**实验报告6**

**57118139顾宸玮**

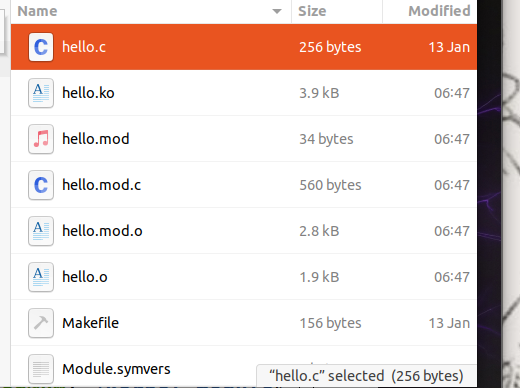
**Task 1: Implementing a Simple Firewall**

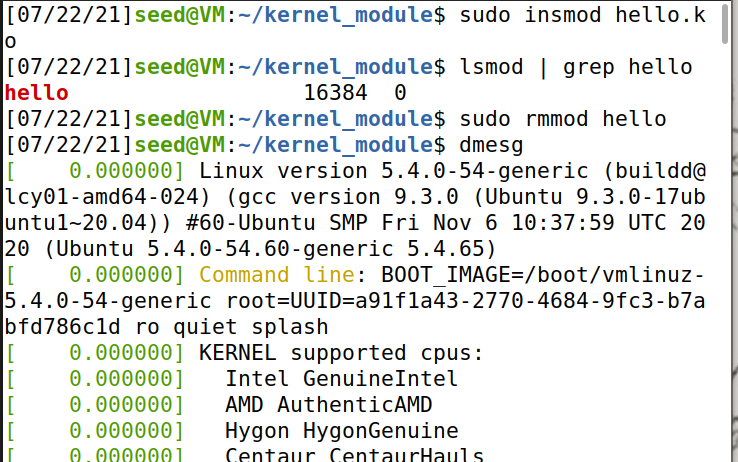
在本任务中，我们将实现一个简单的包过滤类型的防火墙，它检查每个传入和传出的包，并执行管理员设置的防火墙策略。因为包处理是在内核中完成的，所以过滤也必须在内核中完成。因此，实现这样的防火墙似乎需要我们修改Linux内核。在过去，这必须通过修改和重新构建内核来完成。现代Linux操作系统提供了几种新的机制，以简化对包的操作，而无需重新构建内核映像。这两种机制是可加载内核模块(LKM)和Netfilter。

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

LKM允许我们在运行时向内核添加一个新模块。这个新模块使我们能够扩展内核的功能，而无需重新构建内核甚至重新启动计算机。防火墙的包过滤部分可以作为LKM实现。在这个任务中，我们将熟悉LKM。

Make后该文件夹内容：





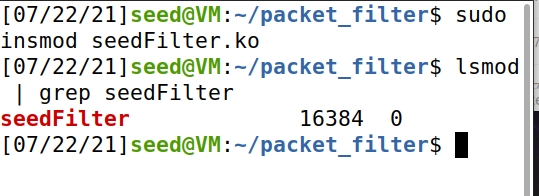
**1.B: Implement a Simple Firewall Using Netfilter**

在这个任务中，我们将以LKM的形式编写包过滤程序，然后将其插入到内核中的包处理路径中。

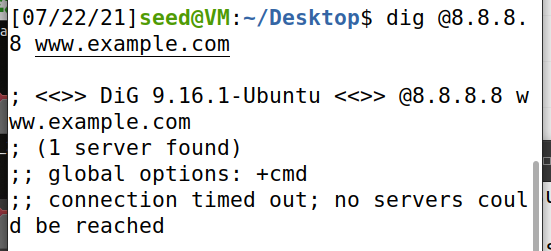
Netfilter是为授权用户操作数据包而设计的。它通过在Linux内核中实现许多hook来实现这个目标。这些hook被插入到不同的位置，包括数据包的进出路径。如果我们想要操作传入的数据包，我们只需要将自己的程序(在LKM内)连接到相应的hook。一旦传入数据包到达，我们的程序将被调用。我们的程序可以决定这个数据包是否应该被阻塞;此外，还可以对程序中的数据包进行修改。

1. 使用提供的Makefile编译示例代码。将它加载到内核中，并演示防火墙按预期工作。可以通过以下命令生成到谷歌的DNS服务器8.8.8.8的UDP报文。如果你的防火墙工作，你的请求将被阻止;否则，您将得到一个响应。

将代码加载到内核：



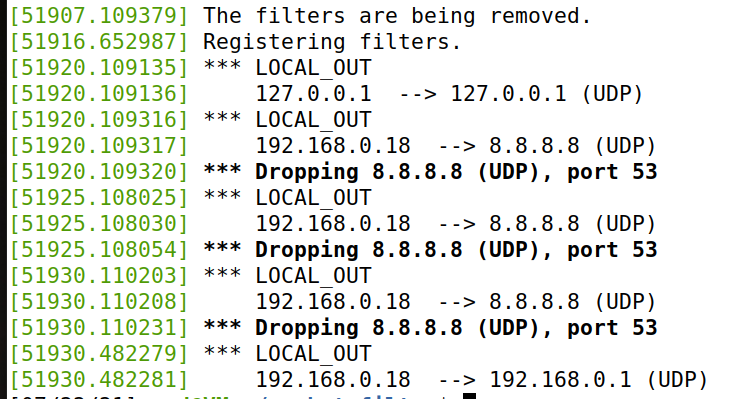
发现请求被阻塞了：



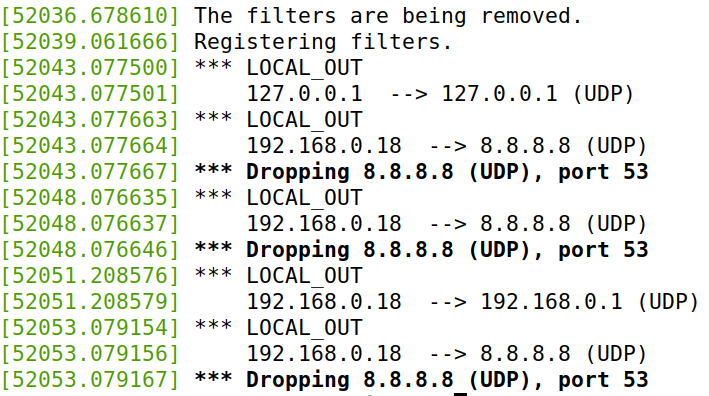
2.将printInfo函数与所有的netfilter hook挂钩。下面是hook号的宏。使用你的实验结果来帮助解释每个hook函数在什么情况下会被调用。

NF\_IP\_LOCAL\_OUT:在数据包以其方式离开主机之前。

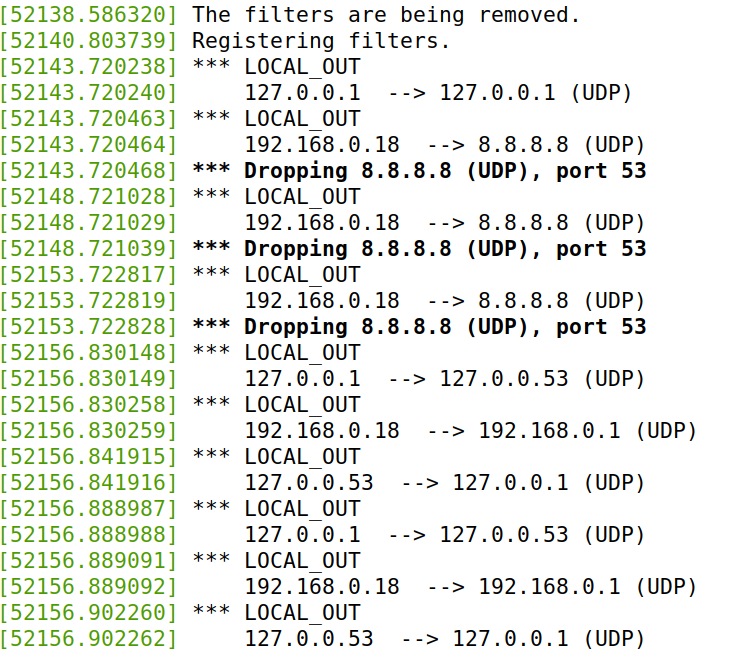
NF\_IP\_POST\_ROUTING:数据包离开主机并进入不同的网络之后。



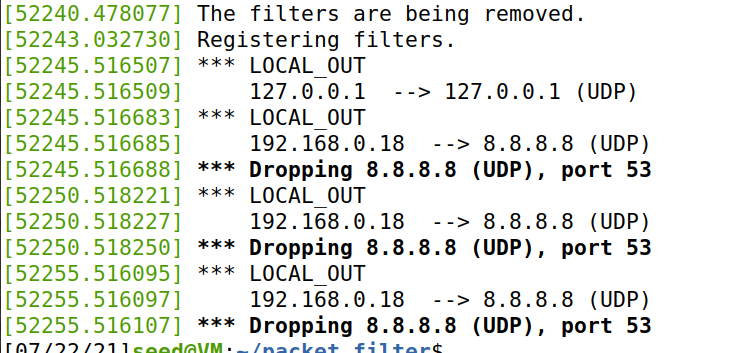
NF\_IP\_PRE\_ROUTING:在做出任何路由决策之前



NF\_IP\_LOCAL\_IN:在发送到网络堆栈之前



NF\_IP\_FORWARD:向其他主机转发报文。



数据报从进入系统，进行IP校验以后，首先经过第一个HOOK函数NF\_INET\_PRE\_ROUTING进行处理；然后就进入路由代码，其决定该数据报是需要转发还是发给本机的；若该数据报是发被本机的，则该数据经过HOOK函数NF\_INET\_LOCAL\_IN处理以后然后传递给上层协议；若该数据报应该被转发则它被NF\_INET\_FORWARD处理；经过转发的数据报经过最后一个HOOK函数NF\_INET\_POST\_ROUTING处理以后，再传输到网络上。本地产生的数据经过HOOK函数NF\_INET\_LOCAL\_OUT 处理后，进行路由选择处理，然后经过NF\_INET\_POST\_ROUTING处理后发送出去。

3. 再实现两个HOOK，实现以下目的:(1)防止其他计算机ping VM，(2)防止其他计算机telnet到VM。请实现两个不同的HOOK函数，但将它们注册到同一个netfilter钩子。

C++代码如下：telnet端口号是23，并且防止10.9.0.5与10.9.0.1之间的icmp包转送

#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 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)

|| (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]==5)))

)

{

printk(KERN\_INFO "Dropping telent packet 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\_FIRST;

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;

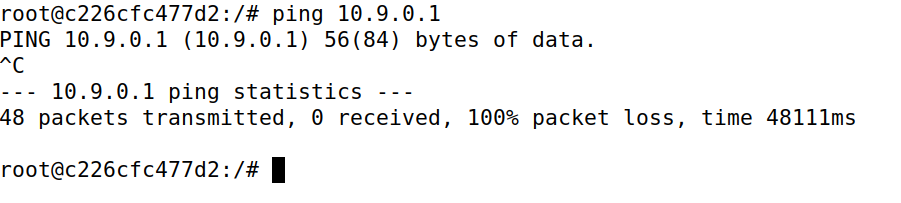
}

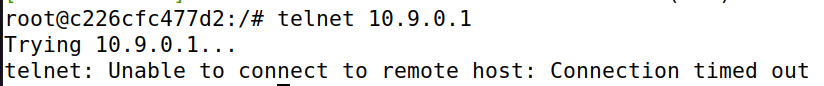
module\_init(setUpFilter);

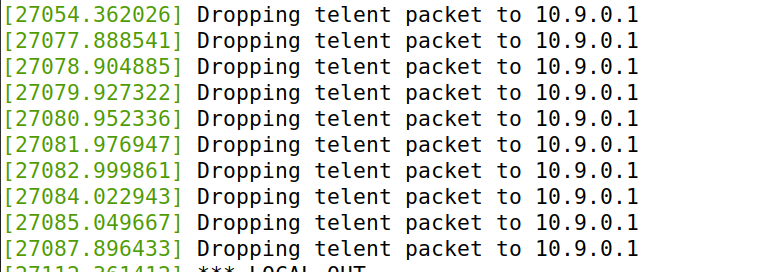
module\_exit(removeFilter);

MODULE\_LICENSE("GPL");

经过测试发现，在10.9.0.5主机上的两个测试包均无法发送，通过dmesg发现都被过滤掉了。







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

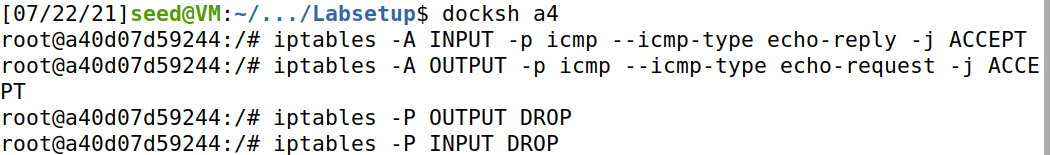
在本任务中，我们将使用iptables设置防火墙。iptables防火墙不仅可以过滤数据包，还可以对数据包进行更改。为了帮助针对不同的目的管理这些防火墙规则，iptables使用分层结构组织所有规则:表、链和规则。

**A: Protecting the Router**

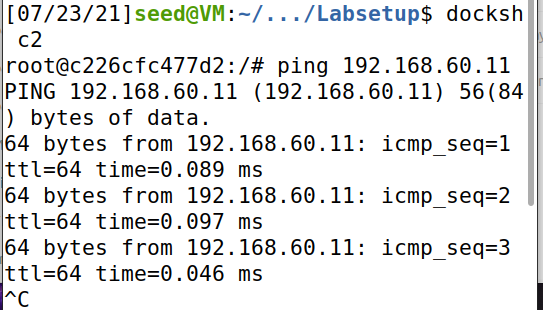
在这个任务中，我们将设置一些规则来防止外部机器访问路由器机器，除了ping。

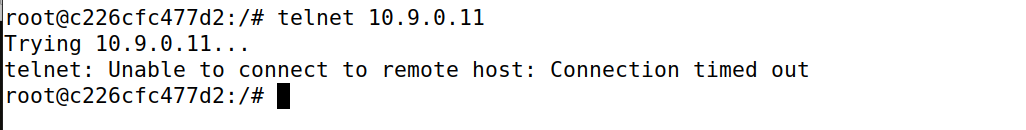
请在路由器容器上执行以下iptables命令，然后尝试从10.9.0.5访问它。（1）你能ping通路由器吗?(2)你能telnet到路由器(一个telnet服务器运行在所有容器上;在它们上面创建了一个名为seed的帐户，密码为dees)。请报告你的观察并解释每条规则的目的。

首先在路由器上设置以下过滤规则：

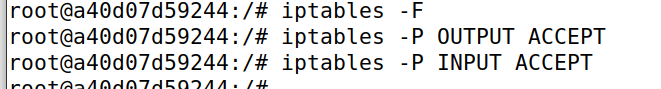


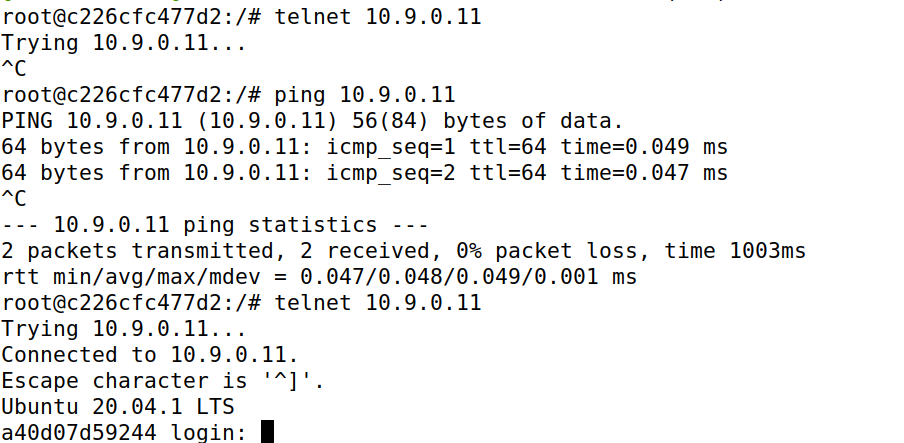
结果发现，从10.9.0.5上可以ping通路由器，但无法telnet到路由器





将上述规则取消掉，发现可以ping和telnet



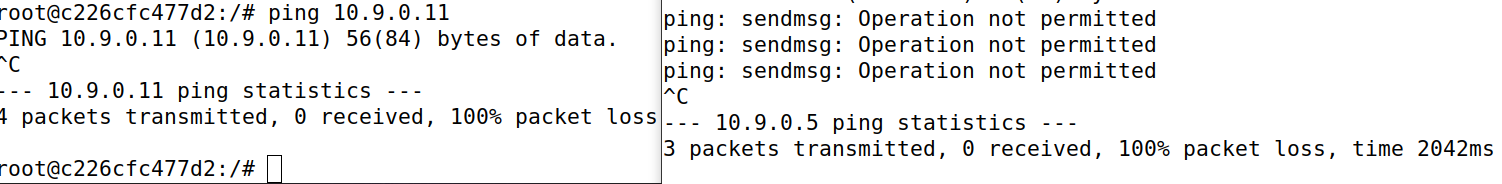


iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT （将手册中的顺序换了一下）

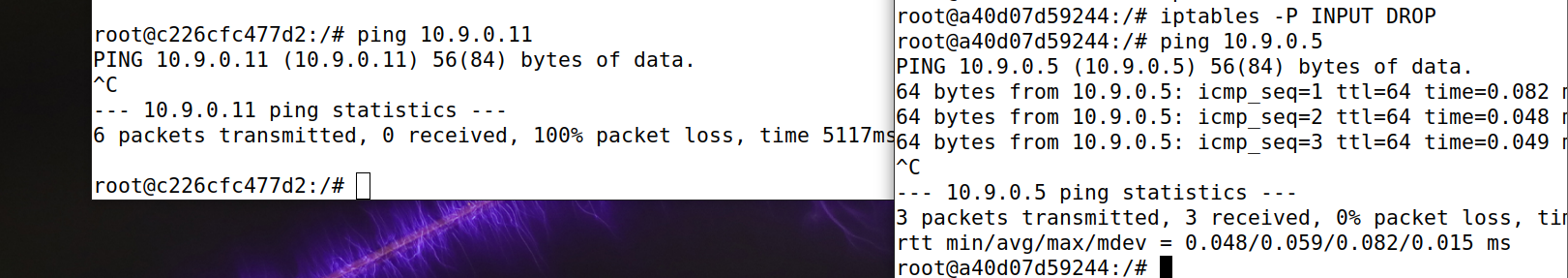
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT

上述两条规则表示外部主机可以ping通防火墙，即其他主机可以ping通防火墙主机（即router），防火墙接收icmp的request请求报文，也可以发出icmp相应报文。

设置了iptables -P OUTPUT DROP后，二者无法ping通，表示丢弃所有外出的包



在单独设置了iptables -P INPUT DROP，可以发现，router可以ping通其他主机，但是其他主机不可以通router，表示所有进入的包都被丢弃了，但是外出的包不受限制。



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

在这个任务中，我们将在路由器上设置防火墙规则来保护内部网络192.168.60.0/24。

为此，我们需要使用FORWARD链。信息包在输入和输出链中的方向很清楚:信息包要么进入(输入)，要么出(输出)。这对于FORWARD链是不正确的，因为它是双向的:进入内部网络或出去到外部网络的数据包都要经过这个链。为了指定方向，我们可以使用“-i xyz”(来自xyz接口)和/或“-o xyz”(来自xyz接口)添加接口选项。内部网络和外部网络的接口不一样。

设置规则如下：

iptables -P OUTPUT DROP

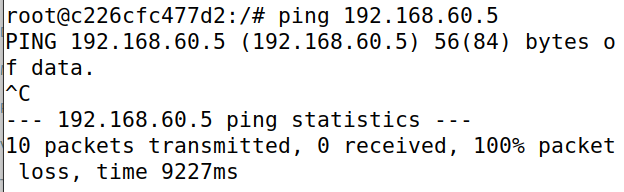
iptables -P INPUT DROP

iptables -A FORWARD -p icmp --icmp-type echo-request -o eth1 -j DROP

iptables -I OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT

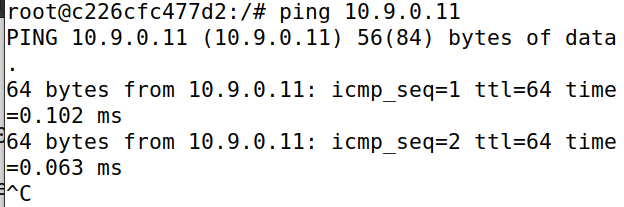
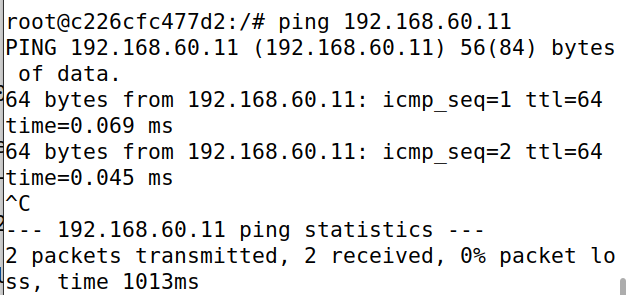
iptables -I INPUT -p icmp --icmp-type echo-request -j ACCEPT

1）外部主机不可以ping通内部主机：发现从10.9.0.5ping到内网全部丢失

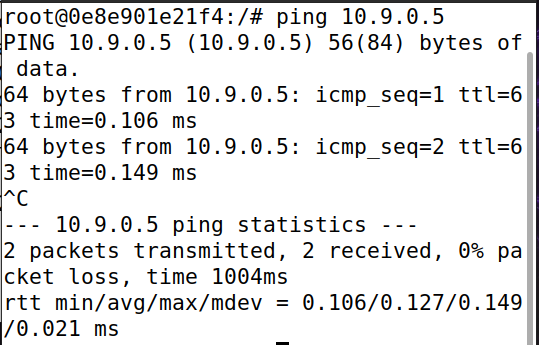


2）外部主机可以ping通router

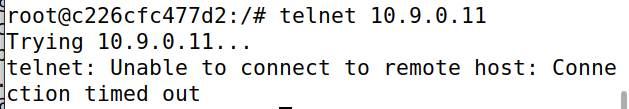
10.9.0.5分别ping router的eth0和eth1接口ip，发现全部可以ping通



3）内部主机可以ping通外部主机：内部主机192.168.60.5 ping 10.9.0.5可以ping通



4）内网外的其他包会被阻塞：尝试发送tcp包的telnet发现无法连接。



**2.C: Protecting Internal Servers**

在这个任务中，我们想要保护内部网络(192.168.60.0/24)内的TCP服务器。更具体地说，我们希望实现以下目标。

规则如下：

iptables -P OUTPUT DROP

iptables -P FORWARD DROP

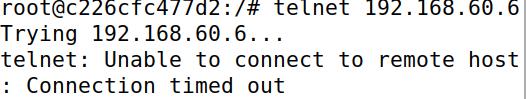
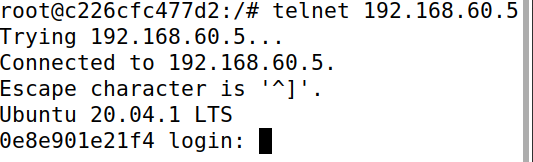
iptables -P INPUT DROP

iptables -A FORWARD -p tcp -m tcp --dport 23 -d 192.168.60.5 -j ACCEPT

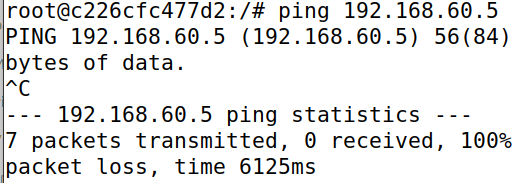
iptables -A FORWARD -p tcp --sport 23 -s 192.168.60.5 -j ACCEPT

iptables -A FORWARD -i eth1 -o eth1 -j ACCEPT

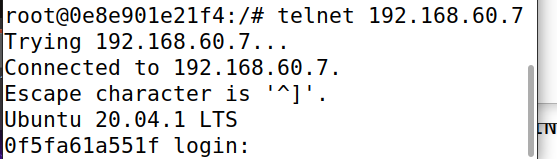
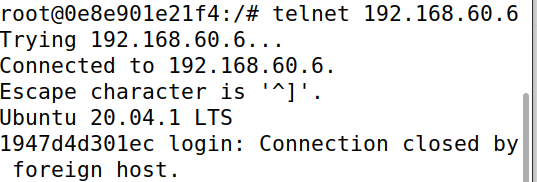
1）所有内部主机都运行telnet服务器(侦听端口23)。外部主机只能访问192.168.60.5上的telnet服务器，不能访问其他内部主机。

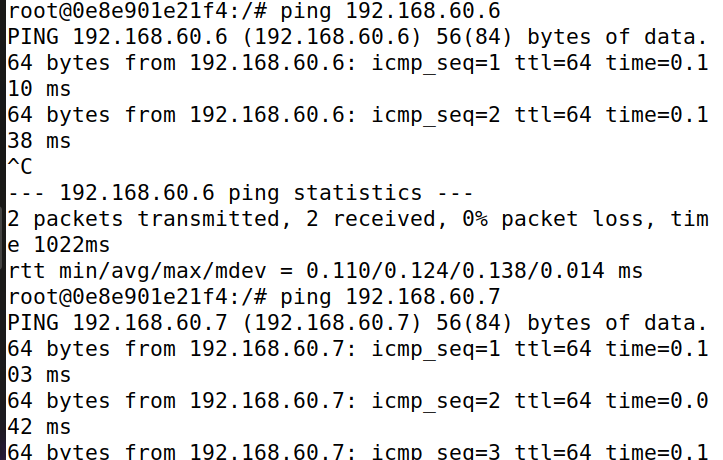


2）外部主机不能访问内部服务器

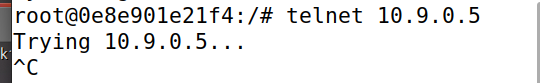
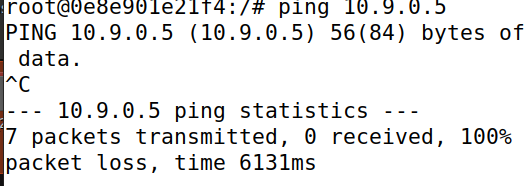


3）内部主机可以访问所有内部服务器





4)内部主机不可以访问外部服务器

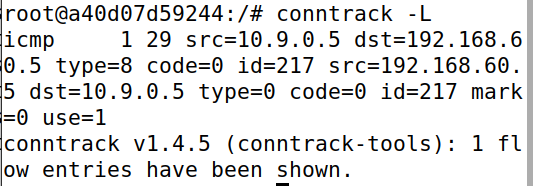
 

**Task 3: Connection Tracking and Stateful Firewall**

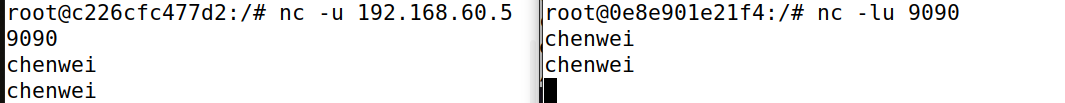
**3.A: Experiment with the Connection Tracking**

ICMP实验:执行如下命令，查看路由器上的连接跟踪信息。描述你的观察。ICMP连接状态保持多长时间?

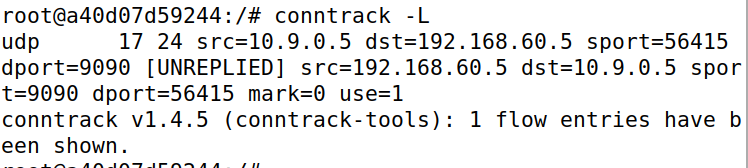
该连接状态保持了29s（默认为30s），type说明ICMP类型，code说明ICMP的代码，



UDP实验:执行如下命令，查看路由器上的连接跟踪信息。描述你的观察。UDP连接状态保持多长时间?

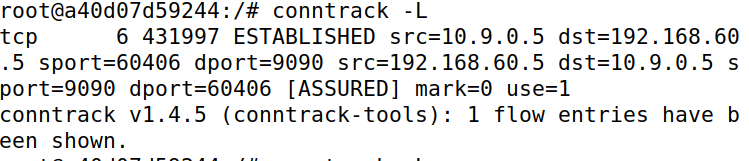


该连接状态保持了24s（默认是30s），发现192.168.60.5【UNREPLIED】说明还未收到响应。

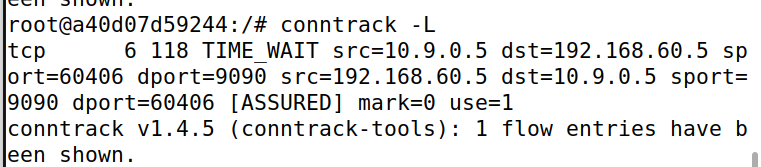


TCP实验:执行如下命令，查看路由器上的连接跟踪信息。描述你的观察。TCP连接状态保持多长时间?

保持时间为431997s（默认为432000s），可以发现连接已经建立



在我终止连接后，状态如下，保持时间为120s



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

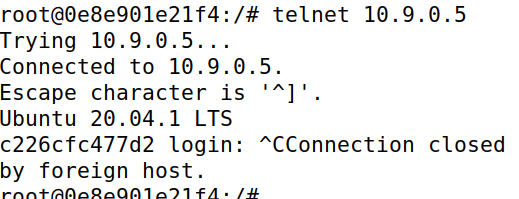
请重写Task 2.C中的防火墙规则，但这次，我们将添加一个允许内部主机访问任何外部服务器的规则(这在Task 2.C中是不允许的)。在使用连接跟踪机制编写规则之后，请考虑如何在不使用连接跟踪机制的情况下执行规则(您不需要实际实现它们)。基于这两套规则，比较这两种不同的方法，并解释每种方法的优点和缺点。当你完成这个任务时，记得清除所有的规则。

在2.c的基础上，增加连接跟踪机制：

iptables -A FORWARD -p tcp -i eth1 --syn -m conntrack --ctstate NEW -j ACCEPT

iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

可以发现，内部主机192.168.60.5可以telnet到外部服务器10.9.0.5.



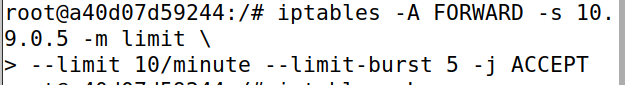
上述两种机制中，如果不是用连接跟踪机制，则还是需要单独设置防火墙可以转发内网访问外网23端口的规则，这样不是通过建立tcp的三次握手跟踪，可能会比设置有上下文关系的跟踪机制麻烦一点，并且即使打开了，也可能不能防止三次握手以外的恶意连接，但是设置了跟踪机制就可以只考虑到三次握手的情况，其余包均丢弃。

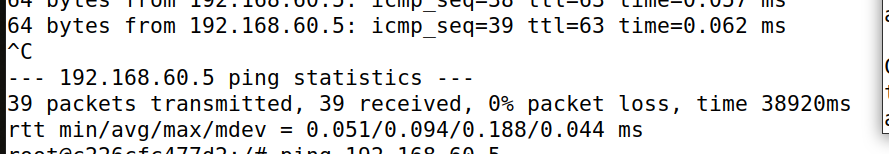
**Task 4: Limiting Network Traffific**

除了阻止数据包，我们还可以限制可以通过防火墙的数据包的数量。

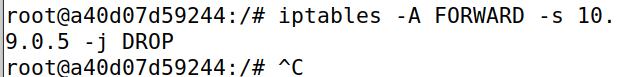
这可以使用iptables的limit模块来完成。在这个任务中，我们将使用这个模块来限制允许从10.9.0.5进入内部网络的数据包的数量。

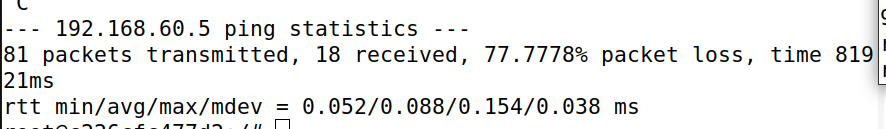
在只设置第一个规则的情况下，可以发现完全可以ping通，并且没有丢包。





在加了第二个规则后，发现有丢包情况，并且icmp请求和响应都会有时间间隔。





第二条规则有必要，因为第一条设置的是最大平均匹配率，并且也有数量的限制，如果没有第二条默认情况下将来自192.168.60.5的回复都丢弃掉，那么第一条规则就无法实现，也就是说默认接收，在有无limit情况下都会返回一个响应，也就无所谓是否有限制的规则。

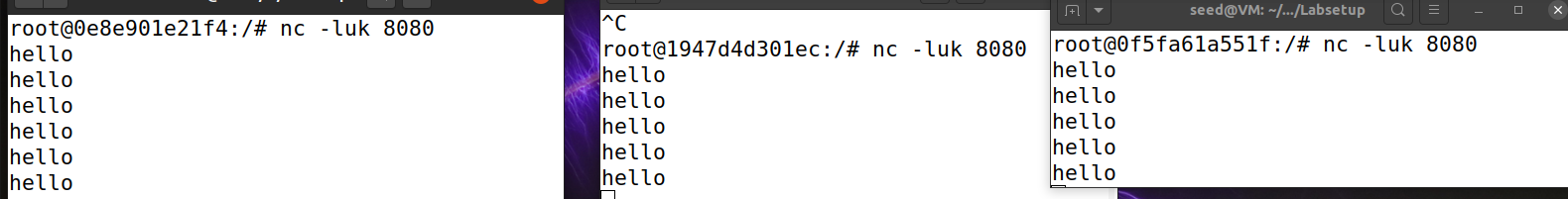
**Task 5: Load Balancing**

轮询法：规则如下：分别表示每3个报文到192.168.60.5上，每两个到192.168.60.6上，每一个报文到192.168.60.7上，这样子三个servers轮询，结果发现.6和.7基本上平均，但是大部分还是在.5server上

iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 0 -j DNAT --to-destination 192.168.60.5:8080

iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 2 --packet 0 -j DNAT --to-destination 192.168.60.6:8080

iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT --to-destination 192.168.60.7:8080



随机法：规则如下，在192.168.60.5的主机上设置概率为0.33，则其有0.33的概率被命中，在192.168.60.6的主机上设置为0.5，则其有0.5\*（1-0.33）≈0.33被命中，192.168.60.7的缺省，以为前两个没命中就一定会命中这个，因此三者被命中都约等于0.33，经过测试发现，因为发送的报文较少，因此不太均匀，还是192.168.60.5较多，另外两个较平均。

iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.33 -j DNAT --to-destination 192.168.60.5:8080

iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.5 -j DNAT --to-destination 192.168.60.6:8080

iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT --to-destination 192.168.60.7:8080

