Ayub Roti
Technical Solutions Engineer, Serianu Limited.
The Content Herein is automatically generated

Linux: 25 Iptables Netfilter Firewall Examples For New SysAdmins - nixCraft

Content Courtesy of
https://www.cyberciti.biz/tips/linux-iptables-examples.html

(//www.cyberciti.biz/tips/category/iptables)

Linux comes with a host based firewall called Netfilter. The netfilter is a set of hooks inside the Linux kernel that allows kernel modules to register callback functions with the network stack. A registered callback function is then called back for every packet that traverses the respective hook within the network stack. This Linux based firewall is controlled by the program called iptables to handles filtering for IPv4, and ip6tables handles filtering for IPv6. I strongly recommend that you first read our quick tutorial that explains how to configure a host-based firewall called Netfilter (//www.cyberciti.biz/faq/rhel-fedorta-linux-iptables-firewall-configuration-tutorial/) (iptables) under CentOS / RHEL / Fedora / Redhat Enterprise Linux. If you are using Ubuntu/Debian Linux, see how to setup UFW for (https://www.cyberciti.biz/faq/howto-configure-setup-firewall-with-ufw-on-ubuntu-linux/) more info. This post lists most simple iptables solutions required by a new Linux user to secure his or her Linux operating system from intruders.

Linux IPTables Firewall Rules and Commands For New Sysadmins
www.cyberciti.biz
(https://www.cyberciti.biz/tips/wp-content/uploads/2011/12/iptables-Essentials-Firewall-Rules-and-Commands-for-Linux-sysadmin-1.jpg)

This guide shows essential iptables command to control your daily life firewall rules and security of Linux server (https://www.cyberciti.biz/tips/linux-security.html) running on the bare metal server, router, or cloud server.

supporter using Patreon (https://www.patreon.com/nixcraft)

.

Linux Iptables Netfilter Firewall Examples For New SysAdmins

- Most of the actions listed in this post written with the assumption that they will be executed by the root user running the bash or any other modern shell. Do not type commands on the remote system as it will disconnect your access.
- For demonstration purpose, I've used RHEL 6.x, but the following command should work with any modern Linux distro that use the netfliter.
- It is NOT a tutorial on how to set iptables. See tutorial here (//www.cyberciti.biz/faq/rhel-fedorta-linux-iptables-firewall-configuration-tutorial/). It is a quick cheat sheet to common iptables commands.

1. Displaying the Status of Your Firewall

Type the following command as root:
`# iptables -L -n -v`
Sample outputs:

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Above output indicates that the firewall is not active. The following sample shows an active firewall:
`# iptables -L -n -v`
Sample outputs:

```
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0           state INVALID
  394 43586 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHE
D
   93 17292 ACCEPT     all  --  br0    *       0.0.0.0/0            0.0.0.0/0
    1   142 ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     all  --  br0    br0     0.0.0.0/0            0.0.0.0/0
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0           state INVALID
    0     0 TCPMSS     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp flags:0x06/0x02 TCPMS
S clamp to PMTU
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHE
D
    0     0 wanin      all  --  vlan2  *       0.0.0.0/0            0.0.0.0/0
    0     0 wanout     all  --  *      vlan2   0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  br0    *       0.0.0.0/0            0.0.0.0/0

Chain OUTPUT (policy ACCEPT 425 packets, 113K bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain wanin (1 references)
 pkts bytes target     prot opt in     out     source               destination

Chain wanout (1 references)
 pkts bytes target     prot opt in     out     source               destination
```

Where,

- **-L** : List rules.
- **-v** : Display detailed information. This option makes the list command show the interface name, the rule options, and the TOS masks. The packet and byte counters are also listed, with the suffix 'K', 'M' or 'G' for 1000, 1,000,000 and 1,000,000,000 multipliers respectively.
- **-n** : Display IP address and port in numeric format. Do not use DNS to resolve names. This will speed up listing.

1.1. To inspect firewall with line numbers, enter:

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

Sample outputs:

```
Chain INPUT (policy DROP)
num  target     prot opt source               destination
1    DROP       all  --  0.0.0.0/0            0.0.0.0/0           state INVALID
2    ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
3    ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
4    ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP)
num  target     prot opt source               destination
1    ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
2    DROP       all  --  0.0.0.0/0            0.0.0.0/0           state INVALID
3    TCPMSS     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp flags:0x06/0x02 TCPMSS clamp to PMTU
4    ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
5    wanin      all  --  0.0.0.0/0            0.0.0.0/0
6    wanout     all  --  0.0.0.0/0            0.0.0.0/0
7    ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination

Chain wanin (1 references)
num  target     prot opt source               destination

Chain wanout (1 references)
num  target     prot opt source               destination
```

You can use line numbers to delete or insert new rules into the firewall.

1.2. To display INPUT or OUTPUT chain rules, enter:

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

2. Stop / Start / Restart the Firewall

If you are using CentOS / RHEL / Fedora Linux, enter:

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

You can use the iptables command itself to stop the firewall and delete all rules:

```
# 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 (called nat or mangle) and delete/flush rules.
- **-P** : Set the default policy (such as DROP, REJECT, or ACCEPT).

3. Delete Firewall Rules

To display line number along with other information for existing rules, enter:

```
# iptables -L INPUT -n --line-numbers
# iptables -L OUTPUT -n --line-numbers
# iptables -L OUTPUT -n --line-numbers | less
# iptables -L OUTPUT -n --line-numbers | grep 202.54.1.1
```

You will get the list of IP. Look at the number on the left, then use number to delete it (https://www.cyberciti.biz/faq/how-to-iptables-delete-postrouting-rule/). For example delete line number 4, enter:

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

OR 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

4. Insert Firewall Rules

To insert one or more rules in the selected chain as the given rule number use the following syntax. First find out line numbers, enter:
# iptables -L INPUT -n –line-numbers
Sample outputs:

```
Chain INPUT (policy DROP)
num  target     prot opt source               destination
1    DROP       all  --  202.54.1.1           0.0.0.0/0
2    ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0           state NEW,ESTABLISHED
```

To insert rule between 1 and 2, enter:

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

To view updated rules, enter:

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

Sample outputs:

```
Chain INPUT (policy DROP)
num  target     prot opt source               destination
1    DROP       all  --  202.54.1.1           0.0.0.0/0
2    DROP       all  --  202.54.1.2           0.0.0.0/0
3    ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0           state NEW,ESTABLISHED
```

5. Save Firewall Rules

To save firewall rules under CentOS / RHEL / Fedora Linux, enter:

```
# service iptables save
```

In this example, drop an IP and save firewall rules:

```
# iptables -A INPUT -s 202.5.4.1 -j DROP
# service iptables save
```

For all other distros use the iptables-save command:

```
# iptables-save > /root/my.active.firewall.rules
# cat /root/my.active.firewall.rules
```

6. Restore Firewall Rules

To restore firewall rules form a file called /root/my.active.firewall.rules, enter:

```
# iptables-restore < /root/my.active.firewall.rules
```

To restore firewall rules under CentOS / RHEL / Fedora Linux, enter:

```
# service iptables restart
```

7. Set the Default Firewall Policies

To drop all traffic:

```
# iptables -P INPUT DROP
# iptables -P OUTPUT DROP
# iptables -P FORWARD DROP
# iptables -L -v -n
#### you will not able to connect anywhere as all traffic is dropped ###
# ping cyberciti.biz
# wget http://www.kernel.org/pub/linux/kernel/v3.0/testing/linux-3.2-rc5.tar.bz2
```

7.1. Only Block Incoming Traffic

To drop all incoming / forwarded packets, but allow outgoing traffic, enter:

```
# iptables -P INPUT DROP
# iptables -P FORWARD DROP
# iptables -P OUTPUT ACCEPT
# iptables -A INPUT -m state --state NEW,ESTABLISHED -j ACCEPT
# iptables -L -v -n
### *** now ping and wget should work *** ###
# ping cyberciti.biz
# wget http://www.kernel.org/pub/linux/kernel/v3.0/testing/linux-3.2-rc5.tar.bz2
```

8. 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
```

8.1. IPv4 Address Ranges For Private Networks (make sure you block them on public interface)

- 10.0.0.0/8 -j (A)
- 172.16.0.0/12 (B)
- 192.168.0.0/16 (C)
- 224.0.0.0/4 (MULTICAST D)
- 240.0.0.0/5 (E)
- 127.0.0.0/8 (LOOPBACK)

9. Blocking an IP Address (BLOCK IP)

To block an attackers ip address called 1.2.3.4, enter:

```
# iptables -A INPUT -s 1.2.3.4 -j DROP
# iptables -A INPUT -s 192.168.0.0/24 -j DROP
```

10. Block Incoming Port Requests (BLOCK PORT)

To block all service requests on port 80, enter:

```
# iptables -A INPUT -p tcp --dport 80 -j DROP
# iptables -A INPUT -i eth1 -p tcp --dport 80 -j DROP
```

To block port 80 only for an ip address 1.2.3.4, enter:

```
# iptables -A INPUT -p tcp -s 1.2.3.4 --dport 80 -j DROP
# iptables -A INPUT -i eth1 -p tcp -s 192.168.1.0/24 --dport 80 -j DROP
```

11. Block Outgoing IP Address

To block outgoing traffic to a particular host or domain such as cyberciti.biz, enter:

```
# host -t a cyberciti.biz
```

Sample outputs:

```
cyberciti.biz has address 75.126.153.206
```

Note down its ip address and type the following to block all outgoing traffic to 75.126.153.206:

```
# iptables -A OUTPUT -d 75.126.153.206 -j DROP
```

You can use a subnet as follows:

```
# iptables -A OUTPUT -d 192.168.1.0/24 -j DROP
# iptables -A OUTPUT -o eth1 -d 192.168.1.0/24 -j DROP
```

11.1. Example - Block Facebook.com Domain

First, find out all ip address of facebook.com, enter:

```
# host -t a www.facebook.com
```

Sample outputs:

```
www.facebook.com has address 69.171.228.40
```

Find CIDR for 69.171.228.40, enter:

```
# whois 69.171.228.40 | grep CIDR
```

Sample outputs:

```
CIDR:           69.171.224.0/19
```

To prevent outgoing access to www.facebook.com, enter:

```
# iptables -A OUTPUT -p tcp -d 69.171.224.0/19 -j DROP
```

You can also use domain name, enter:

```
# iptables -A OUTPUT -p tcp -d www.facebook.com -j DROP
# iptables -A OUTPUT -p tcp -d facebook.com -j DROP
```

From the iptables man page:

> ... specifying any name to be resolved with a remote query such as DNS (e.g., facebook.com is a really bad idea), a network IP address (with /mask), or a plain IP address ...

12. Log and Drop Packets

Type the following to log and block IP spoofing on public interface called eth1

```
# 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
```

By default everything is logged to /var/log/messages file.

```
# tail -f /var/log/messages
# grep --color 'IP SPOOF' /var/log/messages
```

13. Log and Drop Packets with Limited Number of Log Entries

The -m limit module can limit the number of log entries created per time. This is used to prevent flooding your log file. To log and drop spoofing per 5 minutes, in bursts of at most 7 entries .

```
# iptables -A INPUT -i eth1 -s 10.0.0.0/8 -m limit --limit 5/m --limit-burst 7 -j LOG --log-prefix "
# iptables -A INPUT -i eth1 -s 10.0.0.0/8 -j DROP
```

14. Drop or Accept Traffic From Mac Address

Use the following syntax:

```
# iptables -A INPUT -m mac --mac-source 00:0F:EA:91:04:08 -j DROP
## *only accept traffic for TCP port # 8080 from mac 00:0F:EA:91:04:07 * ##
# iptables -A INPUT -p tcp --destination-port 22 -m mac --mac-source 00:0F:EA:91:04:07 -j ACCEPT
```

15. Block or Allow ICMP Ping Request

Type the following command to block ICMP ping requests:

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

Ping responses can also be limited to certain networks or hosts:

```
# iptables -A INPUT -s 192.168.1.0/24 -p icmp --icmp-type echo-request -j ACCEPT
```

The following only accepts limited type of ICMP requests:

```
### ** assumed that default INPUT policy set to DROP ** #############
iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT
iptables -A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT
iptables -A INPUT -p icmp --icmp-type time-exceeded -j ACCEPT
## ** all our server to respond to pings ** ##
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
```

16. Open Range of Ports

Use the following syntax to open a range of ports:

```
iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 7000:7010 -j ACCEPT
```

17. Open Range of IP Addresses

Use the following syntax to open a range of IP address:

```
## only accept connection to tcp port 80 (Apache) if ip is between 192.168.1.100 and 192.168.1.200
iptables -A INPUT -p tcp --destination-port 80 -m iprange --src-range 192.168.1.100-192.168.1.200 -
```

## nat example ##
iptables -t nat -A POSTROUTING -j SNAT --to-source 192.168.1.20-192.168.1.25

18. Established Connections and Restarting The Firewall

When you restart the iptables service it will drop established connections as it unload modules from the system under RHEL / Fedora / CentOS Linux. Edit, /etc/sysconfig/iptables-config and set IPTABLES_MODULES_UNLOAD as follows:

```
IPTABLES_MODULES_UNLOAD = no
```

19. Help Iptables Flooding My Server Screen

Use the crit log level to send messages to a log file instead of console:

```
iptables -A INPUT -s 1.2.3.4 -p tcp --destination-port 80 -j LOG --log-level crit
```

20. Block or Open Common Ports

The following shows syntax for opening and closing common TCP and UDP ports:

```
Replace ACCEPT with DROP to block port:
## open port ssh tcp port 22 ##
iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 22 -j ACCEPT

## open cups (printing service) udp/tcp port 631 for LAN users ##
iptables -A INPUT -s 192.168.1.0/24 -p udp -m udp --dport 631 -j ACCEPT
iptables -A INPUT -s 192.168.1.0/24 -p tcp -m tcp --dport 631 -j ACCEPT

## allow time sync via NTP for lan users (open udp port 123) ##
iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p udp --dport 123 -j ACCEPT

## open tcp port 25 (smtp) for all ##
iptables -A INPUT -m state --state NEW -p tcp --dport 25 -j ACCEPT

# open dns server ports for all ##
iptables -A INPUT -m state --state NEW -p udp --dport 53 -j ACCEPT
iptables -A INPUT -m state --state NEW -p tcp --dport 53 -j ACCEPT

## open http/https (Apache) server port to all ##
iptables -A INPUT -m state --state NEW -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -m state --state NEW -p tcp --dport 443 -j ACCEPT

## open tcp port 110 (pop3) for all ##
iptables -A INPUT -m state --state NEW -p tcp --dport 110 -j ACCEPT

## open tcp port 143 (imap) for all ##
iptables -A INPUT -m state --state NEW -p tcp --dport 143 -j ACCEPT

## open access to Samba file server for lan users only ##
iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 137 -j ACCEPT
iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 138 -j ACCEPT
iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 139 -j ACCEPT
iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 445 -j ACCEPT

## open access to proxy server for lan users only ##
iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 3128 -j ACCEPT

## open access to mysql server for lan users only ##
iptables -I INPUT -p tcp --dport 3306 -j ACCEPT
```

Replace ACCEPT with DROP to block port: ## open port ssh tcp port 22 ## iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 22 -j ACCEPT ## open cups (printing service) udp/tcp port 631 for LAN users ## iptables -A INPUT -s 192.168.1.0/24 -p udp -m udp --dport 631 -j ACCEPT iptables -A INPUT -s 192.168.1.0/24 -p tcp -m tcp --dport 631 -j ACCEPT ## allow time sync via NTP for lan users (open udp port 123) ## iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p udp --dport 123 -j ACCEPT ## open tcp port 25 (smtp) for all ## iptables -A INPUT -m state --state NEW -p tcp --dport 25 -j ACCEPT # open dns server ports for all ## iptables -A INPUT -m state --state NEW -p udp --dport 53 -j ACCEPT iptables -A INPUT -m state --state NEW -p tcp --dport 53 -j ACCEPT ## open http/https (Apache) server port to all ## iptables -A INPUT -m state --state NEW -p tcp --dport 80 -j ACCEPT iptables -A INPUT -m state --state NEW -p tcp --dport 443 -j ACCEPT ## open tcp port 110 (pop3) for all ## iptables -A INPUT -m state --state NEW -p tcp --dport 110 -j ACCEPT ## open tcp port 143 (imap) for all ## iptables -A INPUT -m state --state NEW -p tcp --dport 143 -j ACCEPT ## open access to Samba file server for lan users only ## iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 137 -j ACCEPT iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW

-p tcp --dport 138 -j ACCEPT iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 139 -j ACCEPT iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 445 -j ACCEPT ## open access to proxy server for lan users only ## iptables -A INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 3128 -j ACCEPT ## open access to mysql server for lan users only ## iptables -I INPUT -p tcp --dport 3306 -j ACCEPT

21. Restrict the Number of Parallel Connections To a Server Per Client IP

You can use connlimit module to put such restrictions. To allow 3 ssh connections per client host, enter:

```
# iptables -A INPUT -p tcp --syn --dport 22 -m connlimit --connlimit-above 3 -j REJECT
```

Set HTTP requests to 20:

```
# iptables -p tcp --syn --dport 80 -m connlimit --connlimit-above 20 --connlimit-mask 24 -j DROP
```

Where,

1. **--connlimit-above 3** : Match if the number of existing connections is above 3.
2. **--connlimit-mask 24** : Group hosts using the prefix length. For IPv4, this must be a number between (including) 0 and 32.

22. List NAT rules

The syntax is

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

Sample outputs:

```
Chain PREROUTING (policy ACCEPT 496K packets, 29M bytes)
 pkts bytes target     prot opt in     out     source               destination
43557 2613K DNAT       tcp  --  *      *       0.0.0.0/0            192.168.184.8        tcp dpt:443 to:10.105.2
8.42:443
68700 4122K DNAT       tcp  --  *      *       0.0.0.0/0            192.168.184.8        tcp dpt:80 to:10.105.2
8.42:80
15855  951K DNAT       tcp  --  *      *       0.0.0.0/0            192.168.184.8        tcp dpt:444 to:10.105.2
8.45:444
16009  961K DNAT       tcp  --  *      *       0.0.0.0/0            192.168.184.8        tcp dpt:81 to:10.105.2
8.45:81
63495 3810K DNAT       tcp  --  *      *       0.0.0.0/0            192.168.184.8        tcp dpt:445 to:10.105.2
8.44:445
19615 1177K DNAT       tcp  --  *      *       0.0.0.0/0            192.168.184.8        tcp dpt:82 to:10.105.2
8.44:82

Chain INPUT (policy ACCEPT 488K packets, 29M bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 3280 packets, 207K bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain POSTROUTING (policy ACCEPT 231K packets, 14M bytes)
 pkts bytes target     prot opt in     out     source               destination
 3832  230K MASQUERADE  all  --  *      *       10.105.28.0/24      !10.105.28.0/24       /* generated for LXD n
etwork lxdbr0 */
```

Chain PREROUTING (policy ACCEPT 496K packets, 29M bytes) pkts bytes target prot opt in out source destination 43557 2613K DNAT tcp -- * * 0.0.0.0/0 192.168.184.8 tcp dpt:443 to:10.105.28.42:443 68700 4122K DNAT tcp -- * * 0.0.0.0/0 192.168.184.8 tcp dpt:80 to:10.105.28.42:80 15855 951K DNAT tcp -- * * 0.0.0.0/0 192.168.184.8 tcp dpt:444 to:10.105.28.45:444 16009 961K DNAT tcp -- * * 0.0.0.0/0 192.168.184.8 tcp dpt:81 to:10.105.28.45:81 63495 3810K DNAT tcp -- * * 0.0.0.0/0 192.168.184.8 tcp dpt:445 to:10.105.28.44:445 19615 1177K DNAT tcp -- * * 0.0.0.0/0 192.168.184.8 tcp dpt:82 to:10.105.28.44:82 Chain INPUT (policy ACCEPT 488K packets, 29M bytes) pkts bytes target prot opt in out source destination Chain OUTPUT (policy ACCEPT 3280 packets, 207K bytes) pkts bytes target prot opt in out source destination Chain POSTROUTING (policy ACCEPT 231K packets, 14M bytes) pkts bytes target prot opt in out source destination 3832 230K MASQUERADE all -- * * 10.105.28.0/24 !10.105.28.0/24 /* generated for LXD network lxdbr0 */

Another option:

```
# iptables -t nat -v -L -n --line-number
```

23. Delete NAT rules

The syntax is as follows to list NAT rules on Linux (https://www.cyberciti.biz/faq/how-to-iptables-delete-postrouting-rule/iptables-list-postrouting-rules/):

```
# iptables -t nat -v -L -n --line-number
# iptables -t nat -v -L PREROUTING -n --line-number
# iptables -t nat -v -L POSTROUTING -n --line-number
```

To delete PREROUTING rule, run:

```
# iptables -t nat -D PREROUTING {number-here}
# iptables -t nat -D PREROUTING 42
```

To delete POSTROUTING rule (https://www.cyberciti.biz/faq/how-to-iptables-delete-postrouting-rule/), run:

```
# iptables -t nat -D POSTROUTING {number-here}
# iptables -t nat -D POSTROUTING 42
```

24. How to redirect port AA to BB

The syntax is as follows (https://www.cyberciti.biz/faq/linux-port-redirection-with-iptables/):

```
iptables -t nat -A PREROUTING -i $interfaceName -p tcp --dport $srcPortNumber -j REDIRECT --to-port
```

To redirect all incoming traffic on port 80 redirect to port 8080

```
# iptables -t nat -I PREROUTING --src 0/0 --dst 192.168.1.5 -p tcp --dport 80 -j REDIRECT --to-port
```

25. How to reset packet counters

To see iptables counters run:

```
# iptables -L -n -v
```

To clear/reset the counters for all rules:

```
# iptables -Z
# iptables -L -n -v
```

To reset the counters for INPUT chain only:

```
# iptables -Z INPUT
```

To reset the counters for rule # 13 in the INPUT chain only:

```
# iptables -Z INPUT 13
```

26. HowTO: Use iptables Like a Pro

For more information about iptables, please see the manual page by typing man iptables from the command line:

```
$ man iptables
```

You can see the help using the following syntax too:

```
# iptables -h
```

To see help with specific commands and targets, enter:

```
# iptables -j DROP -h
```

27. Testing Your Firewall

Find out if ports are open or not, enter:

```
# netstat -tulpn
```

Find out if tcp port 80 open or not, enter:

```
# netstat -tulpn | grep :80
```

If port 80 is not open, start the Apache, enter:

```
# service httpd start
```

Make sure iptables allowing access to the port 80:

```
# iptables -L INPUT -v -n | grep 80
```

Otherwise open port 80 using the iptables for all users:

```
# iptables -A INPUT -m state --state NEW -p tcp --dport 80 -j ACCEPT
# service iptables save
```

Use the telnet command to see if firewall allows to connect to port 80:

```
$ telnet www.cyberciti.biz 80
```

Sample outputs:

```
Trying 75.126.153.206...
Connected to www.cyberciti.biz.
Escape character is '^]'.
^]

telnet> quit
Connection closed.
```

You can use the nmap command (https://www.cyberciti.biz/networking/nmap-command-examples-tutorials/) to probe your own server using the following syntax:

```
$ nmap -sS -p 80 www.cyberciti.biz
```

Sample outputs:

```
Starting Nmap 5.00 ( http://nmap.org ) at 2011-12-13 13:19 IST
Interesting ports on www.cyberciti.biz (75.126.153.206):
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 1.00 seconds
```

I also recommend you install and use sniffer such as tcpdupm and ngrep to test your firewall settings.

Conclusion:

This post only list basic rules for new Linux users. You can create and build more complex rules. This requires good understanding of TCP/IP, Linux kernel tuning via sysctl.conf, and good knowledge of your own setup. Stay tuned for next topics:

- Stateful packet inspection.
- Using connection tracking helpers.
- Network address translation.
- Layer 2 filtering.
- Firewall testing tools.
- Dealing with VPNs, DNS, Web, Proxy, and other protocols.

Posted by: Vivek Gite

The author is the creator of nixCraft and a seasoned sysadmin, DevOps engineer, and a trainer for the Linux operating system/Unix shell scripting. Get the **latest tutorials on SysAdmin, Linux/Unix and open source topics via RSS/XML feed (https://www.cyberciti.biz/atom/atom.xml)** or weekly email newsletter (https://www.cyberciti.biz/subscribe-to-weekly-linux-unix-newsletter-for-sysadmin/).

Linux iptables Firewall Simplified Examples - Like Geeks

Content Courtesy of
https://likegeeks.com/linux-iptables-firewall-examples/
In the previous post, we talked about how to Secure Linux Server Using Hardening Best Practices (https://likegeeks.com/secure-linux-server-hardening-best-practices/), some people asked me about the firewall section which was a brief introduction about iptables firewall. Today we will discuss in detail the **Linux iptables firewall** and how to secure your server traffic using that awesome firewall.

Iptables on CentOS 7

If you are using CentOS 7, you will find that firewalld was introduced to manage iptables, so if you want to go back to iptables, you have to stop and mask firewalld.

```
$ systemctl stop firewalld
```

```
$ systemctl mask firewalld
```

Then install iptables service and enable it:

```
$ yum install iptables-services
```

```
$ systemctl enable iptables
```

Then you can start it:

```
$ systemctl start iptables
```

How Linux Firewall Works

Iptables firewall functions are built on Netfilter framework that is available in the Linux kernel for packets filtering.

Firewall Types

There are two types of firewalls:

Stateless firewall process each packet on its own, it means it doesn't see other packets of the same connection.

Stateful firewall this type of firewalls cares about all packets passed through it, so it knows the state of te connection. It gives more control over the traffic.

Netfilter contains **tables**. These tables contain **chains**, and chains contain individual **rules**.

If the passed packet matches any rule, the rule **action** will be applied on that packet.

The actions can be: **accept**, **reject**, **ignore,** or **pass** the packet on to other rules for more processing.

Netfilter can process incoming or outgoing traffic using the IP address and port number

Netfilter is managed and configured by the iptables command.

Before we start writing firewall commands, we need to understand the firewall structure a bit so we can write firewall rules easily.

iptables Firewall Tables

Netfilter has three tables that can carry rules for processing.

The iptables **filter table** is the main table for processing the traffic.

The second is **nat table**, which handles NAT rules.

The third table is the **mangle table**, which is used for mangling packets.

Table Chains

Each table of the above-mentioned tables contains chains, these chains are the container of the rules of iptables.

The filter table contains **FORWARD, INPUT**, and **OUTPUT** chains.

You can create a custom chain to save your rules on it.

If a packet is **coming to the host**, it will be processed by **INPUT chain** rules.

If the packet is **going to another host**, that means it will be processed by **OUTPUT chain** rules.

The iptables **FORWARD chain** is used for handling packets that have **accessed the host** but are destined to another host.

Chain Policy

Each chain in the filter table has a policy. The policy is the default action taken.

The policy could be **DROP, REJECT**, and **ACCEPT**.

The ACCEPT policy allows the packets to pass the firewall. The DROP policy drops a packet without informing the client. The REJECT policy also drops the packet and inform the sender.

From a security perspective, you should drop all the packets coming to the host and accept only the packets that come from trusted sources.

Adding iptables Rules

You can add a new rule using the iptables command like this:

$ iptables -A INPUT -i eth1 -p tcp --dport 80 -d 1.2.3.4 -j ACCEPT

Let's break this command into pieces so we can understand everything about it.

The -A means we are adding a new rule. By default, all new rules are added to **filter table** unless you specify another table.

The -i flag means which device will be used for the traffic to enter the host. If no device specified, the rule will be applied to all incoming traffic regardless the devices.

The -p flag specifies the packet's protocol that you want to process, which is TCP in our case.

The –dport flag specifies the destination port, which is 80.

The -d specifies the destination IP address which is 1.2.3.4. If no destination IP address specified, the rule would apply to all incoming traffic on eth1 regardless of IP address.

The -j specifies the action or the **JUMP** action to do, here we are accepting the packets using the accept policy.

The above rule allows incoming HTTP traffic which is on port 80.

What about allowing outgoing traffic?

```
$ iptables -A OUTPUT -o eth1 -p tcp --sport 80 -j ACCEPT
```

The -A flag is used to add rules to the OUTPUT chain.

The -o flag is used for the device used for outgoing traffic.

The -sport flag specifies the source port.

You can use the service name like http or https instead of the numeric port number on sport or dport. The service names can be found in /etc/services file.

It is recommended to use the service name rather than a port number, which makes reading rules easier.

Iptables Rules Order

When you add a rule, it is added to the end of the chain.

You can add it on the top by using -I option.

The sequence of the rules matters as you will see now.

You can insert your rules exactly where you want using the I flag.

Look at the following rules to understand how rules ordering matters:

```
$ iptables -I INPUT 3 -i eth1 -p udp -j ACCEPT

$ iptables -I INPUT 4 -i eth1 -p udp --dport 80 -j DROP
```

The first rule accepts all UDP traffic comes to eth1, and the number 3 is the rule order.

The second rule drops the traffic that enters port 80.

The first rule will accept all the traffic, then the second rule will be ignored because the first rule already accepts all the traffic so the second rule here makes no sense.

Your rules should make sense since the order of the rules in the chain matters.

List iptables Rules

You can list the rules in a chain using -L flag:

```
$ iptables -L INPUT
```

You can show the line numbers for rules using –line-numbers:

```
$ iptables -L INPUT --line-numbers
```

The list shows the services names, you can show port numbers instead using -n option:

```
$ iptables -L INPUT -n --line-numbers
```

This will make the listing faster because it prevents iptables from DNS (https://likegeeks.com/linux-dns-server/) resolution and service lookups.

You can list all rules for all chains like this:

```
$ iptables -L -n --line-numbers
```

To get how many packets processed by each rule, you can use the -v flag:

```
$ iptables -L -v
```

Also, you can reset the counters to zero using -Z flag.

Now we can add a new rule to any chain we want, we can insert the rule in a specific order and we can list the rules for any chain or all chains, but what about deleting a rule?

Deleting Rules

You can delete a rule using -D flag:

```
$ iptables -D INPUT -i eth1 -p tcp --dport 80 -d 1.2.3.4 -j ACCEPT
```

This command will delete the HTTP rule that you specified earlier.

Before you delete a rule, just make sure of the rule specification by listing it, then delete it.

You can delete the rule using the order number instead of writing the rule specifications.

```
$ iptables -D INPUT 2
```

You can delete all rules in a specific chain using -F flag which means flush all rules.

```
$ iptables -F INPUT
```

If you forget to mention the chain name when using -F flag, then all chain rules will be deleted.

Replacing Rules

You can replace existing rules with your own rule using -R flag:

```
$ iptables -R INPUT 1 -i eth1 -p tcp --dport httpht -d 1.2.3.4 -j ACCEPT
```

This command will replace the first rule in INPUT chain with the typed rule.

Listing Specific Table

To list a specific table, use the -t flag with the table name like this:

```
$ iptables -L -t nat
```

Here we list the rules in nat table.

Iptables User Defined Chain

To create a user defined chain, use the -N flag.

$ iptables -N MY_CHAIN

Also, you can rename it using -E flag.

$ iptables -E MY_CHAIN NEW_NAME

And you can delete the user-defined chain using -X flag.

$ iptables -X MY_CHAIN

If you don't mention the chain name when using -X flag it will delete all user-defined chains. You can't delete built-in chains like INPUT and OUTPUT.

Redirection to a User Defined Chain

You can redirect packets to a user-defined chain like built-in chains using -j flag.

$ iptables -A INPUT -p icmp -j MY_CHAIN

So all incoming ICMP traffic will be redirected to the newly created chain called MY_CHAIN. (https://www.ezoic.com/what-is-ezoic/)

(https://www.ezoic.com/what-is-ezoic/)

Setting The Default Policy for Chains (https://www.ezoic.com/what-is-ezoic/)

You can use the -P flag to set the default policy for a specific chain. The default policy could be ACCEPT, REJECT and DROP. (https://www.ezoic.com/what-is-ezoic/)

$ iptables -P INPUT DROP (https://www.ezoic.com/what-is-ezoic/)

So now the input chain will drop any packet come unless you write a rule to allow any incoming traffic. (https://www.ezoic.com/what-is-ezoic/)

(https://www.ezoic.com/what-is-ezoic/)

SYN Flooding (https://www.ezoic.com/what-is-ezoic/)

The attacker sends SYN packets only without completing the TCP handshake and as a result, the receiving host would have many opened connections, and your server becomes too busy to respond to other clients. (https://www.ezoic.com/what-is-ezoic/)

We can use the limit module of iptables firewall to protect us from SYN flooding. (https://www.ezoic.com/what-is-ezoic/)

$ iptables -A INPUT -i eth1 -p tcp --syn -m limit --limit 10/second -j ACCEPT (https://www.ezoic.com/what-is-ezoic/)

Here we specify 10 SYN packets per second only. You can adjust this value according to your network needs. (https://www.ezoic.com/what-is-ezoic/)

If this will throttle your network, you can use SYN cookies. (https://www.ezoic.com/what-is-ezoic/)

SYN Cookies (https://www.ezoic.com/what-is-ezoic/)

In /etc/sysctl.conf file and add this line: (https://www.ezoic.com/what-is-ezoic/)

net.ipv4.tcp_syncookies = 1 (https://www.ezoic.com/what-is-ezoic/)

Then save and reload. (https://www.ezoic.com/what-is-ezoic/)

$ sysctl -p (https://www.ezoic.com/what-is-ezoic/)

(https://www.ezoic.com/what-is-ezoic/)

Drop INVALID State Packets (https://www.ezoic.com/what-is-ezoic/)

The INVALID state packets are packets that don't belong to any connection and should be dropped. (https://www.ezoic.com/what-is-ezoic/)

$ iptables -A INPUT -m state --state INVALID -j DROP (https://www.ezoic.com/what-is-ezoic/)

This rule will drop all incoming invalid state packets. (https://www.ezoic.com/what-is-ezoic/)

(https://www.ezoic.com/what-is-ezoic/)

Drop Fragmented Packets (https://www.ezoic.com/what-is-ezoic/)

Fragmented packets are broken pieces of large malformed packets and should be dropped (https://www.ezoic.com/what-is-ezoic/)

The -f flag tells iptables firewall to select all fragments. So if you are not using iptables as a router, you can drop fragmented packets. (https://www.ezoic.com/what-is-ezoic/)

$ iptables -A INPUT -f -j DROP (https://www.ezoic.com/what-is-ezoic/)

(https://www.ezoic.com/what-is-ezoic/)

Save iptables Rules (https://www.ezoic.com/what-is-ezoic/)

All the rules we discussed will be lost if you reboot your server, so how to persist them. (https://www.ezoic.com/what-is-ezoic/)

You can save all of your rules using the iptables-save command if you are using CentOS or Red Hat. (https://www.ezoic.com/what-is-ezoic/)

iptables-save > /etc/sysconfig/iptables (https://www.ezoic.com/what-is-ezoic/)

On CentOS 7, you can save rules like this: (https://www.ezoic.com/what-is-ezoic/)

$ service iptables save (https://www.ezoic.com/what-is-ezoic/)

You can save specific table like filter table only: (https://www.ezoic.com/what-is-ezoic/)

$ iptables-save -t filter (https://www.ezoic.com/what-is-ezoic/)

Also, you can use iptables-restore to restore rules that were saved. (https://www.ezoic.com/what-is-ezoic/)

On Debian based distros, you can use the iptables-persistent package to save and restore rules. (https://www.ezoic.com/what-is-ezoic/)

First, install it: (https://www.ezoic.com/what-is-ezoic/)

$ apt-get install iptables-persistent (https://www.ezoic.com/what-is-ezoic/)

Then you can save and restore rules: (https://www.ezoic.com/what-is-ezoic/)

$ netfilter-persistent save (https://www.ezoic.com/what-is-ezoic/)

$ netfilter-persistent reload (https://www.ezoic.com/what-is-ezoic/)

I Hope you find iptables firewall easy. Keep coming back. (https://www.ezoic.com/what-is-ezoic/)

Thank you. (https://www.ezoic.com/what-is-ezoic/)

(https://www.ezoic.com/what-is-ezoic/)

(https://www.ezoic.com/what-is-ezoic/)

(https://www.ezoic.com/what-is-ezoic/)

25 Useful IPtable Firewall Rules Every Linux Administrator Should Know

Content Courtesy of

https://www.tecmint.com/linux-iptables-firewall-rules-examples-commands/
(https://www.ezoic.com/what-is-ezoic/)
(https://www.ezoic.com/what-is-ezoic/)
(https://www.ezoic.com/what-is-ezoic/)
(https://www.ezoic.com/what-is-ezoic/)Managing network traffic (https://www.tecmint.com/manage-and-limit-downloadupload-bandwidth-with-trickle-in-linux/) is one of the toughest jobs a system administrators has to deal with. He must configure the firewall (https://www.tecmint.com/configure-iptables-firewall/) in such a way that it will meet the system and users requirements for both incoming and outgoing connections, without leaving the system vulnerable to attacks.

(https://www.tecmint.com/wp-content/uploads/2016/03/iptables-Firewall-Rules-for-Linux.png)
25 IPtables Firewall Rules for Linux

This is where `iptables` come in handy. **Iptables** is a Linux command line firewall that allows system administrators to manage incoming and outgoing traffic via a set of configurable table rules.

Iptables uses a set of tables which have chains that contain set of built-in or user defined rules. Thanks to them a system administrator can properly filter the network traffic of his system.

Per iptables manual, there are currently 3 types of tables:

1. `FILTER` – this is the default table, which contains the built in chains for:
   1. **INPUT** – packages destined for local sockets
   2. **FORWARD** – packets routed through the system
   3. **OUTPUT** – packets generated locally
2. `NAT` – a table that is consulted when a packet tries to create a new connection. It has the following built-in:
   1. **PREROUTING** – used for altering a packet as soon as it’s received
   2. **OUTPUT** – used for altering locally generated packets
   3. **POSTROUTING** – used for altering packets as they are about to go out
3. `MANGLE` – this table is used for packet altering. Until kernel version **2.4** this table had only two chains, but they are now 5:
   1. **PREROUTING** – for altering incoming connections
   2. **OUTPUT** – for altering locally generated packets
   3. **INPUT** – for incoming packets
   4. **POSTROUTING** – for altering packets as they are about to go out
   5. **FORWARD** – for packets routed through the box

In this article, you will see some useful commands that will help you manage your Linux box firewall through iptables. For the purpose of this article, I will start with simpler commands and go to more complex to the end.

1. Start/Stop/Restart Iptables Firewall

First, you should know how to manage iptables service in different Linux distributions. This is fairly easy:

On SystemD based Linux Distributions

```
------------ On Cent/RHEL 7 and Fedora 22+ ------------
# systemctl start iptables
# systemctl stop iptables
# systemctl restart iptables
```

On SysVinit based Linux Distributions

```
------------ On Cent/RHEL 6/5 and Fedora ------------
# /etc/init.d/iptables start
# /etc/init.d/iptables stop
# /etc/init.d/iptables restart
```

2. Check all IPtables Firewall Rules

If you want to check your existing rules, use the following command:

```
# iptables -L -n -v
```

This should return output similar to the one below:

```
Chain INPUT (policy ACCEPT 1129K packets, 415M bytes)
 pkts bytes target prot opt in out source destination
 0 0 ACCEPT tcp -- lxcbr0 * 0.0.0.0/0 0.0.0.0/0 tcp dpt:53
 0 0 ACCEPT udp -- lxcbr0 * 0.0.0.0/0 0.0.0.0/0 udp dpt:53
 0 0 ACCEPT tcp -- lxcbr0 * 0.0.0.0/0 0.0.0.0/0 tcp dpt:67
 0 0 ACCEPT udp -- lxcbr0 * 0.0.0.0/0 0.0.0.0/0 udp dpt:67
```

```
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target prot opt in out source destination
 0 0 ACCEPT all -- * lxcbr0 0.0.0.0/0 0.0.0.0/0
 0 0 ACCEPT all -- lxcbr0 * 0.0.0.0/0 0.0.0.0/0
```

```
Chain OUTPUT (policy ACCEPT 354K packets, 185M bytes)
 pkts bytes target prot opt in out source destination
```

If you prefer to check the rules for a specific table, you can use the `-t` option followed by the table which you want to check. For example, to check the rules in the `NAT` table, you can use:

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

3. Block Specific IP Address in IPtables Firewall

If you find an unusual or abusive activity from an IP address you can block that IP address with the following rule:

```
# iptables -A INPUT -s xxx.xxx.xxx.xxx -j DROP
```

Where you need to change `"xxx.xxx.xxx.xxx"` with the actual IP address. Be very careful when running this command as you can accidentally block your own IP address. The `-A` option appends the rule in the end of the selected chain.

In case you only want to block **TCP** traffic from that IP address, you can use the `-p` option that specifies the protocol. That way the command will look like this:

```
# iptables -A INPUT -p tcp -s xxx.xxx.xxx.xxx -j DROP
```

4. Unblock IP Address in IPtables Firewall

If you have decided that you no longer want to block requests from specific IP address, you can delete the blocking rule with the following command:

```
# iptables -D INPUT -s xxx.xxx.xxx.xxx -j DROP
```

The `-D` option deletes one or more rules from the selected chain. If you prefer to use the longer option you can use `--delete`.

5. Block Specific Port on IPtables Firewall

Sometimes you may want to block incoming or outgoing connections on a specific port. It's a good security measure and you should really think on that matter when setting up your firewall.

To block outgoing connections on a specific port use:

```
# iptables -A OUTPUT -p tcp --dport xxx -j DROP
```

To allow incoming connections use:

```
# iptables -A INPUT -p tcp --dport xxx -j ACCEPT
```

In both examples change `"xxx"` with the actual port you wish to allow. If you want to block **UDP** traffic instead of **TCP**, simply change `"tcp"` with `"udp"` in the above iptables rule.

6. Allow Multiple Ports on IPtables using Multiport

You can allow multiple ports at once, by using **multiport**, below you can find such rule for both incoming and outgoing connections:

```
# iptables -A INPUT  -p tcp -m multiport --dports 22,80,443 -j ACCEPT
# iptables -A OUTPUT -p tcp -m multiport --sports 22,80,443 -j ACCEPT
```

7. Allow Specific Network Range on Particular Port on IPtables

You may want to limit certain connections on specific port to a given network. Let's say you want to allow outgoing connections on port `22` to network `192.168.100.0/24`.

You can do it with this command:

```
# iptables -A OUTPUT -p tcp -d 192.168.100.0/24 --dport 22 -j ACCEPT
```

8. Block Facebook on IPtables Firewall

Some employers like to block access to **Facebook** to their employees. Below is an example how to block traffic to Facebook.

Note: If you are a system administrator and need to apply these rules, keep in mind that your colleagues may stop talking to you :)

First find the IP addresses used by Facebook:

```
# host facebook.com
facebook.com has address 66.220.156.68
```

```
# whois 66.220.156.68 | grep CIDR
CIDR: 66.220.144.0/20
```

You can then block that Facebook network with:

```
# iptables -A OUTPUT -p tcp -d 66.220.144.0/20 -j DROP
```

Keep in mind that the IP address range used by Facebook may vary in your country.

9. Setup Port Forwarding in IPtables

Sometimes you may want to forward one service's traffic to another port. You can achieve this with the following command:

```
# iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 25 -j REDIRECT --to-port 2525
```

The above command forwards all incoming traffic on network interface `eth0`, from port `25` to port `2525`. You may change the ports with the ones you need.

10. Block Network Flood on Apache Port with IPtables

Sometimes IP addresses may requests too many connections towards web ports on your website. This can cause number of issues and to prevent such problems, you can use the following rule:

```
# iptables -A INPUT -p tcp --dport 80 -m limit --limit 100/minute --limit-burst 200 -j ACCEPT
```

The above command limits the incoming connections from per minute to `100` and sets a limit burst to `200`. You can edit the limit and limit-burst to your own specific requirements.

11. Block Incoming Ping Requests on IPtables

Some system administrators like to block incoming ping requests due to security concerns. While the threat is not that big, it's good to know how to block such request:

```
# iptables -A INPUT -p icmp -i eth0 -j DROP
```

12. Allow loopback Access

Loopback access (access from `127.0.0.1`) is important and you should always leave it active:

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

13. Keep a Log of Dropped Network Packets on IPtables

If you want to log the dropped packets on network interface `eth0`, you can use the following command:

```
# iptables -A INPUT -i eth0 -j LOG --log-prefix "IPtables dropped packets:"
```

You can change the value after `"--log-prefix"` with something by your choice. The messages are logged in `/var/log/messages` and you can search for them with:

```
# grep "IPtables dropped packets:" /var/log/messages
```

14. Block Access to Specific MAC Address on IPtables

You can block access to your system from specific MAC address by using:

```
# iptables -A INPUT -m mac --mac-source 00:00:00:00:00:00 -j DROP
```

Of course, you will need to change `"00:00:00:00:00:00"` with the actual MAC address that you want to block.

15. Limit the Number of Concurrent Connections per IP Address

If you don't want to have too many concurrent connection established from single IP address on given port you can use the command below:

```
# iptables -A INPUT -p tcp --syn --dport 22 -m connlimit --connlimit-above 3 -j REJECT
```

The above command allows no more than `3` connections per client. Of course, you can change the port number to match different service. Also the `--connlimit-above` should be changed to match your requirement.

16. Search within IPtables Rule

Once you have defined your iptables rules, you will want to search from time to time and may need to alter them. An easy way to search within your rules is to use:

```
# iptables -L $table -v -n | grep $string
```

In the above example, you will need to change `$table` with the actual table within which you wish to search and `$string` with the actual string for which you are looking for.

Here is an example:

```
# iptables -L INPUT -v -n | grep 192.168.0.100
```

17. Define New IPTables Chain

With iptables, you can define your own chain and store custom rules in it. To define a chain, use:

```
# iptables -N custom-filter
```

Now you can check if your new filter is there:

```
# iptables -L
```

Sample Output

```
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
```

```
Chain custom-filter (0 references)
target prot opt source destination
```

18. Flush IPtables Firewall Chains or Rules

If you want to flush your firewall chains, you can use:

```
# iptables -F
```

You can flush chains from specific table with:

```
# iptables -t nat -F
```

You can change `"nat"` with the actual table which chains you wish to flush.

19. Save IPtables Rules to a File

If you want to save your firewall rules, you can use the `iptables-save` command. You can use the following to save and store your rules in a file:

```
# iptables-save > ~/iptables.rules
```

It's up to you where will you store the file and how you will name it.

20. Restore IPtables Rules from a File

If you want to restore a list of iptables rules, you can use `iptables-restore`. The command looks like this:

```
# iptables-restore < ~/iptables.rules
```

Of course the path to your rules file might be different.

21. Setup IPtables Rules for PCI Compliance

Some system administrators might be required to configure their servers to be PCI compiliant. There are many requirements by different PCI compliance vendors, but there are few common ones.

In many of the cases, you will need to have more than one IP address. You will need to apply the rules below for the site's IP address. Be extra careful when using the rules below and use them only if you are sure what you are doing:

```
# iptables -I INPUT -d SITE -p tcp -m multiport --dports 21,25,110,143,465,587,993,995 -j DROP
```

If you use cPanel or similar control panel, you may need to block it's' ports as well. Here is an example:

```
# iptables -I in_sg -d DEDI_IP -p tcp -m multiport --dports  2082,2083,2095,2096,2525,2086,2087 -j DROP
```

Note: To make sure you meet your PCI vendor's requirements, check their report carefully and apply the required rules. In some cases you may need to block UDP traffic on certain ports as well.

22. Allow Established and Related Connections

As the network traffic is separate on incoming and outgoing, you will want to allow established and related incoming traffic. For incoming connections do it with:

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

For outgoing use:

```
# iptables -A OUTPUT -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

23. Drop Invalid Packets in IPtables

It's possible to have some network packets marked as invalid. Some people may prefer to log those packages, but others prefer to drop them. To drop invalid the packets, you can use:

```
# iptables -A INPUT -m conntrack --ctstate INVALID -j DROP
```

24. Block Connection on Network Interface

Some systems may have more than one network interface. You can limit the access to that network interface or block connections from certain IP address.

For example:

```
# iptables -A INPUT -i eth0 -s xxx.xxx.xxx.xxx -j DROP
```

Change **“xxx.xxx.xxx.xxx”** with the actual IP address (or network) that you wish to block.

25. Disable Outgoing Mails through IPTables

If your system should not be sending any emails, you can block outgoing ports on SMTP ports. For example you can use this:

```
# iptables -A OUTPUT -p tcp --dports 25,465,587 -j REJECT
```

Conclusion

Iptables is a powerful firewall that you can easily benefit from. It is vital for every system administrator to learn at least the basics of iptables (https://www.tecmint.com/basic-guide-on-iptables-linux-firewall-tips-commands/). If you want to find more detailed information about iptables and its options it is highly recommend to read it’s manual:

```
# man iptables
```

If you think we should add more commands to this list, please share them with us, by submitting them in the comment section below.

Iptables Essentials: Common Firewall Rules and Commands | DigitalOcean

Content Courtesy of https://www.digitalocean.com/community/tutorials/iptables-essentials-common-firewall-rules-and-commands

Introduction

Iptables is the software firewall that is included with most Linux distributions by default. This cheat sheet-style guide provides a quick reference to iptables commands that will create firewall rules are useful in common, everyday scenarios. This includes iptables examples of allowing and blocking various services by port, network interface, and source IP address.

How To Use This Guide

- If you are just getting started with configuring your iptables firewall, check out our introduction to iptables (https://www.digitalocean.com/community/tutorials/how-to-set-up-a-firewall-using-iptables-on-ubuntu-14-04)
- Most of the rules that are described here assume that your iptables is set to **DROP** incoming traffic, through the default input policy, and you want to selectively allow traffic in
- Use whichever subsequent sections are applicable to what you are trying to achieve. Most sections are not predicated on any other, so you can use the examples below independently
- Use the Contents menu on the right side of this page (at wide page widths) or your browser's find function to locate the sections you need
- Copy and paste the command-line examples given, substituting the values in red with your own values

Keep in mind that the order of your rules matter. All of these `iptables` commands use the `-A` option to append the new rule to the end of a chain. If you want to put it somewhere else in the chain, you can use the `-I` option which allows you to specify the position of the new rule (or simply place it at the beginning of the chain by not specifying a rule number).

Note: When working with firewalls, take care not to lock yourself out of your own server by blocking SSH traffic (port 22, by default). If you lose access due to your firewall settings, you may need to connect to it via the console (https://www.digitalocean.com/community/tutorials/how-to-use-the-digitalocean-console-to-access-your-droplet) to fix your access. Once you are connected via the console, you can change your firewall rules to allow SSH access (or allow all traffic). If your saved firewall rules allow SSH access, another method is to reboot your server.

Remember that you can check your current iptables ruleset with `sudo iptables -S` and `sudo iptables -L`.

Let's take a look at the iptables commands!

Saving Rules

Iptables rules are ephemeral, which means they need to be manually saved for them to persist after a reboot.

Ubuntu

On Ubuntu, the easiest way to save iptables rules, so they will survive a reboot, is to use the `iptables-persistent` package. Install it with apt-get like this:

```
sudo apt-get install iptables-persistent
```

During the installation, you will asked if you want to save your current firewall rules.

If you update your firewall rules and want to save the changes, run this command:

```
sudo netfilter-persistent save
```

On versions of Ubuntu prior to 16.04, run this command instead:

```
sudo invoke-rc.d iptables-persistent save
```

CentOS 6 and Older

On CentOS 6 and older—CentOS 7 uses FirewallD by default—you can use the `iptables` init script to save your iptables rules:

```
sudo service iptables save
```

This will save your current iptables rules to the `/etc/sysconfig/iptables` file.

Listing and Deleting Rules

If you want to learn how to list and delete iptables rules, check out this tutorial: How To List and Delete Iptables Firewall Rules (https://www.digitalocean.com/community/tutorials/how-to-list-and-delete-iptables-firewall-rules).

Generally Useful Rules

This section includes a variety of iptables commands that will create rules that are generally useful on most servers.

Allow Loopback Connections

The **loopback** interface, also referred to as `lo`, is what a computer uses to forward network connections to itself. For example, if you run `ping localhost` or `ping 127.0.0.1`, your server will ping itself using the loopback. The loopback interface is also used if you configure your application server to connect to a database server with a "localhost" address. As such, you will want to be sure that your firewall is allowing these connections.

To accept all traffic on your loopback interface, run these commands:

```
sudo iptables -A INPUT -i lo -j ACCEPT
sudo iptables -A OUTPUT -o lo -j ACCEPT
```

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 that the server will allow return traffic to outgoing connections initiated by the server itself. This command will allow that:

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

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
```

Internal to External

Assuming `eth0` is your external network, and `eth1` is your internal network, this will allow your internal to access the external:

```
• sudo iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
```

Drop Invalid Packets

Some network traffic packets get marked as **invalid**. Sometimes it can be useful to log this type of packet but often it is fine to drop them. Do so with this command:

```
• sudo iptables -A INPUT -m conntrack --ctstate INVALID -j DROP
```

Block an IP Address

To block network connections that originate from a specific IP address, `15.15.15.51` for example, run this command:

```
• sudo iptables -A INPUT -s 15.15.15.51 -j DROP
```

In this example, `-s 15.15.15.51` specifies a **source** IP address of "15.15.15.51". The source IP address can be specified in any firewall rule, including an **allow** rule.

If you want to **reject** the connection instead, which will respond to the connection request with a "connection refused" error, replace "DROP" with "REJECT" like this:

```
• sudo iptables -A INPUT -s 15.15.15.51 -j REJECT
```

Block Connections to a Network Interface

To block connections from a specific IP address, e.g. `15.15.15.51`, to a specific network interface, e.g. `eth0`, use this command:

```
• iptables -A INPUT -i eth0 -s 15.15.15.51 -j DROP
```

This is the same as the previous example, with the addition of `-i eth0`. The network interface can be specified in any firewall rule, and is a great way to limit the rule to a particular network.

Service: SSH

If you're using a cloud server, you will probably want to allow incoming SSH connections (port 22) so you can connect to and manage your server. This section covers how to configure your firewall with various SSH-related rules.

Allow All Incoming SSH

To allow all incoming SSH connections run these commands:

```
• sudo iptables -A INPUT -p tcp --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, is only necessary if the `OUTPUT` policy is not set to `ACCEPT`.

Allow Incoming SSH from Specific IP address or subnet

To allow incoming SSH connections from a specific IP address or subnet, specify the source. For example, if you want to allow the 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, is only necessary if the `OUTPUT` policy is not set to `ACCEPT`.

Allow Outgoing SSH

If your firewall `OUTPUT` policy is not set to `ACCEPT`, and you want to allow outgoing SSH connections—your server initiating an SSH connection to another server—you can run these commands:

```
• sudo iptables -A OUTPUT -p tcp --dport 22 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
• sudo iptables -A INPUT -p tcp --sport 22 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

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, if you want to allow the entire `15.15.15.0/24` subnet to be able to rsync to your server, run these commands:

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

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

Service: Web Server

Web servers, such as Apache and Nginx, typically listen for requests on port 80 and 443 for HTTP and HTTPS connections, respectively. If your default policy for incoming traffic is set to drop or deny, you will want to create rules that will allow your server to respond to those requests.

Allow All Incoming HTTP

To allow all incoming HTTP (port 80) connections run these commands:

```
• sudo iptables -A INPUT -p tcp --dport 80 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
• sudo iptables -A OUTPUT -p tcp --sport 80 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

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

Allow All Incoming HTTPS

To allow all incoming HTTPS (port 443) connections run these commands:

```
• sudo iptables -A INPUT -p tcp --dport 443 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
• sudo iptables -A OUTPUT -p tcp --sport 443 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

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

Allow All Incoming HTTP and HTTPS

If you want to allow both HTTP and HTTPS traffic, you 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:

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

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.

Allow MySQL from Specific IP Address or Subnet

To allow incoming MySQL connections from a specific IP address or subnet, specify the source. For example, if you want to allow the entire `15.15.15.0/24` subnet, run these commands:

```
• sudo iptables -A INPUT -p tcp -s 15.15.15.0/24 --dport 3306 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEP
  T
• sudo 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 you have a private network interface `eth1`, for example—use these commands:

```
• sudo iptables -A INPUT -i eth1 -p tcp --dport 3306 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
• sudo iptables -A OUTPUT -o eth1 -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`.

Service: PostgreSQL

PostgreSQL listens for client connections on port 5432. If your PostgreSQL database server is being used by a client on a remote server, you need to be sure to allow that traffic.

PostgreSQL from Specific IP Address or Subnet

To allow incoming PostgreSQL connections from a specific IP address or subnet, specify the source. For example, if you want to allow the entire `15.15.15.0/24` subnet, run these commands:

```
• sudo iptables -A INPUT -p tcp -s 15.15.15.0/24 --dport 5432 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEP
  T
• sudo iptables -A OUTPUT -p tcp --sport 5432 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

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

Allow PostgreSQL to Specific Network Interface

To allow PostgreSQL connections to a specific network interface—say you have a private network interface `eth1`, for example—use these commands:

```
• sudo iptables -A INPUT -i eth1 -p tcp --dport 5432 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
• sudo iptables -A OUTPUT -o eth1 -p tcp --sport 5432 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

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

Service: Mail

Mail servers, such as Sendmail and Postfix, listen on a variety of ports depending on the protocols being used for mail delivery. If you are running a mail server, determine which protocols you are using and allow the appropriate types of traffic. We will also show you how to create a rule to block outgoing SMTP mail.

Block Outgoing SMTP Mail

If your server shouldn't be sending outgoing mail, you may want to block that kind of traffic. To block outgoing SMTP mail, which uses port 25, run this command:

```
• sudo iptables -A OUTPUT -p tcp --dport 25 -j REJECT
```

This configures iptables to **reject** all outgoing traffic on port 25. If you need to reject a different service by its port number, instead of port 25, simply replace it.

Allow All Incoming SMTP

To allow your server to respond to SMTP connections, port 25, run these commands:

```
• sudo iptables -A INPUT -p tcp --dport 25 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
• sudo iptables -A OUTPUT -p tcp --sport 25 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

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

Note: It is common for SMTP servers to use port 587 for outbound mail.

Allow All Incoming IMAP

To allow your server to respond to IMAP connections, port 143, run these commands:

```
• sudo iptables -A INPUT -p tcp --dport 143 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
• sudo iptables -A OUTPUT -p tcp --sport 143 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

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

Allow All Incoming IMAPS

To allow your server to respond to IMAPS connections, port 993, run these commands:

```
• sudo iptables -A INPUT -p tcp --dport 993 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
• sudo iptables -A OUTPUT -p tcp --sport 993 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

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

Allow All Incoming POP3

To allow your server to respond to POP3 connections, port 110, run these commands:

```
• sudo iptables -A INPUT -p tcp --dport 110 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
• sudo iptables -A OUTPUT -p tcp --sport 110 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

The second command, which allows the outgoing traffic of **established** POP3 connections, is only necessary if the `OUTPUT` policy is not set to `ACCEPT`.

Allow All Incoming POP3S

To allow your server to respond to POP3S connections, port 995, run these commands:

```
• sudo iptables -A INPUT -p tcp --dport 995 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
• sudo iptables -A OUTPUT -p tcp --sport 995 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

The second command, which allows the outgoing traffic of **established** POP3S connections, is only necessary if the `OUTPUT` policy is not set to `ACCEPT`.

Conclusion

That should cover many of the commands that are commonly used when configuring an iptables firewall. Of course, iptables is a very flexible tool so feel free to mix and match the commands with different options to match your specific needs if they aren't covered here.

If you're looking for help determining how your firewall should be set up, check out this tutorial: How To Choose an Effective Firewall Policy to Secure your Servers (https://www.digitalocean.com /community/tutorials/how-to-choose-an-effective-firewall-policy-to-secure-your-servers).

Good luck!

Control Network Traffic with iptables

Content Courtesy of https://www.linode.com/docs/security/firewalls/control-network-traffic-with-iptables/

Control Network Traffic with iptables

Iptables Tutorial - Beginners Guide to Linux Firewall

Content Courtesy of

Updated Monday, July 23, 2018 by Linode Written by Linode

Use promo code **DOCS10** for $10 credit on a new account.

iptables is an application that allows users to configure specific rules that will be enforced by the kernel's `netfilter` framework. It acts as a packet filter and firewall that examines and directs traffic based on port, protocol and other criteria. This guide will focus on the configuration and application of iptables rulesets and will provide examples of ways they are commonly used.

Control Network Traffic with iptables

By default, the iptables tool is included with your Linode-supplied distribution. In order to use iptables, you will need root (`sudo`) privileges.

Use Linux iptables to Manage IPv4 Traffic

The iptables Command

Many options can be used with the iptables command. As stated above, iptables sets the rules that control network traffic. You can define different tables to handle these rules through chains, lists of rules that match a subset of packets. The table contains a variety of built-in chains, but you can add your own.

Basic iptables Parameters and Syntax

Before we begin creating rules, let's review the syntax of an iptables rule.

For example, the following command adds a rule to the beginning of the chain that will drop all packets from the address `198.51.100.0`:

```
iptables -I INPUT -s 198.51.100.0 -j DROP
```

The sample command above:

1. Calls the `iptables` program
2. Uses the `-I` option for *insertion*. Using a rule with the insertion option will add it to the beginning of a chain and will be applied first. To indicate a specific placement in the chain, you may also use a number with the `-I` option.
3. The `-s` parameter, along with the IP address (198.51.100.0), indicates the *source*.
4. Finally, the `-j` parameter stands for *jump*. It specifies the target of the rule and what action will be performed if the packet is a match.

Parameter	Description
`-p, --protocol`	The protocol, such as TCP, UDP, etc.
`-s, --source`	Can be an address, network name, hostname, etc.
`-d, --destination`	An address, hostname, network name, etc.
`-j, --jump`	Specifies the target of the rule; i.e. what to do if the packet matches.
`-g, --goto chain`	Specifies that the processing will continue in a user specified chain.
`-i, --in-interface`	Names the interface from where packets are received.
`-o, --out-interface`	Name of the interface by which a packet is being sent.
`-f, --fragment`	The rule will only be applied to the second and subsequent fragments of fragmented packets.
`-c, --set-counters`	Enables the admin to initialize the packet and byte counters of a rule.

Default Tables

Tables are made up of built-in chains and may also contain user-defined chains. The built-in tables will depend on the kernel configuration and the installed modules.

The default tables are as follows:

- **Filter** - This is the default table. Its built-in chains are:
  - Input: packets going to local sockets
  - Forward: packets routed through the server
  - Output: locally generated packets
- **Nat** - When a packet creates a new connection, this table is used. Its built-in chains are:
  - Prerouting: designating packets when they come in
  - Output: locally generated packets before routing takes place
  - Postrouting: altering packets on the way out
- **Mangle** - Used for special altering of packets. Its chains are:
  - Prerouting: incoming packets
  - Postrouting: outgoing packets
  - Output: locally generated packets that are being altered
  - Input: packets coming directly into the server
  - Forward: packets being routed through the server
- **Raw** - Primarily used for configuring exemptions from connection tracking. The built-in chains are:
  - Prerouting: packets that arrive by the network interface
  - Output: processes that are locally generated
- **Security** - Used for Mandatory Access Control (MAC) rules. After the filter table, the security table is accessed next. The built-in chains are:
  - Input: packets entering the server
  - Output: locally generated packets
  - Forward: packets passing through the server

Basic iptables Options

There are many options that may be used with the `iptables` command:

Option	Description
`-A --append`	Add one or more rules to the end of the selected chain.
`-C --check`	Check for a rule matching the specifications in the selected chain.
`-D --delete`	Delete one or more rules from the selected chain.
`-F --flush`	Delete *all* the rules one-by-one.
`-I --insert`	Insert one or more rules into the selected chain as the given rule number.
`-L --list`	Display the rules in the selected chain.
`-n --numeric`	Display the IP address or hostname and post number in numeric format.
`-N --new-chain <name>`	Create a new user-defined chain.
`-v --verbose`	Provide more information when used with the list option.

https://www.hostinger.com/tutorials/iptables-tutorial

Iptables Linux firewall is used to monitor incoming and outgoing traffic to a server and filter it based on user-defined rules to prevent anyone from accessing the system. Using Iptables you can define rules which will allow only selective traffic on your server. In this Iptables tutorial, you will learn how to secure your web application using Iptables.

Note: For RHEL/ CentOS users there is a service named firewallD which is already installed on these operating systems. If you want to use Iptables, you have to disable it first.

What you'll need

Before you begin with Iptables tutorial, you will need the following:

If you want to learn more about SSH and SSH commands follow this tutorial. (https://www.hostinger.com/tutorials/ssh/basic-ssh-commands)

Iptables Basics

All data is sent in the form packets over the internet. Linux kernel provides an interface to filter both incoming and outgoing traffic packets using tables of packet filters. Iptables is a command line application and a Linux firewall that you can use to set-up, maintain and inspect these tables. Multiple tables can be defined. Each table can contain multiple chains. A chain is nothing but a set of rules. Each rule defines what to do with the packet if it matches with that packet. When the packet is matched, it is given a **TARGET**. A target can be another chain to match with or one of the following special values:

- **ACCEPT**: It means the packet will be allowed to pass through.
- **DROP**: It means that packet will not be allowed to pass through.
- **RETURN**: It means to skip the current chain and go back to the next rule from the chain it was called in.

For the scope of this iptables tutorial, we are going to work with one of the default tables called **filter**. Filters table has three chains (sets of rules).

- **INPUT** – This chain is used to control incoming packets to the server. You can block/allow connections based on port, protocol or source IP address.

Option	Description
`-X --delete-chain <name>`	Delete the user-defined chain.

- **FORWARD** – This chain is used to filter packets that are incoming to the server but are to be forwarded somewhere else.
- **OUTPUT** – This chain is used to filter packets that are going out from your server.

Insert, Replace or Delete iptables Rules

iptables rules are enforced top down, so the first rule in the ruleset is applied to traffic in the chain, then the second, third and so on. This means that rules cannot necessarily be added to a ruleset with `iptables -A` or `ip6tables -A`. Instead, rules must be *inserted* with `iptables -I` or `ip6tables -I`.

Insert

Inserted rules need to be placed in the correct order with respect to other rules in the chain. To get a numerical list of your iptables rules:

```
sudo iptables -L -nv --line-numbers
```

For example, let's say you want to insert a rule into the basic ruleset provided in this guide, that will accept incoming connections to port 8080 over the TCP protocol. We'll add it as rule 7 to the INPUT chain, following the web traffic rules:

```
sudo iptables -I INPUT 7 -p tcp --dport 8080 -m state --state NEW -j ACCEPT
```

If you now run `sudo iptables -L -nv` again, you'll see the new rule in the output.

Replace

Replacing a rule is similar to inserting, but instead uses `iptables -R`. For example, let's say you want to reduce the logging of denied entries to only 3 per minute, down from 5 in the original ruleset. The LOG rule is ninth in the INPUT chain:

```
sudo iptables -R INPUT 9 -m limit --limit 3/min -j LOG --log-prefix "iptables_IN
PUT_denied: " --log-level 7
```

Delete

Deleting a rule is also done using the rule number. For example, to delete the rule we just inserted for port 8080:

```
sudo iptables -D INPUT 7
```

Caution
Editing rules does not automatically save them. See our section on deploying rulesets (/docs/security/firewalls/control-network-traffic-with-iptables#deploy-your-iptables-rulesets) for the specific instructions for your distribution.

View Your Current iptables Rules

IPv4:

```
sudo iptables -L -nv
```

IPv6:

```
sudo ip6tables -L -nv
```

On most distributions, iptables has no default rules for either IPv4 and IPv6. As a result, on a newly created Linode you will likely see what is shown below - three empty chains without any firewall rules. This means that all incoming, forwarded and outgoing traffic is *allowed*. It's important to limit inbound and forwarded traffic to only what's necessary.

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Configure iptables

iptables can be configured and used in a variety of ways. The following sections will outline how to configure rules by port and IP, as well as how to blacklist (block) or whitelist (allow) addresses.

Block Traffic by Port

You may use a port to block all traffic coming in on a specific interface. For example:

```
iptables -A INPUT -j DROP -p tcp --destination-port 110 -i eth0
```

Let's examine what each part of this command does:

- `-A` will add or append the rule to the end of the chain.
- `INPUT` will add the rule to the table.
- `DROP` means the packets are discarded.
- `-p tcp` means the rule will only drop TCP packets.
- `--destination-port 110` filters packets targeted to port 110.
- `-i eth0` means this rule will impact only packets arriving on the *eth0* interface.

It is important to understand that iptables do *not* recognize aliases on the network interface. Therefore, if you have several virtual IP interfaces, you will have to specify the destination address to filter the traffic. A sample command is provided below:

```
iptables -A INPUT -j DROP -p tcp --destination-port 110 -i eth0 -d 198.51.100.0
```

You may also use `-D` or `--delete` to remove rules. For example, these commands are equivalent:

```
iptables --delete INPUT -j DROP -p tcp --destination-port 110 -i eth0 -d 198.5
1.100.0
iptables -D INPUT -j DROP -p tcp --destination-port 110 -i eth0 -d 198.51.100.0
```

Drop Traffic from an IP

In order to drop all incoming traffic from a specific IP address, use the iptables command with the following options:

```
iptables -I INPUT -s 198.51.100.0 -j DROP
```

To remove these rules, use the `--delete` or `-D` option:

```
iptables --delete INPUT -s 198.51.100.0 -j DROP
iptables -D INPUT -s 198.51.100.0 -j DROP
```

Block or Allow Traffic by Port Number to Create an iptables Firewall

One way to create a firewall is to block all traffic to the system and then allow traffic on certain ports. Below is a sample sequence of commands to illustrate the process:

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i lo -m comment --comment "Allow loopback connections" -j AC
CEPT
iptables -A INPUT -p icmp -m comment --comment "Allow Ping to work as expected"
-j ACCEPT
iptables -A INPUT -p tcp -m multiport --destination-ports 22,25,53,80,443,465,5
222,5269,5280,8999:9003 -j ACCEPT
iptables -A INPUT -p udp -m multiport --destination-ports 53 -j ACCEPT
iptables -P INPUT DROP
iptables -P FORWARD DROP
```

Let’s break down the example above. The first two commands add or append rules to the `INPUT` chain in order to allow access on specific ports. The `-p tcp` and `-p udp` options specify either UDP or TCP packet types. The `-m multiport` function matches packets on the basis of their source or destination ports, and can accept the specification of up to 15 ports. Multiport also accepts ranges such as `8999:9003` which counts as 2 of the 15 possible ports, but matches ports `8999` , `9000` , `9001` , `9002` , and `9003` . The next command allows all incoming and outgoing packets that are associated with existing connections so that they will not be inadvertently blocked by the firewall. The final two commands use the `-P` option to describe the *default policy* for these chains. As a result, all packets processed by `INPUT` and `FORWARD` will be dropped by default.

Note that the rules described above only control incoming packets, and do not limit outgoing connections.

Whitelist/Blacklist Traffic by Address

You can use iptables to block all traffic and then only allow traffic from certain IP addresses. These firewall rules limit access to specific resources at the network layer. Below is an example sequence of commands:

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i lo -m comment --comment "Allow loopback connections" -j AC
CEPT
iptables -A INPUT -p icmp -m comment --comment "Allow Ping to work as expected"
-j ACCEPT
iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT
iptables -A INPUT -s 198.51.100.0 -j ACCEPT
iptables -P INPUT DROP
iptables -P FORWARD DROP
```

In the first command, the `-s 192.168.1.0/24` statement specifies that all source IPs (`-s`) in the address space of `192.168.1` are allowed. You may specify an IP address range using CIDR (Classless Inter-Domain Routing) notation, or individual IP addresses, as in the second command. The third command allows all incoming and outgoing packets that are associated with existing connections. The final two commands set the default policy for all `INPUT` and `FORWARD` chains to drop all packets.

Use ip6tables to Manage IPv6 Traffic

When you’re working with IPv6, remember that the `iptables` command is not compatible. Instead, there is an `ip6tables` command. The options such as append, check, etc. are the same. The tables used by ip6tables are *raw*, *security*, *mangle* and *filter*. The parameters such as protocol, source, etc. are the same. The syntax is essentially the same as IPv4. Sample syntax is below:

```
ip6tables [-t table] -N chain
```

To view what rules are configured for IPv6, use the command:

```
ip6tables -L
```

Configure Rules for IPv6

ip6tables works by using ports, specific addresses for blacklisting, protocols and so forth. The primary difference is that ip6tables can use extended packet matching modules with the `-m` or `match` options, followed by the module name. Below are some of the extended modules:

- **addrtype** - Matches packets based on their address type. Some of the address types are:
  - Local
  - Unicast
  - Broadcast
  - Multicast
- **ah** - Matches the parameters in the authentication header of IPsec packets.
- **cluster** - You can deploy gateway and backend load-sharing clusters without a load balancer.
- **comment** - Allows you to add a comment to any rule.
- **connbytes** - Matches by how many bytes or packets a connection has transferred, or average bytes per packet.

This is not intended to be a complete or comprehensive list. You may review the full list of extended modules by using the `man` page:

```
man ip6tables
```

Below is a sample rule used in ip6tables:

```
# limit the number of parallel HTTP requests to 16 for the link local network
ip6tables -A INPUT -p tcp --syn --dport 80 -s fe80::/64 -m connlimit --connlimi
t-above 16 --connlimit-mask 64 -j REJECT
ip6tables -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
```

This rule breaks down as follows:

- The first line is a comment.
- `-A` is for append.
- `INPUT` is to add the rule to the table.
- `-p` is for protocol, which is TCP.
- `--syn` only matches TCP packets with the SYN bit set and the ACK, RST, and FIN bits cleared.
- `--dport` is the destination port, which is 80.
- `-s` is the source, which is the local address range fe80::/64.
- `-m` is for match.
- `connlimit` is the extended packet module name, which is connection limit.
- `--connlimit-above 16` means if the number of connections exceeds 16, only the first 16 will be used.
- `--connlimit-mask 64` means the group hosts are using a prefix length of 64.
- `-j` is for jump, it tells the target of the rule what to do if the packet is a match.
- `REJECT` means the packet is dropped.

Required Rules for Non-Static IPv6 Allocations

```
# Below are the rules which are required for your IPv6 address to be properly a
llocated
ip6tables -A INPUT -p icmpv6 --icmpv6-type router-advertisement -m hl --hl-eq 2
55 -j ACCEPT
ip6tables -A INPUT -p icmpv6 --icmpv6-type neighbor-solicitation -m hl --hl-eq
255 -j ACCEPT
ip6tables -A INPUT -p icmpv6 --icmpv6-type neighbor-advertisement -m hl --hl-eq
255 -j ACCEPT
ip6tables -A INPUT -p icmpv6 --icmpv6-type redirect -m hl --hl-eq 255 -j ACCEPT
```

Basic iptables Rulesets for IPv4 and IPv6

Appropriate firewall rules depend on the services being run. Below are iptables rulesets to secure your Linode if you're running a web server.

Caution

These rules are given only as an example. A real production web server may require more or less configuration, and these rules would not be appropriate for a database, Minecraft or VPN server. Iptables rules can always be modified or reset later, but these basic rulesets serve as a demonstration.

IPv4

/tmp/v4

```
*filter

# Allow all loopback (lo0) traffic and reject traffic
# to localhost that does not originate from lo0.
-A INPUT -i lo -j ACCEPT
-A INPUT ! -i lo -s 127.0.0.0/8 -j REJECT

# Allow ping.
-A INPUT -p icmp -m state --state NEW --icmp-type 8 -j ACCEPT

# Allow SSH connections.
-A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT

# Allow HTTP and HTTPS connections from anywhere
# (the normal ports for web servers).
-A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
-A INPUT -p tcp --dport 443 -m state --state NEW -j ACCEPT

# Allow inbound traffic from established connections.
# This includes ICMP error returns.
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Log what was incoming but denied (optional but useful).
-A INPUT -m limit --limit 5/min -j LOG --log-prefix "iptables_INPUT_denied:
" --log-level 7

# Reject all other inbound.
-A INPUT -j REJECT

# Log any traffic that was sent to you
# for forwarding (optional but useful).
-A FORWARD -m limit --limit 5/min -j LOG --log-prefix "iptables_FORWARD_den
ied: " --log-level 7

# Reject all traffic forwarding.
-A FORWARD -j REJECT

COMMIT
```

Optional: If you plan to use Linode Longview (/docs/platform/longview/longview/) or Linode's NodeBalancers (/docs/platform/nodebalancer/getting-started-with-nodebalancers/), add the respective rule after the section for allowing HTTP and HTTPS connections:

```
# Allow incoming Longview connections from longview.linode.com
-A INPUT -s 96.126.119.66 -m state --state NEW -j ACCEPT

# Allow incoming NodeBalancer connections
-A INPUT -s 192.168.255.0/24 -m state --state NEW -j ACCEPT
```

IPv6

If you would like to supplement your web server's IPv4 rules with IPv6 as well, this ruleset will allow HTTP/S access and all ICMP functions.

/tmp/v6

```
*filter

# Allow all loopback (lo0) traffic and reject traffic
# to localhost that does not originate from lo0.
-A INPUT -i lo -j ACCEPT
-A INPUT ! -i lo -s ::1/128 -j REJECT

# Allow ICMP
-A INPUT -p icmpv6 -j ACCEPT

# Allow HTTP and HTTPS connections from anywhere
# (the normal ports for web servers).
-A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
-A INPUT -p tcp --dport 443 -m state --state NEW -j ACCEPT

# Allow inbound traffic from established connections.
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Log what was incoming but denied (optional but useful).
-A INPUT -m limit --limit 5/min -j LOG --log-prefix "ip6tables_INPUT_denie
d: " --log-level 7

# Reject all other inbound.
-A INPUT -j REJECT

# Log any traffic that was sent to you
# for forwarding (optional but useful).
-A FORWARD -m limit --limit 5/min -j LOG --log-prefix "ip6tables_FORWARD_de
nied: " --log-level 7

# Reject all traffic forwarding.
-A FORWARD -j REJECT

COMMIT
```

Note

APT (http://linux.die.net/man/8/apt) attempts to resolve mirror domains to IPv6 as a result of `apt-get update`. If you choose to entirely disable and deny IPv6, this will slow down the update process for Debian and Ubuntu because APT waits for each resolution to time out before moving on.

To remedy this, uncomment the line `precedence ::ffff:0:0/96 100` in `/etc/gai.conf`.

Deploy Your iptables Rulesets

The process for deploying iptables rulesets varies depending on which Linux distribution you're using:

Debian / Ubuntu

UFW is the iptables controller included with Ubuntu, but it is also available in Debian's repositories. If you prefer to use UFW instead of iptables, see our guide: How to Configure a Firewall with UFW (/docs /security/firewalls/configure-firewall-with-ufw).

1. Create the files `/tmp/v4` and `/tmp/v6`. Paste the above rulesets into their respective files.
2. Import the rulesets into immediate use:

   ```
   sudo iptables-restore < /tmp/v4
   sudo ip6tables-restore < /tmp/v6
   ```

3. To apply your iptables rules automatically on boot, see our section on configuring iptables-persistent (/docs/security/firewalls/control-network-traffic-with-iptables#introduction-to-iptables-persistent).

CentOS / Fedora

CentOS 7 or Fedora 20 and above

In these distros, FirewallD is used to implement firewall rules instead of using the iptables command. If you prefer to use it over iptables, see our guide: Introduction to FirewallD on CentOS (/docs/security/firewalls /introduction-to-firewalld-on-centos).

1. If you prefer to use iptables, FirewallD must first be stopped and disabled.

   ```
   sudo systemctl stop firewalld.service && sudo systemctl disable firewalld.se
   rvice
   ```

2. Install `iptables-services` and enable iptables and ip6tables:

   ```
   sudo yum install iptables-services
   sudo systemctl enable iptables && sudo systemctl enable ip6tables
   sudo systemctl start iptables && sudo systemctl start ip6tables
   ```

3. Create the files `/tmp/v4` and `/tmp/v6`. Paste the rulesets above into their respective files.
4. Import the rulesets into immediate use:

   ```
   sudo iptables-restore < /tmp/v4
   sudo ip6tables-restore < /tmp/v6
   ```

5. Save each ruleset:

   ```
   sudo service iptables save
   sudo service ip6tables save
   ```

6. Remove the temporary rule files:

   ```
   sudo rm /tmp/{v4,v6}
   ```

CentOS 6

1. Create the files `/tmp/v4` and `/tmp/v6`. Paste the rulesets above into their respective files.
2. Import the rules from the temporary files:

```
sudo iptables-restore < /tmp/v4
sudo ip6tables-restore < /tmp/v6
```

3. Save the rules:

```
sudo service iptables save
sudo service ip6tables save
```

> **Note**
> Firewall rules are saved to `/etc/sysconfig/iptables` and `/etc/sysconfig/ip6tables`.

4. Remove the temporary rule files:

```
sudo rm /tmp/{v4,v6}
```

Arch Linux

1. Create the files `/etc/iptables/iptables.rules` and `/etc/iptables/ip6tables.rules`. Paste the rulesets above into their respective files.
2. Import the rulesets into immediate use:

```
sudo iptables-restore < /etc/iptables/iptables.rules
sudo ip6tables-restore < /etc/iptables/ip6tables.rules
```

3. iptables does not run by default in Arch. Enable and start the systemd units:

```
sudo systemctl start iptables && sudo systemctl start ip6tables
sudo systemctl enable iptables && sudo systemctl enable ip6tables
```

For more info on using iptables in Arch, see its Wiki entries for iptables (https://wiki.archlinux.org/index.php/Iptables) and a simple stateful firewall (https://wiki.archlinux.org/index.php/Simple_stateful_firewall).

Verify iptables Rulesets

Check your Linode's firewall rules with the `v` option for a verbose output:

```
sudo iptables -vL
sudo ip6tables -vL
```

The output for IPv4 rules should show:

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     all  --  lo     any     anywhere             anywhere
    0     0 REJECT     all  --  !lo    any     loopback/8           anywhere
reject-with icmp-port-unreachable
    0     0 ACCEPT     icmp --  any    any     anywhere             anywhere
icmp destination-unreachable
    0     0 ACCEPT     icmp --  any    any     anywhere             anywhere
icmp echo-request
    0     0 ACCEPT     icmp --  any    any     anywhere             anywhere
icmp time-exceeded
    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere
tcp dpt:ssh state NEW
    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere
tcp dpt:http state NEW
    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere
tcp dpt:https state NEW
    0     0 ACCEPT     all  --  any    any     anywhere             anywhere
state RELATED,ESTABLISHED
    0     0 LOG        all  --  any    any     anywhere             anywhere
limit: avg 5/min burst 5 LOG level debug prefix "iptables_INPUT_denied: "
    0     0 REJECT     all  --  any    any     anywhere             anywhere
reject-with icmp-port-unreachable

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 LOG        all  --  any    any     anywhere             anywhere
limit: avg 5/min burst 5 LOG level debug prefix "iptables_FORWARD_denied: "
    0     0 REJECT     all  --  any    any     anywhere             anywhere
reject-with icmp-port-unreachable

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Output for IPv6 rules will look like this:

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     all      lo     any     anywhere             anywhere
    0     0 REJECT     all      !lo    any     localhost            anywhere
reject-with icmp6-port-unreachable
    0     0 ACCEPT     ipv6-icmp    any    any     anywhere             anywher
e
    0     0 ACCEPT     tcp      any    any     anywhere             anywhere
tcp dpt:http state NEW
    0     0 ACCEPT     tcp      any    any     anywhere             anywhere
tcp dpt:https state NEW
    0     0 ACCEPT     all      any    any     anywhere             anywhere
state RELATED,ESTABLISHED
    0     0 LOG        all      any    any     anywhere             anywhere
limit: avg 5/min burst 5 LOG level debug prefix "ip6tables_INPUT_denied: "
    0     0 REJECT     all      any    any     anywhere             anywhere
reject-with icmp6-port-unreachable

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 LOG        all      any    any     anywhere             anywhere
limit: avg 5/min burst 5 LOG level debug prefix "ip6tables_FORWARD_denied: "
    0     0 REJECT     all      any    any     anywhere             anywhere
reject-with icmp6-port-unreachable

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Your firewall rules are now in place and protecting your Linode. Remember, you may need to edit these rules later if you install other packages that require network access.

Introduction to iptables-persistent

Ubuntu and Debian have a package called **iptables-persistent** that makes it easy to reapply your firewall rules at boot time. After installation, you can save all your rules in two files (one for IPv4 and one for IPv6). If you’ve already configured and applied iptables rules, iptables-persistent will detect them automatically and allow you to add them to the appropriate configuration file.

Install iptables-persistent

On Debian or Ubuntu use the following command to check whether `iptables-persistent` is already installed:

```
dpkg -l iptables-persistent
```

If `dpkg` returns that there are no matching packages, you will need to install the `iptables-persistent` package:

```
apt-get install iptables-persistent
```

During the installation, you will be prompted twice. The first prompt is asking if you would like to save your current IPv4 rules.

Save IPv4 rules prompt. (1721-ipv4-rules.png)

The second prompt is to save the rules configured for IPv6.

Save IPv6 rules prompt. (1722-ipv6-rules.png)

After the install is complete, you should see the iptables's subdirectory. Run the `ls /etc/iptables` command again to verify that your output resembles the following:

```
rules.v4  rules.v6
```

Use iptables-persistent

To view what rules are already configured on your server:

```
iptables -L
```

You should see output similar to:

```
Chain INPUT (policy ACCEPT)
target      prot opt source         destination
DROP        all  --  198.51.100.0    anywhere

Chain FORWARD (policy ACCEPT)
target      prot opt source         destination

CHAIN OUTPUT (policy ACCEPT)
target      prot opt source         destination
```

The rules above allow anyone anywhere access to everything. If your output resembles this, you'll need to set rules that prevent unauthorized access.

iptables-persistent Rules

Use the `rules.v4` or `rules.v6` files to add, delete or edit the rules for your server. These files can be edited using a text editor to function as a proxy, NAT or firewall. The configuration depends on the requirements of your server and what functions are needed. Below is a file excerpt from both the `rules.v4` and `rules.v6` files:

/etc/iptables/rules.v4

```
# Generated by iptables-save v1.4.14 on Wed Apr  2 13:24:27 2014
*security
:INPUT ACCEPT [18483:1240117]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [17288:2887358]
COMMIT
```

/etc/iptables/rules.v6

```
# Generated by ip6tables-save v1.4.14 on Wed Apr  2 13:24:27 2014
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [27:2576]
:POSTROUTING ACCEPT [27:2576]
COMMIT
```

While some rules are configured in these files already, either file can be edited at any time. The syntax for altering table rules is the same as in the sections Configure iptables and Configuring Rules for IPv6.

Save iptables-persistent Rules Through Reboot

By default, iptables-persistent rules save on reboot for IPv4 only. Therefore, if you are running both IPv4 and IPv6 together you will need to manually edit both the `rules.v4` and `rules.v6` files. On older systems, `iptables-save` was used to write the changes to the `rules` file. Now that `iptables-persistent` is an option, do not use the `iptables-save > /etc/iptables/rules.v4` or `iptables-save > /etc/iptables/rules.v6` commands as any IPv6 changes will be overwritten by the IPv4 rules.

To enforce the iptables rules and ensure that they persist after reboot run `dpkg-reconfigure` and respond **Yes** when prompted. (If you ever edit your saved rules in the future, use this same command to save them again.)

```
dpkg-reconfigure iptables-persistent
```

To verify the rules are applied and available after the system reboot use the commands:

```
iptables -L
ip6tables -L
```

Network Lock-out

When you’re applying network rules, especially with both IPv4 and IPv6 and multiple interfaces, it is easy to lock yourself out. In the event you apply the rule and are unable to access your server, you may gain access through Lish (/docs/platform/manager/using-the-linode-shell-lish/) in the Linode Manager. The following steps will guide you through using the graphical interface of your Linode to gain access to your server:

1. Connect to your Linode Manager.
2. Click on the Remote Access tab.
3. Under the section entitled “Console Access,” click on the **Launch Lish Console** link.
4. Login with your root or sudo user name and password.
5. Remove any rules causing the connectivity issues.
6. Log out of the Lish window.
7. Attempt login via a regular SSH session.

This Lish console will function similarly to a regular SSH terminal session.

Troubleshooting: netfilter-persistent doesn’t come back up on reboot.

If you have upgraded to Debian 8 from an earlier version, you may see a situation where netfilter-persistent fails to start during boot when using the Linode kernel. The console output will show similar to:

```
[FAILED] Failed to start Load Kernel Modules.
See 'systemctl status systemd-modules-load.service' for details.
[DEPEND] Dependency failed for netfilter persistent configuration
```

You can also use `journalctl -xn` to see that systemd can not load the `loop` module:

```
systemd-modules-load[3452]: Failed to lookup alias 'loop': Function not impleme
nted
```

To fix this, comment out the line `loop` in `/etc/modules`:

```
sed -i 's/loop/#loop/g' /etc/modules
```

Then restart netfilter-persistent:

```
systemctl restart netfilter-persistent
```

It should then be running fine. Confirm with:

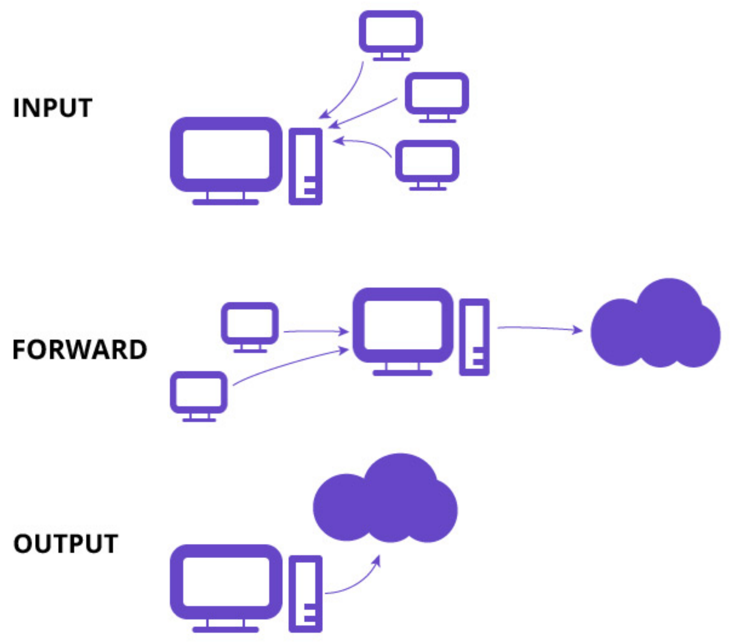
```
systemctl status netfilter-persistent
```

This issue does not occur in new deployments of Debian 8 because the `loop` line isn't present in `/etc/modules`.

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

Find answers, ask questions, and help others. (https://www.linode.com/community/questions/)



(https://www.hostinger.com/tutorials/wp-content/uploads/sites/2/2017/06/iptabes-tutorial-input-forward-output.jpg)

Step 1 – Installing Iptables Linux Firewall

1. Installing Iptables

Iptables comes pre-installed in almost all of the Linux distributions. But if you don't have it installed on

Ubuntu/Debian system use:

```
sudo apt-get update
sudo apt-get install iptables
```

2. Checking current Iptables status

With this command, you can check the status of your current Iptables configuration. Here **-L** option is used to list all the rules and **-v** option is for a more tedious list. Please note that these options are **case sensitive**.

```
sudo iptables -L -v
```

Example output:

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination
```

This is the output of the above command. Here, all three chains are set to default **ACCEPT** policy. There are currently no rules for any of the chains.

To make this Iptables tutorial more practical, we will modify the **INPUT** chain to filter the incoming traffic.

Step 2 – Defining chain rules

Defining a rule means appending it to the list (chain). Here's the Iptables command formatted with regular options. We don't have to specify all of them.

```
sudo iptables -A  -i <interface> -p <protocol (tcp/udp) > -s <source> --dport <port no.>  -j <target>
```

Here **-A** stands for append. The chain refers to the chain we want to append our rules. The **interface** is the network interface on which you want to filter the traffic. The **protocol** refers to the networking protocol of packets you want to filter. You can also specify the **port, no of** the port on which you want to filter the traffic.

For more detailed info about Iptables command and its options, you can check Iptables main page. (http://ipset.netfilter.org/iptables.man.html)

1. Enabling traffic on localhost

We want all communications between applications and databases on the server to continue as usual.

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

Example output:

```
Chain INPUT (policy ACCEPT 7 packets, 488 bytes)
pkts bytes target     prot opt in     out     source                 destination
0     0 ACCEPT     all  --  lo     any     anywhere             anywhere
```

Here **-A** option is used to append the rule to the **INPUT** chain, accept all connections on **lo** interface. lo means loopback interface. It is used for all the communications on the localhost, like communications between a database and a web application on the same machine.

2. Enabling connections on HTTP, SSH, and SSL port

We want our regular HTTP (port 80), https (port 443), ssh (port 22) connections to continue as usual. Enter the following commands to enable them. In the following commands, we have specified protocol with **-p** option and the corresponding port for each protocol with **–dport** (destination port) option.

```
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

Now all TCP protocol connections with specified ports will be accepted.

3. Filtering packets based on source

If you want to accept or reject packets based on the source IP address or the range of IP addresses you can specify it with **-s** option. For example to accept packets from address 192.168.1.3 –

```
sudo iptables -A INPUT -s 192.168.1.3 -j ACCEPT
```

You can drop packets from an IP address with a similar command with option **DROP** .

```
sudo iptables -A INPUT -s 192.168.1.3 -j DROP
```

If you want to drop packets from a range of IP addresses you have to use the **Iprange** module with **-m** option and specify the IP address range with **–src-range.**

```
sudo iptables -A INPUT -m iprange --src-range 192.168.1.100-192.168.1.200 -j DROP
```

4. Dropping all other traffic

Note: It is important to **DROP** all other traffic after defining the rules as it prevents unauthorized access to a server from other open ports.

```
sudo iptables -A INPUT -j DROP
```

This command drops all incoming traffic other than the ports mentioned in the above commands. You can check your set of rules now with:

```
sudo iptables -L -v
```

5. Deleting rules

If you want to remove all rules and start with a clean slate you can use the flush command.

```
sudo iptables -F
```

This command deletes all current rules. If you want to delete a specific rule you can do it with **-D** option. First, list all the rules with numbers by entering following command:

```
sudo iptables -L --line-numbers
```

Then you will get a list of rules with numbers.

```
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all  --  192.168.0.4          anywhere
2    ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:https
3    ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
4    ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
```

To delete a rule specify the number in the list and the chain of the rule. In our case **INPUT** chain and number **3**.

```
sudo iptables -D INPUT 3
```

Step 3 – Persisting changes

Iptables rules we have created are saved in memory. That means we have to redefine them on reboot. To make these changes persistent after reboot, use the following command on Ubuntu/Debian systems:

```
sudo /sbin/iptables-save
```

This command saves current rules to system configuration file which is used to reconfigure the tables at the time of reboot. You should run this command everytime you make changes to the rules. To disable this firewall simply flush all the rules and make the changes persistent.

```
sudo iptables -F
sudo /sbin/iptables-save
```

Conclusion

In this Iptables tutorial, we have used Iptables Linux firewall to only allow traffic on specific ports. We have also made sure that our rules will be saved after reboot. This Linux firewall will drop unwanted packets, but there is a caveat here that Iptables can govern only ipv4 traffic. If your VPS box has enabled ipv6 networking you have to set different rules for that traffic with ip6tables.

iptables command in Linux with Examples - GeeksforGeeks

Content Courtesy of https://www.geeksforgeeks.org/iptables-command-in-linux-with-examples/
iptables is a command line interface used to set up and maintain tables for the Netfilter firewall for IPv4, included in the Linux kernel. The firewall matches packets with rules defined in these tables and then takes the specified action on a possible match.

- *Tables* is the name for a set of chains.
- *Chain* is a collection of rules.
- *Rule* is condition used to match packet.
- *Target* is action taken when a possible rule matches. Examples of the target are ACCEPT, DROP, QUEUE.
- *Policy* is the default action taken in case of no match with the inbuilt chains and can be ACCEPT or DROP.

Syntax:

```
iptables --table TABLE -A/-C/-D... CHAIN rule --jump Target
```

TABLE

There are five possible tables:

- **filter:** Default used table for packet filtering. It includes chains like INPUT, OUTPUT and FORWARD.
- **nat :** Related to Network Address Translation. It includes PREROUTING and POSTROUTING chains.
- **mangle :** For specialised packet alteration. Inbuilt chains include PREROUTING and OUTPUT.
- **raw :** Configures exemptions from connection tracking. Built-in chains are PREROUTING and OUTPUT.
- **security :** Used for Mandatory Access Control (https://en.wikipedia.org/wiki/Mandatory_access_control)

CHAINS

There are few built-in chains that are included in tables. They are:

- **INPUT :**set of rules for packets destined to localhost sockets.
- **FORWARD :**for packets routed through the device.
- **OUTPUT :**for locally generated packets, meant to be transmitted outside.
- **PREROUTING :**for modifying packets as they arrive.
- **POSTROUTING :**for modifying packets as they are leaving.

Note: User-defined chains can also be created.

OPTIONS

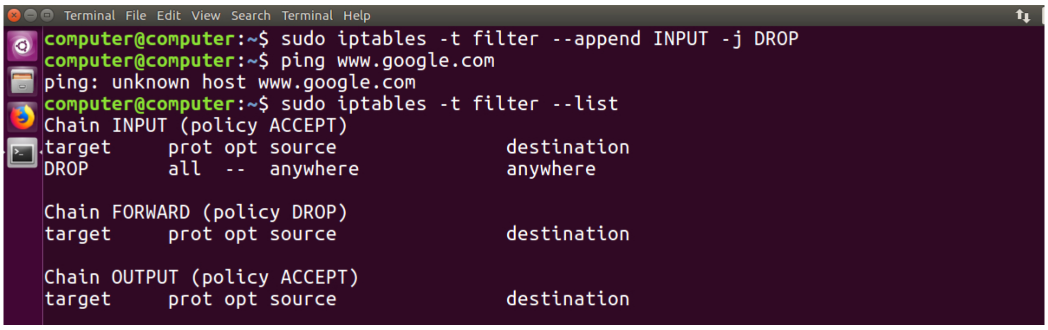
1. **-A, –append :** Append to the chain provided in parameters.
   Syntax:

```
iptables [-t table] --append [chain] [parameters]
```

Example: This command drops all the traffic coming on any port.

```
iptables -t filter --append INPUT -j DROP
```

Output:



```
computer@computer:~$ sudo iptables -t filter --append INPUT -j DROP
computer@computer:~$ ping www.google.com
ping: unknown host www.google.com
computer@computer:~$ sudo iptables -t filter --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

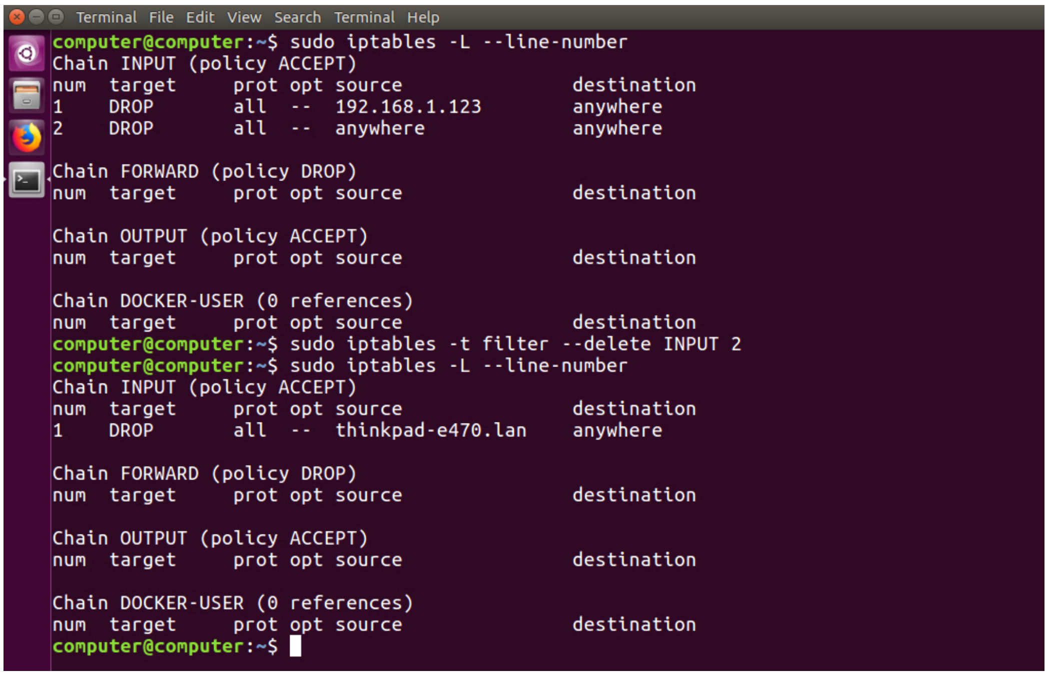
2. **-D, –delete :** Delete rule from the specified chain.
   Syntax:

```
iptables [-t table] --delete [chain] [rule_number]
```

Example: This command deletes the rule 2 from INPUT chain.

```
iptables -t filter --delete INPUT 2
```

Output:



```
computer@computer:~$ sudo iptables -L --line-number
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    DROP       all  --  192.168.1.123        anywhere
2    DROP       all  --  anywhere             anywhere

Chain FORWARD (policy DROP)
num  target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination

Chain DOCKER-USER (0 references)
num  target     prot opt source               destination
computer@computer:~$ sudo iptables -t filter --delete INPUT 2
computer@computer:~$ sudo iptables -L --line-number
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    DROP       all  --  thinkpad-e470.lan    anywhere

Chain FORWARD (policy DROP)
num  target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination

Chain DOCKER-USER (0 references)
num  target     prot opt source               destination
computer@computer:~$
```

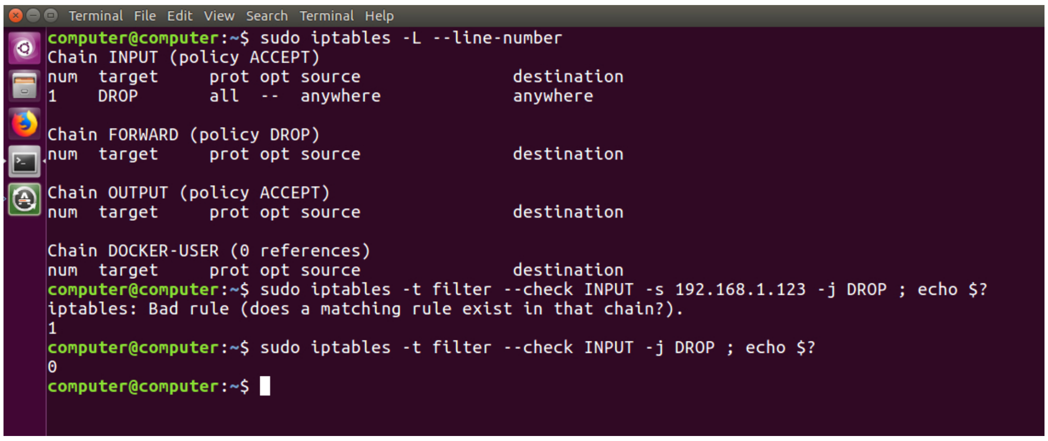
3. **-C, –check :**Check if a rule is present in the chain or not. It returns 0 if the rule exists and returns 1 if it does not.
   Syntax:

```
iptables [-t table] --check [chain] [parameters]
```

Example: This command checks whether the specified rule is present in the INPUT chain.

```
iptables -t filter --check INPUT -s 192.168.1.123 -j DROP
```

Output:



```
computer@computer:~$ sudo iptables -L --line-number
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    DROP       all  --  anywhere             anywhere

Chain FORWARD (policy DROP)
num  target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination

Chain DOCKER-USER (0 references)
num  target     prot opt source               destination
computer@computer:~$ sudo iptables -t filter --check INPUT -s 192.168.1.123 -j DROP ; echo $?
iptables: Bad rule (does a matching rule exist in that chain?).
1
computer@computer:~$ sudo iptables -t filter --check INPUT -j DROP ; echo $?
0
computer@computer:~$
```

PARAMETERS

The parameters provided with the *iptables* command is used to match the packet and perform the specified action. The common parameters are:

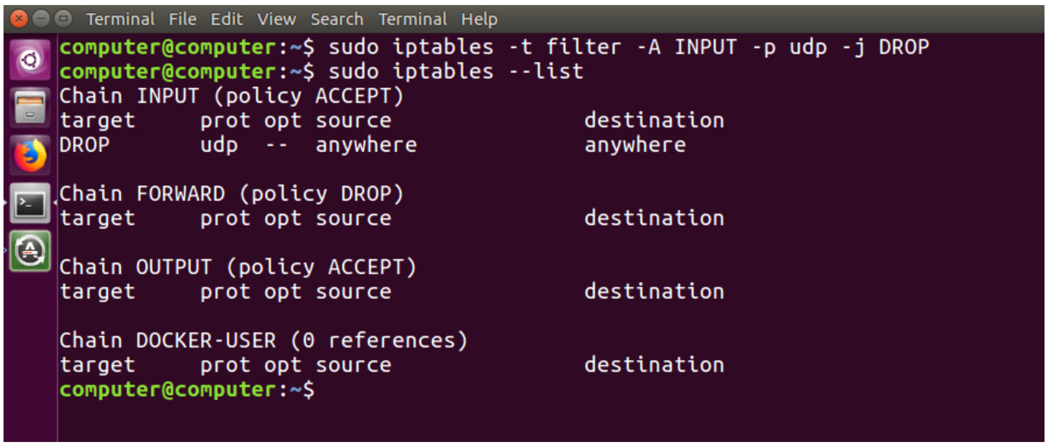
1. **-p, –proto :** is the protocol that the packet follows. Possible values maybe: tcp, udp, icmp, ssh etc.
   Syntax:

   ```
   iptables [-t table] -A [chain] -p {protocol_name} [target]
   ```

   Example: This command appends a rule in the INPUT chain to drop all udp packets.

   ```
   iptables -t filter -A INPUT -p udp -j DROP
   ```

   Output:



```
computer@computer:~$ sudo iptables -t filter -A INPUT -p udp -j DROP
computer@computer:~$ sudo iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       udp  --  anywhere             anywhere

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain DOCKER-USER (0 references)
target     prot opt source               destination
computer@computer:~$
```

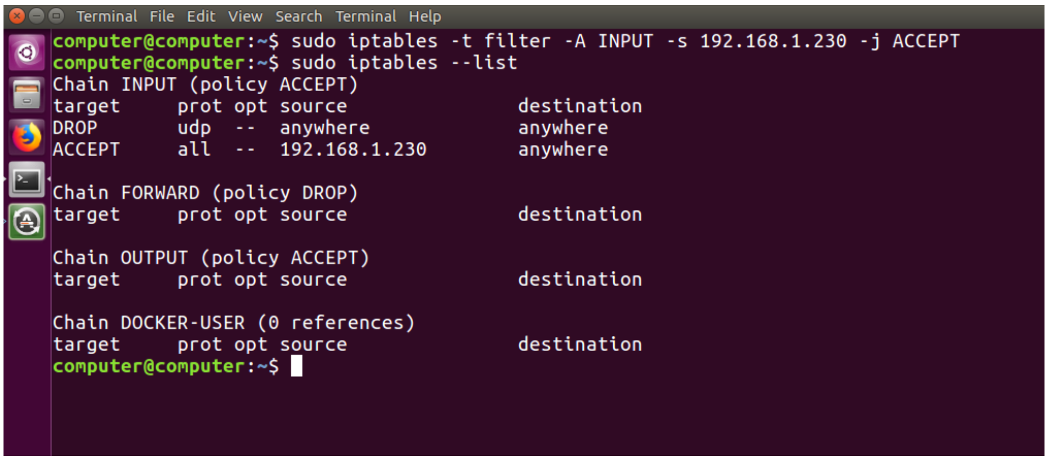
2. **-s, –source:** is used to match with the source address of the packet.
   Syntax:

   ```
   iptables [-t table] -A [chain] -s {source_address} [target]
   ```

   Example: This command appends a rule in the INPUT chain to accept all packets originating from 192.168.1.230.

   ```
   iptables -t filter -A INPUT -s 192.168.1.230 -j ACCEPT
   ```

   Output:



```
computer@computer:~$ sudo iptables -t filter -A INPUT -s 192.168.1.230 -j ACCEPT
computer@computer:~$ sudo iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       udp  --  anywhere             anywhere
ACCEPT     all  --  192.168.1.230        anywhere

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain DOCKER-USER (0 references)
target     prot opt source               destination
computer@computer:~$
```

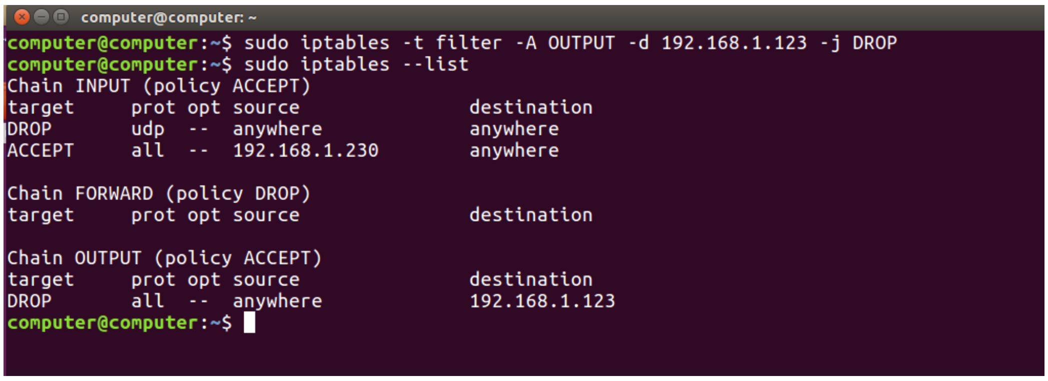
3. **-d, –destination :** is used to match with the destination address of the packet.
   Syntax:

```
iptables [-t table] -A [chain] -d {destination_address} [target]
```

Example: This command appends a rule in the OUTPUT chain to drop all packets destined for 192.168.1.123.

```
iptables -t filter -A OUTPUT -d 192.168.1.123 -j DROP
```

Output:



```
computer@computer:~$ sudo iptables -t filter -A OUTPUT -d 192.168.1.123 -j DROP
computer@computer:~$ sudo iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       udp  --  anywhere             anywhere
ACCEPT     all  --  192.168.1.230        anywhere

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             192.168.1.123
computer@computer:~$
```

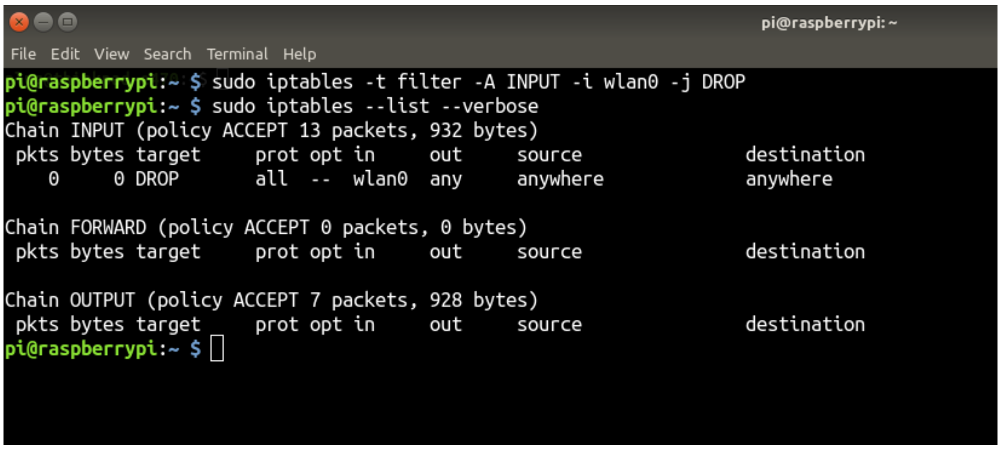
4. **-i, –in-interface :** matches packets with the specified in-interface and takes the action.
   Syntax:

```
iptables [-t table] -A [chain] -i {interface} [target]
```

Example: This command appends a rule in the INPUT chain to drop all packets destined for wireless interface.

```
iptables -t filter -A INPUT -i wlan0 -j DROP
```

Output:



```
pi@raspberrypi:~ $ sudo iptables -t filter -A INPUT -i wlan0 -j DROP
pi@raspberrypi:~ $ sudo iptables --list --verbose
Chain INPUT (policy ACCEPT 13 packets, 932 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       all  --  wlan0  any     anywhere             anywhere

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 7 packets, 928 bytes)
 pkts bytes target     prot opt in     out     source               destination
pi@raspberrypi:~ $
```

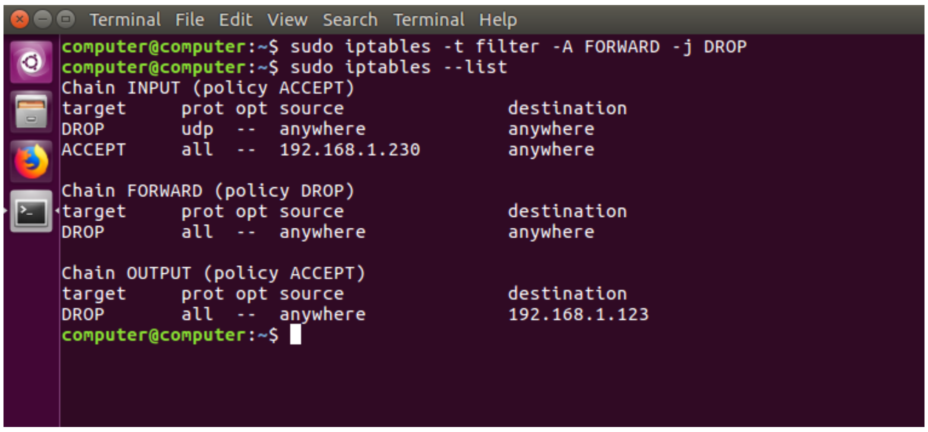
5. **-o, –out-interface :** matches packets with the specified out-interface.
6. **-j, –jump :** this parameter specifies the action to be taken on a match.
   Syntax:

```
iptables [-t table] -A [chain] [parameters] -j {target}
```

Example: This command adds a rule in the FORWARD chain to drop all packets.

```
iptables -t filter -A FORWARD -j DROP
```

Output:



```
computer@computer:~$ sudo iptables -t filter -A FORWARD -j DROP
computer@computer:~$ sudo iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       udp  --  anywhere             anywhere
ACCEPT     all  --  192.168.1.230        anywhere

Chain FORWARD (policy DROP)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             192.168.1.123
computer@computer:~$
```

Note:

- While trying out the commands, you can remove all filtering rules and user created chains.

```
sudo iptables --flush
```

- To save the iptables configuration use:

```
sudo iptables-save
```

- Restoring iptables config can be done with:

```
sudo iptables-restore
```

- There are other interfaces such ip6tables which are used to manage filtering tables for IPv6.

16 iptables tips and tricks for sysadmins | Opensource.com

Content Courtesy of https://opensource.com/article/18/10/iptables-tips-and-tricks
Modern Linux kernels come with a packet-filtering framework named Netfilter (https://en.wikipedia.org /wiki/Netfilter). Netfilter enables you to allow, drop, and modify traffic coming in and going out of a system. The **iptables** userspace command-line tool builds upon this functionality to provide a powerful firewall, which you can configure by adding rules to form a firewall policy. iptables (https://en.wikipedia.org/wiki/Iptables) can be very daunting with its rich set of capabilities and baroque command syntax. Let's explore some of them and develop a set of iptables tips and tricks for many situations a system administrator might encounter.

Avoid locking yourself out

Scenario: You are going to make changes to the iptables policy rules on your company's primary server. You want to avoid locking yourself—and potentially everybody else—out. (This costs time and money and causes your phone to ring off the wall.)

Tip #1: Take a backup of your iptables configuration before you start working on it.

Back up your configuration with the command:

```
/sbin/iptables-save > /root/iptables-works
```

Tip #2: Even better, include a timestamp in the filename.

Add the timestamp with the command:

```
/sbin/iptables-save > /root/iptables-works-`date +%F`
```

You get a file with a name like:

```
/root/iptables-works-2018-09-11
```

If you do something that prevents your system from working, you can quickly restore it:

```
/sbin/iptables-restore < /root/iptables-works-2018-09-11
```

Tip #3: Every time you create a backup copy of the iptables policy, create a link to the file with 'latest' in the name.

```
ln -s /root/iptables-works-`date +%F` /root/iptables-works-latest
```

Tip #4: Put specific rules at the top of the policy and generic rules at the bottom.

Avoid generic rules like this at the top of the policy rules:

```
iptables -A INPUT -p tcp --dport 22 -j DROP
```

The more criteria you specify in the rule, the less chance you will have of locking yourself out. Instead of the very generic rule above, use something like this:

```
iptables -A INPUT -p tcp --dport 22 -s 10.0.0.0/8 -d 192.168.100.101 -j DROP
```

This rule appends (**-A**) to the **INPUT** chain a rule that will **DROP** any packets originating from the CIDR block **10.0.0.0/8** on TCP (**-p tcp**) port 22 (**--dport 22**) destined for IP address 192.168.100.101 (**-d 192.168.100.101**).

There are plenty of ways you can be more specific. For example, using **-i eth0** will limit the processing to a single NIC in your server. This way, the filtering actions will not apply the rule to **eth1**.

Tip #5: Whitelist your IP address at the top of your policy rules.

This is a very effective method of not locking yourself out. Everybody else, not so much.

```
iptables -I INPUT -s <your IP> -j ACCEPT
```

You need to put this as the *first* rule for it to work properly. Remember, **-I** inserts it as the first rule; **-A** appends it to the end of the list.

Tip #6: Know and understand all the rules in your current policy.

Not making a mistake in the first place is half the battle. If you understand the inner workings behind your iptables policy, it will make your life easier. Draw a flowchart if you must. Also remember: What the policy does and what it is supposed to do can be two different things.

Set up a workstation firewall policy

Scenario: You want to set up a workstation with a restrictive firewall policy.

Tip #1: Set the default policy as DROP.

```
# Set a default policy of DROP

*filter

:INPUT DROP [0:0]

:FORWARD DROP [0:0]

:OUTPUT DROP [0:0]
```

Tip #2: Allow users the minimum amount of services needed to get their work done.

The iptables rules need to allow the workstation to get an IP address, netmask, and other important information via DHCP (**-p udp --dport 67:68 --sport 67:68**). For remote management, the rules need to allow inbound SSH (**--dport 22**), outbound mail (**--dport 25**), DNS (**--dport 53**), outbound ping (**-p icmp**), Network Time Protocol (**--dport 123 --sport 123**), and outbound HTTP (**--dport 80**) and HTTPS (**--dport 443**).

```
# Set a default policy of DROP

*filter

:INPUT DROP [0:0]

:FORWARD DROP [0:0]

:OUTPUT DROP [0:0]


# Accept any related or established connections

-I INPUT  1 -m state --state RELATED,ESTABLISHED -j ACCEPT

-I OUTPUT 1 -m state --state RELATED,ESTABLISHED -j ACCEPT


# Allow all traﬃc on the loopback interface

-A INPUT -i lo -j ACCEPT

-A OUTPUT -o lo -j ACCEPT


# Allow outbound DHCP request

-A OUTPUT -o eth0 -p udp --dport 67:68 --sport 67:68 -j ACCEPT


# Allow inbound SSH

-A INPUT -i eth0 -p tcp -m tcp --dport 22 -m state --state NEW  -j ACCEPT


# Allow outbound email

-A OUTPUT -i eth0 -p tcp -m tcp --dport 25 -m state --state NEW  -j ACCEPT


# Outbound DNS lookups

-A OUTPUT -o eth0 -p udp -m udp --dport 53 -j ACCEPT


# Outbound PING requests

-A OUTPUT -o eth0 -p icmp -j ACCEPT


# Outbound Network Time Protocol (NTP) requests

-A OUTPUT -o eth0 -p udp --dport 123 --sport 123 -j ACCEPT


# Outbound HTTP

-A OUTPUT -o eth0 -p tcp -m tcp --dport 80 -m state --state NEW -j ACCEPT

-A OUTPUT -o eth0 -p tcp -m tcp --dport 443 -m state --state NEW -j ACCEPT


COMMIT
```

Restrict an IP address range

Scenario: The CEO of your company thinks the employees are spending too much time on Facebook and not getting any work done. The CEO tells the CIO to do something about the employees wasting time on Facebook. The CIO tells the CISO to do something about employees wasting time on Facebook. Eventually, *you* are told the employees are wasting too much time on Facebook, and you have to do something about it. You decide to block all access to Facebook. First, find out Facebook's

IP address by using the **host** and **whois** commands.

```
host -t a www.facebook.com

www.facebook.com is an alias for star.c10r.facebook.com.

star.c10r.facebook.com has address 31.13.65.17

whois 31.13.65.17 | grep inetnum

inetnum:        31.13.64.0 - 31.13.127.255
```

Then convert that range to CIDR notation by using the CIDR to IPv4 Conversion (http://www.ipaddressguide.com/cidr) page. You get **31.13.64.0/18**. To prevent outgoing access to www.facebook.com (http://www.facebook.com), enter:

```
iptables -A OUTPUT -p tcp -i eth0 -o eth1 -d 31.13.64.0/18 -j DROP
```

Regulate by time

Scenario: The backlash from the company's employees over denying access to Facebook access causes the CEO to relent a little (that and his administrative assistant's reminding him that she keeps HIS Facebook page up-to-date). The CEO decides to allow access to Facebook.com only at lunchtime (12PM to 1PM). Assuming the default policy is DROP, use iptables' time features to open up access.

```
iptables -A OUTPUT -p tcp -m multiport --dport http,https -i eth0 -o eth1 -m time --timestart 12:00 --timestart
12:00 -timestop 13:00 -d

31.13.64.0/18  -j ACCEPT
```

This command sets the policy to allow (**-j ACCEPT**) http and https (**-m multiport --dport http,https**) between noon (**--timestart 12:00**) and 13PM (**--timestop 13:00**) to Facebook.com (**–d 31.13.64.0/18 (http://31.13.64.0/18)**).

Regulate by time—Take 2

Scenario: During planned downtime for system maintenance, you need to deny all TCP and UDP traffic between the hours of 2AM and 3AM so maintenance tasks won't be disrupted by incoming traffic. This will take two iptables rules:

```
iptables -A INPUT -p tcp -m time --timestart 02:00 --timestop 03:00 -j DROP

iptables -A INPUT -p udp -m time --timestart 02:00 --timestop 03:00 -j DROP
```

With these rules, TCP and UDP traffic (**-p tcp and -p udp**) are denied (**-j DROP**) between the hours of 2AM (**--timestart 02:00**) and 3AM (**--timestop 03:00**) on input (**-A INPUT**).

Limit connections with iptables

Scenario: Your internet-connected web servers are under attack by bad actors from around the world attempting to DoS (Denial of Service) them. To mitigate these attacks, you restrict the number of connections a single IP address can have to your web server:

```
iptables -A INPUT -p tcp -syn -m multiport --dport http,https -m connlimit --connlimit-above 20 -j REJECT --reje
ct-with-tcp-reset
```

Let's look at what this rule does. If a host makes more than 20 (**-–connlimit-above 20**) new connections (**–p tcp –syn**) in a minute to the web servers (**-–dport http,https**), reject the new connection (**–j REJECT**) and tell the connecting host you are rejecting the connection (**-–reject-with-tcp-reset**).

Monitor iptables rules

Scenario: Since iptables operates on a "first match wins" basis as packets traverse the rules in a chain, frequently matched rules should be near the top of the policy and less frequently matched rules should be near the bottom. How do you know which rules are traversed the most or the least so they can be ordered nearer the top or the bottom?

Tip #1: See how many times each rule has been hit.

Use this command:

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

The command will list all the rules in the chain (**-L**). Since no chain was specified, all the chains will be listed with verbose output (**-v**) showing packet and byte counters in numeric format (**-n**) with line numbers at the beginning of each rule corresponding to that rule's position in the chain.

Using the packet and bytes counts, you can order the most frequently traversed rules to the top and the least frequently traversed rules towards the bottom.

Tip #2: Remove unnecessary rules.

Which rules aren't getting any matches at all? These would be good candidates for removal from the policy. You can find that out with this command:

```
iptables -nvL | grep -v "0     0"
```

Note: that's not a tab between the zeros; there are five spaces between the zeros.

Tip #3: Monitor what's going on.

You would like to monitor what's going on with iptables in real time, like with **top**. Use this command to monitor the activity of iptables activity dynamically and show only the rules that are actively being traversed:

```
watch --interval=5 'iptables -nvL | grep -v "0     0"'
```

watch runs **'iptables -nvL | grep -v "0   0"'** every five seconds and displays the first screen of its output. This allows you to watch the packet and byte counts change over time.

Report on iptables

Scenario: Your manager thinks this iptables firewall stuff is just great, but a daily activity report would be even better. Sometimes it's more important to write a report than to do the work.

Use the packet filter/firewall/IDS log analyzer FWLogwatch (http://fwlogwatch.inside-security.de/) to create reports based on the iptables firewall logs. FWLogwatch supports many log formats and offers many analysis options. It generates daily and monthly summaries of the log files, allowing the security administrator to free up substantial time, maintain better control over network security, and reduce unnoticed attacks.

Here is sample output from FWLogwatch:

More than just ACCEPT and DROP

We've covered many facets of iptables, all the way from making sure you don't lock yourself out when working with iptables to monitoring iptables to visualizing the activity of an iptables firewall. These will get you started down the path to realizing even more iptables tips and tricks.

Most Frequently Used Linux IPTables Rules with Examples

Content Courtesy of https://www.tutorialspoint.com/articles/most-frequently-used-linux-iptables-rules-with-examples

This article will help you to create IPtables rules that you can directly use for your daily or routine needs, These examples will act as basic templates for you to work on iptables with these rules which suit your specific requirement.

Deleting the IPtables or Existing Rules

Before you start building new IPtables set of rules, you should clean up all the default rules, and existing rules. Use the IPtables flush command, below are some examples –

```
#iptables --flush
(or)
# iptables --F
```

Default Policies Chain

The default policy is ACCEPT, change the policy to DROP for all the INPUT, FORWARD, OUTPUT.

```
# iptables -P INPUT DROP
# iptables -P FORWARD DROP
# iptables -P OUTPUT DROP
```

For every firewall rule, we need to define two rules, i.e., one for In-coming and another for Out-going.

If we trust the internal users, we can use the DROP for incoming rules, and the default outgoing will be ACCEPT.

Allowing HTTP & HTTPS Incoming Connections

The below rules will allow all the incoming traffic of HTTP & HTTPS (80 & 443)

```
iptables -A INPUT -i eth0 -p tcp --dport 80 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 80 -m state --state ESTABLISHED -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --dport 443 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 443 -m state --state ESTABLISHED -j ACCEPT
```

Allowing only SSH to a Network

The below rules will allow only outgoing ssh connection from the internal network means we can ssh only from 192.168.87.0/24 network only

```
iptables -A OUTPUT -o eth0 -p tcp -d 192.168.100.0/24 --dport 3306 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --sport 3306 -m state --state ESTABLISHED -j ACCEPT
```

Allowing the Incoming MySQL port (3306) for TCP Traffic.

Below is the example which has incoming & outgoing traffic on port 3306 (mysql) for eth0 adaptor.

```
iptables -A INPUT -i eth0 -p tcp --dport 3306 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 3306 -m state --state ESTABLISHED -j ACCEPT
```

Allowing Incoming MySQL Port (3306) for a Specific Network

The below example will allow 3306 (mysql) for a specific network 192.168.87.x.

```
iptables -A INPUT -i eth0 -p tcp -s 192.168.87.0/24 --dport 3306 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 3306 -m state --state ESTABLISHED -j ACCEPT
```

Allowing Multiple Ports with a Single Rule

The below rules will allow incoming connections from outside to multiple ports, instead of writing multiple rules, we can also write rules with multiple ports together as shown below.

Here, were are allowing mysql, Http & Https in a single rule.

```
iptables -A INPUT -i eth0 -p tcp -m multiport --dports 3306,80,443 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp -m multiport --sports 3306,80,443 -m state --state ESTABLISHED -j ACCEPT
```

Allowing Outgoing MySQL

This is different from the incoming connection, we allow both new and established connections on the OUTPUT chain, but whereas in INPUT, we allow only the established chain.

This rule will allow only outgoing connection to MySQL when we try to connect to MySQL server from our Linux box.

```
iptables -A OUTPUT -o eth0 -p tcp --dport 3306-m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --sport 3306 -m state --state ESTABLISHED -j ACCEPT
```

Allow Sendmail Traffic

These rules will allow mails using sendmail or postfix port 25.

```
iptables -A INPUT -i eth0 -p tcp --dport 25 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 25 -m state --state ESTABLISHED -j ACCEPT
```

Allowing IMAP & POP3 Ports

This rule will allow to send or receive emails from IMAP or POP3

```
iptables -A INPUT -i eth0 -p tcp --dport 143 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 143 -m state --state ESTABLISHED -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --dport 110 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 110 -m state --state ESTABLISHED -j ACCEPT
```

Forward a Port to 5722 to 22(SSH)

These rules will forward the total traffic which comes from port 5722 to port 22. That means, the incoming connection for ssh can come from both 5722 and 22.

```
iptables -t nat -A PREROUTING -p tcp -d 192.168.87.100 --dport 5722 -j DNAT --to 192.168.87.200:22
```

Allowing Port 873 (rsync) for Backups

These rules will allow to you to take backups or copy data using rsync from a specific network

```
iptables -A INPUT -i eth0 -p tcp -s 192.168.87.0/24 --dport 873 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 873 -m state --state ESTABLISHED -j ACCEPT
```

Blocking an IP address

If we want to block a particular IP address.

```
BLOCK_ADDRESS="192.168.87.100"
# iptables -A INPUT -s "$BLOCK_ADDRESS" -j DROP
```

This will be useful if we want to block some IP address where they are downloading or trying to access the server, where we can block the IP for further investigation.

# iptables -A INPUT -i eth0 -s “$ BLOCK_ADDRESS ” -j DROP
# iptables -A INPUT -i eth0 -p tcp -s “$ BLOCK_ADDRESS ” -j DROP

This above example will block the TCP/IP traffic on the eth0 for that particular IP address.

We can add a network in the variable if you want to restrict access to the server from outside

> By using the above iptables rules or modifying the rules and ports, we can secure the connection or network/server. We can also modify the network or ports accordingly to fit our environment. And these iptables rules are written in a simple shell script format, so we can use them in writing the shell scripts to apply on multiple servers.

References