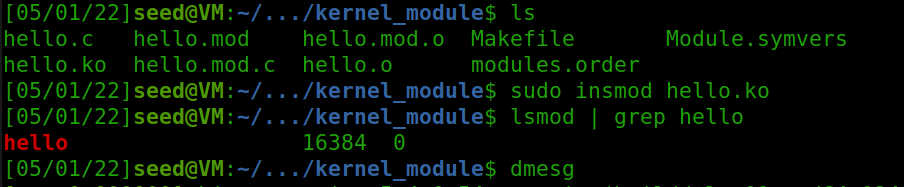
Val Robichaux

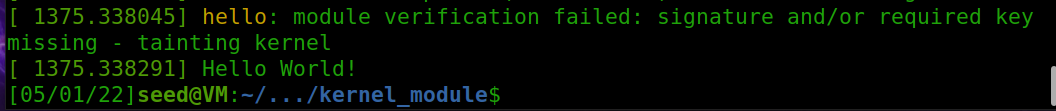
Homework 7

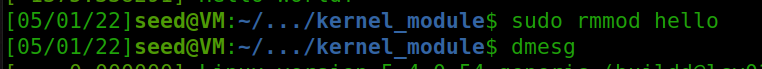
CSCE 465

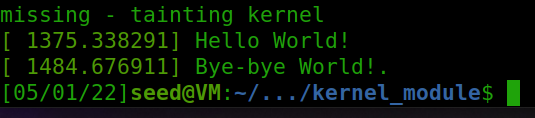
# Deliverables

1. Implementing a Simple Firewall
   1. Implement a Simple Kernel Module

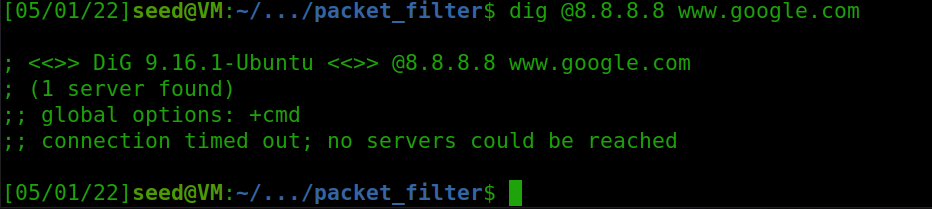




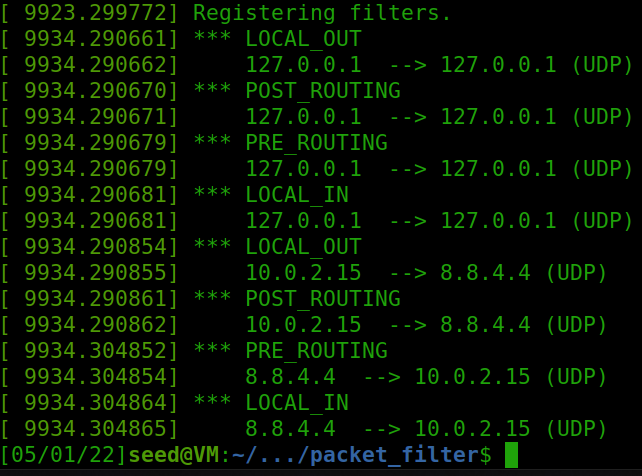




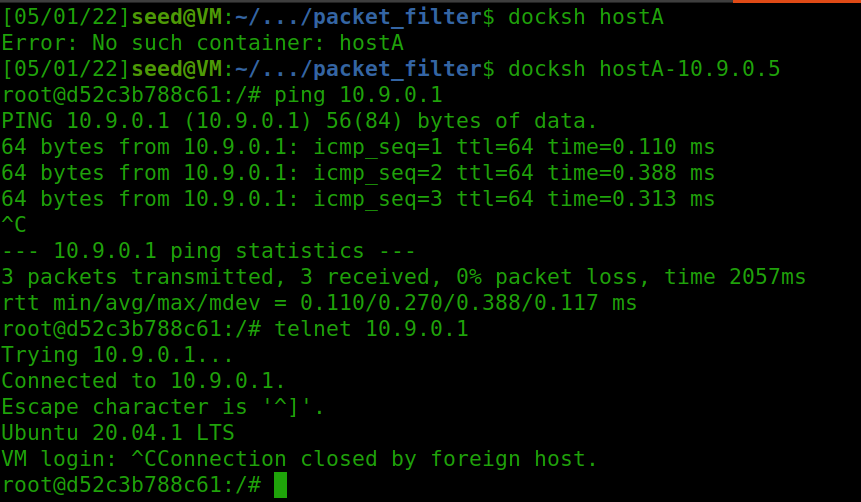
* Following the lab manual just loading and unloading the kernel module for proof that it works.
* Here I load it, read the messages it prints out, then I remove the module and read the messages it print out again
* Everything is working as expected!
  1. Implement a Simple Firewall using Netfilter
* Show your work following 3 steps from the instruction



* + Hook the *printInfo* function to all the *netfilter* hooks as it is described in the instruction.

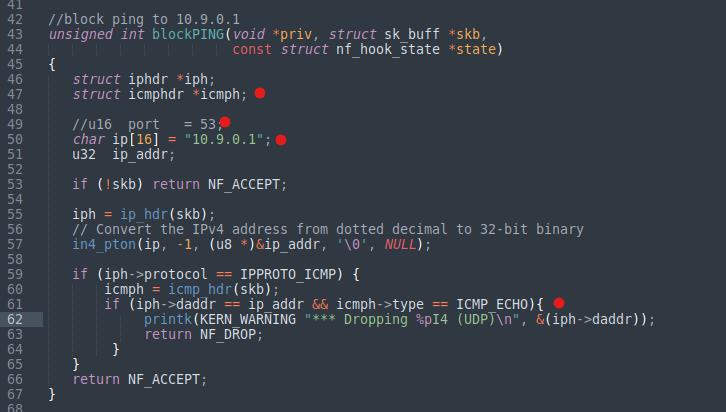


* **PRE\_ROUTING**
  + Triggered after the packet enters the IP layer but before routing the packet
* **LOCAL\_IN**
  + After routing the packet, if the packet is sent to the local machine, this hook will be invoked.
* **FORWARD**
  + After routing the packet, if it is not sent to the local, and before forwarding the packet this will be invoked
  + This is not invoked on our script because this process is usually handled by the router.
* **LOCAL\_OUT**
  + The local output for the output packet before it is routed will invoke this hook.
* **POST\_ROUTING**
  + After routing, the output packet will invoke this after the output packet is routed.
* Implement 2 more hooks to achieve 2 goals



* Initially testing that we can use another container to ping and telnet to our VM

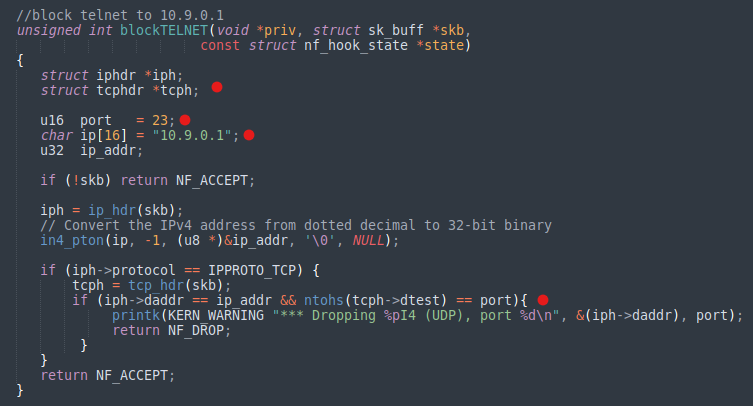
**BLOCK PING**

****

**Red dots explained below in order**

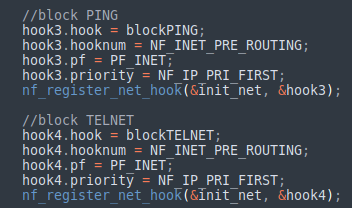
* Changed the icmp header since ping uses ICMP protocols
* ICMP does not use ports so we can comment that out
* Change the IP address to our VM’s own IP address for ease of use
* Check the ICMP type if it specifically receiving and echo req

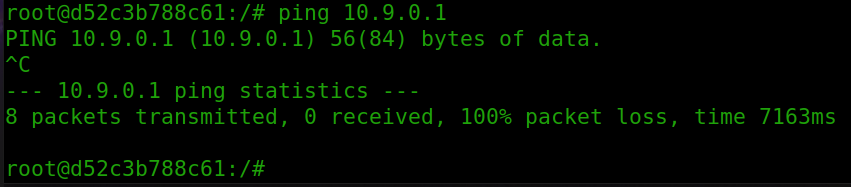
**BLOCK TELNET**

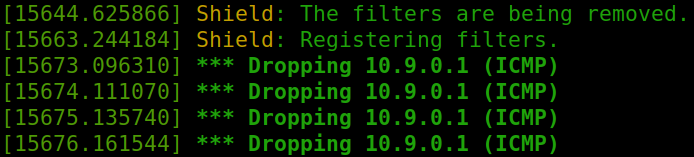
****

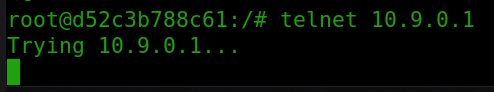
**Red dots explained below in order**

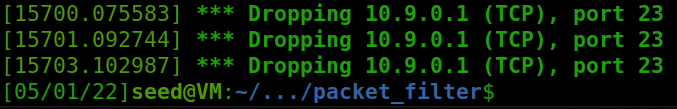
* Changed to TCP header since telnet uses the TCP protocols
* TCP does use port numbers so we are going to add that back and change it to the default port of 23
* IP address should point to our VM
* Finally check if the telnet request is hitting the correct port that we are asking for.





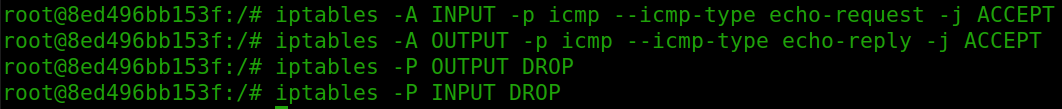


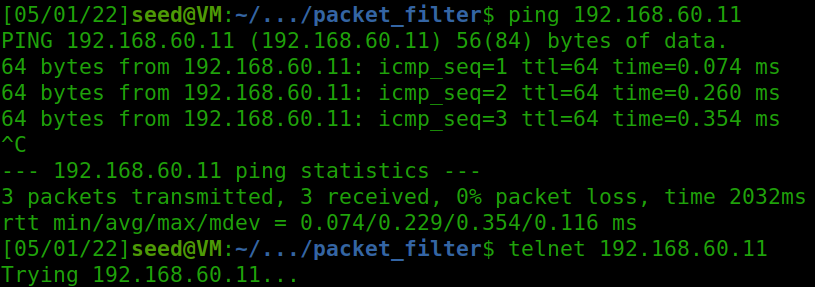




* As we can see in both cases, I am using the PRE\_ROUTING protocol to ensure that the packets are checked and filtered before they are routed to the specific designation
* Then the process is as follows
  + **PING**
    - The packet gets checked before it’s routed and we are checking if there is a response of the type ICMP\_ECHO as well as the destination of 10.9.0.1.
    - If these are triggered and the function is invoked, then the packet will be dropped accordingly.
  + **TELNET**
    - Telnet is similar but instead we are checking for the port number, IP address, and if the port request matches the designated port
    - If these are all triggered and the function is invoked, the packet will be dropped.
    - Port number is important in the second screenshot to see the telnet blocking working accordingly.

1. Experimenting with Stateless Firewall Rules
   1. Protecting the Router

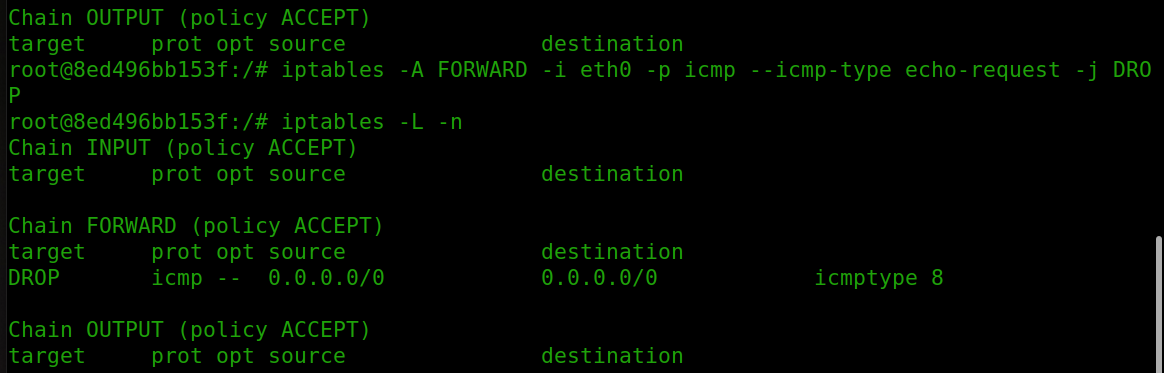


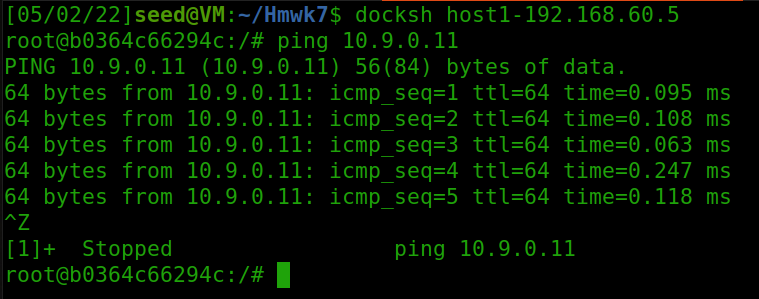


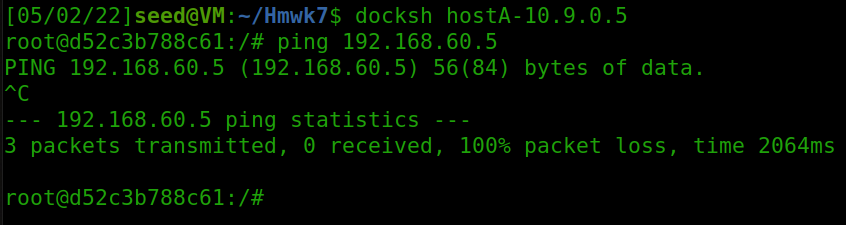
* Yes I can ping into the router but no I can not telnet into the router as it just hangs when trying to connect
* The only rules that are being set are to accept echo requests and send echo requests out by the router, so trying to telnet into the router drops the packet by default.
  1. Protecting the Internal Network
* Using *iptables* command, make the firewalls to protect the internal network *192.168.60.0/24*
  + Be aware of the restrictions listed on the instruction



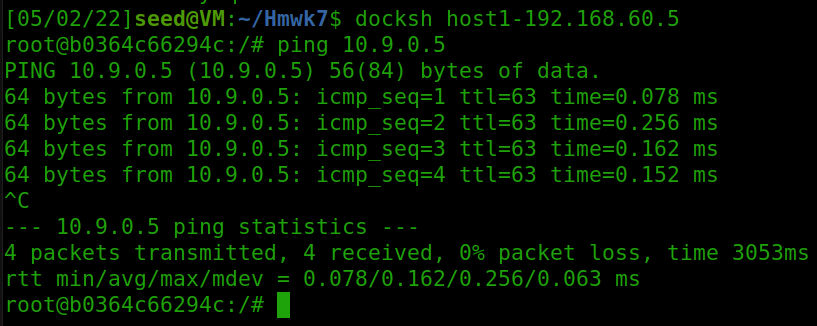
* This policy drops all echo request packets when the outside hosts are trying to ping our inside hosts.



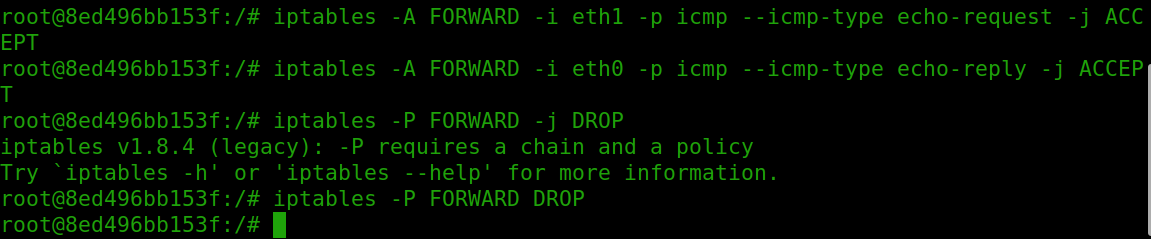




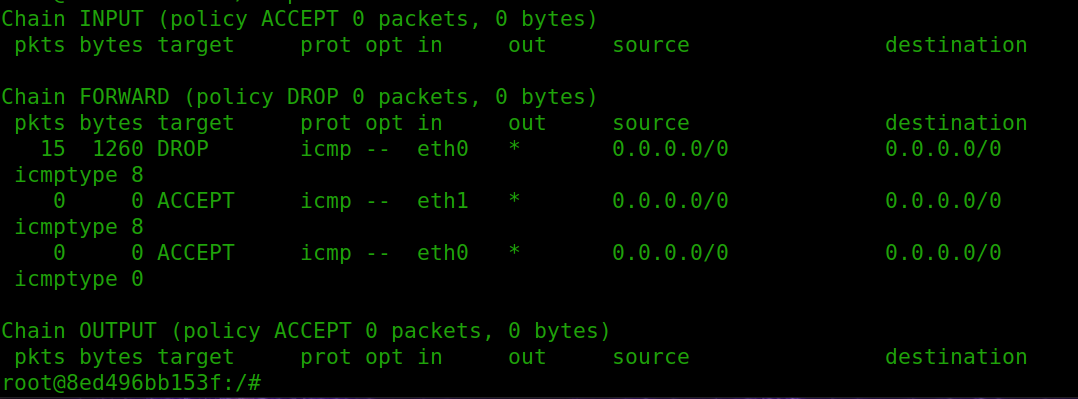
* Here we can see that our outside hosts can ping the router but can not ping internal hosts

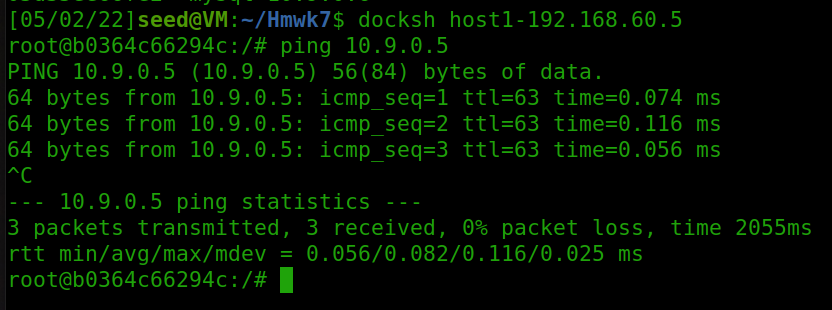


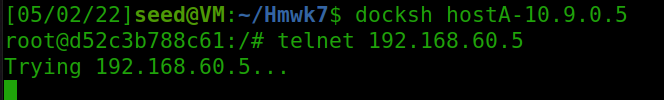
* Here we can see that our internal host can ping the external hosts

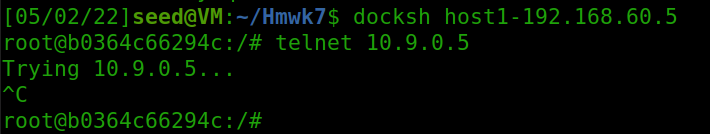


* Here we are setting a policy to accept echo-requests on the first interface (eth1) which should be our internal hosts
* Then we are setting a policy to accept echo-replies on the second interface (eth0) which should be our outside hosts
* This allows for our inside hosts to remain able to ping our outside hosts in the program

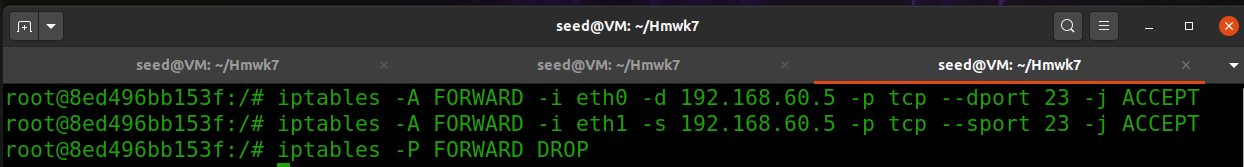




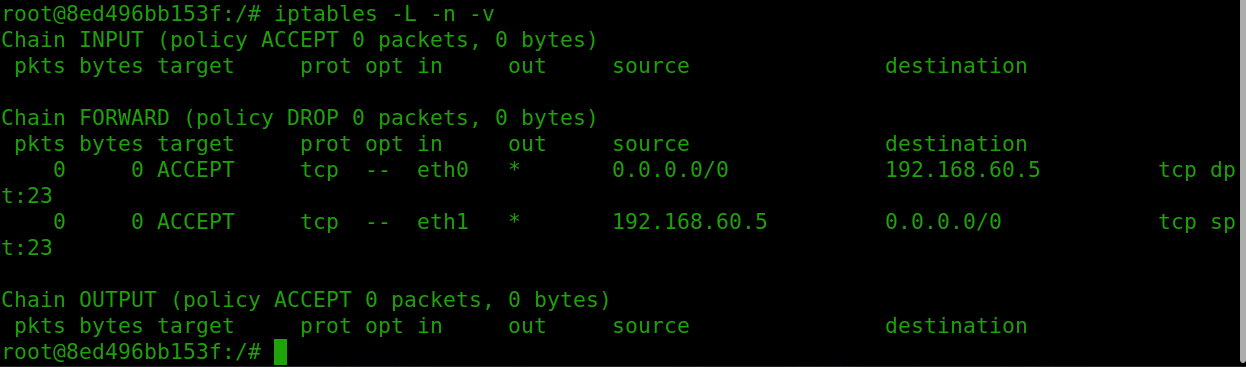


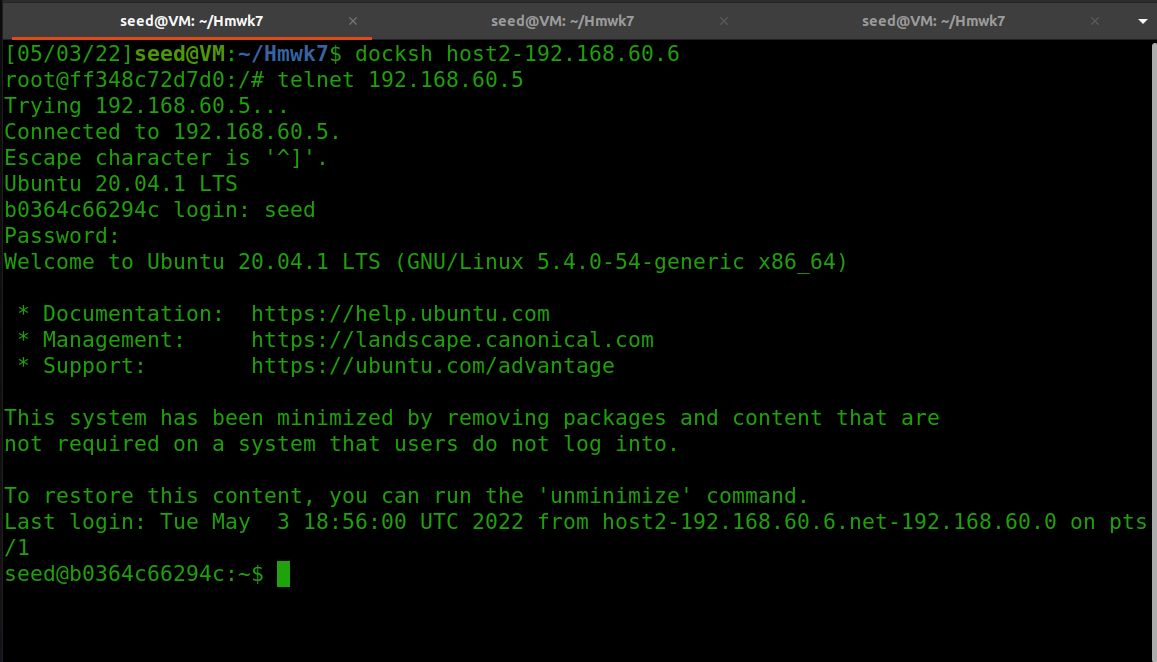


* Finally for the last case we can see that all the other packets between the outside and inside hosts are being blocked
  1. Protecting Internal Servers

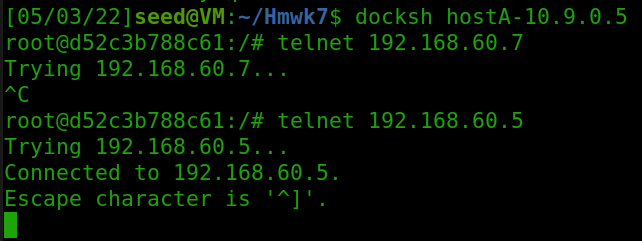


* The first command allows all TCP packets from outside hosts (eth0) to be forwarded to the destination 192.168.60.5 that also include the destination port 23
* The second command allows for all TCP packets from our inside users (eth1) to accept packets where the source IP is 192.168.60.5 and the source port is 23
* The last command is setting a policy that drops any other type of network protocols

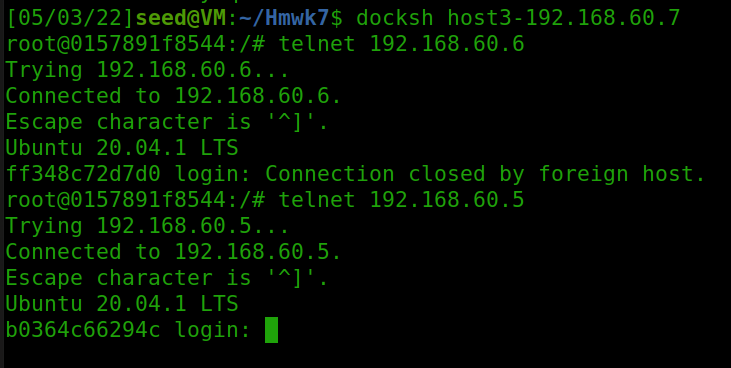




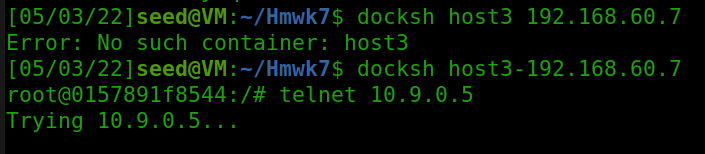
* Demonstrating that my inside hosts can run telnets to other internal servers.



* This demonstrates that the outsider host can only access the internal host located at 192.168.60.5 and not other internal hosts

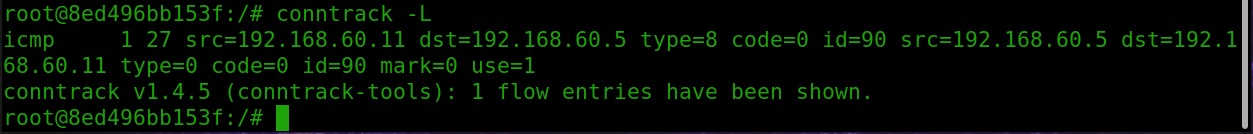


* Here I demonstrate that host 3 can access host 2 and 1, thus all internal hosts and access other internal hosts



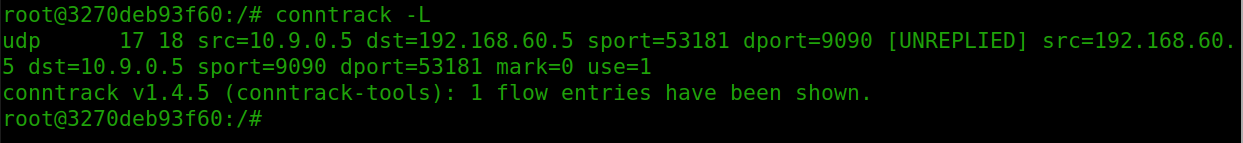
* Here I demonstrate that the internal hosts are not able to access external hosts

1. Connection Tracking and Stateful Firewall
   1. Experiment with the Connection Tracking



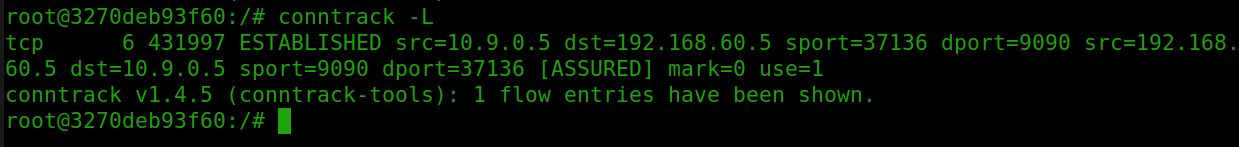


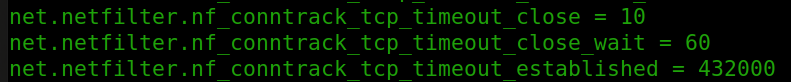
* ICMP timeout = 30



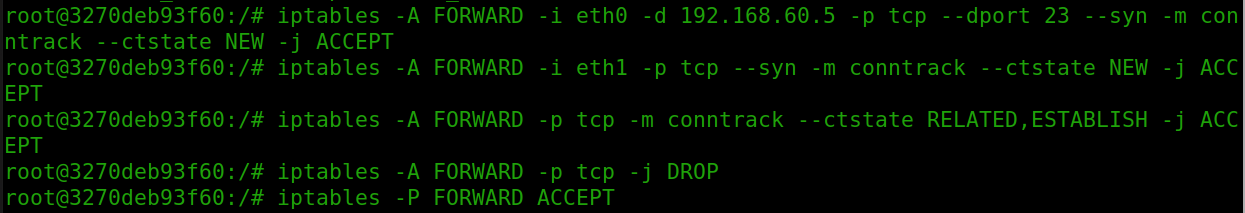


* UDP timeout = 30

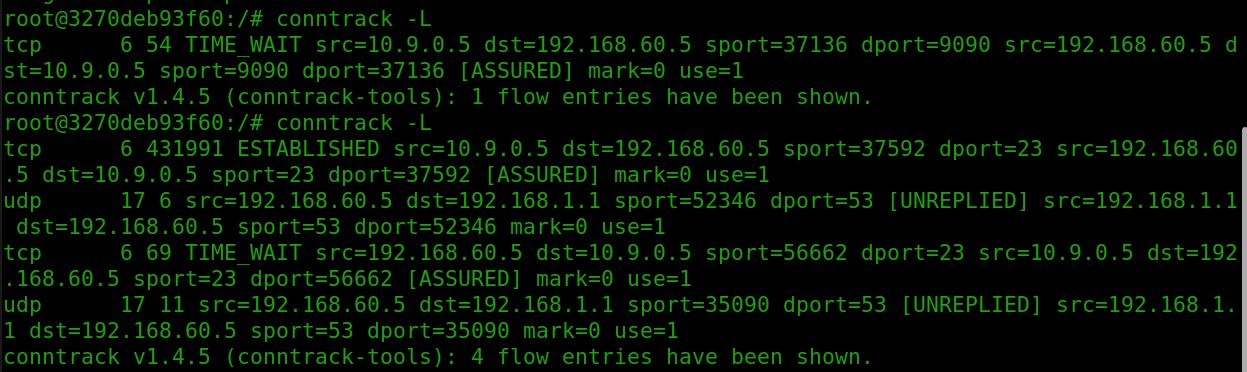




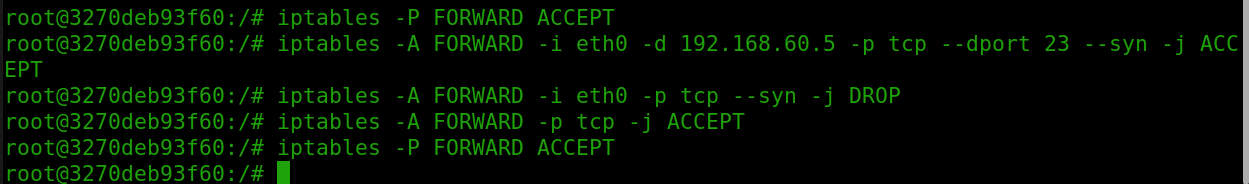
* TCP = 120
  1. Setting Up a Stateful Firewall



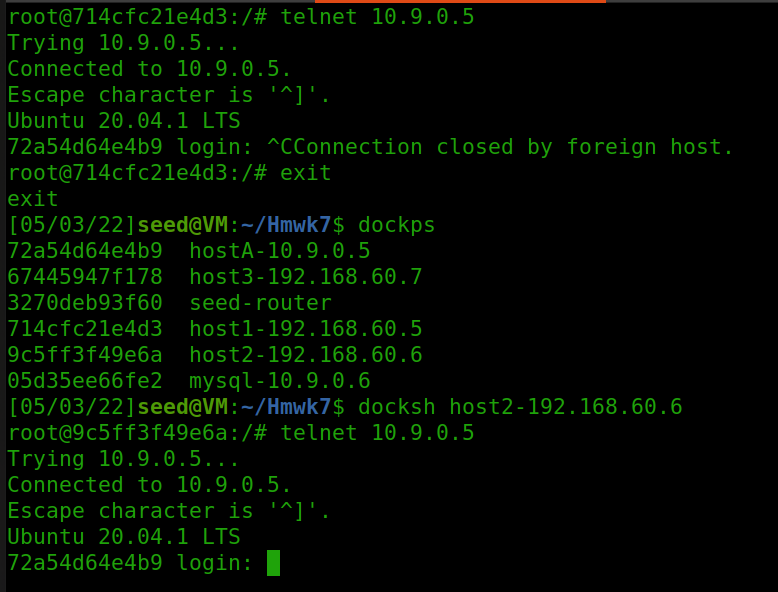
* The first rule is allowing outside hosts to accept replies from internal hosts.
* The second rules is allowing internal hosts to send TCP packets to external hosts that are being created on a new connection
* The third rule is allowing packets that is associated with an existed connection to be sent to outside hosts
* The last rules drop and other packets that should not be sent



* Verifying connections using the conntrack module



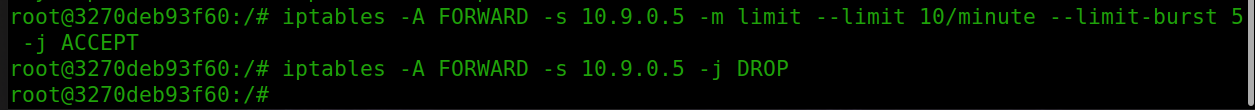
* Here I don’t use the conntrack module when making my rules
* We run the same command in this case but instead of checking for a new connection or connections that are already established, we check and match TCP packets with the SYN bit set so we are able to check which TCP connections are outgoing.

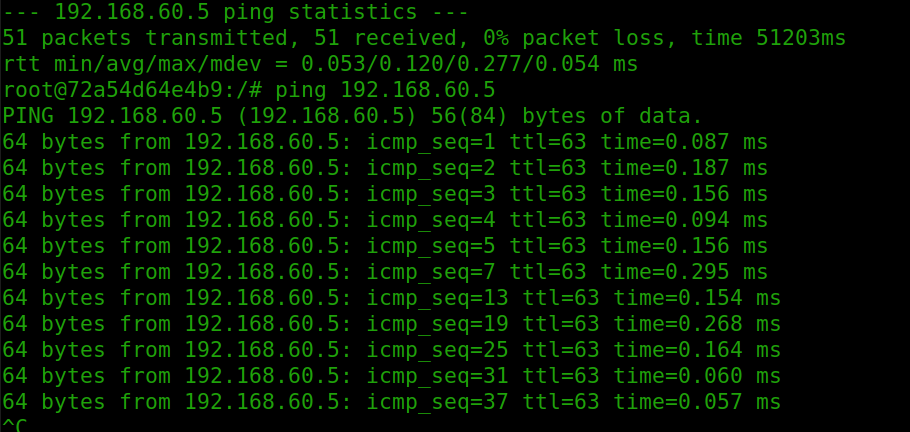


* Verifying our results and connections but this time not using the conntrack module

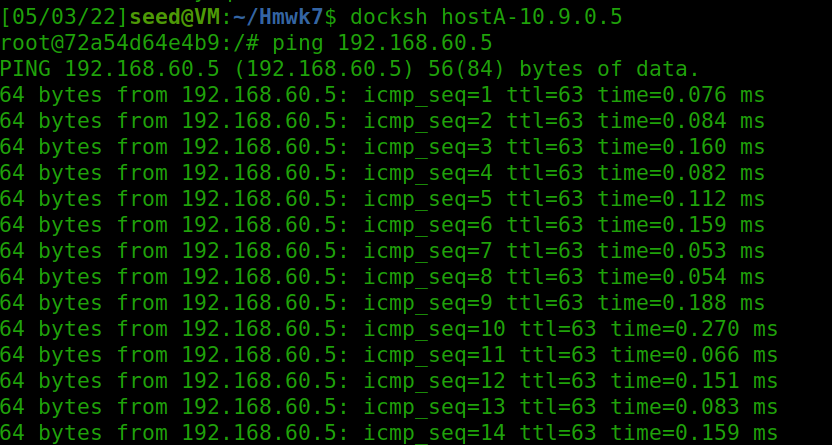
1. Limiting Network Traffic

* Show the result of pinging *192.168.60.5* from *10.9.0.5* in 2 conditions:





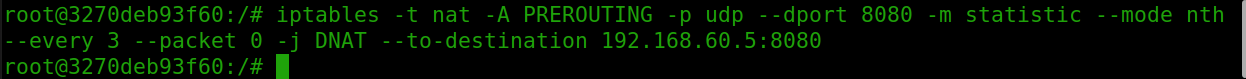


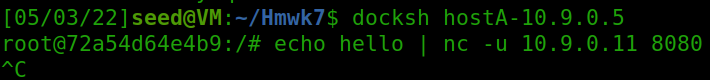


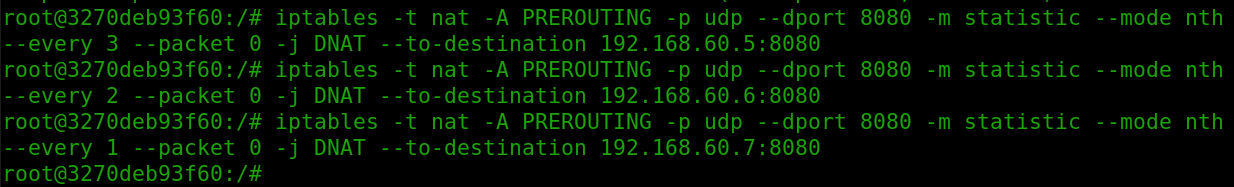
* The second rule is needed because if we don’t have the second rule none of the packets will actually get dropped.
* The limit only takes effect when the packets start getting dropped.
* The first few packets likely do not match the first rule and they pass on like normally without getting dropped.

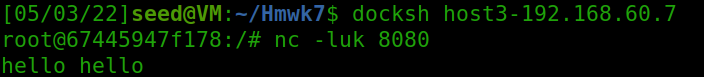
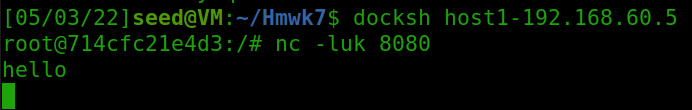
1. Load Balancing

* Show the comparison between two modes:



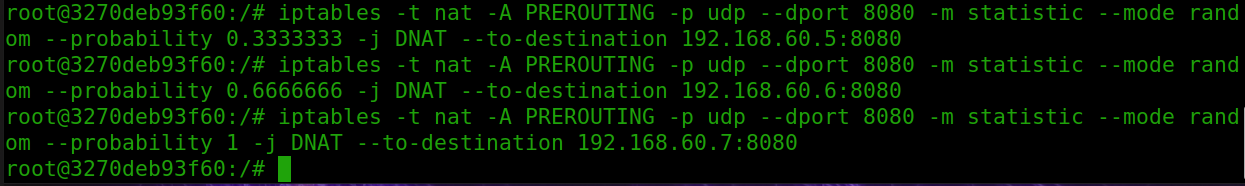


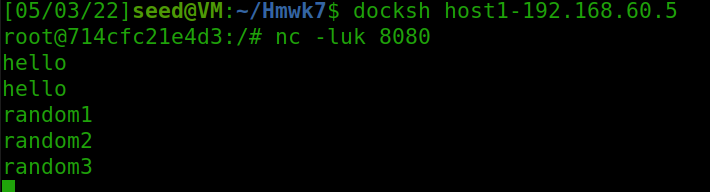


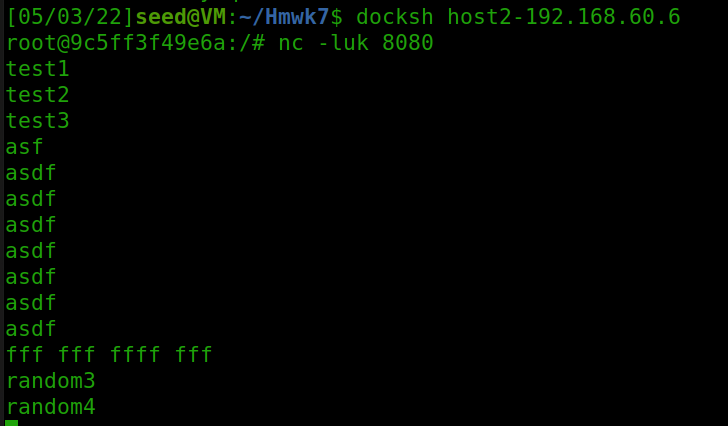


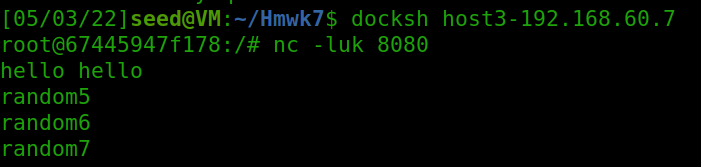


**Random module**











* For the Nth mode operating
  + We have to find certain Nth mode variables in order to all three internal hosts to get the equal number of packers so when we are using the rules I replace the --every flag with different numbers and a different ip according to which host I am going to be sending them to
  + I just went in descending order from 3 to 1 and made sure that I had every internal host’s IP included
    - Every 3rd packet, every 2nd packet every 1st packet etc….
  + This allowed me to send an equal number of packets to each internal host
* For random mode operating
  + We have to set a probability P to use with our rules that would be roughly equivalent when passing packets and balancing the load
  + I determined that it would be easy to keep a standard fractional incrementation of 0333 / 0.666 / 1 that way I could achieve some sort of similarity when using the random operation mode.
    - Every 3rd packet, every 2nd packet out of 3 and then every packet.