Red Hat Enterprise Linux 8

Securing networks

Configuring secured networks and network communication
Red Hat Enterprise Linux 8 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

MAKING OPEN SOURCE MORE INCLUSIVE ......................................................... 6

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION ................................. 7

CHAPTER 1. USING SECURE COMMUNICATIONS BETWEEN TWO SYSTEMS WITH OPENSSH ........ 8

1.1. SSH AND OPENSSH ................................................................. 8
1.2. CONFIGURING AND STARTING AN OPENSSH SERVER ....................... 9
1.3. SETTING AN OPENSSH SERVER FOR KEY-BASED AUTHENTICATION .... 10
1.4. GENERATING SSH KEY PAIRS .................................................. 11
1.5. USING SSH KEYS STORED ON A SMART CARD ............................ 13
1.6. MAKING OPENSSH MORE SECURE .......................................... 14
1.7. CONNECTING TO A REMOTE SERVER USING AN SSH JUMP HOST ....... 16
1.8. CONNECTING TO REMOTE MACHINES WITH SSH KEYS USING SSH-AGENT ... 17
1.9. ADDITIONAL RESOURCES ......................................................... 18

CHAPTER 2. CONFIGURING SECURE COMMUNICATION WITH THE SSH SYSTEM ROLES ........ 19

2.1. SSHD SYSTEM ROLE VARIABLES ............................................. 19
2.2. CONFIGURING OPENSSH SERVERS USING THE SSHD SYSTEM ROLE .. 21
2.3. SSH SYSTEM ROLE VARIABLES ............................................... 24
2.4. CONFIGURING OPENSSH CLIENTS USING THE SSH SYSTEM ROLE .... 25
2.5. USING THE SSH SERVER SYSTEM ROLE FOR NON-EXCLUSIVE CONFIGURATION . 27

CHAPTER 3. PLANNING AND IMPLEMENTING TLS .......................................... 29

3.1. SSL AND TLS PROTOCOLS ...................................................... 29
3.2. SECURITY CONSIDERATIONS FOR TLS IN RHEL 8 ....................... 29
3.2.1. Protocols .......................................................... 30
3.2.2. Cipher suites ......................................................... 30
3.2.3. Public key length .................................................... 31
3.3. HARDENING TLS CONFIGURATION IN APPLICATIONS .......... 31
3.3.1. Configuring the Apache HTTP server ................................. 31
3.3.2. Configuring the Nginx HTTP and proxy server .................. 32
3.3.3. Configuring the Dovecot mail server ................................ 32

CHAPTER 4. CONFIGURING A VPN WITH IPSEC ............................................ 34

4.1. LIBRESWAN AS AN IPSEC VPN IMPLEMENTATION ......................... 34
4.2. AUTHENTICATION METHODS IN LIBRESWAN ............................ 35
4.3. INSTALLING LIBRESWAN ....................................................... 36
4.4. CREATING A HOST-TO-HOST VPN ........................................... 37
4.5. CONFIGURING A SITE-TO-SITE VPN ....................................... 38
4.6. CONFIGURING A REMOTE ACCESS VPN .................................. 39
4.7. CONFIGURING A MESH VPN .................................................. 40
4.8. DEPLOYING A FIPS-COMPLIANT IPSEC VPN ............................. 42
4.9. PROTECTING THE IPSEC NSS DATABASE BY A PASSWORD ........... 44
4.10. CONFIGURING AN IPSEC VPN TO USE TCP ............................ 46
4.11. CONFIGURING AUTOMATIC DETECTION AND USAGE OF ESP HARDWARE OFFLOAD TO ACCELERATE AN IPSEC CONNECTION .................. 46
4.12. CONFIGURING ESP HARDWARE OFFLOAD ON A BOND TO ACCELERATE AN IPSEC CONNECTION ........................................ 47
4.13. CONFIGURING IPSEC CONNECTIONS THAT OPT OUT OF THE SYSTEM-WIDE CRYPTO POLICIES .................................. 49
4.14. TROUBLESHOOTING IPSEC VPN CONFIGURATIONS ................. 50
4.15. ADDITIONAL RESOURCES ..................................................... 54

CHAPTER 5. CONFIGURING VPN CONNECTIONS WITH IPSEC BY USING THE RHEL VPN SYSTEM ROLE ........................... 55
7.7.4. Setting the default target of policy objects
7.8. CONFIGURING NAT USING FIREWALLD
    7.8.1. The different NAT types: masquerading, source NAT, destination NAT, and redirect
    7.8.2. Configuring IP address masquerading
7.9. PORT FORWARDING
    7.9.1. Adding a port to redirect
    7.9.2. Redirecting TCP port 80 to port 88 on the same machine
    7.9.3. Removing a redirected port
    7.9.4. Removing TCP port 80 forwarded to port 88 on the same machine
7.10. MANAGING ICMP REQUESTS
    7.10.1. Listing and blocking ICMP requests
    7.10.2. Configuring the ICMP filter using GUI
7.11. SETTING AND CONTROLLING IP SETS USING FIREWALLD
    7.11.1. Configuring IP set options using CLI
7.12. PRIORITIZING RICH RULES
    7.12.1. How the priority parameter organizes rules into different chains
    7.12.2. Setting the priority of a rich rule
7.13. CONFIGURING FIREWALL LOCKDOWN
    7.13.1. Configuring lockdown using CLI
    7.13.2. Configuring lockdown allowlist options using CLI
    7.13.3. Configuring lockdown allowlist options using configuration files
7.14. ENABLING TRAFFIC FORWARDING BETWEEN DIFFERENT INTERFACES OR SOURCES WITHIN A FIREWALLD ZONE
    7.14.1. The difference between intra-zone forwarding and zones with the default target set to ACCEPT
    7.14.2. Using intra-zone forwarding to forward traffic between an Ethernet and Wi-Fi network
7.15. ADDITIONAL RESOURCES

CHAPTER 8. GETTING STARTED WITH NFTABLES

8.1. MIGRATING FROM IPTABLES TO NFTABLES
    8.1.1. When to use firewalld, nftables, or iptables
    8.1.2. Converting iptables rules to nftables rules
    8.1.3. Comparison of common iptables and nftables commands
8.2. WRITING AND EXECUTING NFTABLES SCRIPTS
    8.2.1. Supported nftables script formats
    8.2.2. Running nftables scripts
    8.2.3. Using comments in nftables scripts
    8.2.4. Using variables in an nftables script
    8.2.5. Including files in an nftables script
    8.2.6. Automatically loading nftables rules when the system boots
8.3. CREATING AND MANAGING NFTABLES TABLES, CHAINS, AND RULES
    8.3.1. Standard chain priority values and textual names
    8.3.2. Displaying the nftables rule set
    8.3.3. Creating an nftables table
    8.3.4. Creating an nftables chain
    8.3.5. Appending a rule to the end of an nftables chain
    8.3.6. Inserting a rule at the beginning of an nftables chain
    8.3.7. Inserting a rule at a specific position of an nftables chain
8.4. CONFIGURING NAT USING NFTABLES
    8.4.1. The different NAT types: masquerading, source NAT, destination NAT, and redirect
    8.4.2. Configuring masquerading using nftables
    8.4.3. Configuring source NAT using nftables
    8.4.4. Configuring destination NAT using nftables
    8.4.5. Configuring a redirect using nftables
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5. USING SETS IN NFTABLES COMMANDS</td>
<td>111</td>
</tr>
<tr>
<td>8.5.1. Using anonymous sets in nftables</td>
<td>111</td>
</tr>
<tr>
<td>8.5.2. Using named sets in nftables</td>
<td>112</td>
</tr>
<tr>
<td>8.5.3. Additional resources</td>
<td>113</td>
</tr>
<tr>
<td>8.6. USING VERDICT MAPS IN NFTABLES COMMANDS</td>
<td>113</td>
</tr>
<tr>
<td>8.6.1. Using anonymous maps in nftables</td>
<td>113</td>
</tr>
<tr>
<td>8.6.2. Using named maps in nftables</td>
<td>115</td>
</tr>
<tr>
<td>8.6.3. Additional resources</td>
<td>116</td>
</tr>
<tr>
<td>8.7. CONFIGURING PORT FORWARDING USING NFTABLES</td>
<td>116</td>
</tr>
<tr>
<td>8.7.1. Forwarding incoming packets to a different local port</td>
<td>117</td>
</tr>
<tr>
<td>8.7.2. Forwarding incoming packets on a specific local port to a different host</td>
<td>117</td>
</tr>
<tr>
<td>8.8. USING NFTABLES TO LIMIT THE AMOUNT OF CONNECTIONS</td>
<td>118</td>
</tr>
<tr>
<td>8.8.1. Limiting the number of connections using nftables</td>
<td>118</td>
</tr>
<tr>
<td>8.8.2. Blocking IP addresses that attempt more than ten new incoming TCP connections within one minute</td>
<td>119</td>
</tr>
<tr>
<td>8.9. DEBUGGING NFTABLES RULES</td>
<td>119</td>
</tr>
<tr>
<td>8.9.1. Creating a rule with a counter</td>
<td>120</td>
</tr>
<tr>
<td>8.9.2. Adding a counter to an existing rule</td>
<td>120</td>
</tr>
<tr>
<td>8.9.3. Monitoring packets that match an existing rule</td>
<td>121</td>
</tr>
<tr>
<td>8.10. BACKING UP AND RESTORING THE NFTABLES RULE SET</td>
<td>122</td>
</tr>
<tr>
<td>8.10.1. Backing up the nftables rule set to a file</td>
<td>122</td>
</tr>
<tr>
<td>8.10.2. Restoring the nftables rule set from a file</td>
<td>122</td>
</tr>
<tr>
<td>8.11. ADDITIONAL RESOURCES</td>
<td>123</td>
</tr>
</tbody>
</table>
MAKING OPEN SOURCE MORE INCLUSIVE

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

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

- 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 feedback via Bugzilla, create a new 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. This
ensures that weak cipher suites and cryptographic algorithms are disabled in the default configuration. To adjust the policy, the administrator must either use the `update-crypto-policies` command to make settings stricter or looser or manually opt-out of the system-wide crypto policies.

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

Additional resources

- Man pages listed by using the `man -k ssh` command
- Using system-wide cryptographic policies

1.2. CONFIGURING AND STARTING AN OPENSSH SERVER

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

Prerequisites

- The `openssh-server` package is installed.

Procedure

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

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

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

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

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

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

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

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

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

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

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

Verification

1. Check that the sshd daemon is running:

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

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

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

```
# ssh user@ssh-server-example.com
ECDSA key fingerprint is SHA256:dXbaS0RG/UzITTkBu8GtXSz0S1++IPeqSy31v3L/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+SqwBKRNaU72oZfaC
joesec@localhost.example.com
The key's randomart image is:
+----[ECDSA 256]----+
 |.oo..o=++        |
 |.. o .oo .       |
 |. .. o. o        |
 |....o.+...       |
 |o.oo.o +S .      |
 |.=.+.   .o       |
 |E.*+.  .  . .    |
 |.=..+ +..  o     |
 |  .  oo*+o.      |
+----[SHA256]-----+

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

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

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

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

   If you do not use the \texttt{ssh-agent} program in your session, the previous command copies the most recently modified \texttt{~/.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 \texttt{ssh-agent}, use the \texttt{ssh-copy-id} command with the \texttt{-i} option.

\begin{center}
\textbf{NOTE}
\end{center}

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

\section*{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

\section*{Additional resources}

   \begin{itemize}
   \item \texttt{ssh-keygen(1) and ssh-copy-id(1) man pages.}
   \end{itemize}
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:
To exclude particular key types for SSH connections, comment out the relevant lines in 
/etc/ssh/sshd_config, and reload the sshd service. For example, to allow only Ed25519 host keys:

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

**Non-default port**

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

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

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

Furthermore, update firewalld configuration:

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

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

**No root login**

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

  Alternatively, set `PermitRootLogin` to `prohibit-password`:

```
PermitRootLogin prohibit-password
```

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

**Using the X Security extension**

- The X server in Red Hat Enterprise Linux clients does not provide the X Security extension. Therefore, clients cannot request another security layer when connecting to untrusted SSH servers with X11 forwarding. Most applications are not able to run with this extension enabled anyway.

  By default, the `ForwardX11Trusted` option in the /etc/ssh/ssh_config.d/05-redhat.conf file is set to `yes`, and there is no difference between the `ssh -X remote_machine` (untrusted host) and `ssh -Y remote_machine` (trusted host) command.
If your scenario does not require the X11 forwarding feature at all, set the `X11Forwarding` directive in the `/etc/ssh/sshd_config` configuration file to `no`.

Restricting access to specific users, groups, or domains

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

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

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

  Note that using 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](#using-system-wide-cryptographic-policies) in the [Security hardening](#security-hardening) title for more information.

Additional resources

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

1.7. 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
   ```

Additional resources

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

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

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

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

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

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

Procedure

1. Optional: Verify you can use the key to authenticate to the remote host:
   a. Connect to the remote host using SSH:

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

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

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

2. Start the `ssh-agent`.

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

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

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

Verification

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

  ```
  $ ssh example.user1@198.51.100.1
  Last login: Mon Sep 14 12:56:37 2020
  ```

  Note that you did not have to enter the passphrase.

1.9. ADDITIONAL RESOURCES

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

- OpenSSH Home Page

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

- Controlling network traffic using firewalld
CHAPTER 2. CONFIGURING SECURE COMMUNICATION WITH THE SSH SYSTEM ROLES

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

2.1. 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:

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

renders as:

```
ListenAddress 0.0.0.0
ListenAddress ::
```

Variables for the SSHD System Role

`sshd_enable`

If set to `False`, the role is completely disabled. Defaults to `True`.

`sshd_skip_defaults`

If set to `True`, the system role does not apply default values. Instead, you specify the complete set of configuration defaults by using either the `sshd` 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_template_service (default: templates/sshd.service.j2)
sshd_service_template_at_service (default: templates/sshd@.service.j2)
sshd_service_template_socket (default: templates/sshd.socket.j2)

**sshd_service**

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

**sshd**

A dict that contains configuration. For example:

```yaml
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:

```yaml
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.

NOTE

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

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

Access and permissions to a control node, which is a system from which Red Hat Ansible Engine configures other systems.

On the control node:

- Red Hat Ansible Engine is installed.
- The rhel-system-roles 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:
        ssdh:
          # root login and password login is enabled only from a particular subnet
          PermitRootLogin: no
          PasswordAuthentication: no
        Match:
          - Condition: "Address 192.0.2.0/24"
            PermitRootLogin: yes
            PasswordAuthentication: yes
```

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

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

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

3. Optional: Verify playbook syntax.

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

4. Run the playbook on your inventory file:
# ansible-playbook -i inventory_file path/custom-playbook.yml

...  

PLAY RECAP

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

Verification

1. Log in to the SSH server:

   $ ssh user1@10.1.1.1

   Where:
   
   - user1 is a user on the SSH server.
   - 10.1.1.1 is the IP address of the SSH server.

2. Check the contents of the sshd_config file on the SSH server:

   $ 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 192.0.2.0/24 subnet:

   a. Determine your IP address:
If the IP address is within the 192.0.2.1 - 192.0.2.254 range, you can connect to the server.

b. Connect to the server as root:

```bash
$ ssh root@10.1.1.1
```

Additional resources

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

### 2.3. SSH SYSTEM ROLE VARIABLES

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

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

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

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

renders as:

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

**NOTE**

The configuration options are case sensitive.

### Variables for the SSH System Role

**ssh_user**

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

**ssh_skip_defaults**

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

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

**WARNING**

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

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

- **ssh**
  A dict that contains configuration options and their respective values.

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

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

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

  - If the system has a drop-in directory, the default value is defined by the template `/etc/ssh/ssh_config.d/{ssh_drop_in_name}.conf`.
  - If the system does not have a drop-in directory, the default value is `/etc/ssh/ssh_config`.
  - If the `ssh_user` variable is defined, the default value is `~/.ssh/config`.

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

## 2.4. CONFIGURING OPENSSH CLIENTS USING THE SSH SYSTEM ROLE

You can use the SSH System Role to configure multiple SSH clients by running an Ansible playbook.
NOTE

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

Prerequisites

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

- Access and permissions to a control node, which is a system from which Red Hat Ansible Engine configures other systems.
  
  On the control node:
  
  - Red Hat Ansible Engine is installed.
  
  - The `rhel-system-roles` package is installed.
  
  - An inventory file which lists the managed nodes.

Procedure

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

```yaml
---
- hosts: all
  tasks:
  - name: "Configure ssh clients"
    include_role:
      name: rhel-system-roles.ssh
    vars:
      ssh_user: root
      ssh:
        Compression: true
        GSSAPIAuthentication: no
        ControlMaster: auto
        ControlPath: ~/.ssh/.cm%C
    Host:
      - Condition: example
        Hostname: example.com
        User: user1
        ssh_ForwardX11: no
```

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

- Compression is enabled.

- ControlMaster multiplexing is set to `auto`.

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

- The `example` host alias is created, which represents a connection to the `example.com` host the with `user1` user name.
• X11 forwarding is disabled.

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

2. Optional: Verify playbook syntax.

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

3. Run the playbook on your inventory file:

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

**Verification**

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

   # vi ~root/.ssh/config

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

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

**2.5. USING THE 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.

**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:

  • Red Hat Ansible Engine is installed.
The `rhel-system-roles` 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_namespace` 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_namespace: <my-application>
  sshd:
    # Environment variables to accept
    AcceptEnv:
      LANG
      LS_COLORS
      EDITOR
```

When you apply the playbook to the inventory, the role adds the following snippet, if not already present, to the `/etc/ssh/sshd_config` file.

```bash
# BEGIN sshd system role managed block: namespace <my-application>
Match all
AcceptEnv LANG LS_COLORS EDITOR
# END sshd system role managed block: namespace <my-application>
```

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 8

In RHEL 8, cryptography-related considerations are significantly simplified thanks to the system-wide
crypto policies. The **DEFAULT** crypto policy allows only TLS 1.2 and 1.3. To allow your system to negotiate connections using the earlier versions of TLS, you need to either opt out from following crypto policies in an application or switch to the **LEGACY** policy with the `update-crypto-policies` command. See [Using system-wide cryptographic policies](#) for more information.

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

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

---

**WARNING**

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

---

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

### 3.2.1. Protocols

The latest version of TLS provides the best security mechanism. Unless you have a compelling reason to include support for older versions of TLS, allow your systems to negotiate connections using at least TLS version 1.2.

Note that even though RHEL 8 supports TLS version 1.3, not all features of this protocol are fully supported by RHEL 8 components. For example, the 0-RTT (Zero Round Trip Time) feature, which reduces connection latency, is not yet fully supported by the Apache web server.

### 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.

---

**Additional resources**

- [System-wide crypto policies in RHEL 8](#)
- [update-crypto-policies(8) man page](#)

### 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:
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 different types of servers 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 different types of servers 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**

- Deploying different types of servers on RHEL 8
- `config(5)` and `ciphers(1)` man pages.
- Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS).
- Mozilla SSL Configuration Generator.
- SSL Server Test.
CHAPTER 4. CONFIGURING A VPN WITH IPSEC

In RHEL 8, a virtual private network (VPN) can be configured using the IPsec protocol, which is supported by the 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-1 and SHA-2. You can further limit it for ECDSA digital signatures using SHA-2 by setting
authby= to ecdsa and RSA Probabilistic Signature Scheme (RSASSA-PSS) digital signatures based authentication with SHA-2 through authby=rsa-sha2. The default value is authby=rsasig,ecdsa.

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

**NULL authentication**

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

**Protection against quantum computers**

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

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

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

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

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

**WARNING**

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

### 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:
   
   ```
   # yum install libreswan
   ```

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

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

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

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

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

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

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:

   ```
   # ipsec newhostkey
   ```

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

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

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

   ```
   # ipsec showhostkey --right --ckaid a9e1f6ce9ecd3608c24e8f79e808f3f41798f03
   ```

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

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

```
# systemctl restart ipsec
```

5. Load the connection:

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

6. Establish the tunnel:

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

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

```
auto=start
```

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:

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

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

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

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

conn mytunnel
   leftid=@west
   left=192.1.2.23
   leftrsasigkey=0sAQOrlo+h0afUZDICQmXFrje/oZm [...] W2n417C/4urYHQkCvulIQ==
   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
   modecfgdnss="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
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
       leftrsasigkey=%cert
   # right
   rightrsasigkey=%cert
   rightid=%fromcert
   right=%opportunisticgroup

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

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

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

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

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

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

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

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

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

```
# systemctl restart ipsec
```

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

```
# ipsec trafficstatus
```

### 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:

```
# yum install libreswan
```

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

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

3. Start the ipsec service, and enable the service to be started automatically on boot:
Configure the firewall to allow 500 and 4500/UDP ports for the IKE, ESP, and AH protocols by adding the `ipsec` service:

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

Switch the system to FIPS mode:

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

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

```
# reboot
```

**Verification**

1. To confirm Libreswan is running in FIPS mode:

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

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

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

3. To see the available algorithms in FIPS mode:

```
# ipsec pluto --selftest 2>&1 | head -11
FIPS Product: YES
FIPS Kernel: YES
FIPS Mode: YES
NSS DB directory: sql:/etc/ipsec.d
Initializing NSS
Opening NSS database "sql:/etc/ipsec.d" read-only
NSS initialized
NSS crypto library initialized
FIPS HMAC integrity support [enabled]
FIPS mode enabled for pluto daemon
NSS library is running in FIPS mode
FIPS HMAC integrity verification self-test passed
```

4. To query disabled algorithms in FIPS mode:

```
# ipsec pluto --selftest 2>&1 | grep disabled
Encryption algorithm CAMELLIA_CTR disabled; not FIPS compliant
Encryption algorithm CAMELLIA_CBC disabled; not FIPS compliant
Encryption algorithm SERPENT_CBC disabled; not FIPS compliant
```
Encryption algorithm TWOFISH_CBC disabled; not FIPS compliant
Encryption algorithm TWOFISH_SSH disabled; not FIPS compliant
Encryption algorithm NULL disabled; not FIPS compliant
Encryption algorithm CHACHA20_POLY1305 disabled; not FIPS compliant
Hash algorithm MD5 disabled; not FIPS compliant
PRF algorithm HMAC_MD5 disabled; not FIPS compliant
PRF algorithm AES_XCBC disabled; not FIPS compliant
Integrity algorithm HMAC_SHA2_256_DISABLED disabled; not FIPS compliant
Integrity algorithm AES_XCBC_96_DISABLED disabled; not FIPS compliant
DH algorithm MODP1024 disabled; not FIPS compliant
DH algorithm MODP1536 disabled; not FIPS compliant
DH algorithm DH31 disabled; not FIPS compliant

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

```bash
# ipsec pluto --selftest 2>&1 | grep ESP | grep FIPS | sed "s/^.*FIPS//"
{256,192,*128}  aes_ccm, aes_ccm_c
{256,192,*128}  aes_ccm_b
{256,192,*128}  aes_ccm_a
[*192]  3des
{256,192,*128}  aes_gcm, aes_gcm_c
{256,192,*128}  aes_gcm_b
{256,192,*128}  aes_gcm_a
{256,192,*128}  aesctr
{256,192,*128}  aes
{256,192,*128}  aes_gmac
sha, sha1, sha1_96, hmac_sha1
sha512, sha2_512, sha2_512_256, hmac_sha2_512
sha384, sha2_384, sha2_384_192, hmac_sha2_384
sha2, sha256, sha2_256, sha2_256_128, hmac_sha2_256
aes_cmac
null
null, dh0
dh14
dh15
dh16
dh17
dh18
ecp_256, ecp256
ecp_384, ecp384
ecp_521, ecp521
```

Additional resources

- Using system-wide cryptographic policies.

### 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.
NOTE

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

Prerequisites

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

Procedure

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

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

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

   ```bash
   # 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:

   ```bash
   # systemctl restart ipsec
   ```

Verification

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

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

2. Optionally, check that the Journal log contains entries confirming a successful initialization:
# journalctl -u ipsec
...
pluto[23001]: NSS DB directory: sql:/etc/ipsec.d
pluto[23001]: Initializing NSS
pluto[23001]: Opening NSS database "sql:/etc/ipsec.d" read-only
pluto[23001]: NSS Password from file "/etc/ipsec.d/nsspassword" for token "NSS Certificate DB" with length 20 passed to NSS
pluto[23001]: NSS crypto library initialized
...

Additional resources

- `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 AUTOMATIC DETECTION AND USAGE OF ESP HARDWARE OFFLOAD TO ACCELERATE AN IPSEC CONNECTION
Offloading Encapsulating Security Payload (ESP) to the hardware accelerates IPsec connections over Ethernet. By default, Libreswan detects if hardware supports this feature and, as a result, enables ESP hardware offload. This procedure describes how to enable the automatic detection in case that the feature was disabled or explicitly enabled.

Prerequisites

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

Procedure

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

2. Ensure the `nic-offload` parameter is not set in the connection’s settings.

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

   ```bash
   # systemctl restart ipsec
   ```

Verification

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

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

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

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

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

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

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

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

Additional resources

- Configuring a VPN with IPsec

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

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

Prerequisites

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

Procedure

1. Enable ESP hardware offload support on the network bond:
   ```
   # nmcli connection modify bond0 ethtool.feature-esp-hw-offload on
   ```
   This command enables ESP hardware offload support on the `bond0` connection.

2. Reactivate the `bond0` connection:
   ```
   # nmcli connection up bond0
   ```

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

4. Restart the `ipsec` service:
   ```
   # systemctl restart ipsec
   ```

Verification

1. Display the active port of the bond:
   ```
   # grep "Currently Active Slave" /proc/net/bonding/bond0
   Currently Active Slave: enp1s0
   ```

2. Display the `tx_ipsec` and `rx_ipsec` counters of the active port:
   ```
   # ethtool enp1s0 | egrep "_ipsec"
   tx_ipsec: 10
   rx_ipsec: 10
   ```
3. Send traffic through the IPsec tunnel. For example, ping a remote IP address:

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

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

```bash
# 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
- Configuring a VPN with IPsec
- Configuring a VPN with IPsec chapter in the Securing networks document.

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

**Overriding system-wide crypto-policies for a connection**

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

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

```bash
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:

```bash
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.
4.14. TROUBLESHOOTING IPSEC VPN CONFIGURATIONS

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

Basic connection troubleshooting

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

To confirm that an IPsec VPN connection is established:

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

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

To check that the problem is in the connection:

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

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

2. Next, initiate the VPN connection:

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

Firewall-related problems

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

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

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

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

  ```
  # ipsec auto --up vpn.example.com
  002 "vpn.example.com" #9: initiating Main Mode
  ```
Because the IKE protocol, which is used to set up IPsec, is encrypted, you can troubleshoot only a limited subset of problems using the `tcpdump` tool. If a firewall is dropping IKE or IPsec packets, you can try to find the cause using the `tcpdump` utility. However, `tcpdump` cannot diagnose other problems with IPsec VPN connections.

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

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

**Mismatched algorithms, protocols, and policies**

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

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

  ```
  # ipsec auto --up vpn.example.com
  ...
  000 "vpn.example.com"[1] 192.0.2.2 #16: ERROR: asynchronous network error report on 
  vlp2s0 (192.0.2.2:500), complainant 198.51.100.1: Connection refused [errno 111, origin 
  ICMP type 3 code 3 (not authenticated)]
  ...
  ```

- Example of mismatched IKE algorithms:

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

- Example of mismatched IPsec algorithms:

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

A mismatched IKE version could also result in the remote endpoint dropping the request without a response. This looks identical to a firewall dropping all IKE packets.
• Example of mismatched IP address ranges for IKEv2 (called Traffic Selectors - TS):

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

• Example of mismatched IP address ranges for IKEv1:

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

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

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

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

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

**Maximum transmission unit**

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

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

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

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

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

```
# iptables -I FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --set-mss 1380
```
Network address translation (NAT)

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

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

The system with address 172.16.0.1 have a NAT rule:

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

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

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

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

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

Kernel IPsec subsystem bugs

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

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

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

Libreswan logs

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

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

To show a live log for the ipsec service:

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

If the default level of logging does not reveal your configuration problem, enable debug logs by adding the `plutodebug=all` option to the `config setup` section in the `/etc/ipsec.conf` file.
Note that debug logging produces a lot of entries, and it is possible that either the `journald` or `syslogd` service rate-limits the `syslog` messages. To ensure you have complete logs, redirect the logging to a file. Edit the `/etc/ipsec.conf`, and add the `logfile=/var/log/pluto.log` in the `config setup` section.

Additional resources

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

4.15. ADDITIONAL RESOURCES

- `ipsec(8)`, `ipsec.conf(5)`, `ipsec.secrets(5)`, `ipsec_auto(8)`, and `ipsec_rsasigkey(8)` man pages.
- `/usr/share/doc/libreswan-version/` directory.
- The website of the upstream project.
- The Libreswan Project Wiki.
- All Libreswan man pages.
- NIST Special Publication 800–77: Guide to IPsec VPNs.
CHAPTER 5. CONFIGURING VPN CONNECTIONS WITH IPSEC BY USING THE RHEL VPN SYSTEM ROLE

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

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

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

5.1. CREATING A HOST-TO-HOST VPN WITH IPSEC USING THE VPN SYSTEM ROLE

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

**Prerequisites**

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the VPN System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Engine configures other systems.

On the control node:

- Red Hat Ansible Engine is installed.
- The `rhel-system-roles` package is installed.
- An inventory file which lists the managed nodes.

**Procedure**

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

```yaml
- name: Host to host VPN
  hosts: managed_node1, managed_node2
  roles:
  - rhel-system-roles.vpn
  vars:
    vpn_connections:
    - hosts:
      - managed_node1:
      - managed_node2:
```
This playbook configures the connection `managed_node1-to-managed_node2` using pre-shared key authentication with keys auto-generated by the system role.

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

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

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

**NOTE**

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

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

```yaml
- name: Multiple VPN
  hosts: managed_node1, managed_node2
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - name: control_plane_vpn
        hosts:
          managed_node1:
            hostname: 192.0.2.0 # IP for the control plane
          managed_node2:
            hostname: 192.0.2.1
      - name: data_plane_vpn
        hosts:
          managed_node1:
            hostname: 10.0.0.1 # IP for the data plane
          managed_node2:
            hostname: 10.0.0.2
```

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

3. Optional: Verify playbook syntax.

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

4. Run the playbook on your inventory file:

```bash
# ansible-playbook -i /path/to/file/inventory_file /path/to/file/playbook.yml
```
CHAPTER 5. CONFIGURING VPN CONNECTIONS WITH IPSEC BY USING THE RHEL VPN SYSTEM ROLE

Verification

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

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

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

**NOTE**

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

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

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

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

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

**NOTE**

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

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

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

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

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

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

**Prerequisites**

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the VPN System Role.
  
  * On all the managed nodes, the NSS database in the `/etc/ipsec.d` directory contains all the certificates necessary for peer authentication. By default, the node name is used as the certificate nickname.

- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Engine configures other systems.
  
  On the control node:
  
  * Red Hat Ansible Engine is installed.
  * The `rhel-system-roles` package is installed.
  * An inventory file which lists the managed nodes.

**Procedure**

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

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

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

3. Optional: Verify playbook syntax.

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

4. Run the playbook on your inventory file:

   ```bash
   # ansible-playbook -i inventory_file /path/to/file/playbook.yml
   ```
5.3. ADDITIONAL RESOURCES

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

- For details about the ansible-playbook command, see the ansible-playbook(1) man page.
CHAPTER 6. 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.

6.1. 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"'
50b71a8ef0bd5751ea76de6d6c98c03a
```

b. Create 32-byte hexadecimal CKN:

```
dd if=/dev/urandom count=32 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898f1c4550
```

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 50b71a8ef0bd5751ea76de6d6c98c03a
macsec.mka-ckn f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898f1c4550
```

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 ipv6.addresses
```

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

### 6.2. ADDITIONAL RESOURCES

- [MACsec: a different solution to encrypt network traffic](blog)
CHAPTER 7. USING AND CONFIGURING FIREWALLD

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

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

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

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

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

7.1. GETTING STARTED WITH FIREWALLD

This section provides information about **firewalld**.

7.1.1. When to use firewalld, nftables, or iptables

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

- **firewalld**: Use the **firewalld** utility for simple firewall use cases. The utility is easy to use and covers the typical use cases for these scenarios.

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

- **iptables**: The **iptables** utility on Red Hat Enterprise Linux uses the **nf_tables** kernel API instead of the legacy back end. The **nf_tables** API provides backward compatibility so that scripts that use **iptables** commands still work on Red Hat Enterprise Linux. For new firewall scripts, Red Hat recommends to use **nftables**.

**IMPORTANT**

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

7.1.2. Zones

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

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

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

**block**

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

**dmz**

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

**drop**

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

**external**

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

**home**

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

**internal**

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

**public**

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

**trusted**

All network connections are accepted.

**work**

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

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

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

Additional resources

- The `firewalld.zone(5)` man page.

7.1.3. Predefined services

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

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

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

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

Additional resources

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

7.1.4. Starting firewalld

Procedure

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

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

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

   ```
   # systemctl enable firewalld
   ```

7.1.5. Stopping firewalld

Procedure

1. To stop `firewalld`, enter the following command as `root`:
# systemctl stop firewalld

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

    # systemctl disable firewalld

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

    # systemctl mask firewalld

7.1.6. Verifying the permanent firewalld configuration

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

Prerequisites

- The firewalld service is running.

Procedure

1. Verify the permanent configuration of the firewalld service:

    # firewall-cmd --check-config
    success

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

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

7.2. VIEWING THE CURRENT STATUS AND SETTINGS OF FIREWALLD

This section covers information about viewing current status, allowed services, and current settings of firewalld.

7.2.1. Viewing the current status of firewalld

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

Procedure

1. To see the status of the service:

    # firewall-cmd --state

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

7.2.2. Viewing allowed services using GUI

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

You can start the graphical firewall configuration tool using the command-line.

Prerequisites

- You installed the `firewall-config` package.

Procedure

- To start the graphical firewall configuration tool using the command-line:

  ```
  $ firewall-config
  ```

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

7.2.3. Viewing firewalld settings using CLI

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

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

Procedure

- To list all the relevant information for the default zone:

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

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

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

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

  ```
  # firewall-cmd --help
  ```

- To see which services are allowed in the current zone:

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

**NOTE**

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

### 7.3. CONTROLLING NETWORK TRAFFIC USING FIREWALLD

This section covers information about controlling network traffic using `firewalld`.

#### 7.3.1. Disabling all traffic in case of emergency using CLI

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

**Procedure**

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

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

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

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

   # firewall-cmd --panic-off

Verification

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

   # firewall-cmd --query-panic

7.3.2. Controlling traffic with predefined services using CLI

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

Procedure

1. Check that the service is not already allowed:

   # firewall-cmd --list-services

   ssh dhcpv6-client

2. List all predefined services:

   # firewall-cmd --get-services

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

3. Add the service to the allowed services:

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

4. Make the new settings persistent:

   # firewall-cmd --runtime-to-permanent

7.3.3. Controlling traffic with predefined services using GUI

This procedure describes how to control the network traffic with predefined services using graphical user interface.

Prerequisites
You installed the `firewall-config` package

**Procedure**

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

2. To edit a service:
   a. Start the `firewall-config` tool.
   b. Select **Permanent** from the menu labeled **Configuration**. Additional icons and menu buttons appear at the bottom of the **Services** window.
   c. Select the service you want to configure.

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

**NOTE**

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

### 7.3.4. Adding new services

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

**NOTE**

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

**Procedure**

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

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

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

2. To add a new service using a local file, use the following command:
3. As soon as service settings are changed, an updated copy of the service is placed into 
/etc/firewalld/services/.
As root, you can enter the following command to copy a service manually:

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

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

### 7.3.5. Opening ports using GUI

To permit traffic through the firewall to a certain port, you can open the port in the GUI.

**Prerequisites**

- You installed the `firewall-config` package

**Procedure**

1. Start the `firewall-config` tool and select the network zone whose settings you want to change.
2. Select the **Ports** tab and click the **Add** button on the right-hand side. The **Port and Protocol** 
   window opens.
3. Enter the port number or range of ports to permit.
4. Select **tcp** or **udp** from the list.

### 7.3.6. Controlling traffic with protocols using GUI

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

**Prerequisites**

- You installed the `firewall-config` package

**Procedure**

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

To permit traffic through the firewall from a certain port, you can use the GUI.

Prerequisites

- You installed the `firewall-config` package

Procedure

1. Start the firewall-config tool and select the network zone whose settings you want to change.
2. Select the Source Port tab and click the Add button on the right-hand side. The Source Port window opens.
3. Enter the port number or range of ports to permit. Select tcp or udp from the list.

7.4. CONTROLLING PORTS USING CLI

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

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

7.4.1. Opening a port

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

Procedure

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

1. List all allowed ports:

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

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

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

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

3. Make the new settings persistent:

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

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

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

Procedure

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

1. List all allowed ports:

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

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

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

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

3. Make the new settings persistent:

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

7.5. WORKING WITH FIREWALLD ZONES

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

7.5.1. Listing zones

This procedure describes how to list zones using the command line.

Procedure

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

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

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

2. To see detailed information for all zones:
7.5.2. Modifying firewalld settings for a certain zone

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

Procedure

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

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

7.5.3. Changing the default zone

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

Procedure

To set up the default zone:

1. Display the current default zone:

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

2. Set the new default zone:

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

NOTE

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

7.5.4. Assigning a network interface to a zone

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

Procedure

To assign the zone to a specific interface:

1. List the active zones and the interfaces assigned to them:
# firewall-cmd --get-active-zones

2. Assign the interface to a different zone:

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

7.5.5. Assigning a zone to a connection using nmcli

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

Procedure

1. Assign the zone to the NetworkManager connection profile:

   # nmcli connection modify profile connection.zone zone_name

2. Activate the connection:

   # nmcli connection up profile

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

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

Procedure

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

   ZONE=zone_name

7.5.7. Creating a new zone

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

Procedure

1. Create a new zone:

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

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

   # firewall-cmd --get-zones

3. Make the new settings persistent:
7.5.8. Zone configuration files

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

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

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

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

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

Additional resources

- firewalld.zone manual page

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

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

- **ACCEPT**: Accepts all incoming packets except those disallowed by specific rules.

- **REJECT**: Rejects all incoming packets except those allowed by specific rules. When firewalld rejects packets, the source machine is informed about the rejection.

- **DROP**: Drops all incoming packets except those allowed by specific rules. When firewalld drops packets, the source machine is not informed about the packet drop.

- **default**: Similar behavior as for REJECT, but with special meanings in certain scenarios. For details, see the Options to Adapt and Query Zones and Policies section in the firewall-cmd(1) man page.

Procedure

To set a target for a zone:

1. List the information for the specific zone to see the default target:
# firewall-cmd --zone=zone-name --list-all

2. Set a new target in the zone:

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

Additional resources

- `firewall-cmd(1)` man page

## 7.6. USING ZONES TO MANAGE INCOMING TRAFFIC DEPENDING ON A SOURCE

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

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

### 7.6.1. Adding a source

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

**NOTE**

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

- To set the source in the current zone:

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

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

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

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

**Procedure**

1. List all available zones:

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

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

   ```bash
   # firewall-cmd --zone=trusted --add-source=192.168.2.15
   ```
3. Make the new settings persistent:

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

### 7.6.2. Removing a source

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

**Procedure**

1. List allowed sources for the required zone:

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

2. Remove the source from the zone permanently:

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

3. Make the new settings persistent:

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

### 7.6.3. Adding a source port

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

**Procedure**

- To add a source port:

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

### 7.6.4. Removing a source port

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

**Procedure**

- To remove a source port:

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

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

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

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

### Procedure

1. List all available zones:

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

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

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

3. Add the http service to the internal zone:

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

4. Make the new settings persistent:

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

### Verification

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

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

### Additional resources

- [firewalld.zones(5) man page](#)

---

**7.7. FILTERING FORWARDED TRAFFIC BETWEEN ZONES**

With a policy object, users can group different identities that require similar permissions in the policy. You can apply policies depending on the direction of the traffic.
The policy objects feature provides forward and output filtering in firewalld. The following describes the usage of firewalld to filter traffic between different zones to allow access to locally hosted VMs to connect the host.

### 7.7.1. The relationship between policy objects and zones

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

```bash
# firewall-cmd --permanent --new-policy myOutputPolicy
# firewall-cmd --permanent --policy myOutputPolicy --add-ingress-zone HOST
# firewall-cmd --permanent --policy myOutputPolicy --add-egress-zone ANY
```

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

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

### 7.7.2. Using priorities to sort policies

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

To set a priority to sort the policies:

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

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

The following rules apply to policy priorities:

- Policies with negative priorities apply before rules in zones.
- Policies with positive priorities apply after rules in zones.
- Priority 0 is reserved and hence is unusable.

### 7.7.3. Using policy objects to filter traffic between locally hosted Containers and a network physically connected to the host

The policy objects feature allows users to filter their container and virtual machine traffic.

#### Procedure

1. Create a new policy.

```bash
# firewall-cmd --permanent --new-policy podmanToHost
```
2. Block all traffic.

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

**NOTE**

Red Hat recommends that you block all traffic to the host by default and then selectively open the services you need for the host.

3. Define the ingress zone to use with the policy.

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

4. Define the egress zone to use with the policy.

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

**Verification**

- Verify information about the policy.

```
# firewall-cmd --info-policy podmanToHost
```

### 7.7.4. Setting the default target of policy objects

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

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

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

**Verification**

- Verify information about the policy

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

### 7.8. CONFIGURING NAT USING FIREWALLD

With `firewalld`, you can configure the following network address translation (NAT) types:
7.8.1. The different NAT types: masquerading, source NAT, destination NAT, and redirect

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

**Masquerading and source NAT (SNAT)**

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers do not route private IP ranges, such as `10.0.0.0/8`. If you use private IP ranges in your network and users should be able to reach servers on the Internet, map the source IP address of packets from these ranges to a public IP address.

Both masquerading and SNAT are very similar. The differences are:

- Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.

- SNAT sets the source IP address of packets to a specified IP and does not dynamically look up the IP of the outgoing interface. Therefore, SNAT is faster than masquerading. Use SNAT if the outgoing interface uses a fixed IP address.

**Destination NAT (DNAT)**

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

**Redirect**

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

7.8.2. Configuring IP address masquerading

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

**Procedure**

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

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

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

2. To enable IP masquerading, enter the following command as `root`:
# firewall-cmd --zone=external --add-masquerade

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

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

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

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

7.9. PORT FORWARDING

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

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

7.9.1. Adding a port to redirect

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

**Prerequisites**

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

**Procedure**

1. To redirect a port to another port:

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

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

   a. Add the port to be forwarded:

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

   b. Enable masquerade:

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

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

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

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

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

2. Make the new settings persistent:

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

3. Check that the port is redirected:

   ```shell
   # firewall-cmd --list-all
   ```

7.9.3. Removing a redirected port

This procedure describes how to remove the redirected port.

Procedure

1. To remove a redirected port:

   ```shell
   # firewall-cmd --remove-forward-port=port=port-number:proto=<tcp|udp>:toport=port-number:toaddr=<IP>
   ```

2. To remove a forwarded port redirected to a different address:
   a. Remove the forwarded port:

      ```shell
      # firewall-cmd --remove-forward-port=port=port-number:proto=<tcp|udp>:toport=port-number:toaddr=<IP>
      ```

   b. Disable masquerade:

      ```shell
      # firewall-cmd --remove-masquerade
      ```

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

This procedure describes how to remove the port redirection.

Procedure

1. List redirected ports:

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

2. Remove the redirected port from the firewall:

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

3. Make the new settings persistent:
7.10. MANAGING ICMP REQUESTS

The **Internet Control Message Protocol (ICMP)** is a supporting protocol that is used by various network devices to send error messages and operational information indicating a connection problem, for example, that a requested service is not available. **ICMP** differs from transport protocols such as TCP and UDP because it is not used to exchange data between systems.

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

### 7.10.1. Listing and blocking ICMP requests

#### Listing ICMP requests

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

- To list all available **ICMP** types:
  ```
  # firewall-cmd --get-icmptypes
  ```

- The **ICMP** request can be used by IPv4, IPv6, or by both protocols. To see for which protocol the **ICMP** request has used:
  ```
  # firewall-cmd --info-icmptype=<icmptype>
  ```

- The status of an **ICMP** request shows **yes** if the request is currently blocked or **no** if it is not. To see if an **ICMP** request is currently blocked:
  ```
  # firewall-cmd --query-icmp-block=<icmptype>
  ```

#### Blocking or unblocking ICMP requests

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

- To see if an **ICMP** request is currently blocked:
  ```
  # firewall-cmd --query-icmp-block=<icmptype>
  ```

- To block an **ICMP** request:
  ```
  # firewall-cmd --add-icmp-block=<icmptype>
  ```

- To remove the block for an **ICMP** request:
  ```
  # firewall-cmd --remove-icmp-block=<icmptype>
  ```
Blocking ICMP requests without providing any information at all

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

- To block and drop all ICMP requests:
  
  ```
  # firewall-cmd --remove-icmp-block=<icmptype>
  ```

- Set the target of your zone to DROP:
  
  ```
  # firewall-cmd --permanent --set-target=DROP
  ```

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

To block and drop certain ICMP requests and allow others:

1. Set the target of your zone to DROP:
   
   ```
   # firewall-cmd --permanent --set-target=DROP
   ```

2. Add the ICMP block inversion to block all ICMP requests at once:
   
   ```
   # firewall-cmd --add-icmp-block-inversion
   ```

3. Add the ICMP block for those ICMP requests that you want to allow:
   
   ```
   # firewall-cmd --add-icmp-block=<icmptype>
   ```

4. Make the new settings persistent:
   
   ```
   # firewall-cmd --runtime-to-permanent
   ```

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

To revert the block inversion to a fully permissive setting:

1. Set the target of your zone to default or ACCEPT:
   
   ```
   # firewall-cmd --permanent --set-target=default
   ```

2. Remove all added blocks for ICMP requests:
   
   ```
   # firewall-cmd --remove-icmp-block=<icmptype>
   ```

3. Remove the ICMP block inversion:
   
   ```
   # firewall-cmd --remove-icmp-block-inversion
   ```
4. Make the new settings persistent:

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

### 7.10.2. Configuring the ICMP filter using GUI

- To enable or disable an **ICMP** filter, start the **firewall-config** tool and select the network zone whose messages are to be filtered. Select the **ICMP Filter** tab and select the check box for each type of **ICMP** message you want to filter. Clear the check box to disable a filter. This setting is per direction and the default allows everything.

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

### 7.11. SETTING AND CONTROLLING IP SETS USING FIREWALLD

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

```
~]# firewall-cmd --get-ipset-types
```

#### 7.11.1. Configuring IP set options using CLI

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  ```bash
  # firewall-cmd --permanent --zone=drop --source=ipset=test
  success
  ```
command as root:

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

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

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

---

### WARNING

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

---

### 7.12. PRIORITIZING RICH RULES

By default, rich rules are organized based on their rule action. For example, `deny` rules have precedence over `allow` rules. The `priority` parameter in rich rules provides administrators fine-grained control over rich rules and their execution order.

#### 7.12.1. How the priority parameter organizes rules into different chains

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

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

- Priority lower than 0: the rule is redirected into a chain with the `_pre` suffix.
- Priority higher than 0: the rule is redirected into a chain with the `_post` suffix.
- Priority equals 0: based on the action, the rule is redirected into a chain with the `_log`, `_deny`, or `_allow` the action.

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

#### 7.12.2. Setting the priority of a rich rule

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

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

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

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

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

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

### 7.13. CONFIGURING FIREWALL LOCKDOWN

Local applications or services are able to change the firewall configuration if they are running as `root` (for example, `libvirt`). With this feature, the administrator can lock the firewall configuration so that either no applications or only applications that are added to the lockdown allow list are able to request firewall changes. The lockdown settings default to disabled. If enabled, the user can be sure that there are no unwanted configuration changes made to the firewall by local applications or services.

#### 7.13.1. Configuring lockdown using CLI

This procedure describes how to enable or disable lockdown using the command line.

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

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

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

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

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

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

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

#### 7.13.2. Configuring lockdown allowlist options using CLI

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

- The context is the security (SELinux) context of a running application or service. To get the context of a running application use the following command:
That command returns all running applications. Pipe the output through the `grep` tool to get the application of interest. For example:

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

- To list all command lines that are in the allowlist, enter the following command as **root**:
  ```
  # firewall-cmd --list-lockdown-whitelist-commands
  ```

- To add a command *command* to the allowlist, enter the following command as **root**:
  ```
  # firewall-cmd --add-lockdown-whitelist-command='/usr/bin/python3 -Es /usr/bin/command'
  ```

- To remove a command *command* from the allowlist, enter the following command as **root**:
  ```
  # firewall-cmd --remove-lockdown-whitelist-command='/usr/bin/python3 -Es /usr/bin/command'
  ```

- To query whether the command *command* is in the allowlist, enter the following command as **root**:
  ```
  # firewall-cmd --query-lockdown-whitelist-command='/usr/bin/python3 -Es /usr/bin/command'
  ```
  The command prints **yes** with exit status **0** if true. It prints **no** with exit status **1** otherwise.

- To list all security contexts that are in the allowlist, enter the following command as **root**:
  ```
  # firewall-cmd --list-lockdown-whitelist-contexts
  ```

- To add a context *context* to the allowlist, enter the following command as **root**:
  ```
  # firewall-cmd --add-lockdown-whitelist-context=context
  ```

- To remove a context *context* from the allowlist, enter the following command as **root**:
  ```
  # firewall-cmd --remove-lockdown-whitelist-context=context
  ```

- To query whether the context *context* is in the allowlist, enter the following command as **root**:
  ```
  # firewall-cmd --query-lockdown-whitelist-context=context
  ```
  Prints **yes** with exit status **0**, if true, prints **no** with exit status **1** otherwise.

- To list all user IDs that are in the allowlist, enter the following command as **root**:
  ```
  # firewall-cmd --list-lockdown-whitelist-uids
  ```

- To add a user ID *uid* to the allowlist, enter the following command as **root**:
  ```
  # firewall-cmd --add-lockdown-whitelist-uid=uid
  ```
To remove a user ID `uid` from the allowlist, enter the following command as `root`:

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

To query whether the user ID `uid` is in the allowlist, enter the following command:

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

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

To list all user names that are in the allowlist, enter the following command as `root`:

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

To add a user name `user` to the allowlist, enter the following command as `root`:

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

To remove a user name `user` from the allowlist, enter the following command as `root`:

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

To query whether the user name `user` is in the allowlist, enter the following command:

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

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

### 7.13.3. Configuring lockdown allowlist options using configuration files

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

+ The allowlist configuration files are stored in the `/etc/firewalld/` directory.

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

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

```xml
<?xml version="1.0" encoding="utf-8"?>
<whitelist>
  <command name="/usr/libexec/platform-python -s /bin/firewall-cmd"/>
  <selinux context="system_u:system_r:NetworkManager_t:s0"/>
  <user id="815"/>
  <user name="user"/>
</whitelist>
```
This example shows both user id and user name, but only one option is required. Python is the interpreter and is prepended to the command line. You can also use a specific command, for example:

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

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

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

The * at the end of the name attribute of a command means that all commands that start with this string match. If the * is not there then the absolute command including arguments must match.

### 7.14. ENABLING TRAFFIC FORWARDING BETWEEN DIFFERENT INTERFACES OR SOURCES WITHIN A FIREWALLD ZONE

Intra-zone forwarding is a firewalld feature that enables traffic forwarding between interfaces or sources within a firewalld zone.

#### 7.14.1. The difference between intra-zone forwarding and zones with the default target set to ACCEPT

When intra-zone forwarding is enabled, the traffic within a single firewalld zone can flow from one interface or source to another interface or source. The zone specifies the trust level of interfaces and sources. If the trust level is the same, communication between interfaces or sources is possible.

Note that, if you enable intra-zone forwarding in the default zone of firewalld, it applies only to the interfaces and sources added to the current default zone.

The trusted zone of firewalld uses a default target set to ACCEPT. This zone accepts all forwarded traffic, and intra-zone forwarding is not applicable for it.

As for other default target values, forwarded traffic is dropped by default, which applies to all standard zones except the trusted zone.

#### 7.14.2. Using intra-zone forwarding to forward traffic between an Ethernet and Wi-Fi network

You can use intra-zone forwarding to forward traffic between interfaces and sources within the same firewalld zone. For example, use this feature to forward traffic between an Ethernet network connected to enp1s0 and a Wi-Fi network connected to wlp0s20.

**Procedure**

1. Enable packet forwarding in the kernel:

   ```
   # echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
   # sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
   ```
2. Ensure that interfaces between which you want to enable intra-zone forwarding are not assigned to a zone different than the `internal` zone:

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

3. If the interface is currently assigned to a zone other than `internal`, reassign it:

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

4. Add the `enp1s0` and `wlp0s20` interfaces to the `internal` zone:

   ```
   # firewall-cmd --zone=internal --add-interface=enp1s0 --add-interface=wlp0s20
   ```

5. Enable intra-zone forwarding:

   ```
   # firewall-cmd --zone=internal --add-forward
   ```

**Verification**

The following verification steps require that the `nmap-ncat` package is installed on both hosts.

1. Log in to a host that is in the same network as the `enp1s0` interface of the host you enabled zone forwarding on.

2. Start an echo service with `ncat` to test connectivity:

   ```
   # ncat -e /usr/bin/cat -l 12345
   ```

3. Log in to a host that is in the same network as the `wlp0s20` interface.

4. Connect to the echo server running on the host that is in the same network as the `enp1s0`:

   ```
   # ncat <other host> 12345
   ```

5. Type something and press `Enter`, and verify the text is sent back.

**Additional resources**

- `firewalld.zones(5)` man page

**7.15. ADDITIONAL RESOURCES**

- `firewalld(1)` man page
- `firewalld.conf(5)` man page
- `firewall-cmd(1)` man page
- `firewall-config(1)` man page
- `firewall-offline-cmd(1)` man page
- `firewalld.icmptype(5)` man page
• firewalld.ipset(5) man page
• firewalld.service(5) man page
• firewalld.zone(5) man page
• firewalld.direct(5) man page
• firewalld.lockdown-whitelist(5)
• firewalld.richlanguage(5)
• firewalld.zones(5) man page
• firewalld.dbus(5) man page
CHAPTER 8. GETTING STARTED WITH NFTABLES

The **nftables** framework provides packet classification facilities. The most notable features are:

- built-in lookup tables instead of linear processing
- a single framework for both the **IPv4** and **IPv6** protocols
- rules all applied atomically instead of fetching, updating, and storing a complete rule set
- support for debugging and tracing in the rule set (**nftrace**) and monitoring trace events (in the **nft** tool)
- more consistent and compact syntax, no protocol-specific extensions
- a Netlink API for third-party applications

The **nftables** framework uses tables to store chains. The chains contain individual rules for performing actions. The **libnftnl** library can be used for low-level interaction with **nftables** Netlink API over the **libmnl** library.

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

8.1. MIGRATING FROM IPTABLES TO NFTABLES

If your firewall configuration still uses **iptables** rules, you can migrate your **iptables** rules to **nftables**.

8.1.1. When to use firewalld, nftables, or iptables

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

- **firewalld**: Use the **firewalld** utility for simple firewall use cases. The utility is easy to use and covers the typical use cases for these scenarios.

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

- **iptables**: The **iptables** utility on Red Hat Enterprise Linux uses the **nf_tables** kernel API instead of the legacy back end. The **nf_tables** API provides backward compatibility so that scripts that use **iptables** commands still work on Red Hat Enterprise Linux. For new firewall scripts, Red Hat recommends to use **nftables**.

  **IMPORTANT**

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

8.1.2. Converting iptables rules to nftables rules

Red Hat Enterprise Linux provides the **iptables-translate** and **ip6tables-translate** tools to convert existing **iptables** or **ip6tables** rules into the equivalent ones for **nftables**.
Note that some extensions lack translation support. If such an extension exists, the tool prints the untranslated rule prefixed with the `#` sign. For example:

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

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

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

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

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

### 8.1.3. Comparison of common `iptables` and `nftables` commands

The following is a comparison of common `iptables` and `nftables` commands:

- **Listing all rules:**

<table>
<thead>
<tr>
<th><code>iptables</code></th>
<th><code>nftables</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iptables-save</code></td>
<td><code>nft list ruleset</code></td>
</tr>
</tbody>
</table>

- **Listing a certain table and chain:**

<table>
<thead>
<tr>
<th><code>iptables</code></th>
<th><code>nftables</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iptables -L</code></td>
<td><code>nft list table ip filter</code></td>
</tr>
<tr>
<td><code>iptables -L INPUT</code></td>
<td><code>nft list chain ip filter INPUT</code></td>
</tr>
<tr>
<td><code>iptables -t nat -L PREROUTING</code></td>
<td><code>nft list chain ip nat PREROUTING</code></td>
</tr>
</tbody>
</table>

The `nft` command does not pre-create tables and chains. They exist only if a user created them manually.

**Example: Listing rules generated by firewalld**

```
# nft list table inet firewalld
# nft list table ip firewalld
# nft list table ip6 firewalld
```

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

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

- add comments
- define variables
- include other rule set files

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

When you install the **nftables** package, Red Hat Enterprise Linux automatically creates *.nft scripts in the `/etc/nftables/` directory. These scripts contain commands that create tables and empty chains for different purposes.

### 8.2.1. Supported nftables script formats

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

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

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

# Flush the rule set
flush ruleset

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

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

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

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

# Flush the rule set
flush ruleset

# Create a table
add table inet example_table

# Create a chain for incoming packets that drops all packets that
# are not explicitly allowed by any rule in this chain
add chain inet example_table example_chain { type filter hook input priority 0; policy drop; }
```
# Add a rule that accepts connections to port 22 (ssh)
add rule inet example_table example_chain tcp dport ssh accept

8.2.2. Running nftables scripts

You can run `nftables` script either by passing it to the `nft` utility or execute the script directly.

Prerequisites

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

Procedure

- To run an `nftables` script by passing it to the `nft` utility, enter:

  ```bash
  # nft -f /etc/nftables/example_firewall.nft
  ```

- To run an `nftables` script directly:
  a. Steps that are required only once:
     i. Ensure that the script starts with the following shebang sequence:

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

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

     ii. Optional: Set the owner of the script to `root`:

        ```bash
        # chown root /etc/nftables/example_firewall.nft
        ```

     iii. Make the script executable for the owner:

        ```bash
        # chmod u+x /etc/nftables/example_firewall.nft
        ```

  b. Run the script:

     ```bash
     # /etc/nftables/example_firewall.nft
     ```

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

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

Additional resources
8.2.3. Using comments in nftables scripts

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

Example 8.1. Comments in an nftables script

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

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

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

8.2.4. Using variables in an nftables script

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

Variables with a single value

The following example defines a variable named `INET_DEV` with the value `enp1s0`:

```
define INET_DEV = enp1s0
```

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

```
...  
add rule inet example_table example_chain ifname $INET_DEV tcp dport ssh accept
...  
```

Variables that contain an anonymous set

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

```
define DNS_SERVERS = { 192.0.2.1, 192.0.2.2 }
```

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

```
add rule inet example_table example_chain ip daddr $DNS_SERVERS accept
```

**NOTE**

Note that curly braces have special semantics when you use them in a rule because they indicate that the variable represents a set.
8.2.5. Including files in an nftables script

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

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

**Example 8.2. Including files from the default search directory**

To include a file from the default search directory:

```plaintext
include "example.nft"
```

**Example 8.3. Including all *.nft files from a directory**

To include all files ending with `*.nft` that are stored in the `/etc/nftables/rulesets/` directory:

```plaintext
include "/etc/nftables/rulesets/*.nft"
```

Note that the `include` statement does not match files beginning with a dot.

Additional resources

- The Include files section in the `nft(8)` man page

8.2.6. Automatically loading nftables rules when the system boots

The nftables systemd service loads firewall scripts that are included in the `/etc/sysconfig/nftables.conf` file. This section explains how to load firewall rules when the system boots.

**Prerequisites**

- The nftables scripts are stored in the `/etc/nftables/` directory.

**Procedure**

1. Edit the `/etc/sysconfig/nftables.conf` file.

   - If you enhance `*.nft` scripts created in `/etc/nftables/` when you installed the nftables package, uncomment the `include` statement for these scripts.

   - If you write scripts from scratch, add `include` statements to include these scripts. For example, to load the `/etc/nftables/example.nft` script when the nftables service starts, add:
include "/etc/nftables/example.nft"

2. Optionally, start the `nftables` service to load the firewall rules without rebooting the system:
   # systemctl start nftables

3. Enable the `nftables` service.
   # systemctl enable nftables

Additional resources
- Supported nftables script formats

8.3. CREATING AND MANAGING NFTABLES TABLES, CHAINS, AND RULES

This section explains how to display nftables rule sets, and how to manage them.

8.3.1. Standard chain priority values and textual names

When you create a chain, the `priority` you can either set an integer value or a standard name that specifies the order in which chains with the same `hook` value traverse.

The names and values are defined based on what priorities are used by `xtables` when registering their default chains.

**NOTE**

The `nft list chains` command displays textual priority values by default. You can view the numeric value by passing the `-y` option to the command.

**Example 8.4. Using a textual value to set the priority**

The following command creates a chain named `example_chain` in `example_table` using the standard priority value 50:

```
# nft add chain inet example_table example_chain { type filter hook input priority 50; policy accept; }
```

Because the priority is a standard value, you can alternatively use the textual value:

```
# nft add chain inet example_table example_chain { type filter hook input priority security; policy accept; }
```

**Table 8.1. Standard priority names, family, and hook compatibility matrix**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Families</th>
<th>Hooks</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw</td>
<td>-300</td>
<td>ip, ip6, inet</td>
<td>all</td>
</tr>
</tbody>
</table>
All families use the same values, but the **bridge** family uses following values:

Table 8.2. Standard priority names, and hook compatibility for the bridge family

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Families</th>
<th>Hooks</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstnat</td>
<td>-300</td>
<td>ip, ip6, inet</td>
<td>prerouting</td>
</tr>
<tr>
<td>filter</td>
<td>-200</td>
<td>ip, ip6, inet, arp, netdev</td>
<td>all</td>
</tr>
<tr>
<td>out</td>
<td>100</td>
<td>ip, ip6, inet</td>
<td>output</td>
</tr>
<tr>
<td>srcnat</td>
<td>300</td>
<td>ip, ip6, inet</td>
<td>postrouting</td>
</tr>
</tbody>
</table>

Additional resources

- The **Chains** section in the **nft(8)** man page

8.3.2. Displaying the nftables rule set

The rule sets of **nftables** contain tables, chains, and rules. This section explains how to display the rule set.

Procedure

- To display the rule set, enter:

```bash
# nft list ruleset
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
tcp dport http accept
tcp dport ssh accept
  }
}
```
8.3.3. Creating an nftables table

A table in nftables is a name space that contains a collection of chains, rules, sets, and other objects. This section explains how to create a table.

Each table must have an address family defined. The address family of a table defines what address types the table processes. You can set one of the following address families when you create a table:

- **ip**: Matches only IPv4 packets. This is the default if you do not specify an address family.
- **ip6**: Matches only IPv6 packets.
- **inet**: Matches both IPv4 and IPv6 packets.
- **arp**: Matches IPv4 address resolution protocol (ARP) packets.
- **bridge**: Matches packets that traverse a bridge device.
- **netdev**: Matches packets from ingress.

**Procedure**

1. Use the `nft add table` command to create a new table. For example, to create a table named `example_table` that processes IPv4 and IPv6 packets:

   ```
   # nft add table inet example_table
   ```

2. Optionally, list all tables in the rule set:

   ```
   # nft list tables
table inet example_table
   ```

**Additional resources**

- The *Address families* section in the *nft(8)* man page
- The *Tables* section in the *nft(8)* man page

8.3.4. Creating an nftables chain

Chains are containers for rules. The following two rule types exists:

- **Base chain**: You can use base chains as an entry point for packets from the networking stack.
- **Regular chain**: You can use regular chains as a `jump` target and to better organize rules.

The procedure describes how to add a base chain to an existing table.

**Prerequisites**
The table to which you want to add the new chain exists.

Procedure

1. Use the `nft add chain` command to create a new chain. For example, to create a chain named `example_chain` in `example_table`:

```
# nft add chain inet example_table example_chain { type filter hook input priority 0; policy accept; }
```

**IMPORTANT**

To avoid that the shell interprets the semicolons as the end of the command, prepend the semicolons the \ escape character.

This chain filters incoming packets. The `priority` parameter specifies the order in which `nftables` processes chains with the same hook value. A lower priority value has precedence over higher ones. The `policy` parameter sets the default action for rules in this chain. Note that if you are logged in to the server remotely and you set the default policy to `drop`, you are disconnected immediately if no other rule allows the remote access.

2. Optionally, display all chains:

```
# nft list chains
table inet example_table {
    chain example_chain {
        type filter hook input priority filter; policy accept;
    }
}
```

Additional resources

- The `Address families` section in the `nft(8)` man page
- The `Chains` section in the `nft(8)` man page

### 8.3.5. Appending a rule to the end of an nftables chain

This section explains how to append a rule to the end of an existing `nftables` chain.

**Prerequisites**

- The chain to which you want to add the rule exists.

**Procedure**

1. To add a new rule, use the `nft add rule` command. For example, to add a rule to the `example_chain` in the `example_table` that allows TCP traffic on port 22:

```
# nft add rule inet example_table example_chain tcp dport 22 accept
```
Instead of the port number, you can alternatively specify the name of the service. In the example, you could use `ssh` instead of the port number `22`. Note that a service name is resolved to a port number based on its entry in the `/etc/services` file.

2. Optionally, display all chains and their rules in `example_table`:

```bash
# nft list table inet example_table
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept
  }
}
```

Additional resources

- The **Address families** section in the `nft(8)` man page
- The **Rules** section in the `nft(8)` man page

### 8.3.6. Inserting a rule at the beginning of an nftables chain

This section explains how to insert a rule at the beginning of an existing `nftables` chain.

**Prerequisites**

- The chain to which you want to add the rule exists.

**Procedure**

1. To insert a new rule, use the `nft insert rule` command. For example, to insert a rule to the `example_chain` in the `example_table` that allows TCP traffic on port 22:

   ```bash
   # nft insert rule inet example_table example_chain tcp dport 22 accept
   ```

   You can alternatively specify the name of the service instead of the port number. In the example, you could use `ssh` instead of the port number `22`. Note that a service name is resolved to a port number based on its entry in the `/etc/services` file.

2. Optionally, display all chains and their rules in `example_table`:

   ```bash
   # nft list table inet example_table
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept
  }
}
```

Additional resources

- The **Address families** section in the `nft(8)` man page
8.3.7. Inserting a rule at a specific position of an nftables chain

This section explains how to insert rules before and after an existing rule in an nftables chain. This way you can place new rules at the right position.

Prerequisites

- The chain to which you want to add the rules exists.

Procedure

1. Use the `nft -a list ruleset` command to display all chains and their rules in the `example_table` including their handle:

   ```
   # nft -a list table inet example_table
   table inet example_table { # handle 1
     chain example_chain { # handle 1
       type filter hook input priority filter; policy accept;
       tcp dport 22 accept # handle 2
       tcp dport 443 accept # handle 3
       tcp dport 389 accept # handle 4
     }
   }
   ```

   Using the `-a` displays the handles. You require this information to position the new rules in the next steps.

2. Insert the new rules to the `example_chain` chain in the `example_table`:

   - To insert a rule that allows TCP traffic on port 636 before handle 3, enter:
     ```
     # nft insert rule inet example_table example_chain position 3 tcp dport 636 accept
     ```

   - To add a rule that allows TCP traffic on port 80 after handle 3, enter:
     ```
     # nft add rule inet example_table example_chain position 3 tcp dport 80 accept
     ```

3. Optionally, display all chains and their rules in `example_table`:

   ```
   # nft -a list table inet example_table
   table inet example_table { # handle 1
     chain example_chain { # handle 1
       type filter hook input priority filter; policy accept;
       tcp dport 22 accept # handle 2
       tcp dport 636 accept # handle 5
       tcp dport 443 accept # handle 3
       tcp dport 80 accept # handle 6
       tcp dport 389 accept # handle 4
     }
   }
   ```

Additional resources
8.4. CONFIGURING NAT USING NFTABLES

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

- Masquerading
- Source NAT (SNAT)
- Destination NAT (DNAT)
- Redirect

**IMPORTANT**

You can only use real interface names in **iifname** and **oifname** parameters, and alternative names (**altname**) are not supported.

8.4.1. The different NAT types: masquerading, source NAT, destination NAT, and redirect

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

**Masquerading and source NAT (SNAT)**

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers do not route private IP ranges, such as `10.0.0.0/8`. If you use private IP ranges in your network and users should be able to reach servers on the Internet, map the source IP address of packets from these ranges to a public IP address.

Both masquerading and SNAT are very similar. The differences are:

- Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.

- SNAT sets the source IP address of packets to a specified IP and does not dynamically look up the IP of the outgoing interface. Therefore, SNAT is faster than masquerading. Use SNAT if the outgoing interface uses a fixed IP address.

**Destination NAT (DNAT)**

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

**Redirect**

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

8.4.2. Configuring masquerading using nftables
Masquerading enables a router to dynamically change the source IP of packets sent through an interface to the IP address of the interface. This means that if the interface gets a new IP assigned, `nftables` automatically uses the new IP when replacing the source IP.

The following procedure describes how to replace the source IP of packets leaving the host through the `ens3` interface to the IP set on `ens3`.

**Procedure**

1. Create a table:
   ```shell
   # nft add table nat
   ```

2. Add the `prerouting` and `postrouting` chains to the table:
   ```shell
   # nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
   # nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
   ```

   **IMPORTANT**

   Even if you do not add a rule to the `prerouting` chain, the `nftables` framework requires this chain to match incoming packet replies.

   Note that you must pass the `--` option to the `nft` command to avoid that the shell interprets the negative priority value as an option of the `nft` command.

3. Add a rule to the `postrouting` chain that matches outgoing packets on the `ens3` interface:
   ```shell
   # nft add rule nat postrouting oifname "ens3" masquerade
   ```

### 8.4.3. Configuring source NAT using nftables

On a router, Source NAT (SNAT) enables you to change the IP of packets sent through an interface to a specific IP address.

The following procedure describes how to replace the source IP of packets leaving the router through the `ens3` interface to `192.0.2.1`.

**Procedure**

1. Create a table:
   ```shell
   # nft add table nat
   ```

2. Add the `prerouting` and `postrouting` chains to the table:
   ```shell
   # nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
   # nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
   ```
IMPORTANT

Even if you do not add a rule to the `postrouting` chain, the `nftables` framework requires this chain to match outgoing packet replies.

Note that you must pass the `--` option to the `nft` command to avoid that the shell interprets the negative priority value as an option of the `nft` command.

3. Add a rule to the `postrouting` chain that replaces the source IP of outgoing packets through `ens3` with `192.0.2.1`:

```bash
# nft add rule nat postrouting oifname "ens3" snat to 192.0.2.1
```

Additional resources

- Forwarding incoming packets on a specific local port to a different host

8.4.4. Configuring destination NAT using nftables

Destination NAT enables you to redirect traffic on a router to a host that is not directly accessible from the Internet.

The following procedure describes how to redirect incoming traffic sent to port `80` and `443` of the router to the host with the `192.0.2.1` IP address.

Procedure

1. Create a table:

```bash
# nft add table nat
```

2. Add the `prerouting` and `postrouting` chains to the table:

```bash
# nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```

IMPORTANT

Even if you do not add a rule to the `postrouting` chain, the `nftables` framework requires this chain to match outgoing packet replies.

Note that you must pass the `--` option to the `nft` command to avoid that the shell interprets the negative priority value as an option of the `nft` command.

3. Add a rule to the `prerouting` chain that redirects incoming traffic on the `ens3` interface sent to port `80` and `443` to the host with the `192.0.2.1` IP:

```bash
# nft add rule nat prerouting ifname ens3 tcp dport { 80, 443 } dnat to 192.0.2.1
```

4. Depending on your environment, add either a SNAT or masquerading rule to change the source address:

a. If the `ens3` interface used dynamic IP addresses, add a masquerading rule:
# nft add rule nat postrouting oifname "ens3" masquerade

b. If the ens3 interface uses a static IP address, add a SNAT rule. For example, if the ens3 uses the 198.51.100.1 IP address:

```bash
# nft add rule nat postrouting oifname "ens3" snat to 198.51.100.1
```

Additional resources

- The different NAT types: masquerading, source NAT, destination NAT, and redirect

8.4.5. Configuring a redirect using nftables

The redirect feature is a special case of destination network address translation (DNAT) that redirects packets to the local machine depending on the chain hook.

The following procedure describes how to redirect incoming and forwarded traffic sent to port 22 of the local host to port 2222.

Procedure

1. Create a table:

   ```bash
   # nft add table nat
   ```

2. Add the prerouting chain to the table:

   ```bash
   # nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
   ```

   Note that you must pass the -- option to the nft command to avoid that the shell interprets the negative priority value as an option of the nft command.

3. Add a rule to the prerouting chain that redirects incoming traffic on port 22 to port 2222:

   ```bash
   # nft add rule nat prerouting tcp dport 22 redirect to 2222
   ```

Additional resources

- The different NAT types: masquerading, source NAT, destination NAT, and redirect

8.5. USING SETS IN NFTABLES COMMANDS

The nftables framework natively supports sets. You can use sets, for example, if a rule should match multiple IP addresses, port numbers, interfaces, or any other match criteria.

8.5.1. Using anonymous sets in nftables

An anonymous set contain comma-separated values enclosed in curly brackets, such as \{ 22, 80, 443 \}, that you use directly in a rule. You can also use anonymous sets also for IP addresses or any other match criteria.
The drawback of anonymous sets is that if you want to change the set, you must replace the rule. For a dynamic solution, use named sets as described in Using named sets in nftables.

### Prerequisites

- The example_chain chain and the example_table table in the inet family exists.

### Procedure

1. For example, to add a rule to example_chain in example_table that allows incoming traffic to port 22, 80, and 443:

   ```
   # nft add rule inet example_table example_chain tcp dport { 22, 80, 443 } accept
   ```

2. Optionally, display all chains and their rules in example_table:

   ```
   # nft list table inet example_table
   table inet example_table {
      chain example_chain {
         type filter hook input priority filter; policy accept;
         tcp dport { ssh, http, https } accept
      }
   }
   ```

### 8.5.2. Using named sets in nftables

The nftables framework supports mutable named sets. A named set is a list or range of elements that you can use in multiple rules within a table. Another benefit over anonymous sets is that you can update a named set without replacing the rules that use the set.

When you create a named set, you must specify the type of elements the set contains. You can set the following types:

- **ipv4_addr** for a set that contains IPv4 addresses or ranges, such as 192.0.2.1 or 192.0.2.0/24.
- **ipv6_addr** for a set that contains IPv6 addresses or ranges, such as 2001:db8:1::1 or 2001:db8:1::1/64.
- **ether_addr** for a set that contains a list of media access control (MAC) addresses, such as 52:54:00:6b:66:42.
- **inet_proto** for a set that contains a list of Internet protocol types, such as tcp.
- **inet_service** for a set that contains a list of Internet services, such as ssh.
- **mark** for a set that contains a list of packet marks. Packet marks can be any positive 32-bit integer value (0 to 2147483647).

### Prerequisites

- The example_chain chain and the example_table table exists.

### Procedure

1. Create an empty set. The following examples create a set for IPv4 addresses:
To create a set that can store multiple individual IPv4 addresses:

```
# nft add set inet example_table example_set { type ipv4_addr ; }
```

To create a set that can store IPv4 address ranges:

```
# nft add set inet example_table example_set { type ipv4_addr ; flags interval ; }
```

**IMPORTANT**

To avoid that the shell interprets the semicolons as the end of the command, you must escape the semicolons with a backslash.

2. Optionally, create rules that use the set. For example, the following command adds a rule to the `example_chain` in the `example_table` that will drop all packets from IPv4 addresses in `example_set`.

```
# nft add rule inet example_table example_chain ip saddr @example_set drop
```

Because `example_set` is still empty, the rule has currently no effect.

3. Add IPv4 addresses to `example_set`:

   - If you create a set that stores individual IPv4 addresses, enter:

     ```
     # nft add element inet example_table example_set { 192.0.2.1, 192.0.2.2 }
     ```

   - If you create a set that stores IPv4 ranges, enter:

     ```
     # nft add element inet example_table example_set { 192.0.2.0-192.0.2.255 }
     ```

     When you specify an IP address range, you can alternatively use the Classless Inter-Domain Routing (CIDR) notation, such as `192.0.2.0/24` in the above example.

**8.5.3. Additional resources**

- The `Sets` section in the `nft(8)` man page

**8.6. USING VERDICT MAPS IN NFTABLES COMMANDS**

Verdict maps, which are also known as dictionaries, enable `nft` to perform an action based on packet information by mapping match criteria to an action.

**8.6.1. Using anonymous maps in nftables**

An anonymous map is a `{ match_criteria : action }` statement that you use directly in a rule. The statement can contain multiple comma-separated mappings.

The drawback of an anonymous map is that if you want to change the map, you must replace the rule. For a dynamic solution, use named maps as described in Using named maps in nftables.

The example describes how to use an anonymous map to route both TCP and UDP packets of the IPv4 and IPv6 protocol to different chains to count incoming TCP and UDP packets separately.
Procedure

1. Create the `example_table`:
   ```bash
   # nft add table inet example_table
   ```

2. Create the `tcp_packets` chain in `example_table`:
   ```bash
   # nft add chain inet example_table tcp_packets
   ```

3. Add a rule to `tcp_packets` that counts the traffic in this chain:
   ```bash
   # nft add rule inet example_table tcp_packets counter
   ```

4. Create the `udp_packets` chain in `example_table`:
   ```bash
   # nft add chain inet example_table udp_packets
   ```

5. Add a rule to `udp_packets` that counts the traffic in this chain:
   ```bash
   # nft add rule inet example_table udp_packets counter
   ```

6. Create a chain for incoming traffic. For example, to create a chain named `incoming_traffic` in `example_table` that filters incoming traffic:
   ```bash
   # nft add chain inet example_table incoming_traffic { type filter hook input priority 0 ; }
   ```

7. Add a rule with an anonymous map to `incoming_traffic`:
   ```bash
   # nft add rule inet example_table incoming_traffic ip protocol vmap { tcp : jump tcp_packets, udp : jump udp_packets }
   ```

   The anonymous map distinguishes the packets and sends them to the different counter chains based on their protocol.

8. To list the traffic counters, display `example_table`:
   ```bash
   # nft list table inet example_table
   table inet example_table {
     chain tcp_packets {
       counter packets 36379 bytes 2103816
     }
     chain udp_packets {
       counter packets 10 bytes 1559
     }
     chain incoming_traffic {
       type filter hook input priority filter; policy accept;
       ip protocol vmap { tcp : jump tcp_packets, udp : jump udp_packets }
     }
   }
   ```
The counters in the tcp_packets and udp_packets chain display both the number of received packets and bytes.

8.6.2. Using named maps in nftables

The nftables framework supports named maps. You can use these maps in multiple rules within a table. Another benefit over anonymous maps is that you can update a named map without replacing the rules that use it.

When you create a named map, you must specify the type of elements:

- **ipv4_addr** for a map whose match part contains an IPv4 address, such as 192.0.2.1.
- **ipv6_addr** for a map whose match part contains an IPv6 address, such as 2001:db8:1::1.
- **ether_addr** for a map whose match part contains a media access control (MAC) address, such as 52:54:00:6b:66:42.
- **inet_proto** for a map whose match part contains an Internet protocol type, such as tcp.
- **inet_service** for a map whose match part contains an Internet services name port number, such as ssh or 22.
- **mark** for a map whose match part contains a packet mark. A packet mark can be any positive 32-bit integer value (0 to 2147483647).
- **counter** for a map whose match part contains a counter value. The counter value can be any positive 64-bit integer value.
- **quota** for a map whose match part contains a quota value. The quota value can be any positive 64-bit integer value.

The example describes how to allow or drop incoming packets based on their source IP address. Using a named map, you require only a single rule to configure this scenario while the IP addresses and actions are dynamically stored in the map. The procedure also describes how to add and remove entries from the map.

Procedure

1. Create a table. For example, to create a table named example_table that processes IPv4 packets:

   ```
   # nft add table ip example_table
   ```

2. Create a chain. For example, to create a chain named example_chain in example_table:

   ```
   # nft add chain ip example_table example_chain { type filter hook input priority 0 \; } 
   ```

   **IMPORTANT**

   To avoid that the shell interprets the semicolons as the end of the command, you must escape the semicolons with a backslash.

3. Create an empty map. For example, to create a map for IPv4 addresses:
4. Create rules that use the map. For example, the following command adds a rule to `example_chain` in `example_table` that applies actions to IPv4 addresses which are both defined in `example_map`:

```
# nft add rule example_table example_chain ip saddr vmap @example_map
```

5. Add IPv4 addresses and corresponding actions to `example_map`:

```
# nft add element ip example_table example_map { 192.0.2.1 : accept, 192.0.2.2 : drop }
```

This example defines the mappings of IPv4 addresses to actions. In combination with the rule created above, the firewall accepts packet from `192.0.2.1` and drops packets from `192.0.2.2`.

6. Optionally, enhance the map by adding another IP address and action statement:

```
# nft add element ip example_table example_map { 192.0.2.3 : accept }
```

7. Optionally, remove an entry from the map:

```
# nft delete element ip example_table example_map { 192.0.2.1 }
```

8. Optionally, display the rule set:

```
# nft list ruleset
table ip example_table {
    map example_map {
        type ipv4_addr : verdict
        elements = { 192.0.2.2 : drop, 192.0.2.3 : accept }
    }

    chain example_chain {
        type filter hook input priority filter; policy accept;
        ip saddr vmap @example_map
    }
}
```

8.6.3. Additional resources

- The Maps section in the `nft(8)` man page

8.7. CONFIGURING PORT FORWARDING USING NFTABLES

Port forwarding enables administrators to forward packets sent to a specific destination port to a different local or remote port.

For example, if your web server does not have a public IP address, you can set a port forwarding rule on your firewall that forwards incoming packets on port 80 and 443 on the firewall to the web server. With this firewall rule, users on the internet can access the web server using the IP or host name of the firewall.
8.7.1. Forwarding incoming packets to a different local port

This section describes an example of how to forward incoming IPv4 packets on port 8022 to port 22 on the local system.

**Procedure**

1. Create a table named `nat` with the `ip` address family:
   ```
   # nft add table ip nat
   ```

2. Add the `prerouting` and `postrouting` chains to the table:
   ```
   # nft -- add chain ip nat prerouting { type nat hook prerouting priority -100 \; } 
   ```

   **NOTE**
   Pass the `--` option to the `nft` command to avoid that the shell interprets the negative priority value as an option of the `nft` command.

3. Add a rule to the `prerouting` chain that redirects incoming packets on port 8022 to the local port 22:
   ```
   # nft add rule ip nat prerouting tcp dport 8022 redirect to :22
   ```

8.7.2. Forwarding incoming packets on a specific local port to a different host

You can use a destination network address translation (DNAT) rule to forward incoming packets on a local port to a remote host. This enables users on the Internet to access a service that runs on a host with a private IP address.

The procedure describes how to forward incoming IPv4 packets on the local port 443 to the same port number on the remote system with the 192.0.2.1 IP address.

**Prerequisites**

- You are logged in as the `root` user on the system that should forward the packets.

**Procedure**

1. Create a table named `nat` with the `ip` address family:
   ```
   # nft add table ip nat
   ```

2. Add the `prerouting` and `postrouting` chains to the table:
   ```
   # nft -- add chain ip nat prerouting { type nat hook prerouting priority -100 \; } 
   # nft add chain ip nat postrouting { type nat hook postrouting priority 100 \; }
   ```
NOTE

Pass the -- option to the `nft` command to avoid that the shell interprets the negative priority value as an option of the `nft` command.

3. Add a rule to the `prerouting` chain that redirects incoming packets on port 443 to the same port on 192.0.2.1:

```
# nft add rule ip nat prerouting tcp dport 443 dnat to 192.0.2.1
```

4. Add a rule to the `postrouting` chain to masquerade outgoing traffic:

```
# nft add rule ip nat postrouting daddr 192.0.2.1 masquerade
```

5. Enable packet forwarding:

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

8.8. USING NFTABLES TO LIMIT THE AMOUNT OF CONNECTIONS

You can use `nftables` to limit the number of connections or to block IP addresses that attempt to establish a given amount of connections to prevent them from using too many system resources.

8.8.1. Limiting the number of connections using `nftables`

The `ct count` parameter of the `nft` utility enables administrators to limit the number of connections. The procedure describes a basic example of how to limit incoming connections.

**Prerequisites**

- The base `example_chain` in `example_table` exists.

**Procedure**

1. Create a dynamic set for IPv4 addresses:

```
# nft add set inet example_table example_meter { type ipv4_addr; flags dynamic ;}
```

2. Add a rule that allows only two simultaneous connections to the SSH port (22) from an IPv4 address and rejects all further connections from the same IP:

```
# nft add rule ip example_table example_chain tcp dport ssh meter example_meter { ip saddr ct count over 2 } counter reject
```

3. Optionally, display the set created in the previous step:

```
# nft list set inet example_table example_meter
```

Red Hat Enterprise Linux 8 Securing networks

118
The `elements` entry displays addresses that currently match the rule. In this example, `elements` lists IP addresses that have active connections to the SSH port. Note that the output does not display the number of active connections or if connections were rejected.

### 8.8.2. Blocking IP addresses that attempt more than ten new incoming TCP connections within one minute

This section explains how you temporarily block hosts that are establishing more than ten IPv4 TCP connections within one minute.

**Procedure**

1. Create the `filter` table with the `ip` address family:

   ```
   # nft add table ip filter
   ```

2. Add the `input` chain to the `filter` table:

   ```
   # nft add chain ip filter input { type filter hook input priority 0 ; }
   ```

3. Add a rule that drops all packets from source addresses that attempt to establish more than ten TCP connections within one minute:

   ```
   # nft add rule ip filter input ip protocol tcp ct state new, untracked meter ratemeter { ip saddr timeout 5m limit rate over 10/minute } drop
   ```

   The `timeout 5m` parameter defines that `nftables` automatically removes entries after five minutes to prevent that the meter fills up with stale entries.

**Verification**

- To display the meter’s content, enter:

  ```
  # nft list meter ip filter ratemeter
table ip filter {
  meter ratemeter {
    type ipv4_addr
    size 65535
    flags dynamic,timeout
    elements = { 192.0.2.1 limit rate over 10/minute timeout 5m expires 4m58s224ms } 
  }
  }
  ```

### 8.9. DEBUGGING NFTABLES RULES

The `nftables` framework provides different options for administrators to debug rules and if packets match them. This section describes these options.
8.9.1. Creating a rule with a counter

To identify if a rule is matched, you can use a counter. This section describes how to create a new rule with a counter.

- For more information on a procedure that adds a counter to an existing rule, see Adding a counter to an existing rule in Configuring and managing networking

Prerequisites

- The chain to which you want to add the rule exists.

Procedure

1. Add a new rule with the `counter` parameter to the chain. The following example adds a rule with a counter that allows TCP traffic on port 22 and counts the packets and traffic that match this rule:

   ```
   # nft add rule inet example_table example_chain tcp dport 22 counter accept
   ```

2. To display the counter values:

   ```
   # nft list ruleset
   table inet example_table {
   chain example_chain {
   type filter hook input priority 0 policy accept;
   tcp dport ssh counter packets 6872 bytes 105448565 accept
   }
   }
   ```

8.9.2. Adding a counter to an existing rule

To identify if a rule is matched, you can use a counter. This section describes how to add a counter to an existing rule.

- For more information on a procedure that adds a new rule with a counter, see Creating a rule with the counter in Configuring and managing networking

Prerequisites

- The rule to which you want to add the counter exists.

Procedure

1. Display the rules in the chain including their handles:

   ```
   # nft --handle list chain inet example_table example_chain
   table inet example_table {
   chain example_chain { # handle 1
   type filter hook input priority 0 policy accept;
   tcp dport ssh accept # handle 4
   }
   }
2. Add the counter by replacing the rule but with the `counter` parameter. The following example replaces the rule displayed in the previous step and adds a counter:

```bash
# nft replace rule inet example_table example_chain handle 4 tcp dport 22 counter accept
```

3. To display the counter values:

```bash
# nft list ruleset
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
tcp dport ssh counter packets 6872 bytes 105448565 accept
  }
}
```

### 8.9.3. Monitoring packets that match an existing rule

The tracing feature in `nftables` in combination with the `nft monitor` command enables administrators to display packets that match a rule. The procedure describes how to enable tracing for a rule as well as monitoring packets that match this rule.

#### Prerequisites

- The rule to which you want to add the counter exists.

#### Procedure

1. Display the rules in the chain including their handles:

```bash
# nft --handle list chain inet example_table example_chain
table inet example_table {
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
tcp dport ssh accept # handle 4
  }
}
```

2. Add the tracing feature by replacing the rule but with the `meta nftrace set 1` parameters. The following example replaces the rule displayed in the previous step and enables tracing:

```bash
# nft replace rule inet example_table example_chain handle 4 tcp dport 22 meta nftrace set 1 accept
```

3. Use the `nft monitor` command to display the tracing. The following example filters the output of the command to display only entries that contain `inet example_table example_chain`:

```bash
# nft monitor | grep "inet example_table example_chain"
trace id 3c5eb15e inet example_table example_chain packet: iif "enp1s0" ether saddr 52:54:00:17:ff:e4 ether daddr 52:54:00:72:2f:6e ip saddr 192.0.2.1 ip daddr 192.0.2.2 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 49710 ip protocol tcp ip length 60 tcp sport 56728 tcp dport ssh tcp flags == syn tcp window 64240
```
8.10. BACKING UP AND RESTORING THE NFTABLES RULE SET

This section describes how to backup nftables rules to a file, as well as restoring rules from a file.

Administrators can use a file with the rules to, for example, transfer the rules to a different server.

8.10.1. Backing up the nftables rule set to a file

This section describes how to back up the nftables rule set to a file.

Procedure

- To backup nftables rules:
  
  - In a format produced by nft list ruleset format:
    
    # nft list ruleset > file.nft
  
  - In JSON format:
    
    # nft -j list ruleset > file.json

8.10.2. Restoring the nftables rule set from a file

This section describes how to restore the nftables rule set.

Procedure

- To restore nftables rules:
  
  - If the file to restore is in the format produced by nft list ruleset or contains nft commands directly:
    
    # nft -f file.nft
e
  - If the file to restore is in JSON format:
    
    # nft -j -f file.json
8.11. ADDITIONAL RESOURCES

- Using nftables in Red Hat Enterprise Linux 8
- What comes after iptables? Its successor, of course: nftables
- Firewalld: The Future is nftables