CSE-343 | Firewall Exploration Lab

Marc Soda Jr

March 30, 2022

Contents

Task 1.A:

- I followed the instructions as specified in the PDF and was successful. The only difference was that I added 'MODULE$_{\text{LICENSE}}$("GPL");' to hello.c because I got a make error because the license was not specified.
- Code:

```c
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");

int initialization(void) {
    printk(KERN_INFO "Hello World!\n");
    return 0;
}

void cleanup(void) {
    printk(KERN_INFO "Bye-bye World!.\n");
}

module_init(initialization);
module_exit(cleanup);
```

- dmesg output:

```
[1370717.558395] hello: loading out-of-tree module taints kernel.
[1370717.558458] hello: module verification failed: signature and/or required key missing - tainting kernel
[1370717.558730] Hello World!
[1370770.346669] Bye-bye World!.
```

Task 1.B:

Task 1:

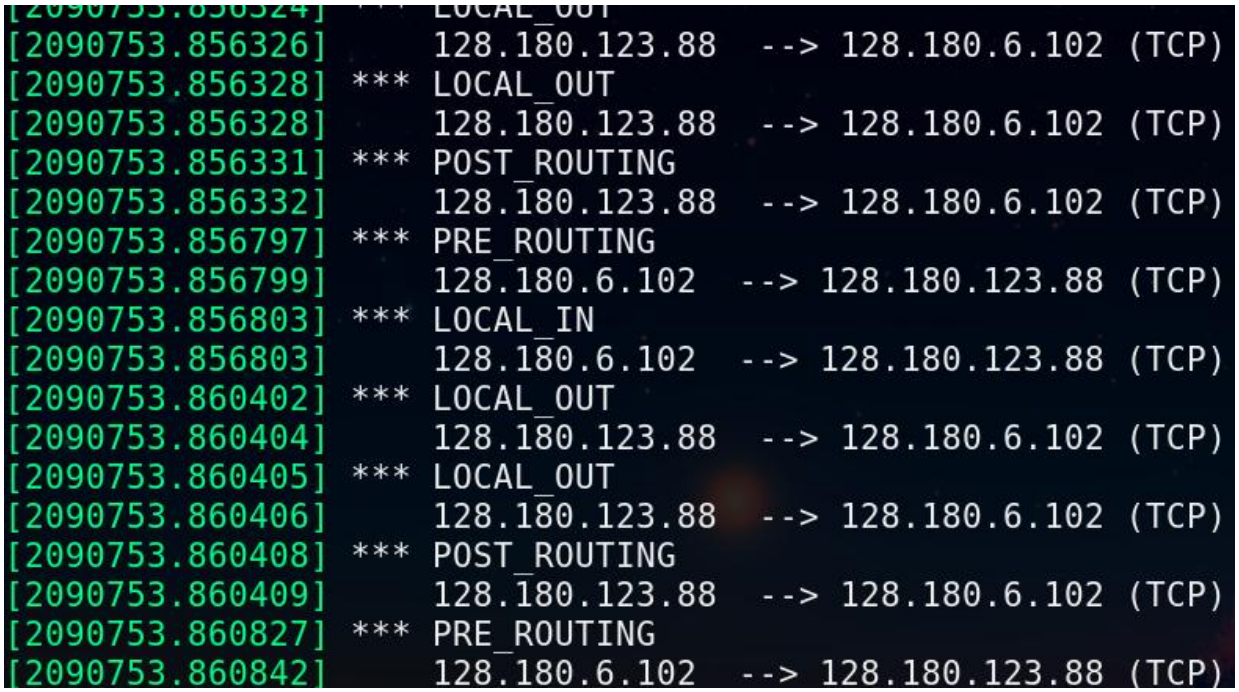
- Before adding the module, I am able to dig google.com with no issue. After adding the module, the request is blocked.
- Proof:

```
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: 22600
;; 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.        20525   IN      A       93.184.216.34

;; Query time: 12 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Mon Mar 21 14:21:38 EDT 2022
;; MSG SIZE  rcvd: 60

seed@VM:~/.../packet_filter$ sudo insmod seedFilter.
seedFilter.ko     seedFilter.mod.o  seedFilter.o
seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
seed@VM:~/.../packet_filter$ lsmod | grep seedF
seedFilter             16384  0
seed@VM:~/.../packet_filter$ !dig
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:~/.../packet_filter$
```

Task 2:

- Code

- After adding the module, I can the information being printed (through dmesg) corresponding to each hook.

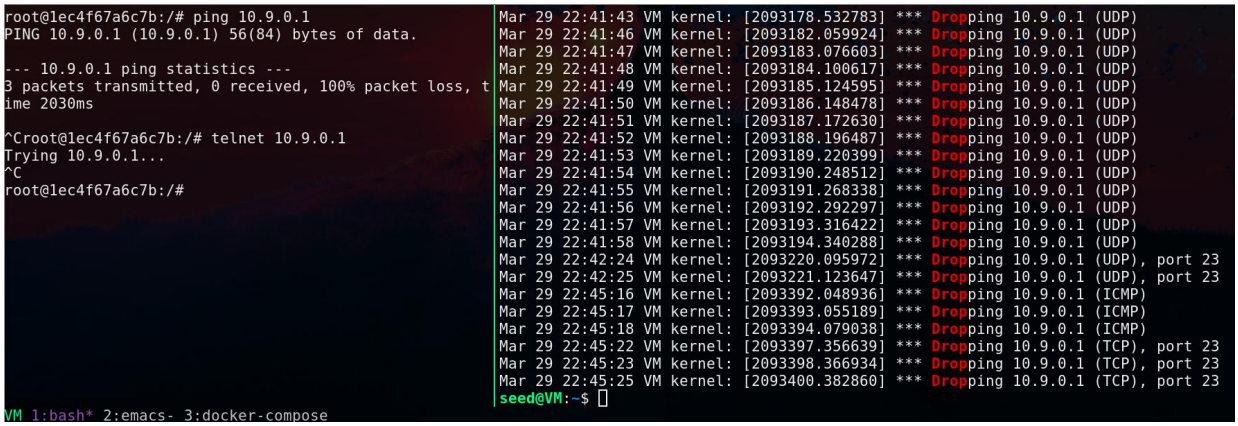


Task 3:

- blockPing and blockTelnet code:

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

   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;
}

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

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

   if (!skb) return NF_ACCEPT;

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

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

unsigned int printInfo(void *priv, struct sk_buff *skb,
```

- The following screenshot shows two shells. One shows HostA running telnet and ping. The other is a filtered cat of syslog showing which packets were dropped as a result of this task. As you can see, the new kernel module is blocking telnet and icmp packets. The reason why I could not show dmesg output is because when I do ‘dmesg -k -w‘, the output scrolls much too fast to read. I believe this is because I am connected to my seed server over ssh.



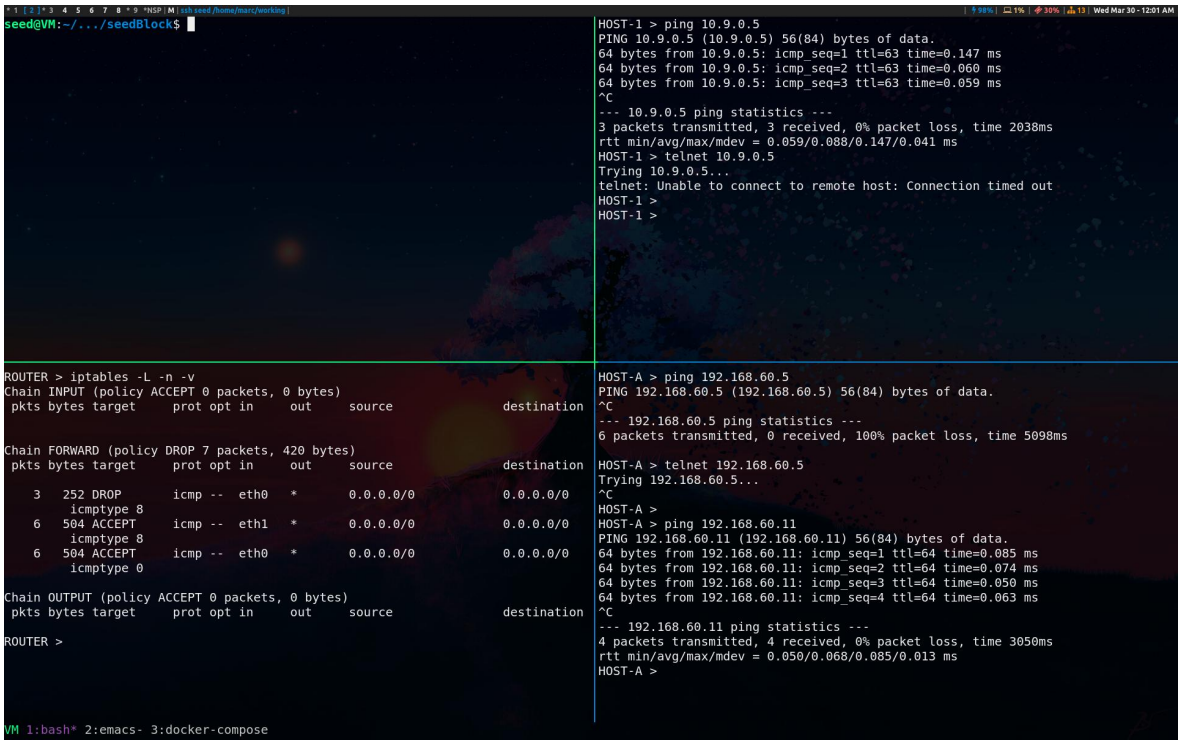
Task 2:

Task 2A:

- Before manipulating the IP tables I can ping and telnet into the router (10.9.0.11) just fine.
- After manipulating the IP tables with the specified command, I can ping the router, but I can't telnet into it.
- These rules drop ICMP packets except for PING.

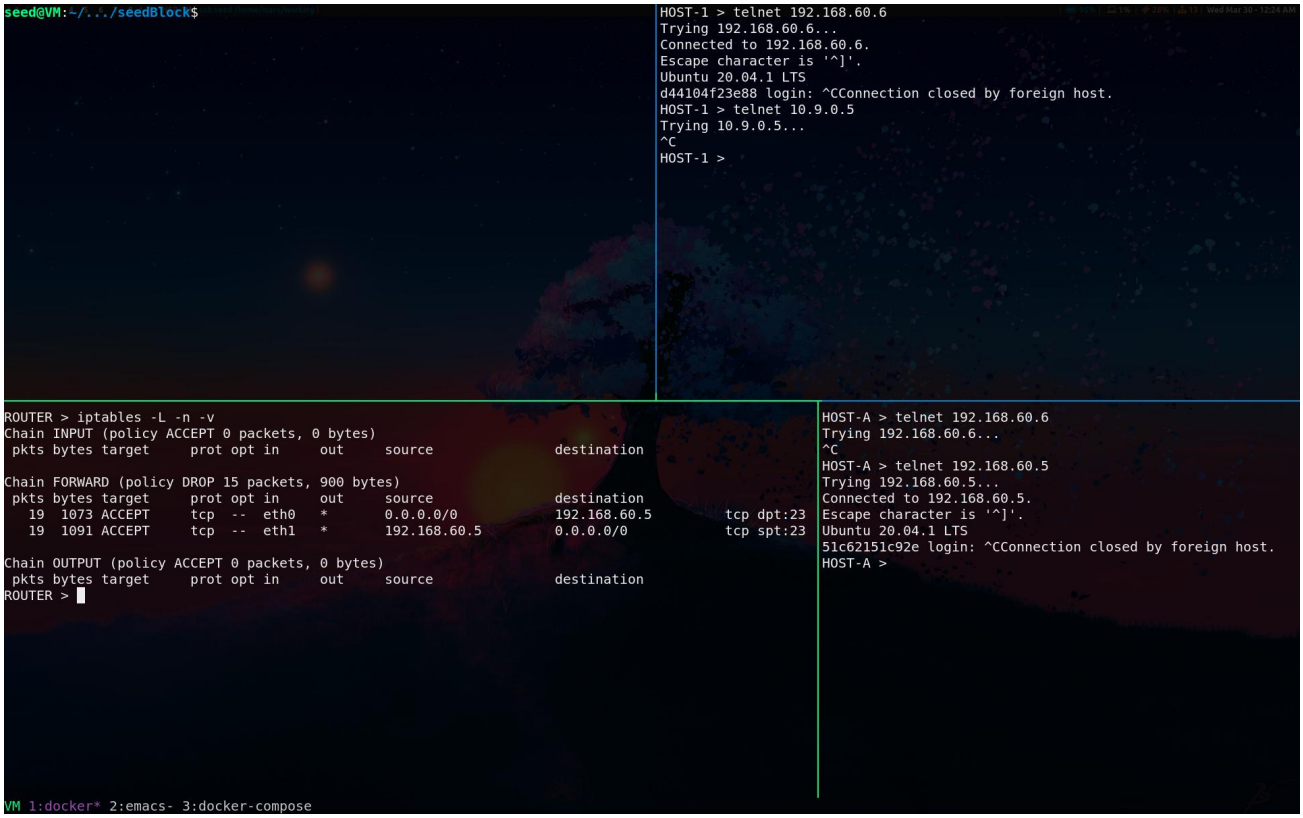
Task 2B:

- Commands (see screenshot for iptables rules output):
  - iptables -A FORWARD -i eth0 -p icmp –icmp-type echo-request -j DROP
  - iptables -A FORWARD -i eth1 -p icmp –icmp-type echo-request -j ACCEPT
  - iptables -A FORWARD -i eth0 -p icmp –icmp-type echo-reply -j ACCEPT
  - iptables -P FORWARD DROP
- Observations (see screenshot for proof):
  - HostA (outside) cannot ping Host1 (inside).
  - HostA (outside) can ping the router.
  - HostA (outside) cannot telnet Host1 (inside).
  - Host1 (inside) can ping HostA (outside)
  - Host1 (inside) cannot telnet HostA (outside)
- Screenshot
  - Top right is Host1. Bottom left is router. Bottom right is HostA. Hostnames reflected in prompt.



Task 2C:

- Commands (see screenshot for iptables rules output)
  - iptables -A FORWARD -i eth0 -p tcp -d 192.160.60.5 –dport 23 -j ACCEPT
  - iptables -A FORWARD -i eth1 -p tcp -s 192.160.60.5 –sport 23 -j ACCEPT
  - iptables -P FORWARD DROP
  - iptables -A FORWARD -i eth0 -p tcp -sport 5000 -j ACCEPT
- Observations (see screenshot for proof)
  - Host1 (inside) is able to telnet into other hosts on the internal network.
  - HostA (outside) is able to telnet to Host1 (inside), but not to any other internal hosts.
  - Host1 (inside) is unable to telnet to HostA (outside).
  - Note that other internal and external hosts were tested and the specifications were met. See screenshot.



Task 3:

Task 3A:

ICMP Experiment:

- After pinging Host1 from the router, conntrack -L shows details about the connection that was made.
- The ICMP connection state was kept for 30 seconds.

UDP Experiment:

- After communicating to Host1 from the router, conntrack -L shows details about the connection that was made.
- The UDP connection state was kept for 30 seconds.

TCP Experiment:

- After communicating from Host1 from the router, conntrack -L shows details about the connection that was made.
- The TCP connection state was kept for 120 seconds.

Task 3B:

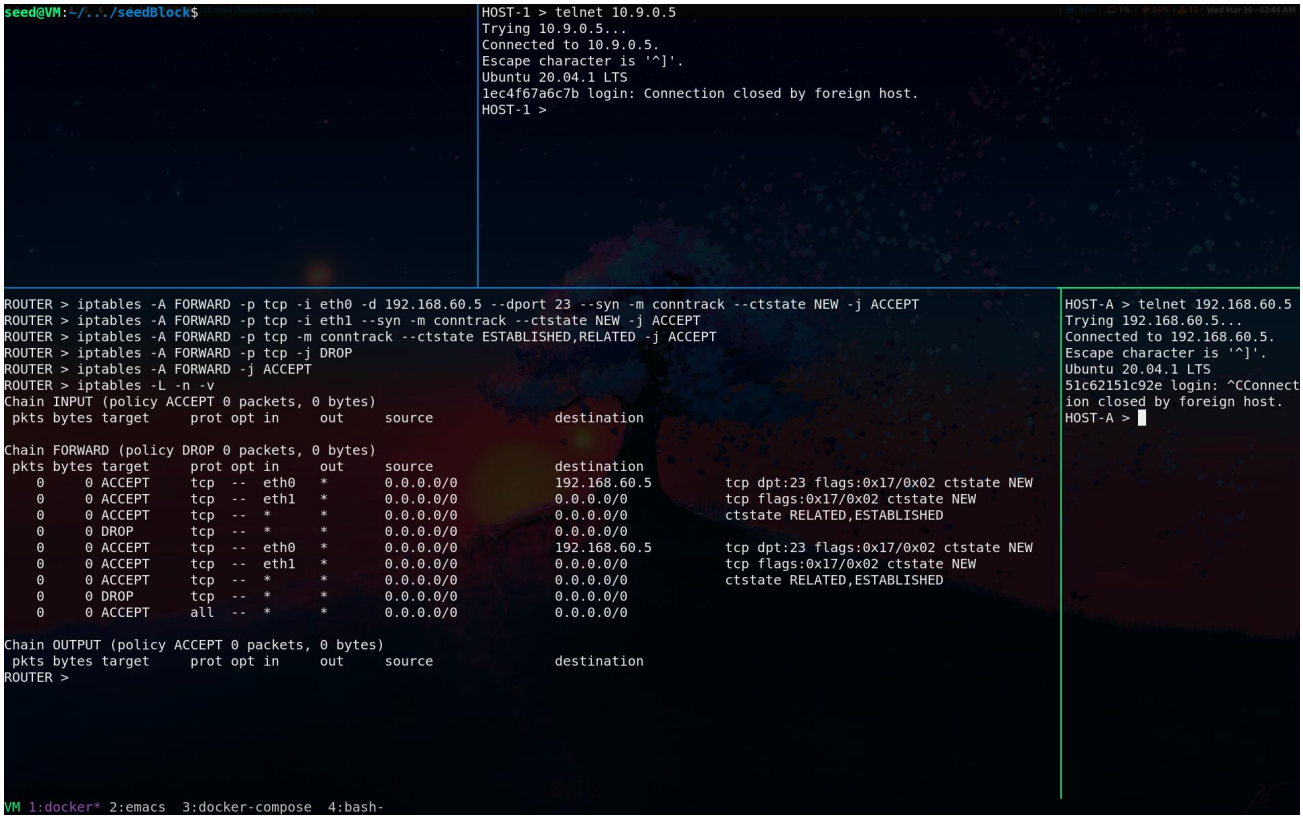
Commands:

- iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 –dport 23 –syn -m conntrack –ctstate NEW -j ACCEPT
- iptables -A FORWARD -p tcp -i eth1 –syn -m conntrack –ctstate NEW -j ACCEPT
- iptables -A FORWARD -p tcp -m conntrack –ctstate ESTABLISHED,RELATED -j ACCEPT

- iptables -A FORWARD -p tcp -j DROP
- iptables -A FORWARD ACCEPT

Observations (see screenshot):

- HostA (outside) can telnet to Host1 (inside)
- Host1 (inside) can telnet to HostA (outside)
- These patterns are maintained for other internal and external hosts.



Conntrack advantages:

- Consumes less CPU because caching

Conntrack disadvantages:

- Consumes more memory because connection states need to be saved for a certain amount of time.
- Can be poor at handling a high volume of connections per second.

Task 4:

- After running the first command only, pinging 192.168.60.5 is unaffected.
- After adding the second rule, the following behavior is observed:
  - The first 5 pings go through as normal.
  - Pings then begin being blocked due to the 5 connection burst limit and the 10/minute limit. I ended up getting a 67% packet loss after 10 successful pings.

Task 5:

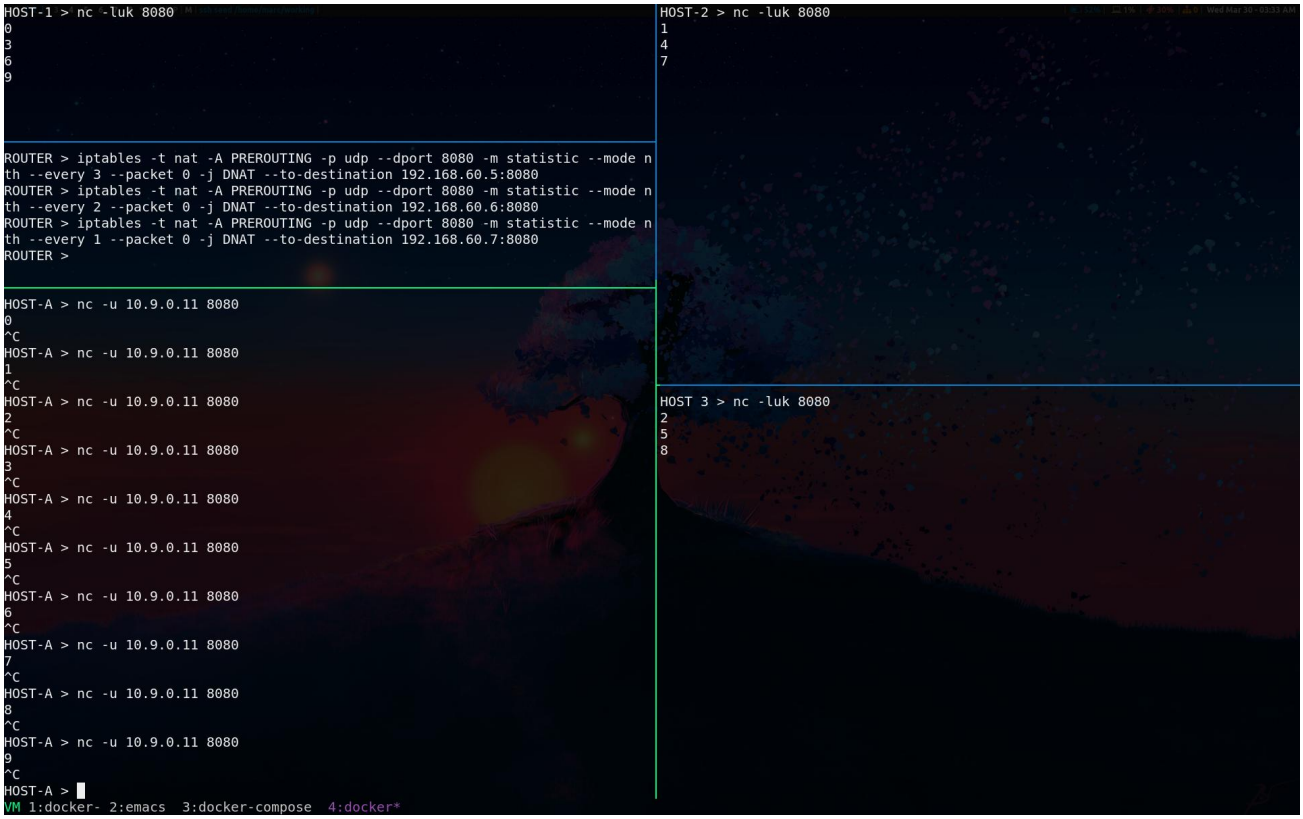
Round Robin Mode

Commands:

- 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 -m statistic –mode nth –every 1 –packet 0 -j DNAT –to-destination 192.168.60.7:8080
- Each rule chooses which number packet (out of three) to send to each different server.

Observations

- After adding the rules, I can make a connection and send a message from HostA to the router. Each time a connection is made, it is forwarded to a different server on the network. First Host1, then Host2, then Host3, then repeat. The load balancing task was successful. See screenshot.



Random Mode

Commands

- iptables -t nat -A PREROUTING -p udp –dport 8080 -m statistic –mode random –probability .3333 -j DNAT –to-destination 192.168.60.5:8080
- iptables -t nat -A PREROUTING -p udp –dport 8080 -m statistic –mode random –probability .5 -j DNAT –to-destination 192.168.60.6:8080

- iptables -t nat -A PREROUTING -p udp –dport 8080 -m statistic –mode random –probability 1 -j DNAT –to-destination 192.168.60.7:8080
- Apparently making the probability for each rule .3333 is wrong. When I did that I found that the connection would get dropped sometimes. I had to look it up and apparently you are supposed to make the first one .3333, the second .5, and the third 1. This has something to do with the fact that the rules are executed sequentially. With this setup, each host has a 33% chance of being selected.

Observations