lab[6]-report

57118123 刘康辉

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

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

```
[07/22/21]seed@VM:~/.../Firewall$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Firewall modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Firewall/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/seed/Desktop/Firewall/hello.o
see include/linux/module.h for more information
  CC [M]  /home/seed/Desktop/Firewall/hello.mod.o
  LD [M]  /home/seed/Desktop/Firewall/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。

```
[07/22/21]seed@VM:~/.../Firewall$ sudo insmod hello.ko
[07/22/21]seed@VM:~/.../Firewall$ lsmod | grep hello
hello                  16384  0
[07/22/21]seed@VM:~/.../Firewall$ 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。

```
[07/22/21]seed@VM:~/.../Firewall$ dmesg | grep World
[ 2494.132049] Hello World!
[ 2516.518487] 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。

```
[07/22/21]seed@VM:~/.../Firewall$ 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
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47252
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        18286   IN      A       93.184.216.34

;; Query time: 48 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu Jul 22 06:02:09 EDT 2021
;; MSG SIZE  rcvd: 60
```

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

```
[07/22/21]seed@VM:~/.../Firewall$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Firewall modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Firewall/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Firewall/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/Firewall/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。

```
[07/22/21]seed@VM:~/.../Firewall$ sudo insmod seedFilter.ko
[07/22/21]seed@VM:~/.../Firewall$ 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>
#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_state *state)
{
   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),
protocol);
   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。

```
[07/23/21]seed@VM:~/.../Firewall$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Firewall modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Firewall/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Firewall/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/Firewall/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[07/23/21]seed@VM:~/.../Firewall$ sudo insmod seedFilter.ko
[07/23/21]seed@VM:~/.../Firewall$ 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。

```
root@36cddb526b95:/# 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=0.222 ms
```

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

```
[  469.983357] *** PRE_ROUTING
[  469.983361]     10.9.0.5  --> 192.168.60.5 (ICMP)
[  469.983372] *** FORWARD
[  469.983373]     10.9.0.5  --> 192.168.60.5 (ICMP)
[  469.983377] *** POST_ROUTING
[  469.983378]     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。

```
root@36cddb526b95:/# 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=0.132 ms
```

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

```
[  704.883109] *** PRE_ROUTING
[  704.883110]     10.9.0.5  --> 10.9.0.1 (ICMP)
[  704.883113] *** LOCAL_IN
[  704.883114]     10.9.0.5  --> 10.9.0.1 (ICMP)
[  704.883123] *** LOCAL_OUT
[  704.883124]     10.9.0.1  --> 10.9.0.5 (ICMP)
[  704.883126] *** POST_ROUTING
[  704.883127]     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。

```c
#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。

```
[07/23/21]seed@VM:~/.../Firewall$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Firewall modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Firewall/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Firewall/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/Firewall/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[07/23/21]seed@VM:~/.../Firewall$ sudo insmod seedFilter.ko
[07/23/21]seed@VM:~/.../Firewall$ 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@53fd31c6c3d3:/# 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 ---
14 packets transmitted, 0 received, 100% packet loss, time 13318ms
```

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@53fd31c6c3d3:/# 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。

```
[07/23/21]seed@VM:~/.../Firewall$ dmesg | grep Dropping
[ 7508.610373] Dropping ICMP packet:10.9.0.5
[ 7509.633199] Dropping ICMP packet:10.9.0.5
[ 7510.654358] Dropping ICMP packet:10.9.0.5
[ 7511.679002] Dropping ICMP packet:10.9.0.5
[ 7512.705140] Dropping ICMP packet:10.9.0.5
[ 7513.727040] Dropping ICMP packet:10.9.0.5
[ 7514.752881] Dropping ICMP packet:10.9.0.5
[ 7515.775368] Dropping ICMP packet:10.9.0.5
[ 7516.800981] Dropping ICMP packet:10.9.0.5
[ 7517.838316] Dropping ICMP packet:10.9.0.5
[ 7518.858580] Dropping ICMP packet:10.9.0.5
[ 7519.878769] Dropping ICMP packet:10.9.0.5
[ 7520.898500] Dropping ICMP packet:10.9.0.5
[ 7521.928230] Dropping ICMP packet:10.9.0.5
[ 7529.024203] Dropping telnet packet:10.9.0.5
[ 7530.086482] Dropping telnet packet:10.9.0.5
[ 7532.320529] Dropping telnet packet:10.9.0.5
[ 7536.415275] Dropping telnet packet:10.9.0.5
[ 7544.607365] Dropping telnet packet:10.9.0.5
[ 7560.734450] Dropping telnet packet:10.9.0.5
[ 7592.992919] 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@53fc248d8be9:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
root@53fc248d8be9:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@53fc248d8be9:/# iptables -P OUTPUT DROP
root@53fc248d8be9:/# 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。

```
root@53fde8986d54:/# 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=0.105 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.060 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.109 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。

```
root@53fde8986d54:/# 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@53fc248d8be9:/# iptables -A FORWARD -p icmp --icmp-type echo-request -i eth1 -j ACCEPT
root@53fc248d8be9:/# iptables -A FORWARD -p icmp --icmp-type echo-reply -i eth0 -j ACCEPT
root@53fc248d8be9:/# 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@53fde8986d54:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3061ms
```

在用户主机上ping路由器，得到结果如下，可知能够连接。

```
root@53fde8986d54:/# 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。

```
root@df9d5cbeb2c8:/# 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。

```
root@53fde8986d54:/# 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。

```
root@df9d5cbeb2c8:/# 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。

```
root@53fc248d8be9:/# iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport 23 -j ACCEPT
root@53fc248d8be9:/# iptables -A FORWARD -o eth0 -p tcp -s 192.168.60.5 --sport 23 -j ACCEPT
root@53fc248d8be9:/# iptables -A FORWARD -i eth1 -p tcp -d 192.168.60.0/24 --dport 23 -j ACCEPT
root@53fc248d8be9:/# iptables -A FORWARD -o eth1 -p tcp -s 192.168.60.0/24 --sport 23 -j ACCEPT
root@53fc248d8be9:/# 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。

```
root@53fde8986d54:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
df9d5cbeb2c8 login: 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。

```
root@53fde8986d54:/# 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。

```
root@df9d5cbeb2c8:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
c15da65f559c login: 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。

```
root@df9d5cbeb2c8:/# 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。

```
root@53fde8986d54:/# 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。

```
root@53fc248d8be9:/# conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=57 src=192.168.60.5
 dst=10.9.0.5 type=0 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。

```
root@53fde8986d54:/# 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。

```
root@df9d5cbeb2c8:/# 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。

```
root@53fc248d8be9:/# 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。

```
root@53fde8986d54:/# 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。

```
root@df9d5cbeb2c8:/# 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。

```
root@53fc248d8be9:/# conntrack -L
tcp      6 431997 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=41558 dport=90
90 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。

```
root@53fc248d8be9:/# iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
root@53fc248d8be9:/# iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
root@53fc248d8be9:/# iptables -A FORWARD -p tcp -i eth1 -s 192.168.60.0/24 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
root@53fc248d8be9:/# 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。

```
root@53fde8986d54:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
df9d5cbeb2c8 login: 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。

```
root@53fde8986d54:/# 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。

```
root@df9d5cbeb2c8:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
c15da65f559c login: 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。

```
root@df9d5cbeb2c8:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
53fde8986d54 login: 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。

```
root@53fc248d8be9:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute
--limit-burst 5 -j ACCEPT
root@53fc248d8be9:/# iptables -A FORWARD -s 10.9.0.5 -j 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。

```
root@53fde8986d54:/# 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=0.099 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.082 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.112 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.111 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.075 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=0.093 ms
^C
--- 192.168.60.5 ping statistics ---
14 packets transmitted, 7 received, 50% packet loss, time 13313ms
rtt min/avg/max/mdev = 0.075/0.092/0.112/0.014 ms
```

在路由器上利用iptables命令，修改流量限制规则如下。

```
root@53fc248d8be9:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute
--limit-burst 5 -j ACCEPT
```

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。

```
root@53fde8986d54:/# 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=0.116 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.086 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.119 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.077 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.076 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.099 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.106 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.106 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.079 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.108 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.096 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.108 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.110 ms
^C
--- 192.168.60.5 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12293ms
rtt min/avg/max/mdev = 0.076/0.098/0.119/0.014 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。

```
root@eaf163c47638:/# 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
root@eaf163c47638:/# 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
root@eaf163c47638:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT --to-
destination 192.168.60.7:8080
```

在用户主机上向路由器的8080端口发送UDP数据包如下。

```
root@36cddb526b95:/# echo hello | nc -u 10.9.0.11 8080
^C
root@36cddb526b95:/# echo hello | nc -u 10.9.0.11 8080
^C
root@36cddb526b95:/# echo hello | nc -u 10.9.0.11 8080
^C
root@36cddb526b95:/# 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。

```
root@44ca35555561:/# nc -luk 8080
hello
hello
```

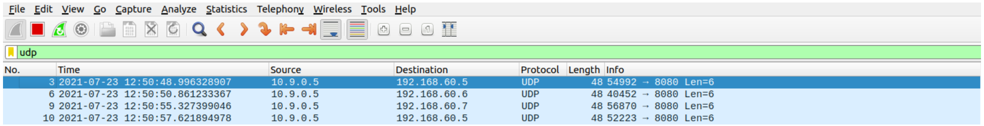
在服务器192.168.60.6上监听8080端口，得到结果如下。

```
root@5479b7208878:/# nc -luk 8080
hello
```

在服务器192.168.60.7上监听8080端口，得到结果如下。

```
root@df14eb7bfad1:/# nc -luk 8080
hello
```

利用wireshark抓包，得到结果如下，可知UDP数据包发送成功，且根据负载均衡规则依次发给三个服务器。



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。

```
root@eaf163c47638:/# 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
root@eaf163c47638:/# 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
root@eaf163c47638:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT --to-
destination 192.168.60.7:8080
```

在用户主机上向路由器的8080端口发送UDP数据包如下。

```
root@36cddb526b95:/# echo hello | nc -u 10.9.0.11 8080
^C
root@36cddb526b95:/# echo hello | nc -u 10.9.0.11 8080
^C
root@36cddb526b95:/# echo hello | nc -u 10.9.0.11 8080
^C
root@36cddb526b95:/# echo hello | nc -u 10.9.0.11 8080
^C
root@36cddb526b95:/# echo hello | nc -u 10.9.0.11 8080
^C
root@36cddb526b95:/# echo hello | nc -u 10.9.0.11 8080
^C
root@36cddb526b95:/# echo hello | nc -u 10.9.0.11 8080
^C
root@36cddb526b95:/# echo hello | nc -u 10.9.0.11 8080
^C
root@36cddb526b95:/# 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。

```
root@44ca35555561:/# 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。

```
root@5479b7208878:/# 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。

```
root@df14eb7bfad1:/# nc -luk 8080
hello
hello
hello
hello
```

利用wireshark抓包，得到结果如下，可知UDP数据包发送成功，且根据负载均衡规则随机发给三个服务器。