**实验二：iptables和netfilter**

linux的防火墙由netfilter和iptables组成。用户空间的iptables制定防火墙规则，内核空间的netfilter实现防火墙功能。netfilter（内核空间）位于Linux内核中的包过滤防火墙功能体系，称为Linux防火墙的“内核态”。iptables(用户空间)位于/sbin/iptables，是用来管理防火墙的命令的工具，为防火墙体系提供过滤规则/策略，决定如何过滤或处理到达防火墙主机的数据包，称为Linux防火墙的“用户态”。

**实验内容**

1. 使用iptables命令实现代理。需要3台虚拟机进行演示。
2. 基于netfilter，对http报文实现包过滤，并窃取http报文的用户名和密码。

**实验环境：**

本文档描述的实验环境是Ubuntu 16.04，其他环境可自行探索。

### 使用iptables命令实现代理

实验需要同一网络下的三台虚拟机，实现一个代理服务器（下列ip仅用于实验说明）：

| IP | 角色 |
| --- | --- |
| 192.168.107.130 | 用户 |
| 192.168.107.131 | 内部服务器 |
| 192.168.107.132 | 代理服务器 |

实现代理，即需要用代理服务器将用户与内部服务器隔离起来，当用户向内部服务器申请资源时，请求报文的目的ip是代理服务器的ip（192.168.107.132），到达代理服务器后，在路由选择之前进行目标地址转换，将目的地址转换为内部服务器的地址（192.168.107.131）。同样，在内部服务器回复请求信息时，响应报文到达代理服务器之后，需要在路由选择之后，将源ip地址（192.168.107.131）转换为代理服务器的ip地址（192.168.107.132）。

### 实验步骤

1．首先在代理机上开启转发数据包的能力：

Sudocd /proc/sys/net/ipv4

echo 1 >ip\_forward

1. 使用Iptables命令对nat表进行配置，查看实验效果。

Iptables命令学习：https://www.cnblogs.com/frankb/p/7427944.html

### 使用netfilter实现包过滤

Netfilter在五个点拦截报文，每个拦截点对应iptable的一个chain

1. NF\_BR\_PRE\_ROUTING: 在报文路由前进行对报文的拦截
2. NF\_INET\_LOCAL\_IN:对到本机的报文进行拦截
3. NF\_BR\_FORWARD:对需要本机进行三层转发的报文进行拦截
4. NF\_INET\_LOCAL\_OUT:对本机生成的报文进行拦截
5. NF\_BR\_POST\_ROUTING:路由后对报文进行拦截

实验要求：

1. 使用wireshark抓包，分析http报文结构。
2. 根据url过滤报文，使得无法下载.exe后缀的文件。
3. 截取本虚拟机发出的http报文，判断是否有用户名和密码，若有则将其使用printk打印出来。

参考：

Netfilter.c实现了过滤tcp报文的功能，Makefile编译并使用LKM加载模块，效果是加载模块后无法上网。

Netfilter.c

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/skbuff.h>

#include <linux/in.h>

#include <linux/ip.h>

#include <linux/tcp.h>

#include <linux/icmp.h>

#include <linux/netdevice.h>

#include <linux/netfilter.h>

#include <linux/netfilter\_ipv4.h>

#include <linux/if\_arp.h>

#include <linux/if\_ether.h>

#include <linux/if\_packet.h>

MODULE\_LICENSE("GPL");

/\* 用于描述我们的Netfilter挂钩

\* nf\_hook\_ops数据结构在linux/netfilter.h中定义

\* 我们定义两个nf\_hook\_ops结构体，一个传入的hook 和 一个传出的hook

struct nf\_hook\_ops

{

struct list\_head list; //钩子链表

nf\_hookfn \*hook; //钩子处理函数

struct module \*owner; //模块所有者

int pf; //钩子协议族

int hooknum; //钩子的位置值（PREROUTING、POSTOUTING、INPUT、FORWARD、OUTPUT五个位置）

int priority; //钩子的的优先级

}

\*/

struct nf\_hook\_opspost\_hook;

static unsigned int watch\_out(void \*priv, struct sk\_buff \*skb, const struct nf\_hook\_state \*state)

{

/\* 让传入的缓冲skb存到sock\_buff中 \*/

struct sk\_buff \*sock\_buff = skb;

if(!sock\_buff){

printk("buffer error!\n");

return NF\_ACCEPT;

}

//丢弃tcp报文

if(ip\_hdr(sock\_buff)->protocol == 6){

return NF\_DROP;

}

return NF\_ACCEPT;

}

/\*

内核模块中的两个函数init\_module() ：表示起始 和 cleanup\_module() ：表示结束

\*/

int init\_module()

{

/\*hook函数指针指向watc\_out\*/

post\_hook.hook = watch\_out;

/\*协议簇为ipv4\*/

post\_hook.pf = PF\_INET;

/\*优先级最高\*/

post\_hook.priority = NF\_IP\_PRI\_FIRST;

post\_hook.hooknum = NF\_INET\_POST\_ROUTING;

/\*将post\_hook注册，注册实际上就是在一个nf\_hook\_ops链表中再插入一个nf\_hook\_ops结构\*/

nf\_register\_net\_hook(&init\_net ,&post\_hook);

return 0;

}

void cleanup\_module()

{

/\*将post\_hook取消注册，取消注册实际上就是在一个nf\_hook\_ops链表中删除一个nf\_hook\_ops结构\*/

nf\_unregister\_net\_hook(&init\_net ,&post\_hook);

}

Makefile：

obj-m +=netfilter.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

注意：若报miss separator错误，检查make前面是否是tab键空格

在包过滤的虚拟机上执行命令：

make

sudo insmod netfilter.ko

其他相关命令

查看内核模块：lsmod

卸载内核模块：rmmod