# PV204 – FIREWALLS

*iptables* are the tables provided by the Linux kernel firewall (implemented as different Netfilter modules) and the chains and rules it stores. Different kernel modules and programs are currently used for different protocols; *iptables* applies to IPv4, *ip6tables* to IPv6, *arptables* to ARP, and *ebtables* to Ethernet frames. For more details see: http://en.wikipedia.org/wiki/Iptables

In REHL or Fedora you can start/stop it by:

```
# service iptables stop
# service iptables start
# service iptables restart
```

Firewall status can be inspected by:

```
# iptables -n -L -v --line-numbers
```

```
Input/output can be listed by:
```

```
# iptables -L INPUT -n -v
# iptables -L OUTPUT -n -v --line-numbers
```

You can use the iptables command itself to stop the firewall, delete all rules and set default policy to accept:

```
# iptables -F
# iptables -X
# iptables -t nat -F
# iptables -t nat -X
# iptables -t mangle -F
# iptables -t mangle -X
# iptables -P INPUT ACCEPT
# iptables -P OUTPUT ACCEPT
# iptables -P FORWARD ACCEPT
```

Where,

**-F :** Deleting (flushing) all the rules.

**-X :** Delete chain.

**-t** table_name : Select table (nat or mangle) and delete/flush rules (default table is filter).

**-P :** Set the default policy (such as DROP or ACCEPT).

You can easily delete a chain rules – this example deletes the rule on line 4 (in the default table filter):

```
# iptables -D INPUT 4
```

Find source IP 202.54.1.1 and delete from rule:

```
# iptables -D INPUT -s 202.54.1.1 -j DROP
```

Where,

**-D** : Delete one or more rules from the selected chain

Save and restore firewall rules to/from a file called `/etc/iptables.rules`

```
# iptables-save > /etc/iptables.rules
# iptables-restore < /etc/iptables.rules
# service iptables restart
```

In some distributions you must put a line into `/etc/rc.local` to automatically load all firewall chains.

To insert a new rule between lines 1 and 2 into INPUT chain, enter:

```
# iptables -I INPUT 2 -s 202.54.1.2 -j DROP
```

All iptables chains have a default policy setting. If a packet doesn't match any of the rules in a relevant chain, it will match the default policy and will be handled according the default policy.

There are 3(4) **basic tables** with predefined chains in iptables:

1. *Filter table* – Filter is default table for iptables. It is where the bulk of the work in an iptables firewall occurs. Avoid filtering in any other table as it may not work. So, if you don't define you own table, you'll be using filter table.
2. *NAT table* – The Network Address Translation or nat table is used to translate the source or destination field in packets. A system with a static IP should use Source Network Address Translation (snat) since it uses fewer system resources. However, iptables also supports hosts with a dynamic connection to the Internet with a masquerade feature. Masquerade uses the current address on the interface for address translation.
3. *Mangle table* – Mangle table is for specialized packet alteration. It can be used to change the Time to Live or TTL, change the Type of Service or TOS field, or mark packets for later filtering.

These are 3 predefined chains in the **filter table** to which we can add rules for processing IP packets passing through those chains. These chains are:

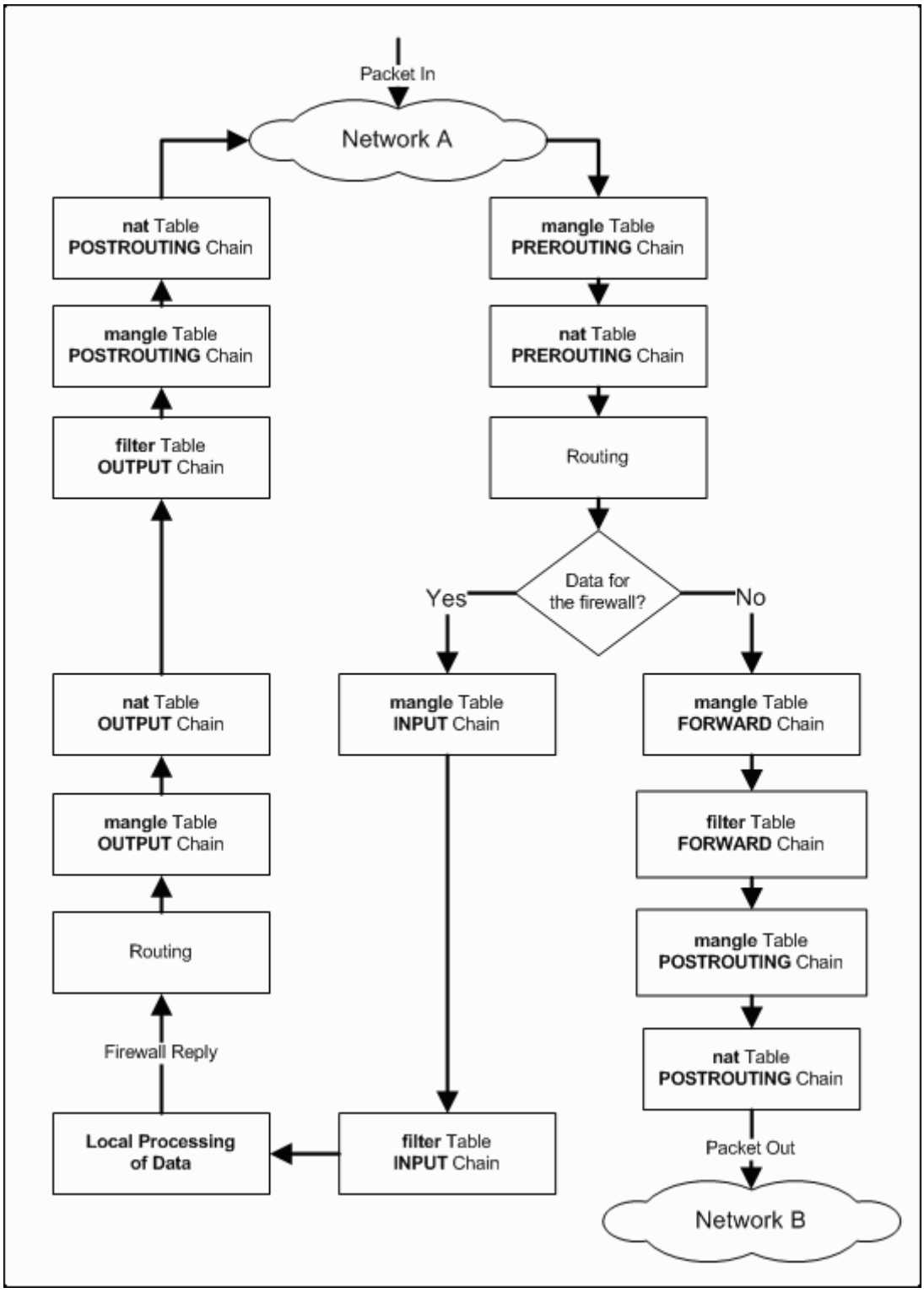
- INPUT - All packets destined for the host computer.
- OUTPUT - All packets originating from the host computer.
- FORWARD - All packets neither destined for nor originating from the host computer, but passing through (routed by) the host computer. This chain is used if you are using your computer as a router.

For the most time, we are going to be dealing with the INPUT chain to filter packets entering our machine.

Rules are added in a list to each chain. A packet is checked against each rule in turn, starting at the top, and if it matches that rule, then an action is taken such as accepting (ACCEPT) or dropping (DROP) the packet. Once a rule has been matched and an action taken, then the packet is processed according to the outcome of that rule and isn't processed by further rules in the chain. If a packet

passes down through all the rules in the chain and reaches the bottom without being matched against any rule, then the default action for that chain is taken. This is referred to as the default policy and may be set to either ACCEPT or DROP the packet.

Iptables packet flow diagram:



**Matching packets (basic examples)**

We need to be able to clearly define which packets we want to block and which we want to allow through.

*Address matching*

The two most basic match conditions are:

1. source address of the packet
2. destination address of the packet

Note: These can either be individual IP addresses or a whole subnet.

If we wanted to block packets heading to 172.25.0.1 from anything on the 10.0.0.0/8 network, we would do:

```
# iptables -A INPUT -s 10.0.0.0/8 -d 172.25.0.1 -j DROP
```

*Protocol matching*

We can also match based on protocol used, (TCP, UDP, ICMP, etc.), as well as the specific port or service type used by that protocol. As an example, a common usage is to block connections to port 113 via TCP, which is used by identd:

```
# iptables -A INPUT -p tcp --dport 113 -j REJECT --reject-with tcp-reset
```

Note: The tcp-reset REJECT option causes the client to reset the TCP connection to our system.

We can mix the protocol and source or destination address into one whole rule:

```
# iptables -I INPUT -p tcp --dport 113 -s 10.0.0.0/8 -j ACCEPT
```

*State matching*

We can also specific a 'match' option, using the **-m** flag. This allows us to use a kernel module to provide extra packet matching capabilities, the most popular usage of which is for connection tracking matching.

The 'state' match has four different types of connection which we can match against:

1. *ESTABLISHED:* corresponds to a connection which is already up and running. If the connection originated within our network, as soon as the packet passes through our firewall on its way to the Internet, it is tracked as ESTABLISHED.
2. *RELATED:* is provided by a protocol helper module. The most common use for this is with FTP by using the ip_conntrack_ftp.o module, which allows us to track FTP connections back into our network properly, as when we download from a FTP server, it will try to make a TCP connection back to our system.

3. *NEW:* means that the packet is part of a new connection, meaning that it has not yet been tracked by the connection tracking system.
4. *INVALID:* means that the connection is in an invalid state, so generally these should be dropped.

As a basic rule, we want to allow all ESTABLISHED and RELATED packets into our network, and selectively allow NEW packets through depending on the destination port.

```
# iptables -A INPUT -m state --state INVALID -j DROP
# iptables -A INPUT -m state --state NEW -j DROP
# iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j DROP
# iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
```

**Matching packets (more tips and examples)**

*Remember localhost*

Lots of applications require access to the lo interface. Ensure that you set up your rules carefully so that the lo interface is not disturbed:

```
# iptables -A INPUT -i lo -j ACCEPT
```

*Be stringent with your rules*

Try to make your rules as specific as possible for your needs. For example, I like to allow ICMP pings on my servers so that I can run network tests against them. I could easily toss a rule into my INPUT chain that looks like this:

```
# iptables -A INPUT -p icmp -m icmp -j ACCEPT
```

However, I don't want to simply allow all ICMP traffic. There have been some ICMP flaws from time to time and I'd rather keep as low of a profile as possible. There are many types of ICMP control messages, but I only want to allow echo requests:

```
# iptables -A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
```

This will allow echo requests (standard ICMP pings), but it won't explicitly allow any other ICMP traffic to pass through the firewall.

*Always comment strange rules*

```
-m comment --comment "limit ssh access"
```

*Drop Private Network Address On Public Interface*

IP spoofing is nothing but to stop the following IPv4 address ranges for private networks on your public interfaces. Packets with non-routable source addresses should be rejected using the following syntax:

```
# iptables -A INPUT -i eth1 -s 192.168.0.0/24 -j DROP
# iptables -A INPUT -i eth1 -s 10.0.0.0/8 -j DROP
```

*Own chain (here named MYCHAIN) and jumping to them*

```
# iptables -N MYCHAIN
# iptables -A MYCHAIN -m limit --limit 5/h --limit-burst 3 -j LOG --log-
prefix "Blacklist: "
# iptables -A MYCHAIN -j DROP
# iptables -A INPUT -i eth0 -s 192.168.0.2 -j MYCHAIN
```

*Multiple ports and addresses*

```
# iptables -A INPUT -p tcp --dport 50:55 -m iprange --dst-range
192.168.0.1-192.168.0.10 -j ACCEPT
```

*Time restrictions*

```
# iptables -A INPUT -m time --timestart 8:00 --timestop 18:00 --days
Mon,Tue,Wed,Thu,Fri -j ACCEPT
```

*Logging(syslog: /var/log/syslog or /var/log/messages) of dropped packets*

```
# iptables -A INPUT -i eth1 -s 10.0.0.0/8 -j LOG --log-prefix "IP_SPOOF A:"
# iptables -A INPUT -i eth1 -s 10.0.0.0/8 -j DROP
```

*Trusted MAC addresses*

First we use `-m mac` to load the mac module and then we use `--mac-source` to specify the mac address of the source IP address (192.168.0.4).

```
# iptables -A INPUT -s 192.168.0.4 -m mac --mac-source
00:70:FD:D1:E1:24 -j ACCEPT
```

*SNAT*

If the server has static IP 192.168.0.1 and wals to share Internet to other users:

```
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 192.168.0.1
```

If the server uses DHCP it is solved by MASQUERADE parameter:

```
# echo "1" > /proc/sys/net/ipv4/ip_forward
# iptables -A FORWARD -i eth1 -o eth0 -s 192.168.0.22 -j ACCEPT
# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Note: forwarding in kernel must be turned on and FORWARD chain in iptables must also allow this forwarding (if the default policy do not allows it).

Note: the exact opposite is DNAT (e.g., server has two IPs and wants just one to be visible for internal network).

*NAT table and port redirecting*

```
# iptables -t nat -A PREROUTING -i eth1 –p tcp --dport 80 -j REDIRECT --to
3128
```

*Mangle table and setting of TTL*

```
# iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-set 64
```

**Some external links**

Basic introduction: http://iptablesguru.blogspot.cz/

For examples see: *http://www.cyberciti.biz/tips/linux-iptables-examples.html*

Some best practices: *http://rackerhacker.com/2010/04/12/best-practices-iptables/*

Mini how-to: *http://www.linode.com/wiki/index.php/Netfilter_IPTables_Mini_Howto*

Other examples: *http://wiki.centos.org/HowTos/Network/IPTables*