山东大学

网络空间安全学院

Firewall Exploration 实验

杜威、李旷达、杨昊

2023 年 12 月 26 日

目录

1 小组成员及分工

姓名	学号	分工
李旷达	202100460124	Task1
杨昊	202100460134	Task2、Task3
杜威	202100460095	Task4、Task5

表 1: 小组成员及分工

2 主要任务

- 防火墙是如何工作的
- 为网络建立一个简单的防火墙
- 使用网络过滤器
- 使用 iptables 建立防火墙规则
- iptables 的应用

3 实验步骤与结果

3.1 Task 1 Implementing a Simple Firewall

3.1.1 Task 1.A:Implement a Simple Kernel Module

- 从实验室官网下载此使用所用到的 Labsetbup 文件，并启动容器
- 我们需要实现一个简单的包过滤类的防火墙，它检查每个传入和传出的包，并强制执行管理员设置的防火墙策略。
- 因为数据包处理是在内核中完成的，所以过滤也必须在内核中完成。因此实现该过滤系统需要我们修改 linux 内核，在过去，这必须通过修改和重建内核来完成。现代 Linux 操作系统提供了几种新的机制，可以方便地操作数据包，而无需重新构建内核映像。目前主要的两种是可载入核心模组 (LKM) 和 Netfilter

再 Task1.A 中我们将使用 LKM，来加载一个简单的核心模组，任务中所需要的代码已经在 Labsetup 文件夹中的 Files/kernel_module 文件夹中提供

编译此文件并查看

```
[12/12/23]seed@VM:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/La
bsetup/Files/kernel_module modules
make[1]: 进入目录“/usr/src/linux-headers-5.4.0-54-generic”
  CC [M]  /home/seed/Desktop/Labsetup/Files/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Labsetup/Files/kernel_module/hello.mod
.o
  LD [M]  /home/seed/Desktop/Labsetup/Files/kernel_module/hello.ko
make[1]: 离开目录“/usr/src/linux-headers-5.4.0-54-generic”
[12/12/23]seed@VM:~/.../kernel_module$ ls
hello.c   hello.mod    hello.mod.o  Makefile       Module.symvers
hello.ko  hello.mod.c  hello.o      modules.order
```

根据指导书将 hello.ko 插入内核，并监看

```
[ 4720.009035] audit: type=1400 audit(1702373519.011:1709):
eration="profile_replace" profile="unconfined" name="snap.s
ware" pid=49115 comm="apparmor_parser"
[ 4720.163243] audit: type=1400 audit(1702373519.167:1710):
eration="profile_replace" profile="unconfined" name="snap.s
ware-local-file" pid=49116 comm="apparmor_parser"
[ 5292.991522] hello: loading out-of-tree module taints ker
[ 5292.991563] hello: module verification failed: signature
missing - tainting kernel
[ 5292.992405] Hello World!
libahci                32768  1 ahci
mptbase                94208  2 mptspi,mptscsih
scsi_transport_spi     32768  1 mptspi
i2c_piix4              28672  0
[12/12/23]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[12/12/23]seed@VM:~/.../kernel_module$
```

成功插入内核并且输出 Hello World!

然后从内核中移除此模块

```
 isa0060/serio0).
[ 5374.936732] atkbd serio0: Use 'setkeycodes 6d <keycode>' to
[ 5374.936926] atkbd serio0: Unknown key released (translated
n isa0060/serio0).
[ 5374.936926] atkbd serio0: Use 'setkeycodes 6d <keycode>' to
[ 5376.820832] atkbd serio0: Unknown key pressed (translated s
 isa0060/serio0).
[ 5376.820834] atkbd serio0: Use 'setkeycodes 6d <keycode>' to
[ 5376.820929] atkbd serio0: Unknown key released (translated
n isa0060/serio0).
[ 5376.820929] atkbd serio0: Use 'setkeycodes 6d <keycode>' to
[ 5469.483111] Bye-bye World!.
```

```
hello.mod.o
hello.o
[12/12/23]seed@VM:~/.../kernel_module$ sudo rmmod hello
[12/12/23]seed@VM:~/.../kernel_module$ lsmod | grep hello
[12/12/23]seed@VM:~/.../kernel_module$
```

3.1.2 Task1.B:Implement a Simple Firewall Using Netfilter

我们将使用 LKM 和 Netfilter 来实现一个数据包过滤模块。

- **我们将对 8.8.8.8 的 UDP 数据包进行过滤**

  实验文件中已经给出示例代码，我们将对目标 IP 是 8.8.8.8，且目标端口是 53 的 UDP 数据包进行拦截，即阻止向 8.8.8.8 命名服务器的 DNS 查询

- 首先测试网络

```
[12/12/23]seed@VM:~/.../packet_filter$ ls
Makefile  seedFilter.c
[12/12/23]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
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15212
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        1556    IN      A       93.184.216.34

;; AUTHORITY SECTION:
example.com.            45197   IN      NS      a.iana-servers.net.
example.com.            45197   IN      NS      b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.     0       IN      AAAA    2001:500:8f::53
b.iana-servers.net.     1504    IN      AAAA    2001:500:8d::53

;; Query time: 144 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: 二 12月 12 05:40:21 EST 2023
;; MSG SIZE  rcvd: 164
```

- 编译程序，并将其插入内核

```
[12/12/23]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/
bsetup/Files/packet_filter modules
make[1]: 进入目录“/usr/src/linux-headers-5.4.0-54-generic”
  CC [M]  /home/seed/Desktop/Labsetup/Files/packet_filter/seedFil
r.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Labsetup/Files/packet_filter/seedFil
r.mod.o
  LD [M]  /home/seed/Desktop/Labsetup/Files/packet_filter/seedFil
r.ko
make[1]: 离开目录“/usr/src/linux-headers-5.4.0-54-generic”
[12/12/23]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[12/12/23]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
```

- 重新测试

```
[12/12/23]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
```

发现连接超时，无法连接到此服务器，过滤成功

再实施两个 hook，以达到

1. 防止其他计算机 ping 到虚拟机
2. 防止其他计算机 telnet 到计算机
3. 实现两个不同的 hook 函数，但将它们注册到同一个 netfilter

– 进入容器 HostA（IP 为 10.9.0.5）测试网络

```
seed@VM: ~/.../packet_filter
root@90a40f3695e3:/# 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=0.111 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.076 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.079 ms
^C
--- 10.9.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2031ms
rtt min/avg/max/mdev = 0.076/0.088/0.111/0.015 ms
root@90a40f3695e3:/# telnet 10.9.0.1
Trying 10.9.0.1...
Connected to 10.9.0.1.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
VM login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

32 updates can be installed immediately.
32 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
*** 需要重启系统 ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

[12/12/23]seed@VM:~$
```

可以看到此时 ping 与 telnet 均可用

– 修改防火墙程序并加入到内核中

* 注：

1. 拦截 ping，就是要拦截 ICMP 数据包
2. 拦截 telnet，是要拦截 TCP 数据包
3. 要先将之前修改的内核清除

修改程序如下：

```
#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/icmp.h>
#include <linux/if_ether.h>
#include <linux/inet.h>

static struct nf_hook_ops hook1, hook2,hook3,hook4;

unsigned int blockUDP(void *priv, struct sk_buff *skb,
     const struct nf_hook_state *state)
{
   struct iphdr *iph;
   struct udphdr *udph;

   u16  port   = 53;
   char ip[16] = "8.8.8.8";
   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_UDP) {
  udph = udp_hdr(skb);
```

```
32          if (iph->daddr == ip_addr && ntohs(udph->dest) == port){
33        printk(KERN_WARNING "*** Dropping %pI4 (UDP), port %d\n",
34        &(iph->daddr), port);
35        return NF_DROP;
36          }
37           }
38           return NF_ACCEPT;
39        }
40        //blocking ping
41        unsigned int blockICMP(void *priv, struct sk_buff *skb,
42             const struct nf_hook_state *state)
43        {
44           struct iphdr *iph;
45           struct icmphdr *icmph;
46
47           char ip[16] = "10.9.0.1";
48           u32  ip_addr;
49
50           if (!skb) return NF_ACCEPT;
51
52           iph = ip_hdr(skb);
53           // Convert the IPv4 address from dotted decimal to 32-
                bit binary
54           in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);
55
56           if (iph->protocol == IPPROTO_ICMP) {
57         icmph = icmp_hdr(skb);
58         if (iph->daddr == ip_addr && icmph->type==ICMP_ECHO){
59        printk(KERN_WARNING "*** Dropping %pI4 (ICMP)\n",
60        &(iph->daddr));
61        return NF_DROP;
62          }
63           }
64           return NF_ACCEPT;
65        }
66        //blocking telnet
67        unsigned int blockTelnet(void *priv, struct sk_buff *skb,
68             const struct nf_hook_state *state)
```

```
69      {
70          struct iphdr *iph;
71          struct tcphdr *tcph;
72
73          u16  port    = 23;
74          char ip[16] = "10.9.0.1";
75          u32  ip_addr;
76
77          if (!skb) return NF_ACCEPT;
78
79          iph = ip_hdr(skb);
80          // Convert the IPv4 address from dotted decimal to 32-
               bit binary
81          in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);
82
83          if (iph->protocol == IPPROTO_TCP) {
84       tcph = tcp_hdr(skb);
85       if (iph->daddr == ip_addr && ntohs(tcph->dest) == port){
86      printk(KERN_WARNING "*** Dropping %pI4 (Telnet), port %d\
           n",
87      &(iph->daddr), port);
88      return NF_DROP;
89        }
90         }
91          return NF_ACCEPT;
92      }
93      unsigned int printInfo(void *priv, struct sk_buff *skb,
94            const struct nf_hook_state *state)
95      {
96          struct iphdr *iph;
97          char *hook;
98          char *protocol;
99
100         switch (state->hook){
101           case NF_INET_LOCAL_IN:     hook = "LOCAL_IN";
                 break;
102           case NF_INET_LOCAL_OUT:    hook = "LOCAL_OUT";
                 break;
```

```
103             case NF_INET_PRE_ROUTING:  hook = "PRE_ROUTING";
                   break;
104             case NF_INET_POST_ROUTING: hook = "POST_ROUTING";
                   break;
105             case NF_INET_FORWARD:hook = "FORWARD";break;
106             default: hook = "IMPOSSIBLE";   break;
107           }
108           printk(KERN_INFO "*** %s\n", hook); // Print out the
                 hook info
109
110           iph = ip_hdr(skb);
111           switch (iph->protocol){
112             case IPPROTO_UDP:  protocol = "UDP";   break;
113             case IPPROTO_TCP:  protocol = "TCP";   break;
114             case IPPROTO_ICMP: protocol = "ICMP";  break;
115             default:     protocol = "OTHER"; break;
116
117           }
118           // Print out the IP addresses and protocol
119           printk(KERN_INFO "    %pI4  --> %pI4 (%s)\n",
120         &(iph->saddr), &(iph->daddr), protocol);
121
122           return NF_ACCEPT;
123       }
124
125       int registerFilter(void) {
126           printk(KERN_INFO "Registering filters.\n");
127
128           hook1.hook = printInfo;
129           hook1.hooknum = NF_INET_LOCAL_OUT;
130           hook1.pf = PF_INET;
131           hook1.priority = NF_IP_PRI_FIRST;
132           nf_register_net_hook(&init_net, &hook1);
133
134           hook2.hook = blockUDP;
135           hook2.hooknum = NF_INET_POST_ROUTING;
136           hook2.pf = PF_INET;
137           hook2.priority = NF_IP_PRI_FIRST;
```

```
138         nf_register_net_hook(&init_net, &hook2);
139
140         hook3.hook = blockICMP;
141         hook3.hooknum = NF_INET_PRE_ROUTING;
142         hook3.pf = PF_INET;
143         hook3.priority = NF_IP_PRI_FIRST;
144         nf_register_net_hook(&init_net, &hook3);
145
146         hook4.hook = blockTelnet;
147         hook4.hooknum = NF_INET_PRE_ROUTING;
148         hook4.pf = PF_INET;
149         hook4.priority = NF_IP_PRI_FIRST;
150         nf_register_net_hook(&init_net, &hook4);
151         return 0;
152     }
153
154     void removeFilter(void) {
155         printk(KERN_INFO "The filters are being removed.\n");
156         nf_unregister_net_hook(&init_net, &hook1);
157         nf_unregister_net_hook(&init_net, &hook2);
158         nf_unregister_net_hook(&init_net, &hook3);
159         nf_unregister_net_hook(&init_net, &hook4);
160     }
161
162     module_init(registerFilter);
163     module_exit(removeFilter);
164
165     MODULE_LICENSE("GPL");
```

– 测试防火墙效果

```
root@90a40f3695e3:/# 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 ---
31 packets transmitted, 0 received, 100% packet loss, time 30727ms

root@90a40f3695e3:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
root@90a40f3695e3:/#
```

ping 与 telnet 均超时

```
[13775.080791]     192.168.247.133  --> 185.125.190.49 (TCP)
[13779.045759] *** LOCAL_OUT
[13779.045783]     192.168.247.133  --> 42.81.193.250 (TCP)
[13779.798868] *** Dropping 10.9.0.1 (Telnet), port 23
[13780.807800] *** Dropping 10.9.0.1 (Telnet), port 23
[13782.824303] *** Dropping 10.9.0.1 (Telnet), port 23
[13784.552276] *** LOCAL_OUT
[13784.552278]     192.168.247.133  --> 113.240.75.252 (TCP)
[13784.591473] *** LOCAL_OUT
[13784.591497]     192.168.247.133  --> 113.240.75.252 (TCP)
[13786.888217] *** Dropping 10.9.0.1 (Telnet), port 23
```

```
root@90a40f3695e3:/# st HostA
bash: st: command not found
root@90a40f3695e3:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of dat
^C
--- 10.9.0.1 ping statistics ---
31 packets transmitted, 0 received, 100% pac

root@90a40f3695e3:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
root@90a40f3695e3:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
root@90a40f3695e3:/#
```

拦截 23 端口的 telnet

```
3812.368884] *** Dropping 10.9.0.1 (ICMP)
3813.384063] *** Dropping 10.9.0.1 (ICMP)
3814.407994] *** Dropping 10.9.0.1 (ICMP)
3815.431955] *** Dropping 10.9.0.1 (ICMP)
3815.442951] *** LOCAL_OUT
3815.442965]     192.168.247.133  --> 42.81.193.250 (TCP)
3816.455590] *** Dropping 10.9.0.1 (ICMP)
```

```
--- 10.9.0.1 ping statistics ---
31 packets transmitted, 0 received, 100% packet loss, t

root@90a40f3695e3:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
root@90a40f3695e3:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
root@90a40f3695e3:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
```

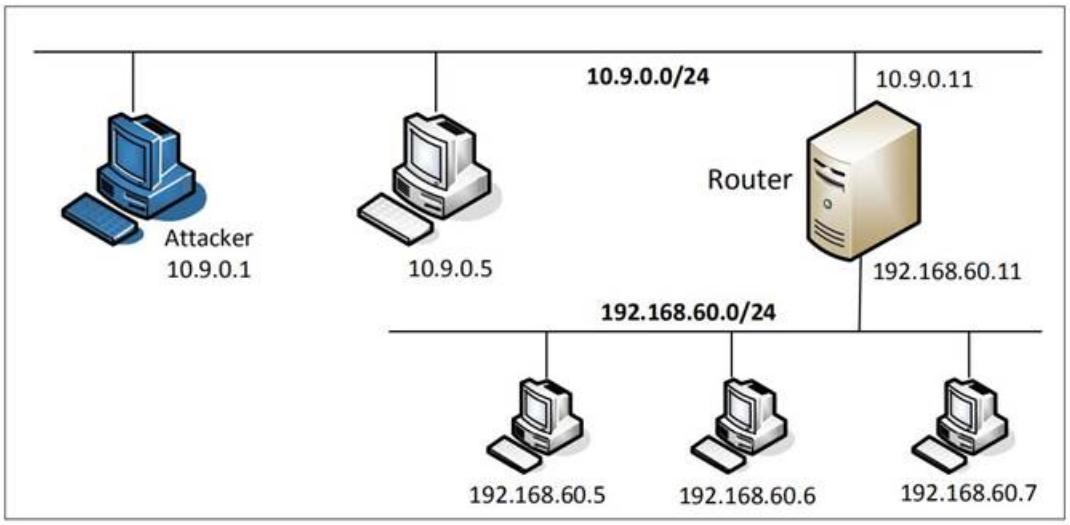
拦截 ping 的 ICMP 数据包

– 任务结束后，从内核中去除

```
[12/12/23]seed@VM:~/.../packet_filter$ sudo rmmod seedFilter
[12/12/23]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
[12/12/23]seed@VM:~/.../packet_filter$ make clean
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/La
bsetup/Files/packet_filter clean
make[1]: 进入目录“/usr/src/linux-headers-5.4.0-54-generic”
  CLEAN   /home/seed/Desktop/Labsetup/Files/packet_filter/Module.sy
mvers
make[1]: 离开目录“/usr/src/linux-headers-5.4.0-54-generic”
[12/12/23]seed@VM:~/.../packet_filter$
```

3.2 Task 2: Experimenting with Stateless Firewall Rules

3.2.1 Task 2.A: Protecting the Router



上面的是实验中容器建立之后的拓扑结构。

```
dd5f32446237  hostA-10.9.0.5
431ad715d95c  host1-192.168.60.5
2234df4a9003  host3-192.168.60.7
e974fc244391  host2-192.168.60.6
528294d68f89  seed-router
```

本任务的要求是，为路由器设置过滤规则，使得来自外部的主机除了可以 PING 以外无法通过其他方式访问路由器。重点在于使用 iptables 设置规则，要学会如何操作 iptables，另外还要记得不用的规则要删除。

PING 命令使用的是 ICMP 数据包，那我们设置过滤规则让 ICMP 可接受可发出然后不接受其他类型的数据包即可。

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

iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT

这两条命令使路由器分别可以接受 ICMP 请求和发出 ICMP 回复。

iptables -P INPUT DROP

iptables -P OUTPUT DROP

这两条命令设置路由器的默认输入输出规则为丢弃，这样就可以无视其他类型的数据包。

```
root@dd5f32446237:/# 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=0.098
ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.055
ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.111
ms
```

现在从 10.9.0.5 主机尝试 PING 路由器，可以 PING 通。

```
rtt min/avg/max/mdev = 0.055/0.088/0.111/0.023 ms
root@dd5f32446237:/# telnet 10.9.0.11
Trying 10.9.0.11...
```

然而尝试 telnet 连接等待许久都没有连接上，telnet 是无法连到路由器了。

3.2.2 Task 2.B: Protecting the Internal Network

本任务的要求如下（这规则有点锥形 NAT 严格那味了）：

1. 外网主机无法 PING 内网主机。
2. 外网主机可以 PING 路由器。
3. 内网主机可以 PING 外网主机。
4. 内外网之间的其他数据包应当丢弃。

iptables -F

iptables -P OUTPUT ACCEPT

iptables -P INPUT ACCEPT

首先输入上面的命令清除先前设置的规则。

iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP

iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply -j ACCEPT

上面两条命令分别是丢弃外网主机的 ICMP 请求不予转发，令外网无法 PING 内网主机，以及允许转发外网主机的 ICMP 回复，这是为了内网主机 PING 外网主机可以收到回复。

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

上面的命令允许转发内网主机的 ICMP 请求，使得内网主机可以 PING 外网。

iptables -P FORWARD DROP

上面的命令设置默认转发规则为丢弃，其他类型的数据包将不予转发。

```
root@dd5f32446237:/# 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=0.176
ms
64 bytes from 10.9.0.11: icmp seq=2 ttl=64 time=0.097
```

规则设置完成后进入 10.9.0.5（外网主机）尝试 PING 路由器，成功。

```
root@dd5f32446237:/# 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 ---
36 packets transmitted, 0 received, 100% packet loss,
time 35894ms
```

尝试从 10.9.0.5（外网主机）PING192.168.60.5（内网主机），失败

```
root@431ad715d95c:/# 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=0.115 m
s
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.153 m
s
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.153 m
```

虽然外网 PING 不通内网主机，但是从 192.168.60.5（内网主机）却可以 PING10.9.0.5（外网主机），如上图所示。

```
root@431ad715d95c:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

虽然内网主机可以 PING 通外网主机，但是其他包会被丢弃，上图所示的内网主机尝试 telnet 外网主机失败了。规则验证完成。

3.2.3 Task 2.C: Protecting Internal Servers

本任务的要求如下：

1. 外网主机可以 telnet 连接到 192.168.60.5，其他的内网主机不可被访问。
2. 其他内网服务器不可被外网主机访问。
3. 内网主机可以访问其他内网主机的 telnet server。
4. 内网主机不能访问外网主机的 telnet server。

iptables -P FORWARD ACCEPT

首先清除前面的规则，这条命令用于清除转发规则。

iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport 23 -j ACCEPT

iptables -A FORWARD -i eth1 -p tcp -s 192.168.60.5 --sport 23 -j ACCEPT

这两条命令分别用于允许转发外网主机的以 192.168.60.5 为目的地址、23 为目的端口的数据包，以及允许转发内网主机以 192.168.60.5 为源地址、23 为源端口的数据包。

iptables -P FORWARD DROP

这条命令设置默认转发规则为丢弃，这样除了 192.168.60.5 的 23 端口的 telnet server，外网主机无法访问内网其他内容，而内网也无法访问外网，但因为路由器管不着内网内部的数据包互相转发，所以内网主机可以轻松访问内网其他主机及其服务器。规则设置完毕。

```
root@dd5f32446237:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
431ad715d95c login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-gene
```

外网主机 10.9.0.5 可以访问 192.168.60.5-23 的 telnet server。

```
root@dd5f32446237:/# telnet 192.168.60.6
Trying 192.168.60.6...
```

外网主机 10.9.0.5 无法访问其他内网主机的 telnet server。

```
root@dd5f32446237:/# 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 ---
2 packets transmitted, 0 received, 100% packet loss, t
ime 1002ms
```

外网主机 10.9.0.5PING 不到内网主机 192.168.60.5，无法访问内网主机。

```
root@431ad715d95c:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e974fc244391 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-gene
```

内网主机 192.168.60.5 可以访问 192.168.60.6 的 telnet server。

```
root@431ad715d95c:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

内网主机 192.168.60.5 不能访问外网主机 10.9.0.5 的 telnet server。规则验证完毕。

3.3 Task 3: Connection Tracking and Stateful Firewall

3.3.1 Task 3.A: Experiment with the Connection Tracking

ICMP：

首先在 10.9.0.5 尝试 PING192.168.60.5，接着在路由器上使用命令 conntrack -L 来追踪连接。

```
root@528294d68f89:/# conntrack -L
icmp     1 5 src=10.9.0.5 dst=192.168.60.5 type=8 code
=0 id=43 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 i
d=43 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries hav
e been shown.
root@528294d68f89:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries hav
e been shown.
```

在 PING 持续过程中和结束没多久时都能在路由器追踪到连接，但是 PING 结束一会后就无法捕获到连接了。在 PING 的过程中两台主机建立了 ICMP 连接，一台发一台回，PING 结束之后，由于一段时间没有收到两台主机之间的数据包，连接就会被认为关闭了。

UDP:

现在使用 UDP 连接，首先在 192.168.60.5 的 9090 端口打开 nc，接着在 10.9.0.5 连接。

```
root@dd5f32446237:/# nc -u 192.168.60.5 9090
SS Marchiert im Feindesland,
Und singt ein Teufelslied.
```

发送消息。

```
root@431ad715d95c:/# nc -lu 9090
SS Marchiert im Feindesland,
Und singt ein Teufelslied.
```

另一边也收到了消息。

```
root@528294d68f89:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries hav
e been shown.
root@528294d68f89:/# conntrack -L
udp      17 23 src=10.9.0.5 dst=192.168.60.5 sport=452
50 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.
5 sport=9090 dport=45250 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries hav
e been shown.
root@528294d68f89:/# conntrack -L
udp      17 18 src=10.9.0.5 dst=192.168.60.5 sport=452
50 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.
5 sport=9090 dport=45250 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries hav
e been shown.
root@528294d68f89:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries hav
e been shown.
```

在消息发送前，没有检测到连接，在消息发出后检测到连接，然而一小会不发消息，路由器就无法检测到连接。这充分说明了 UDP 是面向无连接的协议，如果不发送消息，两台主机之间是不会通过收发数据包来维持状态同步的。

TCP：

在 192.168.60.5 的 9090 端口启动 nc，在 10.9.0.5 连接。

```
root@528294d68f89:/# conntrack -L
tcp      6 431993 ESTABLISHED src=10.9.0.5 dst=192.168
.60.5 sport=49582 dport=9090 src=192.168.60.5 dst=10.9
.0.5 sport=9090 dport=49582 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries hav
e been shown.
```

在路由器这边发现，即便不发送消息，一旦在 10.9.0.5 输入了 nc 连接命令，连接就被持续地检测到了。这说明了 TCP 是面向连接的协议，即便不发送消息，两台主机之间仍然在收发数据包来保持状态同步。

3.3.2 Task 3.B: Setting Up a Stateful Firewall

本任务的要求如下：

1. 外网主机可以 telnet 连接到 192.168.60.5，其他的内网主机不可被访问。
2. 其他内网服务器不可被外网主机访问。

3. 内网主机可以访问其他内网主机的 telnet server。

4. 内网主机可以访问外网主机的 telnet server。

该套规则实际上和 Task 2.C 相似，只有最后一条要求不同（相反）。然而这一次我们要建立的防火墙是基于连接的，指导书还对一些命令做了提示（SYN），因此我们显然不能直接照搬原来的命令。这一次我们要考虑允许通过 SYN 来建立连接，并允许连接的后续数据包的收发。

iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT

这条命令允许外网主机对 192.168.60.5-23 建立 TCP 连接。

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

这条命令允许内网主机和外网主机建立 TCP 连接。

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

这条命令允许路由器转发已建立连接的主机之间的 TCP 数据以继续通讯。

iptables -A FORWARD -p tcp -j DROP

这条命令要求路由器不转发其他 TCP 数据，也就是非连接的。

iptables -P FORWARD ACCEPT

这条命令允许转发其他类型的数据包。

```
root@dd5f32446237:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
431ad715d95c login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-gene
```

外网主机 10.9.0.5 对 192.168.60.5 的 telnet server 的访问成功。

```
root@dd5f32446237:/# telnet 192.168.60.6
Trying 192.168.60.6...
```

外网主机 10.9.0.5 无法访问其他内网服务器。

```
root@431ad715d95c:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e974fc244391 login:
```

内网主机可以自由访问内网服务器。

```
root@431ad715d95c:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
dd5f32446237 login:
```

内网主机 192.168.60.5 可以访问外网主机 10.9.0.5 的 telnet server。规则验证完毕。

3.4 Task 4: Limiting Network Traffic

限制每秒只允许 10 个数据包从 eth0 接口输出：

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

```
root@c1380e305170:/# iptables -A FORWARD -s 10.9.0.5 -m limit --l
it 10/minute --limit-burst 5 -j ACCEPT
root@c1380e305170:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
root@c1380e305170:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  10.9.0.5             0.0.0.0/0            lim
: avg 10/min burst 5
DROP       all  --  10.9.0.5             0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0            lim
: avg 10/sec burst 5
root@c1380e305170:/#
```

- Host2->Host2 通信正常

```
root@cfb303ffef49:/# 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=64 time=0.112 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.073 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=64 time=0.070 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=64 time=0.080 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=64 time=0.071 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=64 time=0.083 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=64 time=0.076 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=64 time=0.074 ms
^C
--- 192.168.60.5 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8175ms
rtt min/avg/max/mdev = 0.070/0.078/0.112/0.012 ms
root@cfb303ffef49:/#
```

- HostA->Host1 通信有丢包

```
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.419 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.114 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.108 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.117 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.222 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.221 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.174 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.192 ms
64 bytes from 192.168.60.5: icmp_seq=25 ttl=63 time=0.149 ms
64 bytes from 192.168.60.5: icmp_seq=31 ttl=63 time=0.118 ms
64 bytes from 192.168.60.5: icmp_seq=37 ttl=63 time=0.137 ms
64 bytes from 192.168.60.5: icmp_seq=42 ttl=63 time=0.156 ms
64 bytes from 192.168.60.5: icmp_seq=48 ttl=63 time=0.123 ms
64 bytes from 192.168.60.5: icmp_seq=54 ttl=63 time=0.117 ms
64 bytes from 192.168.60.5: icmp_seq=60 ttl=63 time=0.125 ms
64 bytes from 192.168.60.5: icmp_seq=66 ttl=63 time=0.158 ms
64 bytes from 192.168.60.5: icmp_seq=72 ttl=63 time=0.108 ms
^C
--- 192.168.60.5 ping statistics ---
72 packets transmitted, 17 received, 76.3889% packet loss, time 727
67ms
rtt min/avg/max/mdev = 0.108/0.162/0.419/0.073 ms
root@38b090f57c45:/#
```

3.5 Task 5: Load Balancing

这个任务的目的是使用 iptables 的 statistic 模块来实现负载均衡的功能，即将网络流量均匀地分配到多个后端服务器上，以提高性能和可用性。statistic 模块支持两种模式：

nth：根据指定的计数器和间隔跳过或执行规则，实现轮询的负载均衡。

- 将 9090 端口的流量随机地分配到三台后端服务器上，每台服务器的概率为 1/3：

```
iptables -t nat -A PREROUTING -p udp --dport 9090 -m statistic --mode nth --every 3
iptables -t nat -A PREROUTING -p udp --dport 9090 -m statistic --mode nth --every 2
```

```
iptables -t nat -A PREROUTING -p udp --dport 9090 -m statistic --mode nth --every 1
```

```
root@c1380e305170:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  10.9.0.5            0.0.0.0/0            limit
: avg 10/min burst 5
DROP       all  --  10.9.0.5            0.0.0.0/0
ACCEPT     all  --  10.9.0.5            0.0.0.0/0            limit
: avg 10/min burst 5
DROP       all  --  10.9.0.5            0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@c1380e305170:/#
```

- 发包效果，自上到下是 Host1、Host2、Host3：

```
root@cfb303ffef49:/# nc -luk 9090
1564
```

```
root@ea1347fffccf:/# nc -luk 9090
1564
21564
75
dzvzd
 dv
451612
xsfcfdv
```

```
root@a4ec8134f9fe:/# nc -luk 9090
1
2
3
4
5
6
```

- Host 收包如下：

```
root@38b090f57c45:/# nc -u 10.9.0.11 9090
1
2
3
4
5
6
7
8
9
1564
21564
```

random：根据指定的概率跳过或执行规则，实现随机的负载均衡。

- 将 9090 端口的流量随机地分配到三台后端服务器上，每台服务器的概率为 1/3：

```
iptables -t nat -A PREROUTING -p udp --dport 9090 -m statistic --mode random --probal
iptables -t nat -A PREROUTING -p udp --dport 9090 -m statistic --mode random --probal
iptables -t nat -A PREROUTING -p udp --dport 9090 -m statistic --mode random --probal
```

```
root@c1380e305170:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  10.9.0.5            0.0.0.0/0            limit
: avg 10/min burst 5
DROP       all  --  10.9.0.5            0.0.0.0/0
ACCEPT     all  --  10.9.0.5            0.0.0.0/0            limit
: avg 10/min burst 5
DROP       all  --  10.9.0.5            0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@c1380e305170:/#
```

- 发包效果，自上到下是 Host1、Host2、Host3：

```
root@cfb303ffef49:/# nc -luk 9090
165
5555
```

```
root@ea1347fffccf:/# nc -luk 9090
156231
26
5
^[^A
```

```
root@a4ec8134f9fe:/# nc -luk 9090
1
2
3
4
```

- Host 收包如下：

```
root@38b090f57c45:/# nc -u 10.9.0.11 9090
1
2
3
4
165
156231
5555
5
```