防火墙 II——利用 netfilter 构建用户级防火墙

2012-12

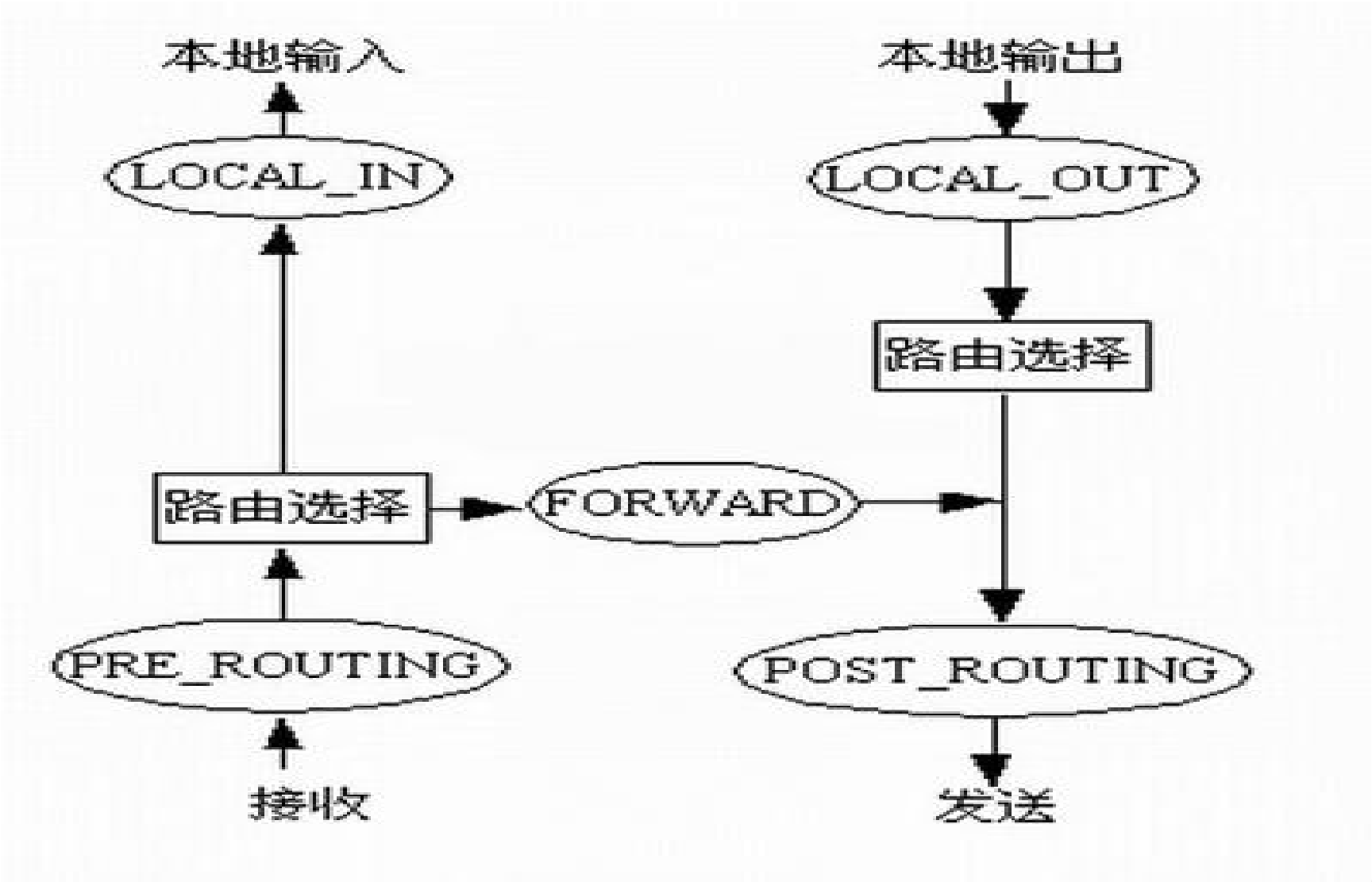
课程内容

- Netfilter 概述
- Netfilter 工作机制
- 利用 netfilter 构建用户级防火墙原理
- Libipq 工作流程与相关函数
- 实验题目

Netfilter 概述

Netfilter 更准确地讲是 Linux 内核中，一个包过滤框架，默认地，它在这个框架上实现了包过滤、状态检测、网络地址转换和包标记等多种功能，因为它设计的开放性，任何有内核开发经验的开发人员，也可以很容易地利用它提供接口，在内核的数据链路层、网络层，实现自己的功能模块。

Netfilter 工作机制



Netfilter 的五种操作

- NF_DROP 丢弃该报文，释放所有与该报文相关的资源；
- NF_ACCEPT 接受该报文，并继续处理；
- NF_STOLEN 该报文已经被 HOOK 函数接管，协议栈无须继续处理；
- NF_QUEUE 将该报文传递到用户态去做进一步的处理；
- NF_REPEAT 再次调用本 HOOK 函数。

利用 netfilter 构建防火墙

- 在内核中注册自己的钩子函数;
- 利用 IP_QUEUE 实现内核与用户层之间的数据包交换，实现用户态防火墙

利用 netfilter 构建用户态防火墙机制

当 HOOK 处理函数返回 NF_QUEUE 值时，内核协议栈将通过 Linux NetLink 通信机制把当前报文传递到用户态，由用户态的防火墙程序进行处理。这样，只要能够在相应的 HOOK 点上返回 NF_QUEUE 值，就可以安心地在用户态使用自己的程序来过滤报文了，这个功能可以由 iptables 实现。

Libipq 简介

Libipq 是 NetFilter 框架的重要组成部分。任何时候在任何 NetFilter 规则链中，数据报都可以被排队转发到用户空间去。用户进程能对数据报进行任何处理。处理结束以后，用户进程可以将该数据报重新注入内核或者设置一个对数据报的目标动作。

开发流程

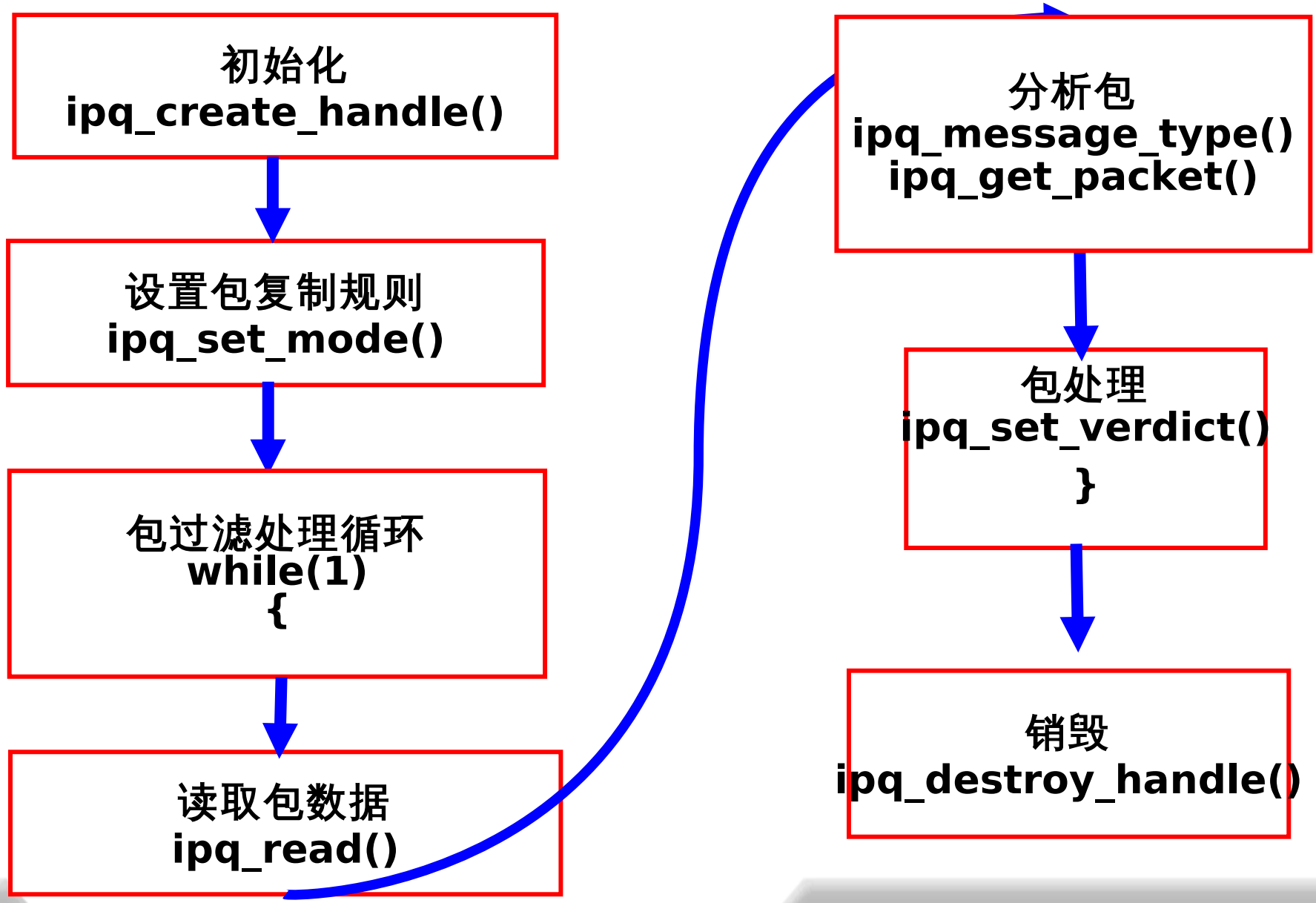
1. 设置过滤环境
2. 编写应用程序

设置过滤环境

1. 加载过滤需要的内核模块
   - # modprobe iptable_filter
   - # modprobe ip_queue

1. 设置过滤规则
   - iptables -A OUTPUT -p icmp -j QUEUE
   - Iptables -A OUTPUT –p tcp –j QUEUE

Libipq 程序框架


初始化
ipq_create_handle()
设置包复制规则
ipq_set_mode()
包过滤处理循环
while(1)
{
读取包数据
ipq_read()
分析包
ipq_message_type()
ipq_get_packet()
包处理
ipq_set_verdict()
}
销毁
ipq_destroy_handle()

Libipq 接口函数（1）

建立 ipq 的 handle:

struct ipq_handle *ipq_create_handle(u_int32_t flags, u_int32_t protocol);

flags: 基本上没用，通常设为 0 ；

protocol: 制定想获取协议的队列， PF_INET 为 IPV4 队列，PF_INET6 为 IPV6 队列

成功：返回一个不为空的指针；

失败：返回一个 NULL 指针

元数据

```
struct nlmsghdr
{
 __u32  nlmsg_len; /* Length of message including header */
 __u16  nlmsg_type; /* Message content */
 __u16  nlmsg_flags; /* Additional flags */
 __u32  nlmsg_seq; /* Sequence number */
 __u32  nlmsg_pid; /* Sending process PID */
};
```

Libipq 接口函数（2）

设置 IPQ 的拷贝模式：用户空间来设置 ip_queue 的接收数据模式的 .

int ipq_set_mode(const struct ipq_handle *h, u_int8_t mode, size_t len)

h: 是通过 *ipq_create_handle ()* 获得的句柄指针;

mode: 设定拷贝模式， IPQ_COPY_META 和 IPQ_COPY_PACKET 。当为 IPQ_COPY_META 时，内核将在其后的报文传递中只传递“报文的元数据”；当为 IPQ_COPY_PACKET 时，内核将同时传递“报文的元数据”和报文本身

len: 请求拷贝的报文长度

成功：返回一个非 0 的正数；

失败：返回 -1 ；

Libipq 接口函数（3）

从内核中的 packet queue 中读取数据包，将其拷贝到 buf 指定的缓冲区

ssize_t ipq_read(const struct ipq_handle *h,unsigned char *buf, size_t len, int timeout);

h:ipq_create_handle 创建的句柄；

buf: 存放数据包的缓冲区；

len: 拷贝的字节数；

timeout: 超时时限， 0 表示函数直到指定的数据读到缓冲区里接触阻塞，负数，表示函数马上接触阻塞。

成功：返回一个大于 0 的数；

失败： -1

Libipq 接口函数（4）

分析数据包的类型

int ipq_message_type(const unsigned char *buf);

1 、 buf: 通过 ipq_read 存放数据包的缓冲区;

返回值存在两种可能:

NLMSG_ERROR： 数据包是一个错误的数据包;

IPQM_PACKET: 元数据或是既包含元数据和负载的数据包

Libipq 接口函数（5）

从缓冲中获取数据包

ipq_packet_msg_t *ipq_get_packet(const unsigned char *buf);

```
typedef struct ipq_packet_msg {
    unsigned long packet_id;        /* ID of queued packet */
    unsigned long mark;             /* Netfilter mark value */
    long timestamp_sec;             /* Packet arrival time (seconds) */
    long timestamp_usec;            /* Packet arrvial time (+useconds) */
    unsigned int hook;              /* Netfilter hook we rode in on */
    char indev_name[IFNAMSIZ];      /* Name of incoming interface */
    char outdev_name[IFNAMSIZ];     /* Name of outgoing interface */
    unsigned short hw_protocol;     /* Hardware protocol (network order) */
    unsigned short hw_type;         /* Hardware type */
    unsigned char hw_addrlen;       /* Hardware address length */
    unsigned char hw_addr[8];       /* Hardware address */
    size_t data_len;                /* Length of packet data */
    unsigned char payload[0];       /* Optional packet data */
} ipq_packet_msg_t;
```

Libipq 接口函数（5）

用户层通过这个函数告诉内核对某一个数据包的处理意见

int ipq_set_verdict(const struct ipq_handle *h,ipq_id_t id,unsigned int verdict,size_t data_len,unsigned char *buf);

h:

id: 数据包的标识符，通过 ipq_get_packet 得到的；

verdict: 对数据包的处理意见， NF_ACCEPT: 接收数据包， NF_DROP: 丢弃数据包；

len:

buf:

成功： >0 的正数；

失败： -1

Libipq 接口函数（6）

释放 ipq 句柄

int ipq_destroy_handle(struct ipq_handle *h);

成功： 0
失败： -1

用 libipq 编写程序

1. 头文件
   - **#include <linux/netfilter.h>**
   - **#include <libipq.h>**
2. 编译、链接
   - **gcc -o myfw myfw.c –lipq**
3. 输出
   - **fprintf(stderr,…..)**

编译环境的搭建

Ubuntu：

```
sudo apt-get install iptables-dev
```

Redhat 9.0:

```
# file  iptables-1.2.7a.tar
# tar xjvf iptables-1.2.7a.tar
# make instll-devel
```

Redhat 高版本系列:

1 、 rpm –ivh iptables 1.XX-dev(在安装盘里能找到安装包) ;
2 、 yum install iptables-dev;

试验题目

1. 设置 iptables 过滤规则为：所有从本机发出的 icmp 包全部到自己编写的应用程序
2. 编写应用程序，功能如下：
    1. 允许从本机出发，目的地址为 win xp ip 的 icmp 包；
    2. 丢弃其他任何 icmp 包；
    3. 当出现错误时，做错误处理，能够清理占用资源，退出程序。