

Lab 6-Firewall

The setup files from seed labs were downloaded and the docker containers were built and run.

```
[12/16/23]seed@VM:~/.../Labsetup$ dcbuild
HostA uses an image, skipping
Host1 uses an image, skipping
Host2 uses an image, skipping
Host3 uses an image, skipping
Building Router
  Firefox Web Browser
Step 1/2 : FROM handsonsecurity/seed-ubuntu:large
    --> cecb04fbf1dd
Step 2/2 : RUN apt-get update      && apt-get install -y kmod
& apt-get clean
--> Using cache
--> 53fe824bdeac

Successfully built 53fe824bdeac
Successfully tagged seed-router-image:latest

[12/16/23]seed@VM:~/.../Labsetup$ dcup
Creating network "net-192.168.60.0" with the default driver
WARNING: Found orphan containers (M-10.9.0.105, B-10.9.0.6) for thi
s project. If you removed or renamed this service in your compose f
ile, you can run this command with the --remove-orphans flag to cle
an it up.
& Recreating A-10.9.0.5 ...
  Creating host1-192.168.60.5 ...
  Creating seed-router ...
  Creating host2-192.168.60.6 ...
  Creating host3-192.168.60.7 ...

[12/16/23]seed@VM:~/.../Labsetup$ dockps
3104fdcf4945  hostA-10.9.0.5
d47e18261c2f  host3-192.168.60.7
7f1f4779e9ea  host2-192.168.60.6
71d80c1c429e  seed-router
71c523a3225c  host1-192.168.60.5
[12/16/23]seed@VM:~/.../Labsetup$
```

Task 1: Implementing a Simple Firewall

The goal of this lab is to implement a packet-filtering firewall that inspects incoming and outgoing packets, enforcing administrator-defined policies within the Linux kernel. Traditionally, kernel modification and rebuilding were required for such implementations, but modern Linux systems offer alternatives, namely Loadable Kernel Modules (LKMs) and Netfilter.

1.1 implementing a simple kernel module in hello.ko

1. Created Makefile:

Developed a Makefile to compile the LKM, using the provided Makefile template (included in the lab setup files).

```
[12/16/23]seed@VM:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/
Labsetup/Files/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
CC [M]  /home/seed/Downloads/Labsetup/Files/kernel_module/hello.o
Building modules, stage 2.
MODPOST 1 modules
CC [M]  /home/seed/Downloads/Labsetup/Files/kernel_module/hello.m
od.o
LD [M]  /home/seed/Downloads/Labsetup/Files/kernel_module/hello.k
o
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[12/16/23]seed@VM:~/.../kernel_module$
```

After running the make file we can see that the kernel module is built.

Cleared out the messages in kernel module using dmesg

```
[12/16/23]seed@VM:~/.../kernel_module$ sudo dmesg --clear
[12/16/23]seed@VM:~/.../kernel_module$ dmesg
[12/16/23]seed@VM:~/.../kernel_module$ dmesg -k -e
[12/16/23]seed@VM:~/.../kernel_module$ dmesg
[35495..275775] eth0: renamed from veth72c38aa
[35495..290414] IPv6: AODRCONF(NETDEV CHANGE): veth6b9913c: link becomes ready
[35495..290455] br-fa82a6f5841: port 3(veth6b9913c) entered blocking state
[35495..290457] br-fa82a6f5841: port 3(veth6b9913c) entered forwarding state
[35495..290459] eth0: renamed from veth31c5e1
[35495..304844] IPv6: AODRCONF(NETDEV CHANGE): veth6ec1c54: link becomes ready
[35495..304845] br-fa82a6f5841: port 1(veth6ec1c54) entered blocking state
[35495..304849] br-fa82a6f5841: port 1(veth6ec1c54) entered forwarding state
[35495..304872] IPv6: AODRCONF(NETDEV CHANGE): vetha8c75c: link becomes ready
[35495..304883] br-fa82a6f5841: port 2(vetha8c75c) entered blocking state
[35495..304883] br-fa82a6f5841: port 2(vetha8c75c) entered forwarding state
[35495..304890] eth0: renamed from vethfb73b9
[35495..573570] IPv6: AODRCONF(NETDEV CHANGE): vethaa938bf: link becomes ready
[35495..573601] br-066027d9bc5d: port 2(vethaa938bf) entered blocking state
[35495..573608] br-066027d9bc5d: port 2(vethaa938bf) entered forwarding state
[35495..632684] eth0: renamed from veth68f62ee
[35495..632684] IPv6: AODRCONF(NETDEV CHANGE): veth91a9754: link becomes ready
[35495..634411] br-066027d9bc5d: port 1(veth91a9754) entered blocking state
[35495..634413] br-066027d9bc5d: port 1(veth91a9754) entered forwarding state
[35495..673180] eth0: renamed from vethaa29e54
[35495..705466] IPv6: AODRCONF(NETDEV CHANGE): veth14e9994: link becomes ready
[35495..705513] br-fa82a6f5841: port 4(veth14e9994) entered blocking state
[35495..705513] br-fa82a6f5841: port 4(veth14e9994) entered forwarding state
[36025..772031] [36025..772031] Radical guest time change: -3 648 702 071 000ns (GuestNow=1 702 770 624 464 989
00 ns GuestLast=1 702 773 273 167 060 000 ns fSetTimeLastLoop=true )
[36055..993110] [36055..993110] 23..50..54..512092..timesync.yavskClineSyncWorker: Radical guest time change: 3 668 705 369 000ns (GuestNow=1 702 774 313 216 970 6
00 ns GuestLast=1 702 770 644 511 681 000 ns fSetTimeLastLoop=false)
[12/16/23]seed@VM:~/.../kernel_module$ dmesg --clear
dmesg: permission denied
[12/16/23]seed@VM:~/.../kernel_module$ sudo dmesg --clear
[12/16/23]seed@VM:~/.../kernel_module$ dmesg
[12/16/23]seed@VM:~/.../kernel_module$ dmesg -k -e
[12/16/23]seed@VM:~/.../kernel_module$ dmesg
```

Then the module is checked to capture messages by writing them with below command

```
[12/16/23]seed@VM:~/.../kernel_module$ dmesg -k -w
```

2. Compiled and Loaded:

The following commands were used to ensure proper running of kernel module:

1. On running hello.ko the kernel should output hello world message as shown below:

```
video          49152  0
[12/16/23]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[12/16/23]seed@VM:~/.../kernel_module$ dmesg -k -w
[37561..853691] Hello World!
[12/16/23]seed@VM:~/.../kernel_module$ lsmod | grep -i hello
hello           16384  0
[12/16/23]seed@VM:~/.../kernel_module$
```

2. To remove it from the module and then check the messages printed the following commands were used:

```
[12/16/23]seed@VM:~/.../kernel_module$ sudo rmmod hello
[12/16/23]seed@VM:~/.../kernel_module$ dmesg
[37561..853691] Hello World!
[37789..753326] Bye-bye World!.
[12/16/23]seed@VM:~/.../kernel_module$
```

Results:

Upon loading the module, "Hello World!" was printed to the system logs.

After removing the module, "Bye-bye World!" was printed.

Verified module presence with `lsmod | grep hello`.

Viewed detailed module information with `modinfo hello.ko`.

1.2 Implement simple firewall using Netfilter

1. Hooking in Netfilter

```
[12/16/23]seed@VM:~/.../packet_filter$ ping www.example.com
PING www.example.com (93.184.216.34) 56(84) bytes of data.
64 bytes from 93.184.216.34 (93.184.216.34): icmp_seq=1 ttl=49 time
=23.5 ms
64 bytes from 93.184.216.34 (93.184.216.34): icmp_seq=2 ttl=49 time
=6.17 ms
64 bytes from 93.184.216.34 (93.184.216.34): icmp_seq=3 ttl=49 time
=13.8 ms
^C
--- www.example.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2012ms
rtt min/avg/max/mdev = 6.170/14.513/23.523/7.099 ms
[12/16/23]seed@VM:~/.../packet_filter$ [REDACTED]
[12/16/23]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com

; <>> DiG 9.16.1-Ubuntu <>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58291
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
L: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.example.com.           IN      A

;; ANSWER SECTION:
www.example.com.      16090    IN      A      93.184.216.34

;; Query time: 8 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Sat Dec 16 21:18:04 EST 2023
;; MSG SIZE  rcvd: 60
[12/16/23]seed@VM:~/.../packet_filter$ [REDACTED]
```

Able to ping example.com

```
[12/16/23]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/
Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFil
ter.o
Building modules, stage 2.
MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFil
ter.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFil
ter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[12/16/23]seed@VM:~/.../packet_filter$ [REDACTED]
```

Lookout for registering filters indicating the module is built properly

```
[12/16/23] seed@VM:~/.../packet_filter$ ls
Makefile      seedFilter.c    seedFilter.mod.c
modules.order  seedFilter.ko   seedFilter.mod.o
Module.symvers seedFilter.mod seedFilter.o
[12/16/23] seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[12/16/23] seed@VM:~/.../packet_filter$ lsmod | grep -i seed
seedFilter      16384  0
[12/16/23] seed@VM:~/.../packet_filter$ dmesg -k -w
[37561.853691] Hello World!
[37789.753326] Bye-bye World!.
[40589.576298] Hello World!
[40609.523746] Bye-bye World!.
[41392.759077] Registering filters.
[41398.140768] *** LOCAL_OUT
[41398.140864]   10.0.2.15 --> 91.189.91.157 (UDP)
[41408.145093] *** LOCAL_OUT
```

```
[12/16/23] seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com
; <>> DiG 9.16.1-Ubuntu <>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

```
seed@VM: ~/.../LabSetup          seed@VM: ~/.../packet_filter          seed@VM: ~/.../kernel_module
[41637.154349] *** Dropping 8.8.8.8 (UDP), port 53
[41638.234928] *** LOCAL_OUT
[41638.234931]   10.0.2.15 --> 91.189.91.157 (UDP)
[41641.047796] *** LOCAL_OUT
[41641.047801]   127.0.0.1 --> 127.0.0.53 (UDP)
[41641.054600] *** LOCAL_OUT
[41641.054603]   10.0.2.15 --> 128.122.0.71 (UDP)
[41641.069777] *** LOCAL_OUT
[41641.069779]   127.0.0.53 --> 127.0.0.1 (UDP)
[41642.143827] *** LOCAL_OUT
[41642.143830]   10.0.2.15 --> 8.8.8.8 (UDP)
[41642.143848] *** Dropping 8.8.8.8 (UDP), port 53
[41647.148452] *** LOCAL_OUT
[41647.148454]   10.0.2.15 --> 8.8.8.8 (UDP)
[41647.148468] *** Dropping 8.8.8.8 (UDP), port 53
[41648.240844] *** LOCAL_OUT
[41648.240846]   10.0.2.15 --> 91.189.91.157 (UDP)
[41658.256736] *** LOCAL_OUT
[41658.256784]   10.0.2.15 --> 91.189.91.157 (UDP)
```

```
[12/16/23] seed@VM:~/.../packet_filter$ sudo rmmod seedFilter
[12/16/23] seed@VM:~/.../packet_filter$ 
```

```
seed@VM: ~/.../LabSetup          seed@VM: ~/.../packet_filter          seed@VM: ~/.../kernel_module
[41678.257979]   10.0.2.15 --> 91.189.91.157 (UDP)
[41688.259106] *** LOCAL_OUT
[41688.259108]   10.0.2.15 --> 91.189.91.157 (UDP)
[41698.264070] *** LOCAL_OUT
[41698.264073]   10.0.2.15 --> 91.189.91.157 (UDP)
[41708.265694] *** LOCAL_OUT
[41708.265696]   10.0.2.15 --> 91.189.91.157 (UDP)
[41718.266395] *** LOCAL_OUT
[41718.266399]   10.0.2.15 --> 91.189.91.157 (UDP)
[41728.295048] *** LOCAL_OUT
[41728.295050]   10.0.2.15 --> 91.189.91.157 (UDP)
[41738.297121] *** LOCAL_OUT
[41738.297123]   10.0.2.15 --> 91.189.91.157 (UDP)
[41748.299722] *** LOCAL_OUT
[41748.299723]   10.0.2.15 --> 91.189.91.157 (UDP)
[41758.302009] *** LOCAL_OUT
[41758.302012]   10.0.2.15 --> 91.189.91.157 (UDP)
[41768.303037] *** LOCAL_OUT
[41768.303039]   10.0.2.15 --> 91.189.91.157 (UDP)
[41771.250530] The filters are being removed.
```

SeedFilters are being removed from the module.

Contents from seedFilter were copied to seedPrint file and worked with the file for next steps.

All hook functions are written from the structure and in the seedFilter file are 5 are added at necessary function parameters and all the hook functions are implemented using make function as shown in the images:

```

78 // NF_INET_PRE_ROUTING
79 hook1.hook = printInfo;
80 hook1.hooknum = NF_INET_PRE_ROUTING;
81 hook1.pf = PF_INET;
82 hook1.priority = NF_IP_PRI_FIRST;
83 nf_register_net_hook(&init_net, &hook1);
84 // NF_INET_LOCAL_IN
85 hook2.hook = printInfo;
86 hook2.hooknum = NF_INET_LOCAL_IN;
87 hook2.pf = PF_INET;
88 hook2.priority = NF_IP_PRI_FIRST;
89 nf_register_net_hook(&init_net, &hook2);
90 // NF_INET_FORWARD
91 hook3.hook = printInfo;
92 hook3.hooknum = NF_INET_FORWARD;
93 hook3.pf = PF_INET;
94 hook3.priority = NF_IP_PRI_FIRST;
95 nf_register_net_hook(&init_net, &hook3);
96 // NF_INET_LOCAL_OUT
97 hook4.hook = printInfo;
98 hook4.hooknum = NF_INET_LOCAL_OUT;
99 hook4.pf = PF_INET;
100 hook4.priority = NF_IP_PRI_FIRST;
101 nf_register_net_hook(&init_net, &hook4);
102 // NF_INET_POST_ROUTING
103 hook5.hook = printInfo;
104 hook5.hooknum = NF_INET_POST_ROUTING;
105 hook5.pf = PF_INET;
106 hook5.priority = NF_IP_PRI_FIRST;
107 nf_register_net_hook(&init_net, &hook5);
108
return v;
}

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);
}

```

```
[12/16/23]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
Building modules, stage 2.
MODPOST 0 modules
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[12/16/23]
```

Modules were built and loaded into the kernel module which can be seen from kernel

```
[12/16/23]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[12/16/23]seed@VM:~/.../packet_filter$ sudo insmod seedPrint.ko
[12/16/23]seed@VM:~/.../packet_filter$ ss
[43930.964474] seedPrint: Registering filters.
[43938.943237] *** LOCAL_OUT
[43938.943241]      10.0.2.15 --> 91.189.91.157 (UDP)
messages:
```

We can cross verify using dig on 8.8.8.8 example.com

```

[44/49/23]seed@VM:~/.../packet_filter$ sudo insmod seedPrint.ko
[12/16/23]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com
; <>> DIG 9.16.1-Ubuntu <>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 48205
;; 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.    18843   IN      A      93.184.216.34
;; Query time: 8 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Sat Dec 16 22:05:22 EST 2023
;; MSG SIZE rcvd: 60
[12/16/23]seed@VM:~/.../packet_filter$ 

```

seed@VM: ~/kernel_module

```

seed@VM: ~/LabSetup          seed@VM: ~/packet_filter          seed@VM: ~/
[44065.044172] 127.0.0.1 --> 127.0.0.1 (UDP)
[44065.044183] *** POST_ROUTING
[44065.044209] 127.0.0.1 --> 127.0.0.1 (UDP)
[44065.044223] *** PRE_ROUTING
[44065.044223] 127.0.0.1 --> 127.0.0.1 (UDP)
[44065.044225] *** LOCAL_IN
[44065.044225] 127.0.0.1 --> 127.0.0.1 (UDP)
[44065.071683] *** LOCAL_OUT
[44065.071685] 10.0.2.15 --> 8.8.8.8 (UDP)
[44065.071697] *** POST_ROUTING
[44065.071697] 10.0.2.15 --> 8.8.8.8 (UDP)
[44065.080281] *** PRE_ROUTING
[44065.080283] 8.8.8.8 --> 10.0.2.15 (UDP)
[44065.080297] *** LOCAL_IN
[44065.080297] 8.8.8.8 --> 10.0.2.15 (UDP)
[44068.975050] *** LOCAL_OUT
[44068.975054] 10.0.2.15 --> 91.189.91.157 (UDP)
[44068.975069] *** POST_ROUTING
[44068.975070] 10.0.2.15 --> 91.189.91.157 (UDP)
[44068.985336] *** PRE_ROUTING
[44068.985338] 91.189.91.157 --> 10.0.2.15 (UDP)

```

2. Once the seedPrint.ko module is built on checking that terminal we see different hooks are invoked at different intervals:

```

[43930.964474] seedPrint: Registering filters.
[43938.943237] *** LOCAL_OUT
[43938.943241] 10.0.2.15 --> 91.189.91.157 (UDP)
[43938.943255] *** POST_ROUTING
[43938.943256] 10.0.2.15 --> 91.189.91.157 (UDP)
[43938.956217] *** PRE_ROUTING
[43938.956220] 91.189.91.157 --> 10.0.2.15 (UDP)
[43938.956235] *** LOCAL_IN
[43938.956236] 91.189.91.157 --> 10.0.2.15 (UDP)
[43948.955099] *** LOCAL_OUT
[43948.955103] 10.0.2.15 --> 91.189.91.157 (UDP)
[43948.955123] *** POST_ROUTING
[43948.955124] 10.0.2.15 --> 91.189.91.157 (UDP)
[43948.968293] *** PRE_ROUTING
[43948.968338] 91.189.91.157 --> 10.0.2.15 (UDP)
[43948.968360] *** LOCAL_IN
[43948.968361] 91.189.91.157 --> 10.0.2.15 (UDP)
[43952.289920] *** LOCAL_OUT
[43952.289926] 10.0.2.15 --> 128.122.0.71 (UDP)
[43952.289949] *** POST_ROUTING
[43952.289950] 10.0.2.15 --> 128.122.0.71 (UDP)
[43952.313820] *** PRE_ROUTING
[43952.314386] 128.122.0.71 --> 10.0.2.15 (UDP)
[43952.314595] *** LOCAL_IN
[43952.314783] 128.122.0.71 --> 10.0.2.15 (UDP)
[43952.441360] *** LOCAL_OUT
[43952.441365] 10.0.2.15 --> 35.224.170.84 (TCP)
[43952.442472] *** POST_ROUTING
[43952.442476] 10.0.2.15 --> 35.224.170.84 (TCP)
[43952.485013] *** PRE_ROUTING
[43952.485016] 35.224.170.84 --> 10.0.2.15 (TCP)

```

On clearly understanding the process we find that:

1. NF_INET_PRE_ROUTING:

Invoked just before routing. It can be used to alter the packet before any routing decisions are made.

2. NF_INET_LOCAL_IN:

Invoked for packets destined to the local machine. It is called after routing, just before the packet is delivered to the local socket.

3. NF_INET_FORWARD:

Invoked for packets that are being forwarded to another destination. It occurs after routing and before the packet is forwarded.

4. NF_INET_LOCAL_OUT:

Invoked for locally generated packets before routing. It allows modification of locally generated packets before they are sent out.

5. NF_INET_POST_ROUTING:

Invoked after the routing decision has been made and before the packet is sent out. It allows modification of the packet before it leaves the system.

3. Ping from hostA to 10.9.0.1 ip addressed to VM

```
[12/16/23]seed@VM:~/.../packet_filter$ docksh hostA-10.9.0.5
root@3104fdcf4945:/# ping 10.9.0.1 -c 2
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=15.5 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.083 ms

--- 10.9.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 0.083/7.813/15.543/7.730 ms
root@3104fdcf4945:/# █
root@3104fdcf4945:/# ping 10.9.0.1 -c 2
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=15.5 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.083 ms

--- 10.9.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 0.083/7.813/15.543/7.730 ms
root@3104fdcf4945:/# telnet 10.9.0.1
Trying 10.9.0.1...
Connected to 10.9.0.1.
Escape character is '^].
Ubuntu 20.04.1 LTS
VM login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

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

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Tue Nov 14 14:30:02 EST 2023 on tty3
[12/16/23]seed@VM:~$ █
```

Now we will make seedBlock.c and start working with the file to check which hook functions are working:

```
//blocking UDP to 8.8.8.8:53
unsigned int blockUDP(void *priv, struct sk_buff *skb,
                      const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct udphdr *udph;

    u16 port = 53;
    char ip[16] = "8.8.8.8";
    u32 ip_addr;

    if (!skb) return NF_ACCEPT;

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

    if (iph->protocol == IPPROTO_UDP) {
        udph = udp_hdr(skb);
        if (iph->daddr == ip_addr && ntohs(udph->dest) == port){
            printk(KERN_WARNING "*** Dropping %pi4 (UDP), port %d\n", &(iph->daddr), port);
            return NF_DROP;
        }
    }
    return NF_ACCEPT;
}
```

```

//blocking ping to vm: 10.9.0.1
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;
}

//blocking telnet to VM: 10.9.0.1:23
unsigned int blockTelnet(void *priv, struct sk_buff *skb,
                        const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcp;

    u16 port = 23;//telnet
    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) {
        tcp = tcp_hdr(skb);
        if (iph->daddr == ip_addr && ntohs(tcp->dest) == port){
            printk(KERN_WARNING "*** Dropping %pI4 (Telnet), port %d\n", &(iph->daddr), port);
            return NF_DROP;
        }
    }
    return NF_ACCEPT;
}

```

As we see each hook function is invoked at mentioned time instances.

```

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 = blockTelent;
    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;
}

```

```

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);
}

```

Now that hook files are built using the make command seedBlock.ko is created which is loaded onto the kernel with the insmod command.

```

[12/16/23]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/Files/seedBlock
s/seedBlock modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/Files/seedBlock.o
Building modules, stage 2.
MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/Files/seedBlock.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/Files/seedBlock.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[12/16/23]seed@VM:~/.../packet_filter$ sudo insmod seedBlock.ko
[12/16/23]seed@VM:~/.../packet_filter$ 
[49200.794055] The filters are being removed.
[49391.016674] SeedBlock: Registering filters.
[49400.401165] *** LOCAL OUT

```

On pinging VM from host -A since it is blocked the ping is dropped as shown below:

The screenshot shows two terminal windows. The top window displays the kernel log with several log entries related to the 'SeedBlock' module, including messages about removing filters, registering filters, and dropping packets. The bottom window shows a ping test from host A to the VM at 10.9.0.1, which is failing due to packet drops.

```

seed@VM: ~/Labsetup [50171.168162] 10.0.2.15 --> 91.189.91.157 (UDP)
[50181.173191] *** LOCAL_OUT
Successfully built 53f
Successfully tagged seed:latest
[12/16/23]seed@VM:~/.../packet_filter$ 
[50184.610202] SeedBlock: The filters are being removed.
[50188.979886] SeedBlock: Registering filters.
Creating network "net-53f4d95c" with 1 interface...
[50191.181816] *** LOCAL_OUT
WADNTNC: Found orphan

seed@VM: ~/.../packet_filter [50231.231438] 10.0.2.15 --> 91.189.91.157 (UDP)
[50231.915041] *** Dropping 10.9.0.1 (ICMP)
host3-192.168.60.7 | [50232.918286] *** Dropping 10.9.0.1 (ICMP)
host2-192.168.60.6 | [50233.943382] *** Dropping 10.9.0.1 (ICMP)
host1-192.168.60.5 | [50234.966316] *** Dropping 10.9.0.1 (ICMP)
seed-router | * Start [50235.991281] *** Dropping 10.9.0.1 (ICMP)
hostA-10.9.0.5 | * St [50237.017351] *** Dropping 10.9.0.1 (ICMP)
[50238.038693] *** Dropping 10.9.0.1 (ICMP)
[50239.063053] *** Dropping 10.9.0.1 (ICMP)

[12/16/23]seed@VM:~/.../packet_filter$ sudo rmmod seedBlock
[12/16/23]seed@VM:~/.../packet_filter$ sudo insmod seedBlock.ko
[12/16/23]seed@VM:~/.../packet_filter$ docksh 31
root@3104fdcf4945:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.

Attaching to host1-192.0.2.15 --> 91.189.91.157 (UDP)
0.7, hostA-10.9.0.5 | [50231.915041] *** Dropping 10.9.0.1 (ICMP)
host3-192.168.60.7 | [50232.918286] *** Dropping 10.9.0.1 (ICMP)
host2-192.168.60.6 | [50233.943382] *** Dropping 10.9.0.1 (ICMP)
host1-192.168.60.5 | [50234.966316] *** Dropping 10.9.0.1 (ICMP)
seed-router | * Start [50235.991281] *** Dropping 10.9.0.1 (ICMP)
hostA-10.9.0.5 | * St [50237.017351] *** Dropping 10.9.0.1 (ICMP)
[50238.038693] *** Dropping 10.9.0.1 (ICMP)
[50239.063053] *** Dropping 10.9.0.1 (ICMP)

```

Task-2

Task 2.1

On pinging router from host A is successful

```
[12/17/23] seed@VM:~$ docksh 31
root@3104fdcf4945:/# ping 10.9.0.11 -c 2
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=25.0 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.089 ms

--- 10.9.0.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 0.089/12.537/24.986/12.448 ms
root@3104fdcf4945:/#
```

Tenet connection host a to router also works

```
root@3104fdcf4945:/# telnet 10.9.0.11
Trying 10.9.0.11...
Connected to 10.9.0.11.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
71d80c1c429e login: seed
Password:
root@71d80c1c429e:~# iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
root@71d80c1c429e:~# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source          destination
ACCEPT    icmp  --  0.0.0.0/0      0.0.0.0/0          icmp type 8
ACCEPT    icmp  --  0.0.0.0/0      0.0.0.0/0          icmp type 8

Chain FORWARD (policy ACCEPT)
target     prot opt source          destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source          destination
ACCEPT    icmp  --  0.0.0.0/0      0.0.0.0/0          icmp type 0
root@71d80c1c429e:~#
root@71d80c1c429e:~# iptables -P OUTPUT DROP
root@71d80c1c429e:~# iptables -P INPUT DROP
root@71d80c1c429e:~# 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          icmp type 8
ACCEPT    icmp  --  0.0.0.0/0      0.0.0.0/0          icmp type 8
ACCEPT    icmp  --  0.0.0.0/0      0.0.0.0/0          icmp type 0

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          icmp type 0
root@71d80c1c429e:~#
```

Now if we ping the router or try a telnet connection it doesn't work.

```
root@3104fdcf4945:/# telnet 10.9.0.11
Trying 10.9.0.11...
```

Task 2.2

The docker containers are restarted before starting next step using docker restart<container id>. After running iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP

```
root@71d80c1c429e:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP
root@71d80c1c429e:/# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source          destination
Chain FORWARD (policy ACCEPT)
target     prot opt source          destination
DROP      icmp --  0.0.0.0/0           0.0.0.0/0          icmptype 8
Chain OUTPUT (policy ACCEPT)
target     prot opt source          destination
root@71d80c1c429e:/# █
```

So we can see now that the ping host A to 10.9.0.11 is working and to 192.168.60.5 is not working as shown below:

```
root@3104fdcf4945:/# 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=8.91 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.105 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.103 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.107 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=0.106 ms
64 bytes from 10.9.0.11: icmp_seq=7 ttl=64 time=0.092 ms
^C
--- 10.9.0.11 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6085ms
rtt min/avg/max/mdev = 0.092/1.360/8.907/3.080 ms
root@3104fdcf4945:/# 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 19456ms
```

Here is the detailed view of iptables after it is set to ACCEPT from VMs both eth0 and eth1 And running iptables -L -n, we can see that it ACCEPTS all three.

```
root@71d80c1c429e:/# iptables -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in     out     source          destination
Chain FORWARD (policy DROP 0 packets, 0 bytes)
pkts bytes target prot opt in     out     source          destination
  20  1680 DROP    icmp   --  eth0    *      0.0.0.0/0      0.0.0.0/0
      0     0 DROP    icmp   --  *      *      0.0.0.0/0      0.0.0.0/0
      0     0 ACCEPT  icmp   --  eth1    *      0.0.0.0/0      0.0.0.0/0
      0     0 ACCEPT  icmp   --  eth0    *      0.0.0.0/0      0.0.0.0/0
      0     0 ACCEPT  icmp   --  *      *      0.0.0.0/0      0.0.0.0/0
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in     out     source          destination
root@71d80c1c429e:/#
```

Task 2.3

running `iptables -A FORWARD -i eth0 -d 192.168.60.5 -p -tcp --dport 23 -j ACCEPT` to make sure all internal hosts run on telnet server listening to port 23 and outside hosts can only have access to server on 192.168.60.5

```
root@71d80c1c429e:/# iptables -A FORWARD -i eth0 -p tcp -d 192.168.60.5 --dport 23 -j ACCEPT
root@71d80c1c429e:/#
```

The command given in the document is run as shown below:

The screenshot shows three terminal windows side-by-side. The left window has tabs for 'Labsetup' and 'packet_filter'. The middle window is titled 'seed@VM: ~/packet_filter' and contains the command: 'root@71d80c1c429e:/# iptables -A FORWARD -i eth0 -p tcp --sport 5000 -j ACCEPT'. The right window has tabs for 'Labsetup' and 'kernel_module'. The bottom of the left window shows some text from a previous command: 'inu... tu.c .can m/ad movi do n the u sy h pr /cop ANTY'.

```
root@71d80c1c429e:/# iptables -A FORWARD -i eth0 -p tcp --sport 5000 -j ACCEPT
```

And we can see that the telnet to VN eth1 is working from Host A

The screenshot shows a single terminal window with the title 'seed@VM: ~'. It displays a successful telnet session to host A. The session starts with 'Trying 192.168.60.5...', followed by a password prompt 'Password:', and a welcome message for Ubuntu 20.04.1 LTS. The session ends with a 'Connection closed by foreign host.' message.

```
root@3104fdcf4945:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^].
Ubuntu 20.04.1 LTS
71c523a3225c 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.
Last login: Sun Dec 17 06:08:36 UTC 2023 from www.SeedLabSQLInjection.com on pts
/1
seed@71c523a3225c:~$ exit
logout
Connection closed by foreign host.
root@3104fdcf4945:/#
```

Sending Hello by using nc -u 192.168.60.5 9090

```
root@d829308fc8df:/# nc -u 192.168.60.5 9090
```

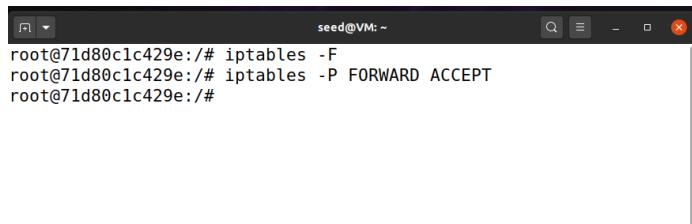
The screenshot shows a terminal window with the command 'nc -u 192.168.60.5 9090' running. The user has typed 'hello' and is pressing the Enter key.

```
hello
```



```
seed@VM: ~
root@71d80c1c429e:/# nc -lu 9090
hello
```

```
root@3104fdcf4945:/# nc -u 192.168.60.5 9090
^C
root@3104fdcf4945:/# nc -ltv 9090
Listening on 0.0.0.0 9090
root@3104fdcf4945:/#
```



```
seed@VM: ~
root@71d80c1c429e:/# iptables -F
root@71d80c1c429e:/# iptables -P FORWARD ACCEPT
root@71d80c1c429e:/#
```

Task 3: Connection Tracking and Stateful Firewall

3.1 ICMP experiment:

Run the following command and check the connection tracking information on the router. Describe your observation. How long is the ICMP connection state be kept?

We will send out ICMP packets to host machine 192.168.60.5 using ping command. Since we need it to continuously send packets. We will run it in the background by adding & character as shown in the picture below.

ICMP Experiment: Execute the provided command and examine the connection tracking details on the router. Provide a description of your observations. Determine the duration for which the ICMP connection state is retained.

To achieve this, initiate the transmission of ICMP packets to the host machine at IP address 192.168.60.5 using the ping command. Given the need for continuous packet transmission, run the command in the background by appending the "&" character, as depicted in the image below.

Subsequently, execute the `conntrack -L` command to monitor the number of active connections.

Then we execute Conntrack -L command to track how many connections are active.

```
root@ala904431131:/# conntrack -L
icmp      1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=65 src=192.168.60.5 dst=10.9.0.5 type
=0 code=0 id=65 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@ala904431131:/#
```

From the above picture we can see that. 1 flow entry is active and it is sending ICMP packets.

Now our task is to end the ping job and observe for how long the Conntrack will hold the status even after termination of the job.

```
root@243fff5590bf:/# conntrack -L
icmp      1 23 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=20 src=192.168.60.5 dst=192.168.60.11 type=0 code=0 id=20 mark=0 use=1
icmp      1 29 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=18 src=192.168.60.5 dst=192.168.60.11 type=0 code=0 id=18 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
[1]+  Terminated                  ping 192.168.60.5 &> /dev/null
root@243fff5590bf:/# conntrack -L
icmp      1 16 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=20 src=192.168.60.5 dst=192.168.60.11 type=0 code=0 id=20 mark=0 use=1
icmp      1 29 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=18 src=192.168.60.5 dst=192.168.60.11 type=0 code=0 id=18 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
root@243fff5590bf:/# conntrack -L
icmp      1 5 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=20 src=192.168.60.5 dst=192.168.60.11 type=0 code=0 id=20 mark=0 use=1
icmp      1 29 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=18 src=192.168.60.5 dst=192.168.60.11 type=0 code=0 id=18 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 2 flow entries have been shown.
root@243fff5590bf:/# conntrack -L
icmp      1 29 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=18 src=192.168.60.5 dst=192.168.60.11 type=0 code=0 id=18 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

In the provided screenshot, we observe that upon executing the `jobs` command, a running ping job is visible in the background. This is also evident when we use the `Conntrack` command.

Executing `kill %1` terminates the ping job, and we promptly run the `Conntrack -L` command multiple times to assess the duration for which the state is retained.

Based on our observation, the state persists for approximately 8 seconds. In the end, it is noticeable that the count of flow streams decreases by one.

UDP experiment: Run the following command and check the connection tracking information on the router. Describe your observation. How long is the UDP connection state be kept?

In order to send out UDP packets. Lets login to host 192.168.60.5 and initiate a Netcat connection.

```
root@ubuntu-seed-labs:~# docker exec -it 833fe3104c4a /bin/bash
root@833fe3104c4a:/# nc -lu 9090
```

Now lets login to external host A on 10.9.0.5 and connect to Netcat server on 192.168.60.5 and try sending some UDP packets. This can be done by typing something on the terminal.

```
root@8681ae8df6c1:/# nc 192.168.60.5 9090
hello
```

Same text will be reflected on the host machine as well.

```
root@657af6b75e9b:/# nc -lu 9090
hello
```

If we execute the Conntrack -L command now, We can observe a flow stream of UDP packets.

Let's end the Netcat server connection and observe Conntrack as to how long the connection state is retained.

By executing the Conntrack -L command again and again, we can get a rough estimate of how long it will store the state.

```
root@0c997aeb4987:/# conntrack -L
tcp      6 431910 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=49576 dport=9090
src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=49576 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@0c997aeb4987:/#
```

Based on the above observation. We can say that Conntrack stores the state for roughly 8 to 10 seconds.

TCP experiment: Run the following command and check the connection tracking information on the router. Describe your observation. How long is the TCP connection state be kept?

To send TCP packets between two hosts. Lets initiate Netcat TCP server on host 192.168.60.5

```
root@657af6b75e9b:/# nc -l 9090
hello
```

Now from host 10.9.0.5 we will connect to the TCP server to send out TCP packets.

```
root@8681ae8df6c1:/# nc 192.168.60.5 9090
hello
```

As seen in the above pictures, The TCP packets have been successfully received by the Server.

Lets Observe Conntrack for the Flow stream information.

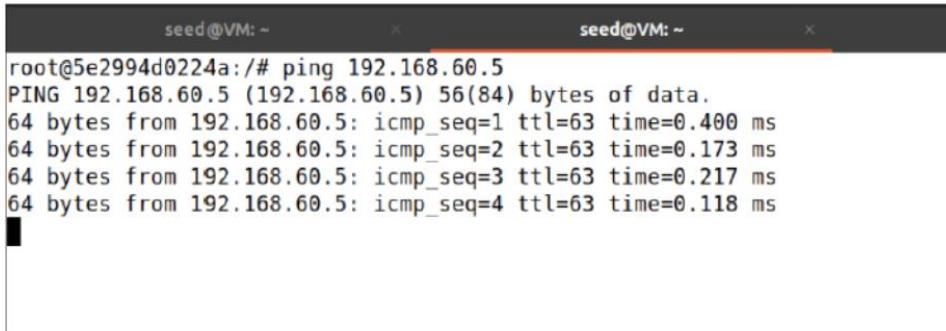
```
11/26/22] seed@VM:-/.../Labsetup$ docksh 8c
root@0c997aeb4987:/# conntrack -L
tcp      6 431910 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=49576 dport=9090
src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=49576 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@0c997aeb4987:/#
```

In the above image, we can see that the status of TCP packets are ESTABLISHED. Meaning the TCP handshake was successful and the connection is formed and retained for data transfer.

To check how long the state will be held by Conntrack, we have to end the TCP connection from both the sides and Observe the Conntrack Results.

ICMP EXPERIMENT:

Ping 192.168.60.5 from 10.9.0.5



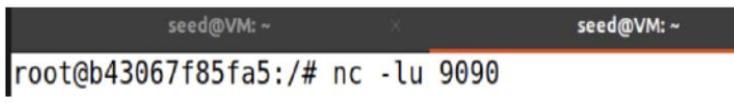
```
seed@VM: ~          seed@VM: ~
root@5e2994d0224a:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.400 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.173 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.217 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.118 ms
```

The ICMP connection status is kept in the connection table for 29 seconds. Because ICMP is connectionless, the state is saved in a shorter timescale, hence we don't see a connection state in the table.

```
root@ala904431131:/# conntrack -L
icmp      1 29 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=65 src=192.168.60.5 dst=10.9.0.5 type
=0 code=0 id=65 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@ala904431131:/#
```

UDP EXPERIMENT:

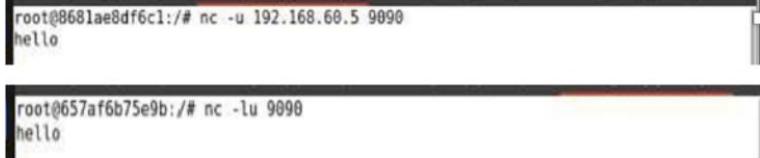
On 192.168.60.5 container I started the netcat server



```
seed@VM: ~          seed@VM: ~
root@b43067f85fa5:/# nc -l -u 9090
```

In 10.9.0.5 container

```
#nc -u 192.168.60.5 9090
```



```
root@8681ae8df6c1:/# nc -u 192.168.60.5 9090
hello
```



```
root@657af6b75e9b:/# nc -l -u 9090
hello
```

Once we ended the TCP connection. We can see the status of TCP packets now changed to TIME_WAIT from ESTABLISHED. It will wait for 113s seconds before discarding the state. As time goes we can observe this timer count getting decreased and finally becoming Zero. that is when the connection state is discarded from Conntrack.

3.2 : Setting Up a Stateful Firewall

A Netcat TCP server is initiated in the External server Host A.

Here in the 192.168.60.5 container can see hello is displayed in the 10.9.0.5 container.

The connection table saves the state of the UDP connection for 33 seconds. Because UDP is a connectionless protocol, the duration in which the state is saved is shorter, and there is no connection state in the table.

TCP EXPERIMENT:

On 192.168.60.5 container I started the netcat TCP server

```
seed@VM: ~/Labsetup          root@a507ff4226b2: /          seed@VM: ~/Labsetup          rc
[11/27/23]seed@VM:~/.../Labsetup$ docksh 38
root@38dd708d0bf5:/# nc -l 9090
```

```
root@8681ae8df6c1:/# nc 192.168.60.5 9090
hello
```

Here in the 192.168.60.5 container, hello is displayed in the 10.9.0.5 container.

```
root@657af6b75e9b:/# nc -l 9090
hello
```

The state of the TCP connection is kept for 431910 seconds. Because TCP is a connection-oriented protocol, we keep the connection state for a longer period of time. As soon as the TCP connection is established, we see the state as ESTABLISHED.

```
root@0c997aeb4987:/# conntrack -L
tcp      6 431910 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=49576 dport=9090
src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=49576 (ASSURED) mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@0c997aeb4987:/#
```

5 Task 3.B: Setting Up a Stateful Firewall

Running the following commands on seed-router

```
seed@VM: ~          seed@VM: ~          seed@VM: ~
root@ala904431131:/# iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport 23 --syn -j ACCEPT
root@ala904431131:/# iptables -A FORWARD -i eth1 -p tcp --syn -j ACCEPT
root@ala904431131:/# iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
root@ala904431131:/# iptables -A FORWARD -p tcp -j DROP
root@ala904431131:/# iptables -P FORWARD ACCEPT
root@ala904431131:/#
```

In the above image, From internal host 1. We are connecting to the Netcat server hosted on external host A.

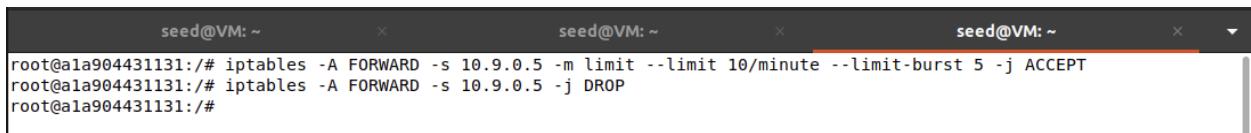
The connection was established successfully and all the input given on the client is reflected on the server.

This completes testing of all the scenarios of the Firewall.

In the end. Lets go ahead and clear all the rules.

Task 4: Limiting Network Traffic

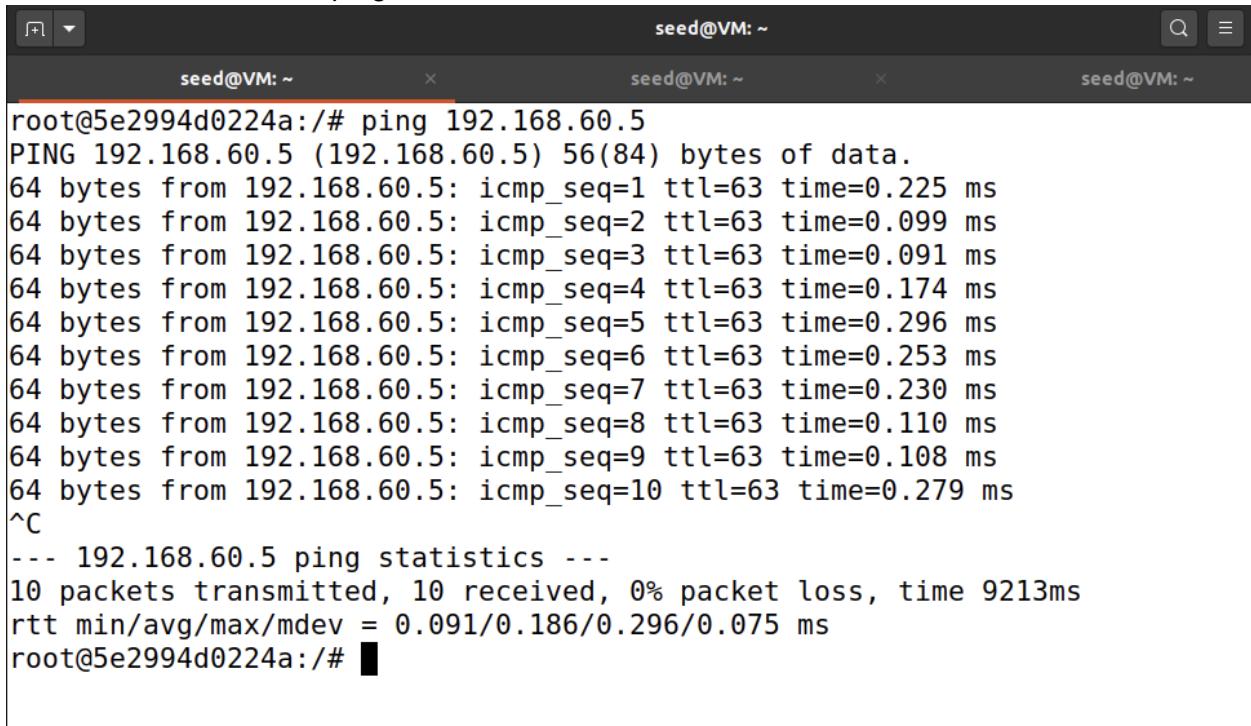
In this task, we will use **limit** module to limit how many packets from 10.9.0.5 are allowed to get into the internal network.



```
seed@VM: ~          seed@VM: ~          seed@VM: ~
root@ala904431131:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
root@ala904431131:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
root@ala904431131:/#
```

The above command restricts traffic at IP 10.9.0.5 and only allows 10 packets per minute. Our task is to find out if this rule is enough on its own to restrict the packet flow.

In order to test that. Lets ping 192.168.60.5 from 10.0.0.5



```
seed@VM: ~          seed@VM: ~          seed@VM: ~          seed@VM: ~          seed@VM: ~
root@5e2994d0224a:/# 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=0.225 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.099 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.091 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.174 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.296 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.253 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.230 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.110 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.108 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.279 ms
^C
--- 192.168.60.5 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9213ms
rtt min/avg/max/mdev = 0.091/0.186/0.296/0.075 ms
root@5e2994d0224a:/#
```

From what we observe from the above image. Every packet is going through. This is because whichever packets did not pass the filter is still going through because we do not have a second

rule to drop such packets which did not pass first filter.

Lets add a second filter to drop packets.

After this lets test it by pinging 192.168.60.5 from 10.0.0.5

```
seed@VM: ~          x      seed@VM: ~          x      seed@VM: ~          x
root@5e2994d0224a:/# 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=0.242 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.227 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.174 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.175 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.437 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.171 ms
^C
--- 192.168.60.5 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5102ms
rtt min/avg/max/mdev = 0.171/0.237/0.437/0.093 ms
root@5e2994d0224a:/#
```

This time we can observe 73.4694 % packet loss. This is because all the packets which didn't go through the first rule are dropped in the second rule. Now the filter is successfully limiting the packet count.

Task-5

Initially the server is started on the three hosts : 192.168.60.5, 192.168.60.6, and 192.168.60.7 and it is set to accept UDP datagrams from multiple hosts.

```
[12/20/23]seed@VM:~$ dockps  
3104fdcf4945 hostA-10.9.0.5  
d47e18261c2f host3-192.168.60.7  
7f1f4779e9ea host2-192.168.60.6  
71d80c1c429e seed-router  
71c523a3225c host1-192.168.60.5  
[12/20/23]seed@VM:~$ docksh 71c  
root@71c523a3225c:/# nc -luk 8080  
[12/20/23]seed@VM:~$ docksh 7f  
root@7f1f4779e9ea:/# nc -luk 8080  
[12/20/23]seed@VM:~$ docksh d4  
root@d47e18261c2f:/# nc -luk 8080
```

Packets from router are sent to host 1

```
[12/20/23]seed@VM:~$ docksh 71d  
root@71d80c1c429e:/# 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  
[12/20/23]seed@VM:~$ docksh 71c  
root@71c523a3225c:/# nc -luk 8080  
hello  
[12/20/23]seed@VM:~$ docksh 7f  
root@7f1f4779e9ea:/# nc -luk 8080
```

```
[12/20/23]seed@VM:~$ docksh d4  
root@d47e18261c2f:/# nc -luk 8080  
[12/20/23]seed@VM:~$ docksh 31  
root@3104fdcf4945:/# echo hello | nc -u 10.9.0.11 8080  
^C  
root@3104fdcf4945:/#
```

WE SEE THAT HELLO IS captured in host 1 and not in any other dockers.

Sending every 3 packets to host 1,2,3.

```
root@71d80clc429e:/# iptables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target    prot opt source          destination
DNAT      udp  --  0.0.0.0/0      0.0.0.0/0      udp dpt:8080 statistic mode nth every 3 to:192.168.60.5:8080
DNAT      udp  --  0.0.0.0/0      0.0.0.0/0      udp dpt:8080 statistic mode nth every 3 packet 2 to:192.168.60.7:808
DNAT      udp  --  0.0.0.0/0      0.0.0.0/0      udp dpt:8080 statistic mode nth every 3 packet 1 to:192.168.60.6:808
DNAT      udp  --  0.0.0.0/0      0.0.0.0/0      udp dpt:8080 statistic mode nth every 3 to:192.168.60.5:8080
```

We can see the policy with 3 DNAT udp packets in the pre routing. Now sending the udp packets from host to router port 8080 as follows:

```
seed@VM: ~
root@71c523a3225c:/# nc -luk 8080
hello1 hello2 hello3 1
```

```
seed@VM: ~
root@7f1f4779e9ea:/# nc -luk 8080
hello1 hello2 hello3 2
```

```
seed@VM: ~
root@d47e18261c2f:/# nc -luk 8080
hello1 hello2 hello3
```

```
seed@VM: ~
root@3104fdcf4945:/# echo "hello1 hello2 hello3" | nc -u 10.9.0.11 8080
^C
root@3104fdcf4945:/# echo "hello1 hello2 hello3 1" | nc -u 10.9.0.11 8080
^C
root@3104fdcf4945:/# echo "hello1 hello2 hello3 2" | nc -u 10.9.0.11 8080
^C
root@3104fdcf4945:/#
```

We can see that after each 2 seconds the packet sent are equally distributed among host 1 2 3.

```
seed@VM: ~
root@71d80clc429e:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.3333 -j DNAT --to-d 192.168.60.5:8080
root@71d80clc429e:/#
root@71d80clc429e:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.5 -j DNAT --to-d 192.168.60.6:8080
root@71d80clc429e:/#
root@71d80clc429e:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 1.00 -j DNAT --to-d 192.168.60.7:8080
root@71d80clc429e:/#
root@71d80clc429e:/# iptables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target    prot opt source          destination
DNAT      udp  --  0.0.0.0/0      0.0.0.0/0      udp dpt:8080 statistic mode random probability 0.333300000006 to:192.168.60.5:8080
DNAT      udp  --  0.0.0.0/0      0.0.0.0/0      udp dpt:8080 statistic mode random probability 0.500000000000 to:192.168.60.6:8080
DNAT      udp  --  0.0.0.0/0      0.0.0.0/0      udp dpt:8080 statistic mode random probability 1.000000000000 to:192.168.60.7:8080
Chain INPUT (policy ACCEPT)
target    prot opt source          destination
```

For random mode, the packets don't run as expected for short packets.

The image displays four terminal windows from a Linux environment, likely Kali Linux, showing the results of network traffic analysis and modification using tools like netcat (nc) and echo.

- Terminal 1:** Shows a sequence of UDP packets sent to port 8080. The payload includes "hello1", "hello2", "hello3", "hello67", and "hello123".
- Terminal 2:** Shows a sequence of UDP packets sent to port 8080. The payload includes "hello1", "hello2", "hello3", "hello34", "123", "5seconds", and "10seconds".
- Terminal 3:** Shows a sequence of UDP packets sent to port 8080. The payload includes "hello1", "hello1", "hello123hgh".
- Terminal 4:** Shows a sequence of UDP packets sent to port 8080. The payload includes "hello1", "hello34", "hello67", "hello123", "hello123hgh", "123", "5seconds", and "10seconds". This terminal also shows the use of echo to generate the payload and nc -u to send UDP packets.

For long packets or over long periods of time the packets behave statistically to the rules.

Conclusion for Task 5

- Round-Robin Load Balancing with nth Mode:

1. The nth mode employs a round-robin strategy, choosing every third packet in the sequence.
2. The designated packet, identified as packet 0, undergoes modification, altering its destination IP address and port to 192.168.60.5:8080.
3. This rule guarantees that one in every three UDP packets directed to the router's 8080 port is rerouted to 192.168.60.5.

Conclusion:

In conclusion, the exploration of network security in this lab encompassed various aspects of firewall implementation and configuration using iptables. Tasks involved the development of stateless and stateful firewall rules, connection tracking mechanisms, and the application of advanced iptables modules. Through ICMP, UDP, and TCP

experiments, students gained practical insights into the intricacies of connection tracking and stateful packet filtering.

The tasks not only reinforced the fundamental concepts of firewall construction but also delved into nuanced scenarios, such as protecting internal networks, controlling access to servers, and implementing packet rate limiting. The utilization of iptables modules like 'limit' and 'statistic' showcased the versatility of iptables in enforcing network traffic restrictions.

Moreover, the exploration of load balancing techniques demonstrated the multifaceted applications of iptables beyond traditional firewall functionalities. By manipulating packet routing based on load balancing policies, students learned to distribute network traffic effectively among multiple servers.

In retrospect, the lab provided a comprehensive hands-on experience for network security students, enabling them to apply theoretical knowledge in a practical setting. The tasks covered a spectrum of firewall-related topics, empowering students to implement robust security measures, optimize network traffic, and critically analyze the advantages and disadvantages of different approaches. Overall, this lab contributed significantly to enhancing the students' proficiency in network security and iptables utilization.