iptables: Linux firewall rules for a basic Web Server

Benjamin Cane

Sep 17, 2012 · 16 min read · Administration, Firewalls, How To and Tutorials, Linux, Networking, Security

For today's article I am going to explain how to create a basic firewall allow and deny filter list using the iptables package. We will be focused on creating a filtering rule-set for a basic everyday Linux web server running Web, FTP, SSH, MySQL, and DNS services.

Before we begin lets get an understanding of iptables and firewall filtering in general.

What is iptables?

`iptables` is a package and kernel module for Linux that uses the netfilter hooks within the Linux kernel to provide filtering, network address translation, and packet mangling. iptables is a powerful tool for turning a regular Linux system into a simple or advanced firewall.

Firewall & iptables basics

Rules are first come first serve

In iptables much like other (but not all) firewall filtering packages the rules are presented in a list. When a packet is being processed, iptables will read through its rule-set list and the first rule that matches this packet completely gets applied.

For example if our rule-set looks like below, all HTTP connections will be denied:

1. Allow all SSH Connections
2. Deny all connections
3. Allow all HTTP Connections

If the packet was for SSH it would be allowed because it matches rule #1, HTTP traffic on the other hand would be denied because it matches both rule #2 and rule #3. Because rule #2 says Deny all connections the HTTP traffic would be denied.

This is an example of why order matters with iptables, keep this in mind as we will see this later in this article.

Two policies for filtering

When creating filtering rule-sets there are two methods **explicit allow** or **explicit deny**.

Explicit Allow (Default Deny)

Explicit Allow uses a default deny policy, when creating these types of rules the base policy is a deny all rule. Once you have a base policy that denies all traffic you must add rules (in the proper order) that explicitly allow access.

Explicit Allow is by default more secure than Explicit Deny, because only traffic that you explicitly allow is allowed into the system. The trade off, however is that you must manage a sometimes complicated rule set. If the rule set is not correct you may even end up locking yourself out of the

system. It is not that uncommon to lock yourself out of your system when creating a explicit allow policy.

Explicit Deny (Default Allow)

Explicit Deny uses a default allow policy, when creating these types of rules the base policy is to allow all. Once you have a base policy that allows all traffic you can create additional rules that explicitly deny access.

When using Explicit Deny only traffic that you specifically block is denied, this means that your system is more vulnerable if you forget to block a port that should be blocked. However this also means that you are less likely to lock yourself out of the system.

Because we are creating a filtering released for a public facing web server we will be covering a Explicit Allow policy, as it is more secure and will show you some best practices when creating iptables rules.

Chains and Tables

Tables

Tables are used by iptable to group rules based on their function. The 3 table types available are Filter, NAT, and Mangle.

- Filter - The filter table is used to filter network traffic based on allow or deny rules the user has put in place. This is the default table to use if no tables were given in a command.
- NAT - The NAT Table is used for Network Address Translation. This is useful for configuring your Linux System as a router or port forwarding.
- Mangle - Mangle is used for packet mangling, this is mainly useful for changing the TOS or TTL on packets

For our examples today we are focusing on only the filter table, which is also the default. When running iptables commands we will not need to specify the table we are using in our examples.

Chains

Chains on the other hand are used to organize rules based on the source of the packets. A chain is essentially a collection of rules within a specific table. The Filter table for example has 3 default chains INPUT, OUTPUT, and FORWARD these chains are used to process packets based on the type of packet.

- INPUT - This chain is used for packets that are incoming on the system from an outside source.
- OUTPUT - This chain is used for packets that are outgoing from the system to an outside source.
- FORWARD - This chain is used for packets that are being forwarded through NAT rules, this allows you to filter traffic that is also NAT'ed

In addition to the chains above custom chains can be created as well. You can add a rule within another chain (such as INPUT) that targets a custom chain. By placing such a rule in the beginning of a chain, you can save unnecessary rule processing.

We will save creating custom chains for another day as you do not really need them for a basic web server.

Creating a rule set for a basic web server

Verify iptables is installed

Before starting to write iptables rules; lets validate that the package is installed and the iptables module is loaded.

Verifying the package is installed (Ubuntu/Debian)

```
# dpkg --list | grep iptables
ii iptables 1.4.12-1ubuntu4 administration tools for packet filtering and NAT
```

Verifying the Kernel Module is loaded

```
# lsmod | grep ip_tables
ip_tables 18106 1 iptable_filter
```

Once you have validated that iptables is loaded and ready, we can start creating rules.

Creating iptables rules

When implementing iptables there are two methods. Rules can be added to a file that gets loaded on reboot/restart of the iptables services, or rules can be added live and then saved to the file that is loaded on reboot/restart of iptables. For today's examples I will show adding the rules live and saving them to a file once complete.

I prefer this method as it provides me with the ability to simply reboot the system and regain access if I mess up any rules. While this may work for virtual machines and physical machines that are close by, this is always a concern when applying firewall rules to remote machines that you do not have quick access to.

Always triple check your rules before applying them. As a best practice if you are applying the rules to a remote machine it is best to test your rules on a machine that is close by first.

Show current rules

Before we start creating rules I want to cover how to show rules that are currently being enforced. We will use this command often to verify whether our rules are in place or not. The below command shows no rules are in place.

```
# iptables -L
Chain INPUT (policy ACCEPT)
 target prot opt source destination

Chain FORWARD (policy ACCEPT)
 target prot opt source destination

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

Add port 22, Before you add the base policy

No matter if I am adding a explicit deny or explicit allow base policy I always add a rule that allows me access to port 22 (SSH) as the first rule. This allows me to ensure that SSH is always available to me even if I mess up the other rules.

Since my SSH connections are coming from the IP `192.168.122.1` I will explicitly allow all `port 22` connections from the IP `192.168.122.1`.

```
# iptables -I INPUT -p tcp --dport 22 -s 192.168.122.1 -j ACCEPT
# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination
ACCEPT tcp -- 192.168.122.1 anywhere tcp dpt:ssh
```

Lets break this command down a bit, and get a better understanding of iptables syntax.

```
-I INPUT
```

The `-I` flag tells iptables to take this rule and **insert** it into the INPUT chain. This will put the given rule as the first rule in the set; there are other methods of adding rules such as appending the rule or adding the rule into a specific line that we will show in the next few examples.

```
-p tcp
```

The `-p` flag is part of the rule that tells iptables what**protocol** that we want to match on, in this case it is TCP.

```
--dport 22
```

The `--dport` flag is short for **destination port** and is used to specify that we are only looking to match on traffic destined for port 22.

```
-s 192.168.122.1
```

The `-s` flag as you can probably guess is used to specify the **source** of the traffic, in our case the source of our port 22 traffic will be 192.168.122.1.As a note you must be careful to ensure you have a proper source when applying this rule.

```
-j ACCEPT
```

The `-j` flag tells iptables to **jump** to the action for this packet, these are known in iptables as targets. A target can be a chain or one of the predefined targets within iptables, the target in our example will tell iptables to ACCEPT the packet and allow it to pass through the firewall.

If you have a dynamic IP the IP address that you are trying to access the server from may change. If that is the case it may be preferable to leave out the source which would allow all port 22 traffic.

Example to allow SSH traffic from all sources:

```
# iptables -I INPUT -p tcp --dport 22 -j ACCEPT
```

Changing the base policy to default deny (Explicit Allow)

Now that we have the SSH traffic allowed we can change the base policy of the INPUT chain to deny all, this means that anything that does not match any of our rules will be denied. In some filtering systems this needs to be done by adding a rule at the end of the set that denies everything. In iptables however this can be changed by changing the INPUT chains default policy.

```
# iptables -P INPUT DROP
# iptables -L
Chain INPUT (policy DROP)
target prot opt source destination
ACCEPT tcp -- 192.168.122.1 anywhere tcp dpt:ssh
```

The `-P` flag stands for **policy**, in this command we are setting the default policy of the INPUT chain to the target DROP.

DROP vs REJECT

Within the default policy I showed using the DROP target. There are two targets in iptables that can deny a packet, DROP and REJECT. There is a very small but possibly very impactful difference in these two targets.

The REJECT target will send a reply icmp packet to the source system telling that system that the packet has been rejected. By default the message will be “port is unreachable”.

The DROP target simply drops the packet without sending any reply packets back.

The REJECT target is vulnerable to DoS style attacks as every packet that is rejected will cause iptables to send back an icmp reply, when an attack is at volume this causes your system to also send icmp replies in volume. In this scenario it is better to simply drop the packet and reduce traffic congestion as best you can.

For this reason I suggest using DROP as a default reply.

Appending our web server rules

Now that we have the default policy and SSH rules in place we need to add rules for the other services we want to make accessible. The first rule we will add will accept and allow all traffic destined to port 80 and port 443 for web traffic.

```
# iptables -A INPUT -p tcp --dport 80 -j ACCEPT
# iptables -A INPUT -p tcp --dport 443 -j ACCEPT
# iptables -L -n
Chain INPUT (policy DROP)
target prot opt source destination
ACCEPT tcp -- 192.168.122.1 0.0.0.0/0 tcp dpt:22
ACCEPT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:80
ACCEPT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:443
```

As you can see in the above example we added the rule by using `-A INPUT`. While `-I` is used to insert a rule into the beginning or a chains rule-set, the `-A` flag is used to **append** a rule at the end of the rule-set.

Allowing Loopback Traffic

In addition to allowing web server traffic we should also allow all traffic on the loopback interface, this is necessary if you are running a local MySQL server and connecting via localhost; but also a good idea in general for the loopback interface.

```
# iptables -I INPUT -i lo -j ACCEPT
# iptables -L -n
Chain INPUT (policy DROP)
target prot opt source destination
ACCEPT all -- 0.0.0.0/0 0.0.0.0/0
ACCEPT tcp -- 192.168.122.1 0.0.0.0/0 tcp dpt:22
```

We can tell iptables to allow all traffic on a specific **interface** with the `-i` flag, and because we didn't add `-p` the default action is to allow all protocols.

I placed this rule first in the chain because in general there is quite a bit of traffic on the loopback interface and having the rule earlier in the ruleset provides faster matching and allows the kernel to spend less time trying to match the packets.

Allowing FTP Traffic

FTP traffic can be tricky when it comes to firewalls; that is because FTP uses multiple ports when transferring data and sometimes not the same port.

Active mode FTP uses port 21 for a control channel and port 20 for data. This is easily enough accounted for with the first two iptables rules in the example below.

Passive mode FTP however is not as simple, once a client has connected to the control port the FTP server, the FTP client can establish a higher port for the client to connect to. Because this higher port is often within a large range it is difficult to open up the entire range without possibly allowing malicious traffic to unwanted ports.

For this scenario iptables uses another module called ip_conntrack; ip_conntrack tracks established connections and allows iptables to create rules that allows related connections to be accepted.

This allows for the FTP connection to establish on port 21 with the first rule in the list and then establish a connection with a higher port via the third rule.

```
# iptables -A INPUT -p tcp --dport 21 -j ACCEPT
# iptables -A INPUT -p tcp --dport 20 -j ACCEPT
# iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
# iptables -L -n
Chain INPUT (policy DROP)
target prot opt source destination
ACCEPT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:21
ACCEPT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:20
ACCEPT all -- 0.0.0.0/0 0.0.0.0/0 ctstate RELATED,ESTABLISHED
```

In the third rule being applied the -m flag stands for match, this allows iptables to match connections based on the **connection state** (-ctstate) ESTABLISHED or RELATED.

RELATED is a connection type within iptables that matches new connections that are related to already established connections. This is really mainly used with FTP based traffic.

You may also want to restrict FTP traffic to a certain network, to do this simply add the `-s` flag and the appropriate ip/ip range to the first and second iptables rules.

Loading the ip_conntrack_ftp module

In order for conntrack to work properly you must ensure that the ip_conntrack_ftp module is loaded.

```
# modprobe ip_conntrack_ftp
# lsmod | grep conntrack
nf_conntrack_ftp 13452 0
nf_conntrack_ipv4 19716 1
nf_defrag_ipv4 12729 1 nf_conntrack_ipv4
xt_conntrack 12760 1
nf_conntrack 81926 4 nf_conntrack_ftp,xt_state,nf_conntrack_ipv4,xt_conntrack
x_tables 29846 7 xt_state,ip6table_filter,ip6_tables,xt_tcpudp,xt_conntrack,iptable_filt
```

Allowing DNS Traffic

DNS based traffic is not as tricky as FTP however DNS traffic primarily uses UDP rather than TCP. However some DNS traffic can be over TCP traffic. In order to allow DNS traffic you must specify 2 iptables commands one opening the port for TCP and the other opening the port for UDP.

```
# iptables -A INPUT -p tcp --dport 53 -j ACCEPT
# iptables -A INPUT -p udp --dport 53 -j ACCEPT
# iptables -L -n
Chain INPUT (policy DROP)
target prot opt source destination
ACCEPT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:53
ACCEPT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:53
```

Blocking IP's

Sometimes the internet is not as friendly as one would like, eventually you may need to block a specific IP address or range of IP's. For this example we are going to block the IP range of 192.168.123.0/24.

```
# iptables -I INPUT 3 -s 192.168.123.0/24 -j DROP
# iptables -L -n
Chain INPUT (policy DROP)
target prot opt source destination
ACCEPT all -- 0.0.0.0/0 0.0.0.0/0
ACCEPT tcp -- 192.168.122.1 0.0.0.0/0 tcp dpt:22
DROP all -- 192.168.123.0/24 0.0.0.0/0
ACCEPT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:80
```

The above rule is a little trickier than others because we already have a rule that accepts all traffic on port 80.

Our rule is to block all traffic from the IP range 192.168.123.0/24; in order for this to be blocked it must be listed in a specific spot within our ruleset. With iptables the first rule that matches a packet will be applied. For our rule to work we must add it before the port 80 rule otherwise the iprange will still be able to connect to port 80.

The number 3 after INPUT is a specification that tells iptables to place the rule 3rd in the list. Because we didn't specify any protocol this rule will block all protocols from the specified ip range.

Saving the rules for reboot

Each distribution of Linux has a different method for saving and restoring the iptables rules for reboot. In my opinion Red Hat variants have the best default.

Red Hat

In the Red Hat distributions you can save the current live iptables rules by using the init script. This saves the rules into the file `/etc/sysconfig/iptables`. This file is later read by the init script during reboot of the server or a simple restart given to the init script.

Saving your active rules

```
# /etc/init.d/iptables save
```

Loading ip_conntrack on boot

In order to load the ip_conntrack_ftp module on boot you will need to edit the `/etc/sysconfig/iptables-config` file. This file is used by Red Hat's init script to load any related modules on boot.

```
# vi /etc/sysconfig/iptables-config
```

Modify to add the following

```
IPTABLES_MODULES=ip_conntrack_netbios_ns ip_conntrack ip_conntrack_ftp
```

Ubuntu/Debian

Debain based distributions which includes Ubuntu, do not offer such init scripts by default. There is however a package called `iptables-persistent` that provides an init script for Ubuntu/Debian variants with similar functionality as the Red Hat version.

Installing iptables-persistent

You can install this package with apt-get

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

Saving your active rules

Once iptables-persistent is installed it will automatically save your current configuration into `/etc/iptables/rules.v4`. Any future modifications however will not be automatically saved, from then on you must save the rules similar to the Red Hat version.

```
# /etc/init.d/iptables-persistent save
 * Saving rules...
 * IPv4...
 * IPv6...
 ...done.

# cat /etc/iptables/rules.v4
# Generated by iptables-save v1.4.12 on Mon Sep 17 05:56:17 2012
*filter
:INPUT DROP [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [42:4296]
-A INPUT -i lo -j ACCEPT
-A INPUT -s 192.168.122.1/32 -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT -s 192.168.123.0/24 -j DROP
-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 21 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 20 -j ACCEPT
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p tcp -m tcp --dport 53 -j ACCEPT
-A INPUT -p udp -m udp --dport 53 -j ACCEPT
COMMIT
# Completed on Mon Sep 17 05:56:17 2012
```

Making sure iptables-persistent is started on boot

Once your rules are saved, iptables-persistent will start them on boot. To verify that the script is added properly simply check that it exists in /etc/rc2.d/.

```
# runlevel
N 2

# ls -la /etc/rc2.d/ | grep iptables
lrwxrwxrwx 1 root root 29 Sep 16 21:44 S37iptables-persistent -> ../init.d/iptables-pers
```

Loading ip_conntrack on boot

For Debian the iptables-persistent package does not include a iptables-config file. You could add this module into the init script itself to ensure it is loaded before iptables starts.

```
# vi /etc/init.d/iptables-persistent
```

Go to line #25 and add:

```
/sbin/modprobe -q ip_conntrack_ftp
```

About Benjamin

Benjamin is a Infrastructure and Software Engineer. On this blog he writes about Linux, Docker, Programming as well as other Systems topics.

Learn more about Linux

If you liked this article, check out Benjamin's book: Red Hat Enterprise Linux Troubleshooting Guide. Where you can learn a lot more about troubleshooting Linux systems. This book is filled with tips and techniques he has learned over years of managing mission critical systems.

Paperback Kindle

Related

- Advanced Linux System Statistics and Diagnostics with SystemTap
- Adding simulated network latency to your Linux server
- ACL: Using Access Control Lists on Linux
- When it's Ok and Not Ok to use rc.local
- Cheat Sheet: 21 useful find commands