Documentation for packet dropper LKM
Mahya Jamshidian
9525133

در این مستند، نحوه توسعه دادن و ایجاد یک کرنل ماژول جهت فیلترینگ پکت‌ها شرح داده می‌شود.
اول به بررسی فایل configuration.c میپردازیم. فایلی که نتیجه کامپایل آن در user space اجرا میشود و وظیفه آن خواندن از روی فایل configuration است و ارسال آن به kernel.

```
int fd = open("/dev/packetdrop", O_RDWR);
```

در این فایل ابتدا دیوایس مربوطه را باز میکنیم و خط اول را که مشخص کننده نوع کانفیگ است ارسال می کنیم.
بعد از آن تمام ip:port های داخل فایل نیز ارسال میشود.

برای نوشتن کرنل ماژول برای لینوکس از زبان برنامه نویسی c استفاده کردیم ولی از آنجایی که نوشتن کرنل ماژول ها با برنامه های معمولی متفاوت است از کتابخانه های سطح user نمی توانیم استفاده کنیم و از کتابخانه هایی که خود سیستم عامل برای نوشتن ماژول در اختیار ما میگذارد استفاده میکنیم.

```
#include <linux/init.h>           //
#include <linux/module.h>         //
#include <linux/device.h>         //
#include <linux/kernel.h>         //
#include <linux/fs.h>             //
#include <linux/uaccess.h>// Require
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/skbuff.h>
#include <linux/udp.h>
#include <linux/tcp.h>
#include <linux/icmp.h>
#include <linux/ip.h>
#include <linux/inet.h>
```

کرنل مازول از دو قسمت اصلی init و exit تشکیل شده است که در هنگام لود شدن کرنل ماژول یکبار تابع init و برای حذف این ماژول از کرنل یکبار تابع exit فراخوانی میشود.

```
module_init(packet_drop_init);
module_exit(packet_drop_exit);
```

پس همه تنظیمات مورد نیاز برای ماژول در هنگام ورود و خروج به kernel mode باید در این دو تابع انجام گیرد.
چون قرار است که کانفیگ ها از یک فایل در سطح user خوانده شود و به kernel فرستاده شود باید در ماژول مورد نظر یک دیوایس بسازیم که user از آن طریق با ماژول ارتباط برقرار کند، با توجه به توضیحات فوق باید ساخت device در init و آزاد کردن منابع آن در هنگام خروج در exit انجام گیرد.

برای اینکه user بتواند به device ما اطلاعات بفرستد باید آن را بشناسیم هر device برای شناخته شدن دارای دو عدد major number و minor number است که ترکیب این دو عدد برای هر device خاص است، بعضی از major number ها به device های خاصی اختصاص دارند ولی بعضی از آن ها هم در هنگام boot به device های مورد نیاز به

صورتی داینامیک اختصاص داده می شوند. برای اینکه ماژول ما portable باشد بهتر است در هنگام ساخت device به صورت داینامیک یک عدد از سیستم عامل دریافت کنیم .

```
majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
if (majorNumber<0){
    printk(KERN_ALERT "PACKET failed to register a major number\n");
    return majorNumber;
}
```

در این کد fops یک struct است از نوع file_system_operations که از طریق هدر fs.h در اختیار ما قرار میگیرد چون دیوایس ما دسترسی به خواندن و نوشتن فایل میخواهد ار این struct استفاده میکنیم. توابع مورد نیاز بعدا پیاده سازی شده اند.

```
static int     device_open(struct inode *, struct file *);
static int     device_release(struct inode *, struct file *);

static ssize_t device_write(struct file *, const char *, size_t, loff_t *);
```

چون minor number به اسم device وابسته است لازم نیست برای آن به صورت dynamic عدد بگیریم. در ادامه device , class مورد نظر ساخته شدند.

```
packetdropClass = class_create(THIS_MODULE, CLASS_NAME);
if (IS_ERR(packetdropClass)){                // Check for error and clean up if there is
    unregister_chrdev(majorNumber, DEVICE_NAME);
    printk(KERN_ALERT "Failed to register device class\n");
    return PTR_ERR(packetdropClass);          // Correct way to return an error on a pointer
}
printk(KERN_INFO "PACKET device class registered correctly\n");

// Register the device driver
packetdropDevice = device_create(packetdropClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE_NAME);
printk(KERN_INFO "PACKET device created correctly with major number %d\n", majorNumber);
if (IS_ERR(packetdropDevice)){               // Clean up if there is an error
    class_destroy(packetdropClass);           // Repeated code but the alternative is goto statements
    unregister_chrdev(majorNumber, DEVICE_NAME);
    printk(KERN_ALERT "Failed to create the device\n");
    return PTR_ERR(packetdropDevice);
}
```

برای فیلتر سوکت های عبوری از سیستم ما هم باید یک device برای این کار بسازیم و در قسمت init آن را مشخص کنیم که شروع به کار کند. برای این کار از struct ای که از طریق هدر netfilter.h در اخیار ماست استفاده میکنیم.

```
packetdropNet = nf_register_net_hook(&init_net,&packet_drop); /*Record in net filtering */
printk(KERN_INFO "PACKET network registered correctly with major number \n");
if(packetdropNet) {

    printk(KERN_ALERT"PACKET Faild to register a record in netfilter\n");
    class_destroy(packetdropClass);           // Repeated code but the alternative is goto statements
    unregister_chrdev(majorNumber, DEVICE_NAME);
    device_destroy(packetdropClass, MKDEV(majorNumber, 0));     // remove the device
    nf_unregister_net_hook(&init_net,&packet_drop); /*UnRecord in net filtering */
    return packetdropNet;
}
```

در این استراکت میخواهیم نشان دهیم که فقط سوکت هایی که با پروتوکل ipv4 شناسایی میشوند را برای فیلتر شدن بفرست و اولویت مارا برای دریاف سوکت ها در بالاترین رده قرار بده یعنی سوکت ها در اولین مرحله به این فیلتر می ایند و سپس اگر فیلتر نشدند عبور میکنند.در قسمت init این فیلتر را فعال میکنیم.

```
static struct nf_hook_ops packet_drop __read_mostly = {
        .pf = NFPROTO_IPV4,
        .priority = NF_IP_PRI_FIRST,
        .hooknum =NF_INET_LOCAL_IN,
        .hook = (nf_hookfn *) packet_hook
};
```

در تابع packet_hook که در آن struct معرفی شد source port , source ip پکت ها جدا میشوند (udp, tcp) و براساس mode و حضور داشتن یانداشتن در لیست مورد نظر در مورد فیلتر شدن یا نشدن آن تصمیم گیری میشود.

در نهایت در قسمت exit باید class و device های ساخته شده را از بین برد!