



Red Hat Enterprise Linux 10

Securing networks

Configuring secured networks and network communication

Red Hat Enterprise Linux 10 Securing networks

Configuring secured networks and network communication

Legal Notice

Copyright © Red Hat.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

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

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	5
CHAPTER 1. USING SECURE COMMUNICATIONS BETWEEN TWO SYSTEMS WITH OPENSSH	6
1.1. SSH AND OPENSSH	6
1.2. GENERATING SSH KEY PAIRS	7
1.3. SETTING KEY-BASED AUTHENTICATION AS THE ONLY METHOD ON AN OPENSSH SERVER	9
1.4. AUTHENTICATING BY SSH KEYS STORED ON A SMART CARD	9
1.5. MAKING OPENSSH MORE SECURE	11
1.6. CONNECTING TO A REMOTE SERVER THROUGH AN SSH JUMP HOST	14
1.7. CONFIGURING THE OPENSSH SERVER AND CLIENT BY USING RHEL SYSTEM ROLES	15
1.7.1. How the sshd RHEL system role maps settings from a playbook to the configuration file	16
1.7.2. Configuring OpenSSH servers by using the sshd RHEL system role	16
1.7.3. Using the sshd RHEL system role for non-exclusive configuration	18
1.7.4. Overriding the system-wide cryptographic policy on an SSH server by using the sshd RHEL system role	20
1.7.5. How the ssh RHEL system role maps settings from a playbook to the configuration file	23
1.7.6. Configuring OpenSSH clients by using the ssh RHEL system role	23
1.8. ADDITIONAL RESOURCES	25
CHAPTER 2. CREATING AND MANAGING TLS KEYS AND CERTIFICATES	26
2.1. TLS CERTIFICATES	26
2.2. POST-QUANTUM CRYPTOGRAPHY ALGORITHMS IN OPENSSL	26
2.3. CREATING A PRIVATE CA BY USING OPENSSL	28
2.4. CREATING A PRIVATE KEY AND A CSR FOR A TLS SERVER CERTIFICATE BY USING OPENSSL	30
2.5. CREATING A PRIVATE KEY AND A CSR FOR A TLS CLIENT CERTIFICATE BY USING OPENSSL	31
2.6. USING A PRIVATE CA TO ISSUE CERTIFICATES FOR CSRS WITH OPENSSL	33
2.7. CREATING A PRIVATE CA BY USING GNUTLS	34
2.8. CREATING A PRIVATE KEY AND A CSR FOR A TLS SERVER CERTIFICATE BY USING GNUTLS	36
2.9. CREATING A PRIVATE KEY AND A CSR FOR A TLS CLIENT CERTIFICATE BY USING GNUTLS	38
2.10. USING A PRIVATE CA TO ISSUE CERTIFICATES FOR CSRS WITH GNUTLS	39
CHAPTER 3. USING SHARED SYSTEM CERTIFICATES	40
3.1. THE SYSTEM-WIDE TRUSTSTORE	40
3.2. ADDING NEW CERTIFICATES TO THE SYSTEM-WIDE TRUSTSTORE	40
3.3. TRUSTED SYSTEM CERTIFICATES MANAGEMENT WITH THE TRUST COMMAND	41
CHAPTER 4. PLANNING AND IMPLEMENTING TLS	43
4.1. SSL AND TLS PROTOCOLS	43
4.2. SECURITY CONSIDERATIONS FOR TLS IN RHEL 10	44
4.2.1. Protocols	44
4.2.2. Cipher suites	45
4.2.3. Public key length	45
4.3. TLS CONFIGURATION HARDENING IN APPLICATIONS	46
4.3.1. TLS configuration of an Apache HTTP server	46
4.3.2. TLS configuration of an Nginx HTTP and proxy server	47
4.3.3. TLS configuration of a Dovecot mail server	47
CHAPTER 5. SECURING SYSTEM DNS TRAFFIC WITH ENCRYPTED DNS (EDNS)	49
5.1. OVERVIEW OF COMPONENTS FOR EDNS IN RHEL	49
5.1.1. eDNS resolution process and core interactions	49
5.2. INSTALLING RHEL WITH EDNS ENABLED FROM A LOCAL INSTALLATION MEDIA	50
5.3. INSTALLING RHEL WITH EDNS ENABLED USING A CUSTOM BOOTABLE ISO	52
5.4. ENABLING EDNS ON AN EXISTING RHEL INSTALLATION	54

5.5. KERNEL PARAMETERS FOR DNS CONFIGURATION	57
5.6. ADDITIONAL RESOURCES	57
CHAPTER 6. SETTING UP AN IPSEC VPN	58
6.1. COMPONENTS IN AN IPSEC VPN	58
6.2. LIBRESWAN AUTHENTICATION METHODS	58
6.3. MANUALLY CONFIGURING AN IPSEC HOST-TO-HOST VPN WITH RAW RSA KEY AUTHENTICATION	59
6.4. MANUALLY CONFIGURING AN IPSEC SITE-TO-SITE VPN WITH RAW RSA KEY AUTHENTICATION	62
6.5. MANUALLY CONFIGURING AN IPSEC HOST-TO-SITE VPN WITH CERTIFICATE-BASED AUTHENTICATION	65
6.5.1. Setting up an IPsec gateway manually	65
6.5.2. Configuring a client to connect to an IPsec VPN gateway by using GNOME Settings	69
6.6. MANUALLY CONFIGURING AN IPSEC MESH VPN WITH CERTIFICATE-BASED AUTHENTICATION	72
6.7. PROTECTING THE IPSEC NSS DATABASE WITH A PASSWORD	76
6.8. USING IPSEC ON A SYSTEM WITH FIPS MODE ENABLED	77
6.9. CONFIGURING TCP FALLBACK FOR AN IPSEC VPN CONNECTION	77
6.10. ENABLING LEGACY CIPHERS AND ALGORITHMS IN LIBRESWAN	79
6.11. ASSIGNING A VPN CONNECTION TO A DEDICATED ROUTING TABLE TO PREVENT THE CONNECTION FROM BYPASSING THE TUNNEL	79
6.12. CONFIGURING IPSEC VPN CONNECTIONS BY USING RHEL SYSTEM ROLES	81
6.12.1. Configuring an IPsec host-to-host VPN with PSK authentication by using the vpn RHEL system role	81
6.12.2. Configuring an IPsec host-to-host VPN with PSK authentication and separate data and control planes by using the vpn RHEL system role	83
6.12.3. Configuring an IPsec site-to-site VPN with PSK authentication by using the vpn RHEL system role	85
6.12.4. Configuring an IPsec mesh VPN with certificate-based authentication by using the vpn RHEL system role	87
6.13. CONFIGURING IPSEC VPN CONNECTIONS BY USING NMSTATECTL	90
6.13.1. Configuring an IPsec host-to-host VPN with raw RSA key authentication by using nmstatectl	91
6.13.2. Configuring an IPsec site-to-site VPN with raw RSA key authentication by using nmstatectl	94
6.13.3. Configuring a client to connect to an IPsec VPN gateway by using nmstatectl	97
6.14. TROUBLESHOOTING IPSEC CONFIGURATIONS	100
6.14.1. Basic connection issues	100
6.14.2. Firewall-related problems	100
6.14.3. Mismatched Configurations	100
6.14.4. MTU issues	102
6.14.5. NAT conflicts	102
6.14.6. Kernel-level IPsec issues	103
6.14.7. Kernel IPsec subsystem bugs	103
6.14.8. Displaying Libreswan logs	104
CHAPTER 7. USING MACSEC TO ENCRYPT LAYER-2 TRAFFIC IN THE SAME PHYSICAL NETWORK	105
7.1. HOW MACSEC INCREASES SECURITY	105
7.2. CONFIGURING A MACSEC CONNECTION BY USING NMCLI	106
7.3. CONFIGURING A MACSEC CONNECTION BY USING NMSTATECTL	107
CHAPTER 8. SECURING NETWORK SERVICES	110
8.1. SECURING THE RPCBIND SERVICE	110
8.2. SECURING THE RPC.MOUNTD SERVICE	111
8.3. SECURING THE NFS SERVICE	112
8.3.1. Export options for securing an NFS server	112
8.3.2. Mount options for securing an NFS client	114
8.3.3. Securing NFS with firewall	115
8.4. SECURING THE FTP SERVICE	116
8.4.1. Securing the FTP greeting banner	116

8.4.2. Preventing anonymous access and uploads in FTP	117
8.4.3. Securing user accounts for FTP	117
8.5. SECURING HTTP SERVERS	118
8.5.1. Security enhancements in httpd.conf	118
8.5.1.1. Removing httpd modules	119
8.5.2. Nginx server configuration hardening	119
8.6. SECURING POSTGRESQL BY LIMITING ACCESS TO AUTHENTICATED LOCAL USERS	121
8.7. SECURING THE MEMCACHED SERVICE	122
8.7.1. Memcached hardening against DDoS attacks	122
8.8. SECURING THE POSTFIX SERVICE	123
8.8.1. Reducing Postfix network-related security risks	123
8.8.2. Postfix configuration options for limiting DoS attacks	124
8.8.3. Configuring Postfix to use SASL	125

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

Learn how to use OpenSSH to establish secure, encrypted communication channels between two systems. This approach helps protect remote login sessions from eavesdropping and intrusions.

1.1. SSH AND OPENSSH

SSH (Secure Shell) is a program for remote machine login and command execution. The SSH protocol provides encrypted communication 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 started for the first time 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.

For more information, refer to man pages listed by using the **man -k ssh** command on your system.



NOTE

In RHEL 9 and later, the Secure copy protocol (SCP) is replaced with the SSH File Transfer Protocol (SFTP) by default. This is because SCP has already caused security issues, for example [CVE-2020-15778](#).

If SFTP is unavailable or incompatible in your scenario, you can use the **scp** command with the **-O** option to force the use of the original SCP/RCP protocol.

For additional information, see the [OpenSSH SCP protocol deprecation in Red Hat Enterprise Linux 9](#) article.

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 the older version 1.

Red Hat Enterprise Linux includes the **openssh**, **openssh-server**, and **openssh-clients** packages. These OpenSSH packages require the OpenSSL package **openssl-lib**, which contains the cryptographic libraries necessary to secure data exchange.

OpenSSH, as one of core cryptographic subsystems of RHEL, uses system-wide cryptographic 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 cryptographic policies. See the [Excluding an application from following system-wide cryptographic policies](#) section for more information.

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. The **/etc/ssh/ssh_config** file contains the client configuration, and the **/etc/ssh/sshd_config** file is the default OpenSSH server configuration file.

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 on your system.

Additional resources

- [Using system-wide cryptographic policies](#)

1.2. GENERATING SSH KEY PAIRS

You can log in to an OpenSSH server without entering a password by generating an SSH key pair on a local system and copying the generated public key to the OpenSSH server. Each user who wants to create a key must run this procedure.

To preserve previously generated key pairs after you reinstall the system, back up the **~/.ssh/** directory before you create new keys. After reinstalling, copy it back to your home directory. You can do this for all users on your system, including **root**.

Prerequisites

- You are logged in as a user who wants to connect to the OpenSSH server by using keys.

- The OpenSSH server is configured to allow key-based authentication.

Procedure

1. Generate an ECDSA key pair:

```
$ ssh-keygen -t ecdsa
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/<username>/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase): <password>
Enter same passphrase again: <password>
Your identification has been saved in /home/<username>/.ssh/id_ecdsa.
Your public key has been saved in /home/<username>/.ssh/id_ecdsa.pub.
The key fingerprint is:
SHA256:Q/x+qms4j7PCQ0qFd09iZEFHA+SqwBKRNuU72oZfaCI
<username>@<localhost.example.com>
The key's randomart image is:
+---[ECDSA 256]---+
|.00..0=++      |
|.. 0 .00 .     |
|. .. 0. 0      |
|....0.+...     |
|0.00.0 +S .    |
|.=.+ .0       |
|E.*+ . . .    |
|.=..+ +.. 0   |
| . 00*+0.     |
+----[SHA256]-----+
```

You can also generate an RSA key pair by using the **ssh-keygen** command without any parameter or an Ed25519 key pair by entering the **ssh-keygen -t ed25519** command. Note that the Ed25519 algorithm is not FIPS-140-compliant, and OpenSSH does not work with Ed25519 keys in FIPS mode.

2. Copy the public key to a remote machine:

```
$ ssh-copy-id <username>@<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
<username>@<ssh-server-example.com>'s password:
...
Number of key(s) added: 1

Now try logging into the machine, with: "ssh '<username>@<ssh-server-example.com>'" and
check to make sure that only the key(s) you wanted were added.
```

Replace **<username>@<ssh-server-example.com>** with your credentials.

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.

Verification

1. Log in to the OpenSSH server by using the key file:

```
$ ssh -o PreferredAuthentications=publickey <username>@<ssh-server-example.com>
```

1.3. SETTING KEY-BASED AUTHENTICATION AS THE ONLY METHOD ON AN OPENSSH SERVER

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.
- You can already connect to the OpenSSH server by using a key. See the [Generating SSH key pairs](#) section for details.

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
```

3. On a system other than a new default installation, check that the **PubkeyAuthentication** parameter is either not set or set to **yes**.
4. Set the **KbdInteractiveAuthentication** directive to **no**.
Note that the corresponding entry is commented out in the configuration file and the default value is **yes**.
5. 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
```

6. If you are connected remotely, not using console or out-of-band access, test the key-based login process before disabling password authentication.
7. Reload the **sshd** daemon to apply the changes:

```
# systemctl reload sshd
```

1.4. AUTHENTICATING BY SSH KEYS STORED ON A SMART CARD

Use SSH keys stored on a smart card for authentication to add a physical layer of protection to your credentials. This method provides enhanced security against unauthorized access.

You can create and store ECDSA and RSA keys on a smart card and authenticate by the smart card on an OpenSSH client. Smart-card authentication replaces the default password authentication.

Prerequisites

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

Procedure

- 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
```

- Transfer the public key to the remote server. Use the **ssh-copy-id** command with the **keys.pub** file created in the previous step:

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

- Connect to **<ssh-server-example.com>** by using the ECDSA key. 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" <ssh-server-example.com>
Enter PIN for 'SSH key':
[ssh-server-example.com] $
```

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

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

- Optional: 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 <ssh-server-example.com>
Enter PIN for 'SSH key':
[ssh-server-example.com] $
```

The **ssh** client utility now automatically uses this URI and the key from the smart card.

1.5. MAKING OPENSSH MORE SECURE

Harden your system security by modifying OpenSSH configurations, such as implementing stronger keys and restricting user access. This helps protect your server against various attacks.

Note that changes in the **/etc/ssh/sshd_config** OpenSSH server configuration file require reloading the **sshd** daemon to take effect:

```
# systemctl reload sshd
```



WARNING

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.

Disabling password-based authentication

Disabling passwords for authentication and allowing only key pairs reduces the attack surface. See the [Setting key-based authentication as the only method on an OpenSSH server](#) section for more information.

Stronger key types

Although the **ssh-keygen** command generates a pair of RSA keys by default, you can instruct it to generate Elliptic Curve Digital Signature Algorithm (ECDSA) or Edwards-Curve 25519 (Ed25519) keys by using the **-t** option. The ECDSA 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
# rm -f /etc/ssh/ssh_host_rsa_key*
# systemctl restart sshd
```



NOTE

In images with the **cloud-init** method 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/ssh-keygen@.service.d/disable-ssh-keygen-if-cloud-init-active.conf** drop-in configuration file disables the **ssh-keygen** units if **cloud-init** is running.

To allow only a particular key type for SSH connections, remove a comment out at the beginning of the relevant line in **/etc/ssh/sshd_config**, and reload the **sshd** service. For example, to allow only Ed25519 host keys, the corresponding lines must be as follows:

```
# 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 on the default port and therefore increases security through obscurity. You can specify the port by 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.

Root login

PermitRootLogin is set to **prohibit-password** by default. 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.

**WARNING**

Enabling logging in as the root user is not a secure practice because the administrator cannot audit which users run which privileged commands. For using administrative commands, log in and use **sudo** instead.

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/50-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 SSH access to specific users, groups, or IP ranges

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
```

This configuration allows only connections if all of the following conditions meet:

- The connection's source IP is within the 192.168.1.0/24 or 10.0.0.0/24 subnet.
- The source IP is not 192.168.1.2.
- The user is a member of the example-group group.

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 with legacy systems, 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.

Opting out of system-wide cryptographic policies

To opt out of the system-wide cryptographic policies for your OpenSSH server, specify the cryptographic policy in a drop-in configuration file located in the `/etc/ssh/sshd_config.d/` directory, with a two-digit number prefix smaller than 50, so that it lexicographically precedes the **50-redhat.conf** file, and with a **.conf** suffix, for example, **49-crypto-policy-override.conf**.

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 50, so that it lexicographically precedes the **50-redhat.conf** file, and with a **.conf** suffix, for example, **49-crypto-policy-override.conf**.

Additional resources

- [Using system-wide cryptographic policies](#)
- [How to disable specific algorithms and ciphers for ssh service only \(Red Hat Knowledgebase\)](#)

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

Connect securely from your local system to a remote server by using a jump host as an intermediary. This approach manages connections between hosts located in different security zones.

Prerequisites

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

Procedure

1. If you connect through a jump server or more intermediary servers once, use the **ssh -J** command and specify the jump servers directly, for example:

```
$ ssh -J <jump-1.example.com>,<jump-2.example.com>,<jump-3.example.com> <target-server-1.example.com>
```

Change the hostname-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 <example.user.1>@<jump-1.example.com>:<75>,<example.user.2>@<jump-2.example.com>:<75>,<example.user.3>@<jump-3.example.com>:<75>
<example.user.f>@<target-server-1.example.com>:<220>
```

2. If you connect to a remote server through jump servers regularly, store the jump-server configuration in your SSH configuration file:
 - a. Define the jump host by editing the **~/.ssh/config** file on your local system, for example:

```
Host <jump-server-1>
  HostName <jump-1.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 hostname, but can also be any string.
 - The **HostName** parameter sets the actual hostname or IP address of the jump host.
- b. Add the remote server jump configuration with the **ProxyJump** directive to **~/.ssh/config** file on your local system, for example:

```
Host <remote-server-1>
  HostName <target-server-1.example.com>
  ProxyJump <jump-server-1>
```

- c. Use your local system to connect to the remote server through the jump server:

```
$ ssh <remote-server-1>
```

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

1.7. CONFIGURING THE OPENSSSH SERVER AND CLIENT BY USING RHEL SYSTEM ROLES

You can use the **sshd** RHEL system role to configure OpenSSH servers and the **ssh** RHEL system role to configure OpenSSH clients consistently, in an automated fashion, and on any number of RHEL systems at the same time.

Such configurations are necessary for any system where secure remote interaction is needed, for example:

- Remote system administration: securely connecting to your machine from another computer by using an SSH client.

- Secure file transfers: the Secure File Transfer Protocol (SFTP) provided by OpenSSH enables you to securely transfer files between your local machine and a remote system.
- Automated DevOps pipelines: automating software deployments that require secure connection to remote servers (CI/CD pipelines).
- Tunneling and port forwarding: forwarding a local port to access a web service on a remote server behind a firewall. For example a remote database or a development server.
- Key-based authentication: more secure alternative to password-based logins.
- Certificate-based authentication: centralized trust management and better scalability.
- Enhanced security: disabling root logins, restricting user access, enforcing strong encryption and other such forms of hardening ensures stronger system security.

1.7.1. How the **sshd** RHEL system role maps settings from a playbook to the configuration file

In the **sshd** RHEL system role playbook, you can define the parameters for the server SSH configuration file. If you do not specify these settings, the role produces the **sshd_config** file that matches the RHEL defaults.

In all cases, booleans correctly render as **yes** and **no** in the final configuration on your managed nodes. You can use lists to define multi-line configuration items. For example:

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

renders as:

```
ListenAddress 0.0.0.0  
ListenAddress ::
```

1.7.2. Configuring OpenSSH servers by using the **sshd** RHEL system role

You can use the **sshd** RHEL system role to configure multiple OpenSSH servers for secure remote access.

The role ensures secure communication environment for remote users by providing namely:

- Management of incoming SSH connections from remote clients
- Credentials verification
- Secure data transfer and command execution



NOTE

You can use the **sshd** RHEL system role alongside with other RHEL system roles that change SSHD configuration, for example the Identity Management in Red Hat Enterprise Linux RHEL system roles. To prevent the configuration from being overwritten, ensure the **sshd** RHEL system role uses namespaces (RHEL 8 and earlier versions) or a drop-in directory (RHEL 9 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 for these nodes.

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: redhat.rhel_system_roles.sshd
      vars:
        sshd_config:
          PermitRootLogin: no
          PasswordAuthentication: no
        Match:
          - Condition: "Address 192.0.2.0/24"
            PermitRootLogin: yes
            PasswordAuthentication: yes
```

The settings specified in the example playbook include the following:

PasswordAuthentication: yes|no

Controls whether the OpenSSH server (**sshd**) accepts authentication from clients that use the username and password combination.

Match:

The match block allows the **root** user to login by using a password only from the subnet **192.0.2.0/24**.

For details about the role variables and the OpenSSH configuration options used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** file and the **sshd_config(5)** manual page on the control node.

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.d/00-ansible_system_role.conf
#
# Ansible managed
#
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>
```

1.7.3. Using the sshd RHEL system role for non-exclusive configuration

By default, applying the **sshd** RHEL system role overwrites the entire configuration. This may be problematic if you have previously adjusted the configuration with a different playbook. You can use the non-exclusive configuration to apply changes only to selected configuration options.

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 for these nodes.

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 environment variables
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.sshd
      vars:
        sshd_config_namespace: <my_application>
        sshd_config:
          # 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 environment variables
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.sshd
      vars:
        sshd_config_file: /etc/ssh/sshd_config.d/<42-my_application>.conf
        sshd_config:
          # Environment variables to accept
          AcceptEnv:
            LANG
            LS_COLORS
            EDITOR
```

The settings specified in the example playbooks include the following:

sshd_config_namespace: <my_application>

The role places the configuration that you specify in the playbook to configuration snippets in the existing configuration file under the given namespace. You need to select a different namespace when running the role from a different context.

sshd_config_file: /etc/ssh/sshd_config.d/<42-my_application>.conf

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.

AcceptEnv:

Controls which environment variables the OpenSSH server (**sshd**) will accept from a client:

- **LANG**: defines the language and locale settings.
- **LS_COLORS**: defines the displaying color scheme for the **ls** command in the terminal.
- **EDITOR**: specifies the default text editor for the command-line programs that need to open an editor.

For details about the role variables and the OpenSSH configuration options used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** file and the **sshd_config(5)** manual page on the control node.

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

```
# 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>
```

- For managed nodes that run RHEL 9 or later:

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

1.7.4. Overriding the system-wide cryptographic policy on an SSH server by using the sshd RHEL system role

When the default cryptographic settings do not meet certain security or compatibility needs, you may want to override the system-wide cryptographic policy on the OpenSSH server by using the **sshd** RHEL system role.

Override the system-wide cryptographic policy in the following notable situations:

- Compatibility with older clients: necessity to use weaker-than-default encryption algorithms, key exchange protocols, or ciphers.
- Enforcing stronger security policies: simultaneously, you can disable weaker algorithms. Such a measure could exceed the default system cryptographic policies, especially in the highly secure and regulated environments.
- Performance considerations: the system defaults could enforce stronger algorithms that can be computationally intensive for some systems.
- Customizing for specific security needs: adapting for unique requirements that are not covered by the default cryptographic policies.



WARNING

It is not possible to override all aspects of the cryptographic policies from the **sshd** RHEL system role. For example, SHA-1 signatures might be forbidden on a different layer so for a more generic solution, see [Setting a custom cryptographic policy by using RHEL system roles](#).

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 for these nodes.

Procedure

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

```
- name: Deploy SSH configuration for OpenSSH server
  hosts: managed-node-01.example.com
  tasks:
    - name: Overriding the system-wide cryptographic policy
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.sshd
      vars:
        sshd_sysconfig: true
        sshd_sysconfig_override_crypto_policy: true
        sshd_KexAlgorithms: ecdh-sha2-nistp521
        sshd_Ciphers: aes256-ctr
        sshd_MACs: hmac-sha2-512-etm@openssh.com
        sshd_HostKeyAlgorithms: rsa-sha2-512,rsa-sha2-256
```

The settings specified in the example playbook include the following:

sshd_KexAlgorithms

You can choose key exchange algorithms, for example, **ecdh-sha2-nistp256**, **ecdh-sha2-nistp384**, **ecdh-sha2-nistp521**, **diffie-hellman-group14-sha1**, or **diffie-hellman-group-exchange-sha256**.

sshd_Ciphers

You can choose ciphers, for example, **aes128-ctr**, **aes192-ctr**, or **aes256-ctr**.

sshd_MACs

You can choose MACs, for example, **hmac-sha2-256**, **hmac-sha2-512**, or **hmac-sha1**.

sshd_HostKeyAlgorithms

You can choose a public key algorithm, for example, **ecdsa-sha2-nistp256**, **ecdsa-sha2-nistp384**, **ecdsa-sha2-nistp521**, or **ssh-rsa**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** file on the control node.



NOTE

On RHEL 9 managed nodes, the system role writes the configuration into the **/etc/ssh/sshd_config.d/00-ansible_system_role.conf** file, where cryptographic options are applied automatically. You can change the file by using the **sshd_config_file** variable. However, to ensure the configuration is effective, use a file name that lexicographically precedes the **/etc/ssh/sshd_config.d/50-redhat.conf** file, which includes the configured crypto policies.

On RHEL 8 managed nodes, you must enable override by setting the **sshd_sysconfig_override_crypto_policy** and **sshd_sysconfig** variables to **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

- You can verify the success of the procedure by using the verbose SSH connection and check the defined variables in the following output:

```
$ ssh -vvv <ssh_server>
...
debug2: peer server KEXINIT proposal
debug2: KEX algorithms: ecdh-sha2-nistp521
debug2: host key algorithms: rsa-sha2-512,rsa-sha2-256
```

```
debug2: ciphers ctos: aes256-ctr
debug2: ciphers stoc: aes256-ctr
debug2: MACs ctos: hmac-sha2-512-etm@openssh.com
debug2: MACs stoc: hmac-sha2-512-etm@openssh.com
...
```

1.7.5. How the **ssh** RHEL system role maps settings from a playbook to the configuration file

In the **ssh** RHEL system role playbook, you can define the parameters for the client SSH configuration file. If you do not specify these settings, the role produces a global **ssh_config** file that matches the RHEL defaults.

In all the cases, booleans correctly render as **yes** or **no** in the final configuration on your managed nodes. You can use lists to define multi-line configuration items. 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.

1.7.6. Configuring OpenSSH clients by using the **ssh** RHEL system role

You can use the **ssh** RHEL system role to configure multiple OpenSSH clients.

OpenSSH clients enable the local user to establish a secure connection with the remote OpenSSH server by ensuring namely:

- Secure connection initiation
- Credentials provision
- Negotiation with the OpenSSH server on the encryption method used for the secure communication channel
- Ability to send files securely to and from the OpenSSH server



NOTE

You can use the **ssh** RHEL system role alongside with other system roles that change SSH configuration, for example the Identity Management in Red Hat Enterprise RHEL system roles. To prevent the configuration from being overwritten, make sure that the **ssh** RHEL system 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 for these nodes.

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: redhat.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_FowardX11: no
```

The settings specified in the example playbook include the following:

ssh_user: root

Configures the **root** user's SSH client preferences on the managed nodes with certain configuration specifics.

Compression: true

Compression is enabled.

ControlMaster: auto

ControlMaster multiplexing is set to **auto**.

Host

Creates alias **example** for connecting to the **server.example.com** host as a user called **user1**.

ssh_FowardX11: no

X11 forwarding is disabled.

For details about the role variables and the OpenSSH configuration options used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.ssh/README.md** file and the **ssh_config(5)** manual page on the control node.

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
```

1.8. ADDITIONAL RESOURCES

- [Configuring SELinux for applications and services with non-standard configurations](#)
- [Controlling network traffic using firewalld](#)

CHAPTER 2. CREATING AND MANAGING TLS KEYS AND CERTIFICATES

Learn how to create and manage TLS private keys and certificates by using toolkits such as OpenSSL and GnuTLS. Properly configuring these assets is critical for secure communication.

2.1. TLS CERTIFICATES

TLS (Transport Layer Security) is a protocol that establishes encrypted data exchange between client/server applications. 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 contains two main toolkits for working with TLS certificates: GnuTLS and OpenSSL. You can create, read, sign, and verify certificates by using the **openssl** utility from the **openssl** package. The **certtool** utility, included in the **gnutls-utils** package, performs the same operations with a different syntax and a distinct set of back-end libraries. See the **openssl(1)**, **x509(1)**, **ca(1)**, **req(1)**, and **certtool(1)** man pages on your system for more information.

Additional resources

- [RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List \(CRL\) Profile](#)

2.2. POST-QUANTUM CRYPTOGRAPHY ALGORITHMS IN OPENSSL

You can use the OpenSSL TLS toolkit to generate keys and certificates with post-quantum algorithms. This helps enhance security against emerging threats while maintaining compatibility with traditional algorithms.

Starting with RHEL 10.1, you can use OpenSSL for generating keys, signing messages, verifying signatures, and creating X.509 certificates with the ML-DSA post-quantum algorithms.

From OpenSSL 3.5, the hybrid ML-KEM (Module-Lattice-Based Key-Encapsulation Mechanism)

method is preferred in TLS 1.3 handshakes. OpenSSL includes keys with both traditional algorithms and ML-KEM. The use of ML-KEM results in a slight delay in the initiation of TLS connections. Still, it does not affect performance after the handshake, as further communication uses a more efficient symmetric key.

Example 2.1. Usage of ML-DSA for keys in OpenSSL

```
$ openssl genpkey -algorithm mldsa65 -out <mldsa-privatekey.pem>
```

Create a private key with the ML-DSA-65 algorithm.

```
$ openssl pkey -in <mldsa-privatekey.pem> -pubout -out <mldsa-publickey.pem>
```

Create a public key based on the ML-DSA-65-encrypted private key.

```
$ openssl dgst -sign <mldsa-privatekey.pem> -out <signature_message>
```

Sign a message with the private key.

```
$ openssl dgst -verify <mldsa-publickey.pem> -signature <signature_message>
```

Verify the ML-DSA-65 signature with the public key.

Example 2.2. Usage of ML-DSA for certificates in OpenSSL

Because no public certificate authorities (CA) currently support post-quantum signatures, you can use only a local CA or self-signed certificates with ML-DSA signatures. For example:

```
$ openssl req \  
  -x509 \  
  -newkey mldsa65 \  
  -keyout <localhost-mldsa.key> \  
  -subj /CN=<localhost> \  
  -addext subjectAltName=DNS:<localhost> \  
  -days <30> \  
  -nodes \  
  -out <localhost-mldsa.crt>
```

Example 2.3. Establishing a connection with PQC key exchange and PQC certificates

An OpenSSL server and client can establish a post-quantum connection and a connection that uses only traditional algorithms.

```
$ openssl s_server \  
  -cert <localhost-mldsa.crt> -key <localhost-mldsa.key> \  
  -dcert <localhost-rsa.crt> -dkey <localhost-rsa.key> >/dev/null &
```

```
$ openssl s_client \  
  -connect <localhost:4433> \  
  -CAfile <localhost-mldsa.crt> </dev/null \  
  |& grep -E '(Peer signature type|Negotiated TLS1.3 group)'  
Peer signature type: mldsa65  
Negotiated TLS1.3 group: X25519MLKEM768
```

Example 2.4. Establishing a connection that uses only non-post-quantum cryptographic algorithms

```
$ openssl s_client \
  -connect <localhost:4433> \
  -CAfile <localhost-rsa.crt> \
  -sigalgs 'rsa_pss_pss_sha256:rsa_pss_rsae_sha256' \
  -groups 'X25519:secp256r1:X448:secp521r1:secp384r1' </dev/null \
  |& grep -E '(Peer signature type|Server Temp Key)'
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
```

You can configure a server to simultaneously use traditional certificates (RSA, ECDSA, and EdDSA) and post-quantum certificates. The server automatically and transparently selects the certificates preferred and supported by clients: the post-quantum for new clients and traditional for legacy ones.

See the **openssl(1)**, **openssl-genpkey(1)**, **openssl-pkey(1)**, **openssl-dgst(1)**, and **openssl-verify(1)** man pages on your system for more information.

Additional resources

- [System-wide cryptographic policies](#)
- [Post-quantum cryptography in Red Hat Enterprise Linux 10 \(Red Hat Blog\)](#)
- [Interoperability of RHEL 10 post-quantum cryptography \(Red Hat Knowledgebase\)](#)
- [Red Hat's path to post-quantum cryptography \(Red Hat Blog\)](#)
- [How Red Hat is integrating post-quantum cryptography into our products \(Red Hat Blog\)](#)

2.3. CREATING A PRIVATE CA BY 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 the [Using shared system certificates](#) chapter 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 2.6, "Using a private CA to issue certificates for CSRs with OpenSSL"](#) and the **ca(1)**, **genpkey(1)**, and **req(1)** man pages on your system 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+hRANCAARb9NxlvkaVjFhOoZbGp/HtIQxbM78E
lwbDP0BI624xBJ8gK68ogSaq2x4SdezFdV1gNeKScDcU+Pj2pELldmdF
-----END PRIVATE KEY-----
```

2.4. CREATING A PRIVATE KEY AND A CSR FOR A TLS SERVER CERTIFICATE BY 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:

```
$ vi <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 [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
...

```

2.5. CREATING A PRIVATE KEY AND A CSR FOR A TLS CLIENT CERTIFICATE BY 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:

```
$ vi <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>

[client_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 the [Using a private CA to issue certificates for CSRs with OpenSSL](#) section 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
...
```

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

To establish a TLS-encrypted data exchange channel, systems must obtain valid certificates from a certificate authority (CA). 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 the [Creating a private CA by using OpenSSL](#) section for more information.
- You have a file containing a CSR. You can find an example of creating the CSR in the [Section 2.4, “Creating a private key and a CSR for a TLS server certificate by using OpenSSL”](#) section.

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
```

Note that the previous example illustrates only the principle and **openssl** does not add all extensions to the certificate automatically. You must add the extensions you require either to the CNF file or append them to parameters of the **openssl** command.

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> -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).

2.7. CREATING A PRIVATE CA BY 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:

```
$ dnf 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 the [Using shared system certificates](#) chapter 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>
```

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

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:

```
$ 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 by 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 **certtool** 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 the [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
...

```

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

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:

```
$ 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 2.10, "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
...

```

2.10. 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 2.7, “Creating a private CA by using GnuTLS”](#) for more information.
- You have a file containing a CSR. You can find an example of creating the CSR in [Section 2.8, “Creating a private key and a CSR for a TLS server certificate by 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>

```

CHAPTER 3. USING SHARED SYSTEM CERTIFICATES

Learn to use the centralized system truststore in RHEL for managing TLS certificates. Using a shared trust location simplifies certificate management and verification across the system.

3.1. THE SYSTEM-WIDE TRUSTSTORE

RHEL contains a centralized system for managing TLS certificates. This shared certificate storage serves as a unified source that NSS, GnuTLS, OpenSSL, and Java use to retrieve system certificate anchors and blocklist information.

By default, the truststore contains the Mozilla CA list, which includes both positive and negative trust. You can update the core Mozilla CA list by using the centralized system.

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/** have lower priority than settings in **/etc/pki/ca-trust/**.

The system treats certificate files based on the subdirectory to which you install them:

- Trust anchors belong to
 - **/usr/share/pki/ca-trust-source/anchors/** or
 - **/etc/pki/ca-trust/source/anchors/**.
- Distrusted certificates are stored in
 - **/usr/share/pki/ca-trust-source/blocklist/** or
 - **/etc/pki/ca-trust/source/blocklist/**.
- Certificates in the extended BEGIN TRUSTED file (OpenSSL trust certificate) format are located in
 - **/usr/share/pki/ca-trust-source/** or
 - **/etc/pki/ca-trust/source/**.

To add a new certificate to the truststore, copy the file containing your certificate to the corresponding directory and use the **update-ca-trust** command to apply the changes. Alternatively, use the **trust anchor** subcommand.

See the **update-ca-trust(8)** and **trust(1)** man pages on your system for more information.



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 first have access to the trust anchor.

3.2. ADDING NEW CERTIFICATES TO THE SYSTEM-WIDE TRUSTSTORE

You can add new certificates to the system-wide truststore so that all cryptographic applications running on the system recognize them as trusted.

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.



NOTE

Even though the Mozilla 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 Mozilla Firefox and Chromium, cache files, and you might have to clear your browser's cache or restart your browser to load the current system certificate configuration.

Prerequisites

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

Procedure

1. 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. Update the system-wide truststore configuration, use the **update-ca-trust** command:

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

3.3. TRUSTED SYSTEM CERTIFICATES MANAGEMENT WITH THE TRUST COMMAND

You can manage certificates within the shared system-wide truststore by using the trust command.

You can add or remove certificates from the system-wide truststore by using either basic file operations with the corresponding files and by using the **update-ca-trust** command as described in the [Adding new certificates to the system-wide truststore](#) section or the **trust** command.

The **trust** command provides a way for managing certificates in the shared system-wide truststore. You can use its subcommands to list, extract, add, remove, or change trust anchors.

- To see the built-in help for the **trust** command, enter it without any arguments or with the **--help** directive. Also, all subcommands of the **trust** commands provide a detailed built-in help, for example:

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

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

■

```
$ trust list
...
pkcs11:id=%DD%04%09%07%A2%F5%7A%7D%52%53%12%92%95%EE%38%80%25%0
D%A6%59;type=cert
  type: certificate
  label: SSL.com Root Certification Authority RSA
  trust: anchor
  category: authority
...
```

- To store a trust anchor into the system-wide truststore, use the **trust anchor** subcommand 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 the ID of a certificate:

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

See the **trust(1)** man page on your system for more information.

CHAPTER 4. PLANNING AND IMPLEMENTING TLS

When hardening TLS configuration, balance strict security settings against client compatibility. Implementing the strictest configuration limits client support, whereas relaxing settings increases compatibility but lowers overall system security.

TLS (Transport Layer Security) is a cryptographic protocol used to secure network communications. When hardening system security by configuring preferred key-exchange protocols, authentication methods, and encryption algorithms, the broader the range of supported clients, the lower the resulting security.

Conversely, strict security settings limit compatibility with clients, potentially locking some users out of the system. Be sure to target the strictest available configuration and relax it only when required for compatibility.

4.1. SSL AND TLS PROTOCOLS

Review the history and usage recommendations for SSL and TLS protocols. This helps you understand which protocol versions are secure for network communication and which should be avoided.

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 cryptographic libraries since RHEL 7.
SSL v3	Do not use. Has serious security vulnerabilities. Removed from the core cryptographic 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 10, disabled in all cryptographic policies.
TLS 1.1	Use for interoperability purposes where needed. Does not support modern cipher suites. In RHEL 10, disabled in all cryptographic policies.
TLS 1.2	Uses the AEAD cipher suites. This version is enabled in all system-wide cryptographic policies. However, optional parts of this protocol contain vulnerabilities, and TLS 1.2 specification also includes support for 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 to the usage of more efficient cryptographic algorithms. TLS 1.3 is also enabled in all system-wide cryptographic policies.

Additional resources

- [IETF: The Transport Layer Security \(TLS\) Protocol Version 1.3](#)

4.2. SECURITY CONSIDERATIONS FOR TLS IN RHEL 10

Review key security aspects when configuring TLS in RHEL. Understanding protocol selection, cipher suites, and key length helps you harden your cryptographic settings.

The default settings provided by libraries included in RHEL 10 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.

In RHEL 10, TLS configuration is performed using the system-wide cryptographic policies mechanism. TLS versions below 1.2 are not supported anymore. **DEFAULT**, **FUTURE**, and **LEGACY** cryptographic policies allow only TLS 1.2 and 1.3. See the [Using system-wide cryptographic policies](#) chapter in the *Security hardening* document for more information.

The most straightforward way to harden your TLS configuration is switching the system-wide cryptographic policy level to **FUTURE** by 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 10 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 cryptographic 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:

4.2.1. Protocols

The latest version of TLS provides the best security mechanism. TLS 1.2 is now the minimum version even when using the **LEGACY** cryptographic policy. Re-enabling older protocol versions is possible through either opting out of cryptographic policies or providing a custom policy, but the resulting configuration will not be supported.

Note that even though that RHEL 10 supports TLS version 1.3, not all features of this protocol are fully supported by RHEL 10 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.

**WARNING**

A RHEL 9.2 and later system running in FIPS mode enforces that any TLS 1.2 connection must use the Extended Master Secret (EMS) extension (RFC 7627) as requires the FIPS 140-3 standard. Thus, legacy clients not supporting EMS or TLS 1.3 cannot connect to RHEL 9 and 10 servers running in FIPS mode, RHEL 9 a 10 clients in FIPS mode cannot connect to servers that support only TLS 1.2 without EMS. See the [TLS Extension "Extended Master Secret" enforced with Red Hat Enterprise Linux 9.2](#) Red Hat Knowledgebase solution for more information.

4.2.2. 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).

4.2.3. 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

- [Using system-wide cryptographic policies](#)

4.3. TLS CONFIGURATION HARDENING IN APPLICATIONS

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

RHEL [system-wide cryptographic policies](#) ensure that your applications that use cryptographic libraries comply with security standards by preventing the use of known insecure protocols, ciphers, or algorithms.

Regardless of the configuration you choose, always ensure that your server application enforces *server-side cipher order*, so that the cipher suite is determined by the order you configure. For more information, see the **crypto-policies(7)**, **config(5)**, and **ciphers(1)** man pages on your system.

4.3.1. TLS configuration of an Apache HTTP server

The **Apache HTTP Server** is compatible with both the OpenSSL and NSS libraries for handling TLS requirements. RHEL 10 includes eponymous packages for the **mod_ssl** functionality. When you install the **mod_ssl** package, it creates the **/etc/httpd/conf.d/ssl.conf** configuration file, which you can use to modify the server's TLS-related settings.

With the **httpd-manual** package, you 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 a 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, if you want to use only the TLS 1.2 and 1.3 protocols, add the line **SSLProtocol all -SSLv3 -TLSv1 -TLSv1.1** to the configuration file.

See the [Configuring TLS encryption on an Apache HTTP Server](#) chapter in the Deploying web servers and reverse proxies document for more information.

4.3.2. TLS configuration of an Nginx HTTP and proxy server

If you want 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, for example:

```
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 web servers and reverse proxies document for more information.

4.3.3. TLS configuration of a Dovecot mail server

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

When modifying the settings in the **/etc/dovecot/conf.d/10-ssl.conf** configuration file, be sure to consider the following three directives at a 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 web servers and reverse proxies](#)
- [Recommendations for Secure Use of Transport Layer Security \(TLS\) and Datagram Transport Layer Security \(DTLS\)](#)

- [Mozilla SSL Configuration Generator](#)
- [SSL Server Test](#)

CHAPTER 5. SECURING SYSTEM DNS TRAFFIC WITH ENCRYPTED DNS (EDNS)

You can enable encrypted DNS (eDNS) to secure DNS communication that uses DNS-over-TLS (DoT) protocol. Encrypted DNS encrypts all DNS traffic end-to-end, with no fallback to insecure protocols, and aligns with the principles of zero trust architecture (ZTA).

The current implementation of eDNS in RHEL uses only the DoT protocol. There are two primary methods to install RHEL with eDNS enabled. You can perform an interactive installation from local media, or you can build a custom bootable ISO to ensure eDNS is configured with an **enforce** policy during and after installation. Alternatively, you can convert an existing RHEL installation to use eDNS.

5.1. OVERVIEW OF COMPONENTS FOR EDNS IN RHEL

Understanding the core components and their layered interactions used in the encrypted DNS (eDNS) setup helps ensure proper configuration and security.

The following components comprise the eDNS setup in RHEL and interact in a layered fashion:

NetworkManager

NetworkManager enables eDNS and enforces the use of encrypted DNS protocols based on the configured policy. It is set to use **dnscconfd** as its backend DNS resolver.

dnscconfd

dnscconfd is a local DNS cache configuration daemon. It simplifies the setup of DNS caching, split DNS, and DNS over TLS (DoT).

unbound

unbound is a validating, recursive, and caching DNS resolver. In the eDNS setup, it serves as the runtime cache service for **dnscconfd**. **unbound** uses TLS for upstream DNS queries, which is essential for encrypting DNS traffic to external DoT servers. **unbound** also manages various caches to store DNS responses, which reduces the need for repeated external queries and improves performance.

5.1.1. eDNS resolution process and core interactions

1. An application requests to resolve a hostname.
2. The system reads the **/etc/resolv.conf** file and sends the query to the local **unbound** service.
3. **unbound** first checks its internal caches for a valid, cached response.
4. If the request record is not found, **unbound** encrypts the DNS query by using TLS and sends it to the configured upstream DoT enabled DNS server.
5. The upstream DoT server processes the query and sends an encrypted DNS response back to **unbound**.
6. **unbound** decrypts, validates, and caches the response.
7. Finally, **unbound** sends the resolved DNS response back to the application.

5.2. INSTALLING RHEL WITH EDNS ENABLED FROM A LOCAL INSTALLATION MEDIA

Install RHEL with encrypted DNS (eDNS) enabled directly from local media using an enforce policy. This helps ensure that all DNS queries remain private and secure during and after the installation process.

If you require a custom CA certificate bundle, you must install it by using the **%certificate** section in the Kickstart file.

During the installation, you must provide both the RHEL installation content and the Kickstart file from local media. You cannot download the Kickstart file from a remote HTTP server because the installation program requires to use DNS to resolve the server's hostname. If your environment is configured to support a fallback to unencrypted DNS, you can perform a standard RHEL installation and configure eDNS afterwards.

Prerequisites

- Commands that start with the **#** command prompt require administrative privileges provided by **sudo** or root user access. For information on how to configure **sudo** access, see [Enabling unprivileged users to run certain commands](#).
- You have the RHEL installation media available locally.
- If you require a custom CA bundle, have your Kickstart file with a **%certificate** section available locally.

Procedure

1. Optional: Create a Kickstart file with a **%certificate** section. Ensure the certificate is saved in a file named **tls-ca-bundle.pem**.

```
%certificate --dir /etc/pki/dns/extracted/pem/ --filename tls-ca-bundle.pem
-----BEGIN CERTIFICATE-----
<Base64-encoded_certificate_content>
-----END CERTIFICATE-----
%end
```

2. Prepare your bootable installation media, and include the Kickstart file if you need a custom CA bundle.
3. [Boot the installation media](#).
4. From the boot menu window, select the required option and press the **e** key to edit the boot parameters.
5. Add the eDNS kernel arguments:

```
linux ($root)/vmlinuz-6.12.0-0.el10_0.x86_64 root=/dev/mapper/rhel-root ro crashkernel=2G-64G:256M,64G-:512M resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet emergency ip=dhcp
rd.net.dns=dns+tls://<server_ip>#<dns_server_hostname> rd.net.dns-resolve-
mode=exclusive rd.net.dns-backend=dnsconfd inst.ks=hd:/dev/sdb1/mykickstart.ks
```

6. When you finish editing, press **Ctrl+X** to start the installation using the specified options.

Verification

- Verify your eDNS configuration:

```
$ dnsconfd status
```

Expected output:

```
Running cache service:
unbound
Resolving mode: exclusive
Config present in service:
{
  ".": [
    "dns+tls://198.51.100.143#dot.dns.example.com"
  ]
}
State of Dnsconfd:
RUNNING
Info about servers: [
  {
    "address": "198.51.100.143",
    "port": 853,
    "name": "dot.dns.example.com",
    "routing_domains": [
      "."
    ],
    "search_domains": [],
    "interface": null,
    "protocol": "dns+tls",
    "dnssec": true,
    "networks": [],
    "firewall_zone": null
  }
]
```

- Verify that DNS server is responsive by using **nslookup**:

```
$ nslookup <domain_name>
```

Replace the **<domain_name>** with the domain that you want to query.

Troubleshooting

- Enable detailed logging in **unbound**:

```
# unbound-control verbosity 5
```

- Review logs for the relevant service:

```
$ journalctl -xe -u <service_name>
```

Replace **<service_name>** with **NetworkManager**, **dnsconfd**, or **unbound**.

Additional resources

- [Creating Kickstart files](#)
- [Kickstart certificates section](#)
- [Creating a bootable installation medium for RHEL](#)
- [Configuring kernel command-line parameters](#)

5.3. INSTALLING RHEL WITH EDNS ENABLED USING A CUSTOM BOOTABLE ISO

Create a custom bootable ISO to install RHEL with encrypted DNS (eDNS) enabled using a strict enforce policy. This method helps ensure that all DNS traffic is private and secure during and after the installation.

If you require a custom CA certificate bundle, you must install it by using the **%certificate** section in the Kickstart file. You then reference this Kickstart file in a script to build a new ISO, which includes kernel arguments to enforce a strict DoT policy. If your environment is configured to support a fallback to unencrypted DNS, you can perform a standard RHEL installation and configure eDNS afterwards.

Prerequisites

- Commands that start with the **#** command prompt require administrative privileges provided by **sudo** or root user access. For information on how to configure **sudo** access, see [Enabling unprivileged users to run certain commands](#).
- You have downloaded the minimal installation Boot ISO image from the Product Downloads page.
- You have a Kickstart file ready with a **%certificate** section if you need a custom CA bundle.
- The **lorax** package is installed.

Procedure

1. Optional: Create a Kickstart file with a **%certificate** section. Ensure the certificate is saved in a file named **tls-ca-bundle.pem**.

```
%certificate --dir /etc/pki/dns/extracted/pem/ --filename tls-ca-bundle.pem
-----BEGIN CERTIFICATE-----
<Base64-encoded_certificate_content>
-----END CERTIFICATE-----
%end
```

2. Add the Kickstart file and kernel arguments into the ISO:
The following script example demonstrates how to create a custom bootable ISO with eDNS enabled. You must create a script file to automate this process.

```
#!/bin/bash

set -ex
```

```

KERNELARGS=""

# Enable network
KERNELARGS+="ip=dhcp "

# Set DoT DNS server
KERNELARGS+="rd.net.dns=dns+tls://_<server_ip>_#_<dns_server_hostname>_"

# Set to 'exclusive' to disable fallback to unencrypted DNS. Other values: 'backup', 'prefer'.
KERNELARGS+="rd.net.dns-resolve-mode=exclusive "

# Set the dnscfgd plugin for NetworkManager
KERNELARGS+="rd.net.dns-backend=dnscfgd "

# Remove any existing ISO to prevent conflicts with the new build
rm -f _<output_iso_filename>_

# Create a new bootable ISO with the Kickstart config file and kernel arguments
mkksiso --ks _<kickstart_file>_ --cmdline "$KERNELARGS" _<input_iso_filename>_
_<output_iso_filename>_

```

3. Run the script.

```
sh <script_filename>
```

4. Install RHEL using the customized ISO file.

Verification

- Verify your eDNS configuration:

```
$ dnscfgd status
```

Expected output:

```

Running cache service:
unbound
Resolving mode: exclusive
Config present in service:
{
  "": [
    "dns+tls://198.51.100.143#dot.dns.example.com"
  ]
}
State of Dnscfgd:
RUNNING
Info about servers: [
  {
    "address": "198.51.100.143",
    "port": 853,
    "name": "dot.dns.example.com",
    "routing_domains": [
      "."
    ],
    "search_domains": [],

```

```
"interface": null,  
"protocol": "dns+tls",  
"dnssec": true,  
"networks": [],  
"firewall_zone": null  
}  
]
```

- Verify that DNS server is responsive by using **nslookup**:

```
$ nslookup <domain_name>
```

Replace the **<domain_name>** with the domain that you want to query.

Troubleshooting

- Enable detailed logging in **unbound**:

```
# unbound-control verbosity 5
```

- Review logs for the relevant service:

```
$ journalctl -xe -u <service_name>
```

Replace **<service_name>** with **NetworkManager**, **dnscconfd**, or **unbound**.

Additional resources

- [Creating Kickstart files](#)
- [Kickstart certificates section](#)
- [Creating a bootable installation medium for RHEL](#)

5.4. ENABLING EDNS ON AN EXISTING RHEL INSTALLATION

You can enable encrypted DNS (eDNS) on an existing RHEL installation to handle all DNS traffic by using DNS-over-TLS.

Prerequisites

- Commands that start with the **#** command prompt require administrative privileges provided by **sudo** or root user access. For information on how to configure **sudo** access, see [Enabling unprivileged users to run certain commands](#).
- Have an existing RHEL installation.
- The following packages are installed on your system:
 - **dnscconfd**
 - **dnscconfd-dracut**
 - **grubby**

- If on an IBM Z system, the **zipl** utility is installed.

Procedure

1. Configure NetworkManager in the **/etc/NetworkManager/conf.d/global-dot.conf** file:

```
[main]
dns=dnsconfd

[global-dns]
resolve-mode=exclusive

[global-dns-domain-*)
servers=dns+tls://<server_ip_1><dns_server_hostname_1>,dns+tls://<server_ip_2><dns_server_hostname_2>
```

For more details on global DNS options, see the **GLOBAL-DNS SECTION** in **NetworkManager.conf(5)** man page on your system.

2. Optional: To use a custom CA bundle for validating upstream DoT servers, copy the PEM-formatted file to the **/etc/pki/dns/extracted/pem/tls-ca-bundle.pem** file.



NOTE

After adding or removing certificates in **/etc/pki/dns/extracted/pem**, restart the **dnscconfd** service to apply the changes.

3. Enable the **dnscconfd** service:

```
# systemctl enable --now dnscconfd
```

4. Reload NetworkManager:

```
# systemctl reload NetworkManager
```

5. Regenerate **initramfs** for all installed kernels to include **dnscconfd** and its configuration:

```
# for kernel in `rpm -q kernel --qf '%{VERSION}-%{RELEASE}-%{ARCH}\n'; do
    dracut -f --kver="$kernel"
done
```

6. Set kernel arguments to the current and newly installed kernel version:

```
# grubby --args="rd.net.dns=dns+tls://<server_ip>#<dns_server_hostname> rd.net.dns-
resolve-mode=exclusive rd.net.dns-backend=dnscconfd" --update-kernel=ALL
```

- If on IBM Z, update the boot menu:

```
# zipl
```

Verification

- Verify your eDNS configuration:

```
$ dnsconfd status
```

Expected output:

```
Running cache service:
unbound
Resolving mode: exclusive
Config present in service:
{
  ".": [
    "dns+tls://198.51.100.143#dot.dns.example.com"
  ]
}
State of Dnsconfd:
RUNNING
Info about servers: [
  {
    "address": "198.51.100.143",
    "port": 853,
    "name": "dot.dns.example.com",
    "routing_domains": [
      "."
    ],
    "search_domains": [],
    "interface": null,
    "protocol": "dns+tls",
    "dnssec": true,
    "networks": [],
    "firewall_zone": null
  }
]
```

- Verify that the DNS server is responsive by using **nslookup**:

```
$ nslookup <domain_name>
```

Replace the **<domain_name>** with the domain that you want to query.

Troubleshooting

- Enable detailed logging in **unbound**:

```
# unbound-control verbosity 5
```

- Review logs for the relevant service:

```
$ journalctl -xe -u <service_name>
```

Replace **<service_name>** with **NetworkManager**, **dnsconfd**, or **unbound**.

Additional resources

- [Changing kernel command-line parameters for all boot entries](#)

5.5. KERNEL PARAMETERS FOR DNS CONFIGURATION

You can use kernel arguments to enable DNS over TLS (DoT) at boot time and set DNS resolution behavior for your system.

rd.net.dns-resolve-mode

Defines how DNS servers from global configuration are used during resolution. The following modes are relevant for both kernel arguments and **NetworkManager.conf** global configuration:

exclusive

Uses only the DNS servers specified by kernel arguments or in **NetworkManager.conf**. Forbids fallback to DNS servers retrieved from connections. This mode is currently relevant only for **dnscfgd** plugin.

prefer

Forbids using DNS servers from connections for general queries unless the queries are subdomains of domains set by connection.

backup

Merges and uses DNS servers from both the global configuration and network connections for the same purposes.

rd.net.dns-servers

Configures the list of DNS servers to use. To define multiple DNS servers, set **rd.net.dns** multiple times:

```
rd.net.dns=dns+tls://<server_ip_1>#<dns_server_hostname_1>
rd.net.dns=dns+tls://<server_ip_2>#<dns_server_hostname_2>
```

For example:

```
rd.net.dns=dns+tls://198.51.100.143#dot.dns.example.com
rd.net.dns=dns+tls://203.0.113.1#dot.dns.example.net
```

rd.net.dns-backend

Specifies the back-end DNS resolver. When set to **dnscfgd**, the system uses **dnscfgd** as a local DNS cache configuration daemon.

5.6. ADDITIONAL RESOURCES

- [Securing DNS with DoT in IdM](#)

CHAPTER 6. SETTING UP AN IPSEC VPN

Configure and manage a secure Virtual Private Network (VPN) by using the Libreswan implementation of the IPsec protocol suite to create encrypted tunnels for secure data transmission over the internet.

IPsec tunnels ensure the confidentiality and integrity of data in transit. Common use cases include connecting branch offices to headquarters or providing remote users with secure access to a corporate network.

RHEL provides different options to configure Libreswan:

- Manually edit the Libreswan configuration files for granular control over advanced options.
- Use the **vpn** RHEL system role to automate the process of creating Libreswan VPN configurations.
- Use Nmstate to configure a Libreswan connection through a declarative API.

Libreswan does not use terms such as "client" and "server". Instead, IPsec refers to endpoints as "left" and "right". This design often enables you to use the same configuration on both hosts because Libreswan dynamically determines which role to adopt. As a convention, administrators typically use "left" for the local host and "right" for the remote host.



NOTE

Libreswan is the only supported VPN technology in RHEL.

IPsec relies on standardized protocols, such as Internet Key Exchange (IKE), to ensure that different systems can communicate effectively. However, in practice, minor differences in how vendors implement these standards can lead to compatibility problems. If you encounter such interoperability issues when connecting Libreswan to a third-party IPsec peer, contact [Red Hat Support](#).

6.1. COMPONENTS IN AN IPSEC VPN

Before setting up an IPsec VPN, it is important to understand its main components: Internet Key Exchange (IKE) for authentication and negotiation, and IPsec for data encryption and transport.

IKE is the protocol two endpoints use to authenticate each other and negotiate connection rules, including encryption algorithms. Libreswan implements IKE in a daemon called **pluto**.

IPsec is the part of the protocol that actually encrypts and transports data according to the policy agreed upon during the IKE negotiation. The Linux kernel implements the IPsec protocol suite.

6.2. LIBRESWAN AUTHENTICATION METHODS

Select the appropriate authentication method to establish a secure VPN connection based on your security needs and network environment.

Libreswan supports the following authentication methods:

Pre-Shared key

The Pre-Shared Key (PSK) method involves both endpoints by using the same secret to authenticate each other. PSKs offer simplicity and broad compatibility, making them suitable for small-scale deployments. However, managing PSKs is risky if the key is reused or not rotated

frequently. For security, PSKs should consist of more than 64 random characters and must meet FIPS strength requirements if your host operates in FIPS mode.

Raw RSA key

This method uses an RSA public and private key pair on each peer for mutual identification. Raw RSA keys provide stronger security than PSKs and are ideal for environments where a full certificate infrastructure is not required.

X.509 certificates

This method uses X.509 certificates issued by a trusted Certificate Authority (CA). Each peer proves its identity by using its certificate and private key, which the other peer verifies against the trusted CA. While providing the highest level of security and scalability for large enterprises, this method is more complex as it requires deploying and maintaining a public key infrastructure (PKI).

NULL authentication

This method provides only encryption with no authentication between peers. Because it does not verify the identity of the remote endpoint, NULL authentication is insecure and offers no protection against man-in-the-middle attacks.

Protection against quantum computers

While not a standalone authentication method, Libreswan offers Post-quantum Pre-shared Keys (PPKs) to protect modern IKEv2 connections from future attacks by quantum computers. This feature is necessary because neither the older IKEv1 protocol nor standard IKEv2 is inherently quantum-resistant on its own. A PPK adds another layer of security on top of the primary authentication method, and its security relies on using a cryptographically strong key that has been distributed securely through an external communication channel.

6.3. MANUALLY CONFIGURING AN IPSEC HOST-TO-HOST VPN WITH RAW RSA KEY AUTHENTICATION

A host-to-host VPN establishes a direct, secure, and encrypted connection between two devices, allowing applications to communicate safely over an insecure network, such as the internet.

For authentication, RSA keys are more secure than pre-shared keys (PSKs) because their asymmetric encryption eliminates the risk of a shared secret. Using RSA keys also simplifies deployment by avoiding the need for a certificate authority (CA), while still providing strong peer-to-peer authentication.

Perform the steps on both hosts.

Procedure

1. If Libreswan is not yet installed, perform the following steps:

- a. Install the **libreswan** package:

```
# dnf install libreswan
```

- b. Initialize the Network Security Services (NSS) database:

```
# ipsec initnss
```

The command creates the database in the **/var/lib/ipsec/nss/** directory.

- c. Enable and start the **ipsec** service:

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

- d. Open the IPsec ports and protocols in the firewall:

```
# firewall-cmd --permanent --add-service="ipsec"
# firewall-cmd --reload
```

2. Create an RSA key pair:

```
# ipsec newhostkey
```

The **ipsec** utility stores the key pair in the NSS database.

3. Designate your peers. In an IPsec tunnel, you must designate one host as *left* and the other as *right*. This is an arbitrary choice. A common practice is to call your local host *left* and the remote host *right*.
4. Display the Certificate Key Attribute ID (CKAID) on both the left and right peer:

```
# ipsec showhostkey --list
<1> RSA keyid: <key_id> ckaid: <ckaid>
```

You require the CKAIDs of both peers in the next steps.

5. Display the public keys:

- a. On the left peer, enter:

```
# ipsec showhostkey --left --ckaid <ckaid_of_left_peer>
# rsakey AwEAAAdKCx
leftrsasigkey=0sAwEAAAdKCxpc9db48cehzQiQD...
```

- b. On the right peer, enter:

```
# ipsec showhostkey --right --ckaid <ckaid_of_right_peer>
# rsakey AwEAAcNWC
rightrsasigkey=0sAwEAAcNWCzZO+PR1j8WbO8X...
```

The commands display the public keys with the corresponding parameters that you must use in the configuration file.

6. Create a **.conf** file for the connection in the **/etc/ipsec.d/** directory. For example, create the **/etc/ipsec.d/host-to-host.conf** file with the following settings:

```
conn <connection_name>
# General setup and authentication type
auto=start
authby=rsasig

# Peer A
left=<ip_address_or_fqdn_of_left_peer>
leftid=@peer_a
leftrsasigkey=<public_key_of_left_peer>

# Peer B
```

```
right=<ip_address_or_fqdn_of_right_peer>
rightid=@peer_b
rightrsasigkey=<public_key_of_right_peer>
```



NOTE

You can use the same configuration file on both hosts, and Libreswan identifies whether it is operating on the left or right host by using internal information. However, it is important that all values in **left*** parameters belong to one peer and the values in **right*** parameters belong to the other.

The settings specified in the example include the following:

conn <connection_name>

Defines the connection name. The name is arbitrary, and Libreswan uses it to identify the connection. You must indent parameters in this connection by at least one space or tab.

auto=<type>

Controls how the connection is initiated. If you set the value to **start**, Libreswan activates the connection automatically when the service starts.

authby=rsasig

Enables RSA signature authentication for this connection.

left=<ip_address_or_fqdn_of_left_peer> and right=<ip_address_or_fqdn_of_right_peer>

Defines the IP address or DNS name of the peers.

leftid=<id> and rightid=<id>

Defines how each peer is identified during the Internet Key Exchange (IKE) negotiation process. This can be a fully-qualified domain name (FQDN), an IP address, or a literal string. In the latter case, precede the string with an @ sign.

leftrsasigkey=<public_key> and rightrsasigkey=<public_key>

Specifies the public key of the peers. Use the values displayed by the **ipsec showhostkey** command in a previous step.

7. Restart the **ipsec** service:

```
# systemctl restart ipsec
```

If you use **auto=start** in the configuration file, the connection is automatically activated. With other methods, additional steps are required to activate the connection. For details, see the **ipsec.conf(5)** man page on your system.

Verification

- Display the IPsec status:

```
# ipsec status
```

If the connection is successfully established, the output contains lines as follows:

- Phase 1 of an Internet Key Exchange version 2 (IKEv2) negotiation has been successfully completed:

■

```
#1: "<connection_name>":500 ESTABLISHED_IKE_SA (established IKE SA); REKEY in 28523s; REPLACE in 28793s; newest; idle;
```

The Security Association (SA) is now ready to negotiate the actual data encryption tunnels, known as child SAs or Phase 2 SAs.

- A child SA has been established:

```
#2: "<connection_name>":500 ESTABLISHED_CHILD_SA (established Child SA); REKEY in 28523s; REPLACE in 28793s; newest; eroute owner; IKE SA #1; idle;
```

This is the actual tunnel that your data traffic flows through.

Next steps

- If you use this host in a network with DHCP or Stateless Address Autoconfiguration (SLAAC), the connection can be vulnerable to being redirected. For details and mitigation steps, see [Assigning a VPN connection to a dedicated routing table to prevent the connection from bypassing the tunnel](#).

6.4. MANUALLY CONFIGURING AN IPSEC SITE-TO-SITE VPN WITH RAW RSA KEY AUTHENTICATION

A site-to-site VPN establishes a secure, encrypted tunnel between two distinct networks, seamlessly linking them across an insecure public network such as the internet.

For example, a site-to-site VPN enables devices in a branch office to access resources at a corporate headquarters just as if they were all part of the same local network.

For authenticating the gateway devices, RSA keys are more secure than pre-shared keys (PSKs) because their asymmetric encryption eliminates the risk of a shared secret. Using RSA keys also simplifies deployment by avoiding the need for a certificate authority (CA), while still providing strong peer-to-peer authentication.

Perform the steps on both gateway devices.

Prerequisites

- Routes in both networks ensure that the traffic to the remote networks is sent through the local VPN gateway devices.

Procedure

1. If Libreswan is not yet installed, perform the following steps:

- a. Install the **libreswan** package:

```
# dnf install libreswan
```

- b. Initialize the Network Security Services (NSS) database:

```
# ipsec initnss
```

The command creates the database in the **/var/lib/ipsec/nss/** directory.

- c. Enable and start the **ipsec** service:

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

- d. Open the IPsec ports and protocols in the firewall:

```
# firewall-cmd --permanent --add-service="ipsec"
# firewall-cmd --reload
```

2. Create an RSA key pair:

```
# ipsec newhostkey
```

The **ipsec** utility stores the key pair in the NSS database.

3. Designate your peers. In an IPsec tunnel, you must designate one host as *left* and the other as *right*. This is an arbitrary choice. A common practice is to call your local host *left* and the remote host *right*.
4. Display the Certificate Key Attribute ID (CKAID) on both the left and right peer:

```
# ipsec showhostkey --list
< 1> RSA keyid: <key_id> ckaid: <ckaid>
```

You require the CKAIDs of both peers in the next steps.

5. Display the public keys:

- a. On the left peer, enter:

```
# ipsec showhostkey --left --ckaid <ckaid_of_left_peer>
# rsakey AwEAAAdKCx
leftrsasigkey=0sAwEAAAdKCxpc9db48cehzQiQD...
```

- b. On the right peer, enter:

```
# ipsec showhostkey --right --ckaid <ckaid_of_right_peer>
# rsakey AwEAAcNWC
rightrsasigkey=0sAwEAAcNWCzZO+PR1j8WbO8X...
```

The commands display the public keys with the corresponding parameters that you must use in the configuration file.

6. Create a **.conf** file for the connection in the **/etc/ipsec.d/** directory. For example, create the **/etc/ipsec.d/site-to-site.conf** file with the following settings:

```
conn <connection_name>
# General setup and authentication type
auto=start
authby=rsasig

# Site A
left=<ip_address_or_fqdn_of_left_peer>
leftid=@site_a
```

```

lefttrsasigkey=<public_key_of_left_peer>
leftsubnet=192.0.2.0/24

# Site B
right=<ip_address_or_fqdn_of_right_peer>
rightid=@site_b
righttrsasigkey=<public_key_of_right_peer>
rightsubnet={198.51.100.0/24, 203.0.113.0/24}

```



NOTE

You can use the same configuration file on both gateway devices, and Libreswan identifies whether it is operating on the left or right host by using internal information. However, it is important that all values in **left*** parameters belong to one peer and the values in **right*** parameters belong to the other.

The settings specified in the example include the following:

conn <connection_name>

Defines the connection name. The name is arbitrary, and Libreswan uses it to identify the connection. You must indent parameters in this connection by at least one space or tab.

auto=<type>

Controls how the connection is initiated. If you set the value to **start**, Libreswan activates the connection automatically when the service starts.

authby=rsasig

Enables RSA signature authentication for this connection.

left=<ip_address_or_fqdn_of_left_peer> and right=<ip_address_or_fqdn_of_right_peer>

Defines the IP address or DNS name of the peers.

leftid=<id> and rightid=<id>

Defines how each peer is identified during the Internet Key Exchange (IKE) negotiation process. This can be a fully-qualified domain name (FQDN), an IP address, or a literal string. In the latter case, precede the string with an @ sign.

lefttrsasigkey=<public_key> and righttrsasigkey=<public_key>

Specifies the public key of the peers. Use the values displayed by the **ipsec showhostkey** command in a previous step.

leftsubnet=<subnet> and rightsubnet=<subnet>

Defines subnets in classless inter-domain routing (CIDR) format that are connected through the tunnel. If you want to tunnel multiple subnets on one side, specify them in curly brackets and separate them with a comma.

7. 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

```

8. Restart the **ipsec** service:

```

# systemctl restart ipsec

```

If you use **auto=start** in the configuration file, the connection is automatically activated. With other methods, additional steps are required to activate the connection. For details, see the **ipsec.conf(5)** man page on your system.

Verification

1. Display the IPsec status:

```
# ipsec status
```

If the connection is successfully established, the output contains lines as follows:

- Phase 1 of an Internet Key Exchange version 2 (IKEv2) negotiation has been successfully completed:

```
#2: "<connection_name>":500 ESTABLISHED_IKE_SA (established IKE SA); REKEY in 28523s; REPLACE in 28793s; newest; idle;
```

The Security Association (SA) is now ready to negotiate the actual data encryption tunnels, known as child SAs or Phase 2 SAs.

- A child SA has been established:

```
#3: "<connection_name>":500 ESTABLISHED_CHILD_SA (established Child SA); REKEY in 28523s; REPLACE in 28793s; newest; eroute owner; IKE SA #2; idle;
```

This is the actual tunnel that your data traffic flows through.

2. From a client in the local subnet, ping a client in the remote subnet.

Next steps

- If you use this host in a network with DHCP or Stateless Address Autoconfiguration (SLAAC), the connection can be vulnerable to being redirected. For details and mitigation steps, see [Assigning a VPN connection to a dedicated routing table to prevent the connection from bypassing the tunnel](#).

6.5. MANUALLY CONFIGURING AN IPSEC HOST-TO-SITE VPN WITH CERTIFICATE-BASED AUTHENTICATION

A host-to-site VPN establishes a secure, encrypted connection between an individual remote computer and a private network, allowing them to be seamlessly linked across an insecure public network, such as the internet.

A host-to-site VPN is ideal for remote employees who need to access resources on their company's internal network from their computer as if they were physically in the office.

For authentication, using digital certificates managed by a Certificate Authority (CA) offers a highly secure and scalable solution. Each connecting host and the gateway presents a certificate signed by a trusted CA. This method provides strong, verifiable authentication and simplifies user management. Access can be granted or revoked centrally at the CA, and Libreswan enforces this by checking each certificate against a certificate revocation list (CRL), denying access if a certificate appears on the list.

6.5.1. Setting up an IPsec gateway manually

You must configure the Libreswan IPsec gateway properly to enable secure remote access. Libreswan reads the server certificate, private key, and CA certificate from a Network Security Services (NSS) database.

The following example permits authenticated clients to access the internal 192.0.2.0/24 subnet and dynamically assigns an IP address from a virtual IP pool to each client. To maintain security, the gateway verifies that client certificates are issued by the same trusted CA and automatically uses a certificate revocation list (CRL) to ensure access is denied for any revoked certificates.

Prerequisites

- The Public Key Cryptography Standards #12 (PKCS #12) file **~/.file.p12** exists on the gateway with the following contents:
 - The private key of the server
 - The server certificate
 - The CA certificate
 - If required, intermediate certificates

For details about creating a private key and certificate signing request (CSR), as well as about requesting a certificate from a CA, see your CA's documentation.

- The server certificate contains the following fields:
 - Extended Key Usage (EKU) is set to **TLS Web Server Authentication**.
 - Common Name (CN) or Subject Alternative Name (SAN) is set to the fully-qualified domain name (FQDN) of the gateway.
 - X509v3 CRL distribution points contain URLs to Certificate Revocation Lists (CRLs).
- A return route for VPN client traffic is configured on the internal network, pointing to the VPN gateway.

Procedure

1. If Libreswan is not yet installed:

- a. Install the **libreswan** package:

```
# dnf install libreswan
```

- b. Initialize the Network Security Services (NSS) database:

```
# ipsec initnss
```

The command creates the database in the **/var/lib/ipsec/nss/** directory.

- c. Enable and start the **ipsec** service:

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

- d. Open the IPsec ports and protocols in the firewall:

```
# firewall-cmd --permanent --add-service="ipsec"
# firewall-cmd --reload
```

2. Import the PKCS #12 file into the NSS database:

```
# ipsec import ~/file.p12
Enter password for PKCS12 file: <password>
pk12util: PKCS12 IMPORT SUCCESSFUL
correcting trust bits for Example-CA
```

3. Display the nicknames of the server and CA certificates:

```
# certutil -L -d /var/lib/ipsec/nss/
Certificate Nickname      Trust Attributes
                        SSL,S/MIME,JAR/XPI

vpn-gateway              u,u,u
Example-CA               CT,,
...
```

You need this information for the configuration file.

4. Create a **.conf** file for the connection in the **/etc/ipsec.d/** directory. For example, create the **/etc/ipsec.d/host-to-site.conf** file with the following settings:

- a. Add a **config setup** section to enable CRL checks:

```
config setup
    crl-strict=yes
    crlcheckinterval=1h
```

The settings specified in the example include the following:

crl-strict=yes

Enables CRL checks. Authenticating clients are rejected if no CRL is available in the NSS database.

crlcheckinterval=1h

Re-fetches the CRL from the URL specified in the server's certificate after the specified period.

- b. Add a section for the gateway:

```
conn <connection_name>
    # General setup and authentication type
    auto=start
    ikev2=insist
    authby=rsasig

    # VPN gateway settings
    left=%defaultroute
    leftid=%fromcert
    leftcert="<server_certificate_nickname>"
    leftrsasigkey=%cert
    leftsendcert=always
```

```

leftsubnet=192.0.2.0/24
rekey=no
mobike=yes
narrowing=yes

# Client-related settings
right=%any
rightid=%fromcert
rightrsasigkey=%cert
rightaddresspool=198.51.100.129-198.51.100.254
rightmodecfgclient=yes
modecfgclient=yes
modecfgdns=192.0.2.5
modecfgdomains="example.com"

# Dead Peer Detection
dpddelay=30
dpdtimeout=120
dpdaction=clear

```

The settings specified in the example include the following:

ikev2=insist

Defines the modern IKEv2 protocol as the only allowed protocol without fallback to IKEv1.

left=%defaultroute

Dynamically sets the IP address of the default route interface when the **ipsec** service starts. Alternatively, you can set the **left** parameter to the IP address or the FQDN of the host.

leftid=%fromcert and rightid=%fromcert

Configures Libreswan to retrieve the identity from the distinguished name (DN) field of the certificate.

leftcert="<server_certificate_nickname>"

Sets the nickname of the server's certificate used in the NSS database.

leftrsasigkey=%cert and rightrsasigkey=%cert

Configures Libreswan to use the RSA public key embedded in the certificate.

leftsendcert=always

Instructs the gateway to always send the certificate, so that clients can validate it against the CA certificate.

leftsubnet=<subnets>

Specifies the subnets connected to the gateway that clients can access through the tunnel.

mobike=yes

Enables clients to seamlessly roam among networks.

rightaddresspool=<ip_range>

Specifies from which range the gateway can assign IP addresses to the clients.

modecfgclient=yes

Enables the gateway to use DNS and IPsec to protect modecfgclient=yes

Enables clients to receive the DNS server IP set in the **modectrgans** parameter and the DNS search domain set in **modectfgdomains**.

For details about all parameters used in the example, see the **ipsec.conf(5)** man page on your system.

5. Enable packet forwarding:

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

6. Restart the **ipsec** service:

```
# systemctl restart ipsec
```

If you use **auto=start** in the configuration file, the connection is automatically activated. With other methods, additional steps are required to activate the connection. For details, see the **ipsec.conf(5)** man page on your system.

Verification

1. [Configure a client and connect to the VPN gateway](#) .
2. Check if the service loaded the CRL and added the entries to the NSS database:

```
# ipsec listcrls

List of CRLs:

issuer: CN=Example-CA
revoked certs: 1
updates: this Tue Jul 15 10:22:36 2025
         next Sun Jan 11 10:22:36 2026

List of CRL fetch requests:

Jul 15 15:13:56 2025, trials: 1
  issuer: 'CN=Example-CA'
  distPts: 'https://ca.example.com/crl.pem'
```

Next steps

- Configure firewall rules to ensure that clients can only communicate with required resources. For details about firewalls, see [Configuring firewalls and packet filters](#).

6.5.2. Configuring a client to connect to an IPsec VPN gateway by using GNOME Settings

To access resources on a remote private network, users must first configure an IPsec VPN connection. The GNOME Settings application provides a graphical solution to create an IPsec VPN connection profile in NetworkManager and to establish the tunnel.

Prerequisites

- You configured the IPsec VPN gateway .
- The **NetworkManager-libreswan-gnome** package is installed.
- The PKCS #12 file `~/file.p12` exists on the client with the following contents:
 - The private key of the user
 - The user certificate
 - The CA certificate
 - If required, intermediate certificates

For details about creating a private key and certificate signing request (CSR), as well as about requesting a certificate from a CA, see your CA's documentation.

- The Extended Key Usage (EKU) in the certificate is set to **TLS Web Client Authentication**.

Procedure

1. Initialize the Network Security Services (NSS) database:

```
# ipsec initnss
```

The command creates the database in the `/var/lib/ipsec/nss/` directory.

2. Import the PKCS #12 file into the NSS database:

```
# ipsec import ~/file.p12
Enter password for PKCS12 file: <password>
pk12util: PKCS12 IMPORT SUCCESSFUL
correcting trust bits for Example-CA
```

3. Display the nicknames of the user and CA certificates:

```
# certutil -L -d /var/lib/ipsec/nss/
Certificate Nickname      Trust Attributes
                        SSL,S/MIME,JAR/XPI

user                      u,u,u
Example-CA                CT,,
...
```

You require this information in the configuration file.

4. Press the **Super** key, type **Settings**, and press **Enter** to open the GNOME **Settings** application.
5. Click the **+** button next to the **VPN** entry.
6. Select **IPsec based VPN** from the list.
7. On the **Identity** tab, fill the fields as follows:

Table 6.1. Identity tab settings

Field name	Value	Corresponding ipsec.conf parameter
Name	<i><networkmanager_profile_name></i>	N/A
Gateway	<i><ip_address_or_fqdn_of_the_gateway></i>	right
Type	IKEv2 (certificate)	authby
Group name	%fromcert	leftid
Certificate name	<i><user_certificate_nickname></i>	leftcert
Remote ID	%fromcert	rightid

8. Click **Advanced**.
9. In the **Advanced properties** window, fill the fields of the **Connectivity** tab as follows:

Table 6.2. Connectivity tab settings

Field name	Value	Corresponding ipsec.conf parameter
Remote Network	192.0.2.0/24	rightsubnet
Narrowing	Selected	narrowing
Enable MOBIKE	yes	mobike
Delay	30	dpddelay
Timeout	120	dpdtimeout
Action	Clear	dpdaction

10. Click **Apply** to return to the connection settings.
11. Click **Apply** to save the connection.
12. In the **Network** tab of the **Settings** application, toggle the switch next to the VPN profile to activate the connection.

Verification

- Establish a connection to a host in the remote network or ping it.

Next steps

- If you use this host in a network with DHCP or Stateless Address Autoconfiguration (SLAAC), the connection can be vulnerable to being redirected. For details and mitigation steps, see [Assigning a VPN connection to a dedicated routing table to prevent the connection from bypassing the tunnel](#).

6.6. MANUALLY CONFIGURING AN IPSEC MESH VPN WITH CERTIFICATE-BASED AUTHENTICATION

An IPsec mesh creates a fully interconnected network where every server can communicate securely and directly with every other server. This is ideal for distributed database clusters or high-availability environments that span multiple data centers or cloud providers.

Establishing a direct, encrypted tunnel between each pair of servers ensures secure communication without a central bottleneck. For authentication, using digital certificates managed by a Certificate Authority (CA) offers a highly secure and scalable solution. Each host in the mesh presents a certificate signed by a trusted CA. This method provides strong, verifiable authentication and simplifies user management. Access can be granted or revoked centrally at the CA, and Libreswan enforces this by checking each certificate against a certificate revocation list (CRL), denying access if a certificate appears on the list.

Prerequisites

- A Public Key Cryptography Standards #12 (PKCS #12) file exists on each peer in the mesh with the following contents:
 - The private key of the server
 - The server certificate
 - The CA certificate
 - If required, intermediate certificates

For details about creating a private key and certificate signing request (CSR), as well as about requesting a certificate from a CA, see your CA's documentation.

- The server certificate contains the following fields:
 - Extended Key Usage (EKU) is set to **TLS Web Server Authentication**.
 - Common Name (CN) or Subject Alternative Name (SAN) is set to the fully-qualified domain name (FQDN) of the host.
 - X509v3 CRL distribution points contain URLs to Certificate Revocation Lists (CRLs).

Procedure

1. If Libreswan is not yet installed, perform the following steps:
 - a. Install the **libreswan** package:

```
# dnf install libreswan
```

-
- b. Initialize the Network Security Services (NSS) database:

```
# ipsec initnss
```

The command creates the database in the `/var/lib/ipsec/nss/` directory.

- c. Enable and start the **ipsec** service:

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

- d. Open the IPsec ports and protocols in the firewall:

```
# firewall-cmd --permanent --add-service="ipsec"
# firewall-cmd --reload
```

2. Import the PKCS #12 file into the NSS database:

```
# ipsec import <file>.p12
Enter password for PKCS12 file: <password>
pk12util: PKCS12 IMPORT SUCCESSFUL
correcting trust bits for Example-CA
```

3. Display the nicknames of the server and CA certificates:

```
# certutil -L -d /var/lib/ipsec/nss/
Certificate Nickname      Trust Attributes
                        SSL,S/MIME,JAR/XPI

server1                   u,u,u
Example-CA                 CT,,
...
```

You need this information for the configuration file.

4. Create a **.conf** file for the connection in the `/etc/ipsec.d/` directory. For example, create the `/etc/ipsec.d/mesh.conf` file with the following settings:

- a. Add a **config setup** section to enable CRL checks:

```
config setup
    crl-strict=yes
    crlcheckinterval=1h
```

The settings specified in the example include the following:

crl-strict=yes

Enables CRL checks. Authenticating peers are rejected if no CRL is available in the NSS database.

crlcheckinterval=1h

Re-fetches the CRL from the URL specified in the server's certificate after the specified period.

- b. Add a section that enforces traffic among members in the mesh:

```
conn <connection_name>
# General setup and authentication type
auto=ondemand
authby=rsasig

# Local settings settings
left=%defaultroute
leftid=%fromcert
leftcert="<server_certificate_nickname>"
lefttrsasigkey=%cert
leftsendcert=always
failureshunt=drop
type=transport

# Settings related to other peers in the mesh
right=%opportunisticgroup
rightid=%fromcert
```

The settings specified in the example include the following:

left=%defaultroute

Dynamically sets the IP address of the default route interface when the **ipsec** service starts. Alternatively, you can set the **left** parameter to the IP address or the FQDN of the host.

leftid=%fromcert and rightid=%fromcert

Configures Libreswan to retrieve the identity from the distinguished name (DN) field of the certificate.

leftcert="<server_certificate_nickname>"

Sets the nickname of the server's certificate used in the NSS database.

lefttrsasigkey=%cert

Configures Libreswan to use the RSA public key embedded in the certificate.

leftsendcert=always

Instructs the peer to always send the certificate, so that peers can validate it against the CA certificate.

failureshunt=drop

Enforces encryption and drops traffic if IPsec negotiation fails. This is critical for a secure mesh.

right=%opportunisticgroup

Specifies that the connection should apply to a dynamic group of remote peers defined in a policy file. This enables Libreswan to instantiate IPsec tunnels opportunistically for each listed IP or subnet in that group.

For details about all parameters used in the example, see the **ipsec.conf(5)** man page on your system.

5. Create the **/etc/ipsec.d/policies/server-mesh** policy file that specifies the peers or subnets in classless inter-domain routing (CIDR) format:

```
192.0.2.0/24
```

```
198.51.100.0/24
```

With these settings, the **ipsec** service encrypts traffic between hosts in these subnets. If a host is not configured as a member of the IPsec mesh, communication between this host and the mesh members fails.

- Restart the **ipsec** service:

```
# systemctl restart ipsec
```

- Repeat the procedure on every host in the subnets you specified in the policy file.

Verification

- Send traffic to a host in the mesh to establish the tunnel. For example, ping the host:

```
# ping -c3 <peer_in_mesh>
```

- Display the IPsec status:

```
# ipsec status
```

If the connection is successfully established, the output contains lines as follows for the peer:

- Phase 1 of an Internet Key Exchange version 2 (IKEv2) negotiation has been successfully completed:

```
#1: "<connection_name>#192.0.2.0/24"[1] ...192.0.2.2:500 ESTABLISHED_IKE_SA
(established IKE SA); REKEY in 12822s; REPLACE in 13875s; newest; idle;
```

The Security Association (SA) is now ready to negotiate the actual data encryption tunnels, known as child SAs or Phase 2 SAs.

- A child SA has been established:

```
#2: "<connection_name>#192.0.2.0/24"[1] ...192.0.2.2:500 ESTABLISHED_CHILD_SA
(established Child SA); REKEY in 13071s; REPLACE in 13875s; newest; eroute owner;
IKE SA #1; idle;
```

This is the actual tunnel that your data traffic flows through.

- Check if the service loaded the CRL and added the entries to the NSS database:

```
# ipsec listcrls
```

List of CRLs:

```
issuer: CN=Example-CA
revoked certs: 1
updates: this Tue Jul 15 10:22:36 2025
         next Sun Jan 11 10:22:36 2026
```

List of CRL fetch requests:

```
Jul 15 15:13:56 2025, trials: 1
issuer: 'CN=Example-CA'
distPts: 'https://ca.example.com/crl.pem'
```

Next steps

- If you use this host in a network with DHCP or Stateless Address Autoconfiguration (SLAAC), the connection can be vulnerable to being redirected. For details and mitigation steps, see [Assigning a VPN connection to a dedicated routing table to prevent the connection from bypassing the tunnel](#).

6.7. PROTECTING THE IPSEC NSS DATABASE WITH A PASSWORD

By default, only the root user can access the IPsec Network Security Services (NSS) database in the `/var/lib/ipsec/nss/` directory. You can additionally protect the database with a password. This is required if you run RHEL in Federal Information Processing Standard (FIPS) mode.

Prerequisites

- The `/var/lib/ipsec/nss/` directory contains the NSS database.

Procedure

1. Enable password protection for the Libreswan NSS database:

```
# certutil -W -d /var/lib/ipsec/nss/
```

2. Enter the current password:

```
Enter Password or Pin for "NSS Certificate DB": <password>
```

If the database is currently not protected by a password, press **Enter**.

3. Enter the new password:

```
Enter new password: <new_password>
Re-enter password: <new_password>
```

4. To unlock the database, the **ipsec** service requires the `/etc/ipsec.d/nsspassword` file. Create the file with the following content:

- If the host does not run in FIPS mode:

```
NSS Certificate DB:<password>
```

- If the host runs in FIPS mode:

```
NSS FIPS 140-2 Certificate DB:<password>
```

5. Set secure permissions on the `/etc/ipsec.d/nsspassword` file:

```
# chmod 600 /etc/ipsec.d/nsspassword
# chown root:root /etc/ipsec.d/nsspassword
```

- 6. Restart the **ipsec** service:

```
# systemctl restart ipsec
```

Verification

1. Verify that the **ipsec** service is running:

```
# systemctl is-active ipsec
```

If the command returns **active**, the service successfully uses the password file to unlock the NSS database.

2. Perform an action on the NSS database that requires the password. For example, display the private keys:

```
# certutil -K -d /var/lib/ipsec/nss/
certutil: Checking token "NSS Certificate DB" in slot "NSS User Private Key and Certificate Services"
Enter Password or Pin for "NSS Certificate DB":
```

Verify that the command prompts for the password.

6.8. USING IPSEC ON A SYSTEM WITH FIPS MODE ENABLED

RHEL in Federal Information Processing Standard (FIPS) mode exclusively uses validated cryptographic modules, automatically disabling legacy protocols and ciphers. Enabling FIPS mode is often a requirement for federal compliance and enhances system security.

The Libreswan IPsec implementation provided by RHEL is fully FIPS-compliant. When the system is in FIPS mode, Libreswan automatically uses the certified cryptographic modules without requiring any additional configuration, regardless of whether Libreswan is installed on a new FIPS-enabled system or when FIPS mode is activated on a system with an existing Libreswan VPN.

If FIPS mode is enabled, you can confirm that Libreswan is running in FIPS mode:

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

To list the allowed algorithms and ciphers in Libreswan in FIPS mode, enter:

```
# ipsec pluto --selftest 2>&1
...
FIPS Encryption algorithms:
AES_CCM_16 {256,192,*128} IKEv1: ESP IKEv2: ESP FIPS aes_ccm, aes_ccm_c
AES_CCM_12 {256,192,*128} IKEv1: ESP IKEv2: ESP FIPS aes_ccm_b
AES_CCM_8 {256,192,*128} IKEv1: ESP IKEv2: ESP FIPS aes_ccm_a
...
```

6.9. CONFIGURING TCP FALLBACK FOR AN IPSEC VPN CONNECTION

Standard IPsec VPNs can fail on restrictive networks that block the UDP and Encapsulating Security Payload (ESP) protocols. To ensure connectivity in such environments, Libreswan can encapsulate all VPN traffic within a TCP connection.



IMPORTANT

Encapsulating VPN packets within TCP can reduce throughput and increase latency. For this reason, use TCP encapsulation only as a fallback option or if UDP-based connections are consistently blocked in your environment.

Prerequisites

- The IPsec connection is configured.

Procedure

1. Edit the `/etc/ipsec.conf` file, and make the following changes in the **config setup** section:

- a. Configure Libreswan to listen on a TCP port:

```
listen-tcp=yes
```

- b. By default, Libreswan listens on port 4500. If you want to use a different port, enter:

```
tcp-remoteport=<port_number>
```

- c. Decide whether TCP should be used as a fallback option if UDP is not available or permanent:

- As a fallback option, enter:

```
enable-tcp=fallback
retransmit-timeout=5s
```

By default, Libreswan waits 60 seconds after a failed attempt to connect by using UDP before retrying the connection over TCP. Lowering the **retransmit-timeout** value shortens the delay, enabling the fallback protocol to initiate more quickly.

- As a permanent replacement for UDP, enter:

```
enable-tcp=yes
```

2. Restart the **ipsec** service:

```
# systemctl restart ipsec
```

3. If you configured a TCP port other than the default 4500, open the port in the firewall:

```
# firewall-cmd --permanent --add-port=<tcp_port>/tcp
# firewall-cmd --reload
```

4. Repeat the procedure on the peers that use this gateway.

6.10. ENABLING LEGACY CIPHERS AND ALGORITHMS IN LIBRESWAN

Enable legacy ciphers and algorithms in Libreswan for backward compatibility with other IPsec peers. This overrides the RHEL system-wide cryptographic policies which, by default, enforce strong encryption ciphers and algorithms for IPsec and Internet Key Exchange (IKE).

The RHEL system-wide cryptographic policies create a special connection called **%default**. This connection sets the default values for the **keyexchange**, **esp**, and **ike** parameters.

Prerequisites

- Libreswan is installed.

Procedure

1. To override the defaults set by the RHEL system-wide cryptographic policies, add the **keyexchange**, **esp**, and **ike** parameters to your connection configuration and set them to the values you require. For example:

```
conn <connection_name>
    keyexchange=ikev1
    ike=aes-sha2,aes-sha1;modp2048
    esp=aes-sha2,aes-sha1
    ...
```

2. Restart the **ipsec** service:

```
# systemctl restart ipsec
```

6.11. ASSIGNING A VPN CONNECTION TO A DEDICATED ROUTING TABLE TO PREVENT THE CONNECTION FROM BYPASSING THE TUNNEL

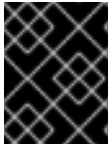
To protect your VPN connection from traffic redirection attacks, assign it to a dedicated routing table. This prevents malicious network servers from bypassing the secure tunnel and compromising data integrity.

Both a DHCP server and Stateless Address Autoconfiguration (SLAAC) can add routes to a client's routing table. For example, a malicious DHCP server can use this feature to force a host with VPN connection to redirect traffic through a physical interface instead of the VPN tunnel. This vulnerability is also known as TunnelVision and described in the [CVE-2024-3661](#) vulnerability article.

To mitigate this vulnerability, you can assign the VPN connection to a dedicated routing table. This prevents the DHCP configuration or SLAAC from manipulating routing decisions for network packets intended for the VPN tunnel.

Follow the steps if at least one of the conditions applies to your environment:

- At least one network interface uses DHCP or SLAAC.
- Your network does not use mechanisms, such as DHCP snooping, that prevent a rogue DHCP server.



IMPORTANT

Routing the entire traffic through the VPN prevents the host from accessing local network resources.

Procedure

1. Decide which routing table you want to use. The following steps use table 75. By default, RHEL does not use the tables 1-254, and you can use any of them.
2. Configure the VPN connection profile to place the VPN routes in a dedicated routing table:

```
# nmcli connection modify <vpn_connection_profile> ipv4.route-table 75 ipv6.route-table 75
```

3. Set a low priority value for the table you used in the previous command:

```
# nmcli connection modify <vpn_connection_profile> ipv4.routing-rules "priority 32345 from all table 75" ipv6.routing-rules "priority 32345 from all table 75"
```

The priority value can be any value between 1 and 32766. The lower the value, the higher the priority.

4. Reconnect the VPN connection:

```
# nmcli connection down <vpn_connection_profile>
# nmcli connection up <vpn_connection_profile>
```

Verification

1. Display the IPv4 routes in table 75:

```
# ip route show table 75
...
192.0.2.0/24 via 192.0.2.254 dev vpn_device proto static metric 50
default dev vpn_device proto static scope link metric 50
```

The output confirms that both the route to the remote network and the default gateway are assigned to routing table 75 and, therefore, all traffic is routed through the tunnel. If you set **ipv4.never-default true** in the VPN connection profile, a default route is not created and, therefore, not visible in this output.

2. Display the IPv6 routes in table 75:

```
# ip -6 route show table 75
...
2001:db8:1::/64 dev vpn_device proto kernel metric 50 pref medium
default dev vpn_device proto static metric 50 pref medium
```

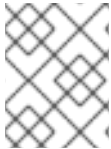
The output confirms that both the route to the remote network and the default gateway are assigned to routing table 75 and, therefore, all traffic is routed through the tunnel. If you set **ipv6.never-default true** in the VPN connection profile, a default route is not created and, therefore, not visible in this output.

Additional resources

- [CVE-2024-3661](#)

6.12. CONFIGURING IPSEC VPN CONNECTIONS BY USING RHEL SYSTEM ROLES

Configure IPsec VPN connections to establish encrypted tunnels over untrusted networks and ensure the integrity of data in transit. By using the RHEL system roles, you can automate the setup for use cases, such as connecting branch offices to headquarters.



NOTE

The **vpn** RHEL system role can only create VPN configurations that use pre-shared keys (PSKs) or certificates to authenticate peers to each other.

6.12.1. Configuring an IPsec host-to-host VPN with PSK authentication by using the **vpn** RHEL system role

A host-to-host VPN establishes an encrypted connection between two devices, allowing applications to communicate safely over an insecure network. By using the **vpn** RHEL system role, you can automate the process of creating IPsec host-to-host connections.

For authentication, a pre-shared key (PSK) is a straightforward method that uses a single, shared secret known only to the two peers. This approach is simple to configure and ideal for basic setups where ease of deployment is a priority. However, you must keep the key strictly confidential. An attacker with access to the key can compromise the connection.

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 for these nodes.

Procedure

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

```
---
- name: Configuring VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  tasks:
    - name: IPsec VPN with PSK authentication
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.vpn
      vars:
        vpn_connections:
          - hosts:
              managed-node-01.example.com:
              managed-node-02.example.com:
            auth_method: psk
```

```

    auto: start
    vpn_manage_firewall: true
    vpn_manage_selinux: true

```

The settings specified in the example playbook include the following:

hosts: *<list>*

Defines a YAML dictionary with the peers between which you want to configure a VPN. If an entry is not an Ansible managed node, you must specify its fully-qualified domain name (FQDN) or IP address in the **hostname** parameter, for example:

```

...
- hosts:
...
  external-host.example.com:
    hostname: 192.0.2.1

```

The role configures the VPN connection on each managed node. The connections are named **<peer_A>-to-<peer_B>**, for example, **managed-node-01.example.com-to-managed-node-02.example.com**. Note that the role cannot configure Libreswan on external (unmanaged) nodes. You must manually create the configuration on these peers.

auth_method: psk

Enables PSK authentication between the peers. The role uses **openssl** on the control node to create the PSK.

auto: *<startup_method>*

Specifies the startup method of the connection. Valid values are **add**, **ondemand**, **start**, and **ignore**. For details, see the **ipsec.conf(5)** man page on a system with Libreswan installed. The default value of this variable is null, which means no automatic startup operation.

vpn_manage_firewall: true

Defines that the role opens the required ports in the **firewalld** service on the managed nodes.

vpn_manage_selinux: true

Defines that the role sets the required SELinux port type on the IPsec ports.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** file on the control node.

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

- Confirm that the connections are successfully started, for example:

```
# ansible managed-node-01.example.com -m shell -a 'ipsec trafficstatus | grep
"managed-node-01.example.com-to-managed-node-02.example.com"'
...
006 #3: "managed-node-01.example.com-to-managed-node-02.example.com", type=ESP,
add_time=1741857153, inBytes=38622, outBytes=324626, maxBytes=2^63B,
id='@managed-node-02.example.com'
```

Note that this command only succeeds if the VPN connection is active. If you set the **auto** variable in the playbook to a value other than **start**, you might need to manually activate the connection on the managed nodes first.

6.12.2. Configuring an IPsec host-to-host VPN with PSK authentication and separate data and control planes by using the `vpn` RHEL system role

Use the **vpn** RHEL system role to automate the process of creating an IPsec host-to-host VPN. To enhance security by minimizing the risk of control messages being intercepted or disrupted, configure separate connections for both the data traffic and the control traffic.

A host-to-host VPN establishes a direct, secure, and encrypted connection between two devices, allowing applications to communicate safely over an insecure network, such as the internet.

For authentication, a pre-shared key (PSK) is a straightforward method that uses a single, shared secret known only to the two peers. This approach is simple to configure and ideal for basic setups where ease of deployment is a priority. However, you must keep the key strictly confidential. An attacker with access to the key can compromise the connection.

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 for these nodes.

Procedure

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

```
---
- name: Configuring VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  tasks:
    - name: IPsec VPN with PSK authentication
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.vpn
  vars:
    vpn_connections:
      - name: control_plane_vpn
        hosts:
          managed-node-01.example.com:
            hostname: 203.0.113.1 # IP address for the control plane
          managed-node-02.example.com:
            hostname: 198.51.100.2 # IP address for the control plane
```

```

    auth_method: psk
    auto: start
  - name: data_plane_vpn
    hosts:
      managed-node-01.example.com:
        hostname: 10.0.0.1 # IP address for the data plane
      managed-node-02.example.com:
        hostname: 172.16.0.2 # IP address for the data plane
    auth_method: psk
    auto: start
  vpn_manage_firewall: true
  vpn_manage_selinux: true

```

The settings specified in the example playbook include the following:

hosts: <list>

Defines a YAML dictionary with the hosts between which you want to configure a VPN. The connections are named **<name>-<IP_address_A>-to-<IP_address_B>**, for example **control_plane_vpn-203.0.113.1-to-198.51.100.2**.

The role configures the VPN connection on each managed node. Note that the role cannot configure Libreswan on external (unmanaged) nodes. You must manually create the configuration on these hosts.

auth_method: psk

Enables PSK authentication between the hosts. The role uses **openssl** on the control node to create the pre-shared key.

auto: <startup_method>

Specifies the startup method of the connection. Valid values are **add**, **ondemand**, **start**, and **ignore**. For details, see the **ipsec.conf(5)** man page on a system with Libreswan installed. The default value of this variable is null, which means no automatic startup operation.

vpn_manage_firewall: true

Defines that the role opens the required ports in the **firewalld** service on the managed nodes.

vpn_manage_selinux: true

Defines that the role sets the required SELinux port type on the IPsec ports.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** file on the control node.

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

- Confirm that the connections are successfully started, for example:

```
# ansible managed-node-01.example.com -m shell -a 'ipsec trafficstatus | grep
"control_plane_vpn-203.0.113.1-to-198.51.100.2"'
...
006 #3: "control_plane_vpn-203.0.113.1-to-198.51.100.2", type=ESP,
add_time=1741860073, inBytes=0, outBytes=0, maxBytes=2^63B, id='198.51.100.2'
```

Note that this command only succeeds if the VPN connection is active. If you set the **auto** variable in the playbook to a value other than **start**, you might need to manually activate the connection on the managed nodes first.

6.12.3. Configuring an IPsec site-to-site VPN with PSK authentication by using the `vpn` RHEL system role

A site-to-site VPN establishes an encrypted tunnel between two distinct networks, seamlessly linking them across an insecure public network. By using the **vpn** RHEL system role, you can automate the process of creating IPsec site-to-site VPN connections.

A site-to-site VPN enables devices in a branch office to access resources at a corporate headquarters just as if they were all part of the same local network.

For authentication, a pre-shared key (PSK) is a straightforward method that uses a single, shared secret known only to the two peers. This approach is simple to configure and ideal for basic setups where ease of deployment is a priority. However, you must keep the key strictly confidential. An attacker with access to the key can compromise the connection.

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 for these nodes.

Procedure

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

```
---
- name: Configuring VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  tasks:
    - name: IPsec VPN with PSK authentication
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.vpn
      vars:
        vpn_connections:
          - hosts:
              managed-node-01.example.com:
                subnets:
                  - 192.0.2.0/24
              managed-node-02.example.com:
                subnets:
                  - 198.51.100.0/24
```

```

- 203.0.113.0/24
  auth_method: psk
  auto: start
  vpn_manage_firewall: true
  vpn_manage_selinux: true

```

The settings specified in the example playbook include the following:

hosts: <list>

Defines a YAML dictionary with the gateways between which you want to configure a VPN. If an entry is not an Ansible-managed node, you must specify its fully-qualified domain name (FQDN) or IP address in the **hostname** parameter, for example:

```

...
- hosts:
  ...
  external-host.example.com:
    hostname: 192.0.2.1

```

The role configures the VPN connection on each managed node. The connections are named **<gateway_A>-to-<gateway_B>**, for example, **managed-node-01.example.com-to-managed-node-02.example.com**. Note that the role cannot configure Libreswan on external (unmanaged) nodes. You must manually create the configuration on these peers.

subnets: <yaml_list_of_subnets>

Defines subnets in classless inter-domain routing (CIDR) format that are connected through the tunnel.

auth_method: psk

Enables PSK authentication between the peers. The role uses **openssl** on the control node to create the PSK.

auto: <startup_method>

Specifies the startup method of the connection. Valid values are **add**, **ondemand**, **start**, and **ignore**. For details, see the **ipsec.conf(5)** man page on a system with Libreswan installed. The default value of this variable is null, which means no automatic startup operation.

vpn_manage_firewall: true

Defines that the role opens the required ports in the **firewalld** service on the managed nodes.

vpn_manage_selinux: true

Defines that the role sets the required SELinux port type on the IPsec ports.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** file on the control node.

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

- Confirm that the connections are successfully started, for example:

```
# ansible managed-node-01.example.com -m shell -a 'ipsec trafficstatus | grep
"managed-node-01.example.com-to-managed-node-02.example.com"'
...
006 #3: "managed-node-01.example.com-to-managed-node-02.example.com", type=ESP,
add_time=1741857153, inBytes=38622, outBytes=324626, maxBytes=2^63B,
id='@managed-node-02.example.com'
```

Note that this command only succeeds if the VPN connection is active. If you set the **auto** variable in the playbook to a value other than **start**, you might need to manually activate the connection on the managed nodes first.

6.12.4. Configuring an IPsec mesh VPN with certificate-based authentication by using the `vpn` RHEL system role

An IPsec mesh creates a fully interconnected network where every server can communicate securely and directly with every other server. By using the **vpn** RHEL system role, you can automate configuring a VPN mesh with certificate-based authentication among managed nodes.

An IPsec mesh is ideal for distributed database clusters or high-availability environments that span multiple data centers or cloud providers. Establishing a direct, encrypted tunnel between each pair of servers ensures secure communication without a central bottleneck.

For authentication, using digital certificates managed by a Certificate Authority (CA) offers a highly secure and scalable solution. Each host in the mesh presents a certificate signed by a trusted CA. This method provides strong, verifiable authentication and simplifies user management. Access can be granted or revoked centrally at the CA, and Libreswan enforces this by checking each certificate against a certificate revocation list (CRL), denying access if a certificate appears on the list.

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 for these nodes.
- You prepared a PKCS #12 file for each managed node:
 - Each file contains:
 - The private key of the server
 - The server certificate
 - The CA certificate
 - If required, intermediate certificates
 - The files are named **<managed_node_name_as_in_the_inventory>.p12**.

- The files are stored in the same directory as the playbook.
- The server certificate contains the following fields:
 - Extended Key Usage (EKU) is set to **TLS Web Server Authentication**.
 - Common Name (CN) or Subject Alternative Name (SAN) is set to the fully-qualified domain name (FQDN) of the host.
 - X509v3 CRL distribution points contain URLs to Certificate Revocation Lists (CRLs).

Procedure

1. Edit the `~/inventory` file, and append the **cert_name** variable:

```
managed-node-01.example.com cert_name=managed-node-01.example.com
managed-node-02.example.com cert_name=managed-node-02.example.com
managed-node-03.example.com cert_name=managed-node-03.example.com
```

Set the **cert_name** variable to the value of the common name (CN) field used in the certificate for each host. Typically, the CN field is set to the fully-qualified domain name (FQDN).

2. Store your sensitive variables in an encrypted file:

- a. Create the vault:

```
$ ansible-vault create ~/vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

- b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the **<key>: <value>** format:

```
pkcs12_pwd: <password>
```

- c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

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

```
- name: Configuring VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com, managed-node-03.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Install LibreSwan
      ansible.builtin.package:
        name: libreswan
        state: present

    - name: Identify the path to IPsec NSS database
      ansible.builtin.set_fact:
        nss_db_dir: "{{ '/etc/ipsec.d' if
          ansible_distribution in ['CentOS', 'RedHat']
          and ansible_distribution_major_version is version('8', '=)
          else '/etc/ipsec.d' }}"
```

```

    else '/var/lib/ipsec/nss/' }}"

- name: Locate IPsec NSS database files
  ansible.builtin.find:
    paths: "{{ nss_db_dir }}"
    patterns: "*.db"
    register: db_files

- name: Initialize IPsec NSS database if not initialized
  ansible.builtin.command:
    cmd: ipsec initnss
    when: db_files.matched == 0

- name: Copy PKCS #12 file to the managed node
  ansible.builtin.copy:
    src: "~/{{ inventory_hostname }}.p12"
    dest: "/etc/ipsec.d/{{ inventory_hostname }}.p12"
    mode: 0600

- name: Import PKCS #12 file in IPsec NSS database
  ansible.builtin.shell:
    cmd: 'pk12util -d {{ nss_db_dir }} -i /etc/ipsec.d/{{ inventory_hostname }}.p12 -W "{{
pkcs12_pwd }}"'

- name: Remove PKCS #12 file
  ansible.builtin.file:
    path: "/etc/ipsec.d/{{ inventory_hostname }}.p12"
    state: absent

- name: Opportunistic mesh IPsec VPN with certificate-based authentication
  ansible.builtin.include_role:
    name: redhat.rhel_system_roles.vpn
  vars:
    vpn_connections:
      - opportunistic: true
        auth_method: cert
        policies:
          - policy: private
            cidr: default
          - policy: private
            cidr: 192.0.2.0/24
          - policy: clear
            cidr: 192.0.2.1/32
    vpn_manage_firewall: true
    vpn_manage_selinux: true

```

The settings specified in the example playbook include the following:

opportunistic: true

Enables an opportunistic mesh among multiple hosts. The **policies** variable defines for which subnets and hosts traffic must or can be encrypted and which of them should continue using plain text connections.

auth_method: cert

Enables certificate-based authentication. This requires that you specify the nickname of each managed node's certificate in the inventory.

policies: <list_of_policies>

Defines the Libreswan policies in YAML list format.

The default policy is **private-or-clear**. To change it to **private**, the above playbook contains an according policy for the default **cidr** entry.

To prevent a loss of the SSH connection during the execution of the playbook if the Ansible control node is in the same IP subnet as the managed nodes, add a **clear** policy for the control node's IP address. For example, if the mesh should be configured for the **192.0.2.0/24** subnet and the control node uses the IP address **192.0.2.1**, you require a **clear** policy for **192.0.2.1/32** as shown in the playbook.

For details about policies, see the **ipsec.conf(5)** man page on a system with Libreswan installed.

vpn_manage_firewall: true

Defines that the role opens the required ports in the **firewalld** service on the managed nodes.

vpn_manage_selinux: true

Defines that the role sets the required SELinux port type on the IPsec ports.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** file on the control node.

4. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

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

5. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

1. On a node in the mesh, ping another node to activate the connection:

```
[root@managed-node-01]# ping managed-node-02.example.com
```

2. Confirm that the connection is active:

```
[root@managed-node-01]# ipsec trafficstatus
006 #2: "private#192.0.2.0/24"[1] ...192.0.2.2, type=ESP, add_time=1741938929,
inBytes=372408, outBytes=545728, maxBytes=2^63B, id='CN=managed-node-
02.example.com'
```

6.13. CONFIGURING IPSEC VPN CONNECTIONS BY USING NMSTATECTL

Configure IPsec VPN connections to establish encrypted tunnels over untrusted networks and ensure the integrity of data in transit. By using Nmstate, you can create IPsec VPN connections by using a declarative API.

You can use the **nmstatectl** utility to configure Libreswan IPsec VPN connections through the Nmstate API. The **nmstatectl** utility is a command-line tool to manage host networking through the declarative Nmstate API. Instead of running multiple imperative commands to configure an interface, you define the expected state in a YAML file. Nmstate then takes this definition and applies it to the system. A key advantage of this approach is an atomic result. Nmstate ensures that the resulting configuration precisely matches your YAML definition. If any part of the configuration fails to apply, it automatically rolls back all changes and prevents the system from entering an incorrect or broken network state.



NOTE

Due to the design of the **NetworkManager-libreswan** plugin, you can use **nmstatectl** only on one peer and must manually configure Libreswan on the other peer.

6.13.1. Configuring an IPsec host-to-host VPN with raw RSA key authentication by using nmstatectl

You can use the declarative Nmstate API to configure a host-to-host VPN between two devices to communicate safely over an insecure network. Nmstate ensures that the result matches the configuration file or rolls back the changes.

For authentication, RSA keys are more secure than pre-shared keys (PSKs) because their asymmetric encryption eliminates the risk of a shared secret. Using RSA keys also simplifies deployment by avoiding the need for a certificate authority (CA), while still providing strong peer-to-peer authentication.



NOTE

In general, the choice of which host is named *left* and *right* is arbitrary. However, NetworkManager always uses the term *left* for the local host and *right* for the remote host.

Prerequisites

- The remote peer runs Libreswan IPsec and is prepared for a [host-to-host](#) connection. Due to the design of the **NetworkManager-libreswan** plugin, Nmstate cannot communicate with other peers that also use this plugin for the same connection.

Procedure

1. If Libreswan is not yet installed, perform the following steps:

- a. Install the required packages:

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

- b. Restart the NetworkManager service:

```
# systemctl restart NetworkManager
```

- c. Initialize the Network Security Services (NSS) database:

■

ipsec initnss

The command creates the database in the `/var/lib/ipsec/nss/` directory.

- d. Open the IPsec ports and protocols in the firewall:

```
# firewall-cmd --permanent --add-service="ipsec"
# firewall-cmd --reload
```

2. Create an RSA key pair:

ipsec newhostkey

The **ipsec** utility stores the key pair in the NSS database.

3. Display the Certificate Key Attribute ID (CKAID) on both the left and right peers:

```
# ipsec showhostkey --list
< 1> RSA keyid: <key_id> ckaid: <ckaid>
```

You require the CKAIDs of both peers in the next steps.

4. Display the public keys:

- a. On the left peer, enter:

```
# ipsec showhostkey --left --ckaid <ckaid_of_left_peer>
# rsakey AwEAAAdKCx
leftrsasigkey=0sAwEAAAdKCxpc9db48cehzQiQD...
```

- b. On the right peer, enter:

```
# ipsec showhostkey --right --ckaid <ckaid_of_right_peer>
# rsakey AwEAAcNWC
rightrsasigkey=0sAwEAAcNWCzZO+PR1j8WbO8X...
```

The commands display the public keys with the corresponding parameters that you must use in the configuration file.

5. Create a YAML file, for example `~/ipsec-host-to-host-rsa-auth.yml`, with the following content:

```
---
interfaces:
- name: '<connection_name>'
  type: ipsec
  libreswan:
    ikev2: insist

    left: <ip_address_or_fqdn_of_left_peer>
    leftid: peer_b
    leftrsasigkey: <public_key_of_left_peer>
    leftmodecfgclient: false

    right: <ip_address_or_fqdn_of_right_peer>
```

```

rightid: peer_a
rightrsasigkey: <public_key_of_right_peer>
rightsubnet: <ip_address_of_right_peer>/32

```

The settings specified in the example include the following:

ikev2: insist

Defines the modern IKEv2 protocol as the only allowed protocol without fallback to IKEv1. This setting is mandatory in a host-to-host configuration with Nmstate.

left=<ip_address_or_fqdn_of_left_peer> and right=<ip_address_or_fqdn_of_right_peer>

Defines the IP address or DNS name of the peers.

leftid=<id> and rightid=<id>

Defines how each peer is identified during the Internet Key Exchange (IKE) negotiation process. This can be an IP address or a literal string. Note that NetworkManager interprets all values other than IP addresses as a literal string and internally adds a leading @ sign. This requires that the Libreswan peer also uses literal strings as IDs or authentication fails.

leftrsasigkey=<public_key> and rightrsasigkey=<public_key>

Specifies the public key of the peers. Use the values displayed by the **ipsec showhostkey** command in a previous step.

leftmodecfgclient: false

Disables dynamic configuration on this host. This setting is mandatory in a host-to-host configuration with Nmstate.

rightsubnet: <ip_address_of_right_peer>/32

Defines that the host can only access this peer. This setting is mandatory in a host-to-host configuration with Nmstate.

6. Apply the settings to the system:

```
# nmstatectl apply ~/ipsec-host-to-host-rsa-auth.yml
```

Verification

- Display the IPsec status:

```
# ipsec status
```

If the connection is successfully established, the output contains lines as follows:

- Phase 1 of an Internet Key Exchange version 2 (IKEv2) negotiation has been successfully completed:

```
000 #1: "<connection_name>":500 STATE_V2_ESTABLISHED_IKE_SA (established
IKE SA); REKEY in 27935s; REPLACE in 28610s; newest; idle;
```

The Security Association (SA) is now ready to negotiate the actual data encryption tunnels, known as child SAs or Phase 2 SAs.

- A child SA has been established:

```
000 #2: "<connection_name>":500 STATE_V2_ESTABLISHED_CHILD_SA (established
Child SA); REKEY in 27671s; REPLACE in 28610s; IKE SA #1; idle;
```

■

This is the actual tunnel that your data traffic flows through.

Troubleshooting

- To display the actual configuration NetworkManager passes to Libreswan, enter:

```
# nmcli connection export <connection_name>
```

The output can help to identify deviating settings, such as IDs and keys, when you compare them with the Libreswan configuration on the remote host.

Next steps

- If you use this host in a network with DHCP or Stateless Address Autoconfiguration (SLAAC), the connection can be vulnerable to being redirected. For details and mitigation steps, see [Assigning a VPN connection to a dedicated routing table to prevent the connection from bypassing the tunnel](#).

6.13.2. Configuring an IPsec site-to-site VPN with raw RSA key authentication by using nmstatectl

You can use the declarative Nmstate API to configure a site-to-site VPN between two distinct networks, seamlessly linking them across an insecure network. Nmstate ensures that the result matches the configuration file or rolls back the changes.

For authenticating the gateway devices, RSA keys are more secure than pre-shared keys (PSKs) because their asymmetric encryption eliminates the risk of a shared secret. Using RSA keys also simplifies deployment by avoiding the need for a certificate authority (CA), while still providing strong peer-to-peer authentication.



NOTE

In general, the choice which host is named *left* and *right* is arbitrary. However, NetworkManager always uses the term *left* for the local host and *right* for the remote host.

Prerequisites

- The remote gateway runs Libreswan IPsec and is prepared for a [site-to-site](#) connection. Due to the design of the **NetworkManager-libreswan** plugin, Nmstate cannot communicate with other peers that also use this plugin for the same connection.

Procedure

1. If Libreswan is not yet installed, perform the following steps:

- a. Install the required packages:

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

- b. Restart the NetworkManager service:

```
# systemctl restart NetworkManager
```

- c. Initialize the Network Security Services (NSS) database:

```
# ipsec initnss
```

The command creates the database in the `/var/lib/ipsec/nss/` directory.

- d. Open the IPsec ports and protocols in the firewall:

```
# firewall-cmd --permanent --add-service="ipsec"
# firewall-cmd --reload
```

2. Create an RSA key pair:

```
# ipsec newhostkey
```

The **ipsec** utility stores the key pair in the NSS database.

3. Display the Certificate Key Attribute ID (CKAID) on both the left and right peer:

```
# ipsec showhostkey --list
< 1> RSA keyid: <key_id> ckaid: <ckaid>
```

You require the CKAIDs of both peers in the following steps.

4. Display the public keys:

- a. On the left peer, enter:

```
# ipsec showhostkey --left --ckaid <ckaid_of_left_peer>
# rsakey AwEAAAdKCx
lefttrsasigkey=0sAwEAAAdKCxpc9db48cehzQiQD...
```

- b. On the right peer, enter:

```
# ipsec showhostkey --right --ckaid <ckaid_of_right_peer>
# rsakey AwEAAcNWC
righttrsasigkey=0sAwEAAcNWCzZO+PR1j8WbO8X...
```

The commands display the public keys with the corresponding parameters that you must use in the configuration file.

5. Create a YAML file, for example `~/ipsec-site-to-site-rsa-auth.yml`, with the following content:

```
---
interfaces:
- name: '<connection_name>'
  type: ipsec
  libreswan:
    ikev2: insist

    left: <ip_address_or_fqdn_of_left_peer>
    leftid: peer_b
    lefttrsasigkey: <public_key_of_left_peer>
```

```

leftmodecfgclient: false
leftsubnet: 198.51.100.0/24

right: <ip_address_or_fqdn_of_right_peer>
rightid: peer_a
rightrsasigkey: <public_key_of_right_peer>
rightsubnet: 192.0.2.0/24

```

The settings specified in the example include the following:

ikev2: insist

Defines the modern IKEv2 protocol as the only allowed protocol without fallback to IKEv1. This setting is mandatory in a site-to-site configuration with Nmstate.

left=<ip_address_or_fqdn_of_left_peer> and right=<ip_address_or_fqdn_of_right_peer>

Defines the IP address or DNS name of the peers.

leftid=<id> and rightid=<id>

Defines how each peer is identified during the Internet Key Exchange (IKE) negotiation process. This can be an IP address or a literal string. Note that NetworkManager interprets all values other than IP addresses as a literal string and internally adds a leading @ sign. This requires that the Libreswan peer also uses literal strings as IDs or authentication fails.

leftrsasigkey=<public_key> and rightrsasigkey=<public_key>

Specifies the public key of the peers. Use the values displayed by the **ipsec showhostkey** command in a previous step.

leftmodecfgclient: false

Disables dynamic configuration on this host. This setting is mandatory in a site-to-site configuration with Nmstate.

leftsubnet=<subnet> and rightsubnet=<subnet>

Defines subnets in classless inter-domain routing (CIDR) format that are connected through the tunnel.

6. 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

```

7. Apply the settings to the system:

```

# nmstatectl apply ~/ipsec-site-to-site-rsa-auth.yml

```

Verification

1. Display the IPsec status:

```

# ipsec status

```

If the connection is successfully established, the output contains lines as follows:

- Phase 1 of an Internet Key Exchange version 2 (IKEv2) negotiation has been successfully completed:

```
000 #1: "<connection_name>":500 STATE_V2_ESTABLISHED_IKE_SA (established
IKE SA); REKEY in 27935s; REPLACE in 28610s; newest; idle;
```

The Security Association (SA) is now ready to negotiate the actual data encryption tunnels, known as child SAs or Phase 2 SAs.

- A child SA has been established:

```
000 #2: "<connection_name>":500 STATE_V2_ESTABLISHED_CHILD_SA (established
Child SA); REKEY in 27671s; REPLACE in 28610s; IKE SA #1; idle;
```

This is the actual tunnel that your data traffic flows through.

2. From a client in the local subnet, ping a client in the remote subnet.

Troubleshooting

- To display the actual configuration NetworkManager passes to Libreswan, enter:

```
# nmcli connection export <connection_name>
```

The output can help to identify deviating settings, such as IDs and keys, when you compare them with the Libreswan configuration on the remote host.

Next steps

- If you use this host in a network with DHCP or Stateless Address Autoconfiguration (SLAAC), the connection can be vulnerable to being redirected. For details and mitigation steps, see [Assigning a VPN connection to a dedicated routing table to prevent the connection from bypassing the tunnel](#).

6.13.3. Configuring a client to connect to an IPsec VPN gateway by using nmstatectl

To access resources on a remote private network, users must first configure an IPsec VPN connection. By using Nmstate, you can create the connection with an existing Libreswan IPsec gateway by using a declarative API.



NOTE

In general, the choice of which host is named *left* and *right* is arbitrary. However, NetworkManager always uses the term *left* for the local host and *right* for the remote host.

Prerequisites

- The remote gateway runs Libreswan IPsec and is prepared for a [host-to-site](#) connection with certificate-based authentication.
Due to the design of the **NetworkManager-libreswan** plugin, Nmstate cannot communicate with other peers that also use this plugin for the same connection.
- The PKCS#12 file **~/file.p12** exists on the client with the following contents:
 - The private key of the user

- The user certificate
- The CA certificate
- If required, intermediate certificates

For details about creating a private key and certificate signing request (CSR), as well as about requesting a certificate from a CA, see your CA's documentation.

- The Extended Key Usage (EKU) in the certificate is set to **TLS Web Client Authentication**.

Procedure

1. If Libreswan is not yet installed:

- a. Install the required packages:

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

- b. Restart the NetworkManager service:

```
# systemctl restart NetworkManager
```

- c. Initialize the Network Security Services (NSS) database:

```
# ipsec initnss
```

The command creates the database in the `/var/lib/ipsec/nss/` directory.

- d. Open the IPsec ports and protocols in the firewall:

```
# firewall-cmd --permanent --add-service="ipsec"
# firewall-cmd --reload
```

2. Import the PKCS #12 file into the NSS database:

```
# ipsec import ~/file.p12
Enter password for PKCS12 file: <password>
pk12util: PKCS12 IMPORT SUCCESSFUL
correcting trust bits for Example-CA
```

3. Display the nicknames of the user and CA certificates:

```
# certutil -L -d /var/lib/ipsec/nss/
Certificate Nickname      Trust Attributes
                        SSL,S/MIME,JAR/XPI

user                      U,u,u
Example-CA                CT,,
...
```

You require this information in the Nmstate YAML file.

4. Create a YAML file, for example, `~/ipsec-host-to-site-cert-auth.yml`, with the following content:

```
---
interfaces:
- name: '<connection_name>'
  type: ipsec
  libreswan:
    ikev2: insist

    left: <ip_address_or_fqdn_of_left_peer>
    leftid: '%fromcert'
    leftcert: <user_certificate_nickname>

    right: <ip_address_or_fqdn_of_right_peer>
    rightid: '%fromcert'
    rightsubnet: 192.0.2.0/24
```

The settings specified in the example include the following:

ikev2: insist

Defines the modern IKEv2 protocol as the only allowed protocol without fallback to IKEv1. This setting is mandatory in a host-to-site configuration with Nmstate.

left=<ip_address_or_fqdn_of_left_peer> and right=<ip_address_or_fqdn_of_right_peer>

Defines the IP address or DNS name of the peers.

leftid=%fromcert and rightid=%fromcert

Configures Libreswan to retrieve the identity from the distinguished name (DN) field of the certificate.

leftcert="<server_certificate_nickname>"

Sets the nickname of the server's certificate used in the NSS database.

rightsubnet: <subnet>

Defines the subnet in classless inter-domain routing (CIDR) format that is connected to the gateway.

5. Apply the settings to the system:

```
# nmstatectl apply ~/ipsec-host-to-site-cert-auth.yml
```

Verification

- Establish a connection to a host in the remote network or ping it.

Troubleshooting

- To display the actual configuration NetworkManager passes to Libreswan, enter:

```
# nmcli connection export <connection_name>
```

The output can help to identify deviating settings, such as IDs and keys, when you compare them with the Libreswan configuration on the remote host.

Next steps

- If you use this host in a network with DHCP or Stateless Address Autoconfiguration (SLAAC), the connection can be vulnerable to being redirected. For details and mitigation steps, see [Assigning a VPN connection to a dedicated routing table to prevent the connection from bypassing the tunnel](#).

6.14. TROUBLESHOOTING IPSEC CONFIGURATIONS

Diagnosing IPsec configuration failures can be challenging, because issues can be caused by mismatched settings, firewall rules, and kernel-level errors. The following information provides a systematic approach to resolving common problems with IPsec VPN connections.

6.14.1. Basic connection issues

Problems with VPN connections often occur due to mismatched configurations between the endpoints.

To confirm that an IPsec connection is established, enter:

```
# 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=198.51.100.1/32
```

For a successful connection, the command shows an entry with the connection's name and details. If the output is empty, the tunnel is not established.

6.14.2. Firewall-related problems

Diagnose IPsec connection failures caused by firewall-related problems.

If a firewall on an endpoint or an intermediate route drops Internet Key Exchange version 2 (IKEv2) packets, the **ipsec up** command times out and shows repeated *retransmission* messages:

```
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 response
...
```

You can use the **tcpdump** utility to capture traffic and verify if IKE or IPsec packets are dropped by a firewall, for example:

```
# tcpdump -i <interface> -n -n esp or udp port 500 or udp port 4500 or tcp port 4500
```

Note that **tcpdump** is of limited use for diagnosing other types of IPsec problems because the IKE protocol is encrypted.

6.14.3. Mismatched Configurations

VPN connections fail if the endpoints are not configured with matching Internet Key Exchange (IKE) versions, algorithms, IP address ranges, or pre-shared keys (PSK). If you identify a mismatch, you must align the settings on both endpoints to resolve the issue.

Remote Peer Not Running IKE/IPsec

If the connection was refused, an ICMP error is displayed:

```
# ipsec 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)]
```

Mismatched IKE Algorithms

The connection fails with a **NO_PROPOSAL_CHOSEN** notification during the initial setup:

```
# ipsec 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
```

Mismatched IPsec Algorithms

The connection fails with a **NO_PROPOSAL_CHOSEN** error after the initial exchange:

```
# ipsec 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
```

Mismatched IP Address Ranges (IKEv2)

The remote peer responds with a **TS_UNACCEPTABLE** error:

```
# ipsec 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
```

Mismatched IP Address Ranges (IKEv1)

The connection times out during quick mode, with a message indicating the peer did not accept the proposal:

```
# ipsec 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
```

Mismatched PSK (IKEv2)

The peer rejects the connection with an **AUTHENTICATION_FAILED** error:

```
# ipsec 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
```

Mismatched PSK (IKEv1)

The hash payload does not match, making the IKE message unreadable and resulting in an **INVALID_HASH_INFORMATION** error:

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

6.14.4. MTU issues

Diagnose intermittent IPsec connection failures caused by Maximum Transmission Unit (MTU) issues. Encryption increases packet size, leading to fragmentation and lost data when packets exceed the network's MTU, often seen with larger data transfers.

A common symptom is that small packets, for example pings, work correctly, but larger packets, such as an SSH session, freeze after the login. To fix the problem, lower the MTU for the tunnel by adding the **mtu=1400** option to the configuration file.

6.14.5. NAT conflicts

Resolve NAT conflicts that occur when an IPsec host also acts as a NAT router. Incorrect NAT application can translate source IP addresses before encryption, causing packets to be sent unencrypted over the network.

For example, if the source IP address of the packet is translated by a masquerade rule before IPsec encryption is applied, the packet's source no longer matches the IPsec policy, and Libreswan sends it unencrypted over the network.

To solve this problem, add a firewall rule that excludes traffic between the IPsec subnets from NAT. This rule should be inserted at the beginning of the **POSTROUTING** chain to ensure it is processed before the general NAT rule.

Example 6.1. Solution by using the **nftables** framework

The following example uses **nftables** to set up a basic NAT environment that excludes traffic between the 192.0.2.0/24 and 198.51.100.0/24 subnets from address translation:

```
# nft add table ip nat
# nft add chain ip nat postrouting { type nat hook postrouting priority 100 \; }
# nft add rule ip nat postrouting ip saddr 192.0.2.0/24 ip daddr 198.51.100.0/24 return
```

6.14.6. Kernel-level IPsec issues

Troubleshoot kernel-level IPsec issues when a VPN tunnel appears established but no traffic flows. In this case, inspect the kernel's IPsec state to check if the tunnel policies and cryptographic keys were correctly installed.

This process involves checking two components:

- The Security Policy Database (SPD): The rule that instructs the kernel what traffic to encrypt.
- The Security Association Database (SAD): The keys that instruct the kernel how to encrypt that traffic.

First, check if the correct policy exists in the SPD:

```
# ip xfrm policy
src 192.0.2.1/32 dst 10.0.0.0/8
dir out priority 666 ptype main
tmpl src 198.51.100.13 dst 203.0.113.22
proto esp reqid 16417 mode tunnel
```

The output should contain the policies matching your **leftsubnet** and **rightsubnet** parameters with both in and out directions. If you do not see a policy for your traffic, Libreswan failed to create the kernel rule, and traffic is not encrypted.

If the policy exists, check if it has a corresponding set of keys in the SAD:

```
# ip xfrm state
src 203.0.113.22 dst 198.51.100.13
proto esp spi 0xa78b3fdb reqid 16417 mode tunnel
auth-trunc hmac(sha1) 0x3763cd3b... 96
enc cbc(aes) 0xd9dba399...
```



WARNING

This command displays private cryptographic keys. Do not share this output, because attackers can use it to decrypt your VPN traffic.

If a policy exists but you see no corresponding state with the same **reqid**, it typically means the Internet Key Exchange (IKE) negotiation failed. The two VPN endpoints could not agree on a set of keys.

For more detailed diagnostics, use the **-s** option with either of the commands. This option adds traffic counters, which can help you identify if the kernel processes packets by a specific rule.

6.14.7. Kernel IPsec subsystem bugs

A defect in the kernel's IPsec subsystem can cause it to lose sync with the IKE daemon. This can lead to discrepancies between negotiated security associations and actual IPsec policy enforcement, disrupting secure network communication.

To check for kernel-level errors, display the transform (XFRM) statistics:

```
# cat /proc/net/xfrm_stat
```

If any of the counters in the output, such as **XfrmInError**, show a nonzero value, it indicates a problem with the kernel subsystem. In this case, open a [support case](#), and attach the output of the command along with the corresponding IKE logs.

6.14.8. Displaying Libreswan logs

Display Libreswan logs to diagnose and troubleshoot IPsec service events and issues. Access the journal for the **ipsec** service to gain insights into connection status and potential problems.

To display the journal, enter:

```
# journalctl -xeu ipsec
```

If the default logging level does not provide enough details, enable comprehensive debug logging by adding the following settings to the **config setup** section in the **/etc/ipsec.conf** file:

```
plutodebug=all  
logfile=/var/log/pluto.log
```

Because debug logging can produce many entries, redirecting the messages to a dedicated log file can prevent the **journald** and **systemd** services from rate-limiting the messages.

CHAPTER 7. USING MACSEC TO ENCRYPT LAYER-2 TRAFFIC IN THE SAME PHYSICAL NETWORK

Configure MACsec to encrypt Layer 2 traffic within the same physical network. This helps secure point-to-point communication between directly connected devices, such as hosts or switches.

7.1. HOW MACSEC INCREASES SECURITY

Use Media Access Control security (MACsec) to encrypt and authenticate LAN traffic at layer 2. This process helps eliminate the need to encrypt individual services at layer 7.

Media Access Control security secures point-to-point communication between devices. For example, you can configure MACsec on the two hosts connecting your branch and central offices over a Metro-Ethernet connection to increase security.

MACsec is a layer-2 protocol that encrypts and authenticates all LAN traffic. It uses a pre-shared key to establish connections. To change this key, you must update the NetworkManager configuration on all hosts that use MACsec.

MACsec secures different traffic types over the Ethernet links, including:

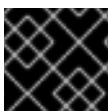
- Dynamic host configuration protocol (DHCP)
- address resolution protocol (ARP)
- IPv4 and IPv6 traffic
- Any traffic over IP such as TCP or UDP

A MACsec connection uses an Ethernet device, such as an Ethernet network card, Virtual Local Area Network (VLAN), or tunnel device, as a parent. You can either set an IP configuration only on the MACsec device to communicate with other hosts only by 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.

That is, you can configure MACsec for two common scenarios:

- Host-to-host
- Host-to-switch and switch-to-other-hosts



IMPORTANT

You can use MACsec only between hosts being in the same physical or virtual LAN.

Using the MACsec security standard for securing communication at the link layer, also known as layer 2 of the Open Systems Interconnection (OSI) model provides the following notable benefits:

- Encryption at layer 2 eliminates the need for encrypting individual services at layer 7. This reduces the overhead associated with managing a large number of certificates for each endpoint on each host.
- Point-to-point security between directly connected network devices such as routers and switches.
- No changes needed for applications and higher-layer protocols.

Additional resources

- [MACsec: a different solution to encrypt network traffic](#)

7.2. CONFIGURING A MACSEC CONNECTION BY USING **nmcli**

You can use the **nmcli** utility to configure Ethernet interfaces to use MACsec. 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
```

Additional resources

- [MACsec: a different solution to encrypt network traffic](#)

7.3. CONFIGURING A MACSEC CONNECTION BY USING NMSTATECTL

You can use the declarative Nmstate API to configure Ethernet interfaces to use MACsec. Nmstate ensures that the result matches the configuration file or rolls back the changes.

Prerequisites

- A physical or virtual Ethernet Network Interface Controller (NIC) exists in the server configuration.
- The **nmstate** package is installed.

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 that you want to connect over a MACsec connection, complete the following steps:

- a. Create a YAML file, for example **create-macsec-connection.yml**, with the following settings:

```
---
routes:
  config:
    - destination: 0.0.0.0/0
      next-hop-interface: macsec0
      next-hop-address: 192.0.2.2
      table-id: 254
    - destination: 192.0.2.2/32
      next-hop-interface: macsec0
      next-hop-address: 0.0.0.0
      table-id: 254
  dns-resolver:
    config:
      search:
        - example.com
      server:
        - 192.0.2.200
        - 2001:db8:1::ffbb
  interfaces:
    - name: macsec0
      type: macsec
      state: up
      ipv4:
        enabled: true
        address:
          - ip: 192.0.2.1
            prefix-length: 32
      ipv6:
        enabled: true
        address:
          - ip: 2001:db8:1::1
            prefix-length: 64
  macsec:
    encrypt: true
    base-iface: enp0s1
    mka-cak: 50b71a8ef0bd5751ea76de6d6c98c03a
    mka-ckn: f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
    port: 0
    validation: strict
    send-sci: true
```

- b. Use the CAK and CKN generated in the previous step in the **mka-cak** and **mka-ckn** parameters. The values must be the same on every host in the MACsec-protected network.
- c. Optional: In the same YAML configuration file, you can also configure the following settings:
 - A static IPv4 address - **192.0.2.1** with the **/32** subnet mask
 - A static IPv6 address - **2001:db8:1::1** with the **/64** subnet mask
 - An IPv4 default gateway - **192.0.2.2**
 - An IPv4 DNS server - **192.0.2.200**
 - An IPv6 DNS server - **2001:db8:1::ffbb**
 - A DNS search domain - **example.com**
3. Apply the settings to the system:

```
# nmstatectl apply create-macsec-connection.yml
```

Verification

1. Display the current state in YAML format:

```
# nmstatectl show macsec0
```

2. Verify that the traffic is encrypted:

```
# tcpdump -nn -i enp0s1
```

3. Optional: Display the unencrypted traffic:

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

4. Display MACsec statistics:

```
# ip macsec show
```

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

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

Additional resources

- [MACsec: a different solution to encrypt network traffic](#)

CHAPTER 8. SECURING NETWORK SERVICES

Learn how to harden and monitor network services to protect your system against various risks. Turning off unused services helps limit exposure.

Red Hat Enterprise Linux supports many different types of network servers. Their network services can expose the system to various kinds of attacks, such as denial-of-service (DoS), distributed denial-of-service (DDoS), script vulnerability, and buffer overflow attacks.

To increase system security against attacks, it is crucial to monitor the active network services you use. For example, when a network service runs on a machine, its daemon listens for connections on network ports, which can reduce security. To limit network exposure to attacks, turn off all unused services, ports, and networking capabilities.

8.1. SECURING THE RPCBIND SERVICE

You can secure **rpcbind** by restricting access to all networks and defining specific exceptions by using firewall rules on the server.

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**.



NOTE

- The **rpcbind** service is required on **NFSv3** servers.
- The **rpcbind** service is not required on **NFSv4**.

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

- 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

- [Configuring an NFSv4-only server](#)
- [Using and configuring firewalld](#)

8.2. 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 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 by 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

- 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](#)

8.3. SECURING THE NFS SERVICE

Learn how to secure Network File System (NFS) by using Kerberos authentication and encryption for file system operations. Proper security configuration helps protect remote hosts mounting file systems over a network.

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, 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.

8.3.1. Export options for securing an NFS server

Use export options in the **/etc/exports** file to define which hosts can access exported file systems and the permissions they hold. This helps control access and limits security risks.

The NFS server determines a list of directories and hosts, along with which file systems to export to which hosts, in the **/etc/exports** file.

You can use the following export options on the **/etc/exports** file:

ro

Exports the NFS volume as read-only.

rw

Controls permission for read and write requests on the NFS volume. Use this option cautiously, as granting write access increases the risk of attacks. If your scenario requires mounting the directories with the **rw** option, make sure they are not writable for all users to reduce possible risks.

root_squash

Maps 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

Turns 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

Restricts 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, restricting to reserved ports is of limited value; it is better to rely on Kerberos, firewalls, and limiting exports to particular clients.

See the **exports(5)** and **nfs(5)** man pages on your system for more information.

**WARNING**

Extra spaces in the syntax of the **/etc/exports** file can lead to significant 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.

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.



WARNING

Export an entire file system because exporting a subdirectory of a file system is not secure. An attacker might access the unexported part of a partially-exported file system.

Additional resources

- [Using Ansible to automount NFS shares for IdM users](#)

8.3.2. Mount options for securing an NFS client

You can apply mount options when configuring an NFS client to help enforce stronger security. These settings ensure that the client/server communication uses required security protocols such as Kerberos.

The following options to the **mount** command might 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 in opposition 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 method for accessing files on the mount point. Valid security methods 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

- [Frequently-used NFS mount options](#)

8.3.3. 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

- List the active ports and RPC programs on the NFS server:

```
$ rpcinfo -p
```

8.4. 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, make sure it is configured securely.

Red Hat Enterprise Linux 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.

8.4.1. Securing the FTP greeting banner

When a user connects to the FTP service, the server displays a greeting banner that, by default, includes version information. Attackers might use this information to identify vulnerabilities in the system. You can hide 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.
```

8.4.2. 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, configure the FTP server to permit anonymous users to upload files to a specific directory and block them from reading data. In the following example procedure, the anonymous user must be able to upload files in the directory owned by the **root** user, but not change it.

Procedure

1. 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
```

2. Add the following lines to the **/etc/vsftpd/vsftpd.conf** file:

```
anon_upload_enable=YES
anonymous_enable=YES
```

3. Optional: If your system has SELinux enabled and enforcing, enable SELinux boolean attributes **allow_ftpd_anon_write** and **allow_ftpd_full_access**.



WARNING

Configuring directories for anonymous read and write access increases risk, because the server might become a repository for stolen software.

8.4.3. 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.

8.5. SECURING HTTP SERVERS

Harden your HTTP servers, such as Apache and Nginx, to mitigate security risks. This involves configuring security options in the main configuration files and checking that scripts run correctly.

8.5.1. 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** 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 run commands. Changing this can allow an attacker to enter commands on the system.

8.5.1.1. 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

- [Setting up the Apache HTTP web server](#)
- [Customizing the SELinux policy for the Apache HTTP server in a non-standard configuration](#)

8.5.2. Nginx server configuration hardening

Harden your Nginx HTTP and proxy server configuration by adjusting security options. This helps protect your system against common web application vulnerabilities.

Nginx is a high-performance HTTP and proxy server. You can harden your Nginx configuration with the following configuration options:

- 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 TLS 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 TLS 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

- [Planning and implementing TLS](#)
- [Mozilla SSL Configuration Generator](#)
- [SSL Server Test](#)

8.6. SECURING POSTGRESQL BY LIMITING ACCESS TO AUTHENTICATED LOCAL USERS

Secure your PostgreSQL database by configuring client authentication to limit access only to authenticated local users. This reduces the risks of unauthorized access and attacks.

PostgreSQL is an object-relational database management system (DBMS). In Red Hat Enterprise Linux, PostgreSQL is provided by the **postgresql-server** package.

The **pg_hba.conf** configuration file, stored in the database cluster's data directory, specifies the client authentication settings. The following procedure details how to configure PostgreSQL for host-based authentication.

Procedure

1. Install PostgreSQL:

```
# dnf 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.

8.7. SECURING THE MEMCACHED SERVICE

To secure the Memcached caching service against denial-of-service (DoS) attacks and unauthorized access, configure it to accept only local traffic and enable user authentication. This prevents DDoS amplification and ensures only authorized clients access stored data.

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 that uses 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, you do not need to expose the **memcached** service to the public internet. Public exposure might cause security problems, making it possible for remote attackers to leak or modify information stored in Memcached.

8.7.1. Memcached hardening against DDoS attacks

Harden the Memcached service against distributed denial-of-service (DDoS) attacks. This helps prevent attackers from overwhelming the service and degrading performance.

To mitigate security risks, perform as many of the following steps as applicable for your configuration:

- 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.

8.8. SECURING THE POSTFIX SERVICE

Secure the Postfix mail transfer agent by configuring it to use encryption and applying settings that mitigate risks from various attacks. This involves configuring SMTP Authentication (AUTH) using SASL and setting limits to reduce vulnerability to denial-of-service (DoS) attacks.

Postfix is a mail transfer agent (MTA) that uses the Simple Mail Transfer Protocol (SMTP) to deliver electronic messages between other MTAs and to email clients or delivery agents. Although MTAs can encrypt traffic between one another, they might not do so by default.

8.8.1. Reducing Postfix network-related security risks

Reduce Postfix security risks related to the network by adjusting configuration options to restrict access.

To reduce the risk of attackers invading your system through the network, perform as many of the following tasks as possible:

- Do not share the **/var/spool/postfix/** mail spool directory on a Network File System (NFS) shared volume. NFSv2 and NFSv3 do not maintain control over user and group IDs. Therefore, if two or more users have the same UID, they can receive and read each other's mail, which is a security risk.



NOTE

This rule does not apply to NFSv4 using Kerberos, because the **SECRPC_GSS** kernel module does not use UID-based authentication. However, to reduce the security risks, you should not put the mail spool directory on NFS shared volumes.

- To reduce the probability of Postfix server exploits, mail users must access the Postfix server using an email program. Do not allow shell accounts on the mail server, and set all user shells in the **/etc/passwd** file to **/sbin/nologin** (with the possible exception of the **root** user).
- To protect Postfix from a network attack, it is set up to only listen to the local loopback address by default. You can verify this by viewing the **inet_interfaces = localhost** line in the **/etc/postfix/main.cf** file. This ensures that Postfix only accepts mail messages (such as **cron** job reports) from the local system and not from the network. This is the default setting and protects Postfix from a network attack. To remove the localhost restriction and allow Postfix to listen on all interfaces, set the **inet_interfaces** parameter to **all** in **/etc/postfix/main.cf**.

8.8.2. Postfix configuration options for limiting DoS attacks

You can limit denial-of-service (DoS) attacks by configuring certain Postfix options. This involves setting strict rate and message-size limits to protect the server from being flooded with traffic.

An attacker can flood the server with traffic or send information that triggers a crash, causing a denial-of-service (DoS) attack. You can configure your system to reduce the risk of such attacks by setting limits in the **/etc/postfix/main.cf** file. You can change the value of the existing directives, or you can add new directives with custom values in the **<directive> = <value>** format.

Use the following list of directives for limiting DoS attacks:

smtpd_client_connection_rate_limit

Limits the maximum number of connection attempts any client can make to this service per time unit. The default value is **0**, which means a client can make as many connections per time unit as Postfix can accept. By default, the directive excludes clients in trusted networks.

anvil_rate_time_unit

Defines a time unit to calculate the rate limit. The default value is **60** seconds.

smtpd_client_event_limit_exceptions

Excludes clients from the connection and rate limit commands. By default, the directive excludes clients in trusted networks.

smtpd_client_message_rate_limit

Defines the maximum number of message deliveries from client to request per time unit (regardless of whether or not Postfix actually accepts those messages).

default_process_limit

Defines the default maximum number of Postfix child processes that provide a given service. You can ignore this rule for specific services in the **master.cf** file. By default, the value is **100**.

queue_minfree

Defines the minimum amount of free space required to receive mail in the queue file system. The directive is currently used by the Postfix SMTP server to decide if it accepts any mail at all. By default, the Postfix SMTP server rejects **MAIL FROM** commands when the amount of free space is less than 1.5 times the **message_size_limit**. To specify a higher minimum free space limit, specify a **queue_minfree** value that is at least 1.5 times the **message_size_limit**. By default, the **queue_minfree** value is **0**.

header_size_limit

Defines the maximum amount of memory in bytes for storing a message header. If a header is large, it discards the excess header. By default, the value is **102400** bytes.

message_size_limit

Defines the maximum size of a message, including the envelope information, in bytes. By default, the value is **10240000** bytes.

8.8.3. Configuring Postfix to use SASL

You can configure the Postfix mail transfer agent to use Simple Authentication and Security Layer (SASL). This strengthens authentication when sending and receiving electronic messages.

Postfix supports SASL-based SMTP Authentication (AUTH). SMTP AUTH is an extension of the Simple Mail Transfer Protocol. Currently, the Postfix SMTP server supports the SASL implementations in the following ways:

Dovecot SASL

The Postfix SMTP server can communicate with the Dovecot SASL implementation by using either a UNIX-domain socket or a TCP socket. Use this method if Postfix and Dovecot applications are running on separate machines.

Cyrus SASL

When enabled, SMTP clients must authenticate with the SMTP server by using an authentication method supported and accepted by both the server and the client.

Prerequisites

- The **dovecot** package is installed on the system

Procedure

1. Set up Dovecot:
 - a. Include the following lines in the **/etc/dovecot/conf.d/10-master.conf** file:

```
service auth {
    unix_listener /var/spool/postfix/private/auth {
        mode = 0660
        user = postfix
        group = postfix
    }
}
```

The previous example uses UNIX-domain sockets for communication between Postfix and Dovecot. The example also assumes default Postfix SMTP server settings, which include the mail queue located in the **/var/spool/postfix/** directory, and the application running under the **postfix** user and group.

- b. Optional: Set up Dovecot to listen for Postfix authentication requests through TCP:

```
service auth {  
  inet_listener {  
    port = <port_number>  
  }  
}
```

- c. Specify the method that the email client uses to authenticate with Dovecot by editing the **auth_mechanisms** parameter in **/etc/dovecot/conf.d/10-auth.conf** file:

```
auth_mechanisms = plain login
```

The **auth_mechanisms** parameter supports different plain text and non-plain text authentication methods.

2. Set up Postfix by modifying the **/etc/postfix/main.cf** file:

- a. Enable SMTP Authentication on the Postfix SMTP server:

```
smtpd_sasl_auth_enable = yes
```

- b. Enable the use of Dovecot SASL implementation for SMTP Authentication:

```
smtpd_sasl_type = dovecot
```

- c. Provide the authentication path relative to the Postfix queue directory. Note that the use of a relative path ensures that the configuration works regardless of whether the Postfix server runs in **chroot** or not:

```
smtpd_sasl_path = private/auth
```

This step uses UNIX-domain sockets for communication between Postfix and Dovecot.

To configure Postfix to look for Dovecot on a different machine in case you use TCP sockets for communication, use configuration values similar to the following:

```
smtpd_sasl_path = inet: <IP_address> : <port_number>
```

In the previous example, replace the **<IP_address>** with the IP address of the Dovecot machine and **<port_number>** with the port number specified in Dovecot's **/etc/dovecot/conf.d/10-master.conf** file.

- d. Specify SASL mechanisms that the Postfix SMTP server makes available to clients. Note that you can specify different mechanisms for encrypted and unencrypted sessions.

```
smtpd_sasl_security_options = noanonymous, noplaintext  
smtpd_sasl_tls_security_options = noanonymous
```

The previous directives specify that during unencrypted sessions, no anonymous authentication is allowed, and no mechanisms that transmit unencrypted user names or passwords are allowed. For encrypted sessions that use TLS, only non-anonymous authentication mechanisms are allowed.

Additional resources

- [Postfix SMTP server policy - SASL mechanism properties](#)
- [Postfix and Dovecot SASL](#)
- [Configuring SASL authentication in the Postfix SMTP server](#)