CSC 435/524 Computer Networks
Instructor: Dr. Natarajan Meghanathan
Spring 2014

Term Project - Choice # 2: Use of IPTables in a Virtual Machine Environment

Due: April 25, 2014 Max. Points: 100

This project is for educational and awareness purposes only. We are not responsible for anyone using this project for any malicious intent. The objective of this project is to educate students how to configure the different tables of IPtables in a virtual machine environment and use the various options to control incoming and outgoing communication from the Ubuntu virtual machine (VM) running on a Windows host machine. This project description includes a detailed tutorial on the configuration of IPtables covering different scenarios. You are then required to execute tasks to answer all the questions (including Question Q0) following the tutorial. You are strongly encouraged to go through the tutorial before attempting the questions.

You will need to download VMware Player which is the virtualization software that will be used for this project. You will also need a total of four virtual machines (one Ubuntu VM, one CentOS VM and two Backtrack VMs) running on the host Windows machine to complete this project. If you do not have sufficient resources to this project on your personal computer, you are advised to do it in the Computer Networks and Systems Security Lab in campus.

Submission Requirements

Hard copy: Include your answers for the questions Q0 through Q9 and the appropriate screenshots to justify each of your answers.

Video Recording: Record your explanation for each question Q0 through Q9 and demonstrate the steps you take to accomplish the tasks asked for in each of those questions. Try to record your responses together for all the questions in one single video file. If needed, you can record in multiple video files (but try to minimize the number of video files). Upload your video(s) through Dropbox or Google Drive and share them with me: natarajan.meghanathan@jsums.edu

Project Description Index

Installations	Page 2
IP Tables Tutorial	Page 5
IP Tables Exercises Q0 - Q8	Page 19

Installations

Installing VMWare Player

Download the latest version (v.5 or v.6) of VMware Player for your Operating System from https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/5_0

Installing Ubuntu OS

1. Download Ubuntu OS http://www.ubuntu.com/download/desktop and save it somewhere on your computer
2. Open up VMWare Player
3. Click on **Create a New Virtual Machine**
4. Select Installer disc image file (iso): browse for your Ubuntu .iso file and click **Next**
5. Type in your full name in the space provided. Use your J-number as Username (with a lowercase j). In my case, I use **natarajan** as the username. For your password, Select a password of your choice (easy to remember; but, difficult to find out by others). Click **Next** after entering the information.
6. Next, type in a name for your virtual machine (use your J-number again). Click **Next**.
7. On the next page, select **Store virtual disk as a single file**, and click **Next**.

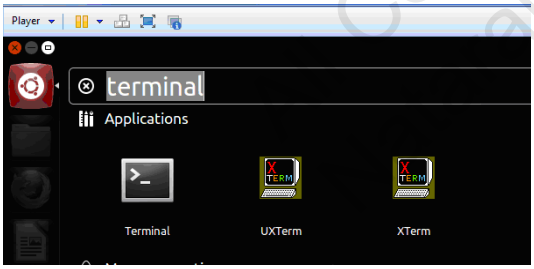
8. Click **Finish** on the next page and wait for the OS to be installed.

9. Next, log into Ubuntu OS with your password and press **Enter**.

10. Click the **Player** menu, and go to **Manage** then **Virtual Machine settings.**

11. When the settings come up, make sure that the **Network Adapter** is set to **NAT**, and click **OK**.

12. Launch a terminal by clicking the **Dash Home** (indicated in the picture below) and typing **terminal** in the box provided. Then click the **Terminal** icon.



Installing CentOS

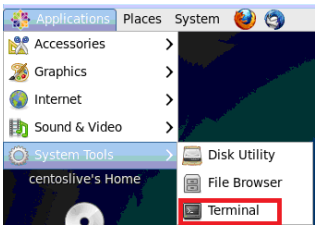
1. Download CentOS (CentOS-6.4-i386-LiveCD.iso) http://centos.icyboards.com/6.4/isos/i386/ and save it somewhere on your computer
2. Open up VMWare Player
3. Click on **Create a New Virtual Machine**
4. Select Installer disc image file (iso): browse for your CentOS .iso file and click **Next**
5. For Guest Operating System, choose Linux --> CentOS (**do not choose** CentOS 64-bit): we are using x86 version. Click **Next**. Give the VM - the name you want.
5. On the next page, select **Store virtual disk as a single file**, and click **Next**.

6. Click **Finish** on the next page.
7. Now Select CentOS from the VM Player menu and click **Play Virtual Machine.** Go through the OS installation process.
8. You can setup automatic login without requiring a password. If you wish to setup a password, you could also do so. You should be now logged into the CentOS system.

9. Click the **Player** menu, and go to **Manage** then **Virtual Machine settings.**

10. When the settings come up, make sure that the **Network Adapter** is set to **NAT**, and click **OK**.

11. Launch a terminal from the Applications --> System --> Terminal menu.



Installing Backtrack 5

1. Download **Backtrack 5** (**not** Backtrack 5 R1, R2, or R3) from
http://www.backtrack-linux.org/downloads/

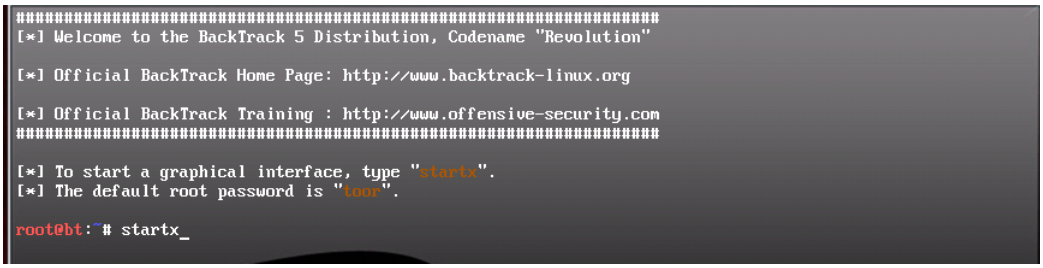
Download the GNOME 32-bit version .iso file, directly to a location in your physical host.
Then create a virtual machine instance of the Backtrack system on the VMWare Player. Choose the Guest Operating System to be Linux - Version: Other Linux 2.6.x kernel. Name the VM as **Backtrack-5.** You could set up the RAM to 512 MB or higher, as feasible for your host machine. The rest of the installation steps should be similar to that you went through for the CentOS VM.

2. When the VM starts, press enter in a black screen where is says **boot:** and press enter again to boot in text mode (the first option) when the Backtrack boot menu appears. If you are not already logged in as root, type in **root** for username and **toor** for password.

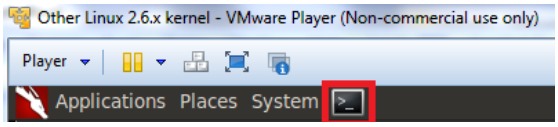
```
BackTrack 5 - Code Name Revolution 32 bitbt tty1
bt login: root
Password:
```

Note: You may need to press **Ctrl+Alt** when you need to bring your mouse pointer out of the Backtrack 5 virtual machine.

3. Type **startx** to launch the graphical interface.



4. You could launch a terminal by clicking the top > terminal icon.



6. Click the **Player** menu, and go to **Manage** then **Virtual Machine settings**.

7. When the settings come up, make sure that the **Network Adapter** is set to **NAT**, and click **OK**.

IPtables Tutorial

IPtables is a packet filter-based implementation of the Linux kernel firewall (netfilter). It defines tables that contain a chain of rules that specify how packets should be treated. The hierarchy is iptables --> tables --> chains --> rules. There may be built-in tables and chains as well as user-defined ones.

There are three independent tables (the presence of a table depends on the kernel configuration options): ***filter***, ***nat*** and ***mangle***. We specify the table to be used through the **-t** option.

- The ***filter*** table is the default table (if no -t option is used) and it has three built-in chains:
  INPUT (for packets destined for the local sockets);
  FORWARD (for packets being routed through a machine) and
  OUTPUT (packets originating from local sockets).
- The ***nat*** table is used when a packet encountered by the router/firewall has to go through network address translation. The nat table consists of three built-in chains:
  PRE-ROUTING - used to change the destination IP address of the incoming packets
  POST-ROUTING - used to change the source IP address of the outgoing packets
  OUTPUT - used to alter and send out the locally generated packets

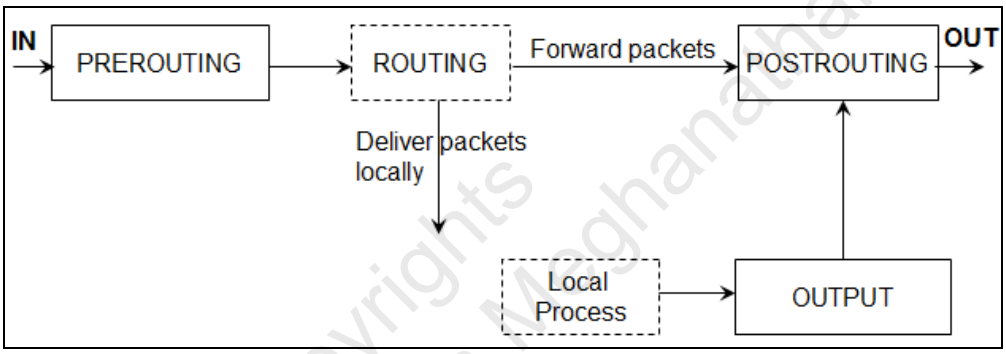


Figure 1: NAT Table

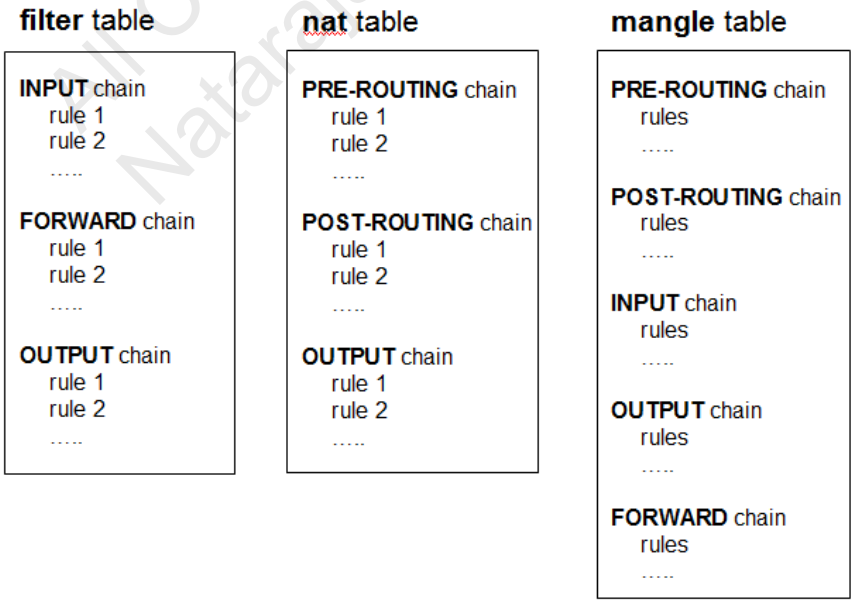


Figure 2: Tables and Chains of IPTables

- The ***mangle*** table is used to do some special alterations to the headers of packets that need some quality of service. Like the nat table, the mangle table has the pre-routing, post-routing and output chains (that have functionalities similar to those in the *nat* table) as well as input and forward chains (that have functionalities similar to those in the *filter* table).

A rule in a chain comprises of criteria and a target action.

Scenarios and IPTables commands

To change the contents or access the IPtables, one needs to have root access. Hence, I would suggest you login as root user. Otherwise, if you want to change/access the contents of IPtables as a regular user, you would have to prefix **sudo** upfront of every command as well as may be asked to enter the root password every time a command is run.

Assumption: Unless otherwise specified, for every scenario in this tutorial, all the chains are assumed to operate under a default-accept policy.

Validation Process: An incoming (or outgoing or transiting) packet is processed by the appropriate chain in the appropriate table (the filter table, by default). If a packet matches to the criteria in the chain, then the packet is subjected to the corresponding target action; otherwise, the packet is validated against the subsequent rules in the chain. If the packet cannot be matched with any of the criteria in the list, the packet is accepted (yes - the default policy for all chains of IPtables is to accept a packet, unless it matches to a criteria because of which the packet needs to be dropped).

S1: To list the contents of the *mangle* table of IPtables

Command: iptables -t mangle -L

As we see in the screenshot, the contents of the chains are empty and the default policy is ACCEPT. We will later see how to change this to DROP using the -P option (note it is uppercase 'P' for Policies and lowercase 'p' for ports).

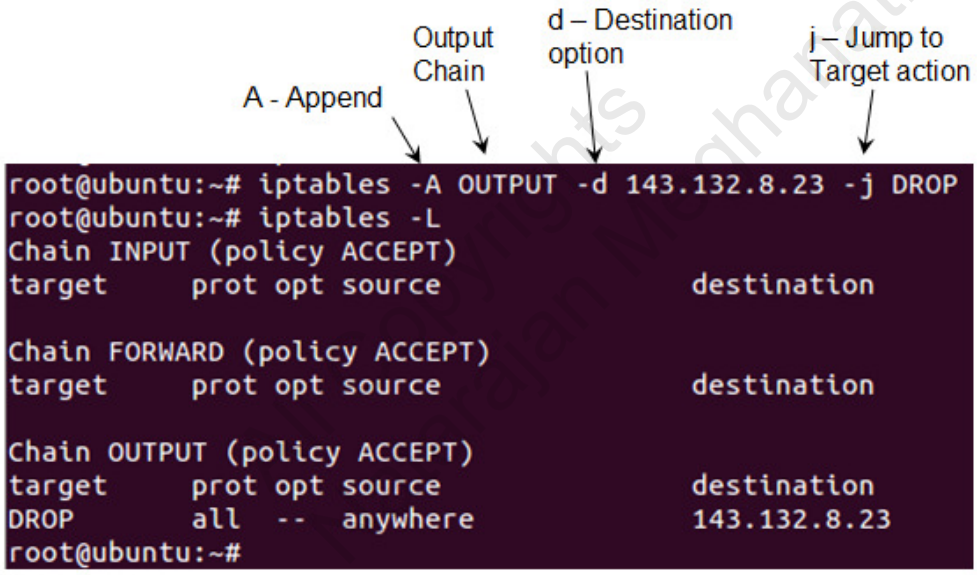
S2: To list the contents of the *filter* table of IPtables

We do not need to use the -t option when we want to access the *filter* table. If we run an iptables command without the -t option, the *filter* table will be processed by default. **Command: iptables -L**

```
root@ubuntu:/etc# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:/etc#
```

S3: To prevent a user on the local machine from visiting the Jackson State University web server whose IP address is 143.132.8.23.

Command: iptables -A OUTPUT -d 143.132.8.23 -j DROP



We could open a web browser (in your virtual machine) and try to visit www.google.com; we could visit without any problem. On the other hand, try to visit www.jsums.edu; you will only see a message on the browser telling "connecting to...," but it could not connect eventually.

S4: To delete all the entries in the IP tables/chains.

Command: iptables -F

This command will delete/flush all the entries in the filter iptable. If you want to delete all the entries in the nat table, you need to then run **iptables -t nat -F**.

IMPORTANT NOTE: Note that the flush operation does not reset the default-accept or drop policy of a chain. One has to manually change the default policy of a chain to the intended policy.

S5: Allow only SSH communications as incoming connection

If the objective is to allow only SSH communications as incoming connections, we could set the firewall to do this through two ways: In the first way, with the default policy being ACCEPT, the two rules are listed in this order: (i) Accept all incoming TCP packets coming to destination port 22 and (ii) Drop all other incoming packets (OR) In the second way, with the default policy changed to DROP, one can just setup a rule to accept all incoming TCP packets to destination port 22.

Method 1:

Commands (run in this order): Under a default-accept/allow policy, Once you have specified the rules to accept incoming an packet, it is better to specify a default rule to drop any incoming packets. Since rules are executed in numerical order, one after the other, starting from the first rule, the default rule to drop any incoming packets should be the last rule.

iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -i eth0 -j DROP

```
root@ubuntu:~# iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
root@ubuntu:~# iptables -A INPUT -i eth0 -j DROP
root@ubuntu:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ssh
DROP       all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:~#
```

One can test the rules from another virtual machine (as shown below) running on the same network.

```
[centoslive@livecd ~]$ ping 192.168.159.131
PING 192.168.159.131 (192.168.159.131) 56(84) bytes of data.
^C
--- 192.168.159.131 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5901ms

[centoslive@livecd ~]$ ssh natarajan@192.168.159.131
Welcome to Ubuntu 11.10
natarajan@192.168.159.131's password:
Welcome to Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic i686)
```

Method 2:

Commands (run in either order should be fine): Note that the uppercase 'P' denotes policy. We are changing the default input policy to DROP. That is, if an incoming packet does not match to any criteria corresponding to the rules in the INPUT chain, the packet will be dropped. This is the default-deny policy.

iptables -P INPUT DROP
iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT

```
root@ubuntu:~# iptables -P INPUT DROP
root@ubuntu:~# iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
root@ubuntu:~# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:~#
```

S6: Lets continue from Method 2 of Scenario 5, where we set the default policy for the INPUT chain to be DROP and configured the firewall to accept only incoming SSH connection requests. Lets first add a rule that would allow only hosts from a particular network (with prefix say 192.168.159.0/24) to send web traffic to the Linux host; all other traffic should be dropped. After configuring the above rule, lets delete the first rule to allow SSH packets.

```
root@ubuntu:~# iptables -A INPUT -i eth0 -p tcp -s 192.168.159.0/24 --dport 80 -
j ACCEPT
root@ubuntu:~# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
ACCEPT     tcp  --  192.168.159.0/24     anywhere             tcp dpt:www

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:~# iptables -D INPUT 1
root@ubuntu:~# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  192.168.159.0/24     anywhere             tcp dpt:www

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:~#
```

S7: Flush the contents of the iptables resulting at the end of Scenario 6. Configure the INPUT chain to default-accept policy (note that this has to be done manually; the flush operation wouldn't do this for us).

Lets say, by mistake, I then configured the firewall with a rule not to accept any incoming traffic. However, I realized later that I want to insert a rule that allowed the firewall to accept any incoming traffic to port 443 (https) and port 22 (ssh).

Commands (in this order): Note that everything following # is considered a comment.
iptables -F
iptables -P INPUT ACCEPT
iptables -A INPUT -j DROP
iptables -L # not needed if you do not want to see the contents of the iptables until now.
iptables -I INPUT 1 -p tcp --dport 443 -j ACCEPT
iptables -I INPUT 2 -p tcp --dport 22 -j ACCEPT

```
root@ubuntu:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:~# iptables -I INPUT 1 -p tcp --dport 443 -j ACCEPT
root@ubuntu:~# iptables -I INPUT 2 -p tcp --dport 22 -j ACCEPT
root@ubuntu:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:https
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
DROP       all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:~#
```

S8: With the configurations setup in Scenario 7, one can notice that we cannot still visit any website, because we need a domain name resolution to find the IP address of the web server whose domain name/website address is entered in the browser and we do not allow DNS traffic (port 53). Also, websites that use HTTPS need to be setup access for both ports 80 and 443. Websites that use only port 80 cannot be visited either as this port is not setup for ACCEPT in Scenario 7. In Scenario 8, we will basically do an enhanced implementation of Scenario 7 permitting packets from any website and drop all other incoming packets, including SSH, which we can test.

When it comes to permitting web traffic, we do not know what other protocols/ports and the corresponding packets need to be let in. So, it is better to insert a rule that lets packets that are related and/or as part of established sessions need to be permitted in. This could be done using the -m option. Note that -m option could be used in three contexts: to limit the number of times the rule has to match; multiport option (both of which we will see later) and the state of new, existing or related connections. We will the -m option for this scenario in the last context. The syntax to use the -m option in this context is **-m state --state ESTABLISHED,RELATED**. Note that there should not be blank space between words RELATED and ESTABLISHED.

Commands (the rule with the -m option can be either the first one or after the two tcp rules):
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -j DROP

```
root@ubuntu:~# iptables -A INPUT -p tcp --dport 80 -j ACCEPT
root@ubuntu:~# iptables -A INPUT -p tcp --dport 443 -j ACCEPT
root@ubuntu:~# iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
root@ubuntu:~# iptables -A INPUT -j DROP
root@ubuntu:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:www
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:https
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED
DROP       all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:~#
```

As we can now see, SSH connection to the Ubuntu host (192.168.159.131) is not accepted.

```
[centoslive@livecd ~]$ ssh natarajan@192.168.159.131
ssh: connect to host 192.168.159.131 port 22: Connection timed out
[centoslive@livecd ~]$
```

S9: We will configure the firewall to allow SSH, HTTP and HTTPS traffic and block any other protocol incoming traffic. We will do this using the multiport option.

The syntax is to use **-m multiport --dports 22,80,443** along with the rest of the parameters for the rule as indicated in the screenshot. Note that there should not be any blank space in between the port numbers.

```
root@ubuntu:~# iptables -A INPUT -p tcp -m multiport --dports 22,80,443 -j ACCEPT
root@ubuntu:~# iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
root@ubuntu:~# iptables -A INPUT -j DROP
root@ubuntu:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere            multiport dports ssh,www,https
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED
DROP       all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:~#
```

Now, SSH traffic is also allowed in.

S10: We want to limit the number of active connections to the web server running on port 80 to 3.

We will use the xt_connlimit module to limit the number of connections. To do so, we first add the xt_connlimit module to the Linux kernel using the modprobe program (a built-in program in Linux). We can then run the iptables **command** as follows:

iptables -A INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 2 -j DROP

where **--syn** indicates that we block the SYN request packets for a web connection
-m connlimit indicates we are using -m option for limiting the number of connections
--connlimit-above is an option to indicate when to take action; in this case, if the number of active connections is more than 2.

It is very important to limit the number of active connections for the servers running on a host/network; this would help to avoid a Denial of Service attack.

```
root@ubuntu:~# modprobe xt_connlimit
root@ubuntu:~# iptables -A INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 2 -j
 DROP
root@ubuntu:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere            tcp dpt:www flags:FIN,SYN,RST,A
CK/SYN #conn/32 > 2

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:~#
```

S11: We want to set the **nat** tables to forward a packet on a multi-hop (two-hop route).

We will use three VMs and datagram sender and receiver Java programs. The Ubuntu VM will serve as the destination and run the datagram receiver program; the Backtrack VM will serve as the source and run the datagram sender program; the CentOS VM will serve as the intermediate host and just forward the UDP datagram received from the Backtrack VM to the Ubuntu VM. The destination program at the Ubuntu VM runs on port 1234.

IP addresses	Machines	Role
192.168.159.131	Ubuntu VM	Destination
192.168.159.132	CentOS	Intermediate Forwarding Host
192.168.159.133	Backtrack VM	Source

Configuring the POSTROUTING chain at the sending host (Backtrack VM)
iptables -t nat -A POSTROUTING -p udp -d 192.168.159.131 --dport 1234 -j SNAT --to 192.168.159.132

Configuring the PREROUTING chain at the intermediate forwarding host (CentOS VM)
iptables -t nat -A PREROUTING -p udp -d 192.168.159.131 --dport 1234 -j DNAT --to 192.168.159.131:1234

Note:
You need to find out the ip addresses for your VMs by running the **ifconfig** command and use them accordingly.
We can see the contents of the nat tables by running the **iptables -t nat -L** command on the VMs.
You could install Java on the Ubuntu VM by running the **sudo apt-get install openjdk-6-jdk** command.

Rule of thumb: Use the POSTROUTING chain to configure forwarding at the source host as well as every intermediate host (except the intermediate host prior to the destination host, a.k.a. penultimate host)
Use the PREROUTING chain to configure forwarding at the intermediate host prior to the destination host (i.e., at the penultimate host)
There is no need to configure anything to receive at the destination host.

Receiver Java Program (start it first, run it on the destination - Ubuntu VM)

```
import java.net.*;
import java.io.*;

class datagramReceiver{
public static void main(String[] args){
 try{
  int MAX_LEN = 40;
  DatagramSocket mySocket = new DatagramSocket(Integer.parseInt(args[0]));
  byte[] buffer = new byte[MAX_LEN];
  DatagramPacket packet = new DatagramPacket(buffer, MAX_LEN);
  mySocket.receive(packet); // receiver is blocked here until it gets the message
  String message = new String(buffer);
  System.out.println(message);
  mySocket.close();
 }
catch(Exception e){e.printStackTrace();}
 }
}
```

Sender Java Program (start it later, run it on the source - Backtrack VM)

```
import java.net.*;
import java.io.*;

class datagramSender{
 public static void main(String[] args){

  try{
    InetAddress receiverHost = InetAddress.getByName(args[0]);
    int receiverPort = Integer.parseInt(args[1]);
    String message = args[2];
    DatagramSocket mySocket = new DatagramSocket();
    byte[] buffer = message.getBytes();
    DatagramPacket packet = new DatagramPacket(buffer, buffer.length, receiverHost, receiverPort);
    mySocket.send(packet);
    mySocket.close();
    System.out.println("sent packet...");
  }
  catch(Exception e){e.printStackTrace();}
  }
 }
```

S12: We want to set the **nat** tables to load balance incoming traffic to different ports.

We can emulate a sequential server as a multi-threaded server by running multiple instances of the sequential server at different ports and let the iptables to forward the incoming client requests (addressed to a common port) to various ports (at which the server instances run), one port at a time, and balance the load. The iptables firewall can be configured to fairly do this in a round-robin fashion so that the number of client requests assigned to the server instances does not differ more than 1. Note that there is still some non-determinism possible as the kernel may not schedule to run the different instances of the server process in a round-robin fashion. In other words, even though the iptables program would fairly forward the incoming packets to the different server instances, the server instances may not be run in a round-robin fashion by the kernel.

We will illustrate this scenario and the appropriate iptables command options using an example of a multiline client-server program wherein the client sends an integer representing the number of lines to be sent by the server and the server responds with that many lines, each identified by a monotonically increasing line number.

iptables commands

root@ubuntu:/home/natarajan# iptables -t nat -A PREROUTING -p tcp --dport 2000 -m state --state NEW -m statistic --mode nth --every 3 --packet 0 -j DNAT --to-destination 192.168.159.131:2001

root@ubuntu:/home/natarajan# iptables -t nat -A PREROUTING -p tcp --dport 2000 -m state --state NEW -m statistic --mode nth --every 3 --packet 1 -j DNAT --to-destination 192.168.159.131:2002

root@ubuntu:/home/natarajan# iptables -t nat -A PREROUTING -p tcp --dport 2000 -m state --state NEW -m statistic --mode nth --every 3 --packet 2 -j DNAT --to-destination 192.168.159.131:2003

root@ubuntu:/home/natarajan# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target prot opt source destination
DNAT tcp -- anywhere anywhere tcp dpt:cisco-sccp state NEW statistic mode nth every 3 to:192.168.159.131:2001
DNAT tcp -- anywhere anywhere tcp dpt:cisco-sccp state NEW statistic mode nth every 3 packet 1 to:192.168.159.131:2002
DNAT tcp -- anywhere anywhere tcp dpt:cisco-sccp state NEW statistic mode nth every 3 packet 2 to:192.168.159.131:2003

Chain INPUT (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination

Chain POSTROUTING (policy ACCEPT)
target prot opt source destination

Multiline Client and Server Programs: We will run three instances of the multiline server program at ports 2001, 2002 and 2003 on the Ubuntu VM (in my case, its IP address is 192.168.159.131). We will run three instances of the client program from the other VMs and the physical host machine, all trying to connect to the Ubuntu VM (192.168.159.131) at port 2000. The premise is the iptables firewall would

forward every incoming packet to the Ubuntu VM to one of the three ports 2001, 2002 and 2003 in a round-robin fashion.

Multiline server program

```
import java.io.*;
import java.net.*;

class multilineServer{

    public static void main(String[] args){

      try{

        int serverPort = Integer.parseInt(args[0]);
        ServerSocket lineServer = new ServerSocket(serverPort);

        while (true){

        Socket clientSocket = lineServer.accept();

        BufferedReader br = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        int count = Integer.parseInt(br.readLine());

        PrintStream ps = new PrintStream(clientSocket.getOutputStream());

        for (int ctr = 1; ctr <= count; ctr++){
          ps.println("Message # "+ctr);
          ps.flush();
          Thread.sleep(200);
        }

        clientSocket.close();

        }

       }
      catch(Exception e){e.printStackTrace();}

   }
}
```

Multiline client program

```
import java.io.*;
import java.net.*;

class multilineClient{

   public static void main(String[] args){

     try{

       InetAddress serverHost = InetAddress.getByName(args[0]);
       int serverPort = Integer.parseInt(args[1]);

       long startTime = System.currentTimeMillis();
```

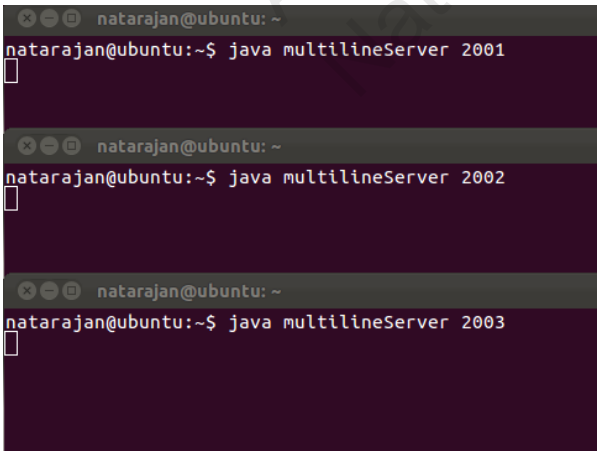
```
        Socket clientSocket = new Socket(serverHost, serverPort);

        int count = Integer.parseInt(args[2]);

        PrintStream ps = new PrintStream(clientSocket.getOutputStream());
        ps.println(count);

        BufferedReader br = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        String line = null;

        while ( (line = br.readLine() ) != null){
           System.out.println(line);
        }

        long endTime = System.currentTimeMillis();

        System.out.println(" Time to receive feedback from the server: "+(endTime-startTime)+" milliseconds");
        clientSocket.close();

        }
     catch(Exception e){e.printStackTrace();}

   }

}
```

To transfer the program: You may type the client and server program at the Ubuntu VM (in my case, its IP address is 192.168.159.131 and the name of the user account is natarajan), and assuming that you already running the SSH server on the Ubuntu VM, you can run the scp command to transfer a file from the Ubuntu VM to another Linux VM; you can transfer to the Windows host machine using the SSH Secure Shell Client application.

scp natarajan@192.168.159.131:multilineClientIterative.java .

Screenshots

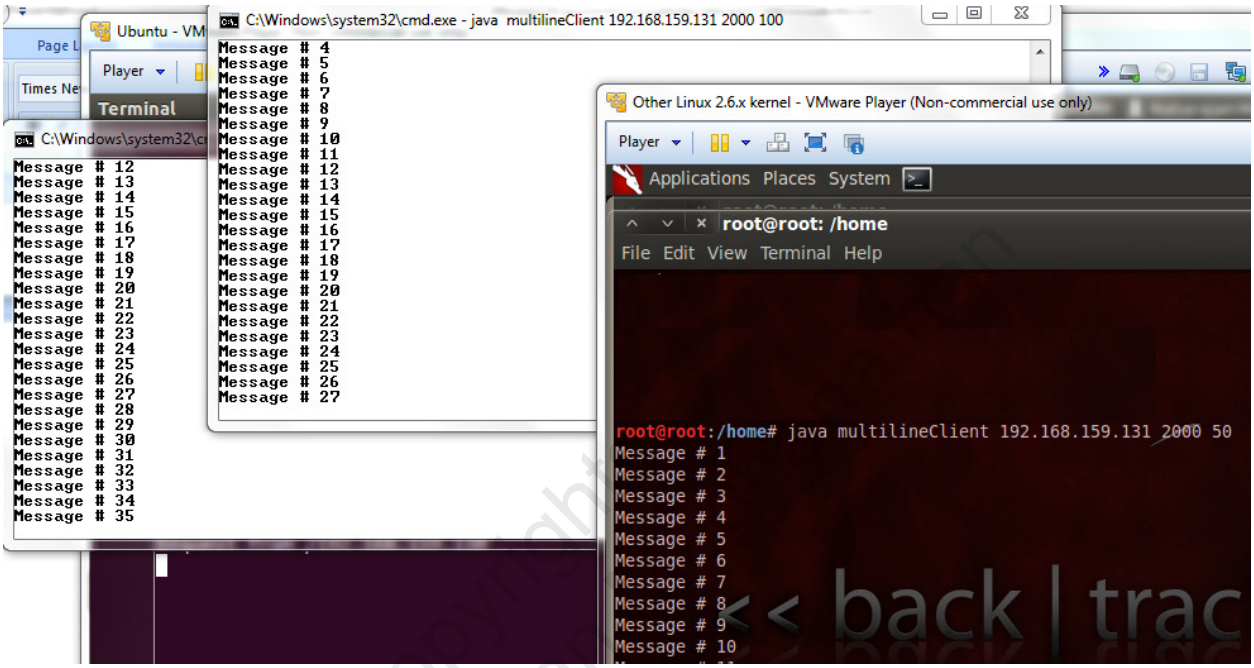
Ubuntu VM - server instances



Client side

```
root@root:/home# java multilineClient 192.168.159.131 2000 50
```

where multilineClient is the name of the Java class file
192.168.159.131 is the IP address at which the multilineServer program is running
2000 is the common port to which the incoming client requests go and then diverted to the different ports at which the server instances run
50 is the number of lines the client is requesting the server to send



S13: Setup iptables in such a way that we block ping testing from outside (i.e., remote machines cannot run a ping test on our machine) and at the same time we are able to ping remote machines.

We will setup our Ubuntu VM (192.168.159.131) to block ping test from other VMs (including our Backtrack VM, 192.168.159.130). To block a remote machine from doing a ping test on our machine, we should block the ICMP Echo-Request messages. Accordingly, we run the iptables command as follows:

root@ubuntu:/home/natarajan# **iptables -A INPUT -p icmp --icmp-type echo-request -j DROP**
root@ubuntu:/home/natarajan# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination
DROP icmp -- anywhere anywhere icmp echo-request

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination
root@ubuntu:/home/natarajan#

By explicitly specifying for the **echo-request** message to be dropped, we are permitting the **echo-reply** message by default (assuming the INPUT chain is configured by default to accept messages).

You can test the above setting by trying **ping 192.168.159.131** from your Backtrack VM to ping to the Ubuntu VM (of IP address 192.168.159.131). You will not be able to see any feedback. On the other hand, you could ping the Backtrack VM (192.168.159.130) from the Ubuntu VM by running **ping 192.168.159.130**.

IP Tables Project Exercises (all questions are with respect to the *filter* table, and are to be executed independent of the other questions, unless otherwise noted. So, remember to flush the iptables after executing each question, unless you are asked to follow up from a previous question):

Run the **iptables -L** command after setting up the configuration rules for each question.
Include screenshots of configuring the iptables firewall for every question.

Q0) Before you proceed with the questions on iptables, you need to install SSH on the Ubuntu VM in which you will be configuring the iptables. If you have already installed SSH on the VM, you could skip this question. Otherwise, complete the steps indicated in this question. You need to have the SSH server running on your Ubuntu VM to execute the steps in some of the questions in this project.

i) You need to setup root access in the Ubuntu VM. To setup root access, run the command **sudo passwd root** on the terminal. Enter the password you setup to login to the Ubuntu VM as a regular user (in my case, the username of the regular user is natarajan). Then, setup a password for the **root** level access and confirm it. The screenshot is shown below.

```
natarajan@ubuntu:~$ sudo passwd root
[sudo] password for natarajan:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
natarajan@ubuntu:~$
```

ii) Login as root using the command **su root**.

```
natarajan@ubuntu:~$ su root
Password:
root@ubuntu:/home/natarajan#
```

iii) Install the OpenSSH server application on the Ubuntu VM (run the command: **sudo apt-get install openssh-server**). After the installation is complete, run the **netstat –ntlp** command to show that the SSH daemon (sshd) is one of the programs actively running on a tcp port listening for incoming connection requests. Identify the port number on which the **sshd** daemon is running. Include appropriate screenshot(s).

Q1) Set the default policy for the INPUT chain to DROP. The firewall should only allow incoming packets from the network prefix 143.132.0.0/16. The default policy for the OUTPUT chain is ACCEPT. So, the user working on the machine could visit any website like www.google.com. Given the above policy for incoming packets, can the web pages visited by the user be displayed in the browser? Explain.

Q2) Set the default policy for the INPUT chain be DROP and the default policy for the OUTPUT chain be ACCEPT. Configure the INPUT chain to accept all incoming web traffic to port 80 and drop any other incoming traffic. Can you visit the website: **www.hotmail.com**? Why or why not? If you cannot visit the website, what aspect of this website is preventing you from visiting it, given that your default OUTPUT policy is ACCEPT and the firewall has been configured to accept traffic coming to port 80? Also, if you cannot visit the website, configure the firewall to let you be able to visit websites of such type. What changes/deletions/additions to the rules had to be done to facilitate this?

Q3) The previous question permitted only incoming packets related to web traffic. Do an insertion to the rules in the INPUT chain to permit SSH traffic. Show that you can connect to the SSH server running on the Ubuntu VM by connecting to it from another VM (centos or anything) or from the physical host machine (Windows). Include appropriate screenshots. You can get the IP address of a Linux machine by running the ifconfig command in the terminal. Refer to the screenshots (for example, under scenarios S5, S8) in the tutorial to see how you could SSH to a machine under a particular username.

Q4) Configure your IPtables filter table on your Ubuntu VM such that sessions/packet exchange originating from the Ubuntu VM (as the source) are successful; on the other hand, sessions/packet exchange originating from a remote machine to the Ubuntu VM (as the destination) are not successful. You need to implement this scenario with the minimal number of rules and policy changes, if any. Also, explain why your set of rules and policies implementing the stated scenario will work.

Q5) Configure your IPtables filter table to limit the number of active SSH connections to the Ubuntu VM (hosting the SSH server) is 2. Test the working of this rule by attempting to open three SSH connections, each in separate terminals, from another VM (like a CentOS VM) or from the host machine itself. Show appropriate screenshots.

Q6) Extend the two-hop packet routing scenario (scenario S11) to a three-hop scenario such that the packet gets routed through two intermediate hosts. You need to create and/or start the following four VMs on your physical host (Use VMware Player). Let the receiver runs port 1234. As can be seen below, you need to create two Backtrack VMs (easiest to create among the different VMs):

UbuntuVM - destination host Run the receiver datagram Java program @ port 1234
Backtrack VM-1 - source host Run the sender datagram Java program
CentOS VM - intermediate host 1
Backtrack VM-2 - intermediate host 2 (a.k.a. penultimate host)

The connectivity between the VMs is as follows:

Backtrack VM -1 ---------> Cent OS VM ----------> Backtrack VM-2 --------> Ubuntu VM
(source host) (intermediate host 1) (intermediate host 2) (destination host)

(a) Clearly show the IP addresses of the four VMs involved.
(b) Show screenshots of the source and destination hosts running the Java programs, sending and receiving packet.
(c) Show screenshots of the commands to configure the iptables and the results of the iptables -L command at each of the VMs, as applicable.
(d) Run the Wireshark application on the penultimate host and show the source and destination IP addresses and the source and destination port numbers as seen in the IP header and UDP header.
To launch Wireshark on Backtrack-VM, just type **wireshark** at the root prompt of the Backtrack terminal and press enter.

78) Consider the sequential (iterative) versions of the Summation Client and Server Java programs given below. Using IPtables, simulate a scenario wherein incoming client requests (to sum from 1, 2, up to a *count* value) are handled by one of three instances of the summation server program in a round-robin fashion. The clients requests to port 1234 and the IPtables is configured to forward the request coming to port 1234 to three different ports, 2234, 2235 and 2236, on each of which we run an instance of the summation server program. The Ubuntu VM, hosting the three instances of the summation server program, is configured with the required IPtables command to do load balancing. The summation client instances could run on the Backtrack VMs and the host machine.

Iterative Summation Server Program

```
import java.io.*;
import java.net.*;

class summationServer{

    public static void main(String[] args){

     try{

        int serverPort = Integer.parseInt(args[0]);
        ServerSocket calcServer = new ServerSocket(serverPort);

        while (true){

        Socket clientSocket = calcServer.accept();

        BufferedReader br = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        int count = Integer.parseInt(br.readLine());

        int sum = 0;
        for (int ctr = 1; ctr <= count; ctr++){
          sum += ctr;
          Thread.sleep(200);
        }

        PrintStream ps = new PrintStream(clientSocket.getOutputStream());
        ps.println(sum);
        ps.flush();

        clientSocket.close();

        }

       }
     catch(Exception e){e.printStackTrace();}

   }
}
```

Iterative Summation Client Program

```
import java.io.*;
import java.net.*;

class summationClient{

   public static void main(String[] args){

     try{

       InetAddress serverHost = InetAddress.getByName(args[0]);
       int serverPort = Integer.parseInt(args[1]);

       long startTime = System.currentTimeMillis();

       Socket clientSocket = new Socket(serverHost, serverPort);
```

```
        int count = Integer.parseInt(args[2]);

        PrintStream ps = new PrintStream(clientSocket.getOutputStream());

        ps.println(count);

        BufferedReader br = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

        int sum = Integer.parseInt(br.readLine());

        System.out.println(" sum = "+sum);

        long endTime = System.currentTimeMillis();

        System.out.println(" Time to receive feedback from the server: "+(endTime-startTime)+" milliseconds");
        clientSocket.close();

        }
     catch(Exception e){e.printStackTrace();}

    }

}
```

Q8) Set the default policy of the INPUT and OUTPUT chains of your filter table of iptables is to DROP using an appropriate command (show a screenshot executing the command and the output of the iptables -L command). You could use the Ubuntu VM and Backtrack VM in your virtual environment to implement this scenario. Now configure your iptables on the Ubuntu VM to (do parts a and b independently):
(a) Only allow remote machines to ping the local machine and block the local machine from pinging others.
(b) Only allow the local machine to ping the remote machines and block the remote machines from pinging the local machine.
(c) Allow ping communication in both directions (from the local machine to remote machine and vice-versa).

Note that you have to use the **--icmp-type echo-request** and **--icmp-type echo-reply** options appropriately.

Show appropriate screenshots executing the iptables commands to realize the above for (a), (b) and (c) and the structure of the iptables. Also, capture the successful or unsuccessful execution of the ping command from the local machine and remote machine (in either direction) for each of the three cases (a), (b), (c).