Securing networks

Configuring secured networks and network communication
Red Hat Enterprise Linux 9.0 Beta Securing networks

Configuring secured networks and network communication
Abstract

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

## RHEL Beta Release

- Making Open Source More Inclusive ........................................... 5
- Providing Feedback on Red Hat Documentation .......................... 6

### Chapter 1. Using Secure Communications Between Two Systems with OpenSSH

1.1. SSH and OpenSSH ......................................................... 7
1.2. Configuring and Starting an OpenSSH Server ................. 8
1.3. Setting an OpenSSH Server for Key-Based Authentication ... 9
1.4. Generating SSH Key Pairs .............................................. 10
1.5. Using SSH Keys Stored on a Smart Card ...................... 12
1.6. Making OpenSSH More Secure ....................................... 13
1.7. Connecting to a Remote Server Using an SSH Jump Host ... 15
1.8. Connecting to Remote Machines with SSH Keys Using SSH-Agent ... 16
1.9. Additional Resources .................................................. 17

### Chapter 2. Configuring Secure Communication with the SSH System Roles

2.1.sshd System Role Variables ............................................ 18
2.2. Configuring OpenSSH Servers Using the sshd System Role ... 20
2.3. SSH System Role Variables ............................................ 23
2.4. Configuring OpenSSH Clients Using the SSH System Role .... 24
2.5. Using the SSH Server System Role for Non-Exclusive Configuration 26

### Chapter 3. Planning and Implementing TLS

3.1. SSL and TLS Protocols .................................................. 28
3.2. Security Considerations for TLS in RHEL 9 ...................... 28
3.2.1. Protocols ............................................................. 29
3.2.2. Cipher suites ......................................................... 29
3.2.3. Public key length .................................................... 30
3.3. Hardening TLS Configuration in Applications ................. 30
3.3.1. Configuring the Apache HTTP Server ...................... 30
3.3.2. Configuring the Nginx HTTP and Proxy Server .......... 31
3.3.3. Configuring the Dovecot Mail Server ...................... 31

### Chapter 4. Configuring a VPN with IPsec

4.1. Libreswan as an IPsec VPN Implementation .................. 33
4.2. Authentication Methods in Libreswan ......................... 34
4.3. Installing Libreswan ................................................... 35
4.4. Creating a Host-to-Host VPN ....................................... 36
4.5. Configuring a Site-to-Site VPN .................................... 37
4.6. Configuring a Remote Access VPN ............................... 38
4.7. Configuring a Mesh VPN .............................................. 39
4.8. Deploying a FIPS-Compliant IPsec VPN ....................... 41
4.9. Protecting the IPsec NSS Database by a Password .......... 43
4.10. Configuring an IPsec VPN to use TCP ....................... 44
4.11. Configuring ESP Hardware Offload on a Bond to Accelerate an IPsec Connection 45
4.13. Troubleshooting IPsec VPN Configurations ................ 47

### Chapter 5. Using MACsec to Encrypt Layer-2 Traffic in the Same Physical Network

5.1. Configuring a MACsec Connection Using nmcli ............... 53
RHEL BETA RELEASE

Red Hat provides Red Hat Enterprise Linux Beta access to all subscribed Red Hat accounts. The purpose of Beta access is to:

- Provide an opportunity to customers to test major features and capabilities prior to the general availability release and provide feedback or report issues.

- Provide Beta product documentation as a preview. Beta product documentation is under development and is subject to substantial change.

Note that Red Hat does not support the usage of RHEL Beta releases in production use cases. For more information, see What does Beta mean in Red Hat Enterprise Linux and can I upgrade a RHEL Beta installation to a General Availability (GA) release?.

Red Hat Enterprise Linux 9.0 Beta Securing networks
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 input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages:
  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the **Add Feedback** pop-up that appears below the highlighted text.
  4. Follow the displayed instructions.

- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the **Bugzilla** website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.
CHAPTER 1. USING SECURE COMMUNICATIONS BETWEEN TWO SYSTEMS WITH OPENSsh

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

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

1.1. SSH AND OPENSSH

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

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

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

- **ssh** is a remote login program (SSH client)
- **sshd** is an OpenSSH SSH daemon
- **scp** is a secure remote file copy program
- **sftp** is a secure file transfer program
- **ssh-agent** is an authentication agent for caching private keys
- **ssh-add** adds private key identities to **ssh-agent**
- **ssh-keygen** generates, manages, and converts authentication keys for **ssh**
- **ssh-copy-id** is a script that adds local public keys to the **authorized_keys** file on a remote SSH server
- **ssh-keyscan** - gathers SSH public host keys

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

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

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

Additional resources

- Man pages 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:

   ```ini
   [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:`
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
```

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

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

```
# ssh user@ssh-server-example.com
```

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

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

### Additional resources

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

### 1.3. SETTING AN OPENSSH SERVER FOR KEY-BASED AUTHENTICATION

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

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

The sshd daemon is running on the server.

Procedure

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

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

2. Change the PasswordAuthentication option to no:

```
PasswordAuthentication no
```

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

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

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

4. Reload the sshd daemon to apply the changes:

```
# systemctl reload sshd
```

Additional resources

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

1.4. GENERATING SSH KEY PAIRS

Use this procedure to generate an SSH key pair on a local system and to copy the generated public key to an OpenSSH server. If the server is configured accordingly, you can log in to the OpenSSH server without providing any password.

**IMPORTANT**

If you complete the following steps as root, only root is able to use the keys.

Procedure

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

```
$ ssh-keygen -t ecdsa
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/joesec/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/joesec/.ssh/id_ecdsa.
Your public key has been saved in /home/joesec/.ssh/id_ecdsa.pub.
```
The key fingerprint is:
SHA256:Q/x+qms4j7PCQ0qFd09iZEFHA+SqwBKRNaU72oZfaCl
joesec@localhost.example.com
The key's randomart image is:
+---[ECDSA 256]---+
| .00...0==++      |
| .. o oo .        |
| . .. o oo        |
| ....0.+...       |
| 0.00.0 +S .      |
| .=+. +.0         |
| E.*+. . .       |
| .=.+ +.. 0       |
| . 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
```

```
Welcome message.
...
Last login: Mon Nov 18 18:28:42 2019 from ::1
```

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

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

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

Non-default port

- By default, the `sshd` daemon listens on TCP port 22. Changing the port reduces the exposure of the system to attacks based on automated network scanning and thus increase security through obscurity. You can specify the port using the `Port` directive in the `/etc/ssh/sshd_config` configuration file.

You also have to update the default SELinux policy to allow the use of a non-default port. To do so, use the `semanage` tool from the `policycoreutils-python-utils` package:

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

Furthermore, update `firewalld` configuration:

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

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

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.

**CAUTION**

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/sshd_config.d/05-redhat.conf` file is set to `yes`, and there is no difference between the `ssh -X remote_machine` (untrusted host) and `ssh -Y remote_machine` (trusted host) command.

If your scenario does not require the X11 forwarding feature at all, set the `X11Forwarding` directive in the `/etc/ssh/sshd_config` configuration file to `no`. 
Restricting access to specific users, groups, or domains

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

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

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

  Note that using 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
  ```

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

  - See [Using system-wide cryptographic policies](#) in the [Security hardening](#) title for more information.

Additional resources

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

**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.example.com:75 joesec@remote1.example.com:220
```

**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. SSHD 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:

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

renders as:

```plaintext
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 dict, or sshd_Key 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
This variable changes the sshd service name, which is useful for configuring a second sshd service instance.

sshd
A dict that contains configuration. For example:

```
sshd:
  Compression: yes
  ListenAddress:
    - 0.0.0.0
```

sshd_OptionName
You can define options by using simple variables consisting of the sshd_prefix and the option name instead of a dict. The simple variables override values in the sshd dict. For example:

```
sshd_Compression: no
```

sshd_match and sshd_match_1 to sshd_match_9
A list of dicts or just a dict for a Match section. Note that these variables do not override match blocks as defined in the sshd dict. All of the sources will be reflected in the resulting configuration file.

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 make sure 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 RHEL-based systems, 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 following configuration. Defaults to `false`.

**sshd_sysconfig_override_crypto_policy**

In RHEL, when set to `true`, this variable overrides the system-wide crypto policy. Defaults to `false`.

**sshd_sysconfig_use_strong_rng**

On RHEL-based systems, 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.

**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 Engine configures other systems.
  
  On the control node:
- The **ansible-core** package is installed.
- An inventory file which lists the managed nodes.

**Procedure**

1. Copy the example playbook for the SSHD System Role:

   ```bash
   # 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:

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

   ```yaml
   ---
   - 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 Server System Role variables](1).

3. Optional: Verify playbook syntax.

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

4. Run the playbook on your inventory file:

   ```bash
   # 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:
   - \textit{user1} is a user on the SSH server.
   - \textit{10.1.1.1} is the IP address of the SSH server.

2. Check the contents of the \texttt{sshd\_config} file on the SSH server:

   $ vim /etc/ssh/sshd\_config

   ```
   # Ansible managed
   HostKey /etc/ssh/ssh\_host\_rsa\_key
   HostKey /etc/ssh/ssh\_host\_ecdsa\_key
   HostKey /etc/ssh/ssh\_host\_ed25519\_key
   AcceptEnv LANG LC\_CTYPE LC\_NUMERIC LC\_TIME LC\_COLLATE LC\_MONETARY
   LC\_MESSAGES
   AcceptEnv LC\_PAPER LC\_NAME LC\_ADDRESS LC\_TELEPHONE LC\_MEASUREMENT
   AcceptEnv LC\_IDENTIFICATION LC\_ALL LANGUAGE
   AcceptEnv XMODIFIERS
   AuthorizedKeysFile .ssh/authorized\_keys
   ChallengeResponseAuthentication no
   GSSAPIAuthentication yes
   GSSAPICleanupCredentials no
   PasswordAuthentication no
   PermitRootLogin no
   PrintMotd no
   Subsystem sftp /usr/libexec/openssh/sftp-server
   SyslogFacility AUTHPRIV
   UsePAM yes
   X11Forwarding yes
   Match Address 192.0.2.0/24
   PasswordAuthentication yes
   PermitRootLogin yes
   ```

3. Check that you can connect to the server as root from the \texttt{192.0.2.0/24} subnet:

   a. Determine your IP address:

      $ hostname -I
      192.0.2.1

      If the IP address is within the \texttt{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
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/sshd/sshd_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/sshd/sshd_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.

#### Prerequisites

- Access and permissions to one or more `managed nodes`, which are systems you want to configure with the SSH System Role.
- The Ansible Core package is installed on the control machine.

#### 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 with user1 user name.
- X11 forwarding is disabled.

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

2. Optional: Verify playbook syntax.

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

3. Run the playbook on your inventory file:

```bash
# 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:

```bash
# 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 SSH SERVER 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 Engine 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. For additional information, see Applying a role.

Procedure

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

```yaml
---
- 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.

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

Verification

- Optional: Verify playbook syntax.

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

Additional resources

- `ansible-playbook(1)` man page.
CHAPTER 3. 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.

3.1. 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.

<table>
<thead>
<tr>
<th>Protocol version</th>
<th>Usage recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL v2</td>
<td>Do not use. Has serious security vulnerabilities. Removed from the core crypto libraries since RHEL 7.</td>
</tr>
<tr>
<td>SSL v3</td>
<td>Do not use. Has serious security vulnerabilities. Removed from the core crypto libraries since RHEL 8.</td>
</tr>
<tr>
<td>TLS 1.0</td>
<td>Not recommended to use. Has known issues that cannot be mitigated in a way that guarantees interoperability, and does not support modern cipher suites. Enabled only in the LEGACY system-wide cryptographic policy profile.</td>
</tr>
<tr>
<td>TLS 1.1</td>
<td>Use for interoperability purposes where needed. Does not support modern cipher suites. Enabled only in the LEGACY policy.</td>
</tr>
<tr>
<td>TLS 1.2</td>
<td>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.</td>
</tr>
<tr>
<td>TLS 1.3</td>
<td>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.</td>
</tr>
</tbody>
</table>

Additional resources


3.2. 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:

### 3.2.1. 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.

### 3.2.2. 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).

3.2.3. 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.

3.3. 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 make sure to mandate that your server application enforces server-side cipher order, so that the cipher suite to be used is determined by the order you configure.

3.3.1. Configuring the Apache HTTP server

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

```
# 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 `/usr/share/httpd/manual/mod/mod_ssl.html`. Examples of various settings are in `/usr/share/httpd/manual/ssl/ssl_howto.html`.

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

**SSLProtocol**

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

**SSLCipherSuite**

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

**SSLHonorCipherOrder**

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

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

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

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

### 3.3.2. Configuring the Nginx HTTP and proxy server

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

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

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

### 3.3.3. Configuring the Dovecot mail server

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

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

**ssl_protocols**

Use this directive to specify the version of TLS or SSL you want to allow or disable.
ssl_cipher_list
Use this directive to specify your preferred cipher suites or disable the ones you want to disallow.

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

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

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

Additional resources

- link: 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 4. 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.

4.1. 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 Level 2 Tunneling Protocol (L2TP) is usually called an L2TP/IPsec VPN, which requires the Optional channel xl2tpd application.

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

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

- Encapsulated Security Payload (ESP), which has protocol number 50.
- Authenticated Header (AH), which has protocol number 51.

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

The IPsec protocol provides two modes of operation:

- Tunnel Mode (the default)
- Transport Mode.

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

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

IMPORTANT

IKE/IPsec VPNs, implemented by Libreswan and the Linux kernel, is the only VPN technology recommended for use in 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.

4.2. 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=
ecdsa and RSA Probabilistic Signature Scheme (RSASSA-PSS) digital signatures based authentication with SHA-2 through \texttt{authby=rsa-sha2}. The default value is \texttt{authby=rsasig,ecdsa}.

The certificates and the \texttt{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 \texttt{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 \texttt{ppk=yes} to the connection definition. To require PPK, add \texttt{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 \texttt{ipsec.secrets}, for example:

\begin{verbatim}
    @west @east : PPKS "user1" "thestringismeanttobearandomstr"
\end{verbatim}

The \texttt{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 \texttt{ipsec.secrets(5)} man page for more information.

\begin{warning}
The implementation of dynamic PPKs is provided as an unsupported Technology Preview. Use with caution.
\end{warning}

### 4.3. INSTALLING LIBRESWAN

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

**Prerequisites**

- The **AppStream** repository is enabled.
Procedure

1. Install the libreswan packages:

```bash
# dnf install libreswan
```

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

```bash
# 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:

```bash
# 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:

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

4.4. CREATING A HOST-TO-HOST VPN

To configure Libreswan to create a host-to-host IPsec VPN between two hosts referred to as left and right using authentication by raw RSA keys, enter the following commands on both of the hosts:

Prerequisites

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

Procedure

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

```bash
# ipsec newhostkey
```

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

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

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

```bash
# 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:

```bash
conn mytunnel
```
4. After importing keys, restart the `ipsec` service:

```bash
# systemctl restart ipsec
```

5. Load the connection:

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

6. Establish the tunnel:

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

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

```bash
auto=start
```

### 4.5. 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:

```bash
# 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:

```plaintext
conn mysubnet
    also=mytunnel
    leftsubnet=192.0.1.0/24
    rightsubnet=192.0.2.0/24
    auto=start
```
conn mysubnet6
  also=mytunnel
  leftsubnet=2001:db8:0:1::/64
  rightsubnet=2001:db8:0:2::/64
  auto=start

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

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

4.6. 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
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
    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
```

### 4.7. 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).

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

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

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

Procedure

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

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

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

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

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

conn private-or-clear
  auto=ondemand
  type=transport
  authby=rsasig
  failureshunt=passthrough
  negotiationshunt=passthrough
  # left
  left=%defaultroute
  leftcert=nodeXXXX
  leftid=%fromcert
    lefrsasigkey=%cert
  # right
  rightrsasigkey=%cert
  rightid=%fromcert
  right=%opportunisticgroup
```
3. Add the IP address of the network in the proper category. For example, if all nodes reside in the 10.15.0.0/16 network, and all nodes should mandate IPsec encryption:

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

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

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

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

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

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

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

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

   # systemctl restart ipsec

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

   # ipsec trafficstatus

4.8. 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:
4. Configure the firewall to allow 500 and 4500/UDP ports for the IKE, ESP, and AH protocols by adding the `ipsec` service:

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

5. Switch the system to FIPS mode:

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

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

```bash
# reboot
```

**Verification**

1. To confirm Libreswan is running in FIPS mode:

```bash
# 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:

```bash
# 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:

```bash
# 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:

```bash
# 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.

4.9. 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:

   ```bash
   # 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.

### 4.10. CONFIGURING AN IPSEC VPN TO USE TCP

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

Prerequisites

- A remote-access VPN is already configured.

Procedure

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

   ```
   listen-tcp=yes
   ```

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

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

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

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

Additional resources

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

4.11. 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 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 enp1s0 | egrep "_ipsec"
```

```
  tx_ipsec: 15
  rx_ipsec: 15
```

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

**Additional resources**

- Configuring network bonding
- The Configuring a VPN with IPsec section in the Securing networks documentation
- Configuring a VPN with IPsec chapter in the Securing networks document.
4.12. 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.

4.13. 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 \texttt{vpn.example.com} connection:

   \begin{verbatim}
   # ipsec auto --add vpn.example.com
   002 added connection description "vpn.example.com"
   \end{verbatim}

2. Next, initiate the VPN connection:

   \begin{verbatim}
   # ipsec auto --up vpn.example.com
   \end{verbatim}

\section*{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:

  \begin{verbatim}
  # 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 response
  ...
  \end{verbatim}

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

  \begin{verbatim}
  # 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
  ...
  \end{verbatim}

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

- To capture the negotiation of the VPN and all encrypted data on the \texttt{eth0} interface:

  \begin{verbatim}
  # tcpdump -i eth0 -n -n esp or udp port 500 or udp port 4500 or tcp port 4500
  \end{verbatim}

\section*{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,N
```

- 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:
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`

### 4.14. 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 5. USING MACSEC TO ENCRYPT LAYER-2 TRAFFIC IN THE SAME PHYSICAL NETWORK

This section describes how to configure MACsec for secure communication for all traffic on Ethernet links.

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.

The following example shows how to configure MACsec between 2 hosts using a pre-shared key.

5.1. CONFIGURING A MACSEC CONNECTION USING NMCLI

You can configure Ethernet interfaces to use MACsec using the nmcli tool. This procedure describes how to create a MACsec connection that uses an Ethernet interface to encrypt the network traffic.

Run this procedure on all the hosts that should communicate in this MACsec-protected network.

Procedure

On Host A:
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 16-byte hexadecimal CAK:

```
dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
50b71a8e0bd5751ea76de6d6c98c03a
```

b. Create 32-byte hexadecimal CKN:

```
dd if=/dev/urandom count=32 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
f2b4297d39da7330910a7abc0449fbeb45b5c0b9fc23df1430e1898fcf1c4550
```

On Host A and B:

1. 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 50b71a8e0bd5751ea76de6d6c98c03a
macsec.mka-ckn f2b4297d39da7330910a7abc0449fbeb45b5c0b9fc23df1430e1898fcf1c4550
```

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.

2. 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
```

3. Activate the connection:

```
# nmcli connection up macsec0
```

**Verification steps**

1. To verify the traffic is encrypted, enter:

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

2. To view the unencrypted traffic, enter:

```
tcpdump -nn -i macsec0
```
3. To display MACsec statistics:

   # ip macsec show

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

   # ip -s macsec show

### 5.2. ADDITIONAL RESOURCES

- [MACsec: a different solution to encrypt network traffic](blog)