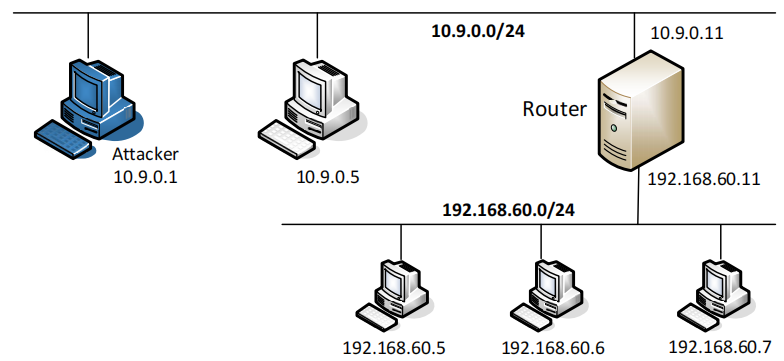
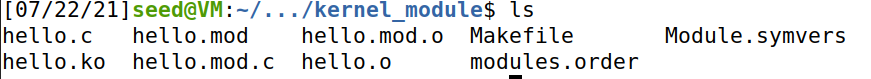
57118130 王嘉麟 lab6

网络结构：

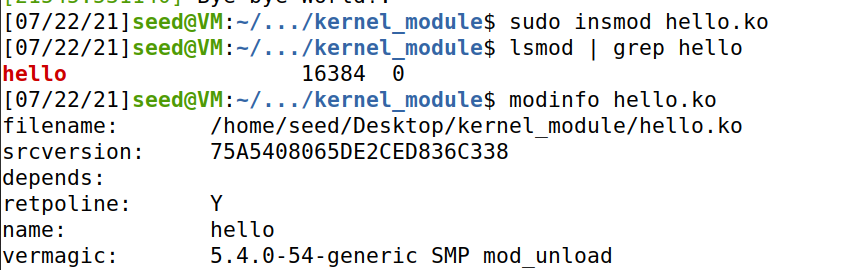


**Task1.A**

将kernel\_model文件夹复制到一个没有空格的路径下，直接make，得到以下文件



使用#sudo insmod hello.ko插入hello.ko模块到LKM，通过#lsmod | grep hello ;#modinfo hello.ko查看模块信息



使用#sudo rmmod hello移除hello.ko模块

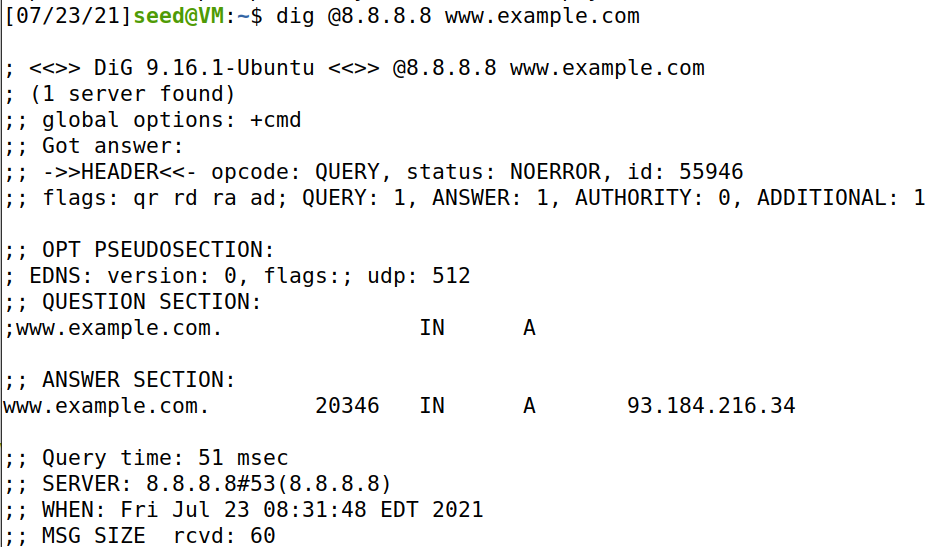
使用#dmesg查看内核日志，确认模块插入并生效，以及最后被移除



**Task1.B**

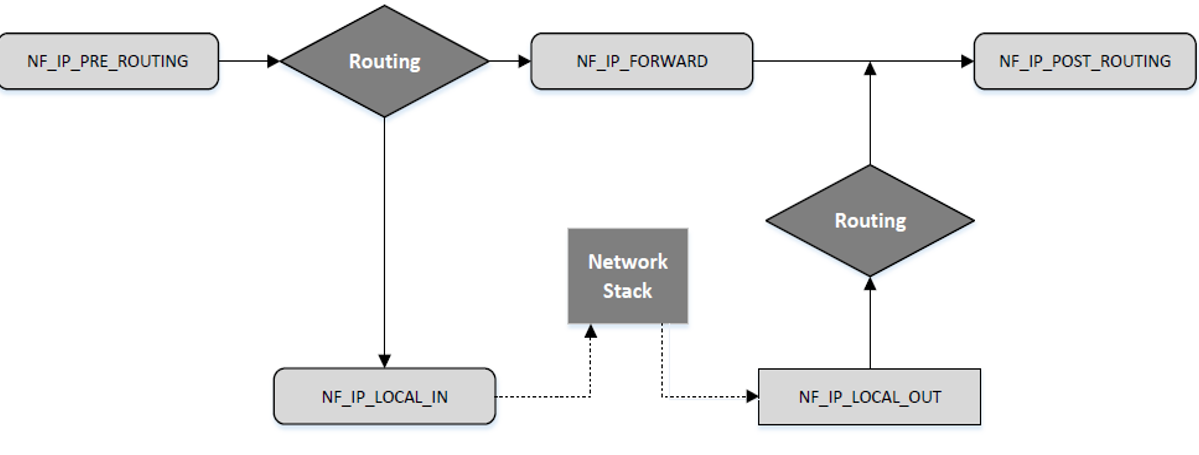
首先编译seedFilter.c，然后使用#sudo insmod seedFilter.ko加载到LKM中，下图为加载前后执行#dig @8.8.8.8 [www.example.com](http://www.example.com)的对比，发现Filter加载后dig命令失效，没有回显

由于hook2绑定到了blockUDP函数，同时将所有路由后的报文给了blockUDP函数，在这个函数中，进行了IP地址，UDP协议和53端口的检测，然后DROP掉这种报文，实现屏蔽DNS回显





五种过滤器之间的关系

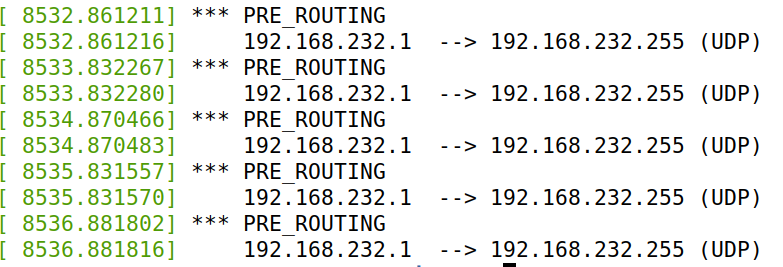


下面演示将printInfo一一和五种过滤器勾上

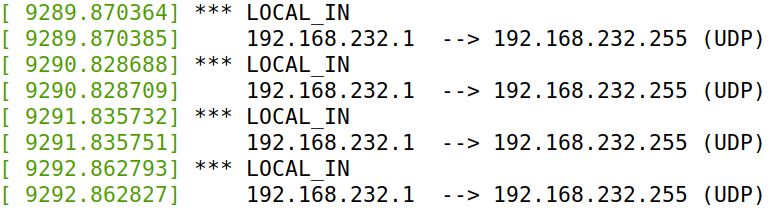
通过#ifconfig查看以太网对外IP地址



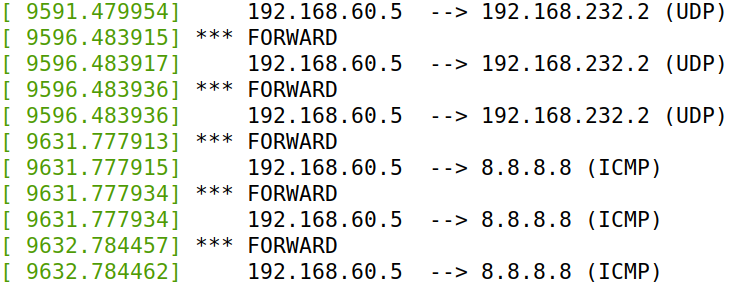
Hook到PRE\_ROUTING上的时候



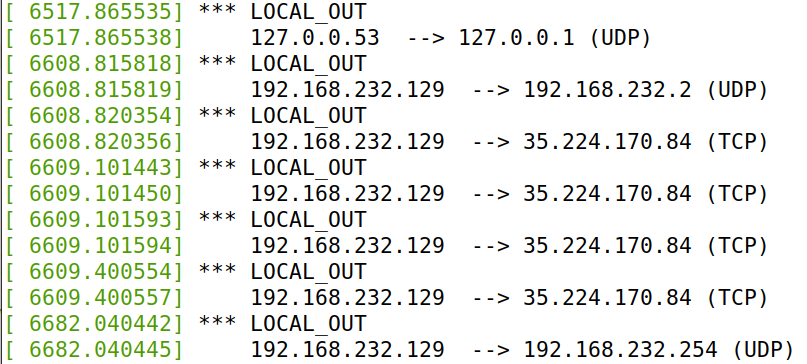
Hook到LOCAL\_IN上的时候



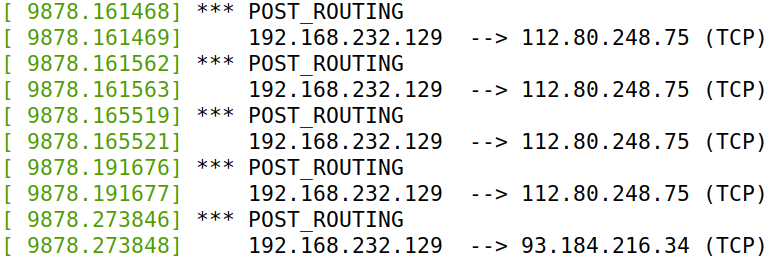
Hook到FORWARD上的时候，同时登录到192.168.60.5上，执行ping和dig等命令，触发FORWARD



Hook到LOCAL\_OUT上的时候



Hook到POST\_ROUTING上的时候



阻止其他主机ping VM或者telnet连接到VM，netfilter hook选择LOCAL\_IN，在报文到达VM网络栈之前过滤之，编写如下两个函数，然后挂载到hook上去

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

const struct nf\_hook\_state \*state)

{

struct iphdr \*iph;

char ip[16] = "10.9.0.1";

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\_ICMP) {

if (iph->daddr == ip\_addr){

printk(KERN\_WARNING "\*\*\* Dropping (ICMP) from %pI4 to 10.9.0.1\n", &(iph->saddr));

return NF\_DROP;

}

}

return NF\_ACCEPT;

}

tcp头的获取没有现成函数，需要通过(void\*)iph + iph->ihl\*4的方式获取

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

const struct nf\_hook\_state \*state)

{

struct iphdr \*iph;

struct tcphdr \*tcph;

u16 port = 23;

char ip[16] = "10.9.0.1";

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\_TCP) {

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

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

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

return NF\_DROP;

}

}

return NF\_ACCEPT;

}

在registerFilter函数中，添加如下，hook到相应hook上去

hook3.hook = blockICMP;

hook3.hooknum = NF\_INET\_LOCAL\_IN;

hook3.pf = PF\_INET;

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

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

hook4.hook = blockTELNET;

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

hook4.pf = PF\_INET;

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

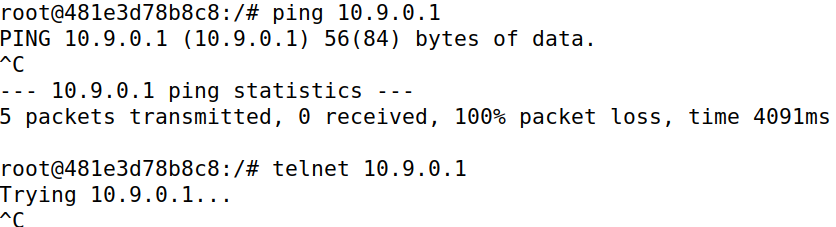
nf\_register\_net\_hook(&init\_net, &hook4);

在removeFilter函数中，添加如下，删除对应hook

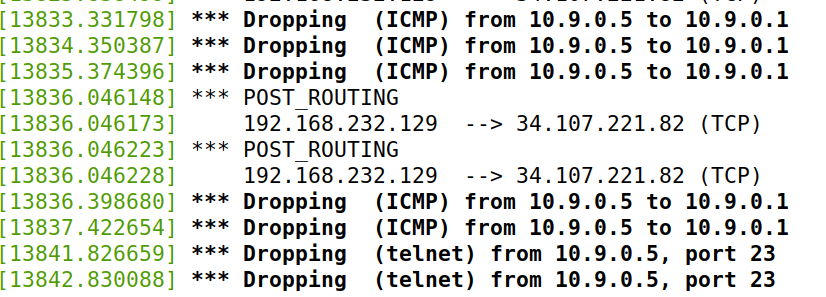
nf\_unregister\_net\_hook(&init\_net, &hook3);

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

先加载模块到LKM中，然后登录到10.9.0.5上执行#ping 10.9.0.1和#telnet 10.9.0.1测试hook是否生效，10.9.0.5上没有回显，如下



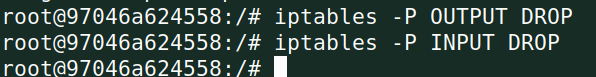
在10.9.0.5上没有回显的基础上，在VM上执行#dmesg查看系统日志如下



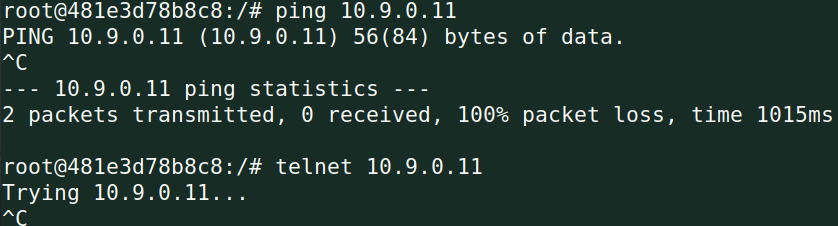
可以看到，对应信息被成功拦截并显示

**Task2.A**

在router上执行了#iptables -P OUTPUT DROP和#iptables -P INPUT DROP后，实现了将路由器的INPUT和OUTPUT报文全部丢弃



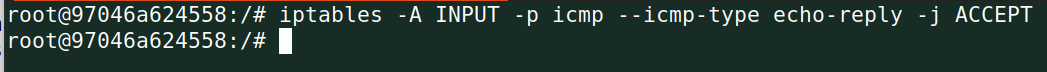
在10.9.0.5上不论telnet还是ping都无法成功，



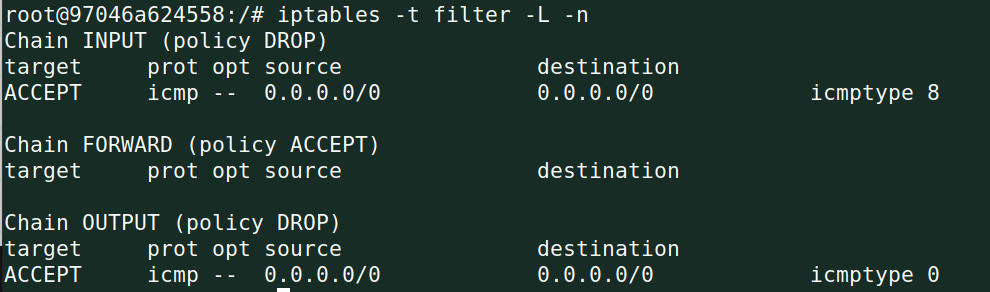
在router上执行

#iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT和

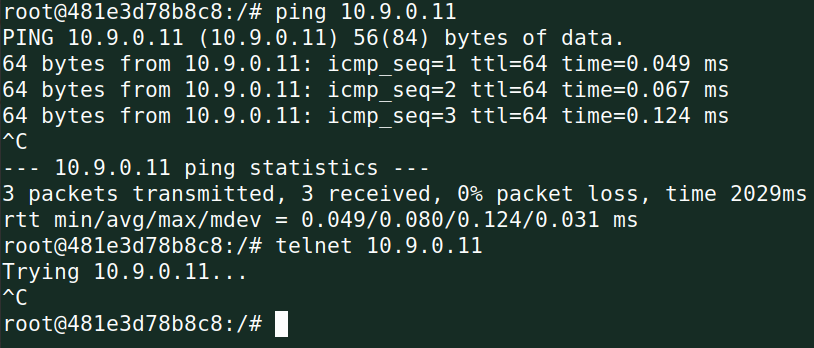
#iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT，第一条指令在INPUT上接收icmp echo-request报文，第二条指令在OUTPUT上允许icmp echo-reply报文



通过#iptables -t filter -L -n查看当前过滤策略



然后在10.9.0.5上ping 10.9.0.11，成功，telnet 10.9.0.11则失败



**Task2.B**

在路由器上设立防火墙规则，实现

1. 外网的主机不能ping通内网主机
2. 外网的主机只能ping通router
3. 内网的主机能ping通外网的主机
4. 其他所有内网外网之间的报文会被阻隔

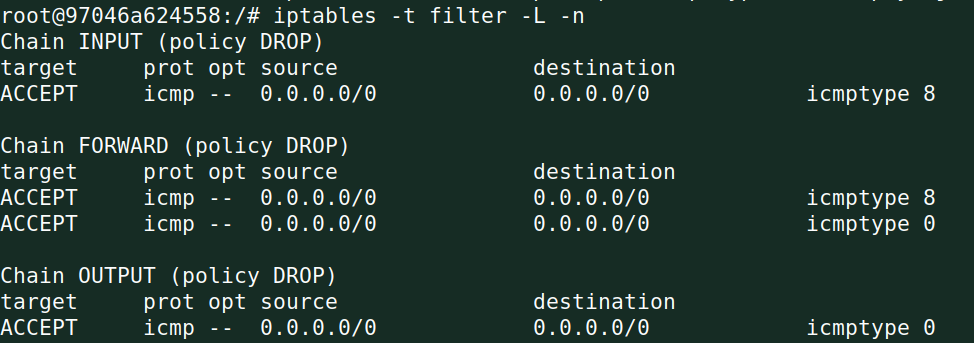
在Task2.A的基础上，增加以下三条规则

#iptables -P FORWARD DROP

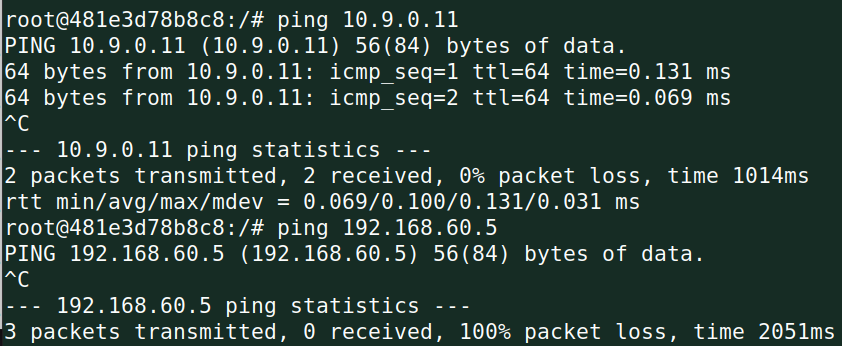
#iptables -A FORWARD -p icmp --icmp-type echo-request -i eth1 -j ACCEPT

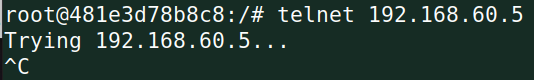
#iptables -A FORWARD -p icmp --icmp-type echo-reply -o eth1 -j ACCEPT

完成配置后，用#iptables -t filter -L -n查看配置，如下

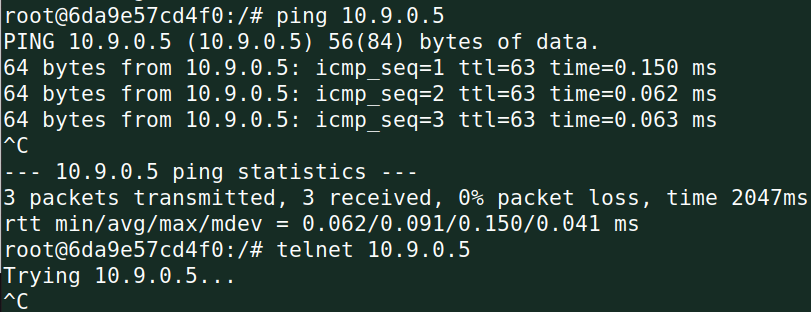


在外网主机10.9.0.5上测试如下，可以ping通路由器，不能ping通内网主机192.168.60.5，telnet 192.168.60.5被拦截





在内网主机192.168.60.5上测试如下，成功ping通外网主机10.9.0.5，同时telnet流量被拦截



**Task2.C**

在路由器上设立防火墙规则，实现

1. 外网的主机只能通过telnet访问内网的192.168.60.5，而不能访问其他主机
2. 外网的主机不能访问内网的其他主机
3. 内网主机互相之间可以访问
4. 内网主机不能访问外网主机
5. 不使用连接跟踪技术

在router上使用如下指令配置防火墙

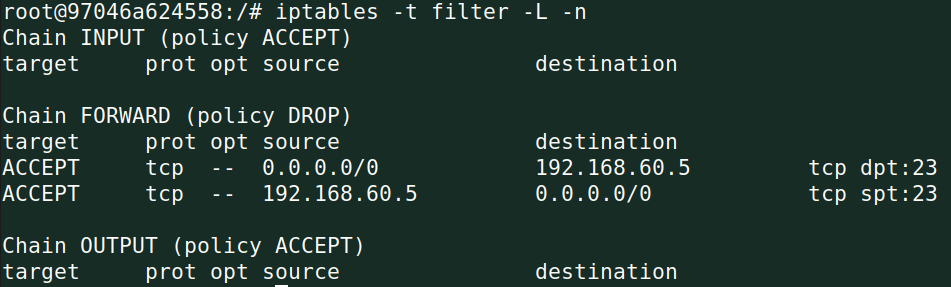
#iptables -P FORWARD DROP

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

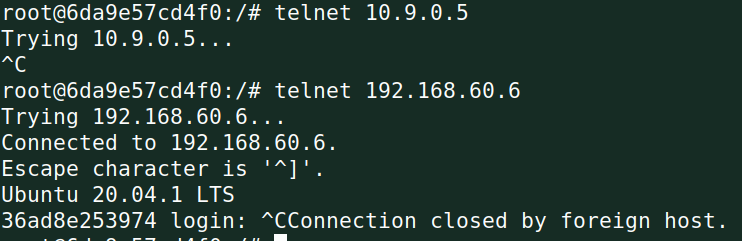
# iptables -A FORWARD -s 192.168.60.5 -p tcp --sport 23 -j ACCEPT

先丢弃所有路由报文，然后允许IP和端口都指定的报文流动，但是很显然，这样的防护无法防护synflood，tcp reset等攻击

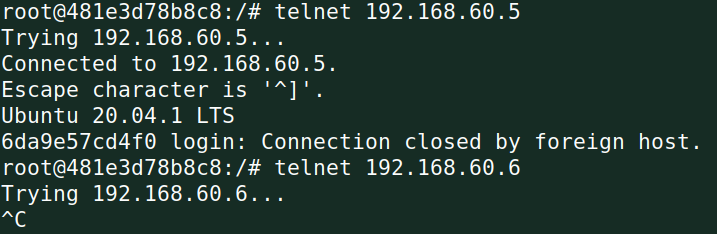
查看配置如下



在内网主机192.168.60.5上验证配置，telnet 10.9.0.5失败，telnet 192.168.60.6成功，因为到192.168.60网段内的路由就是本机网卡，不会被router上的配置干扰

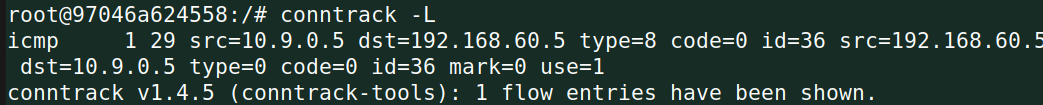


在外网主机10.9.0.5上验证配置，telnet 192.168.60.5成功，telnet 192.168.60.6失败



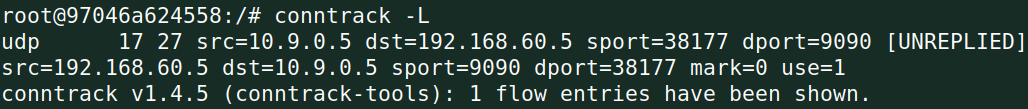
**Task3.A**

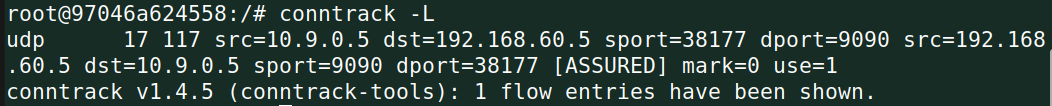
在10.9.0.5上执行#ping 192.168.60.5，然后在router上执行#conntrack -L查看连接跟踪信息，发现除了源和目的地址，icmp的code和type，id也被记录，时长为30秒，随后被清除

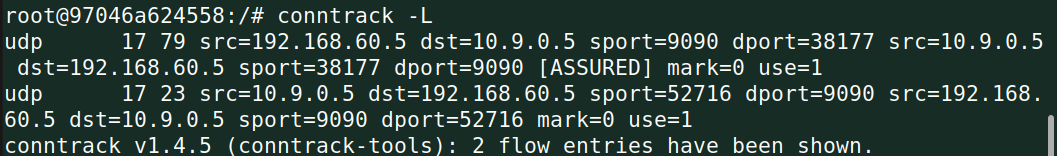


在192.168.60.5上执行#nc -lu 9090，然后在10.9.0.5上执行# nc -u 192.168.60.5 9090，

最后在router上用#conntrack -L查看连接跟踪信息如下，发现一开始在10.9.0.5上输入信息后只会出现一个UNREPLIED的记录，时长为30秒，如果在192.168.60.5上回复，则信息会转变为ASSURED的记录，时长刷新为120秒，期间中断双方连接，信息也不会消失，120s后自动消失，如果这期间再次进行连接，则会变为两个记录

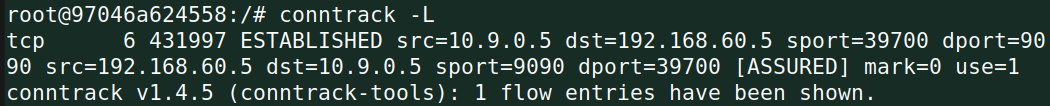






在192.168.60.5上执行#nc -l 9090，然后在10.9.0.5上执行# nc 192.168.60.5 9090，

最后在router上用#conntrack -L查看连接跟踪信息如下，在tcp连接建立的时候就监听432000秒，只要连接建立就处于ASSURED状态



**Task3.B**

使用连接跟踪技术重写Task2.C，但是这次内网主机能访问外网主机

在router上使用以下4条指令配置防火墙

#iptables -P FORWARD DROP

#iptables -A FORWARD -p tcp -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT //允许对主机192.168.60.5 23端口的syn访问，用来建立TCP连接

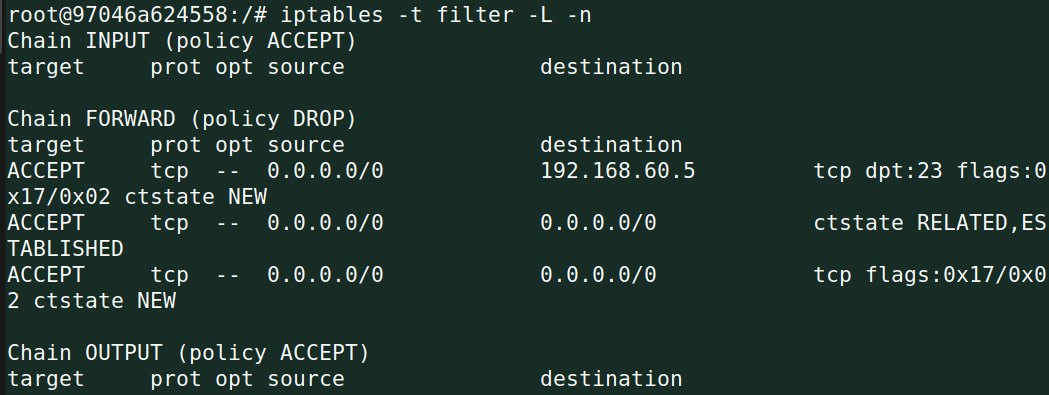
# iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

//允许已经建立的TCP连接

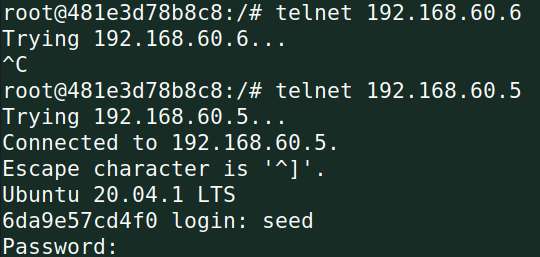
# iptables -A FORWARD -p tcp -i eth1 --syn -m conntrack --ctstate NEW -j ACCEPT

//允许192.168.60网段的SYN报文

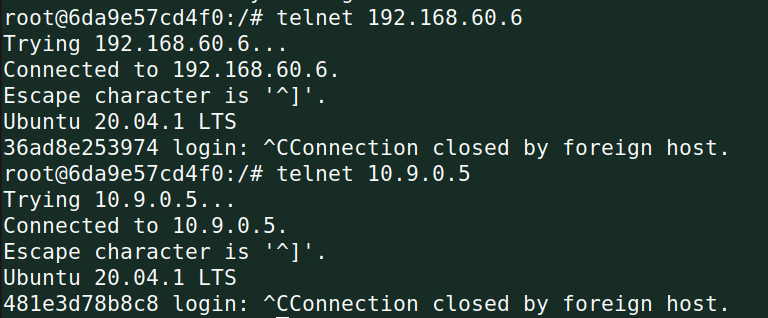
配置完成后，查看配置如下



在外网主机10.9.0.5上验证防火墙，telnet 192.168.60.6不成功，telnet 192.168.60.5成功



在192.168.60.5上验证防火墙，telnet 192.168.60.6成功，telnet 10.9.0.5成功



如果不使用信息跟踪技术的话，想要达成同样的目的，需要在2.C的基础上增加两条

#iptables -P FORWARD DROP

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

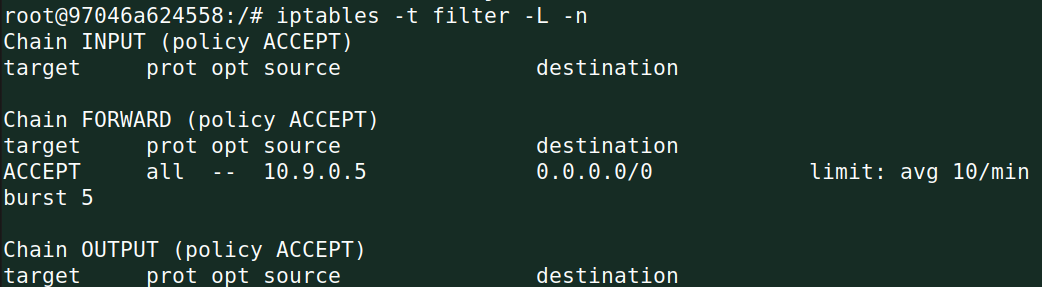
# iptables -A FORWARD -s 192.168.60.5 -p tcp --sport 23 -j ACCEPT

# iptables -A FORWARD -i eth1 -p tcp --dport 23 -j ACCEPT

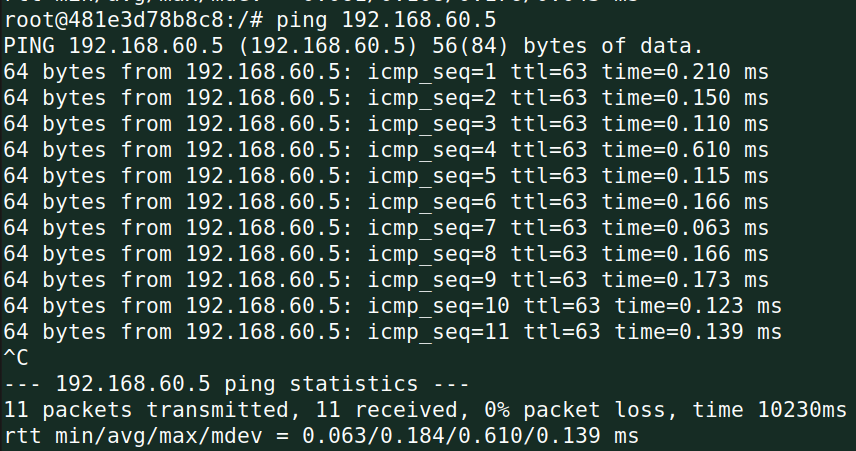
# iptables -A FORWARD -o eth1 -p tcp --sport 23 -j ACCEPT

**Task4**

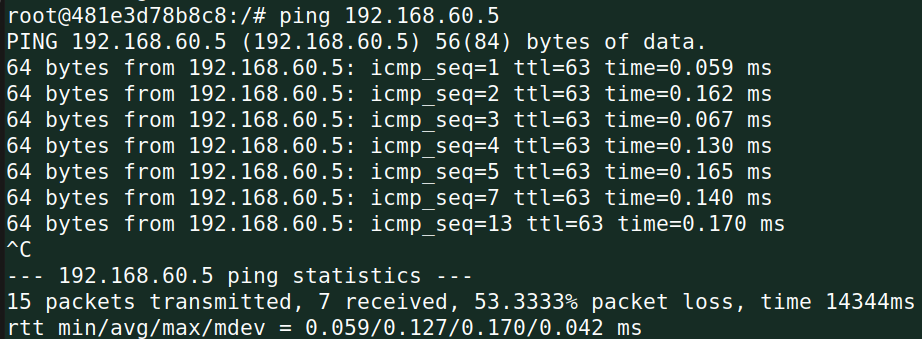
在router上设置了#iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT后，查看防火墙配置，FORWARD报文策略为ACCEPT



此时在10.9.0.5上ping 192.168.60.5，完全没有被限制，不但短时间内超过了5个报文，也超过了每分钟10个的标准



加上了#iptables -A FORWARD -s 10.9.0.5 -j DROP规则后，再次测试如下，前5个报文的回显速度正常，到了第6个报文，因为burst = 5的限制被DROP掉，第7个报文后面因为10/min的限制，一分钟最多10个报文，很多都被DROP掉，所以这个规则必须加上，这个增加的规则表示DROP掉所有来自10.9.0.5的报文，否则默认策略里面的ACCEPT就会接受报文，不会丢弃多余报文，经过测试，**不能使用#iptables -P FORWARD DROP来代替#iptables -A FORWARD -s 10.9.0.5 -j DROP**，不知道为什么limit会在默认策略之后，可能是他要统计数据的原因吧，所以#iptables -P FORWARD DROP会导致报文一个都收不到



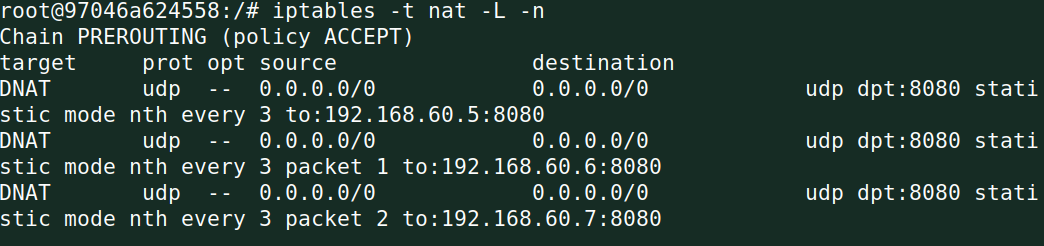
**Task5**

在router上配置如下nat filter，如下指令将在路由之前将报文提取出来，做ip地址的映射，用 -nth模式来实现round robin分配报文

#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 3 --packet 1 -j DNAT --to-destination 192.168.60.6:8080

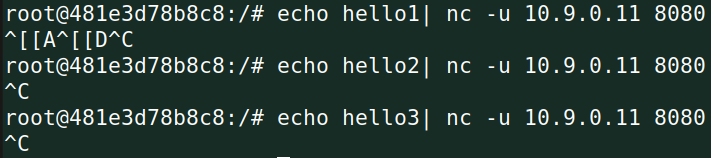
#iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 2 -j DNAT --to-destination 192.168.60.7:8080



然后在192.168.60.5/6/7上分别监听udp 8080端口#nc -luk 8080

最后在10.9.0.5上发送报文

10.9.0.5：



192.168.60.5



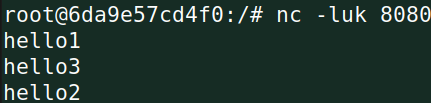
192.168.60.6

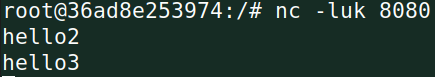


192.168.60.7



看到并没有出现hello2在192.168.60.6上，hello3在192.168.60.7上的情况，如果反复echo hello1/2/3的话，也不一定能保证hello1/2/3一定在5/6/7主机上，有的时候会失效







使用—random模式的时候的概率相等，相加等于1可以实现大体平均分配报文，和-nth模式相比只有这个区别，nat映射都一样

#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 0.33 -j DNAT --to-destination 192.168.60.6:8080

# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.33 -j DNAT --to-destination 192.168.60.7:8080

