Red Hat Enterprise Linux 8

Securing networks

Configuring secured networks and network communication

Last Updated: 2020-09-24

Red Hat Enterprise Linux 8 Securing networks

Configuring secured networks and network communication

Legal Notice

Abstract

This title assists administrators with securing networks, connected machines, and network communication against various attacks.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

- For simple comments on specific passages:

  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.

  2. Use your mouse cursor to highlight the part of text that you want to comment on.

  3. Click the **Add Feedback** pop-up that appears below the highlighted text.

  4. Follow the displayed instructions.

- For submitting more complex feedback, create a Bugzilla ticket:

  1. Go to the Bugzilla website.

  2. As the Component, use **Documentation**.

  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.

  4. Click **Submit Bug**.

CHAPTER 1. USING SECURE COMMUNICATIONS BETWEEN TWO SYSTEMS WITH OPENSSH

SSH (Secure Shell) is a protocol which provides secure communications between two systems using a client-server architecture and allows users to log in to server host systems remotely. Unlike other remote communication protocols, such as FTP or Telnet, SSH encrypts the login session, which prevents intruders to collect unencrypted passwords from the connection.

Red Hat Enterprise Linux includes the basic **OpenSSH** packages: the general **openssh** package, the **openssh-server** package and the **openssh-clients** package. Note that the **OpenSSH** packages require the **OpenSSL** package **openssl-libs**, which installs several important cryptographic libraries that enable **OpenSSH** to provide encrypted communications.

1.1. SSH AND OPENSSH

SSH (Secure Shell) is a program for logging into a remote machine and executing commands on that machine. The SSH protocol provides secure encrypted communications between two untrusted hosts over an insecure network. You can also forward X11 connections and arbitrary TCP/IP ports over the secure channel.

The SSH protocol mitigates security threats, such as interception of communication between two systems and impersonation of a particular host, when you use it for remote shell login or file copying. This is because the SSH client and server use digital signatures to verify their identities. Additionally, all communication between the client and server systems is encrypted.

OpenSSH is an implementation of the SSH protocol supported by a number of Linux, UNIX, and similar operating systems. It includes the core files necessary for both the OpenSSH client and server. The OpenSSH suite consists of the following user-space tools:

- **ssh** is a remote login program (SSH client)

- **sshd** is an **OpenSSH** SSH daemon

- **scp** is a secure remote file copy program

- **sftp** is a secure file transfer program

- **ssh-agent** is an authentication agent for caching private keys

- **ssh-add** adds private key identities to **ssh-agent**

- **ssh-keygen** generates, manages, and converts authentication keys for **ssh**

- **ssh-copy-id** is a script that adds local public keys to the **authorized_keys** file on a remote SSH server

- **ssh-keyscan** - gathers SSH public host keys

Two versions of SSH currently exist: version 1, and the newer version 2. The **OpenSSH** suite in Red Hat Enterprise Linux 8 supports only SSH version 2, which has an enhanced key-exchange algorithm not vulnerable to known exploits in version 1.

OpenSSH, as one of the RHEL core cryptographic subsystems uses system-wide crypto policies. This ensures that weak cipher suites and cryptographic algorithms are disabled in the default configuration. To adjust the policy, the administrator must either use the **update-crypto-policies** command to make settings stricter or looser or manually opt-out of the system-wide crypto policies.

The **OpenSSH** suite uses two different sets of configuration files: those for client programs (that is, **ssh**, **scp**, and **sftp**), and those for the server (the **sshd** daemon). System-wide SSH configuration information is stored in the **/etc/ssh/** directory. User-specific SSH configuration information is stored in ~/**.ssh/** in the user's home directory. For a detailed list of OpenSSH configuration files, see the **FILES** section in the **sshd(8)** man page.

Additional resources

- Man pages for the **ssh** topic listed by the **man -k ssh** command.

- Using system-wide cryptographic policies.

1.2. CONFIGURING AND STARTING AN OPENSSH SERVER

Use the following procedure for a basic configuration that might be required for your environment and for starting an **OpenSSH** server. Note that after the default RHEL installation, the **sshd** daemon is already started and server host keys are automatically created.

Prerequisites

- The **openssh-server** package is installed.

Procedure

1. Start the **sshd** daemon in the current session and set it to start automatically at boot time:

   ```
   # systemctl start sshd
   # systemctl enable sshd
   ```

2. To specify different addresses than the default **0.0.0.0** (IPv4) or **::** (IPv6) for the **ListenAddress** directive in the **/etc/ssh/sshd_config** configuration file and to use a slower dynamic network configuration, add the dependency on the **network-online.target** target unit to the **sshd.service** unit file. To achieve this, create the **/etc/systemd/system/sshd.service.d/local.conf** file with the following content:

   ```
   [Unit]
   Wants=network-online.target
   After=network-online.target
   ```

3. Review if **OpenSSH** server settings in the **/etc/ssh/sshd_config** configuration file meet the requirements of your scenario.

4. Optionally, change the welcome message that your **OpenSSH** server displays before a client authenticates by editing the **/etc/issue** file, for example:

   ```
   Welcome to ssh-server.example.com
   Warning: By accessing this server, you agree to the referenced terms and conditions.
   ```

   Note that to change the message displayed after a successful login you have to edit the **/etc/motd** file on the server. See the **pam_motd** man page for more information.

5. Reload the **systemd** configuration to apply the changes:

   ```
   # systemctl daemon-reload
   ```

Verification steps

1. Check that the **sshd** daemon is running:

   ```
   # systemctl status sshd
   ● sshd.service - OpenSSH server daemon
     Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2019-11-18 14:59:58 CET; 6min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Main PID: 1149 (sshd)
      Tasks: 1 (limit: 11491)
     Memory: 1.9M
     CGroup: /system.slice/sshd.service
             └─1149 /usr/sbin/sshd -D -oCiphers=aes128-ctr,aes256-ctr,aes128-cbc,aes256-cbc -
   oMACs=hmac-sha2-256,>

   Nov 18 14:59:58 ssh-server-example.com systemd[1]: Starting OpenSSH server daemon...
   Nov 18 14:59:58 ssh-server-example.com sshd[1149]: Server listening on 0.0.0.0 port 22.
   Nov 18 14:59:58 ssh-server-example.com sshd[1149]: Server listening on :: port 22.
   Nov 18 14:59:58 ssh-server-example.com systemd[1]: Started OpenSSH server daemon.
   ```

2. Connect to the SSH server with an SSH client.

   ```
   # ssh user@ssh-server-example.com
   ECDSA key fingerprint is SHA256:dXbaS0RG/UzlTTku8GtXSz0S1++IPegSy31v3L/FAEc.
   Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
   Warning: Permanently added 'ssh-server-example.com' (ECDSA) to the list of known hosts.

   user@ssh-server-example.com's password:
   ```

Additional resources

- **sshd(8)** and **sshd_config(5)** man pages

1.3. USING KEY PAIRS INSTEAD OF PASSWORDS FOR SSH AUTHENTICATION

To improve system security even further, generate SSH key pairs and then enforce key-based authentication by disabling password authentication.

1.3.1. Setting an OpenSSH server for key-based authentication

Follow these steps to configure your OpenSSH server for enforcing key-based authentication.

Prerequisites

- The **openssh-server** package is installed.

- The **sshd** daemon is running on the server.

Procedure

1. Open the **/etc/ssh/sshd_config** configuration in a text editor, for example:

```
# vi /etc/ssh/sshd_config
```

2. Change the **PasswordAuthentication** option to **no**:

```
PasswordAuthentication no
```

On a system other than a new default installation, check that **PubkeyAuthentication no** has not been set and the **ChallengeResponseAuthentication** directive is set to **no**. If you are connected remotely, not using console or out-of-band access, test the key-based login process before disabling password authentication.

3. To use key-based authentication with NFS-mounted home directories, enable the **use_nfs_home_dirs** SELinux boolean:

```
# setsebool -P use_nfs_home_dirs 1
```

4. Reload the **sshd** daemon to apply the changes:

```
# systemctl reload sshd
```

Additional resources

- **sshd(8)**, **sshd_config(5)**, and **setsebool(8)** man pages

1.3.2. Generating SSH key pairs

Use this procedure to generate an SSH key pair on a local system and to copy the generated public key to an **OpenSSH** server. If the server is configured accordingly, you can log in to the **OpenSSH** server without providing any password.
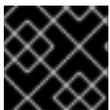
> **IMPORTANT**
>
> If you complete the following steps as **root**, only **root** is able to use the keys.

Procedure

1. To generate an ECDSA key pair for version 2 of the SSH protocol:

```
$ ssh-keygen -t ecdsa
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/joesec/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/joesec/.ssh/id_ecdsa.
Your public key has been saved in /home/joesec/.ssh/id_ecdsa.pub.
The key fingerprint is:
SHA256:Q/x+qms4j7PCQ0qFd09iZEFHA+SqwBKRNaU72oZfaCl
joesec@localhost.example.com
The key's randomart image is:
+---[ECDSA 256]---+
|.oo..o=++        |
|.. o .oo .       |
|. .. o. o        |
```

```
|....o.+...      |
|o.oo.o +S .     |
|.=.+.   .o      |
|E.*+. . . .     |
|.=..+ +.. o     |
| . oo*+o.       |
+----[SHA256]-----+
```

You can also generate an RSA key pair by using the **-t rsa** option with the **ssh-keygen** command or an Ed25519 key pair by entering the **ssh-keygen -t ed25519** command.

2. To copy the public key to a remote machine:

   > $ ssh-copy-id *joesec@ssh-server-example.com*
   > /usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
   > ...
   > Number of key(s) added: 1
   >
   > Now try logging into the machine, with: "ssh 'joesec@ssh-server-example.com'" and check to make sure that only the key(s) you wanted were added.

   If you do not use the **ssh-agent** program in your session, the previous command copies the most recently modified ~/**.ssh**/**id*.pub** public key if it is not yet installed. To specify another public-key file or to prioritize keys in files over keys cached in memory by **ssh-agent**, use the **ssh-copy-id** command with the **-i** option.

> **NOTE**
>
> If you reinstall your system and want to keep previously generated key pairs, back up the ~/**.ssh**/ directory. After reinstalling, copy it back to your home directory. You can do this for all users on your system, including **root**.

Verification steps

1. Log in to the OpenSSH server without providing any password:

   > $ ssh *joesec@ssh-server-example.com*
   > Welcome message.
   > ...
   > Last login: Mon Nov 18 18:28:42 2019 from ::1

Additional resources

- **ssh-keygen(1)** and **ssh-copy-id(1)** man pages

1.4. USING SSH KEYS STORED ON A SMART CARD

Red Hat Enterprise Linux 8 enables you to use RSA and ECDSA keys stored on a smart card on OpenSSH clients. Use this procedure to enable authentication using a smart card instead of using a password.

Prerequisites

- On the client side, the **opensc** package is installed and the **pcscd** service is running.

Procedure

1. List all keys provided by the OpenSC PKCS #11 module including their PKCS #11 URIs and save the output to the *keys.pub* file:

   ```
   $ ssh-keygen -D pkcs11: > keys.pub
   $ ssh-keygen -D pkcs11:
   ssh-rsa AAAAB3NzaC1yc2E...KKZMzcQZzx
   pkcs11:id=%02;object=SIGN%20pubkey;token=SSH%20key;manufacturer=piv_II?module-
   path=/usr/lib64/pkcs11/opensc-pkcs11.so
   ecdsa-sha2-nistp256 AAA...J0hkYnnsM=
   pkcs11:id=%01;object=PIV%20AUTH%20pubkey;token=SSH%20key;manufacturer=piv_II?
   module-path=/usr/lib64/pkcs11/opensc-pkcs11.so
   ```

2. To enable authentication using a smart card on a remote server (*example.com*), transfer the public key to the remote server. Use the **ssh-copy-id** command with *keys.pub* created in the previous step:

   ```
   $ ssh-copy-id -f -i keys.pub username@example.com
   ```

3. To connect to *example.com* using the ECDSA key from the output of the **ssh-keygen -D** command in step 1, you can use just a subset of the URI, which uniquely references your key, for example:

   ```
   $ ssh -i "pkcs11:id=%01?module-path=/usr/lib64/pkcs11/opensc-pkcs11.so" example.com
   Enter PIN for 'SSH key':
   [example.com] $
   ```

4. You can use the same URI string in the **~/.ssh/config** file to make the configuration permanent:

   ```
   $ cat ~/.ssh/config
   IdentityFile "pkcs11:id=%01?module-path=/usr/lib64/pkcs11/opensc-pkcs11.so"
   $ ssh example.com
   Enter PIN for 'SSH key':
   [example.com] $
   ```

   Because OpenSSH uses the **p11-kit-proxy** wrapper and the OpenSC PKCS #11 module is registered to PKCS#11 Kit, you can simplify the previous commands:

   ```
   $ ssh -i "pkcs11:id=%01" example.com
   Enter PIN for 'SSH key':
   [example.com] $
   ```

If you skip the **id=** part of a PKCS #11 URI, OpenSSH loads all keys that are available in the proxy module. This can reduce the amount of typing required:

```
$ ssh -i pkcs11: example.com
Enter PIN for 'SSH key':
[example.com] $
```

Additional resources

- Fedora 28: Better smart card support in OpenSSH

- **p11-kit(8)** man page

- **ssh(1)** man page

- **ssh-keygen(1)** man page

- **opensc.conf(5)** man page

- **pcscd(8)** man page

1.5. MAKING OPENSSH MORE SECURE

The following tips help you to increase security when using OpenSSH. Note that changes in the **/etc/ssh/sshd_config** OpenSSH configuration file require reloading the **sshd** daemon to take effect:

```
# systemctl reload sshd
```
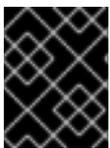
> **IMPORTANT**
>
> The majority of security hardening configuration changes reduce compatibility with clients that do not support up-to-date algorithms or cipher suites.

Disabling insecure connection protocols

- To make SSH truly effective, prevent the use of insecure connection protocols that are replaced by the **OpenSSH** suite. Otherwise, a user's password might be protected using SSH for one session only to be captured later when logging in using Telnet. For this reason, consider disabling insecure protocols, such as telnet, rsh, rlogin, and ftp.

Enabling key-based authentication and disabling password-based authentication

- Disabling passwords for authentication and allowing only key pairs reduces the attack surface and it also might save users' time. On clients, generate key pairs using the **ssh-keygen** tool and use the **ssh-copy-id** utility to copy public keys from clients on the **OpenSSH** server. To disable password-based authentication on your OpenSSH server, edit **/etc/ssh/sshd_config** and change the **PasswordAuthentication** option to **no**:

  ```
  PasswordAuthentication no
  ```

Key types

- Although the **ssh-keygen** command generates a pair of RSA keys by default, you can instruct it to generate ECDSA or Ed25519 keys by using the **-t** option. The ECDSA (Elliptic Curve Digital Signature Algorithm) offers better performance than RSA at the equivalent symmetric key strength. It also generates shorter keys. The Ed25519 public-key algorithm is an implementation of twisted Edwards curves that is more secure and also faster than RSA, DSA, and ECDSA. OpenSSH creates RSA, ECDSA, and Ed25519 server host keys automatically if they are missing. To configure the host key creation in RHEL 8, use the **sshd-keygen@.service** instantiated service. For example, to disable the automatic creation of the RSA key type:

  ```
  # systemctl mask sshd-keygen@rsa.service
  ```

- To exclude particular key types for SSH connections, comment out the relevant lines in **/etc/ssh/sshd_config**, and reload the **sshd** service. For example, to allow only Ed25519 host keys:

```
# HostKey /etc/ssh/ssh_host_rsa_key
# HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key
```

Non-default port

- By default, the **sshd** daemon listens on TCP port 22. Changing the port reduces the exposure of the system to attacks based on automated network scanning and thus increase security through obscurity. You can specify the port using the **Port** directive in the **/etc/ssh/sshd_config** configuration file.
  You also have to update the default SELinux policy to allow the use of a non-default port. To do so, use the **semanage** tool from the **policycoreutils-python-utils** package:

```
# semanage port -a -t ssh_port_t -p tcp port_number
```

Furthermore, update **firewalld** configuration:

```
# firewall-cmd --add-port port_number/tcp
# firewall-cmd --runtime-to-permanent
```

In the previous commands, replace *port_number* with the new port number specified using the **Port** directive.

No root login

- If your particular use case does not require the possibility of logging in as the root user, you should consider setting the **PermitRootLogin** configuration directive to **no** in the **/etc/ssh/sshd_config** file. By disabling the possibility of logging in as the root user, the administrator can audit which users run what privileged commands after they log in as regular users and then gain root rights.
  Alternatively, set **PermitRootLogin** to **prohibit-password**:

```
PermitRootLogin prohibit-password
```

This enforces the use of key-based authentication instead of the use of passwords for logging in as root and reduces risks by preventing brute-force attacks.

Using the X Security extension

- The X server in Red Hat Enterprise Linux clients does not provide the X Security extension. Therefore, clients cannot request another security layer when connecting to untrusted SSH servers with X11 forwarding. Most applications are not able to run with this extension enabled anyway.
  By default, the **ForwardX11Trusted** option in the **/etc/ssh/ssh_config.d/05-redhat.conf** file is set to **yes**, and there is no difference between the **ssh -X remote_machine** (untrusted host) and **ssh -Y remote_machine** (trusted host) command.

  If your scenario does not require the X11 forwarding feature at all, set the **X11Forwarding** directive in the **/etc/ssh/sshd_config** configuration file to **no**.

Restricting access to specific users, groups, or domains

- The **AllowUsers** and **AllowGroups** directives in the **/etc/ssh/sshd_config** configuration file server enable you to permit only certain users, domains, or groups to connect to your OpenSSH server. You can combine **AllowUsers** and **AllowGroups** to restrict access more precisely, for example:

  ```
  AllowUsers *@192.168.1.*,*@10.0.0.*,!*@192.168.1.2
  AllowGroups example-group
  ```

  The previous configuration lines accept connections from all users from systems in 192.168.1.* and 10.0.0.* subnets except from the system with the 192.168.1.2 address. All users must be in the **example-group** group. The OpenSSH server denies all other connections.

  Note that using whitelists (directives starting with Allow) is more secure than using blacklists (options starting with Deny) because whitelists block also new unauthorized users or groups.

Changing system-wide cryptographic policies

- **OpenSSH** uses RHEL system-wide cryptographic policies, and the default system-wide cryptographic policy level offers secure settings for current threat models. To make your cryptographic settings more strict, change the current policy level:

  ```
  # update-crypto-policies --set FUTURE
  Setting system policy to FUTURE
  ```

- To opt-out of the system-wide crypto policies for your **OpenSSH** server, uncomment the line with the **CRYPTO_POLICY=** variable in the **/etc/sysconfig/sshd** file. After this change, values that you specify in the **Ciphers**, **MACs**, **KexAlgoritms**, and **GSSAPIKexAlgorithms** sections in the **/etc/ssh/sshd_config** file are not overridden. Note that this task requires deep expertise in configuring cryptographic options.

- See Using system-wide cryptographic policies in the RHEL 8 Security hardening title for more information.

Additional resources

- **sshd_config(5)**, **ssh-keygen(1)**, **crypto-policies(7)**, and **update-crypto-policies(8)** man pages

1.6. CONNECTING TO A REMOTE SERVER USING AN SSH JUMP HOST

Use this procedure for connecting to a remote server through an intermediary server, also called jump host.

Prerequisites

- A jump host accepts SSH connections from your system.

- A remote server accepts SSH connections only from the jump host.

Procedure

1. Define the jump host by editing the ~/**.ssh/config** file, for example:

> Host *jump-server1*
> HostName *jump1.example.com*

2. Add the remote server jump configuration with the **ProxyJump** directive to ~/**.ssh**/**config**, for example:

> Host *remote-server*
> HostName *remote1.example.com*
> ProxyJump *jump-server1*

3. Connect to the remote server through the jump server:

> $ ssh *remote-server*

The previous command is equivalent to the **ssh -J jump-server1 remote-server** command if you omit the configuration steps 1 and 2.

> NOTE
>
> You can specify more jump servers and you can also skip adding host definitions to the configurations file when you provide their complete host names, for example:
>
> > $ ssh -J *jump1.example.com,jump2.example.com,jump3.example.com*
> > *remote1.example.com*
>
> Change the host name–only notation in the previous command if the user names or SSH ports on the jump servers differ from the names and ports on the remote server, for example:
>
> > $ ssh -J
> > *johndoe@jump1.example.com:75,johndoe@jump2.example.com:75,johndoe@jump3.e*
> > *xample.com:75 joesec@remote1.example.com:220*

Additional resources

- **ssh_config(5)** and **ssh(1)** man pages

1.7. CONNECTING TO REMOTE MACHINES WITH SSH KEYS USING SSH-AGENT

To avoid entering a passphrase each time you initiate an SSH connection, you can use the **ssh-agent** utility to cache the private SSH key. The private key and the passphrase remain secure.

Prerequisites

- You have a remote host with SSH daemon running and reachable through the network.

- You know the IP address or hostname and credentials to log in to the remote host.

- You have generated an SSH key pair with a passphrase and transferred the public key to the remote machine. For more information, see Generating SSH key pairs .

Procedure

1. Optional: Verify you can use the key to authenticate to the remote host:

   a. Connect to the remote host using SSH:

   ```
   $ ssh example.user1@198.51.100.1 hostname
   ```

   b. Enter the passphrase you set while creating the key to grant access to the private key.

   ```
   $ ssh example.user1@198.51.100.1 hostname
    host.example.com
   ```

2. Start the **ssh-agent**.

   ```
   $ eval $(ssh-agent)
   Agent pid 20062
   ```

3. Add the key to **ssh-agent**.

   ```
   $ ssh-add ~/.ssh/id_rsa
   Enter passphrase for ~/.ssh/id_rsa:
   Identity added: ~/.ssh/id_rsa (example.user0@198.51.100.12)
   ```

Verification steps

- Optional: Log in to the host machine using SSH.

   ```
   $ ssh example.user1@198.51.100.1

   Last login: Mon Sep 14 12:56:37 2020
   ```

   Note that you did not have to enter the passphrase.

1.8. ADDITIONAL RESOURCES

For more information on configuring and connecting to **OpenSSH** servers and clients on Red Hat Enterprise Linux, see the resources listed below.

Installed documentation

- **sshd(8)** man page documents available command-line options and provides a complete list of supported configuration files and directories.

- **ssh(1)** man page provides a complete list of available command-line options and supported configuration files and directories.

- **scp(1)** man page provides a more detailed description of the **scp** utility and its usage.

- **sftp(1)** man page provides a more detailed description of the **sftp** utility and its usage.

- **ssh-keygen(1)** man page documents in detail the use of the **ssh-keygen** utility to generate, manage, and convert authentication keys used by ssh.

- **ssh-copy-id(1)** man page describes the use of the **ssh-copy-id** script.

- **ssh_config(5)** man page documents available SSH client configuration options.

- **sshd_config(5)** man page provides a full description of available SSH daemon configuration options.

- **update-crypto-policies(8)** man page provides guidance on managing system-wide cryptographic policies

- **crypto-policies(7)** man page provides an overview of system-wide cryptographic policy levels

Online documentation

- OpenSSH Home Page - contains further documentation, frequently asked questions, links to the mailing lists, bug reports, and other useful resources.

- Configuring SELinux for applications and services with non-standard configurations - you can apply analogous procedures for OpenSSH in a non-standard configuration with SELinux in enforcing mode.

- Controlling network traffic using firewalld - provides guidance on updating **firewalld** settings after changing an SSH port

CHAPTER 2. PLANNING AND IMPLEMENTING TLS

TLS (Transport Layer Security) is a cryptographic protocol used to secure network communications. When hardening system security settings by configuring preferred key-exchange protocols, authentication methods, and encryption algorithms, it is necessary to bear in mind that the broader the range of supported clients, the lower the resulting security. Conversely, strict security settings lead to limited compatibility with clients, which can result in some users being locked out of the system. Be sure to target the strictest available configuration and only relax it when it is required for compatibility reasons.

2.1. SSL AND TLS PROTOCOLS

The Secure Sockets Layer (SSL) protocol was originally developed by Netscape Corporation to provide a mechanism for secure communication over the Internet. Subsequently, the protocol was adopted by the Internet Engineering Task Force (IETF) and renamed to Transport Layer Security (TLS).

The TLS protocol sits between an application protocol layer and a reliable transport layer, such as TCP/IP. It is independent of the application protocol and can thus be layered underneath many different protocols, for example: HTTP, FTP, SMTP, and so on.

Protocol version	Usage recommendation
SSL v2	Do not use. Has serious security vulnerabilities. Removed from the core crypto libraries since RHEL 7.
SSL v3	Do not use. Has serious security vulnerabilities. Removed from the core crypto libraries since RHEL 8.
TLS 1.0	Not recommended to use. Has known issues that cannot be mitigated in a way that guarantees interoperability, and does not support modern cipher suites. Enabled only in the **LEGACY** system-wide cryptographic policy profile.
TLS 1.1	Use for interoperability purposes where needed. Does not support modern cipher suites. Enabled only in the **LEGACY** policy.
TLS 1.2	Supports the modern AEAD cipher suites. This version is enabled in all system-wide crypto policies, but optional parts of this protocol contain vulnerabilities and TLS 1.2 also allows outdated algorithms.
TLS 1.3	Recommended version. TLS 1.3 removes known problematic options, provides additional privacy by encrypting more of the negotiation handshake and can be faster thanks usage of more efficient modern cryptographic algorithms. TLS 1.3 is also enabled in all system-wide crypto policies.

Additional resources

- IETF: The Transport Layer Security (TLS) Protocol Version 1.3

2.2. SECURITY CONSIDERATIONS FOR TLS IN RHEL 8

In RHEL 8, cryptography-related considerations are significantly simplified thanks to the system-wide

crypto policies. The **DEFAULT** crypto policy allows only TLS 1.2 and 1.3. To allow your system to negotiate connections using the earlier versions of TLS, you need to either opt out from following crypto policies in an application or switch to the **LEGACY** policy with the **update-crypto-policies** command. See Using system-wide cryptographic policies for more information.

The default settings provided by libraries included in RHEL 8 are secure enough for most deployments. The TLS implementations use secure algorithms where possible while not preventing connections from or to legacy clients or servers. Apply hardened settings in environments with strict security requirements where legacy clients or servers that do not support secure algorithms or protocols are not expected or allowed to connect.

The most straightforward way to harden your TLS configuration is switching the system-wide cryptographic policy level to **FUTURE** using the **update-crypto-policies --set FUTURE** command.

If you decide to not follow RHEL system-wide crypto policies, use the following recommendations for preferred protocols, cipher suites, and key lengths on your custom configuration:

2.2.1. Protocols

The latest version of TLS provides the best security mechanism. Unless you have a compelling reason to include support for older versions of TLS, allow your systems to negotiate connections using at least TLS version 1.2. Note that despite that RHEL 8 supports TLS version 1.3, not all features of this protocol are fully supported by RHEL 8 components. For example, the 0-RTT (Zero Round Trip Time) feature, which reduces connection latency, is not yet fully supported by Apache or Nginx web servers.

2.2.2. Cipher suites

Modern, more secure cipher suites should be preferred to old, insecure ones. Always disable the use of eNULL and aNULL cipher suites, which do not offer any encryption or authentication at all. If at all possible, ciphers suites based on RC4 or HMAC-MD5, which have serious shortcomings, should also be disabled. The same applies to the so-called export cipher suites, which have been intentionally made weaker, and thus are easy to break.

While not immediately insecure, cipher suites that offer less than 128 bits of security should not be considered for their short useful life. Algorithms that use 128 bits of security or more can be expected to be unbreakable for at least several years, and are thus strongly recommended. Note that while 3DES ciphers advertise the use of 168 bits, they actually offer 112 bits of security.

Always give preference to cipher suites that support (perfect) forward secrecy (PFS), which ensures the confidentiality of encrypted data even in case the server key is compromised. This rules out the fast RSA key exchange, but allows for the use of ECDHE and DHE. Of the two, ECDHE is the faster and therefore the preferred choice.

You should also give preference to AEAD ciphers, such as AES-GCM, before CBC-mode ciphers as they are not vulnerable to padding oracle attacks. Additionally, in many cases, AES-GCM is faster than AES in CBC mode, especially when the hardware has cryptographic accelerators for AES.

Note also that when using the ECDHE key exchange with ECDSA certificates, the transaction is even faster than pure RSA key exchange. To provide support for legacy clients, you can install two pairs of certificates and keys on a server: one with ECDSA keys (for new clients) and one with RSA keys (for legacy ones).

2.2.3. Public key length

When using RSA keys, always prefer key lengths of at least 3072 bits signed by at least SHA-256, which is sufficiently large for true 128 bits of security.

> **WARNING**
>
> The security of your system is only as strong as the weakest link in the chain. For example, a strong cipher alone does not guarantee good security. The keys and the certificates are just as important, as well as the hash functions and keys used by the Certification Authority (CA) to sign your keys.

Additional resources

- System-wide crypto policies in RHEL 8 .

- **update-crypto-policies(8)** man page

2.3. HARDENING TLS CONFIGURATION IN APPLICATIONS

In Red Hat Enterprise Linux 8, system-wide crypto policies provide a convenient way to ensure that your applications using cryptographic libraries do not allow known insecure protocols, ciphers, or algorithms.

If you want to harden your TLS-related configuration with your customized cryptographic settings, you can use the cryptographic configuration options described in this section, and override the system-wide crypto policies just in the minimum required amount.

Regardless of the configuration you choose to use, always make sure to mandate that your server application enforces *server-side cipher order*, so that the cipher suite to be used is determined by the order you configure.

2.3.1. Configuring the Apache HTTP server

The **Apache HTTP Server** can use both **OpenSSL** and **NSS** libraries for its TLS needs. Red Hat Enterprise Linux 8 provides the **mod_ssl** functionality through eponymous packages:

```
# yum install mod_ssl
```

The **mod_ssl** package installs the **/etc/httpd/conf.d/ssl.conf** configuration file, which can be used to modify the TLS-related settings of the **Apache HTTP Server**.

Install the **httpd-manual** package to obtain complete documentation for the **Apache HTTP Server**, including TLS configuration. The directives available in the **/etc/httpd/conf.d/ssl.conf** configuration file are described in detail in /usr/share/httpd/manual/mod/mod_ssl.html. Examples of various settings are in /usr/share/httpd/manual/ssl/ssl_howto.html.

When modifying the settings in the **/etc/httpd/conf.d/ssl.conf** configuration file, be sure to consider the following three directives at the minimum:

SSLProtocol

Use this directive to specify the version of TLS or SSL you want to allow.

SSLCipherSuite

Use this directive to specify your preferred cipher suite or disable the ones you want to disallow.

SSLHonorCipherOrder

Uncomment and set this directive to **on** to ensure that the connecting clients adhere to the order of ciphers you specified.

For example, to use only the TLS 1.2 and 1.3 protocol:

```
SSLProtocol          all -SSLv3 -TLSv1 -TLSv1.1
```

2.3.2. Configuring the `Nginx` HTTP and proxy server

To enable TLS 1.3 support in **Nginx**, add the **TLSv1.3** value to the **ssl_protocols** option in the **server** section of the **/etc/nginx/nginx.conf** configuration file:

```
server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    ....
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers
    ....
}
```

2.3.3. Configuring the Dovecot mail server

To configure your installation of the **Dovecot** mail server to use TLS, modify the **/etc/dovecot/conf.d/10-ssl.conf** configuration file. You can find an explanation of some of the basic configuration directives available in that file in the /usr/share/doc/dovecot/wiki/SSL.DovecotConfiguration.txt file, which is installed along with the standard installation of **Dovecot**.

When modifying the settings in the **/etc/dovecot/conf.d/10-ssl.conf** configuration file, be sure to consider the following three directives at the minimum:

ssl_protocols

Use this directive to specify the version of TLS or SSL you want to allow or disable.

ssl_cipher_list

Use this directive to specify your preferred cipher suites or disable the ones you want to disallow.

ssl_prefer_server_ciphers

Uncomment and set this directive to **yes** to ensure that the connecting clients adhere to the order of ciphers you specified.

For example, the following line in **/etc/dovecot/conf.d/10-ssl.conf** allows only TLS 1.1 and later:

```
ssl_protocols = !SSLv2 !SSLv3 !TLSv1
```

Additional resources

For more information about TLS configuration and related topics, see the resources listed below.

- **config(5)** man page describes the format of the **/etc/ssl/openssl.conf** configuration file.

- **ciphers(1)** man page includes a list of available **OpenSSL** keywords and cipher strings.

- Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)

- Mozilla SSL Configuration Generator can help to create configuration files for **Apache** or **Nginx** with secure settings that disable known vulnerable protocols, ciphers, and hashing algorithms.

- SSL Server Test verifies that your configuration meets modern security requirements.

CHAPTER 3. CONFIGURING A VPN WITH IPSEC

In Red Hat Enterprise Linux 8, a virtual private network (VPN) can be configured using the **IPsec** protocol, which is supported by the **Libreswan** application.

3.1. LIBRESWAN AS AN IPSEC VPN IMPLEMENTATION

In Red Hat Enterprise Linux 8, a Virtual Private Network (VPN) can be configured using the **IPsec** protocol, which is supported by the **Libreswan** application. **Libreswan** is a continuation of the **Openswan** application, and many examples from the **Openswan** documentation are interchangeable with **Libreswan**.

The **IPsec** protocol for a VPN is configured using the Internet Key Exchange (**IKE**) protocol. The terms IPsec and IKE are used interchangeably. An IPsec VPN is also called an IKE VPN, IKEv2 VPN, XAUTH VPN, Cisco VPN or IKE/IPsec VPN. A variant of an IPsec VPN that also uses the Level 2 Tunneling Protocol (**L2TP**) is usually called an L2TP/IPsec VPN, which requires the Optional channel **xl2tpd** application.

Libreswan is an open-source, user-space **IKE** implementation. **IKE** v1 and v2 are implemented as a user-level daemon. The IKE protocol is also encrypted. The **IPsec** protocol is implemented by the Linux kernel, and **Libreswan** configures the kernel to add and remove VPN tunnel configurations.

The **IKE** protocol uses UDP port 500 and 4500. The **IPsec** protocol consists of two protocols:

- Encapsulated Security Payload (**ESP**), which has protocol number 50.

- Authenticated Header (**AH**), which has protocol number 51.

The **AH** protocol is not recommended for use. Users of **AH** are recommended to migrate to **ESP** with null encryption.

The **IPsec** protocol provides two modes of operation:

- **Tunnel Mode** (the default)

- **Transport Mode**.

You can configure the kernel with IPsec without IKE. This is called **Manual Keying**. You can also configure manual keying using the **ip xfrm** commands, however, this is strongly discouraged for security reasons. **Libreswan** interfaces with the Linux kernel using netlink. Packet encryption and decryption happen in the Linux kernel.

Libreswan uses the Network Security Services (**NSS**) cryptographic library. Both **Libreswan** and **NSS** are certified for use with the *Federal Information Processing Standard* (**FIPS**) Publication 140-2.

> IMPORTANT
>
> **IKE**/**IPsec** VPNs, implemented by **Libreswan** and the Linux kernel, is the only VPN technology recommended for use in Red Hat Enterprise Linux 8. Do not use any other VPN technology without understanding the risks of doing so.

In Red Hat Enterprise Linux 8, **Libreswan** follows **system-wide cryptographic policies** by default. This ensures that **Libreswan** uses secure settings for current threat models including **IKEv2** as a default protocol. See Using system-wide crypto policies for more information.

Libreswan does not use the terms "source" and "destination" or "server" and "client" because IKE/IPsec are peer to peer protocols. Instead, it uses the terms "left" and "right" to refer to end points (the hosts). This also allows you to use the same configuration on both end points in most cases. However, administrators usually choose to always use "left" for the local host and "right" for the remote host.

3.2. INSTALLING LIBRESWAN

This procedure describes the steps for installing and starting the **Libreswan** IPsec/IKE VPN implementation.

Prerequisites

- The **AppStream** repository is enabled.

Procedure

1. Install the **libreswan** packages:

    ```
    # yum install libreswan
    ```

2. If you are re-installing **Libreswan**, remove its old database files:

    ```
    # systemctl stop ipsec
    # rm /etc/ipsec.d/*db
    ```

3. Start the **ipsec** service, and enable the service to be started automatically on boot:

    ```
    # systemctl enable ipsec --now
    ```

4. Configure the firewall to allow 500 and 4500/UDP ports for the IKE, ESP, and AH protocols by adding the **ipsec** service:

    ```
    # firewall-cmd --add-service="ipsec"
    # firewall-cmd --runtime-to-permanent
    ```

3.3. CREATING A HOST-TO-HOST VPN

To configure **Libreswan** to create a host-to-host **IPsec** VPN between two hosts referred to as *left* and *right*, enter the following commands on both of the hosts:

Procedure

1. Generate an RSA key pair on each host:

    ```
    # ipsec newhostkey --output /etc/ipsec.d/hostkey.secrets
    ```

2. The previous step returned the generated key's **ckaid**. Use that **ckaid** with the following command on *left*, for example:

    ```
    # ipsec showhostkey --left --ckaid 2d3ea57b61c9419dfd6cf43a1eb6cb306c0e857d
    ```

The output of the previous command generated the **leftrsasigkey=** line required for the configuration. Do the same on the second host (*right*):

```
# ipsec showhostkey --right --ckaid a9e1f6ce9ecd3608c24e8f701318383f41798f03
```

3. In the **/etc/ipsec.d/** directory, create a new **my_host-to-host.conf** file. Write the RSA host keys from the output of the **ipsec showhostkey** commands in the previous step to the new file. For example:

```
conn mytunnel
    leftid=@west
    left=192.1.2.23
    leftrsasigkey=0sAQOrlo+hOafUZDICQmXFrje/oZm [...] W2n417C/4urYHQkCvuIQ==
    rightid=@east
    right=192.1.2.45
    rightrsasigkey=0sAQO3fwC6nSSGgt64DWiYZzuHbc4 [...] D/v8t5YTQ==
    authby=rsasig
```

4. After importing keys, restart the **ipsec** service:

```
# systemctl restart ipsec
```

5. Start **Libreswan**:

```
# ipsec setup start
```

6. Load the connection:

```
# ipsec auto --add mytunnel
```

7. Establish the tunnel:

```
# ipsec auto --up mytunnel
```

8. To automatically start the tunnel when the **ipsec** service is started, add the following line to the connection definition:

```
auto=start
```

3.4. CONFIGURING A SITE-TO-SITE VPN

To create a site-to-site **IPsec** VPN, by joining two networks, an **IPsec** tunnel between the two hosts, is created. The hosts thus act as the end points, which are configured to permit traffic from one or more subnets to pass through. Therefore you can think of the host as gateways to the remote portion of the network.

The configuration of the site-to-site VPN only differs from the host-to-host VPN in that one or more networks or subnets must be specified in the configuration file.

Prerequisites

- A host-to-host VPN is already configured.

Procedure

1. Copy the file with the configuration of your host-to-host VPN to a new file, for example:

   ```
   # cp /etc/ipsec.d/my_host-to-host.conf /etc/ipsec.d/my_site-to-site.conf
   ```

2. Add the subnet configuration to the file created in the previous step, for example:

   ```
   conn mysubnet
       also=mytunnel
       leftsubnet=192.0.1.0/24
       rightsubnet=192.0.2.0/24
       auto=start

   conn mysubnet6
       also=mytunnel
       leftsubnet=2001:db8:0:1::/64
       rightsubnet=2001:db8:0:2::/64
       auto=start

   # the following part of the configuration file is the same for both host-to-host and site-to-site
   connections:

   conn mytunnel
       leftid=@west
       left=192.1.2.23
       leftrsasigkey=0sAQOrlo+hOafUZDICQmXFrje/oZm [...] W2n417C/4urYHQkCvuIQ==
       rightid=@east
       right=192.1.2.45
       rightrsasigkey=0sAQO3fwC6nSSGgt64DWiYZzuHbc4 [...] D/v8t5YTQ==
       authby=rsasig
   ```

3.5. CONFIGURING A REMOTE ACCESS VPN

Road warriors are traveling users with mobile clients with a dynamically assigned IP address, such as laptops. The mobile clients authenticate using certificates.

The following example shows configuration for **IKEv2**, and it avoids using the **IKEv1** XAUTH protocol.

On the server:

```
conn roadwarriors
    ikev2=insist
    # Support (roaming) MOBIKE clients (RFC 4555)
    mobike=yes
    fragmentation=yes
    left=1.2.3.4
    # if access to the LAN is given, enable this, otherwise use 0.0.0.0/0
    # leftsubnet=10.10.0.0/16
    leftsubnet=0.0.0.0/0
    leftcert=gw.example.com
    leftid=%fromcert
    leftxauthserver=yes
    leftmodecfgserver=yes
    right=%any
```

```
# trust our own Certificate Agency
rightca=%same
# pick an IP address pool to assign to remote users
# 100.64.0.0/16 prevents RFC1918 clashes when remote users are behind NAT
rightaddresspool=100.64.13.100-100.64.13.254
# if you want remote clients to use some local DNS zones and servers
modecfgdns="1.2.3.4, 5.6.7.8"
modecfgdomains="internal.company.com, corp"
rightxauthclient=yes
rightmodecfgclient=yes
authby=rsasig
# optionally, run the client X.509 ID through pam to allow/deny client
# pam-authorize=yes
# load connection, don't initiate
auto=add
# kill vanished roadwarriors
dpddelay=1m
dpdtimeout=5m
dpdaction=clear
```

On the mobile client, the road warrior's device, use a slight variation of the previous configuration:

```
conn to-vpn-server
    ikev2=insist
    # pick up our dynamic IP
    left=%defaultroute
    leftsubnet=0.0.0.0/0
    leftcert=myname.example.com
    leftid=%fromcert
    leftmodecfgclient=yes
    # right can also be a DNS hostname
    right=1.2.3.4
    # if access to the remote LAN is required, enable this, otherwise use 0.0.0.0/0
    # rightsubnet=10.10.0.0/16
    rightsubnet=0.0.0.0/0
    fragmentation=yes
    # trust our own Certificate Agency
    rightca=%same
    authby=rsasig
    # allow narrowing to the server's suggested assigned IP and remote subnet
    narrowing=yes
    # Support (roaming) MOBIKE clients (RFC 4555)
    mobike=yes
    # Initiate connection
    auto=start
```

3.6. CONFIGURING A MESH VPN

A mesh VPN network, which is also known as an *any-to-any* VPN, is a network where all nodes communicate using **IPsec**. The configuration allows for exceptions for nodes that cannot use **IPsec**. The mesh VPN network can be configured in two ways:

- To require **IPsec**.

- To prefer **IPsec** but allow a fallback to clear-text communication.

Authentication between the nodes can be based on X.509 certificates or on DNS Security Extensions (DNSSEC).

The following procedure uses X.509 certificates. These certificates can be generated using any kind of Certificate Authority (CA) management system, such as the Dogtag Certificate System. Dogtag assumes that the certificates for each node are available in the PKCS #12 format (.p12 files), which contain the private key, the node certificate, and the Root CA certificate used to validate other nodes' X.509 certificates.

Each node has an identical configuration with the exception of its X.509 certificate. This allows for adding new nodes without reconfiguring any of the existing nodes in the network. The PKCS #12 files require a "friendly name", for which we use the name "node" so that the configuration files referencing the friendly name can be identical for all nodes.

Prerequisites

- **Libreswan** is installed, and the **ipsec** service is started on each node.

Procedure

1. On each node, import PKCS #12 files. This step requires the password used to generate the PKCS #12 files:

   ```
   # ipsec import nodeXXX.p12
   ```

2. Create the following three connection definitions for the **IPsec required** (private), **IPsec optional** (private-or-clear), and **No IPsec** (clear) profiles:

   ```
   # cat /etc/ipsec.d/mesh.conf
   conn clear
    auto=ondemand
    type=passthrough
    authby=never
    left=%defaultroute
    right=%group

   conn private
    auto=ondemand
    type=transport
    authby=rsasig
    failureshunt=drop
    negotiationshunt=drop
    # left
    left=%defaultroute
    leftcert=nodeXXXX
    leftid=%fromcert
        leftrsasigkey=%cert
    # right
    rightrsasigkey=%cert
    rightid=%fromcert
    right=%opportunisticgroup

   conn private-or-clear
    auto=ondemand
    type=transport
   ```

```
authby=rsasig
failureshunt=passthrough
negotiationshunt=passthrough
# left
left=%defaultroute
leftcert=nodeXXXX
leftid=%fromcert
      leftrsasigkey=%cert
# right
rightrsasigkey=%cert
rightid=%fromcert
right=%opportunisticgroup
```

3. Add the IP address of the network in the proper category. For example, if all nodes reside in the 10.15.0.0/16 network, and all nodes should mandate **IPsec** encryption:

   ```
   # echo "10.15.0.0/16" >> /etc/ipsec.d/policies/private
   ```

4. To allow certain nodes, for example, 10.15.34.0/24, to work with and without **IPsec**, add those nodes to the private-or-clear group using:

   ```
   # echo "10.15.34.0/24" >> /etc/ipsec.d/policies/private-or-clear
   ```

5. To define a host, for example, 10.15.1.2, that is not capable of **IPsec** into the clear group, use:

   ```
   # echo "10.15.1.2/32" >> /etc/ipsec.d/policies/clear
   ```

   The files in the **/etc/ipsec.d/policies** directory can be created from a template for each new node, or can be provisioned using Puppet or Ansible.

   Note that every node has the same list of exceptions or different traffic flow expectations. Two nodes, therefore, might not be able to communicate because one requires **IPsec** and the other cannot use **IPsec**.

6. Restart the node to add it to the configured mesh:

   ```
   # systemctl restart ipsec
   ```

7. Once you finish with the addition of nodes, a **ping** command is sufficient to open an **IPsec** tunnel. To see which tunnels a node has opened:

   ```
   # ipsec trafficstatus
   ```

3.7. METHODS OF AUTHENTICATION USED IN LIBRESWAN

You can use the following methods for authentication of end points:

- *Pre-Shared Keys* (PSK) is the simplest authentication method. PSKs should consist of random characters and have a length of at least 20 characters. In FIPS mode, PSKs need to comply to a minimum strength requirement depending on the integrity algorithm used. It is recommended not to use PSKs shorter than 64 random characters.

- *Raw RSA keys* are commonly used for static host-to-host or subnet-to-subnet **IPsec** configurations. The hosts are manually configured with each other's public RSA key. This

method does not scale well when dozens or more hosts all need to setup **IPsec** tunnels to each other.

- *X.509 certificates* are commonly used for large-scale deployments where there are many hosts that need to connect to a common **IPsec** gateway. A central *certificate authority* (CA) is used to sign RSA certificates for hosts or users. This central CA is responsible for relaying trust, including the revocations of individual hosts or users.

- *NULL authentication* is used to gain mesh encryption without authentication. It protects against passive attacks but does not protect against active attacks. However, since **IKEv2** allows asymmetrical authentication methods, NULL authentication can also be used for internet scale opportunistic IPsec, where clients authenticate the server, but servers do not authenticate the client. This model is similar to secure websites using **TLS**.

Protection against quantum computers

In addition to these authentication methods, you can use the *Postquantum Preshared Keys* (PPK) method to protect against possible attacks by quantum computers. Individual clients or groups of clients can use their own PPK by specifying a (PPKID) that corresponds to an out-of-band configured PreShared Key.

Using **IKEv1** with PreShared Keys provided protection against quantum attackers. The redesign of **IKEv2** does not offer this protection natively. **Libreswan** offers the use of *Postquantum Preshared Keys* (PPK) to protect **IKEv2** connections against quantum attacks.

To enable optional PPK support, add **ppk=yes** to the connection definition. To require PPK, add **ppk=insist**. Then, each client can be given a PPK ID with a secret value that is communicated out-of-band (and preferably quantum safe). The PPK's should be very strong in randomness and not be based on dictionary words. The PPK ID and PPK data itself are stored in **ipsec.secrets**, for example:

```
@west @east : PPKS "user1" "thestringismeanttobearandomstr"
```

The **PPKS** option refers to static PPKs. An experimental function uses one-time-pad based Dynamic PPKs. Upon each connection, a new part of a one-time pad is used as the PPK. When used, that part of the dynamic PPK inside the file is overwritten with zeroes to prevent re-use. If there is no more one-time-pad material left, the connection fails. See the **ipsec.secrets(5)** man page for more information.

> **WARNING**
>
> The implementation of dynamic PPKs is provided as a Technology Preview, and this functionality should be used with caution.

3.8. DEPLOYING A FIPS-COMPLIANT IPSEC VPN

Use this procedure to deploy a FIPS-compliant IPsec VPN solution based on Libreswan. The following steps also enable you to identify which cryptographic algorithms are available and which are disabled for Libreswan in FIPS mode.

Prerequisites

- The **AppStream** repository is enabled.

Procedure

1. Install the **libreswan** packages:

   ```
   # yum install libreswan
   ```

2. If you are re-installing **Libreswan**, remove its old NSS database:

   ```
   # systemctl stop ipsec
   # rm /etc/ipsec.d/*db
   ```

3. Start the **ipsec** service, and enable the service to be started automatically on boot:

   ```
   # systemctl enable ipsec --now
   ```

4. Configure the firewall to allow 500 and 4500/UDP ports for the IKE, ESP, and AH protocols by adding the **ipsec** service:

   ```
   # firewall-cmd --add-service="ipsec"
   # firewall-cmd --runtime-to-permanent
   ```

5. Switch the system to FIPS mode in RHEL 8:

   ```
   # fips-mode-setup --enable
   ```

6. Restart your system to allow the kernel to switch to FIPS mode:

   ```
   # reboot
   ```

Verification steps

1. To confirm Libreswan is running in FIPS mode:

   ```
   # ipsec whack --fipsstatus
   000 FIPS mode enabled
   ```

2. Alternatively, check entries for the **ipsec** unit in the **systemd** journal:

   ```
   $ journalctl -u ipsec
   ...
   Jan 22 11:26:50 localhost.localdomain pluto[3076]: FIPS Product: YES
   Jan 22 11:26:50 localhost.localdomain pluto[3076]: FIPS Kernel: YES
   Jan 22 11:26:50 localhost.localdomain pluto[3076]: FIPS Mode: YES
   ```

3. To see the available algorithms in FIPS mode:

   ```
   # ipsec pluto --selftest 2>&1 | head -11
   FIPS Product: YES
   FIPS Kernel: YES
   FIPS Mode: YES
   NSS DB directory: sql:/etc/ipsec.d
   Initializing NSS
   ```

```
Opening NSS database "sql:/etc/ipsec.d" read-only
NSS initialized
NSS crypto library initialized
FIPS HMAC integrity support [enabled]
FIPS mode enabled for pluto daemon
NSS library is running in FIPS mode
FIPS HMAC integrity verification self-test passed
```

4. To query disabled algorithms in FIPS mode:

```
# ipsec pluto --selftest 2>&1 | grep disabled
Encryption algorithm CAMELLIA_CTR disabled; not FIPS compliant
Encryption algorithm CAMELLIA_CBC disabled; not FIPS compliant
Encryption algorithm SERPENT_CBC disabled; not FIPS compliant
Encryption algorithm TWOFISH_CBC disabled; not FIPS compliant
Encryption algorithm TWOFISH_SSH disabled; not FIPS compliant
Encryption algorithm NULL disabled; not FIPS compliant
Encryption algorithm CHACHA20_POLY1305 disabled; not FIPS compliant
Hash algorithm MD5 disabled; not FIPS compliant
PRF algorithm HMAC_MD5 disabled; not FIPS compliant
PRF algorithm AES_XCBC disabled; not FIPS compliant
Integrity algorithm HMAC_MD5_96 disabled; not FIPS compliant
Integrity algorithm HMAC_SHA2_256_TRUNCBUG disabled; not FIPS compliant
Integrity algorithm AES_XCBC_96 disabled; not FIPS compliant
DH algorithm MODP1024 disabled; not FIPS compliant
DH algorithm MODP1536 disabled; not FIPS compliant
DH algorithm DH31 disabled; not FIPS compliant
```

5. To list all allowed algorithms and ciphers in FIPS mode:

```
# ipsec pluto --selftest 2>&1 | grep ESP | grep FIPS | sed "s/^.*FIPS//"
{256,192,*128}  aes_ccm, aes_ccm_c
{256,192,*128}  aes_ccm_b
{256,192,*128}  aes_ccm_a
[*192]  3des
{256,192,*128}  aes_gcm, aes_gcm_c
{256,192,*128}  aes_gcm_b
{256,192,*128}  aes_gcm_a
{256,192,*128}  aesctr
{256,192,*128}  aes
{256,192,*128}  aes_gmac
sha, sha1, sha1_96, hmac_sha1
sha512, sha2_512, sha2_512_256, hmac_sha2_512
sha384, sha2_384, sha2_384_192, hmac_sha2_384
sha2, sha256, sha2_256, sha2_256_128, hmac_sha2_256
aes_cmac
null
null, dh0
dh14
dh15
dh16
dh17
dh18
```

ecp_256, ecp256
ecp_384, ecp384
ecp_521, ecp521

Additional resources

- Using system-wide cryptographic policies

3.9. PROTECTING THE IPSEC NSS DATABASE BY A PASSWORD

By default, the IPsec service creates its Network Security Services (NSS) database with an empty password during the first start. Add password protection by using the following steps.

> **NOTE**
>
> In the previous releases of RHEL up to version 6.6, you had to protect the IPsec NSS database with a password to meet the FIPS 140-2 requirements because the NSS cryptographic libraries were certified for the FIPS 140-2 Level 2 standard. In RHEL 8, NIST certified NSS to Level 1 of this standard, and this status does not require password protection for the database.

Prerequisite

- The **/etc/ipsec.d** directory contains NSS database files.

Procedure

1. Enable password protection for the **NSS** database for **Libreswan**:

   ```
   # certutil -N -d sql:/etc/ipsec.d
   Enter Password or Pin for "NSS Certificate DB":
   Enter a password which will be used to encrypt your keys.
   The password should be at least 8 characters long,
   and should contain at least one non-alphabetic character.

   Enter new password:
   ```

2. Create the **/etc/ipsec.d/nsspassword** file containing the password you have set in the previous step, for example:

   ```
   # cat /etc/ipsec.d/nsspassword
   NSS Certificate DB:MyStrongPasswordHere
   ```

   Note that the **nsspassword** file use the following syntax:

   ```
   token_1_name:the_password
   token_2_name:the_password
   ```

   The default NSS software token is **NSS Certificate DB**. If your system is running in FIPS mode, the name of the token is **NSS FIPS 140-2 Certificate DB**.

3. Depending on your scenario, either start or restart the **ipsec** service after you finish the **nsspassword** file:

```
# systemctl restart ipsec
```

Verification steps

1. Check that the **ipsec** service is running after you have added a non-empty password to its NSS database:

   ```
   # systemctl status ipsec
   ● ipsec.service - Internet Key Exchange (IKE) Protocol Daemon for IPsec
     Loaded: loaded (/usr/lib/systemd/system/ipsec.service; enabled; vendor preset: disable>
     Active: active (running)...
   ```

2. Optionally, check that the **Journal** log contains entries confirming a successful initialization:

   ```
   # journalctl -u ipsec
   ...
   pluto[23001]: NSS DB directory: sql:/etc/ipsec.d
   pluto[23001]: Initializing NSS
   pluto[23001]: Opening NSS database "sql:/etc/ipsec.d" read-only
   pluto[23001]: NSS Password from file "/etc/ipsec.d/nsspassword" for token "NSS Certificate
   DB" with length 20 passed to NSS
   pluto[23001]: NSS crypto library initialized
   ...
   ```

Additional resources

- The **certutil(1)** man page.

- For more information about certifications related to FIPS 140-2, see the Government Standards Knowledgebase article.

3.10. CONFIGURING IPSEC CONNECTIONS THAT OPT OUT OF THE SYSTEM-WIDE CRYPTO POLICIES

Overriding system-wide crypto-policies for a connection

The RHEL system-wide cryptographic policies create a special connection called **%default**. This connection contains the default values for the **ikev2**, **esp**, and **ike** options. However, you can override the default values by specifying the mentioned option in the connection configuration file.

For example, the following configuration allows connections that use IKEv1 with AES and SHA-1 or SHA-2, and IPsec (ESP) with either AES-GCM or AES-CBC:

```
conn MyExample
  ...
  ikev2=never
  ike=aes-sha2,aes-sha1;modp2048
  esp=aes_gcm,aes-sha2,aes-sha1
  ...
```

Note that AES-GCM is available for IPsec (ESP) and for IKEv2, but not for IKEv1.

Disabling system-wide crypto policies for all connections

To disable system-wide crypto policies for all IPsec connections, comment out the following line in the **/etc/ipsec.conf** file:

> include /etc/crypto-policies/back-ends/libreswan.config

Then add the **ikev2=never** option to your connection configuration file.

Additional resources

- See [Using system-wide cryptographic policies](#) for more information.

3.11. RELATED INFORMATION

The following resources provide additional information regarding **Libreswan** and the **ipsec** daemon.

Installed documentation

- **ipsec(8)** man page — Describes command options for **ipsec**.

- **ipsec.conf(5)** man page — Contains information on configuring **ipsec**.

- **ipsec.secrets(5)** man page — Describes the format of the **ipsec.secrets** file.

- **ipsec_auto(8)** man page — Describes the use of the **auto** command-line client for manipulating Libreswan IPsec connections established using automatic exchanges of keys.

- **ipsec_rsasigkey(8)** man page — Describes the tool used to generate RSA signature keys.

- **/usr/share/doc/libreswan-*version*/**

Online documentation

https://libreswan.org
> The website of the upstream project.

https://libreswan.org/wiki
> The Libreswan Project Wiki.

https://libreswan.org/man/
> All Libreswan man pages.

[NIST Special Publication 800-77: Guide to IPsec VPNs](#)
> Practical guidance to organizations on implementing security services based on IPsec.

CHAPTER 4. CONFIGURING MACSEC

The following section provides information on how to configure **Media Control Access Security** (**MACsec**), which is an 802.1AE IEEE standard security technology for secure communication in all traffic on Ethernet links.

4.1. INTRODUCTION TO MACSEC

Media Access Control Security (**MACsec**, IEEE 802.1AE) encrypts and authenticates all traffic in LANs with the GCM-AES-128 algorithm. **MACsec** can protect not only **IP** but also Address Resolution Protocol (ARP), Neighbor Discovery (ND), or **DHCP**. While **IPsec** operates on the network layer (layer 3) and **SSL** or **TLS** on the application layer (layer 7), **MACsec** operates in the data link layer (layer 2). Combine **MACsec** with security protocols for other networking layers to take advantage of different security features that these standards provide.

4.2. USING MACSEC WITH NMCLI TOOL

This procedure shows how to configure **MACsec** with **nmcli** tool.

Prerequisites

- The **NetworkManager** must be running.

- You already have a 16-byte hexadecimal CAK (**$MKA_CAK**) and a 32-byte hexadecimal CKN (**$MKA_CKN**).

Procedure

```
~]# nmcli connection add type macsec \
  con-name test-macsec+ ifname macsec0 \
  connection.autoconnect no \
  macsec.parent enp1s0 macsec.mode psk \
  macsec.mka-cak $MKA_CAK \
  macsec.mka-ckn $MKA_CKN

~]# nmcli connection up test-macsec+
```

After this step, the *macsec0* device is configured and can be used for networking.

4.3. USING MACSEC WITH WPA_SUPPLICANT

This procedure shows how to enable **MACsec** with a switch that performs authentication using a pre-shared Connectivity Association Key/CAK Name (CAK/CKN) pair.

Procedure

1. Create a CAK/CKN pair. For example, the following command generates a 16-byte key in hexadecimal notation:

   ```
   ~]$ dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%02x"'
   ```

2. Create the **wpa_supplicant.conf** configuration file and add the following lines to it:

```
ctrl_interface=/var/run/wpa_supplicant
eapol_version=3
ap_scan=0
fast_reauth=1

network={
    key_mgmt=NONE
    eapol_flags=0
    macsec_policy=1

    mka_cak=0011... # 16 bytes hexadecimal
    mka_ckn=2233... # 32 bytes hexadecimal
}
```

Use the values from the previous step to complete the **mka_cak** and **mka_ckn** lines in the **wpa_supplicant.conf** configuration file.

For more information, see the **wpa_supplicant.conf(5)** man page.

3. Assuming you are using *wlp61s0* to connect to your network, start **wpa_supplicant** using the following command:

```
~]# wpa_supplicant -i wlp61s0 -Dmacsec_linux -c wpa_supplicant.conf
```

4.4. RELATED INFORMATION

For more details, see the What's new in MACsec: setting up MACsec using wpa_supplicant and (optionally) NetworkManager article. In addition, see the MACsec: a different solution to encrypt network traffic article for more information about the architecture of a **MACsec** network, use case scenarios, and configuration examples.

CHAPTER 5. 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.

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

5.1. 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 to configure a firewall on workstations. The utility is easy to use and covers the typical use cases for this scenario.

- **nftables**: Use the **nftables** utility to set up complex firewalls, such as for a whole network.

- **iptables**: The **iptables** utility is deprecated in Red Hat Enterprise Linux 8. Use instead **nftables**.

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

5.2. GETTING STARTED WITH FIREWALLD

5.2.1. firewalld

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.

Additional resources

- **firewalld(1)** man page

5.2.2. Zones

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

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

- **NetworkManager**

- **firewall-config** tool

- **firewall-cmd** command-line tool

- The RHEL web console

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

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

block

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

dmz

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

drop

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

external

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

home

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

internal

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

public

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

trusted

All network connections are accepted.

work

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

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

NOTE

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

Additional resources

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

5.2.3. Predefined services

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

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

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

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

Additional resources

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

5.3. INSTALLING THE FIREWALL-CONFIG GUI CONFIGURATION TOOL

To use the **firewall-config** GUI configuration tool, install the **firewall-config** package.

Procedure

1. Enter the following command as **root**:

   ```
   # yum install firewall-config
   ```

   Alternatively, in **GNOME, use the Super key and type `Software** to launch the **Software Sources** application. Type **firewall** to the search box, which appears after selecting the search button in the top-right corner. Select the **Firewall** item from the search results, and click on the **Install** button.

2. To run **firewall-config**, use either the **firewall-config** command or press the **Super** key to enter the **Activities Overview**, type **firewall**, and press **Enter**.

5.4. VIEWING THE CURRENT STATUS AND SETTINGS OF FIREWALLD

5.4.1. Viewing the current status of firewalld

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

Procedure

1. To see the status of the service:

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

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

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

Additional resources

It is important to know how **firewalld** is set up and which rules are in force before you try to edit the settings. To display the firewall settings, see Section 5.4.2, "Viewing current firewalld settings"

5.4.2. Viewing current firewalld settings

5.4.2.1. Viewing allowed services using GUI

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

Alternatively, to start the graphical firewall configuration tool using the command-line, enter the following command:

```
$ firewall-config
```

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

5.4.2.2. Viewing firewalld settings using CLI

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

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

To list all the relevant information for the default zone:

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

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

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

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

```
# firewall-cmd --help

Usage: firewall-cmd [OPTIONS...]

General Options
  -h, --help          Prints a short help text and exists
  -V, --version        Print the version string of firewalld
  -q, --quiet          Do not print status messages

Status Options
  --state             Return and print firewalld state
  --reload             Reload firewall and keep state information
... [trimmed for clarity]
```

For example, to see which services are allowed in the current zone:

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

NOTE

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

5.5. STARTING FIREWALLD

Procedure

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

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

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

   ```
   # systemctl enable firewalld
   ```

5.6. STOPPING FIREWALLD

Procedure

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

   ```
   # systemctl stop firewalld
   ```

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

   ```
   # systemctl disable firewalld
   ```

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

   ```
   # systemctl mask firewalld
   ```

5.7. RUNTIME AND PERMANENT SETTINGS

Any changes committed in *runtime* mode only apply while **firewalld** is running. When **firewalld** is restarted, the settings revert to their *permanent* values.

To make the changes persistent across reboots, apply them again using the **--permanent** option. Alternatively, to make changes persistent while **firewalld** is running, use the **--runtime-to-permanent firewall-cmd** option.

If you set the rules while **firewalld** is running using only the **--permanent** option, they do not become effective before **firewalld** is restarted. However, restarting **firewalld** closes all open ports and stops the networking traffic.

Modifying settings in runtime and permanent configuration using CLI

Using the CLI, you do not modify the firewall settings in both modes at the same time. You only modify either runtime or permanent mode. To modify the firewall settings in the permanent mode, use the **--permanent** option with the **firewall-cmd** command.

```
# firewall-cmd --permanent <other options>
```

Without this option, the command modifies runtime mode.

To change settings in both modes, you can use two methods:

1. Change runtime settings and then make them permanent as follows:

```
# firewall-cmd <other options>
# firewall-cmd --runtime-to-permanent
```

2. Set permanent settings and reload the settings into runtime mode:

```
# firewall-cmd --permanent <other options>
# firewall-cmd --reload
```

The first method allows you to test the settings before you apply them to the permanent mode.

> **NOTE**
>
> It is possible, especially on remote systems, that an incorrect setting results in a user locking themselves out of a machine. To prevent such situations, use the **--timeout** option. After a specified amount of time, any change reverts to its previous state. Using this options excludes the **--permanent** option.
>
> For example, to add the **SSH** service for 15 minutes:
>
> ```
> # firewall-cmd --add-service=ssh --timeout 15m
> ```

5.8. VERIFYING THE PERMANENT FIREWALLD CONFIGURATION

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

Prerequisites

- The **firewalld** service is running.

Procedure

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

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

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

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

5.9. CONTROLLING NETWORK TRAFFIC USING FIREWALLD

5.9.1. Disabling all traffic in case of emergency using CLI

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

Procedure

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

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

> **IMPORTANT**
>
> Enabling panic mode stops all networking traffic. From 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.

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

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

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

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

5.9.2. Controlling traffic with predefined services using CLI

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

Procedure

1. Check that the service is not already allowed:

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

2. List all predefined services:

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

3. Add the service to the allowed services:

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

4. Make the new settings persistent:

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

5.9.3. Controlling traffic with predefined services using GUI

To enable or disable a predefined or custom service:

1. Start the **firewall-config** tool and select the network zone whose services are to be configured.

2. Select the **Services** tab.

3. Select the check box for each type of service you want to trust or clear the check box to block a service.

To edit a service:

1. Start the **firewall-config** tool.

2. Select **Permanent** from the menu labeled **Configuration**. Additional icons and menu buttons appear at the bottom of the **Services** window.

3. 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**).
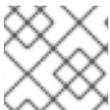
> **NOTE**
>
> It is not possible to alter service settings in **Runtime** mode.

5.9.4. Adding new services

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

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

Procedure

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

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

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

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

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

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

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

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

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

5.9.5. Controlling ports using CLI

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

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

5.9.5.1. Opening a port

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

Procedure

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

1. List all allowed ports:

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

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

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

3. Make the new settings persistent:

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

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

5.9.5.2. Closing a port

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

Procedure

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

1. List all allowed ports:

   ```
   # firewall-cmd --list-ports
   [WARNING]
   ====
   This command will only give you a list of ports that have been opened as ports. You will not
   be able to see any open ports that have been opened as a service. Therefore, you should
   consider using the --list-all option instead of --list-ports.
   ====
   ```

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

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

3. Make the new settings persistent:

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

5.9.6. Opening ports using GUI

To permit traffic through the firewall to a certain port:

1. Start the **firewall-config** tool and select the network zone whose settings you want to change.

2. Select the **Ports** tab and click the **Add** button on the right-hand side. The **Port and Protocol** window opens.

3. Enter the port number or range of ports to permit.

4. Select **tcp** or **udp** from the list.

5.9.7. Controlling traffic with protocols using GUI

To permit traffic through the firewall using a certain protocol:

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.

5.9.8. Opening source ports using GUI

To permit traffic through the firewall from a certain port:

1. Start the firewall-config tool and select the network zone whose settings you want to change.

2. Select the **Source Port** tab and click the **Add** button on the right-hand side. The **Source Port** window opens.

3. Enter the port number or range of ports to permit. Select **tcp** or **udp** from the list.

5.10. 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.

5.10.1. Listing zones

Procedure

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

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

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

2. To see detailed information for all zones:

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

3. To see detailed information for a specific zone:

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

5.10.2. Modifying firewalld settings for a certain zone

The Section 5.9.2, "Controlling traffic with predefined services using CLI" and Section 5.9.5, "Controlling ports using CLI" explain how to add services or modify ports in the scope of the current working zone. Sometimes, it is required to set up rules in a different zone.

Procedure

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

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

5.10.3. Changing the default zone

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

Procedure

To set up the default zone:

1. Display the current default zone:

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

2. Set the new default zone:

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

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

5.10.4. Assigning a network interface to a zone

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

Procedure

To assign the zone to a specific interface:

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

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

2. Assign the interface to a different zone:

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

5.10.5. Assigning a zone to a connection using nmcli

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

Procedure

1. Assign the zone to the NetworkManager connection profile:

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

2. Reload the connection:

   ```
   # nmcli connection up profile
   ```

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

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

Procedure

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

   ZONE=*zone_name*

5.10.7. Creating a new zone

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

Procedure

To create a new zone:

1. Create a new zone:

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

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

   # firewall-cmd --get-zones

3. Make the new settings persistent:

   # firewall-cmd --runtime-to-permanent

5.10.8. Zone configuration files

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

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

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

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

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

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

Additional resources

- For more information, see the **firewalld.zone** manual pages.

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

For every zone, you can set a default behavior that handles incoming traffic that is not further specified. Such behaviour is defined by setting the target of the zone. There are four options – **default**, **ACCEPT**, **REJECT**, and **DROP**. By setting the target to **ACCEPT**, you accept all incoming packets except those disabled by a specific rule. If you set the target to **REJECT** or **DROP**, you disable all incoming packets except those that you have allowed in specific rules. When packets are rejected, the source machine is informed about the rejection, while there is no information sent when the packets are dropped.

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>
   ```

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

5.11.1. Using zones to manage incoming traffic depending on a source

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

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

5.11.2. Adding a source

To route incoming traffic into a specific source, 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
    ```

5.11.3. Removing a source

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

Procedure

1. List allowed sources for the required zone:

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

2. Remove the source from the zone permanently:

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

3. Make the new settings persistent:

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

5.11.4. Adding a source port

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

Procedure

1. To add a source port:

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

5.11.5. Removing a source port

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

Procedure

1. To remove a source port:

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

5.11.6. 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 traffic from *192.168.1.0/24* to be able to reach the *HTTP* service while any other traffic is blocked.

Procedure

1. List all available zones:

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

2. Add the source to the trusted zone to route the traffic originating from the source through the zone:

   ```
   # firewall-cmd --zone=trusted --add-source=192.168.1.0/24
   ```

3. Add the *http* service in the trusted zone:

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

4. Make the new settings persistent:

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

5. Check that the trusted zone is active and that the service is allowed in it:

   ```
   # firewall-cmd --zone=trusted --list-all
   trusted (active)
   target: ACCEPT
   ```

> sources: 192.168.1.0/24
> services: http

5.11.7. Configuring traffic accepted by a zone based on a protocol

You can allow incoming traffic to be accepted by a zone based on a protocol. All traffic using the specified protocol is accepted by a zone, in which you can apply further rules and filtering.

5.11.7.1. Adding a protocol to a zone

By adding a protocol to a certain zone, you allow all traffic with this protocol to be accepted by this zone.

Procedure

1. To add a protocol to a zone:

   > # firewall-cmd --zone=zone-name --add-protocol=port-name/tcp|udp|sctp|dccp|igmp

> **NOTE**
>
> To receive multicast traffic, use the **igmp** value with the **--add-protocol** option.

5.11.7.2. Removing a protocol from a zone

By removing a protocol from a certain zone, you stop accepting all traffic based on this protocol by the zone.

Procedure

1. To remove a protocol from a zone:

   > # firewall-cmd --zone=zone-name --remove-protocol=port-name/tcp|udp|sctp|dccp|igmp

5.12. CONFIGURING IP ADDRESS MASQUERADING

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

Procedure

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

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

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

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

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

3. To make this setting persistent, repeat the command adding the **--permanent** option.

To disable IP masquerading, enter the following command as **root**:

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

5.13. PORT FORWARDING

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

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

5.13.1. Adding a port to redirect

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

Prerequisites

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

Procedure

To redirect a port to another port:

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

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

1. Add the port to be forwarded:

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

2. Enable masquerade:

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

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

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

Procedure

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

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

2. Make the new settings persistent:

> # firewall-cmd --runtime-to-permanent

3. Check that the port is redirected:

> # firewall-cmd --list-all

5.13.3. Removing a redirected port

To remove a redirected port:

> ~]# firewall-cmd --remove-forward-port=port=port-number:proto=<tcp|udp>:toport=port-number:toaddr=<IP/mask>

To remove a forwarded port redirected to a different address, use the following procedure.

Procedure

1. Remove the forwarded port:

> ~]# firewall-cmd --remove-forward-port=port=port-number:proto=<tcp|udp>:toport=port-number:toaddr=<IP/mask>

2. Disable masquerade:

> ~]# firewall-cmd --remove-masquerade

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

To remove the port redirection:

Procedure

1. List redirected ports:

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

2. Remove the redirected port from the firewall::

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

3. Make the new settings persistent:

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

5.14. MANAGING ICMP REQUESTS

The **Internet Control Message Protocol** (**ICMP**) is a supporting protocol that is used by various network devices to send error messages and operational information indicating a connection problem,

for example, that a requested service is not available. **ICMP** differs from transport protocols such as TCP and UDP because it is not used to exchange data between systems.

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

5.14.1. Listing and blocking ICMP requests

Listing **ICMP** requests

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

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

      # firewall-cmd --get-icmptypes

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

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

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

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

Blocking or unblocking **ICMP** requests

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

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

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

- To block an **ICMP** request:

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

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

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

Blocking **ICMP** requests without providing any information at all

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

- To block and drop all **ICMP** requests:

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

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

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

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

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

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

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

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

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

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

4. Make the new settings persistent:

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

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

- To revert the block inversion to a fully permissive setting:

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

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

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

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

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

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

4. Make the new settings persistent:

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

5.14.2. Configuring the ICMP filter using GUI

- To enable or disable an **ICMP** filter, start the **firewall-config** tool and select the network zone

whose messages are to be filtered. Select the **ICMP Filter** tab and select the check box for each type of **ICMP** message you want to filter. Clear the check box to disable a filter. This setting is per direction and the default allows everything.

- To edit an **ICMP** type, start the `firewall-config` tool and select **Permanent** mode from the menu labeled **Configuration**. Additional icons appear at the bottom of the **Services** window. Select **Yes** in the following dialog to enable masquerading and to make forwarding to another machine working.

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

5.15. SETTING AND CONTROLLING IP SETS USING FIREWALLD

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

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

5.15.1. Configuring IP set options using CLI

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.16. 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.

5.16.1. How the priority parameter organizes rules into different chains

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

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

- Priority lower than 0: the rule is redirected into a chain with the **_pre** suffix.

- Priority higher than 0: the rule is redirected into a chain with the **_post** suffix.

- Priority equals 0: based on the action, the rule is redirected into a chain with the **_log**, **_deny**, or **_allow** the action.

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

5.16.2. Setting the priority of a rich rule

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

Procedure

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

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

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

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

```
# nft list chain inet firewalld filter_IN_public_post
table inet firewalld {
  chain filter_IN_public_post {
    log prefix "UNEXPECTED: " limit rate 5/minute
  }
}
```

5.17. 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 whitelist 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.

5.17.1. Configuring lockdown with using CLI

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

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

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

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

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

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

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

5.17.2. Configuring lockdown whitelist options using CLI

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

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

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

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

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

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

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

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

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

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

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

- To query whether the command *command* is on the whitelist, enter the following command as **root**:

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

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

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

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

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

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

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

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

- To query whether the context *context* is on the whitelist, enter the following command as **root**:

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

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

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

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

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

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

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

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

- To query whether the user ID *uid* is on the whitelist, enter the following command:

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

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

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

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

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

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

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

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

- To query whether the user name *user* is on the whitelist, enter the following command:

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

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

5.17.3. Configuring lockdown whitelist options using configuration files

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

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

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

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

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

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

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

In Red Hat Enterprise Linux, all utilities are placed in the **/usr/bin/** directory and the **/bin/** directory is sym-linked to the **/usr/bin/** directory. In other words, although the path for **firewall-cmd** when entered as **root** might resolve to **/bin/firewall-cmd**, **/usr/bin/firewall-cmd** can now be used. All new scripts should use the new location. But be aware that if scripts that run as **root** are written to use the **/bin/firewall-cmd** path, then that command path must be whitelisted 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.

5.18. LOG FOR DENIED PACKETS

With the **LogDenied** option in the **firewalld**, it is possible to add a simple logging mechanism for denied packets. These are the packets that are rejected or dropped. To change the setting of the logging, edit the **/etc/firewalld/firewalld.conf** file or use the command-line or GUI configuration tool.

If **LogDenied** is enabled, logging rules are added right before the reject and drop rules in the INPUT, FORWARD and OUTPUT chains for the default rules and also the final reject and drop rules in zones. The possible values for this setting are: **all**, **unicast**, **broadcast**, **multicast**, and **off**. The default setting is **off**. With the **unicast**, **broadcast**, and **multicast** setting, the **pkttype** match is used to match the link-layer packet type. With **all**, all packets are logged.

To list the actual **LogDenied** setting with firewall-cmd, use the following command as **root**:

```
# firewall-cmd --get-log-denied
off
```

To change the **LogDenied** setting, use the following command as **root**:

```
# firewall-cmd --set-log-denied=all
success
```

To change the **LogDenied** setting with the **firewalld** GUI configuration tool, start **firewall-config**, click the **Options** menu and select **Change Log Denied**. The **LogDenied** window appears. Select the new **LogDenied** setting from the menu and click OK.

5.19. RELATED INFORMATION

The following sources of information provide additional resources regarding **firewalld**.

Installed documentation

- **firewalld(1)** man page — describes command options for **firewalld**.

- **firewalld.conf(5)** man page — contains information to configure **firewalld**.

- **firewall-cmd(1)** man page — describes command options for the **firewalld** command-line client.

- **firewall-config(1)** man page — describes settings for the **firewall-config** tool.

- **firewall-offline-cmd(1)** man page — describes command options for the **firewalld** offline command-line client.

- **firewalld.icmptype(5)** man page — describes XML configuration files for **ICMP** filtering.

- **firewalld.ipset(5)** man page — describes XML configuration files for the **firewalld IP** sets.

- **firewalld.service(5)** man page — describes XML configuration files for **firewalld service**.

- **firewalld.zone(5)** man page — describes XML configuration files for **firewalld** zone configuration.

- **firewalld.direct(5)** man page — describes the **firewalld** direct interface configuration file.

- **firewalld.lockdown-whitelist(5)** man page — describes the **firewalld** lockdown whitelist configuration file.

- **firewalld.richlanguage(5)** man page — describes the **firewalld** rich language rule syntax.

- **firewalld.zones(5)** man page — general description of what zones are and how to configure them.

- **firewalld.dbus(5)** man page — describes the **D-Bus** interface of **firewalld**.

Online documentation

- http://www.firewalld.org/ — **firewalld** home page.

CHAPTER 6. GETTING STARTED WITH NFTABLES

The **nftables** framework enables administrators to configure packet-filtering rules used by the Linux kernel firewall.

6.1. INTRODUCTION TO NFTABLES

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

- lookup tables instead of linear processing

- a single framework for both the **IPv4** and **IPv6** protocols

- rules all applied atomically instead of fetching, updating, and storing a complete rule set

- support for debugging and tracing in the rule set (**nftrace**) and monitoring trace events (in the **nft** tool)

- more consistent and compact syntax, no protocol-specific extensions

- a Netlink API for third-party applications

Similarly to **iptables**, **nftables** use tables for storing chains. The chains contain individual rules for performing actions. The **nft** tool replaces all tools from the previous packet-filtering frameworks. The **libnftnl** library can be used for low-level interaction with **nftables** Netlink API over the **libmnl** library.

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

Additional resources

- The **nft(8)** man page provides a comprehensive reference documentation for configuring and inspecting packet filtering with nftables using the **nft** command-line tool.

6.2. 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 to configure a firewall on workstations. The utility is easy to use and covers the typical use cases for this scenario.

- **nftables**: Use the **nftables** utility to set up complex firewalls, such as for a whole network.

- **iptables**: The **iptables** utility is deprecated in Red Hat Enterprise Linux 8. Use instead **nftables**.

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

6.3. CONVERTING IPTABLES RULES TO NFTABLES RULES

Red Hat Enterprise Linux 8 provides the **iptables-translate** and **ip6tables-translate** tools to convert existing **iptables** or **ip6tables** rules into the equivalent ones for **nftables**.

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

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

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

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

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

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

6.4. WRITING AND EXECUTING NFTABLES SCRIPTS

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

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

- add comments

- define variables

- include other rule set files

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

When you install the **nftables** package, Red Hat Enterprise Linux automatically creates ***.nft** scripts in the **/etc/nftables/** directory. These scripts contain commands that create tables and empty chains for different purposes. You can either extend these files or write your scripts.

6.4.1. The required script header in nftables script

Similar to other scripts, **nftables** scripts require a shebang sequence in the first line of the script that sets the interpreter directive.

An **nftables** script must always start with the following line:

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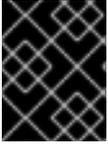
IMPORTANT

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

6.4.2. Supported nftables script formats

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

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

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

  # Flush the rule set
  flush ruleset

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

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

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

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

  # Flush the rule set
  flush ruleset

  # Create a table
  add table inet example_table

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

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

6.4.3. Running nftables scripts

To run an **nftables** script, the script must be executable. Only if the script is included in another script, it does not require to be executable. The procedure describes how to make a script executable and run the script.

Prerequisites

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

Procedure

1. Steps that are required only once:

   a. Optionally, set the owner of the script to **root**:

      > # chown root */etc/nftables/example_firewall.nft*

   b. Make the script executable for the owner:

      > # chmod u+x */etc/nftables/example_firewall.nft*

2. Run the script:

   > # */etc/nftables/example_firewall.nft*

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

   IMPORTANT

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

Additional resources

- For details about setting the owner of a file, see the **chown(1)** man page.

- For details about setting permissions of a file, see the **chmod(1)** man page.

- Section 6.4.7, "Automatically loading nftables rules when the system boots"

6.4.4. Using comments in nftables scripts

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

Example 6.1. Comments in an nftables script

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

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

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

6.4.5. Using variables in an nftables script

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

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

> define INET_DEV = *enp1s0*

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

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

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

> define DNS_SERVERS = { *192.0.2.1*, *192.0.2.2* }

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

> add rule inet example_table example_chain ip daddr **$DNS_SERVERS** accept

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

Additional resources

- For details about sets, see Section 6.11, "Using sets in nftables commands" .

- For details about verdict maps, see Section 6.12, "Using verdict maps in nftables commands" .

6.4.6. Including files in an nftables script

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

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

> **Example 6.2. Including files from the default search directory**
>
> To include a file from the default search directory:
>
> > include "example.nft"

> **Example 6.3. Including all *.nft files from a directory**
>
> To include all files ending in ***.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

- For further details, see the **Include files** section in the **nft(8)** man page.

6.4.7. Automatically loading nftables rules when the system boots

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

Prerequisites

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

Procedure

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

   - If you enhance ***.nft** scripts created in **/etc/nftables/** when you installed the **nftables** package, uncomment the **include** statement for these scripts.

   - If you write scripts from scratch, add **include** statements to include these scripts. For example, to load the **/etc/nftables/*example*.nft** script when the **nftables** service starts, add:

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

2. Enable the **nftables** service.

   ```
   # systemctl enable nftables
   ```

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

   ```
   # systemctl start nftables
   ```

Additional resources

- [Section 6.4.2, "Supported nftables script formats"](#)

6.5. DISPLAYING NFTABLES RULE SETS

Rule sets of **nftables** contain tables, chains, and rules. This section explains how to display these rule sets.

Procedure

1. To display all rule sets, enter:

   ```
   # nft list ruleset
   table inet example_table {
     chain example_chain {
       type filter hook input priority 0; policy accept;
   ```

```
    tcp dport http accept
    tcp dport ssh accept
  }
}
```

NOTE

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

6.6. CREATING AN NFTABLES TABLE

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

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

- **ip**: Matches only IPv4 packets. This is the default if you do not specify an address family.

- **ip6**: Matches only IPv6 packets.

- **inet**: Matches both IPv4 and IPv6 packets.

- **arp**: Matches IPv4 address resolution protocol (ARP) packets.

- **bridge**: Matches packets that traverse a bridge device.

- **netdev**: Matches packets from ingress.

Procedure

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

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

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

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

Additional resources

- For further details about address families, see the **Address families** section in the **nft(8)** man page.

- For details on other actions you can run on tables, see the **Tables** section in the **nft(8)** man page.

6.7. CREATING AN NFTABLES CHAIN

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

- Base chain: You can use base chains as an entry point for packets from the networking stack.

- Regular chain: You can use regular chains as a **jump** target and to better organize rules.

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

Prerequisites

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

Procedure

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

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

   IMPORTANT

   To avoid that the shell interprets the semicolons as the end of the command, you must escape the semicolons with a backslash.

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

2. Optionally, display all chains:

   ```
   # nft list chains
   table inet example_table {
     chain example_chain {
       type filter hook input priority 0; policy accept;
     }
   }
   ```

Additional resources

- For further details about address families, see the **Address families** section in the **nft(8)** man page.

- For details on other actions you can run on chains, see the **Chains** section in the **nft(8)** man page.

6.8. ADDING A RULE TO AN NFTABLES CHAIN

This section explains how to add a rule to an existing **nftables** chain. By default, the **nftables add rule** command appends a new rule to the end of the chain.

If you instead want to insert a rule at the beginning of chain, see Section 6.9, "Inserting a rule into an nftables chain".

Prerequisites

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

Procedure

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

   > # nft add rule *inet example_table example_chain tcp* dport *22 accept*

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

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

   ```
   # nft list table inet example_table
   table inet example_table {
     chain example_chain {
       type filter hook input priority 0; policy accept;
       ...
       tcp dport ssh accept
     }
   }
   ```

Additional resources

- For further details about address families, see the **Address families** section in the **nft(8)** man page.

- For details on other actions you can run on rules, see the **Rules** section in the **nft(8)** man page.

6.9. INSERTING A RULE INTO AN NFTABLES CHAIN

This section explains how to insert a rule at the beginning of an existing **nftables** chain using the **nftables insert rule** command. If you instead want to add a rule to the end of a chain, see Section 6.8, "Adding a rule to an nftables chain".

Prerequisites

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

Procedure

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

   > # nft insert rule *inet example_table example_chain tcp* dport *22 accept*

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

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

```
# nft list table inet example_table
table inet example_table {
  chain example_chain {
    type filter hook input priority 0; policy accept;
    tcp dport ssh accept
    ...
  }
}
```

Additional resources

- For further details about address families, see the **Address families** section in the **nft(8)** man page.

- For details on other actions you can run on rules, see the **Rules** section in the **nft(8)** man page.

6.10. CONFIGURING NAT USING NFTABLES

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

- Masquerading

- Source NAT (SNAT)

- Destination NAT (DNAT)

6.10.1. The different NAT types: masquerading, source NAT, and destination NAT

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

- Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.

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

Destination NAT (DNAT)

Use this NAT type to route incoming traffic to a different host. For example, if your web server uses an IP address from a reserved 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.

6.10.2. Configuring masquerading using nftables

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

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

Procedure

1. Create a table:

   ```
   # nft add table nat
   ```

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

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

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

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

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

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

6.10.3. Configuring source NAT using nftables

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

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

Procedure

1. Create a table:

   ```
   # nft add table nat
   ```

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

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

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

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

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

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

Additional resources

- Section 6.13.2, "Forwarding incoming packets on a specific local port to a different host"

6.10.4. Configuring destination NAT using nftables

Destination NAT enables you to redirect traffic on a router to a host that is not directly accessible from the Internet.

The following procedure describes how to redirect incoming traffic sent to port **80** and **443** of the router to the host with the **192.0.2.1** IP address.

Procedure

1. Create a table:

   ```
   # nft add table nat
   ```

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

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

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

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

3. Add a rule to the **prerouting** chain that redirects incoming traffic on the **ens3** interface sent to port **80** and **443** to the host with the **192.0.2.1** IP:

   ```
   # 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:

   a. If the **ens3** interface used 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
```

Additional resources

- Section 6.10.1, "The different NAT types: masquerading, source NAT, and destination NAT"

6.11. 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.

6.11.1. Using an anonymous sets in nftables

An anonymous set contain comma-separated values enclosed in curly brackets, such as **{ 22, 80, 443 }**, that you use directly in a rule. You can also use anonymous sets also for IP addresses or 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 Section 6.11.2, "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. Optionally, display all chains and their rules in **example_table**:

```
# nft list table inet example_table
table inet example_table {
  chain example_chain {
    type filter hook input priority 0; policy accept;
    tcp dport { ssh, http, https } accept
  }
}
```

6.11.2. 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 avoid that the shell interprets the semicolons as the end of the command, you must escape the semicolons with a backslash.

2. Optionally, 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.

6.11.3. Related information

- For further details about sets, see the **Sets** section in the **nft(8)** man page.

6.12. 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.

6.12.1. Using literal maps in nftables

A literal 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 a literal map is that if you want to change the map, you must replace the rule. For a dynamic solution, use named verdict maps as described in Section 6.12.2, "Using mutable verdict maps in nftables".

The example describes how to use a literal 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 the **example_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 a literal map to **incoming_traffic**:

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

The literal 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 0; 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.

6.12.2. Using mutable verdict maps in nftables

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

When you create a mutable verdict 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.

The example describes how to allow or drop incoming packets based on their source IP address. Using a mutable verdict map, you require only a single rule to configure this scenario while the IP addresses and actions are dynamically stored in the map. The procedure also describes how to add and remove entries from 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 avoid that the shell interprets 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. Optionally, 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. Optionally, remove an entry from the map:

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

8. Optionally, 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 0; policy accept;
    ip saddr vmap @example_map
  }
}
```

6.12.3. Related information

- For further details about verdict maps, see the **Maps** section in the **nft(8)** man page.

6.13. 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.

6.13.1. Forwarding incoming packets to a different local port

This section describes an example of how to 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 avoid that the shell interprets 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
   ```

6.13.2. 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.

The procedure describes how to 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.

Prerequisite

- 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 avoid that the shell interprets 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 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

6.14. LIMITING THE NUMBER OF CONNECTIONS USING NFTABLES

The **ct count** parameter of the **nft** utility enables administrators to limit the number of connections. The procedure describes a basic example of how to limit incoming connections.

Prerequisites

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

Procedure

1. 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

2. Optionally, display the meter created in the previous step:

   ```
   # nft list meter ip example_table example_meter
   table ip 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.

6.15. BLOCKING IP ADDRESSES THAT ATTEMPT MORE THAN TEN NEW INCOMING TCP CONNECTIONS WITHIN ONE MINUTE

The **nftables** framework enables administrators to dynamically update sets. This section explains how you use this feature to temporarily block hosts that are establishing more than ten IPv4 TCP connections within one minute. After five minutes, **nftables** automatically removes the IP address from the blacklist.

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 **blacklist** to the **filter** table:

   # nft add set ip *filter blacklist* { 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 5 minutes from the set.

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 **blacklist** set:

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

5. Add a rule that drops all connections from IP addresses in the **blacklist** set:

```
# nft add rule ip filter input ip saddr @blacklist drop
```

Additional resources

- Section 6.11.2, "Using named sets in nftables"

6.16. DEBUGGING NFTABLES RULES

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

6.16.1. Creating a rule with a counter

To identify if a rule is matched, you can use a counter. This section describes how to create a new rule with a counter.

For a procedure that adds a counter to an existing rule, see Section 6.16.2, "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 0; policy accept;
       tcp dport ssh counter packets 6872 bytes 105448565 accept
     }
   }
   ```

6.16.2. Adding a counter to an existing rule

To identify if a rule is matched, you can use a counter. This section describes how to add a counter to an existing rule.

For a procedure to add a new rule with a counter, see Section 6.16.1, "Creating a rule with a 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 0; 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 0; policy accept;
       tcp dport ssh counter packets 6872 bytes 105448565 accept
     }
   }
   ```

6.16.3. 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. The procedure describes how to enable tracing for a rule as well as 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 0; 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.

6.17. BACKING UP AND RESTORING NFTABLES RULE SETS

This section describes how to backup **nftables** rules to a file, as well as restoring rules from a file.

Administrators can use a file with the rules to, for example, transfer the rules to a different server.

6.17.1. Backing up nftables rule sets to a file

This section describes how to back up **nftables** rule sets to a file.

Procedure

1. To backup **nftables** rules:

   - In **nft list ruleset** format:

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

   - In JSON format:

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

6.17.2. Restoring nftables rule sets from a file

This section describes how to restore **nftables** rule sets.

Procedure

1. To restore **nftables** rules:

- If the file to restore is in **nft list ruleset** format or contains **nft** commands:

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

- If the file to restore is in JSON format:

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

6.18. RELATED INFORMATION

- The Using nftables in Red Hat Enterprise Linux 8 blog post provides an overview about using **nftables** features.

- The What comes after iptables? Its successor, of course: nftables article explains why **nftables** replaces **iptables**.

- The Firewalld: The Future is nftables article provides additional information on **nftables** as a default back end for **firewalld**.