The report of lab 6

57118117 谌雨阳

Task 1.A: Implement a Simple Kernel Module

Result：

编译内核并插入模块：

```
[07/24/21]seed@VM:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/kernel_module m
odules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/seed/Desktop/kernel_module/h
ello.o
see include/linux/module.h for more information
  CC [M]  /home/seed/Desktop/kernel_module/hello.mod.o
  LD [M]  /home/seed/Desktop/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[07/24/21]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[07/24/21]seed@VM:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
```

移除插入的内核模块并使用如下命令查看/var/log/syslog 文件，在显示内容末尾找到 hello.ko 打印的两行内容。

```
[07/24/21]seed@VM:~/.../kernel_module$ sudo rmmod hello
[07/24/21]seed@VM:~/.../kernel_module$ dmesg
[    0.000000] Linux version 5.4.0-54-generic (buildd@lcy01-amd64-024) (gcc vers
ion 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04)) #60-Ubuntu SMP Fri Nov 6 10:37:59 UTC
2020 (Ubuntu 5.4.0-54.60-generic 5.4.65)
```

```
[  835.295118] Hello World!
[  872.078972] Bye-bye World!.
[07/24/21]seed@VM:~/.../kernel_module$
```

Task 1.B: Implement a Simple Firewall Using Netfilter

Result：

1、编译内核并插入模块：

```
[07/24/21]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/packet_filter m
odules
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'
[07/24/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/24/21]seed@VM:~/.../packet_filter$ lsmod | grep seed
seedFilter             16384  0
```

接下来测试防火墙效果，看到请求被拦截了：

```
[07/24/21]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
```

为了测试各个 hook 在拦截过程中的调用情况，将 5 种 hook 都加入代码中，代码如下：

```
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_FOWARD;
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 = blockUDP;
hook5.hooknum = NF_INET_POST_ROUTING;
hook5.pf = PF_INET;
hook5.priority = NF_IP_PRI_FIRST;
nf_register_net_hook(&init_net, &hook5);
```

另需要在代码中声明新增加的 3 个 hook：

```
static struct nf_hook_ops hook1, hook2, hook3, hook4, hook5;
```

在 remove 函数中野添加三个 hook：

```
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);
    nf_unregister_net_hook(&init_net, &hook3);
    nf_unregister_net_hook(&init_net, &hook4);
    nf_unregister_net_hook(&init_net, &hook5);
}
```

再次编译内核并重新插入模块：

```
[07/24/21]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/packet_filter m
odules
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'
[07/24/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
insmod: ERROR: could not insert module seedFilter.ko: File exists
[07/24/21]seed@VM:~/.../packet_filter$ sudo rmmod seedFilter
[07/24/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/24/21]seed@VM:~/.../packet_filter$ lsmod | grep seed
seedFilter             16384  0
```

dig www.example.com 进行测试 ：

```
[07/24/21]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
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55373
;; 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.        21138   IN      A       93.184.216.34

;; Query time: 39 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Sat Jul 24 10:56:16 EDT 2021
;; MSG SIZE  rcvd: 60
```

使用 dmesg 查看打印内容：

```
[ 3136.038435] *** LOCAL_OUT
[ 3136.038436]     192.168.43.57  --> 8.8.8.8 (UDP)
[ 3136.038440] *** POST_ROUTING
[ 3136.038440]     192.168.43.57  --> 8.8.8.8 (UDP)
[ 3136.096335] *** PRE_ROUTING
[ 3136.096337]     8.8.8.8  --> 192.168.43.57 (UDP)
[ 3136.096346] *** LOCAL_IN
[ 3136.096347]     8.8.8.8  --> 192.168.43.57 (UDP)
[ 3144.431438] The filters are being removed.
```

发送请求过程中：本机产生的数据包第一个到达的是 LOCAL_OUT 钩子点；随后到达

POST_ROUTING 钩子点，本机产生的书包和需要被转发的数据包都会经过这个钩子点；

接收应答过程中：先到达 PRE_ROUTING 钩子点，这是除了混杂模式以外所有数据包都将经过的钩子点，其钩子函数会在路由判决之前被调用；之后到达 LOCAL_IN 钩子点，在这里数据包进行路由判决，决定转发还是发往主机。

整个过程中没有出现 FORWARD 钩子点，因为只有需要被转发的数据包会到达这个钩子点。

2、在代码中添加两个 hook block 函数如下：

```
unsigned int blockTelnet(void *priv, struct sk_buff *skb,
                              const struct nf_hook_state *state)
{
   struct iphdr *iph;
   struct tcphdr *tcph;

   if (!skb) return NF_ACCEPT;

   iph = ip_hdr(skb);
   tcph = tcp_hdr(skb);

   if (iph->protocol == IPPROTO_TCP && ntohs(tcph->dest) == 23 ) {
        printk(KERN_WARNING "*** Dropping Telnet packet from %p\n", &(iph->daddr));
        return NF_DROP;
   }
   return NF_ACCEPT;
}

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

   if (!skb) return NF_ACCEPT;

   iph = ip_hdr(skb);

   if (iph->protocol == IPPROTO_ICMP) {
        printk(KERN_WARNING "*** Dropping ICMP packet from %p\n", &(iph->daddr));
        return NF_DROP;
   }
   return NF_ACCEPT;
}
```

修改两个 hook 配置如下：

```c
int registerFilter(void) {
    printk(KERN_INFO "Registering filters.\n");

    hook1.hook = blockTelnet;
    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 = blockICMP;
    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;
}
```

重新编译内核并插入模块：

```
[07/24/21]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/packet_filter m
odules
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'
[07/24/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/24/21]seed@VM:~/.../packet_filter$ lsmod | grep seed
seedFilter             16384  0
```

进入 user 机（10.9.0.5）中对 10.9.0.1 进行 ping 和 telnet 操作，都不能成功：

```
root@e7cb5792fc68:/# 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 ---
4 packets transmitted, 0 received, 100% packet loss, time 3053ms

root@e7cb5792fc68:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
root@e7cb5792fc68:/#
```

使用 dmesg 命令查看内核打印内容，打印了 hook 函数的拦截记录：

```
[ 6335.863003] *** Dropping ICMP packet from 00000000f19839d8
[ 6336.865238] *** Dropping ICMP packet from 00000000f19839d8
[ 6337.889966] *** Dropping ICMP packet from 000000005bc0b73c
[ 6338.915551] *** Dropping ICMP packet from 000000005bc0b73c
[ 6347.762824] *** Dropping Telnet packet from 00000000a4ba782f
[ 6348.769143] *** Dropping Telnet packet from 00000000a4ba782f
[ 6350.785329] *** Dropping Telnet packet from 00000000a4ba782f
```

Task 2.A: Protecting the Router

Result：

在配置 iptables 防火墙之前，先在 user 机上对 router 进行 ping 操作和 telnet 操作的测试，发现两种操作都能成功：

```
root@e7cb5792fc68:/# 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=0.119 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.054 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.081 ms
^C
--- 10.9.0.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2054ms
rtt min/avg/max/mdev = 0.054/0.084/0.119/0.026 ms
root@e7cb5792fc68:/# telnet 10.9.0.11
Trying 10.9.0.11...
Connected to 10.9.0.11.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e57f6a8caf43 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

然后使用如下命令在防火墙 Filter 表中添加规则，四条命令的含义分别是：允许发送 ICMP 应答，允许接收 ICMP 请求，丢弃所有发送报文和丢弃所有接收报文。

```
root@e57f6a8caf43:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@e57f6a8caf43:/# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEP
T
```

```
root@e57f6a8caf43:/# iptables -P OUTPUT DROP
root@e57f6a8caf43:/# iptables -P INPUT DROP
```

查看 filter 表中存有的规则：

```
root@e57f6a8caf43:/# iptables -t filter -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0
```

在 user 机上再次尝试对 router 进行 ping 和 telnet 操作，发现 ping 依旧可以 ping 通，但 telnet 无法连接。说明我们为防火墙添加的规则完成了只允许 ICMP 报文的交互的目的：

```
root@e7cb5792fc68:/# 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=0.063 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.052 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.053 ms
^C
--- 10.9.0.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2057ms
rtt min/avg/max/mdev = 0.052/0.056/0.063/0.005 ms
root@e7cb5792fc68:/# telnet 10.9.0.11
Trying 10.9.0.11...
```

Task 2.B: Protecting the Internal Network

Result：

先通过 ifconfig 命令找到 router 接口与内外网的对应关系，得知 eth0 对应外网，eth1 对应内网（192.168.60.0/24）。

```
root@e57f6a8caf43:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.11  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:0b  txqueuelen 0  (Ethernet)
        RX packets 73  bytes 8083 (8.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.60.11  netmask 255.255.255.0  broadcast 192.168.60.255
        ether 02:42:c0:a8:3c:0b  txqueuelen 0  (Ethernet)
        RX packets 72  bytes 8045 (8.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

新增 FORWARD 表中的规则如下：

```
root@e57f6a8caf43:/# iptables -A FORWARD -p icmp --icmp-type echo-request -i eth1 -j ACCEPT
root@e57f6a8caf43:/# iptables -A FORWARD -p icmp --icmp-type echo-request -o eth0 -j ACCEPT
root@e57f6a8caf43:/# iptables -A FORWARD -p icmp --icmp-type echo-reply -i eth0 -j ACCEPT
root@e57f6a8caf43:/# iptables -A FORWARD -p icmp --icmp-type echo-reply -o eth1 -j ACCEPT
```

```
root@e57f6a8caf43:/# iptables -P FORWARD DROP
```

上面的规则分别完成的功能是：允许接收从内网主机中发来的 icmp_request 包，允许转发向外网主机发送的 icmp_request 包，允许接收从外网主机中发来的 icmp_reply 包，允许转发向内网主机发送的 icmp_reply 包，丢弃所有其他 FORWARD 报文。

规则表如下：

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0
```

由此可以实现题设所需的保护内网功能，测试如下：

（1）外网主机无法 ping 通内网主机：

```
root@e7cb5792fc68:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

（2）外网主机可以 ping 通路由器：

```
root@e7cb5792fc68:/# 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=0.088 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.089 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.067 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.085 ms
```

（3）内网主机可以 ping 通外网主机：

```
root@0aefa4cfbc36:/# 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=0.356 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.082 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.064 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.097 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.060 ms
```

（4）其它报文会被阻碍（此处以 telnet 为例）：

```
root@0aefa4cfbc36:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

Task 2.C: Protecting Internal Servers

Result：

新增 FORWARD 表中的规则如下：

```
root@e57f6a8caf43:/# iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport 23 -j ACCEPT
root@e57f6a8caf43:/# iptables -A FORWARD -i eth1 -p tcp -s 192.168.60.5 -j ACCEPT
root@e57f6a8caf43:/# iptables -A FORWARD -o eth0 -p tcp -s 192.168.60.5 -j ACCEPT
root@e57f6a8caf43:/# iptables -A FORWARD -o eth1 -p tcp -d 192.168.60.5 --dport 23 -j ACCEPT
```

上述四条规则含义如下：允许接收来自外网，宿地址为 192.168.60.5，宿端口为 23 的 tcp 包；允许接受来自内网，源地址为 192.168.60.5 的 tcp 包；允许转发去往外网，源地址为 192.168.60.5 的 tcp 包；允许转发去往内网，宿地址为 192.168.60.5，宿端口为 23 的 tcp 包。

规则表如下：

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            192.168.60.5         tcp dpt:23
ACCEPT     tcp  --  192.168.60.5         0.0.0.0/0
ACCEPT     tcp  --  192.168.60.5         0.0.0.0/0
ACCEPT     tcp  --  0.0.0.0/0            192.168.60.5         tcp dpt:23

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0
```

（1）外网主机只能 telnet 192.168.60.5 主机：

```
root@e7cb5792fc68:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
0aefa4cfbc36 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

（2）外网主机无法 telnet 192.168.60.0/24 的其他主机：

```
root@e7cb5792fc68:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
```

（3）内网主机可以相互访问：

```
root@0aefa4cfbc36:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
35b9d7975a47 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

（4）内网主机无法 telnet 外网主机：

```
seed@35b9d7975a47:~$ telnet 10.9.0.5
Trying 10.9.0.5...
```

Task 3.A: Experiment with the Connection Tracking

Result：

ICMP：

在 router 上运行如下命令，可以查看连接跟踪信息：

```
root@e57f6a8caf43:/# conntrack -L
tcp      6 427723 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=54338 dport=23 src=192.168.60
.5 dst=10.9.0.5 sport=23 dport=54338 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

测试发现一个新的 ICMP 连接持续时间为 30s，若该连接不断有报文发送则时间不断刷新，否则就减少直到 0，因此若在上一次 ping 后 30s 内再 ping 一次就会有两个连接：

刷新：

```
root@c17ff66987d7:/# conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=30 src=192.168.60.5 dst=10.9.0.5 t
ype=0 code=0 id=30 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@c17ff66987d7:/# conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=30 src=192.168.60.5 dst=10.9.0.5 t
ype=0 code=0 id=30 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@c17ff66987d7:/# conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=30 src=192.168.60.5 dst=10.9.0.5 t
ype=0 code=0 id=30 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

下降：

```
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=31 src=192.168.60.5 dst=10.9.0.5 t
ype=0 code=0 id=31 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@c17ff66987d7:/# conntrack -L
icmp     1 25 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=31 src=192.168.60.5 dst=10.9.0.5 t
ype=0 code=0 id=31 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@c17ff66987d7:/# conntrack -L
icmp     1 24 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=31 src=192.168.60.5 dst=10.9.0.5 t
ype=0 code=0 id=31 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@c17ff66987d7:/# conntrack -L
icmp     1 22 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=31 src=192.168.60.5 dst=10.9.0.5 t
ype=0 code=0 id=31 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@c17ff66987d7:/# conntrack -L
icmp     1 14 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=31 src=192.168.60.5 dst=10.9.0.5 t
ype=0 code=0 id=31 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

两个连接：

```
root@c17ff66987d7:/# conntrack -L
icmp     1 23 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=36 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=36 mark=0 use=1
icmp     1 28 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=37 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=37 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
```

UDP：

同样也是持续 30s：

```
udp      17 28 src=10.9.0.5 dst=192.168.60.5 sport=40325 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=40325 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@c17ff66987d7:/# conntrack -L
udp      17 20 src=10.9.0.5 dst=192.168.60.5 sport=40325 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=40325 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@c17ff66987d7:/# conntrack -L
udp      17 17 src=10.9.0.5 dst=192.168.60.5 sport=40325 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=40325 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

TCP：

测试发现 TCP 连接持续时间是 432000s。在断开连接后还会在 conntrack 中显示约 120s。

```
root@c17ff66987d7:/# conntrack -L
tcp      6 431997 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=36446 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=36446 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@c17ff66987d7:/# conntrack -L
tcp      6 431990 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=36446 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=36446 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

断开连接后：

```
tcp      6 116 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=36446 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=36446 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@c17ff66987d7:/# conntrack -L
tcp      6 113 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=36446 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=36446 [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

Result：

在 Task2.C 基础上新增如下两条规则：

```
root@c17ff66987d7:/# iptables -A FORWARD -i eth1 -p tcp --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
root@c17ff66987d7:/# iptables -A FORWARD -o eth0 -p tcp --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
```

增加规则后内网主机就可以任意访问外网主机了：

```
root@8b62376fa8fb:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
989424c60fde login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

而外网主机还是只能访问内网的 192.168.60.5 主机：

```
root@989424c60fde:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@989424c60fde:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
8b62376fa8fb login:
```

Task 4: Limiting Network Traffic

Result：

先只加入第一条规则：

```
root@48bf36efc8ea:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute
--limit-burst 5 -j ACCEPT
```

在 10.9.0.5 中 ping 192.168.60.5，结果如下，能够 ping 通且没有丢包：

```
[07/24/21]seed@VM:~$ docksh 9c
root@9c1021450aa8:/# 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=0.136 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.065 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.100 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.115 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.080 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.123 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.070 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.069 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.105 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.065 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.066 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.062 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.155 ms
64 bytes from 192.168.60.5: icmp_seq=14 ttl=63 time=0.062 ms
64 bytes from 192.168.60.5: icmp_seq=15 ttl=63 time=0.065 ms
64 bytes from 192.168.60.5: icmp_seq=16 ttl=63 time=0.098 ms
64 bytes from 192.168.60.5: icmp_seq=17 ttl=63 time=0.070 ms
64 bytes from 192.168.60.5: icmp_seq=18 ttl=63 time=0.064 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.385 ms
^C
--- 192.168.60.5 ping statistics ---
19 packets transmitted, 19 received, 0% packet loss, time 18429ms
rtt min/avg/max/mdev = 0.062/0.102/0.385/0.071 ms
```

然后再加入第二条规则：

```
root@48bf36efc8ea:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
```

再次尝试在 10.9.0.5 中 ping 192.168.60.5，结果如下，能够 ping 通但存在丢包问题：

```
root@0fd7acea6749:/# 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=0.071 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.071 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.180 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.064 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.065 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.062 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.088 ms
^C
--- 192.168.60.5 ping statistics ---
22 packets transmitted, 7 received, 68.1818% packet loss, time 21486ms
rtt min/avg/max/mdev = 0.062/0.085/0.180/0.039 ms
```

该实验结果表明在仅有一条流量限制规则的情况下，超出该规则限制的报文会去匹配 FORWARD 的默认规则，所以会被接收；而添加了第二条规则后超出第一条规则的报文会匹配第二条，所以才会被丢弃。

Task 5: Load Balancing

Result：

采用 nth 方法进行负载均衡：

在 router 中添加如下规则，为了达到三台主机负载均衡的目的，设置第一条：每 3 个包中有一个到达 192.168.60.5，之后匹配第二条：每 2 个包有一个到达 192.168.60.6，之后匹配第三条：剩下的包都到达 192.168.60.7：

```
root@17f4b4166abf:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode nth --every 3 --packet 0 -j DNAT --to-destination 192.168.60.5:8080
root@17f4b4166abf:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode nth --every 2 --packet 0 -j DNAT --to-destination 192.168.60.6:8080
root@17f4b4166abf:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT -
-to-destination 192.168.60.7:8080
```

在 192.168.60.5、192.168.60.6、192.168.60.7 上分别开启 8080 端口监听：

```
root@1a5ce7f60d4b:/# nc -luk 8080
```

```
root@f637782beaa5:/# nc -luk 8080
```

```
root@e92aa023af4f:/# nc -luk 8080
```

在 10.9.0.5 端口上通过管道输出到 router（10.9.0.11）：

```
root@2c1b8a9f730d:/# echo hello | nc -u 10.9.0.11 8080
^C
```

每进行 3 次发送，192.168.60.5，192.168.60.6,192.168.60.7 上各能接收到一个 hello：

```
root@2e37538e7dbf:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2e37538e7dbf:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2e37538e7dbf:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2e37538e7dbf:/# echo hello | nc -u 10.9.0.11 8080
^C^C
root@2e37538e7dbf:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2e37538e7dbf:/# echo hello | nc -u 10.9.0.11 8080
^C
```

192.168.60.5：

```
root@1a5ce7f60d4b:/# nc -luk 8080
hello
hello
```

192.168.60.6：

```
root@f637782beaa5:/# nc -luk 8080
hello
hello
```

192.168.60.7：

```
root@e92aa023af4f:/# nc -luk 8080
hello
hello
```

采用 random mode 进行负载均衡：

加入如下规则：

```
root@dbd96c5c33e0:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode random --probability 0.33 -j DNAT --to-destination 192.168.60.5:8080
root@dbd96c5c33e0:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode random --probability 0.5 -j DNAT --to-destination 192.168.60.6:8080
root@dbd96c5c33e0:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT -
-to-destination 192.168.60.7:8080
```

由于各个规则是独立进行匹配的，因此对 192.168.60.5 分配概率为 0.33 后，60.6 和 60.7 平分剩下 0.67，此时应将 60.6 概率设置为 0.5，60.7 为全部接收，则可以达到三台主机负载均衡。

```
root@2287c180a04a:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2287c180a04a:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2287c180a04a:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2287c180a04a:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2287c180a04a:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2287c180a04a:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2287c180a04a:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2287c180a04a:/# echo hello | nc -u 10.9.0.11 8080
^C
root@2287c180a04a:/# echo hello | nc -u 10.9.0.11 8080
^C
```

共发出 9 个 hello，三台主机各接收到 3 个，结果如下：

192.168.60.5：

```
root@89cfb30ddae1:/# nc -luk 8080
hello
hello
hello
```

192.168.60.6：

```
root@2c99f0245780:/# nc -luk 8080
hello
hello
hello
```

192.168.60.7：

```
root@a87dcd5de207:/# nc -luk 8080
hello
hello
hello
```

实验结果表明 random mode 的方法在发出报文数量较多的时候能够达到比较好的负载均衡效果。