Academic Year: 2025-26 **Semester: V**

Class / Branch: TE IT

Subject: Security Lab

Subject Lab Incharge: Prof. Apeksha Mohite

Experiment No. 03

1. **Aim: To study installation and configuration of Linux Kernel firewall iptables.**
2. **Software Required** : Ubuntu 14.04 OS, iptables 1.6
3. **Theory :**

A firewall is a network security system designed to prevent unauthorized access to or from a private network. Firewalls can be implemented in both hardware and software, or a combination of both. Network firewalls are frequently used to prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets. Firewalls can be either hardware or software but the ideal configuration will consist of both. In addition to limiting access to your computer and network, a firewall is also useful for allowing remote access to a private network through secure authentication certificates and logins. Software firewalls are installed on your computer and can be customized which gives administrator control over its function and protection features. A software firewall will protect computer from outside attempts to control or gain access.

Setting up a good firewall is an essential step to take in securing any modern operating system. Most Linux distributions ship with a few different firewall tools that can be used to configure firewalls. Ipptables is a standard firewall included in most Linux distributions by default. It is actually a front end to the kernel-level netfilter hooks that can manipulate the Linux network stack. It works by matching each packet that crosses the networking interface against a set of rules to decide what to do.

IPTABLES : TABLES and CHAINS

Iptables command allows the system administrators to manage incoming and outgoing traffics. IPtables contains set of tables, tables consists of chains and chains consists of rules. The iptables firewall operates by comparing network traffic against a set of **rules**. The rules define the characteristics that a packet must have to match the rule, and the action that should be taken for matching packets. There are many options to establish which packets match a specific rule. i.e. packet protocol type, the source or destination address or port, the interface that is being used, its relation to previous packets. When the defined pattern matches, the action that takes place is called a **target**. A target can be a final policy decision for the packet, such as accept, or drop. These rules are organized into groups called **chains**. A chain is a set of rules that a packet is checked against sequentially. When the packet matches one of the rules, it executes the associated action and is not checked against the remaining rules in the chain.

IPTables has the following 3 built-in tables.

1. Filter Table

Filter is default table for iptables. So, if you don't define you own table, you'll be using filter table.

Iptables's filter table has the following built-in chains.

•INPUT chain

•OUTPUT chain

•FORWARD chain

2. NAT table

Iptable's NAT table has the following built-in chains.

•PREROUTING chain – Alters packets before routing. i.e Packet translation happens immediately after the packet comes to the system (and before routing). This helps to translate the destination ip address of the packets to something that matches the routing on the local server. This is used forDNAT (destination NAT).

•POSTROUTING chain – Alters packets after routing. i.e Packet translation happens when the packets are leaving the system. This helps to translate the source ip address of the packets to something that might match the routing on the destination server. This is used for SNAT (source NAT).

•OUTPUT chain – NAT for locally generated packets on the firewall.

3. Mangle table

Iptables's Mangle table is for specialized packet alteration. This alters QOS bits in the TCP header. Mangle table has the following built-in chains.

•PREROUTING chain

•OUTPUT chain

•FORWARD chain

•INPUT chain

•POSTROUTING chain

A user can create chains as needed. There are three chains defined by default. They are:

•**INPUT**: This chain handles all packets that are addressed to your server.

•**OUTPUT**: This chain contains rules for traffic created by your server.

•**FORWARD**: This chain is used to deal with traffic destined for other servers that are not created on your server. This chain is basically a way to configure your server to route requests toother machines.

Each chain can contain zero or more rules, and has a default **policy**. The policy determines what happens when a packet drops through all of the rules in the chain and does not match any rule. Firewall can either drop the packet or accept the packet if no rules match. Through a module that can be loaded via rules, iptables can also track connections. This means rules can be created that can define what happens to a packet based on its relationship to previous packets. This capability is called "state tracking", "connection tracking", or configuring the "state machine".

Usage of Iptables

An iptable command-line utility can be followed by an argument denoting the command to execute. To add a new rule to a chain, you use -A . Use -D to remove it, and -R to replace it. The -s option specifies the source address attached to the packet, -d specifies the destination address, and the -j option specifies the target of the rule. The ACCEPT target will allow a packet to pass. The -i option now indicates the input device and can be used only with the INPUT and FORWARD chains. The -o option indicates the output device and can be used only for OUTPUT and FORWARD chains.

Set Default Chain Policies

The default chain policy is ACCEPT. Change this to DROP for all INPUT, FORWARD, and OUTPUT chains as shown below.

```
iptables -P INPUT DROP

iptables -P FORWARD DROP

iptables -P OUTPUT DROP
```

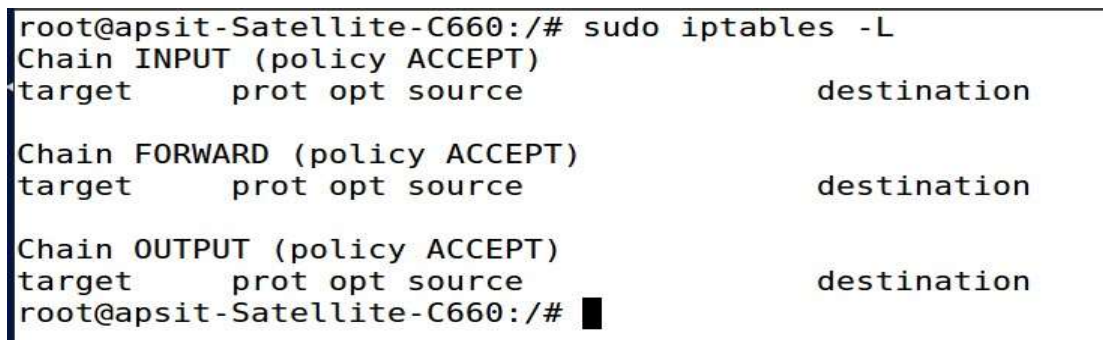
When you make both INPUT, and OUTPUT chain’s default policy as DROP, for every firewall rule requirement you have, you should define two rules. i.e one for incoming and one for outgoing. In all our examples below, we have two rules for each scenario, as we’ve set DROP as default policy for both INPUT and OUTPUT chain.

Basic iptables Commands

iptables commands must be run with root privilege

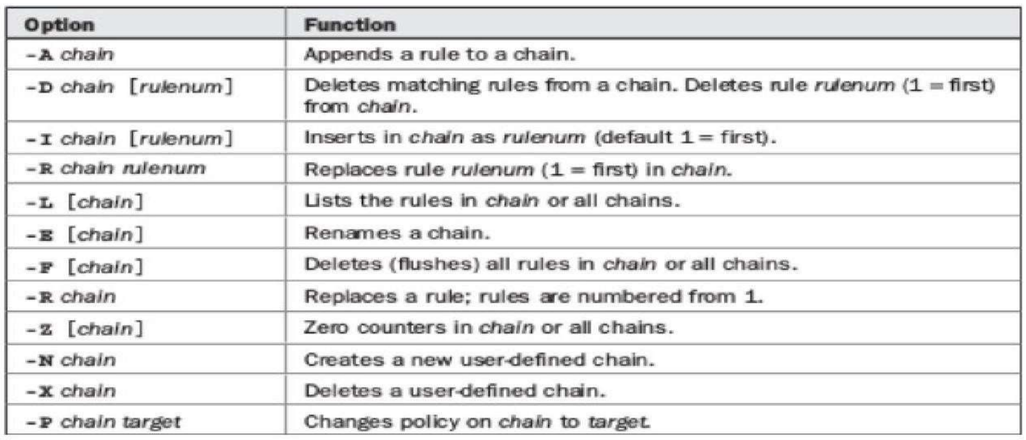
1. list the current rules that are configured for iptables

sudo iptables -L

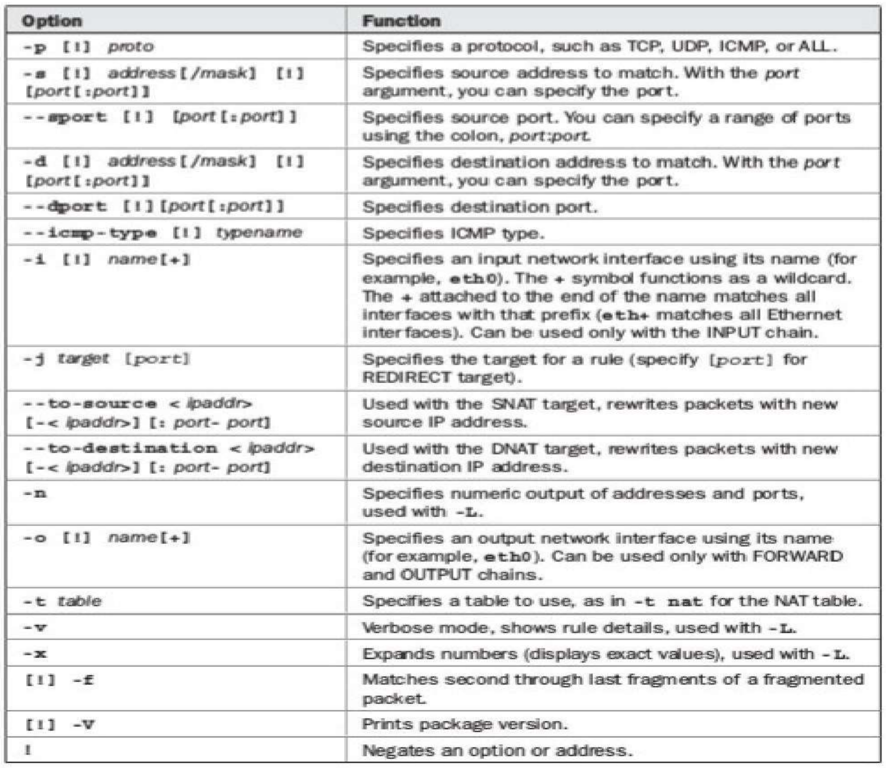


```
root@apsit-Satellite-C660:/# sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@apsit-Satellite-C660:/# 
```

Following table lists several basic options.

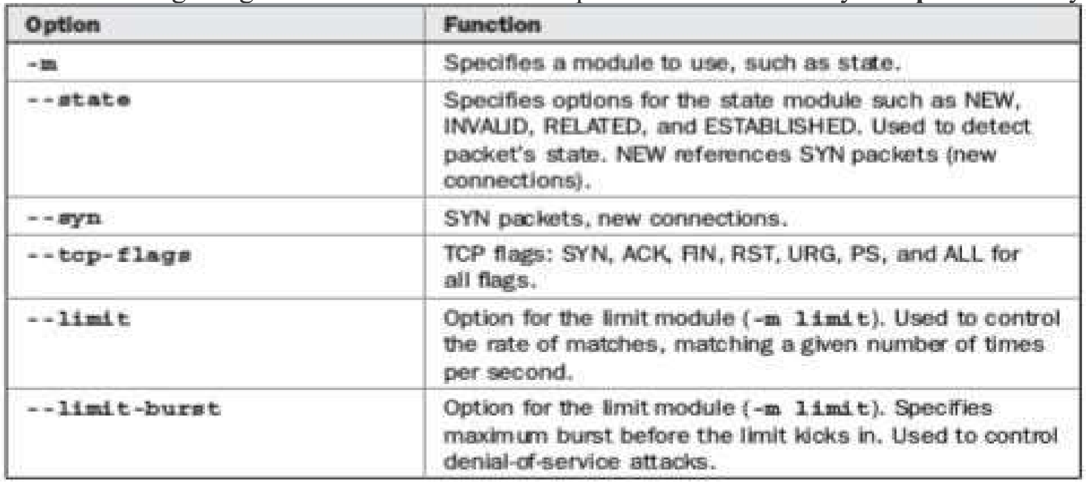


Option	Function
-A *chain*	Appends a rule to a chain.
-D *chain* [*rulenum*]	Deletes matching rules from a chain. Deletes rule *rulenum* (1 = first) from *chain*.
-I *chain* [*rulenum*]	Inserts in *chain* as *rulenum* (default 1 = first).
-R *chain rulenum*	Replaces rule *rulenum* (1 = first) in *chain*.
-L [*chain*]	Lists the rules in *chain* or all chains.
-E [*chain*]	Renames a chain.
-F [*chain*]	Deletes (flushes) all rules in *chain* or all chains.
-R *chain*	Replaces a rule; rules are numbered from 1.
-Z [*chain*]	Zero counters in *chain* or all chains.
-N *chain*	Creates a new user-defined chain.
-X *chain*	Deletes a user-defined chain.
-P *chain target*	Changes policy on *chain* to *target*.

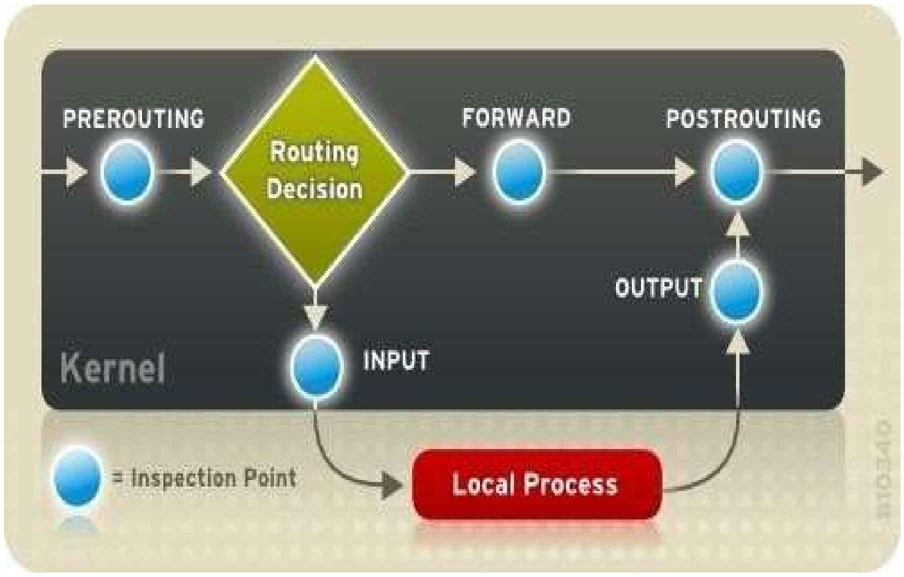


Option	Function
`-p [!]` *proto*	Specifies a protocol, such as TCP, UDP, ICMP, or ALL.
`-s [!]` *address*`[/`*mask*`] [!] [`*port*`[:`*port*`]]`	Specifies source address to match. With the *port* argument, you can specify the port.
`--sport [!] [`*port*`[:`*port*`]]`	Specifies source port. You can specify a range of ports using the colon, *port:port*.
`-d [!]` *address*`[/`*mask*`] [!] [`*port*`[:`*port*`]]`	Specifies destination address to match. With the *port* argument, you can specify the port.
`--dport [!][`*port*`[:`*port*`]]`	Specifies destination port.
`--icmp-type [!]` *typename*	Specifies ICMP type.
`-i [!]` *name*`[+]`	Specifies an input network interface using its name (for example, `eth0`). The `+` symbol functions as a wildcard. The `+` attached to the end of the name matches all interfaces with that prefix (`eth+` matches all Ethernet interfaces). Can be used only with the INPUT chain.
`-j` *target* `[port]`	Specifies the target for a rule (specify `[port]` for REDIRECT target).
`--to-source <`*ipaddr*`> [-<`*ipaddr*`>] [:`*port*`-`*port*`]`	Used with the SNAT target, rewrites packets with new source IP address.
`--to-destination <`*ipaddr*`> [-<`*ipaddr*`>] [:`*port*`-`*port*`]`	Used with the DNAT target, rewrites packets with new destination IP address.
`-n`	Specifies numeric output of addresses and ports, used with `-L`.
`-o [!]` *name*`[+]`	Specifies an output network interface using its name (for example, `eth0`). Can be used only with FORWARD and OUTPUT chains.
`-t` *table*	Specifies a table to use, as in `-t nat` for the NAT table.
`-v`	Verbose mode, shows rule details, used with `-L`.
`-x`	Expands numbers (displays exact values), used with `-L`.
`[!] -f`	Matches second through last fragments of a fragmented packet.
`[!] -V`	Prints package version.
`!`	Negates an option or address.

The following image outlines how the flow of packets is examined by the **iptables** subsystem:



Option	Function
`-m`	Specifies a module to use, such as state.
`--state`	Specifies options for the state module such as NEW, INVALID, RELATED, and ESTABLISHED. Used to detect packet's state. NEW references SYN packets (new connections).
`--syn`	SYN packets, new connections.
`--tcp-flags`	TCP flags: SYN, ACK, FIN, RST, URG, PS, and ALL for all flags.
`--limit`	Option for the limit module (`-m limit`). Used to control the rate of matches, matching a given number of times per second.
`--limit-burst`	Option for the limit module (`-m limit`). Specifies maximum burst before the limit kicks in. Used to control denial-of-service attacks.



IPTables and Connection Tracking

Administrator can inspect and restrict connections to services based on their connection state. A module within **iptables** uses a method called connection tracking to store information about incoming connections. Access can be allowed or denied based on the following connection states:

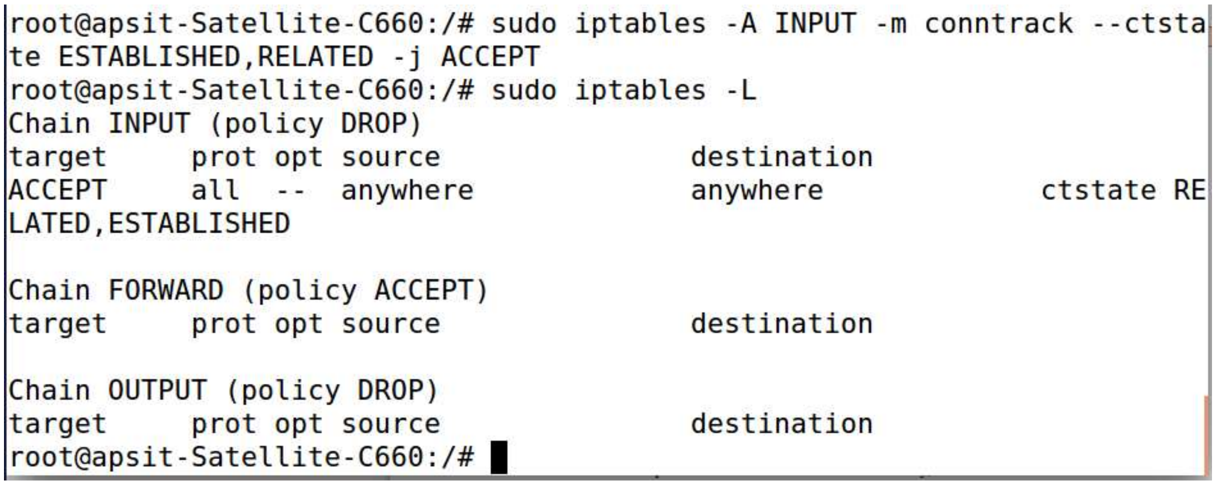
- NEW — A packet requesting a new connection, such as an HTTP request.
- ESTABLISHED — A packet that is part of an existing connection.
- RELATED — A packet that is requesting a new connection but is part of an existing connection. For example, FTP uses port 21 to establish a connection, but data is transferred on a different port (typically port 20).
- INVALID — A packet that is not part of any connections in the connection tracking table.

Stateful functionality of **iptables** can be used for connection tracking with any network protocol, even if the protocol itself is stateless (such as UDP).

Allow Established and Related Incoming Connections

As network traffic generally needs to be two-way—incoming and outgoing—to work properly, it is typical to create a firewall rule that allows **established** and **related** incoming traffic, so thatthe server will allow return traffic to outgoing connections initiated by the server itself.
Following command will allow that

sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT


```
root@apsit-Satellite-C660:/# sudo iptables -A INPUT -m conntrack --ctsta
te ESTABLISHED,RELATED -j ACCEPT
root@apsit-Satellite-C660:/# sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             ctstate RE
LATED,ESTABLISHED

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
root@apsit-Satellite-C660:/# 
```

- **-A INPUT**: The -A flag *appends* a rule to the end of a chain. This is the portion of the command that tells iptables that we wish to add a new rule, that we want that rule added to the end of the chain, and that the chain we want to operate on is the INPUT chain.
- **-m conntrack**: iptables has a set of core functionality, but also has a set of extensions or modules that provide extra capabilities.

In this portion of the command, we're stating that we wish to have access to the functionality provided by the conntrack module. This module gives access to commands that can be used to make decisions based on the packet's relationship to previous connections.

- **--ctstate**: This is one of the commands made available by calling the conntrack module. This command allows us to match packets based on how they are related to packets we've seen before.

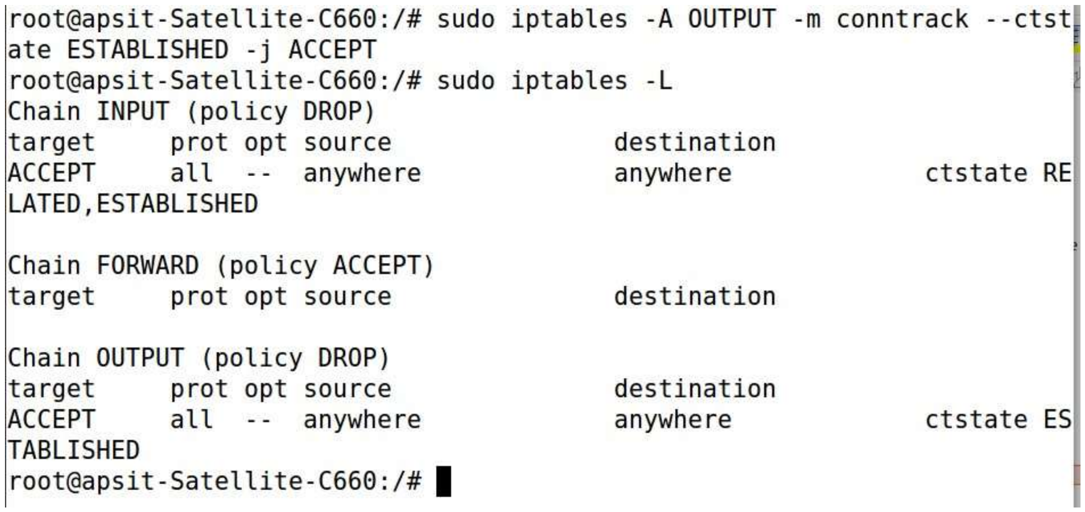
We pass it the value of ESTABLISHED to allow packets that are part of an existing connection. We pass it the value of RELATED to allow packets that are associated with an established connection. This is the portion of the rule that matches our current SSH session.

- **-j ACCEPT**: This specifies the target of matching packets. Here, we tell iptables that packets that match the preceding criteria should be accepted and allowed through.

Allow Established Outgoing Connections

You may want to allow outgoing traffic of all **established** connections, which are typically the response to legitimate incoming connections. This command will allow that:

sudo iptables -A OUTPUT -m conntrack --ctstate ESTABLISHED -j ACCEPT



```
root@apsit-Satellite-C660:/# sudo iptables -A OUTPUT -m conntrack --ctst
ate ESTABLISHED -j ACCEPT
root@apsit-Satellite-C660:/# sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             ctstate RE
LATED,ESTABLISHED

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             ctstate ES
TABLISHED
root@apsit-Satellite-C660:/# 
```

Service : SSH

Allow All Incoming SSH

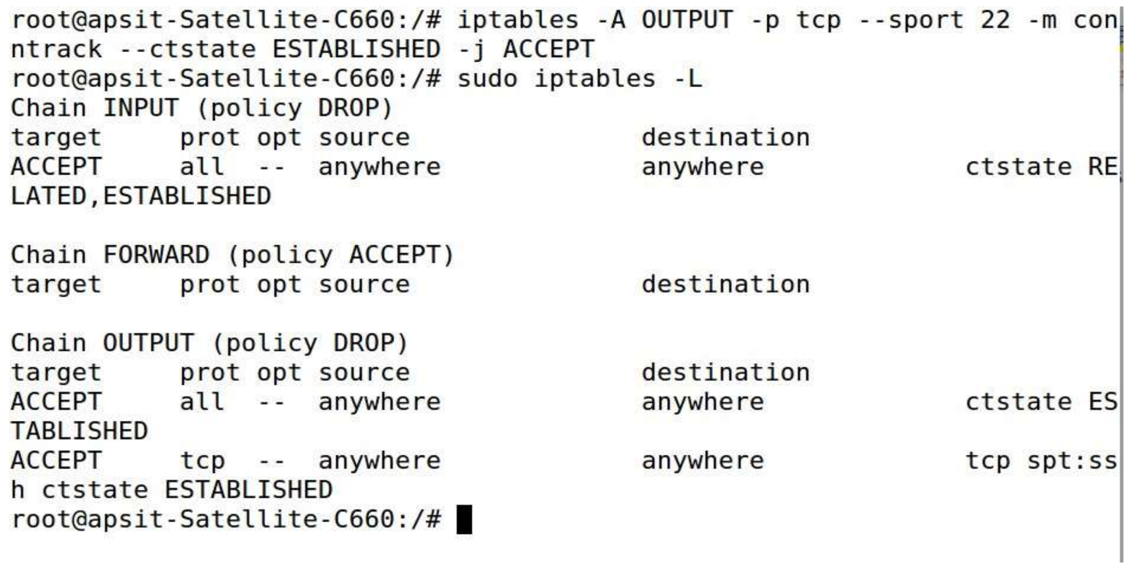
If hosting a cloud server or hosting Web server then this will probably requires allowing incoming SSH connections (port 22) so administrator can connect to and can manage server.

iptables -A INPUT -p tcp --dport 22 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT

iptables -A OUTPUT -p tcp --sport 22 -m conntrack --ctstate ESTABLISHED -j ACCEPT

The second command, which allows the outgoing traffic of **established** SSH connections, isonly necessary if the OUTPUT policy is not set to ACCEPT.

Allow outgoing SSH to Specific IP address or subnet



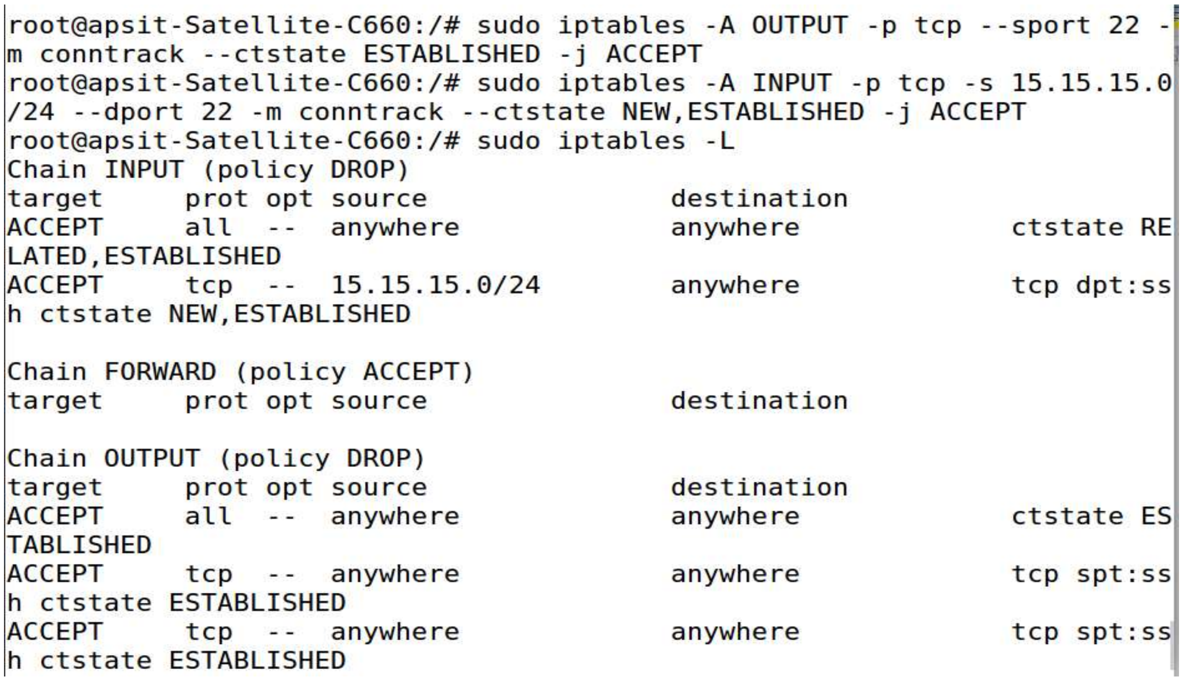
```
root@apsit-Satellite-C660:/# iptables -A OUTPUT -p tcp --sport 22 -m con
ntrack --ctstate ESTABLISHED -j ACCEPT
root@apsit-Satellite-C660:/# sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             ctstate RE
LATED,ESTABLISHED

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             ctstate ES
TABLISHED
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:ss
h ctstate ESTABLISHED
root@apsit-Satellite-C660:/# 
```

To allow outgoing SSH connections to a specific IP address or subnet, specify the destination.For example, to allow outgoing ssh to entire 15.15.15.0/24 subnet, run these commands:

sudo iptables -A INPUT -p tcp -s 15.15.15.0/24 --dport 22 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT

sudo iptables -A OUTPUT -p tcp --sport 22 -m conntrack --ctstate ESTABLISHED -j ACCEPT

The second command, which allows the outgoing traffic of **established** SSH connections, isonly necessary if the OUTPUT policy is not set to ACCEPT.



```
root@apsit-Satellite-C660:/# sudo iptables -A OUTPUT -p tcp --sport 22 -
m conntrack --ctstate ESTABLISHED -j ACCEPT
root@apsit-Satellite-C660:/# sudo iptables -A INPUT -p tcp -s 15.15.15.0
/24 --dport 22 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
root@apsit-Satellite-C660:/# sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             ctstate RE
LATED,ESTABLISHED
ACCEPT     tcp  --  15.15.15.0/24        anywhere             tcp dpt:ss
h ctstate NEW,ESTABLISHED

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             ctstate ES
TABLISHED
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:ss
h ctstate ESTABLISHED
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:ss
h ctstate ESTABLISHED
```

Allow Incoming Rsync from Specific IP Address or Subnet

Rsync, which runs on port 873, can be used to transfer files from one computer to another. To allow incoming rsync connections from a specific IP address or subnet, specify the source IP address and the destination port. For example, to allow the entire 15.15.15.0/24 subnet to be able to rsync to your server, run these commands

iptables -A INPUT -p tcp -s 15.15.15.0/24 --dport 873 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT

iptables -A OUTPUT -p tcp --sport 873 -m conntrack –ctstate ESTABLISHED -j ACCEPT

Service : Web Server

Allow All Incoming HTTP and HTTPS

Web servers, such as Apache and Nginx, typically listen for requests on port 80 and 443 for HTTP and HTTPS connections, respectively. If default policy for incoming traffic is set to drop or deny, then create rules that will allow web server to respond to those requests. To allow both HTTP and HTTPS traffic, administrator can use the **multiport** module to create a rule that allows both ports. To allow all incoming HTTP and HTTPS (port 443) connections run these commands.

```
iptables -A INPUT -p tcp -m multiport --dports 80,443 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
```

```
iptables -A OUTPUT -p tcp -m multiport --dports 80,443 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

The second command, which allows the outgoing traffic of **established** HTTP and HTTPS connections, is only necessary if the OUTPUT policy is not set to ACCEPT

Service : MySQL

Allow MySQL from Specific IP Address or Subnet

MySQL listens for client connections on port 3306. If your MySQL database server is being used by a client on a remote server, you need to be sure to allow that traffic. To allow incoming MySQL connections from a specific IP address or subnet, specify the source. For example,to allow the entire 15.15.15.0/24 subnet, run these commands

```
iptables -A INPUT -p tcp -s 15.15.15.0/24 --dport 3306 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
```

```
iptables -A OUTPUT -p tcp --sport 3306 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

The second command, which allows the outgoing traffic of **established** MySQL connections, is only necessary if the OUTPUT policy is not set to ACCEPT.

Allow MySQL to Specific Network Interface

To allow MySQL connections to a specific network interface—say server has a private network interface eth1, for example—use these commands:

```
iptables -A INPUT -i eth1 -p tcp --dport 3306 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
```

```
iptables -A OUTPUT -o eth1 -p tcp --sport 3306 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

Note: Default Policy for INPUT and OUTPUT chain is considered as DROP for all examples.

Conclusion: Hence we have successfully studied commands that are commonly used when configuring an iptables firewall and also configured a linux machine as Firewall(iptables). iptables is a very flexible tool that allows to mix and match the commands with different options to match specific needs .