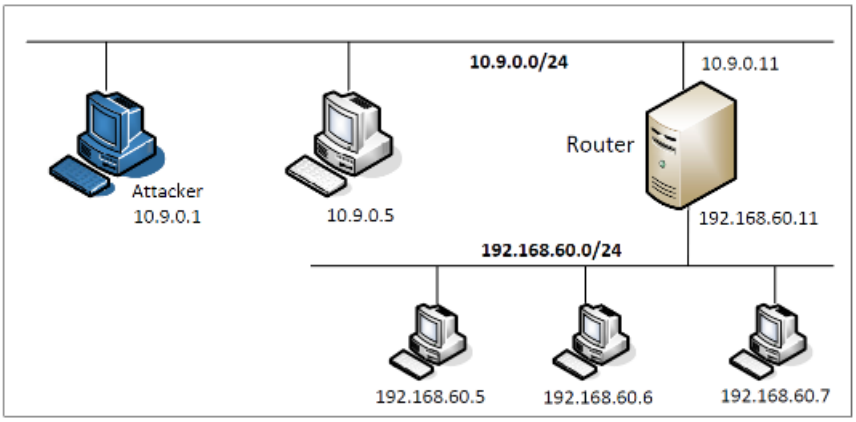
# Chapter 6:Firewall Exploration Lab

57118132 吴嘉琪
852809049@qq.com
Southeast University — 2021 年 7 月 25 日

## 网络拓扑




### Task 1: Implementing a Simple Firewall

用户主机的IP地址为10.9.0.5，攻击者主机的IP地址为10.9.0.1，内网主机的IP地址为192.168.60.5。

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

利用make命令编译可装载内核模块如下。

```
seed@VM:~/kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/seed/kernel_module/hello.o
see include/linux/module.h for more information
  CC [M]  /home/seed/kernel_module/hello.mod.o
  LD [M]  /home/seed/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

利用insmod和rmmod命令将内核模块进行操作如下，可知成功插入和移除。

```
[07/25/21]seed@VM:~/kernel_module$ sudo insmod hello.ko
```

```
[07/25/21]seed@VM:~/kernel_module$ lsmod |grep hello
hello                  16384  0
[07/25/21]seed@VM:~/kernel_module$ sudo rmmod hello
```

利用dmesg命令查看/var/log/syslog文件中的信息，得到结果如下。

```
[07/25/21]seed@VM:~/kernel_module$ dmesg | grep World
[ 2999.044572] Hello World!
[ 3029.777412] Bye-bye World!.
```

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

### 1.Compile the sample code using the provided Makefile

在主机上利用dig查询www.example.com的DNS如下，可知能够获得相关信息。

```
[07/25/21]seed@VM:~/kernel_module$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

利用make命令编译可装载内核模块如下

```
[07/25/21]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/packet_filter m
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

利用insmod命令插入内核模块，在主机上利用dig查询www.example.com的DNS如下，可知无法获得相关信息。

```
[07/25/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/25/21]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

### 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>
```

```c
#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 printInfo(void *priv, struct sk_buff *skb, const struct nf_hook_sta
{
        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);
        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;
        }
        printk(KERN_INFO " %pI4 --> %pI4 (%s)\n", &(iph->saddr), &(iph->daddr),
        return NF_ACCEPT;
}
int registerFilter(void) {
        printk(KERN_INFO "Registering filters.\n");
        hook1.hook = printInfo;
        hook1.hooknum = NF_INET_PRE_ROUTING;
        hook1.pf = PF_INET;
        hook1.priority = NF_IP_PRI_FIRST;
        nf_register_net_hook(&init_net, &hook1);
        hook2.hook = printInfo;
        hook2.hooknum = NF_INET_LOCAL_IN;
```

```
        hook2.pf = PF_INET;
        hook2.priority = NF_IP_PRI_FIRST;
        nf_register_net_hook(&init_net, &hook2);
        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);
        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);
        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);
}
module_init(registerFilter);
module_exit(removeFilter);
MODULE_LICENSE("GPL");
```

利用make命令编译可装载内核模块，并且利用insmod命令插入内核模块如下。

```
[07/25/21]seed@VM:~/.../packet_filter$ sudo rmmod seedFilter
[07/25/21]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/packet_filter m
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[07/25/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/25/21]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
```

在用户主机上ping内网主机，得到结果如下，可知能够连接。

```
[07/25/21]seed@VM:~/Desktop$ docksh 6f
```

```
root@6f2a6da2277b:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

利用dmesg命令查看/var/log/syslog文件中的信息，得到结果如下。

```
[ 6421.412436] *** PRE_ROUTING
[ 6421.412437]  10.9.0.5 --> 192.168.60.5 (ICMP)
[ 6421.412443] *** FORWARD
[ 6421.412443]  10.9.0.5 --> 192.168.60.5 (ICMP)
[ 6421.412446] *** POST_ROUTING
[ 6421.412446]  10.9.0.5 --> 192.168.60.5 (ICMP)
```

在用户主机上ping攻击者主机，得到结果如下，可知能够连接。

```
# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
64 bytes from 10.9.0.1: icmp_seq=1 ttl=64 time=0.113 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.046 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.053 ms
```

利用dmesg命令查看/var/log/syslog文件中的信息，得到结果如下。

```
[ 8803.410311] *** PRE_ROUTING
[ 8803.410311]  10.9.0.5 --> 10.9.0.1 (ICMP)
[ 8803.410315] *** LOCAL_IN
[ 8803.410316]  10.9.0.5 --> 10.9.0.1 (ICMP)
[ 8803.410320] *** LOCAL_OUT
[ 8803.410320]  10.9.0.1 --> 10.9.0.5 (ICMP)
[ 8803.410321] *** POST_ROUTING
[ 8803.410321]  10.9.0.1 --> 10.9.0.5 (ICMP)
```

根据实验结果可知，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的时候调用；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的时候调用。 **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>
#include <linux/udp.h>
#include <linux/if_ether.h>
#include <linux/inet.h>
static struct nf_hook_ops hook1, hook2;
```

```
unsigned int ICMPFilter(void *priv, struct sk_buff *skb, const struct
nf_hook_state *state)
{
        struct iphdr *iph;
        iph = ip_hdr(skb);
        if (iph->protocol == IPPROTO_ICMP) {
                printk(KERN_INFO "Dropping ICMP packet:%pI4\n", &(iph->saddr));
                return NF_DROP;
        }
        return NF_ACCEPT;
}
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)) {
                printk(KERN_INFO "Dropping telnet packet:%pI4\n", &(iph->saddr))
                return NF_DROP;
        }
        return NF_ACCEPT;
}
int registerFilter(void) {
        printk(KERN_INFO "Registering filters.\n");
        hook1.hook = ICMPFilter;
        hook1.hooknum = NF_INET_LOCAL_IN;
        hook1.pf = PF_INET;
        hook1.priority = NF_IP_PRI_FIRST;
        nf_register_net_hook(&init_net, &hook1);
        hook2.hook = telnetFilter;
        hook2.hooknum = NF_INET_LOCAL_IN;
        hook2.pf = PF_INET;
        hook2.priority = NF_IP_PRI_FIRST;
        nf_register_net_hook(&init_net, &hook2);
        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);
```

```
}
module_init(registerFilter);
module_exit(removeFilter);
MODULE_LICENSE("GPL");
```

利用make命令编译可装载内核模块，并且利用insmod命令插入内核模块如下。

```
[07/25/21]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/packet_filter m
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[07/25/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/25/21]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
```

在用户主机上ping攻击者主机，得到结果如下，可知无法连接。

```
root@6f2a6da2277b:~# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
^C
--- 10.9.0.1 ping statistics ---
24 packets transmitted, 0 received, 100% packet loss, time 23547ms
```

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。

```
root@6f2a6da2277b:~# telnet 10.9.0.1
Trying 10.9.0.1...
telnet: Unable to connect to remote host: Connection timed out
```

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。

```
root@6f2a6da2277b:~# dmesg | grep Dropping
[ 9436.951853] Dropping ICMP packet:10.9.0.5
[ 9437.969995] Dropping ICMP packet:10.9.0.5
[ 9438.994882] Dropping ICMP packet:10.9.0.5
[ 9440.017479] Dropping ICMP packet:10.9.0.5
[ 9441.044620] Dropping ICMP packet:10.9.0.5
[ 9442.070628] Dropping ICMP packet:10.9.0.5
[ 9443.090399] Dropping ICMP packet:10.9.0.5
[ 9444.116344] Dropping ICMP packet:10.9.0.5
[ 9445.140409] Dropping ICMP packet:10.9.0.5
```

```
[ 9446.161628] Dropping ICMP packet:10.9.0.5
[ 9447.186582] Dropping ICMP packet:10.9.0.5
[ 9448.210779] Dropping ICMP packet:10.9.0.5
[ 9449.234708] Dropping ICMP packet:10.9.0.5
[ 9450.257601] Dropping ICMP packet:10.9.0.5
[ 9451.282588] Dropping ICMP packet:10.9.0.5
[ 9452.306899] Dropping ICMP packet:10.9.0.5
[ 9453.330034] Dropping ICMP packet:10.9.0.5
[ 9454.356552] Dropping ICMP packet:10.9.0.5
[ 9455.386289] Dropping ICMP packet:10.9.0.5
[ 9456.414812] Dropping ICMP packet:10.9.0.5
[ 9457.439054] Dropping ICMP packet:10.9.0.5
[ 9458.449733] Dropping ICMP packet:10.9.0.5
[ 9459.485914] Dropping ICMP packet:10.9.0.5
[ 9460.499052] Dropping ICMP packet:10.9.0.5
[ 9573.049608] Dropping telnet packet:10.9.0.5
[ 9574.067686] Dropping telnet packet:10.9.0.5
[ 9576.081784] Dropping telnet packet:10.9.0.5
[ 9580.306944] Dropping telnet packet:10.9.0.5
[ 9588.509638] Dropping telnet packet:10.9.0.5
[ 9604.628969] Dropping telnet packet:10.9.0.5
[ 9611.987225] Dropping telnet packet:10.9.0.5
[ 9613.010705] Dropping telnet packet:10.9.0.5
[ 9615.025146] Dropping telnet packet:10.9.0.5
[ 9619.218598] Dropping telnet packet:10.9.0.5
[ 9627.411575] Dropping telnet packet:10.9.0.5
[ 9643.539862] Dropping telnet packet:10.9.0.5
[ 9675.799853] Dropping telnet packet:10.9.0.5
```

### 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。

```
root@38daf6a0160e:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCE
root@38daf6a0160e:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEP
root@38daf6a0160e:/# iptables -P OUTPUT DROP
root@38daf6a0160e:/# iptables -P INPUT DROP
```

在用户主机上ping路由器，得到结果如下，可知能够连接。

```
# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.045 ms
```

```
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.048 ms
```

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。

```
# telnet 10.9.0.11
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
```

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响应报文输出，ping的报文可以进行传输，而telnet的报文无法进行传输。

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

在路由器上利用iptables命令，创建过滤规则如下。

```
root@38daf6a0160e:/# iptables -A FORWARD -p icmp --icmp-type echo-request -i eth
root@38daf6a0160e:/# iptables -A FORWARD -p icmp --icmp-type echo-reply -i eth0
root@38daf6a0160e:/# iptables -P FORWARD DROP
```

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。

```
root@6f2a6da2277b:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

在用户主机上ping路由器，得到结果如下，可知能够连接。

```
# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp seq=1 ttl-64 time-0.167 ms
64 bytes from 10.9.0.11: icmp seq-2 ttl-64 time=0.087 ms
64 bytes from 10.9.0.11: icmp seq=3 ttl-64 time=0.073 ms
```

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。

```
#ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data
64 bytes from 10.9.0.5: icmp_seq=1 ttl-63 time=0.168 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl-63 time-0.108 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.062 ms
```

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，可知连接失败。

```
# telnet 192.168.60.5
Trying 192.168.60.5..
telnet: Unable to connect to remote host: Connection timed out
```

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。

```
# telnet 10.9.0.5
Trying 10.9.0.5..
telnet: Unable to connect to remote host: Connection timed out
```

### Task 2.C:Protecting Internal Servers

在路由器上利用iptables命令，创建过滤规则如下。

```
# iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport 23-i ACCEPT
# iptables -A FORWARD -o eth0 -p tcp -s 192.168.60.5 --sport 23 -i ACCEPT
# iptables -A FORWARD -i eth1 -p tcp -d 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
```

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。

```
# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
6f2a6da2277b loqin: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86 64)
```

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。

```
# telnet 192.168.60.6
Trying 192.168.60.6....
telnet: Unable to connect to remote host: Connection timed out
```

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。

```
# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e47d80f3aa5d loqin: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86 64)
```

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。

```
# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
```

## 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。

```
# ping 192.168,60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq-1 ttl-63 time-0.250 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.073 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl-63 time=0.115 ms
```

在路由器上利用conntrack -L命令实现连接跟踪，得到结果如下，可知ICMP连接时间约为30s。

```
# conntrack –L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 1d=57 src=192.168.60.5
dst=10.9.0.5 type=e code=0 id=57 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

**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9090端口，并发送消息如下。

```
# nc –u 192.168.60.5 9090
whoami
```

在内网主机192.168.60.5上监听9090端口的UDP连接，得到结果如下。

```
#nc –lu 9090
whoami
```

在路由器上利用conntrack -L命令实现连接跟踪，得到结果如下，可知UDP连接时间约为30s。

```
# conntrack –L
udp      17 28 src=10.9.0.5 dst=192.168.60.5 sport=49265 dport-9090 [UNREPLIED]
src=192.168.60.5  dst=10.9.0.5 sport=9090 dport=49265 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

**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，并发送消息如下。

```
# nc  192.168.60.5 9090
whoami
```

在内网主机192.168.60.5上监听9090端口的TCP连接，得到结果如下。

```
#nc –l 9090
whoami
```

在路由器上利用conntrack -L命令实现连接跟踪，得到结果如下，可知TCP连接时间约为432000s。

```
# conntrack –L
tcp      6 431997 src=10.9.0.5 dst=192.168.60.5 sport=41558 dport=9090
src=192.168.60.5  dst=10.9.0.5 sport=9090 dport=41558 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

## 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。

```
# iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCE
# iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn -m conntr
# iptables -A FORWARD -p tcp -i eth1 -s 192,168.60.0/24 --dport 23 --syn -m con
# iptables -P FORWARD DROP
```

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。

```
# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e47d80f3aa5d loqin: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86 64)
```

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。

```
# telnet 192.168.60.6
Trying 192.168.60.6...
telnet: Unable to connect to remote host: Connection timed out
```

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。

```
# telnet 192.168.60.6
Trying 192.168.60.6 ...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
a8beed46aa46 loqin: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86 64)
```

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。

```
# telnet 10.9.0.5
Trying 10.9.0.5 ...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
6f2a6da2277b loqin: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86 64)
```

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。

```
# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j
# iptables -A FORWARD-s 10.9.0.5 -i DROP
```

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。

```
# ping 192.168.60.5
PING 192.168.60.5 (v) 56(84) bytes of data
64 bytes from 192.168.60.5: icmp_seq=1 ttl-63 time=0.128 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl-63 time-0.108 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.072 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl-63 time=0.106 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl-63 time-0.112 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.076 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl-63 time=0.098 ms
```

在路由器上利用iptables命令，修改流量限制规则如下。

```
# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j
```

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。

```
# ping 192.168.60.5
PING 192.168.60.5 (v) 56(84) bytes of data
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.263 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.053 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.055 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.058 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.060 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.057 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.056 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.058 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.057 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.060 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.057 ms
```

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，可知上述第二条规则是必需的。

### 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)**

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。

```
# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --e
# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --e
# iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT --to-destination 19
```

在用户主机上向路由器的8080端口发送UDP数据包如下。

```
# echo hello | nc -u 10.9.0.11 8080
^C
# echo hello | nc -u 10.9.0.11 8080
^C
# echo hello | nc -u 10.9.0.11 8080
^C
# echo hello | nc -u 10.9.0.11 8080
^C
```

在服务器192.168.60.5上监听8080端口，得到结果如下。

```
# nc -luk 8080
hello
hello
```

在服务器192.168.60.6上监听8080端口，得到结果如下。

```
# nc -luk 8080
hello
```

在服务器192.168.60.7上监听8080端口，得到结果如下。

```
# nc -luk 8080
hello
```

**Using the random mode**

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

```
# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random -
# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random -
# iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT --to-destination 19
```

在用户主机上向路由器的8080端口发送UDP数据包如下。

```
# echo hello | nc -u 10.9.0.11 8080
^C
# echo hello | nc -u 10.9.0.11 8080
^C
# echo hello | nc -u 10.9.0.11 8080
^C
# echo hello | nc -u 10.9.0.11 8080
```

```
^C
# echo hello | nc -u 10.9.0.11 8080
^C
# echo hello | nc -u 10.9.0.11 8080
^C
# echo hello | nc -u 10.9.0.11 8080
^C
# echo hello | nc -u 10.9.0.11 8080
^C
# echo hello | nc -u 10.9.0.11 8080
^C
```

在服务器192.168.60.5上监听8080端口，得到结果如下。

```
# nc -luk 8080
hello
hello
hello
```

在服务器192.168.60.6上监听8080端口，得到结果如下。

```
# nc -luk 8080
hello
hello
```

在服务器192.168.60.7上监听8080端口，得到结果如下。

```
# nc -luk 8080
hello
hello
hello
hello
```