Red Hat Enterprise Linux 8

Securing networks

Configuring secured networks and network communication

Last Updated: 2024-05-23

Red Hat Enterprise Linux 8 Securing networks

Configuring secured networks and network communication

Legal Notice

Abstract

Learn the tools and techniques to improve the security of your networks and lower the risks of data breaches and intrusions.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

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

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

Submitting feedback through Jira (account required)

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

CHAPTER 1. USING 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.

A host key authenticates hosts in the SSH protocol. Host keys are cryptographic keys that are generated automatically when OpenSSH is first installed, or when the host boots for the first time.

OpenSSH is an implementation of the SSH protocol supported by 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 RHEL supports only SSH version 2. It has an enhanced key-exchange algorithm that is not vulnerable to exploits known in version 1.

OpenSSH, as one of core cryptographic subsystems of RHEL, uses system-wide crypto policies. This

ensures that weak cipher suites and cryptographic algorithms are disabled in the default configuration. To modify the policy, the administrator must either use the **update-crypto-policies** command to adjust the settings or manually opt out of the system-wide crypto policies.

The OpenSSH suite uses two sets of configuration files: one for client programs (that is, **ssh**, **scp**, and **sftp**), and another 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 listed by using 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.
```

Ensure that the **Banner** option is not commented out in **/etc/ssh/sshd_config** and its value contains **/etc/issue**:

```
# less /etc/ssh/sshd_config | grep Banner
Banner /etc/issue
```

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 and restart **sshd** to apply the changes:

```
# systemctl daemon-reload
# systemctl restart sshd
```

Verification

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++lPegSy31v3L/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. SETTING AN OPENSSH SERVER FOR KEY-BASED AUTHENTICATION

To improve system security, enforce key-based authentication by disabling password authentication on your OpenSSH server.

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:

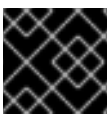
```
# systemctl reload sshd
```

Additional resources

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

1.4. 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+SqwBKRNaU72oZfaCI
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
joesec@ssh-server-example.com's password:
...
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

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.

1.5. USING SSH KEYS STORED ON A SMART CARD

Red Hat Enterprise Linux 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] $
```

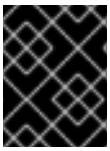
Additional resources

- Fedora 28: Better smart card support in OpenSSH
- **p11-kit(8)**, **opensc.conf(5)**, **pcscd(8)**, **ssh(1)**, and **ssh-keygen(1)** man pages

1.6. 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, use the **sshd-keygen@.service** instantiated service. For example, to disable the automatic creation of the RSA key type:

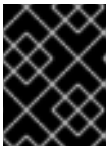
```
# systemctl mask sshd-keygen@rsa.service
```

NOTE

In images with **cloud-init** enabled, the **ssh-keygen** units are automatically disabled. This is because the **ssh-keygen template** service can interfere with the **cloud-init** tool and cause problems with host key generation. To prevent these problems the **etc/systemd/system/sshd-keygen@.service.d/disable-sshd-keygen-if-cloud-init-active.conf** drop-in configuration file disables the **ssh-keygen** units if **cloud-init** is running.

- 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
```



IMPORTANT

The Ed25519 algorithm is not FIPS-140-compliant, and OpenSSH does not work with Ed25519 keys in FIPS mode.

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 therefore 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 --remove-port=22/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 can set 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.

  The OpenSSH server permits only connections that pass all Allow and Deny directives in **/etc/ssh/sshd_config**. For example, if the **AllowUsers** directive lists a user that is not part of a group listed in the **AllowGroups** directive, then the user cannot log in.

  Note that using allowlists (directives starting with Allow) is more secure than using blocklists (options starting with Deny) because allowlists 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
```

WARNING

If your system communicates on the internet, you might face interoperability problems due to the strict setting of the **FUTURE** policy.

You can also disable only specific ciphers for the SSH protocol through the system-wide cryptographic policies. See the Customizing system-wide cryptographic policies with subpolicies section in the Security hardening document for more information.

To opt out of the system-wide cryptographic 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.

See the **sshd_config(5)** man page for more information.

To opt out of system-wide cryptographic policies for your OpenSSH client, perform one of the following tasks:

- For a given user, override the global **ssh_config** with a user-specific configuration in the **~/.ssh/config** file.
- For the entire system, specify the cryptographic policy in a drop-in configuration file located in the **/etc/ssh/ssh_config.d/** directory, with a two-digit number prefix smaller than 5, so that it lexicographically precedes the **05-redhat.conf** file, and with a **.conf** suffix, for example, **04-crypto-policy-override.conf**.

Additional resources

- **sshd_config(5)**, **ssh-keygen(1)**, **crypto-policies(7)**, and **update-crypto-policies(8)** man pages.
- Using system-wide cryptographic policies in the Security hardening document.
- How to disable specific algorithms and ciphers for ssh service only article.

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

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

Prerequisites

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

Procedure

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

```
Host jump-server1
  HostName jump1.example.com
```

- The **Host** parameter defines a name or alias for the host you can use in **ssh** commands. The value can match the real host name, but can also be any string.
- The **HostName** parameter sets the actual host name or IP address of the jump host.

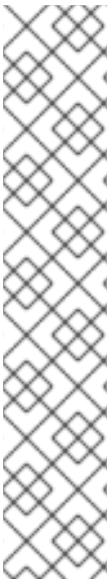
2. Add the remote server jump configuration with the **ProxyJump** directive to **~/.ssh/config** file on your local system, for example:

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

3. Use your local system to 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.

1.8. 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.

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

- 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.

1.9. ADDITIONAL RESOURCES

- **sshd(8)**, **ssh(1)**, **scp(1)**, **sftp(1)**, **ssh-keygen(1)**, **ssh-copy-id(1)**, **ssh_config(5)**, **sshd_config(5)**, **update-crypto-policies(8)**, and **crypto-policies(7)** man pages.
- OpenSSH Home Page
- Configuring SELinux for applications and services with non-standard configurations
- Controlling network traffic using firewalld

CHAPTER 2. CONFIGURING SECURE COMMUNICATION WITH THE SSH SYSTEM ROLES

As an administrator, you can use the **sshd** system role to configure SSH servers and the **ssh** system role to configure SSH clients consistently on any number of RHEL systems at the same time by using Red Hat Ansible Automation Platform.

2.1. VARIABLES OF THE SSHD RHEL SYSTEM ROLE

In an **sshd** system role playbook, you can define the parameters for the SSH configuration file according to your preferences and limitations.

If you do not configure these variables, the system role produces an **sshd_config** file that matches the RHEL defaults.

In all cases, Booleans correctly render as **yes** and **no** in **sshd** configuration. You can define multi-line configuration items using lists. For example:

```
sshd_ListenAddress:
  - 0.0.0.0
  - '::'
```

renders as:

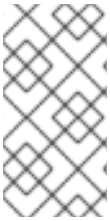
```
ListenAddress 0.0.0.0
ListenAddress ::
```

Additional resources

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

2.2. CONFIGURING OPENSSH SERVERS BY USING THE SSHD RHEL SYSTEM ROLE

You can use the **sshd** system role to configure multiple SSH servers by running an Ansible playbook.



NOTE

You can use the **sshd** system role with other system roles that change SSH and SSHD configuration, for example the Identity Management RHEL system roles. To prevent the configuration from being overwritten, make sure that the **sshd** role uses namespaces (RHEL 8 and earlier versions) or a drop-in directory (RHEL 9).

Prerequisites

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

Procedure

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

```
---
- name: SSH server configuration
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure sshd to prevent root and password login except from particular subnet
      ansible.builtin.include_role:
        name: rhel-system-roles.sshd
      vars:
        sshd:
          PermitRootLogin: no
          PasswordAuthentication: no
          Match:
            - Condition: "Address 192.0.2.0/24"
              PermitRootLogin: yes
              PasswordAuthentication: yes
```

   The playbook configures the managed node as an SSH server configured so that:

   - password and **root** user login is disabled
   - password and **root** user login is enabled only from the subnet **192.0.2.0/24**

2. Validate the playbook syntax:

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

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

3. Run the playbook:

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

Verification

1. Log in to the SSH server:

```
$ ssh <username>@<ssh_server>
```

2. Verify the contents of the **sshd_config** file on the SSH server:

```
$ cat /etc/ssh/sshd_config
...
PasswordAuthentication no
PermitRootLogin no
...
Match Address 192.0.2.0/24
  PasswordAuthentication yes
  PermitRootLogin yes
...
```

3. Check that you can connect to the server as root from the **192.0.2.0/24** subnet:

   a. Determine your IP address:

   ```
   $ hostname -I
   192.0.2.1
   ```

   If the IP address is within the **192.0.2.1** - **192.0.2.254** range, you can connect to the server.

   b. Connect to the server as **root**:

   ```
   $ ssh root@<ssh_server>
   ```

Additional resources

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

2.3. VARIABLES OF THE SSH RHEL SYSTEM ROLE

In an **ssh** system role playbook, you can define the parameters for the client SSH configuration file according to your preferences and limitations.

If you do not configure these variables, the system role produces a global **ssh_config** file that matches the RHEL defaults.

In all cases, booleans correctly render as **yes** or **no** in **ssh** configuration. You can define multi-line configuration items using lists. For example:

```
LocalForward:
  - 22 localhost:2222
  - 403 localhost:4003
```

renders as:

```
LocalForward 22 localhost:2222
LocalForward 403 localhost:4003
```

NOTE

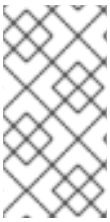
The configuration options are case sensitive.

Additional resources

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

2.4. CONFIGURING OPENSSH CLIENTS BY USING THE SSH RHEL SYSTEM ROLE

You can use the **ssh** system role to configure multiple SSH clients by running an Ansible playbook.



NOTE

You can use the **ssh** system role with other system roles that change SSH and SSHD configuration, for example the Identity Management RHEL system roles. To prevent the configuration from being overwritten, make sure that the **ssh** role uses a drop-in directory (default in RHEL 8 and later).

Prerequisites

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

Procedure

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

```
---
- name: SSH client configuration
  hosts: managed-node-01.example.com
  tasks:
    - name: "Configure ssh clients"
      ansible.builtin.include_role:
        name: rhel-system-roles.ssh
      vars:
        ssh_user: root
        ssh:
          Compression: true
          GSSAPIAuthentication: no
          ControlMaster: auto
          ControlPath: ~/.ssh/.cm%C
          Host:
            - Condition: example
              Hostname: server.example.com
              User: user1
        ssh_ForwardX11: no
```

   This playbook configures the **root** user's SSH client preferences on the managed nodes with the following configurations:

   - Compression is enabled.
   - ControlMaster multiplexing is set to **auto**.
   - The **example** alias for connecting to the **server.example.com** host is **user1**.
   - The **example** host alias is created, which represents a connection to the **server.example.com** host the with the **user1** user name.
   - X11 forwarding is disabled.

2. Validate the playbook syntax:

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

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

3. Run the playbook:

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

Verification

- Verify that the managed node has the correct configuration by displaying the SSH configuration file:

```
# cat ~/root/.ssh/config
# Ansible managed
Compression yes
ControlMaster auto
ControlPath ~/.ssh/.cm%C
ForwardX11 no
GSSAPIAuthentication no
Host example
  Hostname example.com
  User user1
```

Additional resources

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

2.5. USING THE SSHD RHEL SYSTEM ROLE FOR NON-EXCLUSIVE CONFIGURATION

Normally, applying the **sshd** system role overwrites the entire configuration. This may be problematic if you have previously adjusted the configuration, for example, with a different system role or playbook. To apply the **sshd** system role for only selected configuration options while keeping other options in place, you can use the non-exclusive configuration.

You can apply a non-exclusive configuration:

- In RHEL 8 and earlier by using a configuration snippet.
- In RHEL 9 and later by using files in a drop-in directory. The default configuration file is already placed in the drop-in directory as **/etc/ssh/sshd_config.d/00-ansible_system_role.conf**.

Prerequisites

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

Procedure

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

   - For managed nodes that run RHEL 8 or earlier:

     ```
     ---
     - name: Non-exclusive sshd configuration
       hosts: managed-node-01.example.com
       tasks:
         - name: <Configure SSHD to accept some useful environment variables>
           ansible.builtin.include_role:
             name: rhel-system-roles.sshd
           vars:
             sshd_config_namespace: <my-application>
             sshd:
               # Environment variables to accept
               AcceptEnv:
                 LANG
                 LS_COLORS
                 EDITOR
     ```

   - For managed nodes that run RHEL 9 or later:

     ```
     - name: Non-exclusive sshd configuration
       hosts: managed-node-01.example.com
       tasks:
         - name: <Configure sshd to accept some useful environment variables>
           ansible.builtin.include_role:
             name: rhel-system-roles.sshd
           vars:
             sshd_config_file: /etc/ssh/sshd_config.d/<42-my-application>.conf
             sshd:
               # Environment variables to accept
               AcceptEnv:
                 LANG
                 LS_COLORS
                 EDITOR
     ```

     In the **sshd_config_file** variable, define the **.conf** file into which the **sshd** system role writes the configuration options. Use a two-digit prefix, for example ***42-*** to specify the order in which the configuration files will be applied.

2. Validate the playbook syntax:

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

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

3. Run the playbook:

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

Verification

Verification

- Verify the configuration on the SSH server:
  - For managed nodes that run RHEL 8 or earlier:

```
# cat /etc/ssh/sshd_config.d/42-my-application.conf
# Ansible managed
#
AcceptEnv LANG LS_COLORS EDITOR
```

  - For managed nodes that run RHEL 9 or later:

```
# cat /etc/ssh/sshd_config
...
# BEGIN sshd system role managed block: namespace <my-application>
Match all
  AcceptEnv LANG LS_COLORS EDITOR
# END sshd system role managed block: namespace <my-application>
```

Additional resources

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

CHAPTER 3. CREATING AND MANAGING TLS KEYS AND CERTIFICATES

You can encrypt communication transmitted between two systems by using the TLS (Transport Layer Security) protocol. This standard uses asymmetric cryptography with private and public keys, digital signatures, and certificates.

3.1. TLS CERTIFICATES

TLS (Transport Layer Security) is a protocol that enables client-server applications to pass information securely. TLS uses a system of public and private key pairs to encrypt communication transmitted between clients and servers. TLS is the successor protocol to SSL (Secure Sockets Layer).

TLS uses X.509 certificates to bind identities, such as hostnames or organizations, to public keys using digital signatures. X.509 is a standard that defines the format of public key certificates.

Authentication of a secure application depends on the integrity of the public key value in the application's certificate. If an attacker replaces the public key with its own public key, it can impersonate the true application and gain access to secure data. To prevent this type of attack, all certificates must be signed by a certification authority (CA). A CA is a trusted node that confirms the integrity of the public key value in a certificate.

A CA signs a public key by adding its digital signature and issues a certificate. A digital signature is a message encoded with the CA's private key. The CA's public key is made available to applications by distributing the certificate of the CA. Applications verify that certificates are validly signed by decoding the CA's digital signature with the CA's public key.

To have a certificate signed by a CA, you must generate a public key, and send it to a CA for signing. This is referred to as a certificate signing request (CSR). A CSR contains also a distinguished name (DN) for the certificate. The DN information that you can provide for either type of certificate can include a two-letter country code for your country, a full name of your state or province, your city or town, a name of your organization, your email address, and it can also be empty. Many current commercial CAs prefer the Subject Alternative Name extension and ignore DNs in CSRs.

RHEL provides two main toolkits for working with TLS certificates: GnuTLS and OpenSSL. You can create, read, sign, and verify certificates using the **openssl** utility from the **openssl** package. The **certtool** utility provided by the **gnutls-utils** package can do the same operations using a different syntax and above all a different set of libraries in the back end.

Additional resources

- RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- **openssl(1)**, **x509(1)**, **ca(1)**, **req(1)**, and **certtool(1)** man pages

3.2. CREATING A PRIVATE CA USING OPENSSL

Private certificate authorities (CA) are useful when your scenario requires verifying entities within your internal network. For example, use a private CA when you create a VPN gateway with authentication based on certificates signed by a CA under your control or when you do not want to pay a commercial CA. To sign certificates in such use cases, the private CA uses a self-signed certificate.

Prerequisites

- You have **root** privileges or permissions to enter administrative commands with **sudo**. Commands that require such privileges are marked with **#**.

Procedure

1. Generate a private key for your CA. For example, the following command creates a 256-bit Elliptic Curve Digital Signature Algorithm (ECDSA) key:

```
$ openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:P-256 -out <ca.key>
```

   The time for the key-generation process depends on the hardware and entropy of the host, the selected algorithm, and the length of the key.

2. Create a certificate signed using the private key generated in the previous command:

```
$ openssl req -key <ca.key> -new -x509 -days 3650 -addext
keyUsage=critical,keyCertSign,cRLSign -subj "/CN=<Example CA>" -out <ca.crt>
```

   The generated **ca.crt** file is a self-signed CA certificate that you can use to sign other certificates for ten years. In the case of a private CA, you can replace *<Example CA>* with any string as the common name (CN).

3. Set secure permissions on the private key of your CA, for example:

```
# chown <root>:<root> <ca.key>
# chmod 600 <ca.key>
```

Next steps

- To use a self-signed CA certificate as a trust anchor on client systems, copy the CA certificate to the client and add it to the clients' system-wide truststore as **root**:

```
# trust anchor <ca.crt>
```

   See Chapter 4, *Using shared system certificates* for more information.

Verification

1. Create a certificate signing request (CSR), and use your CA to sign the request. The CA must successfully create a certificate based on the CSR, for example:

```
$ openssl x509 -req -in <client-cert.csr> -CA <ca.crt> -CAkey <ca.key> -CAcreateserial -
days 365 -extfile <openssl.cnf> -extensions <client-cert> -out <client-cert.crt>
Signature ok
subject=C = US, O = Example Organization, CN = server.example.com
Getting CA Private Key
```

   See Section 3.5, “Using a private CA to issue certificates for CSRs with OpenSSL” for more information.

2. Display the basic information about your self-signed CA:

```
$ openssl x509 -in <ca.crt> -text -noout
Certificate:
```

```
…
        X509v3 extensions:
            …
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
…
```

3. Verify the consistency of the private key:

```
$ openssl pkey -check -in <ca.key>
Key is valid
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgcagSaTEBn74xZAwO

18wRpXoCVC9vcPki7WIT+gnmCI+hRANCAARb9NxIvkaVjFhOoZbGp/HtIQxbM78E
lwbDP0BI624xBJ8gK68ogSaq2x4SdezFdV1gNeKScDcU+Pj2pELldmdF
-----END PRIVATE KEY-----
```

Additional resources

- **openssl(1)**, **ca(1)**, **genpkey(1)**, **x509(1)**, and **req(1)** man pages

3.3. CREATING A PRIVATE KEY AND A CSR FOR A TLS SERVER CERTIFICATE USING OPENSSL

You can use TLS-encrypted communication channels only if you have a valid TLS certificate from a certificate authority (CA). To obtain the certificate, you must create a private key and a certificate signing request (CSR) for your server first.

Procedure

1. Generate a private key on your server system, for example:

```
$ openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:P-256 -out <server-private.key>
```

2. Optional: Use a text editor of your choice to prepare a configuration file that simplifies creating your CSR, for example:

```
$ vim <example_server.cnf>
[server-cert]
keyUsage = critical, digitalSignature, keyEncipherment, keyAgreement
extendedKeyUsage = serverAuth
subjectAltName = @alt_name

[req]
distinguished_name = dn
prompt = no

[dn]
C = <US>
```

```
O = <Example Organization>
CN = <server.example.com>

[alt_name]
DNS.1 = <example.com>
DNS.2 = <server.example.com>
IP.1 = <192.168.0.1>
IP.2 = <::1>
IP.3 = <127.0.0.1>
```

The **extendedKeyUsage = serverAuth** option limits the use of a certificate.

3. Create a CSR using the private key you created previously:

```
$ openssl req -key <server-private.key> -config <example_server.cnf> -new -out <server-cert.csr>
```

If you omit the **-config** option, the **req** utility prompts you for additional information, for example:

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]: <US>
State or Province Name (full name) []: <Washington>
Locality Name (eg, city) [Default City]: <Seattle>
Organization Name (eg, company) [Default Company Ltd]: <Example Organization>
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []: <server.example.com>
Email Address []: <server@example.com>
```

Next steps

- Submit the CSR to a CA of your choice for signing. Alternatively, for an internal use scenario within a trusted network, use your private CA for signing. See Section 3.5, “Using a private CA to issue certificates for CSRs with OpenSSL” for more information.

Verification

1. After you obtain the requested certificate from the CA, check that the human-readable parts of the certificate match your requirements, for example:

```
$ openssl x509 -text -noout -in <server-cert.crt>
Certificate:
…
        Issuer: CN = Example CA
        Validity
            Not Before: Feb  2 20:27:29 2023 GMT
            Not After : Feb  2 20:27:29 2024 GMT
        Subject: C = US, O = Example Organization, CN = server.example.com
        Subject Public Key Info:
```

```
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
…
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment, Key Agreement
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
            X509v3 Subject Alternative Name:
                DNS:example.com, DNS:server.example.com, IP Address:192.168.0.1, IP
…
```

Additional resources

- **openssl(1)**, **x509(1)**, **genpkey(1)**, **req(1)**, and **config(5)** man pages

3.4. CREATING A PRIVATE KEY AND A CSR FOR A TLS CLIENT CERTIFICATE USING OPENSSL

You can use TLS-encrypted communication channels only if you have a valid TLS certificate from a certificate authority (CA). To obtain the certificate, you must create a private key and a certificate signing request (CSR) for your client first.

Procedure

1. Generate a private key on your client system, for example:

```
$ openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:P-256 -out <client-private.key>
```

2. Optional: Use a text editor of your choice to prepare a configuration file that simplifies creating your CSR, for example:

```
$ vim <example_client.cnf>
[client-cert]
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth
subjectAltName = @alt_name

[req]
distinguished_name = dn
prompt = no

[dn]
CN = <client.example.com>

[clnt_alt_name]
email= <client@example.com>
```

   The **extendedKeyUsage = clientAuth** option limits the use of a certificate.

3. Create a CSR using the private key you created previously:

```
$ openssl req -key <client-private.key> -config <example_client.cnf> -new -out <client-cert.csr>
```

If you omit the **-config** option, the **req** utility prompts you for additional information, for example:

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
…
Common Name (eg, your name or your server's hostname) []: <client.example.com>
Email Address []: <client@example.com>
```

Next steps

- Submit the CSR to a CA of your choice for signing. Alternatively, for an internal use scenario within a trusted network, use your private CA for signing. See Section 3.5, "Using a private CA to issue certificates for CSRs with OpenSSL" for more information.

Verification

1. Check that the human-readable parts of the certificate match your requirements, for example:

```
$ openssl x509 -text -noout -in <client-cert.crt>
Certificate:
…
            X509v3 Extended Key Usage:
                TLS Web Client Authentication
            X509v3 Subject Alternative Name:
                email:client@example.com
…
```

Additional resources

- **openssl(1)**, **x509(1)**, **genpkey(1)**, **req(1)**, and **config(5)** man pages

3.5. USING A PRIVATE CA TO ISSUE CERTIFICATES FOR CSRS WITH OPENSSL

To enable systems to establish a TLS-encrypted communication channel, a certificate authority (CA) must provide valid certificates to them. If you have a private CA, you can create the requested certificates by signing certificate signing requests (CSRs) from the systems.

Prerequisites

- You have already configured a private CA. See Section 3.2, "Creating a private CA using OpenSSL" for more information.
- You have a file containing a CSR. You can find an example of creating the CSR in Section 3.3, "Creating a private key and a CSR for a TLS server certificate using OpenSSL" .

Procedure

1. Optional: Use a text editor of your choice to prepare an OpenSSL configuration file for adding extensions to certificates, for example:

```
$ vim <openssl.cnf>
[server-cert]
extendedKeyUsage = serverAuth

[client-cert]
extendedKeyUsage = clientAuth
```

2. Use the **x509** utility to create a certificate based on a CSR, for example:

```
$ openssl x509 -req -in <server-cert.csr> -CA <ca.crt> -CAkey <ca.key> -CAcreateserial -
days 365 -extfile <openssl.cnf> -extensions <server-cert> -out <server-cert.crt>
Signature ok
subject=C = US, O = Example Organization, CN = server.example.com
Getting CA Private Key
```

To increase security, delete the serial-number file before you create another certificate from a CSR. This way, you ensure that the serial number is always random. If you omit the **CAserial** option for specifying a custom file name, the serial-number file name is the same as the file name of the certificate, but its extension is replaced with the **.srl** extension (**server-cert.srl** in the previous example).

Additional resources

- **openssl(1)**, **ca(1)**, and **x509(1)** man pages

3.6. CREATING A PRIVATE CA USING GNUTLS

Private certificate authorities (CA) are useful when your scenario requires verifying entities within your internal network. For example, use a private CA when you create a VPN gateway with authentication based on certificates signed by a CA under your control or when you do not want to pay a commercial CA. To sign certificates in such use cases, the private CA uses a self-signed certificate.

Prerequisites

- You have **root** privileges or permissions to enter administrative commands with **sudo**. Commands that require such privileges are marked with **#**.
- You have already installed GnuTLS on your system. If you did not, you can use this command:

```
$ yum install gnutls-utils
```

Procedure

1. Generate a private key for your CA. For example, the following command creates a 256-bit ECDSA (Elliptic Curve Digital Signature Algorithm) key:

```
$ certtool --generate-privkey --sec-param High --key-type=ecdsa --outfile <ca.key>
```

The time for the key-generation process depends on the hardware and entropy of the host, the selected algorithm, and the length of the key.

2. Create a template file for a certificate.

   a. Create a file with a text editor of your choice, for example:

   ```
   $ vi <ca.cfg>
   ```

   b. Edit the file to include the necessary certification details:

   ```
   organization = "Example Inc."
   state = "Example"
   country = EX
   cn = "Example CA"
   serial = 007
   expiration_days = 365
   ca
   cert_signing_key
   crl_signing_key
   ```

3. Create a certificate signed using the private key generated in step 1:
   The generated *<ca.crt>* file is a self-signed CA certificate that you can use to sign other certificates for one year. *<ca.crt>* file is the public key (certificate). The loaded file *<ca.key>* is the private key. You should keep this file in safe location.

   ```
   $ certtool --generate-self-signed --load-privkey <ca.key> --template <ca.cfg> --outfile
   <ca.crt>
   ```

4. Set secure permissions on the private key of your CA, for example:

   ```
   # chown <root>:<root> <ca.key>
   # chmod 600 <ca.key>
   ```

Next steps

- To use a self-signed CA certificate as a trust anchor on client systems, copy the CA certificate to the client and add it to the clients' system-wide truststore as **root**:

  ```
  # trust anchor <ca.crt>
  ```

  See Chapter 4, *Using shared system certificates* for more information.

Verification

1. Display the basic information about your self-signed CA:

   ```
   $ certtool --certificate-info --infile <ca.crt>
   Certificate:
   …
       X509v3 extensions:
           …
           X509v3 Basic Constraints: critical
               CA:TRUE
           X509v3 Key Usage: critical
               Certificate Sign, CRL Sign
   ```

2. Create a certificate signing request (CSR), and use your CA to sign the request. The CA must successfully create a certificate based on the CSR, for example:

   a. Generate a private key for your CA:

   ```
   $ certtool --generate-privkey --outfile <example-server.key>
   ```

   b. Open a new configuration file in a text editor of your choice, for example:

   ```
   $ vi <example-server.cfg>
   ```

   c. Edit the file to include the necessary certification details:

   ```
   signing_key
   encryption_key
   key_agreement

   tls_www_server

   country = "US"
   organization = "Example Organization"
   cn = "server.example.com"

   dns_name = "example.com"
   dns_name = "server.example.com"
   ip_address = "192.168.0.1"
   ip_address = "::1"
   ip_address = "127.0.0.1"
   ```

   d. Generate a request with the previously created private key:

   ```
   $ certtool --generate-request --load-privkey <example-server.key> --template <example-server.cfg> --outfile <example-server.crq>
   ```

   e. Generate the certificate and sign it with the private key of the CA:

   ```
   $ certtool --generate-certificate --load-request <example-server.crq> --load-ca-certificate <ca.crt> --load-ca-privkey <ca.key> --outfile <example-server.crt>
   ```

   Additional resources

   - **certtool(1)** and **trust(1)** man pages

3.7. CREATING A PRIVATE KEY AND A CSR FOR A TLS SERVER CERTIFICATE USING GNUTLS

To obtain the certificate, you must create a private key and a certificate signing request (CSR) for your server first.

Procedure

1. Generate a private key on your server system, for example:

```
$ certtool --generate-privkey --sec-param High --outfile <example-server.key>
```

2. Optional: Use a text editor of your choice to prepare a configuration file that simplifies creating your CSR, for example:

```
$ vim <example_server.cnf>
signing_key
encryption_key
key_agreement

tls_www_server

country = "US"
organization = "Example Organization"
cn = "server.example.com"

dns_name = "example.com"
dns_name = "server.example.com"
ip_address = "192.168.0.1"
ip_address = "::1"
ip_address = "127.0.0.1"
```

3. Create a CSR using the private key you created previously:

```
$ certtool --generate-request --template <example-server.cfg> --load-privkey <example-server.key> --outfile <example-server.crq>
```

If you omit the **--template** option, the **certool** utility prompts you for additional information, for example:

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Generating a PKCS #10 certificate request...
Country name (2 chars): <US>
State or province name: <Washington>
Locality name: <Seattle>
Organization name: <Example Organization>
Organizational unit name:
Common name: <server.example.com>
```

Next steps

- Submit the CSR to a CA of your choice for signing. Alternatively, for an internal use scenario within a trusted network, use your private CA for signing. See Section 3.9, “Using a private CA to issue certificates for CSRs with GnuTLS” for more information.

Verification

1. After you obtain the requested certificate from the CA, check that the human-readable parts of the certificate match your requirements, for example:

```
$ certtool --certificate-info --infile <example-server.crt>
Certificate:
…
        Issuer: CN = Example CA
        Validity
            Not Before: Feb  2 20:27:29 2023 GMT
            Not After : Feb  2 20:27:29 2024 GMT
        Subject: C = US, O = Example Organization, CN = server.example.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
…
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment, Key Agreement
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
            X509v3 Subject Alternative Name:
                DNS:example.com, DNS:server.example.com, IP Address:192.168.0.1, IP
…
```

Additional resources

- **certtool(1)** man page

3.8. CREATING A PRIVATE KEY AND A CSR FOR A TLS CLIENT CERTIFICATE USING GNUTLS

To obtain the certificate, you must create a private key and a certificate signing request (CSR) for your client first.

Procedure

1. Generate a private key on your client system, for example:

```
$ certtool --generate-privkey --sec-param High --outfile <example-client.key>
```

2. Optional: Use a text editor of your choice to prepare a configuration file that simplifies creating your CSR, for example:

```
$ vim <example_client.cnf>
signing_key
encryption_key

tls_www_client

cn = "client.example.com"
email = "client@example.com"
```

3. Create a CSR using the private key you created previously:

```
$ certtool --generate-request --template <example-client.cfg> --load-privkey <example-client.key> --outfile <example-client.crq>
```

If you omit the **--template** option, the **certtool** utility prompts you for additional information, for example:

```
Generating a PKCS #10 certificate request...
Country name (2 chars): <US>
State or province name: <Washington>
Locality name: <Seattle>
Organization name: <Example Organization>
Organizational unit name:
Common name: <server.example.com>
```

Next steps

- Submit the CSR to a CA of your choice for signing. Alternatively, for an internal use scenario within a trusted network, use your private CA for signing. See Section 3.9, "Using a private CA to issue certificates for CSRs with GnuTLS" for more information.

Verification

1. Check that the human-readable parts of the certificate match your requirements, for example:

```
$ certtool --certificate-info --infile <example-client.crt>
Certificate:
…
        X509v3 Extended Key Usage:
            TLS Web Client Authentication
        X509v3 Subject Alternative Name:
            email:client@example.com
…
```

Additional resources

- **certtool(1)** man page

3.9. USING A PRIVATE CA TO ISSUE CERTIFICATES FOR CSRS WITH GNUTLS

To enable systems to establish a TLS-encrypted communication channel, a certificate authority (CA) must provide valid certificates to them. If you have a private CA, you can create the requested certificates by signing certificate signing requests (CSRs) from the systems.

Prerequisites

- You have already configured a private CA. See Section 3.6, "Creating a private CA using GnuTLS" for more information.
- You have a file containing a CSR. You can find an example of creating the CSR in Section 3.7, "Creating a private key and a CSR for a TLS server certificate using GnuTLS" .

Procedure

1. Optional: Use a text editor of your choice to prepare an GnuTLS configuration file for adding extensions to certificates, for example:

```
$ vi <server-extensions.cfg>
honor_crq_extensions
ocsp_uri = "http://ocsp.example.com"
```

2. Use the **certtool** utility to create a certificate based on a CSR, for example:

```
$ certtool --generate-certificate --load-request <example-server.crq> --load-ca-privkey
<ca.key> --load-ca-certificate <ca.crt> --template <server-extensions.cfg> --outfile
<example-server.crt>
```

Additional resources

- **certtool(1)** man page

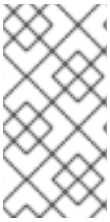
CHAPTER 4. USING SHARED SYSTEM CERTIFICATES

The shared system certificates storage enables NSS, GnuTLS, OpenSSL, and Java to share a default source for retrieving system certificate anchors and block-list information. By default, the truststore contains the Mozilla CA list, including positive and negative trust. The system allows updating the core Mozilla CA list or choosing another certificate list.

4.1. THE SYSTEM-WIDE TRUSTSTORE

In RHEL, the consolidated system-wide truststore is located in the **/etc/pki/ca-trust/** and **/usr/share/pki/ca-trust-source/** directories. The trust settings in **/usr/share/pki/ca-trust-source/** are processed with lower priority than settings in **/etc/pki/ca-trust/**.

Certificate files are treated depending on the subdirectory they are installed to. For example, trust anchors belong to the **/usr/share/pki/ca-trust-source/anchors/** or **/etc/pki/ca-trust/source/anchors/** directory.



NOTE

In a hierarchical cryptographic system, a trust anchor is an authoritative entity that other parties consider trustworthy. In the X.509 architecture, a root certificate is a trust anchor from which a chain of trust is derived. To enable chain validation, the trusting party must have access to the trust anchor first.

Additional resources

- **update-ca-trust(8)** and **trust(1)** man pages

4.2. ADDING NEW CERTIFICATES

To acknowledge applications on your system with a new source of trust, add the corresponding certificate to the system-wide store, and use the **update-ca-trust** command.

Prerequisites

- The **ca-certificates** package is present on the system.

Procedure

1. To add a certificate in the simple PEM or DER file formats to the list of CAs trusted on the system, copy the certificate file to the **/usr/share/pki/ca-trust-source/anchors/** or **/etc/pki/ca-trust/source/anchors/** directory, for example:

```
# cp ~/certificate-trust-examples/Cert-trust-test-ca.pem /usr/share/pki/ca-trust-source/anchors/
```

2. To update the system-wide truststore configuration, use the **update-ca-trust** command:

```
# update-ca-trust
```

NOTE

Even though the Firefox browser can use an added certificate without a prior execution of **update-ca-trust**, enter the **update-ca-trust** command after every CA change. Also note that browsers, such as Firefox, Chromium, and GNOME Web cache files, and you might have to clear your browser's cache or restart your browser to load the current system certificate configuration.

Additional resources

- **update-ca-trust(8)** and **trust(1)** man pages

4.3. MANAGING TRUSTED SYSTEM CERTIFICATES

The **trust** command provides a convenient way for managing certificates in the shared system-wide truststore.

- To list, extract, add, remove, or change trust anchors, use the **trust** command. To see the built-in help for this command, enter it without any arguments or with the **--help** directive:

```
$ trust
usage: trust command <args>...

Common trust commands are:
  list            List trust or certificates
  extract         Extract certificates and trust
  extract-compat  Extract trust compatibility bundles
  anchor          Add, remove, change trust anchors
  dump            Dump trust objects in internal format

See 'trust <command> --help' for more information
```

- To list all system trust anchors and certificates, use the **trust list** command:

```
$ trust list
pkcs11:id=%d2%87%b4%e3%df%37%27%93%55%f6%56%ea%81%e5%36%cc%8c%1e%3
f%bd;type=cert
    type: certificate
    label: ACCVRAIZ1
    trust: anchor
    category: authority

pkcs11:id=%a6%b3%e1%2b%2b%49%b6%d7%73%a1%aa%94%f5%01%e7%73%65%4c%
ac%50;type=cert
    type: certificate
    label: ACEDICOM Root
    trust: anchor
    category: authority
...
```

- To store a trust anchor into the system-wide truststore, use the **trust anchor** sub-command and specify a path to a certificate. Replace *<path.to/certificate.crt>* by a path to your certificate and its file name:

```
# trust anchor <path.to/certificate.crt>
```

- To remove a certificate, use either a path to a certificate or an ID of a certificate:

```
# trust anchor --remove <path.to/certificate.crt>
# trust anchor --remove "pkcs11:id=<%AA%BB%CC%DD%EE>;type=cert"
```

Additional resources

- All sub-commands of the **trust** commands offer a detailed built-in help, for example:

```
$ trust list --help
usage: trust list --filter=<what>

  --filter=<what>     filter of what to export
                      ca-anchors        certificate anchors
...
  --purpose=<usage>   limit to certificates usable for the purpose
                      server-auth       for authenticating servers
...
```

Additional resources

- **update-ca-trust(8)** and **trust(1)** man pages

CHAPTER 5. 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.

5.1. 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. In RHEL 8, 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. In RHEL 8, 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 .

5.2. 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.

WARNING

Algorithms disabled for the **LEGACY** cryptographic policy do not conform to Red Hat's vision of RHEL 8 security, and their security properties are not reliable. Consider moving away from using these algorithms instead of re-enabling them. If you do decide to re-enable them, for example for interoperability with old hardware, treat them as insecure and apply extra protection measures, such as isolating their network interactions to separate network segments. Do not use them across public networks.

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

5.2.1. 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 even though 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 the Apache web server.

5.2.2. 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 prefer 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 prefer AEAD ciphers, such as AES-GCM, over 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 a 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).

5.2.3. 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.

5.3. HARDENING TLS CONFIGURATION IN APPLICATIONS

In RHEL, system-wide crypto policies provide a convenient way to ensure that your applications that use 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 ensure that your server application enforces *server-side cipher order* , so that the cipher suite to be used is determined by the order you configure.

5.3.1. Configuring the Apache HTTP server to use TLS

The **Apache HTTP Server** can use both **OpenSSL** and **NSS** libraries for its TLS needs. RHEL 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 the **/usr/share/httpd/manual/mod/mod_ssl.html** file. Examples of various settings are described in the **/usr/share/httpd/manual/ssl/ssl_howto.html** file.

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
```

See the Configuring TLS encryption on an Apache HTTP Server chapter in the Deploying different types of servers document for more information.

5.3.2. Configuring the Nginx HTTP and proxy server to use TLS

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
    ....
}
```

See the Adding TLS encryption to an Nginx web server chapter in the Deploying different types of servers document for more information.

5.3.3. Configuring the Dovecot mail server to use TLS

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

- Deploying different types of servers on RHEL 8
- **config(5)** and **ciphers(1)** man pages.
- Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS).
- Mozilla SSL Configuration Generator.
- SSL Server Test.

CHAPTER 6. CONFIGURING A VPN WITH IPSEC

In RHEL 8, a virtual private network (VPN) can be configured using the **IPsec** protocol, which is supported by the [application]Libreswan application.

6.1. LIBRESWAN AS AN IPSEC VPN IMPLEMENTATION

In RHEL, 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 Layer 2 Tunneling Protocol (L2TP) is usually called an L2TP/IPsec VPN, which requires the **xl2tpd** package provided by the **optional** repository.

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 communicates with the Linux kernel using the Netlink interface. The kernel performs packet encryption and decryption.

Libreswan uses the Network Security Services (NSS) cryptographic library. NSS is 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 RHEL. Do not use any other VPN technology without understanding the risks of doing so.

In RHEL, 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.

The **leftid** and **rightid** options serve as identification of the respective hosts in the authentication process. See the **ipsec.conf(5)** man page for more information.

6.2. AUTHENTICATION METHODS IN LIBRESWAN

Libreswan supports several authentication methods, each of which fits a different scenario.

Pre-Shared key (PSK)

Pre-Shared Key (PSK) is the simplest authentication method. For security reasons, do not use PSKs shorter than 64 random characters. In FIPS mode, PSKs must comply with a minimum-strength requirement depending on the integrity algorithm used. You can set PSK by using the **authby=secret** connection.

Raw RSA keys

Raw RSA keys are commonly used for static host-to-host or subnet-to-subnet IPsec configurations. Each host is manually configured with the public RSA keys of all other hosts, and Libreswan sets up an IPsec tunnel between each pair of hosts. This method does not scale well for large numbers of hosts.

You can generate a raw RSA key on a host using the **ipsec newhostkey** command. You can list generated keys by using the **ipsec showhostkey** command. The **leftrsasigkey=** line is required for connection configurations that use CKA ID keys. Use the **authby=rsasig** connection option for raw RSA keys.

X.509 certificates

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

For example, you can generate X.509 certificates using the **openssl** command and the NSS **certutil** command. Because Libreswan reads user certificates from the NSS database using the certificates' nickname in the **leftcert=** configuration option, provide a nickname when you create a certificate.

If you use a custom CA certificate, you must import it to the Network Security Services (NSS) database. You can import any certificate in the PKCS #12 format to the Libreswan NSS database by using the **ipsec import** command.

WARNING

Libreswan requires an Internet Key Exchange (IKE) peer ID as a subject alternative name (SAN) for every peer certificate as described in section 3.1 of RFC 4945 . Disabling this check by changing the **require-id-on-certificated=** option can make the system vulnerable to man-in-the-middle attacks.

Use the **authby=rsasig** connection option for authentication based on X.509 certificates using RSA with SHA-1 and SHA-2. You can further limit it for ECDSA digital signatures using SHA-2 by setting

authby= to **ecdsa** and RSA Probabilistic Signature Scheme (RSASSA-PSS) digital signatures based authentication with SHA-2 through **authby=rsa-sha2**. The default value is **authby=rsasig,ecdsa**.

The certificates and the **authby=** signature methods should match. This increases interoperability and preserves authentication in one digital signature system.

NULL authentication

NULL authentication is used to gain mesh encryption without authentication. It protects against passive attacks but not against active attacks. However, because IKEv2 allows asymmetric authentication methods, NULL authentication can also be used for internet-scale opportunistic IPsec. In this model, clients authenticate the server, but servers do not authenticate the client. This model is similar to secure websites using TLS. Use **authby=null** for NULL authentication.

Protection against quantum computers

In addition to the previously mentioned authentication methods, you can use the *Post-quantum Pre-shared Key* (PPK) method to protect against possible attacks by quantum computers. Individual clients or groups of clients can use their own PPK by specifying a PPK ID that corresponds to an out-of-band configured pre-shared key.

Using IKEv1 with pre-shared keys protects against quantum attackers. The redesign of IKEv2 does not offer this protection natively. Libreswan offers the use of a *Post-quantum Pre-shared Key* (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 based on dictionary words. The PPK ID and PPK data are stored in the **ipsec.secrets** file, for example:

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

The **PPKS** option refers to static PPKs. This experimental function uses one-time-pad-based Dynamic PPKs. Upon each connection, a new part of the one-time pad is used as the PPK. When used, that part of the dynamic PPK inside the file is overwritten with zeros 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 an unsupported Technology Preview. Use with caution.

6.3. INSTALLING LIBRESWAN

Before you can set a VPN through the Libreswan IPsec/IKE implementation, you must install the corresponding packages, start the **ipsec** service, and allow the service in your firewall.

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 and create a new database:

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

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
```

6.4. CREATING A HOST-TO-HOST VPN

You can configure Libreswan to create a host-to-host IPsec VPN between two hosts referred to as *left* and *right* using authentication by raw RSA keys.

Prerequisites

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

Procedure

1. Generate a raw RSA key pair on each host:

```
# ipsec newhostkey
```

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+hOafUZDlCQmXFrje/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. Load the connection:

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

6. Establish the tunnel:

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

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

```
auto=start
```

6.5. 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+hOafUZDlCQmXFrje/oZm [...] W2n417C/4urYHQkCvuIQ==
   rightid=@east
   right=192.1.2.45
   rightrsasigkey=0sAQO3fwC6nSSGgt64DWiYZzuHbc4 [...] D/v8t5YTQ==
   authby=rsasig
```

6.6. CONFIGURING A REMOTE ACCESS VPN

Road warriors are traveling users with mobile clients and a dynamically assigned IP address. The mobile clients authenticate using X.509 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 or deny client
   # pam-authorize=yes
   # load connection, do not 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
```

6.7. 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).

NOTE

You can use any regular IKEv2 authentication method for *opportunistic IPsec*, because these connections are regular Libreswan configurations, except for the opportunistic IPsec that is defined by **right=%opportunisticgroup** entry. A common authentication method is for hosts to authenticate each other based on X.509 certificates using a commonly shared certification authority (CA). Cloud deployments typically issue certificates for each node in the cloud as part of the standard procedure.

Do not use PreSharedKey (PSK) authentication because one compromised host would result in group PSK secret being compromised as well.

You can use NULL authentication to deploy encryption between nodes without authentication, which protects only against passive attackers.

The following procedure uses X.509 certificates. You can generate these certificates by using any kind of 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.
- A new NSS database is initialized.

1. If you already have an old NSS database, remove the old database files:

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

2. You can initialize a new database with the following command:

```
# ipsec initnss
```

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 (1)
 type=passthrough
```

```
 authby=never
 left=%defaultroute
 right=%group

conn private
 auto=ondemand
 type=transport
 authby=rsasig
 failureshunt=drop
 negotiationshunt=drop
 ikev2=insist
 left=%defaultroute
 leftcert=nodeXXXX
 leftid=%fromcert (2)
 rightid=%fromcert
 right=%opportunisticgroup

conn private-or-clear
 auto=ondemand
 type=transport
 authby=rsasig
 failureshunt=passthrough
 negotiationshunt=passthrough
 # left
 left=%defaultroute
 leftcert=nodeXXXX (3)
 leftid=%fromcert
 leftrsasigkey=%cert
 # right
 rightrsasigkey=%cert
 rightid=%fromcert
 right=%opportunisticgroup
```

1 The **auto** variable has several options:

You can use the **ondemand** connection option with opportunistic IPsec to initiate the IPsec connection, or for explicitly configured connections that do not need to be active all the time. This option sets up a trap XFRM policy in the kernel, enabling the IPsec connection to begin when it receives the first packet that matches that policy.

You can effectively configure and manage your IPsec connections, whether you use Opportunistic IPsec or explicitly configured connections, by using the following options:

The add option
: Loads the connection configuration and prepares it for responding to remote initiations. However, the connection is not automatically initiated from the local side. You can manually start the IPsec connection by using the command **ipsec auto --up**.

The start option
: Loads the connection configuration and prepares it for responding to remote initiations. Additionally, it immediately initiates a connection to the remote peer. You can use this option for permanent and always active connections.

2 **leftid** and **rightid** variables identifies the right and the left channel of the IPsec tunnel connection. You can use these variables to obtain the value of the local IP address or the subject DN of the local certificate, if you have configured one.

3 **leftcert** variable defines the nickname of the NSS database, which you want to use.

3. Add the IP address of the network to the corresponding category. For example, if all nodes reside in the 10.15.0.0/16 network, and all nodes must use 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:

```
# 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
```

You have the option to create the files in the **/etc/ipsec.d/policies** directory from a template for each new node, or you can provision them by 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
```

Verification

You can verify the procedure by opening a IPsec tunnel between two nodes.

1. Open an IPsec tunnel by using the **ping** command:

```
# ping <nodeYYY>
```

2. Display the NSS database with the imported certification:

```
# certutil -L -d sql:/etc/ipsec.d

Certificate Nickname    Trust Attributes
                        SSL,S/MIME,JAR/XPI

west                     u,u,u
ca                      CT,,
```

3. See which tunnels a node has opened:

```
# ipsec trafficstatus
006 #2: "private#10.15.0.0/16"[1] ...nodeYYY, type=ESP, add_time=1691399301,
inBytes=512, outBytes=512, maxBytes=2^63B, id='C=US, ST=NC, O=Example
Organization, CN=east'
```

Additional resources

- **ipsec.conf(5)** man page.
- For more information about the **authby** variable, see 6.2. Authentication methods in Libreswan .

6.8. 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:

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

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

```
# reboot
```

Verification

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 .

6.9. 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.

Prerequisites

- 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

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[6214]: Initializing NSS using read-write database "sql:/etc/ipsec.d"
pluto[6214]: NSS Password from file "/etc/ipsec.d/nsspassword" for token "NSS Certificate
DB" with length 20 passed to NSS
pluto[6214]: NSS crypto library initialized
...
```

Additional resources

- **certutil(1)** man page.
- Government Standards Knowledgebase article.

6.10. CONFIGURING AN IPSEC VPN TO USE TCP

Libreswan supports TCP encapsulation of IKE and IPsec packets as described in RFC 8229. With this feature, you can establish IPsec VPNs on networks that prevent traffic transmitted via UDP and Encapsulating Security Payload (ESP). You can configure VPN servers and clients to use TCP either as a fallback or as the main VPN transport protocol. Because TCP encapsulation has bigger performance costs, use TCP as the main VPN protocol only if UDP is permanently blocked in your scenario.

Prerequisites

- A remote-access VPN is already configured.

Procedure

1. Add the following option to the **/etc/ipsec.conf** file in the **config setup** section:

```
listen-tcp=yes
```

2. To use TCP encapsulation as a fallback option when the first attempt over UDP fails, add the following two options to the client's connection definition:

```
enable-tcp=fallback
tcp-remoteport=4500
```

Alternatively, if you know that UDP is permanently blocked, use the following options in the client's connection configuration:

```
enable-tcp=yes
tcp-remoteport=4500
```

Additional resources

- IETF RFC 8229: TCP Encapsulation of IKE and IPsec Packets .

6.11. CONFIGURING AUTOMATIC DETECTION AND USAGE OF ESP HARDWARE OFFLOAD TO ACCELERATE AN IPSEC CONNECTION

Offloading Encapsulating Security Payload (ESP) to the hardware accelerates IPsec connections over Ethernet. By default, Libreswan detects if hardware supports this feature and, as a result, enables ESP hardware offload. In case that the feature was disabled or explicitly enabled, you can switch back to automatic detection.

Prerequisites

- The network card supports ESP hardware offload.
- The network driver supports ESP hardware offload.
- The IPsec connection is configured and works.

Procedure

1. Edit the Libreswan configuration file in the **/etc/ipsec.d/** directory of the connection that should use automatic detection of ESP hardware offload support.
2. Ensure the **nic-offload** parameter is not set in the connection's settings.
3. If you removed **nic-offload**, restart the **ipsec** service:

```
# systemctl restart ipsec
```

Verification

If the network card supports ESP hardware offload support, following these steps to verify the result:

1. Display the **tx_ipsec** and **rx_ipsec** counters of the Ethernet device the IPsec connection uses:

```
# ethtool -S enp1s0 | egrep "_ipsec"
    tx_ipsec: 10
    rx_ipsec: 10
```

2. Send traffic through the IPsec tunnel. For example, ping a remote IP address:

```
# ping -c 5 remote_ip_address
```

3. Display the **tx_ipsec** and **rx_ipsec** counters of the Ethernet device again:

```
# ethtool -S enp1s0 | egrep "_ipsec"
    tx_ipsec: 15
    rx_ipsec: 15
```

If the counter values have increased, ESP hardware offload works.

Additional resources

- Configuring a VPN with IPsec

6.12. CONFIGURING ESP HARDWARE OFFLOAD ON A BOND TO ACCELERATE AN IPSEC CONNECTION

Offloading Encapsulating Security Payload (ESP) to the hardware accelerates IPsec connections. If you use a network bond for fail-over reasons, the requirements and the procedure to configure ESP hardware offload are different from those using a regular Ethernet device. For example, in this scenario, you enable the offload support on the bond, and the kernel applies the settings to the ports of the bond.

Prerequisites

- All network cards in the bond support ESP hardware offload.
- The network driver supports ESP hardware offload on a bond device. In RHEL, only the **ixgbe** driver supports this feature.
- The bond is configured and works.
- The bond uses the **active-backup** mode. The bonding driver does not support any other modes for this feature.
- The IPsec connection is configured and works.

Procedure

1. Enable ESP hardware offload support on the network bond:

```
# nmcli connection modify bond0 ethtool.feature-esp-hw-offload on
```

This command enables ESP hardware offload support on the **bond0** connection.

2. Reactivate the **bond0** connection:

```
# nmcli connection up bond0
```

3. Edit the Libreswan configuration file in the **/etc/ipsec.d/** directory of the connection that should use ESP hardware offload, and append the **nic-offload=yes** statement to the connection entry:

```
conn example
    ...
    nic-offload=yes
```

4. Restart the **ipsec** service:

```
# systemctl restart ipsec
```

Verification

1. Display the active port of the bond:

```
# grep "Currently Active Slave" /proc/net/bonding/bond0
Currently Active Slave: enp1s0
```

2. Display the **tx_ipsec** and **rx_ipsec** counters of the active port:

```
# ethtool -S enp1s0 | egrep "_ipsec"
     tx_ipsec: 10
     rx_ipsec: 10
```

3. Send traffic through the IPsec tunnel. For example, ping a remote IP address:

```
# ping -c 5 remote_ip_address
```

4. Display the **tx_ipsec** and **rx_ipsec** counters of the active port again:

```
# ethtool -S enp1s0 | egrep "_ipsec"
     tx_ipsec: 15
     rx_ipsec: 15
```

If the counter values have increased, ESP hardware offload works.

Additional resources

- Configuring network bonding
- Configuring a VPN with IPsec

6.13. 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

- Using system-wide cryptographic policies .

6.14. TROUBLESHOOTING IPSEC VPN CONFIGURATIONS

Problems related to IPsec VPN configurations most commonly occur due to several main reasons. If you are encountering such problems, you can check if the cause of the problem corresponds to any of the following scenarios, and apply the corresponding solution.

Basic connection troubleshooting

Most problems with VPN connections occur in new deployments, where administrators configured endpoints with mismatched configuration options. Also, a working configuration can suddenly stop working, often due to newly introduced incompatible values. This could be the result of an administrator changing the configuration. Alternatively, an administrator may have installed a firmware update or a package update with different default values for certain options, such as encryption algorithms.

To confirm that an IPsec VPN connection is established:

```
# ipsec trafficstatus
006 #8: "vpn.example.com"[1] 192.0.2.1, type=ESP, add_time=1595296930, inBytes=5999,
outBytes=3231, id='@vpn.example.com', lease=100.64.13.5/32
```

If the output is empty or does not show an entry with the connection name, the tunnel is broken.

To check that the problem is in the connection:

1. Reload the *vpn.example.com* connection:

```
# ipsec auto --add vpn.example.com
002 added connection description "vpn.example.com"
```

2. Next, initiate the VPN connection:

```
# ipsec auto --up vpn.example.com
```

Firewall-related problems

The most common problem is that a firewall on one of the IPsec endpoints or on a router between the endpoints is dropping all Internet Key Exchange (IKE) packets.

- For IKEv2, an output similar to the following example indicates a problem with a firewall:

```
# ipsec auto --up vpn.example.com
181 "vpn.example.com"[1] 192.0.2.2 #15: initiating IKEv2 IKE SA
181 "vpn.example.com"[1] 192.0.2.2 #15: STATE_PARENT_I1: sent v2I1, expected v2R1
010 "vpn.example.com"[1] 192.0.2.2 #15: STATE_PARENT_I1: retransmission; will wait 0.5
seconds for response
010 "vpn.example.com"[1] 192.0.2.2 #15: STATE_PARENT_I1: retransmission; will wait 1
seconds for response
010 "vpn.example.com"[1] 192.0.2.2 #15: STATE_PARENT_I1: retransmission; will wait 2
seconds for
...
```

- For IKEv1, the output of the initiating command looks like:

```
# ipsec auto --up vpn.example.com
002 "vpn.example.com" #9: initiating Main Mode
102 "vpn.example.com" #9: STATE_MAIN_I1: sent MI1, expecting MR1
010 "vpn.example.com" #9: STATE_MAIN_I1: retransmission; will wait 0.5 seconds for
response
010 "vpn.example.com" #9: STATE_MAIN_I1: retransmission; will wait 1 seconds for
response
010 "vpn.example.com" #9: STATE_MAIN_I1: retransmission; will wait 2 seconds for
response
...
```

Because the IKE protocol, which is used to set up IPsec, is encrypted, you can troubleshoot only a limited subset of problems using the **tcpdump** tool. If a firewall is dropping IKE or IPsec packets, you can try to find the cause using the **tcpdump** utility. However, **tcpdump** cannot diagnose other problems with IPsec VPN connections.

- To capture the negotiation of the VPN and all encrypted data on the **eth0** interface:

```
# tcpdump -i eth0 -n -n esp or udp port 500 or udp port 4500 or tcp port 4500
```

Mismatched algorithms, protocols, and policies

VPN connections require that the endpoints have matching IKE algorithms, IPsec algorithms, and IP address ranges. If a mismatch occurs, the connection fails. If you identify a mismatch by using one of the following methods, fix it by aligning algorithms, protocols, or policies.

- If the remote endpoint is not running IKE/IPsec, you can see an ICMP packet indicating it. For example:

```
# ipsec auto --up vpn.example.com
...
000 "vpn.example.com"[1] 192.0.2.2 #16: ERROR: asynchronous network error report on
```

```
wlp2s0 (192.0.2.2:500), complainant 198.51.100.1: Connection refused [errno 111, origin
ICMP type 3 code 3 (not authenticated)]
...
```

- Example of mismatched IKE algorithms:

```
# ipsec auto --up vpn.example.com
...
003 "vpn.example.com"[1] 193.110.157.148 #3: dropping unexpected IKE_SA_INIT message
containing NO_PROPOSAL_CHOSEN notification; message payloads: N; missing payloads:
SA,KE,Ni
```

- Example of mismatched IPsec algorithms:

```
# ipsec auto --up vpn.example.com
...
182 "vpn.example.com"[1] 193.110.157.148 #5: STATE_PARENT_I2: sent v2I2, expected
v2R2 {auth=IKEv2 cipher=AES_GCM_16_256 integ=n/a prf=HMAC_SHA2_256
group=MODP2048}
002 "vpn.example.com"[1] 193.110.157.148 #6: IKE_AUTH response contained the error
notification NO_PROPOSAL_CHOSEN
```

  A mismatched IKE version could also result in the remote endpoint dropping the request without a response. This looks identical to a firewall dropping all IKE packets.

- Example of mismatched IP address ranges for IKEv2 (called Traffic Selectors - TS):

```
# ipsec auto --up vpn.example.com
...
1v2 "vpn.example.com" #1: STATE_PARENT_I2: sent v2I2, expected v2R2 {auth=IKEv2
cipher=AES_GCM_16_256 integ=n/a prf=HMAC_SHA2_512 group=MODP2048}
002 "vpn.example.com" #2: IKE_AUTH response contained the error notification
TS_UNACCEPTABLE
```

- Example of mismatched IP address ranges for IKEv1:

```
# ipsec auto --up vpn.example.com
...
031 "vpn.example.com" #2: STATE_QUICK_I1: 60 second timeout exceeded after 0
retransmits.  No acceptable response to our first Quick Mode message: perhaps peer likes
no proposal
```

- When using PreSharedKeys (PSK) in IKEv1, if both sides do not put in the same PSK, the entire IKE message becomes unreadable:

```
# ipsec auto --up vpn.example.com
...
003 "vpn.example.com" #1: received Hash Payload does not match computed value
223 "vpn.example.com" #1: sending notification INVALID_HASH_INFORMATION to
192.0.2.23:500
```

- In IKEv2, the mismatched-PSK error results in an AUTHENTICATION_FAILED message:

```
# ipsec auto --up vpn.example.com
```

```
...
002 "vpn.example.com" #1: IKE SA authentication request rejected by peer:
AUTHENTICATION_FAILED
```

Maximum transmission unit

Other than firewalls blocking IKE or IPsec packets, the most common cause of networking problems relates to an increased packet size of encrypted packets. Network hardware fragments packets larger than the maximum transmission unit (MTU), for example, 1500 bytes. Often, the fragments are lost and the packets fail to re-assemble. This leads to intermittent failures, when a ping test, which uses small-sized packets, works but other traffic fails. In this case, you can establish an SSH session but the terminal freezes as soon as you use it, for example, by entering the 'ls -al /usr' command on the remote host.

To work around the problem, reduce MTU size by adding the **mtu=1400** option to the tunnel configuration file.

Alternatively, for TCP connections, enable an iptables rule that changes the MSS value:

```
# iptables -I FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-to-pmtu
```

If the previous command does not solve the problem in your scenario, directly specify a lower size in the **set-mss** parameter:

```
# iptables -I FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --set-mss 1380
```

Network address translation (NAT)

When an IPsec host also serves as a NAT router, it could accidentally remap packets. The following example configuration demonstrates the problem:

```
conn myvpn
    left=172.16.0.1
    leftsubnet=10.0.2.0/24
    right=172.16.0.2
    rightsubnet=192.168.0.0/16
…
```

The system with address 172.16.0.1 have a NAT rule:

```
iptables -t nat -I POSTROUTING -o eth0 -j MASQUERADE
```

If the system on address 10.0.2.33 sends a packet to 192.168.0.1, then the router translates the source 10.0.2.33 to 172.16.0.1 before it applies the IPsec encryption.

Then, the packet with the source address 10.0.2.33 no longer matches the **conn myvpn** configuration, and IPsec does not encrypt this packet.

To solve this problem, insert rules that exclude NAT for target IPsec subnet ranges on the router, in this example:

```
iptables -t nat -I POSTROUTING -s 10.0.2.0/24 -d 192.168.0.0/16 -j RETURN
```

Kernel IPsec subsystem bugs

The kernel IPsec subsystem might fail, for example, when a bug causes a desynchronizing of the IKE user space and the IPsec kernel. To check for such problems:

```
$ cat /proc/net/xfrm_stat
XfrmInError               0
XfrmInBufferError         0
...
```

Any non-zero value in the output of the previous command indicates a problem. If you encounter this problem, open a new support case, and attach the output of the previous command along with the corresponding IKE logs.

Libreswan logs

Libreswan logs using the **syslog** protocol by default. You can use the **journalctl** command to find log entries related to IPsec. Because the corresponding entries to the log are sent by the **pluto** IKE daemon, search for the "pluto" keyword, for example:

```
$ journalctl -b | grep pluto
```

To show a live log for the **ipsec** service:

```
$ journalctl -f -u ipsec
```

If the default level of logging does not reveal your configuration problem, enable debug logs by adding the **plutodebug=all** option to the **config setup** section in the **/etc/ipsec.conf** file.

Note that debug logging produces a lot of entries, and it is possible that either the **journald** or **syslogd** service rate-limits the **syslog** messages. To ensure you have complete logs, redirect the logging to a file. Edit the **/etc/ipsec.conf**, and add the **logfile=/var/log/pluto.log** in the **config setup** section.

Additional resources

- Troubleshooting problems using log files.
- **tcpdump(8)** and **ipsec.conf(5)** man pages.
- Using and configuring firewalld

6.15. ADDITIONAL RESOURCES

- **ipsec(8)**, **ipsec.conf(5)**, **ipsec.secrets(5)**, **ipsec_auto(8)**, and **ipsec_rsasigkey(8)** man pages.
- **/usr/share/doc/libreswan-*version*/** directory.
- The website of the upstream project.
- The Libreswan Project Wiki.
- All Libreswan man pages.
- NIST Special Publication 800-77: Guide to IPsec VPNs .

CHAPTER 7. CONFIGURING VPN CONNECTIONS WITH IPSEC BY USING THE RHEL SYSTEM ROLE

With the **vpn** system role, you can configure VPN connections on RHEL systems by using Red Hat Ansible Automation Platform. You can use it to set up host-to-host, network-to-network, VPN Remote Access Server, and mesh configurations.

For host-to-host connections, the role sets up a VPN tunnel between each pair of hosts in the list of **vpn_connections** using the default parameters, including generating keys as needed. Alternatively, you can configure it to create an opportunistic mesh configuration between all hosts listed. The role assumes that the names of the hosts under **hosts** are the same as the names of the hosts used in the Ansible inventory, and that you can use those names to configure the tunnels.

NOTE

The **vpn** RHEL system role currently supports only Libreswan, which is an IPsec implementation, as the VPN provider.

7.1. CREATING A HOST-TO-HOST VPN WITH IPSEC BY USING THE VPN RHEL SYSTEM ROLE

You can use the **vpn** system role to configure host-to-host connections by running an Ansible playbook on the control node, which configures all managed nodes listed in an inventory file.

Prerequisites

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

Procedure

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

```
- name: Host to host VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - hosts:
          managed-node-01.example.com:
          managed-node-02.example.com:
    vpn_manage_firewall: true
    vpn_manage_selinux: true
```

This playbook configures the connection **managed-node-01.example.com-to-managed-node-02.example.com** by using pre-shared key authentication with keys auto-generated by the system role. Because **vpn_manage_firewall** and **vpn_manage_selinux** are both set to **true**, the **vpn** role uses the **firewall** and **selinux** roles to manage the ports used by the **vpn** role.

To configure connections from managed hosts to external hosts that are not listed in the inventory file, add the following section to the **vpn_connections** list of hosts:

```
vpn_connections:
  - hosts:
      managed-node-01.example.com:
      <external_node>:
        hostname: <IP_address_or_hostname>
```

This configures one additional connection: **managed-node-01.example.com-to-*<external_node>***

NOTE

The connections are configured only on the managed nodes and not on the external node.

2. Optional: You can specify multiple VPN connections for the managed nodes by using additional sections within **vpn_connections**, for example, a control plane and a data plane:

```
- name: Multiple VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - name: control_plane_vpn
        hosts:
          managed-node-01.example.com:
            hostname: 192.0.2.0 # IP for the control plane
          managed-node-02.example.com:
            hostname: 192.0.2.1
      - name: data_plane_vpn
        hosts:
          managed-node-01.example.com:
            hostname: 10.0.0.1 # IP for the data plane
          managed-node-02.example.com:
            hostname: 10.0.0.2
```

3. Validate the playbook syntax:

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

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

4. Run the playbook:

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

Verification

1. On the managed nodes, confirm that the connection is successfully loaded:

```
# ipsec status | grep <connection_name>
```

Replace *<connection_name>* with the name of the connection from this node, for example **managed_node1-to-managed_node2**.

NOTE

By default, the role generates a descriptive name for each connection it creates from the perspective of each system. For example, when creating a connection between **managed_node1** and **managed_node2**, the descriptive name of this connection on **managed_node1** is **managed_node1-to-managed_node2** but on **managed_node2** the connection is named **managed_node2-to-managed_node1**.

2. On the managed nodes, confirm that the connection is successfully started:

```
# ipsec trafficstatus | grep <connection_name>
```

3. Optional: If a connection does not successfully load, manually add the connection by entering the following command. This provides more specific information indicating why the connection failed to establish:

```
# ipsec auto --add <connection_name>
```

NOTE

Any errors that may occur during the process of loading and starting the connection are reported in the **/var/log/pluto.log** file. Because these logs are hard to parse, manually add the connection to obtain log messages from the standard output instead.

Additional resources

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

7.2. CREATING AN OPPORTUNISTIC MESH VPN CONNECTION WITH IPSEC BY USING THE VPN RHEL SYSTEM ROLE

You can use the **vpn** system role to configure an opportunistic mesh VPN connection that uses certificates for authentication by running an Ansible playbook on the control node, which will configure all the managed nodes listed in an inventory file.

Prerequisites

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

- The IPsec Network Security Services (NSS) crypto library in the **/etc/ipsec.d/** directory contains the necessary certificates.

Procedure

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

```
- name: Mesh VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com, managed-node-03.example.com
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - opportunistic: true
        auth_method: cert
        policies:
          - policy: private
            cidr: default
          - policy: private-or-clear
            cidr: 198.51.100.0/24
          - policy: private
            cidr: 192.0.2.0/24
          - policy: clear
            cidr: 192.0.2.7/32
    vpn_manage_firewall: true
    vpn_manage_selinux: true
```

Authentication with certificates is configured by defining the **auth_method: cert** parameter in the playbook. By default, the node name is used as the certificate nickname. In this example, this is **managed-node-01.example.com**. You can define different certificate names by using the **cert_name** attribute in your inventory.

In this example procedure, the control node, which is the system from which you will run the Ansible playbook, shares the same classless inter-domain routing (CIDR) number as both of the managed nodes (192.0.2.0/24) and has the IP address 192.0.2.7. Therefore, the control node falls under the private policy which is automatically created for CIDR 192.0.2.0/24.

To prevent SSH connection loss during the play, a clear policy for the control node is included in the list of policies. Note that there is also an item in the policies list where the CIDR is equal to default. This is because this playbook overrides the rule from the default policy to make it private instead of private-or-clear.

Because **vpn_manage_firewall** and **vpn_manage_selinux** are both set to **true**, the **vpn** role uses the **firewall** and **selinux** roles to manage the ports used by the **vpn** role.

2. Validate the playbook syntax:

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

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

3. Run the playbook:

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

-

Additional resources

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

CHAPTER 8. USING MACSEC TO ENCRYPT LAYER-2 TRAFFIC IN THE SAME PHYSICAL NETWORK

You can use MACsec to secure the communication between two devices (point-to-point). For example, your branch office is connected over a Metro-Ethernet connection with the central office, you can configure MACsec on the two hosts that connect the offices to increase the security.

Media Access Control security (MACsec) is a layer 2 protocol that secures different traffic types over the Ethernet links including:

- dynamic host configuration protocol (DHCP)
- address resolution protocol (ARP)
- Internet Protocol version 4 / 6 (**IPv4** / **IPv6**) and
- any traffic over IP such as TCP or UDP

MACsec encrypts and authenticates all traffic in LANs, by default with the GCM-AES-128 algorithm, and uses a pre-shared key to establish the connection between the participant hosts. If you want to change the pre-shared key, you need to update the NM configuration on all hosts in the network that uses MACsec.

A MACsec connection uses an Ethernet device, such as an Ethernet network card, VLAN, or tunnel device, as parent. You can either set an IP configuration only on the MACsec device to communicate with other hosts only using the encrypted connection, or you can also set an IP configuration on the parent device. In the latter case, you can use the parent device to communicate with other hosts using an unencrypted connection and the MACsec device for encrypted connections.

MACsec does not require any special hardware. For example, you can use any switch, except if you want to encrypt traffic only between a host and a switch. In this scenario, the switch must also support MACsec.

In other words, there are 2 common methods to configure MACsec;

- host to host and
- host to switch then switch to other host(s)

IMPORTANT

You can use MACsec only between hosts that are in the same (physical or virtual) LAN.

8.1. CONFIGURING A MACSEC CONNECTION BY USING NMCLI

You can configure Ethernet interfaces to use MACsec using the **nmcli** utility. For example, you can create a MACsec connection between two hosts that are connected over Ethernet.

Procedure

1. On the first host on which you configure MACsec:
   - Create the connectivity association key (CAK) and connectivity-association key name (CKN) for the pre-shared key:

a. Create a 16-byte hexadecimal CAK:

```
# dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
50b71a8ef0bd5751ea76de6d6c98c03a
```

b. Create a 32-byte hexadecimal CKN:

```
# dd if=/dev/urandom count=32 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
```

2. On both hosts you want to connect over a MACsec connection:

3. Create the MACsec connection:

```
# nmcli connection add type macsec con-name macsec0 ifname macsec0
connection.autoconnect yes macsec.parent enp1s0 macsec.mode psk macsec.mka-
cak 50b71a8ef0bd5751ea76de6d6c98c03a macsec.mka-ckn
f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
```

Use the CAK and CKN generated in the previous step in the **macsec.mka-cak** and **macsec.mka-ckn** parameters. The values must be the same on every host in the MACsec-protected network.

4. Configure the IP settings on the MACsec connection.

a. Configure the **IPv4** settings. For example, to set a static **IPv4** address, network mask, default gateway, and DNS server to the **macsec0** connection, enter:

```
# nmcli connection modify macsec0 ipv4.method manual ipv4.addresses
'192.0.2.1/24' ipv4.gateway '192.0.2.254' ipv4.dns '192.0.2.253'
```

b. Configure the **IPv6** settings. For example, to set a static **IPv6** address, network mask, default gateway, and DNS server to the **macsec0** connection, enter:

```
# nmcli connection modify macsec0 ipv6.method manual ipv6.addresses
'2001:db8:1::1/32' ipv6.gateway '2001:db8:1::fffe' ipv6.dns '2001:db8:1::fffd'
```

5. Activate the connection:

```
# nmcli connection up macsec0
```

Verification

1. Verify that the traffic is encrypted:

```
# tcpdump -nn -i enp1s0
```

2. Optional: Display the unencrypted traffic:

```
# tcpdump -nn -i macsec0
```

3. Display MACsec statistics:

```
# ip macsec show
```

4. Display individual counters for each type of protection: integrity-only (encrypt off) and encryption (encrypt on)

```
# ip -s macsec show
```

8.2. ADDITIONAL RESOURCES

- MACsec: a different solution to encrypt network traffic blog.

CHAPTER 9. USING AND CONFIGURING FIREWALLD

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

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

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

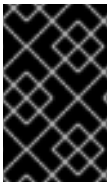
Services use one or more ports or addresses for network communication. Firewalls filter communication based on ports. To allow network traffic for a service, its ports must be open. **firewalld** blocks all traffic on ports that are not explicitly set as open. Some zones, such as trusted, allow all traffic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

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

9.1. WHEN TO USE FIREWALLD, NFTABLES, OR IPTABLES

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

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



IMPORTANT

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

9.2. FIREWALL ZONES

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

firewalld follows strict principles in regards to zones:

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

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

Principles 4 and 5 are a consequence of principle 3.

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

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

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

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

The **/usr/lib/firewalld/zones/** directory stores the predefined zones, and you can instantly apply them to any available network interface. These files are copied to the **/etc/firewalld/zones/** directory only after they are modified. The default settings of the predefined zones are as follows:

block

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

dmz

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

drop

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

- Accepts: Only outgoing network connections.

external

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

home

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

internal

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

public

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

trusted

- Accepts: All network connections.

work

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

- Accepts: Only selected incoming connections.

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

NOTE

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

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

Additional resources

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

9.3. FIREWALL POLICIES

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

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

- Network address translations (NAT)

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

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

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

9.4. FIREWALL RULES

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

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

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

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

9.5. ZONE CONFIGURATION FILES

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

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

```
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>My Zone</short>
  <description>Here you can describe the characteristic features of the zone.</description>
  <service name="ssh"/>
  <port protocol="udp" port="1025-65535"/>
  <port protocol="tcp" port="1025-65535"/>
</zone>
```

Additional resources

- **firewalld.zone** manual page

9.6. PREDEFINED FIREWALLD SERVICES

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

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

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

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

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

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

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

Additional resources

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

9.7. WORKING WITH FIREWALLD ZONES

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

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

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

For example, you can achieve the following benefits:

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

Prerequisites

- The **firewalld** service is running.

Procedure

1. List the available firewall zones:

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

   The **firewall-cmd --get-zones** command displays all zones that are available on the system, but it does not show any details for particular zones. To see more detailed information for all zones, use the **firewall-cmd --list-all-zones** command.

2. Choose the zone you want to use for this configuration.

3. Modify firewall settings for the chosen zone. For example, to allow the **SSH** service and remove the **ftp** service:

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

4. Assign a network interface to the firewall zone:

   a. List the available network interfaces:

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

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

   b. Assign a network interface to the chosen zone:

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

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

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

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

Verification

1. Display the updated settings for your chosen zone:

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

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

9.7.2. Changing the default zone

System administrators assign a zone to a networking interface in its configuration files. If an interface is not assigned to a specific zone, it is assigned to the default zone. After each restart of the **firewalld** service, **firewalld** loads the settings for the default zone and makes it active. Note that settings for all other zones are preserved and ready to be used.

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

Prerequisites

- The **firewalld** service is running.

Procedure

To set up the default zone:

1. Display the current default zone:

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

2. Set the new default zone:

```
# firewall-cmd --set-default-zone <zone_name>
```

NOTE

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

9.7.3. Assigning a network interface to a zone

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

Procedure

To assign the zone to a specific interface:

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

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

2. Assign the interface to a different zone:

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

9.7.4. Assigning a zone to a connection using nmcli

You can add a **firewalld** zone to a **NetworkManager** connection using the **nmcli** utility.

Procedure

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

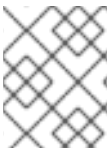
```
# nmcli connection modify profile connection.zone zone_name
```

2. Activate the connection:

```
# nmcli connection up profile
```

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

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



NOTE

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

Procedure

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

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

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

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

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

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

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

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

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

```
ZONE=internal
```

3. Reload the connection profiles:

```
# nmcli connection reload
```

4. Reactivate the connection profiles

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

Verification

- Display the zone of the interface, for example:

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

9.7.6. 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 profile, a zone can be specified, which provides the flexibility of various firewall settings according to the location of the computer with portable devices. Thus, zones and settings can be specified for different locations, such as company or home.

Procedure

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

```
ZONE=zone_name
```

9.7.7. 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.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Create a new zone:

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

2. Make the new zone usable:

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

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

Verification

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

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

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

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

- **ACCEPT**: Accepts all incoming packets except those disallowed by specific rules.
- **REJECT**: Rejects all incoming packets except those allowed by specific rules. When **firewalld** rejects packets, the source machine is informed about the rejection.
- **DROP**: Drops all incoming packets except those allowed by specific rules. When **firewalld** drops packets, the source machine is not informed about the packet drop.
- **default**: Similar behavior as for **REJECT**, but with special meanings in certain scenarios.

Prerequisites

- The **firewalld** service is running.

Procedure

To set a target for a zone:

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

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

2. Set a new target in the zone:

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

Additional resources

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

9.8. CONTROLLING NETWORK TRAFFIC USING FIREWALLD

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

9.8.1. Controlling traffic with predefined services using the CLI

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

Prerequisites

- The **firewalld** service is running.

Procedure

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

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

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

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

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

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

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

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

   The command adds the specified service to the default zone.

4. Make the new settings persistent:

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

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

Verification

1. List all permanent firewall rules:

```
# firewall-cmd --list-all --permanent
public
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
```

```
services: cockpit dhcpv6-client ssh
ports:
protocols:
forward: no
masquerade: no
forward-ports:
source-ports:
icmp-blocks:
rich rules:
```

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

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

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

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

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

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

9.8.2. Controlling traffic with predefined services using the GUI

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

Prerequisites

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

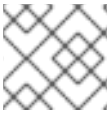
Procedure

1. To enable or disable a predefined or custom service:
   a. Start the **firewall-config** utility and select the network zone whose services are to be configured.
   b. Select the **Zones** tab and then the **Services** tab below.
   c. Select the checkbox for each type of service you want to trust or clear the checkbox to block a service in the selected zone.
2. To edit a service:
   a. Start the **firewall-config** utility.

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

c. Select the service you want to configure.

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

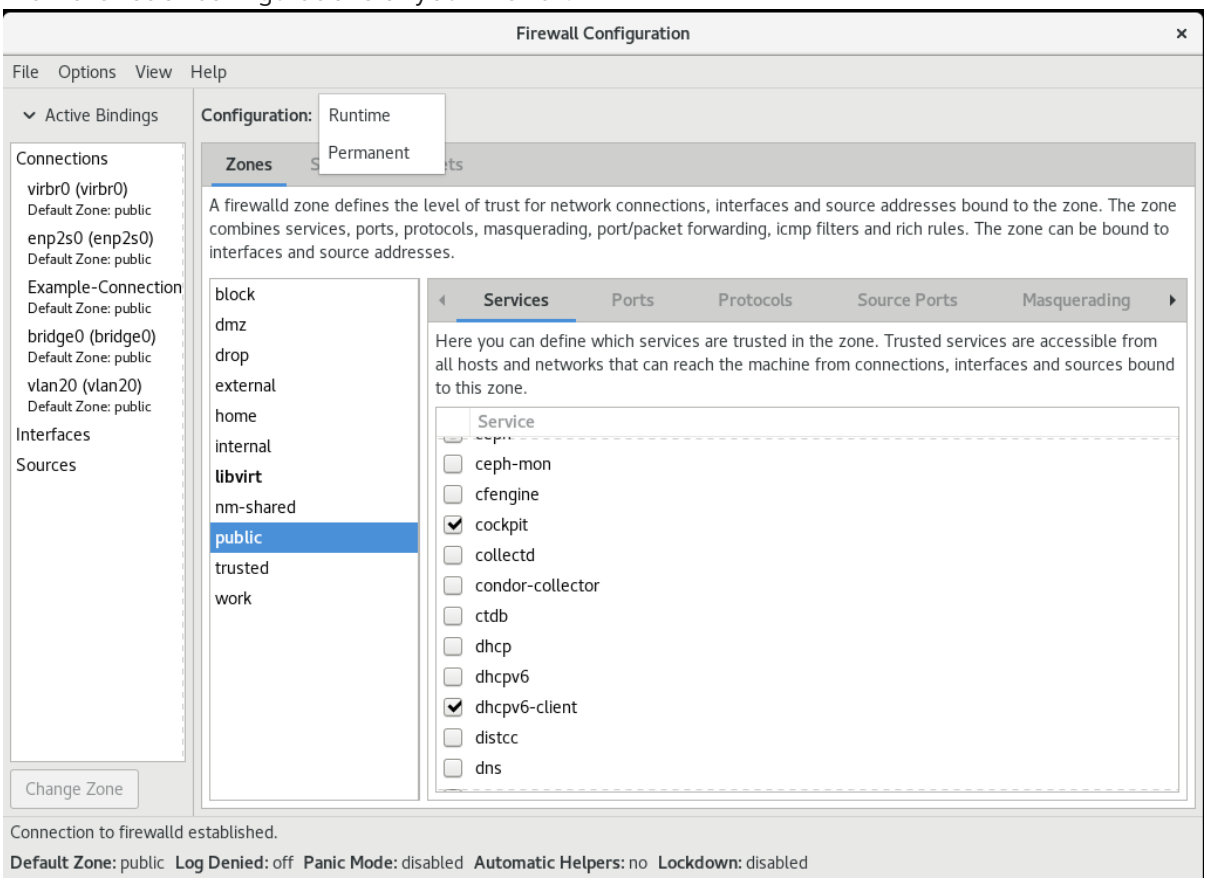


NOTE

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

Verification

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



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

9.8.3. Configuring firewalld to allow hosting a secure web server

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

Normally, system services listen on standard ports that are reserved for them. The **httpd** daemon, for example, listens on port 80. However, system administrators can directly specify the port number instead of the service name.

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

Prerequisites

- The **firewalld** service is running.

Procedure

1. Check the currently active firewall zone:

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

2. Add the HTTPS service to the appropriate zone:

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

3. Reload the firewall configuration:

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

Verification

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

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

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

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

9.8.4. Closing unused or unnecessary ports to enhance network security

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

IMPORTANT

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

Procedure

1. List all allowed ports:

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

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

NOTE

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

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

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

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

3. Make the new settings persistent:

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

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

Verification

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

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

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

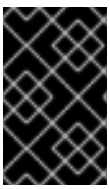
```
# firewall-cmd --zone=<zone_to_inspect> --list-ports
```

9.8.5. Controlling traffic through the CLI

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

- disable networking traffic
- enable networking traffic

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



IMPORTANT

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

Procedure

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

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

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

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

Verification

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

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

9.8.6. Controlling traffic with protocols using GUI

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

Prerequisites

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

Procedure

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

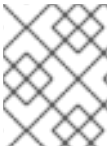
9.9. USING ZONES TO MANAGE INCOMING TRAFFIC DEPENDING ON A SOURCE

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

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

9.9.1. Adding a source

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



NOTE

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

- To set the source in the current zone:

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

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

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

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

Procedure

1. List all available zones:

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

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

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

3. Make the new settings persistent:

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

9.9.2. Removing a source

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

Procedure

1. List allowed sources for the required zone:

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

2. Remove the source from the zone permanently:

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

3. Make the new settings persistent:

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

9.9.3. Removing a source port

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

Procedure

- To remove a source port:

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

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

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

WARNING

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

Procedure

1. List all available zones:

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

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

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

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

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

4. Make the new settings persistent:

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

Verification

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

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

Additional resources

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

9.10. FILTERING FORWARDED TRAFFIC BETWEEN ZONES

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

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

9.10.1. The relationship between policy objects and zones

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

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

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

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

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

- The **HOST** symbolic zone allows policies for the traffic originating from or has a destination to the host running firewalld.
- The **ANY** symbolic zone applies policy to all the current and future zones. **ANY** symbolic zone acts as a wildcard for all zones.

9.10.2. Using priorities to sort policies

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

To set a priority to sort the policies:

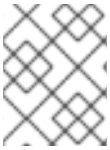
```
# firewall-cmd --permanent --policy mypolicy --set-priority -500
```

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

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

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

The policy objects feature allows users to filter traffic between Podman and firewalld zones.



NOTE

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

Procedure

1. Create a new firewall policy:

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

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

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

3. Create a new Podman zone:

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

4. Define the ingress zone for the policy:

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

5. Define the egress zone for all other zones:

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

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

6. Restart the firewalld service:

```
# systemctl restart firewalld
```

Verification

- Verify the Podman firewall policy to other zones:

```
# firewall-cmd --info-policy podmanToAny
podmanToAny (active)
  ...
  target: REJECT
```

```
ingress-zones: podman
egress-zones: ANY
services: dhcp dns https
...
```

9.10.4. Setting the default target of policy objects

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

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

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

Verification

- Verify information about the policy

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

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

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

Prerequisites

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

Procedure

1. Create a firewall policy:

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

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

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

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

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

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

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

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

4. Reload the firewall configuration files:

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

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

   - For persistent changes, run:

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

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

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

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

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

Verification

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

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

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

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

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

```
# firewall-cmd --info-policy=<example_policy>
example_policy (active)
  priority: -1
  target: CONTINUE
  ingress-zones: HOST
  egress-zones: ANY
  services:
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
 rule family="ipv4" destination address="192.0.2.1" forward-port port="443" protocol="tcp" to-port="443" to-addr="192.51.100.20"
```

Additional resources

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

9.11. CONFIGURING NAT USING FIREWALLD

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

- Masquerading
- Destination NAT (DNAT)
- Redirect

9.11.1. Network address translation types

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

Masquerading

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers (ISPs) do not route private IP ranges, such as **10.0.0.0/8**. If you use private IP ranges in your network and users should be able to reach servers on the internet, map the source IP address of packets from these ranges to a public IP address.
Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.

Destination NAT (DNAT)

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

Redirect

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

9.11.2. Configuring IP address masquerading

You can enable IP masquerading on your system. IP masquerading hides individual machines behind a gateway when accessing the internet.

Procedure

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

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

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

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

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

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

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

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

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

9.11.3. Using DNAT to forward incoming HTTP traffic

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

Prerequisites

- The **firewalld** service is running.

Procedure

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

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

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

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

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

3. Forward incoming HTTP traffic:

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

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

- **--zone=public** - The firewall zone for which you configure the DNAT rule. You can adjust this to whatever zone you need.
- **--add-forward-port** - The option that indicates you are adding a port-forwarding rule.
- **port=80** - The external destination port.
- **proto=tcp** - The protocol indicating that you forward TCP traffic.
- **toaddr=198.51.100.10** - The destination IP address.
- **toport=8080** - The destination port of the internal server.
- **--permanent** - The option that makes the DNAT rule persistent across reboots.

4. Reload the firewall configuration to apply the changes:

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

Verification

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

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

Alternatively, view the corresponding XML configuration file:

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

Additional resources

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

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

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

Prerequisites

- The **firewalld** service is running.

Procedure

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

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

   This setting enables IP forwarding in the kernel.

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

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

3. Create the NAT redirect rule:

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

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

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

4. Reload the firewall configuration to apply the changes:

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

Verification

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

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

Alternatively, view the corresponding XML configuration file:

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

Additional resources

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

9.12. MANAGING ICMP REQUESTS

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

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

9.12.1. Configuring ICMP filtering

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

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

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

Prerequisites

- The **firewalld** service is running.

Procedure

1. List available ICMP types and codes:

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

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

2. Filter specific ICMP types by:

   - Allowing ICMP types:

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

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

   - Blocking ICMP types:

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

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

3. Reload the firewall configuration to apply the changes:

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

Verification

- Verify your filtering rules are in effect:

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

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

Additional resources

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

9.13. SETTING AND CONTROLLING IP SETS USING FIREWALLD

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

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

- Handle large lists of IP addresses

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

WARNING

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

9.13.1. Configuring dynamic updates for allowlisting with IP sets

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

Prerequisites

- The **firewalld** service is running.

Procedure

1. Create an IP set with a meaningful name:

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

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

2. Add a dynamic update to the IP set:

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

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

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

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

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

4. Reload the firewall configuration to apply the changes:

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

Verification

1. List all IP sets:

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

2. List the active rules:

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

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

3. Explore the contents of your IP set:

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

Next steps

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

Additional resources

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

9.14. PRIORITIZING RICH RULES

By default, rich rules are organized based on their rule action. For example, **deny** rules have precedence over **allow** rules. The **priority** parameter in rich rules provides administrators fine-grained control over rich rules and their execution order. When using the **priority** parameter, rules are sorted first by their priority values in ascending order. When more rules have the same **priority**, their order is determined by the rule action, and if the action is also the same, the order may be undefined.

9.14.1. How the priority parameter organizes rules into different chains

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

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

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

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

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

9.14.2. Setting the priority of a rich rule

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

Procedure

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

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

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

Verification

- Display the **nftables** rule that the command in the previous step created:

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

9.15. CONFIGURING FIREWALL LOCKDOWN

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

9.15.1. Configuring lockdown using CLI

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

Procedure

1. To query whether lockdown is enabled:

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

2. Manage lockdown configuration by either:

- Enabling lockdown:

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

- Disabling lockdown:

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

9.15.2. Overview of lockdown allowlist configuration files

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

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

```
<?xml version="1.0" encoding="utf-8"?>
 <whitelist>
  <command name="/usr/bin/python3 -s /usr/bin/firewall-config"/>
   <selinux context="system_u:system_r:NetworkManager_t:s0"/>
   <selinux context="system_u:system_r:virtd_t:s0-s0:c0.c1023"/>
   <user id="0"/>
 </whitelist>
```

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

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

This example shows both **user id** and **user name**, but only one option is required. Python is the interpreter and is prepended to the command line.

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

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

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

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

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

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

NOTE

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

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

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

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

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

You can use intra-zone forwarding to forward traffic between interfaces and sources within the same **firewalld** zone. This feature brings the following benefits:

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

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

Procedure

1. Enable packet forwarding in the kernel:

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

2. Ensure that interfaces between which you want to enable intra-zone forwarding are assigned only to the **internal** zone:

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

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

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

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

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

5. Enable intra-zone forwarding:

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

Verification

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

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

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

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

```
# ncat <other_host> 12345
```

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

Additional resources

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

9.17. CONFIGURING FIREWALLD BY USING THE RHEL SYSTEM ROLE

You can use the **firewall** RHEL system role to configure settings of the **firewalld** service on multiple clients at once. This solution:

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

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

9.17.1. Introduction to the firewall RHEL system role

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

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

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

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

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

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

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file
- **/usr/share/doc/rhel-system-roles/firewall/** directory
- Working with playbooks
- How to build your inventory

9.17.2. Resetting the firewalld settings by using the firewall RHEL system role

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

Perform this procedure on the Ansible control node.

Prerequisites

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

Procedure

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

```
---
- name: Reset firewalld example
  hosts: managed-node-01.example.com
  tasks:
    - name: Reset firewalld
```

```
    ansible.builtin.include_role:
      name: rhel-system-roles.firewall
    vars:
      firewall:
        - previous: replaced
```

2. Validate the playbook syntax:

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

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

3. Run the playbook:

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

Verification

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

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

Additional resources

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

9.17.3. Forwarding incoming traffic in firewalld from one local port to a different local port by using the firewall RHEL system role

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

Perform this procedure on the Ansible control node.

Prerequisites

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

Procedure

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

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
```

```
- name: Forward incoming traffic on port 8080 to 443
  ansible.builtin.include_role:
    name: rhel-system-roles.firewall
  vars:
    firewall:
      - { forward_port: 8080/tcp;443;, state: enabled, runtime: true, permanent: true }
```

2. Validate the playbook syntax:

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

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

3. Run the playbook:

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

Verification

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

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

Additional resources

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

9.17.4. Managing ports in firewalld by using the firewall RHEL system role

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

Perform this procedure on the Ansible control node.

Prerequisites

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

Procedure

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

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
```

```
tasks:
  - name: Allow incoming HTTPS traffic to the local host
    ansible.builtin.include_role:
      name: rhel-system-roles.firewall
    vars:
      firewall:
        - port: 443/tcp
          service: http
          state: enabled
          runtime: true
          permanent: true
```

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

2. Validate the playbook syntax:

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

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

3. Run the playbook:

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

Verification

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

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

Additional resources

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

9.17.5. Configuring a firewalld DMZ zone by using the firewall RHEL system role

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

Perform this procedure on the Ansible control node.

Prerequisites

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

Procedure

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

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

2. Validate the playbook syntax:

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

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

3. Run the playbook:

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

Verification

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

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

Additional resources

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

- **/usr/share/doc/rhel-system-roles/firewall/** directory

CHAPTER 10. GETTING STARTED WITH NFTABLES

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

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

The **nftables** framework uses tables to store chains. The chains contain individual rules for performing actions. The **nft** utility replaces all tools from the previous packet-filtering frameworks. You can use the **libnftnl** library for low-level interaction with **nftables** Netlink API through the **libmnl** library.

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

10.1. MIGRATING FROM IPTABLES TO NFTABLES

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

10.1.1. When to use firewalld, nftables, or iptables

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

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

IMPORTANT

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

10.1.2. Converting iptables and ip6tables rule sets to nftables

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

Prerequisites

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

Procedure

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

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

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

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

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

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

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

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

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

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

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

Verification

- Display the **nftables** rule set:

```
# nft list ruleset
```

Additional resources

- Automatically loading nftables rules when the system boots

10.1.3. Converting single iptables and ip6tables rules to nftables

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

Prerequisites

- The **nftables** package is installed.

Procedure

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

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

Note that some extensions lack translation support. In these cases, the utility prints the untranslated rule prefixed with the **#** sign, for example:

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

Additional resources

- **iptables-translate --help**

10.1.4. Comparison of common iptables and nftables commands

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

- Listing all rules:

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

- Listing a certain table and chain:

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

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

Listing rules generated by firewalld:

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

10.2. WRITING AND EXECUTING NFTABLES SCRIPTS

The major benefit of using the **nftables** framework is that the execution of scripts is atomic. This means that the system either applies the whole script or prevents the execution if an error occurs. This guarantees that the firewall is always in a consistent state.

Additionally, with the **nftables** script environment, you can:

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

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

10.2.1. Supported nftables script formats

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

- The same format as the **nft list ruleset** command displays the rule set:

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

# Flush the rule set
flush ruleset

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

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

- The same syntax as for **nft** commands:

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

# Flush the rule set
flush ruleset

# Create a table
add table inet example_table

# Create a chain for incoming packets that drops all packets
```

```
# that are not explicitly allowed by any rule in this chain
add chain inet example_table example_chain { type filter hook input priority 0 ; policy drop ; }

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

10.2.2. Running nftables scripts

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

Procedure

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

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

- To run an **nftables** script directly:
  a. For the single time that you perform this:
     i. Ensure that the script starts with the following shebang sequence:

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

IMPORTANT

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

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

```
# chown root /etc/nftables/<example_firewall_script>.nft
```

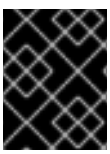
     iii. Make the script executable for the owner:

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

  b. Run the script:

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

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



IMPORTANT

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

Additional resources

- **chown(1)** man page

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

10.2.3. Using comments in nftables scripts

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

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

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

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

10.2.4. Using variables in nftables script

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

Variables with a single value

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

```
define INET_DEV = enp1s0
```

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

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

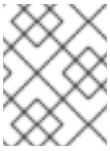
Variables that contain an anonymous set

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

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

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

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



NOTE

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

Additional resources

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

10.2.5. Including files in nftables scripts

In the **nftables** scripting environment, you can include other scripts by using the **include** statement.

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

Example 10.1. Including files from the default search directory

To include a file from the default search directory:

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

Example 10.2. Including all *.nft files from a directory

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

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

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

Additional resources

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

10.2.6. Automatically loading nftables rules when the system boots

The **nftables** systemd service loads firewall scripts that are included in the **/etc/sysconfig/nftables.conf** file.

Prerequisites

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

Procedure

1. Edit the **/etc/sysconfig/nftables.conf** file.
   - If you modified the ***.nft** scripts that were created in **/etc/nftables/** with the installation of the **nftables** package, uncomment the **include** statement for these scripts.
   - If you wrote new scripts, add **include** statements to include these scripts. For example, to load the **/etc/nftables/*example*.nft** script when the **nftables** service starts, add:

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

2. Optional: Start the **nftables** service to load the firewall rules without rebooting the system:

```
# systemctl start nftables
```

3. Enable the **nftables** service.

```
# systemctl enable nftables
```

Additional resources

- Supported nftables script formats

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

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

10.3.1. Basics of nftables tables

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

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

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

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

- In scripts in native syntax, use:

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

- In shell scripts, use:

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

10.3.2. Basics of nftables chains

Tables consist of chains which in turn are containers for rules. The following two rule types exists:

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

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

- In scripts in native syntax, use:

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

- In shell scripts, use:

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

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

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

Chain types

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

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

Chain priorities

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

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

Textual value	Numeric value	Address families	Hooks
raw	**-300**	**ip**, **ip6**, **inet**	all
mangle	**-150**	**ip**, **ip6**, **inet**	all

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

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

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

10.3.3. Basics of nftables rules

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

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

- In scripts in native syntax, use:

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

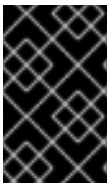
- In shell scripts, use:

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

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

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

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



IMPORTANT

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

Procedure

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

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

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

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

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

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

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

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

4. Display the current rule set including handles:

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

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

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

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

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

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

```
# nft -a list table inet nftables_svc
table inet nftables_svc { # handle 13
  chain INPUT { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 636 accept # handle 5
    tcp dport 443 accept # handle 3
    tcp dport 80 accept # handle 6
    reject # handle 4
  }
}
```

8. Remove the rule with handle 6:

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

To remove a rule, you must specify the handle.

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

```
# nft -a list table inet nftables_svc
table inet nftables_svc { # handle 13
  chain INPUT { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 636 accept # handle 5
    tcp dport 443 accept # handle 3
    reject # handle 4
  }
}
```

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

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

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

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

12. Delete the **INPUT** chain:

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

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

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

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

14. Delete the **nftables_svc** table:

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

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

NOTE

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

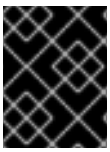
Additional resources

nft(8) man page

10.4. CONFIGURING NAT USING NFTABLES

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

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



IMPORTANT

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

10.4.1. NAT types

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

Masquerading and source NAT (SNAT)

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

- Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.

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

Destination NAT (DNAT)

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

Redirect

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

10.4.2. Configuring masquerading using nftables

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

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

Procedure

1. Create a table:

```
# nft add table nat
```

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

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

IMPORTANT

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

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

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

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

10.4.3. Configuring source NAT using nftables

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

Procedure

1. Create a table:

```
# nft add table nat
```

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

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

IMPORTANT

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

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

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

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

Additional resources

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

10.4.4. Configuring destination NAT using nftables

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

For example, with DNAT the router redirects incoming traffic sent to port **80** and **443** to a web server with the IP address **192.0.2.1**.

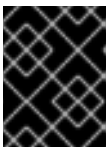
Procedure

1. Create a table:

```
# nft add table nat
```

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

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



IMPORTANT

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

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

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

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

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

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

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

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

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

5. Enable packet forwarding:

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

Additional resources

- NAT types

10.4.5. Configuring a redirect using nftables

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

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

Procedure

1. Create a table:

```
# nft add table nat
```

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

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

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

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

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

Additional resources

- NAT types

10.4.6. Configuring flowtable by using nftables

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

The flowtable mechanism has the following features:

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

Procedure

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

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

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

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

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

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

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

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

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

Verification

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

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

    chain example-forwardchain {
type filter hook forward priority filter; policy accept;
```

```
        ct state established flow add @example-flowtable
    }
}
```

Additional resources

- **nft(8)** man page

10.5. USING SETS IN NFTABLES COMMANDS

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

10.5.1. Using anonymous sets in nftables

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

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

Prerequisites

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

Procedure

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

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

2. Optional: Display all chains and their rules in **example_table**:

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

10.5.2. Using named sets in nftables

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

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

- **ipv4_addr** for a set that contains IPv4 addresses or ranges, such as **192.0.2.1** or **192.0.2.0/24**.

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

Prerequisites

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

Procedure

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

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

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

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

     IMPORTANT

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

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

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

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

3. Add IPv4 addresses to **example_set**:
   - If you create a set that stores individual IPv4 addresses, enter:

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

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

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

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

10.5.3. Additional resources

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

10.6. USING VERDICT MAPS IN NFTABLES COMMANDS

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

10.6.1. Using anonymous maps in nftables

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

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

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

Procedure

1. Create a new table:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  chain udp_packets {
    counter packets 10 bytes 1559
  }

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

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

10.6.2. Using named maps in nftables

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

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

- **ipv4_addr** for a map whose match part contains an IPv4 address, such as **192.0.2.1**.
- **ipv6_addr** for a map whose match part contains an IPv6 address, such as **2001:db8:1::1**.
- **ether_addr** for a map whose match part contains a media access control (MAC) address, such as **52:54:00:6b:66:42**.
- **inet_proto** for a map whose match part contains an internet protocol type, such as **tcp**.
- **inet_service** for a map whose match part contains an internet services name port number, such as **ssh** or **22**.
- **mark** for a map whose match part contains a packet mark. A packet mark can be any positive 32-bit integer value (**0** to **2147483647**).
- **counter** for a map whose match part contains a counter value. The counter value can be any positive 64-bit integer value.
- **quota** for a map whose match part contains a quota value. The quota value can be any positive 64-bit integer value.

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

Procedure

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

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

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

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

IMPORTANT

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

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

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

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

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

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

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

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

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

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

7. Optional: Remove an entry from the map:

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

8. Optional: Display the rule set:

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

  chain example_chain {
    type filter hook input priority filter; policy accept;
```

```
    ip saddr vmap @example_map
  }
}
```

10.6.3. Additional resources

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

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

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

IMPORTANT

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

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

10.7.1. Network conditions

The network in this example has the following conditions:

- The router is connected to the following networks:
  - The internet through interface **enp1s0**
  - The internal LAN through interface **enp7s0**
  - The DMZ through **enp8s0**
- The internet interface of the router has both a static IPv4 address (**203.0.113.1**) and IPv6 address (**2001:db8:a::1**) assigned.
- The clients in the internal LAN use only private IPv4 addresses from the range **10.0.0.0/24**. Consequently, traffic from the LAN to the internet requires source network address translation (SNAT).
- The administrator PCs in the internal LAN use the IP addresses **10.0.0.100** and **10.0.0.200**.
- The DMZ uses public IP addresses from the ranges **198.51.100.0/24** and **2001:db8:b::/56**.
- The web server in the DMZ uses the IP addresses **198.51.100.5** and **2001:db8:b::5**.
- The router acts as a caching DNS server for hosts in the LAN and DMZ.

10.7.2. Security requirements to the firewall script

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

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

10.7.3. Configuring logging of dropped packets to a file

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

Prerequisites

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

Procedure

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

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

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

2. Restart the **rsyslog** service:

```
# systemctl restart rsyslog
```

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

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

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

Additional resources

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

10.7.4. Writing and activating the nftables script

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

WARNING

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

Prerequisites

- The network is configured as described in Network conditions.

Procedure

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

```
# Remove all rules
flush ruleset


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

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


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


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

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

  # Accept incoming traffic on loopback interface
  iifname lo accept

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

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

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


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

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

  # Accept outgoing traffic on loopback interface
  oifname lo accept

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

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


# Chain for forwarding traffic. Default policy: drop
```

```
  chain FORWARD {
    type filter hook forward priority filter
    policy drop

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

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

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

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

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


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

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

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

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

3. Enable IPv4 forwarding:

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

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

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

Verification

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

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

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

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

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

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

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

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

10.8. CONFIGURING PORT FORWARDING USING NFTABLES

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

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

10.8.1. Forwarding incoming packets to a different local port

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

Procedure

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

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

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

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

NOTE

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

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

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

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

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

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

Prerequisites

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

Procedure

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

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

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

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

NOTE

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

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

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

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

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

5. Enable packet forwarding:

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

10.9. USING NFTABLES TO LIMIT THE AMOUNT OF CONNECTIONS

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

10.9.1. Limiting the number of connections using nftables

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

Prerequisites

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

Procedure

1. Create a dynamic set for IPv4 addresses:

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

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

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

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

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

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

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

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

Procedure

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

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

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

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

3. Add a rule that drops all packets from source addresses that attempt to establish more than ten TCP connections within one minute:

```
# nft add rule ip filter input ip protocol tcp ct state new, untracked meter ratemeter { ip saddr timeout 5m limit rate over 10/minute } drop
```

The **timeout 5m** parameter defines that **nftables** automatically removes entries after five minutes to prevent that the meter fills up with stale entries.

Verification

- To display the meter's content, enter:

```
# nft list meter ip filter ratemeter
table ip filter {
  meter ratemeter {
    type ipv4_addr
    size 65535
    flags dynamic,timeout
    elements = { 192.0.2.1 limit rate over 10/minute timeout 5m expires 4m58s224ms }
  }
}
```

10.10. DEBUGGING NFTABLES RULES

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

10.10.1. Creating a rule with a counter

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

- For more information about a procedure that adds a counter to an existing rule, see Adding a counter to an existing rule in **Configuring and managing networking**

Prerequisites

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

Procedure

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

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

2. To display the counter values:

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

10.10.2. Adding a counter to an existing rule

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

- For more information about a procedure that adds a new rule with a counter, see Creating a rule with the counter in **Configuring and managing networking**

Prerequisites

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

Procedure

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

```
# nft --handle list chain inet example_table example_chain
table inet example_table {
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept # handle 4
  }
}
```

2. Add the counter by replacing the rule but with the **counter** parameter. The following example replaces the rule displayed in the previous step and adds a counter:

```
# nft replace rule inet example_table example_chain handle 4 tcp dport 22 counter accept
```

3. To display the counter values:

```
# nft list ruleset
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport ssh counter packets 6872 bytes 105448565 accept
  }
}
```

10.10.3. Monitoring packets that match an existing rule

The tracing feature in **nftables** in combination with the **nft monitor** command enables administrators to display packets that match a rule. You can enable tracing for a rule an use it to monitoring packets that match this rule.

Prerequisites

- The rule to which you want to add the counter exists.

Procedure

1. Display the rules in the chain including their handles:

```
# nft --handle list chain inet example_table example_chain
table inet example_table {
```

```
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept # handle 4
  }
}
```

2. Add the tracing feature by replacing the rule but with the **meta nftrace set 1** parameters. The following example replaces the rule displayed in the previous step and enables tracing:

```
# nft replace rule inet example_table example_chain handle 4 tcp dport 22 meta nftrace set 1 accept
```

3. Use the **nft monitor** command to display the tracing. The following example filters the output of the command to display only entries that contain **inet example_table example_chain**:

```
# nft monitor | grep "inet example_table example_chain"
trace id 3c5eb15e inet example_table example_chain packet: iif "enp1s0" ether saddr 52:54:00:17:ff:e4 ether daddr 52:54:00:72:2f:6e ip saddr 192.0.2.1 ip daddr 192.0.2.2 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 49710 ip protocol tcp ip length 60 tcp sport 56728 tcp dport ssh tcp flags == syn tcp window 64240
trace id 3c5eb15e inet example_table example_chain rule tcp dport ssh nftrace set 1 accept (verdict accept)
...
```

WARNING

Depending on the number of rules with tracing enabled and the amount of matching traffic, the **nft monitor** command can display a lot of output. Use **grep** or other utilities to filter the output.

10.11. BACKING UP AND RESTORING THE NFTABLES RULE SET

You can backup **nftables** rules to a file and later restoring them. Also, administrators can use a file with the rules to, for example, transfer the rules to a different server.

10.11.1. Backing up the nftables rule set to a file

You can use the **nft** utility to back up the **nftables** rule set to a file.

Procedure

- To backup **nftables** rules:
  - In a format produced by **nft list ruleset** format:

```
# nft list ruleset > file.nft
```

  - In JSON format:

```
# nft -j list ruleset > file.json
```

10.11.2. Restoring the nftables rule set from a file

You can restore the **nftables** rule set from a file.

Procedure

- To restore **nftables** rules:
  - If the file to restore is in the format produced by **nft list ruleset** or contains **nft** commands directly:

    ```
    # nft -f file.nft
    ```

  - If the file to restore is in JSON format:

    ```
    # nft -j -f file.json
    ```

10.12. ADDITIONAL RESOURCES

- Using nftables in Red Hat Enterprise Linux 8
- What comes after iptables? Its successor, of course: nftables
- Firewalld: The Future is nftables

CHAPTER 11. SECURING NETWORK SERVICES

Red Hat Enterprise Linux 8 supports many different types of network servers. Their network services can expose the system security to risks of various types of attacks, such as denial of service attacks (DoS), distributed denial of service attacks (DDoS), script vulnerability attacks, and buffer overflow attacks.

To increase the system security against attacks, it is important to monitor active network services that you use. For example, when a network service is running on a machine, its daemon listens for connections on network ports, and this can reduce the security. To limit exposure to attacks over the network, all services that are unused should be turned off.

11.1. SECURING THE RPCBIND SERVICE

The **rpcbind** service is a dynamic port-assignment daemon for remote procedure calls (RPC) services such as Network Information Service (NIS) and Network File System (NFS). Because it has weak authentication mechanisms and can assign a wide range of ports for the services it controls, it is important to secure **rpcbind**.

You can secure **rpcbind** by restricting access to all networks and defining specific exceptions using firewall rules on the server.

NOTE

- The **rpcbind** service is required on **NFSv2** and **NFSv3** servers.
- **NFSv4** does not require the **rpcbind** service to listen on the network.

Prerequisites

- The **rpcbind** package is installed.
- The **firewalld** package is installed and the service is running.

Procedure

1. Add firewall rules, for example:
   - Limit TCP connection and accept packages only from the **192.168.0.0/24** host via the **111** port:

     ```
     # firewall-cmd --add-rich-rule='rule family="ipv4" port port="111" protocol="tcp" source address="192.168.0.0/24" invert="True" drop'
     ```

   - Limit TCP connection and accept packages only from local host via the **111** port:

     ```
     # firewall-cmd --add-rich-rule='rule family="ipv4" port port="111" protocol="tcp" source address="127.0.0.1" accept'
     ```

   - Limit UDP connection and accept packages only from the **192.168.0.0/24** host via the **111** port:

     ```
     # firewall-cmd --permanent --add-rich-rule='rule family="ipv4" port port="111" protocol="udp" source address="192.168.0.0/24" invert="True" drop'
     ```

To make the firewall settings permanent, use the **--permanent** option when adding firewall rules.

2. Reload the firewall to apply the new rules:

```
# firewall-cmd --reload
```

Verification steps

- List the firewall rules:

```
# firewall-cmd --list-rich-rule
rule family="ipv4" port port="111" protocol="tcp" source address="192.168.0.0/24"
invert="True" drop
rule family="ipv4" port port="111" protocol="tcp" source address="127.0.0.1" accept
rule family="ipv4" port port="111" protocol="udp" source address="192.168.0.0/24"
invert="True" drop
```

Additional resources

- For more information about **NFSv4-only** servers, see Configuring an NFSv4-only server.
- Using and configuring firewalld

11.2. SECURING THE RPC.MOUNTD SERVICE

The **rpc.mountd** daemon implements the server side of the NFS mount protocol. The NFS mount protocol is used by NFS version 2 (RFC 1904) and NFS version 3 (RFC 1813).

You can secure the **rpc.mountd** service by adding firewall rules to the server. You can restrict access to all networks and define specific exceptions using firewall rules.

Prerequisites

- The **rpc.mountd** package is installed.
- The **firewalld** package is installed and the service is running.

Procedure

1. Add firewall rules to the server, for example:
   - Accept **mountd** connections from the **192.168.0.0/24** host:

```
# firewall-cmd --add-rich-rule 'rule family="ipv4" service name="mountd" source
address="192.168.0.0/24" invert="True" drop'
```

   - Accept **mountd** connections from the local host:

```
# firewall-cmd --permanent --add-rich-rule 'rule family="ipv4" source address="127.0.0.1"
service name="mountd" accept'
```

     To make the firewall settings permanent, use the **--permanent** option when adding firewall rules.

2. Reload the firewall to apply the new rules:

```
# firewall-cmd --reload
```

Verification steps

- List the firewall rules:

```
# firewall-cmd --list-rich-rule
rule family="ipv4" service name="mountd" source address="192.168.0.0/24" invert="True"
drop
rule family="ipv4" source address="127.0.0.1" service name="mountd" accept
```

Additional resources

- Using and configuring firewalld

11.3. SECURING THE NFS SERVICE

You can secure Network File System version 4 (NFSv4) by authenticating and encrypting all file system operations using Kerberos. When using NFSv4 with Network Address Translation (NAT) or a firewall, you can turn off the delegations by modifying the **/etc/default/nfs** file. Delegation is a technique by which the server delegates the management of a file to a client. In contrast, NFSv2 and NFSv3 do not use Kerberos for locking and mounting files.

The NFS service sends the traffic using TCP in all versions of NFS. The service supports Kerberos user and group authentication, as part of the **RPCSEC_GSS** kernel module.

NFS allows remote hosts to mount file systems over a network and interact with those file systems as if they are mounted locally. You can merge the resources on centralized servers and additionally customize NFS mount options in the **/etc/nfsmount.conf** file when sharing the file systems.

11.3.1. Export options for securing an NFS server

The NFS server determines a list structure of directories and hosts about which file systems to export to which hosts in the **/etc/exports** file.

WARNING

Extra spaces in the syntax of the exports file can lead to major changes in the configuration.

In the following example, the **/tmp/nfs/** directory is shared with the **bob.example.com** host and has read and write permissions.

```
/tmp/nfs/     bob.example.com(rw)
```

The following example is the same as the previous one but shares the same directory to the **bob.example.com** host with read-only permissions and shares it to the *world* with read and write permissions due to a single space character after the hostname.

```
/tmp/nfs/     bob.example.com (rw)
```

You can check the shared directories on your system by entering the **showmount -e <hostname>** command.

Use the following export options on the **/etc/exports** file:

WARNING

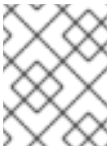
Export an entire file system because exporting a subdirectory of a file system is not secure. An attacker can possibly access the unexported part of a partially-exported file system.

ro

Use the **ro** option to export the NFS volume as read-only.

rw

Use the **rw** option to allow read and write requests on the NFS volume. Use this option cautiously because allowing write access increases the risk of attacks.



NOTE

If your scenario requires to mount the directories with the **rw** option, make sure they are not writable for all users to reduce possible risks.

root_squash

Use the **root_squash** option to map requests from **uid/gid** 0 to the anonymous **uid/gid**. This does not apply to any other **uids** or **gids** that might be equally sensitive, such as the **bin** user or the **staff** group.

no_root_squash

Use the **no_root_squash** option to turn off root squashing. By default, NFS shares change the **root** user to the **nobody** user, which is an unprivileged user account. This changes the owner of all the **root** created files to **nobody**, which prevents the uploading of programs with the **setuid** bit set. When using the **no_root_squash** option, remote root users can change any file on the shared file system and leave applications infected by trojans for other users.

secure

Use the **secure** option to restrict exports to reserved ports. By default, the server allows client communication only through reserved ports. However, it is easy for anyone to become a **root** user on a client on many networks, so it is rarely safe for the server to assume that communication through a reserved port is privileged. Therefore the restriction to reserved ports is of limited value; it is better to rely on Kerberos, firewalls, and restriction of exports to particular clients.

Additionally, consider the following best practices when exporting an NFS server:

- Exporting home directories is a risk because some applications store passwords in plain text or in a weakly encrypted format. You can reduce the risk by reviewing and improving the application code.
- Some users do not set passwords on SSH keys which again leads to risks with home directories. You can reduce these risks by enforcing the use of passwords or using Kerberos.
- Restrict the NFS exports only to required clients. Use the **showmount -e** command on the NFS server to review what the server is exporting. Do not export anything that is not specifically required.
- Do not allow unnecessary users to log in to a server to reduce the risk of attacks. You can periodically check who and what can access the server.

Additional resources

- Secure NFS with Kerberos when using Red Hat Identity Management
- **exports(5)** and **nfs(5)** man pages

11.3.2. Mount options for securing an NFS client

You can pass the following options to the **mount** command to increase the security of NFS-based clients:

nosuid

Use the **nosuid** option to disable the **set-user-identifier** or **set-group-identifier** bits. This prevents remote users from gaining higher privileges by running a **setuid** program and you can use this option opposite to **setuid** option.

noexec

Use the **noexec** option to disable all executable files on the client. Use this to prevent users from accidentally executing files placed in the shared file system.

nodev

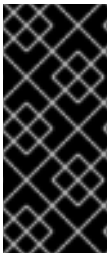
Use the **nodev** option to prevent the client's processing of device files as a hardware device.

resvport

Use the **resvport** option to restrict communication to a reserved port and you can use a privileged source port to communicate with the server. The reserved ports are reserved for privileged users and processes such as the **root** user.

sec

Use the **sec** option on the NFS server to choose the RPCGSS security flavor for accessing files on the mount point. Valid security flavors are **none**, **sys**, **krb5**, **krb5i**, and **krb5p**.



IMPORTANT

The MIT Kerberos libraries provided by the **krb5-libs** package do not support the Data Encryption Standard (DES) algorithm in new deployments. DES is deprecated and disabled by default in Kerberos libraries because of security and compatibility reasons. Use newer and more secure algorithms instead of DES, unless your environment requires DES for compatibility reasons.

Additional resources

- Common NFS mount options.

11.3.3. Securing NFS with firewall

To secure the firewall on an NFS server, keep only the required ports open. Do not use the NFS connection port numbers for any other service.

Prerequisites

- The **nfs-utils** package is installed.
- The **firewalld** package is installed and running.

Procedure

- On NFSv4, the firewall must open TCP port **2049**.
- On NFSv3, open four additional ports with **2049**:

1. **rpcbind** service assigns the NFS ports dynamically, which might cause problems when creating firewall rules. To simplify this process, use the **/etc/nfs.conf** file to specify which ports to use:

   a. Set TCP and UDP port for **mountd** (**rpc.mountd**) in the **[mountd]** section in **port=*<value>*** format.

   b. Set TCP and UDP port for **statd** (**rpc.statd**) in the **[statd]** section in **port=*<value>*** format.

2. Set the TCP and UDP port for the NFS lock manager (**nlockmgr**) in the **/etc/nfs.conf** file:

   a. Set TCP port for **nlockmgr** (**rpc.statd**) in the **[lockd]** section in **port=value** format. Alternatively, you can use the **nlm_tcpport** option in the **/etc/modprobe.d/lockd.conf** file.

   b. Set UDP port for **nlockmgr** (**rpc.statd**) in the **[lockd]** section in **udp-port=value** format. Alternatively, you can use the **nlm_udpport** option in the **/etc/modprobe.d/lockd.conf** file.

Verification steps

- List the active ports and RPC programs on the NFS server:

```
$ rpcinfo -p
```

Additional resources

- Secure NFS with Kerberos when using Red Hat Identity Management
- **exports(5)** and **nfs(5)** man pages

11.4. SECURING THE FTP SERVICE

You can use the File Transfer Protocol (FTP) to transfer files over a network. Because all FTP transactions with the server, including user authentication, are unencrypted, you should ensure it is configured securely.

RHEL 8 provides two FTP servers:

- Red Hat Content Accelerator (tux) - a kernel-space web server with FTP capabilities.
- Very Secure FTP Daemon (vsftpd) - a standalone, security-oriented implementation of the FTP service.

The following security guidelines are for setting up the **vsftpd** FTP service.

11.4.1. Securing the FTP greeting banner

When a user connects to the FTP service, FTP shows a greeting banner, which by default includes version information that could be useful for attackers to identify weaknesses in a system. You can prevent the attackers from accessing this information by changing the default banner.

You can define a custom banner by editing the **/etc/banners/ftp.msg** file to either directly include a single-line message, or to refer to a separate file, which can contain a multi-line message.

Procedure

- To define a single line message, add the following option to the **/etc/vsftpd/vsftpd.conf** file:

```
ftpd_banner=Hello, all activity on ftp.example.com is logged.
```

- To define a message in a separate file:
  - Create a **.msg** file which contains the banner message, for example **/etc/banners/*ftp*.msg**:

```
######### Hello, all activity on ftp.example.com is logged. #########
```

    To simplify the management of multiple banners, place all banners into the **/etc/banners/** directory.

  - Add the path to the banner file to the **banner_file** option in the **/etc/vsftpd/vsftpd.conf** file:

```
banner_file=/etc/banners/ftp.msg
```

Verification

- Display the modified banner:

```
$ ftp localhost
Trying ::1…
Connected to localhost (::1).
Hello, all activity on ftp.example.com is logged.
```

11.4.2. Preventing anonymous access and uploads in FTP

By default, installing the **vsftpd** package creates the /**var**/**ftp**/ directory and a directory tree for anonymous users with read-only permissions on the directories. Because anonymous users can access the data, do not store sensitive data in these directories.

To increase the security of the system, you can configure the FTP server to allow anonymous users to upload files to a specific directory and prevent anonymous users from reading data. In the following procedure, the anonymous user must be able to upload files in the directory owned by the **root** user but not change it.

Procedure

- Create a write-only directory in the /**var**/**ftp**/**pub**/ directory:

```
# mkdir /var/ftp/pub/upload
# chmod 730 /var/ftp/pub/upload
# ls -ld /var/ftp/pub/upload
drwx-wx---. 2 root ftp 4096 Nov 14 22:57 /var/ftp/pub/upload
```

- Add the following lines to the /**etc**/**vsftpd**/**vsftpd.conf** file:

```
anon_upload_enable=YES
anonymous_enable=YES
```

- Optional: If your system has SELinux enabled and enforcing, enable SELinux boolean attributes **allow_ftpd_anon_write** and **allow_ftpd_full_access**.

WARNING

Allowing anonymous users to read and write in directories might lead to the server becoming a repository for stolen software.

11.4.3. Securing user accounts for FTP

FTP transmits usernames and passwords unencrypted over insecure networks for authentication. You can improve the security of FTP by denying system users access to the server from their user accounts.

Perform as many of the following steps as applicable for your configuration.

Procedure

- Disable all user accounts in the **vsftpd** server, by adding the following line to the **/etc/vsftpd/vsftpd.conf** file:

```
local_enable=NO
```

- Disable FTP access for specific accounts or specific groups of accounts, such as the **root** user and users with **sudo** privileges, by adding the usernames to the **/etc/pam.d/vsftpd** PAM configuration file.

- Disable user accounts, by adding the usernames to the **/etc/vsftpd/ftpusers** file.

11.4.4. Additional resources

- **ftpd_selinux(8)** man page

11.5. SECURING HTTP SERVERS

11.5.1. Security enhancements in httpd.conf

You can enhance the security of the Apache HTTP server by configuring security options in the **/etc/httpd/conf/httpd.conf** file.

Always verify that all scripts running on the system work correctly before putting them into production.

Ensure that only the **root** user has write permissions to any directory containing scripts or Common Gateway Interfaces (CGI). To change the directory ownership to **root** user with write permissions, enter the following commands:

```
# chown root directory-name
# chmod 755 directory-name
```

In the **/etc/httpd/conf/httpd.conf** file, you can configure the following options:

FollowSymLinks

This directive is enabled by default and follows symbolic links in the directory.

Indexes

This directive is enabled by default. Disable this directive to prevent visitors from browsing files on the server.

UserDir

This directive is disabled by default because it can confirm the presence of a user account on the system. To activate user directory browsing for all user directories other than **/root/**, use the **UserDir enabled** and **UserDir disabled** root directives. To add users to the list of disabled accounts, add a space-delimited list of users on the **UserDir disabled** line.

ServerTokens

This directive controls the server response header field which is sent back to clients. You can use the following parameters to customize the information:

ServerTokens Full

provides all available information such as web server version number, server operating system details, installed Apache modules, for example:

```
Apache/2.4.37 (Red Hat Enterprise Linux) MyMod/1.2
```

ServerTokens Full-Release

provides all available information with release versions, for example:

```
Apache/2.4.37 (Red Hat Enterprise Linux) (Release 41.module+el8.5.0+11772+c8e0c271)
```

ServerTokens Prod / ServerTokens ProductOnly

provides the web server name, for example:

```
Apache
```

ServerTokens Major

provides the web server major release version, for example:

```
Apache/2
```

ServerTokens Minor

provides the web server minor release version, for example:

```
Apache/2.4
```

ServerTokens Min / ServerTokens Minimal

provides the web server minimal release version, for example:

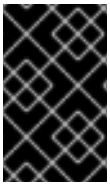
```
Apache/2.4.37
```

ServerTokens OS

provides the web server release version and operating system, for example:

```
Apache/2.4.37 (Red Hat Enterprise Linux)
```

Use the **ServerTokens Prod** option to reduce the risk of attackers gaining any valuable information about your system.



IMPORTANT

Do not remove the **IncludesNoExec** directive. By default, the Server Side Includes (SSI) module cannot execute commands. Changing this can allow an attacker to enter commands on the system.

Removing httpd modules

You can remove the **httpd** modules to limit the functionality of the HTTP server. To do so, edit configuration files in the **/etc/httpd/conf.modules.d/** or **/etc/httpd/conf.d/** directory. For example, to remove the proxy module:

```
echo '# All proxy modules disabled' > /etc/httpd/conf.modules.d/00-proxy.conf
```

Additional resources

- The Apache HTTP server
- Customizing the SELinux policy for the Apache HTTP server

11.5.2. Securing the Nginx server configuration

Nginx is a high-performance HTTP and proxy server. You can harden your Nginx configuration with the following configuration options.

Procedure

- To disable version strings, modify the **server_tokens** configuration option:

  ```
  server_tokens off;
  ```

  This option stops displaying additional details such as server version number. This configuration displays only the server name in all requests served by Nginx, for example:

  ```
  $ curl -sI http://localhost | grep Server
  Server: nginx
  ```

- Add extra security headers that mitigate certain known web application vulnerabilities in specific **/etc/nginx/** conf files:
  - For example, the **X-Frame-Options** header option denies any page outside of your domain to frame any content served by Nginx, mitigating clickjacking attacks:

    ```
    add_header X-Frame-Options "SAMEORIGIN";
    ```

  - For example, the **x-content-type** header prevents MIME-type sniffing in certain older browsers:

    ```
    add_header X-Content-Type-Options nosniff;
    ```

  - For example, the **X-XSS-Protection** header enables Cross-Site Scripting (XSS) filtering, which prevents browsers from rendering potentially malicious content included in a response by Nginx:

    ```
    add_header X-XSS-Protection "1; mode=block";
    ```

- You can limit the services exposed to the public and limit what they do and accept from the visitors, for example:

  ```
  limit_except GET {
    allow 192.168.1.0/32;
    deny  all;
  }
  ```

  The snippet will limit access to all methods except **GET** and **HEAD**.

- You can disable HTTP methods, for example:

```
# Allow GET, PUT, POST; return "405 Method Not Allowed" for all others.
if ( $request_method !~ ^(GET|PUT|POST)$ ) {
    return 405;
}
```

- You can configure SSL to protect the data served by your Nginx web server, consider serving it over HTTPS only. Furthermore, you can generate a secure configuration profile for enabling SSL in your Nginx server using the Mozilla SSL Configuration Generator. The generated configuration ensures that known vulnerable protocols (for example, SSLv2 and SSLv3), ciphers, and hashing algorithms (for example, 3DES and MD5) are disabled. You can also use the SSL Server Test to verify that your configuration meets modern security requirements.

Additional resources

- Mozilla SSL Configuration Generator
- SSL Server Test

11.6. SECURING POSTGRESQL BY LIMITING ACCESS TO AUTHENTICATED LOCAL USERS

PostgreSQL is an object-relational database management system (DBMS). In Red Hat Enterprise Linux, PostgreSQL is provided by the **postgresql-server** package.

You can reduce the risks of attacks by configuring client authentication. The **pg_hba.conf** configuration file stored in the database cluster's data directory controls the client authentication. Follow the procedure to configure PostgreSQL for host-based authentication.

Procedure

1. Install PostgreSQL:

```
# yum install postgresql-server
```

2. Initialize a database storage area using one of the following options:

   a. Using the **initdb** utility:

```
$ initdb -D /home/postgresql/db1/
```

   The **initdb** command with the **-D** option creates the directory you specify if it does not already exist, for example */**home**/**postgresql**/**db1**/*. This directory then contains all the data stored in the database and also the client authentication configuration file.

   b. Using the **postgresql-setup** script:

```
$ postgresql-setup --initdb
```

   By default, the script uses the /**var**/**lib**/**pgsql**/**data**/ directory. This script helps system administrators with basic database cluster administration.

3. To allow any authenticated local users to access any database with their usernames, modify the following line in the **pg_hba.conf** file:

```
local   all             all                                     trust
```

This can be problematic when you use layered applications that create database users and no local users. If you do not want to explicitly control all user names on the system, remove the **local** line entry from the **pg_hba.conf** file.

4. Restart the database to apply the changes:

```
# systemctl restart postgresql
```

The previous command updates the database and also verifies the syntax of the configuration file.

11.7. SECURING THE MEMCACHED SERVICE

Memcached is an open source, high-performance, distributed memory object caching system. It can improve the performance of dynamic web applications by lowering database load.

Memcached is an in-memory key-value store for small chunks of arbitrary data, such as strings and objects, from results of database calls, API calls, or page rendering. Memcached allows assigning memory from underutilized areas to applications that require more memory.

In 2018, vulnerabilities of DDoS amplification attacks by exploiting Memcached servers exposed to the public internet were discovered. These attacks took advantage of Memcached communication using the UDP protocol for transport. The attack was effective because of the high amplification ratio where a request with the size of a few hundred bytes could generate a response of a few megabytes or even hundreds of megabytes in size.

In most situations, the **memcached** service does not need to be exposed to the public Internet. Such exposure may have its own security problems, allowing remote attackers to leak or modify information stored in Memcached.

Follow the section to harden the system using Memcached service against possible DDoS attacks.

11.7.1. Hardening Memcached against DDoS

To mitigate security risks, perform as many of the following steps as applicable for your configuration.

Procedure

- Configure a firewall in your LAN. If your Memcached server should be accessible only in your local network, do not route external traffic to ports used by the **memcached** service. For example, remove the default port **11211** from the list of allowed ports:

```
# firewall-cmd --remove-port=11211/udp
# firewall-cmd --runtime-to-permanent
```

- If you use a single Memcached server on the same machine as your application, set up **memcached** to listen to localhost traffic only. Modify the **OPTIONS** value in the **/etc/sysconfig/memcached** file:

```
OPTIONS="-l 127.0.0.1,::1"
```

- Enable Simple Authentication and Security Layer (SASL) authentication:

  1. Modify or add the **/etc/sasl2/memcached.conf** file:

     ```
     sasldb_path: /path.to/memcached.sasldb
     ```

  2. Add an account in the SASL database:

     ```
     # saslpasswd2 -a memcached -c cacheuser -f /path.to/memcached.sasldb
     ```

  3. Ensure that the database is accessible for the **memcached** user and group:

     ```
     # chown memcached:memcached /path.to/memcached.sasldb
     ```

  4. Enable SASL support in Memcached by adding the **-S** value to the **OPTIONS** parameter in the **/etc/sysconfig/memcached** file:

     ```
     OPTIONS="-S"
     ```

  5. Restart the Memcached server to apply the changes:

     ```
     # systemctl restart memcached
     ```

  6. Add the username and password created in the SASL database to the Memcached client configuration of your application.

- Encrypt communication between Memcached clients and servers with TLS:

  1. Enable encrypted communication between Memcached clients and servers with TLS by adding the **-Z** value to the **OPTIONS** parameter in the **/etc/sysconfig/memcached** file:

     ```
     OPTIONS="-Z"
     ```

  2. Add the certificate chain file path in the PEM format using the **-o ssl_chain_cert** option.

  3. Add a private key file path using the **-o ssl_key** option.