

# Firewall Exploration Lab

57118203 陈萱妍

## Task 1: Implementing a Simple Firewall

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

创建一个路径不含空格的文件夹，将路径/home/seed/Desktop/Labs\_20.04/Network Security/Firewall Exploration Lab/Labsetup/Files/kernel\_module 下的 hello.c 和 makefile 两个文件拷贝到新创建的文件夹中。

输入命令 make 编译内核。

```
[07/28/21]seed@VM:~/.../CXY$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Labs_20.04/CXY/modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M] /home/seed/Desktop/Labs_20.04/CXY/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/seed/Desktop/Labs_20.04/CXY/hello.o
see include/linux/module.h for more information
  CC [M] /home/seed/Desktop/Labs_20.04/CXY/hello.mod.o
  LD [M] /home/seed/Desktop/Labs_20.04/CXY/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

输入命令 sudo insmod hello.ko, 将内核模块 hello 载入内核。输入命令, 查看 hello 模块载入内核的情况。

```
[07/28/21]seed@VM:~/.../CXY$ sudo insmod hello.ko
[07/28/21]seed@VM:~/.../CXY$ lsmod | grep hello
hello                  16384  0
```

输入命令 dmesg, 查看模块输出的详细信息。

```
[13008.360917] eth0: renamed from veth12d00c7
[13008.380389] eth0: renamed from veth1e33eae
[13008.430453] IPv6: ADDRCONF(NETDEV_CHANGE): veth227517d: link becomes ready
[13008.430955] br-c866e28ac0e2: port 2(veth227517d) entered blocking state
[13008.430964] br-c866e28ac0e2: port 2(veth227517d) entered forwarding state
[13008.448691] IPv6: ADDRCONF(NETDEV_CHANGE): vethedd196d: link becomes ready
[13008.448773] br-9a87950f0869: port 3(vethedd196d) entered blocking state
[13008.448776] br-9a87950f0869: port 3(vethedd196d) entered forwarding state
[13008.545444] eth1: renamed from vethec13044
[13008.630726] IPv6: ADDRCONF(NETDEV_CHANGE): veth5fd93c4: link becomes ready
[13008.631125] br-9a87950f0869: port 4(veth5fd93c4) entered blocking state
[13008.631130] br-9a87950f0869: port 4(veth5fd93c4) entered forwarding state
[13336.241064] hello: loading out-of-tree module taints kernel.
[13336.241069] hello: module license 'unspecified' taints kernel.
[13336.241072] Disabling lock debugging due to kernel taint
[13336.241185] hello: module verification failed: signature and/or required key
missing - tainting kernel
[13336.244057] Hello World!
```

可观察到刚才载入的 hello 模块。

输入命令 `sudo rmmod hello`，把内核模块 hello 移除内核。

```
[07/28/21]seed@VM:~/.../CXY$ sudo rmmod hello
```

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

创建一个路径不含空格的文件夹，将路径/home/seed/Desktop/Labs\_20.04/Network Security/Firewall Exploration Lab/Labsetup/Files/packet\_filter 下的 seedFilter.c 和 makefile 两个文件拷贝到新创建的文件夹中。

输入命令 `make` 编译内核。

```
[07/28/21]seed@VM:~/.../cxy$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Labs_20.04/cxy
modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Labs_20.04/cxy/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Labs_20.04/cxy/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/Labs_20.04/cxy/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

## Tasks.

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

在未载入内核模块的情况下输入命令 `dig @8.8.8.8 www.example.com.`

```
[07/28/21]seed@VM:~/.../cxy$ 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: 28400
;; 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.                4414    IN      A      93.184.216.34

;; Query time: 104 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Jul 28 04:17:45 EDT 2021
;; MSG SIZE rcvd: 60
```

输入命令 `sudo insmod seedFilter.ko`，将内核模块载入内核。

再次输入命令 `dig @8.8.8.8 www.example.com.`

```
[07/28/21]seed@VM:~/.../cxy$ sudo insmod seedFilter.ko
[07/28/21]seed@VM:~/.../cxy$ 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
```

可观察到连接超时，无法得到应答。

将该模块移除，重复上述命令。

```
[07/28/21]seed@VM:~/.../cxy$ sudo rmmod seedFilter
[07/28/21]seed@VM:~/.../cxy$ 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: 28603
;; 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.                3350    IN      A      93.184.216.34

;; Query time: 104 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Jul 28 04:20:11 EDT 2021
;; MSG SIZE rcvd: 60
```

可以得到应答，说明防火墙起效。

## 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); // Print out the hook info

    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;
    }
    // Print out the IP addresses and protocol
    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_LOCAL_OUT;
    hook1.pf = PF_INET;
    hook1.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook1);

    hook2.hook = printInfo;

```

```

hook2.hooknum = NF_INET_PRE_ROUTING;
hook2.pf = PF_INET;
hook2.priority = NF_IP_PRI_FIRST;
nf_register_net_hook(&init_net, &hook2);

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

hook4.hook = printInfo;
hook4.hooknum = NF_INET_FORWARD;
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);
    nf_unregister_net_hook(&init_net, &hook3);
    nf_unregister_net_hook(&init_net, &hook4);
    nf_unregister_net_hook(&init_net, &hook5);
}

module_init(registerFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");

```



```

Open  [icon] *seedFilter.c ~/Desktop/Labs_20.04/cy Save [icon] [icon] [icon]
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/if_ether.h>
9 #include <linux/inet.h>
10
11 static struct nf_hook_ops hook1, hook2, hook3, hook4, hook5;
12
13 unsigned int printInfo(void *priv, struct sk_buff *skb,
14                        const struct nf_hook_state *state)
15 {
16     struct iphdr *iph;
17     char *hook;
18     char *protocol;
19
20     switch (state->hook){
21         case NF_INET_LOCAL_IN:    hook = "LOCAL_IN";    break;
22         case NF_INET_LOCAL_OUT:   hook = "LOCAL_OUT";   break;
23         case NF_INET_PRE_ROUTING: hook = "PRE_ROUTING";  break;
24         case NF_INET_POST_ROUTING: hook = "POST_ROUTING"; break;
25         case NF_INET_FORWARD:     hook = "FORWARD";     break;
26         default:                  hook = "IMPOSSIBLE";   break;
27     }
28     printk(KERN_INFO "*** %s\n", hook); // Print out the hook info

```

输入命令 make，编译内核。

输入命令 sudo insmod seedFilter.ko，将内核模块载入内核。

在 docker 上输入 dcup，启动 docker。

在 VM 输入命令 ping 10.9.0.5（VM 是 10.9.0.1）。

输入命令 dmesg。

```

[14183.963002] 10.9.0.5 --> 10.9.0.1 (ICMP)
[14183.963009] *** PRE_ROUTING
[14183.963009] 10.9.0.5 --> 10.9.0.1 (ICMP)
[14183.963012] *** LOCAL_IN
[14183.963013] 10.9.0.5 --> 10.9.0.1 (ICMP)
[14184.988078] *** LOCAL_OUT
[14184.988089] 10.9.0.1 --> 10.9.0.5 (ICMP)
[14184.988130] *** POST_ROUTING
[14184.988131] 10.9.0.1 --> 10.9.0.5 (ICMP)
[14184.988364] *** PRE_ROUTING
[14184.988366] 10.9.0.5 --> 10.9.0.1 (ICMP)
[14184.988378] *** PRE_ROUTING
[14184.988379] 10.9.0.5 --> 10.9.0.1 (ICMP)
[14184.988386] *** LOCAL_IN
[14184.988387] 10.9.0.5 --> 10.9.0.1 (ICMP)
[14186.010189] *** LOCAL_OUT

```

可观察到没有 FORWARD 项。

输入命令 sudo rmmod seedFilter，将模块移除。

根据实验结果和查阅到的资料，得出这五个宏的作用分别是：

(1)NF\_INET\_PRE\_ROUTING: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之前被调用。

(2)NF\_INET\_LOCAL\_IN:数据包要进行路由判决，以决定需要被转发还是发往本机。前一种情况下，数据包将前往转发路径;而后一种情况下，数据包将通过这个钩

子点，之后被发送到网络协议栈，并最终被主机接收。

(3)NF\_INET\_FORWARD:需要被转发的数据包会到达这个钩子点。这个钩子点对于实现一个防火墙是十分重要的。

(4)NF\_INET\_LOCAL\_OUT:这是本机产生的数据包到达的第一个钩子点。

(5)NF\_INET\_POST\_ROUTING:需要被转发或者由本机产生的数据包都会经过这个钩子点。源网络地址转换(source network translation, SNAT)就是用这个钩子点实现的。

### 3. Implement two more hooks.

修改 seedFilter.c 文件，代码如下。

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/if_ether.h>
#include <linux/inet.h>
```

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

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

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

    if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23)){
        return NF_DROP;
    }

    else {
        return NF_ACCEPT;
    }
}
```

```

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

    iph = ip_hdr(skb);

    if (iph->protocol == IPPROTO_ICMP){
        return NF_DROP;
    }

    else {
        return NF_ACCEPT;
    }
}

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;
}

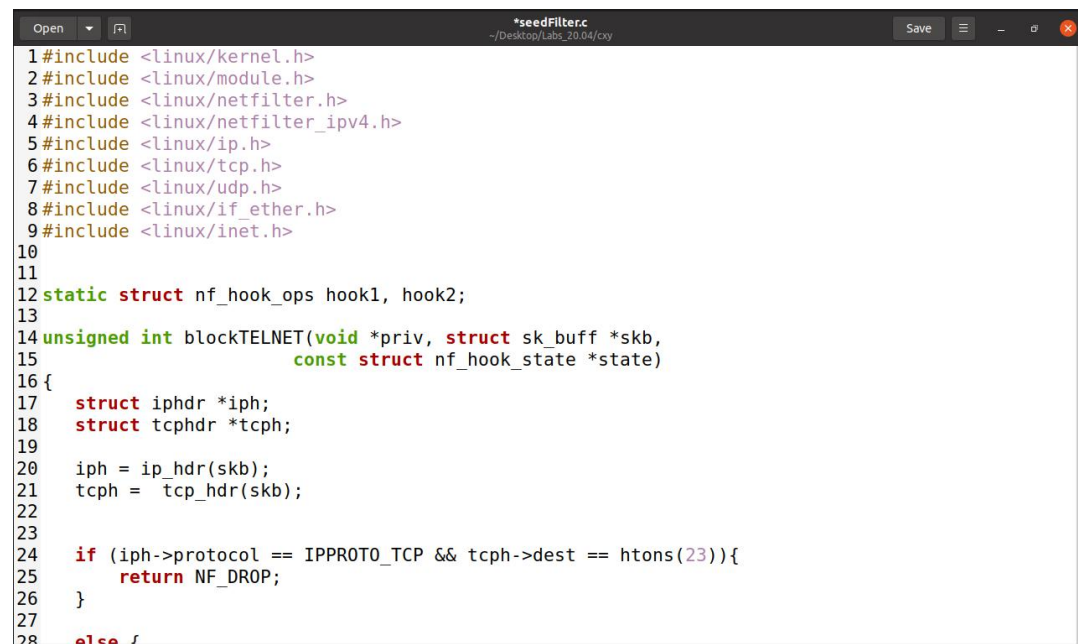
void removeFilter(void) {
    printk(KERN_INFO "The filters are being removed.\n");
    nf_unregister_net_hook(&init_net, &hook1);
    nf_unregister_net_hook(&init_net, &hook2);
}

module_init(registerFilter);
module_exit(removeFilter);

```



```
MODULE_LICENSE("GPL");
```



```
*seedFilter.c
~/Desktop/Labs_20.04/cxy

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/if_ether.h>
9 #include <linux/inet.h>
10
11
12 static struct nf_hook_ops hook1, hook2;
13
14 unsigned int blockTELNET(void *priv, struct sk_buff *skb,
15                          const struct nf_hook_state *state)
16 {
17     struct iphdr *iph;
18     struct tcphdr *tcph;
19
20     iph = ip_hdr(skb);
21     tcph = tcp_hdr(skb);
22
23
24     if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23)){
25         return NF_DROP;
26     }
27
28     else {
```

输入命令 make，编译内核。

输入命令 sudo insmod seedFilter.ko，将内核模块载入内核。

```
[07/28/21]seed@VM:~/.../cxy$ sudo rmmod seedFilter
[07/28/21]seed@VM:~/.../cxy$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Labs_20.04/cxy
modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
CC [M] /home/seed/Desktop/Labs_20.04/cxy/seedFilter.o
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/seed/Desktop/Labs_20.04/cxy/seedFilter.mod.o
LD [M] /home/seed/Desktop/Labs_20.04/cxy/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[07/28/21]seed@VM:~/.../cxy$ sudo insmod seedFilter.ko
```

在用户主机 A 上输入命令 ping 10.9.0.1。

```
[07/28/21]seed@VM:~/.../volumes$ dockcps
ce3774a479ba hostA-10.9.0.5
d48416fe301b host3-192.168.60.7
53b26239b841 seed-router
5cd2495d24dd host2-192.168.60.6
789e24fbf6ec host1-192.168.60.5
[07/28/21]seed@VM:~/.../volumes$ docksh ce
root@ce3774a479ba:/# 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 ---
47 packets transmitted, 0 received, 100% packet loss, time 47100ms
```

可观察到无法 ping 通。

在用户主机 A 上 telnet 10.9.0.1。

```
root@ce3774a479ba:/# telnet 10.9.0.1
Trying 10.9.0.1...
telnet: Unable to connect to remote host: Connection timed out
```

可观察到 telnet 连接失败。  
输入命令 `sudo rmmod seedFilter`，将模块移除。

## Task 2: Experimenting with Stateless Firewall Rules

用户主机 A 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，依次输入如下命令创建过滤规则：

```
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
iptables -P OUTPUT DROP
iptables -P INPUT DROP
```

其中，各项含义为

-A OUTPUT：把此规则加到 OUTPUT 链上

-A INPUT：把该规则加到 INPUT 链上

-p icmp --icmp-type echo-request：该规则只用于 icmp 响应报文

-p icmp --icmp-type echo-reply：该规则只用于 icmp 请求报文

-j ACCEPT：接受满足此规则的包

输入 `iptables -L` 查看过滤规则。

```
root@53b26239b841:/# iptables -L
Chain INPUT (policy DROP)
target     prot opt source                destination            icmp echo-request
ACCEPT     icmp -- anywhere            anywhere              

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy DROP)
target     prot opt source                destination            icmp echo-reply
ACCEPT     icmp -- anywhere            anywhere               
```

在用户主机 10.9.0.5 上 ping 路由器 10.9.0.11。

```
root@ce3774a479ba:/# 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=0.922 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.160 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.368 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.187 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.059 ms
^C
--- 10.9.0.11 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4061ms
rtt min/avg/max/mdev = 0.059/0.339/0.922/0.307 ms
```

可观察到可以 ping 通。

在用户主机 10.9.0.5 上 telnet 远程连接路由器。

```
root@ce3774a479ba:/# telnet 10.9.0.11
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
```

可观察到 telnet 连接失败。

路由器的过滤规则只允许 icmp 请求报文输入和 icmp 响应报文输入, ping 的报文可以进行传输而 telnet 的报文无法进行传输。

在路由器上输入命令 iptables -F, 清空过滤规则。

## Task 2.B: Protecting the Internal Network

在路由器上输入命令 ip addr 查看 interface。

```
root@53b26239b841:/# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
24: eth0@if25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:0b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.11/24 brd 10.9.0.255 scope global eth0
        valid_lft forever preferred_lft forever
28: eth1@if29: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:c0:a8:3c:0b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.60.11/24 brd 192.168.60.255 scope global eth1
        valid_lft forever preferred_lft forever
```

可观察到 10.9.0.0/24 一侧 interface 值为 eth0, 可看到 192.168.60.0/24 一侧 interface 值为 eth1。

在路由器上利用 iptables 命令, 依次输入如下命令创建过滤规则:

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

不允许转发 interface 值为 eth0 一侧的 ICMP 请求报文, 满足外部无法 ping 通内部。

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

允许转发 interface 值为 eth0 一侧的 ICMP 响应报文, 满足外部无法 ping 通内部。

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

允许转发 interface 值为 eth1 一侧的 ICMP 请求报文, 满足外部无法 ping 通内部。

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

允许接受 ICMP 请求报文, 满足外部 ping 通路由器。

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



允许发出 ICMP 回应报文，满足外部 ping 通路由器。

```
iptables -P OUTPUT DROP
```

```
iptables -P INPUT DROP
```

```
iptables -P FORWARD DROP
```

默认设为丢弃，满足其他数据包均阻塞。

输入 iptables -L 查看过滤规则。

```
root@53b26239b841:/# iptables -L
Chain INPUT (policy DROP)
target     prot opt source                destination           icmp echo-request
ACCEPT     icmp -- anywhere              anywhere              icmp echo-request

Chain FORWARD (policy DROP)
target     prot opt source                destination           icmp echo-request
DROP       icmp -- anywhere              anywhere              icmp echo-request
ACCEPT     icmp -- anywhere              anywhere              icmp echo-reply
ACCEPT     icmp -- anywhere              anywhere              icmp echo-request

Chain OUTPUT (policy DROP)
target     prot opt source                destination           icmp echo-reply
ACCEPT     icmp -- anywhere              anywhere              icmp echo-reply
```

在用户主机上 ping 内网主机 192.168.60.5。

```
root@ce3774a479ba:/# 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 ---
20 packets transmitted, 0 received, 100% packet loss, time 19463ms
```

可观察到无法 ping 通。

在用户主机上 ping 路由器。

```
root@ce3774a479ba:/# 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=0.068 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.078 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.403 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.101 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.054 ms
^C
--- 10.9.0.11 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4083ms
rtt min/avg/max/mdev = 0.054/0.140/0.403/0.131 ms
```

可观察到可以 ping 通。

在内网主机 192.168.60.5 上 ping 用户主机。

```

root@789e24fbf6ec:/# 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=0.564 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.161 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.061 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.278 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.456 ms
^C
--- 10.9.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4094ms
rtt min/avg/max/mdev = 0.061/0.304/0.564/0.184 ms

```

可观察到可以 ping 通。

在用户主机上 telnet 远程连接内网主机 192.168.60.5。

```

root@ce3774a479ba:/# telnet 192.168.60.5
Trying 192.168.60.5...
telnet: Unable to connect to remote host: Connection timed out

```

可观察到 telnet 连接失败。

在内网主机 192.168.60.5 上 telnet 远程连接用户主机。

```

root@789e24fbf6ec:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out

```

可观察到 telnet 连接失败。

在路由器上输入命令 iptables -F，清空过滤规则。

## Task 2.C: Protecting Internal Servers

在路由器上利用 iptables 命令，依次输入如下命令创建过滤规则：

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

允许转发 interface 为 eth0 一侧的主机的目的端口为 23、目的地址为 192.168.60.5 的 tcp 报文，满足外部主机只能 telnet 登录 192.168.60.5。

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

允许转发 interface 为 eth1 一侧的 IP 地址为 192.168.60.5 的主机的 tcp 报文，满足外部主机只能 telnet 登录 192.168.60.5。

```
iptables -P FORWARD DROP
```

默认设为丢弃。

输入 iptables -L 查看过滤规则。

```

root@53b26239b841:/# iptables -L
Chain INPUT (policy DROP)
target     prot opt source                destination

Chain FORWARD (policy DROP)
target     prot opt source                destination
ACCEPT     tcp  --  anywhere              192.168.60.5          tcp dpt:telnet
ACCEPT     tcp  --  192.168.60.5          anywhere

Chain OUTPUT (policy DROP)
target     prot opt source                destination

```

在用户主机上 telnet 远程连接网络 192.168.60.5。

```

root@ce3774a479ba:/# telnet 192.168.60.5
Trying 192.168.60.5...
telnet: Unable to connect to remote host: Connection timed out
root@ce3774a479ba:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
789e24fbf6ec login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

```

可观察到 telnet 连接成功。

在用户主机上 telnet 远程连接网络 192.168.60.6。

```

root@ce3774a479ba:/# telnet 192.168.60.6
Trying 192.168.60.6...
telnet: Unable to connect to remote host: Connection timed out

```

可观察到 telnet 连接失败，外部主机只能 telnet 到 192.168.60.5。

在内网主机 192.168.60.5 上 telnet 远程连接内网主机 192.168.60.6。

```

root@789e24fbf6ec:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
5cd2495d24dd login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

```

可观察到 telnet 连接成功，内部主机可以访问内部服务器。

在内网主机 192.168.60.5 上 telnet 远程连接用户主机。

```

root@789e24fbf6ec:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out

```

可观察到 telnet 连接失败，内部主机不能访问外部服务器。

在路由器上输入命令 iptables -F，清空过滤规则。



## Task 3: Connection Tracking and Stateful Firewall

用户主机 A 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

在路由器上利用 iptables 命令，依次输入如下命令创建过滤规则：

```
iptables -F
iptables -P OUTPUT ACCEPT
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
```

在用户主机上 ping 内网主机 192.168.60.5，然后在路由器上输入 conntrack -L 追踪状态。

```
root@53b26239b841:/# conntrack -L
icmp      1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=48 src=192.168.60.5
dst=10.9.0.5 type=0 code=0 id=48 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

在内网主机 192.168.60.5 上输入 nc -lu 9090，在用户主机上输入 nc -u 192.168.60.5 9090。

在用户主机上输入内容，然后在路由器上输入 conntrack -L 追踪状态。

```
root@53b26239b841:/# conntrack -L
udp       17 27 src=10.9.0.5 dst=192.168.60.5 sport=54554 dport=9090 [UNREPLIED]
src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=54554 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

在内网主机上输入内容，然后在路由器上输入 conntrack -L 追踪状态。

```
root@53b26239b841:/# conntrack -L
udp       17 21 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=54554 [UNREPLIED]
src=10.9.0.5 dst=192.168.60.5 sport=54554 dport=9090 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

在内网主机 192.168.60.5 上输入 nc -l 9090，在用户主机上输入 nc 192.168.60.5 9090。

在用户主机上或内网主机上输入内容，然后在路由器上输入 conntrack -L 追踪状态。

```
root@53b26239b841:/# conntrack -L
tcp       6 431995 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=45942 dport=90
90 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=45942 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

在路由器上输入命令 iptables -F，清空过滤规则。

### Task 3.B: Setting Up a Stateful Firewall

在路由器上利用 iptables 命令，依次输入如下命令创建过滤规则：

```
iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport 23 -m conntrack --ctstate
```

```
NEW,ESTABLISHED -j ACCEPT
```

```
iptables -A FORWARD -i eth1 -p tcp -s 192.168.60.5 -m conntrack --ctstate
```

```
ESTABLISHED,RELATED -j ACCEPT
```

以上两步保证了只允许外部 telnet 到 192.168.60.5。

```
iptables -A FORWARD -i eth1 -p tcp -d 10.9.0.5 --dport 23 -m conntrack --ctstate
```

```
ESTABLISHED,RELATED -j ACCEPT
```

```
iptables -A FORWARD -i eth0 -p tcp -s 10.9.0.5 -m conntrack --ctstate
```

```
ESTABLISHED,RELATED -j ACCEPT
```

以上两步保证了允许内部 telnet 到外部所有主机（其实只有一台 10.9.0.5）。

```
iptables -P FORWARD DROP
```

默认设为丢弃。

输入 iptables -L 查看过滤规则。

```
root@53b26239b841:/# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy DROP)
target     prot opt source                destination
ACCEPT     tcp  --  anywhere              host1-192.168.60.5.net-192.168.60.0  tcp
p dpt:telnet ctstate NEW,ESTABLISHED
ACCEPT     tcp  --  host1-192.168.60.5.net-192.168.60.0  anywhere              ct
state RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere              hostA-10.9.0.5.net-10.9.0.0  tcp dpt:te
lnet ctstate RELATED,ESTABLISHED
ACCEPT     tcp  --  hostA-10.9.0.5.net-10.9.0.0  anywhere              ctstate RE
LATED,ESTABLISHED

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

在用户主机上 telnet 远程连接网络 192.168.60.5。

```
root@ce3774a479ba:/# telnet 192.168.60.5
```

```
Trying 192.168.60.5...
```

```
Connected to 192.168.60.5.
```

```
Escape character is '^]'.

```

```
Ubuntu 20.04.1 LTS
```

```
789e24fbf6ec login: seed
```

```
Password:
```

```
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

可观察到 telnet 连接成功。

在用户主机上 telnet 远程连接网络 192.168.60.6。

```
root@ce3774a479ba:/# telnet 192.168.60.6
```

```
Trying 192.168.60.6...
```

```
telnet: Unable to connect to remote host: Connection timed out
```

可观察到 telnet 连接失败。外部主机只能 telnet 访问 192.168.60.5。

在内网主机 192.168.60.5 上 telnet 远程连接内网主机 192.168.60.6。

```
root@789e24fbf6ec:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
5cd2495d24dd login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

可观察到 telnet 连接成功。

在内网主机 192.168.60.5 上 telnet 远程连接用户主机。

```
root@789e24fbf6ec:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
ce3774a479ba login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

可观察到 telnet 连接成功。内部主机可以 telnet 访问外部服务器。

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，输入命令 iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT 创建过滤规则。

在用户主机上 ping 内网主机 192.168.60.5。

```
root@ce3774a479ba:/# 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=0.210 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.149 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.061 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.323 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.223 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.326 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.419 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.878 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.100 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.169 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.560 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.226 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.082 ms
^C
--- 192.168.60.5 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12240ms
rtt min/avg/max/mdev = 0.061/0.286/0.878/0.218 ms
```



可观察到报文正常发送。

在路由器上输入命令 `iptables -A FORWARD -s 10.9.0.5 -j DROP`

再次在用户主机上 ping 内网主机 192.168.60.5。

```
root@ce3774a479ba:/# 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=0.439 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.196 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.094 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.173 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.081 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.075 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.631 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.061 ms
64 bytes from 192.168.60.5: icmp_seq=25 ttl=63 time=0.062 ms
^C
--- 192.168.60.5 ping statistics ---
25 packets transmitted, 9 received, 64% packet loss, time 24554ms
rtt min/avg/max/mdev = 0.061/0.201/0.631/0.189 ms
```

可观察到存在丢包现象。

在实验中可以观察到前 4 个包的回应正常，但是从第 5 个包开始，每 10 秒才能收到一个正常的回应。这是因为设定了每分钟内允许通过的数据包的速率是每 6 秒钟一个；也就是说每过 6 秒钟，第一条命令才会“接管”一次，在这 6s 的空窗期内，根据优先级，将由第二条命令“接管”，这时就会丢弃掉来自 10.9.0.5 的包。

## 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，依次输入如下命令创建过滤规则：

```
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
```

```
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
```

```
iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT --to-destination
192.168.60.7:8080
```

在内网主机 192.168.60.5 上输入命令 `nc -luk 8080`。

在内网主机 192.168.60.6 上输入命令 `nc -luk 8080`。

在内网主机 192.168.60.7 上输入命令 `nc -luk 8080`。

在用户主机 10.9.0.5 上输入命令。重复三次。

```
root@ce3774a479ba:/# echo hello | nc -u 10.9.0.11 8080
^C
root@ce3774a479ba:/# echo hello | nc -u 10.9.0.11 8080
^C
root@ce3774a479ba:/# echo hello | nc -u 10.9.0.11 8080
^C
```

可观察到 192.168.60.5、192.168.60.6、192.168.60.7 依次按序收到 “hello”。

```
root@789e24fbf6ec:/# nc -luk 8080
hello
■

root@5cd2495d24dd:/# nc -luk 8080
hello
■

root@d48416fe301b:/# nc -luk 8080
hello
■
```

### Using the random mode.

在路由器上输入命令 iptables -F，清空过滤规则。

在路由器上利用 iptables 命令，依次输入如下命令创建过滤规则以达到负载均衡的效果：

```
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
```

以 0.33 的概率将报文发送到 192.168.60.5:8080 端口。

```
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
```

剩下的报文将以 0.5 的概率发送到 192.168.60.6:8080 端口。

```
iptables -t nat -A PREROUTING -p udp --dport 8080 -j DNAT --to-destination
192.168.60.7:8080
```

剩下的所有报文发送到 192.168.60.7:8080 端口。

在用户主机上反复输入命令 echo hello | nc -u 10.9.0.11 8080。

在 192.168.60.5:8080 端口发送的数据包：

```
root@789e24fbf6ec:/# nc -luk 8080
hello
■
```

在 192.168.60.5:8080 端口发送的数据包:

```
root@5cd2495d24dd:/# nc -luk 8080
hello
hello
hello
hello
```

在 192.168.60.5:8080 端口发送的数据包:

```
root@d48416fe301b:/# nc -luk 8080
hello
hello
hello
hello
hello
hello
hello
hello
hello
■
```

可观察到 hello 数量不同。