Please use a Screen or Video Capture software to save your works!

OBJECTIVE & PREPARATION

In this lab, you will learn how to use **iptables** to build a simple **Linux firewall** on your servers.

iptables is a very complex topic. Fortunately, you are not required to become an iptables expert, but by the end of the course, you should be able to use iptables to properly secure your servers.

Some basic iptables commands are provided in this lab for reference, but it is also essential that you know how to obtain help (man pages and online) in order to become self-reliant.

firewalld

In this course, we will be using iptables, **not** firewalld. Although firewalld can present information in a similiar iptables format, learning both would be too advanced at this point of learning Linux network administration.

You can also check the status of the firewalld service by issuing the command. systemctl status firewalld

You can also check if the firewalld service is running by issuing iptables -L and noting a high volume of unexpected output (i.e. "a strange result").

```
sudo -s          # all of our labs require you to have wheel access, it is better to switch to root.
# Disable firewalld if installed #
systemctl status firewalld.service
systemctl stop firewalld.service
systemctl disable firewalld.service
systemctl mask firewalld.service
```

install package to save iptables rules using the yum command

yum install iptables-services

systemctl enable iptables

systemctl start iptables

systemctl status iptables

reboot

Do the above steps on both **vm1** and **vm2**

now, run the iptables -L again, do you find it is much clean?

Critical iptables Elements

This may seem like another task to perform, but it is an essential task! You need to "become one" with basic iptables and focus on these important elements on this section, since you will be troubleshooting MANY connection issues with MANY VMs for labs and assignments! You need to become comfortable when using iptables to not only set policy, but troubleshoot and fix mistakes when you set your firewall policies!

The more you practice and get comfortable with iptables, the quicker you will be able to isolate and fix connection issues.

We don't expect you to become firewall experts, but there are some basics that you need to become familiar for this and future labs:

- What is a **chain**?

- **Which chain** applies to which traffic?
- What's the **default action** for a chain and when that applies?

```
[root@vm1 ops345]# iptables -L INPUT
Chain INPUT (policy ACCEPT)
```

- Understanding the differences between **setting policies**, **adding rules**, and **inserting rules**.
- In what **order are the rules executed**?
- **Reading and/or creating a rule** for a specific service. That includes a basic understanding of:
  - Protocols
  - Ports
  - Source/Destination IPADDR
  - HWADDR (MAC Address)
  - Network Interface name
- The best way to learn that is to practice.

Case 1: Preparation & Getting to Know iptables

Confirming Existing Network Connections

Before proceeding with iptables, we should first verify that your **vm1** and **vm2** can connect with each other. We can also take the opportunity to record some observations which could be used for future labs.

NOTE: You must **not** change any settings on the 10.0.0.* IPs on Azure. Unless otherwise specified, we will not use 10.0.0.* IPs throughout the semester.

vm1nic's IP address must be 192.168.0.10 through out the semester.

vm2nic's IP address must be 192.168.0.20 through out the semester.

Perform the Following Steps:

1. Determine the MAC address of the virtual network device on your servers and the IP addresses assigned to them. Start both of your vm1 and vm2 VMs. Record this information here:
   a. vm1 (Internet): MAC: _____ IP(s): _____.
   b. vm1nic (local): MAC: _____ IP(s): 192.168.0.10 _____.
   c. vm2 (Internet): MAC: _____ IP(s): _____.
   d. vm2nic (local): MAC: _____ IP(s): 192.168.0.20 _____.
2. SSH to vm1, open a terminal window, and perform the following connectivity tests to vm2.

   ping -c 1 192.168.0.20

   ssh 192.168.0.20

Default vs Updated Firewall Rules for VMs

You should have learned in OPS245 how to view existing iptables rules with a command similar to:

iptables -L -v

Although you may assume that this listing of rules should be empty, they may not be! In fact, several rules were **automatically added** to your chains.

Let's make a backup of the current iptables rules before we mess up!

cp /etc/sysconfig/iptables /etc/sysconfig/iptables.org

Practice Setting Firewall Rules on Server

We will run some iptables commands on your **vm1** to practice and get a basic understanding of how to set rules. We will NOT be saving the iptables rules in this section, so you don't have to worry about "messing-up" your server - you can simply reboot your server to load the default iptables rules.

The remaining iptables rules will relate to that same **inbound** traffic chain:

1. From the **vm1**, issue the command iptables -L INPUT, is there any ssh rule?
   ACCEPT tcp -- anywhere anywhere state NEW tcp dpt:ssh
2. Add a new ssh rule to accept the ssh from vm2: 192.168.0.20
   iptables -I INPUT -p tcp -s 192.168.0.20 --dport 22 -j ACCEPT
3. Delete the ssh rule from step 1.
   iptables -L INPUT --line-numbers
   # get the line number of the rule from step 1. In my case, it is #5
   iptables -D INPUT 5
   iptables -L INPUT

check if the rule removed successfully, and why you didn't get disconnected right away? Hint: rule #2
disconnect ssh and see if you can ssh again to the Public IP of vm1?

```
[root@vm1 ops345]# iptables -L INPUT --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     tcp  --  192.168.0.20         anywhere             tcp dpt:ssh
2    ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
3    ACCEPT     icmp --  anywhere             anywhere
4    ACCEPT     all  --  anywhere             anywhere
5    ACCEPT     tcp  --  anywhere             anywhere             state NEW tcp dpt:ssh
6    REJECT     all  --  anywhere             anywhere             reject-with icmp-host-prohibited
```

4. Confirm you can ssh to the vm2 through the Public IP of vm2.
   Are you able to ssh to vm1 from within vm2? Why you can? Hint: the rule you created on step 2.
   ssh 192.168.0.10
5. Add back the iptables rule to allow ssh
   iptables -I INPUT -p tcp --dport 22 -j ACCEPT
   # now check if you can ssh to the Public IP of vm1.
6. **Shut down your VMs** and reboot your server. What happens to the iptables rules you created for your server? How to save and restore your iptables rules, and what is **flushing iptables rules**?
   iptables-save > /etc/sysconfig/iptables

   iptables-save > filename
   iptables-restore < filename

   iptables -F INPUT
   iptables -F

Case 2: Best Practices & Customized Chains

In this case study, we will use shell scripting to help automate our firewalls, and create our own customized chains for packet filtering.

Best Practices for iptables

Refer to this "best practices" chart when using iptables:

Tip	Explanation
Always back-up the default iptables settings	When you install iptables in CentOS it already has some rules predefined. Make a copy of the file that creates these rules (including the ones that allow communication with your other machines). This way you can always restore them to have a functional machine even if you completely mess up your rules.
Place your iptables commands (i.e. Rules) within a Bash shell script	If you need to reset iptables, then you can run a shell script to quickly re-apply rules to save time.
Don't Panic if disconnected from a VM	Some of the traffic between your home computer and the server and client VM goes through IPtables. When you mess with iptables rules on the VMs, you might end up losing the console connection to the virtual machines. **Don't worry, the virtual machines are still running and you can still use them once you re-establish your connection. Talk to the professor if your connection doesn't re-establish within few minutes.**
If your most recent iptables Rule messes up your system	**Reload the default rules**. You can do that by restarting the iptables services (you can also do that at the beginning of your shell script). Then **run your script with all the working iptables commands** that you already finished. **Return to work on creating the rule that didn't work**.

Creating Customized Chains

You have the ability to create your own customized chains - you can actually name them!

The purpose of creating your own customized chains is to separate all the rules related to a single service (e.g. SSH, HTTP, FTP, ICMP, etc) from other unrelated rules.

We will now create a new chain in order to create rules just relating to the ssh service:

1. On vm1, create a new chain named **MYSSH** in the filter table.
   iptables -N MYSSH

2. Remove any SSH rules from INPUT. (refer to Case 1, step 3 on how to delete a rule).
   Note: in my case here, I will need to remove rule #1 and #2. Be careful, after you deleted rule #1, then rule #2 become rule #1 in the filter table. Therefore, if you start removal from the lowest number, the command you will run would be multiple times of iptables -D INPUT 1

```
[root@vm1 ops345]# iptables -L INPUT --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ssh
2    ACCEPT     tcp  --  192.168.0.20         anywhere            tcp dpt:ssh
3    ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED
4    ACCEPT     icmp --  anywhere             anywhere
5    ACCEPT     all  --  anywhere             anywhere
6    REJECT     all  --  anywhere             anywhere            reject-with icmp-host-prohibited
```

3. At this step, you should have a filter table rule like the following.

```
[root@vm1 ops345]# iptables -L INPUT --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
2    ACCEPT     icmp --  anywhere             anywhere
3    ACCEPT     all  --  anywhere             anywhere
4    REJECT     all  --  anywhere             anywhere             reject-with icmp-host-prohibited
```

4. Add a rule to the INPUT chain of your filter table that sends all ssh traffic to your MYSSH chain. Make sure this new rule follows (not proceeds) the RELATED,ESTABLISHED rule, so it doesn't apply to existing connections! (add the rule after rule #1, so we should INSERT the rule as #2)
   Note: Use --jump or -j (not -g or --goto) to move to a target.
   iptables -I INPUT 2 -p tcp --dport 22 -j MYSSH
   iptables -I INPUT 2 -p tcp --dport ssh -j MYSSH # use name instead of port number also works

```
[root@vm1 ops345]# iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
MYSSH      tcp  --  anywhere             anywhere             tcp dpt:ssh
ACCEPT     icmp --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere
REJECT     all  --  anywhere             anywhere             reject-with icmp-host-prohibited
```

5. Recall the step 1, we created an empty MYSSH chain. Now, add a rule to your MYSSH chain to accept all traffic on your virtual interface from vm1nic's subnet, 192.168.0.0/24 (i.e. your internal network).
   iptables -I MYSSH -s 192.168.0.0/24 -j ACCEPT
   # Note that how I add a subnet instead of a single IP address?
6. Add a rule to the end of the MYSSH chain to drop all remaining ssh connections, but to log these denied packets with log level 4 'info' and log prefix "DENIED BY MYSSH" before doing so.
   Note: you cannot put both DROP and LOG in the same iptables command.

LOG first and then DROP.

iptables -A MYSSH -p tcp --dport ssh -j LOG --log-level 4 --log-prefix "DENIED BY MYSSH"

iptables -A MYSSH -p tcp --dport ssh -j DROP

7. Save the iptables rules to make the change permanent.
   iptables-save > /etc/sysconfig/iptables
8. Restart iptables service. systemctl restart iptables
9. Issue iptables -L -v to view your newly created iptables rules remains. Then disconnect current ssh session and try to connect again.

If everything done correctly, you are not able to ssh from home, but able to ssh from vm2.

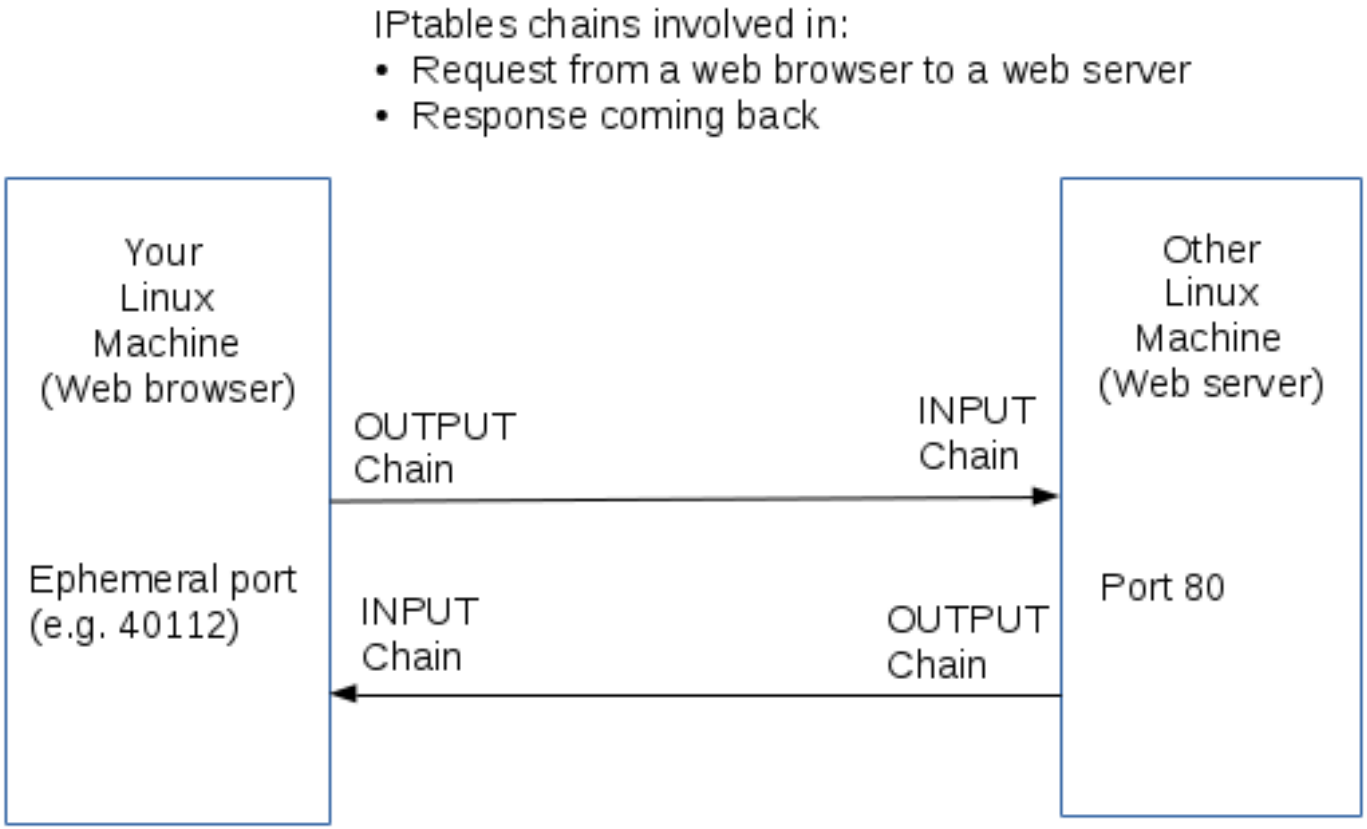
10. Issue journalctl –dmesg or tail /var/log/messages and check the last few lines, you should see something like this.

```
Sep 28 11:15:00 vm1 kernel: DENIED BY MYSSHIN=eth0 OUT= MAC=60:45:bd:c7:ba:dc:12:34:56:78:9a:bc:08:0
Sep 28 11:15:01 vm1 kernel: DENIED BY MYSSHIN=eth0 OUT= MAC=60:45:bd:c7:ba:dc:12:34:56:78:9a:bc:08:0
Sep 28 11:15:03 vm1 kernel: DENIED BY MYSSHIN=eth0 OUT= MAC=60:45:bd:c7:ba:dc:12:34:56:78:9a:bc:08:0
Sep 28 11:15:07 vm1 kernel: DENIED BY MYSSHIN=eth0 OUT= MAC=60:45:bd:c7:ba:dc:12:34:56:78:9a:bc:08:0
Sep 28 11:15:23 vm1 kernel: DENIED BY MYSSHIN=eth0 OUT= MAC=60:45:bd:c7:ba:dc:12:34:56:78:9a:bc:08:0
Sep 28 11:15:24 vm1 kernel: DENIED BY MYSSHIN=eth0 OUT= MAC=60:45:bd:c7:ba:dc:12:34:56:78:9a:bc:08:0
Sep 28 11:15:26 vm1 kernel: DENIED BY MYSSHIN=eth0 OUT= MAC=60:45:bd:c7:ba:dc:12:34:56:78:9a:bc:08:0
Sep 28 11:15:30 vm1 kernel: DENIED BY MYSSHIN=eth0 OUT= MAC=60:45:bd:c7:ba:dc:12:34:56:78:9a:bc:08:0
Sep 28 11:15:38 vm1 kernel: DENIED BY MYSSHIN=eth0 OUT= MAC=60:45:bd:c7:ba:dc:12:34:56:78:9a:bc:08:0
```

Case 3: How Firewalls (iptables) relate to the Labs in this Course

We will use an example of setting up a firewall to secure a web server. You will be installing, configuring, protecting, and maintaining a web-server of one of your VMs in a later lab.

The diagram displayed below shows how iptables can be used with a web-server:



There are some important things to be aware of in terms of this diagram:

1. There are **two sets** of iptables rules (chains) that apply: **OUTPUT/INPUT on the client** and **INPUT/OUTPUT on the server**.
    a. It is important to think about traffic from the perspective from the client as well as the server.
2. Outbound traffic is rarely blocked unless there is a security policy to prevent some kind of traffic.
    a. Even in that case, that security policy is usually performed on a router.
3. **Inbound traffic is of two distinct types**. Our diagram shows:
    a. **New incoming connections** (what you normally think of as **inbound traffic**): the web server receives a **new incoming connection**.
    b. **Incoming data that client receives as a response from the server**: the web page that the server sent back in the diagram above.
    c. The analogy would be like making a telephone call:
        i. A **NEW** packet is like the phone ringing
        ii. An **ESTABLISHED** packet is the connection and the packet say "hello", along with any further communication.
        iii. A **RELATED** packet would be the same person calling on a second line. (eg. a second connection that is made because of something that happened in the first, like an ftp transfer).
    d. We normally don't want to do anything special for the response. It is safe to assume that **a connection that was allowed to be established should be allowed to receive a response**. This is accomplished with the following **INPUT chain rule** that should be there by default on your machines:

```
iptables -L INPUT
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
```

```
[root@vm1 ops345]# iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
ACCEPT     icmp --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere
ACCEPT     tcp  --  anywhere             anywhere             state NEW tcp dpt:ssh
REJECT     all  --  anywhere             anywhere             reject-with icmp-host-prohibited
[root@vm1 ops345]#
```

4. **Rules are applied to: chains** (e.g. INPUT, OUTPUT) and contain information regarding the type of traffic they apply to. For example, **protocols** such as tcp/udp/icmp, **port numbers** such as 22 (SSH), 80 (HTTP), 443 (HTTPS), **addresses**, and many other things.
   a. Let's look at how these rules would apply to a simple web connection (HTTP - port 80):
   b. For the *request*, the **source port (sport) for the example in the above diagram is 40112** and the **destination port (dport) is 80**
   c. For the *response*, the **source port (sport) is 80** and the **destination port (dport) is 40112**
   d. Since the **RELATED,ESTABLISHED** rule already exists, we are only concerned about **controlling** the **incoming traffic on the server**, which in our example, the **chain is: INPUT**, the **protocol is: tcp**, and the **destination is: port 80**.
   e. If we standing on the webserver (vm2), iptables rule should be created as:
   iptables -I INPUT -p tcp -s 192.168.0.10 --dport 80 -j ACCEPT
   f. How do we know the above iptables rule works? On webserver vm2

```
yum install epel-release        # Install the EPEL repository
yum install nginx               # Install webserver nginx
systemctl start nginx           # make sure you start the nginx service
systemctl enable nginx          # optional, it only decides if the nginx service will be load at boot
```

   g. On the client (vm1)

```
yum install lynx                # Install browser lynx
lynx 192.168.0.20               # try to connect to webserver (vm2)
```

5. **Basically, most other services work in a similar way as discussed above**.
6. **Extra (optional)**: How do you connect to the webserver from the Internet port?

Save the captured file(s) as OPS345_Lab02_yourusername and upload to Blackboard.

If it is video recording, upload to OneDrive and share with jason.pang@senecacollege.ca