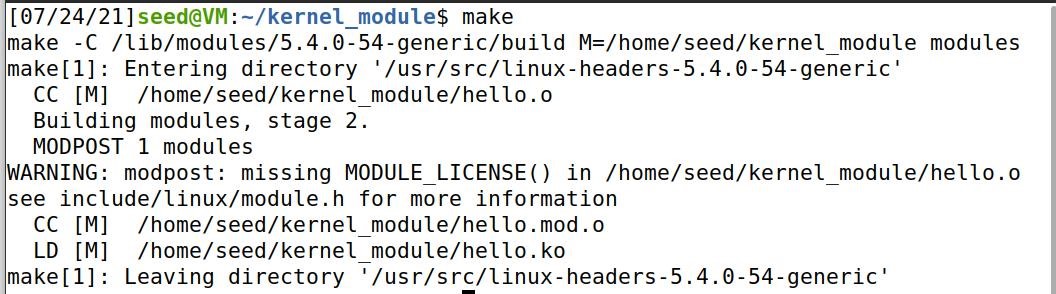
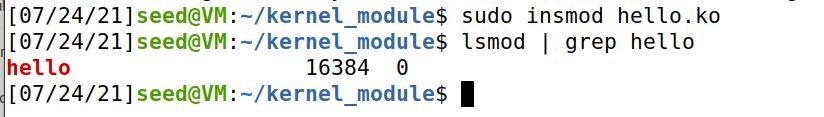
**Firewall Exploration**

57118114 蔡欣明

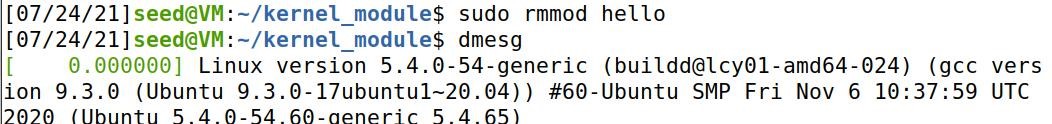
# Task 1.A: Implement a Simple Kernel Module 将 kernel\_module 文件夹拷贝到 home 目录下编译：



使用 sudo insmod hello.ko 命令加载模块并使用 lsmod | grep hello 命令查看:



使用 sudo rmmod hello 命令删除模块，并使用 dmesg 命令输出消息：



结尾可以看到函数输出的消息：

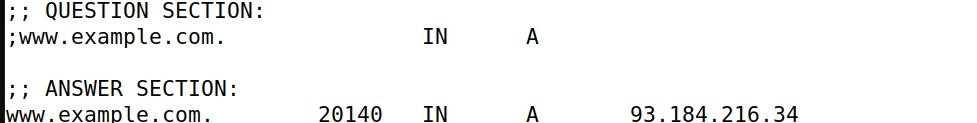


# Task 1.B: Implement a Simple Firewall Using Netfilter

1.编译示例代码

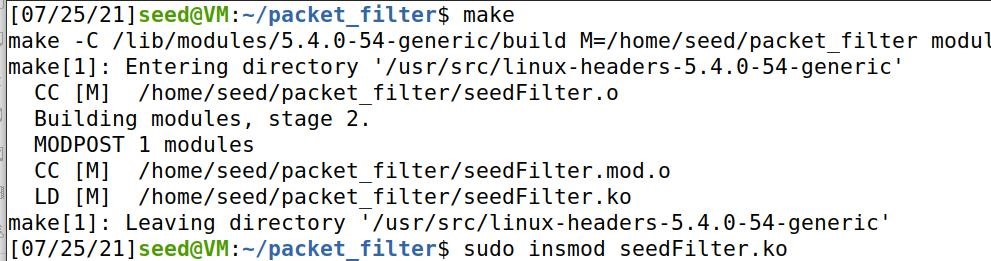
编译示例代码前使用命令 dig @8.8.8.8 www.example.com 生成到谷歌服务器

的 udp 数据包：

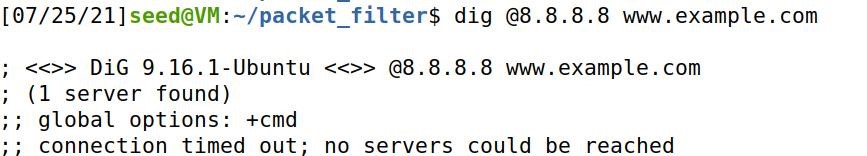


成功查询到 DNS 信息，说明数据包能够成功到达。

在 packet\_filter 文件夹编译代码并加载到内核中：



再次使用 dig 命令：

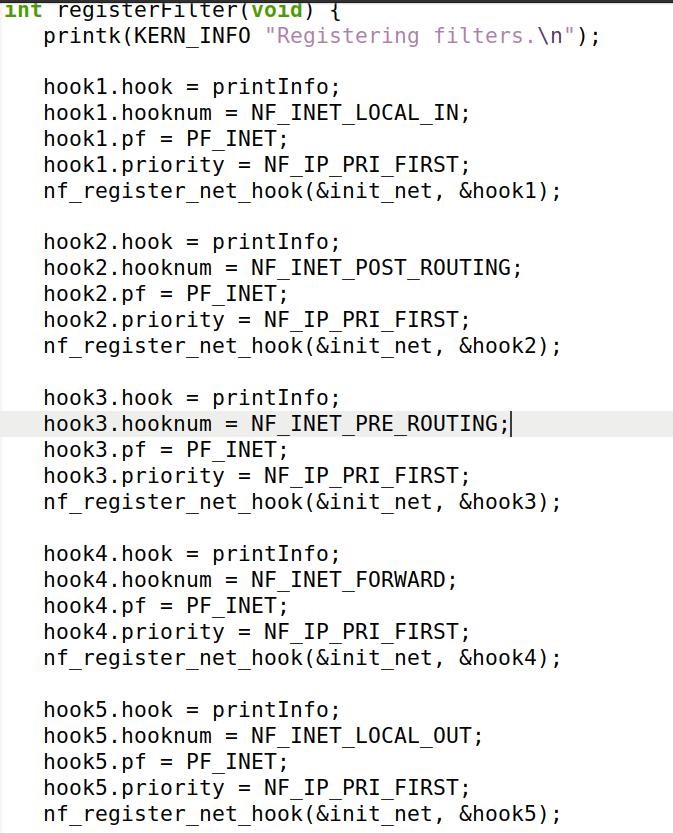


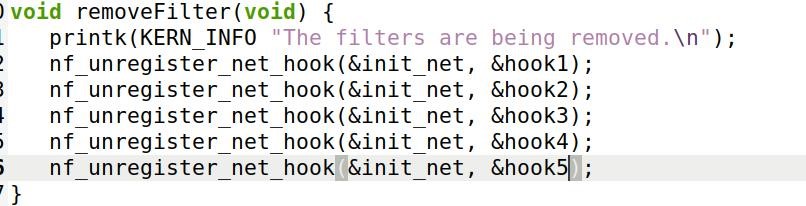
发现连接超时，无法达到服务器。说明防火墙成功工作。

**2.** 将 **printInfo** 函数挂接到所有的 **netfilter** 的 **hook** 上添加hook：



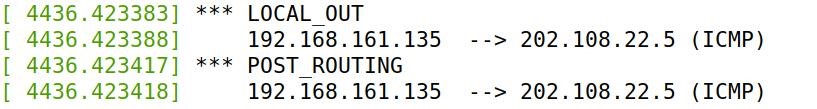
修改registerFilter和removeFilter函数：





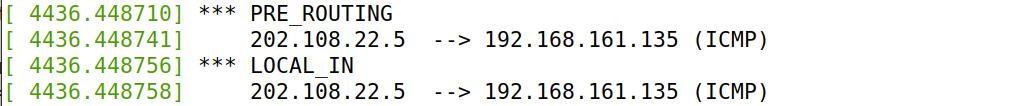
重新编译和加载（加载前删除以前的模块）后，我们可以 ping 下百度地址

202.108.22.5，如何 dmesg 查看输出：



可以看到本地的数据包发送时先是调用了 NF\_INET\_LOCAL\_OUT,然后是

NF\_INET\_POST\_ROUTING。



可以看到接收到发送给本地的数据包时先是调用 NF\_INET\_PRE\_ROUTING，然后是 NF\_INET\_LOCAL\_IN。

结合我们的实验现象以及查阅相关资料可知：NF\_INET\_PRE\_ROUTING 在数据包进入系统，进行 ip 校验后调用。然后进入路由代码，如果数据包是发送给本机，则调用 NF\_INET\_LOCAL\_IN；如果数据包需要转发则调用 NF\_INET\_FORWARD。本地产生数据包时先调用 NF\_IP\_LOCAL\_OUT ，进行路由选择处理，然后调用

NF\_IP\_POST\_ROUTING 后发送出去。

3.实现另外两个hook 再添加两个hook函数，代码如下：

|  |
| --- |
| unsigned int blockICMP(void \*priv, struct sk\_buff \*skb, const struct nf\_hook\_state \*state) { struct iphdr \*iph; if (!skb) return NF\_ACCEPT; iph = ip\_hdr(skb);  if (iph->protocol == IPPROTO\_ICMP) { printk(KERN\_WARNING "\*\*\* Dropping %pI4 (ICMP)\n", &(iph->daddr)); return NF\_DROP;  } return NF\_ACCEPT;  } |

这个函数负责阻止ICMP报文。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| unsigned int blockTCP(void \*priv, struct sk\_buff const struct nf\_hook\_state { struct iphdr \*iph; struct tcphdr \*tcph; u16 port = 23; if (!skb) return NF\_ACCEPT; iph = ip\_hdr(skb); if (iph->protocol == IPPROTO\_TCP) { tcph = tcp\_hdr(skb); if (ntohs(tcph->dest) == port){  printk(KERN\_WARNING "\*\*\*  &(iph->daddr), port); return NF\_DROP;  } } return NF\_ACCEPT;  } | \*skb,  \*state)  Dropping | %pI4 | (TCP), | port | %d\n", |

这个函数负责阻止目的端口为23的TCP报文。

在registerFilter函数中添加两个hook的调用（removeFilter函数中也

要有对应的注销hook，代码与前面task1.B.2类似，不再介绍）：

hook6.hook = blockICMP;

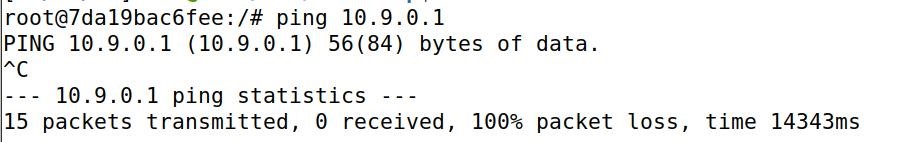
hook6.hooknum = NF\_INET\_LOCAL\_IN; hook6.pf = PF\_INET; hook6.priority = NF\_IP\_PRI\_FIRST; nf\_register\_net\_hook(&init\_net, &hook6);

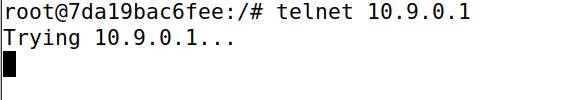
hook7.hook = blockTCP; hook7.hooknum = NF\_INET\_LOCAL\_IN; hook7.pf = PF\_INET; hook7.priority = NF\_IP\_PRI\_FIRST; nf\_register\_net\_hook(&init\_net, &hook7);

前面我们已经讨论过什么时候调用了什么hook，这里我们要阻止特定数据包

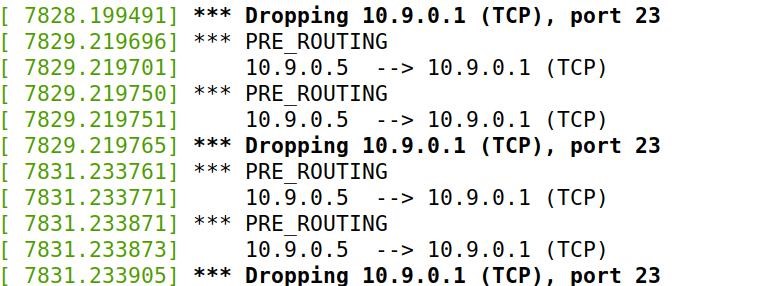
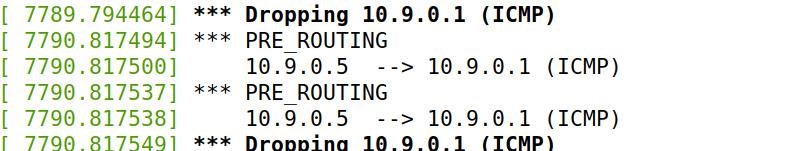
发给虚拟机，所以我们使用NF\_INET\_LOCAL\_IN在数据包发给本机时调用我们的 hook函数，阻止特定数据包接收。

在容器10.9.0.5中ping我们虚拟机的ip地址并进行Telnet连接：





发现 ping 和 Telnet 都无法成功。Dmesg 命令查看内核输出：



发现发送给 10.9.0.1 的 ICMP 数据包和目的地址为 23 的 TCP 数据包成功被阻

止。说明我们防火墙设置成功。

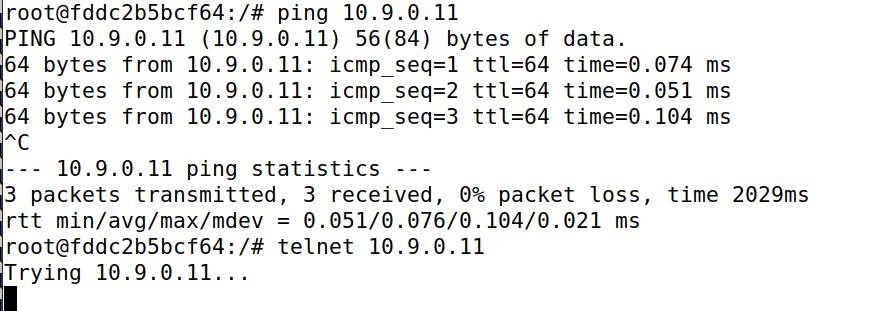
# Task 2: Experimenting with Stateless Firewall Rules

## Task 2.A: Protecting the Router

路由器容器上执行下列 iptables 命令：

iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT iptables -P OUTPUT DROP ¥ Set default rule for OUTPUT iptables -P INPUT DROP ¥ Set default rule for INPUT

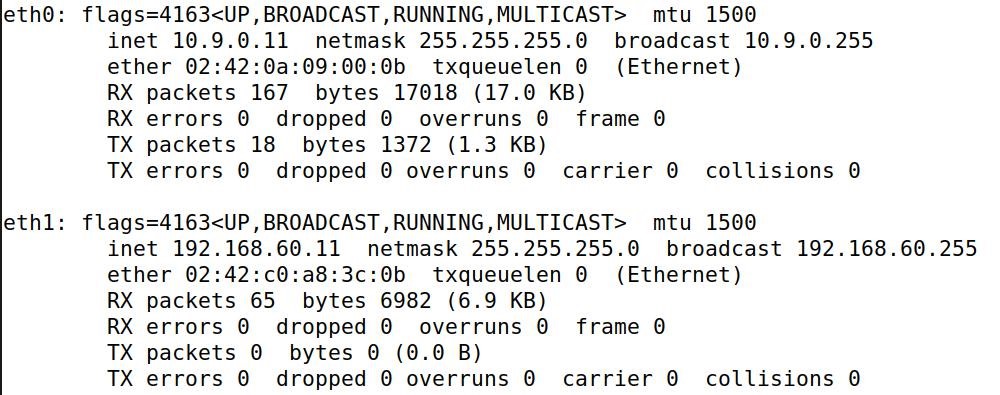
pdf 上 request 和 reply 写反了，需要交换过来。在 10.9.0.5 主机上 ping 路由器以及 Telnet 连接：



发现可以 ping 通，但是 Telnet 无法连接。这是因为我们在 iptables 规则中允许 icmp 请求报文输入，允许 icmp 响应报文输出，所以可以 ping 通。其他报文都不允许通过，所以无法 Telnet 成功。

## Task 2.B: Protecting the Internal Network

查看路由器接口：



可以看到外部接口为 eth0,内部接口为 eth1。防火墙要求如下：

1. 外部主机无法 ping 内部主机。 2. 外部主机可以 ping 通路由器。

1. 内部主机可以 ping 外部主机。
2. 应该阻止内部和外部网络之间的所有其他数据包。

设 iptables -P FORWARD DROP 阻止内部和外部之间所有数据包。但是路由器

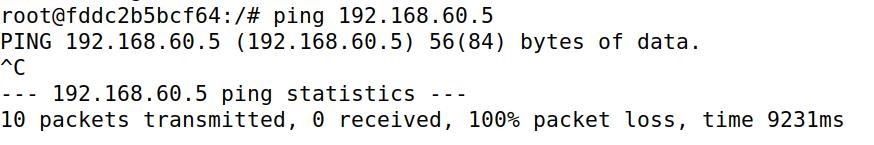
可以 ping 通。

可设 iptables -A FORWARD -p icmp --icmp-type echo-request -i eth1 -j ACCEPT 允许转发内部网络的 ICMP 请求报文。

可设 iptables -A FORWARD -p icmp --icmp-type echo-reply -i eth0 -j ACCEPT 允许转发外部网络的 ICMP 响应报文。这样就能使得内部主机可以 ping 外部主机，但是外部主机的请求报文无法转发，使得外部主机无法 ping 通内部主机。所以我们执行如下规则：

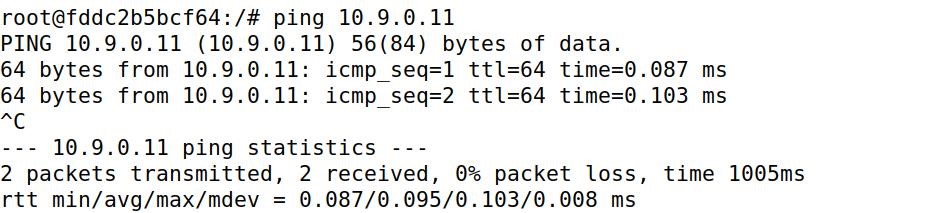
iptables -A FORWARD -p icmp --icmp-type echo-request -i eth1 -j ACCEPT iptables -A FORWARD -p icmp --icmp-type echo-reply -i eth0 -j ACCEPT iptables -P FORWARD DROP

### 在外部主机 10.9.0.5 中 ping 内部主机 192.168.60.5



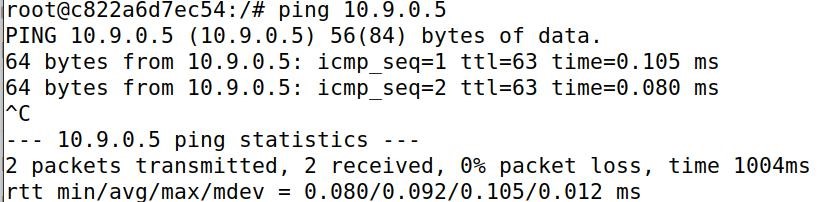
无法 ping 通。

在外部主机 **10.9.0.5** 中 **ping** 路由器地址 **10.9.0.11**

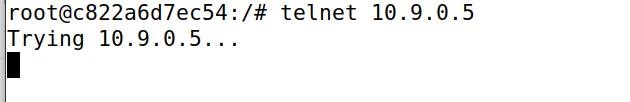


可以 ping 通。

在内部主机 **192.168.60.5** 中 **ping** 外部主机 **10.9.0.5**

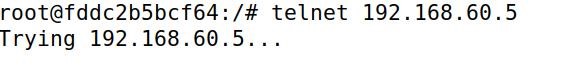


可以 ping 通。在内部主机 **192.168.60.5** 中向外部主机 **10.9.0.5** 建立 **Telnet**



无法建立 Telnet。

在外部主机 **10.9.0.5** 中向内部主机 **192.168.60.5** 建立 **Telnet**



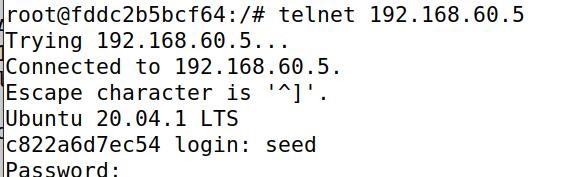
无法建立 Telnet。说明我们防火墙设置成功。

## Task 2.C: Protecting Internal Servers

过滤规则如下：

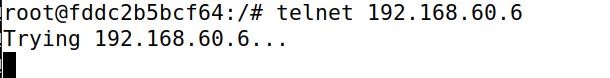
iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport 23 -j ACCEPT iptables -A FORWARD -o eth0 -p tcp -s 192.168.60.5 --sport 23 -j ACCEPT iptables -A FORWARD -i eth1 -p tcp -s 192.168.60.0/24 --dport 23 -j ACCEPT iptables -A FORWARD -o eth1 -p tcp -s 192.168.60.0/24 --sport 23 -j ACCEPT iptables -P FORWARD DROP

### 外部主机 10.9.0.5 访问内部主机 192.168.60.5：



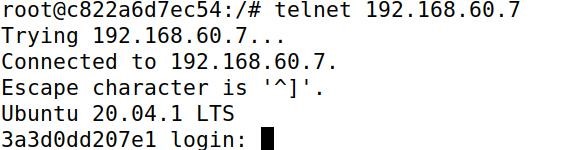
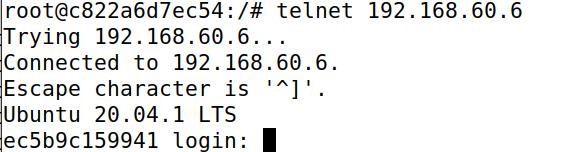
能够成功访问。

### 外部主机 10.9.0.5 访问其他内部主机 192.168.60.6：



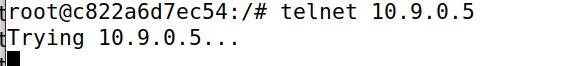
无法访问。

### 内部主机 192.168.60.5 访问内部主机 192.168.60.6 和 192.168.60.7：



都可以成功访问。

### 内部主机 192.168.60.5 访问外部主机 10.9.0.5：



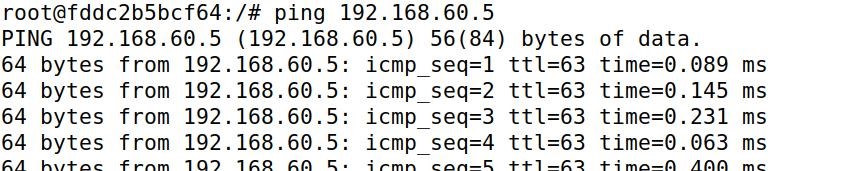
无法访问。符合要求，防火墙规则设置成功。

# Task 3: Connection Tracking and Stateful Firewall

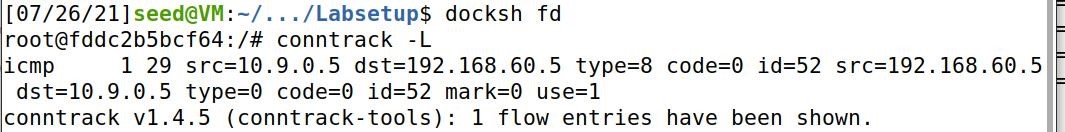
## Task 3.A: Experiment with the Connection Tracking

### （1）ICMP 实验

在主机 10.9.0.5 上执行命令 ping 192.168.60.5：

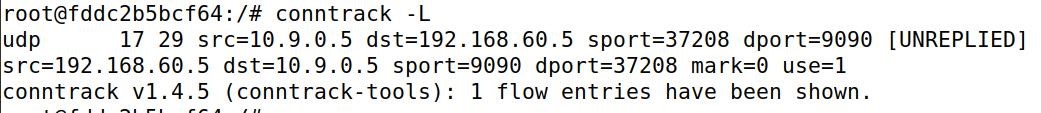


然后执行 conntrack -L 命令查看追踪信息：



发现 icmp 连接时间为 29 秒。

1. **udp** 实验在主机 192.168.60.5 上执行 nc -lu 9090，然后在 10.9.0.5 主机上执行 nc -u 192.168.60.5 9090 命令。在 10.9.0.5 主机上随便输入字符后快速执行 conntrack -L 命令。



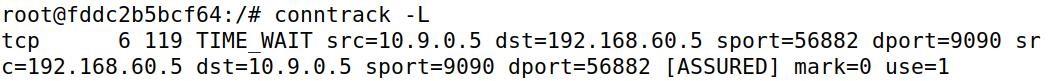
可以看到 udp 连接的时间也是 29 秒。

1. **Tcp** 实验在主机 192.168.60.5 上执行 nc -l 9090，然后在 10.9.0.5 主机上执行 nc 192.168.60.5 9090 命令。在 10.9.0.5 主机上随便输入字符后快速执行 conntrack -L 命令。





发现 tcp 连接保持时间约为 432000 秒。



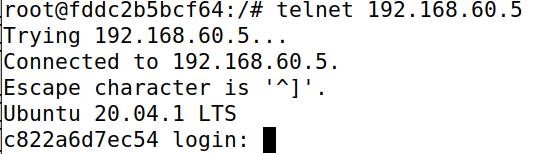
当结束连接时，看到 tcp 的连接时间为 119 秒。

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

使用连接追踪机制编写规则：

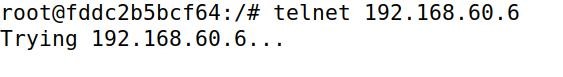
|  |
| --- |
| iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn -m conntrack  --ctstate NEW -j ACCEPT iptables -A FORWARD -p tcp -i eth1 -s 192.168.60.0/24 --dport 23 --syn -m conntrack  --ctstate NEW -j ACCEPT iptables -P FORWARD DROP |

### 外部主机 10.9.0.5 访问内部主机 192.168.60.5：



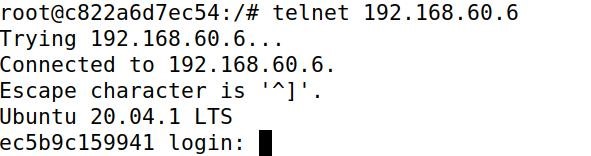
能够成功访问。

### 外部主机 10.9.0.5 访问其他内部主机 192.168.60.6：



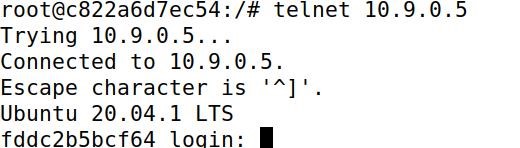
无法访问。

### 内部主机 192.168.60.5 访问内部主机 192.168.60.6：



可以成功访问。

### 内部主机 192.168.60.5 访问外部主机 10.9.0.5：



访问成功。符合要求，防火墙规则设置成功。

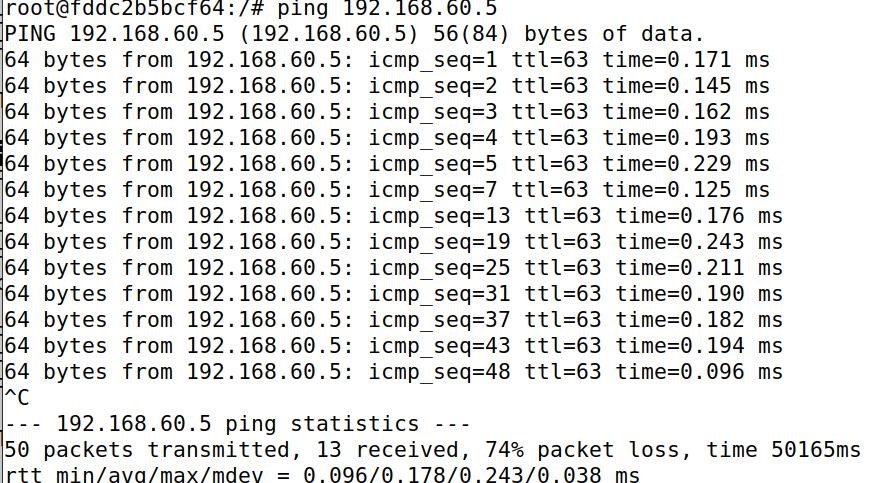
不使用连接跟踪机制的过滤规则检查比较简单，优点是处理速度快，缺点是不会考虑数据包的上下文，因此会导致不准确、不安全或复杂的防火墙规则；使用连接跟踪机制的过滤规则对数据包的状态进行检查，优点是能建立有状态防火墙，安全性更高，更准确，缺点是无法对数据包内容进行识别。

## Task 4: Limiting Network Traffic

限制可以通过防火墙的数据包数量，规则如下：



在 10.9.0.5 主机上 ping 196.168.60.5，结果如下：

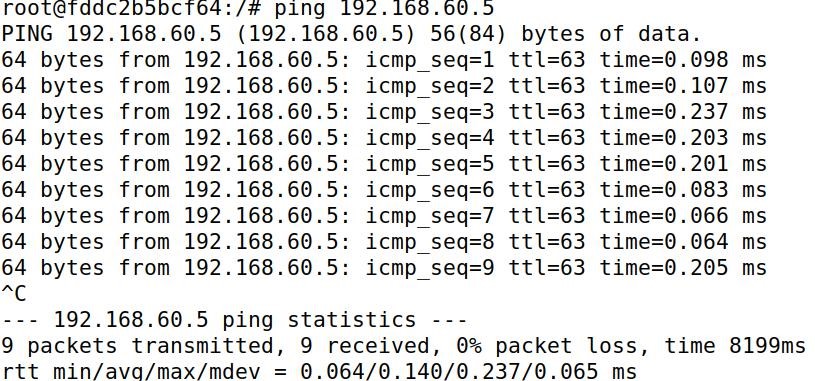


可以发现 ping 的过程中丢了很多包，导致收到包的速率降低，通过防火墙的数据包数量被限制。

不使用第二条规则时：



重新 ping 196.168.60.5，结果如下：



此时没有出现丢包，数据包没有得到限制。

实验结果说明需要第二条规则。这是因为如果不设置第二条规则，所有数据

包默认ACCEPT，不会发生丢包，也就不能限制数据包数量。

## Task 5: Load Balancing

### Using the nth mode (round-robin)

使用nth模式实现负载平衡,规则如下：

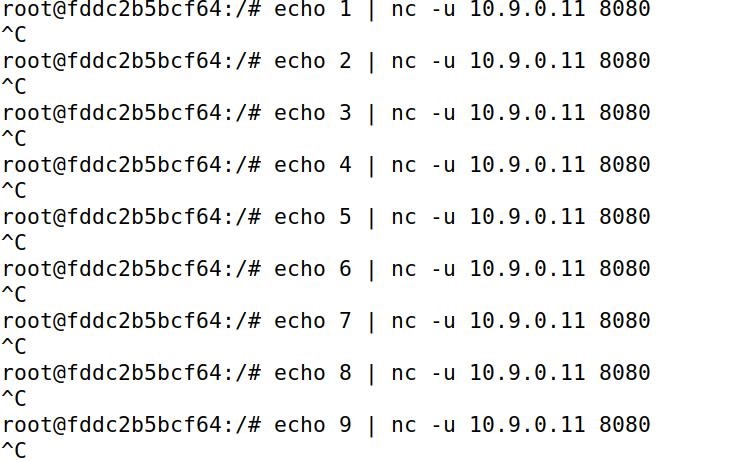
iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3

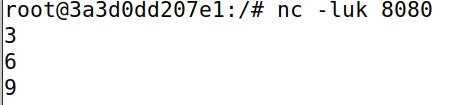
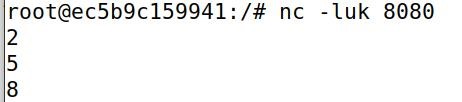
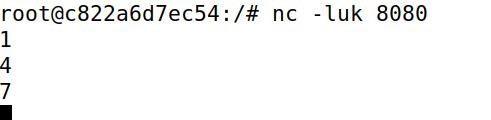
--packet 0 -j DNAT --to-destination 192.168.60.5:8080 iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 2

--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

这里第一条规则将每三个数据包中的第一个发送给192.168.60.5，然后第二条规则让剩下的两个中的数据包中的第一个发给192.168.60.6，第三条规则让剩下的一个数据包发给192.168.60.7。这样就使得三个数据包平均分给三个主机。结果如下：





在 10.9.0.5 主机上发送了 9 个数据包，三个主机按顺序接收到了数据包，最

后都收到三个数据包。

### Using the random mode

随机模式下的规则如下:

iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random

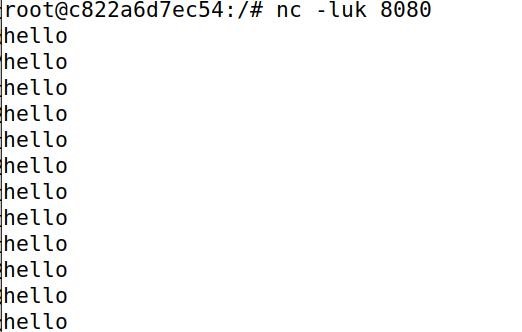
--probability 0.33 -j DNAT --to-destination 192.168.60.5:8080 iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random

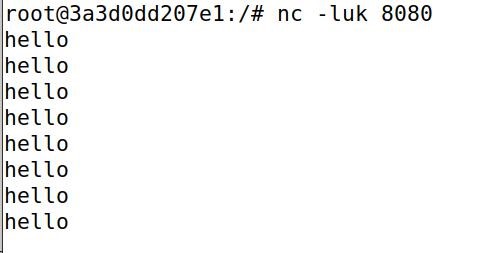
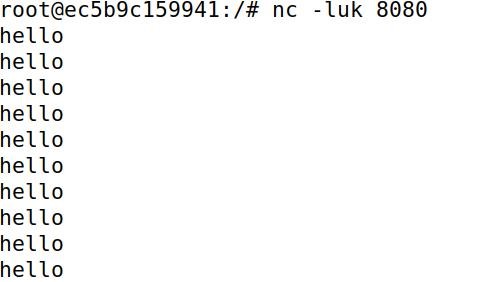
--probability 0.50 -j DNAT --to-destination 192.168.60.6:8080 iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT --to-destination

192.168.60.7:8080

第一条规则以1/3的概率将数据包发给192.168.60.5，如果第一条没有捕获，则以 1/2 的概率发给 192.168.60.6,如果第二条也没捕获，则发给 192.168.60.7，这样三个主机收到数据包的概率都是1/3。

10.9.0.5主机上发送30个数据包，结果如下：





三个数据包分别收到 12、10、8 个数据报，概率基本一致，数据包总数越大，数量越相同。实验成功。