Configuring firewalls for Linux (kernel 2.4.x) using netfilter/iptables

Mugdha Vairagade (vmugdha@indiatimes.com)
Independent Developer
September 2002

The *netfilter/iptables* is the IP packet filtering system that is integrated with the latest 2.4.x versions of the Linux kernel. This system facilitates greater control over IP packet filtering and firewall configuration on Linux systems, be they systems connected to the Internet or a LAN, servers, or proxy servers interfacing between a LAN and the Internet. Mugdha Vairagade provides an introduction to the netfilter/iptables system, how it works, its advantages, installing and configuring, and how to use it to configure firewalls on Linux systems to filter IP packets.

Note: Minimum intermediate level knowledge of Linux OS and experience of configuring Linux kernels will be helpful in understanding this article.

For this article, we are using iptables userspace tool version 1.2.6a and kernel version 2.4.9.

Linux security and netfilter/iptables

Linux has become extremely popular in the IT industry because of its robustness, reliability, flexibility, and seemingly unlimited scope for customization. Linux has many inbuilt capabilities that let the developer customize its tools, behavior, and appearance according to his needs without requiring expensive third-party tools. One such inbuilt capability is firewall configuration for Linux systems on a network, be they systems connected to the Internet or a LAN, servers, or proxy servers interfacing between a LAN and the Internet. This capability can be put to use with the help of the *netfilter/iptables IP packet filtering system* that comes integrated in versions 2.4.x of Linux kernels.

The netfilter/iptables IP packet filtering system is the latest among Linux packet filtering solutions like *ipfwadm* and *ipchains* and is also the first one to be integrated into the Linux kernel. The netfilter/iptables system is ideal for Linux system administrators, network administrators, and home users who want to configure firewalls according to their specific needs, save money on firewall solutions, and have total control over IP packet filtering.

Understanding firewall configuration and packet filtering

For a Linux system connected to a network, a firewall is the essential defense mechanism that allows only legitimate network traffic in and out of the system and disallows everything else. To determine whether the network traffic is legitimate or not, a firewall relies on a set of *rules* it contains that are predefined by a network or system administrator. These rules tell the firewall whether to consider as legitimate and what to do with the network traffic coming from a certain source, going to a certain destination, or having a certain protocol type. The term "configuring the firewall" refers to adding, modifying, and removing these rules. I will discuss these *rules* in detail a little later.

Network traffic is made up of IP packets or simply *packets* -- small chunks of data traveling in streams from a source system to a destination system. These packets have *headers*, i.e. bits of data prefixed to every packet that contain information about the packet's source, destination, and protocol types. Based on a set of rules, a firewall checks these headers to determine which packet to accept and which packet to reject. This process is known as *packet filtering*.

Why do we want to configure our own firewalls?

The need for configuring a firewall according to specific needs arises from various factors and reasons. Probably the most important reason is security.

Related content:

Subscribe to the developerWorks newsletter
More dW Security resources

Also in the Linux zone:

Tutorials
Tools and products
Code and components
Articles

Administrators might want their firewalls to stop unauthorized sources from accessing their Linux systems by using telnet, for example. They might also want to restrict the network traffic in and out of their system, so that only the traffic from trusted sources can enter their system and only authorized traffic can go out. A home user might want to configure a firewall to a lower security level by allowing all outgoing packets to pass through.

Another reason behind this could be to free up the bandwidth by blocking unnecessary traffic coming from sources like advertisement sites.

Thus firewall configuration can be tailored to suit any specific need and any security level requirement. This is where the netfilter/iptables system can help.

How does the netfilter/iptables system work?
The netfilter/iptables IP packet filtering system is a powerful tool that is used to add, edit and remove the rules that a firewall follows and consists of while making packet filtering decisions. These rules are stored in special-purpose packet filtering tables integrated in the Linux kernel. Inside the packet filtering tables the rules are grouped together in what are known as *chains*. I will discuss the rules in detail shortly as well as how to build these rules and group them in chains.

The netfilter/iptables IP packet filtering system, although referred to as a single entity, is actually made up of two components: *netfilter* and *iptables*.

The netfilter component, also known as *kernelspace*, is the part of the kernel that consists of packet filtering tables containing sets of rules that are used by the kernel to control the process of packet filtering.

The iptables component is a tool, also known as *userspace*, which facilitates inserting, modifying and removing rules in the packet filtering tables. You need to download this tool from netfilter.org and install it to use it, unless you are using Red Hat Linux 7.1 or higher.

By using userspace, you can build your own customized rules that are saved in packet filtering tables in kernelspace. These rules have *targets* that tell the kernel what to do with packets coming from certain sources, heading for certain destinations or have certain protocol types. If a packet matches a rule, the packet can be allowed to pass through using target `ACCEPT`. A packet can also be blocked and killed using target `DROP` or `REJECT`. There are many more targets available for other actions that can be performed on packets.

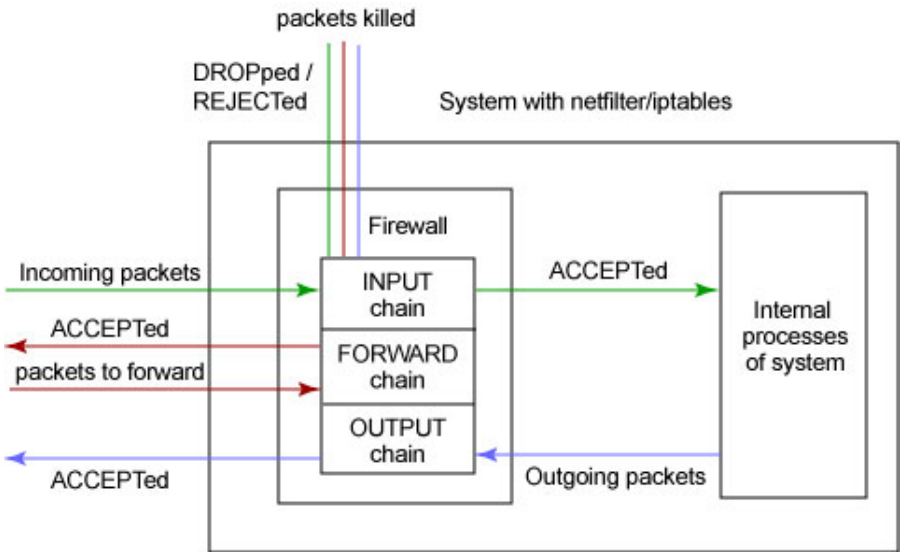
The rules are grouped in chains, according to the types of packets they deal with. Rules dealing with incoming packets are added to the `INPUT` chain. Rules dealing with outgoing packets are added to the `OUTPUT` chain. And rules dealing with packets being forwarded are added to the `FORWARD` chain. These three chains are the main chains built-in by default inside basic packet-filtering tables. There are many other chain types available like `PREROUTING` and `POSTROUTING` and there is also provision for user-defined chains. Each chain can have a *policy* that defines "a default target", i.e. a default action to perform, if a packet doesn't match any rule in that chain.

After the rules are built and chains are in place, the real work of packet filtering starts. Here is where the kernelspace takes over from userspace. When a packet reaches the firewall, the kernel first examines the header information of the packet, particularly the destination of the packet. This process is known as *routing*.

If the packet originated from outside and is destined for the system and the firewall is on, the kernel passes it on to the `INPUT` chain of the kernelspace packet filtering table. If the packet originated from inside the system or another source on an internal network the system is connected to and is destined for another outside system, the packet is passed on to the `OUTPUT` chain. Similarly, packets originating from outside systems and destined for outside systems are passed on to the `FORWARD` chain.

Next the packet's header information is compared with each rule in the chain it is passed on to, unless it perfectly matches a rule. If a packet matches a rule, the kernel performs the action specified by the target of that rule on the packet. But if the packet doesn't match a rule, then it is compared to the next rule in the chain. Finally, if the packet doesn't match to any rule in the chain, then the kernel consults the policy of that chain to decide what to do with the packet. Ideally the policy should tell the kernel to `DROP` that packet. Figure 1 graphically illustrates this packet filtering process.

Figure 1. Packet filtering process



Installing the netfilter/iptables system
Since the netfilter component of netfilter/iptables comes integrated with the kernel 2.4.x, you only need to download and install the iptables userspace tool.

Requirements
These are the requirements for installing the netfilter/iptables system:

- **Hardware:** To use netfilter/iptables, you need to have a system running a Linux OS that is connected to the Internet, a LAN or a WAN.
- **Software:** Any version of the Linux OS with kernel 2.4 or higher. The latest version of the kernel can be downloaded from http://www.kernel.org. You will also need to download the iptables userspace tool from http://www.netfilter.org, since this tool is not part of the kernel. But this is not necessary for RedHat Linux version 7.1 or higher, since this tool comes bundled into the standard install of this Linux version.
- **User:** You need to have at least an intermediate level of knowledge of the Linux OS and have experience of configuring Linux kernels.

Pre-install preparations
Before you start installing the iptables userspace tool, you'll need to make certain modifications to your system. First of all you'll need to configure your kernel's options using the command `make config`. During configuration you must turn on the options `CONFIG_NETFILTER` and `CONFIG_IP_NF_IPTABLES` by setting them to `Y`, since it is necessary to make netfilter/iptables work. Other options you might want to turn on are as follows:

- **`CONFIG_PACKET`:** This option is useful, if you want to allow applications and programs to work directly to certain network devices.
- **`CONFIG_IP_NF_MATCH_STATE`:** This option is very important and useful, if you want to configure *stateful* firewalls. Such firewalls can remember previous decisions taken regarding packet filtering and make new decisions based on them. I will further discuss this aspect in the section Advantages of the netfilter/iptables system.
- **`CONFIG_IP_NF_FILTER`:** This option provides a basic packet filtering framework. Turning this option on will add a basic filter table to kernelspace with built-in `INPUT`, `FORWARD`, and `OUTPUT` chains.
- **`CONFIG_IP_NF_TARGET_REJECT`:** This option allows you to specify that an ICMP error message should be sent in reply to incoming packets that are `DROP`ped, instead of just killing them.

Now you are ready to install the userspace tool.

Installing the userspace tool
After downloading the source for the iptables userspace tool (it will be something like iptables-1.2.6a.tar.bz2), you can start installation. You'll need to log in as `root` to perform installation. Listing 1 gives an example that specifies the commands necessary to install the tool, their required order and their explanation.

Listing 1. Example of userspace tool installation

```
First, unpack the tool package into a directory:

# bzip2 -d iptables-1.2.6a.tar.bz2
# tar -xvf  iptables-1.2.6a.tar

This will unpack the tool source into a directory named iptables-1.2.6a.
Now change to the iptables-1.2.6a directory:

# cd iptables-1.2.6a

The INSTALL file in this directory contains a lot of useful information
on compiling and installing this tool.
Now compile the userspace tool using the following command:

# make KERNEL_DIR=/usr/src/linux/

Here the KERNEL_DIR=/usr/src/linux/ specifies the path to the kernel's
directory. If the directory of kernel happens to be different on some
systems, the appropriate directory path should be substituted for
/usr/src/linux.
Now install the source binaries using the following command:

# make install KERNEL_DIR=/usr/src/linux/

Now the installation is complete.
```

Note: In case you have RedHat Linux version 7.1 or higher, you don't need to go through the two previous steps specified here. As we already know, the iptables userspace tool is available with the standard install of this Linux distribution. But this tool is turned off by default. To make this tool run, you'll need to follow these steps (Listing 2):

Listing 2. Example of setting up the userspace tool on a RedHat 7.1 system

```
First you'll have to turn off the old ipchains module (predecessor of
iptables) available in this OS package.
This can be done using the following command:

# chkconfig --level 0123456 ipchains off

Next, to completely stop the ipchains module from running, so that it
doesn't conflict with the iptables tool, you will have to stop the ipchains
service using the following command:

# service ipchains stop

Now if you don't want to keep this old ipchains module on your system,
uninstall it using the following command:

# rpm -e ipchains

Now you can turn on the iptables userspace tool with the following command:

# chkconfig --level 235 iptables on

Finally, you'll have to activate the iptables service to make the userspace
tool work by using this command:

# service iptables start

Now the userspace tool is ready to work on a RedHat 7.1 or higher system.
```

As you now have everything in place and the netfilter/iptables system should be running, you will need to build rules and chains to filter packets.

Building rules and chains
Rules govern packet filtering by providing the firewall with instructions on what to do with packets coming from a certain source, going to a certain destination or having a specific protocol type. These rules are built and added to chains inside special packet filtering tables of the kernelspace using a special command `iptables`, provided by the netfilter/iptables system. The general syntax of the command to add/remove/edit a rule is as follows:

```
$ iptables [-t table] command [match] [target]
```

Table
The `[-t table]` option allows you to use any table other than the standard table. A table is a packet filtering table that contains rules and chains dealing with specific kinds of packets only. There are three table options available: `filter`, `nat` and `mangle`. This option is not necessary and, if not specified, the `filter` is the default table to be used.

The filter table is used for general packet filtering and consists of `INPUT`, `OUTPUT`, and `FORWARD` chains. The nat table is used for packets to be forwarded and consists of `PREROUTING`, `OUTPUT`, and `POSTROUTING` chains. The mangle table is used if there are any changes to be made in packets and their headers. This table contains rules to mark packets for advanced routing and it consists of `PREROUTING` and `OUTPUT` chains.

Note: The `PREROUTING` chain consists of rules that specify altering of packets as soon as they get in to the firewall, while the `POSTROUTING` chain consists of rules that specify altering of packets just as they are about to leave the firewall.

Command
The compulsory command section of the command above is the most important part of the `iptables` command. It tells the `iptables` command what to do, for example, to insert a rule, to add a rule to the end of the chain, or to delete a rule. The following are the most often-used commands:

- **`-A` or `--append`:** This command appends a rule to the end of a chain.
  Example:

  ```
  $ iptables -A INPUT -s 205.168.0.1 -j ACCEPT
  ```

  This example command appends a rule at the end of the `INPUT` chain that specifies packets coming from source address 205.168.0.1 to be `ACCEPT`ed.

- **`-D` or `--delete`:** This command deletes a rule from the chain, either by specifying the rule to match with `-D` or by specifying the rule's position number in the chain. The following examples shows both ways.
  Examples:

  ```
  $ iptables -D INPUT --dport 80 -j DROP
  $ iptables -D OUTPUT 3
  ```

  The first command deletes a rule from the `INPUT` chain that specifies packets destined for port 80 to be `DROP`ped. The second command simply deletes rule number 3 from the `OUTPUT` chain.

- **`-P` or `--policy`:** This command sets a default target, i.e. policy, for a chain. All packets that don't match any rule in the chain will then be forced to use the policy of the chain.
  Example:

  ```
  $ iptables -P INPUT DROP
  ```

  This command specifies the default target of the `INPUT` chain to be `DROP`. That means all the packets not matching any rule in the `INPUT` chain will be dropped.

- **`-N` or `--new-chain`:** This creates a new chain with the name specified in the command.
  Example:

  ```
  $ iptables -N allowed-chain
  ```

- **`-F` or `--flush`:** This command deletes all rules inside a chain if a chain name is specified or all rules in all chains if no chain name is specified. Used for quick cleanup.
  Examples:

  ```
  $ iptables -F FORWARD
  $ iptables -F
  ```

- **`-L` or `--list`:** Lists all rules in the specified chain.
  Example:

  ```
  $ iptables -L allowed-chain
  ```

Match
The optional match section of the iptables command specifies the characteristics that a packet should have to match the rule, such as source and destination address, protocol, etc. The matches are divided in two major categories: *generic matches* and *protocol-specific matches*. Here I will explore generic matches that can be used for packets having any protocols. The following are important and often-used generic matches with their examples and explanations:

- **`-p` or `--protocol`:** This generic protocol match is used to check for certain protocols. Examples of protocols are `TCP`, `UDP`, `ICMP`, comma-delimited list of any combination of these three protocols and `ALL` (for all protocols). `ALL` is the default match. This option can be inverted by using the `!` sign.
  Examples:

  ```
  $ iptables -A INPUT -p TCP, UDP
  $ iptables -A INPUT -p ! ICMP
  ```

  In these examples, both commands perform the same task -- they specify that all `TCP` and `UDP` packets will match this rule. By specifying `! ICMP`, we mean to allow all other protocols (`TCP` and `UDP`, in this case) except `ICMP`.

- **`-s` or `--source`:** This source match is used to match packets based on their source IP address. This match also allows IP address range matching and it can be inverted using the `!` sign. The default source match matches all IP addresses.
  Examples:

  ```
  $ iptables -A OUTPUT -s 192.168.1.1
  $ iptables -A OUTPUT -s 192.168.0.0/24
  $ iptables -A OUTPUT -s ! 203.16.1.89
  ```

  The second command specifies that this rule matches all packets coming from IP addresses ranging 192.168.0.0 to 192.168.0.24. The third command specifies that this rule will match any packets not from source address 203.16.1.89.

- **`-d` or `--destination`:** This destination match is used to match packets based on their destination IP address. This match also allows IP address range matching and it can be inverted using the `!` sign.
  Examples:

  ```
  $ iptables -A INPUT -d 192.168.1.1
  $ iptables -A INPUT -d 192.168.0.0/24
  $ iptables -A OUTPUT -d ! 203.16.1.89
  ```

Target
We already know that targets are the actions specified by rules to be performed on packets that match those rules. There are many target options available along with allowances for user-defined targets. The following are often-used targets, their examples and explanations:

- **`ACCEPT`:** When a packet is perfectly matched with a rule that has an `ACCEPT` target, it is accepted (allowed to go wherever it is destined to) and it will stop traversing the chain (though that packet may traverse through another chain in another table and may be dropped there). This target is specified as `-j ACCEPT`.

- **`DROP`:** A packet that matches a rule perfectly that has a `DROP` target will be blocked and no further processing will be done on it. This target is specified as `-j DROP`.

- **REJECT:** This target works the same way as the DROP target, except that it is better than DROP. Unlike DROP, REJECT doesn't leave dead sockets around on the server and client. Also, REJECT sends back an error message to the sender of the packet. This target is specified as `-j REJECT`.
  Example:

  ```
  $ iptables -A FORWARD -p TCP --dport 22 -j REJECT
  ```

- **RETURN:** The RETURN target set in a rule makes the packet matching that rule stop traversing through the chain containing the rule. If the chain is a main chain like INPUT, the packet will be handled using the default policy of that chain. It is specified as `-jump RETURN`. Example:

  ```
  $ iptables -A FORWARD -d 203.16.1.89 -jump RETURN
  ```

There are many other targets like LOG, REDIRECT, MARK, MIRROR, MASQUERADE etc. that are used for advanced rule building.

Saving rules
Now you've learned how to build basic rules and chains and how to add or remove them from the packet filtering tables. But you should remember that rules built the way described above are saved into the kernel and are lost when the system is rebooted. So if you have an error-free and efficient set of rules added to the packet filtering tables, to use the rules again after a reboot, you have to save the rule set in a file. This can be done using the **iptables-save** command:

```
$ iptables-save > iptables-script
```

Now all rules in the packet filtering tables are saved in the file iptables-script. Whenever you boot your system again, you can restore the rule set into the packet filtering tables from this script file using the **iptables-restore** command as shown below:

```
$ iptables-restore iptables-script
```

And if you prefer to restore the rule set automatically every time you boot your system, then you can put the command specified above into any of your initialization shell-scripts.

Advantages of the netfilter/iptables system
The biggest advantage of netfilter/iptables is that it can configure stateful firewalls, an important facility that its predecessors like ipfwadm and ipchains were unable to provide. A stateful firewall is capable of assigning and remembering the state of connections made for sending or receiving packets. Firewall gets this information from the connection tracking state of the packets. This information about states is used by a firewall when it is making new packet filtering decisions to increase its efficiency and speed. There are 4 valid states, namely ESTABLISHED, INVALID, NEW and RELATED.

The state ESTABLISHED indicates that the packet is part of an already established connection that has been used to both send and receive packets and is fully valid. INVALID state indicates that the packet is not associated with any known stream or connection and it may contain faulty data or headers. The state NEW means that the packet has or will start a new connection or that it is associated with a connection that has not been used to both send and receive packets. Finally, RELATED means that the packet is starting a new connection and it is associated with an already established connection.

Another important advantage of netfilter/iptables is that it gives users total control over firewall configuration and packet filtering. You can make your own rules that suit your specific needs to allow only the network traffic you want into your system.

Also, netfilter/iptables is free and thus ideal for the people who want an alternative to expensive firewall solutions to save money.

Conclusion
The latest Linux kernel 2.4.x has a built-in IP packet filtering facility in the form of the netfilter/iptables system, which makes configuring firewalls and packet filtering cheap and easy. The netfilter/iptables system gives its users total control over firewall configuration and packet filtering. It allows the creation of customized rules for firewalls that govern packet filtering. It also allows configuration of stateful firewalls.

Resources

- Get the latest information about the netfilter/iptables system and download the iptables userspace tool at the Netfilter Web

site.

- Visit Linux 2.4 Packet Filtering HOWTO Web pages for a quick reference on netfilter/iptables.
- Check out this detailed tutorial on netfilter/iptables (Iptables Tutorial 1.1.9).
- Get answers to any questions on netfilter/iptables at netfilter/iptables FAQ.

About the author
Mugdha Vairagade is a developer with work experience in various organizations. She has sound experience in wireless application development and specialization in component architecture. She has a special interest in Open Source projects and is associated with Linux Documentation Project, Forum Nokia (WAP developers' forum), and W3C. She also writes technical articles on Linux and XML technologies. You can contact Mugdha at vmugdha@indiatimes.com.