Technical Guide

Network control of virtual servers using iptables

Netfilter

May 2021

SAMSUNG SDS

Contents

1. Overview

In this document, we will cover how netfilter, the basis framework for iptables, works and how to use iptables.

What enables Linux firewall functions is not actually iptables but the framework already built into the Linux kernel called netfilter. Netfilter primarily refers to the packet filtering framework, which literally takes the role of filtering packets.

In other words, netfilter itself implements packet filtering without all rules but provides a tool that runs in a user space called iptables. When rules are added using this tool, packets enter the Linux kernel and netfilter begin to run. If there are pre-determined rules for passing through each set point, netfilter makes sure that the packet goes through these rules with a value, including drop or accept.

2. Hook points provided by netfilter

Netfilter provides 5 hook points. As packets pass through each point, the kernel module registered in each hook point is triggered.

Since multiple kernel modules can be registered and used in the same hook point, when the kernel module registers to catch and use the hook point, input the priority number and trigger the kernel module according to the priority.

1. NF_IP_PRE_ROUTING: A hook that occurs immediately after a packet from outside enters the network stack of the Linux kernel, which takes place before routing the packet.
2. NF_IP_LOCAL_IN: This hook occurs before sending the packet to the local process when the destination is itself after the packet is routed.
3. NF_IP_FORWARD: This is a hook that occurs when a packet is forwarded to another place when the destination is not itself after a packet is routed.
4. NF_IP_LOCAL_OUT: A hook that occurs before a packet leaves the process and passes through the network stack.
5. NF_IP_POST_ROUTING: This is a hook that occurs after a packet passes through the network stack and before being sent out.

The above 5 hook points are shown in the figure below.

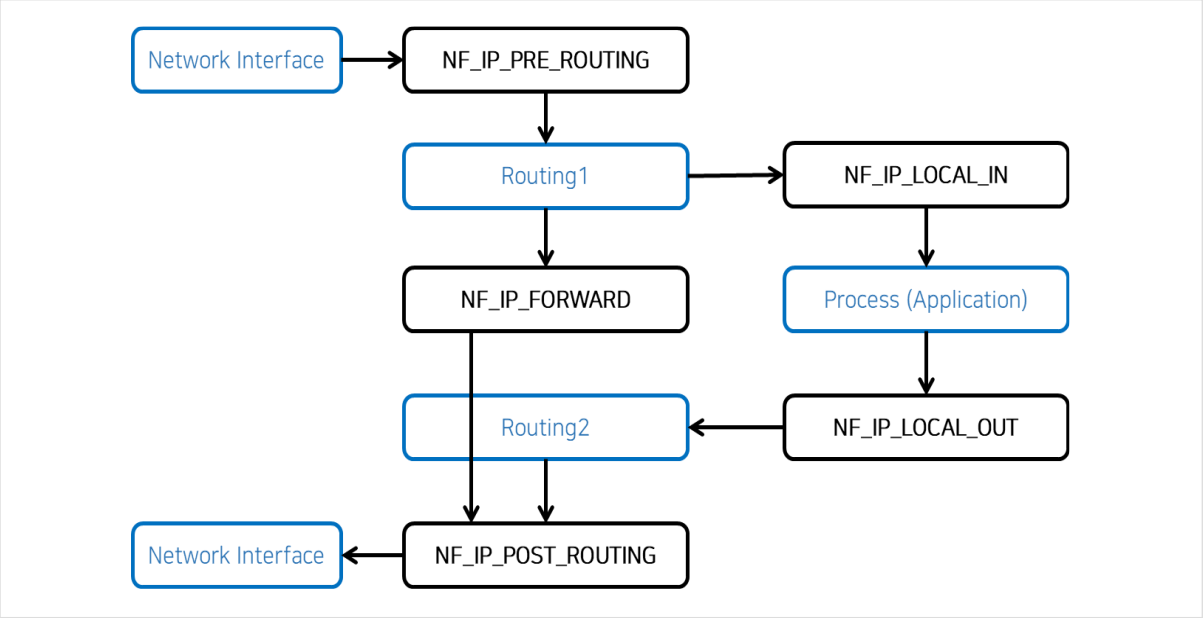

Figure 1. Packet path in netfilter

The figure above shows the packet flow in netfilter.

Routing1 represents the process of routing by distinguishing whether a packet received from a network interface card (NIC) is a packet that a local process should receive or a packet that another host should receive.
Routing2 represents the process of re-routing when a packet sent by the process is DNATed at NF_IP_LOCAL_OUT.

In summary, packet flows can be classified into the following three:

1. If the destination of the packet from outside is itself: NF_IP_PRE_ROUTING -> NF_IP_LOCAL_IN -> Process
2. If the destination of the packet from outside is not itself: NF_IP_PRE_ROUTING -> NF_IP_FORWARD -> NF_IP_POST_ROUTING -> Network Interface
3. In case of packet transmitted by a local process: NF_IP_LOCAL_OUT -> NF_IP_POST_ROUTING -> Network Interface

3. How iptables works

To understand the packet filtering mechanism by iptables, you need to understand the following four terms/structures:

- Tables, chains, rules, and targets

Tables are already built into the kernel. Each of the 5 tables has predefined chains, and rules are put in these chains, specifying a target. If a packet matches a rule, you can go to the target and check if it matches another rule, or you can drop or accept it.

Now, let's look at each structure a little more in detail.

4. Table

A total of 5 tables are provided as follows:

1. Filter Table: This is a table for filtering packets. Decide whether to deliver the packet to its original destination or drop it. Firewall function can be built through filter table.
2. Mangle Table: This table is used to change the IP header of the packet. For example, you can change the packet's Time to Live (TTL) or mark the packet so that other iptables tables or network tools can distinguish the packet.
3. NAT Table: This is a table for Network Address Translation (NAT). Change the source address or destination address of the packet.
4. Raw Table: Netfilter framework provides not only hooks but also the connection tracking function. The connection of the just arrived packet is traced based on the previously arrived packets. The raw table also sets certain packets to be excluded from connection tracking.
5. Security Table: This table decides how to handle packets in SELinux.

5. Chain

Each table has built-in chains, which are named after the netfilter hook.

1. PREROUTING: Triggered by NF_IP_PRE_ROUTING Hook.
2. INPUT: Triggered by NF_IP_LOCAL_IN Hook.
3. FORWARD: Triggered by NF_IP_FORWARD Hook.
4. OUTPUT: Triggered by NF_IP_LOCAL_OUT Hook.
5. POSTROUTING: Triggered by NF_IP_POST_ROUTING Hook.

6. Packet flows

In iptables, the flows of the packet are as follows:

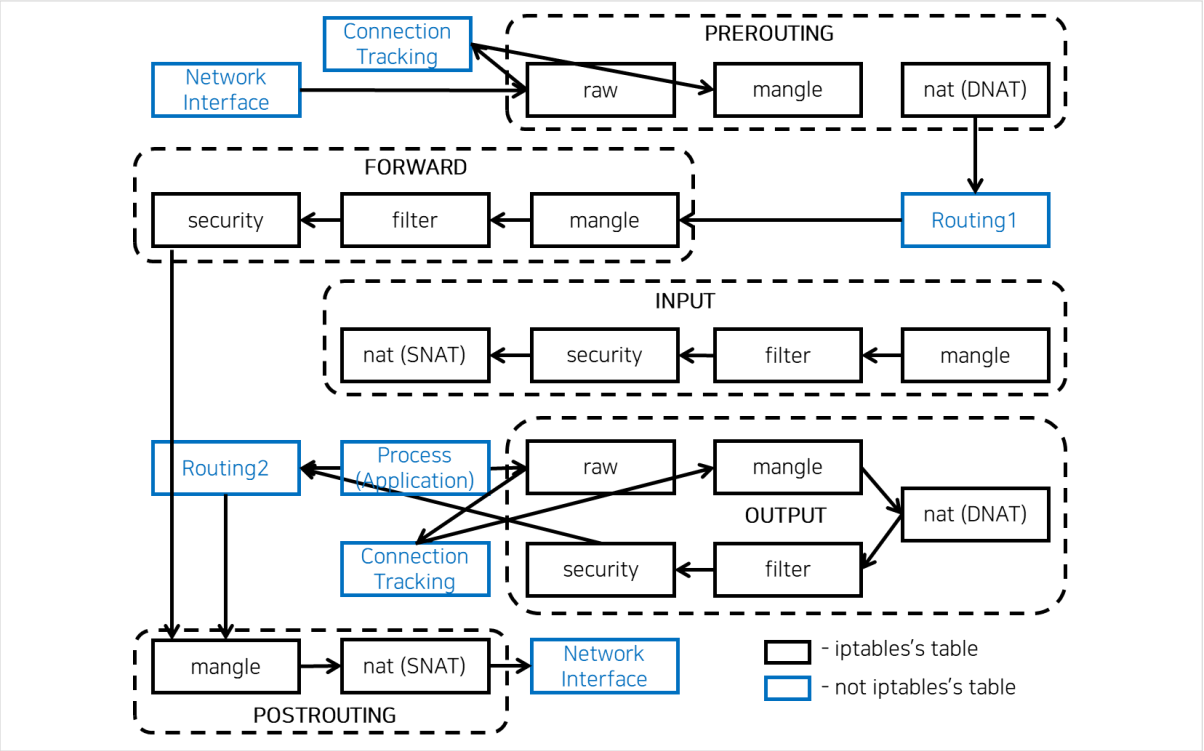

Figure 2. Packet flows in iptables

As shown in the picture above, packets are processed while passing through several tables in each hook.

Of course, there are multiple chains in each table and rules are registered in each chain. If a packet matches a certain rule and its fate is determined (e.g. accept or drop), it does not proceed to the next chain or table.

7. Rules of iptables

A rule is registered in a specific chain in a specific table, and when each chain is called, the action is decided after checking if the packet matches the registered rule.

The rule matching criteria can be a combination of protocol type, source address/port, destination address/port, input/output interface, etc.

8. Target

A target is an action triggered when a packet matches a registered rule.

Targets use the -j or --jump option and can be a user-defined chain, one of the built-in targets, or a target extension.

If it is a custom chain, it jumps to that specified custom chain and continues trying to match the rule.

Built-in targets are ACCEPT, DROP, QUEUE, and RETURN. Details are as follows:

1. ACCEPT - Iptables accepts packets.
2. DROP - Iptables drop packets. To anyone trying to connect to the system, it appears that the system does not exist.
3. QUEUE - Iptables forwards packets to userspace.
4. RETURN - Iptables stops executing the next set of rules in the current chain for this packet. Control is returned to the call chain.

Target extensions include REJECT and LOG.

1. REJECT: Iptables "rejects" packets. It sends a “connection reset” packet for TCP, and a “destination host unreachable” packet for UDP or ICMP. (It is different from the abovementioned built-in target DROP.)

If the target is a built-in target, the packet's fate is determined immediately, so packet processing in the current table does not proceed any further.

If the target is a user-defined chain and the packet's fate is not determined by the chain, it returns to the original chain and tries to match to the rest of the rules.

9. Default policy chain behavior

Given that rules are added to each of the chains mentioned above, you need to decide what to do if the packet does not match to any rules.

Options	Description
-N (--new-chain)	Add a new chain
-F(--flush)	Delete all rules in the chain
-P (--policy)	Change a policy
-R (--replace)	Replace a rule
-A (--append)	Append a new rule
-I (--insert)	Insert a new rule

-X(--delete-chain)	Delete an empty chain
-L (--list)	Check the rule defined in the table
-D (--delete)	Delete Rules

To view the current policy, do the following:

```
$ sudo iptables -L | grep policy
Chain INPUT (policy ACCEPT)
Chain FORWARD (policy ACCEPT)
Chain OUTPUT (policy ACCEPT)
```

If a table is not specified with the -t option in the iptables command, the default value is filter table. If you want to know the default policy for chains of other tables (e.g. NAT), run the following:

```
$ sudo iptables –t nat –L | grep policy
```

If you want to change the default policy for each chain of the table, use the -p option as shown below.

The following is the default value in case of accepting all packets.

```
$ sudo iptables -P INPUT ACCEPT
$ sudo iptables -P OUTPUT ACCEPT
$ sudo iptables -P FORWARD ACCEPT
```

The following is the default value in case of dropping all packets.

```
$ sudo iptables -P INPUT DROP
$ sudo iptables -P OUTPUT DROP
$ sudo iptables -P FORWARD DROP
```

To check the current rule, execute the following command.

```
$ sudo iptables –L
$ sudo iptables -S
```

To add a rule to the end of the chain, execute the following command.

```
$ sudo iptables -A
```

To insert a rule at a specific location in the chain, run the following command:

```
$ sudo iptables -I
```

To replace a rule, execute the following command.

```
$ sudo iptables -R
```

To delete a rule, execute the following command.

```
$ sudo iptables -D
```

10. Examples of using iptables

Let's look at frequently used options as follows:

Options	Description
-s (--source)	IP address setting for source
-d (--destination)	IP address setting for destination
-j (--jump)	Jump as a target such as ACCEPT, DROP, etc.
-p (--protocol)	Set protocols such as TCP, UDP, etc.
-t (--table)	Specify the five tables mentioned above
--sport	Source port
--dport	Destination port
--m(--match)	When using a specific module
--state	Designate connection status such as FIN_WAIT
--string	Match a given string
-!	Exclude specific IP address
-i (--in-interface)	Designate incoming interface
-o (--out-interface)	Specify outgoing interface

Here are some examples of iptables usage for several situations.

10.1 Allowing Incoming Traffic on Specific Ports

Execute the following command to allow TCP connection to the default ssh port 22.

```
$ sudo iptables -A INPUT -p tcp --dport ssh -j ACCEPT
or
$ sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

If you view the result after adding a rule as above, it will look similar to the one below:

```
$ sudo iptables -L
Chain INPUT
(policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp --  anywhere             anywhere            tcp dpt:ssh
```

Let's allow web-related traffic.

The above command can also be performed as below:

```
$ sudo iptables -A INPUT -p tcp -m multiport --dports http,https -j ACCEPT
```

If you view the result after adding a rule, it will look similar to the one below:

```
$ sudo iptables -L
Chain INPUT
(policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp --  anywhere             anywhere            tcp dpt:ssh
ACCEPT     tcp --  anywhere             anywhere            tcp dpt:http
ACCEPT     tcp --  anywhere             anywhere            tcp dpt:https
```

10.2 Blocking Incoming Traffic

As above, if you want to allow only ssh, http, https and block all the other incoming traffic, you can execute the command as follows:

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

Chain INPUT
(policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp --  anywhere             anywhere            tcp dpt:ssh
ACCEPT     tcp --  anywhere             anywhere            tcp dpt:http
ACCEPT     tcp --  anywhere             anywhere            tcp dpt:https
DROP       all --  anywhere             anywhere
```

Since the rule is added with the -A INPUT option, the rule is added to the end of the INPUT chain. When a packet comes in, rule matching is attempted from the top. When a match is made, the action is applied immediately and does not proceed to the next rule.

In other words, ssh can still be connected.

10.3 Blocking traffic from specific IP address

Enter the IP address for the -s option.

```
$ sudo iptables -A INPUT -s 59.45.175.62 -j REJECT
or
$ sudo iptables -A INPUT -s 59.45.175.62 -j DROP
```

Classless Inter-Domain Routing (CIDR) representation is also possible in the source IP address. It means that if you want to block all IP addresses in the 59.45.17.0 - 59.45.17.255 range, you can run the command like the following:

```
$ sudo iptables -A INPUT -s 59.45.175.0/24 -j REJECT
```

If you want to block outgoing traffic to a specific IP address, you need to register the rule in the OUTPUT chain, not the INPUT chain, and use the -d option as it is the destination IP address.

```
$ sudo iptables -A OUTPUT -d 31.13.78.35 -j DROP
```

10.4 Retrieving rules by line number

If you enter the --line-numbers option when querying rules, the line number before the line is also shown when displaying each chain and rule. This can be helpful sometimes. (Especially when deleting a rule.)

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

The result is the line number for each chain as shown below:

```
Chain INPUT (policy ACCEPT)
num target prot opt source destination

1 DROP all -- 59.45.175.0/24 anywhere
2 DROP all -- 221.194.47.0/24 anywhere
3 DROP all -- 91.197.232.104/29 anywhere

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

Chain OUTPUT (policy ACCEPT)
num target prot opt source destination
1 DROP all -- anywhere 31.13.78.0/24
```

The -n option can also be used if this process is unnecessary because iptables performs reverse DNS lookup for the IPs registered in the rule or if you want to print more quickly.

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

10.5 Deleting rules by line number

If you want to delete the added rule, use the -d option as follows, just by replacing -a with -d in the command line created from the -a option.

```
$ sudo iptables -D INPUT -s 221.194.47.0/24 -j REJECT
```

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

Alternatively, you can use iptables -l --line-numbers to display the line number for the rule registered in each chain, and then delete it with the command below.

One thing to note when deleting with line numbers is that the line numbers of the rules can change.

For example, when deleting two rules with line numbers 9 and 12, if you delete 9 first, rule 12 becomes 11. You can check the updated line number by executing the iptables -L --line-numbers command again, but checking them every time can be a burden. In

```
$ sudo iptables -D INPUT 12
$ sudo iptables -D INPUT 9
```

this regard, it is recommended to delete the highest line number first (i.e., the rule at the bottom) as follows:

If you want to delete all rules in a specific chain of a specific table, execute the command as shown below:

```
$ sudo iptables -F INPUT
```

In the above command, if a specific table is not defined with the -t option, filter table will be the default, deleting all rules.

10.6 Putting a rule in a specific location

If you want to put a rule on the first line of the INPUT chain, run the command like this:

```
$ sudo iptables -I INPUT 1 -s 59.45.175.10 -j ACCEPT
```

10.7 Changing the rules

If the source IP address in the rule put in the first line of the INPUT chain is accidentally entered as 59.45.175.12 and you want to change it to 59.45.17.10, execute the command as follows:

```
$ sudo iptables -I INPUT 1 -s 59.45.175.10 -j ACCEPT
```

10.8 Protocol-based matching

If you want to block all incoming TCP traffic, run the command like this:

```
$ sudo iptables -A INPUT -p tcp -j DROP
```

If you want to block ssh access with a source IP address range, you can execute the command as below:

```
$ sudo iptables -A INPUT -p tcp -m tcp --dport 22 -s 59.45.175.0/24 -j DROP
```

If you want to block access to ssh and vnc with a single rule, you can execute the command as follows using the -m multiport option.

```
$ sudo iptables -A INPUT -p tcp -m tcp --dport 22 -s 59.45.175.0/24 -j DROP
```

If you want to block ICMP address mask requests (type 17) for the ICMP protocol, you can execute the command as follows:

```
$ sudo iptables -A INPUT -p icmp --icmp-type 17 -j DROP
```

10.9 Selecting a specific network interface

If you want to allow packets coming through the Loopback Interface (lo in general), use the -i option and execute the command as shown below. The -i option refers to the input interface.

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

For OUTPUT chains, you must use the -o option. The -o option indicates the output interface.

```
$ sudo iptables -A OUTPUT -o wlan0 -d 121.18.238.0/29 -j DROP
```

10.10 User-defined chain

Suppose you entered the rules below for some circumstances.

```
$ sudo iptables -A INPUT -p tcp -m tcp --dport 22 -s 18.130.0.0/16 -j ACCEPT
$ suoo iptables -A INPUT -p tcp -m tcp --dport 22 -s 18.11.0.0/16 -j ACCEPT
$ sudo iptables -A INPUT -p tcp -m tcp --dport 22 -j DROP
```

Although the above rules work fine, they can be inefficient and not look neat. By creating a custom chain for these rules, you can organize them as follows.
For this, you can use the -n option to create a new chain as shown below:

```
$ sudo iptables -N ssh-rules
```

Then, you can add rules to the newly created ssh-rules chain as shown below:

```
$ sudo iptables -A ssh-rules -s 18.130.0.0/16 -j ACCEPT
$ sudo iptables -A ssh-rules -s 18.11.0.0/16 -j ACCEPT
$ sudo iptables -A ssh-rules -j DROP
```

First, create a new chain and the rules that go into that chain. However, if it stops here, no packets can enter this chain and check for a match. This is because packets basically

```
$ sudo iptables -A INPUT -p tcp --dport 22 -j ssh-rules
```

flow through built-in chains such as INPUT and OUPUT. Therefore, in the built-in chain INPUT, we need to jump to the chain we created when certain conditions are met.

If you want to clear the custom chain, run the command as below. Note that you must delete all rules referencing this chain before deleting the chain.

```
$ sudo iptables -X ssh-rules
```

10.11 iptables-save and iptables-restore

If you want to know all rules applied to iptables, you can run iptables -l command. You can also create a text file by redirecting the output of iptables-save command.

```
$ sudo iptables-save > iptables.rules
```

You can open the created text file with an editor such as vi, modify/add/delete rules, and then apply the content of the file again with the command below.

```
$ sudo iptables-restore < iptables.rules
```

10.12 Permanent rules registered in iptables

Rules registered using the iptables command are instantly applied during operation, but disappear after rebooting. Therefore, in order to apply them continuously regardless of reboot, you need to install the appropriate package as follows according to each Linux distribution.

RHEL family

```
$ sudo iptables-save > iptables.rules
```

Debian and Ubuntu family

```
$ sudo apt install iptables-persistent
```

If your distribution does not have a package that functions similar to the one above, you can create a service file yourself and load the iptables rule file in the beginning and save it before shutdown.

So far, we have looked at the concept of netfilter and various ways of using iptables. We hope this guide helps you set up complete network routing of virtual servers in a cloud environment.