LINUX FIREWALL with IPTABLES

What Is iptables

- **The most popular firewall / NAT package running on Linux was ipchains**. It had a number of limitations, the primary one being that it ran as a separate program and not as part of the kernel. The **Netfilter** organization decided to create a new product called **iptables** in order to rectify this shortcoming. As a result of this, **iptables is considered a faster and more secure alternative. iptables has now become the default firewall package installed under RedHat and Fedora Linux.**
- The Linux kernel has the built-in ability to filter packets, allowing some of them into the system while stopping others. The 2.4 kernel's netfilter has **three built-in *tables*** or *rules lists*. They are as follows:
  - **filter** :The default table for handling network packets.
  - **nat** : Used to alter packets that create a new connection.
  - **mangle** : Used for specific types of packet alteration.

Packets Processing By The Firewall

Queue Type	Queue Function	chain in Queue	Chain Function
Filter	Packet filtering	FORWARD	Filters packets to servers accessible by another NIC on the firewall.
		INPUT	Filters packets destined to the firewall.
		OUTPUT	Filters packets originating from the firewall
Mangle	TCP header modification	PREROUTING POSTROUTING OUTPUT INPUT FORWARD	Modification of the TCP packet quality of service bits before routing occurs (Rarely used in SOHO environments)

Packets Processing By The Firewall

Queue Type	Queue Function	Chain in Queue	Chain Function
Nat	Network Address Translation	PREROUTING	Address translation occurs before routing. Used with NAT of the destination IP address, also known as destination NAT or DNAT.
		POSTROUTING	Address translation occurs after routing. This implies that there was no need to modify the destination IP address of the packet as in pre-routing. Used with NAT of the source IP address using either one-to-one or many-to-one NAT. This is known as source NAT, or SNAT.
		OUTPUT	Network address translation for packets generated by the firewall. (Rarely used in SOHO environments)

What Is iptables

- The built-in chains for the **filter** table are as follows:
  - ***INPUT*** — Applies to network packets that are targeted for the host.
  - ***OUTPUT*** — Applies to locally-generated network packets.
  - ***FORWARD*** — Applies to network packets routed through the host.
- The built-in chains for the **nat** table are as follows:
  - ***PREROUTING*** — Alters network packets when they arrive.
  - ***OUTPUT*** — Alters locally-generated network packets before they are sent out.
  - ***POSTROUTING*** — Alters network packets before they are sent out.

What Is iptables

- The built-in chains for the **mangle** table are as follows:
  - ***INPUT*** — Alters network packets targeted for the host.
  - ***OUTPUT*** — Alters locally-generated network packets before they are sent out.
  - ***FORWARD*** — Alters network packets routed through the host.
  - ***PREROUTING*** — Alters incoming network packets before they are routed.
  - ***POSTROUTING*** — Alters network packets before they are sent out.

Download – install – start the IPTABLES

- The latest version of the RPM for Fedora Core 1 is iptables-1.2.9-1.0.i386.rpm. Install the package using the following command:

```
[root@MAIL tmp]# rpm -Uvh iptables-1.2.9-1.0.i386.rpm
Preparing...    ########################################### [100%]
1:iptables      ########################################### [100%]
[root@MAIL tmp]#
```

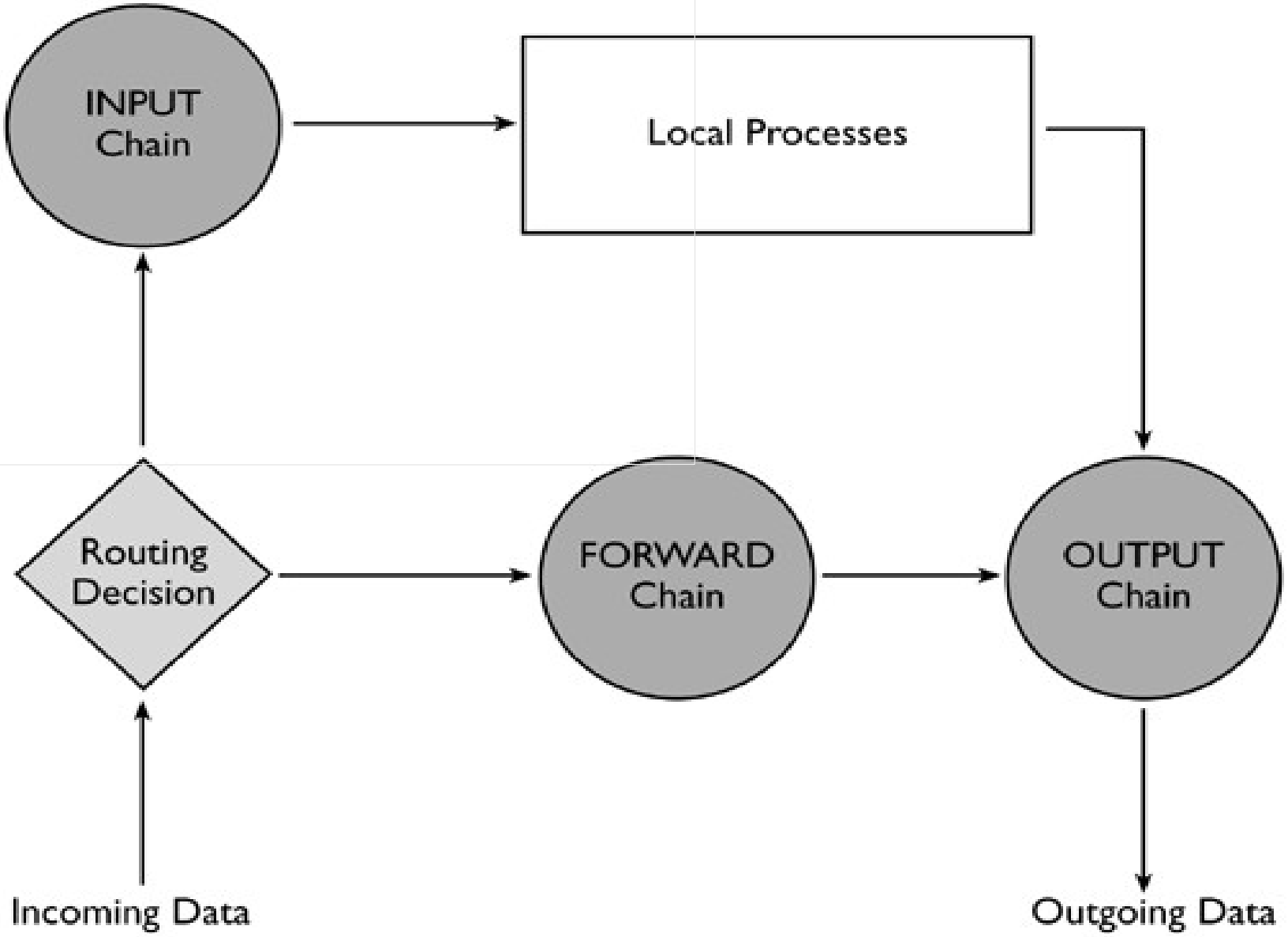
- You can start/stop/restart iptables after booting by using the following commands:

```
[root@MAIL tmp]# /etc/init.d/iptables start
[root@MAIL tmp]# /etc/init.d/iptables stop
[root@MAIL tmp]# /etc/init.d/iptables restart
```

- To get iptables configured to start at boot:

```
[root@MAIL tmp]# chkconfig --level 345 iptables on
```

Packet Processing In iptables



Packet Processing

Options Used in iptables Commands

- Rules that allow packets to be filtered by the kernel are put in place by running the iptables command. When using the iptables command, specify the following options:
  - ***Packet Type*** : Dictates what type of packets the command filters.
  - ***Packet Source/Destination*** : Dictates which packets the command filters based on the source or destination of the packet.
  - ***Target*** : Dictates what action is taken on packets matching the above criteria.
- The options used with given iptables rule must be grouped logically, based on the purpose and conditions of the overall rule, in order for the rule to be valid.

Structure

- Many iptables commands have the following structure:

#iptables [-t <*table-name*>]
<***command***> <***chain-name***>
<***parameter-1***> <***option-1***>
<***parameter-n***> <***option-n***>

- <table-name> : allows the user to select a table other than the default *filter* table to use with the command.
- <command> : dictates a specific action to perform, such as appending or deleting the rule specified by the <chain-name> option.
- <chain-name> :are pairs of parameters and options that define what will happen when a packet matches the rule

iptables -A INPUT -s 0/0 -i eth0 -d 192.168.1.1 -p TCP -j ACCEPT

Commands

- **–A** : Appends the iptables rule to the end of the specified chain.
- **–F** : Flushes the selected chain, which effectively deletes every rule in the the chain. If no chain is specified, this command flushes every rule from every chain
- **–L** : Lists all of the rules in the chain specified after the command. To list all rules in all chains in the default filter table, do not specify a chain or table. Otherwise, the following syntax should be used to list the rules in a specific chain in a particular table:

  iptables –L <chain-name> –t <table-name>
- **–N** : Creates a new chain with a user-specified name.
- **–P** : Sets the default policy for a particular chain, so that when packets traverse an entire chain without matching a rule, they will be sent on to a particular target, such as ACCEPT or DROP.

  Iptables –P INPUT DROP

The Most Commonly Used Targets

Target	Description	Most common options
ACCEPT	> `iptables` stops further processing. > The packet is handed over to the end application or the operating system for processing	N/A
DROP	> `iptables` stops further processing. > The packet is blocked	N/A
REJECT	> Works like the DROP target, but will also return an error message to the host sending the packet that the packet was blocked	--reject-with *qualifier* Qualifiers include: icmp-port-unreachable (default) icmp-net-unreachable icmp-host-unreachable

The Most Commonly Used Targets

Target	Description	Most common options
DNAT	> Used to do destination network address translation. ie. rewriting the destination IP address of the packet	--to-destination *ipaddress* Tells iptables what the destination IP address should be
SNAT	> Used to do source network address translation rewriting the source IP address of the packet > The source IP address is user defined	--to-source *<address>[-<address>][:<port>-<port>]* Specifies the source IP address and ports to be used by SNAT.
MASQUERADE	> Used to do Source Network Address Translation. > By default the source IP address is the same as that used by the firewall's interface	*[--to-ports <port>[-<port>]]* Specifies the range of source ports to which the original source port can be mapped.

General Iptables Match Criteria

Iptables command	Description
-t <table>	If you don't specify a table, then the filter table is assumed. As discussed before, the possible built-in tables include: filter, nat, mangle
-j <target>	Jump to the specified target chain when the packet matches the current rule.
-p <protocol-type>	Match protocol. Types include, icmp, tcp, udp, and all
-s <ip-address>	Match source IP address
-d <ip-address>	Match destination IP address
-i <interface-name>	Match "input" interface on which the packet enters.
-o <interface-name>	Match "output" interface on which the packet exits

Parameters

- **-d : Sets the destination** hostname, IP address, or network of a packet that will match the rule. When matching a network, the following IP address/netmask formats are supported:
  - ***N.N.N.N/M.M.M.M*** — Where ***N.N.N.N*** is the IP address range and ***M.M.M.M*** is the netmask.
  - ***N.N.N.N/M*** — Where ***N.N.N.N*** is the IP address range and ***M*** is the netmask length.
- **-s** : **Sets the source** for a particular packet using the same syntax as the destination (-d) parameter.
- **-f** : **Applies this rule only to fragmented packets**. By using the ! option after this parameter, only unfragmented packets will be matched.
- **-p** — **Sets the IP protocol for the rule**, which can be either **icmp**, **tcp**, **udp**, or **all**, to match every supported protocol. In addition, any protocols listed in /etc/protocols may also be used. If this option is omitted when creating a rule, the all option is the default.

Parameters

- **-i** : **Sets the incoming network interface**, such as **eth0** or **ppp0**. With iptables, this optional parameter may only be used with the **INPUT** and **FORWARD** chains when used with the filter table and the **PREROUTING** chain with the nat and mangle tables.
- This parameter also supports the following special options:
  - **!** : Tells this parameter not to match, meaning that any specified interfaces are specifically excluded from this rule.
  - **+** : A wildcard character used to match all interfaces which match a particular string. For example, the parameter **-i eth+** would apply this rule to any Ethernet interfaces but exclude any other interfaces, such as **ppp0**.
- If the **-i** parameter is used but no interface is specified, then every interface is affected by the rule.

Parameters

- **-j** :Tells iptables to **jump to a particular target** when a packet matches a particular rule. Valid targets to be used after the -j option are: **ACCEPT**, **DROP**, **LOG**, **REJECT** ….
- You may also direct a packet matching this rule to a user-defined chain outside of the current chain so that other rules can be applied to the packet.
- If no target is specified, the packet moves past the rule with no action taken. However, the counter for this rule is still increased by one, as the packet matched the specified rule.
- **-o** : **Sets the outgoing network interface** for a rule and may only be used with **OUTPUT** and **FORWARD** chains in the filter table, and the **POSTROUTING** chain in the nat and mangle tables. This parameter's options are the same as those of the incoming network interface parameter **(-i**).

Common TCP Match Criteria

Switch	Description
`-p tcp --sport <port>`	TCP source port Can be a single value or a range in the format: *start-port-number:end-port-number*
`-p tcp --dport <port>`	TCP destination port Can be a single value or a range in the format: *starting-port:ending-port*
`-p tcp --syn`	Used to identify a new TCP connection request

Common UDP Match Criteria

Switch	Description
`-p udp --sport <port>`	UDP source port Can be a single value or a range in the format: *starting-port:ending-port*
`-p udp --dport <port>`	UDP destination port Can be a single value or a range in the format: *starting-port:ending-port*

iptables -A FORWARD -s 0/0 -i eth0 -d 192.168.1.58 -o eth1 **-p TCP --sport** 1024:65535 **--dport** 80 -j ACCEPT

TCP Protocol match options (-p tcp ...)

- **--dport** : Sets the destination port for the packet. To browse the names and aliases of network services and the port numbers they use, view the /**etc/services** file.
- To specify a specific range of port numbers, separate the two numbers with a colon (**:**), such as **-p tcp --dport 3000:3200**. The largest acceptable valid range is **0:65535**.
- Use an exclamation point character (**!**) after the **--dport** option to tell iptables to match all packets which do not use that network service or port.

UDP Protocol match options (-p udp ...)

- **--sport** : Sets the source port of the packet using the same options as --dport.
- **--syn** : Applies to all **TCP** packets designed to initiate communication, commonly called **SYN** packets. Any packets that carry a data payload are not touched. Placing an exclamation point character (**!**) as a flag after the **--syn** option causes all non-SYN packets to be matched.
  - **--dport** — Specifies the destination port of the UDP packet.
  - **--sport** — Specifies the source port of the UDP packet.

ICMP Protocol match options (-p icmp)

- These match options are available for the Internet Control Message Protocol (ICMP) (-p icmp):
- **--icmp-type <type>** : Sets the name or number of the **ICMP** type to match with the rule

iptables -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT

iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT

- In this example iptables is being configured to allow the firewall send **ICMP echo-requests** (pings) and in turn, **accept** the expected **ICMP echo-replies**.

Common Extended Match Criteria

Switch	Description
`-m multiport --port <port, port>`	A variety of TCP/UDP source ports separated by commas. Unlike when `-m` isn't used, they do not have to be within a range.
`-m multiport --dport <port, port>`	A variety of TCP/UDP destination ports separated by commas. Unlike when `-m` isn't used, they do not have to be within a range.
`-m multiport --ports <port, port>`	A variety of TCP/UDP ports separated by commas. Source and destination ports are assumed to be the same and they do not have to be within a range.

iptables -A FORWARD -s 0/0 -i eth0 -d 192.168.1.58 -o eth1 -p TCP --sport 1024:65535 **-m multiport --dport 80,443** -j ACCEPT

Common Extended Match Criteria

Switch	Description
`-m --state <state>`	The most frequently tested states are: ESTABLISHED: The packet is part of a connection that has seen packets in both directions NEW: The packet is the start of a new connection RELATED: The packet is starting a new secondary connection. This is a common feature of such protocols such as an FTP data transfer, or an ICMP error. INVALID: The packet couldn't be identified. Could be due to insufficient system resources, or ICMP errors that don't match an existing data flow.

iptables -A FORWARD -d 0/0 -o eth0 -s 192.168.1.58 -i eth1 -p TCP **-m state --state ESTABLISHED** -j ACCEPT

Modules with Additional Match Options

- To use a match option module, load the module by name using the **-m** option, such as **-m <*module-name*>** (replacing <***module-name***> with the name of the module).
- state module: The **state module (-m state --state** … match a packet with the following connection states:
  - **ESTABLISHED** — The packet is associated with other packets in an established connection.
  - **INVALID** — The packet can't be matched to a known connection, and thus may be forged .
  - **NEW** — The packet is trying to establish a new connection .
  - **RELATED** — The packet is not part of an existing connection, but it's related, such as an **ICMP** error packet .

  These connection states can be used in combination with one another by separating them with commas, such as **-m state --state INVALID,NEW**.
- **mac module** — **(-m mac** …) Enables hardware **MAC** address matching with the following option:
  - **--mac-source** — Matches a **MAC** address of the network interface card that sent the packet. To exclude a **MAC** address from a rule, place an exclamation point (!) after the **--mac-source** match option.

Loading Kernel Modules Needed By iptables

- The iptables application requires you to load certain kernel modules to activate some of its functions. Whenever any type of NAT is required, the iptable_nat module needs to be loaded. The ip_conntrack_ftp module needs to be added for FTP support and should always be loaded with the ip_conntrack module which tracks TCP connection states. As most scripts probably will keep track of connection states, the ip_conntrack module will be needed in any case. The ip_nat_ftp module also needs to be loaded for FTP servers behind a NAT firewall.

- ```
  # File: /etc/rc.local
  # Module to track the state of connections
        modprobe     ip_conntrack
  # Load the iptables active FTP module, requires ip_conntrack
        modprobe ip_conntrack_ftp
  # Load iptables NAT module when required
        modprobe     iptable_nat
  # Module required for active an FTP server using NAT
  modprobe ip_nat_ftp
  ```
  ```

Example: Allowing DNS Access To Firewall

```
#-----------------------------------------------------
# Allow outbound DNS queries from the FW and
 the replies too
# Interface eth0 is the internet interface
# Zone transfers use TCP and not UDP. Most home
 networks
# websites using a single DNS server won't
 require TCP statements
#-----------------------------------------------------
iptables -A OUTPUT -p udp -o eth0
 --dport 53 --sport 1024:65535 -j ACCEPT


iptables -A INPUT -p udp -i eth0
 --sport 53 --dport 1024:65535 -j ACCEPT
```

Allowing WWW And SSH Access To Firewall

```
#-----------------------------------------------------
# Allow previously established connections
# - Interface eth0 is the internet interface
#-----------------------------------------------------

iptables -A OUTPUT -o eth0 -m state --state
 ESTABLISHED,RELATED -j ACCEPT
#-----------------------------------------------------
# Allow port 80 (www) and 22 (SSH) connections to the firewall
#-----------------------------------------------------


iptables -A INPUT -p tcp -i eth0 --dport 22
 --sport 1024:65535 -m state --state NEW
 -j ACCEPT
iptables -A INPUT -p tcp -i eth0 --dport 80
 --sport 1024:65535 -m state --state NEW
 -j ACCEPT
```

Allowing Firewall To Access The Internet

Allow port 80 (www) and 443 (https) connections from the firewall

```
iptables -A OUTPUT -j ACCEPT -m state --state
 NEW,ESTABLISHED,RELATED -o eth0 -p tcp -m
 multiport --dport 80,443 -m multiport --sport
 1024:65535
```

Allow previously established connections

- Interface eth0 is the internet interface

```
iptables -A INPUT -j ACCEPT -m state --state
 ESTABLISHED,RELATED -i eth0 -p tcp
```

If you want all TCP traffic originating from the firewall to be accepted, then remove the line:

```
-m multiport --dport 80,443 -m multiport
 --sport 1024:65535
```

IPTABLES Summary

- **Flush the rules from a chain:**
- **# iptables -F INPUT**
  # iptables -F OUTPUT
  # iptables -F FORWARD
- **Setting a Default Policy (-P or --policy)**
- **# iptables -P INPUT DROP**
  # iptables -P OUTPUT DROP
  # iptables -P FORWARD DROP
- **Creating Firewall Rules**
- **# iptables --append *CHAIN selection-criteria* --jump {DROP|ACCEPT|REJECT}**
  # iptables -A *CHAIN selection-criteria* -j {DROP|ACCEPT|REJECT}
- **Opening and Closing Specific Ports**
- **# iptables -A INPUT -p tcp --dport 25 -j ACCEPT**
  # iptables -A OUTPUT -p tcp --sport 25 -j ACCEPT

IPTABLES Summary

- **Using Source and Destination IP Addresses**
- **# iptables -A INPUT -s 172.24.0.0/16 -j DROP**
  # iptables -A OUTPUT -d 172.24.0.0/16 -j DROP
- **Filtering by Interface**
- **# iptables -A INPUT -s 192.168.9.0/24 -i eth0 -j DROP**
  # iptables -A FORWARD -s 192.168.9.0/24 -i eth0 -j DROP
  # iptables -A FORWARD -s !192.168.9.0/24 -i eth1 -j DROP
  # iptables -A OUTPUT -s !192.168.9.0/24 -i eth1 -j DROP
- **Performing Stateful Inspection**
- **# iptables -A INPUT -m state -p tcp --dport 80 **
  --state NEW,ESTABLISHED,RELATED -j ACCEPT
  **# iptables -A OUTPUT -m state -p tcp --sport 80 **
  --state ESTABLISHED,RELATED -j ACCEPT

Masquerading (Many to One NAT)

- NAT allows a router to modify the contents of TCP/IP packets. In particular, **NAT enables changes to the source and destination addresses of the packets. Masquerading is another word for what many call "many to aone" NAT**. In other words, traffic from all devices on one or more protected networks will appear as if it originated from a single IP address on the Internet side of the firewall
- **iptables** requires the **iptables_nat** module to be loaded with the "**modprobe**" command for the masquerade feature to work. Masquerading also depends on the Linux operating system being configured to support routing between the internet and private network interfaces of the firewall. This is done by enabling "**IP forwarding**" or routing by giving the file /**proc/sys/net/ipv4/ip_forward** the value "**1**" as opposed to the default disabled value of "**0**".

Example of Masquerading

```
# Load the NAT module
modprobe iptable_nat
# Enable routing by modifying the ip_forward
# - Interface eth0 is the internet interface
# - Interface eth1 is the private network interface

echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -A POSTROUTING -t nat -o eth0 -s \
192.168.1.0/24 -d 0/0 -j MASQUERADE
# Prior to masquerading, the packets are routed via the filter
# table's FORWARD chain.
# Allowed outbound: New, established and related
connections
# Allowed inbound : Established and related connections
iptables -A FORWARD -t filter -i eth1 -m state \
        - - state NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -t filter -i eth0 -m state \
        - state ESTABLISHED,RELATED     -j ACCEPT
```

Forwarding Ports with iptables

- In many cases home users may get a single public IP address from their ISP. If their Linux firewall is their interface to the Internet and they want to host a website on one of the NAT protected home servers then they will have to use the "port forwarding" technique.
- Port forwarding is handled by the **PREROUTING** chain of the "**nat**" table. As in masquerading, the **iptables_nat** module will have to be loaded and routing enabled for port forwarding to work. Routing too will have to be allowed in iptables with the **FORWARD** chain, this would include all **NEW** inbound connections from the Internet matching the port forwarding port plus all future packets related to the **ESTABLISHED** connection in both directions.

Forwarding Ports with iptables

Port forwarding can be an extremely useful tool in certain situations, including the following:

- **You move a server from one computer to another**, but your DNS entries have not yet been properly updated. You can also use port forwarding to *temporarily* make such a change.
- **You want a server to respond to multiple ports** on a single computer. You can set up the system to forward one port to another on the same system. Some servers can listen directly to multiple ports, though, which is often a simpler approach.
- **You want to run an externally accessible server within a NAT- protected network**. You can set up the NAT router to forward traffic directed at one of its external ports to an internal system.

Example of port forwarding

Load the NAT module and Enable routing

modprobe iptable_nat

echo 1 > /proc/sys/net/ipv4/ip_forward

external_int="eth0"

external_ip="203.132.168.1"

Allow port forwarding for traffic destined to port 80 of the firewall's IP address to be forwarded to port 8080 on server 192.168.1.200. Interface eth0 is the internet interface Interface eth1 is the private network interface

iptables -t nat -A PREROUTING -p tcp -i eth0 -d $external_ip --dport 80 --sport 1024:65535 -j DNAT --to 192.168.1.200:8080

Example of port forwarding


```
# After DNAT, the packets are routed via the filter's
FORWARD chain.
# Connections on port 80 to the target machine on
the private  network # must be allowed.
iptables -A FORWARD -p tcp -i eth0 -o eth1 -d \
192.168.1.200 --dport 8080 --sport 1024:65535 -m \
state --state NEW -j ACCEPT
iptables -A FORWARD -t filter -i eth1 -m state \
        --state NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -t filter -i eth0 -m state \
         --state ESTABLISHED,RELATED -j ACCEPT
```

Static NAT

- See doc. file