Red Hat Enterprise Linux 9

Configuring firewalls and packet filters

Managing the firewalld service, the nftables framework, and XDP packet filtering features

Last Updated: 2024-05-01

Red Hat Enterprise Linux 9 Configuring firewalls and packet filters

Managing the firewalld service, the nftables framework, and XDP packet filtering features

Legal Notice

Abstract

Packet filters, such as firewalls, use rules to control incoming, outgoing, and forwarded network traffic. In Red Hat Enterprise Linux (RHEL), you can use the firewalld service and the nftables framework to filter network traffic and build performance-critical firewalls. You can also use the Express Data Path (XDP) feature of the kernel to process or drop network packets at the network interface at a very high rate.

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 feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

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.

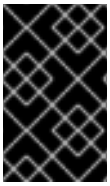
Services use one or more ports or addresses for network communication. Firewalls filter communication based on ports. To allow network traffic for a service, its ports must be open. **firewalld** blocks all traffic on ports that are not explicitly set as open. Some zones, such as trusted, allow all traffic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

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

1.1. 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 prevent the different firewall-related services (**firewalld**, **nftables**, or **iptables**) from influencing each other, run only one of them on a RHEL host, and disable the other services.

1.2. FIREWALL ZONES

You can use the **firewalld** utility to separate networks into different zones according to the level of trust that you have with the interfaces and traffic within that network. A connection can only be part of one zone, but you can use that zone for many network connections.

firewalld follows strict principles in regards to zones:

1. Traffic ingresses only one zone.
2. Traffic egresses only one zone.

3. A zone defines a level of trust.
4. Intrazone traffic (within the same zone) is allowed by default.
5. Interzone traffic (from zone to zone) is denied by default.

Principles 4 and 5 are a consequence of principle 3.

Principle 4 is configurable through the zone option **--remove-forward**. Principle 5 is configurable by adding new policies.

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

- **NetworkManager**
- **firewall-config** utility
- **firewall-cmd** utility
- The RHEL web console

The RHEL web console, **firewall-config**, and **firewall-cmd** 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 /**usr**/**lib**/**firewalld**/**zones**/ directory stores the predefined zones, and you can instantly apply them 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

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

dmz

- Suitable for: Computers in your DMZ that are publicly-accessible with limited access to your internal network.
- Accepts: Only selected incoming connections.

drop

Suitable for: Any incoming network packets are dropped without any notification.

**Accepts: Only outgoing network connections.

external

- Suitable for: External networks with masquerading enabled, especially for routers. Situations when you do not trust the other computers on the network.
- Accepts: Only selected incoming connections.

home

- Suitable for: Home environment where you mostly trust the other computers on the network.
- Accepts: Only selected incoming connections.

internal

- Suitable for: Internal networks where you mostly trust the other computers on the network.
- Accepts: Only selected incoming connections.

public

- Suitable for: Public areas where you do not trust other computers on the network.
- Accepts: Only selected incoming connections.

trusted

- Accepts: All network connections.

work

Suitable for: Work environment where you mostly trust the other computers on the network.

- Accepts: Only selected incoming connections.

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 the **public** zone. You can change the default zone.

NOTE

Make network zone names self-explanatory to help users understand them quickly.

To avoid any security problems, review the default zone configuration and disable any unnecessary services according to your needs and risk assessments.

Additional resources

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

1.3. FIREWALL POLICIES

The firewall policies specify the desired security state of your network. They outline rules and actions to take for different types of traffic. Typically, the policies contain rules for the following types of traffic:

- Incoming traffic
- Outgoing traffic
- Forward traffic
- Specific services and applications

- Network address translations (NAT)

Firewall policies use the concept of firewall zones. Each zone is associated with a specific set of firewall rules that determine the traffic allowed. Policies apply firewall rules in a stateful, unidirectional manner. This means you only consider one direction of the traffic. The traffic return path is implicitly allowed due to stateful filtering of **firewalld**.

Policies are associated with an ingress zone and an egress zone. The ingress zone is where the traffic originated (received). The egress zone is where the traffic leaves (sent).

The firewall rules defined in a policy can reference the firewall zones to apply consistent configurations across multiple network interfaces.

1.4. FIREWALL RULES

You can use the firewall rules to implement specific configurations for allowing or blocking network traffic. As a result, you can control the flow of network traffic to protect your system from security threats.

Firewall rules typically define certain criteria based on various attributes. The attributes can be as:

- Source IP addresses
- Destination IP addresses
- Transfer Protocols (TCP, UDP, ...)
- Ports
- Network interfaces

The **firewalld** utility organizes the firewall rules into zones (such as **public**, **internal**, and others) and policies. Each zone has its own set of rules that determine the level of traffic freedom for network interfaces associated with a particular zone.

1.5. ZONE CONFIGURATION FILES

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>
```

Additional resources

- **firewalld.zone** manual page

1.6. PREDEFINED FIREWALLD SERVICES

The **firewalld** service is a predefined set of firewall rules that define access to a specific application or network service. Each service represents a combination of the following elements:

- Local port
- Network protocol
- Associated firewall rules
- Source ports and destinations
- Firewall helper modules that load automatically if a service is enabled

A service simplifies packet filtering and saves you time because it achieves several tasks at once. For example, **firewalld** can perform the following tasks at once:

- Open a port
- Define network protocol
- Enable packet forwarding

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**.

You can configure **firewalld** in the following ways:

- Use utilities:
  - **firewall-config** - graphical utility
  - **firewall-cmd** - command-line utility
  - **firewall-offline-cmd** - command-line utility
- Edit the XML files in the **/etc/firewalld/services/** directory.
  If you do not add or change the service, no corresponding XML file exists in **/etc/firewalld/services/**. You can use the files in **/usr/lib/firewalld/services/** as templates.

Additional resources

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

1.7. 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.

1.7.1. Customizing firewall settings for a specific zone to enhance security

You can strengthen your network security by modifying the firewall settings and associating a specific network interface or connection with a particular firewall zone. By defining granular rules and restrictions for a zone, you can control inbound and outbound traffic based on your intended security levels.

For example, you can achieve the following benefits:

- Protection of sensitive data
- Prevention of unauthorized access
- Mitigation of potential network threats

Prerequisites

- The **firewalld** service is running.

Procedure

1. List the available firewall zones:

```
# 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. To see more detailed information for all zones, use the **firewall-cmd --list-all-zones** command.

2. Choose the zone you want to use for this configuration.
3. Modify firewall settings for the chosen zone. For example, to allow the **SSH** service and remove the **ftp** service:

```
# firewall-cmd --add-service=ssh --zone=<your_chosen_zone>
# firewall-cmd --remove-service=ftp --zone=<same_chosen_zone>
```

4. Assign a network interface to the firewall zone:
   a. List the available network interfaces:

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

      Activity of a zone is determined by the presence of network interfaces or source address ranges that match its configuration. The default zone is active for unclassified traffic but is not always active if no traffic matches its rules.

   b. Assign a network interface to the chosen zone:

```
# firewall-cmd --zone=<your_chosen_zone> --change-interface=<interface_name> --permanent
```

      Assigning a network interface to a zone is more suitable for applying consistent firewall settings to all traffic on a particular interface (physical or virtual).

      The **firewall-cmd** command, when used with the **--permanent** option, often involves

updating NetworkManager connection profiles to make changes to the firewall configuration permanent. This integration between **firewalld** and NetworkManager ensures consistent network and firewall settings.

Verification

1. Display the updated settings for your chosen zone:

```
# firewall-cmd --zone=<your_chosen_zone> --list-all
```

The command output displays all zone settings including the assigned services, network interface, and network connections (sources).

1.7.2. 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. Note that settings for all other zones are preserved and ready to be used.

Typically, zones are assigned to interfaces by NetworkManager according to the **connection.zone** setting in NetworkManager connection profiles. Also, after a reboot NetworkManager manages assignments for "activating" those zones.

Prerequisites

- The **firewalld** service is running.

Procedure

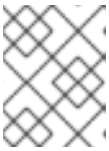
To set up the default zone:

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.

1.7.3. 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
```

1.7.4. Assigning a zone to a connection using nmcli

You can 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
```

1.7.5. Manually assigning a zone to a network connection in a connection profile file

If you cannot use the **nmcli** utility to modify a connection profile, you can manually edit the corresponding file of the profile to assign a **firewalld** zone.

NOTE

Modifying the connection profile with the **nmcli** utility to assign a **firewalld** zone is more efficient. For details, see Assigning a network interface to a zone .

Procedure

1. Determine the path to the connection profile and its format:

```
# nmcli -f NAME,FILENAME connection
NAME    FILENAME
enp1s0  /etc/NetworkManager/system-connections/enp1s0.nmconnection
enp7s0  /etc/sysconfig/network-scripts/ifcfg-enp7s0
```

NetworkManager uses separate directories and file names for the different connection profile formats:

- Profiles in **/etc/NetworkManager/system-connections/*<connection_name>*.nmconnection** files use the keyfile format.
- Profiles in **/etc/sysconfig/network-scripts/ifcfg-*<interface_name>*** files use the ifcfg format.

2. Depending on the format, update the corresponding file:

- If the file uses the keyfile format, append **zone=*<name>*** to the **[connection]** section of the

/etc/NetworkManager/system-connections/*<connection_name>*.nmconnection file:

```
[connection]
...
zone=internal
```

- If the file uses the ifcfg format, append **ZONE=*<name>*** to the **/etc/sysconfig/network-scripts/ifcfg-*<interface_name>*** file:

```
ZONE=internal
```

3. Reload the connection profiles:

```
# nmcli connection reload
```

4. Reactivate the connection profiles

```
# nmcli connection up <profile_name>
```

Verification

- Display the zone of the interface, for example:

```
# firewall-cmd --get-zone-of-interface enp1s0
internal
```

1.7.6. 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.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Create a new zone:

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

2. Make the new zone usable:

```
# firewall-cmd --reload
```

The command applies recent changes to the firewall configuration without interrupting network services that are already running.

Verification

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

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

1.7.7. 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.

Prerequisites

- The **firewalld** service is running.

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

1.8. CONTROLLING NETWORK TRAFFIC USING FIREWALLD

The **firewalld** package installs a large number of predefined service files and you can add more or customize them. You can then use these service definitions to open or close ports for services without knowing the protocol and port numbers they use.

1.8.1. Controlling traffic with predefined services using the 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*.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Check that the service in **firewalld** is not already allowed:

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

   The command lists the services that are enabled in the default zone.

2. List all predefined services in **firewalld**:

```
# 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 ...
```

   The command displays a list of available services for the default zone.

3. Add the service to the list of services that **firewalld** allows:

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

   The command adds the specified service to the default zone.

4. Make the new settings persistent:

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

   The command applies these runtime changes to the permanent configuration of the firewall. By default, it applies these changes to the configuration of the default zone.

Verification

1. List all permanent firewall rules:

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

   The command displays complete configuration with the permanent firewall rules of the default firewall zone (**public**).

2. Check the validity of the permanent configuration of the **firewalld** service.

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

If the permanent configuration is invalid, the command returns an error with further details:

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

You can also manually inspect the permanent configuration files to verify the settings. The main configuration file is **/etc/firewalld/firewalld.conf**. The zone-specific configuration files are in the **/etc/firewalld/zones/** directory and the policies are in the **/etc/firewalld/policies/** directory.

1.8.2. Controlling traffic with predefined services using the GUI

You can control the network traffic with predefined services using a graphical user interface. The Firewall Configuration application provides an accessible and user-friendly alternative to the command-line utilities.

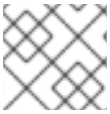
Prerequisites

- You installed the **firewall-config** package.
- The **firewalld** service is running.

Procedure

1. To enable or disable a predefined or custom service:
   a. Start the **firewall-config** utility 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 checkbox for each type of service you want to trust or clear the checkbox to block a service in the selected zone.
2. To edit a service:
   a. Start the **firewall-config** utility.
   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**).

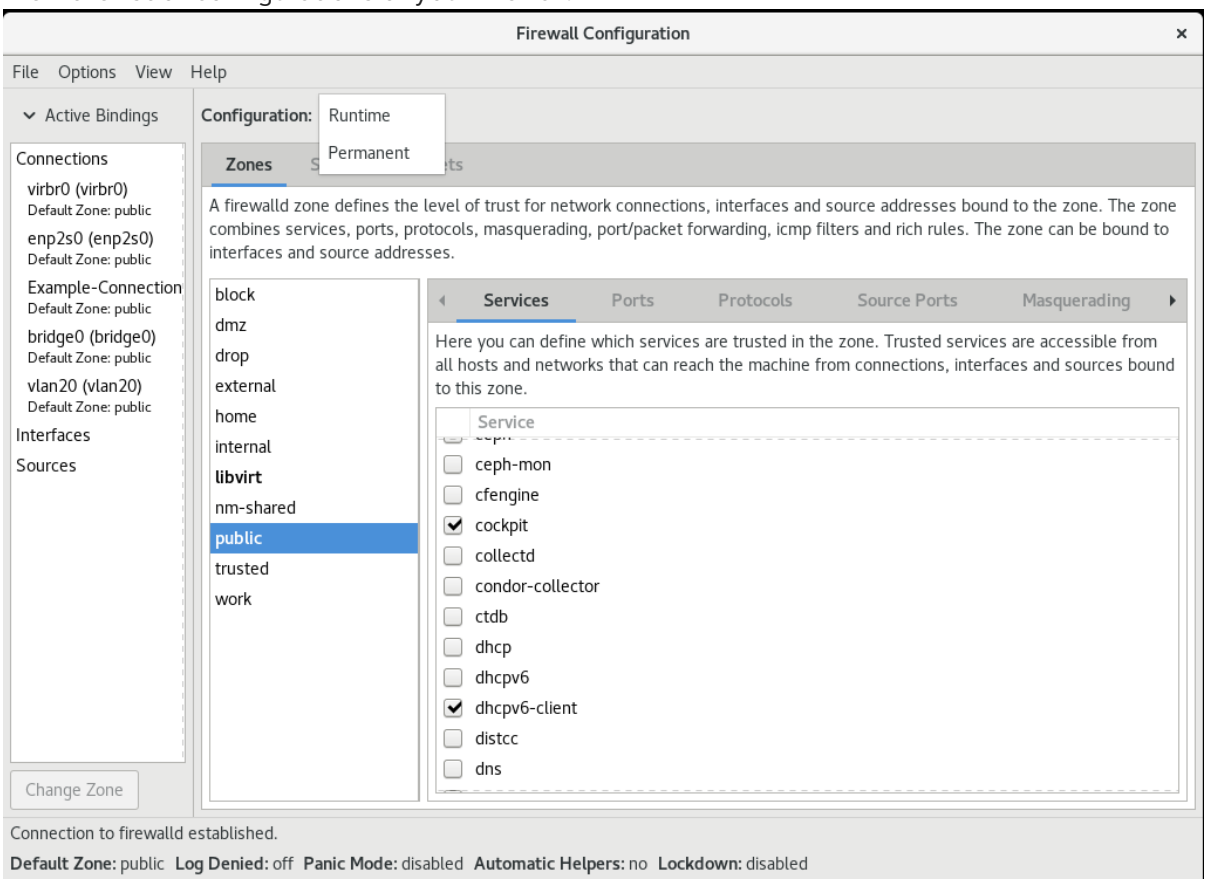


NOTE

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

Verification

- Press the **Super** key to enter the Activities overview.
- Select the Firewall Configuration utility.
  - You can also start the graphical firewall configuration utility using the command-line, by entering the **firewall-config** command.
- View the list of configurations of your firewall:



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.

1.8.3. Configuring firewalld to allow hosting a secure web server

Ports are logical services that enable an operating system to receive and distinguish network traffic and forward it to system services. The system services are represented by a daemon that listens on the port and 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 can directly specify the port number instead of the service name.

You can use the **firewalld** service to configure access to a secure web server for hosting your data.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Check the currently active firewall zone:
   -

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

2. Add the HTTPS service to the appropriate zone:

```
# firewall-cmd --zone=<zone_name> --add-service=https --permanent
```

3. Reload the firewall configuration:

```
# firewall-cmd --reload
```

Verification

1. Check if the port is open in **firewalld**:
   - If you opened the port by specifying the port number, enter:

```
# firewall-cmd --zone=<zone_name> --list-all
```

   - If you opened the port by specifying a service definition, enter:

```
# firewall-cmd --zone=<zone_name> --list-services
```

1.8.4. Closing unused or unnecessary ports to enhance network security

When an open port is no longer needed, you can use the **firewalld** utility to close it.

IMPORTANT

Close all unnecessary ports to reduce the potential attack surface and minimize the risk of unauthorized access or exploitation of vulnerabilities.

Procedure

1. List all allowed ports:

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

By default, this command lists the ports that are enabled in the default zone.

NOTE

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

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

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

This command removes a port from a zone. If you do not specify a zone, it will remove the port from the default zone.

3. Make the new settings persistent:

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

Without specifying a zone, this command applies runtime changes to the permanent configuration of the default zone.

Verification

1. List the active zones and choose the zone you want to inspect:

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

2. List the currently open ports in the selected zone to check if the unused or unnecessary ports are closed:

```
# firewall-cmd --zone=<zone_to_inspect> --list-ports
```

1.8.5. Controlling traffic through the CLI

You can use the **firewall-cmd** command to:

- disable networking traffic
- enable networking traffic

As a result, you can for example enhance your system defenses, ensure data privacy or optimize network resources.

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.

Procedure

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

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

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
```

1.8.6. 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.

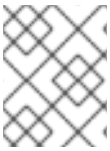
1.9. USING ZONES TO MANAGE INCOMING TRAFFIC DEPENDING ON A SOURCE

You can use zones to manage incoming traffic based on its source. Incoming traffic in this context is any data that is destined for your system, or passes through the host running **firewalld**. The source typically refers to the IP address or network range from which the traffic originates. As a result, you can sort incoming traffic and assign it to different zones to allow or disallow services that can be reached by that traffic.

Matching by source address takes precedence over matching by interface name. When you add a source to a zone, the firewall will prioritize the source-based rules for incoming traffic over interface-based rules. This means that if incoming traffic matches a source address specified for a particular zone, the zone associated with that source address will determine how the traffic is handled, regardless of the interface through which it arrives. On the other hand, interface-based rules are generally a fallback for traffic that does not match specific source-based rules. These rules apply to traffic, for which the source is not explicitly associated with a zone. This allows you to define a default behavior for traffic that does not have a specific source-defined zone.

1.9.1. 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
```

1.9.2. Removing a source

When you remove a source from a zone, the traffic which originates from the source is no longer directed through the rules specified for that source. Instead, the traffic falls back to the rules and settings of the zone associated with the interface from which it originates, or goes to the default zone.

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
```

1.9.3. 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>
```

1.9.4. 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

1.10. FILTERING FORWARDED TRAFFIC BETWEEN ZONES

firewalld enables you to control the flow of network data between different **firewalld** zones. By defining rules and policies, you can manage how traffic is allowed or blocked when it moves between these zones.

The policy objects feature provides forward and output filtering in **firewalld**. You can use **firewalld** to filter traffic between different zones to allow access to locally hosted VMs to connect the host.

1.10.1. 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.

1.10.2. 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.

Lower numerical priority values have higher precedence and are applied first.

1.10.3. 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 traffic between Podman and firewalld zones.

NOTE

Red Hat recommends blocking all traffic by default and opening the selective services needed for the Podman utility.

Procedure

1. Create a new firewall policy:

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

2. Block all traffic from Podman to other zones and allow only necessary services on Podman:

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

3. Create a new Podman zone:

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

4. Define the ingress zone for the policy:

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

5. Define the egress zone for all other zones:

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

Setting the egress zone to ANY means that you filter from Podman to other zones. If you want to filter to the host, then set the egress zone to HOST.

6. Restart the firewalld service:

```
# systemctl restart firewalld
```

Verification

- Verify the Podman firewall policy to other zones:

```
# firewall-cmd --info-policy podmanToAny
podmanToAny (active)
  ...
  target: REJECT
  ingress-zones: podman
  egress-zones: ANY
  services: dhcp dns https
  ...
```

1.10.4. 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
```

1.10.5. Using DNAT to forward HTTPS traffic to a different host

If your web server runs in a DMZ with private IP addresses, you can configure destination network address translation (DNAT) to enable clients on the internet to connect to this web server. In this case, the host name of the web server resolves to the public IP address of the router. When a client establishes a connection to a defined port on the router, the router forwards the packets to the internal web server.

Prerequisites

- The DNS server resolves the host name of the web server to the router's IP address.
- You know the following settings:
  - The private IP address and port number that you want to forward
  - The IP protocol to be used
  - The destination IP address and port of the web server where you want to redirect the packets

Procedure

1. Create a firewall policy:

```
# firewall-cmd --permanent --new-policy <example_policy>
```

   The policies, as opposed to zones, allow packet filtering for input, output, and forwarded traffic. This is important, because forwarding traffic to endpoints on locally run web servers, containers, or virtual machines requires such capability.

2. Configure symbolic zones for the ingress and egress traffic to also enable the router itself to connect to its local IP address and forward this traffic:

```
# firewall-cmd --permanent --policy=<example_policy> --add-ingress-zone=HOST
# firewall-cmd --permanent --policy=<example_policy> --add-egress-zone=ANY
```

   The **--add-ingress-zone=HOST** option refers to packets generated locally and transmitted out of the local host. The **--add-egress-zone=ANY** option refers to traffic moving to any zone.

3. Add a rich rule that forwards traffic to the web server:

```
# firewall-cmd --permanent --policy=<example_policy> --add-rich-rule='rule
family="ipv4" destination address="192.0.2.1" forward-port port="443" protocol="tcp"
to-port="443" to-addr="192.51.100.20"'
```

The rich rule forwards TCP traffic from port 443 on the IP address of the router (192.0.2.1) to port 443 of the IP address of the web server (192.51.100.20).

4. Reload the firewall configuration files:

```
# firewall-cmd --reload
success
```

5. Activate routing of 127.0.0.0/8 in the kernel:

   - For persistent changes, run:

```
# echo "net.ipv4.conf.all.route_localnet=1" > /etc/sysctl.d/90-enable-route-localnet.conf
```

     The command persistently configures the **route_localnet** kernel parameter and ensures that the setting is preserved after the system reboots.

   - For applying the settings immediately without a system reboot, run:

```
# sysctl -p /etc/sysctl.d/90-enable-route-localnet.conf
```

     The **sysctl** command is useful for applying on-the-fly changes, however the configuration will not persist across system reboots.

Verification

1. Connect to the IP address of the router and to the port that you have forwarded to the web server:

```
# curl https://192.0.2.1:443
```

2. Optional: Verify that the **net.ipv4.conf.all.route_localnet** kernel parameter is active:

```
# sysctl net.ipv4.conf.all.route_localnet
net.ipv4.conf.all.route_localnet = 1
```

3. Verify that **<example_policy>** is active and contains the settings you need, especially the source IP address and port, protocol to be used, and the destination IP address and port:

```
# firewall-cmd --info-policy=<example_policy>
example_policy (active)
  priority: -1
  target: CONTINUE
  ingress-zones: HOST
  egress-zones: ANY
  services:
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
```

```
 rich rules:
 rule family="ipv4" destination address="192.0.2.1" forward-port port="443" protocol="tcp" to-port="443" to-addr="192.51.100.20"
```

Additional resources

- **firewall-cmd(1)**, **firewalld.policies(5)**, **firewalld.richlanguage(5)**, **sysctl(8)**, and **sysctl.conf(5)** man pages
- Using configuration files in /etc/sysctl.d/ to adjust kernel parameters

1.11. CONFIGURING NAT USING FIREWALLD

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

- Masquerading
- Destination NAT (DNAT)
- Redirect

1.11.1. Network address translation types

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

Masquerading

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers (ISPs) 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.
Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic 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 a different port on the local machine. 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.

1.11.2. Configuring IP address masquerading

You can 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.

1.11.3. Using DNAT to forward incoming HTTP traffic

You can use destination network address translation (DNAT) to direct incoming traffic from one destination address and port to another. Typically, this is useful for redirecting incoming requests from an external network interface to specific internal servers or services.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Create the **/etc/sysctl.d/90-enable-IP-forwarding.conf** file with the following content:

```
net.ipv4.ip_forward=1
```

This setting enables IP forwarding in the kernel. It makes the internal RHEL server act as a router and forward packets from network to network.

2. Load the setting from the **/etc/sysctl.d/90-enable-IP-forwarding.conf** file:

```
# sysctl -p /etc/sysctl.d/90-enable-IP-forwarding.conf
```

3. Forward incoming HTTP traffic:

```
# firewall-cmd --zone=public --add-forward-
port=port=80:proto=tcp:toaddr=198.51.100.10:toport=8080 --permanent
```

The previous command defines a DNAT rule with the following settings:

- **--zone=public** - The firewall zone for which you configure the DNAT rule. You can adjust this to whatever zone you need.
- **--add-forward-port** - The option that indicates you are adding a port-forwarding rule.
- **port=80** - The external destination port.
- **proto=tcp** - The protocol indicating that you forward TCP traffic.

- **toaddr=198.51.100.10** - The destination IP address.
- **toport=8080** - The destination port of the internal server.
- **--permanent** - The option that makes the DNAT rule persistent across reboots.

4. Reload the firewall configuration to apply the changes:

```
# firewall-cmd --reload
```

Verification

- Verify the DNAT rule for the firewall zone that you used:

```
# firewall-cmd --list-forward-ports --zone=public
port=80:proto=tcp:toport=8080:toaddr=198.51.100.10
```

Alternatively, view the corresponding XML configuration file:

```
# cat /etc/firewalld/zones/public.xml
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>Public</short>
  <description>For use in public areas. You do not trust the other computers on networks to
not harm your computer. Only selected incoming connections are accepted.</description>
  <service name="ssh"/>
  <service name="dhcpv6-client"/>
  <service name="cockpit"/>
  <forward-port port="80" protocol="tcp" to-port="8080" to-addr="198.51.100.10"/>
  <forward/>
</zone>
```

Additional resources

- Configuring kernel parameters at runtime
- **firewall-cmd(1)** manual page

1.11.4. Redirecting traffic from a non-standard port to make the web service accessible on a standard port

You can use the redirect mechanism to make the web service that internally runs on a non-standard port accessible without requiring users to specify the port in the URL. As a result, the URLs are simpler and provide better browsing experience, while a non-standard port is still used internally or for specific requirements.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Create the **/etc/sysctl.d/90-enable-IP-forwarding.conf** file with the following content:

```
net.ipv4.ip_forward=1
```

This setting enables IP forwarding in the kernel.

2. Load the setting from the **/etc/sysctl.d/90-enable-IP-forwarding.conf** file:

```
# sysctl -p /etc/sysctl.d/90-enable-IP-forwarding.conf
```

3. Create the NAT redirect rule:

```
# firewall-cmd --zone=public --add-forward-
port=port=<standard_port>:proto=tcp:toport=<non_standard_port> --permanent
```

The previous command defines the NAT redirect rule with the following settings:

- **--zone=public** - The firewall zone, for which you configure the rule. You can adjust this to whatever zone you need.
- **--add-forward-port=port=*<non_standard_port>*** - The option that indicates you are adding a port-forwarding (redirecting) rule with source port on which you initially receive the incoming traffic.
- **proto=tcp** - The protocol indicating that you redirect TCP traffic.
- **toport=*<standard_port>*** - The destination port, to which the incoming traffic should be redirected after being received on the source port.
- **--permanent** - The option that makes the rule persist across reboots.

4. Reload the firewall configuration to apply the changes:

```
# firewall-cmd --reload
```

Verification

- Verify the redirect rule for the firewall zone that you used:

```
# firewall-cmd --list-forward-ports
port=8080:proto=tcp:toport=80:toaddr=
```

Alternatively, view the corresponding XML configuration file:

```
# cat /etc/firewalld/zones/public.xml
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>Public</short>
  <description>For use in public areas. You do not trust the other computers on networks to
not harm your computer. Only selected incoming connections are accepted.</description>
  <service name="ssh"/>
  <service name="dhcpv6-client"/>
  <service name="cockpit"/>
  <forward-port port="8080" protocol="tcp" to-port="80"/>
  <forward/>
</zone>
```

Additional resources

- Configuring kernel parameters at runtime
- **firewall-cmd(1)** manual page

1.12. MANAGING ICMP REQUESTS

The **Internet Control Message Protocol** (**ICMP**) is a supporting protocol that is used by various network devices for testing, troubleshooting, and diagnostics. **ICMP** differs from transport protocols such as TCP and UDP because it is not used to exchange data between systems.

You can use the **ICMP** messages, especially **echo-request** and **echo-reply**, to reveal information about a network and misuse such information for various kinds of fraudulent activities. Therefore, **firewalld** enables controlling the **ICMP** requests to protect your network information.

1.12.1. Configuring ICMP filtering

You can use ICMP filtering to define which ICMP types and codes you want the firewall to permit or deny from reaching your system. ICMP types and codes are specific categories and subcategories of ICMP messages.

ICMP filtering helps, for example, in the following areas:

- Security enhancement - Block potentially harmful ICMP types and codes to reduce your attack surface.
- Network performance - Permit only necessary ICMP types to optimize network performance and prevent potential network congestion caused by excessive ICMP traffic.
- Troubleshooting control - Maintain essential ICMP functionality for network troubleshooting and block ICMP types that represent potential security risk.

Prerequisites

- The **firewalld** service is running.

Procedure

1. List available ICMP types and codes:

```
# firewall-cmd --get-icmptypes
address-unreachable bad-header beyond-scope communication-prohibited destination-unreachable echo-reply echo-request failed-policy fragmentation-needed host-precedence-violation host-prohibited host-redirect host-unknown host-unreachable
...
```

   From this predefined list, select which ICMP types and codes to allow or block.

2. Filter specific ICMP types by:
   - Allowing ICMP types:

```
# firewall-cmd --zone=<target-zone> --remove-icmp-block=echo-request --permanent
```

The command removes any existing blocking rules for the echo requests ICMP type.

- Blocking ICMP types:

```
# firewall-cmd --zone=<target-zone> --add-icmp-block=redirect --permanent
```

The command ensures that the redirect messages ICMP type is blocked by the firewall.

3. Reload the firewall configuration to apply the changes:

```
# firewall-cmd --reload
```

Verification

- Verify your filtering rules are in effect:

```
# firewall-cmd --list-icmp-blocks
redirect
```

The command output displays the ICMP types and codes that you allowed or blocked.

Additional resources

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

1.13. SETTING AND CONTROLLING IP SETS USING FIREWALLD

IP sets are a RHEL feature for grouping of IP addresses and networks into sets to achieve more flexible and efficient firewall rule management.

The IP sets are valuable in scenarios when you need to for example:

- Handle large lists of IP addresses
- Implement dynamic updates to those large lists of IP addresses
- Create custom IP-based policies to enhance network security and control

WARNING

Red Hat recommends using the **firewall-cmd** command to create and manage IP sets.

1.13.1. Configuring dynamic updates for allowlisting with IP sets

You can make near real-time updates to flexibly allow specific IP addresses or ranges in the IP sets even in unpredictable conditions. These updates can be triggered by various events, such as detection of security threats or changes in the network behavior. Typically, such a solution leverages automation to reduce manual effort and improve security by responding quickly to the situation.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Create an IP set with a meaningful name:

```
# firewall-cmd --permanent --new-ipset=allowlist --type=hash:ip
```

   The new IP set called **allowlist** contains IP addresses that you want your firewall to allow.

2. Add a dynamic update to the IP set:

```
# firewall-cmd --permanent --ipset=allowlist --add-entry=198.51.100.10
```

   This configuration updates the **allowlist** IP set with a newly added IP address that is allowed to pass network traffic by your firewall.

3. Create a firewall rule that references the previously created IP set:

```
# firewall-cmd --permanent --zone=public --add-source=ipset:allowlist
```

   Without this rule, the IP set would not have any impact on network traffic. The default firewall policy would prevail.

4. Reload the firewall configuration to apply the changes:

```
# firewall-cmd --reload
```

Verification

1. List all IP sets:

```
# firewall-cmd --get-ipsets
allowlist
```

2. List the active rules:

```
# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp0s1
  sources: ipset:allowlist
  services: cockpit dhcpv6-client ssh
  ports:
  protocols:
  ...
```

   The **sources** section of the command-line output provides insights to what origins of traffic (hostnames, interfaces, IP sets, subnets, and others) are permitted or denied access to a particular firewall zone. In this case, the IP addresses contained in the **allowlist** IP set are allowed to pass traffic through the firewall for the **public** zone.

3. Explore the contents of your IP set:

```
# cat /etc/firewalld/ipsets/allowlist.xml
<?xml version="1.0" encoding="utf-8"?>
<ipset type="hash:ip">
  <entry>198.51.100.10</entry>
</ipset>
```

Next steps

- Use a script or a security utility to fetch your threat intelligence feeds and update **allowlist** accordingly in an automated fashion.

Additional resources

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

1.14. 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. When using the **priority** parameter, rules are sorted first by their priority values in ascending order. When more rules have the same **priority**, their order is determined by the rule action, and if the action is also the same, the order may be undefined.

1.14.1. 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 numerical 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.

1.14.2. Setting the priority of a rich rule

The following is 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

- 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.

Verification

- 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
  }
}
```

1.15. 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.

1.15.1. Configuring lockdown using CLI

You can enable or disable the lockdown feature using the command line.

Procedure

1. To query whether lockdown is enabled:

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

2. Manage lockdown configuration by either:
   - Enabling lockdown:

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

   - Disabling lockdown:

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

1.15.2. Overview of lockdown allowlist 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>
  <command name="/usr/bin/python3 -s /usr/bin/firewall-config"/>
   <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.

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.

1.16. 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.

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

With intra-zone forwarding 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, the traffic stays inside the same zone.

NOTE

Enabling intra-zone forwarding in the default zone of **firewalld**, applies only to the interfaces and sources added to the current default zone.

firewalld uses different zones to manage incoming and outgoing traffic. Each zone has its own set of rules and behaviors. For example, the **trusted** zone, allows all forwarded traffic by default.

Other zones can have different default behaviors. In standard zones, forwarded traffic is typically dropped by default when the target of the zone is set to **default**.

To control how the traffic is forwarded between different interfaces or sources within a zone, make sure you understand and configure the target of the zone accordingly.

1.16.2. 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. This feature brings the following benefits:

- Seamless connectivity between wired and wireless devices (you can forward traffic between an Ethernet network connected to **enp1s0** and a Wi-Fi network connected to **wlp0s20**)
- Support for flexible work environments
- Shared resources that are accessible and used by multiple devices or users within a network (such as printers, databases, network-attached storage, and others)
- Efficient internal networking (such as smooth communication, reduced latency, resource accessibility, and others)

You can enable this functionality for individual **firewalld** zones.

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 assigned only to 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 on the same network as the **enp1s0** interface of the host on which you enabled zone forwarding.
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**. Verify the text is sent back.

Additional resources

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

1.17. CONFIGURING FIREWALLD BY USING RHEL SYSTEM ROLES

You can use the **firewall** RHEL 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 RHEL system role applies the **firewalld** parameters to the managed node immediately and makes them persistent across reboots.

1.17.1. 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 RHEL system role, and also the reference documentation.

To apply the **firewalld** parameters on one or more systems in an automated fashion, use the **firewall** RHEL 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

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file
- **/usr/share/doc/rhel-system-roles/firewall/** directory

- Working with playbooks
- How to build your inventory

1.17.2. Resetting the firewalld settings by using a RHEL system role

With the **firewall** RHEL system role, you can reset the **firewalld** settings to their default state. If you add the **previous:replaced** parameter to the variable list, the RHEL system role removes all existing user-defined settings and resets **firewalld** to the defaults. If you combine the **previous:replaced** parameter with other settings, the **firewall** role removes all existing settings before applying new ones.

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example ~/**playbook.yml**, with the following content:

```
---
- name: Reset firewalld example
  hosts: managed-node-01.example.com
  tasks:
    - name: Reset firewalld
      ansible.builtin.include_role:
        name: rhel-system-roles.firewall
      vars:
        firewall:
          - previous: replaced
```

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Run this command as **root** on the managed node to check all the zones:

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

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file
- **/usr/share/doc/rhel-system-roles/firewall/** directory

1.17.3. Forwarding incoming traffic in **firewalld** from one local port to a different local port by using a RHEL system role

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

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example ~/**playbook.yml**, with the following content:

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Forward incoming traffic on port 8080 to 443
      ansible.builtin.include_role:
        name: rhel-system-roles.firewall
      vars:
        firewall:
          - { forward_port: 8080/tcp;443;, state: enabled, runtime: true, permanent: true }
```

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- On the managed host, display the **firewalld** settings:

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

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file
- **/usr/share/doc/rhel-system-roles/firewall/** directory

1.17.4. Managing ports in firewalld by using a RHEL system role

You can use the **firewall** RHEL system role to open or close ports in the local firewall for incoming traffic and make the new configuration persist across reboots. For example you can configure the default zone to permit incoming traffic for the HTTPS service.

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example ~/**playbook.yml**, with the following content:

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Allow incoming HTTPS traffic to the local host
      ansible.builtin.include_role:
        name: rhel-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.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- On the managed node, verify that the **443/tcp** port associated with the **HTTPS** service is open:

```
# firewall-cmd --list-ports
443/tcp
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file
- **/usr/share/doc/rhel-system-roles/firewall/** directory

1.17.5. Configuring a firewalld DMZ zone by using a RHEL system role

As a system administrator, you can use the **firewall** RHEL 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.

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example ~/**playbook.yml**, with the following content:

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Creating a DMZ with access to HTTPS port and masquerading for hosts in DMZ
      ansible.builtin.include_role:
        name: rhel-system-roles.firewall
      vars:
        firewall:
          - zone: dmz
            interface: enp1s0
            service: https
            state: enabled
            runtime: true
            permanent: true
```

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

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:
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file
- **/usr/share/doc/rhel-system-roles/firewall/** directory

CHAPTER 2. GETTING STARTED WITH NFTABLES

The **nftables** framework classifies packets and it is the successor to the **iptables**, **ip6tables**, **arptables**, **ebtables**, and **ipset** utilities. It offers numerous improvements in convenience, features, and performance over previous packet-filtering tools, most notably:

- Built-in lookup tables instead of linear processing
- A single framework for both the **IPv4** and **IPv6** protocols
- All rules 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 **nft** utility replaces all tools from the previous packet-filtering frameworks. You can use the **libnftnl** library for low-level interaction with **nftables** Netlink API through the **libmnl** library.

To display the effect of rule set changes, use the **nft list ruleset** command. Because these utilities 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 **iptables** command.

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 prevent the different firewall-related services (**firewalld**, **nftables**, or **iptables**) from influencing each other, run only one of them on a RHEL host, and disable the other services.

2.1.2. Converting iptables and ip6tables rule sets to nftables

Use the **iptables-restore-translate** and **ip6tables-restore-translate** utilities to translate **iptables** and **ip6tables** rule sets to **nftables**.

Prerequisites

- The **nftables** and **iptables** packages are installed.
- The system has **iptables** and **ip6tables** rules configured.

Procedure

1. Write the **iptables** and **ip6tables** rules to a file:

```
# iptables-save >/root/iptables.dump
# ip6tables-save >/root/ip6tables.dump
```

2. Convert the dump files to **nftables** instructions:

```
# iptables-restore-translate -f /root/iptables.dump > /etc/nftables/ruleset-migrated-from-iptables.nft
# ip6tables-restore-translate -f /root/ip6tables.dump > /etc/nftables/ruleset-migrated-from-ip6tables.nft
```

3. Review and, if needed, manually update the generated **nftables** rules.
4. To enable the **nftables** service to load the generated files, add the following to the **/etc/sysconfig/nftables.conf** file:

```
include "/etc/nftables/ruleset-migrated-from-iptables.nft"
include "/etc/nftables/ruleset-migrated-from-ip6tables.nft"
```

5. Stop and disable the **iptables** service:

```
# systemctl disable --now iptables
```

If you used a custom script to load the **iptables** rules, ensure that the script no longer starts automatically and reboot to flush all tables.

6. Enable and start the **nftables** service:

```
# systemctl enable --now nftables
```

Verification

- Display the **nftables** rule set:

```
# nft list ruleset
```

Additional resources

- Automatically loading nftables rules when the system boots

2.1.3. Converting single iptables and ip6tables rules to nftables

Red Hat Enterprise Linux provides the **iptables-translate** and **ip6tables-translate** utilities to convert an **iptables** or **ip6tables** rule into the equivalent one for **nftables**.

Prerequisites

- The **nftables** package is installed.

Procedure

- Use the **iptables-translate** or **ip6tables-translate** utility instead of **iptables** or **ip6tables** to display the corresponding **nftables** rule, for example:

```
# iptables-translate -A INPUT -s 192.0.2.0/24 -j ACCEPT
nft add rule ip filter INPUT ip saddr 192.0.2.0/24 counter accept
```

Note that some extensions lack translation support. In these cases, the utility 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
```

Additional resources

- **iptables-translate --help**

2.1.4. 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	nftables
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.

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 major benefit of using the **nftables** framework is that 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, with the **nftables** script environment, you can:

- Add comments
- Define variables
- Include other rule-set files

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

You can write scripts in the **nftables** scripting environment in the following formats:

- 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
  }
}
```

- The same syntax as for **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 an **nftables** script either by passing it to the **nft** utility or by executing the script directly.

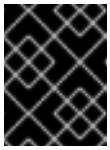
Procedure

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

```
# nft -f /etc/nftables/<example_firewall_script>.nft
```

- To run an **nftables** script directly:
  a. For the single time that you perform this:
     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_script>.nft
```

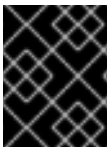
     iii. Make the script executable for the owner:

```
# chmod u+x /etc/nftables/<example_firewall_script>.nft
```

  b. Run the script:

```
# /etc/nftables/<example_firewall_script>.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 to the end of a line as a comment.

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

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

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

2.2.4. Using variables in 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 entering the **$** sign followed by the variable name:

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

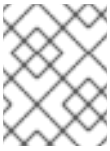
Variables that contain an anonymous set

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

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 nftables scripts

In the **nftables** scripting environment, you can 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 2.1. Including files from the default search directory

To include a file from the default search directory:

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

Example 2.2. 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.

Prerequisites

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

Procedure

1. Edit the **/etc/sysconfig/nftables.conf** file.

- If you modified the ***.nft** scripts that were created in **/etc/nftables/** with the installation of the **nftables** package, uncomment the **include** statement for these scripts.
- If you wrote new scripts, 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. Optional: 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

You can display **nftables** rule sets and manage them.

2.3.1. Basics of nftables tables

A table in **nftables** is a namespace that contains a collection of chains, rules, sets, and other objects.

Each table must have an address family assigned. The address family defines the packet types that this 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 pass through a bridge device.
- **netdev**: Matches packets from ingress.

If you want to add a table, the format to use depends on your firewall script:

- In scripts in native syntax, use:

```
table <table_address_family> <table_name> {
}
```

- In shell scripts, use:

```
nft add table <table_address_family> <table_name>
```

-

2.3.2. Basics of nftables chains

Tables consist of chains which in turn 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 to better organize rules.

If you want to add a base chain to a table, the format to use depends on your firewall script:

- In scripts in native syntax, use:

```
table <table_address_family> <table_name> {
  chain <chain_name> {
    type <type> hook <hook> priority <priority>
    policy <policy> ;
  }
}
```

- In shell scripts, use:

```
nft add chain <table_address_family> <table_name> <chain_name> { type <type> hook <hook> priority <priority> \; policy <policy> \; }
```

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

Both examples create **base chains**. To create a **regular chain**, do not set any parameters in the curly brackets.

Chain types
The following are the chain types and an overview with which address families and hooks you can use them:

Type	Address families	Hooks	Description
filter	all	all	Standard chain type
nat	**ip**, **ip6**, **inet**	**prerouting**, **input**, **output**, **postrouting**	Chains of this type perform native address translation based on connection tracking entries. Only the first packet traverses this chain type.
route	**ip**, **ip6**	**output**	Accepted packets that traverse this chain type cause a new route lookup if relevant parts of the IP header have changed.

Chain priorities
The priority parameter specifies the order in which packets traverse chains with the same hook value. You can set this parameter to an integer value or use a standard priority name.

The following matrix is an overview of the standard priority names and their numeric values, and with which address families and hooks you can use them:

Textual value	Numeric value	Address families	Hooks
raw	**-300**	**ip**, **ip6**, **inet**	all
mangle	**-150**	**ip**, **ip6**, **inet**	all
dstnat	**-100**	**ip**, **ip6**, **inet**	**prerouting**
	-300	**bridge**	**prerouting**
filter	**0**	**ip**, **ip6**, **inet**, **arp**, **netdev**	all
	-200	**bridge**	all
security	**50**	**ip**, **ip6**, **inet**	all
srcnat	**100**	**ip**, **ip6**, **inet**	**postrouting**
	300	**bridge**	**postrouting**
out	**100**	**bridge**	**output**

Chain policies

The chain policy defines whether **nftables** should accept or drop packets if rules in this chain do not specify any action. You can set one of the following policies in a chain:

- **accept** (default)
- **drop**

2.3.3. Basics of nftables rules

Rules define actions to perform on packets that pass a chain that contains this rule. If the rule also contains matching expressions, **nftables** performs the actions only if all previous expressions apply.

If you want to add a rule to a chain, the format to use depends on your firewall script:

- In scripts in native syntax, use:

```
table <table_address_family> <table_name> {
  chain <chain_name> {
    type <type> hook <hook> priority <priority> ; policy <policy> ;
      <rule>
  }
}
```

- In shell scripts, use:

```
nft add rule <table_address_family> <table_name> <chain_name> <rule>
```

This shell command appends the new rule at the end of the chain. If you prefer to add a rule at the beginning of the chain, use the **nft insert** command instead of **nft add**.

2.3.4. Managing tables, chains, and rules using nft commands

To manage an **nftables** firewall on the command line or in shell scripts, use the **nft** utility.

IMPORTANT

The commands in this procedure do not represent a typical workflow and are not optimized. This procedure only demonstrates how to use **nft** commands to manage tables, chains, and rules in general.

Procedure

1. Create a table named **nftables_svc** with the **inet** address family so that the table can process both IPv4 and IPv6 packets:

```
# nft add table inet nftables_svc
```

2. Add a base chain named **INPUT**, that processes incoming network traffic, to the **inet nftables_svc** table:

```
# nft add chain inet nftables_svc INPUT { type filter hook input priority filter \; policy accept \; }
```

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

3. Add rules to the **INPUT** chain. For example, allow incoming TCP traffic on port 22 and 443, and, as the last rule of the **INPUT** chain, reject other incoming traffic with an Internet Control Message Protocol (ICMP) port unreachable message:

```
# nft add rule inet nftables_svc INPUT tcp dport 22 accept
# nft add rule inet nftables_svc INPUT tcp dport 443 accept
# nft add rule inet nftables_svc INPUT reject with icmpx type port-unreachable
```

If you enter the **nft add rule** commands as shown, **nft** adds the rules in the same order to the chain as you run the commands.

4. Display the current rule set including handles:

```
# nft -a list table inet nftables_svc
table inet nftables_svc { # handle 13
  chain INPUT { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 443 accept # handle 3
    reject # handle 4
  }
}
```

5. Insert a rule before the existing rule with handle 3. For example, to insert a rule that allows TCP traffic on port 636, enter:

```
# nft insert rule inet nftables_svc INPUT position 3 tcp dport 636 accept
```

6. Append a rule after the existing rule with handle 3. For example, to insert a rule that allows TCP traffic on port 80, enter:

```
# nft add rule inet nftables_svc INPUT position 3 tcp dport 80 accept
```

7. Display the rule set again with handles. Verify that the later added rules have been added to the specified positions:

```
# nft -a list table inet nftables_svc
table inet nftables_svc { # handle 13
  chain INPUT { # 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
    reject # handle 4
  }
}
```

8. Remove the rule with handle 6:

```
# nft delete rule inet nftables_svc INPUT handle 6
```

To remove a rule, you must specify the handle.

9. Display the rule set, and verify that the removed rule is no longer present:

```
# nft -a list table inet nftables_svc
table inet nftables_svc { # handle 13
  chain INPUT { # 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
    reject # handle 4
  }
}
```

10. Remove all remaining rules from the **INPUT** chain:

```
# nft flush chain inet nftables_svc INPUT
```

11. Display the rule set, and verify that the **INPUT** chain is empty:

```
# nft list table inet nftables_svc
table inet nftables_svc {
  chain INPUT {
```

```
    type filter hook input priority filter; policy accept
  }
}
```

12. Delete the **INPUT** chain:

```
# nft delete chain inet nftables_svc INPUT
```

You can also use this command to delete chains that still contain rules.

13. Display the rule set, and verify that the **INPUT** chain has been deleted:

```
# nft list table inet nftables_svc
table inet nftables_svc {
}
```

14. Delete the **nftables_svc** table:

```
# nft delete table inet nftables_svc
```

You can also use this command to delete tables that still contain chains.

NOTE

To delete the entire rule set, use the **nft flush ruleset** command instead of manually deleting all rules, chains, and tables in separate commands.

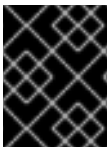
Additional resources

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. NAT types

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 (ISPs) 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.
Masquerading and SNAT are very similar to one another. 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.

Replace the source IP of packets leaving the host through the **ens3** interface to the IP set on **ens3**.

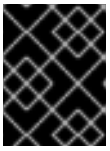
Procedure

1. Create a table:

```
# nft add table nat
```

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

```
# nft add chain nat 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 prevent the shell from interpreting 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 router then replaces the source IP of outgoing packets.

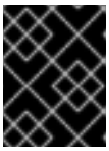
Procedure

1. Create a table:

```
# nft add table nat
```

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

```
# 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 prevent the shell from interpreting 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
```

Additional resources

- Forwarding incoming packets on a specific local port to a different host

2.4.4. Configuring destination NAT using nftables

Destination NAT (DNAT) enables you to redirect traffic on a router to a host that is not directly accessible from the internet.

For example, with DNAT the router redirects incoming traffic sent to port **80** and **443** to a web server with the IP address **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 prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming traffic to port **80** and **443** on the **ens3** interface of the router to the web server with the IP address **192.0.2.1**:

```
# nft add rule nat prerouting iifname ens3 tcp dport { 80, 443 } dnat to 192.0.2.1
```

4. Depending on your environment, add either a SNAT or masquerading rule to change the source address for packets returning from the web server to the sender:

   a. If the **ens3** interface uses a dynamic IP addresses, add a masquerading rule:

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

   b. If the **ens3** interface uses a static IP address, add a SNAT rule. For example, if the **ens3** uses the **198.51.100.1** IP address:

```
# nft add rule nat postrouting oifname "ens3" snat to 198.51.100.1
```

5. Enable packet forwarding:

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

Additional resources

- NAT types

2.4.5. Configuring a redirect using nftables

The **redirect** feature is a special case of destination network address translation (DNAT) that redirects packets to the local machine depending on the chain hook.

For example, you can redirect incoming and forwarded traffic sent to port **22** of the local host to port **2222**.

Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** chain to the table:

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

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

3. Add a rule to the **prerouting** chain that redirects incoming traffic on port **22** to port **2222**:

```
# nft add rule nat prerouting tcp dport 22 redirect to 2222
```

Additional resources

- NAT types

2.4.6. Configuring flowtable by using nftables

The **nftables** utility uses the **netfilter** framework to provide network address translation (NAT) for network traffic and provides the fastpath feature-based **flowtable** mechanism to accelerate packet forwarding.

The flowtable mechanism has the following features:

- Uses connection tracking to bypass the classic packet forwarding path.
- Avoids revisiting the routing table by bypassing the classic packet processing.
- Works only with TCP and UDP protocols.
- Hardware independent software fast path.

Procedure

1. Add an **example-table** table of **inet** family:

```
# nft add table inet <example-table>
```

2. Add an **example-flowtable** flowtable with **ingress** hook and **filter** as a priority type:

```
# nft add flowtable inet <example-table> <example-flowtable> { hook ingress priority filter \; devices = { enp1s0, enp7s0 } \; }
```

3. Add an **example-forwardchain** flow to the flowtable from a packet processing table:

```
# nft add chain inet <example-table> <example-forwardchain> { type filter hook forward priority filter \; }
```

This command adds a flowtable of **filter** type with **forward** hook and **filter** priority.

4. Add a rule with **established** connection tracking state to offload **example-flowtable** flow:

```
# nft add rule inet <example-table> <example-forwardchain> ct state established flow add @<example-flowtable>
```

Verification

- Verify the properties of **example-table**:

```
# nft list table inet <example-table>
table inet example-table {
    flowtable example-flowtable {
      hook ingress priority filter
          devices = { enp1s0, enp7s0 }
    }

    chain example-forwardchain {
type filter hook forward priority filter; policy accept;
ct state established flow add @example-flowtable
   }
}
```

Additional resources

- **nft(8)** man page

2.5. USING SETS IN NFTABLES COMMANDS

The **nftables** framework natively supports sets. You can use sets, for example, if a rule should match multiple IP addresses, port numbers, interfaces, or any other match criteria.

2.5.1. Using anonymous sets in nftables

An anonymous set contains comma-separated values enclosed in curly brackets, such as **{ 22, 80, 443 }**, that you use directly in a rule. You can use anonymous sets also for IP addresses and any other match criteria.

The drawback of anonymous sets is that if you want to change the set, you must replace the rule. For a dynamic solution, use named sets as described in Using named sets in nftables .

Prerequisites

- The **example_chain** chain and the **example_table** table in the **inet** family exists.

Procedure

1. For example, to add a rule to **example_chain** in **example_table** that allows incoming traffic to port **22**, **80**, and **443**:

```
# nft add rule inet example_table example_chain tcp dport { 22, 80, 443 } accept
```

2. Optional: 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, http, https } accept
  }
}
```

2.5.2. Using named sets in nftables

The **nftables** framework supports mutable named sets. A named set is a list or range of elements that you can use in multiple rules within a table. Another benefit over anonymous sets is that you can update a named set without replacing the rules that use the set.

When you create a named set, you must specify the type of elements the set contains. You can set the following types:

- **ipv4_addr** for a set that contains IPv4 addresses or ranges, such as **192.0.2.1** or **192.0.2.0/24**.
- **ipv6_addr** for a set that contains IPv6 addresses or ranges, such as **2001:db8:1::1** or **2001:db8:1::1/64**.
- **ether_addr** for a set that contains a list of media access control (MAC) addresses, such as **52:54:00:6b:66:42**.
- **inet_proto** for a set that contains a list of internet protocol types, such as **tcp**.
- **inet_service** for a set that contains a list of internet services, such as **ssh**.
- **mark** for a set that contains a list of packet marks. Packet marks can be any positive 32-bit integer value (**0** to **2147483647**).

Prerequisites

- The **example_chain** chain and the **example_table** table exists.

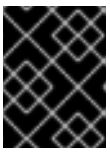
Procedure

1. Create an empty set. The following examples create a set for IPv4 addresses:
   - To create a set that can store multiple individual IPv4 addresses:

     ```
     # nft add set inet example_table example_set { type ipv4_addr \; }
     ```

   - To create a set that can store IPv4 address ranges:

     ```
     # nft add set inet example_table example_set { type ipv4_addr \; flags interval \; }
     ```

   IMPORTANT

   To prevent the shell from interpreting the semicolons as the end of the command, you must escape the semicolons with a backslash.

2. Optional: Create rules that use the set. For example, the following command adds a rule to the **example_chain** in the **example_table** that will drop all packets from IPv4 addresses in **example_set**.

   ```
   # nft add rule inet example_table example_chain ip saddr @example_set drop
   ```

   Because **example_set** is still empty, the rule has currently no effect.

3. Add IPv4 addresses to **example_set**:

- If you create a set that stores individual IPv4 addresses, enter:

```
# nft add element inet example_table example_set { 192.0.2.1, 192.0.2.2 }
```

- If you create a set that stores IPv4 ranges, enter:

```
# nft add element inet example_table example_set { 192.0.2.0-192.0.2.255 }
```

When you specify an IP address range, you can alternatively use the Classless Inter-Domain Routing (CIDR) notation, such as **192.0.2.0/24** in the above example.

2.5.3. Additional resources

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

2.6. USING VERDICT MAPS IN NFTABLES COMMANDS

Verdict maps, which are also known as dictionaries, enable **nft** to perform an action based on packet information by mapping match criteria to an action.

2.6.1. Using anonymous maps in nftables

An anonymous map is a **{ *match_criteria* : *action* }** statement that you use directly in a rule. The statement can contain multiple comma-separated mappings.

The drawback of an anonymous map is that if you want to change the map, you must replace the rule. For a dynamic solution, use named maps as described in Using named maps in nftables .

For example, you can use an anonymous map to route both TCP and UDP packets of the IPv4 and IPv6 protocol to different chains to count incoming TCP and UDP packets separately.

Procedure

1. Create a new table:

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

2. Create the **tcp_packets** chain in **example_table**:

```
# nft add chain inet example_table tcp_packets
```

3. Add a rule to **tcp_packets** that counts the traffic in this chain:

```
# nft add rule inet example_table tcp_packets counter
```

4. Create the **udp_packets** chain in **example_table**

```
# nft add chain inet example_table udp_packets
```

5. Add a rule to **udp_packets** that counts the traffic in this chain:

```
# nft add rule inet example_table udp_packets counter
```

6. Create a chain for incoming traffic. For example, to create a chain named **incoming_traffic** in **example_table** that filters incoming traffic:

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

7. Add a rule with an anonymous map to **incoming_traffic**:

```
# nft add rule inet example_table incoming_traffic ip protocol vmap { tcp : jump
tcp_packets, udp : jump udp_packets }
```

The anonymous map distinguishes the packets and sends them to the different counter chains based on their protocol.

8. To list the traffic counters, display **example_table**:

```
# nft list table inet example_table
table inet example_table {
  chain tcp_packets {
    counter packets 36379 bytes 2103816
  }

  chain udp_packets {
    counter packets 10 bytes 1559
  }

  chain incoming_traffic {
    type filter hook input priority filter; policy accept;
    ip protocol vmap { tcp : jump tcp_packets, udp : jump udp_packets }
  }
}
```

The counters in the **tcp_packets** and **udp_packets** chain display both the number of received packets and bytes.

2.6.2. Using named maps in nftables

The **nftables** framework supports named maps. You can use these maps in multiple rules within a table. Another benefit over anonymous maps is that you can update a named map without replacing the rules that use it.

When you create a named map, you must specify the type of elements:

- **ipv4_addr** for a map whose match part contains an IPv4 address, such as **192.0.2.1**.
- **ipv6_addr** for a map whose match part contains an IPv6 address, such as **2001:db8:1::1**.
- **ether_addr** for a map whose match part contains a media access control (MAC) address, such as **52:54:00:6b:66:42**.
- **inet_proto** for a map whose match part contains an internet protocol type, such as **tcp**.
- **inet_service** for a map whose match part contains an internet services name port number, such as **ssh** or **22**.

- **mark** for a map whose match part contains a packet mark. A packet mark can be any positive 32-bit integer value (**0** to **2147483647**).
- **counter** for a map whose match part contains a counter value. The counter value can be any positive 64-bit integer value.
- **quota** for a map whose match part contains a quota value. The quota value can be any positive 64-bit integer value.

For example, you can allow or drop incoming packets based on their source IP address. Using a named map, you require only a single rule to configure this scenario while the IP addresses and actions are dynamically stored in the map.

Procedure

1. Create a table. For example, to create a table named **example_table** that processes IPv4 packets:

```
# nft add table ip example_table
```

2. Create a chain. For example, to create a chain named **example_chain** in **example_table**:

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

IMPORTANT

To prevent the shell from interpreting the semicolons as the end of the command, you must escape the semicolons with a backslash.

3. Create an empty map. For example, to create a map for IPv4 addresses:

```
# nft add map ip example_table example_map { type ipv4_addr : verdict \; }
```

4. Create rules that use the map. For example, the following command adds a rule to **example_chain** in **example_table** that applies actions to IPv4 addresses which are both defined in **example_map**:

```
# nft add rule example_table example_chain ip saddr vmap @example_map
```

5. Add IPv4 addresses and corresponding actions to **example_map**:

```
# nft add element ip example_table example_map { 192.0.2.1 : accept, 192.0.2.2 : drop }
```

This example defines the mappings of IPv4 addresses to actions. In combination with the rule created above, the firewall accepts packet from **192.0.2.1** and drops packets from **192.0.2.2**.

6. Optional: Enhance the map by adding another IP address and action statement:

```
# nft add element ip example_table example_map { 192.0.2.3 : accept }
```

7. Optional: Remove an entry from the map:

```
# nft delete element ip example_table example_map { 192.0.2.1 }
```

8. Optional: Display the rule set:

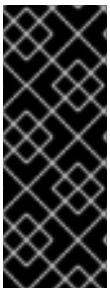
```
# nft list ruleset
table ip example_table {
  map example_map {
    type ipv4_addr : verdict
    elements = { 192.0.2.2 : drop, 192.0.2.3 : accept }
  }

  chain example_chain {
    type filter hook input priority filter; policy accept;
    ip saddr vmap @example_map
  }
}
```

2.6.3. Additional resources

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

2.7. EXAMPLE: PROTECTING A LAN AND DMZ USING AN NFTABLES SCRIPT

Use the **nftables** framework on a RHEL router to write and install a firewall script that protects the network clients in an internal LAN and a web server in a DMZ from unauthorized access from the internet and from other networks.



IMPORTANT

This example is only for demonstration purposes and describes a scenario with specific requirements.

Firewall scripts highly depend on the network infrastructure and security requirements. Use this example to learn the concepts of **nftables** firewalls when you write scripts for your own environment.

2.7.1. Network conditions

The network in this example has the following conditions:

- The router is connected to the following networks:
  - The internet through interface **enp1s0**
  - The internal LAN through interface **enp7s0**
  - The DMZ through **enp8s0**
- The internet interface of the router has both a static IPv4 address (**203.0.113.1**) and IPv6 address (**2001:db8:a::1**) assigned.
- The clients in the internal LAN use only private IPv4 addresses from the range **10.0.0.0/24**. Consequently, traffic from the LAN to the internet requires source network address translation (SNAT).

- The administrator PCs in the internal LAN use the IP addresses **10.0.0.100** and **10.0.0.200**.
- The DMZ uses public IP addresses from the ranges **198.51.100.0/24** and **2001:db8:b::/56**.
- The web server in the DMZ uses the IP addresses **198.51.100.5** and **2001:db8:b::5**.
- The router acts as a caching DNS server for hosts in the LAN and DMZ.

2.7.2. Security requirements to the firewall script

The following are the requirements to the **nftables** firewall in the example network:

- The router must be able to:
  - Recursively resolve DNS queries.
  - Perform all connections on the loopback interface.
- Clients in the internal LAN must be able to:
  - Query the caching DNS server running on the router.
  - Access the HTTPS server in the DMZ.
  - Access any HTTPS server on the internet.
- The PCs of the administrators must be able to access the router and every server in the DMZ using SSH.
- The web server in the DMZ must be able to:
  - Query the caching DNS server running on the router.
  - Access HTTPS servers on the internet to download updates.
- Hosts on the internet must be able to:
  - Access the HTTPS servers in the DMZ.
- Additionally, the following security requirements exists:
  - Connection attempts that are not explicitly allowed should be dropped.
  - Dropped packets should be logged.

2.7.3. Configuring logging of dropped packets to a file

By default, **systemd** logs kernel messages, such as for dropped packets, to the journal. Additionally, you can configure the **rsyslog** service to log such entries to a separate file. To ensure that the log file does not grow infinitely, configure a rotation policy.

Prerequisites

- The **rsyslog** package is installed.
- The **rsyslog** service is running.

Procedure

1. Create the **/etc/rsyslog.d/nftables.conf** file with the following content:

```
:msg, startswith, "nft drop" -/var/log/nftables.log
& stop
```

   Using this configuration, the **rsyslog** service logs dropped packets to the **/var/log/nftables.log** file instead of **/var/log/messages**.

2. Restart the **rsyslog** service:

```
# systemctl restart rsyslog
```

3. Create the **/etc/logrotate.d/nftables** file with the following content to rotate **/var/log/nftables.log** if the size exceeds 10 MB:

```
/var/log/nftables.log {
  size +10M
  maxage 30
  sharedscripts
  postrotate
    /usr/bin/systemctl kill -s HUP rsyslog.service >/dev/null 2>&1 || true
  endscript
}
```

   The **maxage 30** setting defines that **logrotate** removes rotated logs older than 30 days during the next rotation operation.

Additional resources

- **rsyslog.conf(5)** man page
- **logrotate(8)** man page

2.7.4. Writing and activating the nftables script

This example is an **nftables** firewall script that runs on a RHEL router and protects the clients in an internal LAN and a web server in a DMZ. For details about the network and the requirements for the firewall used in the example, see Network conditions and Security requirements to the firewall script.

WARNING

This **nftables** firewall script is only for demonstration purposes. Do not use it without adapting it to your environments and security requirements.

Prerequisites

- The network is configured as described in Network conditions.

Procedure

1. Create the **/etc/nftables/firewall.nft** script with the following content:

```
# Remove all rules
flush ruleset


# Table for both IPv4 and IPv6 rules
table inet nftables_svc {

  # Define variables for the interface name
  define INET_DEV = enp1s0
  define LAN_DEV  = enp7s0
  define DMZ_DEV  = enp8s0


  # Set with the IPv4 addresses of admin PCs
  set admin_pc_ipv4 {
    type ipv4_addr
    elements = { 10.0.0.100, 10.0.0.200 }
  }


  # Chain for incoming trafic. Default policy: drop
  chain INPUT {
    type filter hook input priority filter
    policy drop

    # Accept packets in established and related state, drop invalid packets
    ct state vmap { established:accept, related:accept, invalid:drop }

    # Accept incoming traffic on loopback interface
    iifname lo accept

    # Allow request from LAN and DMZ to local DNS server
    iifname { $LAN_DEV, $DMZ_DEV } meta l4proto { tcp, udp } th dport 53 accept

    # Allow admins PCs to access the router using SSH
    iifname $LAN_DEV ip saddr @admin_pc_ipv4 tcp dport 22 accept

    # Last action: Log blocked packets
    # (packets that were not accepted in previous rules in this chain)
    log prefix "nft drop IN : "
  }


  # Chain for outgoing traffic. Default policy: drop
  chain OUTPUT {
    type filter hook output priority filter
    policy drop

    # Accept packets in established and related state, drop invalid packets
    ct state vmap { established:accept, related:accept, invalid:drop }

    # Accept outgoing traffic on loopback interface
```

```
    oifname lo accept

    # Allow local DNS server to recursively resolve queries
    oifname $INET_DEV meta l4proto { tcp, udp } th dport 53 accept

    # Last action: Log blocked packets
    log prefix "nft drop OUT: "
  }


  # Chain for forwarding traffic. Default policy: drop
  chain FORWARD {
    type filter hook forward priority filter
    policy drop

    # Accept packets in established and related state, drop invalid packets
    ct state vmap { established:accept, related:accept, invalid:drop }

    # IPv4 access from LAN and internet to the HTTPS server in the DMZ
    iifname { $LAN_DEV, $INET_DEV } oifname $DMZ_DEV ip daddr 198.51.100.5 tcp dport
443 accept

    # IPv6 access from internet to the HTTPS server in the DMZ
    iifname $INET_DEV oifname $DMZ_DEV ip6 daddr 2001:db8:b::5 tcp dport 443 accept

    # Access from LAN and DMZ to HTTPS servers on the internet
    iifname { $LAN_DEV, $DMZ_DEV } oifname $INET_DEV tcp dport 443 accept

    # Last action: Log blocked packets
    log prefix "nft drop FWD: "
  }


  # Postrouting chain to handle SNAT
  chain postrouting {
    type nat hook postrouting priority srcnat; policy accept;

    # SNAT for IPv4 traffic from LAN to internet
    iifname $LAN_DEV oifname $INET_DEV snat ip to 203.0.113.1
  }
}
```

2. Include the **/etc/nftables/firewall.nft** script in the **/etc/sysconfig/nftables.conf** file:

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

3. Enable IPv4 forwarding:

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

4. Enable and start the **nftables** service:

```
# systemctl enable --now nftables
```

Verification

1. Optional: Verify the **nftables** rule set:

```
# nft list ruleset
...
```

2. Try to perform an access that the firewall prevents. For example, try to access the router using SSH from the DMZ:

```
# ssh router.example.com
ssh: connect to host router.example.com port 22: Network is unreachable
```

3. Depending on your logging settings, search:
   - The **systemd** journal for the blocked packets:

```
# journalctl -k -g "nft drop"
Oct 14 17:27:18 router kernel: nft drop IN : IN=enp8s0 OUT= MAC=...
SRC=198.51.100.5 DST=198.51.100.1 ... PROTO=TCP SPT=40464 DPT=22 ... SYN ...
```

   - The **/var/log/nftables.log** file for the blocked packets:

```
Oct 14 17:27:18 router kernel: nft drop IN : IN=enp8s0 OUT= MAC=...
SRC=198.51.100.5 DST=198.51.100.1 ... PROTO=TCP SPT=40464 DPT=22 ... SYN ...
```

2.8. CONFIGURING PORT FORWARDING USING NFTABLES

Port forwarding enables administrators to forward packets sent to a specific destination port to a different local or remote port.

For example, if your web server does not have a public IP address, you can set a port forwarding rule on your firewall that forwards incoming packets on port **80** and **443** on the firewall to the web server. With this firewall rule, users on the internet can access the web server using the IP or host name of the firewall.

2.8.1. Forwarding incoming packets to a different local port

You can use **nftables** to forward packets. For example, you can forward incoming IPv4 packets on port **8022** to port **22** on the local system.

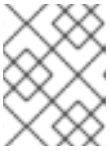
Procedure

1. Create a table named **nat** with the **ip** address family:

```
# nft add table ip nat
```

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

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



NOTE

Pass the **--** option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming packets on port **8022** to the local port **22**:

```
# nft add rule ip nat prerouting tcp dport 8022 redirect to :22
```

2.8.2. Forwarding incoming packets on a specific local port to a different host

You can use a destination network address translation (DNAT) rule to forward incoming packets on a local port to a remote host. This enables users on the internet to access a service that runs on a host with a private IP address.

For example, you can forward incoming IPv4 packets on the local port **443** to the same port number on the remote system with the **192.0.2.1** IP address.

Prerequisites

- You are logged in as the **root** user on the system that should forward the packets.

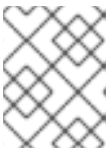
Procedure

1. Create a table named **nat** with the **ip** address family:

```
# nft add table ip nat
```

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

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



NOTE

Pass the **--** option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming packets on port **443** to the same port on **192.0.2.1**:

```
# nft add rule ip nat prerouting tcp dport 443 dnat to 192.0.2.1
```

4. Add a rule to the **postrouting** chain to masquerade outgoing traffic:

```
# nft add rule ip nat postrouting daddr 192.0.2.1 masquerade
```

5. Enable packet forwarding:

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

2.9. USING NFTABLES TO LIMIT THE AMOUNT OF CONNECTIONS

You can use **nftables** to limit the number of connections or to block IP addresses that attempt to establish a given amount of connections to prevent them from using too many system resources.

2.9.1. Limiting the number of connections using nftables

The **ct count** parameter of the **nft** utility enables administrators to limit the number of connections.

Prerequisites

- The base **example_chain** in **example_table** exists.

Procedure

1. Create a dynamic set for IPv4 addresses:

```
# nft add set inet example_table example_meter { type ipv4_addr\; flags dynamic \;}
```

2. Add a rule that allows only two simultaneous connections to the SSH port (22) from an IPv4 address and rejects all further connections from the same IP:

```
# nft add rule ip example_table example_chain tcp dport ssh meter example_meter { ip saddr ct count over 2 } counter reject
```

3. Optional: Display the set created in the previous step:

```
# nft list set inet example_table example_meter
table inet example_table {
  meter example_meter {
    type ipv4_addr
    size 65535
    elements = { 192.0.2.1 ct count over 2 , 192.0.2.2 ct count over 2  }
  }
}
```

The **elements** entry displays addresses that currently match the rule. In this example, **elements** lists IP addresses that have active connections to the SSH port. Note that the output does not display the number of active connections or if connections were rejected.

2.9.2. Blocking IP addresses that attempt more than ten new incoming TCP connections within one minute

You can temporarily block hosts that are establishing more than ten IPv4 TCP connections within one minute.

Procedure

1. Create the **filter** table with the **ip** address family:

```
# nft add table ip filter
```

2. Add the **input** chain to the **filter** table:

```
# nft add chain ip filter input { type filter hook input priority 0 \; }
```

3. Add a set named **denylist** to the **filter** table:

```
# nft add set ip filter denylist { type ipv4_addr \; flags dynamic, timeout \; timeout 5m \; }
```

This command creates a dynamic set for IPv4 addresses. The **timeout 5m** parameter defines that **nftables** automatically removes entries after five minutes to prevent that the set fills up with stale entries.

4. Add a rule that automatically adds the source IP address of hosts that attempt to establish more than ten new TCP connections within one minute to the **denylist** set:

```
# nft add rule ip filter input ip protocol tcp ct state new, untracked add @denylist { ip saddr limit rate over 10/minute } drop
```

Additional resources

- Using named sets in nftables

2.10. DEBUGGING NFTABLES RULES

The **nftables** framework provides different options for administrators to debug rules and if packets match them.

2.10.1. Creating a rule with a counter

To identify if a rule is matched, you can use a counter.

- For more information about a procedure that adds a counter to an existing rule, see Adding a counter to an existing rule.

Prerequisites

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

Procedure

1. Add a new rule with the **counter** parameter to the chain. The following example adds a rule with a counter that allows TCP traffic on port 22 and counts the packets and traffic that match this rule:

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

2. To display the counter values:

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

```
    tcp dport ssh counter packets 6872 bytes 105448565 accept
  }
}
```

2.10.2. Adding a counter to an existing rule

To identify if a rule is matched, you can use a counter.

- For more information about a procedure that adds a new rule with a counter, see Creating a rule with the counter.

Prerequisites

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

Procedure

1. Display the rules in the chain including their handles:

```
# nft --handle list chain inet example_table example_chain
table inet example_table {
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept # handle 4
  }
}
```

2. Add the counter by replacing the rule but with the **counter** parameter. The following example replaces the rule displayed in the previous step and adds a counter:

```
# nft replace rule inet example_table example_chain handle 4 tcp dport 22 counter accept
```

3. To display the counter values:

```
# nft list ruleset
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport ssh counter packets 6872 bytes 105448565 accept
  }
}
```

2.10.3. Monitoring packets that match an existing rule

The tracing feature in **nftables** in combination with the **nft monitor** command enables administrators to display packets that match a rule. You can enable tracing for a rule an use it to monitoring packets that match this rule.

Prerequisites

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

Procedure

1. Display the rules in the chain including their handles:

```
# nft --handle list chain inet example_table example_chain
table inet example_table {
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept # handle 4
  }
}
```

2. Add the tracing feature by replacing the rule but with the **meta nftrace set 1** parameters. The following example replaces the rule displayed in the previous step and enables tracing:

```
# nft replace rule inet example_table example_chain handle 4 tcp dport 22 meta nftrace set 1 accept
```

3. Use the **nft monitor** command to display the tracing. The following example filters the output of the command to display only entries that contain **inet example_table example_chain**:

```
# nft monitor | grep "inet example_table example_chain"
trace id 3c5eb15e inet example_table example_chain packet: iif "enp1s0" ether saddr 52:54:00:17:ff:e4 ether daddr 52:54:00:72:2f:6e ip saddr 192.0.2.1 ip daddr 192.0.2.2 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 49710 ip protocol tcp ip length 60 tcp sport 56728 tcp dport ssh tcp flags == syn tcp window 64240
trace id 3c5eb15e inet example_table example_chain rule tcp dport ssh nftrace set 1 accept (verdict accept)
...
```

WARNING

Depending on the number of rules with tracing enabled and the amount of matching traffic, the **nft monitor** command can display a lot of output. Use **grep** or other utilities to filter the output.

2.11. BACKING UP AND RESTORING THE NFTABLES RULE SET

You can backup **nftables** rules to a file and later restoring them. Also, administrators can use a file with the rules to, for example, transfer the rules to a different server.

2.11.1. Backing up the nftables rule set to a file

You can use the **nft** utility to back up the **nftables** rule set to a file.

Procedure

- To backup **nftables** rules:
  - In a format produced by **nft list ruleset** format:

```
# nft list ruleset > file.nft
```

- In JSON format:

```
# nft -j list ruleset > file.json
```

2.11.2. Restoring the nftables rule set from a file

You can restore the **nftables** rule set from a file.

Procedure

- To restore **nftables** rules:
  - If the file to restore is in the format produced by **nft list ruleset** or contains **nft** commands directly:

```
# nft -f file.nft
```

  - If the file to restore is in JSON format:

```
# nft -j -f file.json
```

2.12. ADDITIONAL RESOURCES

- Using nftables in Red Hat Enterprise Linux 8
- What comes after iptables? Its successor, of course: nftables
- Firewalld: The Future is nftables

CHAPTER 3. USING XDP-FILTER FOR HIGH-PERFORMANCE TRAFFIC FILTERING TO PREVENT DDOS ATTACKS

Compared to packet filters, such as **nftables**, Express Data Path (XDP) processes and drops network packets right at the network interface. Therefore, XDP determines the next step for the package before it reaches a firewall or other applications. As a result, XDP filters require less resources and can process network packets at a much higher rate than conventional packet filters to defend against distributed denial of service (DDoS) attacks. For example, during testing, Red Hat dropped 26 million network packets per second on a single core, which is significantly higher than the drop rate of **nftables** on the same hardware.

The **xdp-filter** utility allows or drops incoming network packets using XDP. You can create rules to filter traffic to or from specific:

- IP addresses
- MAC addresses
- Ports

Note that, even if **xdp-filter** has a significantly higher packet-processing rate, it does not have the same capabilities as, for example, **nftables**. Consider **xdp-filter** a conceptual utility to demonstrate packet filtering using XDP. Additionally, you can use the code of the utility for a better understanding of how to write your own XDP applications.

IMPORTANT

On other architectures than AMD and Intel 64-bit, the **xdp-filter** utility is provided as a Technology Preview only. Technology Preview features are not supported with Red Hat production Service Level Agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These previews provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

3.1. DROPPING NETWORK PACKETS THAT MATCH AN XDP-FILTER RULE

You can use **xdp-filter** to drop network packets:

- To a specific destination port
- From a specific IP address
- From a specific MAC address

The **allow** policy of **xdp-filter** defines that all traffic is allowed and the filter drops only network packets that match a particular rule. For example, use this method if you know the source IP addresses of packets you want to drop.

Prerequisites

- The **xdp-tools** package is installed.
- A network driver that supports XDP programs.

Procedure

1. Load **xdp-filter** to process incoming packets on a certain interface, such as **enp1s0**:

```
# xdp-filter load enp1s0
```

   By default, **xdp-filter** uses the **allow** policy, and the utility drops only traffic that matches any rule.

   Optionally, use the **-f *feature*** option to enable only particular features, such as **tcp**, **ipv4**, or **ethernet**. Loading only the required features instead of all of them increases the speed of packet processing. To enable multiple features, separate them with a comma.

   If the command fails with an error, the network driver does not support XDP programs.

2. Add rules to drop packets that match them. For example:
   - To drop incoming packets to port **22**, enter:

```
# xdp-filter port 22
```

     This command adds a rule that matches TCP and UDP traffic. To match only a particular protocol, use the **-p *protocol*** option.

   - To drop incoming packets from **192.0.2.1**, enter:

```
# xdp-filter ip 192.0.2.1 -m src
```

     Note that **xdp-filter** does not support IP ranges.

   - To drop incoming packets from MAC address **00:53:00:AA:07:BE**, enter:

```
# xdp-filter ether 00:53:00:AA:07:BE -m src
```

Verification

- Use the following command to display statistics about dropped and allowed packets:

```
# xdp-filter status
```

Additional resources

- **xdp-filter(8)** man page
- If you are a developer and interested in the code of **xdp-filter**, download and install the corresponding source RPM (SRPM) from the Red Hat Customer Portal.

3.2. DROPPING ALL NETWORK PACKETS EXCEPT THE ONES THAT MATCH AN XDP-FILTER RULE

You can use **xdp-filter** to allow only network packets:

- From and to a specific destination port
- From and to a specific IP address
- From and to specific MAC address

To do so, use the **deny** policy of **xdp-filter** which defines that the filter drops all network packets except the ones that match a particular rule. For example, use this method if you do not know the source IP addresses of packets you want to drop.

WARNING

If you set the default policy to **deny** when you load **xdp-filter** on an interface, the kernel immediately drops all packets from this interface until you create rules that allow certain traffic. To avoid being locked out from the system, enter the commands locally or connect through a different network interface to the host.

Prerequisites

- The **xdp-tools** package is installed.
- You are logged in to the host either locally or using a network interface for which you do not plan to filter the traffic.
- A network driver that supports XDP programs.

Procedure

1. Load **xdp-filter** to process packets on a certain interface, such as **enp1s0**:

```
# xdp-filter load enp1s0 -p deny
```

   Optionally, use the **-f *feature*** option to enable only particular features, such as **tcp**, **ipv4**, or **ethernet**. Loading only the required features instead of all of them increases the speed of packet processing. To enable multiple features, separate them with a comma.

   If the command fails with an error, the network driver does not support XDP programs.

2. Add rules to allow packets that match them. For example:
   - To allow packets to port **22**, enter:

```
# xdp-filter port 22
```

     This command adds a rule that matches TCP and UDP traffic. To match only a particular protocol, pass the **-p *protocol*** option to the command.

   - To allow packets to **192.0.2.1**, enter:

```
# xdp-filter ip 192.0.2.1
```

Note that **xdp-filter** does not support IP ranges.

- To allow packets to MAC address **00:53:00:AA:07:BE**, enter:

```
# xdp-filter ether 00:53:00:AA:07:BE
```

IMPORTANT

The **xdp-filter** utility does not support stateful packet inspection. This requires that you either do not set a mode using the **-m *mode*** option or you add explicit rules to allow incoming traffic that the machine receives in reply to outgoing traffic.

Verification

- Use the following command to display statistics about dropped and allowed packets:

```
# xdp-filter status
```

Additional resources

- **xdp-filter(8)** man page.
- If you are a developer and you are interested in the code of **xdp-filter**, download and install the corresponding source RPM (SRPM) from the Red Hat Customer Portal.