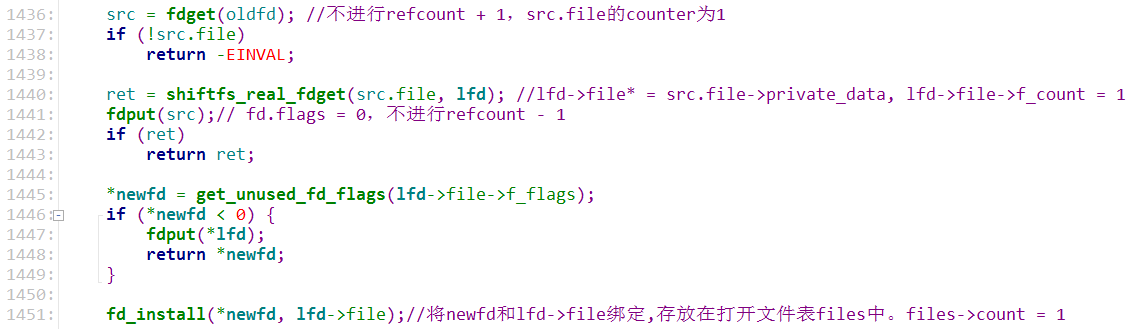
先获取当前task唯一的打开文件表files。Files会维护一个当前进程所打开的所有文件的文件描述符fd数组，并和file结构体对应起来。这里就是找到oldfd所对应的file，通过调试可以确定files->count 为1，所以走上面一个分支，返回fdt->fd[fd]。



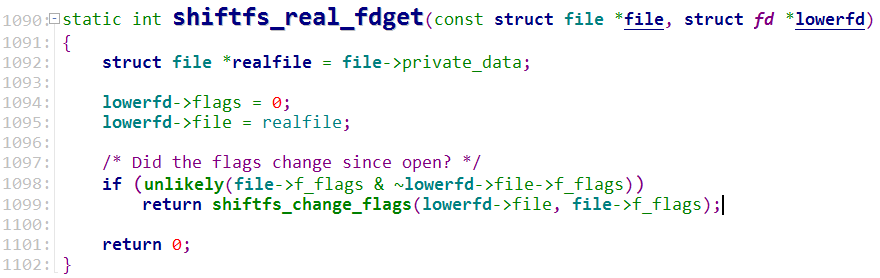
然后有其他进程共享打开文件表，那就会走下面一个路径，下面路径最终会多一个将fd对应的file的引用计数加一的操作。



这里只是简单讲述fdget的操作。

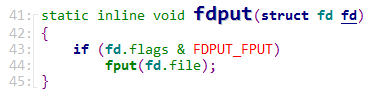


然后调用shiftfs\_real\_fdget函数，该函数会将一个指针指向oldfd对应的file。



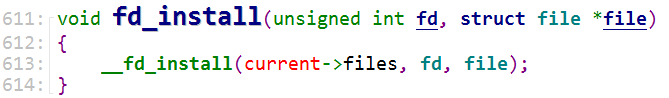
Lowerfd->file 指向了realfile，src.file->private\_data多了一个指针引用，而该函数中并没有进行引用计数加一的操作。

然后调用fdput释放src。Fdget和fdput是对应的，要么同时加一减一，要么都不加减。

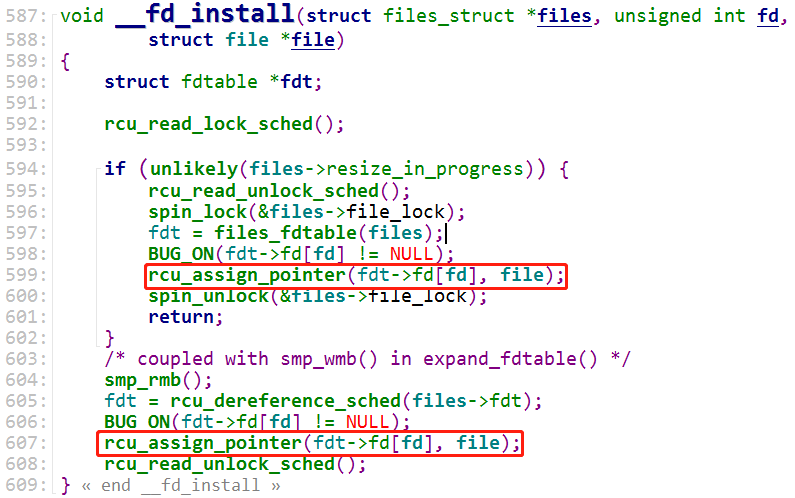


因此对于src.file自身的引用计数的操作来说，都是平衡的，但是就是多了一个指针引用指向src.file->private\_data，最后也就是这个指针引用出现了问题。

后面调用fd\_install将newfd和lfd->file绑定到一起，实现如下：

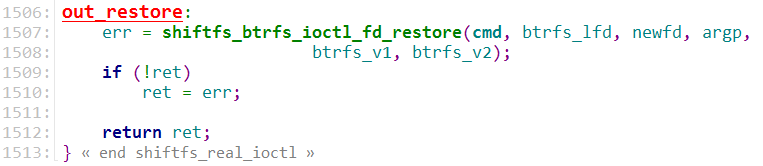


调用\_\_fd\_install。



将file放到fdt->fd[fd]中，这里rcu\_assign\_pointer是rcu模式赋值

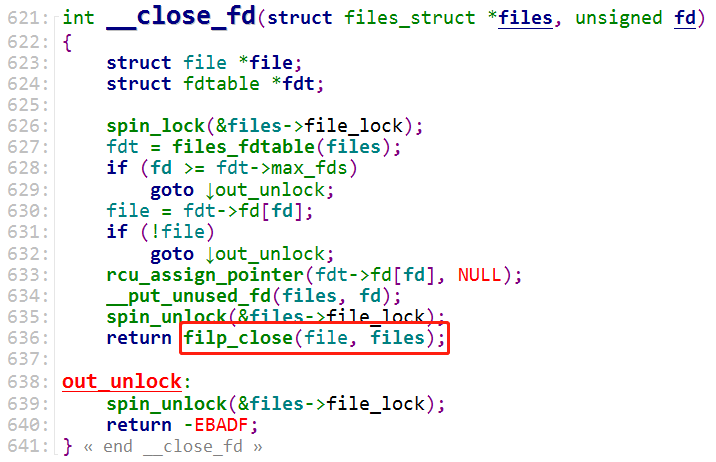
该操作也不会操作引用计数。后面操作都是引用计数平衡的。从shiftfs\_btrfs\_ioctl\_fd\_replace函数返回后，最后会调用shiftfs\_btrfs\_ioctl\_fd\_restore，传入newfd。



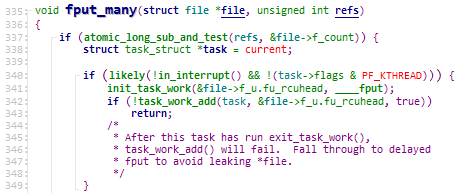
shiftfs\_btrfs\_ioctl\_fd\_restore实现如下：



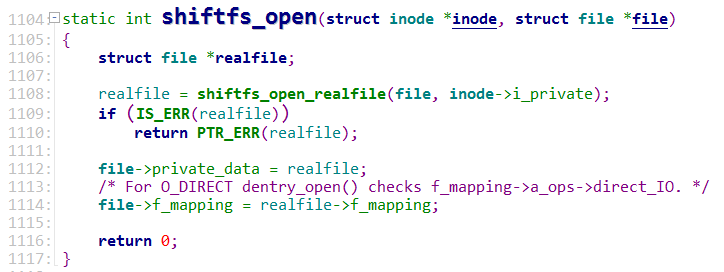
行1401，调用\_\_close\_fd，fd就是newfd。\_\_close\_fd实现如下：



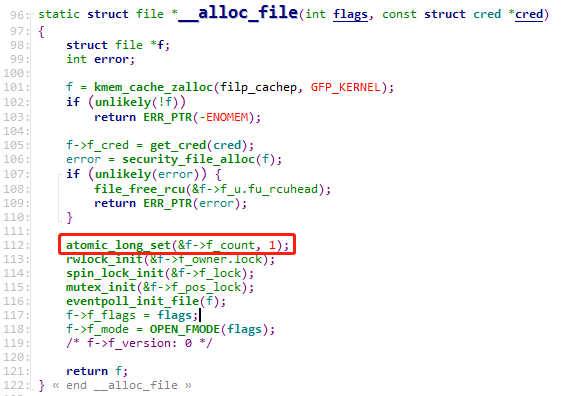
获取newfd对应的file，调用filp\_close，最后会调用fput\_many



先通过atomic\_long\_sub\_and\_test判断lfd->file->count是否为1，lfd->file->count的引用计数也就是src.file->private\_data的引用计数。Src.file->private\_data指向是哪个file？在shiftfs\_open中找到答案。

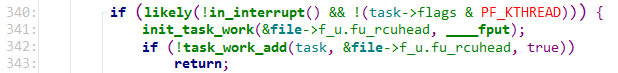


然后shiftfs\_open\_realfile->open\_with\_fake\_path->alloc\_empty\_file\_noaccount->\_\_alloc\_file，\_\_alloc\_file实现如下：



f->f\_count加一。在没有其他额外操作下，可以确定src.file->private\_data的引用计数为一。

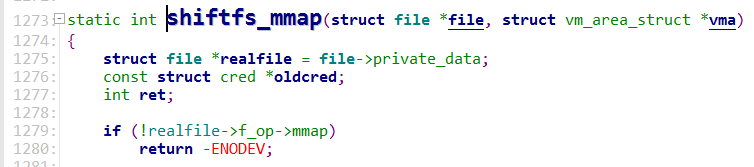
回到fput\_many函数中，判断引用计数为1，



会将释放例程放入task任务队列中，在系统调用结束后，返回到用户层的时候，再进行回调处理，回调函数为\_\_\_fput。

到这里，我们明白，src.file->private\_data被通过newfd释放了，那src.file->private\_data也就是成为了野指针。

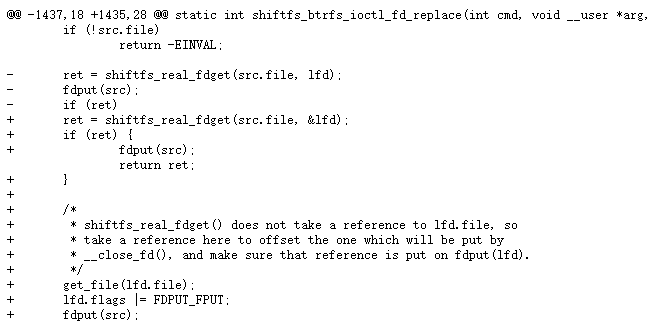
最终如何触发漏洞。Poc中直接通过调用mmap函数，mmap系统调用的接口函数为shiftfs\_mmap函数。



通过file->private\_data获取file。在realfile->f\_op解引用中，发生内存访问错误。

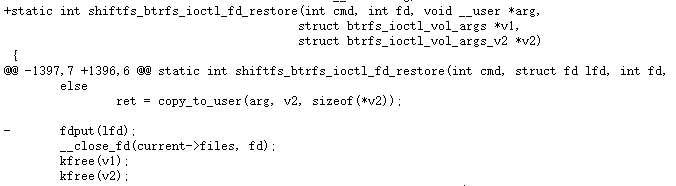
Pacth分析：

对shiftfs\_btrfs\_ioctl\_fd\_replace函数的修改如下



在shiftfs\_real\_fdget返回后，调用get\_file函数给lfd.file加上一次引用计数，平衡\_\_close\_fd。

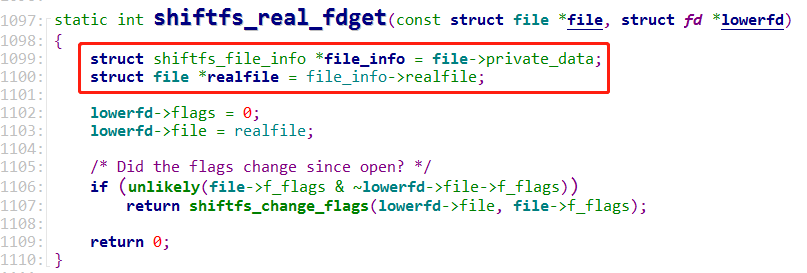
对shiftfs\_btrfs\_ioctl\_fd\_restore的修改如下，



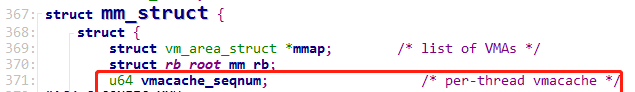
去掉了fdput对lfd的操作，让\_\_close\_fd去平衡引用计数。

2、CVE-2019- 15792是一个类型混淆漏洞。

file-> private\_data是一个void \*，它指向依赖于文件系统的类型；这个file->private\_data是进程私有数据，等同于全局变量的指针，可以通过它进行指针或数据传递。在shiftfs\_real\_fdget函数中，有如下转换。



将file->private\_data空类型指针隐式转换成shiftfs\_file\_info\*类型。接着下面就对file\_info进行解引用，获取->realfile。这里realfile数据域在结构体中偏移0x10，因此进行伪造混淆。Poc中，作者找到mm\_struct中的vmacache\_sequm。



看如何触发漏洞

int main(void) {

// make our vmacache sequence number something like 0x4242

for (int i=0; i<0x4242; i++) {

void \*x = mmap((void\*)0x100000000UL, 0x1000, PROT\_READ,

MAP\_ANONYMOUS|MAP\_PRIVATE, -1, 0);

if (x == MAP\_FAILED) err(1, "mmap vmacache seqnum");

munmap(x, 0x1000);

}

int root = open("mnt/shiftfs", O\_RDONLY);

if (root == -1) err(1, "open shiftfs root");

int foofd = open("/proc/self/environ", O\_RDONLY);

if (foofd == -1) err(1, "open foofd");

// trigger the confusion

struct btrfs\_ioctl\_vol\_args iocarg = {

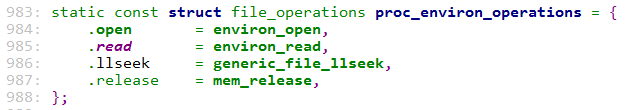
.fd = foofd

};

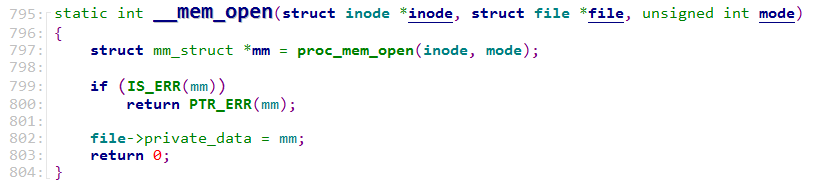
ioctl(root, BTRFS\_IOC\_SNAP\_CREATE, &iocarg);

}

先任意扩大vmacache\_sequm，然后打开/proc/self/environ，会触发environ\_open接口。

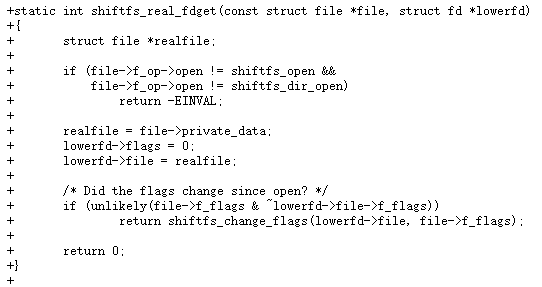


environ\_open函数调用\_\_mem\_open函数。该函数实现如下：



File->private\_data指向mm\_struct。然后将打开的文件描述符foofd通过ioctl传入触发漏洞。

该漏洞patch分析：



添加了对file->f\_op->open的判断。