



湖南大学

HUNAN UNIVERSITY

操作系统实验报告

课 程 名 称： 操作系统

实验项目名称： 操作系统内核编程实验

专 业 班 级： 软件 2203

姓 名： 白旭

学 号： 202226010306

指 导 教 师： 周军海

完 成 时 间： 2024 年 5 月 16 日

信息科学与工程学院

实验题目：实验五 内核网络管理

实验目的：

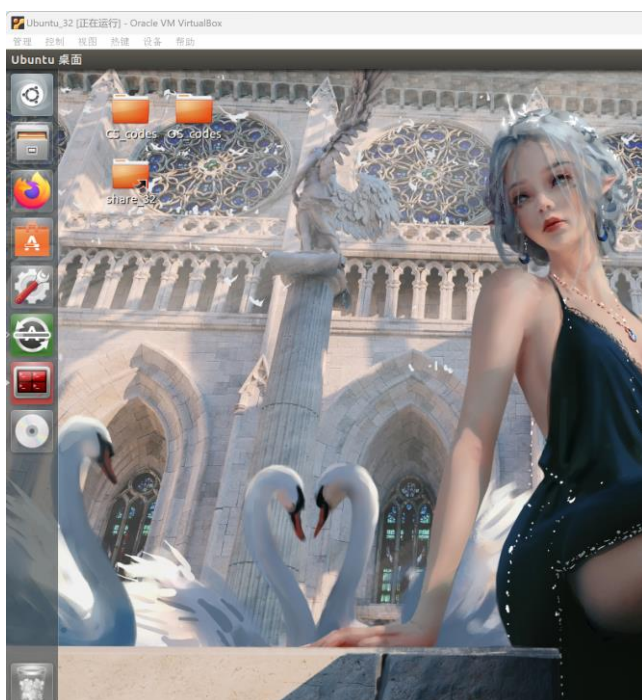
- 学习了解 XDP 与 eBPF
- 编写 eBPF 过滤程序、编译 eBPF 过滤程序，并将过滤程序应用到网卡中
- 思考题：
Linux 高级网络栈架构通常包括哪些层？

实验环境：

- Ubuntu 20.04

实验内容及操作步骤：

1. 打开 Ubuntu 20.04:



2. 使用 `lshw -C network` 查看网口命名：

```
baijue@baijue-VirtualBox:~$ lshw -C network
WARNING: you should run this program as super-user.
*-network
   description: Ethernet interface
   product: 82540EM Gigabit Ethernet Controller
   vendor: Intel Corporation
   physical id: 3
   bus info: pci@0000:00:03.0
   logical name: enp0s3
   version: 02
   serial: 08:00:27:e5:fe:53
   size: 1Gbit/s
   capacity: 1Gbit/s
   width: 32 bits
   clock: 66MHz
   capabilities: bus_master cap_list ethernet physical tp 10bt 10bt-fd 100bt
100bt-fd 1000bt-fd autonegotiation
   configuration: autonegotiation=on broadcast=yes driver=e1000 driverversion=7.3.21-k8-NAPI duplex=full ip=10.0.2.15 latency=64 link=yes mingnt=255 multica
st=yes port=twisted pair speed=1Gbit/s
   resources: irq:19 memory:f0200000-f021ffff ioport:d020(size=8)
WARNING: output may be incomplete or inaccurate, you should run this program as
super-user.
```

得知网口命名为 enp0s3

3. 编写代码

1) enp0s3.c

禁止所有数据包

```
#include <linux/bpf.h>
#include <linux/if_ether.h>
#include <linux/ip.h>
#include <linux/in.h>

// 将后面的函数或变量放到特定的段(section)中
// __attribute__((section(NAME), used))是GCC编译器的一个属性
// 表示将函数或变量放到名为NAME的段中, 并且标记为已使用, 防止编译器优化时将其删除
#define __section(NAME) \
    __attribute__((section(NAME), used))

// 这部分代码定义了一个BPF程序, 放在名为"prog"的段中
// 函数drop_all接收一个参数struct xdp_md *ctx
// 这是XDP (eXpress Data Path) 程序的上下文, 包含了数据包的信息
__section("prog") int drop_all(struct xdp_md *ctx)
{
    // 函数的返回值XDP_DROP表示将数据包丢弃
    return XDP_DROP;
}

// 许可证是GPL
char __license[] __section("license") = "GPL";
```

其中 struct xdp_md 的实现

```
struct xdp_md {
    __u32 data; // 指向数据包的起点
    __u32 data_end; // 指向数据包的末尾
    __u32 data_meta; // 数据包元数据
    __u32 ingress_ifindex; // rxq->dev->ifindex 网卡的序号, ip link显示的那个
    __u32 rx_queue_index; // rxq->queue_index 网卡接收队列的序号

    __u32 egress_ifindex; // txq->dev->ifindex 这个参数旧一些的内核是没有的
};
```

2) enp0s3_1.c

禁止所有的 TCP 数据包

以太网头部结构体 ethhdr:

```
#define ETH_ALEN 6 // 定义了以太网接口的MAC地址的长度为6个字节
#define ETH_HLEN 14 // 定义了以太网帧的头长度为14个字节
#define ETH_ZLEN 60 // 定义了以太网帧的最小长度为 ETH_ZLEN + ETH_FCS_LEN = 64个字节
#define ETH_DATA_LEN 1500 // 定义了以太网帧的最大负载为1500个字节
#define ETH_FRAME_LEN 1514 // 定义了以太网帧的最大长度为ETH_DATA_LEN + ETH_FCS_LEN = 1518个字节
#define ETH_FCS_LEN 4 // 定义了以太网帧的CRC值占4个字节

struct ethhdr
{
    unsigned char h_dest[ETH_ALEN]; // 目的MAC地址
    unsigned char h_source[ETH_ALEN]; // 源MAC地址
    __u16 h_proto; // 网络层所使用的协议类型
} __attribute__((packed)) // 用于告诉编译器不要对这个结构体中的缝隙部分进行填充操作;
```

ip 头部结构体:

```
struct iphdr {
    #if defined(__LITTLE_ENDIAN_BITFIELD)
        __u8    ihl:4,           // 首部长度(4位):首部长度指的是IP层头部包含多少个4字节
        version:4; // 版本(4位)。目前的协议版本号是4, 因此IP有时也称作IPv4
    #elif defined(__BIG_ENDIAN_BITFIELD)
        __u8    version:4,
        ihl:4;
    #else
        #error "Please fix <asm/byteorder.h>"
    #endif
    __u8    tos;
    __be16  -tot_len; // 总长度字段(16位)是指整个IP数据报的长度
    __be16  -id;
    __be16  -frag_off;
    __u8    ttl;
    __u8    protocol;
    __be16  -check;
    __be32  -s_addr;
    __be32  -d_addr;
};
```

```
#include <linux/bpf.h>
#include <linux/in.h>
#include <linux/if_ether.h>
#include <linux/ip.h>

#define __section(NAME) \
    __attribute__((section(NAME), used))

// struct xdp_md *ctx 是XDP程序的上下文参数, 包含了数据包的信息
__section("prog") int drop_tcp(struct xdp_md *ctx)
{
    // 获取数据包的起始位置
    void *data = (void *) (long) ctx->data;
    // 获取数据包的结束位置
    void *data_end = (void *) (long) ctx->data_end;

    // 检查以太网头部是否在数据包内
    if (data + sizeof(struct ethhdr) > data_end)
    {
        // XDP_ABORTED 是 XDP 程序返回的一种特殊值, 表示程序由于某种错误而终止处理数据包
        return XDP_ABORTED;
    }

    // 定义一个指向IP头部的指针
    struct iphdr *ip;
    // struct ethhdr 结构体来表示以太网帧的头部
    // 将指针 ip 移动到以太网头之后的位置, 这里是IP头的开始位置
    ip = data + sizeof(struct ethhdr);

    // 检查IP头部是否在数据包内
    if ((void *) (ip + 1) > data_end)
    {
        return XDP_ABORTED;
    }

    // 检查IP头部中的协议字段是否为TCP协议
    if (ip->protocol == IPPROTO_TCP)
    {
        // 如果是TCP数据包, 返回 XDP_drop, 表示丢弃数据包
        return XDP_DROP;
    }
    // 如果不是TCP数据包, 返回 XDP_PASS, 表示允许数据包通过
    return XDP_PASS;
}

char _license[] __section("license") = "GPL";
```

4. 模块功能验证

1) enp0s3

```
20:55:45.343731 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:55:46.373601 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:55:47.387336 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:55:48.410643 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:55:49.446195 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:55:55.584012 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:56:02.748509 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:56:03.771914 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:56:12.990263 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:56:14.010614 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:56:15.037321 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:56:16.059058 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:56:17.083239 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:56:18.108969 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
20:56:19.130745 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
^C
45 packets captured
64 packets received by filter
19 packets dropped by kernel
baijue@baijue-VirtualBox:~/OS_5VS$
```

除了 ARP 包之外的包都被丢弃

2) enp0s3_1

```
7016497 ecr 0,nop,wscale 7], length 0
20:58:44.322694 IP baijue-VirtualBox.49820 > ubuntu-content-cache-3.ps6.canonical.
com.http: Flags [S], seq 796166945, win 64240, options [mss 1460,sackOK,TS val 387
1564475 ecr 0,nop,wscale 7], length 0
20:58:46.018900 IP baijue-VirtualBox.57010 > ubuntu-content-cache-2.ps6.canonical.
com.http: Flags [S], seq 603486234, win 64240, options [mss 1460,sackOK,TS val 403
7018517 ecr 0,nop,wscale 7], length 0
20:58:46.362951 IP baijue-VirtualBox.49830 > ubuntu-content-cache-3.ps6.canonical.
com.http: Flags [S], seq 1764881023, win 64240, options [mss 1460,sackOK,TS val 38
71566515 ecr 0,nop,wscale 7], length 0
20:58:50.219187 IP baijue-VirtualBox.55776 > ubuntu-content-cache-3.ps6.canonical.
com.http: Flags [S], seq 155564631, win 64240, options [mss 1460,sackOK,TS val 387
1570372 ecr 0,nop,wscale 7], length 0
20:58:50.222895 IP baijue-VirtualBox.57010 > ubuntu-content-cache-2.ps6.canonical.
com.http: Flags [S], seq 603486234, win 64240, options [mss 1460,sackOK,TS val 403
7022721 ecr 0,nop,wscale 7], length 0
20:58:51.226769 ARP, Request who-has _gateway tell baijue-VirtualBox, length 28
20:58:51.227081 ARP, Reply _gateway is-at 52:54:00:12:35:02 (oui Unknown), length
46
^C
59 packets captured
59 packets received by filter
0 packets dropped by kernel
baijue@baijue-VirtualBox:~/OS_5VS$
```

所有 TCP 包都被丢弃

实验结果及分析:

1. 实验结果与预期相符, 第五次实验学习了 XDP 与 eBPF 的知识, 完成得比较顺利, 源代码可以正常运行

收获与体会:

1. 应全部使用 sudo 来提升权限, 否则会报错
2. 做完实验要及时将 ESC 关机以节约经费
3. 掌握了如何编写 eBPF 程序

思考题:

1. Linux 高级网络栈架构通常包括哪些层?

TCP/IP五层网络模型:

应用层: 概念同上, 一般在客户端的应用程序, 例如访问web的HTTP协议, 域名转换协议DNS等, 类似于客户的数据

传输层: TCP提供面向连接的、有状态的数据流传输, UDP是数据报传输面向无连接无状态

网络层: 提供IP选路, 在这一层往上的数据都叫做数据包

数据链路层: 这一层在路由器中数据广域网或二者局域网的接入方式, 例如选择以太、帧中继或者ATM等。这一层的数据成为帧。

物理层: 中继器, 集线器等, 是具体数据传输的媒介, 数据在这一层是以比特流的形式传输。

实验 成绩	
----------	--