**Linux IPTables Firewall Configuration**

<http://www.justlinux.com/nhf/Security/IPtables_Basics.html>  
Written By: [Prince\_Kenshi](mailto:snailboy1@yahoo.com)

First you need to know how the firewall treats packets leaving, entering, or passing through your computer. Basically there is a chain for each of these. Any packet entering your computer goes through the **INPUT** chain. Any packet that your computer sends out to the network goes through the **OUTPUT** chain. Any packet that your computer picks up on one network and sends to another goes through the **FORWARD** chain. The chains are half of the logic behind iptables themselves.

Now the way that iptables works is that you set up certain rules in each of these chains that decide what happens to packets of data that pass through them. For instance, if your computer was to send out a packet to www.yahoo.com to request an HTML page, it would first pass through the OUTPUT chain. The kernel would look through the rules in the chain and see if any of them match. The first one that matches will decide the outcome of that packet. If none of the rules match, then the policy of the whole chain will be the final decision maker. Then whatever reply Yahoo! sent back would pass through the INPUT chain. It's no more complicated than that.

Now that we have the basics out of the way, we can start working on putting all this to practical use. There are a lot of different letters to memorize when using iptables and you'll probably have to peek at the man page often to remind yourself of a certain one. Now let's start with manipulation of certain IP addresses. Suppose you wanted to **block all packets coming from 200.200.200.1**. First of all, -s is used to specify a source IP or DNS name. So from that, to refer to traffic coming from this address, we'd use this:

**iptables -s 200.200.200.1**

But that doesn't tell what to do with the packets. **The -j option is used to specify what happens to the packet.** The most common three are **ACCEPT**, **REJECT**, and **DROP**. Now you can probably figure out what ACCEPT does and it's not what we want. REJECT sends a message back that this computer isn't accepting connections. DROP just totally ignores the packet. If we're really suspicious about this certain IP address, we'd probably prefer DROP over REJECT. So here is the command with the result:

**iptables -A INPUT -s 200.200.200.1**

The command above will work correctly. You use -A to specific the chain that you want to append the rule to. –I and –R can also be used to Insert a Rule and Replace a rule when you specify the rule number. The default action is DROP, so the above command will work. So here's the entire command:

**iptables -A INPUT -s 200.200.200.1 -j DROP**

This single command **iptables -A INPUT -s 200.200.200.1 -j DROP** would ignore everything coming from 200.200.200.1 (with exceptions, but we'll get into that later). **The order of the options doesn't matter**; the -j DROP could go before -s 200.200.200.1. I just like to put the outcome part at the end of the command. Ok, we're now capable of ignoring a certain computer on a network. **If you wanted to keep your computer from talking to it, you'd simply change INPUT to OUTPUT and change the -s to -d for destination**. Now that's not too hard, is it?

So what if we only wanted to **ignore telnet requests from 200.200.200.1**? Well that's not very hard either. You might know that port 23 is for telnet, but you can just use the word telnet if you like. There are at least 3 protocols that can be specified: TCP, UDP, and ICMP. Telnet, like most services, runs on TCP so we're going with it. The -p option specifies the protocol. But TCP doesn't tell it everything; telnet is only a specific protocol used on the larger protocol of TCP. After we specify that the protocol is TCP, we can use --destination-port to denote the port that they're trying to contact us on. Make sure you don't get source and destination ports mixed up. Remember, the client can run on any port, it's the server that will be running the service on port 23. Any time you want to block out a certain service, you'll use --destination-port. The opposite is --source-port in case you need it. So let's put this all together. This should be the command that accomplishes what we want:

**iptables -A INPUT -s 200.200.200.1 -p tcp --dport 23 -j DROP**

And there you go. If you wanted to **specify a range of IP's**, you could use **200.200.200.0/24**. This would specify any IP that matched **200.200.200.\***. Now it's time to fry some bigger fish. Let's say that, like me, you have a local area network and then you have a connection to the internet. We're going to also say that the **LAN is eth0** while the Internet connection is eth1. Now suppose we wanted to allow telnet to run as a service to computers on the LAN but not on the insecure internet. Well there is an easy way to do this. We can use **-i for the input interface** and **-o for the output interface**. You could always block it on the OUTPUT chain, but we'd rather block it on the INPUT so that the telnet daemon never even sees the request. Therefore we'll use -i. This should set up just the rule:

**iptables -A INPUT -p tcp --dport 23 -i eth1 -j DROP**

So this should close off the port to anyone on the Internet yet kept it open to the LAN. Now before we go on to more intense stuff, I'd like to briefly explain other ways to manipulate rules. The **-A option appends a rule to the end of the list**, meaning any matching rule before it will have say before this one does. If we wanted to put a rule before the end of the chain, we use **-I for insert**. This will **put the rule in a numerical location in the chain**. For example, if we wanted to put it at the top of the INPUT chain, we'd use "**-I INPUT 1**" along with the rest of the command. Just change the 1 to whatever place you want it to be in. Now let's say we wanted to replace whatever rule was already in that location. Just **use -R to replace a rule**. It has the same syntax as -I and works the same way except that it **deletes the rule at that position** rather than bumping everything down. And finally, if you just want **to delete a rule, use -D**. This also has a similar syntax but you can either use a number for the rule or type out all the options that you would if you created the rule. The number method is usually the optimal choice. There are two more simple options to learn though. **-L lists all the rules set so far**. This is obviously helpful when you forget where you're at. And **-F** flushes a certain chain. (It removes all of the rules on the chain.) **If you don't specify a chain, it will basically flush everything**.

Well let's get a bit more advanced. We know that these packets use a certain protocol, and if that protocol is TCP, then it also uses a certain port. Now you might be compelled to just close all ports to incoming traffic, but remember, after your computer talks to another computer, that computer must talk back. If you close all of your incoming ports, you'll essentially render your connection useless. And for most non-service programs, you can't predict which port they're going to be communicating on. But there's still a way. Whenever two computers are talking over a TCP connection, that connection must first be initialized. This is the job of a SYN packet. A SYN packet simply tells the other computer that it's ready to talk. Now only the computer requesting the service sends a SYN packet. So if you only block incoming SYN packets, it stops other computers from opening services on your computer but doesn't stop you from communicating with them. It roughly makes your computer ignore anything that it didn't speak to first. It's mean but it gets the job done. Well the option for this is **--syn** after you've specified the TCP protocol. So to make a rule that would **block all incoming connections only on the Internet**:

**iptables -A INPUT -i ppp0 -p tcp --syn -j DROP**

That's a likely rule that you'll be using unless you have a web service running. If you want to leave one port open, for example 80 (HTTP), there's a simple way to do this too. As with many programming languages, an exclamation mark means not. For instance, if you wanted to **block all SYN packets on all ports except 80**, I believe it would look something like this:

**iptables -A INPUT -i eth1 -p tcp --syn --dport ! 80 -j DROP**

It's somewhat complicated but it's not so hard to comprehend. There's one last thing I'd like to cover and that's changing the policy for a chain. **The chains INPUT and OUTPUT are usually set to ACCEPT by default and FORWARD is set to REJECT**. Well if you want to use this computer as a router, you would probably want to set the FORWARD policy to ACCEPT. How do we do this you ask? Well it's really very simple. All you have to do is **use the -P option**. Just follow it by the chain name and the new policy and you have it made. To change the FORWARD chain to an ACCEPT policy, we'd do this:

**iptables -P FORWARD ACCEPT**

Nothing to it, huh? This is really just the basics of iptables. It should help you set up a limited firewall but there's still a lot more that I couldn't talk about. You can look at the man page "man iptables" to learn more of the options (or refresh your memory when you forget). You can find more advanced documents if you want to learn some of the more advanced features of iptables. **Iptables Connection Tracking - TCP**

A tcp connection is initiated via a three-way handshake involving a synchronization request from the client, a synchronization and an acknowledgement from the server, and finally an acknowledgement from the client. Subsequent traffic flowing between server and client is acknowledged in all cases. The sequence looks like:

Client Server

SYN ---->

<---- SYN+ACK

ACK ---->

.….

.….

SYN and ACK refer to flags set in the tcp header. There are also 32 bit sequence and acknowledgement numbers stored in the tcp header which are passed back and forth and updated during the session. To get connection tracking to work for a tcp connection you need a rule that looks something like this:

**iptables -A INPUT -p tcp -m state –-state ESTABLISHED -j ACCEPT**

**iptables -A OUTPUT -p tcp -m state –-state NEW,ESTABLISHED -j ACCEPT**

Connection tracking only knows about NEW,ESTABLISHED,RELATED and INVALID, classified as described above.

When a packet with the SYN+ACK flags set arrives in response to a packet with SYN set the connection tracking thinks: “I have been just seeing a packet with SYN+ACK which answers a SYN I had previously seen, so this is an ESTABLISHED connection.”

If you want to drop all new inbound tcp connections originating from the outside (i.e. initiating a three-way handshake), use the following rule:

**iptables -A INPUT-p tcp --syn -m state --state NEW-j DROP**

**Iptables Connection Tracking - ICMP**

In iptables parlance, there are only four types of icmp that can be categorized as NEW, or ESTABLISHED:

1) Echo request (ping, type=8) and echo reply (pong, type=0).

2) Timestamp request (13)and reply (14).

3) Information request (15) and reply (16).

4) Address mask request (17) and reply (18).

The request in each case is classified as NEW and the reply as ESTABLISHED.

Other types of icmp are not request-reply based and can only be RELATED to other connections.

Let us consider a sample ruleset and a few examples:

**iptables -A OUTPUT -p icmp -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT**

**iptables -A INPUT -p icmp -m state --state ESTABLISHED,RELATED -j ACCEPT**

* An icmp echo request is NEW and so is allowed in the OUTPUT chain.
* An icmp echo reply, provided it is in response to an echo request, is ESTABLISHED and so is allowed in the INPUT chain. An echo reply cannot be allowed in the OUTPUT chain for the rules above because there is no NEW in the INPUT chain to allow echo requests and a reply has to be in response to a request.
* An icmp redirect, because it is not request-reply based, is RELATED and so can be allowed in both the INPUT and the OUTPUT chains provided there is already a tcp or udp connection in the state table already that it can be matched against.

Firstly, you need to load the ip\_conntrack\_ftp module.Assuming you have a single-homed box, a simple ruleset to allow an ftp connection would be:

**iptables -A INPUT -p tcp --sport 21 -m state --state ESTABLISHED -j ACCEPT**

**iptables -A OUTPUT -p tcp --dport 21 -m state --state NEW,ESTABLISHED -j ACCEPT**

(Please note, I am assuming here you have a separate ruleset to allow any icmp RELATED to the conection. Please see my example ruleset for this).

**Iptables Connection Tracking - FTP**

An ftp connection also needs a data-channel, which can be provided in one of two ways:

**(1) Active Ftp**

The ftp client sends a port number over the ftp channel via a PORT command to the ftp server. The ftp server then connects from port 20 to this port to send data, such as a file, or the output from an ls command. The ftp-data connection is in the opposite sense from the original ftp connection.

To allow active ftp without knowing the port number that has been passed we need a general rule which allows connections from port 20 on remote ftp servers to high ports (port numbers > 1023) on ftp clients. This is simply too general to ever be secure.

Enter the ip\_conntrack\_ftp module. This module is able to recognize the PORT command and pick-out the port number. As such, the **ftp-data connection can be classified as RELATED** to the original outgoing connection to port 21 so we don’t need NEW as a state match for the connection in the INPUT chain. The following rules will serve our purposes grandly:

**iptables -A INPUT-p tcp --sport 20 -m state --state ESTABLISHED,RELATED -j ACCEPT**

**iptables -A OUTPUT -p tcp --dport 20 -m state --state ESTABLISHED -j ACCEPT**

**(2) Passive Ftp**

A PORT command is again issued, but this time it is from the server to the client. The client connects to the server for data transfer. Since the connection is in the same sense as the original ftp connection, passive ftp is inherently more secure than active ftp, but note that this time we know even less about the port numbers. Now we have a connection between almost arbitrary port numbers.

Enter the ip\_conntrack\_ftp module once more. Again, this module is able to recognize the PORT command and pick-out the port number. Instead of NEW in the state match for the OUTPUT chain, we can use RELATED. The following rules will suffice:

**iptables -A INPUT -p tcp --sport 1024: --dport 1024**

**-m state --state ESTABLISHED -j ACCEPT**

**iptables -A OUTPUT -p tcp –sport 1024: --dport 1024**

**-m state --state ESTABLISHED,RELATED -j ACCEPT**

**iptables command examples**

**# delete old rules**

**iptables -F**

**iptables -t nat -F**

**# enable ip forwarding**

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

**# enable masquerading**

**iptables -t nat -A POSTROUTING -o eth1 -s 141.28.227.0/16 -j MASQUERADE**

**# network**

**iptables -A INPUT -s 141.28.227.0/24 -i eth0 -j ACCEPT**

**iptables -A OUTPUT -d 141.28.227.0/24 -o eth0 -j ACCEPT**

**# ip forwarding**

**iptables -A FORWARD -i eth0 -d 141.28.227.178 -j ACCEPT**

**iptables -A FORWARD -o eth0 -s 141.28.227.178 -j ACCEPT**

**# router rules**

**iptables -A INPUT -i ppp0 -p tcp --dport 53 -j ACCEPT**

**iptables -A OUTPUT -o ppp0 -p tcp --sport 53 -j ACCEPT**

**iptables -A INPUT -i ppp0 -p udp --dport 53 -j ACCEPT**

**iptables -A OUTPUT -o ppp0 -p udp --sport 53 -j ACCEPT**

**iptables -A INPUT -i ppp0 -p tcp --dport 80 -j ACCEPT**

**iptables -A OUTPUT -o ppp0 -p tcp --sport 80 -j ACCEPT**

**# loopback**

**iptables -A INPUT -i lo -j ACCEPT**

**iptables -A OUTPUT -o lo -j ACCEPT**

**# route to the other networks**

**route add -net 141.28.224.0 gw 141.28.227.254 netmask 255.255.255.0**

**route add -net 141.28.228.0 gw 141.28.227.254 netmask 255.255.255.0**

**# policies**

**iptables -P INPUT DROP**

**iptables -P OUTPUT DROP**

**iptables -P FORWARD DROP**