Red Hat Enterprise Linux 9

Configuring firewalls and packet filters

A guide to configuring and managing firewalls and packet-filtering technologies in Red Hat Enterprise Linux 9

Last Updated: 2022-06-23

Red Hat Enterprise Linux 9 Configuring firewalls and packet filters

A guide to configuring and managing firewalls and packet-filtering technologies in Red Hat Enterprise Linux 9

Legal Notice

Abstract

This document describes how to configure firewalls and filter network packets in Red Hat Enterprise Linux 9.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better.

- For simple comments on specific passages:
    1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
    2. Use your mouse cursor to highlight the part of text that you want to comment on.
    3. Click the **Add Feedback** pop-up that appears below the highlighted text.
    4. Follow the displayed instructions.
- For submitting feedback via Bugzilla, create a new ticket:
    1. Go to the Bugzilla website.
    2. As the Component, use **Documentation**.
    3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
    4. Click **Submit Bug**.

CHAPTER 1. USING AND CONFIGURING FIREWALLD

A *firewall* is a way to protect machines from any unwanted traffic from outside. It enables users to control incoming network traffic on host machines by defining a set of *firewall rules*. These rules are used to sort the incoming traffic and either block it or allow through.

firewalld is a firewall service daemon that provides a dynamic customizable host-based firewall with a D-Bus interface. Being dynamic, it enables creating, changing, and deleting the rules without the necessity to restart the firewall daemon each time the rules are changed.

firewalld uses the concepts of zones and services, that simplify the traffic management. Zones are predefined sets of rules. Network interfaces and sources can be assigned to a zone. The traffic allowed depends on the network your computer is connected to and the security level this network is assigned. Firewall services are predefined rules that cover all necessary settings to allow incoming traffic for a specific service and they apply within a zone.

Services use one or more ports or addresses for network communication. Firewalls filter communication based on ports. To allow network traffic for a service, its ports must be open. **firewalld** blocks all traffic on ports that are not explicitly set as open. Some zones, such as trusted, allow all traffic by default.

Note that **firewalld** with **nftables** backend does not support passing custom **nftables** rules to **firewalld**, using the **--direct** option.

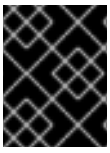
1.1. GETTING STARTED WITH FIREWALLD

This section provides information about **firewalld**.

1.1.1. When to use firewalld, nftables, or iptables

The following is a brief overview in which scenario you should use one of the following utilities:

- **firewalld**: Use the **firewalld** utility for simple firewall use cases. The utility is easy to use and covers the typical use cases for these scenarios.
- **nftables**: Use the **nftables** utility to set up complex and performance critical firewalls, such as for a whole network.
- **iptables**: The **iptables** utility on Red Hat Enterprise Linux uses the **nf_tables** kernel API instead of the **legacy** back end. The **nf_tables** API provides backward compatibility so that scripts that use **iptables** commands still work on Red Hat Enterprise Linux. For new firewall scripts, Red Hat recommends to use **nftables**.



IMPORTANT

To avoid that the different firewall services influence each other, run only one of them on a RHEL host, and disable the other services.

1.1.2. Zones

firewalld can be used to separate networks into different zones according to the level of trust that the user has decided to place on the interfaces and traffic within that network. A connection can only be part of one zone, but a zone can be used for many network connections.

NetworkManager notifies **firewalld** of the zone of an interface. You can assign zones to interfaces with:

- **NetworkManager**
- **firewall-config** tool
- **firewall-cmd** command-line tool
- The RHEL web console

The latter three can only edit the appropriate **NetworkManager** configuration files. If you change the zone of the interface using the web console, **firewall-cmd** or **firewall-config**, the request is forwarded to **NetworkManager** and is not handled by **firewalld**.

The predefined zones are stored in the **/usr/lib/firewalld/zones/** directory and can be instantly applied to any available network interface. These files are copied to the **/etc/firewalld/zones/** directory only after they are modified. The default settings of the predefined zones are as follows:

block

Any incoming network connections are rejected with an icmp-host-prohibited message for **IPv4** and icmp6-adm-prohibited for **IPv6**. Only network connections initiated from within the system are possible.

dmz

For computers in your demilitarized zone that are publicly-accessible with limited access to your internal network. Only selected incoming connections are accepted.

drop

Any incoming network packets are dropped without any notification. Only outgoing network connections are possible.

external

For use on external networks with masquerading enabled, especially for routers. You do not trust the other computers on the network to not harm your computer. Only selected incoming connections are accepted.

home

For use at home when you mostly trust the other computers on the network. Only selected incoming connections are accepted.

internal

For use on internal networks when you mostly trust the other computers on the network. Only selected incoming connections are accepted.

public

For use in public areas where you do not trust other computers on the network. Only selected incoming connections are accepted.

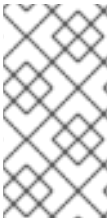
trusted

All network connections are accepted.

work

For use at work where you mostly trust the other computers on the network. Only selected incoming connections are accepted.

One of these zones is set as the *default* zone. When interface connections are added to **NetworkManager**, they are assigned to the default zone. On installation, the default zone in **firewalld** is set to be the **public** zone. The default zone can be changed.



NOTE

The network zone names should be self-explanatory and to allow users to quickly make a reasonable decision. To avoid any security problems, review the default zone configuration and disable any unnecessary services according to your needs and risk assessments.

Additional resources

- The **firewalld.zone(5)** man page.

1.1.3. Predefined services

A service can be a list of local ports, protocols, source ports, and destinations, as well as a list of firewall helper modules automatically loaded if a service is enabled. Using services saves users time because they can achieve several tasks, such as opening ports, defining protocols, enabling packet forwarding and more, in a single step, rather than setting up everything one after another.

Service configuration options and generic file information are described in the **firewalld.service(5)** man page. The services are specified by means of individual XML configuration files, which are named in the following format: ***service-name*.xml**. Protocol names are preferred over service or application names in **firewalld**.

Services can be added and removed using the graphical **firewall-config** tool, **firewall-cmd**, and **firewall-offline-cmd**.

Alternatively, you can edit the XML files in the **/etc/firewalld/services/** directory. If a service is not added or changed by the user, then no corresponding XML file is found in **/etc/firewalld/services/**. The files in the **/usr/lib/firewalld/services/** directory can be used as templates if you want to add or change a service.

Additional resources

- The **firewalld.service(5)** man page

1.1.4. Starting firewalld

Procedure

1. To start **firewalld**, enter the following command as **root**:

```
# systemctl unmask firewalld
# systemctl start firewalld
```

2. To ensure **firewalld** starts automatically at system start, enter the following command as **root**:

```
# systemctl enable firewalld
```

1.1.5. Stopping firewalld

Procedure

1. To stop **firewalld**, enter the following command as **root**:

```
# systemctl stop firewalld
```

2. To prevent **firewalld** from starting automatically at system start:

```
# systemctl disable firewalld
```

3. To make sure firewalld is not started by accessing the **firewalld D-Bus** interface and also if other services require **firewalld**:

```
# systemctl mask firewalld
```

1.1.6. Verifying the permanent firewalld configuration

In certain situations, for example after manually editing **firewalld** configuration files, administrators want to verify that the changes are correct. This section describes how to verify the permanent configuration of the **firewalld** service.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Verify the permanent configuration of the **firewalld** service:

```
# firewall-cmd --check-config
success
```

If the permanent configuration is valid, the command returns **success**. In other cases, the command returns an error with further details, such as the following:

```
# firewall-cmd --check-config
Error: INVALID_PROTOCOL: 'public.xml': 'tcpx' not from {'tcp'|'udp'|'sctp'|'dccp'}
```

1.2. VIEWING THE CURRENT STATUS AND SETTINGS OF FIREWALLD

This section covers information about viewing current status, allowed services, and current settings of **firewalld**.

1.2.1. Viewing the current status of firewalld

The firewall service, **firewalld**, is installed on the system by default. Use the **firewalld** CLI interface to check that the service is running.

Procedure

1. To see the status of the service:

```
# firewall-cmd --state
```

2. For more information about the service status, use the **systemctl status** sub-command:

```
# systemctl status firewalld
firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor pr
   Active: active (running) since Mon 2017-12-18 16:05:15 CET; 50min ago
     Docs: man:firewalld(1)
 Main PID: 705 (firewalld)
    Tasks: 2 (limit: 4915)
   CGroup: /system.slice/firewalld.service
           └─705 /usr/bin/python3 -Es /usr/sbin/firewalld --nofork --nopid
```

1.2.2. Viewing allowed services using GUI

To view the list of services using the graphical **firewall-config** tool, press the **Super** key to enter the Activities Overview, type **firewall**, and press **Enter**. The **firewall-config** tool appears. You can now view the list of services under the **Services** tab.

You can start the graphical firewall configuration tool using the command-line.

Prerequisites

- You installed the **firewall-config** package.

Procedure

- To start the graphical firewall configuration tool using the command-line:

```
$ firewall-config
```

The **Firewall Configuration** window opens. Note that this command can be run as a normal user, but you are prompted for an administrator password occasionally.

1.2.3. Viewing firewalld settings using CLI

With the CLI client, it is possible to get different views of the current firewall settings. The **--list-all** option shows a complete overview of the **firewalld** settings.

firewalld uses zones to manage the traffic. If a zone is not specified by the **--zone** option, the command is effective in the default zone assigned to the active network interface and connection.

Procedure

- To list all the relevant information for the default zone:

```
# firewall-cmd --list-all
public
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: ssh dhcpv6-client
  ports:
  protocols:
  masquerade: no
  forward-ports:
```

```
  source-ports:
  icmp-blocks:
  rich rules:
```

- To specify the zone for which to display the settings, add the **--zone=*zone-name*** argument to the **firewall-cmd --list-all** command, for example:

```
# firewall-cmd --list-all --zone=home
home
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: ssh mdns samba-client dhcpv6-client
... [trimmed for clarity]
```

- To see the settings for particular information, such as services or ports, use a specific option. See the **firewalld** manual pages or get a list of the options using the command help:

```
# firewall-cmd --help
```

- To see which services are allowed in the current zone:

```
# firewall-cmd --list-services
ssh dhcpv6-client
```

NOTE

Listing the settings for a certain subpart using the CLI tool can sometimes be difficult to interpret. For example, you allow the **SSH** service and **firewalld** opens the necessary port (22) for the service. Later, if you list the allowed services, the list shows the **SSH** service, but if you list open ports, it does not show any. Therefore, it is recommended to use the **--list-all** option to make sure you receive a complete information.

1.3. CONTROLLING NETWORK TRAFFIC USING FIREWALLD

This section covers information about controlling network traffic using **firewalld**.

1.3.1. Disabling all traffic in case of emergency using CLI

In an emergency situation, such as a system attack, it is possible to disable all network traffic and cut off the attacker.

Procedure

1. To immediately disable networking traffic, switch panic mode on:

```
# firewall-cmd --panic-on
```

IMPORTANT

Enabling panic mode stops all networking traffic. For this reason, it should be used only when you have the physical access to the machine or if you are logged in using a serial console.

2. Switching off panic mode reverts the firewall to its permanent settings. To switch panic mode off, enter:

```
# firewall-cmd --panic-off
```

Verification

- To see whether panic mode is switched on or off, use:

```
# firewall-cmd --query-panic
```

1.3.2. Controlling traffic with predefined services using CLI

The most straightforward method to control traffic is to add a predefined service to **firewalld**. This opens all necessary ports and modifies other settings according to the *service definition file* .

Procedure

1. Check that the service is not already allowed:

```
# firewall-cmd --list-services
ssh dhcpv6-client
```

2. List all predefined services:

```
# firewall-cmd --get-services
RH-Satellite-6 amanda-client amanda-k5-client bacula bacula-client bitcoin bitcoin-rpc
bitcoin-testnet bitcoin-testnet-rpc ceph ceph-mon cfengine condor-collector ctdb dhcp dhcpv6
dhcpv6-client dns docker-registry ...
[trimmed for clarity]
```

3. Add the service to the allowed services:

```
# firewall-cmd --add-service=<service-name>
```

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

1.3.3. Controlling traffic with predefined services using GUI

This procedure describes how to control the network traffic with predefined services using graphical user interface.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. To enable or disable a predefined or custom service:
   a. Start the **firewall-config** tool and select the network zone whose services are to be configured.
   b. Select the **Zones** tab and then the **Services** tab below.
   c. Select the check box for each type of service you want to trust or clear the check box to block a service in the selected zone.
2. To edit a service:
   a. Start the **firewall-config** tool.
   b. Select **Permanent** from the menu labeled **Configuration**. Additional icons and menu buttons appear at the bottom of the **Services** window.
   c. Select the service you want to configure.

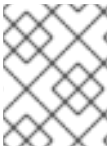
The **Ports**, **Protocols**, and **Source Port** tabs enable adding, changing, and removing of ports, protocols, and source port for the selected service. The modules tab is for configuring **Netfilter** helper modules. The **Destination** tab enables limiting traffic to a particular destination address and Internet Protocol (**IPv4** or **IPv6**).

NOTE

It is not possible to alter service settings in the **Runtime** mode.

1.3.4. Adding new services

Services can be added and removed using the graphical **firewall-config** tool, **firewall-cmd**, and **firewall-offline-cmd**. Alternatively, you can edit the XML files in **/etc/firewalld/services/**. If a service is not added or changed by the user, then no corresponding XML file are found in **/etc/firewalld/services/**. The files **/usr/lib/firewalld/services/** can be used as templates if you want to add or change a service.



NOTE

Service names must be alphanumeric and can, additionally, include only _ (underscore) and **-** (dash) characters.

Procedure

To add a new service in a terminal, use **firewall-cmd**, or **firewall-offline-cmd** in case of not active **firewalld**.

1. Enter the following command to add a new and empty service:

```
$ firewall-cmd --new-service=service-name --permanent
```

2. To add a new service using a local file, use the following command:

```
$ firewall-cmd --new-service-from-file=service-name.xml --permanent
```

You can change the service name with the additional **--name=*service-name*** option.

3. As soon as service settings are changed, an updated copy of the service is placed into **/etc/firewalld/services/**.
   As **root**, you can enter the following command to copy a service manually:

```
# cp /usr/lib/firewalld/services/service-name.xml /etc/firewalld/services/service-name.xml
```

firewalld loads files from **/usr/lib/firewalld/services** in the first place. If files are placed in **/etc/firewalld/services** and they are valid, then these will override the matching files from **/usr/lib/firewalld/services**. The overridden files in **/usr/lib/firewalld/services** are used as soon as the matching files in **/etc/firewalld/services** have been removed or if **firewalld** has been asked to load the defaults of the services. This applies to the permanent environment only. A reload is needed to get these fallbacks also in the runtime environment.

1.3.5. Opening ports using GUI

To permit traffic through the firewall to a certain port, you can open the port in the GUI.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. Start the **firewall-config** tool and select the network zone whose settings you want to change.
2. Select the **Ports** tab and click the **Add** button on the right-hand side. The **Port and Protocol** window opens.
3. Enter the port number or range of ports to permit.
4. Select **tcp** or **udp** from the list.

1.3.6. Controlling traffic with protocols using GUI

To permit traffic through the firewall using a certain protocol, you can use the GUI.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. Start the **firewall-config** tool and select the network zone whose settings you want to change.
2. Select the **Protocols** tab and click the **Add** button on the right-hand side. The **Protocol** window opens.
3. Either select a protocol from the list or select the **Other Protocol** check box and enter the protocol in the field.

1.3.7. Opening source ports using GUI

To permit traffic through the firewall from a certain port, you can use the GUI.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. Start the firewall-config tool and select the network zone whose settings you want to change.
2. Select the **Source Port** tab and click the **Add** button on the right-hand side. The **Source Port** window opens.
3. Enter the port number or range of ports to permit. Select **tcp** or **udp** from the list.

1.4. CONTROLLING PORTS USING CLI

Ports are logical devices that enable an operating system to receive and distinguish network traffic and forward it accordingly to system services. These are usually represented by a daemon that listens on the port, that is it waits for any traffic coming to this port.

Normally, system services listen on standard ports that are reserved for them. The **httpd** daemon, for example, listens on port 80. However, system administrators by default configure daemons to listen on different ports to enhance security or for other reasons.

1.4.1. Opening a port

Through open ports, the system is accessible from the outside, which represents a security risk. Generally, keep ports closed and only open them if they are required for certain services.

Procedure

To get a list of open ports in the current zone:

1. List all allowed ports:

```
# firewall-cmd --list-ports
```

2. Add a port to the allowed ports to open it for incoming traffic:

```
# firewall-cmd --add-port=port-number/port-type
```

The port types are either **tcp**, **udp**, **sctp**, or **dccp**. The type must match the type of network communication.

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

The port types are either **tcp**, **udp**, **sctp**, or **dccp**. The type must match the type of network communication.

1.4.2. Closing a port

When an open port is no longer needed, close that port in **firewalld**. It is highly recommended to close all unnecessary ports as soon as they are not used because leaving a port open represents a security risk.

Procedure

To close a port, remove it from the list of allowed ports:

1. List all allowed ports:

```
# firewall-cmd --list-ports
```

WARNING

This command will only give you a list of ports that have been opened as ports. You will not be able to see any open ports that have been opened as a service. Therefore, you should consider using the **--list-all** option instead of **--list-ports**.

2. Remove the port from the allowed ports to close it for the incoming traffic:

```
# firewall-cmd --remove-port=port-number/port-type
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

1.5. CONFIGURING PORTS USING SYSTEM ROLES

You can use the Red Hat Enterprise Linux (RHEL) **firewalld** System Role to open or close ports in the local firewall for incoming traffic and make the new configuration persist across reboots. The example describes how to configure the default zone to permit incoming traffic for the **HTTPS** service.

Run this procedure on the Ansible control node.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **firewalld** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
- The **ansible-core** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.
- The host uses NetworkManager to configure the network.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

```
node.example.com
```

2. Create the **~/adding-and-removing-ports.yml** playbook with the following content:

```
---
- name: Allow incoming HTTPS traffic to the local host
  hosts: node.example.com
  become: true

  tasks:
    - include_role:
        name: linux-system-roles.firewall

      vars:
        firewall:
          - port: 443/tcp
            service: http
            state: enabled
            runtime: true
            permanent: true
```

The **permanent: true** option makes the new settings persistent across reboots.

3. Run the playbook:

- To connect as **root** user to the managed host, enter:

```
# ansible-playbook -u root ~/adding-and-removing-ports.yml
```

- To connect as a user to the managed host, enter:

```
# ansible-playbook -u user_name --ask-become-pass ~/adding-and-removing-ports.yml
```

The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Verification

1. Connect to the managed node:

```
$ ssh user_name@node.example.com
```

2. Verify that the **443/tcp** port associated with the **HTTPS** service is open:

```
$ sudo firewall-cmd --list-ports
443/tcp
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**
- **ansible-playbook(1)** man page

1.6. WORKING WITH FIREWALLD ZONES

Zones represent a concept to manage incoming traffic more transparently. The zones are connected to networking interfaces or assigned a range of source addresses. You manage firewall rules for each zone independently, which enables you to define complex firewall settings and apply them to the traffic.

1.6.1. Listing zones

This procedure describes how to list zones using the command line.

Procedure

1. To see which zones are available on your system:

   ```
   # firewall-cmd --get-zones
   ```

   The **firewall-cmd --get-zones** command displays all zones that are available on the system, but it does not show any details for particular zones.

2. To see detailed information for all zones:

   ```
   # firewall-cmd --list-all-zones
   ```

3. To see detailed information for a specific zone:

   ```
   # firewall-cmd --zone=zone-name --list-all
   ```

1.6.2. Modifying firewalld settings for a certain zone

The Controlling traffic with predefined services using cli and Controlling ports using cli explain how to add services or modify ports in the scope of the current working zone. Sometimes, it is required to set up rules in a different zone.

Procedure

- To work in a different zone, use the **--zone=*zone-name*** option. For example, to allow the **SSH** service in the zone *public*:

  ```
  # firewall-cmd --add-service=ssh --zone=public
  ```

1.6.3. Changing the default zone

System administrators assign a zone to a networking interface in its configuration files. If an interface is not assigned to a specific zone, it is assigned to the default zone. After each restart of the **firewalld** service, **firewalld** loads the settings for the default zone and makes it active.

Procedure

To set up the default zone:

1. Display the current default zone:

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

2. Set the new default zone:

```
# firewall-cmd --set-default-zone zone-name
```

NOTE

Following this procedure, the setting is a permanent setting, even without the **--permanent** option.

1.6.4. Assigning a network interface to a zone

It is possible to define different sets of rules for different zones and then change the settings quickly by changing the zone for the interface that is being used. With multiple interfaces, a specific zone can be set for each of them to distinguish traffic that is coming through them.

Procedure

To assign the zone to a specific interface:

1. List the active zones and the interfaces assigned to them:

```
# firewall-cmd --get-active-zones
```

2. Assign the interface to a different zone:

```
# firewall-cmd --zone=zone_name --change-interface=interface_name --permanent
```

1.6.5. Assigning a zone to a connection using nmcli

This procedure describes how to add a **firewalld** zone to a **NetworkManager** connection using the **nmcli** utility.

Procedure

1. Assign the zone to the **NetworkManager** connection profile:

```
# nmcli connection modify profile connection.zone zone_name
```

2. Activate the connection:

```
# nmcli connection up profile
```

1.6.6. Manually assigning a zone to a network connection in an ifcfg file

When the connection is managed by **NetworkManager**, it must be aware of a zone that it uses. For every network connection, a zone can be specified, which provides the flexibility of various firewall settings according to the location of the computer with portable devices. Thus, zones and settings can be specified for different locations, such as company or home.

Procedure

- To set a zone for a connection, edit the **/etc/sysconfig/network-scripts/ifcfg-*connection_name*** file and add a line that assigns a zone to this connection:

```
ZONE=zone_name
```

1.6.7. Creating a new zone

To use custom zones, create a new zone and use it just like a predefined zone. New zones require the **--permanent** option, otherwise the command does not work.

Procedure

1. Create a new zone:

```
# firewall-cmd --permanent --new-zone=zone-name
```

2. Check if the new zone is added to your permanent settings:

```
# firewall-cmd --get-zones
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

1.6.8. Zone configuration files

Zones can also be created using a *zone configuration file*. This approach can be helpful when you need to create a new zone, but want to reuse the settings from a different zone and only alter them a little.

A **firewalld** zone configuration file contains the information for a zone. These are the zone description, services, ports, protocols, icmp-blocks, masquerade, forward-ports and rich language rules in an XML file format. The file name has to be ***zone-name*.xml** where the length of *zone-name* is currently limited to 17 chars. The zone configuration files are located in the **/usr/lib/firewalld/zones/** and **/etc/firewalld/zones/** directories.

The following example shows a configuration that allows one service (**SSH**) and one port range, for both the **TCP** and **UDP** protocols:

```
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>My Zone</short>
  <description>Here you can describe the characteristic features of the zone.</description>
  <service name="ssh"/>
```

```
  <port protocol="udp" port="1025-65535"/>
  <port protocol="tcp" port="1025-65535"/>
</zone>
```

To change settings for that zone, add or remove sections to add ports, forward ports, services, and so on.

Additional resources

- **firewalld.zone** manual page

1.6.9. Using zone targets to set default behavior for incoming traffic

For every zone, you can set a default behavior that handles incoming traffic that is not further specified. Such behavior is defined by setting the target of the zone. There are four options:

- **ACCEPT**: Accepts all incoming packets except those disallowed by specific rules.
- **REJECT**: Rejects all incoming packets except those allowed by specific rules. When **firewalld** rejects packets, the source machine is informed about the rejection.
- **DROP**: Drops all incoming packets except those allowed by specific rules. When **firewalld** drops packets, the source machine is not informed about the packet drop.
- **default**: Similar behavior as for **REJECT**, but with special meanings in certain scenarios. For details, see the **Options to Adapt and Query Zones and Policies** section in the **firewall-cmd(1)** man page.

Procedure

To set a target for a zone:

1. List the information for the specific zone to see the default target:

```
# firewall-cmd --zone=zone-name --list-all
```

2. Set a new target in the zone:

```
# firewall-cmd --permanent --zone=zone-name --set-target=
<default|ACCEPT|REJECT|DROP>
```

Additional resources

- **firewall-cmd(1)** man page

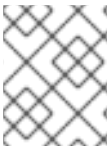
1.7. USING ZONES TO MANAGE INCOMING TRAFFIC DEPENDING ON A SOURCE

You can use zones to manage incoming traffic based on its source. That enables you to sort incoming traffic and route it through different zones to allow or disallow services that can be reached by that traffic.

If you add a source to a zone, the zone becomes active and any incoming traffic from that source will be directed through it. You can specify different settings for each zone, which is applied to the traffic from the given sources accordingly. You can use more zones even if you only have one network interface.

1.7.1. Adding a source

To route incoming traffic into a specific zone, add the source to that zone. The source can be an IP address or an IP mask in the classless inter-domain routing (CIDR) notation.



NOTE

In case you add multiple zones with an overlapping network range, they are ordered alphanumerically by zone name and only the first one is considered.

- To set the source in the current zone:

```
# firewall-cmd --add-source=<source>
```

- To set the source IP address for a specific zone:

```
# firewall-cmd --zone=zone-name --add-source=<source>
```

The following procedure allows all incoming traffic from *192.168.2.15* in the **trusted** zone:

Procedure

1. List all available zones:

```
# firewall-cmd --get-zones
```

2. Add the source IP to the trusted zone in the permanent mode:

```
# firewall-cmd --zone=trusted --add-source=192.168.2.15
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

1.7.2. Removing a source

Removing a source from the zone cuts off the traffic coming from it.

Procedure

1. List allowed sources for the required zone:

```
# firewall-cmd --zone=zone-name --list-sources
```

2. Remove the source from the zone permanently:

```
# firewall-cmd --zone=zone-name --remove-source=<source>
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

1.7.3. Adding a source port

To enable sorting the traffic based on a port of origin, specify a source port using the **--add-source-port** option. You can also combine this with the **--add-source** option to limit the traffic to a certain IP address or IP range.

Procedure

- To add a source port:

```
# firewall-cmd --zone=zone-name --add-source-port=<port-name>/<tcp|udp|sctp|dccp>
```

1.7.4. Removing a source port

By removing a source port you disable sorting the traffic based on a port of origin.

Procedure

- To remove a source port:

```
# firewall-cmd --zone=zone-name --remove-source-port=<port-name>/<tcp|udp|sctp|dccp>
```

1.7.5. Using zones and sources to allow a service for only a specific domain

To allow traffic from a specific network to use a service on a machine, use zones and source. The following procedure allows only HTTP traffic from the **192.0.2.0/24** network while any other traffic is blocked.

WARNING

When you configure this scenario, use a zone that has the **default** target. Using a zone that has the target set to **ACCEPT** is a security risk, because for traffic from **192.0.2.0/24**, all network connections would be accepted.

Procedure

1. List all available zones:

```
# firewall-cmd --get-zones
block dmz drop external home internal public trusted work
```

2. Add the IP range to the **internal** zone to route the traffic originating from the source through the zone:

```
# firewall-cmd --zone=internal --add-source=192.0.2.0/24
```

3. Add the **http** service to the **internal** zone:

```
# firewall-cmd --zone=internal --add-service=http
```

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

Verification

- Check that the **internal** zone is active and that the service is allowed in it:

```
# firewall-cmd --zone=internal --list-all
internal (active)
  target: default
  icmp-block-inversion: no
  interfaces:
  sources: 192.0.2.0/24
  services: cockpit dhcpv6-client mdns samba-client ssh http
  ...
```

Additional resources

- **firewalld.zones(5)** man page

1.8. FILTERING FORWARDED TRAFFIC BETWEEN ZONES

With a policy object, users can group different identities that require similar permissions in the policy. You can apply policies depending on the direction of the traffic.

The policy objects feature provides forward and output filtering in firewalld. The following describes the usage of firewalld to filter traffic between different zones to allow access to locally hosted VMs to connect the host.

1.8.1. The relationship between policy objects and zones

Policy objects allow the user to attach firewalld's primitives' such as services, ports, and rich rules to the policy. You can apply the policy objects to traffic that passes between zones in a stateful and unidirectional manner.

```
# firewall-cmd --permanent --new-policy myOutputPolicy

# firewall-cmd --permanent --policy myOutputPolicy --add-ingress-zone HOST

# firewall-cmd --permanent --policy myOutputPolicy --add-egress-zone ANY
```

HOST and **ANY** are the symbolic zones used in the ingress and egress zone lists.

- The **HOST** symbolic zone allows policies for the traffic originating from or has a destination to the host running firewalld.

- The **ANY** symbolic zone applies policy to all the current and future zones. **ANY** symbolic zone acts as a wildcard for all zones.

1.8.2. Using priorities to sort policies

Multiple policies can apply to the same set of traffic, therefore, priorities should be used to create an order of precedence for the policies that may be applied.

To set a priority to sort the policies:

```
# firewall-cmd --permanent --policy mypolicy --set-priority -500
```

In the above example *-500* is a lower priority value but has higher precedence. Thus, -500 will execute before -100. Higher priority values have precedence over lower values.

The following rules apply to policy priorities:

- Policies with negative priorities apply before rules in zones.
- Policies with positive priorities apply after rules in zones.
- Priority 0 is reserved and hence is unusable.

1.8.3. Using policy objects to filter traffic between locally hosted Containers and a network physically connected to the host

The policy objects feature allows users to filter their container and virtual machine traffic.

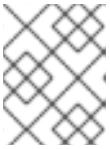
Procedure

1. Create a new policy.

```
# firewall-cmd --permanent --new-policy podmanToHost
```

2. Block all traffic.

```
# firewall-cmd --permanent --policy podmanToHost --set-target REJECT
# firewall-cmd --permanent --policy podmanToHost --add-service dhcp
# firewall-cmd --permanent --policy podmanToHost --add-service dns
```



NOTE

Red Hat recommends that you block all traffic to the host by default and then selectively open the services you need for the host.

3. Define the ingress zone to use with the policy.

```
# firewall-cmd --permanent --policy podmanToHost --add-ingress-zone podman
```

4. Define the egress zone to use with the policy.

```
# firewall-cmd --permanent --policy podmanToHost --add-egress-zone ANY
```

Verification

- Verify information about the policy.

```
# firewall-cmd --info-policy podmanToHost
```

1.8.4. Setting the default target of policy objects

You can specify --set-target options for policies. The following targets are available:

- **ACCEPT** - accepts the packet
- **DROP** - drops the unwanted packets
- **REJECT** - rejects unwanted packets with an ICMP reply
- **CONTINUE (default)** - packets will be subject to rules in following policies and zones.

```
# firewall-cmd --permanent --policy mypolicy --set-target CONTINUE
```

Verification

- Verify information about the policy

```
# firewall-cmd --info-policy mypolicy
```

1.9. CONFIGURING NAT USING FIREWALLD

With **firewalld**, you can configure the following network address translation (NAT) types:

- Masquerading
- Source NAT (SNAT)
- Destination NAT (DNAT)
- Redirect

1.9.1. The different NAT types: masquerading, source NAT, destination NAT, and redirect

These are the different network address translation (NAT) types:

Masquerading and source NAT (SNAT)

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers do not route private IP ranges, such as **10.0.0.0/8**. If you use private IP ranges in your network and users should be able to reach servers on the Internet, map the source IP address of packets from these ranges to a public IP address.
Both masquerading and SNAT are very similar. The differences are:

- Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.
- SNAT sets the source IP address of packets to a specified IP and does not dynamically look up the IP of the outgoing interface. Therefore, SNAT is faster than masquerading. Use SNAT if the outgoing interface uses a fixed IP address.

Destination NAT (DNAT)

Use this NAT type to rewrite the destination address and port of incoming packets. For example, if your web server uses an IP address from a private IP range and is, therefore, not directly accessible from the Internet, you can set a DNAT rule on the router to redirect incoming traffic to this server.

Redirect

This type is a special case of DNAT that redirects packets to the local machine depending on the chain hook. For example, if a service runs on a different port than its standard port, you can redirect incoming traffic from the standard port to this specific port.

1.9.2. Configuring IP address masquerading

The following procedure describes how to enable IP masquerading on your system. IP masquerading hides individual machines behind a gateway when accessing the Internet.

Procedure

1. To check if IP masquerading is enabled (for example, for the **external** zone), enter the following command as **root**:

```
# firewall-cmd --zone=external --query-masquerade
```

   The command prints **yes** with exit status **0** if enabled. It prints **no** with exit status **1** otherwise. If **zone** is omitted, the default zone will be used.

2. To enable IP masquerading, enter the following command as **root**:

```
# firewall-cmd --zone=external --add-masquerade
```

3. To make this setting persistent, pass the **--permanent** option to the command.
4. To disable IP masquerading, enter the following command as **root**:

```
# firewall-cmd --zone=external --remove-masquerade
```

   To make this setting permanent, pass the **--permanent** option to the command.

1.10. PORT FORWARDING

Redirecting ports using this method only works for IPv4-based traffic. For IPv6 redirecting setup, you must use rich rules.

To redirect to an external system, it is necessary to enable masquerading. For more information, see Configuring IP address masquerading.

NOTE

You cannot access a service through a redirected port from the host on which you have configured local forwarding.

1.10.1. Adding a port to redirect

Using **firewalld**, you can set up port redirection so that any incoming traffic that reaches a certain port on your system is delivered to another internal port of your choice or to an external port on another machine.

Prerequisites

- Before you redirect traffic from one port to another port, or another address, you have to know three things: which port the packets arrive at, what protocol is used, and where you want to redirect them.

Procedure

1. To redirect a port to another port:

```
# firewall-cmd --add-forward-port=port=port-number:proto=tcp|udp|sctp|dccp:toport=port-number
```

2. To redirect a port to another port at a different IP address:

   a. Add the port to be forwarded:

```
# firewall-cmd --add-forward-port=port=port-number:proto=tcp|udp:toport=port-number:toaddr=IP
```

   b. Enable masquerade:

```
# firewall-cmd --add-masquerade
```

1.10.2. Redirecting TCP port 80 to port 88 on the same machine

Follow the steps to redirect the TCP port 80 to port 88.

Procedure

1. Redirect the port 80 to port 88 for TCP traffic:

```
# firewall-cmd --add-forward-port=port=80:proto=tcp:toport=88
```

2. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

3. Check that the port is redirected:

```
# firewall-cmd --list-all
```

1.10.3. Removing a redirected port

This procedure describes how to remove the redirected port.

Procedure

1. To remove a redirected port:

```
# firewall-cmd --remove-forward-port=port=port-number:proto=<tcp|udp>:toport=port-number:toaddr=<IP>
```

2. To remove a forwarded port redirected to a different address:

   a. Remove the forwarded port:

```
# firewall-cmd --remove-forward-port=port=port-number:proto=<tcp|udp>:toport=port-number:toaddr=<IP>
```

   b. Disable masquerade:

```
# firewall-cmd --remove-masquerade
```

1.10.4. Removing TCP port 80 forwarded to port 88 on the same machine

This procedure describes how to remove the port redirection.

Procedure

1. List redirected ports:

```
~]# firewall-cmd --list-forward-ports
port=80:proto=tcp:toport=88:toaddr=
```

2. Remove the redirected port from the firewall::

```
~]# firewall-cmd  --remove-forward-port=port=80:proto=tcp:toport=88:toaddr=
```

3. Make the new settings persistent:

```
~]# firewall-cmd --runtime-to-permanent
```

1.11. MANAGING ICMP REQUESTS

The **Internet Control Message Protocol** (**ICMP**) is a supporting protocol that is used by various network devices to send error messages and operational information indicating a connection problem, for example, that a requested service is not available. **ICMP** differs from transport protocols such as TCP and UDP because it is not used to exchange data between systems.

Unfortunately, it is possible to use the **ICMP** messages, especially **echo-request** and **echo-reply**, to reveal information about your network and misuse such information for various kinds of fraudulent activities. Therefore, **firewalld** enables blocking the **ICMP** requests to protect your network information.

1.11.1. Listing and blocking ICMP requests

Listing ICMP requests

The **ICMP** requests are described in individual XML files that are located in the **/usr/lib/firewalld/icmptypes/** directory. You can read these files to see a description of the request. The **firewall-cmd** command controls the **ICMP** requests manipulation.

- To list all available **ICMP** types:

```
# firewall-cmd --get-icmptypes
```

- The **ICMP** request can be used by IPv4, IPv6, or by both protocols. To see for which protocol the **ICMP** request has used:

```
# firewall-cmd --info-icmptype=<icmptype>
```

- The status of an **ICMP** request shows **yes** if the request is currently blocked or **no** if it is not. To see if an **ICMP** request is currently blocked:

```
# firewall-cmd --query-icmp-block=<icmptype>
```

Blocking or unblocking ICMP requests

When your server blocks **ICMP** requests, it does not provide the information that it normally would. However, that does not mean that no information is given at all. The clients receive information that the particular **ICMP** request is being blocked (rejected). Blocking the **ICMP** requests should be considered carefully, because it can cause communication problems, especially with IPv6 traffic.

- To see if an **ICMP** request is currently blocked:

```
# firewall-cmd --query-icmp-block=<icmptype>
```

- To block an **ICMP** request:

```
# firewall-cmd --add-icmp-block=<icmptype>
```

- To remove the block for an **ICMP** request:

```
# firewall-cmd --remove-icmp-block=<icmptype>
```

Blocking ICMP requests without providing any information at all

Normally, if you block **ICMP** requests, clients know that you are blocking it. So, a potential attacker who is sniffing for live IP addresses is still able to see that your IP address is online. To hide this information completely, you have to drop all **ICMP** requests.

- To block and drop all **ICMP** requests:
- Set the target of your zone to **DROP**:

```
# firewall-cmd --permanent --set-target=DROP
```

Now, all traffic, including **ICMP** requests, is dropped, except traffic which you have explicitly allowed.

To block and drop certain **ICMP** requests and allow others:

1. Set the target of your zone to **DROP**:

```
# firewall-cmd --permanent --set-target=DROP
```

2. Add the ICMP block inversion to block all **ICMP** requests at once:

```
# firewall-cmd --add-icmp-block-inversion
```

3. Add the ICMP block for those **ICMP** requests that you want to allow:

```
# firewall-cmd --add-icmp-block=<icmptype>
```

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

The *block inversion* inverts the setting of the **ICMP** requests blocks, so all requests, that were not previously blocked, are blocked because of the target of your zone changes to **DROP**. The requests that were blocked are not blocked. This means that if you want to unblock a request, you must use the blocking command.

To revert the block inversion to a fully permissive setting:

1. Set the target of your zone to **default** or **ACCEPT**:

```
# firewall-cmd --permanent --set-target=default
```

2. Remove all added blocks for **ICMP** requests:

```
# firewall-cmd --remove-icmp-block=<icmptype>
```

3. Remove the **ICMP** block inversion:

```
# firewall-cmd --remove-icmp-block-inversion
```

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

1.11.2. Configuring the ICMP filter using GUI

- To enable or disable an **ICMP** filter, start the **firewall-config** tool and select the network zone whose messages are to be filtered. Select the **ICMP Filter** tab and select the check box for each type of **ICMP** message you want to filter. Clear the check box to disable a filter. This setting is per direction and the default allows everything.

- To enable inverting the **ICMP Filter**, click the **Invert Filter** check box on the right. Only marked **ICMP** types are now accepted, all other are rejected. In a zone using the DROP target, they are dropped.

1.12. SETTING AND CONTROLLING IP SETS USING FIREWALLD

To see the list of IP set types supported by **firewalld**, enter the following command as root.

```
~]# firewall-cmd --get-ipset-types
hash:ip hash:ip,mark hash:ip,port hash:ip,port,ip hash:ip,port,net hash:mac hash:net hash:net,iface
hash:net,net hash:net,port hash:net,port,net
```

1.12.1. Configuring IP set options using CLI

IP sets can be used in **firewalld** zones as sources and also as sources in rich rules. In Red Hat Enterprise Linux, the preferred method is to use the IP sets created with **firewalld** in a direct rule.

- To list the IP sets known to **firewalld** in the permanent environment, use the following command as **root**:

  ```
  # firewall-cmd --permanent --get-ipsets
  ```

- To add a new IP set, use the following command using the permanent environment as **root**:

  ```
  # firewall-cmd --permanent --new-ipset=test --type=hash:net
  success
  ```

  The previous command creates a new IP set with the name *test* and the **hash:net** type for **IPv4**. To create an IP set for use with **IPv6**, add the **--option=family=inet6** option. To make the new setting effective in the runtime environment, reload **firewalld**.

- List the new IP set with the following command as **root**:

  ```
  # firewall-cmd --permanent --get-ipsets
  test
  ```

- To get more information about the IP set, use the following command as **root**:

  ```
  # firewall-cmd --permanent --info-ipset=test
  test
  type: hash:net
  options:
  entries:
  ```

  Note that the IP set does not have any entries at the moment.

- To add an entry to the *test* IP set, use the following command as **root**:

  ```
  # firewall-cmd --permanent --ipset=test --add-entry=192.168.0.1
  success
  ```

  The previous command adds the IP address *192.168.0.1* to the IP set.

- To get the list of current entries in the IP set, use the following command as **root**:

  ```
  # firewall-cmd --permanent --ipset=test --get-entries
  192.168.0.1
  ```

- Generate a file containing a list of IP addresses, for example:

```
# cat > iplist.txt <<EOL
192.168.0.2
192.168.0.3
192.168.1.0/24
192.168.2.254
EOL
```

The file with the list of IP addresses for an IP set should contain an entry per line. Lines starting with a hash, a semi-colon, or empty lines are ignored.

- To add the addresses from the *iplist.txt* file, use the following command as **root**:

```
# firewall-cmd --permanent --ipset=test --add-entries-from-file=iplist.txt
success
```

- To see the extended entries list of the IP set, use the following command as **root**:

```
# firewall-cmd --permanent --ipset=test --get-entries
192.168.0.1
192.168.0.2
192.168.0.3
192.168.1.0/24
192.168.2.254
```

- To remove the addresses from the IP set and to check the updated entries list, use the following commands as **root**:

```
# firewall-cmd --permanent --ipset=test --remove-entries-from-file=iplist.txt
success
# firewall-cmd --permanent --ipset=test --get-entries
192.168.0.1
```

- You can add the IP set as a source to a zone to handle all traffic coming in from any of the addresses listed in the IP set with a zone. For example, to add the *test* IP set as a source to the *drop* zone to drop all packets coming from all entries listed in the *test* IP set, use the following command as **root**:

```
# firewall-cmd --permanent --zone=drop --add-source=ipset:test
success
```

The **ipset:** prefix in the source shows **firewalld** that the source is an IP set and not an IP address or an address range.

Only the creation and removal of IP sets is limited to the permanent environment, all other IP set options can be used also in the runtime environment without the **--permanent** option.

WARNING

Red Hat does not recommend using IP sets that are not managed through **firewalld**. To use such IP sets, a permanent direct rule is required to reference the set, and a custom service must be added to create these IP sets. This service needs to be started before **firewalld** starts, otherwise **firewalld** is not able to add the direct rules using these sets. You can add permanent direct rules with the **/etc/firewalld/direct.xml** file.

1.13. PRIORITIZING RICH RULES

By default, rich rules are organized based on their rule action. For example, **deny** rules have precedence over **allow** rules. The **priority** parameter in rich rules provides administrators fine-grained control over rich rules and their execution order.

1.13.1. How the priority parameter organizes rules into different chains

You can set the **priority** parameter in a rich rule to any number between **-32768** and **32767**, and lower values have higher precedence.

The **firewalld** service organizes rules based on their priority value into different chains:

- Priority lower than 0: the rule is redirected into a chain with the **_pre** suffix.
- Priority higher than 0: the rule is redirected into a chain with the **_post** suffix.
- Priority equals 0: based on the action, the rule is redirected into a chain with the **_log**, **_deny**, or **_allow** the action.

Inside these sub-chains, **firewalld** sorts the rules based on their priority value.

1.13.2. Setting the priority of a rich rule

The procedure describes an example of how to create a rich rule that uses the **priority** parameter to log all traffic that is not allowed or denied by other rules. You can use this rule to flag unexpected traffic.

Procedure

1. Add a rich rule with a very low precedence to log all traffic that has not been matched by other rules:

```
# firewall-cmd --add-rich-rule='rule priority=32767 log prefix="UNEXPECTED: " limit value="5/m"'
```

   The command additionally limits the number of log entries to **5** per minute.

2. Optionally, display the **nftables** rule that the command in the previous step created:

```
# nft list chain inet firewalld filter_IN_public_post
table inet firewalld {
```

```
  chain filter_IN_public_post {
    log prefix "UNEXPECTED: " limit rate 5/minute
  }
}
```

1.14. CONFIGURING FIREWALL LOCKDOWN

Local applications or services are able to change the firewall configuration if they are running as **root** (for example, **libvirt**). With this feature, the administrator can lock the firewall configuration so that either no applications or only applications that are added to the lockdown allow list are able to request firewall changes. The lockdown settings default to disabled. If enabled, the user can be sure that there are no unwanted configuration changes made to the firewall by local applications or services.

1.14.1. Configuring lockdown using CLI

This procedure describes how to enable or disable lockdown using the command line.

- To query whether lockdown is enabled, use the following command as **root**:

  ```
  # firewall-cmd --query-lockdown
  ```

  The command prints **yes** with exit status **0** if lockdown is enabled. It prints **no** with exit status **1** otherwise.

- To enable lockdown, enter the following command as **root**:

  ```
  # firewall-cmd --lockdown-on
  ```

- To disable lockdown, use the following command as **root**:

  ```
  # firewall-cmd --lockdown-off
  ```

1.14.2. Configuring lockdown allowlist options using CLI

The lockdown allowlist can contain commands, security contexts, users and user IDs. If a command entry on the allowlist ends with an asterisk "*", then all command lines starting with that command will match. If the "*" is not there then the absolute command including arguments must match.

- The context is the security (SELinux) context of a running application or service. To get the context of a running application use the following command:

  ```
  $ ps -e --context
  ```

  That command returns all running applications. Pipe the output through the **grep** tool to get the application of interest. For example:

  ```
  $ ps -e --context | grep example_program
  ```

- To list all command lines that are in the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --list-lockdown-whitelist-commands
  ```

- To add a command *command* to the allowlist, enter the following command as **root**:

```
# firewall-cmd --add-lockdown-whitelist-command='/usr/bin/python3 -Es /usr/bin/command'
```

- To remove a command *command* from the allowlist, enter the following command as **root**:

```
# firewall-cmd --remove-lockdown-whitelist-command='/usr/bin/python3 -Es
/usr/bin/command'
```

- To query whether the command *command* is in the allowlist, enter the following command as **root**:

```
# firewall-cmd --query-lockdown-whitelist-command='/usr/bin/python3 -Es /usr/bin/command'
```

The command prints **yes** with exit status **0** if true. It prints **no** with exit status **1** otherwise.

- To list all security contexts that are in the allowlist, enter the following command as **root**:

```
# firewall-cmd --list-lockdown-whitelist-contexts
```

- To add a context *context* to the allowlist, enter the following command as **root**:

```
# firewall-cmd --add-lockdown-whitelist-context=context
```

- To remove a context *context* from the allowlist, enter the following command as **root**:

```
# firewall-cmd --remove-lockdown-whitelist-context=context
```

- To query whether the context *context* is in the allowlist, enter the following command as **root**:

```
# firewall-cmd --query-lockdown-whitelist-context=context
```

Prints **yes** with exit status **0**, if true, prints **no** with exit status **1** otherwise.

- To list all user IDs that are in the allowlist, enter the following command as **root**:

```
# firewall-cmd --list-lockdown-whitelist-uids
```

- To add a user ID *uid* to the allowlist, enter the following command as **root**:

```
# firewall-cmd --add-lockdown-whitelist-uid=uid
```

- To remove a user ID *uid* from the allowlist, enter the following command as **root**:

```
# firewall-cmd --remove-lockdown-whitelist-uid=uid
```

- To query whether the user ID *uid* is in the allowlist, enter the following command:

```
$ firewall-cmd --query-lockdown-whitelist-uid=uid
```

Prints **yes** with exit status **0**, if true, prints **no** with exit status **1** otherwise.

- To list all user names that are in the allowlist, enter the following command as **root**:

```
# firewall-cmd --list-lockdown-whitelist-users
```

- To add a user name *user* to the allowlist, enter the following command as **root**:

```
# firewall-cmd --add-lockdown-whitelist-user=user
```

- To remove a user name *user* from the allowlist, enter the following command as **root**:

```
# firewall-cmd --remove-lockdown-whitelist-user=user
```

- To query whether the user name *user* is in the allowlist, enter the following command:

```
$ firewall-cmd --query-lockdown-whitelist-user=user
```

  Prints **yes** with exit status **0**, if true, prints **no** with exit status **1** otherwise.

1.14.3. Configuring lockdown allowlist options using configuration files

The default allowlist configuration file contains the **NetworkManager** context and the default context of **libvirt**. The user ID 0 is also on the list.

+ The allowlist configuration files are stored in the /**etc**/**firewalld**/ directory.

```
<?xml version="1.0" encoding="utf-8"?>
 <whitelist>
   <selinux context="system_u:system_r:NetworkManager_t:s0"/>
   <selinux context="system_u:system_r:virtd_t:s0-s0:c0.c1023"/>
   <user id="0"/>
 </whitelist>
```

Following is an example allowlist configuration file enabling all commands for the **firewall-cmd** utility, for a user called *user* whose user ID is **815**:

```
<?xml version="1.0" encoding="utf-8"?>
 <whitelist>
   <command name="/usr/libexec/platform-python -s /bin/firewall-cmd*"/>
   <selinux context="system_u:system_r:NetworkManager_t:s0"/>
   <user id="815"/>
   <user name="user"/>
 </whitelist>
```

This example shows both **user id** and **user name**, but only one option is required. Python is the interpreter and is prepended to the command line. You can also use a specific command, for example:

```
/usr/bin/python3 /bin/firewall-cmd --lockdown-on
```

In that example, only the **--lockdown-on** command is allowed.

In Red Hat Enterprise Linux, all utilities are placed in the /**usr**/**bin**/ directory and the /**bin**/ directory is sym-linked to the /**usr**/**bin**/ directory. In other words, although the path for **firewall-cmd** when entered as **root** might resolve to /**bin**/**firewall-cmd**, /**usr**/**bin**/**firewall-cmd** can now be used. All new scripts

should use the new location. But be aware that if scripts that run as **root** are written to use the **/bin/firewall-cmd** path, then that command path must be added in the allowlist in addition to the **/usr/bin/firewall-cmd** path traditionally used only for non- **root** users.

The * at the end of the name attribute of a command means that all commands that start with this string match. If the * is not there then the absolute command including arguments must match.

1.15. ENABLING TRAFFIC FORWARDING BETWEEN DIFFERENT INTERFACES OR SOURCES WITHIN A FIREWALLD ZONE

Intra-zone forwarding is a **firewalld** feature that enables traffic forwarding between interfaces or sources within a **firewalld** zone.

1.15.1. The difference between intra-zone forwarding and zones with the default target set to ACCEPT

When intra-zone forwarding is enabled, the traffic within a single **firewalld** zone can flow from one interface or source to another interface or source. The zone specifies the trust level of interfaces and sources. If the trust level is the same, communication between interfaces or sources is possible.

Note that, if you enable intra-zone forwarding in the default zone of **firewalld**, it applies only to the interfaces and sources added to the current default zone.

The **trusted** zone of **firewalld** uses a default target set to **ACCEPT**. This zone accepts all forwarded traffic, and intra-zone forwarding is not applicable for it.

As for other default target values, forwarded traffic is dropped by default, which applies to all standard zones except the trusted zone.

1.15.2. Using intra-zone forwarding to forward traffic between an Ethernet and Wi-Fi network

You can use intra-zone forwarding to forward traffic between interfaces and sources within the same **firewalld** zone. For example, use this feature to forward traffic between an Ethernet network connected to **enp1s0** and a Wi-Fi network connected to **wlp0s20**.

Procedure

1. Enable packet forwarding in the kernel:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf

# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

2. Ensure that interfaces between which you want to enable intra-zone forwarding are not assigned to a zone different than the **internal** zone:

```
# firewall-cmd --get-active-zones
```

3. If the interface is currently assigned to a zone other than **internal**, reassign it:

```
# firewall-cmd --zone=internal --change-interface=interface_name --permanent
```

4. Add the **enp1s0** and **wlp0s20** interfaces to the **internal** zone:

```
# firewall-cmd --zone=internal --add-interface=enp1s0 --add-interface=wlp0s20
```

5. Enable intra-zone forwarding:

```
# firewall-cmd --zone=internal --add-forward
```

Verification

The following verification steps require that the **nmap-ncat** package is installed on both hosts.

1. Log in to a host that is in the same network as the **enp1s0** interface of the host you enabled zone forwarding on.
2. Start an echo service with **ncat** to test connectivity:

```
# ncat -e /usr/bin/cat -l 12345
```

3. Log in to a host that is in the same network as the **wlp0s20** interface.
4. Connect to the echo server running on the host that is in the same network as the **enp1s0**:

```
# ncat <other host> 12345
```

5. Type something and press **Enter**, and verify the text is sent back.

Additional resources

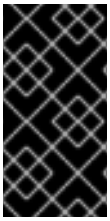
- **firewalld.zones(5)** man page

1.16. USING RHEL SYSTEM ROLES WITH ANSIBLE TO CONFIGURE FIREWALLD SETTINGS

You can use the Ansible firewall System Role to configure settings of the **firewalld** service on multiple clients at once. This solution:

- Provides an interface with efficient input settings.
- Keeps all intended **firewalld** parameters in one place.

After you run the **firewall** role on the control node, the System Role applies the **firewalld** parameters to the managed node immediately and makes them persistent across reboots.



IMPORTANT

Note that RHEL System Roles delivered over RHEL channels are available to RHEL customers as an RPM package in the default **AppStream** repository. RHEL System Roles are also available as a collection to customers with Ansible subscriptions over Ansible Automation Hub.

1.16.1. Introduction to the firewall RHEL System Role

RHEL System Roles is a set of contents for the Ansible automation utility. This content together with the Ansible automation utility provides a consistent configuration interface to remotely manage multiple systems.

The **rhel-system-roles.firewall** role from the RHEL System Roles was introduced for automated configurations of the **firewalld** service. The **rhel-system-roles** package contains this system role, and also the reference documentation.

To apply the **firewalld** parameters on one or more systems in an automated fashion, use the **firewall** System Role variable in a playbook. A playbook is a list of one or more plays that is written in the text-based YAML format.

You can use an inventory file to define a set of systems that you want Ansible to configure.

With the **firewall** role you can configure many different **firewalld** parameters, for example:

- Zones.
- The services for which packets should be allowed.
- Granting, rejection, or dropping of traffic access to ports.
- Forwarding of ports or port ranges for a zone.

Additional resources

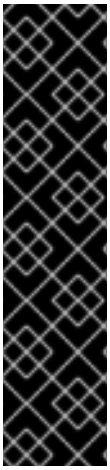
- **README.md** and **README.html** files in the **/usr/share/doc/rhel-system-roles/firewall/** directory
- Working with playbooks
- How to build your inventory

1.16.2. Forwarding incoming traffic from one local port to a different local port

With the **rhel-system-roles.firewall** role you can remotely configure **firewalld** parameters with persisting effect on multiple managed hosts.

Prerequisites

- Entitled by your RHEL subscription, you installed the **ansible-core** and **rhel-system-roles** packages on the control node.
- An inventory of managed hosts is present on the control machine and Ansible is able to connect to them.
- You have permission to run Ansible playbooks on the managed hosts.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed host.
- The inventory file lists the hosts on which the playbook should perform the actions. The playbook in this procedure runs on hosts in the group **testinservers**.



IMPORTANT

RHEL 8.0 - 8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**; connectors such as **docker** and **podman**; and the entire world of plugins and modules. For information on how to obtain and install Ansible Engine, refer to How do I Download and Install Red Hat Ansible Engine? .

RHEL 8.6 and later has introduced Ansible Core (provided as **ansible-core** RPM), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. The AppStream repository provides **ansible-core**, which has a limited scope of support. You can learn more by reviewing Scope of support for the ansible-core package included in the RHEL 9 AppStream.

Procedure

1. Create the ~/**port_forwarding.yml** file and add the following content:

```
---
- name: Forward incoming traffic on port 8080 to 443
  hosts: testingservers

  tasks:
    - include_role:
        name: rhel-system-roles.firewall

  vars:
    firewall:
      - { forward_port: 8080/tcp;443;, state: enabled, runtime: true, permanent: true }
```

This file represents a playbook and usually contains an ordered list of tasks, also called *plays*, that are run against specific managed hosts selected from your **inventory** file. In this case, the playbook will run against the **testingservers** group of managed hosts.

The **hosts** key in the play specifies the hosts against which the play is run. You can provide the value or values for this key as individual names of managed hosts or as groups of hosts as defined in the **inventory** file.

The **tasks** section has the **include_role** key, which specifies what system role is going to configure the parameters and values mentioned in the **vars** section.

The **vars** section contains a role variable called **firewall**. This variable is a list of dictionary values and specifies parameters that will be applied to **firewalld** on managed hosts. The example role will forward the traffic coming to port 8080 to port 443. The settings will come to effect immediately and will also persist across reboots.

2. Optionally, verify that the syntax in the playbook is correct:

```
# ansible-playbook --syntax-check ~/port_forwarding.yml

playbook: port_forwarding.yml
```

This example shows the successful verification of a playbook.

3. Execute the playbook:

```
# ansible-playbook ~/port_forwarding.yml
```

Verification

- On the managed host:
  - Restart the host to verify if the **firewalld** settings are still in place after a reboot:

    ```
    # reboot
    ```

  - Display the **firewalld** settings:

    ```
    # firewall-cmd --list-forward-ports
    ```

Additional resources

- Getting started with RHEL System Roles
- **README.html** and **README.md** files in the **/usr/share/doc/rhel-system-roles/firewall/** directory
- Build Your Inventory
- Configuring Ansible
- Working With Playbooks
- Using Variables
- Roles

1.16.3. Configuring ports using System Roles

You can use the Red Hat Enterprise Linux (RHEL) **firewalld** System Role to open or close ports in the local firewall for incoming traffic and make the new configuration persist across reboots. The example describes how to configure the default zone to permit incoming traffic for the **HTTPS** service.

Run this procedure on the Ansible control node.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **firewalld** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
- The **ansible-core** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.
- The host uses NetworkManager to configure the network.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

```
node.example.com
```

2. Create the **~/adding-and-removing-ports.yml** playbook with the following content:

```
---
- name: Allow incoming HTTPS traffic to the local host
  hosts: node.example.com
  become: true

  tasks:
    - include_role:
        name: linux-system-roles.firewall

      vars:
        firewall:
          - port: 443/tcp
            service: http
            state: enabled
            runtime: true
            permanent: true
```

The **permanent: true** option makes the new settings persistent across reboots.

3. Run the playbook:

- To connect as **root** user to the managed host, enter:

```
# ansible-playbook -u root ~/adding-and-removing-ports.yml
```

- To connect as a user to the managed host, enter:

```
# ansible-playbook -u user_name --ask-become-pass ~/adding-and-removing-ports.yml
```

The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Verification

1. Connect to the managed node:

```
$ ssh user_name@node.example.com
```

2. Verify that the **443/tcp** port associated with the **HTTPS** service is open:

```
$ sudo firewall-cmd --list-ports
443/tcp
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**
- **ansible-playbook(1)** man page

1.16.4. Configuring a DMZ firewalld zone by using the firewalld RHEL System Role

As a system administrator, you can use the RHEL **firewalld** System Role to configure a **dmz** zone on the **enp1s0** interface to permit **HTTPS** traffic to the zone. In this way, you enable external users to access your web servers.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the VPN System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
- An inventory file that lists the managed nodes.
- The **ansible-core** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.
- The managed nodes use **NetworkManager** to configure the network.

Procedure

1. Create the **~/configuring-a-dmz-using-the-firewall-system-role.yml** playbook with the following content:

```
---
- name: Creating a DMZ with access to HTTPS port and masquerading for hosts in DMZ
  hosts: node.example.com
  become: true

  tasks:
    - include_role:
        name: linux-system-roles.firewall

      vars:
        firewall:
          - zone: dmz
            interface: enp1s0
            service: https
            state: enabled
            runtime: true
            permanent: true
```

2. Run the playbook:

   - To connect as **root** user to the managed host, enter:

     ```
     $ ansible-playbook -u root ~/configuring-a-dmz-using-the-firewall-system-role.yml
     ```

   - To connect as a user to the managed host, enter:

     ```
     $ ansible-playbook -u user_name --ask-become-pass ~/configuring-a-dmz-using-the-firewall-system-role.yml
     ```

     The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

   If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Verification

- On the managed node, view detailed information about the **dmz** zone:

```
# firewall-cmd --zone=dmz --list-all
dmz (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0
  sources:
  services: https ssh
  ports:
  protocols:
  forward: no
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
```

1.17. ADDITIONAL RESOURCES

- **firewalld(1)** man page
- **firewalld.conf(5)** man page
- **firewall-cmd(1)** man page
- **firewall-config(1)** man page
- **firewall-offline-cmd(1)** man page
- **firewalld.icmptype(5)** man page
- **firewalld.ipset(5)** man page
- **firewalld.service(5)** man page
- **firewalld.zone(5)** man page

- **firewalld.direct(5)** man page
- **firewalld.lockdown-whitelist(5)**
- **firewalld.richlanguage(5)**
- **firewalld.zones(5)** man page
- **firewalld.dbus(5)** man page

CHAPTER 2. GETTING STARTED WITH NFTABLES

The **nftables** framework provides packet classification facilities. The most notable features are:

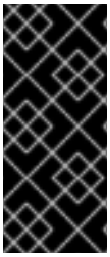
- built-in lookup tables instead of linear processing
- a single framework for both the **IPv4** and **IPv6** protocols
- rules all applied atomically instead of fetching, updating, and storing a complete rule set
- support for debugging and tracing in the rule set (**nftrace**) and monitoring trace events (in the **nft** tool)
- more consistent and compact syntax, no protocol-specific extensions
- a Netlink API for third-party applications

The **nftables** framework uses tables to store chains. The chains contain individual rules for performing actions. The **libnftnl** library can be used for low-level interaction with **nftables** Netlink API over the **libmnl** library.

To display the effect of rule set changes, use the **nft list ruleset** command. Since these tools add tables, chains, rules, sets, and other objects to the **nftables** rule set, be aware that **nftables** rule-set operations, such as the **nft flush ruleset** command, might affect rule sets installed using the formerly separate legacy commands.

2.1. MIGRATING FROM IPTABLES TO NFTABLES

If your firewall configuration still uses **iptables** rules, you can migrate your **iptables** rules to **nftables**.



IMPORTANT

The **ipset** and **iptables-nft** packages have been deprecated in Red Hat Enterprise Linux 9. This includes deprecation of **nft-variants** such as **iptables**, **ip6tables**, **arptables**, and **ebtables** utilities. If you are using any of these tools, for example, because you upgraded from an earlier RHEL version, Red Hat recommends migrating to the **nft** command line tool provided by the **nftables** package.

2.1.1. When to use firewalld, nftables, or iptables

The following is a brief overview in which scenario you should use one of the following utilities:

- **firewalld**: Use the **firewalld** utility for simple firewall use cases. The utility is easy to use and covers the typical use cases for these scenarios.
- **nftables**: Use the **nftables** utility to set up complex and performance critical firewalls, such as for a whole network.
- **iptables**: The **iptables** utility on Red Hat Enterprise Linux uses the **nf_tables** kernel API instead of the **legacy** back end. The **nf_tables** API provides backward compatibility so that scripts that use **iptables** commands still work on Red Hat Enterprise Linux. For new firewall scripts, Red Hat recommends to use **nftables**.

IMPORTANT

To avoid that the different firewall services influence each other, run only one of them on a RHEL host, and disable the other services.

2.1.2. Converting iptables rules to nftables rules

Red Hat Enterprise Linux provides the **iptables-translate** and **ip6tables-translate** tools to convert existing **iptables** or **ip6tables** rules into the equivalent ones for **nftables**.

Note that some extensions lack translation support. If such an extension exists, the tool prints the untranslated rule prefixed with the **#** sign. For example:

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

Additionally, users can use the **iptables-restore-translate** and **ip6tables-restore-translate** tools to translate a dump of rules. Note that before that, users can use the **iptables-save** or **ip6tables-save** commands to print a dump of current rules. For example:

```
# iptables-save >/tmp/iptables.dump
# iptables-restore-translate -f /tmp/iptables.dump

# Translated by iptables-restore-translate v1.8.0 on Wed Oct 17 17:00:13 2018
add table ip nat
...
```

For more information and a list of possible options and values, enter the **iptables-translate --help** command.

2.1.3. Comparison of common iptables and nftables commands

The following is a comparison of common **iptables** and **nftables** commands:

- Listing all rules:

iptables	nftables
iptables-save	**nft list ruleset**

- Listing a certain table and chain:

iptables	nftables
iptables -L	**nft list table ip filter**
iptables -L INPUT	**nft list chain ip filter INPUT**
iptables -t nat -L PREROUTING	**nft list chain ip nat PREROUTING**

The **nft** command does not pre-create tables and chains. They exist only if a user created them manually.

Example: Listing rules generated by firewalld

```
# nft list table inet firewalld
# nft list table ip firewalld
# nft list table ip6 firewalld
```

2.2. WRITING AND EXECUTING NFTABLES SCRIPTS

The **nftables** framework provides a native scripting environment that brings a major benefit over using shell scripts to maintain firewall rules: the execution of scripts is atomic. This means that the system either applies the whole script or prevents the execution if an error occurs. This guarantees that the firewall is always in a consistent state.

Additionally, the **nftables** script environment enables administrators to:

- add comments
- define variables
- include other rule set files

This section explains how to use these features, as well as creating and executing **nftables** scripts.

When you install the **nftables** package, Red Hat Enterprise Linux automatically creates ***.nft** scripts in the **/etc/nftables/** directory. These scripts contain commands that create tables and empty chains for different purposes.

2.2.1. Supported nftables script formats

The **nftables** scripting environment supports scripts in the following formats:

- You can write a script in the same format as the **nft list ruleset** command displays the rule set:

```
#!/usr/sbin/nft -f

# Flush the rule set
flush ruleset

table inet example_table {
  chain example_chain {
    # Chain for incoming packets that drops all packets that
    # are not explicitly allowed by any rule in this chain
    type filter hook input priority 0; policy drop;

    # Accept connections to port 22 (ssh)
    tcp dport ssh accept
  }
}
```

- You can use the same syntax for commands as in **nft** commands:

```
#!/usr/sbin/nft -f

# Flush the rule set
```

```
flush ruleset

# Create a table
add table inet example_table

# Create a chain for incoming packets that drops all packets
# that are not explicitly allowed by any rule in this chain
add chain inet example_table example_chain { type filter hook input priority 0 ; policy drop ; }

# Add a rule that accepts connections to port 22 (ssh)
add rule inet example_table example_chain tcp dport ssh accept
```

2.2.2. Running nftables scripts

You can run **nftables** script either by passing it to the **nft** utility or execute the script directly.

Prerequisites

- The procedure of this section assumes that you stored an **nftables** script in the **/etc/nftables/example_firewall.nft** file.

Procedure

- To run an **nftables** script by passing it to the **nft** utility, enter:

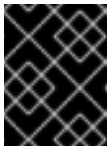
```
# nft -f /etc/nftables/example_firewall.nft
```

- To run an **nftables** script directly:

  a. Steps that are required only once:

     i. Ensure that the script starts with the following shebang sequence:

```
#!/usr/sbin/nft -f
```



IMPORTANT

If you omit the **-f** parameter, the **nft** utility does not read the script and displays: **Error: syntax error, unexpected newline, expecting string**.

     ii. Optional: Set the owner of the script to **root**:

```
# chown root /etc/nftables/example_firewall.nft
```

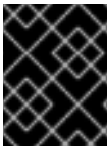
     iii. Make the script executable for the owner:

```
# chmod u+x /etc/nftables/example_firewall.nft
```

  b. Run the script:

```
# /etc/nftables/example_firewall.nft
```

  If no output is displayed, the system executed the script successfully.



IMPORTANT

Even if **nft** executes the script successfully, incorrectly placed rules, missing parameters, or other problems in the script can cause that the firewall behaves not as expected.

Additional resources

- **chown(1)** man page
- **chmod(1)** man page
- Automatically loading nftables rules when the system boots

2.2.3. Using comments in nftables scripts

The **nftables** scripting environment interprets everything to the right of a **#** character as a comment.

Example 2.1. Comments in an nftables script

Comments can start at the beginning of a line, as well as next to a command:

```
...
# Flush the rule set
flush ruleset

add table inet example_table  # Create a table
...
```

2.2.4. Using variables in an nftables script

To define a variable in an **nftables** script, use the **define** keyword. You can store single values and anonymous sets in a variable. For more complex scenarios, use sets or verdict maps.

Variables with a single value

The following example defines a variable named **INET_DEV** with the value **enp1s0**:

```
define INET_DEV = enp1s0
```

You can use the variable in the script by writing the **$** sign followed by the variable name:

```
...
add rule inet example_table example_chain iifname $INET_DEV tcp dport ssh accept
...
```

Variables that contain an anonymous set

The following example defines a variable that contains an anonymous set:

```
define DNS_SERVERS = { 192.0.2.1, 192.0.2.2 }
```

You can use the variable in the script by writing the **$** sign followed by the variable name:

```
add rule inet example_table example_chain ip daddr $DNS_SERVERS accept
```

NOTE

Note that curly braces have special semantics when you use them in a rule because they indicate that the variable represents a set.

Additional resources

- Using sets in nftables commands
- Using verdict maps in nftables commands

2.2.5. Including files in an nftables script

The **nftables** scripting environment enables administrators to include other scripts by using the **include** statement.

If you specify only a file name without an absolute or relative path, **nftables** includes files from the default search path, which is set to **/etc** on Red Hat Enterprise Linux.

Example 2.2. Including files from the default search directory

To include a file from the default search directory:

```
include "example.nft"
```

Example 2.3. Including all *.nft files from a directory

To include all files ending with ***.nft** that are stored in the **/etc/nftables/rulesets/** directory:

```
include "/etc/nftables/rulesets/*.nft"
```

Note that the **include** statement does not match files beginning with a dot.

Additional resources

- The **Include files** section in the **nft(8)** man page

2.2.6. Automatically loading nftables rules when the system boots

The **nftables** systemd service loads firewall scripts that are included in the **/etc/sysconfig/nftables.conf** file. This section explains how to load firewall rules when the system boots.

Prerequisites

- The **nftables** scripts are stored in the **/etc/nftables/** directory.

Procedure

1. Edit the **/etc/sysconfig/nftables.conf** file.
   - If you enhance ***.nft** scripts created in **/etc/nftables/** when you installed the **nftables** package, uncomment the **include** statement for these scripts.
   - If you write scripts from scratch, add **include** statements to include these scripts. For example, to load the **/etc/nftables/*example*.nft** script when the **nftables** service starts, add:

     ```
     include "/etc/nftables/example.nft"
     ```

2. Optionally, start the **nftables** service to load the firewall rules without rebooting the system:

   ```
   # systemctl start nftables
   ```

3. Enable the **nftables** service.

   ```
   # systemctl enable nftables
   ```

Additional resources

- Supported nftables script formats

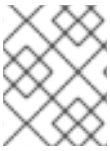
2.3. CREATING AND MANAGING NFTABLES TABLES, CHAINS, AND RULES

This section explains how to display **nftables** rule sets, and how to manage them.

2.3.1. Standard chain priority values and textual names

When you create a chain, the **priority** you can either set an integer value or a standard name that specifies the order in which chains with the same **hook** value traverse.

The names and values are defined based on what priorities are used by **xtables** when registering their default chains.



NOTE

The **nft list chains** command displays textual priority values by default. You can view the numeric value by passing the **-y** option to the command.

Example 2.4. Using a textual value to set the priority

The following command creates a chain named **example_chain** in **example_table** using the standard priority value **50**:

 # nft add chain inet *example_table example_chain* { type *filter* hook *input* priority *50* \; policy accept \; }

Because the priority is a standard value, you can alternatively use the textual value:

```
# nft add chain inet example_table example_chain { type filter hook input priority security \;
policy accept \; }
```

Table 2.1. Standard priority names, family, and hook compatibility matrix

Name	Value	Families	Hooks
raw	-300	**ip**, **ip6**, **inet**	all
mangle	-150	**ip**, **ip6**, **inet**	all
dstnat	-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

All families use the same values, but the **bridge** family uses following values:

Table 2.2. Standard priority names, and hook compatibility for the bridge family

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

Additional resources

- The **Chains** section in the **nft(8)** man page

2.3.2. Displaying the nftables rule set

The rule sets of **nftables** contain tables, chains, and rules. This section explains how to display the rule set.

Procedure

- To display the rule set, enter:

```
# nft list ruleset
table inet example_table {
  chain example_chain {
```

```
        type filter hook input priority filter; policy accept;
        tcp dport http accept
        tcp dport ssh accept
    }
}
```

NOTE

By default, **nftables** does not pre-create tables. As a consequence, displaying the rule set on a host without any tables, the **nft list ruleset** command shows no output.

2.3.3. Creating an nftables table

A table in **nftables** is a name space that contains a collection of chains, rules, sets, and other objects. This section explains how to create a table.

Each table must have an address family defined. The address family of a table defines what address types the table processes. You can set one of the following address families when you create a table:

- **ip**: Matches only IPv4 packets. This is the default if you do not specify an address family.
- **ip6**: Matches only IPv6 packets.
- **inet**: Matches both IPv4 and IPv6 packets.
- **arp**: Matches IPv4 address resolution protocol (ARP) packets.
- **bridge**: Matches packets that traverse a bridge device.
- **netdev**: Matches packets from ingress.

Procedure

1. Use the **nft add table** command to create a new table. For example, to create a table named **example_table** that processes IPv4 and IPv6 packets:

```
# nft add table inet example_table
```

2. Optionally, list all tables in the rule set:

```
# nft list tables
table inet example_table
```

Additional resources

- The **Address families** section in the **nft(8)** man page
- The **Tables** section in the **nft(8)** man page

2.3.4. Creating an nftables chain

Chains are containers for rules. The following two rule types exists:

- Base chain: You can use base chains as an entry point for packets from the networking stack.
- Regular chain: You can use regular chains as a **jump** target and to better organize rules.

The procedure describes how to add a base chain to an existing table.

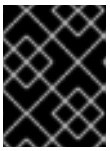
Prerequisites

- The table to which you want to add the new chain exists.

Procedure

1. Use the **nft add chain** command to create a new chain. For example, to create a chain named **example_chain** in **example_table**:

```
# nft add chain inet example_table example_chain { type filter hook input priority 0 \; policy accept \; }
```



IMPORTANT

To avoid that the shell interprets the semicolons as the end of the command, prepend the semicolons the \ escape character.

This chain filters incoming packets. The **priority** parameter specifies the order in which **nftables** processes chains with the same hook value. A lower priority value has precedence over higher ones. The **policy** parameter sets the default action for rules in this chain. Note that if you are logged in to the server remotely and you set the default policy to **drop**, you are disconnected immediately if no other rule allows the remote access.

2. Optionally, display all chains:

```
# nft list chains
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
  }
}
```

Additional resources

- The **Address families** section in the **nft(8)** man page
- The **Chains** section in the **nft(8)** man page

2.3.5. Appending a rule to the end of an nftables chain

This section explains how to append a rule to the end of an existing **nftables** chain.

Prerequisites

- The chain to which you want to add the rule exists.

Procedure

1. To add a new rule, use the **nft add rule** command. For example, to add a rule to the **example_chain** in the **example_table** that allows TCP traffic on port 22:

```
# nft add rule inet example_table example_chain tcp dport 22 accept
```

Instead of the port number, you can alternatively specify the name of the service. In the example, you could use **ssh** instead of the port number **22**. Note that a service name is resolved to a port number based on its entry in the /**etc**/**services** file.

2. Optionally, display all chains and their rules in **example_table**:

```
# nft list table inet example_table
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    ...
    tcp dport ssh accept
  }
}
```

Additional resources

- The **Address families** section in the **nft(8)** man page
- The **Rules** section in the **nft(8)** man page

2.3.6. Inserting a rule at the beginning of an nftables chain

This section explains how to insert a rule at the beginning of an existing **nftables** chain.

Prerequisites

- The chain to which you want to add the rule exists.

Procedure

1. To insert a new rule, use the **nft insert rule** command. For example, to insert a rule to the **example_chain** in the **example_table** that allows TCP traffic on port 22:

```
# nft insert rule inet example_table example_chain tcp dport 22 accept
```

You can alternatively specify the name of the service instead of the port number. In the example, you could use **ssh** instead of the port number **22**. Note that a service name is resolved to a port number based on its entry in the /**etc**/**services** file.

2. Optionally, display all chains and their rules in **example_table**:

```
# nft list table inet example_table
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept
```

```
    ...
  }
}
```

Additional resources

- The **Address families** section in the **nft(8)** man page
- The **Rules** section in the **nft(8)** man page

2.3.7. Inserting a rule at a specific position of an nftables chain

This section explains how to insert rules before and after an existing rule in an **nftables** chain. This way you can place new rules at the right position.

Prerequisites

- The chain to which you want to add the rules exists.

Procedure

1. Use the **nft -a list ruleset** command to display all chains and their rules in the **example_table** including their handle:

```
# nft -a list table inet example_table
table inet example_table { # handle 1
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 443 accept # handle 3
    tcp dport 389 accept # handle 4
  }
}
```

   Using the **-a** displays the handles. You require this information to position the new rules in the next steps.

2. Insert the new rules to the **example_chain** chain in the **example_table**:

   - To insert a rule that allows TCP traffic on port **636** before handle **3**, enter:

```
# nft insert rule inet example_table example_chain position 3 tcp dport 636 accept
```

   - To add a rule that allows TCP traffic on port **80** after handle **3**, enter:

```
# nft add rule inet example_table example_chain position 3 tcp dport 80 accept
```

3. Optionally, display all chains and their rules in **example_table**:

```
# nft -a list table inet example_table
table inet example_table { # handle 1
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
```

```
    tcp dport 636 accept # handle 5
    tcp dport 443 accept # handle 3
    tcp dport 80 accept # handle 6
    tcp dport 389 accept # handle 4
  }
}
```

Additional resources

- The **Address families** section in the **nft(8)** man page
- The **Rules** section in the **nft(8)** man page

2.4. CONFIGURING NAT USING NFTABLES

With **nftables**, you can configure the following network address translation (NAT) types:

- Masquerading
- Source NAT (SNAT)
- Destination NAT (DNAT)
- Redirect

IMPORTANT

You can only use real interface names in **iifname** and **oifname** parameters, and alternative names (**altname**) are not supported.

2.4.1. The different NAT types: masquerading, source NAT, destination NAT, and redirect

These are the different network address translation (NAT) types:

Masquerading and source NAT (SNAT)

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers do not route private IP ranges, such as **10.0.0.0/8**. If you use private IP ranges in your network and users should be able to reach servers on the Internet, map the source IP address of packets from these ranges to a public IP address.
Both masquerading and SNAT are very similar. The differences are:

- Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.
- SNAT sets the source IP address of packets to a specified IP and does not dynamically look up the IP of the outgoing interface. Therefore, SNAT is faster than masquerading. Use SNAT if the outgoing interface uses a fixed IP address.

Destination NAT (DNAT)

Use this NAT type to rewrite the destination address and port of incoming packets. For example, if your web server uses an IP address from a private IP range and is, therefore, not directly accessible from the Internet, you can set a DNAT rule on the router to redirect incoming traffic to this server.

Redirect

This type is a special case of DNAT that redirects packets to the local machine depending on the chain hook. For example, if a service runs on a different port than its standard port, you can redirect incoming traffic from the standard port to this specific port.

2.4.2. Configuring masquerading using nftables

Masquerading enables a router to dynamically change the source IP of packets sent through an interface to the IP address of the interface. This means that if the interface gets a new IP assigned, **nftables** automatically uses the new IP when replacing the source IP.

The following procedure describes how to replace the source IP of packets leaving the host through the **ens3** interface to the IP set on **ens3**.

Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```

IMPORTANT

Even if you do not add a rule to the **prerouting** chain, the **nftables** framework requires this chain to match incoming packet replies.

Note that you must pass the **--** option to the **nft** command to avoid that the shell interprets the negative priority value as an option of the **nft** command.

3. Add a rule to the **postrouting** chain that matches outgoing packets on the **ens3** interface:

```
# nft add rule nat postrouting oifname "ens3" masquerade
```

2.4.3. Configuring source NAT using nftables

On a router, Source NAT (SNAT) enables you to change the IP of packets sent through an interface to a specific IP address.

The following procedure describes how to replace the source IP of packets leaving the router through the **ens3** interface to **192.0.2.1**.

Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```

IMPORTANT

Even if you do not add a rule to the **postrouting** chain, the **nftables** framework requires this chain to match outgoing packet replies.

Note that you must pass the **--** option to the **nft** command to avoid that the shell interprets the negative priority value as an option of the **nft** command.

3. Add a rule to the **postrouting** chain that replaces the source IP of outgoing packets through **ens3** with **192.0.2.1**:

```
# nft add rule nat postrouting oifname "ens3" snat to 192.0.2.1
```