**IPTABLES**

* Layer 1: Physical Layer
* Layer 2: Data Link Layer
* Layer 3: Network Layer
* Layer 4: Transport Layer
* Layer 5: Session Layer
* Layer 6: Presentation Layer
* Layer 7: Application Layer

**some of the Protocols in each Layer are given below.**  
**7. Application Layer**  
*NNTP  · SIP  · SSI  · DNS  · FTP  · Gopher  · HTTP  · NFS  · NTP  · SMPP  · SMTP  · SNMP  · Telnet (more)*  
**6. Presentation Layer**  
*MIME  · XDR  · SSL  · TLS*  
**5. Session Layer**  
*Named Pipes  · NetBIOS  · SAP*  
**4. Transport Layer**  
*TCP  · UDP  · PPTP  · L2TP  · SCTP*  
**3. Network Layer**  
*IP  · ICMP  · IPsec  · IGMP*  
**2. Data Link Layer**  
*ARP  · CSLIP  · SLIP  · Frame relay  · ITU-T G.hn DLL*  
**1. Physical Layer**  
*RS-232  · V.35  · V.34  · I.430  · I.431  · T1  · E1  · Ethernet  · POTS  · SONET  · DSL  · 802.11a/b/g/n PHY  · ITU-T G.hn PHY*  
      
    IPTables is a front end user space tool to manage Netfilter in Linux kernel. IPTables functions primarily in the Transport (Layer4) and Network (Layer 3), even it can work in the DataLink layer too. IPTables can manage the ICMP .

**Layer 4 -Transport-** Focuses on Protocols & Ports (TCP/UDP & Ports(0-65535)). The ports are based on 16bit value  
**Layer 3 -Network-** Focuses on Source & Destination (IP Address). The IP address is based on 32 bit value

**Installing IPTables**  
         The package IPTables will be installed by default in most of the Linux distro.

***# rpm -qa |grep -i IPTables***

    Or download the Latest package of IPTables from [http://www.netfiler.org](http://www.netfiler.org/)

**Checking the kernel for the support of the IPTables.**

Find the area for “NETFILTER” in Kernel config file.

***# uname -a***  
 ***# vim /boot/config-***  
 *CONFIG\_NETFILTER=y*

    (y)This means the netfilter basic support has been integrated and compiled to the kernel.If (m) option is defined then this means the module can be loaded on the fly so here we need to check the iptables modules has been loaded by command “lsmod”.

**Default Tables & Chains in IPTables**  
    There are 3 default tables which cannot be deleted. Each table contains chains and the rules are written to the chains  
**1. Mangle**  
    This allows to alter packets eg:- Type Of Service, Time To Live etc.  
**2. NAT**  
    Network Address Translation, This allows to change IP Address & Ports. Eg:- Source NAT / DST NAT etc  
**3. Filter**  
    Here we perform the Filtering the traffic (INPUT, OUTPUT & FORWARD). It works between Layer 3 & Layer 4.

**Rule Syntax IPTables.**

***# /sbin/iptables***  
commands are used in the following syntax:  
*name of chain – action done to chain (Append/Incert or Replace)  
    name of table – default it will append to filter table  
    Layer 3 object – src or dst of ip address  
    Layer 4 object – protocols & ports   
    Jump/Target – if the above criteria meets the do this action*  
 **Example of iptables**  
Drop All the packages from a Host

***# iptables -A INPUT -t filter -s 192.168.1.233    -j DROP***

    This will Drop all the packages coming from the source 192.168.1.233.  
Now Test by pinging to the destination host 192.168.1.233  
     Here we have the OUTPUT chain opened and the rule is defined in INPUT chain. This means our system is able to send the packages to the destination and while the destination machines replies back we drop the packets.

**Saving and Restoring the rules in IPTables**

***# iptables-save***

    This will dump the rules to STDOUT(to the terminal). The output will be in the iptables default format.

***# iptables-save > firewall-rules***

    This will write the rule the file firewall-rules

***# iptables-restore***

    Default reads the rule from STDIN and loads in to the kernel.

***# iptables-restore < firewall-rules***

**IPTables Chain Management**  
**Listing all the chains in Table Filter**

***# iptables -L -n***

    It lists INPUT, FORWARD and OUTPUT chains and rules associated with. Each of the chains will have a default policy. i.e the default policy is accept the traffic in IPTables.

***# iptables -L OUTPUT***

    It will list all the rules in the chain OUTPUT for the default table.

**Listing all the chains in Table NAT.**

***# iptables -L -n -t nat***

     It contains the chains PREROUTING(will use NAT before routing occurred -destination nat-),  POSTROUTING(uses NAT to after the packets get routed  -source nat-) & OUTPUT (Reserved for packets that sourced locally that need the NAT)

**Listing all the chains in Table Mangle.**

***# iptables -L -n -t mangle***

    It contains chains INPUT,OUTPUT,FORWARD, PREROUTE & POSTROUTE. Mangle Table is the ANDing of Filter & NAT table.

**To List the amount of traffic that processed by the a chain**

***# iptables -L -n -v -t filter***

    This will show the total amount of traffic in each chains. Even if there is no rule defined it shows the traffic in chains. This is because there is default rule of accept all in IPTables.

**Determine the Line number of the rule in a chain**

***# iptables -L -n -v –line-numbers  
# iptables -L -n -v –line-numbers -t nat***

    This will show the line numbers column for all the chains.

**Appending(-A) and Inserting(-I) rules to Chains**

    We will  try to understand each chains with a real time scenario  
*Source (192.168.1.1)pings to -> destination (192.168.254)*  
    In this case the source sends a ICMP (echo-request) packet to 192.168.254 which pass across the OUTPUT chain in filter table. Once the request reaches the destination it responds with a echo-reply to the source 192.168.1.1 which pass across the INPUT chain in filter table.

Now we will create rule in source all traffic for SSH will be permitted and Telnet traffic will be denied  
**Appending a rule(-A):**

***# iptables -A INPUT -p tcp –dport 22 -j ACCEPT  
# iptables -A INPUT -p tcp –dport 23 -j DROP***

    This will make the server to accept only the ssh based connection and telnet sessions wll be droped.   
The append (-A) will add the rule to the last rule in the chain(to the end of the rule list in chain).

***# iptables -L -n -v –line-number***

    This will list the newly added rule

**Inserting a rule (-I):**  
We can insert the rule into a particular line number using this option. (Keep in mind the iptable checks the rule from above to bottom and once it matches the criteria it executes the rule).(We can even insert a same rule to the chain, creating a duplicate rule. IPTables doesn’t have a feature to detect the duplicate rules that have appended or inserted.

***# iptables -I INPUT 1 -p tcp –dport 23 -j DROP  
# iptables -L -n -v –line-number***

    Here we can see that the rule for dropping the telnet session has been added to first line in the chain. So IPTables will process the rule number 1 before it hitting the rule number 2.  
Other examples:

***# iptables -I INPUT 2 -p tcp –dport 21 -j DROP***

    It inserts a rule to line number 2 in INPUT Chain for the table Filter,for Dropping all FTP traffic.

**Deleting(-D) and Replacing(-R) Rules**  
**Deleting a Rule:**  
Syntax for deleting the rule from the chain:  
***# iptables -D   
# iptables -D***  
**Type 1**

***# iptables -L -n -v –line-number  
# iptables -D INPUT 2***

    This will delete the 2nd rule in the chain INPUT.  
**Type 2**

***# iptables -D INPUT -p tcp –dport 21 -j DROP***

    This will delete the rule as mentioned . This need the exact match and in case of any duplicate rules the first match will be deleted.

**Replacing Rules:**  
Syntax:  
***# iptables -R***

***# iptables -L -n -v –line-number  
# iptables -R INPUT 1 -p tcp –dport 23 -j ACCEPT***

    This will replace the existing rule from DROP to ACCEPT (we had previously denied the telnet access)

**Flush(-F) rules and Zero counters (-Z)**

**Flush rules:**  
syntax:  
***# iptables -F***

***# iptables -F INPUT***

    This will flush all the rules in the chain INPUT.

***# iptables -F***

    This will flush all the rules from all the chains in default Table. But the flusing will not zero the packet counters (iptables -L -v).

**Zero Counters:**  
Syntax:  
***# iptables -Z***

***# iptables -Z INPUT  
# iptables -L -n -v***

    This will reset the packet count for the chain INPUT

***# iptables -Z***

    This will reset all the chain packet counts in default table.

***# iptables -Z POSTROUTING -t nat***

    This will reset the packet counter for the chain POSTROUTING for table nat.

**User Defined Tables/Chains (Creating (-N) and Renaming (-E old new)):**  
    IPTables ships with 3 default tables which cannot be deleted.

**Creating a New chain called INTRANET**

***# iptables -N INTRANET  
# iptables -L -n***

    This will create a new chain called INTRANET in the filter table. This will create a chain with the default refereces as “0”. reference is the link towards the default chains(INPUT, OUTPUT & FORWARD).  
Now we define the new chain INTRANET how to behave. i.e, which traffic should be this chain responsilbe for.

***# iptables -R INPUT 1 -s 192.168.1.0/24 -j INTRANET***

    This will tell IPTables that – In rule number 1, any trafic having the source network ID 192.168.1.0 should be contacted the chain INTRANET

***# iptables -L -n***

    Here we can see that a new entry for the Chain is added into the Line number 1 stating that for all the packages having source address in the network 192.168.1.0/24 should jump to target chain INTRANET.

**Now create the rules**

***# iptables -A INTRANET    -p tcp –dport 23 -j DROP***

    So when a packet comes with a source address in 192.168.1.0/24 with the destination port 23. The iptables will refer from the INPUT chain to the INTRANET chain and Then IPTables will start matching the rule. If the packet has the destitantion port 23 then it will DROP.

Note:-  
   *User defined chains must have unique names. Because it function has the target (-j).*

**Rename a Chain(-E):**  
    If we need to rename a user defined chain   
Syntax:  
***# iptables -E***

***# iptables -E INTRANET    SUBINTRANET***

    This will rename the chain to SUBINTRANET. The iptables “will update the references as well”(The reference to the default chain).  
**Chain Policy (-P):**  
    It is usually “accept” in RedHat environment for all the chains in filter table. should be very careful while setting the chain default policy to DROP(Update the iptables to permit the appropriate access, else if we are using a remote session this may freez the access).  
Syntax:  
***# iptables -P***

***# iptables -P INPUT DROP***  
    This will make the default policy of INPUT chain to DROP.  
***# iptables -L -n***

    Check the “chain INPUT (policy DROP)” to verify.  
Note:-  
    Default DROP Policy may prevent typical TCP/UDP/ICMP communication.So a state matching rule should be added in case of such scenarios.

**IPTables Building Rules with Source, Destination of IP, MAC, Protocols & Port**  
      
    Here we will deal with the possibilities to match the traffic to define the rule, i.e, matching destination & source IP/MAC/PORT/PROTOCOL, Interfaces,Usage of Wildcards etc.

Matching the traffic based on Source and destination:  
*–src/-s/–source  
–dst/-d/–destination*  
    These are the switches used to match the source and destination of the traffic. Widely used while rules created based on source and destination address  
Eg:-  
**Blocking all the traffic from a source (192.168.1.200) (–src)**

***# iptables -A INPUT –src 192.168.1.200 -j DROP***

    This Drops all the incoming traffic to out server from the Source 192.168.1.200. Here the match of source is used by “–src”.  
**Blocking all the traffic To a destination from our server (–dst)**

***# iptables -A OUTPUT –dst 192.168.1.200 -j DROP***

    This Drops all the outgoing traffic in our server to 192.168.1.200. Here the match of destination is used by “–dst”.

Matching Based on Interface:  
    It is useful while creating the rules based on a particular interface.  
*(-i eth0/eth1.. etc)*  
    switch “-i” is used to match the traffic with the interface to define the rule.  
Eg:-  
(-i eth1)

***# iptables -A INPUT -i eth1 –src 192.168.1.200 -j DROP***

    Any incoming traffic from the ip address on the interface eth1 will be dropped.

Negation rule:

***# iptables -A INPUT -i eth1 –src !192.168.1.200 -j DROP***

    This will Drop all the incoming traffic to the interface eth1 other than the IP 192.168.1.200. Only the incoming traffic from ip 192.168.1.200 will be accepted.

***# iptables -A INPUT -i eth1 -j DROP***

    This Drops all the incoming traffic on the interface eth1.

Wildcard for Matching all interfaces(eth+):  
For eg:-   
    IF we have more interfaces like eth0, eth1, eth2, eth3, eth4 etc and need to define a rule that matched all the interface, we can use the wild-card eth+ . eth+ will match all the interfaces starting with “eth”.  
For Eg:-

***# iptables -A INPUT -i eth+ -p tcp –dport 23 -j DROP***

    This will drop all the incoming telnet traffic to all interfaces, which starts with eth.

TCP Based Matching (–protocol/-p): (Connection Oriented)  
     Majority of the rules are based on TCP . TCP is on Transport Layer (layer 4).  
***-p tcp/ –protocol tcp***  
    This switch will make IPTables to initiate the tcp modules and allow/deny the tcp based traffic. This switch makes sense to IPTables about the three way handshake of TCP. The protocol type (tcp/udp) has to be specified while using the “-p” match.  
***–sport/–source-port***  
    Generally the –sport of TCP client will be greater than 1024, and it is generaly picked arbitrarily from greater than 1024. So usally we wont filter based on the source port for TCP based traffic until and unless we know exactly how a application behaves.  
***–dport/–destination-port***  
    This is the common match that used along with the “-p” switch. Each and every TCP connection will have a well defined destination port. so based on this destination port we created/matched the rule.  
***–tcp-flags SYN, ACK SYN, ACK, FIN***  
    This is used to match the three way handshake of the tcp protocols.  
    SYN – Step 1 of Three way Handshake (Initial synchronization) (From Server)  
    ACK SYN – Step 2 of three way Handshake (To Acknowledge that the SYN has recieved) (From Client)  
    ACK – Step 3 of Three way HandShake(From Server)  
    FIN (Finishing a TCP Session)  
Eg:-

***# iptables -A INPUT -p tcp –dport 23 -j DROP***

    Here Match is made with the protocol TCP having the destination port of 23. So all the incoming traffic to telnet will be dropped.

***# iptables -A OUTPUT -p tcp –dport 21 -j DROP***

    This will Drop all the FTP outbound traffic(all request to ftp access from our server)

UDP Based Match: (Connection Less)  
    Some of the UDP based applications are TFTP:69, Syslog:514, NTP:123, DHCP:67/68, DNS:53  
***-p udp/–protocol udp  
–dport/–destination-port  
–sport/–source-port***  
    In majority of the cases, the UDP based traffic having same source port as the destination port.Eg:- The NTP client packets has same destination-port and source-port as 123 in header.  
Eg:-  
If we are running the syslogd daemon we have to block all other traffic to the service other than the syslog server.

***# iptables -A INPUT -p udp –dport 514 -s !192.168.1.3 -j DROP***

    So here only the traffic from the host 192.168.1.3 with UDP:514 will be accepted and all other source will be denied. Here the match is made with the protocol UDP and –dport 514 along with the Source(-s) using Negation(!).

ICMP based traffic Match.  
    This is designed to communicate the status information.  
**various types of ICMP:**  
    echo-request – PING (sends the request via output chains using echo-request to destination)  
    echo-reply –   PONG (Remote system Recieves the echo-request and responds with an echo-reply (PONG))  
***-p icmp/–protocol icmp***  
    Here defines the protocol type  
***–icmp-type name/number of icmp type***  
    Here we specifies the ICMP-Types. It can be name or number.eg:- echo-reply, icmp-request etc.  
To get the list of icmp types that supported by the IPTables

***# iptables -p icmp –help***

        Using this we can build the rules. The above command can be used for both the tcp and udp protocols

***# iptables -p tcp –help  
    # iptables -p udp –help***

Eg:-

***# iptables -A INPUT -p icmp –icmp-type echo-reply -j DROP***

    All the echo-reply from outside will be droped.  
**Rule to drop all the echo-request to our filrewall from all outbound destination.**

***# iptables -A INPUT -i eth1 -p icmp –icmp-type echo-request -j DROP***

    This will disable all the echo-request from the outside interface. But from this server we will be able to ping to any other system because we have not doped any incoming echo-reply.

Multiport Matching in single rule (-m):  
    This feature uses to match multiple ports in a single rule.  
***-m multiport***  
**Checking the Multiport module installation**

***# rpm -ql iptables |grep multiport***  
*/lib/iptables/libipt\_multiport.so*

    This is the modlue responsible for multiport  
Eg:-

***# iptables -A INPUT -p tcp -m multiport –dport 21,23 -j DROP***

    Here we defined the multiple ports in single rule.

Matching Layer 2 Traffic (MAC-address):  
    The MAC address is least changable.  
**Checking the capability of iptables to match the Layer 2 traffic**

***# rpm -ql iptables |grep mac***  
*/lib/iptables/libipt\_mac.so*

    This is the modlue responsible for mac address based rule.  
***-m mac***  
    This will tell iptables to consult the libipt\_mac.so module for processing the rule  
***–mac-source***  
    Source MAC address. Same as the –src option in Layer 3 (IP Adress)  
***–mac-destination***  
    Destination MAC address. Same as the –dst option in Layer 3 (IP Address)  
Eg:-

***# iptables -A INPUT -p tcp -m mac –mac-source 00:09:8F:3E:10:3A -j DROP***

    IF the source mac address is matched then the traffic will be DROPed.  
Filtering based on Layer 2 (MAC Address) is more secure because the IP Address can easily be changed.

**IPTables Statefullness(-m state –state):**  
    IPTables provide state fullness. The state full firewall is considered more secure than stateless firewall because of their connection tracking capability and their ability to determine whether or not the session is new,related, invalid or established. Based on this criteria we can create more powerfull rules.  
**State Module:**

***# rpm -ql iptables | grep -i conntrack***  
 */lib/iptables/libipt\_conntrack.so*

    This is the module that makes IPTables to behave as statefull. It is applicable for all the protocols (TCP/UDP/ICMP)  
**The states are:**  
*NEW (The First SYN traffic)  
ESTABLISHED  
RELATED(SESSION/STATE)  
INVALID*  
    When a user creates a TCP/UDP based session IPTables can follow the connection. Here IPTable will keep a track with SYN, ACK-SYN, ACK and labelled with NEW(for SYN), ESTABLISHED or RELATED (For all other subsequent connections).

**Example:**  
Permit Host to Initialte the connection and deny other hosts from initiating traffic to our host.

# Default Policy to Drop All connection  
***# iptables -P INPUT DROP  
# iptables -P OUTPUT DROP***  
# State Rule   
***# iptables -A OUTPUT -m state –state NEW,ESTABLISHED -j ACCEPT***  
    This will allow creating a NEW session (SYN) with outside and continue the ESTABLISHED  connections(regardless of protocol(UDP/TCP))  
***# iptables -A INPUT -m state –state ESTABLISHED -j ACCEPT***  
    After initiating a traffic to any other machine, the traffic will be permitted when it comes back.(regardles of protocol(UDP/TCP))

**End Result:**  
The host will be able to make all connections to out side(NEW & ESTABLISHED is allowed in OUTPUT chain).   
All new connection coming to our system will be dropped(No NEW is defined in INPUT chain only  ESTABLISED as well the default rule of DROP) only allows the ESTABLISHED connections(Initiated by our host)

**The details of the connection tracking will be stored in**

***# cat /proc/net/ip\_conntrack***

    This file contains the status of all the established connections in the system for all protocols. The number of packets that transmitted, The   
source and destination address, source and destination port etc.

**IPTables Targets(-j)**  
Commonly used targets are   
**1. ACCEPT**  
     Sends packtes to other rule or process

**2. DROP**  
     Drops the packet silently. Remote machine will not be aware about what happend to the packet.

**3. REJECT**  
     When the rule met an error msg is send to client.  
Eg of Reject:-

***# iptables -A INPUT -p icmp –icmp-type echo-request -j REJECT***

This will reject all the echo request part with a msg icmp-port-unreachable. If we ping to the host we will get a destination host unreachable.

**4. REDIRECT**  
     This is used to redirect a current traffic to a desired target. It is applied to PREROUTING chain of NAT table.  
Eg:-

***# iptables -t nat -A PREROUTING -p tcp –dport 3128 -j REDIRECT –to-port 80***

     This will redirect all the trafic coming to the destination port 3128 to 80.

***# iptables -L -n -t nat -v***

     Test with the verbose mode to get the packet count which hits the rule.

**5. LOG**  
     This allow us to log the traffic which meets the rules from the level of debug to emergency using syslog.

**IPTable Logs:**  
     It relies upon the kernel(kern) facility in syslog. So have to setup the syslog for logging the iptables activities.

**Setup Logging**  
     Primarily we enable the logging in IPTables  
**Enabling the Log for a chain**

***# iptables -I INPUT 1 -p tcp –dport 22 -j LOG***

This will start logging for the traffic which meets the above rule.(Logs all the incoming ssh request.) The default level of logging is warning. The Log level corresponds to the syslog.

***# iptables -L -n -v***

Check any packets hits the log

***# tail -f /var/log/messages***

This is the default place where undefined facilities logs to.So we the kern facility has been logging to /var/log/messages.  
**Configure syslog to log iptables activity separately:**  
We will change the facility to log to a seperate file

***# vi /etc/syslog.conf***  
*kern.none /var/log/messeges*  
*kern.\* /var/log/firewall.log*

This will stop the kern facility to log to /var/log/messeges and redirects all levels of logs to /var/log/firewall.log

***# service syslog reload***

This will restart the syslog daemon and creates the file /var/log/firewall.log.  
Test the Logging information by creating the traffic to port 22 on host.

***# tail -f /var/log/firewall.log***

brief about the log format:-   
time- syslog facility – interface that revived the tracfic- MAC address of the remote system- MAC address of the local system – SRC IP- DSTIP – ID=packet sequence number – SPT=source port – DPT=destination port etc

Note:-  
Generally logging should be enabled for separate chains & a specific rule. A catch all log for all the traffic will grow the log file numerously.

**Loging All trafic**

***# iptables -A INPUT -j LOG***  
***# tail -f /var/log/firewall.log***

This will Log all traffic destined to the local server(INPUT). This will log all the protocols

**Log All except a perticular protocol from host 192.168.1.53**

***# iptables -I INPUT 1 -p tcp ! –dport 22 -src 192.168.1.53 -j LOG***  
***# tail -f /var/log/firewall.log***

This will log everything except traffic to destination port 22

**Log Excluding Multiple port in single rule**

***# iptables -I INPUT 1 -m multiport -p tcp –dport !80,8080 -j LOG***  
***# tail -f /var/log/firewall.log***

This will log all traffic except packet destined to port 80 and 8080.

**Log using separate chains**  
Now we will check how to create a separate chain in IPTables for logging activities.  
Create a New chain

***# iptables -N LOGGER***

Create a reference in INPUT chain to new chain LOG

***# iptables -I INPUT 1 -j LOGGER***

Create the logging rule in chain LOG

***# iptables -A LOGGER -m multiport -p tcp –dport 21,22,80,143,8080 -j LOG***  
***# tail -f /var/log/firewall.log***

This will start logging ports 21,22,80,143 & 8080.

**Loging the ssh access to the console.**  
In iptables:  
Create a New CHAIN

***# iptables -N SSHLOG***

Create a reference in INPUT chain to new chain SSHLOG

***# iptables -I INPUT 1 -j SSHLOG***

Create the loging rule in chain LOG

***# iptables -A LOGGER -p tcp –dport 22 -j LOG***

In syslog:

***# vim /etc/syslog.conf***  
*kern.\* /dev/console*  
***# service syslogd restart***

This will start logging any ssh access to the console.

**Prefixing Interesting Traffic with a Log Prefix(–log-prefix “log prefix”)**

***# iptables -A LOGGER -p tcp –dport 22 -j LOG –log-prefix “SSH Access Logs”***

This will prefix the given string to the log. So it is easy to grep/awk the content from the log file.  
Note:-  
The Maximum prefix length is 29 characters.

Note:-  
–log-level (debug to emer)

**PTables Routing (Forward Chain)**  
           The Forward chain holds the rules that take care of routing  
**Enabling the Routing.**

***#sysctl***

This is the key utilities which shows the running kernel parameters.

***#syscltl net.ipv4.ip\_forward***

This will show the status of the IPV4 routing in our local system.

***# echo 1 > /proc/sys/net/ipv4/ip\_forward***

This will turn on the routing in kernel.

***# vim /etcv/sysctl.conf***

***net.ipv4.ip\_forward = 1***

This will make the routing permanent.

***# route add -net 10.0.0.0 netmask 255.0.0.0 gw 192.168.1.10***

This will make the net routing in Linux host.

Forward Chain to Manage the Routing.  
     All the packets that is subjected to route will traverse through Forward Chain in a Linux router.

**Defining the Forward chain policy**  
1. Initially make the default policy to Drop all the routing traffic in firewall

***# iptables -P FORWARD DROP***

This will make all the routing traffic to be dropped as a default policy.  
2. Specify only certain source network to be routed

***# iptables -A FORWARD -s 192.168.1.0/24 -d 10.0.0.0/8 -j ACCEPT***

This will allow the traffic from 192.168.1.0 network to 10.0.0.0. But the traffic from 10.0.0.0/8 network if comes back will not be accepted until & unless we define a state rule or a rule that allows the traffic from the given source.  
or

***# iptables -A FORWARD -m state –state NEW,ESTABLISHED -s 192.168.1.0/24 -j ACCPET***

This will allow and route all the new and established connection from the network 192.168.1.0 to any destination  
3. Accept the return traffic

***# iptables -A FORWARD -m state –state ESTABLISHED -j ACCEPT***

This will allow/accept all the established connection in the forward chain. This will allow the return traffic.  
or define a rule that allows the return traffic from the network 10.0.0.0/8. Here usage of the “state” rule makes the definition of the firewall rule more easier and secure.

Logging the routing traffic in FORWARD Chain:

***# iptables -N ROUTELOG***

***# iptables -A FORWARD -j ROUTELOG***

***# IPTABLES -I ROUTELOG -j LOG***

This will create a new chain and starts logging all the routing activities.

Allowing a subnet to access outer world web

***# iptables -A FORWARD -s 10.0.0.0/24 -p tcp –dport 80 -j ACCEPT***

Allow the UDP(DNS) queries to outside

***# iptables -A FORWARD -s 10.0.0.0/24 -p udp –dport 53 -j ACCEPT***

**IPTables with DMZ**  
Let consider the interface to setup/understand the DMZ.

* eth0: external interface (192.168.1.0/24)
* eth1: Internal Interface (10.0.0.0/8)
* eth2: The DMZ zone (172.16.0.0/16)

**Step 1:**  
Create DNAT for all the servers in the DMZ zone (eth2) for accessing the service externally

***# iptables -t nat -A PREROUTING -d 192.168.1.2 -p tcp –dport 80 -j DNAT –to-destination 172.16.0.2***

***# iptables -t nat -A PREROUTING -d 192.168.1.2 -p tcp –dport 443 -j DNAT –to-destination 172.16.0.2***

If any request comes to firewall with the destination IP as 192.168.1.2 and port as 80 will be DNATed to 172.16.0.2 in DMZone.  
Now test accessing the service in DMZone from Internel as well externel network. From both the network we will be able to access the server in the DMZone using the IP 192.168.1.2.

**Step2:**  
Configure the split DNS or 2 DNS systems (Inside&Outside of the DMZone).  
**Step3:**  
Setup rule for trusted network from the outside network(Internet) for the traffic which will allow system access (SSH).

***# iptables -A FORWARD -s 10.0.0.0/8 -j ACCEPT***

***# iptables -A FORWARD -s 172.16.0.0/16 -m state –state ESTABLISHED -j ACCEPT***

***# iptables -P FORWARD DROP***

This will deny all access to the DMZone from the internet hosts, only allows the Internal network. Because the default policy of FORWARD chain is set to drop, we need to create the “state match” for the hosts in the DMZone(This will deny sourcing a new connection from the DMZone, only established connection will be permitted).

**Dual DMZ Configuration**  
This is the way of segmenting the servers to separate DMZones.  
Let consider the interface to setup/understand the Dual DMZ.

* eth0: externel interface (192.168.1.0/24)
* eth1: Internel Interface (10.0.0.0/8)
* eth2: The DMZ1 zone (172.16.0.0/16) (Web servers)
* eth3: The DMZ2 zone (172.17.0.0/16) (DBMS, App servers like JBOSS, TOMCAT etc)

Using this method we will be able to control the traffic from one DMZone to another. This is used for the scenarios of Application servers which need to contact the DB Servers located on separate server.

Here we have to permit only the DMZ1 to contact the DMZ2. all other traffic will be denied.So the servers in the DMZ2 zone will be more secured.

***# iptables -t nat -A FORWARD -s 172.16.0.0/16 -d 172.7.0.0/16 -j ACCEPT***

***# iptables -t nat -A FORWARD -m state –state ESTABLISED -s 172.17.0.0/16 -j ACCEPT***

***# iptables -t nat -P FORWARD DROP***

This will make only the DMZ1 to contact the DMZ2. And from DMZ2 only the established connection will be permitted. All other request will be dropped in the FORWARD chain.  
Note:-  
These rules are the basic backbone for setting up the routing and Natting in DMZone. All other rules should be defined according to our network need.