**Firewall Exploration Lab**

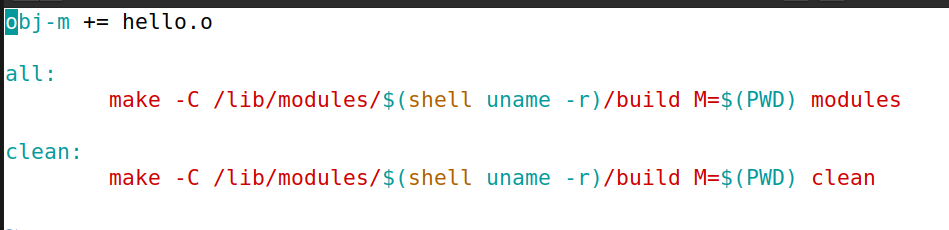
**Task 1: Implementing a Simple Firewall**

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

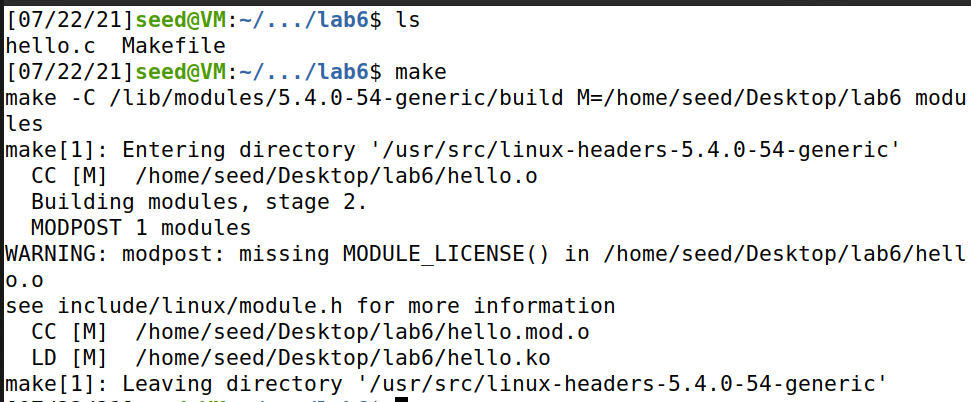
第一个实验需要我们进行简单的内核编译

下面是一个简单的可加载内核模块。 它打印出“Hello World！” 加载模块时； 当模块从内核中移除时，它会打印出“Bye-bye World！”。 消息不会打印在屏幕上； 它们实际上被打印到 /var/log/syslog 文件中。 您可以使用“dmesg”查看消息

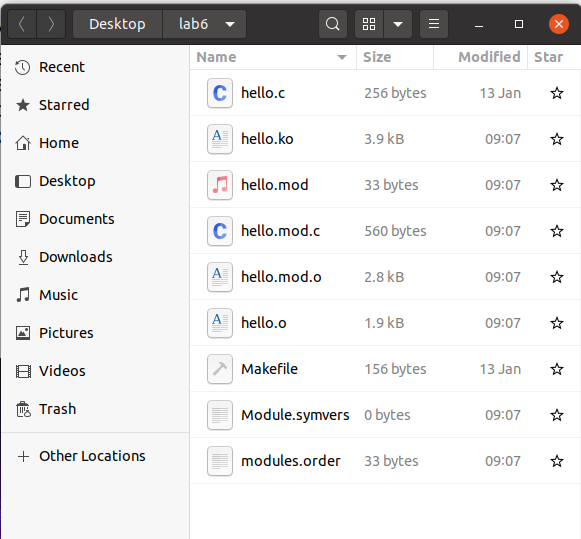
首先我们需要创建 Makefile，它包含以下内容（该文件包含在实验室设置文件中）。 只需输入make，上面的程序就会被编译成一个可加载的内核模块（如果将以下内容复制并粘贴到Makefile中，请确保用制表符替换make命令之前的空格）

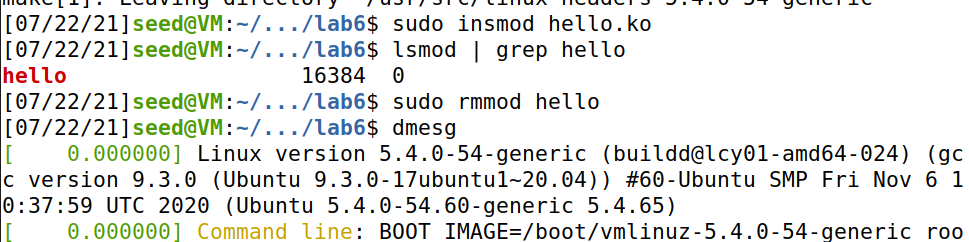


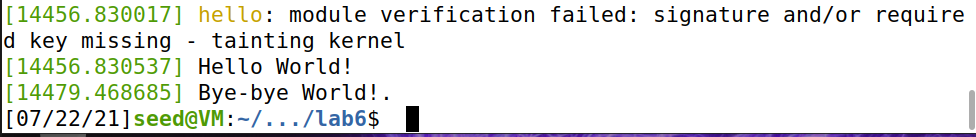
使用make命令编译内核



生成的内核模块在 hello.ko 中。 您可以使用以下命令加载模块、列出所有模块并删除模块。 此外，您可以使用“modinfo hello.ko”来显示有关 Linux 内核模块的信息。



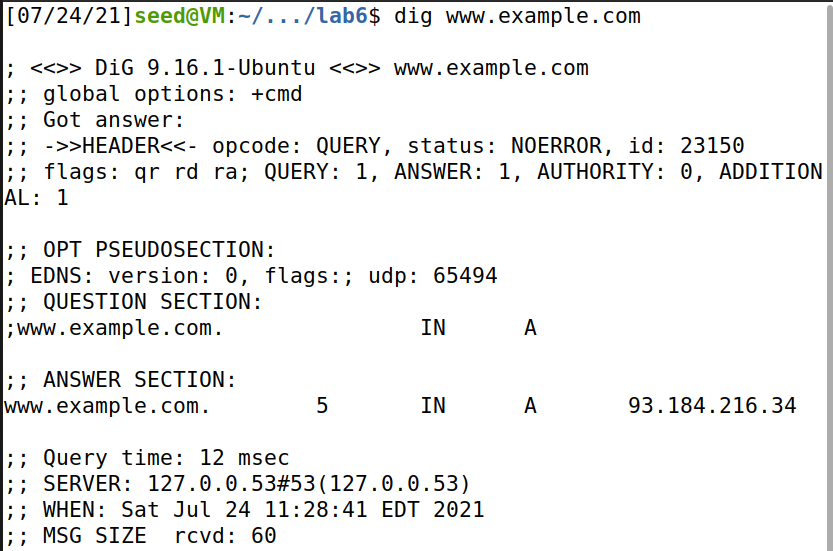




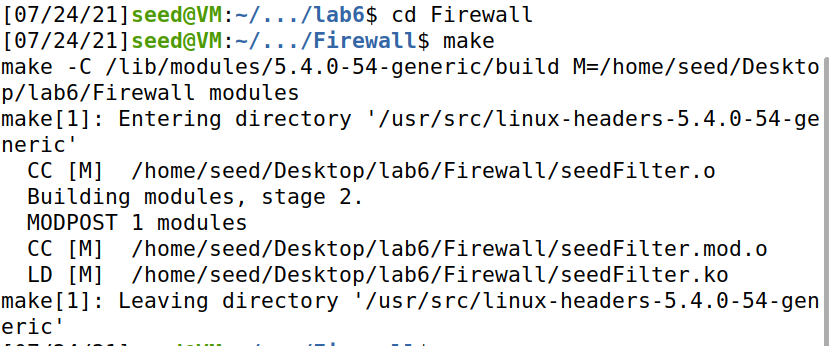
**Task 1.B: Implement a Simple Firewall Using Netfilter**

1. **Compile the sample code using the provided Makefike**

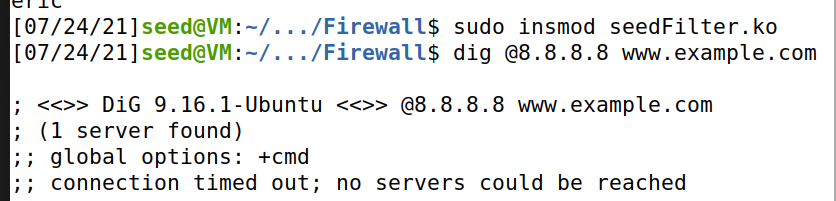
我们选择dig www.example.com 来获取所需的DNS信息**。**



**创建文件夹，保证完整路径中不含有空格，利用make命令编译可装载内核模块如下。**

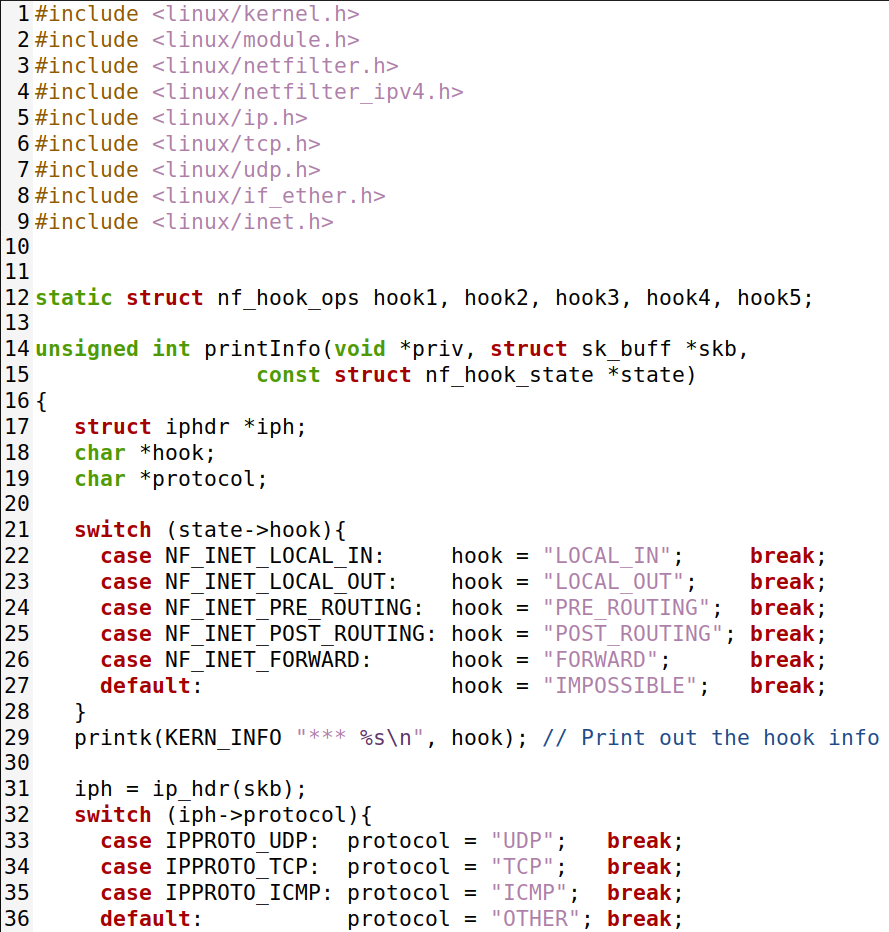


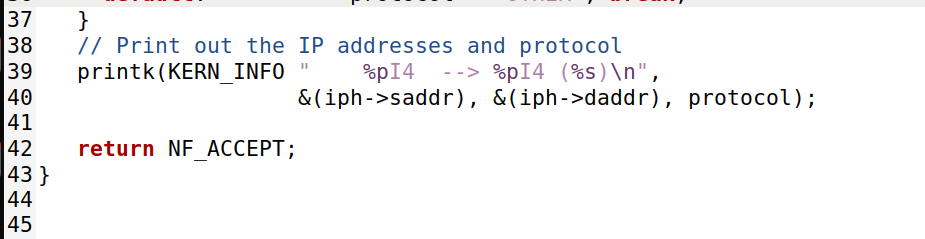
**利用insmod命令插入内核模块，在主机上利用dig查询DNS如下，无法获得相关信息**

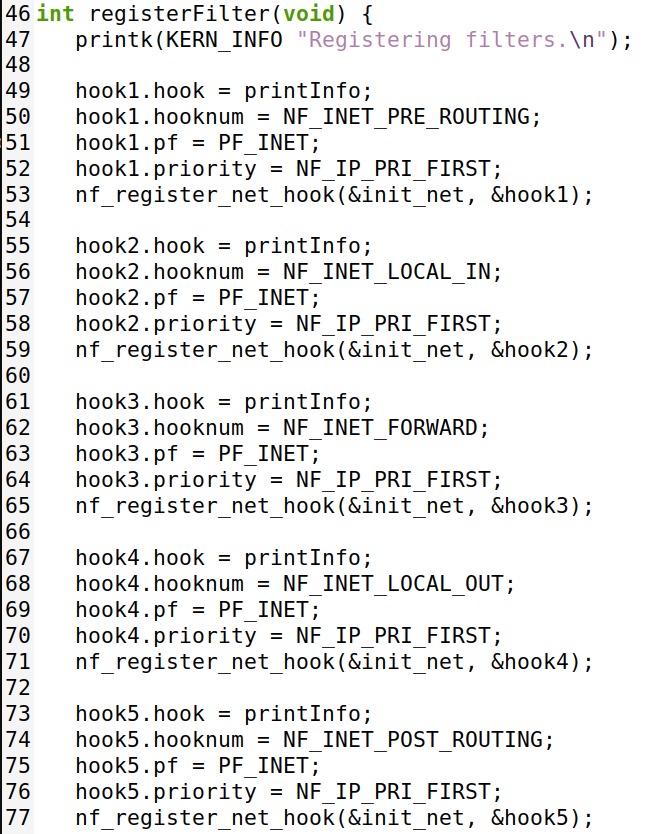


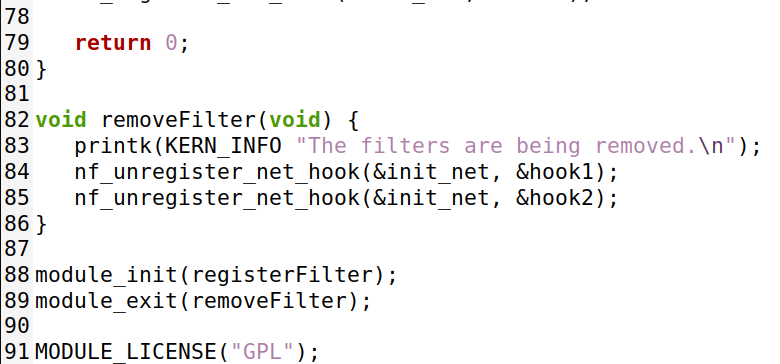
1. **Hook the printInfo function to all of the netfilter hooks**

**修改seedFilter.c，代码如下：**

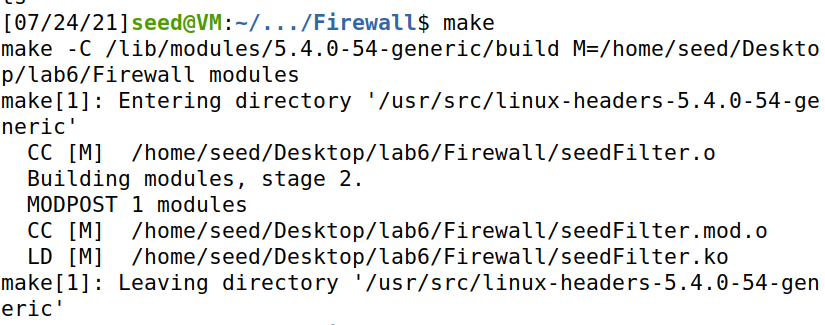


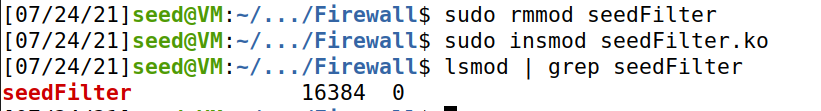




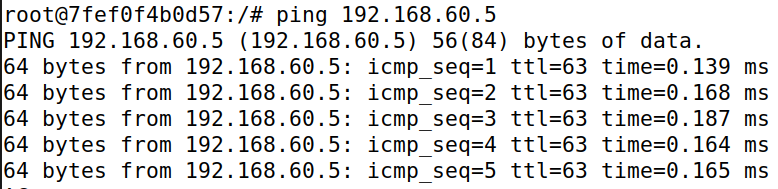


**利用make命令装载内核模块，并且利用insmod命令插入内核模块如下。**

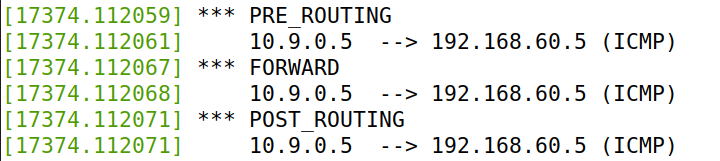




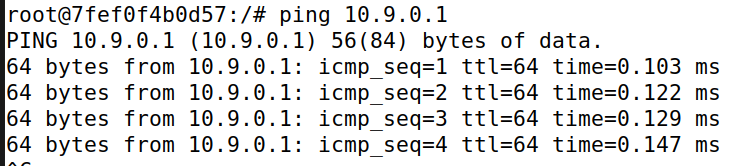
**在用户主机是ping 内网主机，得到结果如下，可知能够连接**



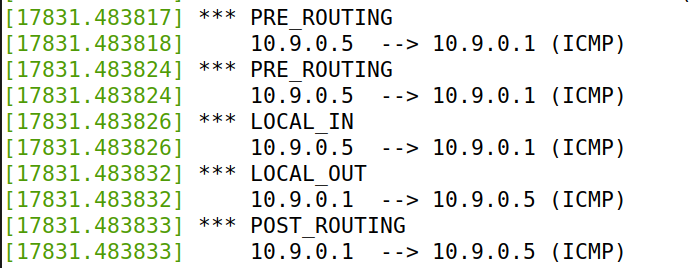
**利用dmesg命令查看/var/log/syslog文件中的信息，得到的结果如下**



**在用户主机上ping攻击者主机，得到结果如下，确认可以连接**



**利用dmesg命令查看/var/log/syslog文件中的信息，得到的结果如下**



**根据以上实验结果可知：**

**NF\_IP\_PRE\_ROUTING在数据包刚进入主机进行处理的时候调用；**

**NF\_IP\_LOCAL\_IN在确认数据包的目的地址为本机时调用；**

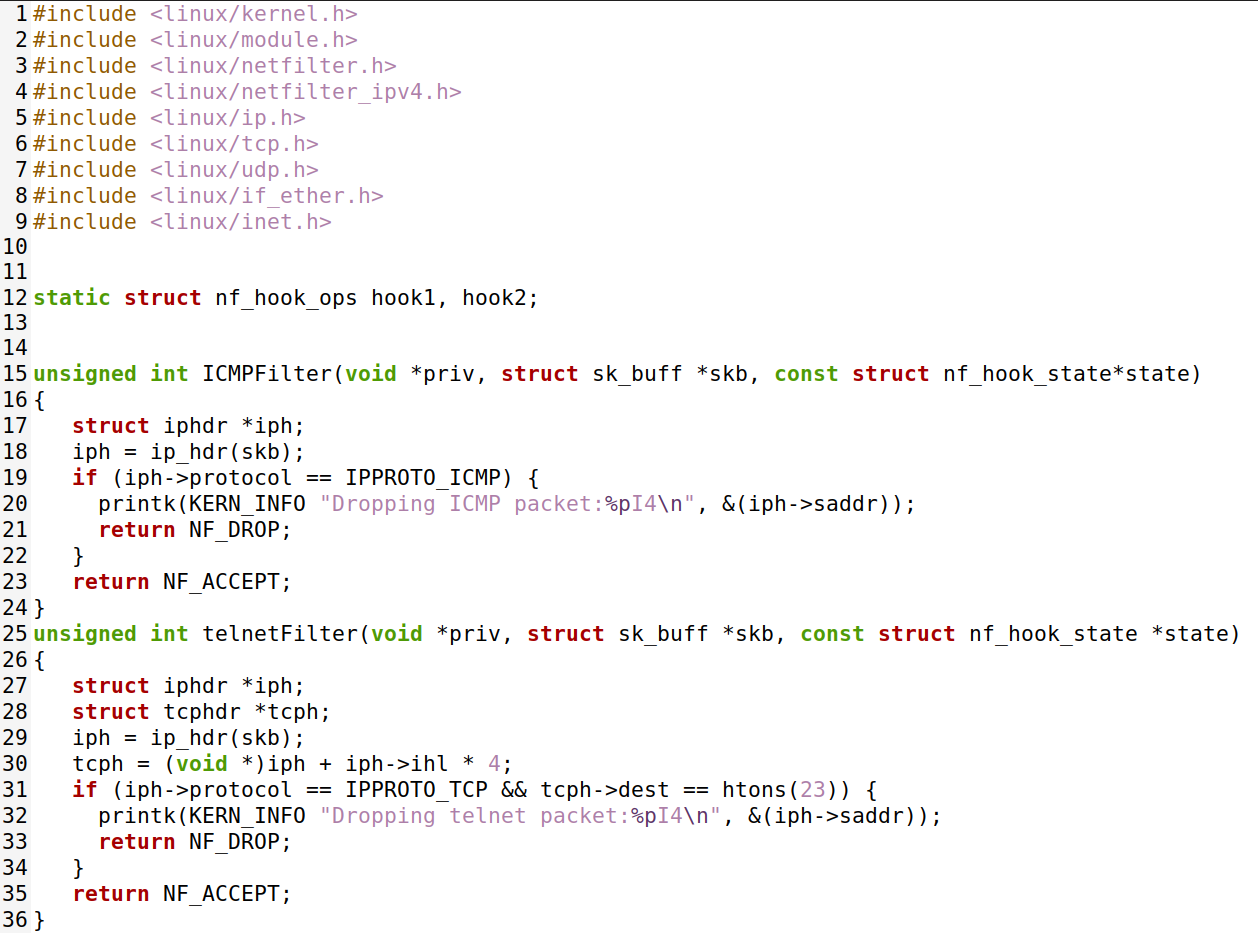
**NF\_IP\_FORWARD在要数据包通过主机进行转发的时候调用；**

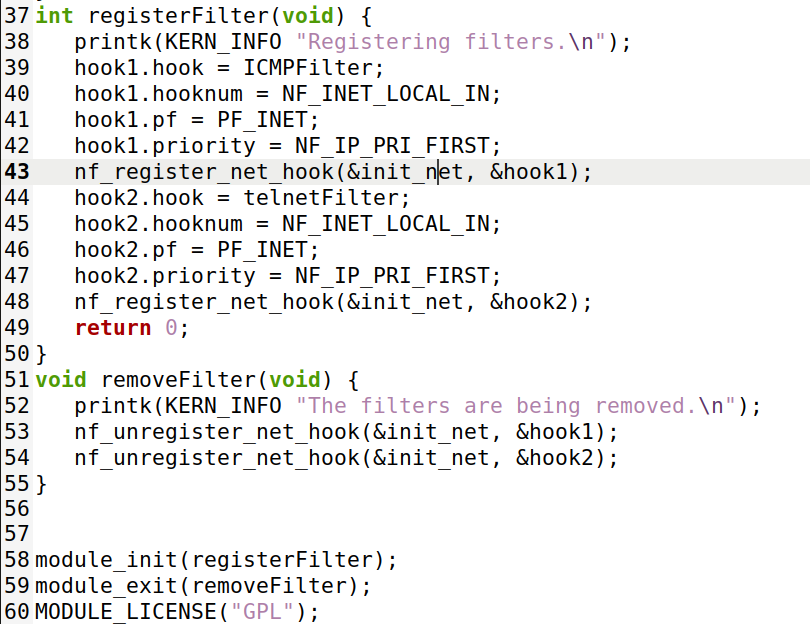
**NF\_IP\_LOCAL\_OUT在确认数据包的源地址为本机的时候调用；**

**NF\_IP\_POST\_ROUTING在数据包将离开主机进行处理时调用。**

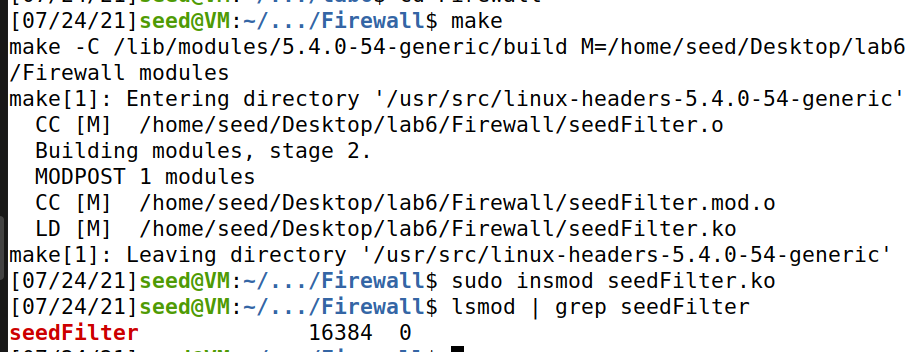
1. **Implement two more hooks**

**再次修改seedFilter.c文件，代码如下：**

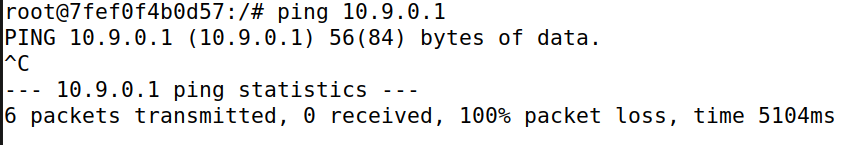




**利用make命令编译可装载内核模块，并利用insmod命令插入内核模块**



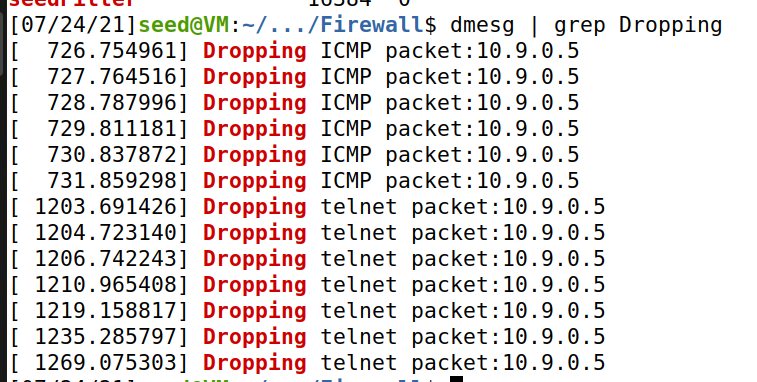
**在用户主机上Ping攻击者主机，得到结果如下，可知无法连接**



**在用户主机上telnet远程连接攻击者主机，可知无法连接**



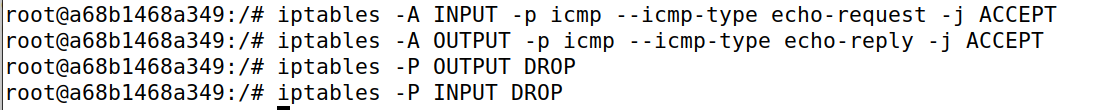
**利用dmesg命令查看/var/log/syslog文件中的信息，得到结果如下。可知ICMP和telnet报文都被丢弃**



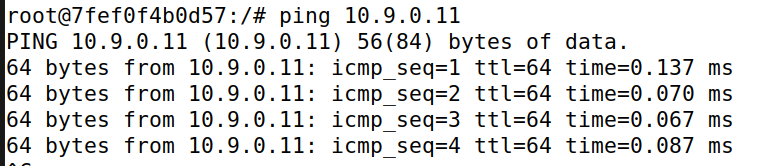
**Task 2: Experimenting with Stateless Firewall Rules**

**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**



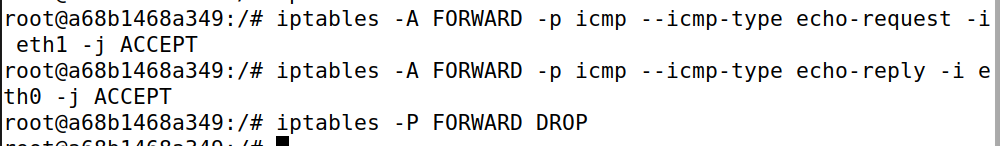
**在用户主机上telnet远程连接路由器，得到结果如下，可知连接失败**



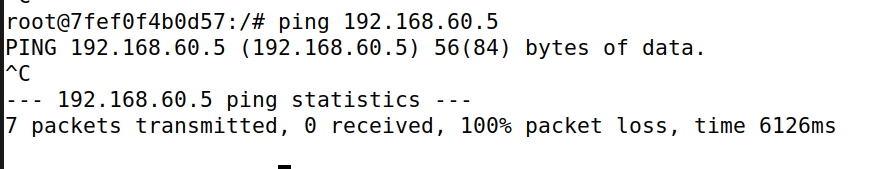
**该现象的原因是路由器的过滤规则只允许icmp请求报文输入和icmp响应报文输入，ping报文可以进行传输，而telnet的报文无法进行传输。**

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

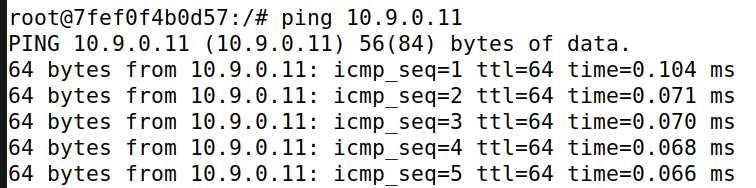
**在路由器上使用iptables命令设置规则如下：**



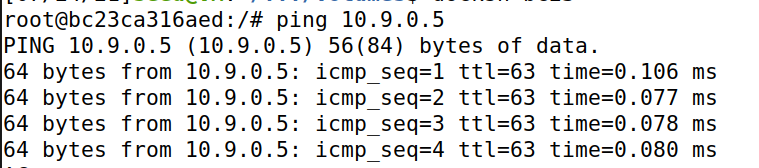
**在用户主机上ping内网主机192.168.60.5，得到结果如下，可知无法连接**



**在主机上ping路由器，发现可以连接**



**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，可知可以连接**



**在用户主机上telnet远程连接内网主机192.168.60.5，连接失败**

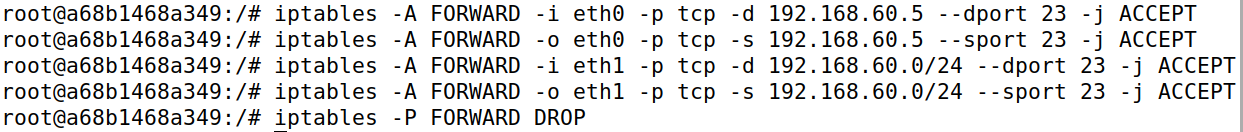


**在IP地址为192.168.60.5的内网主机telnet远程用户主机，连接失败**

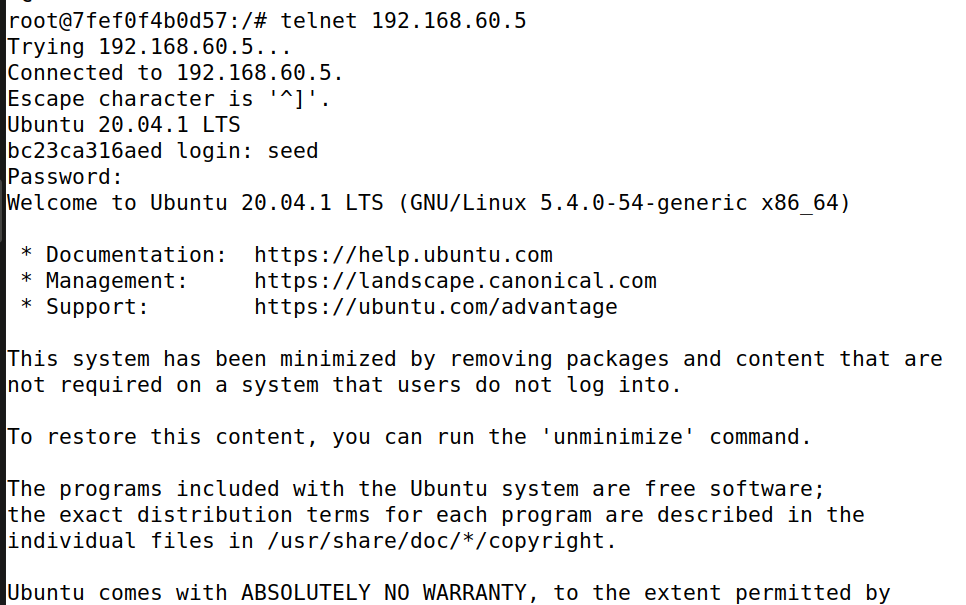


**Task 2.C: Protecting Internal Servers**

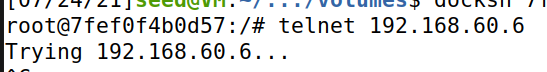
**在路由器上利用ptables命令，创建过滤规则如下。**



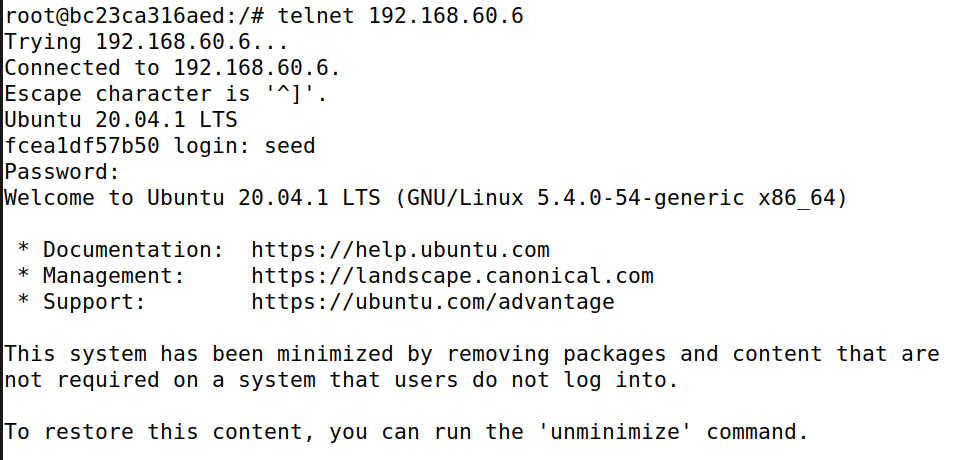
**在用户主机上telnet远程内网主机192.168.60.5**



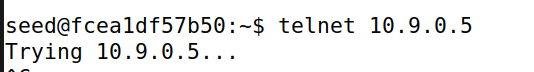
**在用户主机上telnet远程连接内网主机192.168.60.6连接失败**



**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，连接成功**



**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，可知连接失败**

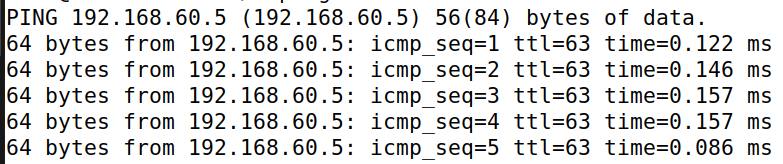


**Task 3: Connection Tracking and Stateful Firewall**

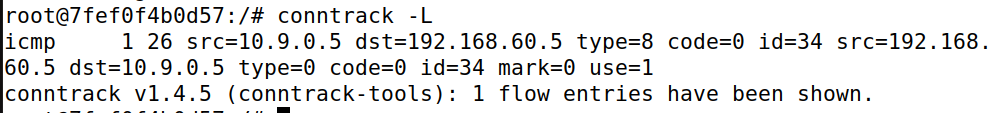
**Task 3.A: Experiment with the Connection Tracking**

**ICMP experiment**

**在用户主机10.9.0.5上ping 192.169.60.5**

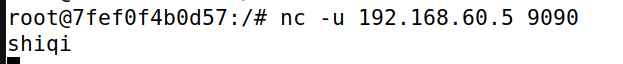


**之后使用conntrack -L检查路由器中的追踪信息，结果如下，ICMP 的连接时间约为27s**



**UDP experiment**

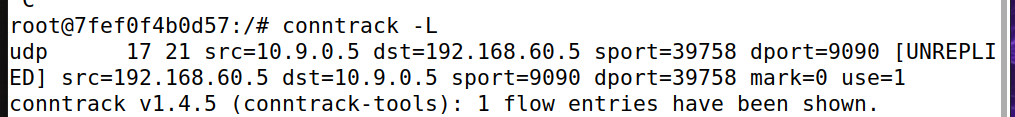
**在用户主机10.9.0.5上发送UDP报文**



**在192.168.60.5上开启netcat服务，监听结果如下**



**在路由器上利用connntracl -L实现连接跟踪，结果如下UDP的连接时间大约为21s**



**TCP experiment**

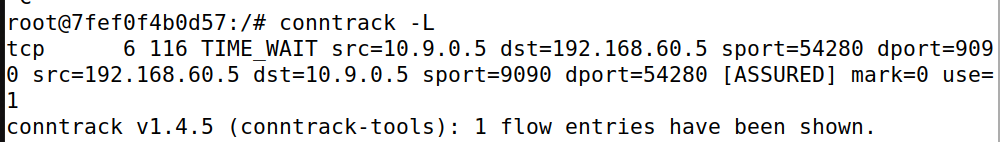
**在用户主机上利用TCP远程连接192.168.60.5，并发送如下消息**



**在192.168.60.5上，开启TCP服务监听结果如下**

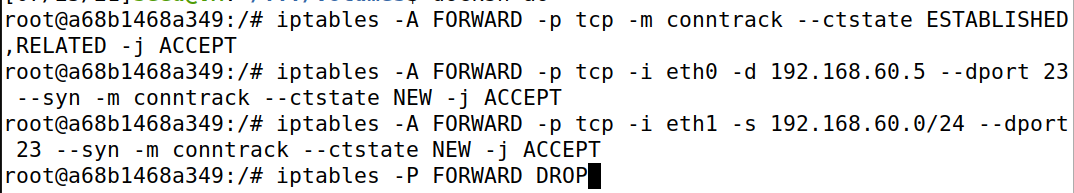


**在路由器上利用connntracl -L实现连接跟踪，结果如下，TCP的连接时间大约为116s**

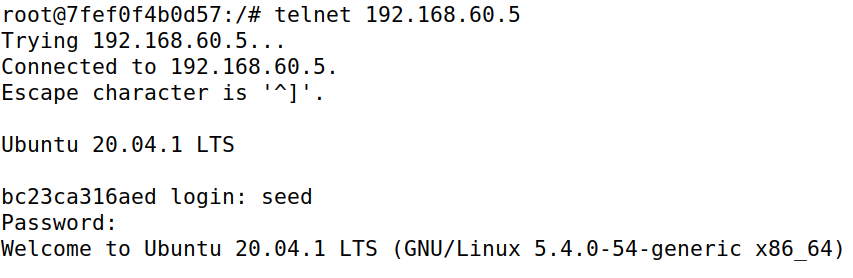


**Task 3.B: Setting Up a Sateful Firewall**

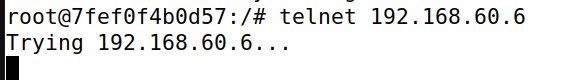
**在路由器上利用iptables命令追踪连接并设置过滤规则**



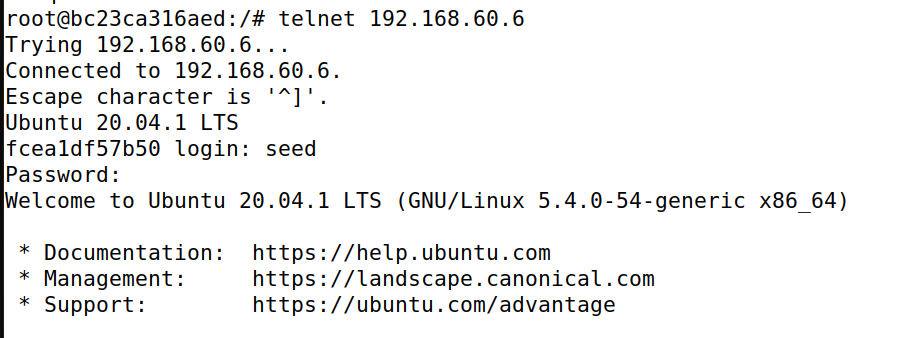
**在用户主机上telnet远程连接内网192.168.60.5，得到结果如下，连接成功**



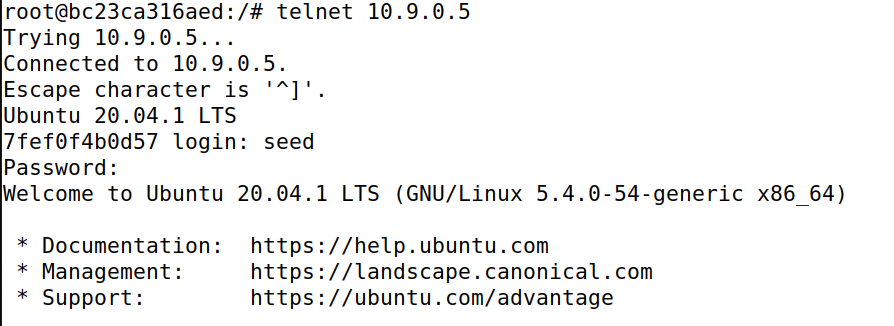
**在用户主机上telnet远程连接内网主机192.168.60.6，得到结果如下，可知连接失败**



**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，可知连接成功**



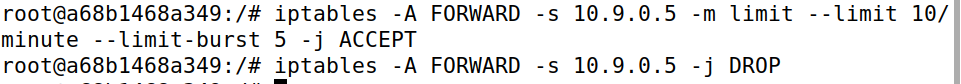
**在IP地址为192.168.60.5的内网主机上telent远程连接用户主机，得到结果如下，可知连接成功**



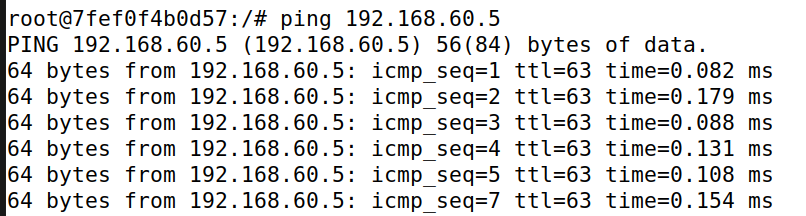
**将本次实验与task 2.C进行对比可以发现，不利于连接跟踪机制的过滤规则仅对数据包的首部进行检查，其优点是处理速度快，缺点是无法定义精细的规则，不适合复杂的访问控制；而利用连接跟踪机制的过滤规则对数据包的状态也进行检查，其优点是能够定义更加严格的规则、应用范围更广、安全性更高，缺点是无法对数据包内容进行识别。**

**Task 4: Limiting Network Traffic**

**利用iptables命令限制经过防火墙的包的数量，限制规则如下**



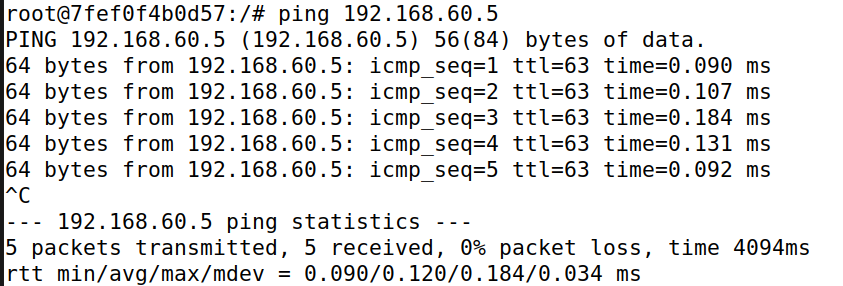
**在用户主机上ping内网主机192.168.60.5，得到结果如下。可知能够连接，但部分报文因流量限制而丢失**



**在路由器上利用iptables命令修改 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，且无报文丢失**

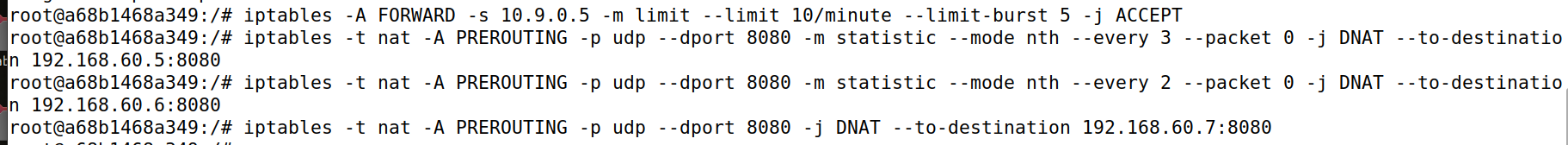


**通过对比发现两者之间的差距，区别在于第一次过滤规则中多了第二条规则，出现该现象的原因是路由器的转发链的默认规则为ACCEPT，即使超过流量限制，报文根据默认规则也可以进行传输，会使过滤失败。因此第二条规则是必要的。**

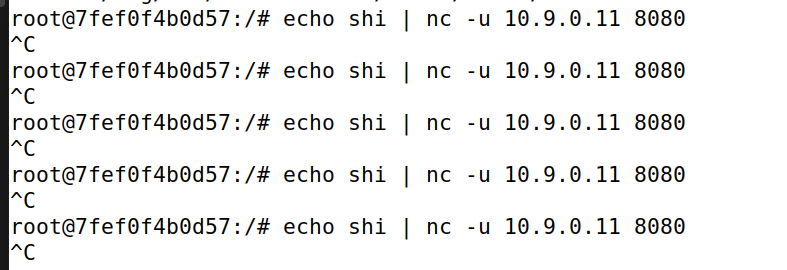
**Task 5: Load Balancing**

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

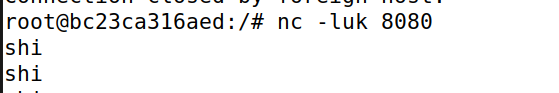
**在路由器上利用iptables命令，采用nth模式创建负载均衡规则如下**



**在用户主机上向路由器发送UDP数据包如下：**



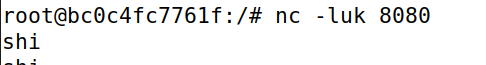
**在服务器192.168.60.5上监听8080端口，结果如下**



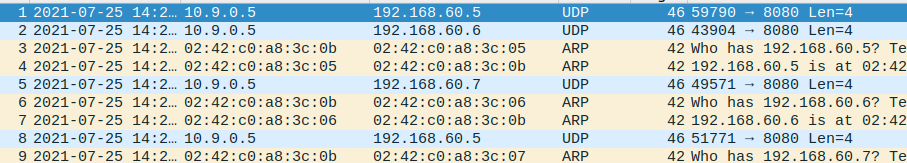
**在服务器192.168.60.6上监听808端口，结果如下：**



**在服务器192.168.60.7上监听808端口，结果如下：**

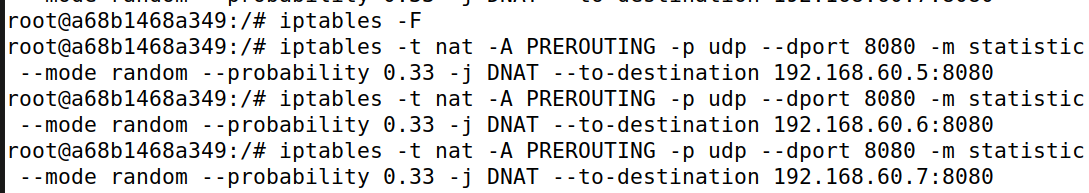


**通过wireshark抓包可以发现，UDP报文发送成功，根据设置的规则按顺序发送至对应的服务器**

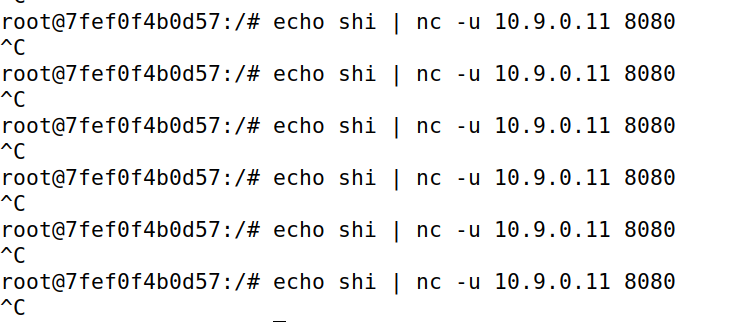


**Using the random mode**

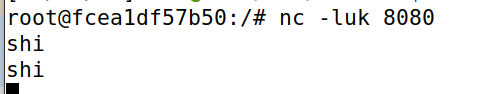
**修改过滤规则如下：**



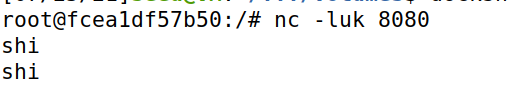
**同上，再次发送数据包**



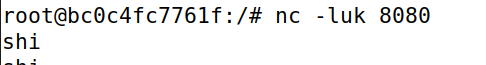
**在服务器192.168.60.5上监听8080端口，结果如下**



**在服务器192.168.60.6上监听808端口，结果如下：**



**在服务器192.168.60.7上监听808端口，结果如下：**



**通过wireshark抓包可以发现，UDP报文发送成功，根据设置的规则随机发送至对应的服务器**

