greenthom – 300536064
Lab 4: Packet Filtering Firewall

pfSense Firewall

1. **Explain why the ping command in "Step 23" of "Section 1" of the lab fails**

Ping command in step 23 of the lab fails because a new firewall rule has been added to the pfSense firewall, which blocks ICMP (ping) traffic from the Kali system to the Ubuntu system. In step 14 of the lab, the instruction is to add a new firewall rule set to block and protocol set to ICMP which effectively prevents any ICMP traffic, including ping requests from reaching the Ubuntu system.

When ping command is executed (23), Kali system sends out ICMP echo request packets to the Ubuntu system's IP address (192.168.1.50), the Ubuntu system does not receive these packets due to the firewall protocol. The ping command reports that 4 packets were transmitted but 0 were received, indicating an unsuccessful ping attempt.

2. **Compare the screeshot in "Step 1" of "Section 2.1" of the lab with the screenshot in "Step 1" of "Section 2.2". Identify the difference, and then, explain why we see this difference.**

'2.1' the screenshot shows through using the nmap scan that there is no SSH port open on the firewall appliance -> it is closed or filtered. This is because no port forwarding rule has been configured yet to redirect SSH traffic to an internal host. '2.2' after configuring port forward for SSH traffic in the previous steps, port 22 (ssh port) on the firewall appliance is now open shown through the nmap scan result showing that port 22 is accessible.

Difference is due to the configuration change made in 2.1 where a port forwarding rule was added to redirect SSH traffic to an internal host (192.168.1.50). This configuration change allows SSH traffic to be redirected from the firewall appliance to the specified internal host -> opening port 22 on the firewall appliance.

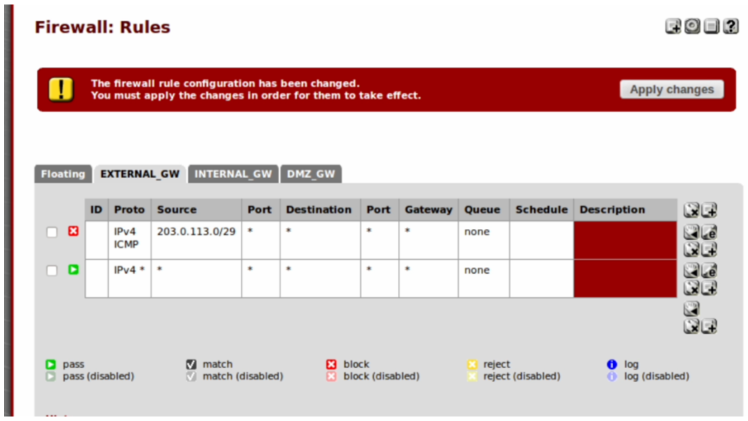
3. **In "Step 5" of Section 2.2 of the lab, explain why we do not see the SSH service.**

In 2.1 we configured the firewall to forward SSH traffic to 192.168.1.50 on port 22 which means that any SSH traffic directed at the firewall's external IP address 203.0.113.1 will be redirected to the Ubuntu system. When conducting the nmap scan against default gateway 192.168.1.1 (firewall) the SSH traffic is now redirected to the internal host so scanning the firewall interface won't reveal the SSH service. Redirection of SSH traffic effectively means there is no SSH service directly accessible on the firewall.

4. **Is pfsense a stateful or stateless firewall? Demonstrate the statefulless or statelessness of pfsense firewall using the lab "Configuring a Network-Based Firewall".**

Stateful firewalls keep track of the state/connections through using a state table -> allows them to differentiate legit packets (established connections) and malicious packets (unauthorized connections). pfSense is a stateful firewall -> retains information of connections traversing through it, retains state of each connection and facilitates traffic management and security -> maintains a state table that records information about incoming connections (src and dest IP) -> allow traffic for established connections and authorize outer traffic response to connections. Keeps track of network by monitoring traffic.

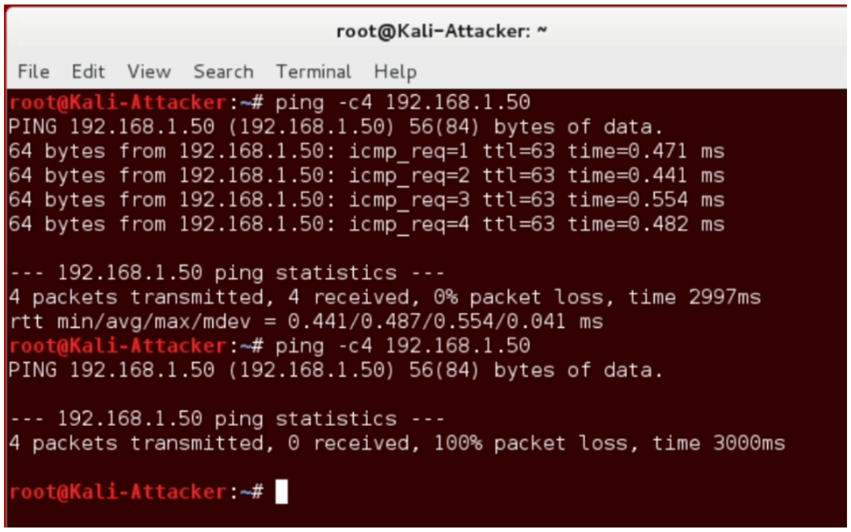
In 1.1 when ICMP requests are blocked by adding a firewall rule, attempts to ping the Ubuntu system from Kali result in unsuccessful attempts to ping. In 2.1 when configuring port forwarding for SSH, the firewall is changed to redirect SSH traffic from external interface to internal host. When performing a nmap scan on the firewall appliance IP address 192.168.1.1 the SSH service is not detected.



Firewall: Rules

The firewall rule configuration has been changed.
You must apply the changes in order for them to take effect.

	ID	Proto	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description
☐ ❎		IPv4 ICMP	203.0.113.0/29	*	*	*	*	none		[redacted]
☐ ▶		IPv4 *	*	*	*	*	*	none		[redacted]



root@Kali-Attacker: ~

```
root@Kali-Attacker:~# ping -c4 192.168.1.50
PING 192.168.1.50 (192.168.1.50) 56(84) bytes of data.
64 bytes from 192.168.1.50: icmp_req=1 ttl=63 time=0.471 ms
64 bytes from 192.168.1.50: icmp_req=2 ttl=63 time=0.441 ms
64 bytes from 192.168.1.50: icmp_req=3 ttl=63 time=0.554 ms
64 bytes from 192.168.1.50: icmp_req=4 ttl=63 time=0.482 ms

--- 192.168.1.50 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.441/0.487/0.554/0.041 ms
root@Kali-Attacker:~# ping -c4 192.168.1.50
PING 192.168.1.50 (192.168.1.50) 56(84) bytes of data.

--- 192.168.1.50 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3000ms

root@Kali-Attacker:~# █
```

Walking on Firewalls!

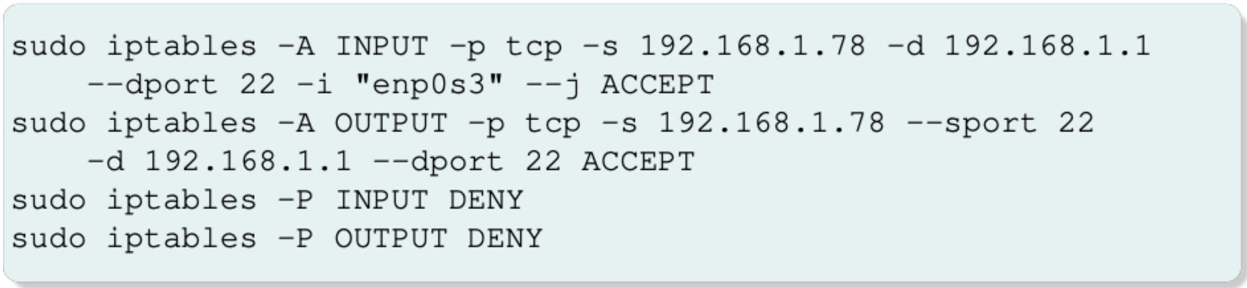
5. **What is Firewalking (the technique and/or the tool) and how can it be used in an attack process against a potential target?**

THIS LAB DID NOT WORK

Firewalking is a tool used to assess the effectiveness of firewall rules or ACLs within a network. Works like the traceroute command -> utilizes time-to-live (TTL/hop limits) that you'll find in IP header information. When routers look at TTL if the value is not set to 0 the router decreases the value by 1 and it'll continue forwarding the packet onto its destination address. If the hop limit reaches 0 the router sends back an ICMP time exceeded message back to the sender. With firewalking, it sends packets one at a time and it keeps increasing the TTL value until it reaches a target. Once it reaches a target it increases the TTL one more time and then begins the requested scan that is set. If it doesn't get a response, we assume whatever we tried to send it is being blocked or there is no firewall set -> no access control list. It is used in an attack to identify open ports, detect ACL rules, and avoid detection by intrusion systems or logging in firewall. Essentially an attack to plan for an attack -> understand system layout. Used to gather information to plan for further attacks mostly used in the investigation phase into conducting an attack by identifying hosts beyond a firewall for exploitation.

Packet Filtering by iptables

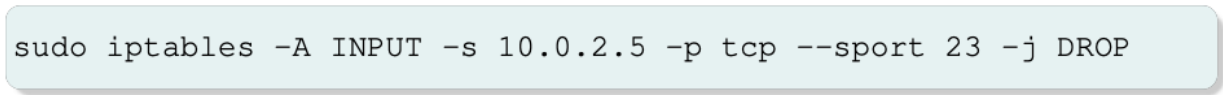
6. **We have the following rule in our iptables on the server (192.168.1.1). The client (192.168.1.78) however fails to initiate an SSH connection. Explain why this is the case.**



```
sudo iptables -A INPUT -p tcp -s 192.168.1.78 -d 192.168.1.1
    --dport 22 -i "enp0s3" --j ACCEPT
sudo iptables -A OUTPUT -p tcp -s 192.168.1.78 --sport 22
    -d 192.168.1.1 --dport 22 ACCEPT
sudo iptables -P INPUT DENY
sudo iptables -P OUTPUT DENY
```

The issue with the given IPTABLE rules is that the policy defaults for INPUT and OUTPUT are to deny. This would mean the rules for incoming and outgoing traffic will be 'deny' regardless of if they have been set. The two rules allowing SSH traffic from client (192.168.1.78) to server (192.1.1) are appended to INPUT and OUTPUT chains on port 22. If they had been set after the sudo iptables -P INPUT/OUTPUT DENY they would have been effective, however all incoming and outgoing traffic is denied. The way to change this to allow all necessary traffic is to change DENY to ACCEPT.

7. **Our server is running a Telnet service listening on port 23. We would like to stop any new Telnet connections from a client with the IP address of 10.0.2.5. Is the following a good rule to block the Client? Explain.**



```
sudo iptables -A INPUT -s 10.0.2.5 -p tcp --sport 23 -j DROP
```

Provided rule attempts to block incoming Telnet traffic from client 10.0.2.5 IP on port 23 (TELNET) by dropping packets with src IP 10.0.2.5 on port 23 using –sport. The –sport option in iptables specifies the src port of the incoming packet, not the dest port on which the service is running. Therefore, the rule would then block incoming traffic from 10.0.2.5 on src port 23. The 23 port is the port on the client not server. Telnet is used on destination port not source port. Therefore, to block connections from the IP address the rule should target dest port 23 rather than src using –dport. Using dport will target incoming TCP traffic from 10.0.2.5 on port 23 and instruct iptables to drop packets matching the criteria. Proper rule: sudo iptables -A INPUT -s 10.0.2.5 -p tcp --dport 23 -j DROP.

8. **Write a rule to drop all the new and established outgoing SSH service requests received on your primary interface (eth0) which has a source MAC address of 30:65:EC:22:14:D1.**

sudo iptables -A OUTPUT -o eth0 -m mac --mac-source 30:65:EC:22:14:D1 -p tcp --dport 22 -m state --state NEW,ESTABLISHED -j DROP

9. **Write a rule to drop any incoming packets with “INVALID” state.**

sudo iptables -A INPUT -m state --state INVALID -j DROP