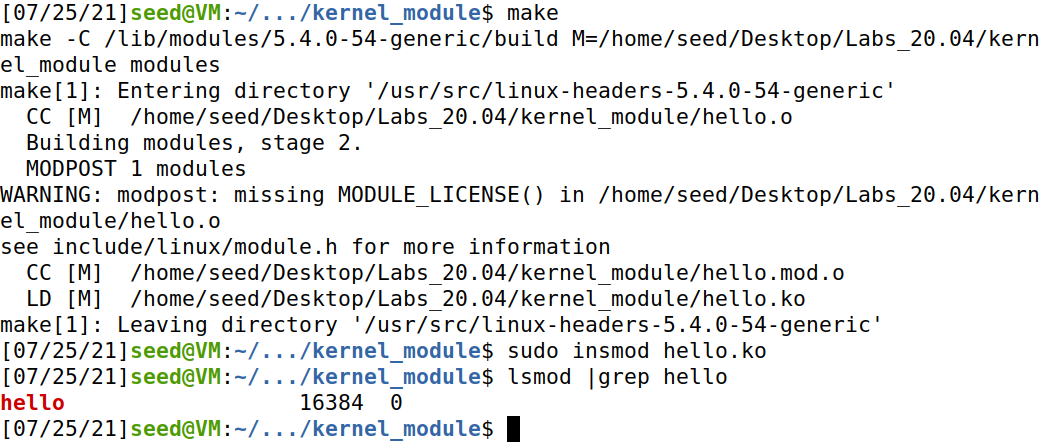
Lab6

57118111王雯

**Task1：Implementing a Simple Firewall**

**Task 1.A: Implement a Simple Kernel Module**

将kernel\_module这个文件夹移动到一个没有空格的目录下，然后直接make编译



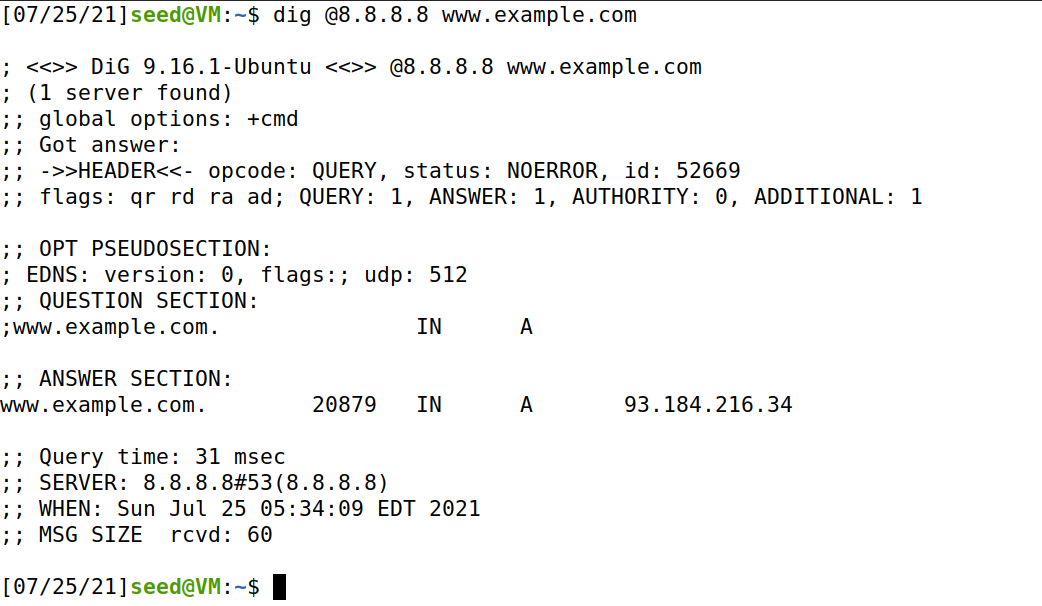
在将其文件夹移除，恢复原始的环境，通过dmesg可查看日志看见其输出：



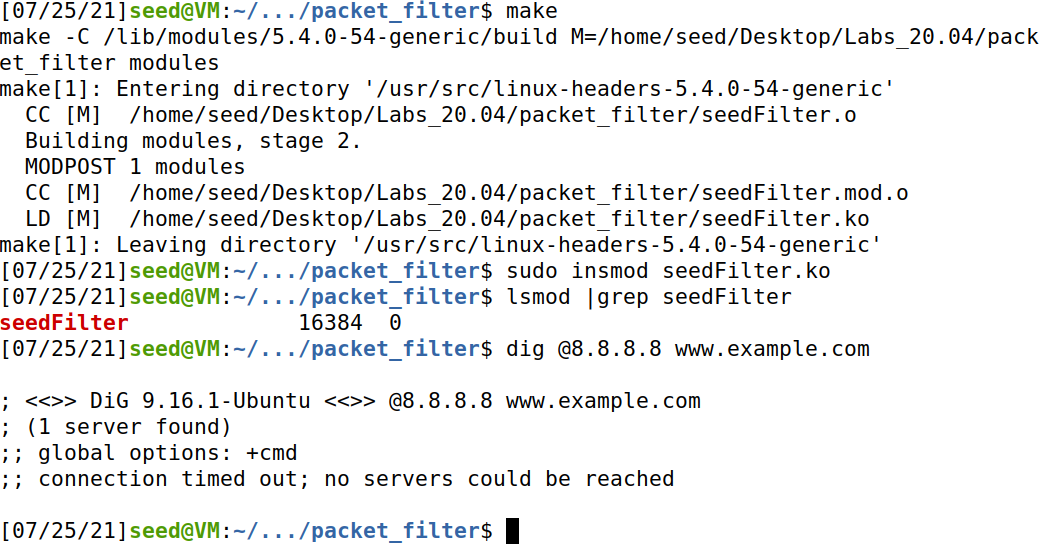
**Task 1.B: Implement a Simple Firewall Using Netfilter**

**1、 Compile the sample code using the provided Makefile**

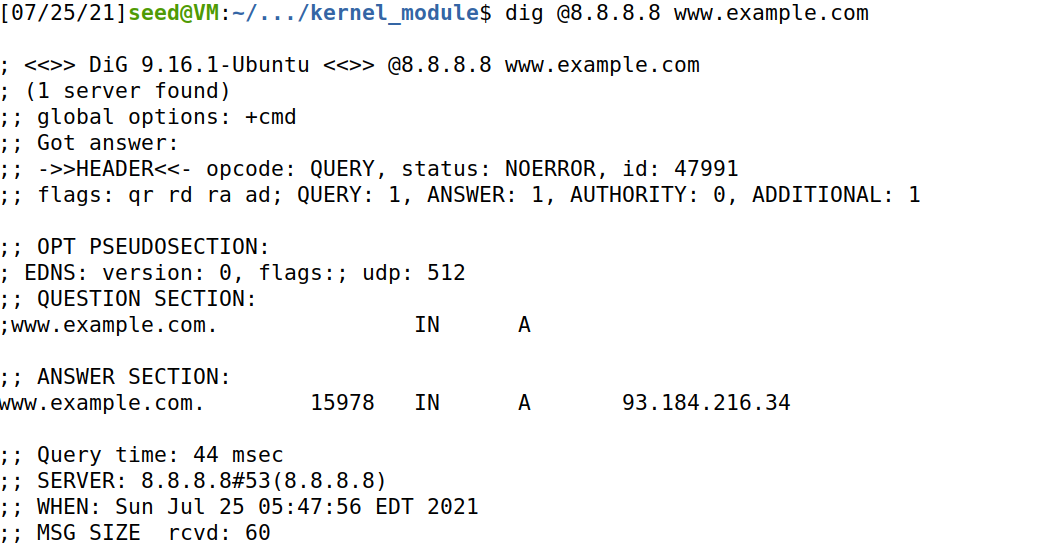
首先现在什么也不做的情况下运行命令，查看结果：



使用makefile编译seedFilter.c，并将其加载入内核，再次使用dig，发现申请被阻塞，得不到应答：



将该模块移除后，重复上述命令，可以得到应答，说明防火墙起效：



**2、 Hook the printInfo function to all of the netfilter hooks**

修改seedFilter.c文件，代码如下：

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/netfilter.h>

#include <linux/netfilter\_ipv4.h>

#include <linux/ip.h>

#include <linux/tcp.h>

#include <linux/udp.h>

#include <linux/if\_ether.h>

#include <linux/inet.h>

static struct nf\_hook\_ops hook1, hook2,hook3, hook4, hook5;

unsigned int blockUDP(void \*priv, struct sk\_buff \*skb,

const struct nf\_hook\_state \*state)

{

struct iphdr \*iph;

struct udphdr \*udph;

u16 port = 53;

char ip[16] = "8.8.8.8";

u32 ip\_addr;

if (!skb) return NF\_ACCEPT;

iph = ip\_hdr(skb);

// Convert the IPv4 address from dotted decimal to 32-bit binary

in4\_pton(ip, -1, (u8 \*)&ip\_addr, '\0', NULL);

if (iph->protocol == IPPROTO\_UDP) {

udph = udp\_hdr(skb);

if (iph->daddr == ip\_addr && ntohs(udph->dest) == port){

printk(KERN\_WARNING "\*\*\* Dropping %pI4 (UDP), port %d\n", &(iph->daddr), port);

return NF\_DROP;

}

}

return NF\_ACCEPT;

}

unsigned int printInfo(void \*priv, struct sk\_buff \*skb,

const struct nf\_hook\_state \*state)

{

struct iphdr \*iph;

char \*hook;

char \*protocol;

switch (state->hook){

case NF\_INET\_LOCAL\_IN: hook = "LOCAL\_IN"; break;

case NF\_INET\_LOCAL\_OUT: hook = "LOCAL\_OUT"; break;

case NF\_INET\_PRE\_ROUTING: hook = "PRE\_ROUTING"; break;

case NF\_INET\_POST\_ROUTING: hook = "POST\_ROUTING"; break;

case NF\_INET\_FORWARD: hook = "FORWARD"; break;

default: hook = "IMPOSSIBLE"; break;

}

printk(KERN\_INFO "\*\*\* %s\n", hook); // Print out the hook info

iph = ip\_hdr(skb);

switch (iph->protocol){

case IPPROTO\_UDP: protocol = "UDP"; break;

case IPPROTO\_TCP: protocol = "TCP"; break;

case IPPROTO\_ICMP: protocol = "ICMP"; break;

default: protocol = "OTHER"; break;

}

// Print out the IP addresses and protocol

printk(KERN\_INFO " %pI4 --> %pI4 (%s)\n",

&(iph->saddr), &(iph->daddr), protocol);

return NF\_ACCEPT;

}

int registerFilter(void) {

printk(KERN\_INFO "Registering filters.\n");

// Hook 1

hook1.hook = printInfo;

hook1.hooknum = NF\_INET\_LOCAL\_IN;

hook1.pf = PF\_INET;

hook1.priority = NF\_IP\_PRI\_FIRST;

nf\_register\_net\_hook(&init\_net, &hook1);

// Hook 2

hook2.hook = printInfo;

hook2.hooknum = NF\_INET\_PRE\_ROUTING;

hook2.pf = PF\_INET;

hook2.priority = NF\_IP\_PRI\_FIRST;

nf\_register\_net\_hook(&init\_net, &hook2);

// Hook 3

hook3.hook = printInfo;

hook3.hooknum = NF\_INET\_FORWARD;

hook3.pf = PF\_INET;

hook3.priority = NF\_IP\_PRI\_FIRST;

nf\_register\_net\_hook(&init\_net, &hook3);

// Hook 4

hook4.hook = printInfo;

hook4.hooknum = NF\_INET\_LOCAL\_OUT;

hook4.pf = PF\_INET;

hook4.priority = NF\_IP\_PRI\_FIRST;

nf\_register\_net\_hook(&init\_net, &hook4);

// Hook 5

hook5.hook = printInfo;

hook5.hooknum = NF\_INET\_POST\_ROUTING;

hook5.pf = PF\_INET;

hook5.priority = NF\_IP\_PRI\_FIRST;

nf\_register\_net\_hook(&init\_net, &hook5);

return 0;

}

void removeFilter(void) {

printk(KERN\_INFO "The filters are being removed.\n");

nf\_unregister\_net\_hook(&init\_net, &hook1);

nf\_unregister\_net\_hook(&init\_net, &hook2);

nf\_unregister\_net\_hook(&init\_net, &hook3);

nf\_unregister\_net\_hook(&init\_net, &hook4);

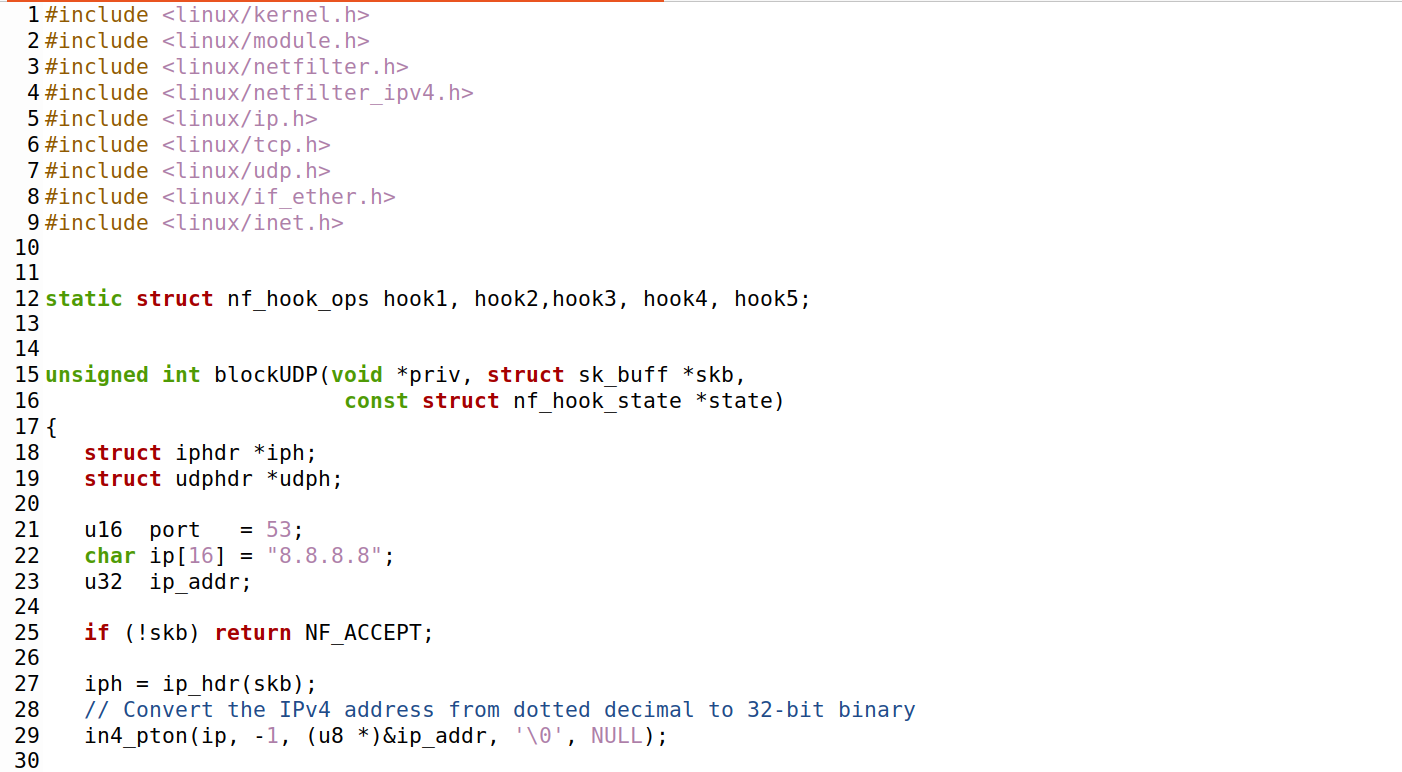
nf\_unregister\_net\_hook(&init\_net, &hook5);

}

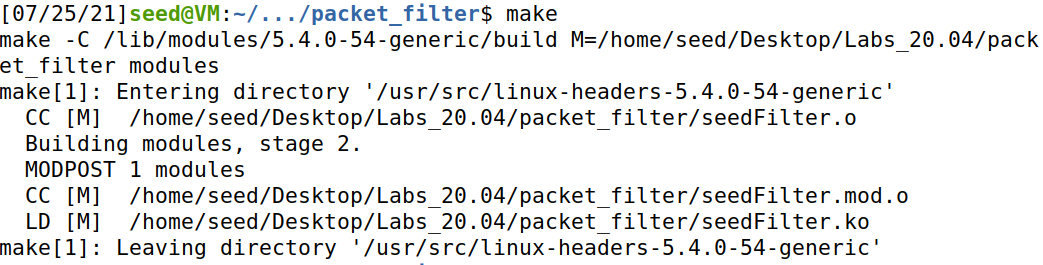
module\_init(registerFilter);

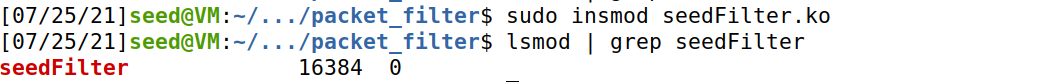
module\_exit(removeFilter);

MODULE\_LICENSE("GPL");

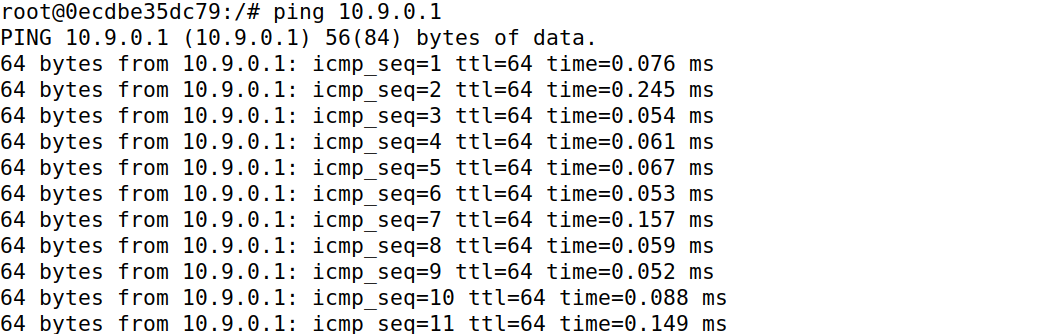


利用make命令编译，并且使用ismod命令插入内核如下：

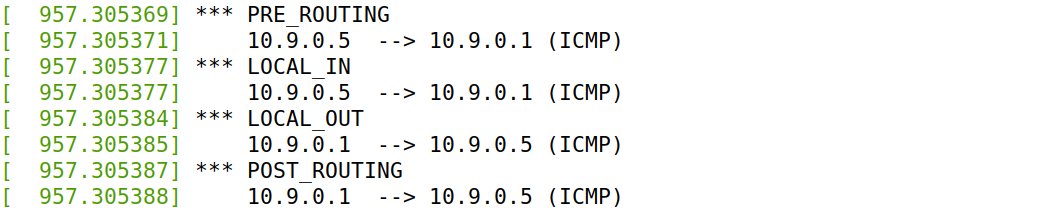




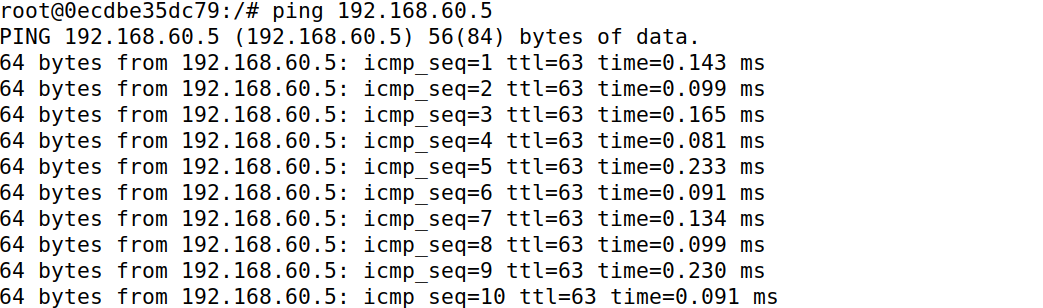
在用户主机上ping攻击者主机，得到结果如下，可知能够连接。



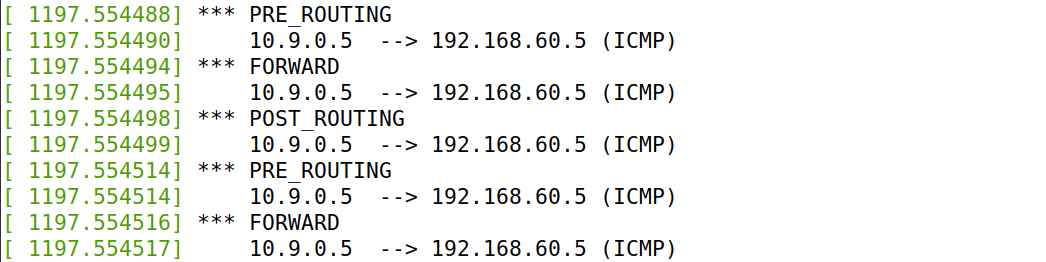
利用dmesg命令查看：



在用户主机上ping内网主机，得到结果如下，可知能够连接：



利用dmesg命令查看文件中的信息：



**3.、Implement two more hooks**

修改seedFilter.c文件，代码如下：

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/netfilter.h>

#include <linux/netfilter\_ipv4.h>

#include <linux/ip.h>

#include <linux/tcp.h>

static struct nf\_hook\_ops telnetFilterHook;

unsigned int telnetFilter(void \*priv, struct sk\_buff \* skb, const struct nf\_hook\_state \*state){

struct iphdr \*iph;

struct tcphdr \*tcph;

iph = ip\_hdr(skb);

tcph = (void \*)iph+iph->ihl\*4;

if((iph->protocol == IPPROTO\_TCP && (tcph->dest == htons(23)

|| tcph->dest== htons(22)

|| tcph->dest== htons(21)))

|| (iph->protocol == IPPROTO\_ICMP &&((((unsigned char \*)&iph->daddr)[0]==10 &&

((unsigned char \*)&iph->daddr)[1]==9

&& ((unsigned char \*)&iph->daddr)[2]==0 && ((unsigned char \*)&iph->daddr)[3]==1)

|| (((unsigned char \*)&iph->daddr)[0]==10 && ((unsigned char \*)&iph->daddr)[1]==9

&& ((unsigned char \*)&iph->daddr)[2]==0 && ((unsigned char \*)&iph->daddr)[3]==1)))){

printk(KERN\_INFO "Dropping telent packdt to %d.%d.%d.%d\n",

((unsigned char \*)&iph->daddr)[0],

((unsigned char \*)&iph->daddr)[1],

((unsigned char \*)&iph->daddr)[2],

((unsigned char \*)&iph->daddr)[3]);

return NF\_DROP;

}else{

return NF\_ACCEPT;

}

}

void removeFilter(void){

printk(KERN\_INFO "Telnet filter has been removed.\n");

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

}

int setUpFilter(void){

telnetFilterHook.hook = telnetFilter;

telnetFilterHook.hooknum = NF\_INET\_PRE\_ROUTING;

telnetFilterHook.pf = PF\_INET;

telnetFilterHook.priority = NF\_IP\_PRI\_FILTER;

if(nf\_register\_net\_hook(&init\_net,&telnetFilterHook)!=0){

printk(KERN\_WARNING "register Telnet filter hook error!\n");

goto err;

}

printk(KERN\_INFO "Registering a Telnet filter");

return 0;

err:

removeFilter();

return -1;

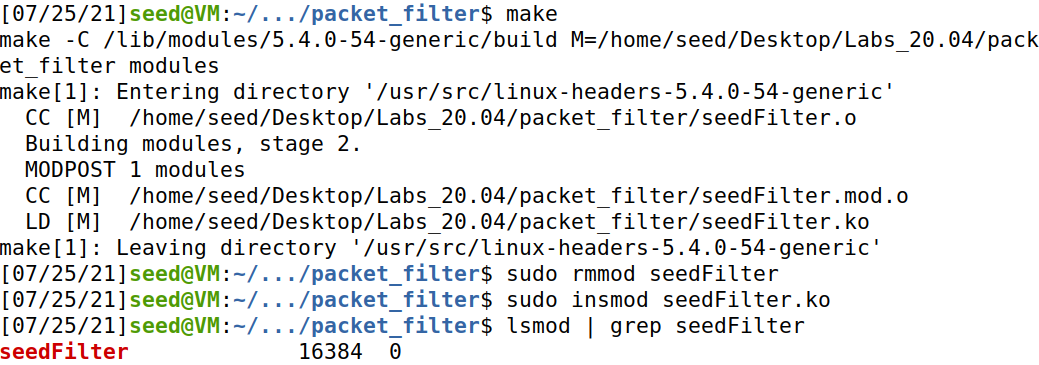
}

module\_init(setUpFilter);

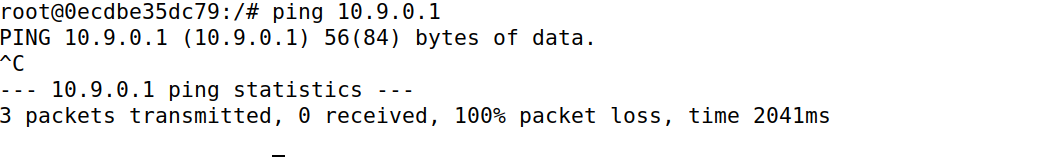
module\_exit(removeFilter);

MODULE\_LICENSE("GPL");

利用make命令编译可装载内核模块，并利用ismod命令插入内核模块如下：



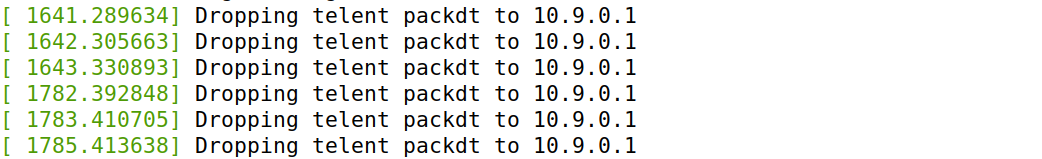
在用户主机上ping攻击者主机，得到结果如下：



在用户主机上telnet远程连接攻击者主机，得到结果如下，可知连接失败：



利用dmesg命令查看，可知报文被丢弃：

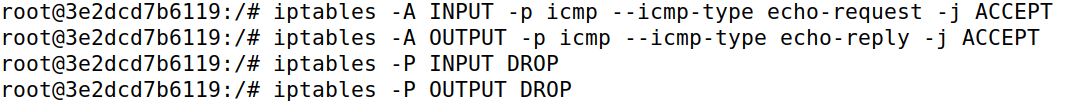


**Task 2: Experimenting with Stateless Firewall Rules**

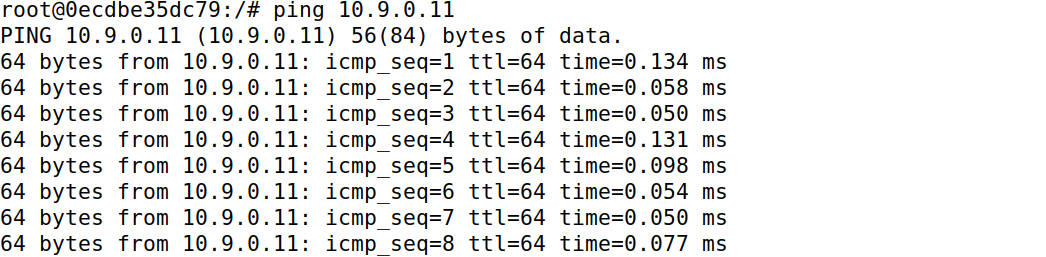
用户主机的IP地址为10.9.0.5，路由器的IP地址为10.9.0.11，内网网段的IP地址为192.168.60.0/24。

**Task 2.A: Protecting the Router**

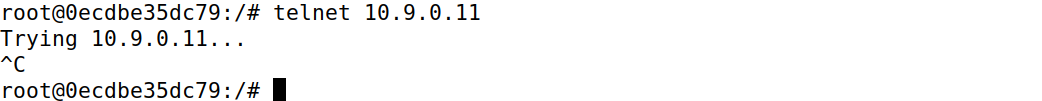
在路由器上利用iptables命令，创建过滤规则如下。



在用户主机上ping路由器，得到结果如下，可知能够连接：



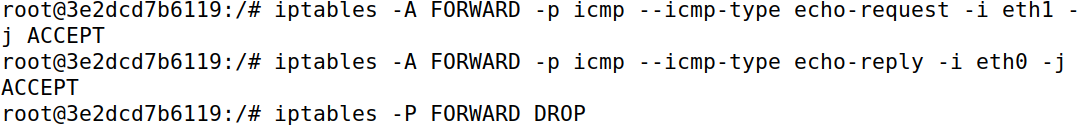
在用户主机上telnet远程连接路由器，得到结果如下，可知连接失败：



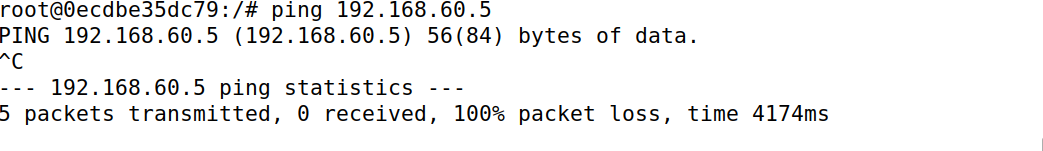
可知该现象的原因是路由器的过滤规则只允许icmp请求报文输入和icmp响应报文输入，ping的报文可以进行传输，儿telent的报文无法进行传输。

**Task 2.B: Protecting the Internal Network**

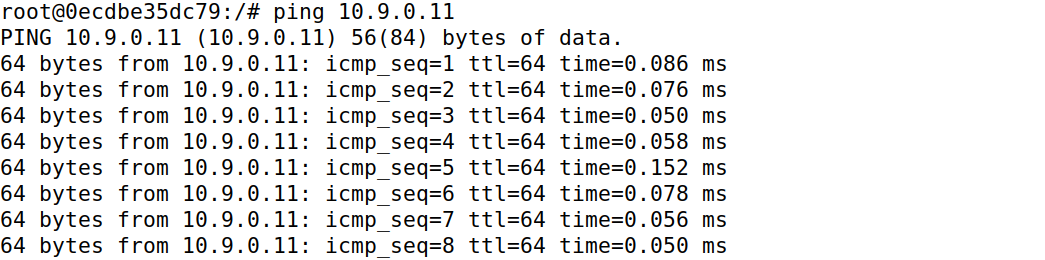
在路由器上利用iptables命令，创建过滤规则如下。



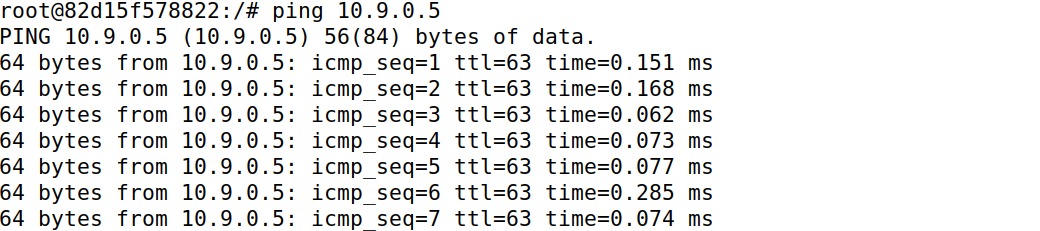
在用户主机上ping内网主机192.168.60.5，得到结果如下，可知无法连接：



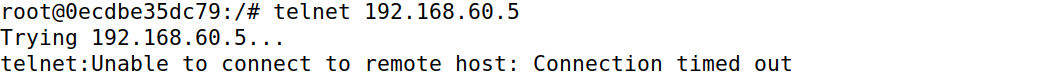
在用户主机上ping路由器，得到结果如下，可知能够连接：



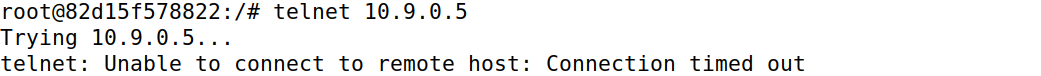
在ip地址为192.168.60.5的内网主机上ping用户主机，得到结果如下，可知能够连接：



在用户主机上telent远程连接内网主机192.168.60.5，得到结果如下，可知连接失败：

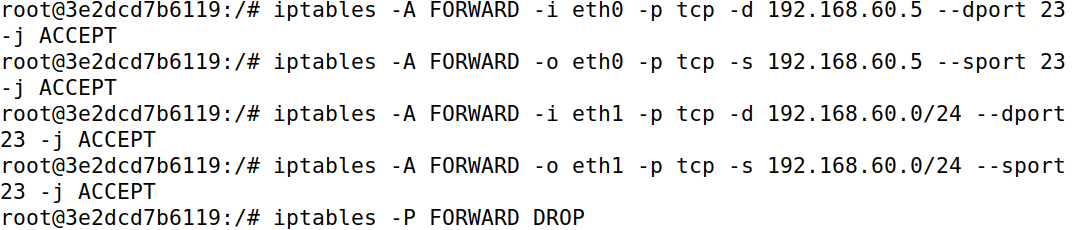


在地址为192.168.60.5的内网主机上telnet远程连接用户主机，得到结果如下，可知连接失败：

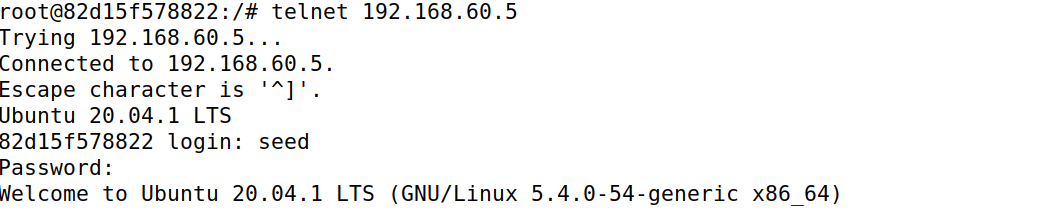


**Task 2.C: Protecting Internal Servers**

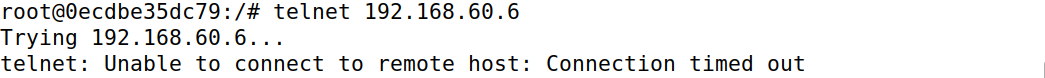
在路由器上利用iptables命令，创建过滤规则如下。



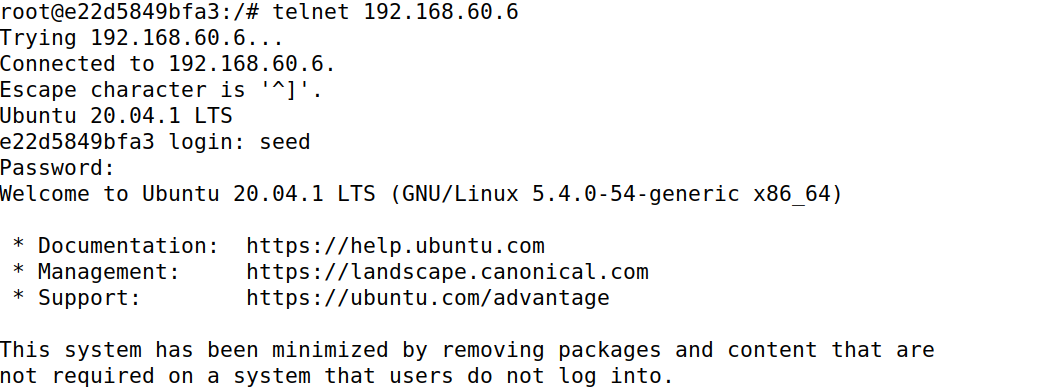
在用户主机上telnet远程连接网络192.168.60.5，得到结果如下：



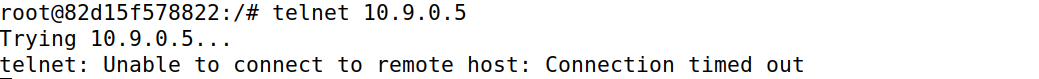
在用户主机上telnet远程连接网络192.168.60.6，得到结果如下：



在ip地址为192.168.60.5的内网主机上telnet远程连接内网主机192.168.60.6，得到结果如下，可知连接成功：



在ip地址为192.168.60.5的内网主机上远程连接用户主机，得到结果如下：



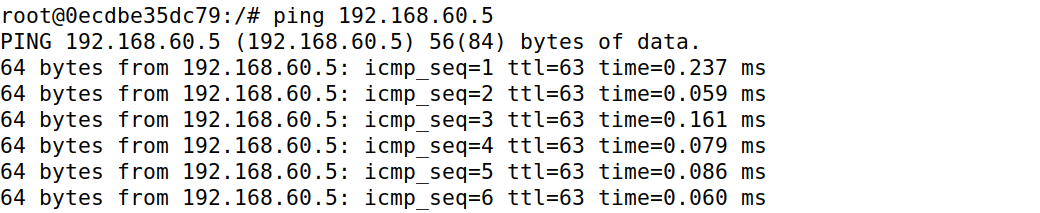
**Task 3: Connection Tracking and Stateful Firewall**

用户主机的IP地址为10.9.0.5，路由器的IP地址为10.9.0.11，内网网段的IP地址为192.168.60.0/24。

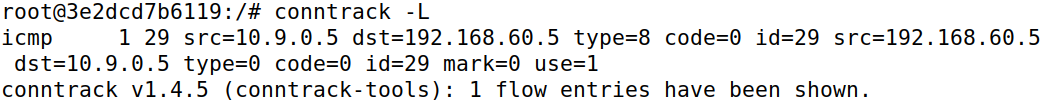
**Task 3.A: Experiment with the Connection Tracking**

**ICMP experiment**

在用户主机上ping内网主机192.168.60.5，得到结果如下，可知能够连接。

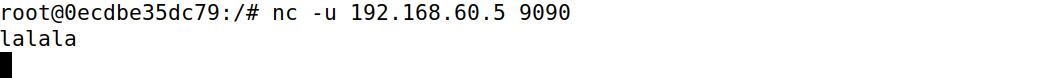


在路由器上利用conntrack -L命令实现连接跟踪，得到结果如下：



**UDP experiment**

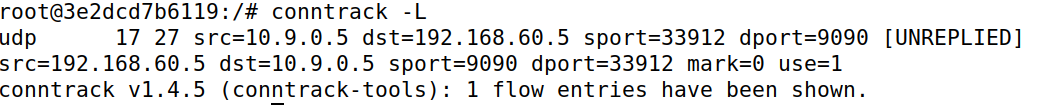
在用户主机上利用UDP远程连接IP地址为192.168.60.5的内网主机的9090端口，并发送消息：



在内网主机192.168.60.5上监听9090端口的UDP连接：

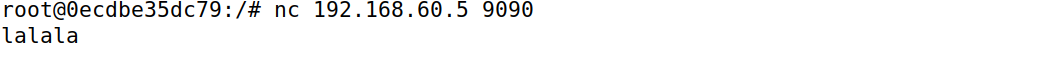


在路由器上利用conntrack -L命令实现追踪，得到结果如下：



**TCP experiment**

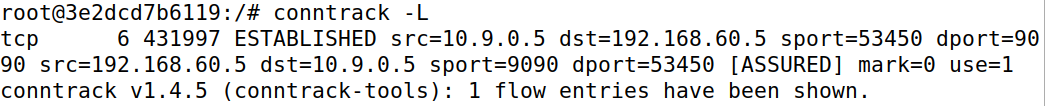
在用户主机上利用TCP远程连接ip地址为192.168.60.5的内网主机9090端口，并发送消息：



在内网主机192.168.60.5上监听9090端口的TCP连接：

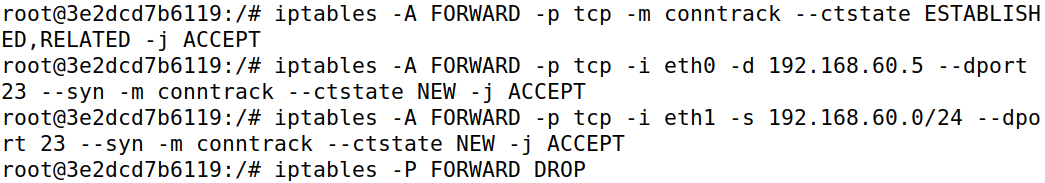


在路由器上利用conntrack -L命令实现追踪，得到结果如下：

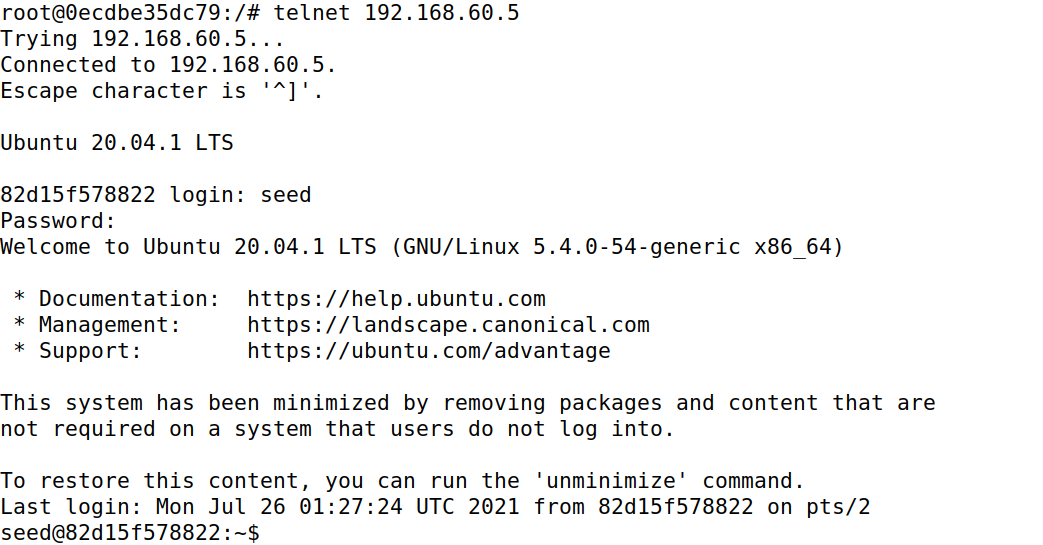


**Task 3.B: Setting Up a Stateful Firewall**

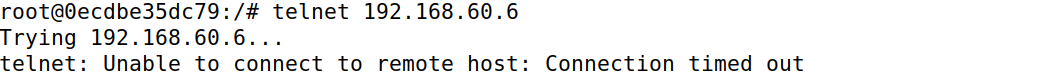
在路由器上利用iptables命令和连接跟踪机制，创建过滤规则如下。



在用户主机上telnet远程连接内网主机192.168.60.5，得到结果如下，可知连接成功：



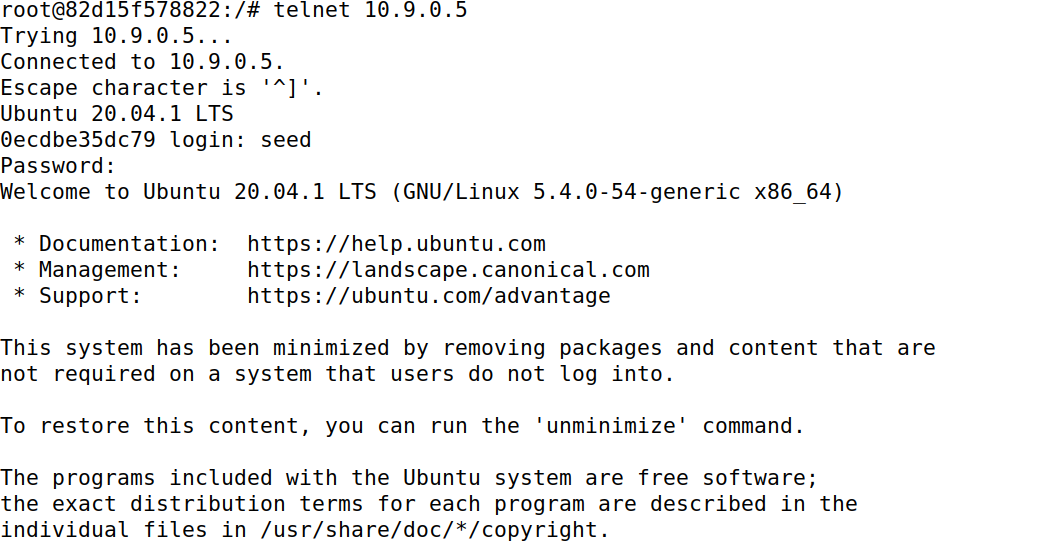
在用户主机上telent远程连接内网主机192.168.60.6，得到结果如下，可知连接失败：



在ip地址为192.168.60.5的内网主机上telnet远程连接内网主机192.168.60.6，得到结果如下：



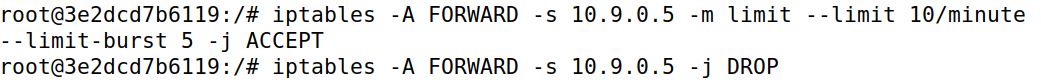
在ip地址为192.168.60.5的内网主机上telnet远程连接用户主机，得到结果如下，可知连接成功：



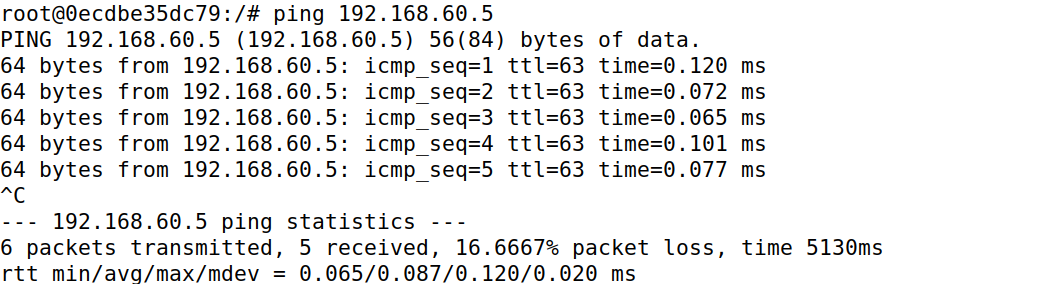
不利用连接跟踪机制的过滤规则仅对数据包的首部进行检查，其优点是处理速度快，缺点是无法定义精细的规则、不适合复杂的访问控制；而利用连接跟踪机制的过滤规则对数据包的状态也进行检查，其优点是能够定义更加严格的规则、应用范围更广、安全性更高，缺点是无法对数据包的内容进行识别。

**Task 4: Limiting Network Traffic**

在路由器上利用iptables命令，创建流量限制规则如下。



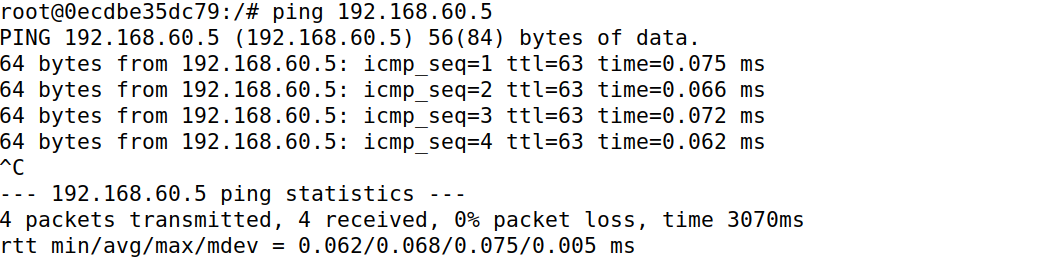
在用户主机上ping内网主机192.168.60.5，得到结果如下，可知能够连接，但部分报文因流量限制儿丢失：



在路由器上利用iptables命令，修改流量规则如下：



在用户主机上ping内网主机192.168.60.5，得到结果如下，可知能够连接，且无报文丢失：

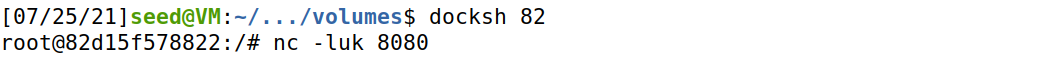


可知该现象是由路由器的转发链的默认规则是ACCEPT，即使超过流量限制，报文根据默认规则也可以进行传输，且第二条规则是必须的的。

**Task 5: Load Balancing**

用户主机的IP地址为10.9.0.5，路由器的IP地址为10.9.0.11，三个服务器的IP地址为192.168.60.5、192.168.60.6和192.168.60.7。

Using the nth mode (round-robin)

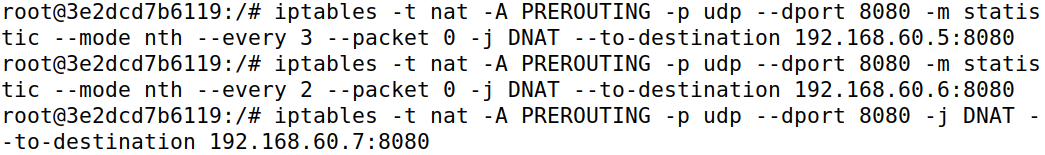




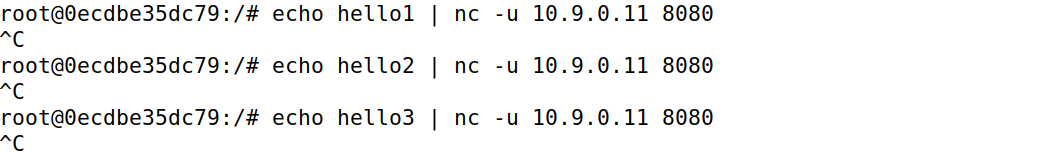


1. 使用nth mode：

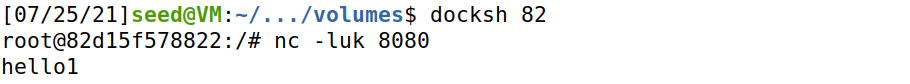
在路由器上利用iptables命令，采用nth模式创建负载均衡规则如下。



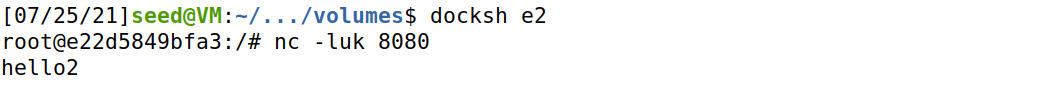
在用户主机10.9.0.5上输入命令：



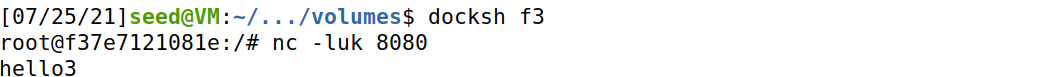
发现hello1被发送到192.168.60.5:8080端口：



发现hello2被发送到192.168.60.6:8080端口：

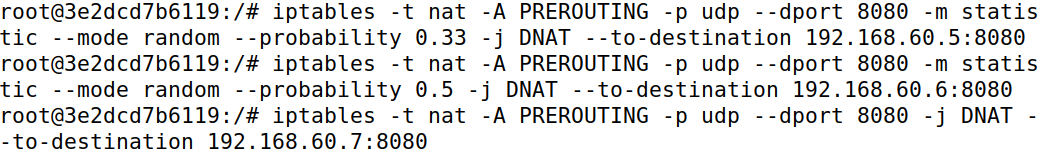


发现hello3被发送到192.168.60.7:8080端口：

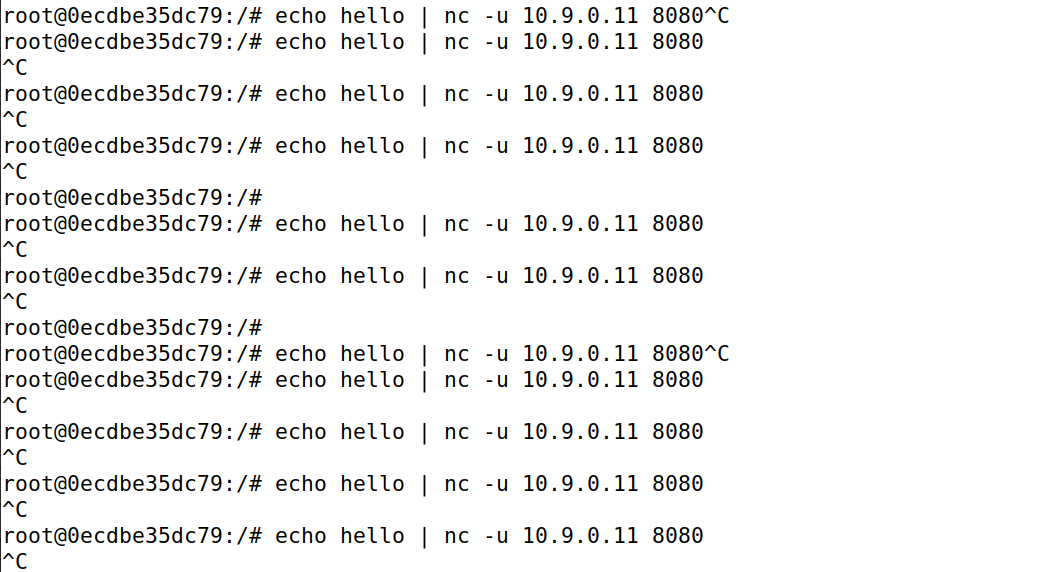


1. 使用random mode

清除之前的iptables规则，路由器中输入以下规则，即将已0.33的概率将报文发送到192.168.60.5:8080端口，在发送后，剩下的报文将以0.5的概率发送到192.168.60.6:8080端口，剩下的所有报文发送到192.168.60.7:8080端口，来达到负载均衡的效果



在用户主机上向路由器的8080端口发送UDP数据包：



发现在192.168.60.5:8080端口发送的UDP数据包：



发现在192.168.60.6:8080端口发送的UDP数据包：



发现在192.168.60.7:8080端口发送的UDP数据包：

