Red Hat Enterprise Linux 9

Securing networks

Configuring secured networks and network communication

Red Hat Enterprise Linux 9 Securing networks

Configuring secured networks and network communication

Legal Notice

Abstract

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

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

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

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

Submitting feedback through Jira (account required)

1. Log in to the Jira website.

2. Click **Create** in the top navigation bar

3. Enter a descriptive title in the **Summary** field.

4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.

5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. USING SECURE COMMUNICATIONS BETWEEN TWO SYSTEMS WITH OPENSSH

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

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

1.1. SSH AND OPENSSH

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

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

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

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

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

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

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

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

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

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

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

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

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

NOTE

In RHEL 9, 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 **-O** option to force use of the original SCP/RCP protocol.

For additional information, see the OpenSSH SCP protocol deprecation in Red Hat Enterprise Linux 9 article.

Two versions of SSH currently exist: version 1, and the newer version 2. The OpenSSH suite in RHEL supports only SSH version 2. It has an enhanced key-exchange algorithm that is not vulnerable to exploits known in version 1.

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

The OpenSSH suite uses two sets of configuration files: one for client programs (that is, **ssh**, **scp**, and **sftp**), and another for the server (the **sshd** daemon).

System-wide SSH configuration information is stored in the **/etc/ssh/** directory. User-specific SSH configuration information is stored in ~/**.ssh/** in the user's home directory. For a detailed list of OpenSSH configuration files, see the **FILES** section in the **sshd(8)** man page.

Additional resources

- Man pages listed by using the **man -k ssh** command

- Using system-wide cryptographic policies

1.2. CONFIGURING AND STARTING AN OPENSSH SERVER

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

Prerequisites

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

Procedure

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

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

2. To specify different addresses than the default **0.0.0.0** (IPv4) or **::** (IPv6) for the **ListenAddress** directive in the **/etc/ssh/sshd_config** configuration file and to use a slower dynamic network configuration, add the dependency on the **network-online.target** target unit

to the **sshd.service** unit file. To achieve this, create the
/etc/systemd/system/sshd.service.d/local.conf file with the following content:

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

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

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

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

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

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

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

5. Reload the **systemd** configuration and restart **sshd** to apply the changes:

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

Verification

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

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

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

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

```
# ssh user@ssh-server-example.com
ECDSA key fingerprint is SHA256:dXbaS0RG/UzITTku8GtXSz0S1++IPegSy31v3L/FAEc.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ssh-server-example.com' (ECDSA) to the list of known hosts.

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

Additional resources

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

1.3. SETTING AN OPENSSH SERVER FOR KEY-BASED AUTHENTICATION

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

Prerequisites

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

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

Procedure

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

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

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

   ```
   PasswordAuthentication no
   ```

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

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

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

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

   ```
   # systemctl reload sshd
   ```

Additional resources

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

1.4. GENERATING SSH KEY PAIRS

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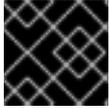
> **IMPORTANT**
>
> If you complete the following steps as **root**, only **root** is able to use the keys.

Procedure

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

   ```
   $ ssh-keygen -t ecdsa
   Generating public/private ecdsa key pair.
   Enter file in which to save the key (/home/joesec/.ssh/id_ecdsa):
   Enter passphrase (empty for no passphrase):
   Enter same passphrase again:
   Your identification has been saved in /home/joesec/.ssh/id_ecdsa.
   Your public key has been saved in /home/joesec/.ssh/id_ecdsa.pub.
   The key fingerprint is:
   SHA256:Q/x+qms4j7PCQ0qFd09iZEFHA+SqwBKRNaU72oZfaCI
   joesec@localhost.example.com
   The key's randomart image is:
   +---[ECDSA 256]---+
   |.oo..o=++        |
   |.. o .oo .       |
   |. .. o. o        |
   |....o.+...        |
   |o.oo.o +S .      |
   |.=.+.  .o        |
   |E.*+.  . . .     |
   |.=..+ +.. o      |
   | .  oo*+o.       |
   +----[SHA256]-----+
   ```

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

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

   ```
   $ ssh-copy-id joesec@ssh-server-example.com
   /usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are
   already installed
   joesec@ssh-server-example.com's password:
   ...
   Number of key(s) added: 1

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

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

ssh-copy-id command with the **-i** option.

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

Verification

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

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

Additional resources

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

1.5. USING SSH KEYS STORED ON A SMART CARD

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

Prerequisites

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

Procedure

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

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

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

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

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

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

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

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

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

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

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

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

Additional resources

- Fedora 28: Better smart card support in OpenSSH

- **p11-kit(8)**, **opensc.conf(5)**, **pcscd(8)**, **ssh(1)**, and **ssh-keygen(1)** man pages

1.6. MAKING OPENSSH MORE SECURE

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

```
# systemctl reload sshd
```

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

Disabling insecure connection protocols

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

Enabling key-based authentication and disabling password-based authentication

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

  > PasswordAuthentication no

Key types

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

  > # systemctl mask sshd-keygen@*rsa*.service

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

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

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

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

Non-default port

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

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

Furthermore, update **firewalld** configuration:

```
# firewall-cmd --add-port <port_number>/tcp
# firewall-cmd --remove-port=22/tcp
# firewall-cmd --runtime-to-permanent
```

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

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 access to specific users, groups, or domains

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

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

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

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

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

Changing system-wide cryptographic policies

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

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

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

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

To opt out of the system-wide cryptographic policies for your OpenSSH server, 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

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

- Using system-wide cryptographic policies in the Security hardening document.

- How to disable specific algorithms and ciphers for ssh service only article.

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

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

Prerequisites

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

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

Procedure

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

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

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

   - The **HostName** parameter sets the actual host name or IP address of the jump host.

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

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

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

   > $ ssh *remote-server*

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

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

Additional resources

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

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

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

Prerequisites

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

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

- You have generated an SSH key pair with a passphrase and transferred the public key to the remote machine.

Procedure

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

   a. Connect to the remote host using SSH:

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

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

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

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

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

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

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

Verification

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

   ```
   $ ssh example.user1@198.51.100.1

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

   Note that you did not have to enter the passphrase.

1.9. ADDITIONAL RESOURCES

- **sshd(8)**, **ssh(1)**, **scp(1)**, **sftp(1)**, **ssh-keygen(1)**, **ssh-copy-id(1)**, **ssh_config(5)**, **sshd_config(5)**, **update-crypto-policies(8)**, and **crypto-policies(7)** man pages.

- OpenSSH Home Page

- Configuring SELinux for applications and services with non-standard configurations

- Controlling network traffic using firewalld

CHAPTER 2. CONFIGURING SECURE COMMUNICATION WITH THE SSH SYSTEM ROLES

As an administrator, you can use the **sshd** System Role to configure SSH servers and the **ssh** System Role to configure SSH clients consistently on any number of RHEL systems at the same time using the Ansible Core package.

2.1. SSH SERVER SYSTEM ROLE VARIABLES

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

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

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

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

renders as:

```
ListenAddress 0.0.0.0
ListenAddress ::
```

Variables for the sshd System Role

sshd_enable

If set to **false**, the role is completely disabled. Defaults to **true**.

sshd_skip_defaults

If set to **true**, the System Role does not apply default values. Instead, you specify the complete set of configuration defaults by using either the **sshd** dictionary or **sshd_<OptionName>** variables. Defaults to **false**.

sshd_manage_service

If set to **false**, the service is not managed, which means it is not enabled on boot and does not start or reload. Defaults to **true** except when running inside a container or AIX, because the Ansible service module does not currently support **enabled** for AIX.

sshd_allow_reload

If set to **false**, **sshd** does not reload after a change of configuration. This can help with troubleshooting. To apply the changed configuration, reload **sshd** manually. Defaults to the same value as **sshd_manage_service** except on AIX, where **sshd_manage_service** defaults to **false** but **sshd_allow_reload** defaults to **true**.

sshd_install_service

If set to **true**, the role installs service files for the **sshd** service. This overrides files provided in the operating system. Do not set to **true** unless you are configuring a second instance and you also change the **sshd_service** variable. Defaults to **false**.

The role uses the files pointed by the following variables as templates:

> sshd_service_template_service (default: templates/sshd.service.j2)
> sshd_service_template_at_service (default: templates/sshd@.service.j2)
> sshd_service_template_socket (default: templates/sshd.socket.j2)

sshd_service

This variable changes the **sshd** service name, which is useful for configuring a second **sshd** service instance.

sshd

A dictionary that contains configuration. For example:

> sshd:
>   Compression: yes
>   ListenAddress:
>   - 0.0.0.0

The **sshd_config(5)** lists all options for the **sshd** dictionary.

sshd_*<OptionName>*

You can define options by using simple variables consisting of the **sshd_** prefix and the option name instead of a dictionary. The simple variables override values in the **sshd** dictionary. For example:

> sshd_Compression: no

The **sshd_config(5)** lists all options for **sshd**.

sshd_manage_firewall

Set this variable to **true** if you are using a different port than the default port **22**. When set to **true**, the **sshd** role uses the **firewall** role to automatically manage port access.

> **NOTE**
>
> The **sshd_manage_firewall** variable can only add ports. It cannot remove ports. To remove ports, use the **firewall** System Role directly. For more information about managing ports by using the **firewall** System Role, see Configuring ports by using System Roles.

sshd_manage_selinux

Set this variable to **true** if you are using a different port than the default port **22**. When set to **true**, the **sshd** role uses the **selinux** role to automatically manage port access.

> **NOTE**
>
> The **sshd_manage_selinux** variable can only add ports. It cannot remove ports. To remove ports, use the **selinux** System Role directly.

sshd_match and **sshd_match_1** to **sshd_match_9**

A list of dictionaries or just a dictionary for a Match section. Note that these variables do not override match blocks as defined in the **sshd** dictionary. All of the sources will be reflected in the resulting configuration file.

sshd_backup

When set to **false**, the original **sshd_config** file is not backed up. Default is **true**.

Secondary variables for the **sshd** System Role

You can use these variables to override the defaults that correspond to each supported platform.

sshd_packages

You can override the default list of installed packages using this variable.

sshd_config_owner, **sshd_config_group**, and **sshd_config_mode**

You can set the ownership and permissions for the **openssh** configuration file that this role produces using these variables.

sshd_config_file

The path where this role saves the **openssh** server configuration produced.

sshd_config_namespace

The default value of this variable is null, which means that the role defines the entire content of the configuration file including system defaults. Alternatively, you can use this variable to invoke this role from other roles or from multiple places in a single playbook on systems that do not support drop-in directory. The **sshd_skip_defaults** variable is ignored and no system defaults are used in this case. When this variable is set, the role places the configuration that you specify to configuration snippets in an existing configuration file under the given namespace. If your scenario requires applying the role several times, you need to select a different namespace for each application.

> **NOTE**
>
> Limitations of the **openssh** configuration file still apply. For example, only the first option specified in a configuration file is effective for most of the configuration options.

Technically, the role places snippets in "Match all" blocks, unless they contain other match blocks, to ensure they are applied regardless of the previous match blocks in the existing configuration file. This allows configuring any non-conflicting options from different roles invocations.

sshd_binary

The path to the **sshd** executable of **openssh**.

sshd_service

The name of the **sshd** service. By default, this variable contains the name of the **sshd** service that the target platform uses. You can also use it to set the name of the custom **sshd** service when the role uses the **sshd_install_service** variable.

sshd_verify_hostkeys

Defaults to **auto**. When set to **auto**, this lists all host keys that are present in the produced configuration file, and generates any paths that are not present. Additionally, permissions and file owners are set to default values. This is useful if the role is used in the deployment stage to verify the service is able to start on the first attempt. To disable this check, set this variable to an empty list **[]**.

sshd_hostkey_owner, **sshd_hostkey_group**, **sshd_hostkey_mode**

Use these variables to set the ownership and permissions for the host keys from **sshd_verify_hostkeys**.

sshd_sysconfig

On systems based on RHEL 8 and earlier versions, this variable configures additional details of the

sshd service. If set to **true**, this role manages also the **/etc/sysconfig/sshd** configuration file based on the **sshd_sysconfig_override_crypto_policy** and **sshd_sysconfig_use_strong_rng** variables. Defaults to **false**.

sshd_sysconfig_override_crypto_policy

In RHEL 8, setting it to **true** allows overriding the system-wide cryptographic policy by using the following configuration options in the **sshd** dictionary or in the **sshd_<OptionName>** format:

- **Ciphers**

- **MACs**

- **GSSAPIKexAlgorithms**

- **GSSAPIKeyExchange** (FIPS-only)

- **KexAlgorithms**

- **HostKeyAlgorithms**

- **PubkeyAcceptedKeyTypes**

- **CASignatureAlgorithms**
  Defaults to **false**.

  In RHEL 9, this variable has no effect. Instead, you can override system-wide cryptographic policies by using the following configuration options in the **sshd** dictionary or in the **sshd_<OptionName>** format:

- **Ciphers**

- **MACs**

- **GSSAPIKexAlgorithms**

- **GSSAPIKeyExchange** (FIPS-only)

- **KexAlgorithms**

- **HostKeyAlgorithms**

- **PubkeyAcceptedAlgorithms**

- **HostbasedAcceptedAlgorithms**

- **CASignatureAlgorithms**

- **RequiredRSASize**
  If you enter these options into custom configuration files in the drop-in directory defined in the **sshd_config_file** variable, use a file name that lexicographically precedes the **/etc/ssh/sshd_config.d/50-redhat.conf** file that includes the cryptographic policies.

sshd_sysconfig_use_strong_rng

On systems based on RHEL 8 and earlier versions, this variable can force **sshd** to reseed the **openssl** random number generator with the number of bytes given as the argument. The default is **0**, which disables this functionality. Do not turn this on if the system does not have a hardware random number generator.

2.2. CONFIGURING OPENSSH SERVERS USING THE SSHD SYSTEM ROLE

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

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

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **sshd** System Role.

- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:

  - The **ansible-core** and **rhel-system-roles** packages are installed.

> **IMPORTANT**
>
> RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information about how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.
>
> RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Copy the example playbook for the **sshd** System Role:

   ```
   # cp /usr/share/doc/rhel-system-roles/sshd/example-root-login-playbook.yml path/custom-playbook.yml
   ```

2. Open the copied playbook by using a text editor, for example:

   ```
   # vim path/custom-playbook.yml

   ---
   ```

```
- hosts: all
  tasks:
  - name: Configure sshd to prevent root and password login except from particular subnet
    include_role:
      name: rhel-system-roles.sshd
    vars:
      sshd:
        # root login and password login is enabled only from a particular subnet
        PermitRootLogin: no
        PasswordAuthentication: no
        Match:
        - Condition: "Address 192.0.2.0/24"
          PermitRootLogin: yes
          PasswordAuthentication: yes
```

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

- password and **root** user login is disabled

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

You can modify the variables according to your preferences. For more details, see sshd System Role variables.

3. Optional: Verify playbook syntax.

   ```
   # ansible-playbook --syntax-check path/custom-playbook.yml
   ```

4. Run the playbook on your inventory file:

   ```
   # ansible-playbook -i inventory_file path/custom-playbook.yml

   ...

   PLAY RECAP
   **************************************************

   localhost : ok=12 changed=2 unreachable=0 failed=0
   skipped=10 rescued=0 ignored=0
   ```

Verification

1. Log in to the SSH server:

   ```
   $ ssh user1@10.1.1.1
   ```

   Where:

   - **user1** is a user on the SSH server.

   - **10.1.1.1** is the IP address of the SSH server.

2. Check 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@10.1.1.1
   ```

Additional resources

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

- **ansible-playbook(1)** man page.

2.3. SSH SYSTEM ROLE VARIABLES

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

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

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

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

renders as:

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

> **NOTE**
>
> The configuration options are case sensitive.

Variables for the ssh System Role

ssh_user

You can define an existing user name for which the System Role modifies user-specific configuration. The user-specific configuration is saved in ~/**.ssh**/**config** of the given user. The default value is null, which modifies global configuration for all users.

ssh_skip_defaults

Defaults to **auto**. If set to **auto**, the System Role writes the system-wide configuration file /**etc**/**ssh**/**ssh_config** and keeps the RHEL defaults defined there. Creating a drop-in configuration file, for example by defining the **ssh_drop_in_name** variable, automatically disables the **ssh_skip_defaults** variable.

ssh_drop_in_name

Defines the name for the drop-in configuration file, which is placed in the system-wide drop-in directory. The name is used in the template /**etc**/**ssh**/**ssh_config.d**/**{ssh_drop_in_name}.conf** to reference the configuration file to be modified. If the system does not support drop-in directory, the default value is null. If the system supports drop-in directories, the default value is **00-ansible**.

> **WARNING**
>
> If the system does not support drop-in directories, setting this option will make the play fail.

The suggested format is **NN-name**, where **NN** is a two-digit number used for ordering the configuration files and **name** is any descriptive name for the content or the owner of the file.

ssh

A dict that contains configuration options and their respective values.

ssh_*OptionName*

You can define options by using simple variables consisting of the **ssh_** prefix and the option name instead of a dict. The simple variables override values in the **ssh** dict.

ssh_additional_packages

This role automatically installs the **openssh** and **openssh-clients** packages, which are needed for the most common use cases. If you need to install additional packages, for example, **openssh-keysign** for host-based authentication, you can specify them in this variable.

ssh_config_file

The path to which the role saves the configuration file produced. Default value:

- If the system has a drop-in directory, the default value is defined by the template /**etc**/**ssh**/**ssh_config.d**/**{ssh_drop_in_name}.conf**.

- If the system does not have a drop-in directory, the default value is /**etc**/**ssh**/**ssh_config**.

- if the **ssh_user** variable is defined, the default value is ~/**.ssh**/**config**.

ssh_config_owner, ssh_config_group, ssh_config_mode

The owner, group and modes of the created configuration file. By default, the owner of the file is **root:root**, and the mode is **0644**. If **ssh_user** is defined, the mode is **0600**, and the owner and group are derived from the user name specified in the **ssh_user** variable.

2.4. CONFIGURING OPENSSH CLIENTS USING THE SSH SYSTEM ROLE

You can use the **ssh** System Role to configure multiple SSH clients by running an Ansible playbook.
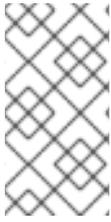
> **NOTE**
>
> You can use the **ssh** System Role with other System Roles that change SSH and SSHD configuration, for example the Identity Management RHEL System Roles. To prevent the configuration from being overwritten, make sure that the **ssh** role uses a drop-in directory (default from RHEL 8).

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **ssh** System Role.

- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:

  - The **ansible-core** and **rhel-system-roles** packages are installed.

> **IMPORTANT**
>
> RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information about how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.
>
> RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Create a new *playbook.yml* file with the following content:

```
---
- hosts: all
  tasks:
  - name: "Configure ssh clients"
    include_role:
      name: rhel-system-roles.ssh
    vars:
      ssh_user: root
      ssh:
        Compression: true
```

```
        GSSAPIAuthentication: no
        ControlMaster: auto
        ControlPath: ~/.ssh/.cm%C
        Host:
          - Condition: example
            Hostname: example.com
            User: user1
      ssh_ForwardX11: no
```

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

- Compression is enabled.

- ControlMaster multiplexing is set to **auto**.

- The **example** alias for connecting to the **example.com** host is **user1**.

- The **example** host alias is created, which represents a connection to the **example.com** host the with **user1** user name.

- X11 forwarding is disabled.

Optionally, you can modify these variables according to your preferences. For more details, see ssh System Role variables.

2. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check path/custom-playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory_file path/custom-playbook.yml
```

Verification

- Verify that the managed node has the correct configuration by opening the SSH configuration file in a text editor, for example:

```
# vi ~root/.ssh/config
```

After application of the example playbook shown above, the configuration file should have the following content:

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

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

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

In RHEL 8 and earlier, you can apply the non-exclusive configuration with a configuration snippet. For more information, see Using the SSH Server System Role for non-exclusive configuration in RHEL 8 documentation.

In RHEL 9, you can apply the non-exclusive configuration 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

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **sshd** System Role.

- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:

  - The **ansible-core** package is installed.

  - An inventory file which lists the managed nodes.

  - A playbook for a different RHEL System Role.

Procedure

1. Add a configuration snippet with the **sshd_config_file** variable to the playbook:

   ```
   ---
   - hosts: all
     tasks:
     - name: <Configure sshd to accept some useful environment variables>
       include_role:
         name: rhel-system-roles.sshd
       vars:
         sshd_config_file: /etc/ssh/sshd_config.d/<42-my-application>.conf
         sshd:
           # Environment variables to accept
           AcceptEnv:
             LANG
             LS_COLORS
             EDITOR
   ```

   In the **sshd_config_file** variable, define the **.conf** file into which the **sshd** System Role writes the configuration options.

   Use a two-digit prefix, for example **42-** to specify the order in which the configuration files will be applied.

When you apply the playbook to the inventory, the role adds the following configuration options to the file defined by the **sshd_config_file** variable.

```
# Ansible managed
#
AcceptEnv LANG LS_COLORS EDITOR
```

Verification

- Optional: Verify playbook syntax.

    ```
    # ansible-playbook --syntax-check playbook.yml -i inventory_file
    ```

Additional resources

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

- **ansible-playbook(1)** man page.

CHAPTER 3. CREATING AND MANAGING TLS KEYS AND CERTIFICATES

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

3.1. TLS CERTIFICATES

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

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

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

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

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

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

Additional resources

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

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

3.2. CREATING A PRIVATE CA USING OPENSSL

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

Prerequisites

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

Procedure

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

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

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

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

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

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

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

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

Next steps

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

   # trust anchor *<ca.crt>*

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

Verification

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

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

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

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

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

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

3. Verify the consistency of the private key:

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

18wRpXoCVC9vcPki7WlT+gnmCI+hRANCAARb9NxIvkaVjFhOoZbGp/HtIQxbM78E
lwbDP0Bl624xBJ8gK68ogSaq2x4SdezFdV1gNeKScDcU+Pj2pELldmdF
-----END PRIVATE KEY-----
```

Additional resources

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

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

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

Procedure

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

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

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

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

[req]
distinguished_name = dn
prompt = no

[dn]
C = <US>
```

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

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

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

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

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

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

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

Next steps

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

Verification

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

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

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

Additional resources

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

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

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

Procedure

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

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

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

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

   [req]
   distinguished_name = dn
   prompt = no

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

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

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

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

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

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

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

Next steps

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

Verification

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

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

Additional resources

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

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

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

Prerequisites

- You have already configured a private CA. See Section 3.2, "Creating a private CA using OpenSSL" for more information.

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

Procedure

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

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

   [client-cert]
   extendedKeyUsage = clientAuth
   ```

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

   ```
   $ openssl x509 -req -in <server-cert.csr> -CA <ca.crt> -CAkey <ca.key> -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
   ```

Additional resources

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

3.6. CREATING A PRIVATE CA USING GNUTLS

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

Prerequisites

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

- You have already installed GnuTLS on your system. If you did not, you can use this command:

  ```
  $ 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 your favourite text editor, for example, vi:

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

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

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

3. Create a certificate signed using the private key generated in step 1:

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

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

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

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

Next steps

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

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

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

Verification

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

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

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

a. Generate a private key for your CA:

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

b. Create a file with your favourite text editor, for example, vi:

```
$ 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 private key of the CA:

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

Additional resources

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

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

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

Procedure

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

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

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

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

tls_www_server

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

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

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

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

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

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

Next steps

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

Verification

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

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

Additional resources

- **certtool(1)** man page

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

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

Procedure

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

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

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

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

   tls_www_client

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

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

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

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

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

Next steps

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

Verification

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

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

Additional resources

- **certtool(1)** man page

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

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

Prerequisites

- You have already configured a private CA. See Section 3.6, "Creating a private CA using GnuTLS" for more information.

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

Procedure

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

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

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

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

Additional resources

- **certtool(1)** man page

CHAPTER 4. USING SHARED SYSTEM CERTIFICATES

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

4.1. THE SYSTEM-WIDE TRUSTSTORE

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

Certificate files are treated depending on the subdirectory they are installed to:

- 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 format are located in

  - **/usr/share/pki/ca-trust-source/** or

  - **/etc/pki/ca-trust/source/**.

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

Additional resources

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

4.2. ADDING NEW CERTIFICATES

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

Prerequisites

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

Procedure

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

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

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

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

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

Additional resources

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

4.3. MANAGING TRUSTED SYSTEM CERTIFICATES

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

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

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

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

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

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

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

  pkcs11:id=%a6%b3%e1%2b%2b%49%b6%d7%73%a1%aa%94%f5%01%e7%73%65%4c%
  ```

```
ac%50;type=cert
    type: certificate
    label: ACEDICOM Root
    trust: anchor
    category: authority
...
```

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

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

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

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

Additional resources

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

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

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

Additional resources

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

CHAPTER 5. PLANNING AND IMPLEMENTING TLS

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

5.1. SSL AND TLS PROTOCOLS

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

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

Protocol version	Usage recommendation
SSL v2	Do not use. Has serious security vulnerabilities. Removed from the core crypto libraries since RHEL 7.
SSL v3	Do not use. Has serious security vulnerabilities. Removed from the core crypto libraries since RHEL 8.
TLS 1.0	Not recommended to use. Has known issues that cannot be mitigated in a way that guarantees interoperability, and does not support modern cipher suites. In RHEL 9, disabled in all cryptographic policies.
TLS 1.1	Use for interoperability purposes where needed. Does not support modern cipher suites. In RHEL 9, disabled in all cryptographic policies.
TLS 1.2	Supports the modern AEAD cipher suites. This version is enabled in all system-wide crypto policies, but optional parts of this protocol contain vulnerabilities and TLS 1.2 also allows outdated algorithms.
TLS 1.3	Recommended version. TLS 1.3 removes known problematic options, provides additional privacy by encrypting more of the negotiation handshake and can be faster thanks usage of more efficient modern cryptographic algorithms. TLS 1.3 is also enabled in all system-wide crypto policies.

Additional resources

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

5.2. SECURITY CONSIDERATIONS FOR TLS IN RHEL 9

In RHEL 9, 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 Using system-wide cryptographic policies for more information.

The default settings provided by libraries included in RHEL 9 are secure enough for most deployments. The TLS implementations use secure algorithms where possible while not preventing connections from or to legacy clients or servers. Apply hardened settings in environments with strict security requirements where legacy clients or servers that do not support secure algorithms or protocols are not expected or allowed to connect.

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

> **WARNING**
>
> Algorithms disabled for the **LEGACY** cryptographic policy do not conform to Red Hat's vision of RHEL 9 security, and their security properties are not reliable. Consider moving away from using these algorithms instead of re-enabling them. If you do decide to re-enable them, for example for interoperability with old hardware, treat them as insecure and apply extra protection measures, such as isolating their network interactions to separate network segments. Do not use them across public networks.

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

5.2.1. Protocols

The latest version of TLS provides the best security mechanism. 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 9 supports TLS version 1.3, not all features of this protocol are fully supported by RHEL 9 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 servers running in FIPS mode, RHEL 9 clients in FIPS mode cannot connect to servers that support only TLS 1.2 without EMS. See TLS Extension "Extended Master Secret" enforced with Red Hat Enterprise Linux 9.2

5.2.2. Cipher suites

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

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

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

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

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

5.2.3. Public key length

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

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

5.3. HARDENING TLS CONFIGURATION IN APPLICATIONS

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

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

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

5.3.1. Configuring the Apache HTTP server to use TLS

The **Apache HTTP Server** can use both **OpenSSL** and **NSS** libraries for its TLS needs. RHEL 9 provides the **mod_ssl** functionality through eponymous packages:

```
# dnf install mod_ssl
```

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

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

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

SSLProtocol

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

SSLCipherSuite

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

SSLHonorCipherOrder

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

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

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

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

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

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

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

See the Adding TLS encryption to an Nginx web server chapter in the Deploying web servers and reverse proxies document for more information.

5.3.3. Configuring the Dovecot mail server to use TLS

To configure your installation of the **Dovecot** mail server to use TLS, modify the

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

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

ssl_protocols

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

ssl_cipher_list

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

ssl_prefer_server_ciphers

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

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

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

Additional resources

- Deploying web servers and reverse proxies

- **config(5)** and **ciphers(1)** man pages.

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

- Mozilla SSL Configuration Generator.

- SSL Server Test.

CHAPTER 6. CONFIGURING A VPN WITH IPSEC

In RHEL 9, a virtual private network (VPN) can be configured using the **IPsec** protocol, which is supported by the **Libreswan** application.

6.1. LIBRESWAN AS AN IPSEC VPN IMPLEMENTATION

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

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

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

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

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

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

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

The IPsec protocol provides two modes of operation:

- Tunnel Mode (the default)

- Transport Mode.

You can configure the kernel with IPsec without IKE. This is called *manual keying*. You can also configure manual keying using the **ip xfrm** commands, however, this is strongly discouraged for security reasons. Libreswan communicates with the Linux kernel using the Netlink interface. The kernel performs packet encryption and decryption.

Libreswan uses the Network Security Services (NSS) cryptographic library. NSS is certified for use with the *Federal Information Processing Standard* (*FIPS*) Publication 140-2.
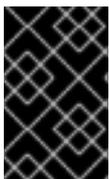
> **IMPORTANT**
>
> IKE/IPsec VPNs, implemented by Libreswan and the Linux kernel, is the only VPN technology recommended for use in RHEL. Do not use any other VPN technology without understanding the risks of doing so.

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

Libreswan does not use the terms "source" and "destination" or "server" and "client" because IKE/IPsec

are peer to peer protocols. Instead, it uses the terms "left" and "right" to refer to end points (the hosts). This also allows you to use the same configuration on both end points in most cases. However, administrators usually choose to always use "left" for the local host and "right" for the remote host.

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

6.2. AUTHENTICATION METHODS IN LIBRESWAN

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

Pre-Shared key (PSK)

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

Raw RSA keys

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

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

X.509 certificates

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

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

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

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

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

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

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

NULL authentication

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

Protection against quantum computers

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

Using IKEv1 with pre-shared keys provides protection against quantum attackers. The redesign of IKEv2 does not offer this protection natively. Libreswan offers the use of *Post-quantum Pre-shared Key* (PPK) to protect IKEv2 connections against quantum attacks.

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

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

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

> ⚠️ **WARNING**
>
> The implementation of dynamic PPKs is provided as an unsupported Technology Preview. Use with caution.

6.3. INSTALLING LIBRESWAN

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

Prerequisites

- The **AppStream** repository is enabled.

Procedure

1. Install the **libreswan** packages:

   ```
   # dnf install libreswan
   ```

2. If you are re-installing Libreswan, remove its old database files and create a new database:

   ```
   # systemctl stop ipsec
   # rm /var/lib/ipsec/nss/*db
   # ipsec initnss
   ```

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

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

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

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

6.4. CREATING A HOST-TO-HOST VPN

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

Prerequisites

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

Procedure

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

   ```
   # ipsec newhostkey
   ```

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

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

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

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

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

   ```
   conn mytunnel
   ```

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

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

   ```
   # systemctl restart ipsec
   ```

5. Load the connection:

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

6. Establish the tunnel:

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

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

   ```
   auto=start
   ```

6.5. CONFIGURING A SITE-TO-SITE VPN

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

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

Prerequisites

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

Procedure

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

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

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

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

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

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

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

6.6. CONFIGURING A REMOTE ACCESS VPN

Road warriors are traveling users with mobile clients and a dynamically assigned IP address. The mobile clients authenticate using X.509 certificates.

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

On the server:

```
conn roadwarriors
    ikev2=insist
    # support (roaming) MOBIKE clients (RFC 4555)
    mobike=yes
    fragmentation=yes
    left=1.2.3.4
    # if access to the LAN is given, enable this, otherwise use 0.0.0.0/0
    # leftsubnet=10.10.0.0/16
    leftsubnet=0.0.0.0/0
    leftcert=gw.example.com
    leftid=%fromcert
    leftxauthserver=yes
    leftmodecfgserver=yes
    right=%any
    # trust our own Certificate Agency
    rightca=%same
    # pick an IP address pool to assign to remote users
    # 100.64.0.0/16 prevents RFC1918 clashes when remote users are behind NAT
    rightaddresspool=100.64.13.100-100.64.13.254
    # if you want remote clients to use some local DNS zones and servers
    modecfgdns="1.2.3.4, 5.6.7.8"
    modecfgdomains="internal.company.com, corp"
    rightxauthclient=yes
    rightmodecfgclient=yes
    authby=rsasig
    # optionally, run the client X.509 ID through pam to allow or deny client
    # pam-authorize=yes
    # load connection, do not initiate
```

```
    auto=add
    # kill vanished roadwarriors
    dpddelay=1m
    dpdtimeout=5m
    dpdaction=clear
```

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

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

6.7. CONFIGURING A MESH VPN

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

- To require IPsec.

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

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

NOTE

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

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

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

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

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

Prerequisites

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

- A new NSS database is initialized.

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

        ```
        # systemctl stop ipsec
        # rm /var/lib/ipsec/nss/*db
        ```

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

        ```
        # ipsec initnss
        ```

Procedure

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

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

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

    ```
    # cat /etc/ipsec.d/mesh.conf
    conn clear
     auto=ondemand 1
     type=passthrough
    ```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
# systemctl restart ipsec
```

Verification

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

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

```
# ping <nodeYYY>
```

2. Display the NSS database with the imported certification:

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

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

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

3. See which tunnels a node has opened:

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

Additional resources

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

- For more information about the **authby** variable, see 6.2. Authentication methods in Libreswan .

6.8. DEPLOYING A FIPS-COMPLIANT IPSEC VPN

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

Prerequisites

- The **AppStream** repository is enabled.

Procedure

1. Install the **libreswan** packages:

   ```
   # dnf install libreswan
   ```

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

   ```
   # systemctl stop ipsec

   # rm /var/lib/ipsec/nss/*db
   ```

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

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

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

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

5. Switch the system to FIPS mode:

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

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

   ```
   # reboot
   ```

Verification

1. To confirm Libreswan is running in FIPS mode:

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

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

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

3. To see the available algorithms in FIPS mode:

```
# ipsec pluto --selftest 2>&1 | head -6
Initializing NSS using read-write database "sql:/var/lib/ipsec/nss"
FIPS Mode: YES
NSS crypto library initialized
FIPS mode enabled for pluto daemon
NSS library is running in FIPS mode
FIPS HMAC integrity support [disabled]
```

4. To query disabled algorithms in FIPS mode:

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

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

```
# ipsec pluto --selftest 2>&1 | grep ESP | grep FIPS | sed "s/^.*FIPS//"
aes_ccm, aes_ccm_c
aes_ccm_b
aes_ccm_a
NSS(CBC)  3des
NSS(GCM)  aes_gcm, aes_gcm_c
NSS(GCM)  aes_gcm_b
NSS(GCM)  aes_gcm_a
NSS(CTR)  aesctr
NSS(CBC)  aes
aes_gmac
NSS       sha, sha1, sha1_96, hmac_sha1
NSS       sha512, sha2_512, sha2_512_256, hmac_sha2_512
NSS       sha384, sha2_384, sha2_384_192, hmac_sha2_384
NSS       sha2, sha256, sha2_256, sha2_256_128, hmac_sha2_256
aes_cmac
null
NSS(MODP) null, dh0
NSS(MODP) dh14
NSS(MODP) dh15
NSS(MODP) dh16
```

NSS(MODP) dh17
NSS(MODP) dh18
NSS(ECP) ecp_256, ecp256
NSS(ECP) ecp_384, ecp384
NSS(ECP) ecp_521, ecp521

Additional resources

- Using system-wide cryptographic policies .

6.9. PROTECTING THE IPSEC NSS DATABASE BY A PASSWORD

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

Prerequisites

- The **/var/lib/ipsec/nss/** directory contains NSS database files.

Procedure

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

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

   Enter new password:
   ```

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

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

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

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

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

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

   ```
   # systemctl restart ipsec
   ```

Verification

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

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

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

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

Additional resources

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

- Government Standards Knowledgebase article.

6.10. CONFIGURING AN IPSEC VPN TO USE TCP

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

Prerequisites

- A remote-access VPN is already configured.

Procedure

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

   ```
   listen-tcp=yes
   ```

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

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

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

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

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

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

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

Prerequisites

- The network card supports ESP hardware offload.

- The network driver supports ESP hardware offload.

- The IPsec connection is configured and works.

Procedure

1. Edit the Libreswan configuration file in the **/etc/ipsec.d/** directory of the connection that should use automatic detection of ESP hardware offload support.

2. Ensure the **nic-offload** parameter is not set in the connection's settings.

3. If you removed **nic-offload**, restart the **ipsec** service:

   ```
   # systemctl restart ipsec
   ```

Verification

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

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

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

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

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

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

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

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

Additional resources

-

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

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

Prerequisites

- All network cards in the bond support ESP hardware offload.

- The network driver supports ESP hardware offload on a bond device. In RHEL, only the **ixgbe** driver supports this feature.

- The bond is configured and works.

- The bond uses the **active-backup** mode. The bonding driver does not support any other modes for this feature.

- The IPsec connection is configured and works.

Procedure

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

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

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

2. Reactivate the **bond0** connection:

   # nmcli connection up *bond0*

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

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

4. Restart the **ipsec** service:

   # systemctl restart ipsec

Verification

1. Display the active port of the bond:

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

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

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

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

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

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

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

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

Additional resources

- Configuring network bonding

- Configuring a VPN with IPsec

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

Overriding system-wide crypto-policies for a connection

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

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

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

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

Disabling system-wide crypto policies for all connections

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

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

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

Additional resources

- Using system-wide cryptographic policies .

6.14. TROUBLESHOOTING IPSEC VPN CONFIGURATIONS

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

Basic connection troubleshooting

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

To confirm that an IPsec VPN connection is established:

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

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

To check that the problem is in the connection:

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

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

2. Next, initiate the VPN connection:

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

Firewall-related problems

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

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

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

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

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

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

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

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

Mismatched algorithms, protocols, and policies

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

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

```
# ipsec auto --up vpn.example.com
...
000 "vpn.example.com"[1] 192.0.2.2 #16: ERROR: asynchronous network error report on
wlp2s0 (192.0.2.2:500), complainant 198.51.100.1: Connection refused [errno 111, origin
ICMP type 3 code 3 (not authenticated)]
...
```

- Example of mismatched IKE algorithms:

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

- Example of mismatched IPsec algorithms:

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

-

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

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

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

- Example of mismatched IP address ranges for IKEv1:

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

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

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

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

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

Maximum transmission unit

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

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

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

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

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

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

Network address translation (NAT)

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

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

The system with address 172.16.0.1 have a NAT rule:

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

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

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

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

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

Kernel IPsec subsystem bugs

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

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

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

Libreswan logs

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

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

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

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

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

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

Additional resources

- Troubleshooting problems using log files .

- **tcpdump(8)** and **ipsec.conf(5)** man pages.

- Using and configuring firewalld

6.15. ADDITIONAL RESOURCES

- **ipsec(8)**, **ipsec.conf(5)**, **ipsec.secrets(5)**, **ipsec_auto(8)**, and **ipsec_rsasigkey(8)** man pages.

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

- The website of the upstream project .

- The Libreswan Project Wiki .

- All Libreswan man pages .

- NIST Special Publication 800-77: Guide to IPsec VPNs .

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

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

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

> **NOTE**
>
> The **vpn** RHEL System Role currently supports only Libreswan, which is an IPsec implementation, as the VPN provider.

7.1. CREATING A HOST-TO-HOST VPN WITH IPSEC USING THEvpn SYSTEM ROLE

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

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **vpn** System Role.

- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:

    - The **ansible-core** and **rhel-system-roles** packages are installed.

> **IMPORTANT**
>
> RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information about how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.
>
> RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Create a new *playbook.yml* file with the following content:

   ```
   - name: Host to host VPN
     hosts: managed_node1, managed_node2
     roles:
       - rhel-system-roles.vpn
     vars:
       vpn_connections:
         - hosts:
             managed_node1:
             managed_node2:
       vpn_manage_firewall: true
       vpn_manage_selinux: true
   ```

   This playbook configures the connection *managed_node1*-to-*managed_node2* using pre-shared key authentication with keys auto-generated by the system role. Because **vpn_manage_firewall** and **vpn_manage_selinux** are both set to **true**, the **vpn** role uses the **firewall** and **selinux** roles to manage the ports used by the **vpn** role.

2. Optional: Configure connections from managed hosts to external hosts that are not listed in the inventory file by adding the following section to the **vpn_connections** list of hosts:

   ```
   vpn_connections:
     - hosts:
         managed_node1:
         managed_node2:
         external_node:
           hostname: 192.0.2.2
   ```

   This configures two additional connections: *managed_node1*-to-*external_node* and *managed_node2*-to-*external_node*.

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

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

   ```
   - name: Multiple VPN
     hosts: managed_node1, managed_node2
     roles:
       - rhel-system-roles.vpn
     vars:
       vpn_connections:
         - name: control_plane_vpn
           hosts:
             managed_node1:
               hostname: 192.0.2.0 # IP for the control plane
             managed_node2:
               hostname: 192.0.2.1
         - name: data_plane_vpn
   ```

```
hosts:
  managed_node1:
    hostname: 10.0.0.1 # IP for the data plane
  managed_node2:
    hostname: 10.0.0.2
```

2. Optional: You can modify the variables according to your preferences. For more details, see the **/usr/share/doc/rhel-system-roles/vpn/README.md** file.

3. Optional: Verify playbook syntax.

```
# ansible-playbook --syntax-check /path/to/file/playbook.yml -i /path/to/file/inventory_file
```

4. Run the playbook on your inventory file:

```
# ansible-playbook -i /path/to/file/inventory_file /path/to/file/playbook.yml
```

Verification

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

```
# ipsec status | grep connection.name
```

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

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

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

```
# ipsec trafficstatus | grep connection.name
```

2. Optional: If a connection did not successfully load, manually add the connection by entering the following command. This will provide more specific information indicating why the connection failed to establish:

```
# ipsec auto --add connection.name
```

> **NOTE**
>
> Any errors that may have occurred during the process of loading and starting the connection are reported in the logs, which can be found in **/var/log/pluto.log**. Because these logs are hard to parse, try to manually add the connection to obtain log messages from the standard output instead.

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

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

Authentication with certificates is configured by defining the **auth_method: cert** parameter in the playbook. The **vpn** System Role assumes that the IPsec Network Security Services (NSS) crypto library, which is defined in the **/etc/ipsec.d** directory, contains the necessary certificates. By default, the node name is used as the certificate nickname. In this example, this is **managed_node1**. You can define different certificate names by using the **cert_name** attribute in your inventory.

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

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

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **vpn** System Role.

  - On all the managed nodes, the NSS database in the **/etc/ipsec.d** directory contains all the certificates necessary for peer authentication. By default, the node name is used as the certificate nickname.

- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
  On the control node:

  - The **ansible-core** and **rhel-system-roles** packages are installed.

> **IMPORTANT**
>
> RHEL 8.0-8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information about how to obtain and install Ansible Engine, see the How to download and install Red Hat Ansible Engine Knowledgebase article.
>
> RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. Create a new **_playbook.yml_** file with the following content:

   ```
   - name: Mesh VPN
     hosts: managed_node1, managed_node2, managed_node3
     roles:
       - rhel-system-roles.vpn
     vars:
      vpn_connections:
        - opportunistic: true
          auth_method: cert
          policies:
            - policy: private
              cidr: default
            - policy: private-or-clear
              cidr: 198.51.100.0/24
            - policy: private
              cidr: 192.0.2.0/24
            - policy: clear
              cidr: 192.0.2.7/32
      vpn_manage_firewall: true
      vpn_manage_selinux: true
   ```

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

2. Optional: You can modify the variables according to your preferences. For more details, see the **/usr/share/doc/rhel-system-roles/vpn/README.md** file.

3. Optional: Verify playbook syntax.

   ```
   # ansible-playbook --syntax-check playbook.yml
   ```

4. Run the playbook on your inventory file:

   ```
   # ansible-playbook -i inventory_file /path/to/file/playbook.yml
   ```

7.3. ADDITIONAL RESOURCES

- For details about the parameters used in the **vpn** System Role and additional information about the role, see the **/usr/share/doc/rhel-system-roles/vpn/README.md** file.

- For details about the **ansible-playbook** command, see the **ansible-playbook(1)** man page.

CHAPTER 8. SECURING NETWORK SERVICES

Red Hat Enterprise Linux 9 supports many different types of network servers. Their network services can expose the system security to risks of various types of attacks, such as denial of service attacks (DoS), distributed denial of service attacks (DDoS), script vulnerability attacks, and buffer overflow attacks.

To increase the system security against attacks, it is important to monitor active network services that you use. For example, when a network service is running on a machine, its daemon listens for connections on network ports, and this can reduce the security. To limit exposure to attacks over the network, all services that are unused should be turned off.

8.1. SECURING THE RPCBIND SERVICE

The **rpcbind** service is a dynamic port-assignment daemon for remote procedure calls (RPC) services such as Network Information Service (NIS) and Network File System (NFS). Because it has weak authentication mechanisms and can assign a wide range of ports for the services it controls, it is important to secure **rpcbind**.

You can secure **rpcbind** by restricting access to all networks and defining specific exceptions using firewall rules on the server.

> **NOTE**
>
> - The **rpcbind** service is required on **NFSv3** servers.
>
> - **NFSv4** does not require the **rpcbind** service to listen on the network.

Prerequisites

- The **rpcbind** package is installed.

- The **firewalld** package is installed and the service is running.

Procedure

1. Add firewall rules, for example:

   - Limit TCP connection and accept packages only from the **192.168.0.0/24** host via the **111** port:

     ```
     # firewall-cmd --add-rich-rule='rule family="ipv4" port port="111" protocol="tcp" source address="192.168.0.0/24" invert="True" drop'
     ```

   - Limit TCP connection and accept packages only from local host via the **111** port:

     ```
     # firewall-cmd --add-rich-rule='rule family="ipv4" port port="111" protocol="tcp" source address="127.0.0.1" accept'
     ```

   - Limit UDP connection and accept packages only from the **192.168.0.0/24** host via the **111** port:

     ```
     # firewall-cmd --permanent --add-rich-rule='rule family="ipv4" port port="111" protocol="udp" source address="192.168.0.0/24" invert="True" drop'
     ```

To make the firewall settings permanent, use the **--permanent** option when adding firewall rules.

2. Reload the firewall to apply the new rules:

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

Verification steps

- List the firewall rules:

```
# firewall-cmd --list-rich-rule
rule family="ipv4" port port="111" protocol="tcp" source address="192.168.0.0/24"
invert="True" drop
rule family="ipv4" port port="111" protocol="tcp" source address="127.0.0.1" accept
rule family="ipv4" port port="111" protocol="udp" source address="192.168.0.0/24"
invert="True" drop
```

Additional resources

- For more information about **NFSv4-only** servers, see the Configuring an NFSv4-only server section.

- 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 using firewall rules.

Prerequisites

- The **rpc.mountd** package is installed.

- The **firewalld** package is installed and the service is running.

Procedure

1. Add firewall rules to the server, for example:

- Accept **mountd** connections from the **192.168.0.0/24** host:

```
# firewall-cmd --add-rich-rule 'rule family="ipv4" service name="mountd" source
address="192.168.0.0/24" invert="True" drop'
```

- Accept **mountd** connections from the local host:

```
# firewall-cmd --permanent --add-rich-rule 'rule family="ipv4" source address="127.0.0.1"
service name="mountd" accept'
```

To make the firewall settings permanent, use the **--permanent** option when adding firewall rules.

2. Reload the firewall to apply the new rules:

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

Verification steps

- List the firewall rules:

```
# firewall-cmd --list-rich-rule
rule family="ipv4" service name="mountd" source address="192.168.0.0/24" invert="True" drop
rule family="ipv4" source address="127.0.0.1" service name="mountd" accept
```

Additional resources

- [Using and configuring firewalld](#)

8.3. SECURING THE NFS SERVICE

You can secure Network File System version 4 (NFSv4) by authenticating and encrypting all file system operations using Kerberos. When using NFSv4 with Network Address Translation (NAT) or a firewall, you can turn off the delegations by modifying the **/etc/default/nfs** file. Delegation is a technique by which the server delegates the management of a file to a client.

In contrast, 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

The NFS server determines a list structure of directories and hosts about which file systems to export to which hosts in the **/etc/exports** file.

> **WARNING**
>
> Extra spaces in the syntax of the exports file can lead to major changes in the configuration.
>
> In the following example, the **/tmp/nfs/** directory is shared with the **bob.example.com** host and has read and write permissions.
>
> > /tmp/nfs/   bob.example.com(rw)
>
> The following example is the same as the previous one but shares the same directory to the **bob.example.com** host with read-only permissions and shares it to the *world* with read and write permissions due to a single space character after the hostname.
>
> > /tmp/nfs/   bob.example.com (rw)
>
> You can check the shared directories on your system by entering the **showmount -e <hostname>** command.

Use the following export options on the **/etc/exports** file:

> **WARNING**
>
> Export an entire file system because exporting a subdirectory of a file system is not secure. An attacker can possibly access the unexported part of a partially-exported file system.

ro

Use the **ro** option to export the NFS volume as read-only.

rw

Use the **rw** option to allow read and write requests on the NFS volume. Use this option cautiously because allowing write access increases the risk of attacks.

> **NOTE**
>
> If your scenario requires to mount the directories with the **rw** option, make sure they are not writable for all users to reduce possible risks.

root_squash

Use the **root_squash** option to map requests from **uid/gid** 0 to the anonymous **uid/gid**. This does not apply to any other **uids** or **gids** that might be equally sensitive, such as the **bin** user or the **staff** group.

no_root_squash

Use the **no_root_squash** option to turn off root squashing. By default, NFS shares change the **root** user to the **nobody** user, which is an unprivileged user account. This changes the owner of all the **root** created files to **nobody**, which prevents the uploading of programs with the **setuid** bit set. When using the **no_root_squash** option, remote root users can change any file on the shared file system and leave applications infected by trojans for other users.

secure

Use the **secure** option to restrict exports to reserved ports. By default, the server allows client communication only through reserved ports. However, it is easy for anyone to become a **root** user on a client on many networks, so it is rarely safe for the server to assume that communication through a reserved port is privileged. Therefore the restriction to reserved ports is of limited value; it is better to rely on Kerberos, firewalls, and restriction of exports to particular clients.

Additionally, consider the following best practices when exporting an NFS server:

- Exporting home directories is a risk because some applications store passwords in plain text or in a weakly encrypted format. You can reduce the risk by reviewing and improving the application code.

- Some users do not set passwords on SSH keys which again leads to risks with home directories. You can reduce these risks by enforcing the use of passwords or using Kerberos.

- Restrict the NFS exports only to required clients. Use the **showmount -e** command on the NFS server to review what the server is exporting. Do not export anything that is not specifically required.

- Do not allow unnecessary users to log in to a server to reduce the risk of attacks. You can periodically check who and what can access the server.

Additional resources

- Secure NFS with Kerberos when using Red Hat Identity Management

- NFS server configuration.

- **exports(5)** and **nfs(5)** man pages

8.3.2. Mount options for securing an NFS client

You can pass the following options to the **mount** command to increase the security of NFS-based clients:

nosuid

Use the **nosuid** option to disable the **set-user-identifier** or **set-group-identifier** bits. This prevents remote users from gaining higher privileges by running a **setuid** program and you can use this option opposite to **setuid** option.

noexec

Use the **noexec** option to disable all executable files on the client. Use this to prevent users from accidentally executing files placed in the shared file system.

nodev

Use the **nodev** option to prevent the client's processing of device files as a hardware device.

resvport

Use the **resvport** option to restrict communication to a reserved port and you can use a privileged source port to communicate with the server. The reserved ports are reserved for privileged users and processes such as the **root** user.

sec

Use the **sec** option on the NFS server to choose the RPCGSS security flavor for accessing files on the mount point. Valid security flavors are **none**, **sys**, **krb5**, **krb5i**, and **krb5p**.

> **IMPORTANT**
>
> The MIT Kerberos libraries provided by the **krb5-libs** package do not support the Data Encryption Standard (DES) algorithm in new deployments. DES is deprecated and disabled by default in Kerberos libraries because of security and compatibility reasons. Use newer and more secure algorithms instead of DES, unless your environment requires DES for compatibility reasons.

Additional resources

- [Common NFS mount options](#).

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 steps

- List the active ports and RPC programs on the NFS server:

  ```
  $ rpcinfo -p
  ```

Additional resources

- [Secure NFS with Kerberos](#) when using Red Hat Identity Management

- **exports(5)** and **nfs(5)** man pages

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, you should ensure it is configured securely.

RHEL 9 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, FTP shows a greeting banner, which by default includes version information that could be useful for attackers to identify weaknesses in a system. You can prevent the attackers from accessing this information by changing the default banner.

You can define a custom banner by editing the **/etc/banners/ftp.msg** file to either directly include a single-line message, or to refer to a separate file, which can contain a multi-line message.

Procedure

- To define a single line message, add the following option to the **/etc/vsftpd/vsftpd.conf** file:

  ```
  ftpd_banner=Hello, all activity on ftp.example.com is logged.
  ```

- To define a message in a separate file:

  - Create a **.msg** file which contains the banner message, for example **/etc/banners/_ftp_.msg**:

    ```
    ######### Hello, all activity on ftp.example.com is logged. #########
    ```

    To simplify the management of multiple banners, place all banners into the **/etc/banners/** directory.

  - Add the path to the banner file to the **banner_file** option in the **/etc/vsftpd/vsftpd.conf** file:

    ```
    banner_file=/etc/banners/ftp.msg
    ```

Verification

- Display the modified banner:

  ```
  $ ftp localhost
  Trying ::1…
  Connected to localhost (::1).
  Hello, all activity on ftp.example.com is logged.
  ```

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, you can configure the FTP server to allow anonymous users to upload files to a specific directory and prevent anonymous users from reading data. In the following procedure, the anonymous user must be able to upload files in the directory owned by the **root** user but not change it.

Procedure

- Create a write-only directory in the **/var/ftp/pub/** directory:

  ```
  # mkdir /var/ftp/pub/upload
  # chmod 730 /var/ftp/pub/upload
  # ls -ld /var/ftp/pub/upload
  drwx-wx---. 2 root ftp 4096 Nov 14 22:57 /var/ftp/pub/upload
  ```

- Add the following lines to the **/etc/vsftpd/vsftpd.conf** file:

  ```
  anon_upload_enable=YES
  anonymous_enable=YES
  ```

- Optional: If your system has SELinux enabled and enforcing, enable SELinux boolean attributes **allow_ftpd_anon_write** and **allow_ftpd_full_access**.

> ⚠️ **WARNING**
>
> Allowing anonymous users to read and write in directories might lead to the server becoming a repository for stolen software.

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.

8.4.4. Additional resources

- **ftpd_selinux(8)** man page

8.5. SECURING HTTP SERVERS

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** user with write permissions, enter the following commands:

```
# chown root directory-name
# chmod 755 directory-name
```

In the **/etc/httpd/conf/httpd.conf** file, you can configure the following options:

FollowSymLinks

This directive is enabled by default and follows symbolic links in the directory.

Indexes

This directive is enabled by default. Disable this directive to prevent visitors from browsing files on the server.

UserDir

This directive is disabled by default because it can confirm the presence of a user account on the system. To activate user directory browsing for all user directories other than /**root**/, use the **UserDir enabled** and **UserDir disabled** root directives. To add users to the list of disabled accounts, add a space-delimited list of users on the **UserDir disabled** line.

ServerTokens

This directive controls the server response header field which is sent back to clients. You can use the following parameters to customize the information:

ServerTokens Full

provides all available information such as web server version number, server operating system details, installed Apache modules, for example:

> Apache/2.4.37 (Red Hat Enterprise Linux) MyMod/1.2

ServerTokens Full-Release

provides all available information with release versions, for example:

> Apache/2.4.37 (Red Hat Enterprise Linux) (Release 41.module+el8.5.0+11772+c8e0c271)

ServerTokens Prod / ServerTokens ProductOnly

provides the web server name, for example:

> Apache

ServerTokens Major

provides the web server major release version, for example:

> Apache/2

ServerTokens Minor

provides the web server minor release version, for example:

> Apache/2.4

ServerTokens Min / ServerTokens Minimal

provides the web server minimal release version, for example:

> Apache/2.4.37

ServerTokens OS

provides the web server release version and operating system, for example:

> Apache/2.4.37 (Red Hat Enterprise Linux)

Use the **ServerTokens Prod** option to reduce the risk of attackers gaining any valuable information about your system.

> **IMPORTANT**
>
> Do not remove the **IncludesNoExec** directive. By default, the Server Side Includes (SSI) module cannot execute commands. Changing this can allow an attacker to enter commands on the system.

Removing httpd modules

You can remove the **httpd** modules to limit the functionality of the HTTP server. To do so, edit configuration files in the **/etc/httpd/conf.modules.d/** or **/etc/httpd/conf.d/** directory. For example, to remove the proxy module:

```
echo '# All proxy modules disabled' > /etc/httpd/conf.modules.d/00-proxy.conf
```

Additional resources

- The Apache HTTP server

- Customizing the SELinux policy for the Apache HTTP server

8.5.2. Securing the Nginx server configuration

Nginx is a high-performance HTTP and proxy server. You can harden your Nginx configuration with the following configuration options.

Procedure

- To disable version strings, modify the **server_tokens** configuration option:

  ```
  server_tokens off;
  ```

  This option stops displaying additional details such as server version number. This configuration displays only the server name in all requests served by Nginx, for example:

  ```
  $ curl -sI http://localhost | grep Server
  Server: nginx
  ```

- Add extra security headers that mitigate certain known web application vulnerabilities in specific **/etc/nginx/** conf files:

  - For example, the **X-Frame-Options** header option denies any page outside of your domain to frame any content served by Nginx, mitigating clickjacking attacks:

    ```
    add_header X-Frame-Options "SAMEORIGIN";
    ```

  - For example, the **x-content-type** header prevents MIME-type sniffing in certain older browsers:

    ```
    add_header X-Content-Type-Options nosniff;
    ```

  - For example, the **X-XSS-Protection** header enables Cross-Site Scripting (XSS) filtering, which prevents browsers from rendering potentially malicious content included in a response by Nginx:

    ```
    add_header X-XSS-Protection "1; mode=block";
    ```

- You can limit the services exposed to the public and limit what they do and accept from the visitors, for example:

  ```
  limit_except GET {
      allow 192.168.1.0/32;
      deny  all;
  }
  ```

  The snippet will limit access to all methods except **GET** and **HEAD**.

- You can disable HTTP methods, for example:

  ```
  # Allow GET, PUT, POST; return "405 Method Not Allowed" for all others.
  if ( $request_method !~ ^(GET|PUT|POST)$ ) {
      return 405;
  }
  ```

- You can configure SSL to protect the data served by your Nginx web server, consider serving it over HTTPS only. Furthermore, you can generate a secure configuration profile for enabling SSL in your Nginx server using the Mozilla SSL Configuration Generator. The generated configuration ensures that known vulnerable protocols (for example, SSLv2 and SSLv3), ciphers, and hashing algorithms (for example, 3DES and MD5) are disabled. You can also use the SSL Server Test to verify that your configuration meets modern security requirements.

Additional resources

- [Mozilla SSL Configuration Generator](#)

- [SSL Server Test](#)

8.6. SECURING POSTGRESQL BY LIMITING ACCESS TO AUTHENTICATED LOCAL USERS

PostgreSQL is an object-relational database management system (DBMS). In Red Hat Enterprise Linux, PostgreSQL is provided by the **postgresql-server** package.

You can reduce the risks of attacks by configuring client authentication. The **pg_hba.conf** configuration file stored in the database cluster's data directory controls the client authentication. Follow the procedure to configure PostgreSQL for host-based authentication.

Procedure

1. Install PostgreSQL:

   ```
   # yum install postgresql-server
   ```

2. Initialize a database storage area using one of the following options:

   a. Using the **initdb** utility:

      ```
      $ initdb -D /home/postgresql/db1/
      ```

      The **initdb** command with the **-D** option creates the directory you specify if it does not already exist, for example **/home/postgresql/db1/**. This directory then contains all the data stored in the database and also the client authentication configuration file.

   b. Using the **postgresql-setup** script:

      ```
      $ postgresql-setup --initdb
      ```

      By default, the script uses the **/var/lib/pgsql/data/** directory. This script helps system administrators with basic database cluster administration.

3. To allow any authenticated local users to access any database with their usernames, modify the following line in the **pg_hba.conf** file:

```
local   all         all                         trust
```

This can be problematic when you use layered applications that create database users and no local users. If you do not want to explicitly control all user names on the system, remove the **local** line entry from the **pg_hba.conf** file.

4. Restart the database to apply the changes:

```
# systemctl restart postgresql
```

The previous command updates the database and also verifies the syntax of the configuration file.

8.7. SECURING THE MEMCACHED SERVICE

Memcached is an open source, high-performance, distributed memory object caching system. It can improve the performance of dynamic web applications by lowering database load.

Memcached is an in-memory key-value store for small chunks of arbitrary data, such as strings and objects, from results of database calls, API calls, or page rendering. Memcached allows assigning memory from underutilized areas to applications that require more memory.

In 2018, vulnerabilities of DDoS amplification attacks by exploiting Memcached servers exposed to the public internet were discovered. These attacks took advantage of Memcached communication using the UDP protocol for transport. The attack was effective because of the high amplification ratio where a request with the size of a few hundred bytes could generate a response of a few megabytes or even hundreds of megabytes in size.

In most situations, the **memcached** service does not need to be exposed to the public Internet. Such exposure may have its own security problems, allowing remote attackers to leak or modify information stored in Memcached.

Follow the section to harden the system using Memcached service against possible DDoS attacks.

8.7.1. Hardening Memcached against DDoS

To mitigate security risks, perform as many of the following steps as applicable for your configuration.

Procedure

- Configure a firewall in your LAN. If your Memcached server should be accessible only in your local network, do not route external traffic to ports used by the **memcached** service. For example, remove the default port **11211** from the list of allowed ports:

```
# firewall-cmd --remove-port=11211/udp
# firewall-cmd --runtime-to-permanent
```

- If you use a single Memcached server on the same machine as your application, set up **memcached** to listen to localhost traffic only. Modify the **OPTIONS** value in the **/etc/sysconfig/memcached** file:

```
OPTIONS="-l 127.0.0.1,::1"
```

- Enable Simple Authentication and Security Layer (SASL) authentication:

  1. Modify or add the **/etc/sasl2/memcached.conf** file:

     ```
     sasldb_path: /path.to/memcached.sasldb
     ```

  2. Add an account in the SASL database:

     ```
     # saslpasswd2 -a memcached -c cacheuser -f /path.to/memcached.sasldb
     ```

  3. Ensure that the database is accessible for the **memcached** user and group:

     ```
     # chown memcached:memcached /path.to/memcached.sasldb
     ```

  4. Enable SASL support in Memcached by adding the **-S** value to the **OPTIONS** parameter in the **/etc/sysconfig/memcached** file:

     ```
     OPTIONS="-S"
     ```

  5. Restart the Memcached server to apply the changes:

     ```
     # systemctl restart memcached
     ```

  6. Add the username and password created in the SASL database to the Memcached client configuration of your application.

- Encrypt communication between Memcached clients and servers with TLS:

  1. Enable encrypted communication between Memcached clients and servers with TLS by adding the **-Z** value to the **OPTIONS** parameter in the **/etc/sysconfig/memcached** file:

     ```
     OPTIONS="-Z"
     ```

  2. Add the certificate chain file path in the PEM format using the **-o ssl_chain_cert** option.

  3. Add a private key file path using the **-o ssl_key** option.

CHAPTER 9. USING MACSEC TO ENCRYPT LAYER-2 TRAFFIC IN THE SAME PHYSICAL NETWORK

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

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

- dynamic host configuration protocol (DHCP)

- address resolution protocol (ARP)

- Internet Protocol version 4 / 6 (**IPv4** / **IPv6**) and

- any traffic over IP such as TCP or UDP

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

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

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

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

- host to host and

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

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

9.1. CONFIGURING A MACSEC CONNECTION USING NMCLI

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

Procedure

1. On the first host on which you configure MACsec:

   - Create the connectivity association key (CAK) and connectivity-association key name (CKN) for the pre-shared key:

a. Create a 16-byte hexadecimal CAK:

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

b. Create a 32-byte hexadecimal CKN:

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

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

3. Create the MACsec connection:

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

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

4. Configure the IP settings on the MACsec connection.

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

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

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

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

5. Activate the connection:

```
# nmcli connection up macsec0
```

Verification

1. Verify that the traffic is encrypted:

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

2. Optional: Display the unencrypted traffic:

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

3. Display MACsec statistics:

```
# ip macsec show
```

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

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

9.2. ADDITIONAL RESOURCES

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

CHAPTER 10. SECURING THE POSTFIX SERVICE

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. You can also mitigate risks to various attacks by changing setting to more secure values.

10.1. REDUCING POSTFIX NETWORK-RELATED SECURITY RISKS

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**.

10.2. POSTFIX CONFIGURATION OPTIONS FOR LIMITING DOS ATTACKS

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 a DoS attack:

smtpd_client_connection_rate_limit

   This directive 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

   This directive is a time unit to calculate the rate limit. The default value is **60** seconds.

smtpd_client_event_limit_exceptions

This directive excludes clients from the connection and rate limit commands. By default, the directive excludes clients in trusted networks.

smtpd_client_message_rate_limit

This directive 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

This directive 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

This directive 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

This directive 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

This directive defines the maximum size of a message including the envelope information in bytes. By default, the value is **10240000** bytes.

10.3. CONFIGURING POSTFIX TO USE SASL

Postfix supports Simple Authentication and Security Layer (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 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 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 plaintext and non-plaintext 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