Firewall Exploration Lab

GITHUB REPOSITORY:
https://github.com/matteodalgrande/Ethical-Hacking-UniPD-Challenges

Task 1: Implement a simple firewall

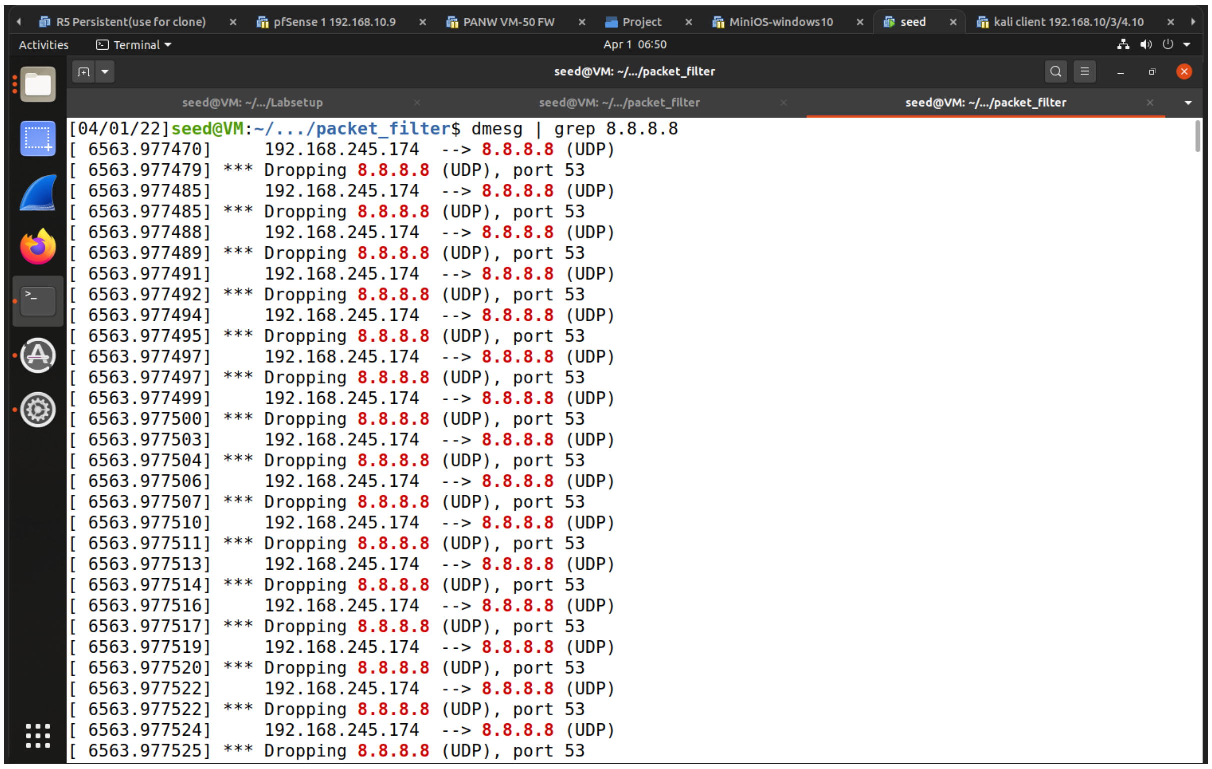
Task 1.B: Implement a Simple Firewall Using Netfilter

We ran the modules named seedFilter.ko

Then by changing the DNS of the virtual machine setting 8.8.8.8. To apply this change we ran the command sudo service network-manager restart.

After that we open a terminal an try to *dig google.com* url and we saw that the resolution of the web server was not successful.

By checking the kernel logs using *dmesg* | *grep 8.8.8.8* we can se how there has been a *Dropping 8.8.8.8 (UDP), port 53.*


R5 Persistent(use for clone)
pfSense 1 192.168.10.9
PANW VM-50 FW
Project
MiniOS-windows10
seed
kali client 192.168.10/3/4.10
Activities
Terminal
Apr 1 06:50
seed@VM: ~/.../packet_filter
seed@VM: ~/.../Labsetup
seed@VM: ~/.../packet_filter
seed@VM: ~/.../packet_filter

```
[04/01/22]seed@VM:~/.../packet_filter$ dmesg | grep 8.8.8.8
[ 6563.977470]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977479] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977485]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977485] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977488]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977489] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977491]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977492] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977494]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977495] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977497]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977497] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977499]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977500] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977503]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977504] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977506]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977507] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977510]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977511] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977513]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977514] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977516]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977517] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977519]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977520] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977522]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977522] *** Dropping 8.8.8.8 (UDP), port 53
[ 6563.977524]     192.168.245.174  --> 8.8.8.8 (UDP)
[ 6563.977525] *** Dropping 8.8.8.8 (UDP), port 53
```

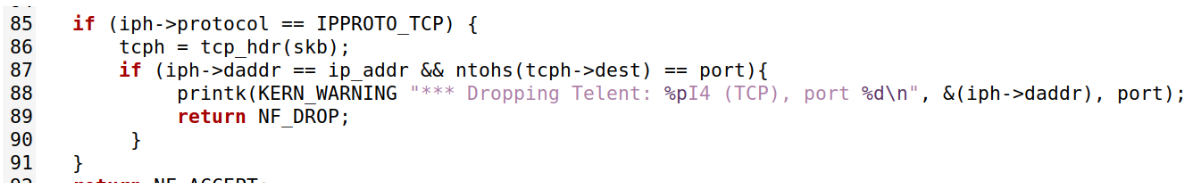
1. **Coompile the sample code using the provided `Makefile`. Load it into the kernel, and demonstrate that the firewall is working as expected. You can use the following command to generate UDP packets to `8.8.8.8`, which is Google's DNS server. If your firewall works, your request will be blocked; otherwise, you will get a response. `dig @8.8.8.8 www.example.com`**

2. **Hook the `printInfo` function to all of the `netfilter` hooks. Here are the macros of the hook numbers.**
   Using your experiment results to help explain at what condition will each of the hook function be invoked.

3. **Implement two more hooks to achieve the following:**
   a. **Preventing other computers to ping the VM**
   b. **Preventing other computers to telnet into the VM. Please implement two different hook functions, but register them to the same `netfilter` hook. You should decide what hook to use. Telnet's default port is TCP port 23. To test it, you can start the containers, go to `10.9.0.5`, run the following commands (`10.9.0.1` is the IP address assigned to the VM; for the sake of simplicity, you can hardcode this IP address in your firewall rules)**

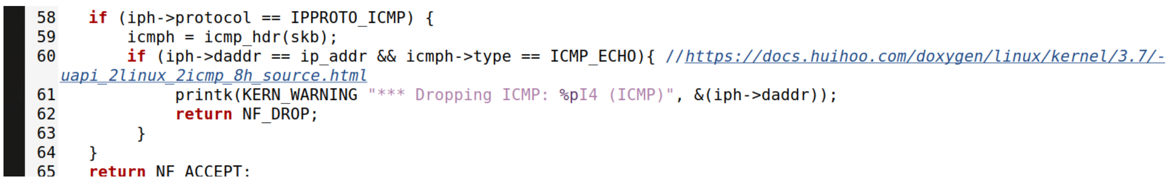
   In order to complete this task is necessary to create a C file program in a way that can be compiled and loaded by the kernel.
   The C file used is in
   Labsetup/Files/04_preventing_other_ping_to_vm/preventing_ping.c
   The most relevant part are:
   1. create a function to register the hook
   2. create a function to remove the hook
   3. create two functions that are called **blockICMP** and **blockTELNET** that check the IPaddress and the destination port and the protocol in order to match the packets passing through the kernel. After a match we can do some stuff by dropping the packet.

The following script shows how to match a protocol and then look inside the packet checking the destination address and the destination port for TCP protocol(in the case of telnet connection).



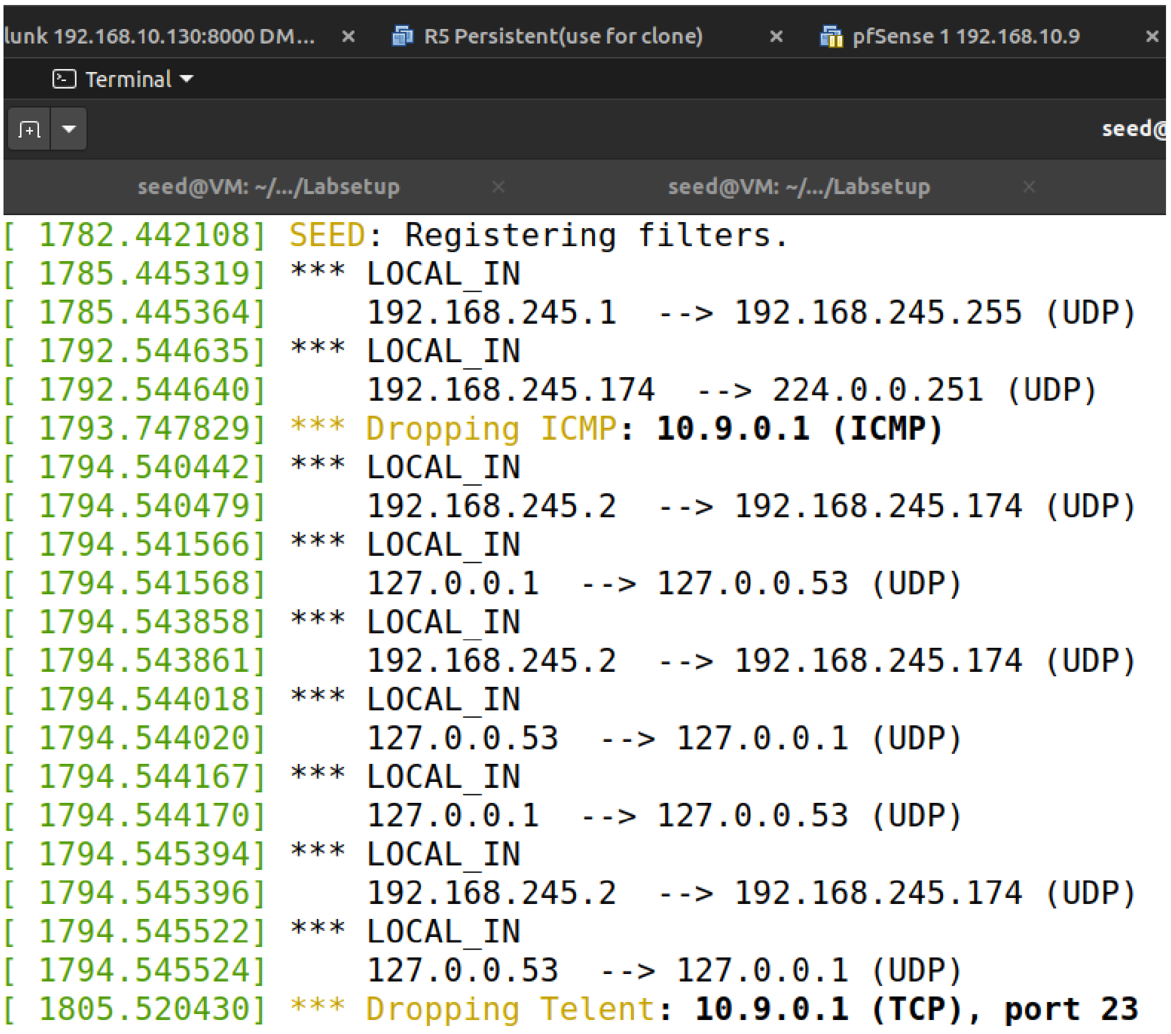
```
    if (iph->protocol == IPPROTO_TCP) {
        tcph = tcp_hdr(skb);
        if (iph->daddr == ip_addr && ntohs(tcph->dest) == port){
             printk(KERN_WARNING "*** Dropping Telent: %pI4 (TCP), port %d\n", &(iph->daddr), port);
             return NF_DROP;
         }
    }
```

The following script shows how to match a protocol and then look inside the packet checking the destination address and the type of the icmp packets(in the case of icmp connection).



```
    if (iph->protocol == IPPROTO_ICMP) {
        icmph = icmp_hdr(skb);
        if (iph->daddr == ip_addr && icmph->type == ICMP_ECHO){ //https://docs.huihoo.com/doxygen/linux/kernel/3.7/-
uapi_2linux_2icmp_8h_source.html
             printk(KERN_WARNING "*** Dropping ICMP: %pI4 (ICMP)", &(iph->daddr));
             return NF_DROP;
         }
    }
    return NF_ACCEPT;
```

The following script shows the log file of the kernel to verify that the script is running correctly.



```
[ 1782.442108] SEED: Registering filters.
[ 1785.445319] *** LOCAL_IN
[ 1785.445364]     192.168.245.1  --> 192.168.245.255 (UDP)
[ 1792.544635] *** LOCAL_IN
[ 1792.544640]     192.168.245.174  --> 224.0.0.251 (UDP)
[ 1793.747829] *** Dropping ICMP: 10.9.0.1 (ICMP)
[ 1794.540442] *** LOCAL_IN
[ 1794.540479]     192.168.245.2  --> 192.168.245.174 (UDP)
[ 1794.541566] *** LOCAL_IN
[ 1794.541568]     127.0.0.1  --> 127.0.0.53 (UDP)
[ 1794.543858] *** LOCAL_IN
[ 1794.543861]     192.168.245.2  --> 192.168.245.174 (UDP)
[ 1794.544018] *** LOCAL_IN
[ 1794.544020]     127.0.0.53  --> 127.0.0.1 (UDP)
[ 1794.544167] *** LOCAL_IN
[ 1794.544170]     127.0.0.1  --> 127.0.0.53 (UDP)
[ 1794.545394] *** LOCAL_IN
[ 1794.545396]     192.168.245.2  --> 192.168.245.174 (UDP)
[ 1794.545522] *** LOCAL_IN
[ 1794.545524]     127.0.0.53  --> 127.0.0.1 (UDP)
[ 1805.520430] *** Dropping Telent: 10.9.0.1 (TCP), port 23
```

Task 2: Experimenting with Stateless Firewall Rules

Task 2.A: Protecting the Router

1. Can you ping the router? 2. Can you telnet into the router (a telnet server is running on all the containers; an account called `seed` was created on them with a password `dees`).

Before inserting the INPUT and OUTPUT policies as DROP into the filter table everything it's working fine and we can ping and telnet the router 10.9.0.11 from the host 10.9.0.5. After the insertion of the DROP rules and the rules that accept only echo-request and echo-reply, we can only ping the router from the host container and no more connect using telnet.

Task 2.B: Protecting the Internal Network

In order to perform this task it's necessary to create very simple rules to drop or accept packets passing through the router.

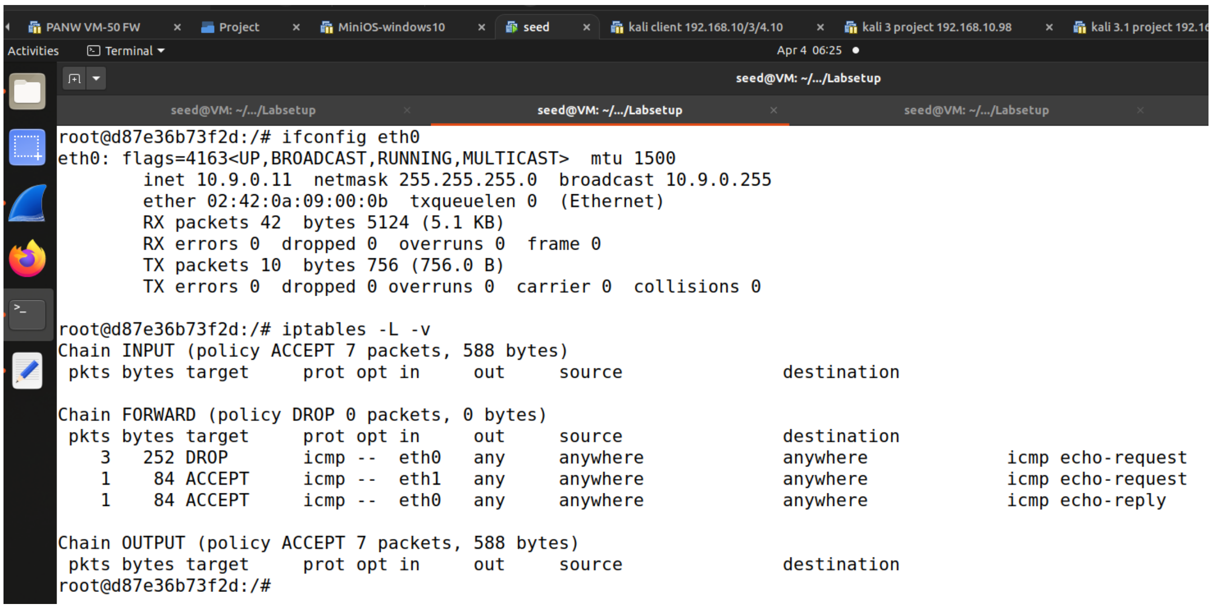
The commands to execute are the below.

iptables -t filter -A FORWARD -p icmp --icmp-type echo-request -i eth0 -j DROP &&

iptables -t filter -A FORWARD -p icmp --icmp-type echo-request -i eth1 -j ACCEPT &&

iptables -t filter -A FORWARD -p icmp --icmp-type echo-reply -i eth0 -j ACCEPT &&

iptables -P FORWARD DROP



```
root@d87e36b73f2d:/# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.11  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:0b  txqueuelen 0  (Ethernet)
        RX packets 42  bytes 5124 (5.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 756 (756.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@d87e36b73f2d:/# iptables -L -v
Chain INPUT (policy ACCEPT 7 packets, 588 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
    3   252 DROP       icmp --  eth0   any     anywhere             anywhere             icmp echo-request
    1    84 ACCEPT     icmp --  eth1   any     anywhere             anywhere             icmp echo-request
    1    84 ACCEPT     icmp --  eth0   any     anywhere             anywhere             icmp echo-reply

Chain OUTPUT (policy ACCEPT 7 packets, 588 bytes)
 pkts bytes target     prot opt in     out     source               destination
root@d87e36b73f2d:/#
```

The above screenshot shows the rules inside the seed-router container.

Task 2.C: Protecting Internal Server

The commands that have to be executed inside the seed-router container are the following two.

```
iptables -t filter -A FORWARD -d 192.168.60.0/24 ! 192.168.60.5 -p tcp --sport 1024:65535 -dport 23 -i
eth0 -j REJECT &&
iptables -P FORWARD DROP
```

These two commands allow traffic only for the telnet connection on port 23 from external network to the host 192.168.60.5 and dropping all the others with destination host in the subnetwork 192.168.60.0/24.