实验 6

57118109 徐一鸣

Task 1: Implementing a Simple Firewall

Task 1.A: Implement a Simple Kernel Module

LKM 允许我们在运行时向内核添加新模块。这个新模块使我们能够扩展内核的功能，而无需重新构建内核甚至重新启动计算机。防火墙的包过滤部分可以实现为 LKM。在这个任务中，我们将熟悉 LKM。

在本任务中，用户主机的 IP 地址为 10.9.0.5，攻击者主机的 IP 地址为 10.9.0.1，内网主机的 IP 地址为 192.168.60.5。

请在您的 VM 上编译这个简单的内核模块，并在 VM 上运行它，对于这个任务，我们将不使用容器，请在实验报告上显示你的跑步结果。

具体步骤如下所示：

- 首先进行环境配置：

```
[07/25/21]seed@VM:~/.../Labsetup$ dockps
67eb2d986df5  seed-router
4a63855706cb  host3-192.168.60.7
b178038d9d7b  host1-192.168.60.5
6fd5893f82aa  hostA-10.9.0.5
76fd4cb1039e  host2-192.168.60.6
```

- 将 kernel_module 文件夹移动到没有空格的文件夹目录下，利用 make 命令编译内核模块：

```
[07/25/21]seed@VM:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Labs_20.04/kern
el_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Labs_20.04/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/seed/Desktop/Labs_20.04/kern
el_module/hello.o
see include/linux/module.h for more information
  CC [M]  /home/seed/Desktop/Labs_20.04/kernel_module/hello.mod.o
  LD [M]  /home/seed/Desktop/Labs_20.04/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

- 利用 insmod 和 rmmod 命令可以将内核模块可进行 hello 的插入和移除：

```
[07/25/21]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[07/25/21]seed@VM:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
[07/25/21]seed@VM:~/.../kernel_module$ sudo rmmod hello
[07/25/21]seed@VM:~/.../kernel_module$ lsmod | grep hello
[07/25/21]seed@VM:~/.../kernel_module$ 
```

- 利用 dmesg 命令可以查看/var/log/syslog 文件中的信息：

```
[07/25/21]seed@VM:~/.../kernel_module$ dmesg | grep World
[  993.049195] Hello World!
[ 1031.030047] Bye-bye World!.
```

- 利用 modinfo 可以查看模块相关信息：

```
[07/25/21]seed@VM:~/.../kernel_module$ modinfo hello.ko
filename:       /home/seed/Desktop/Labs_20.04/kernel_module/hello.ko
srcversion:     75A5408065DE2CED836C338
depends:
retpoline:      Y
name:           hello
vermagic:       5.4.0-54-generic SMP mod_unload
```

Task 1.B: Implement a Simple Firewall Using Netfilter

在本任务中，需要使用 LKM 和 Netfilter 实现一个包过滤模块。该模块将从数据结构中获取防火墙策略，并使用这些策略来决定是否应该阻止数据包。我们希望学生关注过滤部分，防火墙的核心，所以学生被允许硬编码防火墙政策的程序。

具体步骤如下所示：

（1）Compile the sample code using the provided Makefile

● 使用 dig 命令，观察在正常情况下得到的信息：

```
; <<>> 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: 57953
;; 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.        18933   IN      A       93.184.216.34

;; Query time: 51 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu Jul 22 02:43:05 EDT 2021
;; MSG SIZE  rcvd: 60
```

● 使用 make 命令编译内核模块：

```
[07/25/21]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Labs_20.04/pack
et_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Labs_20.04/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Labs_20.04/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/Labs_20.04/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

● 使用 insmod 命令插入内核模块：

```
[07/25/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
```

● 此时无法连接 8.8.8.8：

```
[07/25/21]seed@VM:~$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

（2）Hook the printInfo function to all of the netfilter hooks

● 修改 seedFilter.c 文件：

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

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

unsigned int printInfo (void *priv , struct sk_buff *skb , const struct nf_hook_state *state )
{
	struct iphdr *iph ;
	char *hook ;
	char * protocol ;
	switch ( state -> hook ){
		case NF_INET_LOCAL_IN: hook = "LOCAL_IN"; break;
		case NF_INET_LOCAL_OUT : hook = "LOCAL_OUT"; break;
		case NF_INET_PRE_ROUTING : hook = "PRE_ROUTING"; break;
		case NF_INET_POST_ROUTING : hook = "POST_ROUTING"; break;
		case NF_INET_FORWARD : hook = "FORWARD"; break;
		default : hook = " IMPOSSIBLE "; break;
	}
	printk ( KERN_INFO " *** %s \n ", hook );
	iph = ip_hdr(skb);
	switch ( iph -> protocol ){
		case IPPROTO_UDP : protocol = "UDP"; break;
		case IPPROTO_TCP : protocol = "TCP"; break;
		case IPPROTO_ICMP : protocol = "ICMP "; break;
		default : protocol = "OTHER"; break;
	}
	printk ( KERN_INFO " %pI4 --> %pI4 (%s)\n ", &( iph->saddr), &( iph->daddr ),
protocol );
	return NF_ACCEPT ;
}
int registerFilter(void) {
	printk(KERN_INFO "Registering filters.\n");
	hook1.hook = printInfo;
	hook1.hooknum = NF_INET_PRE_ROUTING;
```

```
        hook1.pf = PF_INET;
        hook1.priority = NF_IP_PRI_FIRST;
        nf_register_net_hook (&init_net, &hook1 );
        hook2.hook = printInfo;
        hook2.hooknum = NF_INET_LOCAL_IN;
        hook2. pf = PF_INET;
        hook2 . priority = NF_IP_PRI_FIRST;
        nf_register_net_hook (& init_net, & hook2 );
        hook3 . hook = printInfo;
        hook3 . hooknum = NF_INET_FORWARD;
        hook3 . pf = PF_INET;
        hook3 . priority = NF_IP_PRI_FIRST;
        nf_register_net_hook (& init_net, & hook3 );
        hook4 . hook = printInfo;
        hook4 . hooknum = NF_INET_LOCAL_OUT;
        hook4 . pf = PF_INET;
        hook4 . priority = NF_IP_PRI_FIRST;
        nf_register_net_hook (& init_net, & hook4 );
        hook5 . hook = printInfo;
        hook5 . hooknum = NF_INET_POST_ROUTING;
        hook5 . pf = PF_INET;
        hook5 . priority = NF_IP_PRI_FIRST;
        nf_register_net_hook (& init_net, & hook5 );
        return 0;
}
void removeFilter (void) {
        printk ( KERN_INFO " The filters are being removed.\n " );
        nf_unregister_net_hook(&init_net , &hook1 );
        nf_unregister_net_hook(&init_net , &hook2 );
}
module_init (registerFilter);
module_exit (removeFilter);
MODULE_LICENSE ("GPL");
```

- 使用 make 命令编译内核模块：

```
[07/25/21]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Labs_20.04/pack
et_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Labs_20.04/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Labs_20.04/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/Labs_20.04/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

- 利用 insmod 命令插入内核模块：

```
[07/25/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[07/25/21]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
```

● 在用户主机上 ping 内网主机，发现可以 ping 通：

```
root@6fd5893f82aa:/# 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=0.237 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.088 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.082 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.081 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.084 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.159 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.090 ms
```

● 使用 dmesg 命令查看/var/log/syslog 文件中的信息：

```
[ 8573.725616]  10.9.0.5 --> 192.168.60.5 (ICMP )

[ 8573.725630]  *** FORWARD

[ 8573.725633]  10.9.0.5 --> 192.168.60.5 (ICMP )

[ 8573.725639]  *** POST_ROUTING

[ 8573.725641]  10.9.0.5 --> 192.168.60.5 (ICMP )

[ 8573.725671]  *** PRE_ROUTING

[ 8573.725673]  10.9.0.5 --> 192.168.60.5 (ICMP )

[ 8573.725678]  *** FORWARD

[ 8573.725681]  10.9.0.5 --> 192.168.60.5 (ICMP )

[ 8573.725685]  *** POST_ROUTING

[ 8573.725687]  10.9.0.5 --> 192.168.60.5 (ICMP )

[ 8573.725724]  *** PRE_ROUTING
```

● 在用户主机上 ping 攻击者主机，发现可以 ping 通：

```
[07/26/21]seed@VM:~/.../Labsetup$ docksh 6f
root@6fd5893f82aa:/# 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=0.383 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.089 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.088 ms
64 bytes from 10.9.0.1: icmp_seq=4 ttl=64 time=0.142 ms
64 bytes from 10.9.0.1: icmp_seq=5 ttl=64 time=0.089 ms
64 bytes from 10.9.0.1: icmp_seq=6 ttl=64 time=0.112 ms
64 bytes from 10.9.0.1: icmp_seq=7 ttl=64 time=0.088 ms
64 bytes from 10.9.0.1: icmp_seq=8 ttl=64 time=0.154 ms
64 bytes from 10.9.0.1: icmp_seq=9 ttl=64 time=0.129 ms
64 bytes from 10.9.0.1: icmp_seq=10 ttl=64 time=0.161 ms
64 bytes from 10.9.0.1: icmp_seq=11 ttl=64 time=0.102 ms
```

● 再次使用 dmesg 命令查看/var/log/syslog 文件中的信息。根据数据发现 NF_IP_PRE_ROUTING 在数据包刚进入主机进行处理的时候调用，NF_IP_LOCAL_IN 在确认数据包的目的地址为本机的时候调用，NF_IP_FORWARD 在要数据包通过主机进行转发的时候调用，NF_IP_LOCAL_OUT 在确认数据包的源地址为本机的时候调用，NF_IP_POST_ROUTING 在数据包将离开主机进行处理的时候调用：

```
[27819.418449]  10.9.0.5 --> 10.9.0.1 (ICMP )
[27819.418457]  *** PRE_ROUTING
[27819.418459]  10.9.0.5 --> 10.9.0.1 (ICMP )
[27819.418465]  *** LOCAL_IN
[27819.418467]  10.9.0.5 --> 10.9.0.1 (ICMP )
[27819.418477]  *** LOCAL_OUT
[27819.418478]  10.9.0.1 --> 10.9.0.5 (ICMP )
[27819.418481]  *** POST_ROUTING
[27819.418482]  10.9.0.1 --> 10.9.0.5 (ICMP )
[27820.393646]  *** PRE_ROUTING
```

（3）Implement two more hooks to achieve the following

- 修改 seedFilter.c 文件：

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


static struct nf_hook_ops hook1, hook2;

unsigned int ICMPFilter ( void *priv , struct sk_buff *skb , const struct nf_hook_state *state )
{
        struct iphdr *iph;
        iph = ip_hdr ( skb );
        if ( iph->protocol == IPPROTO_ICMP ) {
                printk ( KERN_INFO " Dropping ICMP packet:%pI4\n ", &( iph->saddr ));
                return NF_DROP;
                }
        return NF_ACCEPT ;
}
unsigned int telnetFilter ( void *priv , struct sk_buff *skb , const struct nf_hook_state *state )
{
        struct iphdr *iph;
        struct tcphdr *tcph;
```

```
        iph = ip_hdr(skb);
        tcph = ( void *) iph + iph -> ihl * 4;
        if ( iph -> protocol == IPPROTO_TCP && tcph -> dest == htons (23)) {
                printk ( KERN_INFO " Dropping telnet packet:%pI4\n ", &( iph -> saddr ));
                return NF_DROP ;
        }
        return NF_ACCEPT ;
}
int registerFilter (void) {
        printk ( KERN_INFO " Registering filters.\n" );
        hook1.hook = ICMPFilter;
        hook1.hooknum = NF_INET_LOCAL_IN;
        hook1.pf = PF_INET;
        hook1.priority = NF_IP_PRI_FIRST;
        nf_register_net_hook (&init_net , &hook1 );
        hook2.hook = telnetFilter;
        hook2.hooknum = NF_INET_LOCAL_IN;
        hook2.pf = PF_INET;
        hook2.priority = NF_IP_PRI_FIRST;
        nf_register_net_hook (&init_net , &hook2 );
        return 0;
}
void removeFilter ( void ) {
        printk ( KERN_INFO " The filters are being removed. \n " );
        nf_unregister_net_hook (&init_net , &hook1 );
        nf_unregister_net_hook (&init_net , &hook2 );
}
module_init ( registerFilter );
module_exit ( removeFilter );
MODULE_LICENSE ("GPL");
```

- 使用 make 编译内核模块，并且使用 insmod 命令插入内核模块：

```
[07/26/21]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Desktop/Labs_20.04/pack
et_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Desktop/Labs_20.04/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Desktop/Labs_20.04/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Desktop/Labs_20.04/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[07/26/21]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
insmod: ERROR: could not insert module seedFilter.ko: File exists
[07/26/21]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
```

- 在用户主机上 ping 攻击者主机，发现 ping 不通：

```
root@6fd5893f82aa:/# 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 ---
24 packets transmitted, 0 received, 100% packet loss, time 23547 ms
```

- 使用 dmesg 查看/var/log/syslog 文件中的信息，发现 ICMP 报文被丢弃：

```
[ 4137.073754] *** Dropping 10.9.0.1 (ICMP)
[ 4138.096985] *** Dropping 10.9.0.1 (ICMP)
[ 4139.120823] *** Dropping 10.9.0.1 (ICMP)
[ 4140.146303] *** Dropping 10.9.0.1 (ICMP)
[ 4141.172263] *** Dropping 10.9.0.1 (ICMP)
```

Task 2.A:Protecting the Router

在前面的任务中，我们有机会使用 netfilter 构建一个简单的防火墙。实际上，Linux 已经有一个内置的防火墙，也是基于 netfilter。这个防火墙称为 iptables。

从技术上讲,防火墙的内核部分实现称为 Xtables，而 iptables 是一个用于配置防火墙的用户空间程序。然而，iptables 通常用于指内核部分实现和用户空间程序。

Task 2.A:Protecting the Router

在这个任务中，我们将设置一些规则来防止外部机器访问路由器机器，除了 ping。请在路由器容器上执行以下 iptables 命令，然后尝试从 10.9.0.5 访问它。你能 ping 通路由器吗?你能 telnet 到路由器吗？请报告你的观察并解释每条规则的目的。

具体步骤如下所示：

- 在 10.9.0.5 上 ping 路由器，发现可以 ping 通：

```
[07/30/21]seed@VM:~/.../Labsetup$ docksh 6f
root@6fd5893f82aa:/# 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=0.118 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.081 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.092 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.642 ms
64 bytes from 10.9.0.11: icmp_seq=6 ttl=64 time=0.078 ms
64 bytes from 10.9.0.11: icmp_seq=7 ttl=64 time=0.079 ms
64 bytes from 10.9.0.11: icmp_seq=8 ttl=64 time=0.092 ms
64 bytes from 10.9.0.11: icmp_seq=9 ttl=64 time=0.075 ms
64 bytes from 10.9.0.11: icmp_seq=10 ttl=64 time=0.073 ms
64 bytes from 10.9.0.11: icmp_seq=11 ttl=64 time=0.104 ms
64 bytes from 10.9.0.11: icmp_seq=12 ttl=64 time=0.074 ms
64 bytes from 10.9.0.11: icmp_seq=13 ttl=64 time=0.077 ms
64 bytes from 10.9.0.11: icmp_seq=14 ttl=64 time=0.072 ms
64 bytes from 10.9.0.11: icmp_seq=15 ttl=64 time=0.087 ms
```

- 使用 iptables 命令，创建过滤规则如下：

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

- 在 10.9.0.5 上 telnet 路由器，发现无法建立连接，说明规则正确：

```
root@6fd5893f82aa:/# telnet 10.9.0.11
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
```

Task 2.B:Protecting the Internal Network

在你的实验报告中，请包括你的规则和屏幕截图，以证明你的防火墙按照预期工作。当你完成了这个任务，请记得清理桌子或重新启动容器，然后继续下一

个任务。

具体步骤如下所示：

- 编写 iptables 规则程序：

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

#iptables -A FORWARD -p icmp --icmp-type echo-request -j ACCEPT -i eth0 -o eth1
iptables -A FORWARD -p icmp --icmp-type echo-reply -j ACCEPT -i eth0 -o eth1
iptables -A FORWARD -p icmp --icmp-type echo-request -j ACCEPT -i eth1 -o eth0
#iptables -A FORWARD -p icmp --icmp-type echo-reply -j ACCEPT -i eth1 -o eth0

iptables -P OUTPUT DROP
iptables -P INPUT DROP
iptables -P FORWARD DROP
```

- 在 router 上运行，并在外部 ping 192.168.60.5,发现无法 ping 通：

```
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C^C
--- 192.168.60.5 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6134ms
```

- 在外部 ping 路由器，发现可以 ping 通：

```
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=0.046 ms
```

- 在内部 ping 外部，发现可以 ping 通：

```
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=0.063 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.056 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.060 ms
```

- 使用 telnet 重复上述操作，发现无法 telnet 成功：

```
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out

Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
```

Task 2.C:Protecting Internal Servers

在这个任务中，我们想要保护内部网络(192.168.60.0/24)内的 TCP 服务器。

具体步骤如下所示：

- 编写相关的 iptables 规则：

```
iptables -A FORWARD -p tcp --dport 23 -j ACCEPT -d 192.168.60.5
iptables -A FORWARD -p tcp --sport 23 -j ACCEPT -s 192.168.60.5

iptables -P FORWARD DROP
```

- 在 router 上运行后在 10.9.0.5 上 telnet 192.168.60.5，发现可以连通，但无法连通其他服务器：

```
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
0fde30699856 login: ^CConnection closed by foreign host.
root@dbc777c11bd5:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
```

```
Trying 192.168.60.6...
^C
```

● 在内部的 telnet 外部，发现无法连通，但可以连通其他服务器：

```
root@6841249b9287:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
root@6841249b9287:/# telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
d413102eeba5 login:
```

Task 3: Connection Tracking and Stateful Firewall

Task 3.A: Experiment with the Connection Tracking

为了支持有状态防火墙，我们需要能够跟踪连接。这是通过内核中的 conntrack 机制实现的。在本任务中，我们将进行与此模块相关的实验，并熟悉连接跟踪机制。在我们的实验中，我们将检查路由器容器上的连接跟踪信息。

具体步骤如下所示：

- 在 10.6.0.5 上 ping 192.168.60.5，发现可以 ping 通：

```
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=16 ttl=63 time=0.129 ms
64 bytes from 192.168.60.5: icmp_seq=17 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=18 ttl=63 time=0.074 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=20 ttl=63 time=0.057 ms
64 bytes from 192.168.60.5: icmp_seq=21 ttl=63 time=0.060 ms
64 bytes from 192.168.60.5: icmp_seq=22 ttl=63 time=0.057 ms
64 bytes from 192.168.60.5: icmp_seq=23 ttl=63 time=0.058 ms
```

- 观测相应的记录项，发现存活时间为 30s：

```
root@699033525cb1:/volumes# conntrack -L
icmp     1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=80 src=192.168.60.5
 dst=10.9.0.5 type=0 code=0 id=80 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

- 使用 udp 连接，发现存活时间同样为 30s：

```
root@699033525cb1:/volumes# conntrack -L
udp      17 22 src=10.9.0.5 dst=192.168.60.5 sport=55340 dport=9090 [UNREPLIED]
src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=55340 mark=0 use=1
tcp      6 25 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=49200 dport=9090 src
=192.168.60.5 dst=10.9.0.5 sport=9090 dport=49200 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
```

- 使用 tcp 连接 track 到相应的表项，发现在连接建立状态下，存活时间很长：

```
root@699033525cb1:/volumes# conntrack -L
tcp      6 431986 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=49200 dport=90
90 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=49200 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

- 连接切断后进入等待状态，发现等待时间为 120s：

```
tcp      6 115 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=49200 dport=9090 s
c=192.168.60.5 dst=10.9.0.5 sport=9090 dport=49200 [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

请重写 Task2.C 中的防火墙规则，但这次，我们将添加一个允许内部主机访问任何外部服务器的规则（Task2.C 中不允许这样做）。在使用连接跟踪机制编写规则之后，请考虑如何在不使用连接跟踪机制的情况下执行（您不需要实际实现它们）。基于这两组规则，比较这两种不同的方法，并解释每种方法的优缺点。完成此任务后，请记住清除所有规则。

具体步骤如下所示：

- 使用 conntrack 编写 iptables 相关规则，并在 router 上运行：

```
iptables -A FORWARD -p tcp --dport 23 -j ACCEPT -d 192.168.60.5
iptables -A FORWARD -p tcp --sport 23 -j ACCEPT -s 192.168.60.5

iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -p tcp -m conntrack --ctstate NEW -i eth1 -j ACCEPT


iptables -P FORWARD DROP
```

- 在内部 192.168.60.7 上 telnet 外网，发现可以连接：

```
root@0fde30699856:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
dbc777c11bd5 login: █
```

- 在外网 tennet 内网服务，如 192.168.60.7，发现无法连接：

```
root@dbc777c11bd5:/# telnet 192.168.60.7
Trying 192.168.60.7...
telnet: Unable to connect to remote host: Connection timed out
```

Task 4: Limiting Network Traffic

请在路由器上运行以下命令，然后从 10.9.0.5 ping 192.168.60.5。描述你的观察结果。请使用第二条规则和不使用第二条规则进行实验，然后解释是否需要第二条规则，以及为什么需要第二条规则。

具体步骤如下所示：

- 编写相关规则并在 router 上运行：

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

- ping 192.168.60.5，发现除了前五个 burst 限制的报文，其余报文的接受速度基本上是每分钟 10 个：

```
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=0.790 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.058 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.058 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.144 ms
64 bytes from 192.168.60.5: icmp_seq=25 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=31 ttl=63 time=0.060 ms
64 bytes from 192.168.60.5: icmp_seq=37 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=43 ttl=63 time=0.060 ms
64 bytes from 192.168.60.5: icmp_seq=48 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=54 ttl=63 time=0.060 ms
^C
--- 192.168.60.5 ping statistics ---
```

- 注释掉第二条规则，再次运行 ping，发现报文并没有受到规则的限制。因为没有最后的 DROP 规则，没有在第一条规则下通过的报文会直接通过最底层的 ACCEPT 规则通过：

```
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=0.064 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.060 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.057 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.067 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.077 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.058 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.057 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.060 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.063 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.067 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.056 ms
64 bytes from 192.168.60.5: icmp_seq=14 ttl=63 time=0.059 ms
64 bytes from 192.168.60.5: icmp_seq=15 ttl=63 time=0.059 ms
^C
--- 192.168.60.5 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14332ms
rtt min/avg/max/mdev = 0.056/0.061/0.077/0.005 ms
```

Task 5: Load Balancing

请使用此模式实现负载平衡规则，以便每个内部服务器获得大致相同的通信量（可能不完全相同，但当数据包总数较大时应接近）。请解释一下规则。

具体步骤如下所示：

- 编写相关规则并在 router 上运行：

```
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
```

- 针对负载均衡的要求更改规则，并在 router 运行：

```
iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --ever
y 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 --ever
y 3 --packet 1 -j DNAT --to-destination 192.168.60.6:8080
iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --ever
y 3 --packet 2 -j DNAT --to-destination 192.168.60.7:8080
```

- 发现在不停的发送 hello 时，三个服务器接受的报文是被负载均衡过的：

```
seed@VM: ~/.../Labsetup
root@dbc777c11bd5:/# echo hello | nc -u 10.9.0.11 8080
^C
root@dbc777c11bd5:/# echo hello | nc -u 10.9.0.11 8080
^C
root@dbc777c11bd5:/# echo hello | nc -u 10.9.0.11 8080
^C
root@dbc777c11bd5:/# echo hello | nc -u 10.9.0.11 8080
^C
root@dbc777c11bd5:/# echo hello | nc -u 10.9.0.11 8080
root@dbc777c11bd5:/# echo hello | nc -u 10.9.0.11 8080
^C
root@dbc777c11bd5:/# echo hello | nc -u 10.9.0.11 8080
^C
root@dbc777c11bd5:/# echo hello | nc -u 10.9.0.11 8080
^C
root@dbc777c11bd5:/# echo hello | nc -u 10.9.0.11 8080
^C
root@dbc777c11bd5:/# echo hello | nc -u 10.9.0.11 8080
^C
root@dbc777c11bd5:/# echo hello | nc -u 10.9.0.11 8080
^C
```

```
seed@VM: ~/.../Labsetup
root@0fde30699856:/# nc -luk 8080
hello
hello
hello
hello
```

```
seed@VM: ~/.../Labsetup
root@6841249b9287:/# nc -luk 8080
hello
hello
hello
```

```
seed@VM: ~/.../Labsetup
root@d413102eeba5:/# nc -luk 8080
hello
hello
hello
```