哈尔滨工业大学(深圳)

《网络与系统安全》 实验报告

实验六

防火墙 实验

学 院: 计算机科学与技术学院

姓 名: 房煊梓

学 号: 210010101

专 业: 智能强基-计算机

日 期: 2024 年 5 月 17 日

1. Task1：加载 seedFilter 模块，执行 dig dig @8.8.8.8 www.example.com，卸载 seedFilter 后再执行 dmesg 命令查看内核日志，把日志信息中加载、卸载 seefFilter 模块以及阻止 UDP 数据包的信息截图，并进行分析说明。

(1)日志信息中加载 seedFilter 模块的截图如下：

```
[  312.812387] Hello World!
[  328.122383] Bye-bye World!.
[  699.985117] Registering filters.
[  728.287854] *** LOCAL_OUT
[  728.287869]     10.0.2.15  --> 34.149.100.209 (TCP)
[  728.287897] *** LOCAL_OUT
[  728.287901]     10.0.2.15  --> 34.149.100.209 (TCP)
```

分析：由图可知 Registering filters 这一行是在加载 seedFilter 模块。

(2) 日志信息中卸载 seedFilter 模块的截图如下：

```
[  837.340001] *** LOCAL_OUT
[  837.340003]     10.0.2.15  --> 34.149.100.209 (TCP)
[  837.596075] *** LOCAL_OUT
[  837.596077]     10.0.2.15  --> 34.149.100.209 (TCP)
[  851.676039] The filters are being removed.
```

分析：由图可知 The filters are being removed 这一行是在卸载 seedFilter 模块。

(3) 日志信息中阻止 UDP 数据包的信息截图如下：

```
[  756.521740] *** LOCAL_OUT
[  756.521740]     10.0.2.15  --> 8.8.8.8 (UDP)
[  756.521743] *** Dropping 8.8.8.8 (UDP), port 53
[  757.344454] *** LOCAL_OUT
[  757.344459]     10.0.2.15  --> 34.149.100.209 (TCP)
[  757.600161] *** LOCAL_OUT
[  757.600174]     10.0.2.15  --> 34.149.100.209 (TCP)
[  761.523983] *** LOCAL_OUT
[  761.524025]     10.0.2.15  --> 8.8.8.8 (UDP)
[  761.524155] *** Dropping 8.8.8.8 (UDP), port 53
[  761.568299] *** LOCAL_OUT
[  761.568305]     10.0.2.15  --> 34.149.100.209 (TCP)
[  761.824391] *** LOCAL_OUT
[  761.824397]     10.0.2.15  --> 34.149.100.209 (TCP)
[  766.527910] *** LOCAL_OUT
[  766.527912]     10.0.2.15  --> 8.8.8.8 (UDP)
[  766.527919] *** Dropping 8.8.8.8 (UDP), port 53
[  769.760667] *** LOCAL_OUT
[  769.760671]     10.0.2.15  --> 34.149.100.209 (TCP)
[  770.016880] *** LOCAL_OUT
[  770.016895]     10.0.2.15  --> 34.149.100.209 (TCP)
[  778.135105] *** LOCAL_OUT
[  778.135112]     10.0.2.15  --> 185.125.190.56 (UDP)
[  781.160497] *** LOCAL_OUT
[  781.160498]     127.0.0.1  --> 127.0.0.1 (UDP)
[  781.160587] *** LOCAL_OUT
[  781.160587]     10.0.2.15  --> 8.8.8.8 (UDP)
[  781.160591] *** Dropping 8.8.8.8 (UDP), port 53
[  785.887652] *** LOCAL_OUT
[  785.887654]     10.0.2.15  --> 34.149.100.209 (TCP)
[  786.143491] *** LOCAL_OUT
[  786.143492]     10.0.2.15  --> 34.149.100.209 (TCP)
[  786.159490] *** LOCAL_OUT
[  786.159491]     10.0.2.15  --> 8.8.8.8 (UDP)
[  786.159498] *** Dropping 8.8.8.8 (UDP), port 53
[  791.159194] *** LOCAL_OUT
[  791.159199]     10.0.2.15  --> 8.8.8.8 (UDP)
[  791.159219] *** Dropping 8.8.8.8 (UDP), port 53
[  805.962818] *** LOCAL_OUT
[  805.962824]     10.0.2.15  --> 34.149.100.209 (TCP)
[  806.216158] *** LOCAL_OUT
[  806.216163]     10.0.2.15  --> 34.149.100.209 (TCP)
```

分析：由图可知，Dropping 8.8.8.8(UDP),port 53 是丢弃所有目的 IP 地址为 8.8.8.8，目的端口号为 53 的 UDP 数据包，说明成功阻止了 UDP 数据包。

2. Task2：阻止 TCP 端口和 PING，把增加和修改的代码截图，并在卸载模块后将 dmesg 的日志信息的截图，并分析说明原因。

(1)增加和修改的代码截图：

```
1 #include <linux/kernel.h>
2 #include <linux/module.h>
3 #include <linux/netfilter.h>
4 #include <linux/netfilter_ipv4.h>
5 #include <linux/ip.h>
6 #include <linux/tcp.h>
7 #include <linux/udp.h>
8 #include <linux/icmp.h>
9 #include <linux/if_ether.h>
10 #include <linux/inet.h>
```

头文件：添加了<linux/icmp.h>用于后续程序编写。

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

定义结构体变量：增加 hook3 和 hook4。

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

   //u16  port   = 53;
   char ip[16] = "10.9.0.1";
   u32  ip_addr;

   if (!skb) return NF_ACCEPT;

   iph = ip_hdr(skb);
   // Convert the IPv4 address from dotted decimal to 32-bit binary
   in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);

   if (iph->protocol == IPPROTO_ICMP) {
       icmph = icmp_hdr(skb);
       if (iph->daddr == ip_addr && icmph->type==ICMP_ECHO){
            printk(KERN_WARNING "*** Dropping %pI4 (ICMP)\n", &(iph->daddr));
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
}
```

blockICMP 函数：根据实验指导书可知需要使 ping 10.9.0.1 和 telnet 10.9.0.1 命令执行失败，而 ping 命令使用 ICMP 协议，故 blockICMP 函数要阻止 ping 10.9.0.1。因此设置 char ip[16]字符串为“10.9.0.1”。若 iph 的协议为 ICMP，则获取相应的 icmph，若 iph->daddr 与 ip_addr 相同且 icmph->type 为 ICMP_ECHO，即目的 IP 地址为 10.9.0.1 的 ping 请求，则打印相应的 warning 信息并返回 NF_DROP 表示进行丢弃的操作。

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

   u16  port   = 23;
   char ip[16] = "10.9.0.1";
   u32  ip_addr;

   if (!skb) return NF_ACCEPT;

   iph = ip_hdr(skb);
   // Convert the IPv4 address from dotted decimal to 32-bit binary
   in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);

   if (iph->protocol == IPPROTO_TCP) {
       tcph = tcp_hdr(skb);
       if (iph->daddr == ip_addr && ntohs(tcph->dest) == port){
           printk(KERN_WARNING "*** Dropping %pI4 (TCP), port %d\n", &(iph->daddr), port);
           return NF_DROP;
        }
   }
   return NF_ACCEPT;
}
```

blockTCP 函数：根据实验指导书可知需要使 ping 10.9.0.1 和 telnet 10.9.0.1 命令执行失败，而 telnet 命令使用 TCP 协议，故 blockTCP 函数要阻止 telnet 10.9.0.1。因此设置 char ip[16]字符串为“10.9.0.1”，并且参照实验指导书设置 port 为 23。若 iph 的协议为 TCP，则获取相应的 tcph，若 iph->daddr 与 ip_addr 相同且 ntohs(tcph->dest)为 port，即目的 IP 地址为 10.9.0.1，目的端口号为 23 的 telnet 请求，则打印相应的 warning 信息并返回 NF_DROP 表示进行丢弃的操作。

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

   hook1.hook = printInfo;
   hook1.hooknum = NF_INET_LOCAL_OUT;
   hook1.pf = PF_INET;
   hook1.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook1);

   hook2.hook = blockUDP;
   hook2.hooknum = NF_INET_POST_ROUTING;
   hook2.pf = PF_INET;
   hook2.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook2);

   hook3.hook = blockICMP;
   hook3.hooknum = NF_INET_PRE_ROUTING;
   hook3.pf = PF_INET;
   hook3.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook3);

   hook4.hook = blockTCP;
   hook4.hooknum = NF_INET_PRE_ROUTING;
   hook4.pf = PF_INET;
   hook4.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook4);

   return 0;
}
```

registerFilter 函数：仿照 hook1 和 hook2 的写法，增加了 hook3 和 hook4 的部分，其中 hooknum 不同。hook3 和 hook4 使用 NT_INET_PRE_ROUTING，因为除了混杂模式，所有数据包都将经过这个钩子点，它上面注册的钩子函数在路由判决前被调用，适用于 hook3 和 hook4 对应的情况。

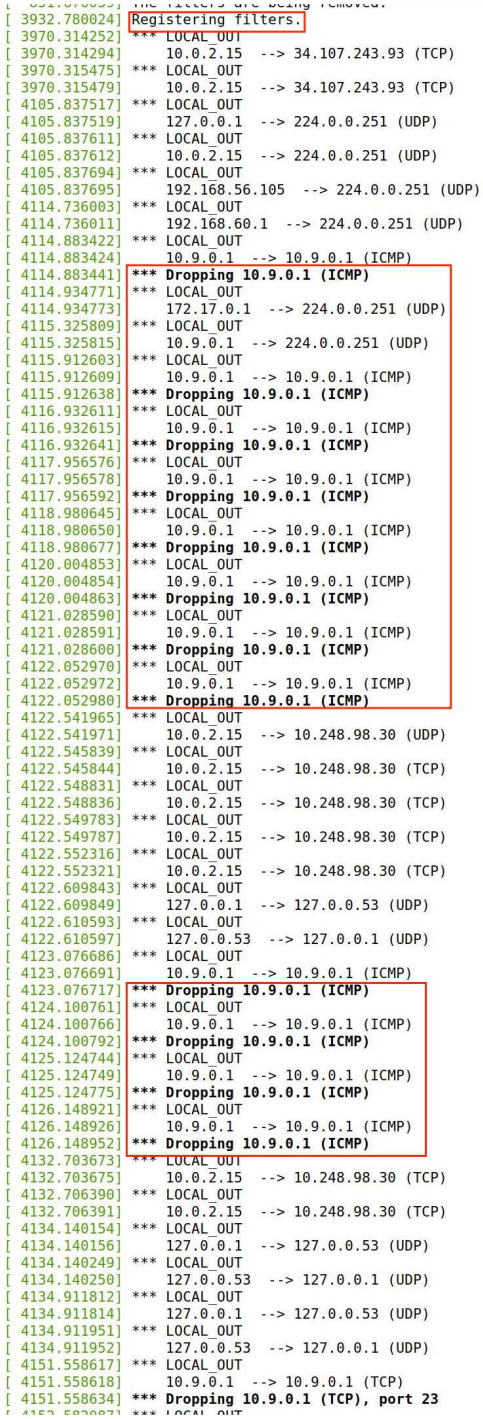
```
void removeFilter(void) {
    printk(KERN_INFO "The filters are being removed.\n");
    nf_unregister_net_hook(&init_net, &hook1);
    nf_unregister_net_hook(&init_net, &hook2);
    nf_unregister_net_hook(&init_net, &hook3);
    nf_unregister_net_hook(&init_net, &hook4);
}

module_init(registerFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

removeFilter 函数：仿照 hook1 和 hook2，增加了 hook3 和 hook4 的部分。

(2)卸载模块后的日志信息截图如下两张图：



```
[ 4151.558618]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4151.558634] *** Dropping 10.9.0.1 (TCP), port 23
[ 4152.582087] *** LOCAL_OUT
[ 4152.582133]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4152.582856] *** Dropping 10.9.0.1 (TCP), port 23
[ 4154.597605] *** LOCAL_OUT
[ 4154.597646]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4154.598297] *** Dropping 10.9.0.1 (TCP), port 23
[ 4158.757988] *** LOCAL_OUT
[ 4158.757992]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4158.758069] *** Dropping 10.9.0.1 (TCP), port 23
[ 4166.949966] *** LOCAL_OUT
[ 4166.949993]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4166.950050] *** Dropping 10.9.0.1 (TCP), port 23
[ 4183.078106] *** LOCAL_OUT
[ 4183.078153]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4183.078243] *** Dropping 10.9.0.1 (TCP), port 23
[ 4196.662690] *** LOCAL_OUT
[ 4196.662691]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4196.662709] *** Dropping 10.9.0.1 (TCP), port 23
[ 4197.670006] *** LOCAL_OUT
[ 4197.670052]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4197.670773] *** Dropping 10.9.0.1 (TCP), port 23
[ 4199.686687] *** LOCAL_OUT
[ 4199.686700]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4199.686902] *** Dropping 10.9.0.1 (TCP), port 23
[ 4203.814033] *** LOCAL_OUT
[ 4203.814049]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4203.814267] *** Dropping 10.9.0.1 (TCP), port 23
[ 4212.006796] *** LOCAL_OUT
[ 4212.006819]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4212.007212] *** Dropping 10.9.0.1 (TCP), port 23
[ 4212.345299] *** LOCAL_OUT
[ 4212.345305]     192.168.56.105  --> 192.168.56.100 (UDP)
[ 4212.510499] *** LOCAL_OUT
[ 4212.510500]     10.0.2.15  --> 10.248.98.30 (UDP)
[ 4212.513226] *** LOCAL_OUT
[ 4212.513228]     10.0.2.15  --> 91.189.91.48 (TCP)
[ 4212.756350] *** LOCAL_OUT
[ 4212.756392]     10.0.2.15  --> 91.189.91.48 (TCP)
[ 4212.757590] *** LOCAL_OUT
[ 4212.757595]     10.0.2.15  --> 91.189.91.48 (TCP)
[ 4213.000300] *** LOCAL_OUT
[ 4213.000314]     10.0.2.15  --> 91.189.91.48 (TCP)
[ 4213.000664] *** LOCAL_OUT
[ 4213.000665]     10.0.2.15  --> 91.189.91.48 (TCP)
[ 4228.134701] *** LOCAL_OUT
[ 4228.134707]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4228.134739] *** Dropping 10.9.0.1 (TCP), port 23
[ 4261.159100] *** LOCAL_OUT
[ 4261.159146]     10.9.0.1  --> 10.9.0.1 (TCP)
[ 4261.159868] *** Dropping 10.9.0.1 (TCP), port 23
[ 4270.533671] *** LOCAL_OUT
[ 4270.533677]     10.0.2.15  --> 34.107.243.93 (TCP)
[ 4363.707897] *** LOCAL_OUT
[ 4363.707899]     10.0.2.15  --> 185.125.190.56 (UDP)
[ 4422.573957] *** LOCAL_OUT
[ 4422.573959]     10.0.2.15  --> 10.248.98.30 (UDP)
[ 4422.605634] *** LOCAL_OUT
[ 4422.605636]     127.0.0.1  --> 127.0.0.53 (UDP)
[ 4422.605747] *** LOCAL_OUT
[ 4422.605748]     127.0.0.53  --> 127.0.0.1 (UDP)
[ 4434.137939] *** LOCAL_OUT
[ 4434.137945]     127.0.0.1  --> 127.0.0.53 (UDP)
[ 4434.138334] *** LOCAL_OUT
[ 4434.138337]     127.0.0.53  --> 127.0.0.1 (UDP)
[ 4434.907515] *** LOCAL_OUT
[ 4434.907516]     127.0.0.1  --> 127.0.0.53 (UDP)
[ 4434.907612] *** LOCAL_OUT
[ 4434.907613]     127.0.0.53  --> 127.0.0.1 (UDP)
[ 4450.121586] The filters are being removed.
[05/15/24]seed@VM:~$
```

分析：由以上两张图可知 Dropping 10.9.0.1(ICMP)是丢弃所有目的 IP 地址为 10.9.0.1 的 ICMP 数据包，由于 ping 使用 ICMP 协议，因此成功阻止 ping。Dropping 10.9.0.1(TCP),port 23 是丢弃所有目的 IP 地址为 10.9.0.1，目的端口号为 23 的 TCP 数据包，说明成功阻止了 TCP 数据包。

3. Task3：保护 Router，将配置 iptables 规则前后 ping 和 telnet 的连通性测试结果截图，并分析说明原因。

(1)配置规则前的连通性测试截图：

```
root@636271af4666:/# 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=0.102 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.102 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.102 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.103 ms
64 bytes from 10.9.0.11: icmp_seq=6 ttl=64 time=0.101 ms
^C
--- 10.9.0.11 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5108ms
rtt min/avg/max/mdev = 0.063/0.095/0.103/0.014 ms
root@636271af4666:/# telnet 10.9.0.11
Trying 10.9.0.11...
Connected to 10.9.0.11.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
73277cae9bb6 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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

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.
```

由图可知在配置规则前 ping 10.9.0.11 和 telnet 10.9.0.11 均可以连通。

(2)配置规则后的连通性测试截图：

```
[05/15/24]seed@VM:~$ docksh 63
root@636271af4666:/# 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=0.050 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.104 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.105 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.105 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.106 ms
64 bytes from 10.9.0.11: icmp_seq=6 ttl=64 time=0.109 ms
^C
--- 10.9.0.11 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5119ms
rtt min/avg/max/mdev = 0.050/0.096/0.109/0.020 ms
root@636271af4666:/# telnet 10.9.0.11
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
```

由图可知配置规则后 ping 10.9.0.11 可以连通而 telnet 10.9.0.11 则不能连通。

(3)**分析**：配置规则前面两行是设置 Router 允许 ICMP 类型协议的应答，下面两行是其他没有设置的协议类型默认拒绝。由于 ping 使用 ICMP 协议，telnet 使用 TCP 协议，因此配置规则后，ping 时允许应答，能够连通；telnet 时默认拒绝，不能连通。

4、Task4：保护内网，将配置 iptables 规则前后 ping 的连通性测试结果截图，并分析说明原因。

(1)配置规则前的连通性测试截图：

```
root@dcda94e610eb:/# 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=0.092 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.052 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.133 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.056 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.083 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.140 ms
^C
--- 192.168.60.5 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5097ms
rtt min/avg/max/mdev = 0.052/0.092/0.140/0.034 ms
root@dcda94e610eb:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
7490ad0943a4 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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

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.

seed@7490ad0943a4:~$ exit
logout
Connection closed by foreign host.
root@dcda94e610eb:/#
```

由图可知在配置规则前在 HostA 容器中执行 ping 192.168.60.5 和 telent 192.168.60.5 均可以连通。

(2)配置规则后的连通性测试截图：

HostA：

```
[05/16/24]seed@VM:~/.../Labsetup$ docksh dc
root@dcda94e610eb:/# 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 ---
7 packets transmitted, 0 received, 100% packet loss, time 6126ms

root@dcda94e610eb:/# telnet 192.168.60.5
Trying 192.168.60.5...
telnet: Unable to connect to remote host: Connection timed out
```

由图可知在配置规则后在 HostA 容器中执行 ping 192.168.60.5 和 telent 192.168.60.5 均不能连通。

Host1:

```
[05/16/24]seed@VM:~/.../Labsetup$ docksh 74
root@7490ad0943a4:/# ping 192.168.60.11
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
64 bytes from 192.168.60.11: icmp_seq=1 ttl=64 time=0.058 ms
64 bytes from 192.168.60.11: icmp_seq=2 ttl=64 time=0.105 ms
64 bytes from 192.168.60.11: icmp_seq=3 ttl=64 time=0.035 ms
64 bytes from 192.168.60.11: icmp_seq=4 ttl=64 time=0.037 ms
64 bytes from 192.168.60.11: icmp_seq=5 ttl=64 time=0.104 ms
^C
--- 192.168.60.11 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4096ms
rtt min/avg/max/mdev = 0.035/0.067/0.105/0.031 ms
root@7490ad0943a4:/# 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=0.058 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.134 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.136 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.139 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.134 ms
^C
--- 10.9.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4084ms
rtt min/avg/max/mdev = 0.058/0.120/0.139/0.031 ms
root@7490ad0943a4:/#
```

由图可知在配置规则后在 Host1 容器中执行 ping 192.168.60.11 和 ping 10.9.0.5 均可以连通。

(3)**分析**：配置规则第一行设置 Router 不允许通过 eth0 转发的 ICMP 类型协议的请求，第二行设置 Router 允许通过 eth1 转发的 ICMP 类型协议的请求，第三行设置 Router 允许 ICMP 类型协议的应答，第四行设置其他没有设置的协议类型默认拒绝。因此，配置规则之后：

①HostA 执行 ping 192.168.60.5 时，是通过 eth0 转发 ICMP 请求，根据第一行设置可知不允许通过请求，因此不能连通。

②HostA 执行 telent 192.168.60.5 时，由于 telnet 使用 TCP 协议，根据第四行设置可知默认拒绝，因此不能连通。

③Host1 执行 ping 192.168.60.11 时，是通过 eth1 转发 ICMP 请求，根据第二行设置可知允许通过请求，并且根据第三行设置可知允许应答，因此能连通。

④Host1 执行 ping 10.9.0.5 时，是通过 eth1 转发 ICMP 请求，根据第二行设置可知允许通过请求，并且根据第三行设置可知允许应答，因此能连通。