1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Институт кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 2**

1. «Разработка модуля ядра ОС Linux»
2. по дисциплине «Безопасность операционных систем»

Вариант 2

1. Выполнил
2. студент гр. 4851003/70801 Гасанов Э.А.

<*подпись*>

1. Преподаватель
2. ассистент Ахтямов Д.Р.

<*подпись*>

1. Санкт-Петербург
2. 2020

**Цель работы**

Изучить внутреннее устройство ядра ОС Linux и создать собственный модуль ядра. Задание второго варианта - перехватить сетевой трафик и сохранить его в формате pcap

**Задачи**

* Ознакомиться со способом исполнения кода модуля ядра
* Узнать, как загружается и выгружается модуль ядра
* Узнать, где и как логируется информация о событиях в ядре
* Захват трафика
* Запись трафика в текстовый файл из модуля ядра
* Вызов user-mode приложения из модуля ядра для конвертирования текстового файла в файл pcap

# Ход работы

Код в модуле ядра исполняется “событийно” (event), то есть он исполняется как бы “отвечая” на события. Таким образом, например, загружая и выгружая минимальный модуль ядра с помощью *insmod ./module\_name.ko* и *rmmod ./module\_name* соответственно, мы сначала должны как минимум определить макросы module\_init() module\_exit(), определенные в linux/init.h.

Затем нам необходимо следить за тем, как ведет себя модуль ядра. Для этого существует два способа:

* Просмотр syslog: root@ubuntu:/var/log# cat syslog
* Интерактивный просмотр: root@ubuntu:/# tail -f /var/log/kern.log

В полученном варианте необходимо было захватывать трафик и записывать его в pcap. Опишем действия необходимые для этого действия.

# Инициализация модуля ядра

Создадим переменную типа struct packet\_type работающую с пакетами. Эта структура имеет поля для заполнения:

1. type
2. dev
3. func

Первое поле инициализируем с помощью функции htons (HostToNetworkShort), чтобы изменить порядок байтов к сетевому виду. Аргументом этой функции являются значения:

* ETH\_P\_IP 0x0800 - IP-протокол
* ETH\_P\_ARP 0x0806 - ARP-протокол
* ETH\_P\_LOOPBACK 0x9000
* ETH\_P\_ALL 0x0003 - все пакеты
* ETH\_P\_802\_2 0x0004

Второе поле отвечает за то, какой интерфейс прослушивается:

ji\_proto.dev = dev\_get\_by\_name (&init\_net, "enp0s3");

Третье поле – это указатель на функцию, которая логирует захваченный трафик с помощью printk и записывает изменения в тексовый файл.

Затем добавим созданную структуру на стек протоколов модели OSI на 2 уровень с помощью функции dev\_add\_pack (&ji\_proto); для того, чтобы захватывать трафик.

Для того чтобы записывать захваченные пакеты в текстовый файл откроем файл со всеми правами чтения, записи, дозаписи, создания (O\_RDWR | O\_APPEND | O\_CREAT) и выставленными битами S\_IRWXU для доступа к файлу.

# Выгрузка модуля

Закрываем файловый указатель и с помощью функции call\_usermodehelper() вызываем user-mode приложение text2pcap, которое переведет текстовый файл с сохранённым hex-дампом пакетов в формат pcap. Пример вызова:

|  |
| --- |
| char\*argv[]={"/usr/bin/text2pcap","o","hex","/home/elvin/myfile.txt","/home/elvin/outputNEW.pcap",  NULL};  rc = call\_usermodehelper(argv[0], argv, envp, GFP\_ATOMIC); |

А также удаляем структуру со стека протоколов функцией dev\_remove\_pack.

# Примеры запуска

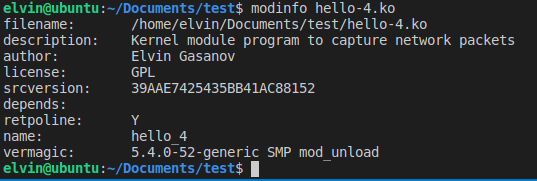


Рисунок 1 - Лицензия

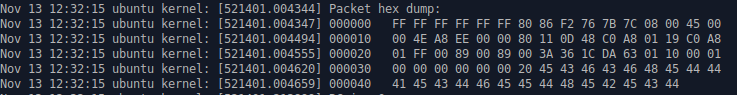


Рисунок 2 – Логирование hex-дампа захваченных пакетов.

На рисунке ниже в текстовый файл записан дамп пакетов. При выгрузке модуля ядра модуль ядра вызвал функцию call\_usermodehelper, которая позволяет вызывать программы из user-mode. Этой функции передали нужные параметры для вызова text2pcap, которая перевела дамп в pcap.

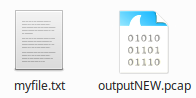


Рисунок 3 – Сохраненный дамп пакетов трафика в текстовом файле переведён в pcap.

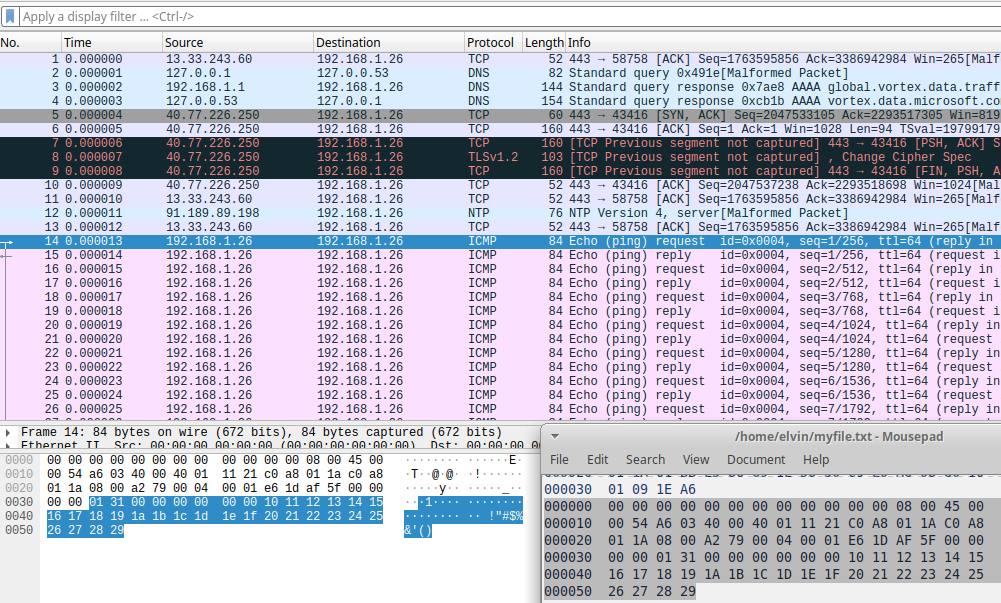


Рисунок 4 – Захваченный трафик уже в pcap формате, читаемый Wireshark.

# Вывод

В данной лабораторной мы познакомились с модулями ядра. Научились работать с файлами в модуле ядра и перехватывать трафик. А также с помощью call\_usermodehelper удаётся вызвать user-mode приложение text2pcap и совершить конвертацию без вмешательства пользователя посредством модуля ядра.

# Листинг

Makefile:

obj-m += hello-4.o

all:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

Модуль ядра:

#include <linux/module.h> // included for all kernel modules

#include <linux/kernel.h> // included for KERN\_INFO

#include <linux/init.h> // included for \_\_init and \_\_exit macros

#include <linux/skbuff.h> // included for struct sk\_buff

#include <linux/if\_packet.h> // include for packet info

#include <linux/ip.h> // include for ip\_hdr

#include <linux/netdevice.h> // include for dev\_add/remove\_pack

#include <linux/if\_ether.h> // include for ETH\_P\_ALL

#include <linux/unistd.h>

//#include<linux/printk.h>

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("Elvin Gasanov");

MODULE\_DESCRIPTION("Kernel module program to capture network packets");

static struct file \*filp = NULL;

struct packet\_type ji\_proto;

struct file \*file\_open(const char \*path, int flags, int rights)

{

//struct file \*filp = NULL;

mm\_segment\_t oldfs;

int err = 0;

oldfs = get\_fs();

set\_fs(KERNEL\_DS);

filp = filp\_open(path, flags, rights);

set\_fs(oldfs);

if (IS\_ERR(filp)) {

err = PTR\_ERR(filp);

return NULL;

}

printk("File opened to write");

return filp;

}

unsigned long long file\_write(unsigned long long offset, unsigned char \*data, unsigned int size)

{

mm\_segment\_t oldfs;

int ret;

oldfs = get\_fs();

set\_fs(KERNEL\_DS);

ret = vfs\_write(filp, data, size, &offset);

set\_fs(oldfs);

return offset;

}

void pkt\_hex\_dump(struct sk\_buff \*skb)

{

char buffer\_mass[512]="\0";

unsigned long long offset=0;

size\_t len;

int rowsize = 16;

int i, l, linelen, remaining;

int li = 0;

uint8\_t \*data, ch;

printk("Packet hex dump:\n");

data = (uint8\_t \*) skb\_mac\_header(skb);

if (skb\_is\_nonlinear(skb)) {

len = skb->data\_len;

} else {

len = skb->len;

}

remaining = len;

for (i = 0; i < len; i += rowsize)

{

printk("%06d\t", li);

sprintf(buffer\_mass,"%06d\t",li);

offset=file\_write(offset,buffer\_mass,strlen(buffer\_mass));

linelen = min(remaining, rowsize);

remaining -= rowsize;

for (l = 0; l < linelen; l++) {

ch = data[l];

memset(buffer\_mass,'\0',512);

printk(KERN\_CONT "%02X ", (uint32\_t) ch);

sprintf(buffer\_mass,"%02X ",(uint32\_t) ch);

offset=file\_write(offset,buffer\_mass,strlen(buffer\_mass));

}

data += linelen;

li += 10;

memset(buffer\_mass,'\0',512);

printk(KERN\_CONT "\n");

sprintf(buffer\_mass,"\n");

offset=file\_write(offset,buffer\_mass,strlen(buffer\_mass));

}

}

int ji\_packet\_rcv (struct sk\_buff \*skb, struct net\_device \*dev,struct packet\_type \*pt, struct net\_device \*orig\_dev)

{

printk(KERN\_INFO "New packet captured.\n");

/\* linux/if\_packet.h : Packet types \*/

// #define PACKET\_HOST 0 /\* To us \*/

// #define PACKET\_BROADCAST 1 /\* To all \*/

// #define PACKET\_MULTICAST 2 /\* To group \*/

// #define PACKET\_OTHERHOST 3 /\* To someone else \*/

// #define PACKET\_OUTGOING 4 /\* Outgoing of any type \*/

// #define PACKET\_LOOPBACK 5 /\* MC/BRD frame looped back \*/

// #define PACKET\_USER 6 /\* To user space \*/

// #define PACKET\_KERNEL 7 /\* To kernel space \*/

/\* Unused, PACKET\_FASTROUTE and PACKET\_LOOPBACK are invisible to user space \*/

// #define PACKET\_FASTROUTE 6 /\* Fastrouted frame \*/

switch (skb->pkt\_type)

{

case PACKET\_HOST:

//printk(KERN\_INFO "@JI : PACKET to us âˆ’ ");

break;

case PACKET\_BROADCAST:

//printk(KERN\_INFO "@JI : PACKET to all âˆ’ ");

break;

case PACKET\_MULTICAST:

//printk(KERN\_INFO "@JI : PACKET to group âˆ’ ");

break;

case PACKET\_OTHERHOST:

//printk(KERN\_INFO "@JI : PACKET to someone else âˆ’ ");

break;

case PACKET\_OUTGOING:

//printk(KERN\_INFO "@JI : PACKET outgoing âˆ’ ");

break;

case PACKET\_LOOPBACK:

//printk(KERN\_INFO "@JI : PACKET LOOPBACK âˆ’ ");

break;

case PACKET\_FASTROUTE:

//printk(KERN\_INFO "@JI : PACKET FASTROUTE âˆ’ ");

break;

}

//printk(KERN\_CONT "Dev: %s ; 0x%.4X ; 0x%.4X \n", skb->dev->name, ntohs(skb->protocol), ip\_hdr(skb)->protocol);

struct ethhdr \*ether = eth\_hdr(skb);

//struct file \*myfile=file\_open("/home/elvin/myfile.txt",O\_RDWR|O\_APPEND|O\_CREAT,S\_IRWXU);

//printk("Source: %x:%x:%x:%x:%x:%x\n", ether->h\_source[0], ether->h\_source[1], ether->h\_source[2], ether->h\_source[3], ether->h\_source[4], ether->h\_source[5]);

//printk("Destination: %x:%x:%x:%x:%x:%x\n", ether->h\_dest[0], ether->h\_dest[1], ether->h\_dest[2], ether->h\_dest[3], ether->h\_dest[4], ether->h\_dest[5]);

//printk("Protocol: %d\n", ether->h\_proto);

pkt\_hex\_dump(skb);

kfree\_skb (skb);

return 0;

}

static int \_\_init

ji\_init(void)

{

/\* See the <linux/if\_ether.h>

When protocol is set to htons(ETH\_P\_ALL), then all protocols are received.

All incoming packets of that protocol type will be passed to the packet

socket before they are passed to the protocols implemented in the kernel. \*/

/\* Few examples \*/

//ETH\_P\_LOOP 0x0060 /\* Ethernet Loopback packet \*/

//ETH\_P\_IP 0x0800 /\* Internet Protocol packet \*/

//ETH\_P\_ARP 0x0806 /\* Address Resolution packet \*/

//ETH\_P\_LOOPBACK 0x9000 /\* Ethernet loopback packet, per IEEE 802.3 \*/

//ETH\_P\_ALL 0x0003 /\* Every packet (be careful!!!) \*/

//ETH\_P\_802\_2 0x0004 /\* 802.2 frames \*/

//ETH\_P\_SNAP 0x0005 /\* Internal only \*/

ji\_proto.type = htons(ETH\_P\_IP);

/\* NULL is a wildcard \*/

//ji\_proto.dev = NULL;

ji\_proto.dev = dev\_get\_by\_name (&init\_net, "enp0s3");

ji\_proto.func = ji\_packet\_rcv;

/\* Packet sockets are used to receive or send raw packets at the device

driver (OSI Layer 2) level. They allow the user to implement

protocol modules in user space on top of the physical layer. \*/

/\* Add a protocol handler to the networking stack.

The passed packet\_type is linked into kernel lists and may not be freed until

it has been removed from the kernel lists. \*/

dev\_add\_pack (&ji\_proto);

struct file \*myfile=file\_open("/home/elvin/myfile.txt",O\_RDWR|O\_APPEND|O\_CREAT,S\_IRWXU);

printk(KERN\_INFO "Module insertion completed successfully!\n");

return 0; // Non-zero return means that the module couldn't be loaded.

}

static void \_\_exit

ji\_cleanup(void)

{

filp\_close(filp, NULL);

int rc=0;

static char \*envp[] = {

"SHELL=/bin/bash",

"HOME=/home/tester",

"USER=tester",

"PATH=/usr/bin/",

"DISPLAY=:0",

"PWD=/home/tester",

NULL};

//static char\* envp[]={NULL};

char \*argv[] = {"/usr/bin/text2pcap", "-o","hex","/home/elvin/myfile.txt","/home/elvin/outputNEW.pcap",NULL};

rc = call\_usermodehelper(argv[0], argv, envp, GFP\_ATOMIC);

printk("RC is: %i \n", rc);

dev\_remove\_pack(&ji\_proto);

printk(KERN\_INFO "Cleaning up module....\n");

}

module\_init(ji\_init);

module\_exit(ji\_cleanup);