**Configure Iptable Rules**

**Iptables Lab: Port 4000 and Packet Dropping**

**Overview**

This lab provides a hands-on experience with iptables, focusing on managing traffic on port 4000 and observing packet dropping behavior. It's designed for users who want to understand firewall rules, packet filtering, and network diagnostics using tools like iptables and tcpdump.

Prerequisites

* Two Virtual Machines (VM1 and VM2) running Ubuntu 22.04.5 LTS
* Both VMs in the same VPC with private IP addresses
* SSH access to both VMs
* Basic understanding of Linux command line and networking concepts

**Lab Steps**

**Step 1: Set up the VMs**

1. Ensure both VMs are running and can communicate with each other.
2. Note down the private IP addresses of both VMs. For this lab, we'll assume:
   * VM1: 10.0.0.10
   * VM2: 10.0.0.20

**Step 2: Switch to legacy iptables (if needed)**

On both VMs, switch to the legacy iptables backend:

sudo update-alternatives --set iptables /usr/sbin/iptables-legacy

sudo update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy

**Step 3: Initial connectivity test**

SSH into VM1:

ssh [user@10.0.0.10](mailto:user@10.0.0.10)

From VM1, ping VM2 to ensure basic connectivity:

ping 10.0.0.20

If the ping is successful, you have confirmed that the VMs can communicate.

**Step 4: Set up basic iptables rules on VM1**

On VM1, set up the following rules:

sudo iptables -F

sudo iptables -A INPUT -i lo -j ACCEPT

sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

sudo iptables -A INPUT -p tcp --dport 4000 -j ACCEPT

sudo iptables -P INPUT DROP

Explanation of rules:

* Clear existing rules
* Allow loopback traffic
* Allow established connections
* Allow SSH (port 22)
* Allow traffic on port 4000
* Set default policy to DROP for incoming traffic

Save the rules:

sudo iptables-save > /etc/iptables/rules.v4

**Step 5: Set up a simple server on VM1**

On VM1, start a simple HTTP server on port 4000:

python3 -m http.server 4000

**Step 6: Test connectivity from VM2**

On VM2, try to connect to VM1 on port 4000:

curl <http://10.0.0.10:4000>

This should succeed as we've allowed traffic on port 4000.

**Step 7: Modify rules to drop packets on port 4000**

On VM1, modify the rule to drop packets on port 4000:

sudo iptables -D INPUT -p tcp --dport 4000 -j ACCEPT

sudo iptables -A INPUT -p tcp --dport 4000 -j DROP

Save the updated rules:

sudo iptables-save > /etc/iptables/rules.v4

**Step 8: Use tcpdump to observe packet dropping**

On VM1, start tcpdump to monitor traffic on port 4000:

sudo tcpdump -i any port 4000 -nn -v

Leave this running in a separate terminal window.

**Step 9: Test connectivity again from VM2**

On VM2, try to connect to VM1 on port 4000 again:

curl <http://10.0.0.10:4000>

This should now fail, and you should see the connection attempts in the tcpdump output on VM1.

**Step 10: Analyze tcpdump output**

When you run tcpdump on VM1 and attempt to connect from VM2, you should see output similar to this:

root@00153a2e9972278f:~/code# sudo tcpdump -i any port 4000 -nn -v

tcpdump: data link type LINUX\_SLL2

tcpdump: listening on any, link-type LINUX\_SLL2 (Linux cooked v2), snapshot length 262144 bytes

20:44:50.467073 eth0 In IP (tos 0x0, ttl 64, id 3122, offset 0, flags [DF], proto TCP (6), length 60)

10.62.2.83.35030 > 10.62.2.82.4000: Flags [S], cksum 0x194f (incorrect -> 0x6d42), seq 755547984, win 64240, options [mss 1460,sackOK,TS val 992413366 ecr 0,nop,wscale 7], length 0

20:44:51.476574 eth0 In IP (tos 0x0, ttl 64, id 3123, offset 0, flags [DF], proto TCP (6), length 60)

10.62.2.83.35030 > 10.62.2.82.4000: Flags [S], cksum 0x194f (incorrect -> 0x6950), seq 755547984, win 64240, options [mss 1460,sackOK,TS val 992414376 ecr 0,nop,wscale 7], length 0

20:44:53.492606 eth0 In IP (tos 0x0, ttl 64, id 3124, offset 0, flags [DF], proto TCP (6), length 60)

10.62.2.83.35030 > 10.62.2.82.4000: Flags [S], cksum 0x194f (incorrect -> 0x6170), seq 755547984, win 64240, options [mss 1460,sackOK,TS val 992416392 ecr 0,nop,wscale 7], length 0

Let's break down this output:

**First line:**

tcpdump: data link type LINUX\_SLL2

tcpdump: listening on any, link-type LINUX\_SLL2 (Linux cooked v2), snapshot length 262144 bytes

This indicates that tcpdump is listening on all interfaces using the LINUX\_SLL2 (Linux cooked v2) data link type.

**For each packet captured, we see a line like this:**

20:44:50.467073 eth0 In IP (tos 0x0, ttl 64, id 3122, offset 0, flags [DF], proto TCP (6), length 60)

* 20:44:50.467073: Timestamp of the packet
* eth0: The interface that received the packet
* In: Direction (incoming packet)
* IP: Internet Protocol version (IPv4 in this case)
* tos 0x0: Type of Service (0 is normal)
* ttl 64: Time to Live
* id 3122: IP identification number
* offset 0: Fragment offset (0 means it's not fragmented)
* flags [DF]: Don't Fragment flag is set
* proto TCP (6): Protocol is TCP
* length 60: Total length of the IP packet

**The next line details the TCP connection:**

10.62.2.83.35030 > 10.62.2.82.4000: Flags [S], cksum 0x194f (incorrect -> 0x6d42), seq 755547984, win 64240, options [mss 1460,sackOK,TS val 992413366 ecr 0,nop,wscale 7], length 0

* 10.62.2.83.35030 > 10.62.2.82.4000: Source IP and port to destination IP and port
* Flags [S]: SYN flag is set, indicating this is a connection initiation attempt
* cksum: Checksum information
* seq 755547984: Initial sequence number
* win 64240: Window size
* options: Various TCP options including Maximum Segment Size (mss), Selective Acknowledgment (sackOK), Timestamp (TS), and Window Scale (wscale)
* length 0: No data in this packet (it's just a SYN packet)

Key observations:

1. All these packets are SYN packets (Flags [S]) from 10.62.2.83 (VM2) trying to connect to 10.62.2.82 (VM1) on port 4000.
2. There are three attempts (at 20:44:50, 20:44:51, and 20:44:53) from the same source port (35030) to the same destination.
3. The sequence number (seq 755547984) remains the same for all attempts, confirming it's the same connection attempt.
4. There are no response packets visible, which indicates that these SYN packets are being dropped by your iptables rule.

This output demonstrates that the client (VM2) is trying to establish a TCP connection to your server (VM1) on port 4000, but the connection is not being established. The client is making multiple attempts (TCP retransmissions) because it's not receiving any response. This behavior is consistent with the iptables rule that's dropping packets on port 4000.

**Step 11: Clean up**

To reset iptables to its default state on VM1:

sudo iptables -F

sudo iptables -X

sudo iptables -t nat -F

sudo iptables -t nat -X

sudo iptables -t mangle -F

sudo iptables -t mangle -X

sudo iptables -P INPUT ACCEPT

sudo iptables -P FORWARD ACCEPT

sudo iptables -P OUTPUT ACCEPT

Save the default state:

sudo iptables-save > /etc/iptables/rules.v4

**Conclusion**

This lab has demonstrated how to:

1. Set up basic iptables rules
2. Allow and then block traffic on a specific port (4000)
3. Use tcpdump to observe network traffic
4. Analyze tcpdump output to understand packet behavior

These skills are fundamental to network administration and security, allowing you to control traffic flow and diagnose network issues effectively.

**Additional Resources**

* [Iptables Documentation](https://netfilter.org/documentation/)
* [Tcpdump Manual](https://www.tcpdump.org/manpages/tcpdump.1.html)

**Troubleshooting**

If you encounter issues:

1. Verify VM connectivity
2. Check iptables rules with sudo iptables -L -v -n
3. Ensure you're using the correct IP addresses for your VMs
4. Check system logs for any error messages

Remember to always be cautious when modifying firewall rules, especially on production systems. It's easy to lock yourself out if you're not careful!