Oracle Linux 8

Configuring the Firewall

F20786-12
October 2025

Oracle Linux 8 Configuring the Firewall,

F20786-12

Contents

Preface

Oracle Linux 8: Configuring the Firewall describes how to secure the network by using `firewalld` to implement rules that control traffic that flows to and from Oracle Linux 8 systems. The document also describes how to use the `nftables` framework to further control network access to these systems.

1

About Packet-Filtering Firewalls

A firewall can be configured to:

- Filter incoming and outgoing network packets based on packet header information,
- Redirect packets, such as with network address translation (NAT),
- Perform packet mirroring,
- Perform deep packet inspection,
- Accepted or rejected packets based on rules.

The Oracle Linux kernel uses the Netfilter feature to provide packet filtering functionality for IPv4, IPv6, inet, arp, bridge, and netdev.

Netfilter consists of the following components:

- A `netfilter` kernel component consisting of a set of tables in memory for the rules that the kernel uses to control network packet filtering.
- Utilities to create, maintain, and display the rules that `netfilter` stores. In Oracle Linux 8, the default firewall utility is the `firewall-cmd`, which is provided by the `firewalld` package.
- The `nftables` framework is the default network packet filtering framework used by `firewalld` in Oracle Linux . `nftables` functions as the backend for `firewalld` and integrates with `netfilter`. The `nftables` framework includes packet classification facilities, added convenience, and improved performance.

The `firewalld`-based firewall has the following advantages:

- The `firewalld-cmd` utility doesn't restart the firewall and disrupt established TCP connections.
- `firewalld` supports dynamic zones, which enable you to implement different sets of firewall rules for systems such as laptops that can connect to networks with different levels of trust. However, this feature isn't typically used on server systems.
- `firewalld` supports D-Bus for better integration with services that depend on firewall configuration.
- `firewalld` covers most basic use cases

For more complex scenarios consider creating and configuring `nftables` directly instead of using `firewalld`. For example, consider configuring `nftables` directly for scenarios such as:

- Where you need direct control over `netfilter`,
- Where you require high performance,
- When using complex rules,
- When dealing with specific or advanced networking requirements.

Disable the `firewalld` service before configuring and using `nftables` directly to avoid situations where each service might influence one another.

2

Configuring the Firewall with Firewalld

This chapter describes the concepts, tools, and methods for configuring the firewall with `firewalld` based tools. It also provides examples for displaying the firewall settings that enforce network security on a system.

firewalld Configuration Tools

You can configure the firewall by using one of the following tools:

- By using the `firewall-cmd` command and its several options.
- By using the Firewall Configuration GUI

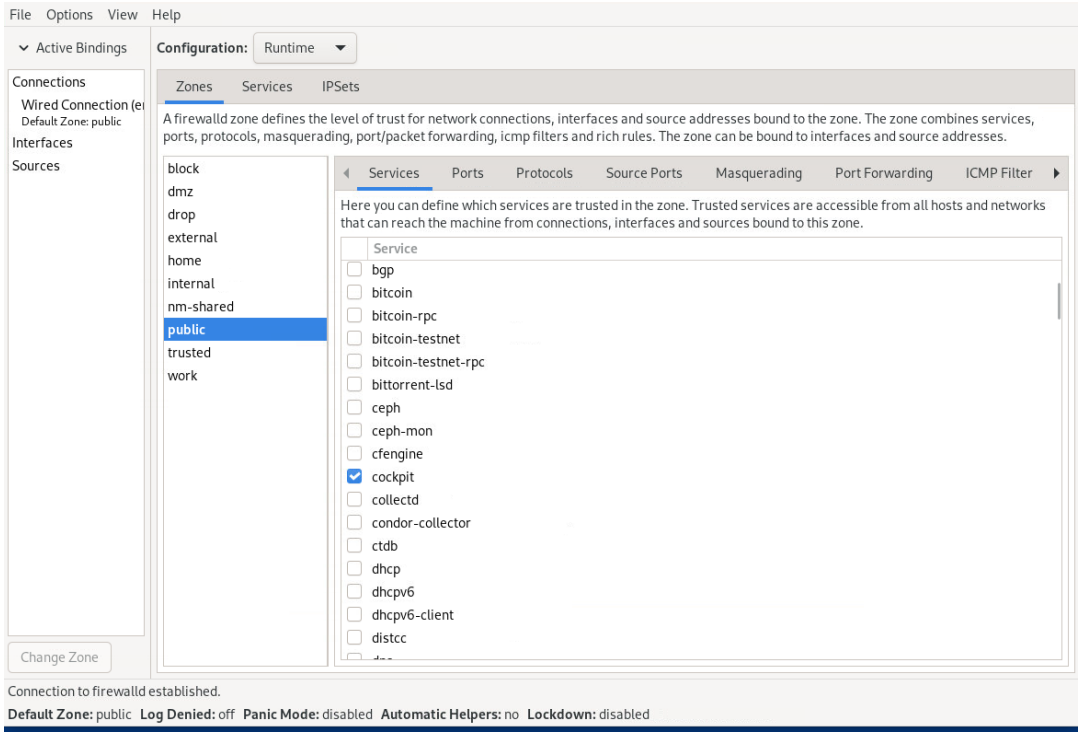
  To use this tool you must install the `firewall-config` package first, then start it by using the same command as the package name, for example:

  ```
  sudo dnf install firewall-config
  ```

  ```
  sudo firewall-config &
  ```

  The command opens the configuration tool, as shown in the following figure:

Figure 2-1 Firewall Configuration



- Cockpit is a browser-based configuration tool that you can also use to perform firewall configurations. See Oracle Linux: Using the Cockpit Web Console.

Controlling the firewalld Service

In Oracle Linux 8, the firewall service, `firewalld`, is enabled by default. The service is controlled by the `systemctl` command.

To start the service:

```
sudo systemctl unmask firewalld

sudo systemctl start firewalld
```

To ensure that the service starts automatically when the system starts, run the following command after starting the firewall:

```
sudo systemctl enable firewalld
```

To stop the firewall service and prevent it from automatically starting when the system starts, run the following command:

```
sudo systemctl stop firewalld

sudo systemctl disable firewalld
```

To prevent the firewall service from being started by other services or through the `firewalld` D-Bus interface, run the following command after disabling the firewall:

```
sudo systemctl mask firewalld
```

To display the current status of the firewall service:

```
sudo systemctl status firewalld

firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset:
enabled)
   Active: active (running) since Wed 2022-04-20 18:33:59 BST; 2 weeks 5 days ago
     Docs: man:firewalld(1)
 Main PID: 4261 (firewalld)
    Tasks: 3 (limit: 99538)
   Memory: 4.7M
   CGroup: /system.slice/firewalld.service
           └─4261 /usr/libexec/platform-python -s /usr/sbin/firewalld --nofork --nopid
```

About Zones and Services

Firewall security is implemented through the concepts of zones and services.

Zones are predefined sets of filtering rules that correspond to levels of trust for network access. You can add to the default filtering rules of a zone by reconfiguring the zone's settings and therefore refine the zone's control of traffic flow. When you install Oracle Linux, a default zone called `public` is automatically assigned to the system.

Firewall rules are applied through services that are assigned to a zone. The service ports are the access points of network traffic. Services assigned to a zone automatically have their ports opened to receive and send network packets.

For more information about zones and firewall-related services, see the `firewalld.zone(5)` and the `firewalld.service(5)` manual pages.

Displaying Information About Zones

When you configure the firewall for zones, displaying the current zone and service settings and other information as part of the configuration steps is a good practice. With this approach you can monitor the changes you're introducing to the firewall and identify potential errors that would make the changes invalid.

To display the system's default zone, run the following command:

```
firewall-cmd --get-default
```

List all the predefined zones that are included in the installation as follows:

```
firewall-cmd --get-zones

block dmz drop external home internal public trusted work
```

You can configure any zone in the list. As you change settings of a particular zone, that zone becomes an active zone. To identify the active zone, type the following:

```
firewall-cmd --get-active-zone
```

> **Note**
>
> By default, all configurations are implemented on the default zone. Note also that an active zone isn't necessarily the default zone. Therefore, you must specify the zone name in the command to define settings for that specific zone. Otherwise, the definitions are applied to the default zone.

Displaying Zone Settings

To obtain the settings of a zone:

```
sudo firewall-cmd --list-all [--zone=zonename]
```

Without specifying a zone, the command displays the settings of the default zone. Thus, to list the settings of the `work` zone, you would use the following command;

```
sudo firewall-cmd --list-all --zone=work

work
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: cockpit dhcpv6-client ssh
  ports:
  protocols:
  forward: yes
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Configuring firewalld Zones

The following tasks describe how to use the `firewall-cmd` command to configure firewall rules for a zone. The rules are then recorded in the `/etc/firewalld` hierarchy for `firewalld`.

Configuring the firewall means setting all or some of a zone settings to specific values to enable the firewall to control network traffic according to specifications.

Controlling Access to Services

Setting the `services` of a zone is the default way to configure the firewall. Each zone has predefined services assigned to it. To configure this setting further, you either add services to the zone or remove services from the zone.

To list predefined services, use the `firewall-cmd --list-services` command.

For example, the following command shows that the `work` zone has the `cockpit`, `dhcpv6-client`, and `ssh` services assigned to it:

```
sudo firewall-cmd --list-services --zone=work

cockpit dhcpv6-client ssh
```

To open access to a new service, use the `--add-service` *service* option. Optionally, include the `--permanent` option to make the rule persistent across reboots.

For example, to add the HTTP and NFS services to the `work` zone, you would use the following command:

```
sudo firewall-cmd --permanent --zone=work --add-service=http --add-service=nfs
sudo firewall-cmd --list-services --zone=work

cockpit dhcpv6-client ssh http nfs
```

To remove access to a service, use the `--remove-service` *service* option:

```
sudo firewall-cmd --permanent --zone=work --remove-service=cockpit
sudo firewall-cmd --list-services --zone=work

dhcpv6-client ssh http nfs
```

Controlling Access to Ports

Network traffic through the zone's services uses the ports of those services. Ports must be opened to accept traffic. You can open more ports for network access by specifying the port number and the associated protocol.

The `--list-ports` option lists the ports and associated protocols to which you have explicitly allowed access. However, ports that have been opened as a service aren't included in this command's output. Therefore, when listing ports, the best practice is to use the `--list-all` option to obtain more complete information.

Use the `--add-port` option to allow access to specific ports. Ports must be specified by using the format *port-number*/*port-type*. Port types can be `tcp`, `udp`, `sctp`, or `dccp`. Ensure that the type and the network traffic match, for example:

```
sudo firewall-cmd --permanent --zone=work --add-port=5353/udp --add-port=3689/tcp
sudo firewall-cmd --list-all --zone=work
```

```
work
  target: default
  icmp-clock-inversion: no
  interfaces:
  sources:
  services: dhcpv6-client ssh http nfs
  ports: 5353/udp 3689/tcp
...
```

Similarly, the `--remove-port` option removes access to a port. Remember to use the `--permanent` option to make the change persist.

For more information, see the `firewall-cmd(1)` manual page.

Assigning a Network Interface to a Zone

A system's network interface is automatically assigned to the default zone. In Oracle Linux, you can configure multiple zones with their specific services, ports, and so on. You then activate a specific zone's rules to become operative by assigning the interface to that zone. Thus, you have the flexibility to easily change the firewall rules that are active on the system by reassigning the network interface.

Suppose that you want to activate the firewall configuration of the `work` zone. You would assign the interface to the zone as follows:

```
sudo firewall-cmd --zone=work --change-interface=enp0s1
firewall-cmd --get-active-zone

work
  interfaces: enp0s1
```

> **Note**
>
> You don't need to use the `--permanent` option to make the setting persist across reboots. If you set the zone to be the default zone, as explained in Changing the Default Zone, then the interface reassignment becomes permanent.

Changing the Default Zone

You can change a system's default zone as follows:

```
sudo firewall-cmd --set-default-zone=work
```

You can also verify that the changes have been applied:

```
firewall-cmd --get-default-zone
```

To display the entire and final results of the configuration:

```
sudo firewall-cmd --zone=work --list-all

work (active)
  target: default
  interfaces:  enp0s1
  sources:
  services: dhcpv6-client ssh http nfs
```

```
  ports: 5353/udp 3689/tcp
...
```

Setting a Default Rule for Controlling Incoming Traffic

The `target` setting establishes the default behavior of the firewall when managing incoming traffic. This zone setting is automatically configured to `default` for all the predefined zones. To change the default behavior of a zone, use the following command;

```
sudo firewall-cmd --zone=zone-name --set-target=ACCEPT|REJECT|DROP
```

You can specify the following options:

- `ACCEPT` accpets all incoming traffic except those you have set to be rejected in another rule.
- `REJECT` blocks all incoming traffic except those you have allowed in another rule. The source machine is informed about the rejecion.
- `DROP` is similar to `REJECT` but no notice of the rejection is sent to the source machine.

Managing Incoming Traffic Based on Sources

You can manage incoming traffic to a zone based on the traffic source. The two following two zone settings enable you to specify the origin of the packets:

- `source` identifies the sending node or network.
- `source-ports` identifies the port from which traffic originates.

To accept incoming traffic from a sending node, use the following command:

```
sudo firewall-cmd --zone=zone-name --add-source=IP-address
```

Note that the IP address can include the netmask in CIDR notation, such as `192.0.2.0/24`.

Run the following command to transform the current runtime ruleset to a permanent ruleset:

```
sudo firewall-cmd --runtime-to-permanent
```

Omit this command if you're setting a temporary configuration that's dropped if the system is rebooted.

The following similar syntax is used to set the `source-port` setting:

```
sudo firewall-cmd --zone=<zone-name> --add-source-port=<portid>/<protocol>
```

In the previous, <*zone-name*> is the zone name, <*portid*> is the port number, and <*protocol*> is one of the following protocol types:

- `tcp`
- `udp`
- `sctp`
- `dccp`

You can combine different settings to configure the firewall. The `trusted` zone can be configured to accept HTTP traffic from the `192.0.2.0` network source, as shown in the following example:

```
sudo firewall-cmd --zone=trusted --add-source=192.0.2.0/24
sudo firewall-cmd --zone=trusted --add-service=http
sudo firewall-cmd --zone=trusted --list-all

trusted (active)
  target: ACCEPT
  sources: 192.0.2.0/24
  services: http
```

Creating Customized Zones

You can create zones and then configure the zone's settings for a customized firewall protection.

Using the firewall-cmd Command

As shown in the following example, you can use the `firewall-cmd` CLI to create an empty zone, which means that no default services are assigned. When configuring a customized zone, you must always include the `--permanent` option in the command. Otherwise, an error message is generated.

```
sudo firewall-cmd --permanent --new-zone=testzone
sudo firewall-cmd --permanent --get-zones

block dmz drop external home internal testzone public trusted work

sudo firewall-cmd --permanent --info-zone=testzone

testzone
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services:
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Without the `--permanent` option, the `--get-zones` option does not display the created zone.

The `--info-zone=`*`zone-name`* option generates the same output as the `--list-all` option.

To make this zone creation persistent, add the following command:

```
sudo firewall-cmd --runtime-to-permanent
```

After creating the zone, you can add services, ports, assign interfaces, and so on, by using the command options that are provided in the previous examples:

```
sudo firewall-cmd --zone=testzone --add-service=http

Error: INVALID ZONE: testzone

sudo firewall-cmd --permanent --zone=testzone --add-service=http
```

Ensure that you use the `--permanent` option when using these commands.

Using a Zone Configuration File

All zones have corresponding configuration files. For the predefined zones that are installed with the operating system, the configuration files are in the `/usr/lib/firewalld/zones` directory.

When you configure a predefined zone, the configuration file is copied to the `/etc/firewalld/zones` directory and the changes are stored in that location. If you use a configuration file to create new zones, you must also use `/etc/firewalld/zones` as the working directory.

If you're creating a zone with only minor differences from the settings of predefined zones, copying an existing configuration file to the working directory is the easiest approach. You can use either of the following commands:

```
sudo cp /etc/firewalld/zones/existing-conf-file.xml new-zone.xml

sudo cp /usr/lib/firewalld/zones/existing-conf-file.xml /etc/firewalld/zones/new-zone.xml
```

Then, using a text editor, revise the settings in the new configuration file. The following example shows what the configuration file of `testzone` might contain. `testzone` accepts traffic for one service (SSH) and one port range for the TCP and UDP protocols:

```
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>testzone</short>
  <description>Put description here</description>
    <service name="ssh"/>
    <port port="1025-65535" protocol="tcp"/>
    <port port="1025-65535" protocol="udp"/>
</zone>
```

3

Configuring the Firewall with nftables

This chapter describes configuring the firewall with `nftables`. It also provides examples for configuring `nftables` tables, chains, and rules that enforce network security on a system with the `nft` command. These examples are appropriate for learning about nftables; however, for more advanced users, consider editing `nftables` configurations from a file. For more information about `nftables` file syntax, see the `nft(8)` manual page.

This chapter also describes how to convert `iptables` and `ip6tables` to the `nftables` framework.

> **Note**
>
> When you create `nftables` configurations using `nft` commands, these configurations reside in the system's memory until you flush the memory or restart the system. To make these configurations persistent across system boots, consider exporting the configurations to an `.nft` file and setup the system to include the file when starting the `nftables` service. For more information about exporting configurations to a file, see Exporting Configurations to a File. For more information about loading a configuration file into `nftables` either manually or automatically, see Loading Configurations from a File.

> **Note**
>
> When working with nftables, it's good practice to keep a local connection (for example, with a serial console if possible) to recover from mistakes that might lock you out of the system.

Disabling the firewalld Service

In Oracle Linux 8, `nftables` isn't enabled by default because the `firewalld` service uses it as its backend. Disable `firewalld` before beginning to work with `nftables`.

Do the following:

1. Disable the `firewalld` service if it isn't already disabled:

   ```
   sudo systemctl disable --now firewalld
   ```

2. Clear any preexisting rulesets in `nftables` created by `firewalld`:

   ```
   sudo nft flush ruleset
   ```

3. Display the current status of the `firewalld` service and ensure that the service is listed as `inactive (dead)`:

```
systemctl status firewalld
```

```
firewalld.service
   Loaded: masked (Reason: Unit firewalld.service is masked.)
   Active: inactive (dead)

Jan 20 15:16:07 localhost.localdomain systemd[1]: Starting firewalld - dynamic
firewall daemon...
Jan 20 15:16:08 localhost.localdomain systemd[1]: Started firewalld - dynamic
firewall daemon.
Jan 20 15:16:08 localhost.localdomain firewalld[1635]: WARNING: AllowZoneDrifting is
enabled. This is considered an insecure configuration option. It will be removed in
a future release. Please consider disabling it now.
Jan 20 15:52:27 localhost.localdomain systemd[1]: Stopping firewalld - dynamic
firewall daemon...
Jan 20 15:52:27 localhost.localdomain systemd[1]: firewalld.service: Succeeded.
Jan 20 15:52:27 localhost.localdomain systemd[1]: Stopped firewalld - dynamic
firewall daemon.
```

About Rulesets and Tables

`nftables` includes rulesets that contain all configuration structures within nftables. Tables are the top level structure in rulesets within which are contained various objects such as chains, rules, and so on.

Managing Rulesets and Tables

To manage `nftable` rulesets and tables in memory, do the following:

1. Create a table for a specific address family type using the following syntax:

   ```
   sudo nft add table <address_family> <table_name>
   ```

   In the previous,

   - *<address_family>* can be `ip`, `ip6`, `inet`, `arp`, `bridge`, or `netdev`. All `nftables` objects are in one of these address families. For more information about these address families, see the `nft(8)` manual page.
   - *<table_name>* is the name of the table. Tables contain chains, which in turn contain rules.

   For example, the following command creates a table called `mytable` with the inet family, which includes addresses for IP version 4 and 6:

   ```
   sudo nft add table inet mytable
   ```

2. Do the following:

   - To view all rulesets, do the following:

     ```
     sudo nft list ruleset
     ```

- To view a single table, do the following:

```
sudo nft list table <address_family> <table_name>
```

About Hooks and Chains

Hooks are points in a system where packets can be intercepted for processing. These hooks are where `nftables` applies chains and rules that decide what happens to a packet. Different hooks are available depending on the address family applied to a table.

The following table shows the hooks available to each address family.

Address Family	Hook	Description
IP, IPv6, inet / Bridge	prerouting	Processes all incoming packets before routing decisions.
	input	Handles packets destined for the local system.
	forward	Manages packets being forwarded to another host.
	output	Processes packets originating from the local system.
	postrouting	Deals with all outgoing packets after routing.
	ingress	Manages incoming packets before the prerouting, available from Linux kernel 5.10 for inet family.
ARP	input	Processes packets for the local system.
	output	Handles packets sent from the local system.
Netdev	ingress	Processes incoming packets after network taps, such as tcpdump, before layer 3 handling.
	egress	Manages outgoing packets after layer 3 handling but before final network exit.

Base chains are attached to hooks. When a packet arrives at a hook, configured with a base chain, the packet then traverses the chain, and each rule in the chain is evaluated in order until a rule matches, or the end of the chain is reached. `nftables` includes the following chain types:

- **Base Chains**: These are chains directly built on a hook. For example, a base chain named `myinput` might be linked to the inet input hook for IPv4 or IPv6 packets arriving at the local system.
- **Regular Chains**: These are chains that jump from base chains or other regular chains. They aren't built directly at a hook but can be part of the decision-making process within a base chain.

Base chains must include the base chain type, hook, and priority parameters. Chain types include:

- `filter`: Primarily used for packet filtering, such as whether to permit or block packets based on various criteria such as source and destination IP, ports, protocols, and so on. This chain type works with all address families and their hooks.
- `nat`: Primarily used to change IP addresses, ports, or both in packet headers for NAT operations. This includes changes to Source NAT (SNAT) for outgoing traffic and Destination NAT (DNAT) for incoming traffic. This chain type can be configured with ip, ipv6, and inet address families and can take the prerouting, input, output, and postrouting hooks.
- `route`: Primarily used for routing decisions such as altering a routing table or marking packets for specific routing policies. This chain type can be used with ip and ipv6 address families and can take the output hook only.

When using the ingress or egress hooks, specify a network interface name as a string with the `device` parameter. Any ingress or egress chains only filters traffic from the interface specified in the device parameter.

The priority parameter uses a signed integer or a standard priority name to determine the order in which chains with the same hook are processed. Chains process from lower priority values or names to higher priority values or names.

The following table shows priority names and values for each associated address family and hook type.

Name	Value	Family	Hooks
raw	-300	ip, ip6, inet	all
mangle	-150	ip, ip6, inet	all
dsnat	-100	ip, ip6, inet	prerouting
filter	0	ip, ip6, inet, arp, netdev	all
security	50	ip, ip6, inet	all
srcnat	100	ip, ip6, inet	postrouting

The following table shows priority names and values for the bridge address family and hook type.

Name	Value	Hooks
dsnat	-300	prerouting
filter	-200	all
out	100	output
srcnat	300	postrouting

You can also choose to set a policy on a chain that defines whether to `accept` or `drop` a packet when none of the rules defined on the chain matches the packet. By default, base chains accept all packets. However, setting a policy to drop all traffic not explicitly allowed by a chain is good practice for security reasons.

Creating Base Chains

To create `nftable` base chains in memory, do the following:

1. Create a chain for a specific address family type and table using the following syntax:

```
sudo nft add chain <address_family> <table_name> <chain_name>{ type <chain_type>
hook <hook_type> device <network_interface_name> priority <priority> policy <policy>
comment <comment> ; }
```

In the previous,

- *<address_family>* can be `ip`, `ip6`, `inet`, `arp`, `bridge`, or `netdev`. All `nftables` objects are in one of these address family. For more information about these address families, see the `nft(8)` manual page.
- *<table_name>* is the name of the table. Tables are containers for chains, which in turn are containers for rules.
- *<chain_name>* is an arbitrary name for the chain. People migrating from iptables based firewalls often use traditional iptables naming.
- *<chain_type>* is the chain type. Valid values for base chains are `filter`, `nat`, and `route`. For more information, see About Hooks and Chains.
- *<hook_type>* is the hook type. Possible values depend on the selected address family and chain type. For more information, see About Hooks and Chains.
- *<network_interface_name>* is the network interface name for the device parameter. This parameter is only required when using the ingress or egress hooks.
- *<priority>* is the priority value or name for the chain. Priority depends on the selected address family and hook type. For more information, see About Hooks and Chains.
- *<policy>* is the action taken if all rules defined on the chain fail to match the packet. Valid values are `accept` or `drop`. If not specified, the default value is `accept`.

For example, the following command creates a chain in the `mytable` table called `mychain` with the `inet` family. The chain type is `filter`, the hook is `input`, and the priority is `0`. Finally, the policy is set to drop all packets that don't match a rule:

```
sudo nft add chain inet mytable mychain "{ type filter hook input priority
0 ; policy drop ; }"
```

2. Do the following:
   - To view all chains, do the following:

     ```
     sudo nft list chains
     ```

   - To view a single chain, do the following:

     ```
     sudo nft list chain <address_family> <table_name> <chain_name>
     ```

About Rules

You can add rules to chains in a table. Rules are made up of instructions that match packets based on various criteria and apply actions such as accept, drop, reject, and so on. These instructions include elements such as:

- Sets, which are collections of elements used for the matching.
- Expressions, which are building blocks for rules that define how packets are matched or manipulated. Examples include:
  - Protocol-specific matches (for example, ip, ip6, tcp, udp).

  - Address matching (saddr, daddr).
  - Port matching (sport, dport).
  - Network interface matches (iifname, oifname).
- Maps, which are similar to sets but can map one value to another. Maps are used for more complex matching or transformation scenarios.
- Verdict maps, which can change verdicts based on packet content, enable more dynamic policy decisions.
- Counters, which track the number of packets matching a rule and display when running a list `nft` command that includes the rule. Counters are a useful way to test whether a rule is working.
- Quotas, which limit the amount of data that can pass through a rule before an action changes (for example, from `accept` to `drop`).
- Flowtables, which allows fast path packet forwarding, improving performance by bypassing regular packet processing for certain traffic.
- Statements, which includes operations such as `log`, `reject`, `jump`, `goto` which change packet handling behavior.

Rules added to a chain are evaluated from top to bottom and from left to right.

For more information about configuring these instructions, see the `nft(8)` manual page.

Creating Rules - Examples

This section includes some common example of rules created within chains.

Allow local traffic

The following command creates a rule in `mytable` within `mychain` for the `inet` family that allows local traffic to enter through the loopback interface (iff lo):

```
sudo nft add rule inet mytable mychain iif lo accept
```

Allow incoming traffic for an existing connection or related to an existing connection

This command adds a rule to the `mytable`'s `mychain` chain that accepts all incoming traffic that are part of or related to an existing connection:

```
sudo nft add rule inet mytable mychain ct state established, related accept
```

In the previous example,

- `ct` is a connection tracking helper for IPv4, IPv6, or inet which is part of the `nf_conntrack` module. This module is typically loaded by default on most systems. You can list the state of all connections to the system using the following command:

  ```
  sudo cat /proc/net/nf_conntrack
  ```

  Possible connection states are:

  - `NEW`: The packet has started a new connection, or is associated with a connection that hasn't received and sent packets.

  - `ESTABLISHED`: The packet is associated with a connection which has received and sent packets.
  - `RELATED`: The packet is starting a new connection, but is associated with an existing connection.
  - `INVALID`: The packet is associated with no known connection.
- `state established, related` indicates that the command applies only to connections in the `ESTABLISHED` or `RELATED` state.
- `accept` indicates that any packet with the appropriate state can be accepted.

This is a common rule in firewall configurations to ensure that responses to outgoing traffic are allowed back in, enabling normal operations of network services such as web browsing, SSH connectivity, and so on, without needing to explicitly open all ports for incoming connections.

Allow incoming SSH traffic

The following adds a rule to the `mytable`'s `mychain` chain that accepts all incoming TCP traffic on destination port 22.

```
sudo nft add rule inet mytable mychain tcp dport 22 accept
```

This port is typically used for SSH traffic and assumes that the SSH daemon is setup and running on the system.

Restricting all IPv4 and IPv6 traffic (Panic button)

The following adds a rule to the `mytable` table that drops all incoming and outgoing IPv4 and IPv6 and acts as a kind of panic button.

```
sudo nft add rule inet mytable drop
```

Exporting Configurations to a File

To keep `nftable` configurations across boots or to switch from one configuration to another, you can export an nftable in memory to a file.

To export `nftable` configurations to a file, do the following:

1. List the rulesets and save the output to a file:

   ```
   sudo nft list ruleset > /etc/nftables/<export_file_name>.nft
   ```

   In the previous, *<export_file_name>* is the name of the file for the exported information. This file now contains all the tables, chains, and rules available in memory.
2. List a table and save the output to a file:

   ```
   sudo nft list table <address_family> <table_name> > /etc/nftables/
   <export_file_name>.nft
   ```

   This file now contains a table and all associated chains and rules available in memory.
3. List a chain and save the output to a file:

   ```
   sudo nft list chain <address_family> <table_name> <chain_name> > /etc/nftables/
   <export_file_name>.nft
   ```

This file now contains a chain in a table and all associated rules within the chain available in memory.

4. Ensure the files included in `/etc/nftables` are executable:

   ```
   sudo chmod +x /etc/nftables/<export_file_name>.nft
   ```

Loading Configurations from a File

To load an `nftable` ruleset, table, or chain from a file into memory, you can perform this task manually or automatically when rebooting a system.

To manually load an `nftables` file, do the following:

1. Before loading a new configuration from a file, drop the existing tables:

   ```
   sudo nft flush ruleset
   ```

   > **Note**
   >
   > This step is crucial to avoid conflicts between the new and old configurations and ensures a clean and consistent application of the new rules.

2. Run the following command to load the file into memory:

   ```
   sudo nft -f /etc/nftables/<import_file_name>.nft
   ```

   In the previous, *<import_file_name>* is the name of the file with the information to be imported. This file might contain a ruleset, one or more tables, one or more chains within a table, and any associated rules.

   > **Note**
   >
   > The atomic reload is an `nftables` feature that ensures that connection tracking is preserved during rule reloading, providing a seamless transition to the new configuration.

3. List a ruleset to verify that the file imported correctly:

   ```
   sudo nft list rulesets
   ```

To automatically load a ruleset from a file when restarting the system, do the following:

1. Edit the `/etc/sysconfig/nftables.conf` file to include the .nft table files you want to include at startup. If this file doesn't exist, create it. For example, the following shows that the `/etc/sysconfig/nftables.conf` now includes the exported `/etc/nftables/myruleset.nft` file.

   ```
   # Uncomment the include statement here to load the default config sample
   # in /etc/nftables for nftables service.

   include "/etc/nftables/myruleset.nft"

   # To customize, either edit the samples in /etc/nftables, append further
   # commands to the end of this file or overwrite it after first service
   # start by calling: 'nft list ruleset >/etc/sysconfig/nftables.conf'.
   ```

2. Enable and start the `nftables` service:

```
sudo systemctl enable --now nftables
```

Converting iptables to nftables

If you query the system's `iptables` version, Oracle Linux 8 would clearly indicate that `nftables` is used as the packet filtering framework:

```
sudo iptables --version

iptables v1.8.2 (nf_tables)
```

Otherwise, the output would be similar to the following example:

```
sudo iptables --version

iptables version (legacy)
```

Utilities are available to convert filter rules in `iptables` and `ip6tables` to their equivalents in the `nftables` framework. Choose from one of the following ways.

- Use the `iptables-translate` or `ip6tables-translate` commands, depending on the type of tables you want to convert. If a rule can't be translated because of an unrecognized extension in the rule, the command prints the untranslated rule preceded by the # sign.

  ```
  sudo iptables-translate -A INPUT -j CHECKSUM --checksum-fill

  nft #  -A INPUT -j CHECKSUM --checksum-fill
  ```

- Save the rules to a dump file, then use the `iptables-restore-translate` or `ip6tables-restore-translate` command, depending on the type of tables you want to convert.

  ```
  sudo iptables-save > /tmp/iptables.dump

  sudo iptables-restore-translate -f /tmp/iptables.dump

  translated-rules
  ```