Red Hat Enterprise Linux 9

ネットワークのセキュリティー保護

セキュリティー保護されたネットワークおよびネットワーク通信の設定

Last Updated: 2024-09-09

Red Hat Enterprise Linux 9 ネットワークのセキュリティー保護

セキュリティー保護されたネットワークおよびネットワーク通信の設定

法律上の通知

概要

ネットワークのセキュリティーを向上させ、データ侵害や侵入のリスクを軽減するためのツールとテクニックを学びます。

目次

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. Jira の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 2 台のシステム間で OPENSSH を使用した安全な通信の使用

SSH (Secure Shell) は、クライアント/サーバーアーキテクチャーを使用する 2 つのシステム間で安全な通信を提供し、ユーザーがリモートでサーバーホストシステムにログインできるようにするプロトコルです。FTP や Telnet などの他のリモート通信プロトコルとは異なり、SSH はログインセッションを暗号化します。これにより、侵入者が接続から暗号化されていないパスワードを収集するのを防ぎます。

1.1. SSH と OPENSSH

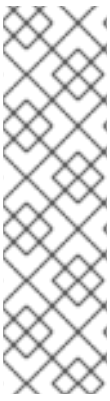
SSH (Secure Shell) は、リモートマシンにログインしてそのマシンでコマンドを実行するプログラムです。SSH プロトコルは、安全でないネットワーク上で、信頼されていないホスト間で安全な通信を提供します。また、X11 接続と任意の TCP/IP ポートを安全なチャンネルで転送することもできます。

SSH プロトコルは、リモートシェルのログインやファイルコピー用に使用する場合に、システム間の通信の傍受や特定ホストの偽装など、セキュリティーの脅威を軽減します。これは、SSH クライアントとサーバーがデジタル署名を使用してそれぞれの ID を確認するためです。さらに、クライアントシステムとサーバーシステムとの間の通信はすべて暗号化されます。

ホストキーは、SSH プロトコルのホストを認証します。ホストキーは、OpenSSH が初めて起動したとき、またはホストが初めて起動したときに自動的に生成される暗号鍵です。

OpenSSH は、Linux、UNIX、および同様のオペレーティングシステムでサポートされている SSH プロトコルの実装です。OpenSSH クライアントとサーバー両方に必要なコアファイルが含まれます。OpenSSH スイートは、以下のユーザー空間ツールで構成されます。

- **SSH** は、リモートログインプログラム (SSH クライアント) です。
- **sshd** は、OpenSSH SSH デーモンです。
- **scp** は、安全なリモートファイルコピープログラムです。
- **sftp** は、安全なファイル転送プログラムです。
- **ssh-agent** は、秘密鍵をキャッシュする認証エージェントです。
- **ssh-add** は、秘密鍵の ID を **ssh-agent** に追加します。
- **ssh-keygen** が、**ssh** の認証キーを生成、管理、および変換します。
- **ssh-copy-id** は、ローカルの公開鍵をリモート SSH サーバーの **authorized_keys** ファイルに追加するスクリプトです。
- **ssh-keyscan** - SSH パブリックホストキーを収集します。



注記

RHEL 9 では、Secure copy protocol (SCP) がデフォルトで SSH File Transfer Protocol (SFTP) に置き換えられています。これは、CVE-2020-15778 など、SCP が原因のセキュリティーの問題が発生しているためです。

SFTP が使用できないか互換性がない場合は、**-O** オプションを指定した **scp** コマンドを使用すると、元の SCP/RCP プロトコルの使用を強制できます。

詳細は、記事 OpenSSH SCP protocol deprecation in Red Hat Enterprise Linux 9 を参照してください。

RHEL の OpenSSH スイートは、SSH バージョン 2 のみをサポートします。このスイートは、旧バージョン (バージョン 1) の既知のエクスプロイトに対して脆弱ではない拡張キー交換アルゴリズムを備えています。

Red Hat Enterprise Linux には、**OpenSSH** パッケージが (一般的な **openssh** パッケージ、**openssh-server** パッケージ、**openssh-clients** パッケージ) が含まれています。**OpenSSH** パッケージには、**OpenSSL** パッケージ (**openssl-libs**) が必要です。このパッケージは、重要な暗号化ライブラリーをいくつかインストールして、暗号化通信を提供する **OpenSSH** を有効にします。

RHEL コア暗号化サブシステムの 1 つである OpenSSH は、システム全体の暗号化ポリシーを使用します。これにより、弱い暗号スイートおよび暗号化アルゴリズムがデフォルト設定で無効になります。ポリシーを変更するには、管理者が **update-crypto-policies** コマンドを使用して設定を調節するか、システム全体の暗号化ポリシーを手動でオプトアウトする必要があります。詳細は、システム全体の暗号化ポリシーに従わないようにアプリケーションを除外 セクションを参照してください。

OpenSSH スイートは、2 セットの設定ファイルを使用します。1 つはクライアントプログラム (つまり、**ssh**、**scp**、および **sftp**) 用で、もう 1 つはサーバー (**sshd** デーモン) 用です。

システム全体の SSH 設定情報が **/etc/ssh/** ディレクトリーに保存されます。**/etc/ssh/ssh_config** ファイルには、クライアント設定が含まれています。**/etc/ssh/sshd_config** ファイルは、デフォルトの OpenSSH サーバー設定ファイルです。

ユーザー固有の SSH 設定情報は、ユーザーのホームディレクトリーの **~/.ssh/** に保存されます。OpenSSH 設定ファイルの詳細なリストは、**sshd(8)** man ページの **FILES** セクションを参照してください。

関連情報

- システム上で **man -k ssh** コマンドを使用してリスト表示される man ページ
- システム全体の暗号化ポリシーの使用

1.2. SSH 鍵ペアの生成

ローカルシステムで SSH 鍵ペアを生成し、生成された公開鍵を OpenSSH サーバーにコピーすることで、パスワードを入力せずに OpenSSH サーバーにログインできます。鍵を作成する各ユーザーは、この手順を実行する必要があります。

システムを再インストールした後も以前に生成した鍵ペアを保持するには、新しい鍵を作成する前に **~/.ssh/** ディレクトリーをバックアップします。再インストール後に、このディレクトリーをホームディレクトリーにコピーします。これは、(**root** を含む) システムの全ユーザーで実行できます。

前提条件

- OpenSSH サーバーに鍵を使用して接続するユーザーとしてログインしている。
- OpenSSH サーバーが鍵ベースの認証を許可するように設定されている。

手順

1. ECDSA 鍵ペアを生成します。

```
$ ssh-keygen -t ecdsa
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/<username>/.ssh/id_ecdsa):
```

```
Enter passphrase (empty for no passphrase): <password>
Enter same passphrase again: <password>
Your identification has been saved in /home/<username>/.ssh/id_ecdsa.
Your public key has been saved in /home/<username>/.ssh/id_ecdsa.pub.
The key fingerprint is:
SHA256:Q/x+qms4j7PCQ0qFd09iZEFHA+SqwBKRNaU72oZfaCI
<username>@<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]-----+
```

パラメーターなしで **ssh-keygen** コマンドを使用して RSA 鍵ペアを生成することも、**ssh-keygen -t ed25519** コマンドを入力して Ed25519 鍵ペアを生成することもできます。Ed25519 アルゴリズムは FIPS-140 に準拠しておらず、FIPS モードでは OpenSSH は Ed25519 鍵で機能しないことに注意してください。

2. 公開鍵をリモートマシンにコピーします。

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

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

<username>@<ssh-server-example.com> は、認証情報に置き換えます。

セッションで **ssh-agent** プログラムを使用しない場合は、上記のコマンドで、最後に変更した **~/.ssh/id*.pub** 公開鍵をコピーします (インストールされていない場合)。別の公開キーファイルを指定したり、**ssh-agent** により、メモリーにキャッシュされた鍵よりもファイル内の鍵の方が優先順位を高くするには、**-i** オプションを指定して **ssh-copy-id** コマンドを使用します。

検証

1. 鍵ファイルを使用して OpenSSH サーバーにログインします。

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

関連情報

- システム上の **ssh-keygen(1)** および **ssh-copy-id(1)** man ページ

1.3. OPENSSH サーバーで鍵ベースの認証を唯一の方法として設定する

システムのセキュリティーを強化するには、OpenSSH サーバーでパスワード認証を無効にして鍵ベースの認証を有効にします。

前提条件

- **openssh-server** パッケージがインストールされている。
- サーバーで **sshd** デーモンが実行している。
- 鍵を使用して OpenSSH サーバーに接続できる。
  詳細は、SSH 鍵ペアの生成 セクションを参照してください。

手順

1. テキストエディターで **/etc/ssh/sshd_config** 設定を開きます。以下に例を示します。

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

2. **PasswordAuthentication** オプションを **no** に変更します。

```
PasswordAuthentication no
```

3. 新しいデフォルトインストール以外のシステムでは、**PubkeyAuthentication** パラメーターが設定されていないか、**yes** に設定されていることを確認します。
4. **KbdInteractiveAuthentication** ディレクティブを **no** に設定します。
   設定ファイル内では対応するエントリーがコメントアウトされていること、およびデフォルト値が **yes** であることに注意してください。
5. NFS がマウントされたホームディレクトリーで鍵ベースの認証を使用するには、SELinux ブール値 **use_nfs_home_dirs** を有効にします。

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

6. リモートで接続している場合は、コンソールもしくは帯域外アクセスを使用せず、パスワード認証を無効にする前に、鍵ベースのログインプロセスをテストします。
7. **sshd** デーモンを再読み込みし、変更を適用します。

```
# systemctl reload sshd
```

関連情報

- システム上の **sshd_config(5)** および **setsebool(8)** man ページ

1.4. SSH-AGENT を使用した SSH 認証情報のキャッシュ

SSH 接続を開始するたびにパスフレーズを入力しなくても済むように、**ssh-agent** ユーティリティーを使用して、ログインセッションの SSH 秘密鍵をキャッシュできます。エージェントが実行中で、鍵のロックが解除されている場合、鍵のパスワードを再入力することなく、この鍵を使用して SSH サーバーにログインできます。秘密鍵とパスフレーズのセキュリティーが確保されます。

前提条件

- SSH デーモンが実行されており、ネットワーク経由でアクセスできるリモートホストがある。
- リモートホストにログインするための IP アドレスまたはホスト名および認証情報を把握している。
- パスフレーズで SSH キーペアを生成し、公開鍵をリモートマシンに転送している。
  詳細は、SSH 鍵ペアの生成 セクションを参照してください。

手順

1. セッションで **ssh-agent** を自動的に起動するためのコマンドを ~/**.bashrc** ファイルに追加します。
   a. 任意のテキストエディターで ~/**.bashrc** を開きます。次に例を示します。

   ```
   $ vi ~/.bashrc
   ```

   b. 以下の行をファイルに追加します。

   ```
   eval $(ssh-agent)
   ```

   c. 変更を保存し、エディターを終了します。
2. ~/**.ssh**/**config** ファイルに次の行を追加します。

   ```
   AddKeysToAgent yes
   ```

   セッションでこのオプションを使用して **ssh-agent** が起動されると、エージェントはホストに初めて接続するときにのみパスワードを要求します。

検証

- エージェントにキャッシュされた秘密鍵に対応する公開鍵を使用するホストにログインします。次に例を示します。

  ```
  $ ssh <example.user>@<ssh-server@example.com>
  ```

  パスフレーズを入力する必要がないことに注意してください。

1.5. スマートカードに保存した SSH 鍵による認証

スマートカードに ECDSA 鍵と RSA 鍵を作成して保存し、そのスマートカードを使用して OpenSSH クライアントで認証することができます。スマートカード認証は、デフォルトのパスワード認証に代わるものです。

前提条件

- クライアントで、**opensc** パッケージをインストールして、**pcscd** サービスを実行している。

手順

1. PKCS #11 の URI を含む OpenSC PKCS #11 モジュールが提供する鍵のリストを表示し、その出力を **keys.pub** ファイルに保存します。

```
$ ssh-keygen -D pkcs11: > keys.pub
```

2. 公開鍵をリモートサーバーに転送します。**ssh-copy-id** コマンドを使用し、前の手順で作成した **keys.pub** ファイルを指定します。

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

3. ECDSA 鍵を使用して **<ssh-server-example.com>** に接続します。鍵を一意に参照する URI のサブセットのみを使用することもできます。次に例を示します。

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

OpenSSH は **p11-kit-proxy** ラッパーを使用し、OpenSC PKCS #11 モジュールが **p11-kit** ツールに登録されているため、前のコマンドを簡略化できます。

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

PKCS #11 の URI の **id=** の部分を飛ばすと、OpenSSH が、プロキシーモジュールで利用可能な鍵をすべて読み込みます。これにより、必要な入力の量を減らすことができます。

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

4. オプション: ~/**.ssh**/**config** ファイルで同じ URI 文字列を使用して、設定を永続化できます。

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

ssh クライアントユーティリティーが、この URI とスマートカードの鍵を自動的に使用するようになります。

関連情報

- システム上の **p11-kit(8)**、**opensc.conf(5)**、**pcscd(8)**、**ssh(1)**、および **ssh-keygen(1)** man ページ

1.6. OPENSSH のセキュリティーの強化

OpenSSH を使用すると、システムを調整してセキュリティーを強化できます。

OpenSSH サーバー設定ファイル **/etc/ssh/sshd_config** の変更を有効にするには、**sshd** デーモンを再ロードする必要があることに注意してください。

```
# systemctl reload sshd
```

警告

ほとんどのセキュリティー強化の設定変更により、最新のアルゴリズムまたは暗号スイートに対応していないクライアントとの互換性が低下します。

安全ではない接続プロトコルの無効化

SSH を本当の意味で有効なものにするため、OpenSSH スイートに置き換えられる安全ではない接続プロトコルを使用しないようにします。このような接続プロトコルを使用すると、ユーザーのパスワード自体は SSH を使用した 1 回のセッションで保護されても、その後に Telnet を使用してログインした時に傍受されてしまうためです。

パスワードベースの認証の無効化

認証用のパスワードを無効にし、鍵ペアのみを許可すると、攻撃対象領域が縮小されます。詳細は、OpenSSH サーバーで鍵ベースの認証を唯一の方法として設定する セクションを参照してください。

より強力な鍵タイプ

ssh-keygen コマンドはデフォルトで RSA キーのペアを生成しますが、**-t** オプションを使用すると、Elliptic Curve Digital Signature Algorithm (ECDSA) 鍵または Edwards-Curve 25519 (Ed25519) 鍵を生成するように指定できます。ECDSA は、同等の対称鍵強度において RSA よりも優れたパフォーマンスを実現します。また、より短い鍵を生成します。Ed25519 公開鍵アルゴリズムは、RSA、DSA、ECDSA よりもセキュアで高速な Twisted Edwards 曲線の実装です。
サーバーホストの鍵の RSA、ECDSA、および Ed25519 がない場合は、OpenSSH が自動的に作成します。RHEL でホストの鍵の作成を設定するには、インスタンス化したサービス **sshd-keygen@.service** を使用します。たとえば、RSA 鍵タイプの自動作成を無効にするには、次のコマンドを実行します。

```
# systemctl mask sshd-keygen@rsa.service
# rm -f /etc/ssh/ssh_host_rsa_key*
# systemctl restart sshd
```

注記

cloud-init 方式が有効になっているイメージでは、**ssh-keygen** ユニットが自動的に無効になります。これは、**ssh-keygen template** サービスが **cloud-init** ツールに干渉し、ホストキーの生成で問題が発生する可能性があるためです。これらの問題を回避するには、**cloud-init** が実行している場合に、**etc/systemd/system/sshd-keygen@.service.d/disable-sshd-keygen-if-cloud-init-active.conf** ドロップイン設定ファイルにより **ssh-keygen** ユニットが無効になります。

SSH 接続に特定の鍵タイプのみを許可するには、**/etc/ssh/sshd_config** の該当する行の先頭のコメントを削除し、**sshd** サービスを再ロードします。たとえば、Ed25519 ホスト鍵のみを許可する場合、対応する行は次のようになります。

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

重要

Ed25519 アルゴリズムは FIPS-140 に準拠しておらず、FIPS モードでは OpenSSH は Ed25519 鍵で機能しません。

デフォルト以外のポート

デフォルトでは、**sshd** デーモンは TCP ポート 22 をリッスンします。ポートを変更すると、システムがデフォルトポートで自動ネットワークスキャンに基づく攻撃にさらされる可能性が減るため、匿名化によってセキュリティーが強化されます。ポートは、**/etc/ssh/sshd_config** 設定ファイルの **Port** ディレクティブを使用して指定できます。
また、デフォルト以外のポートを使用できるように、デフォルトの SELinux ポリシーも更新する必要があります。そのためには、**policycoreutils-python-utils** パッケージの **semanage** ツールを使用します。

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

さらに、**firewalld** 設定を更新します。

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

前のコマンドの **<port-number>** は、**Port** ディレクティブを使用して指定した新しいポート番号に置き換えます。

Root ログイン

PermitRootLogin はデフォルトで **prohibit-password** に設定されています。これにより、root としてログインしてパスワードを使用する代わりに鍵ベースの認証が使用され、ブルートフォース攻撃を防ぐことでリスクが軽減します。

警告

root ユーザーとしてログインを有効にすることは、どのユーザーがどの特権コマンドを実行するかを監査できないため、安全ではありません。管理コマンドを使用するには、ログインして、代わりに **sudo** を使用します。

X セキュリティー拡張機能の使用

Red Hat Enterprise Linux クライアントの X サーバーは、X セキュリティー拡張を提供しません。そのため、クライアントは X11 転送を使用して信頼できない SSH サーバーに接続するときに別のセキュリティー層を要求できません。ほとんどのアプリケーションは、この拡張機能を有効にしても実行できません。

デフォルトでは、**/etc/ssh/ssh_config.d/50-redhat.conf** ファイルの **ForwardX11Trusted** オプションが **yes** に設定され、**ssh -X remote_machine** コマンド (信頼できないホスト) と **ssh -Y remote_machine** コマンド (信頼できるホスト) には違いがありません。

シナリオで X11 転送機能を必要としない場合は、**/etc/ssh/sshd_config** 設定ファイルの **X11Forwarding** ディレクティブを **no** に設定します。

特定のユーザー、グループ、または IP 範囲への SSH アクセスの制限

/etc/ssh/sshd_config 設定ファイルの **AllowUsers** ディレクティブおよび **AllowGroups** ディレクティブを使用すると、特定のユーザー、ドメイン、またはグループのみが OpenSSH サーバーに接続することを許可できます。**AllowUsers** および **AllowGroups** を組み合わせて、アクセスをより正確に制限できます。以下に例を示します。

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

この設定では、次の条件がすべて満たされる場合にのみ接続が許可されます。

- 接続の送信元 IP が、192.168.1.0/24 または 10.0.0.0/24 サブネット内にある。
- 送信元 IP が 192.168.1.2 ではない。
- ユーザーが example-group グループのメンバーである。

OpenSSH サーバーは、**/etc/ssh/sshd_config** 内のすべての Allow および Deny ディレクティブを渡す接続のみを許可します。たとえば、**AllowUsers** ディレクティブに、**AllowGroups** ディレクティブにリストされているグループの一部ではないユーザーがリストされている場合、そのユーザーはログインできません。

許可リストは、許可されていない新しいユーザーまたはグループもブロックするため、許可リスト (Allow で始まるディレクティブ) の使用は、拒否リスト (Deny で始まるオプション) を使用するよりも安全です。

システム全体の暗号化ポリシーの変更

OpenSSH は、RHEL のシステム全体の暗号化ポリシーを使用し、デフォルトのシステム全体の暗号化ポリシーレベルは、現在の脅威モデルに安全な設定を提供します。暗号化の設定をより厳格にするには、現在のポリシーレベルを変更します。

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

警告

システムがレガシーシステムと通信する場合、**FUTURE** ポリシーの厳格な設定により相互運用性の問題が発生する可能性があります。

システム全体の暗号化ポリシーにより、SSH プロトコルの特定の暗号のみを無効にすることもできます。詳細は、「セキュリティーの強化」ドキュメントの サブポリシーを使用したシステム全体の暗号化ポリシーのカスタマイズ セクションを参照してください。

システム全体の暗号化ポリシーのオプトアウト

OpenSSH サーバーのシステム全体の暗号化ポリシーをオプトアウトするには、**/etc/ssh/sshd_config.d/** ディレクトリーにあるドロップイン設定ファイルに暗号化ポリシーを指定します。このとき、辞書式順序で **50-redhat.conf** ファイルよりも前に来るように、50 未満の 2 桁の数字接頭辞と、**.conf** という接尾辞を付けます (例: **49-crypto-policy-override.conf**)。
詳細は、**sshd_config(5)** の man ページを参照してください。

OpenSSH クライアントのシステム全体の暗号化ポリシーをオプトアウトするには、次のいずれかのタスクを実行します。

- 指定のユーザーの場合は、**~/.ssh/config** ファイルのユーザー固有の設定でグローバルの **ssh_config** を上書きします。
- システム全体の場合は、**/etc/ssh/ssh_config.d/** ディレクトリーにあるドロップイン設定ファイルに暗号化ポリシーを指定します。このとき、辞書式順序で **50-redhat.conf** ファイルよりも前に来るように、50 未満の 2 桁の接頭辞と、**.conf** という接尾辞を付けます (例: **49-crypto-policy-override.conf**)。

関連情報

- システム上の **sshd_config(5)**、**ssh-keygen(1)**、**crypto-policies(7)**、および **update-crypto-policies(8)** man ページ
- 「セキュリティー強化」ドキュメントの システム全体の暗号化ポリシーの使用
- How to disable specific algorithms and ciphers for ssh service only (Red Hat ナレッジベース)

1.7. SSH ジャンプホストを介したリモートサーバーへの接続

ローカルシステムから中間サーバー (ジャンプホストとも呼ばれます) を介してリモートサーバーに接続できます。ジャンプサーバーは、さまざまなセキュリティーゾーンのホストをブリッジし、複数のクライアントサーバー接続を管理できます。

前提条件

- ジャンプホストでローカルシステムからの SSH 接続に対応している。
- リモートサーバーがジャンプホストからの SSH 接続を受け入れる。

手順

1. ジャンプサーバーまたは複数の中間サーバーを経由して一度に接続する場合は、**ssh -J** コマンドを使用してジャンプサーバーを直接指定します。次に例を示します。

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

ジャンプサーバーのユーザー名または SSH ポートが、リモートサーバーの名前およびポートと異なる場合は、上記のコマンドのホスト名のみの表記を変更します。以下に例を示します。

```
$ ssh -J <example.user.1>@<jump-1.example.com>:<75>,<example.user.2>@<jump-2.example.com>:<75>,<example.user.3>@<jump-3.example.com>:<75> <example.user.f>@<target-server-1.example.com>:<220>
```

2. ジャンプサーバーを介してリモートサーバーに定期的に接続する場合は、ジャンプサーバーの設定を SSH 設定ファイルに保存します。

   a. ローカルシステムの **~/.ssh/config** ファイルを編集してジャンプホストを定義します。以下に例を示します。

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

   - **Host** パラメーターは、**ssh** コマンドで使用できるホストの名前またはエイリアスを定義します。値は実際のホスト名と一致可能ですが、任意の文字列にすることもできます。
   - **HostName** パラメーターは、ジャンプホストの実際のホスト名または IP アドレスを設定します。

   b. **ProxyJump** ディレクティブを使用してリモートサーバーのジャンプ設定を、ローカルシステムの **~/.ssh/config** ファイルに追加します。以下に例を示します。

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

   c. ローカルシステムを使用して、ジャンプサーバー経由でリモートサーバーに接続します。

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

   このコマンドは、前の設定ステップを省略したときの **ssh -J jump-server1 remote-server** コマンドと同じです。

関連情報

- システム上の **ssh_config(5)** および **ssh(1)** man ページ

1.8. SSH システムロールを使用したセキュアな通信の設定

管理者は、Ansible Core パッケージを使用すると、**sshd** システムロールを使用して SSH サーバーを設定できます。また、**ssh** システムロールを使用して任意の数の RHEL システムで SSH クライアントを一貫して同時に設定できます。

1.8.1. sshd RHEL システムロールの変数

sshd システムロール Playbook では、必要や制限に応じて SSH 設定ファイルのパラメーターを定義できます。

これらの変数が設定されていない場合には、システムロールは RHEL のデフォルト値と同じ **sshd_config** ファイルを作成します。

どのような場合でも、ブール値は **sshd** 設定で適切に **yes** と **no** としてレンダリングされます。リストを使用して複数行の設定項目を定義できます。以下に例を示します。

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

レンダリングは以下のようになります。

```
ListenAddress 0.0.0.0
ListenAddress ::
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/sshd/** ディレクトリー

1.8.2. sshd RHEL システムロールを使用した OpenSSH サーバーの設定

sshd RHEL システムロールを使用して、Ansible Playbook を実行し、複数の SSH サーバーを設定できます。

注記

sshd RHEL システムロールは、SSH および SSHD 設定を変更する他の RHEL システムロール (ID 管理 RHEL システムロールなど) とともに使用できます。設定が上書きされないようにするには、**sshd** ロールがネームスペース (RHEL 8 以前のバージョン) またはドロップインディレクトリー (RHEL 9) を使用していることを確認してください。

前提条件

- コントロールノードと管理対象ノードの準備が完了している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

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

```
    - Condition: "Address 192.0.2.0/24"
      PermitRootLogin: yes
      PasswordAuthentication: yes
```

Playbook は、以下のように、マネージドノードを SSH サーバーとして設定します。

- パスワードと **root** ユーザーのログインが無効である
- **192.0.2.0/24** のサブネットからのパスワードおよび **root** ユーザーのログインのみが有効である

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. SSH サーバーにログインします。

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

2. SSH サーバー上の **sshd_config** ファイルの内容を確認します。

```
$ cat /etc/ssh/sshd_config.d/00-ansible_system_role.conf
#
# Ansible managed
#
PasswordAuthentication no
PermitRootLogin no
Match Address 192.0.2.0/24
  PasswordAuthentication yes
  PermitRootLogin yes
```

3. **192.0.2.0/24** サブネットから root としてサーバーに接続できることを確認します。

a. IP アドレスを確認します。

```
$ hostname -I
192.0.2.1
```

IP アドレスが **192.0.2.1** - **192.0.2.254** 範囲にある場合は、サーバーに接続できます。

b. **root** でサーバーに接続します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/sshd/** ディレクトリー

1.8.3. ssh RHEL システムロールの変数

ssh システムロール Playbook では、必要や制限に応じてクライアント SSH 設定ファイルのパラメーターを定義できます。

これらの変数が設定されていない場合には、システムロールは RHEL のデフォルト値と同じグローバル **ssh_config** ファイルを作成します。

どのような場合でも、ブール値は **ssh** 設定で適切に **yes** または **no** とレンダリングされます。リストを使用して複数行の設定項目を定義できます。以下に例を示します。

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

レンダリングは以下のようになります。

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

注記

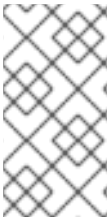
設定オプションでは、大文字と小文字が区別されます。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ssh/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ssh/** ディレクトリー

1.8.4. ssh RHEL システムロールを使用した OpenSSH クライアントの設定

ssh RHEL システムロールを使用して、Ansible Playbook を実行し、複数の SSH クライアントを設定できます。



注記

ssh RHEL システムロールは、SSH および SSHD 設定を変更する他のシステムロール (ID 管理 RHEL システムロールなど) とともに使用できます。設定が上書きされないようにするには、**ssh** ロールがドロップインディレクトリーを使用していること (RHEL 8 以降ではデフォルト) を確認してください。

前提条件

- コントロールノードと管理対象ノードの準備が完了している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。

- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

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

この Playbook は、以下の設定を使用して、マネージドノードで **root** ユーザーの SSH クライアント設定を行います。

- 圧縮が有効になっている。
- ControlMaster multiplexing が **auto** に設定されている。
- **server.example.com** ホストに接続するための **example** エイリアスが **user1** である。
- **example** ホストエイリアスが作成済みで、このエイリアスがユーザー名 **user1** を持つ **server.example.com** ホストへの接続を表している。
- X11 転送が無効化されている。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- SSH 設定ファイルを表示して、管理対象ノードの設定が正しいことを確認します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.ssh/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/ssh/** ディレクトリー

1.8.5. 非排他的設定に sshd RHEL システムロールを使用する

通常、**sshd** システムロールを適用すると、設定全体が上書きされます。これは、たとえば別のシステムロールや Playbook などを使用して、以前に設定を調整している場合に問題を生じる可能性があります。他のオプションを維持しながら、選択した設定オプションにのみ **sshd** システムロールを適用するには、非排他的設定を使用できます。

非排他的設定は、以下を使用して適用できます。

- RHEL 8 以前では、設定スニペットを使用します。
- RHEL 9 以降では、ドロップインディレクトリー内のファイルを使用します。デフォルトの設定ファイルは、**/etc/ssh/sshd_config.d/00-ansible_system_role.conf** としてドロップインディレクトリーにすでに配置されています。

前提条件

- コントロールノードと管理対象ノードの準備が完了している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。
   - RHEL 8 以前を実行する管理対象ノードの場合:

```
---
- name: Non-exclusive sshd configuration
  hosts: managed-node-01.example.com
  tasks:
    - name: <Configure SSHD to accept some useful environment variables>
      ansible.builtin.include_role:
        name: rhel-system-roles.sshd
      vars:
```

```
    sshd_config_namespace: <my-application>
    sshd:
      # Environment variables to accept
      AcceptEnv:
        LANG
        LS_COLORS
        EDITOR
```

- RHEL 9 以降を実行する管理対象ノードの場合:

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

sshd_config_file 変数では、**sshd** システムロールによる設定オプションの書き込み先の **.conf** ファイルを定義します。設定ファイルが適用される順序を指定するには、2 桁の接頭辞 (例: **42-**) を使用します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

- SSH サーバーの設定を確認します。
  - RHEL 8 以前を実行する管理対象ノードの場合:

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

  - RHEL 9 以降を実行する管理対象ノードの場合:

```
# cat /etc/ssh/sshd_config
```

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.sshd/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/sshd/** ディレクトリー

1.9. 関連情報

- **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)**、および **crypto-policies(7)** の man ページ
- 非標準設定でのアプリケーションとサービスの SELinux 設定
- Controlling network traffic using firewalld

第2章 TLS キーと証明書の作成と管理

TLS (Transport Layer Security) プロトコルを使用して、2 つのシステム間で送信される通信を暗号化できます。この標準は、秘密鍵と公開鍵、デジタル署名、および証明書を用いた非対称暗号化を使用します。

2.1. TLS 証明書

TLS (Transport Layer Security) は、クライアントサーバーアプリケーションが情報を安全に受け渡すことを可能にするプロトコルです。TLS は、公開鍵と秘密鍵のペアのシステムを使用して、クライアントとサーバー間で送信される通信を暗号化します。TLS は、SSL (Secure Sockets Layer) の後継プロトコルです。

TLS は、X.509 証明書を使用して、ホスト名や組織などの ID を、デジタル署名を使用する公開鍵にバインドします。X.509 は、公開鍵証明書の形式を定義する標準です。

セキュアなアプリケーションの認証は、アプリケーションの証明書の公開鍵値の整合性によって異なります。攻撃者が公開鍵を独自の公開鍵に置き換えると、真のアプリケーションになりすまして安全なデータにアクセスできるようになります。この種の攻撃を防ぐには、すべての証明書が証明局 (CA) によって署名されている必要があります。CA は、証明書の公開鍵値の整合性を確認する信頼できるノードです。

CA はデジタル署名を追加して公開鍵に署名し、証明書を発行します。デジタル署名は、CA の秘密鍵でエンコードされたメッセージです。CA の公開鍵は、CA の証明書を配布することによって、アプリケーションで使用できるようになります。アプリケーションは、CA の公開鍵を使用して CA のデジタル署名をデコードして、証明書が有効で署名されていることを確認します。

CA によって署名された証明書を取得するには、公開鍵を生成し、署名のために CA に送信する必要があります。これは、証明書署名要求 (CSR) と呼ばれます。CSR には、証明書の識別名 (DN) も含まれています。どちらのタイプの証明書にも提供できる DN 情報には、国を表す 2 文字の国コード、州または県の完全な名前、市または町、組織の名前、メールアドレスを含めることも、空にすることもできます。現在の商用 CA の多くは、Subject Alternative Name 拡張を好み、CSR の DN を無視します。

RHEL は、TLS 証明書GnuTLS と OpenSSL を操作するための 2 つの主要なツールキットを提供します。**openssl** パッケージの **openssl** ユーティリティーを使用して、証明書を作成、読み取り、署名、および検証できます。**gnutls-utils** パッケージで提供される **certtool** ユーティリティーは、異なる構文を使用して同じ操作を行うことができ、特にバックエンドの異なるライブラリーセットを使用できます。

関連情報

- RFC 5280:インターネット X.509 公開鍵インフラストラクチャー証明書および証明書失効リスト (CRL) プロファイル
- **openssl(1)**、**x509(1)**、**ca(1)**、**req(1)**、および **certtool(1)** man ページ

2.2. OPENSSL を使用したプライベート CA の作成

プライベート証明機関 (CA) は、シナリオで内部ネットワーク内のエンティティーを検証する必要がある場合に役立ちます。たとえば、管理下にある CA によって署名された証明書に基づく認証を使用して VPN ゲートウェイを作成する場合、または商用 CA への支払いを希望しない場合は、プライベート CA を使用します。このようなユースケースで証明書に署名するために、プライベート CA は自己署名証明書を使用します。

前提条件

- **sudo** を使用して管理コマンドを入力するため **の root** 権限または権限がある。そのような特権を必要とするコマンドは、**#** でマークされています。

手順

1. CA の秘密鍵を生成します。たとえば、次のコマンドは、256 ビットの楕円曲線デジタル署名アルゴリズム (ECDSA) キーを作成します。

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

キー生成プロセスの時間は、ホストのハードウェアとエントロピー、選択したアルゴリズム、およびキーの長さによって異なります。

2. 前のコマンドで生成された秘密鍵を使用して署名された証明書を作成します。

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

生成された **ca.crt** ファイルは、10 年間、他の証明書の署名に使用できる自己署名 CA 証明書です。プライベート CA の場合、**<Example CA>** を共通名 (CN) として任意の文字列に置き換えることができます。

3. CA の秘密鍵に安全なアクセス許可を設定します。次に例を示します。

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

次のステップ

- 自己署名 CA 証明書をクライアントシステムのトラストアンカーとして使用するには、CA 証明書をクライアントにコピーし、クライアントのシステム全体のトラストストアに **root** として追加します。

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

詳細は、*3章共通システム証明書の使用* を参照してください。

検証

1. 証明書署名要求 (CSR) を作成し、CA を使用して要求に署名します。CA は、CSR に基づいて証明書を正常に作成する必要があります。次に例を示します。

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

詳細は、「プライベート CA を使用した OpenSSL での CSR の証明書の発行」 を参照してください。

2. 自己署名 CA に関する基本情報を表示します。

```
$ openssl x509 -in <ca.crt> -text -noout
Certificate:
…
        X509v3 extensions:
            …
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
…
```

3. 秘密鍵の一貫性を確認します。

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

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

関連情報

- **openssl(1)**、**ca(1)**、**genpkey(1)**、**x509(1)**、および **req(1)** man ページ

2.3. OPENSSL を使用した TLS サーバー証明書の秘密鍵と CSR の作成

認証局 (CA) からの有効な TLS 証明書がある場合にのみ、TLS 暗号化通信チャネルを使用できます。証明書を取得するには、最初にサーバーの秘密鍵と証明書署名要求 (CSR) を作成する必要があります。

手順

1. 以下のようにサーバーシステムで秘密鍵を生成します。

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

2. オプション: 次の例のように、選択したテキストエディターを使用して、CSR の作成を簡素化する設定ファイルを準備します。

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

[req]
distinguished_name = dn
prompt = no

[dn]
C = <US>
```

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

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

extendedKeyUsage = serverAuth オプションは、証明書の使用を制限します。

3. 前に作成した秘密鍵を使用して CSR を作成します。

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

-config オプションを省略すると、**req** ユーティリティーは次のような追加情報の入力を求めます。

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

次のステップ

- 署名のために選択した CA に CSR を送信します。または、信頼できるネットワーク内での内部使用シナリオでは、署名にプライベート CA を使用します。詳細は、「プライベート CA を使用した OpenSSL での CSR の証明書の発行」 を参照してください。

検証

1. 要求された証明書を CA から取得したら、次の例のように、証明書の中で人間が判読できる部分が要件と一致することを確認します。

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

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

関連情報

- **openssl(1)**、**x509(1)**、**genpkey(1)**、**req(1)**、および **config(5)** man ページ

2.4. OPENSSL を使用した TLS クライアント証明書の秘密鍵と CSR の作成

認証局 (CA) からの有効な TLS 証明書がある場合にのみ、TLS 暗号化通信チャネルを使用できます。証明書を取得するには、最初にクライアントの秘密鍵と証明書署名要求 (CSR) を作成する必要があります。

手順

1. 次の例のように、クライアントシステムで秘密鍵を生成します。

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

2. オプション: 次の例のように、選択したテキストエディターを使用して、CSR の作成を簡素化する設定ファイルを準備します。

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

[req]
distinguished_name = dn
prompt = no

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

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

extendedKeyUsage = clientAuth オプションは、証明書の使用を制限します。

3. 前に作成した秘密鍵を使用して CSR を作成します。

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

-config オプションを省略すると、**req** ユーティリティーは次のような追加情報の入力を求めます。

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

次のステップ

- 署名のために選択した CA に CSR を送信します。または、信頼できるネットワーク内での内部使用シナリオでは、署名にプライベート CA を使用します。詳細は、「プライベート CA を使用した OpenSSL での CSR の証明書の発行」 を参照してください。

検証

1. 次の例のように、証明書の中で人間が判読できる部分が要件と一致することを確認します。

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

関連情報

- **openssl(1)**、**x509(1)**、**genpkey(1)**、**req(1)**、および **config(5)** man ページ

2.5. プライベート CA を使用した OPENSSL での CSR の証明書の発行

システムが TLS 暗号化通信チャネルを確立できるようにするには、認証局 (CA) が有効な証明書をシステムに提供する必要があります。プライベート CA がある場合は、システムからの証明書署名要求 (CSR) に署名することにより、要求された証明書を作成できます。

前提条件

- プライベート CA が設定済みである。詳細は、「OpenSSL を使用したプライベート CA の作成」 を参照してください。
- CSR を含むファイルがある。CSR の作成例は、「OpenSSL を使用した TLS サーバー証明書の秘密鍵と CSR の作成」 にあります。

手順

1. オプション: 任意のテキストエディターを使用して、次の例のように、証明書に拡張機能を追加するための OpenSSL 設定ファイルを準備します。

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

```
extendedKeyUsage = serverAuth

[client-cert]
extendedKeyUsage = clientAuth
```

2. **x509** ユーティリティーを使用して、CSR に基づいて証明書を作成します。次に例を示します。

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

関連情報

- **openssl(1)**、**ca(1)**、および **x509(1)** man ページ

2.6. GNUTLS を使用したプライベート CA の作成

プライベート証明機関 (CA) は、シナリオで内部ネットワーク内のエンティティーを検証する必要がある場合に役立ちます。たとえば、管理下にある CA によって署名された証明書に基づく認証を使用して VPN ゲートウェイを作成する場合、または商用 CA への支払いを希望しない場合は、プライベート CA を使用します。このようなユースケースで証明書に署名するために、プライベート CA は自己署名証明書を使用します。

前提条件

- **sudo** を使用して管理コマンドを入力するため **の root** 権限または権限がある。そのような特権を必要とするコマンドは、**#** でマークされています。
- システムにはすでに GnuTLS がインストールされている。インストールしていない場合は、次のコマンドを使用できます。

```
$ dnf install gnutls-utils
```

手順

1. CA の秘密鍵を生成します。たとえば、次のコマンドは、256 ビットの楕円曲線デジタル署名アルゴリズム (ECDSA) キーを作成します。

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

キー生成プロセスの時間は、ホストのハードウェアとエントロピー、選択したアルゴリズム、およびキーの長さによって異なります。

2. 証明書のテンプレートファイルを作成します。

   a. 任意のテキストエディターでファイルを作成します。以下に例を示します。

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

   b. ファイルを編集して、必要な証明書の詳細を含めます。

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

3. 手順 1 で生成した秘密鍵を使用して署名された証明書を作成します。
生成された **<ca.crt>** ファイルは、1 年間他の証明書に署名するために使用できる自己署名 CA 証明書です。**<ca.crt>** ファイルは公開キー (証明書) です。ロードされたファイル **<ca.key>** が秘密鍵です。このファイルは安全な場所に保管してください。

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

4. CA の秘密鍵に安全なアクセス許可を設定します。次に例を示します。

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

次のステップ

- 自己署名 CA 証明書をクライアントシステムのトラストアンカーとして使用するには、CA 証明書をクライアントにコピーし、クライアントのシステム全体のトラストストアに **root** として追加します。

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

詳細は、*3章共通システム証明書の使用* を参照してください。

検証

1. 自己署名 CA に関する基本情報を表示します。

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

2. 証明書署名要求 (CSR) を作成し、CA を使用して要求に署名します。CA は、CSR に基づいて証明書を正常に作成する必要があります。次に例を示します。

   a. CA の秘密鍵を生成します。

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

b. 任意のテキストエディターで新しい設定ファイルを開きます。以下に例を示します。

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

c. ファイルを編集して、必要な証明書の詳細を含めます。

```
signing_key
encryption_key
key_agreement

tls_www_server

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

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

d. 以前に作成した秘密鍵を使用してリクエストを生成します

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

e. 証明書を生成し、CA の秘密鍵で署名します。

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

関連情報

- **certtool(1)** および **trust(1)** の man ページ

2.7. GNUTLS を使用した TLS サーバー証明書の秘密鍵と CSR の作成

証明書を取得するには、最初にサーバーの秘密鍵と証明書署名要求 (CSR) を作成する必要があります。

手順

1. 以下のようにサーバーシステムで秘密鍵を生成します。

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

2. オプション: 次の例のように、選択したテキストエディターを使用して、CSR の作成を簡素化する設定ファイルを準備します。

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

```
tls_www_server

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

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

3. 前に作成した秘密鍵を使用して CSR を作成します。

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

--template オプションを省略すると、**certool** ユーティリティーは次のような追加情報の入力を求めます。

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

次のステップ

- 署名のために選択した CA に CSR を送信します。または、信頼できるネットワーク内での内部使用シナリオでは、署名にプライベート CA を使用します。詳細は、「プライベート CA を使用した GnuTLS での CSR の証明書の発行」 を参照してください。

検証

1. 要求された証明書を CA から取得したら、次の例のように、証明書の中で人間が判読できる部分が要件と一致することを確認します。

```
$ certtool --certificate-info --infile <example-server.crt>
Certificate:
…
        Issuer: CN = Example CA
        Validity
            Not Before: Feb  2 20:27:29 2023 GMT
            Not After : Feb  2 20:27:29 2024 GMT
```

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

関連情報

- **certtool(1)** の man ページ

2.8. GNUTLS を使用した TLS クライアント証明書の秘密鍵と CSR の作成

証明書を取得するには、最初にクライアントの秘密鍵と証明書署名要求 (CSR) を作成する必要があります。

手順

1. 次の例のように、クライアントシステムで秘密鍵を生成します。

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

2. オプション: 次の例のように、選択したテキストエディターを使用して、CSR の作成を簡素化する設定ファイルを準備します。

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

tls_www_client

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

3. 前に作成した秘密鍵を使用して CSR を作成します。

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

--template オプションを省略すると、**certtool** ユーティリティーは次のような追加情報の入力を求めます。

```
Generating a PKCS #10 certificate request...
Country name (2 chars): <US>
State or province name: <Washington>
Locality name: <Seattle>
```

```
Organization name: <Example Organization>
Organizational unit name:
Common name: <server.example.com>
```

次のステップ

- 署名のために選択した CA に CSR を送信します。または、信頼できるネットワーク内での内部使用シナリオでは、署名にプライベート CA を使用します。詳細は、「プライベート CA を使用した GnuTLS での CSR の証明書の発行」 を参照してください。

検証

1. 次の例のように、証明書の中で人間が判読できる部分が要件と一致することを確認します。

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

関連情報

- **certtool(1)** の man ページ

2.9. プライベート CA を使用した GNUTLS での CSR の証明書の発行

システムが TLS 暗号化通信チャネルを確立できるようにするには、認証局 (CA) が有効な証明書をシステムに提供する必要があります。プライベート CA がある場合は、システムからの証明書署名要求 (CSR) に署名することにより、要求された証明書を作成できます。

前提条件

- プライベート CA が設定済みである。詳細は、「GnuTLS を使用したプライベート CA の作成」 を参照してください。
- CSR を含むファイルがある。CSR の作成例は、「GnuTLS を使用した TLS サーバー証明書の秘密鍵と CSR の作成」 にあります。

手順

1. オプション: 任意のテキストエディターを使用して、次の例のように、証明書に拡張機能を追加するための GnuTLS 設定ファイルを準備します。

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

2. **certtool** ユーティリティーを使用して、CSR に基づいて証明書を作成します。次に例を示します。

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

関連情報

- **certtool(1)** の man ページ

第3章 共通システム証明書の使用

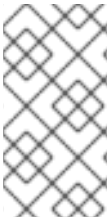
共有システム証明書ストレージは、NSS、GnuTLS、OpenSSL、および Java が、システムの証明書アンカーと、拒否リスト情報を取得するデフォルトソースを共有します。トラストストアには、デフォルトで、Mozilla CA リスト (信頼できるリストおよび信頼できないリスト) が含まれています。システムでは、コア Mozilla CA リストを更新したり、別の証明書リストを選択したりできます。

3.1. システム全体のトラストストア

RHEL では、統合されたシステム全体のトラストストアが **/etc/pki/ca-trust/** ディレクトリーおよび **/usr/share/pki/ca-trust-source/** ディレクトリーに置かれています。**/usr/share/pki/ca-trust-source/** のトラスト設定は、**/etc/pki/ca-trust/** の設定よりも低い優先順位で処理されます。

証明書ファイルは、インストールされているサブディレクトリーによって扱われ方が異なります。

- トラストアンカーの所属先
  - **/usr/share/pki/ca-trust-source/anchors/** または
  - **/etc/pki/ca-trust/source/anchors/**
- 信頼されていない証明書の保存先
  - **/usr/share/pki/ca-trust-source/blocklist/** または
  - **/etc/pki/ca-trust/source/blocklist/**
- 拡張 BEGIN TRUSTED ファイル形式の証明書の配置先
  - **/usr/share/pki/ca-trust-source/** または
  - **/etc/pki/ca-trust/source/**



注記

階層暗号化システムでは、トラストアンカーとは、他のパーティーが信頼できると想定する権威あるエンティティーです。X.509 アーキテクチャーでは、ルート証明書はトラストチェーンの元となるトラストアンカーです。チェーンの検証を有効にするには、信頼元がまずトラストアンカーにアクセスできる必要があります。

関連情報

- **update-ca-trust(8)** および **trust(1)** の man ページ

3.2. 新しい証明書の追加

新しいソースでシステムのアプリケーションを確認するには、対応する証明書をシステム全体のストアに追加し、**update-ca-trust** コマンドを使用します。

前提条件

- **ca-certificates** パッケージがシステムにインストールされている。

手順

1. システムで信頼されている CA のリストに、シンプルな PEM または DER のファイルフォーマットに含まれる証明書を追加するには、**/usr/share/pki/ca-trust-source/anchors/** ディレクトリーまたは **/etc/pki/ca-trust/source/anchors/** ディレクトリーに証明書ファイルをコピーします。以下に例を示します。

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

2. システム全体のトラストストア設定を更新するには、**update-ca-trust** コマンドを実行します。

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

注記

update-ca-trust を事前に実行しなくても、Firefox ブラウザーは追加された証明書を使用できますが、CA を変更するたびに **update-ca-trust** コマンドを入力してください。Firefox、Chromium および GNOME Web などのブラウザーはファイルをキャッシュするので、ブラウザーのキャッシュをクリアするか、ブラウザーを再起動して、現在のシステム証明書の設定を読み込む必要がある場合があります。

関連情報

- **update-ca-trust(8)** および **trust(1)** の man ページ

3.3. 信頼されているシステム証明書の管理

trust コマンドを使用すると、システム全体の共有トラストストアの証明書を便利な方法で管理できます。

- トラストアンカーのリスト表示、抽出、追加、削除、または変更を行うには、**trust** コマンドを使用します。このコマンドの組み込みヘルプを表示するには、引数を付けずに、または **--help** ディレクティブを付けて実行します。

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

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

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

- すべてのシステムのトラストアンカーおよび証明書のリストを表示するには、**trust list** コマンドを実行します。

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

```
    category: authority

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

- トラストアンカーをシステム全体のトラストストアに保存するには、**trust anchor** サブコマンドを使用し、証明書のパスを指定します。**<path.to/certificate.crt>** を、証明書およびそのファイル名へのパスに置き換えます。

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

- 証明書を削除するには、証明書のパス、または証明書の ID を使用します。

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

関連情報

- **trust** コマンドのすべてのサブコマンドは、以下のような詳細な組み込みヘルプを提供します。

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

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

関連情報

- **update-ca-trust(8)** および **trust(1)** の man ページ

第4章 TLS の計画および実施

TLS (トランスポート層セキュリティー) は、ネットワーク通信のセキュリティー保護に使用する暗号化プロトコルです。優先する鍵交換プロトコル、認証方法、および暗号化アルゴリズムを設定してシステムのセキュリティー設定を強化する際には、対応するクライアントの範囲が広ければ広いほど、セキュリティーのレベルが低くなることを認識しておく必要があります。反対に、セキュリティー設定を厳密にすると、クライアントとの互換性が制限され、システムからロックアウトされるユーザーが出てくる可能性もあります。可能な限り厳密な設定を目指し、互換性に必要な場合に限り、設定を緩めるようにしてください。

4.1. SSL プロトコルおよび TLS プロトコル

Secure Sockets Layer (SSL) プロトコルは、元々はインターネットを介した安全な通信メカニズムを提供するために、Netscape Corporation により開発されました。その後、このプロトコルは、Internet Engineering Task Force (IETF) により採用され、Transport Layer Security (TLS) に名前が変更になりました。

TLS プロトコルは、アプリケーションプロトコル層と、TCP/IP などの信頼性の高いトランスポート層の間にあります。これは、アプリケーションプロトコルから独立しているため、さまざまなプロトコルの下に階層化できます。(HTTP、FTP、SMTP など)

プロトコルのバージョン	推奨される使用方法
SSL v2	使用しないでください。深刻なセキュリティー上の脆弱性があります。RHEL 7 以降、コア暗号ライブラリーから削除されました。
SSL v3	使用しないでください。深刻なセキュリティー上の脆弱性があります。RHEL 8 以降、コア暗号ライブラリーから削除されました。
TLS 1.0	使用は推奨されません。相互運用性を保証した方法では軽減できない既知の問題があり、最新の暗号スイートには対応しません。RHEL 9 では、すべての暗号化ポリシーで無効になります。
TLS 1.1	必要に応じて相互運用性の目的で使用します。最新の暗号スイートには対応しません。RHEL 9 では、すべての暗号化ポリシーで無効になります。
TLS 1.2	最新の AEAD 暗号スイートに対応します。このバージョンは、システム全体のすべての暗号化ポリシーで有効になっていますが、このプロトコルの必須ではない部分に脆弱性があります。また、TLS 1.2 では古いアルゴリズムも使用できます。
TLS 1.3	推奨されるバージョン。TLS 1.3 は、既知の問題があるオプションを取り除き、より多くのネゴシエーションハンドシェイクを暗号化することでプライバシーを強化し、最新の暗号アルゴリズムをより効果的に使用することで速度を速めることができます。TLS 1.3 は、システム全体のすべての暗号化ポリシーでも有効になっています。

関連情報

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

4.2. RHEL 9 における TLS のセキュリティー上の検討事項

RHEL 9 では、TLS 設定はシステム全体の暗号化ポリシーメカニズムを使用して実行されます。1.2 未満の TLS バージョンはサポートされなくなりました。**DEFAULT**、**FUTURE**、および **LEGACY** 暗号化ポリシーは、TLS 1.2 および 1.3 のみを許可します。詳細は、Using system-wide cryptographic policies を参照してください。

RHEL 9 に含まれるライブラリーが提供するデフォルト設定は、ほとんどのデプロイメントで十分に安全です。TLS 実装は、可能な場合は、安全なアルゴリズムを使用する一方で、レガシーなクライアントまたはサーバーとの間の接続は妨げません。セキュリティーが保護されたアルゴリズムまたはプロトコルに対応しないレガシーなクライアントまたはサーバーの接続が期待できないまたは許可されない場合に、厳密なセキュリティー要件の環境で、強化設定を適用します。

TLS 設定を強化する最も簡単な方法は、**update-crypto-policies --set FUTURE** コマンドを実行して、システム全体の暗号化ポリシーレベルを **FUTURE** に切り替えます。

警告

LEGACY 暗号化ポリシーで無効にされているアルゴリズムは、Red Hat の RHEL 9 セキュリティーのビジョンに準拠しておらず、それらのセキュリティープロパティーは信頼できません。これらのアルゴリズムを再度有効化するのではなく、使用しないようにすることを検討してください。たとえば、古いハードウェアとの相互運用性のためにそれらを再度有効化することを決めた場合は、それらを安全でないものとして扱い、ネットワークの相互作用を個別のネットワークセグメントに分離するなどの追加の保護手段を適用します。パブリックネットワーク全体では使用しないでください。

RHEL システム全体の暗号化ポリシーに従わない場合、またはセットアップに適したカスタム暗号化ポリシーを作成する場合は、カスタム設定で必要なプロトコル、暗号スイート、および鍵の長さについて、以下の推奨事項を使用します。

4.2.1. プロトコル

最新バージョンの TLS は、最高のセキュリティーメカニズムを提供します。TLS 1.2 は、**LEGACY** 暗号化ポリシーを使用する場合でも最小バージョンになりました。古いプロトコルバージョンを再度有効にするには、暗号化ポリシーをオプトアウトするか、カスタムポリシーを提供することで可能ですが、この結果生成される設定はサポートされません。

RHEL 9 は TLS バージョン 1.3 をサポートしていますが、このプロトコルのすべての機能が RHEL 9 コンポーネントで完全にサポートされているわけではない点に注意してください。たとえば、接続レイテンシーを短縮する 0-RTT (Zero Round Trip Time) 機能は、Apache Web サーバーではまだ完全にはサポートされていません。

警告

FIPS モードで実行されている RHEL 9.2 以降のシステムでは、FIPS 140-3 標準の要件に従って、TLS 1.2 接続で Extended Master Secret (EMS) 拡張機能 (RFC 7627) を使用する必要があります。したがって、EMS または TLS 1.3 をサポートしていないレガシークライアントは、FIPS モードで実行されている RHEL 9 サーバーに接続できません。FIPS モードの RHEL 9 クライアントは、EMS なしで TLS 1.2 のみをサポートするサーバーに接続できません。TLS Extension "Extended Master Secret" enforced with Red Hat Enterprise Linux 9.2 を参照してください。

4.2.2. 暗号化スイート

旧式で、安全ではない暗号化スイートではなく、最近の、より安全なものを使用してください。暗号化スイートの eNULL および aNULL は、暗号化や認証を提供しないため、常に無効にしてください。RC4 や HMAC-MD5 をベースとした暗号化スイートには深刻な欠陥があるため、可能な場合はこれも無効にしてください。いわゆるエクスポート暗号化スイートも同様です。エクスポート暗号化スイートは意図的に弱くなっているため、侵入が容易になっています。

128 ビット未満のセキュリティーしか提供しない暗号化スイートでは直ちにセキュリティーが保護されなくなるというわけではありませんが、使用できる期間が短いため考慮すべきではありません。アルゴリズムが 128 ビット以上のセキュリティーを使用している場合は、少なくとも数年間は解読不可能であることが期待されているため、強く推奨されます。3DES 暗号は 168 ビットを使用していると言われていますが、実際に提供されているのは 112 ビットのセキュリティーであることに注意してください。

サーバーの鍵が危険にさらされた場合でも、暗号化したデータの機密性を保証する (完全な) 前方秘匿性 (PFS) に対応する暗号スイートを常に優先します。ここでは、速い RSA 鍵交換は除外されますが、ECDHE および DHE は使用できます。この 2 つを比べると、ECDHE の方が速いため推奨されます。

また、AES-GCM などの AEAD 暗号は、パディングオラクル攻撃の影響は受けないため、CBC モード暗号よりも推奨されます。さらに、多くの場合、特にハードウェアに AES 用の暗号化アクセラレーターがある場合、AES-GCM は CBC モードの AES よりも高速です。

ECDSA 証明書で ECDHE 鍵交換を使用すると、トランザクションは純粋な RSA 鍵交換よりもさらに高速になります。レガシークライアントに対応するため、サーバーには証明書と鍵のペアを 2 つ (新しいクライアント用の ECDSA 鍵と、レガシー用の RSA 鍵) インストールできます。

4.2.3. 公開鍵の長さ

RSA 鍵を使用する際は、SHA-256 以上で署名され、鍵の長さが 3072 ビット以上のものが常に推奨されます (これは、実際に 128 ビットであるセキュリティーに対して十分な大きさです)。

警告

システムのセキュリティー強度は、チェーンの中の最も弱いリンクが示すものと同じになります。たとえば、強力な暗号化だけではすぐれたセキュリティーは保証されません。鍵と証明書も同様に重要で、認証機関 (CA) が鍵の署名に使用するハッシュ機能と鍵もまた重要になります。

4.3. アプリケーションで TLS 設定の強化

RHEL では、システム全体の暗号化ポリシー は、暗号化ライブラリーを使用するアプリケーションが、既知の安全でないプロトコル、暗号化、またはアルゴリズムを許可しないようにするための便利な方法を提供します。

暗号化設定をカスタマイズして、TLS 関連の設定を強化する場合は、このセクションで説明する暗号化設定オプションを使用して、必要最小量でシステム全体の暗号化ポリシーを上書きできます。

いずれの設定を選択しても、サーバーアプリケーションが **サーバー側が指定した順序** で暗号を利用することを確認し、使用される暗号化スイートの選択がサーバーでの設定順に行われるように設定してください。

4.3.1. TLS を使用するように Apache HTTP サーバーを設定

Apache HTTP Server は、TLS のニーズに **OpenSSL** ライブラリーおよび **NSS** ライブラリーの両方を使用できます。RHEL 9 では、mod_ss パッケージで **mod_ssl** 機能が提供されます。

```
# dnf install mod_ssl
```

mod_ssl パッケージは、**/etc/httpd/conf.d/ssl.conf** 設定ファイルをインストールします。これは、**Apache HTTP Server** の TLS 関連の設定を変更するのに使用できます。

httpd-manual パッケージをインストールして、TLS 設定を含む **Apache HTTP Server** の完全ドキュメントを取得します。**/etc/httpd/conf.d/ssl.conf** 設定ファイルで利用可能なディレクティブの詳細は、**/usr/share/httpd/manual/mod/mod_ssl.html** を参照してください。さまざまな設定の例は、**/usr/share/httpd/manual/ssl/ssl_howto.html** ファイルに記載されています。

/etc/httpd/conf.d/ssl.conf 設定ファイルの設定を修正する場合は、少なくとも下記の 3 つのディレクティブを確認してください。

SSLProtocol
: このディレクティブを使用して、許可する TLS または SSL のバージョンを指定します。

SSLCipherSuite
: 優先する暗号化スイートを指定する、もしくは許可しないスイートを無効にするディレクティブです。

SSLHonorCipherOrder
: コメントを解除して、このディレクティブを **on** に設定すると、接続先のクライアントは指定した暗号化の順序に従います。

たとえば、TLS 1.2 プロトコルおよび 1.3 プロトコルだけを使用する場合は、以下を実行します。

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

詳細は、Deploying web servers and reverse proxies の Configuring TLS encryption on an Apache HTTP Server の章を参照してください。

4.3.2. TLS を使用するように Nginx HTTP およびプロキシーサーバーを設定

Nginx で TLS 1.3 サポートを有効にするには、**/etc/nginx/nginx.conf** 設定ファイルの **server** セクションで、**ssl_protocols** オプションに **TLSv1.3** 値を追加します。

```
server {
```

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

詳細は、Deploying web servers and reverse proxies の Adding TLS encryption to an Nginx web server の章を参照してください。

4.3.3. TLS を使用するように Dovecot メールサーバーを設定

Dovecot メールサーバーのインストールが TLS を使用するように設定するには、**/etc/dovecot/conf.d/10-ssl.conf** 設定ファイルを修正します。このファイルで利用可能な基本的な設定ディレクティブの一部は、**/usr/share/doc/dovecot/wiki/SSL.DovecotConfiguration.txt** ファイルで説明されています。このファイルは **Dovecot** の標準インストールに含まれています。

/etc/dovecot/conf.d/10-ssl.conf 設定ファイルの設定を修正する場合は、少なくとも下記の 3 つのディレクティブを確認してください。

ssl_protocols

このディレクティブを使用して、許可または無効にする TLS または SSL のバージョンを指定します。

ssl_cipher_list

優先する暗号化スイートを指定する、もしくは許可しないスイートを無効にするディレクティブです。

ssl_prefer_server_ciphers

コメントを解除して、このディレクティブを **yes** に設定すると、接続先のクライアントは指定した暗号化の順序に従います。

たとえば、**/etc/dovecot/conf.d/10-ssl.conf** 内の次の行が、TLS 1.1 以降だけを許可します。

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

関連情報

- Web サーバーとリバースプロキシーのデプロイ
- **config(5)** および **ciphers(1)** の man ページ
- Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)
- Mozilla SSL Configuration Generator
- SSL Server Test

第5章 IPSEC を使用した VPN の設定

RHEL 9 では、**IPsec** プロトコルを使用して仮想プライベートネットワーク (VPN) を設定できます。これは、Libreswan アプリケーションによりサポートされます。

5.1. IPSEC VPN 実装としての LIBRESWAN

RHEL では、IPsec プロトコルを使用して仮想プライベートネットワーク (VPN) を設定できます。これは、Libreswan アプリケーションによりサポートされます。Libreswan は、Openswan アプリケーションの延長であり、Openswan ドキュメントの多くの例は Libreswan でも利用できます。

VPN の IPsec プロトコルは、IKE (Internet Key Exchange) プロトコルを使用して設定されます。IPsec と IKE は同義語です。IPsec VPN は、IKE VPN、IKEv2 VPN、XAUTH VPN、Cisco VPN、または IKE/IPsec VPN とも呼ばれます。Layer 2 Tunneling Protocol (L2TP) も使用する IPsec VPN のバリアントは、通常、L2TP/IPsec VPN と呼ばれ、**optional** のリポジトリーによって提供される **xl2tpd** パッケージが必要です。

Libreswan は、オープンソースのユーザー空間の IKE 実装です。IKE v1 および v2 は、ユーザーレベルのデーモンとして実装されます。IKE プロトコルも暗号化されています。IPsec プロトコルは Linux カーネルで実装され、Libreswan は、VPN トンネル設定を追加および削除するようにカーネルを設定します。

IKE プロトコルは、UDP ポート 500 および 4500 を使用します。IPsec プロトコルは、以下の 2 つのプロトコルで構成されます。

- 暗号セキュリティーペイロード (ESP) (プロトコル番号が 50)
- 認証ヘッダー (AH) (プロトコル番号 51)

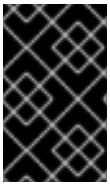
AH プロトコルの使用は推奨されていません。AH のユーザーは、null 暗号化で ESP に移行することが推奨されます。

IPsec プロトコルは、以下の 2 つの操作モードを提供します。

- トンネルモード (デフォルト)
- トランスポートモード

IKE を使用せずに IPsec を使用してカーネルを設定できます。これは、**手動キーリング** と呼ばれます。また、**ip xfrm** コマンドを使用して手動キーを設定できますが、これはセキュリティー上の理由からは強く推奨されません。Libreswan は、Netlink インターフェイスを使用して Linux カーネルと通信します。カーネルはパケットの暗号化と復号化を実行します。

Libreswan は、ネットワークセキュリティーサービス (NSS) 暗号化ライブラリーを使用します。NSS は、**連邦情報処理標準** (**FIPS**) の公開文書 140-2 での使用が認定されています。



重要

Libreswan および Linux カーネルが実装する IKE/IPsec の VPN は、RHEL で使用することが推奨される唯一の VPN 技術です。その他の VPN 技術は、そのリスクを理解せずに使用しないでください。

RHEL では、Libreswan はデフォルトで **システム全体の暗号化ポリシー** に従います。これにより、Libreswan は、デフォルトのプロトコルとして IKEv2 を含む現在の脅威モデルに対して安全な設定を使用するようになります。詳細は、Using system-wide crypto policies を参照してください。

IKE/IPsec はピアツーピアプロトコルであるため、Libreswan では、ソースおよび宛先、またはサーバーおよびクライアントという用語を使用しません。終了点 (ホスト) を参照する場合は、代わりに左と右という用語を使用します。これにより、ほとんどの場合、両方の終了点で同じ設定も使用できます。ただし、管理者は通常、ローカルホストに左を使用し、リモートホストに右を使用します。

leftid と **rightid** オプションは、認証プロセス内の各ホストの識別として機能します。詳細は、man ページの **ipsec.conf(5)** を参照してください。

5.2. LIBRESWAN の認証方法

Libreswan は複数の認証方法をサポートしますが、それぞれは異なるシナリオとなっています。

事前共有キー (PSK)

事前共有キー (PSK) は、最も簡単な認証メソッドです。セキュリティー上の理由から、PSK は 64 文字未満は使用しないでください。FIPS モードでは、PSK は、使用される整合性アルゴリズムに応じて、、最低強度の要件に準拠する必要があります。**authby=secret** 接続を使用して PSK を設定できます。

Raw RSA 鍵

Raw RSA 鍵 は、静的なホスト間またはサブネット間の IPsec 設定で一般的に使用されます。各ホストは、他のすべてのホストのパブリック RSA 鍵を使用して手動で設定され、Libreswan はホストの各ペア間で IPsec トンネルを設定します。この方法は、多数のホストでは適切にスケーリングされません。

ipsec newhostkey コマンドを使用して、ホストで Raw RSA 鍵を生成できます。 **ipsec showhostkey** コマンドを使用して、生成された鍵をリスト表示できます。**leftrsasigkey=** の行は、CKA ID キーを使用する接続設定に必要です。Raw RSA 鍵に **authby=rsasig** 接続オプションを使用します。

X.509 証明書

X.509 証明書 は、共通の IPsec ゲートウェイに接続するホストが含まれる大規模なデプロイメントに一般的に使用されます。中央の **認証局** (CA) は、ホストまたはユーザーの RSA 証明書に署名します。この中央 CA は、個別のホストまたはユーザーの取り消しを含む、信頼のリレーを行います。

たとえば、**openssl** コマンドおよび NSS **certutil** コマンドを使用して X.509 証明書を生成できます。Libreswan は、**leftcert=** 設定オプションの証明書のニックネームを使用して NSS データベースからユーザー証明書を読み取るため、証明書の作成時にニックネームを指定します。

カスタム CA 証明書を使用する場合は、これを Network Security Services(NSS) データベースにインポートする必要があります。**ipsec import** コマンドを使用して、PKCS #12 形式の証明書を Libreswan NSS データベースにインポートできます。

警告

Libreswan は、section 3.1 of RFC 4945 で説明されているように、すべてのピア証明書のサブジェクト代替名 (SAN) としてインターネット鍵 Exchange(IKE) ピア ID を必要とします。**require-id-on-certificated=** オプションを変更してこのチェックを無効にすると、システムが中間者攻撃に対して脆弱になる可能性があります。

SHA-2 で RSA を使用した X.509 証明書に基づく認証に **authby=rsasig** 接続オプションを使用します。**authby=** を **ecdsa** に設定し、**authby=rsa-sha2** を介した SHA-2 による RSA Probabilistic Signature Scheme (RSASSA-PSS) デジタル署名ベースの認証を設定することにより、SHA-2 を使用す

る ECDSA デジタル署名に対してさらに制限することができます。デフォルト値は **authby=rsasig,ecdsa** です。

証明書と **authby=** 署名メソッドが一致する必要があります。これにより、相互運用性が向上し、1 つのデジタル署名システムでの認証が維持されます。

NULL 認証

null 認証 は、認証なしでメッシュの暗号化を取得するために使用されます。これは、パッシブ攻撃は防ぎますが、アクティブ攻撃は防ぎません。ただし、IKEv2 は非対称認証メソッドを許可するため、NULL 認証はインターネットスケールのオポチュニスティック IPsec にも使用できます。このモデルでは、クライアントはサーバーを認証しますが、サーバーはクライアントを認証しません。このモデルは、TLS を使用して Web サイトのセキュリティーを保護するのと似ています。NULL 認証に **authby=null** を使用します。

量子コンピューターに対する保護

上記の認証方法に加えて、**Post-quantum Pre-shared Key** (PPK) メソッドを使用して、量子コンピューターによる潜在的な攻撃から保護することができます。個々のクライアントまたはクライアントグループは、帯域外で設定された事前共有鍵に対応する PPK ID を指定することにより、独自の PPK を使用できます。

事前共有キーで IKEv1 を使用すると、量子攻撃者から保護されます。IKEv2 の再設計は、この保護をネイティブに提供しません。Libreswan は、**Post-quantum Pre-shared Key** (PPK) を使用して、量子攻撃から IKEv2 接続を保護します。

任意の PPK 対応を有効にする場合は、接続定義に **ppk=yes** を追加します。PPK が必要な場合は **ppk=insist** を追加します。次に、各クライアントには、帯域外で通信する (および可能であれば量子攻撃に対して安全な) シークレット値を持つ PPK ID を付与できます。PPK はランダム性において非常に強力で、辞書の単語に基づいていません。PPK ID と PPK データは、次のように **ipsec.secrets** ファイルに保存されます。

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

PPKS オプションは、静的な PPK を参照します。実験的な関数は、ワンタイムパッドに基づいた動的 PPK を使用します。各接続では、ワンタイムパッドの新しい部分が PPK として使用されます。これを使用すると、ファイル内の動的な PPK の部分がゼロで上書きされ、再利用を防ぐことができます。複数のタイムパッドマテリアルが残っていないと、接続は失敗します。詳細は、man ページの **ipsec.secrets(5)** を参照してください。

警告

動的 PPK の実装はサポート対象外のテクノロジープレビューとして提供されます。注意して使用してください。

5.3. LIBRESWAN のインストール

Libreswan IPsec/IKE 実装を通じて VPN を設定する前に、対応するパッケージをインストールし、**ipsec** サービスを開始して、ファイアウォールでサービスを許可する必要があります。

前提条件

- **AppStream** リポジトリーが有効になっている。

手順

1. **libreswan** パッケージをインストールします。

```
# dnf install libreswan
```

2. Libreswan を再インストールする場合は、古いデータベースファイルを削除し、新しいデータベースを作成します。

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

3. **ipsec** サービスを開始して有効にし、システムの起動時にサービスを自動的に開始できるようにします。

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

4. ファイアウォールで、**ipsec** サービスを追加して、IKE プロトコル、ESP プロトコル、および AH プロトコルの 500/UDP ポートおよび 4500/UDP ポートを許可するように設定します。

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

5.4. ホスト間の VPN の作成

raw RSA キーによる認証を使用して、**左** および **右** と呼ばれる 2 つのホスト間に、ホストツーホスト IPsec VPN を作成するように Libreswan を設定できます。

前提条件

- Libreswan がインストールされ、**ipsec** サービスが各ノードで開始している。

手順

1. 各ホストで Raw RSA 鍵ペアを生成します。

```
# ipsec newhostkey
```

2. 前の手順で生成した鍵の **ckaid** を返します。**左** で次のコマンドを実行して、その **ckaid** を使用します。以下に例を示します。

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

上のコマンドの出力により、設定に必要な **leftrsasigkey=** 行が生成されます。次のホスト (**右**) でも同じ操作を行います。

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

3. **/etc/ipsec.d/** ディレクトリーで、新しい **my_host-to-host.conf** ファイルを作成します。上の手順の **ipsec showhostkey** コマンドの出力から、RSA ホストの鍵を新規ファイルに書き込みます。以下に例を示します。

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

4. 鍵をインポートしたら、**ipsec** サービスを再起動します。

```
# systemctl restart ipsec
```

5. 接続を読み込みます。

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

6. トンネルを確立します。

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

7. **ipsec** サービスの開始時に自動的にトンネルを開始するには、以下の行を接続定義に追加します。

```
auto=start
```

5.5. サイト間 VPN の設定

2 つのネットワークを結合してサイト間の IPsec VPN を作成する場合は、その 2 つのホスト間の IPsec トンネルを作成します。これにより、ホストは終了点として動作し、1 つまたは複数のサブネットからのトラフィックが通過できるように設定されます。したがって、ホストを、ネットワークのリモート部分にゲートウェイとして見なすことができます。

サイト間の VPN の設定は、設定ファイル内で複数のネットワークまたはサブネットを指定する必要がある点のみが、ホスト間の VPN とは異なります。

前提条件

- ホスト間の VPN が設定されている。

手順

1. ホスト間の VPN の設定が含まれるファイルを、新規ファイルにコピーします。以下に例を示します。

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

2. 上の手順で作成したファイルに、サブネット設定を追加します。以下に例を示します。

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

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

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

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

5.6. リモートアクセスの VPN の設定

ロードウォーリアーとは、モバイルクライアントと動的に割り当てられた IP アドレスを使用する移動するユーザーのことです。モバイルクライアントは、X.509 証明書を使用して認証します。

以下の例では、**IKEv2** の設定を示しています。**IKEv1** XAUTH プロトコルは使用していません。

サーバー上では以下の設定になります。

```
conn roadwarriors
   ikev2=insist
   # support (roaming) MOBIKE clients (RFC 4555)
   mobike=yes
   fragmentation=yes
   left=1.2.3.4
   # if access to the LAN is given, enable this, otherwise use 0.0.0.0/0
   # leftsubnet=10.10.0.0/16
   leftsubnet=0.0.0.0/0
   leftcert=gw.example.com
   leftid=%fromcert
   leftxauthserver=yes
   leftmodecfgserver=yes
   right=%any
   # trust our own Certificate Agency
   rightca=%same
   # pick an IP address pool to assign to remote users
   # 100.64.0.0/16 prevents RFC1918 clashes when remote users are behind NAT
   rightaddresspool=100.64.13.100-100.64.13.254
   # if you want remote clients to use some local DNS zones and servers
   modecfgdns="1.2.3.4, 5.6.7.8"
   modecfgdomains="internal.company.com, corp"
```

```
    rightxauthclient=yes
    rightmodecfgclient=yes
    authby=rsasig
    # optionally, run the client X.509 ID through pam to allow or deny client
    # pam-authorize=yes
    # load connection, do not initiate
    auto=add
    # kill vanished roadwarriors
    dpddelay=1m
    dpdtimeout=5m
    dpdaction=clear
```

ロードウォーリアーのデバイスであるモバイルクライアントでは、上記の設定に多少変更を加えて使用します。

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

5.7. メッシュ VPN の設定

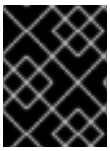
any-to-any VPN とも呼ばれるメッシュ VPN ネットワークは、全ノードが IPsec を使用して通信するネットワークです。この設定では、IPsec を使用できないノードの例外が許可されます。メッシュの VPN ネットワークは、以下のいずれかの方法で設定できます。

- IPSec を必要とする。
- IPsec を優先するが、平文通信へのフォールバックを可能にする。

ノード間の認証は、X.509 証明書または DNSSEC (DNS Security Extensions) を基にできます。

これらの接続は通常の Libreswan 設定であるため、**オポチュニスティック IPsec** に通常の IKEv2 認証方法を使用できます。ただし、**right=%opportunisticgroup** エントリーで定義されるオポチュニスティック IPsec を除きます。一般的な認証方法は、一般に共有される認証局 (CA) を使用して、X.509 証明書

に基づいてホストを相互に認証させる方法です。クラウドデプロイメントでは通常、標準の手順の一部として、クラウド内の各ノードに証明書を発行します。



重要

1 つのホストが侵害されると、グループの PSK シークレットも侵害されるため、PreSharedKey (PSK) 認証は使用しないでください。

NULL 認証を使用すると、認証なしでノード間に暗号化をデプロイできます。これを使用した場合、受動的な攻撃者からのみ保護されます。

以下の手順では、X.509 証明書を使用します。この証明書は、Dogtag Certificate System などの任意の種類の CA 管理システムを使用して生成できます。Dogtag は、各ノードの証明書が PKCS #12 形式 (**.p12** ファイル) で利用可能であることを前提としています。これには、秘密鍵、ノード証明書、およびその他のノードの X.509 証明書を検証するのに使用されるルート CA 証明書が含まれます。

各ノードでは、その X.509 証明書を除いて、同じ設定を使用します。これにより、ネットワーク内で既存ノードを再設定せずに、新規ノードを追加できます。PKCS #12 ファイルには分かりやすい名前が必要であるため、名前には node を使用します。これにより、すべてのノードに対して、この名前を参照する設定ファイルが同一になります。

前提条件

- Libreswan がインストールされ、**ipsec** サービスが各ノードで開始している。
- 新しい NSS データベースが初期化されている。

  1. すでに古い NSS データベースがある場合は、古いデータベースファイルを削除します。

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

  2. 次のコマンドを使用して、新しいデータベースを初期化できます。

     ```
     # ipsec initnss
     ```

手順

1. 各ノードで PKCS #12 ファイルをインポートします。この手順では、PKCS #12 ファイルの生成に使用するパスワードが必要になります。

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

2. **IPsec required** (private)、**IPsec optional** (private-or-clear)、および **No IPsec** (clear) プロファイルに、以下のような 3 つの接続定義を作成します。

   ```
   # cat /etc/ipsec.d/mesh.conf
   conn clear
    auto=ondemand (1)
    type=passthrough
    authby=never
    left=%defaultroute
    right=%group
   ```

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

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

(1) **auto** 変数にはいくつかのオプションがあります。

ondemand 接続オプションは、IPsec 接続を開始するオポチュニスティック IPsec や、常にアクティブにする必要のない明示的に設定した接続に使用できます。このオプションは、カーネル内にトラップ XFRM ポリシーを設定し、そのポリシーに一致する最初のパケットを受信したときに IPsec 接続を開始できるようにします。

オポチュニスティック IPsec を使用する場合も、明示的に設定した接続を使用する場合も、次のオプションを使用すると、IPsec 接続を効果的に設定および管理できます。

add オプション
: 接続設定をロードし、リモート開始に応答できるように準備します。ただし、接続はローカル側から自動的に開始されません。コマンド **ipsec auto --up** を使用して、IPsec 接続を手動で開始できます。

start オプション
: 接続設定をロードし、リモート開始に応答できるように準備します。さらに、リモートピアへの接続を即座に開始します。このオプションは、永続的かつ常にアクティブな接続に使用できます。

(2) **leftid** 変数と **rightid** 変数は、IPsec トンネル接続の右チャネルと左チャネルを指定します。これらの変数を使用して、ローカル IP アドレスの値、またはローカル証明書のサブジェクト DN を取得できます (設定している場合)。

(3) **leftcert** 変数は、使用する NSS データベースのニックネームを定義します。

3. ネットワークの IP アドレスを対応するカテゴリーに追加します。たとえば、すべてのノードが **10.15.0.0/16** ネットワーク内に存在し、すべてのノードで IPsec 暗号化を使用する必要がある場合は、次のコマンドを実行します。

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

4. 特定のノード (**10.15.34.0/24** など) を IPsec の有無にかかわらず機能させるには、そのノードを private-or-clear グループに追加します。

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

5. ホストを、**10.15.1.2** など、IPsec の機能がない clear グループに定義する場合は、次のコマンドを実行します。

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

/etc/ipsec.d/policies ディレクトリーのファイルは、各新規ノードのテンプレートから作成することも、Puppet または Ansible を使用してプロビジョニングすることもできます。

すべてのノードでは、例外のリストが同じか、異なるトラフィックフローが期待される点に注意してください。したがって、あるノードで IPsec が必要になり、別のノードで IPsec を使用できないために、ノード間の通信ができない場合もあります。

6. ノードを再起動して、設定したメッシュに追加します。

```
# systemctl restart ipsec
```

検証

1. **ping** コマンドを使用して IPsec トンネルを開きます。

```
# ping <nodeYYY>
```

2. インポートされた証明書を含む NSS データベースを表示します。

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

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

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

3. ノード上の開いているトンネルを確認します。

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

関連情報

- **ipsec.conf(5)** man ページ。
- **authby** 変数の詳細は、6.2.Libreswan の認証方法 を参照してください。

5.8. FIPS 準拠の IPSEC VPN のデプロイ

Libreswan を使用して、FIPS 準拠の IPsec VPN ソリューションをデプロイできます。これを行うには、FIPS モードの Libreswan で使用できる暗号化アルゴリズムと無効になっている暗号化アルゴリズムを特定します。

前提条件

- **AppStream** リポジトリーが有効になっている。

手順

1. **libreswan** パッケージをインストールします。

```
# dnf install libreswan
```

2. Libreswan を再インストールする場合は、古い NSS データベースを削除します。

```
# systemctl stop ipsec

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

3. **ipsec** サービスを開始して有効にし、システムの起動時にサービスを自動的に開始できるようにします。

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

4. ファイアウォールで、**ipsec** サービスを追加して、IKE プロトコル、ESP プロトコル、および AH プロトコルの **500** および **4500** UDP ポートを許可するように設定します。

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

5. システムを FIPS モードに切り替えます。

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

6. システムを再起動して、カーネルを FIPS モードに切り替えます。

```
# reboot
```

検証

1. Libreswan が FIPS モードで実行されていることを確認します。

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

2. または、**systemd** ジャーナルで **ipsec** ユニットのエントリーを確認します。

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

3. FIPS モードで使用可能なアルゴリズムを表示するには、次のコマンドを実行します。

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

4. FIPS モードで無効化されたアルゴリズムをクエリーするには、次のコマンドを実行します。

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

5. FIPS モードで許可されているすべてのアルゴリズムと暗号のリストを表示するには、次のコマンドを実行します。

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

```
NSS(MODP) dh16
NSS(MODP) dh17
NSS(MODP) dh18
NSS(ECP)  ecp_256, ecp256
NSS(ECP)  ecp_384, ecp384
NSS(ECP)  ecp_521, ecp521
```

関連情報

- Using system-wide cryptographic policies.

5.9. パスワードによる IPSEC NSS データベースの保護

デフォルトでは、IPsec サービスは、初回起動時に空のパスワードを使用して Network Security Services (NSS) データベースを作成します。セキュリティーを強化するために、パスワード保護を追加できます。

前提条件

- **/var/lib/ipsec/nss/** ディレクトリーには NSS データベースファイルが含まれます。

手順

1. Libreswan の **NSS** データベースのパスワード保護を有効にします。

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

Enter new password:
```

2. 前の手順で設定したパスワードを含む **/etc/ipsec.d/nsspassword** ファイルを作成します。次に例を示します。

```
# cat /etc/ipsec.d/nsspassword
NSS Certificate DB:_<password>_
```

nsspassword ファイルは次の構文を使用します。

```
<token_1>:<password1>
<token_2>:<password2>
```

デフォルトの NSS ソフトウェアトークンは **NSS Certificate DB** です。システムが FIPS モードで実行し場合は、トークンの名前が **NSS FIPS 140-2 Certificate DB** になります。

3. 選択したシナリオに応じて、**nsspassword** ファイルの完了後に **ipsec** サービスを起動または再起動します。

```
# systemctl restart ipsec
```

検証

1. NSS データベースに空でないパスワードを追加した後に、**ipsec** サービスが実行中であることを確認します。

```
# 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)...
```

検証

- **Journal** ログに初期化が成功したことを確認するエントリーが含まれていることを確認します。

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

関連情報

- **certutil(1)** man ページ
- ナレッジベース記事 Compliance Activities and Government Standards の FIPS 140-2 および FIPS 140-3

5.10. TCP を使用するように IPSEC VPN を設定

Libreswan は、RFC 8229 で説明されているように、IKE パケットおよび IPsec パケットの TCP カプセル化に対応します。この機能により、UDP 経由でトラフィックが転送されないように、IPsec VPN をネットワークに確立し、セキュリティーのペイロード (ESP) を強化できます。フォールバックまたはメインの VPN トランスポートプロトコルとして TCP を使用するように VPN サーバーおよびクライアントを設定できます。TCP カプセル化にはパフォーマンスコストが大きくなるため、UDP がシナリオで永続的にブロックされている場合に限り、TCP を主な VPN プロトコルとして使用してください。

前提条件

- リモートアクセス VPN が設定されている。

手順

1. **config setup** セクションの **/etc/ipsec.conf** ファイルに以下のオプションを追加します。

```
listen-tcp=yes
```

2. UDP で最初の試行に失敗した場合に TCP カプセル化をフォールバックオプションとして使用するには、クライアントの接続定義に以下の 2 つのオプションを追加します。

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

または、UDP を永続的にブロックしている場合は、クライアントの接続設定で以下のオプションを使用します。

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

関連情報

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

5.11. IPSEC 接続を高速化するために、ESP ハードウェアオフロードの自動検出と使用を設定

Encapsulating Security Payload (ESP) をハードウェアにオフロードすると、Ethernet で IPsec 接続が加速します。デフォルトでは、Libreswan は、ハードウェアがこの機能に対応しているかどうかを検出するため、ESP ハードウェアのオフロードを有効にします。機能が無効になっているか、明示的に有効になっている場合は、自動検出に戻すことができます。

前提条件

- ネットワークカードは、ESP ハードウェアオフロードに対応します。
- ネットワークドライバーは、ESP ハードウェアのオフロードに対応します。
- IPsec 接続が設定され、動作する。

手順

1. ESP ハードウェアオフロードサポートの自動検出を使用する接続の **/etc/ipsec.d/** ディレクトリーにある Libreswan 設定ファイルを編集します。
2. 接続の設定で **nic-offload** パラメーターが設定されていないことを確認します。
3. **nic-offload** を削除した場合は、**ipsec** を再起動します。

```
# systemctl restart ipsec
```

検証

1. IPsec 接続が使用するイーサネットデバイスの **tx_ipsec** および **rx_ipsec** カウンターを表示します。

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

2. IPsec トンネルを介してトラフィックを送信します。たとえば、リモート IP アドレスに ping します。

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

3. イーサネットデバイスの **tx_ipsec** および **rx_ipsec** カウンターを再度表示します。

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

カウンターの値が増えると、ESP ハードウェアオフロードが動作します。

関連情報

- IPsec を使用した VPN の設定

5.12. IPSEC 接続を加速化するためにボンディングでの ESP ハードウェアオフロードの設定

Encapsulating Security Payload (ESP) をハードウェアにオフロードすると、IPsec 接続が加速します。フェイルオーバーの理由でネットワークボンディングを使用する場合、ESP ハードウェアオフロードを設定する要件と手順は、通常のイーサーネットデバイスを使用する要件と手順とは異なります。たとえば、このシナリオでは、ボンディングでオフロードサポートを有効にし、カーネルはボンディングのポートに設定を適用します。

前提条件

- ボンディングのすべてのネットワークカードが、ESP ハードウェアオフロードをサポートしている。
- ネットワークドライバーが、ボンドデバイスで ESP ハードウェアオフロードに対応している。RHEL では、**ixgbe** ドライバーのみがこの機能をサポートします。
- ボンディングが設定されており動作する。
- ボンディングで **active-backup** モードを使用している。ボンディングドライバーは、この機能の他のモードはサポートしていません。
- IPsec 接続が設定され、動作する。

手順

1. ネットワークボンディングで ESP ハードウェアオフロードのサポートを有効にします。

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

このコマンドにより、**bond0** 接続での ESP ハードウェアオフロードのサポートが有効になります。

2. **bond0** 接続を再度アクティブにします。

```
# nmcli connection up bond0
```

3. ESP ハードウェアオフロードに使用すべき接続の **/etc/ipsec.d/** ディレクトリーにある Libreswan 設定ファイルを編集し、**nic-offload=yes** ステートメントを接続エントリーに追加します。

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

4. **ipsec** サービスを再起動します。

```
# systemctl restart ipsec
```

検証

1. ボンディングのアクティブなポートを表示します。

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

2. アクティブなポートの **tx_ipsec** カウンターおよび **rx_ipsec** カウンターを表示します。

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

3. IPsec トンネルを介してトラフィックを送信します。たとえば、リモート IP アドレスに ping します。

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

4. アクティブなポートの **tx_ipsec** カウンターおよび **rx_ipsec** カウンターを再度表示します。

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

カウンターの値が増えると、ESP ハードウェアオフロードが動作します。

関連情報

- ネットワークボンディングの設定
- IPsec を使用した VPN の設定

5.13. RHEL システムロールを使用した IPSEC による VPN 接続の設定

vpn システムロールを使用すると、Red Hat Ansible Automation Platform を使用して RHEL システムで VPN 接続を設定できます。これを使用して、ホスト間、ネットワーク間、VPN リモートアクセスサーバー、およびメッシュ設定をセットアップできます。

ホスト間接続の場合、ロールは、必要に応じてキーを生成するなど、デフォルトのパラメーターを使用して、**vpn_connections** のリスト内のホストの各ペア間に VPN トンネルを設定します。または、リストされているすべてのホスト間にオポチュニスティックメッシュ設定を作成するように設定することもできます。このロールは、**hosts** の下にあるホストの名前が Ansible インベントリーで使用されているホストの名前と同じであり、それらの名前を使用してトンネルを設定できることを前提としています。

注記

vpn RHEL システムロールは、現在 VPN プロバイダーとして、IPsec 実装である Libreswan のみをサポートしています。

5.13.1. vpn RHEL システムロールを使用して IPsec によるホスト間 VPN を作成する

vpn システムロールを使用して、コントロールノードで Ansible Playbook を実行することにより、ホスト間接続を設定できます。これにより、インベントリーファイルにリストされているすべての管理対象ノードが設定されます。

前提条件

- コントロールノードと管理対象ノードの準備が完了している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

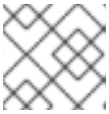
```
- name: Host to host VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - hosts:
          managed-node-01.example.com:
          managed-node-02.example.com:
    vpn_manage_firewall: true
    vpn_manage_selinux: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

この Playbook は、システムロールによって自動生成されたキーを使用した事前共有キー認証を使用して、接続 **managed-node-01.example.com-to-managed-node-02.example.com** を設定します。**vpn_manage_firewall** と **vpn_manage_selinux** は両方とも **true** に設定されているため、**vpn** ロールは **firewall** ロールと **selinux** ロールを使用して、**vpn** ロールが使用するポートを管理します。

管理対象ホストから、インベントリーファイルにリストされていない外部ホストへの接続を設定するには、ホストの **vpn_connections** リストに次のセクションを追加します。

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

これにより、追加の接続 **managed-node-01.example.com-to-<external_node>** が 1 つ設定されます。



注記

接続は管理対象ノードでのみ設定され、外部ノードでは設定されません。

2. オプション: **vpn_connections** 内の追加セクション (コントロールプレーンやデータプレーンなど) を使用して、マネージドノードに複数の VPN 接続を指定できます。

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

3. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

4. Playbook を実行します。

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

検証

1. マネージドノードで、接続が正常にロードされていることを確認します。

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

<connection_name> は、このノードからの接続の名前に置き換えます (例: **managed_node1-to-managed_node2**)。

注記

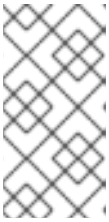
デフォルトでは、ロールは、各システムの観点から作成する接続ごとにわかりやすい名前を生成します。たとえば、**managed_node1** と **managed_node2** との間の接続を作成するときに、**managed_node1** 上のこの接続のわかりやすい名前は **managed_node1-to-managed_node2** ですが、**managed_node2** では、この接続の名前は **managed_node2-to-managed_node1** となります。

2. マネージドノードで、接続が正常に開始されたことを確認します。

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

3. オプション: 接続が正常にロードされない場合は、次のコマンドを入力して手動で接続を追加します。これにより、接続の確立に失敗した理由を示す、より具体的な情報が提供されます。

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



注記

接続のロードおよび開始のプロセスで発生する可能性のあるエラーは、**/var/log/pluto.log** ファイルに報告されます。これらのログは解析が難しいため、代わりに接続を手動で追加して、標準出力からログメッセージを取得してください。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/vpn/** ディレクトリー

5.13.2. vpn RHEL システムロールを使用して IPsec によるオポチュニスティックメッシュ VPN 接続を作成する

vpn システムロールを使用して、コントロールノードで Ansible Playbook を実行することにより、認証に証明書を使用するオポチュニスティックメッシュ VPN 接続を設定できます。これにより、インベントリーファイルにリストされているすべての管理対象ノードが設定されます。

前提条件

- コントロールノードと管理対象ノードの準備が完了している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- **/etc/ipsec.d/** ディレクトリーの IPsec ネットワークセキュリティーサービス (NSS) 暗号ライブラリーに、必要な証明書が含まれている。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

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

```
        cidr: 198.51.100.0/24
      - policy: private
        cidr: 192.0.2.0/24
      - policy: clear
        cidr: 192.0.2.7/32
  vpn_manage_firewall: true
  vpn_manage_selinux: true
```

証明書による認証は、Playbook で **auth_method: cert** パラメーターを定義することによって設定されます。デフォルトでは、ノード名が証明書のニックネームとして使用されます。この例では、**managed-node-01.example.com** です。インベントリーで **cert_name** 属性を使用して、さまざまな証明書名を定義できます。

この例の手順では、Ansible Playbook の実行元のシステムであるコントロールノードが、両方の管理対象ノードと同じ Classless Inter-Domain Routing (CIDR) 番号 (192.0.2.0/24) を共有し、IP アドレス 192.0.2.7 を持ちます。したがって、コントロールノードは、CIDR 192.0.2.0/24 用に自動的に作成されるプライベートポリシーに該当します。

再生中の SSH 接続の損失を防ぐために、コントロールノードの明確なポリシーがポリシーのリストに含まれています。ポリシーリストには、CIDR がデフォルトと等しい項目もあることに注意してください。これは、この Playbook がデフォルトポリシーのルールを上書きして、private-or-clear ではなく private にするためです。

vpn_manage_firewall と **vpn_manage_selinux** は両方とも **true** に設定されているため、**vpn** ロールは **firewall** ロールと **selinux** ロールを使用して、**vpn** ロールが使用するポートを管理します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/vpn/** ディレクトリー

5.14. システム全体の暗号化ポリシーをオプトアウトする IPSEC 接続の設定

接続向けのシステム全体の暗号化ポリシーのオーバーライド

RHEL のシステム全体の暗号化ポリシーでは、**%default** と呼ばれる特別な接続が作成されます。この接続には、**ikev2** オプション、**esp** オプション、および **ike** オプションのデフォルト値が含まれます。ただし、接続設定ファイルに上記のオプションを指定すると、デフォルト値を上書きできます。

たとえば、次の設定では、AES および SHA-1 または SHA-2 で IKEv1 を使用し、AES-GCM または AES-CBC で IPsec (ESP) を使用する接続が可能です。

-

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

AES-GCM は IPsec (ESP) および IKEv2 で利用できますが、IKEv1 では利用できません。

全接続向けのシステム全体の暗号化ポリシーの無効化

すべての IPsec 接続のシステム全体の暗号化ポリシーを無効にするには、**/etc/ipsec.conf** ファイルで次の行をコメントアウトします。

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

次に、接続設定ファイルに **ikev2=never** オプションを追加してください。

関連情報

- Using system-wide cryptographic policies.

5.15. IPSEC VPN 設定のトラブルシューティング

IPsec VPN 設定に関連する問題は主に、一般的な理由が原因で発生する可能性が高くなっています。このような問題が発生した場合は、問題の原因が以下のシナリオのいずれかに該当するかを確認して、対応するソリューションを適用します。

基本的な接続のトラブルシューティング

VPN 接続関連の問題の多くは、管理者が不適当な設定オプションを指定してエンドポイントを設定した新しいデプロイメントで発生します。また、互換性のない値が新たに実装された場合に、機能していた設定が突然動作が停止する可能性があります。管理者が設定を変更した場合など、このような結果になることがあります。また、管理者が暗号化アルゴリズムなど、特定のオプションに異なるデフォルト値を使用して、ファームウェアまたはパッケージの更新をインストールした場合などです。

IPsec VPN 接続が確立されていることを確認するには、次のコマンドを実行します。

```
# 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
```

出力が空の場合や、エントリーで接続名が表示されない場合など、トンネルが破損します。

接続に問題があることを確認するには、以下を実行します。

1. **vpn.example.com** 接続をもう一度読み込みます。

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

2. 次に、VPN 接続を開始します。

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

ファイアウォール関連の問題

最も一般的な問題は、IPSec エンドポイントの 1 つ、またはエンドポイント間にあるルーターにあるファイアウォールで Internet Key Exchange (IKE) パケットがドロップされるという点が挙げられます。

- IKEv2 の場合には、以下の例のような出力は、ファイアウォールに問題があることを示しています。

```
# 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
...
```

- IKEv1 の場合は、最初のコマンドの出力は以下のようになります。

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

IPsec の設定に使用される IKE プロトコルは暗号化されているため、**tcpdump** ツールを使用して、トラブルシューティングできるサブセットは一部のみです。ファイアウォールが IKE パケットまたは IPsec パケットをドロップしている場合は、**tcpdump** ユーティリティーを使用して原因を見つけることができます。ただし、**tcpdump** は IPsec VPN 接続に関する他の問題を診断できません。

- **eth0** インターフェイスで VPN および暗号化データすべてのネゴシエーションを取得するには、次のコマンドを実行します。

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

アルゴリズム、プロトコル、およびポリシーが一致しない場合

VPN 接続では、エンドポイントが IKE アルゴリズム、IPsec アルゴリズム、および IP アドレス範囲に一致する必要があります。不一致が発生した場合には接続は失敗します。以下の方法のいずれかを使用して不一致を特定した場合は、アルゴリズム、プロトコル、またはポリシーを調整して修正します。

- リモートエンドポイントが IKE/IPsec を実行していない場合は、そのパケットを示す ICMP パケットが表示されます。以下に例を示します。

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

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

- IKE アルゴリズムが一致しない例:

```
# 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
```

- IPsec アルゴリズムが一致しない例:

```
# 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
```

また、IKE バージョンが一致しないと、リモートエンドポイントが応答なしの状態でリクエストをドロップする可能性がありました。これは、すべての IKE パケットをドロップするファイアウォールと同じです。

- IKEv2 (Traffic Selectors - TS) の IP アドレス範囲が一致しない例:

```
# 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
```

- IKEv1 の IP アドレス範囲で一致しない例:

```
# 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
```

- IKEv1 で PreSharedKeys (PSK) を使用する場合には、どちらでも同じ PSK に配置されなければ、IKE メッセージ全体の読み込みができなくなります。

```
# 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
```

- IKEv2 では、 mismatched-PSK エラーが原因で AUTHENTICATION_FAILED メッセージが表示されます。

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

最大伝送単位 (MTU)

ファイアウォールが IKE または IPSec パケットをブロックする以外で、ネットワークの問題の原因として、暗号化パケットのパケットサイズの増加が最も一般的です。ネットワークハードウェアは、最大伝送単位 (MTU) を超えるパケットを 1500 バイトなどのサイズに断片化します。多くの場合、断片化されたパケットは失われ、パケットの再アセンブルに失敗します。これにより、小さいサイズのパケットを使用する ping テスト時には機能し、他のトラフィックでは失敗するなど、断続的な問題が発生します。このような場合に、SSH セッションを確立できますが、リモートホストに 'ls -al /usr' コマンドに入力した場合など、すぐにターミナルがフリーズします。

この問題を回避するには、トンネル設定ファイルに **mtu=1400** のオプションを追加して、MTU サイズを縮小します。

または、TCP 接続の場合は、MSS 値を変更する iptables ルールを有効にします。

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

各シナリオで上記のコマンドを使用して問題が解決されない場合は、**set-mss** パラメーターで直接サイズを指定します。

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

ネットワークアドレス変換 (NAT)

IPsec ホストが NAT ルーターとしても機能すると、誤ってパケットが再マッピングされる可能性があります。以下の設定例はこの問題について示しています。

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

アドレスが 172.16.0.1 のシステムには NAT ルールが 1 つあります。

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

アドレスが 10.0.2.33 のシステムがパケットを 192.168.0.1 に送信する場合に、ルーターは IPsec 暗号化を適用する前にソースを 10.0.2.33 から 172.16.0.1 に変換します。

次に、ソースアドレスが 10.0.2.33 のパケットは **conn myvpn** 設定と一致しなくなるので、IPsec ではこのパケットが暗号化されません。

この問題を解決するには、ルーターのターゲット IPsec サブネット範囲の NAT を除外するルールを挿入します。以下に例を示します。

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

カーネル IPsec サブシステムのバグ

たとえば、バグが原因で IKE ユーザー空間と IPsec カーネルの同期が解除される場合など、カーネル IPsec サブシステムに問題が発生する可能性があります。このような問題がないかを確認するには、以下を実行します。

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

上記のコマンドの出力でゼロ以外の値が表示されると、問題があることを示しています。この問題が発生した場合は、新しい サポートケース を作成し、1 つ前のコマンドの出力と対応する IKE ログを添付してください。

Libreswan のログ

デフォルトでは、Libreswan は **syslog** プロトコルを使用してログに記録します。**journalctl** コマンドを使用して、IPsec に関連するログエントリーを検索できます。ログへの対応するエントリーは **pluto** IKE デーモンにより送信されるため、以下のように、キーワード pluto を検索します。

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

ipsec サービスのライブログを表示するには、次のコマンドを実行します。

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

ロギングのデフォルトレベルで設定問題が解決しない場合は、**/etc/ipsec.conf** ファイルの **config setup** セクションに **plutodebug=all** オプションを追加してデバッグログを有効にします。

デバッグロギングは多くのエントリーを生成し、**journald** サービスまたは **syslogd** サービスレートのいずれかが **syslog** メッセージを制限する可能性があることに注意してください。完全なログを取得するには、ロギングをファイルにリダイレクトします。**/etc/ipsec.conf** を編集し、**config setup** セクションに **logfile=/var/log/pluto.log** を追加します。

関連情報

- ログファイルを使用した問題のトラブルシューティング
- **tcpdump(8)** および **ipsec.conf(5)** の man ページ
- firewalld の使用および設定

5.16. 関連情報

- **ipsec(8)**、**ipsec.conf(5)**、**ipsec.secrets(5)**、**ipsec_auto(8)**、および **ipsec_rsasigkey(8)** の man ページ
- **/usr/share/doc/libreswan-version/** ディレクトリー
- The Libreswan プロジェクトの Wiki
- All Libreswan のすべての man ページ
- NIST Special Publication 800-77:Guide to IPsec VPNs

第6章 ネットワークサービスのセキュリティー保護

Red Hat Enterprise Linux 9 は、さまざまな種類のネットワークサーバーをサポートしています。RHEL 9 ネットワークサービスを使用すると、システムのセキュリティーが DoS 攻撃 (Denial of Service)、DDoS 攻撃 (Distributed Denial of Service)、スクリプト脆弱性攻撃、バッファーオーバーフロー攻撃など、さまざまな種類の攻撃のリスクにさらされる可能性があります。

攻撃に対するシステムのセキュリティーを強化するには、使用しているアクティブなネットワークサービスを監視することが重要です。たとえば、ネットワークサービスがマシンで実行されている場合に、そのデーモンはネットワークポートでの接続をリッスンするのでセキュリティーが低下する可能性があります。ネットワークに対する攻撃に対する公開を制限するには、未使用のすべてのサービスをオフにする必要があります。

6.1. RPCBIND サービスのセキュリティー保護

rpcbind サービスは、Network Information Service (NIS) や Network File System (NFS) などの Remote Procedure Calls (RPC) サービス用の動的ポート割り当てデーモンです。その認証メカニズムは弱く、制御するサービスに幅広いポート範囲を割り当てる可能性があるため、**rpcbind** をセキュア化することが重要です。

すべてのネットワークへのアクセスを制限し、サーバーのファイアウォールルールを使用して特定の例外を定義することにより、**rpcbind** のセキュリティーを確保できます。

注記

- **NFSv3** サーバーでは、**rpcbind** サービスが必要です。
- **NFSv4** では **rpcbind** サービスは必要ありません。

前提条件

- **rpcbind** パッケージがインストールされている。
- **Firewalld** パッケージがインストールされ、サービスが実行されている。

手順

1. 次に、ファイアウォールルールを追加します。
   - TCP 接続を制限し、**111** ポート経由の **192.168.0.0/24** ホストからのパッケージだけを受け入れます。

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

   - TCP 接続を制限し、**111** ポート経由のローカルホストからのパッケージだけを受け入れます。

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

   - UDP 接続を制限し、**111** ポート経由の **192.168.0.0/24** ホストからのパッケージだけを受け入れます。

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

ファイアウォール設定を永続化するには、ファイアウォールルールを追加するときに **--permanent** オプションを使用します。

2. ファイアウォールをリロードして、新しいルールを適用します。

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

検証

- ファイアウォールルールをリストします。

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

関連情報

- **NFSv4-only** サーバーの詳細は、Configuring an NFSv4-only server を参照してください。
- firewalld の使用および設定

6.2. RPC.MOUNTD サービスのセキュリティー保護

rpc.mountd デーモンは、NFS マウントプロトコルのサーバー側を実装します。NFS マウントプロトコルは、NFS バージョン 3 (RFC 1813) で使用されます。

rpc.mountd サービスは、サーバーにファイアウォールルールを追加することでセキュリティー保護できます。すべてのネットワークへのアクセスを制限し、ファイアウォールルールを使用して特定の例外を定義できます。

前提条件

- **rpc.mountd** パッケージがインストールされている。
- **Firewalld** パッケージがインストールされ、サービスが実行されている。

手順

1. 以下のように、サーバーにファイアウォールルールを追加します。
   - **192.168.0.0/24** ホストからの **mountd** 接続を許可します。

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

   - ローカルホストからの **mountd** 接続を受け入れます。

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

ファイアウォール設定を永続化するには、ファイアウォールルールを追加するときに **--permanent** オプションを使用します。

2. ファイアウォールをリロードして、新しいルールを適用します。

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

検証

- ファイアウォールルールをリストします。

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

関連情報

- firewalld の使用および設定

6.3. NFS サービスの保護

Kerberos を使用してすべてのファイルシステム操作を認証および暗号化して、ネットワークファイルシステムバージョン 4 (NFSv4) のセキュリティーを保護できます。ネットワークアドレス変換 (NAT) またはファイアウォールで NFSv4 を使用する場合に、**/etc/default/nfs** ファイルを変更することで委譲をオフにできます。委譲は、サーバーがファイルの管理をクライアントに委譲する手法です。

対照的に、NFSv3 ではファイルのロックとマウントに Kerberos は使用されません。

NFS サービスは、すべてのバージョンの NFS で TCP を使用してトラフィックを送信します。このサービスは、**RPCSEC_GSS** カーネルモジュールの一部として Kerberos ユーザーおよびグループ認証をサポートします。

NFS を利用すると、リモートのホストがネットワーク経由でファイルシステムをマウントし、そのファイルシステムを、ローカルにマウントしているファイルシステムのように操作できるようになります。集約サーバーのリソースを統合して、ファイルシステムを共有するときに **/etc/nfsmount.conf** ファイルの NFS マウントオプションをさらにカスタマイズできます。

6.3.1. NFS サーバーのセキュリティーを保護するエクスポートオプション

NFS サーバーは、**/etc/exports** ファイル内のどのファイルシステムにどのファイルシステムをエクスポートするかなど、ディレクトリーとホストのリスト構造を決定します。

警告

/etc/exports ファイルの構文に余分なスペースがあると、設定が大幅に変更される可能性があります。

以下の例では、**/tmp/nfs/** ディレクトリーは **bob.example.com** ホストと共有され、読み取りおよび書き込みのパーミッションを持ちます。

```
/tmp/nfs/     bob.example.com(rw)
```

以下の例は上記と同じになりますが、同じディレクトリーを読み取り専用パーミッションで **bob.example.com** ホストに共有し、ホスト名の後の 1 つのスペース文字が原因で読み取りと書き込み権限で **すべてのユーザー** に共有します。

```
/tmp/nfs/     bob.example.com (rw)
```

showmount -e <hostname> コマンドを入力すると、システム上の共有ディレクトリーを確認できます。

/etc/exports ファイルでは次のエクスポートオプションを使用できます。

警告

ファイルシステムのサブディレクトリーをエクスポートするのはセキュアではないため、ファイルシステム全体をエクスポートしてください。攻撃者が、部分的にエクスポートされたファイルシステムのエクスポートされていない部分にアクセスする可能性があります。

ro

NFS ボリュームを読み取り専用としてエクスポートします。

rw

NFS ボリュームに対する読み取りおよび書き込み要求を許可します。書き込みアクセスが許可されると攻撃のリスクが高まるため、このオプションは注意して使用してください。**rw** オプションを使用してディレクトリーをマウントする必要がある場合は、起こりうるリスクを軽減するために、すべてのユーザーがディレクトリーに書き込み可能にしないようにしてください。

root_squash

uid/**gid** 0 からの要求を匿名の **uid**/**gid** にマップします。これは、**bin** ユーザーや **staff** グループなど、同様に機密である可能性の高い他の **uid** または **gid** には適用されません。

no_root_squash

root squash をオフにします。デフォルトでは、NFS 共有は **root** ユーザーを、非特権ユーザーである **nobody** ユーザーに変更します。これにより、**root** が作成したすべてのファイルの所有者が **nobody** に変更され、**setuid** ビットが設定されたプログラムのアップロードができなくなりま

す。**no_root_squash** オプションを使用すると、リモートの root ユーザーは共有ファイルシステムの任意のファイルを変更し、他のユーザーに対してアプリケーションが Trojans に感染した状態のままにします。

secure

予約ポートへのエクスポートを制限します。デフォルトでは、サーバーは予約済みポートからのクライアント通信のみを許可します。ただし、多くのネットワークで、クライアント上で **root** ユーザーになるのは簡単です。そのため、サーバーで予約されたポートからの通信が特権であると仮定することは安全ではありません。そのため、予約ポートの制限は効果が限定的です。Kerberos、ファイアウォール、および特定クライアントへのエクスポートを制限することに依存すると良いでしょう。

また、NFS サーバーをエクスポートする際に、以下のベストプラクティスを考慮してください。

- 一部のアプリケーションでは、パスワードをプレーンテキストまたは弱い暗号化形式で保存するため、ホームディレクトリーをエクスポートすることはリスクがあります。アプリケーションコードを確認して改善することで、リスクを軽減できます。
- 一部のユーザーは SSH キーにパスワードを設定していないため、この場合もホームディレクトリーによるリスクが発生します。パスワードの使用を強制するか、Kerberos を使用することで、これらのリスクを軽減できます。
- NFS エクスポートを必要なクライアントのみに制限します。NFS サーバーで **showmount -e** コマンドを使用して、サーバーのエクスポート内容を確認します。特に必要のないものはエクスポートしないでください。
- 攻撃のリスクを減らすために、不要なユーザーがサーバーにログインできないようにしてください。サーバーにアクセスできるユーザーを定期的に確認してください。

関連情報

- Red Hat Identity Management 使用時における IdM での自動マウントの使用
- **exports(5)** および **nfs(5)** の man ページ

6.3.2. NFS クライアントのセキュリティーを保護するマウントオプション

mount コマンドに次のオプションを渡すと、NFS ベースのクライアントのセキュリティーを強化できます。

nosuid

nosuid オプションを使用して **set-user-identifier** または **set-group-identifier** ビットを無効にします。これにより、リモートユーザーが **setuid** プログラムを実行してより高い特権を取得するのを防ぎ、**setuid** オプションの反対となるこのオプションを使用できます。

noexec

noexec オプションを使用して、クライアント上の実行可能なファイルをすべて無効にします。これを使用して、ユーザーが共有ファイルシステムに配置されたファイルを誤って実行するのを防ぎます。

nodev

nodev オプションを使用して、クライアントがデバイスファイルをハードウェアデバイスとして処理するのを防ぎます。

resvport

resvport オプションを使用して、通信を予約済みポートに制限し、特権送信元ポートを使用してサーバーと通信できます。予約済みポートは、**root** ユーザーなどの特権ユーザーおよびプロセス用に予約されています。

秒

NFS サーバーの **sec** オプションを使用して、マウントポイント上のファイルにアクセスするための RPCGSS セキュリティーフレーバーを選択します。有効なセキュリティーフレーバーは、**none**、**sys**、**krb5**、**krb5i**、および **krb5p** です。

重要

krb5-libs パッケージが提供する MIT Kerberos ライブラリーは、新しいデプロイメントで Data Encryption Standard (DES) アルゴリズムに対応しなくなりました。DES は、セキュリティーと互換性の理由から、Kerberos ライブラリーでは非推奨であり、デフォルトで無効になっています。互換性の理由でご使用の環境で DES が必要な場合を除き、DES の代わりに新しくより安全なアルゴリズムを使用してください。

関連情報

- よく使用される NFS マウントオプション

6.3.3. ファイアウォールでの NFS のセキュリティー保護

NFS サーバーでファイアウォールを保護するには、必要なポートのみを開いてください。他のサービスには NFS 接続ポート番号を使用しないでください。

前提条件

- **nfs-utils** パッケージがインストールされている。
- **Firewalld** パッケージがインストールされ、実行されている。

手順

- NFSv4 では、ファイアウォールは TCP ポート **2049** を開く必要があります。
- NFSv3 では、**2049** で 4 つのポートを追加で開きます。

1. **rpcbind** サービスは NFS ポートを動的に割り当て、ファイアウォールルールの作成時に問題が発生する可能性があります。このプロセスを簡素化するには、**/etc/nfs.conf** ファイルを使用して、使用するポートを指定します。

   a. **[mountd]** セクションの **mountd** (**rpc.mountd**) の TCP および UDP ポートを **port=<value>** 形式で設定します。

   b. **[statd]** セクションの **statd** (**rpc.statd**) の TCP および UDP ポートを **port=<value>** 形式で設定します。

2. **/etc/nfs.conf** ファイルで NFS ロックマネージャー (**nlockmgr**) の TCP および UDP ポートを設定します。

   a. **[lockd]** セクションの **nlockmgr** (**rpc.statd**) の TCP ポートを **port=value** 形式で設定します。または、**/etc/modprobe.d/lockd.conf** ファイルの **nlm_tcpport** オプションを使用することもできます。

b. **[lockd]** セクションの **nlockmgr** (**rpc.statd**) の UDP ポートを **udp-port=value** 形式で設定します。または、**/etc/modprobe.d/lockd.conf** ファイルの **nlm_udpport** オプションを使用することもできます。

検証

- NFS サーバー上のアクティブなポートと RPC プログラムをリスト表示します。

```
$ rpcinfo -p
```

関連情報

- Red Hat Identity Management 使用時における IdM での自動マウントの使用
- **exports(5)** および **nfs(5)** の man ページ

6.4. FTP サービスのセキュリティー保護

ファイル転送プロトコル (FTP) を使用して、ネットワーク経由でファイルを転送できます。ユーザー認証を含むサーバーとの FTP トランザクションは、すべて暗号化されるわけではないため、サーバーがセキュアに設定されていることを確認してください。

RHEL 9 は、2 つの FTP サーバーを提供します。

Red Hat Content Accelerator (tux)
: FTP 機能を備えたカーネル空間 Web サーバー。

Very Secure FTP Daemon (vsftpd)
: セキュリティーを重視した、FTP サービスのスタンドアロンの実装。

vsftpd FTP サービスをセットアップするためのセキュリティーガイドラインを以下に示します。

6.4.1. FTP グリーティングバナーのセキュリティー保護

ユーザーが FTP サービスに接続すると、FTP はグリーティングバナーを表示します。このバナーには、デフォルトでバージョン情報が含まれています。攻撃者がこの情報を利用してシステムの弱点を特定する可能性があります。デフォルトのバナーを変更することで、この情報を非表示にすることができます。

/etc/banners/ftp.msg ファイルを編集して、単一行のメッセージを直接含めるか、複数行のメッセージを含めることができる別のファイルを参照して、カスタムバナーを定義できます。

手順

- 1 行のメッセージを定義するには、次のオプションを **/etc/vsftpd/vsftpd.conf** ファイルに追加します。

```
ftpd_banner=Hello, all activity on ftp.example.com is logged.
```

- 別のファイルでメッセージを定義するには以下を実行します。
  - バナーメッセージを含む **.msg** ファイルを作成します。(例: **/etc/vendors/ftp .msg**)

```
######### Hello, all activity on ftp.example.com is logged. #########
```

複数のバナーの管理を簡素化するには、すべてのバナーを **/etc/vendors/** ディレクトリーに配置します。

- バナーファイルへのパスを **/etc/vsftpd/vsftpd.conf** ファイルの **banner_file** オプションに追加します。

```
banner_file=/etc/banners/ftp.msg
```

検証

- 変更されたバナーを表示します。

```
$ ftp localhost
Trying ::1…
Connected to localhost (::1).
Hello, all activity on ftp.example.com is logged.
```

6.4.2. FTP での匿名アクセスとアップロードの防止

デフォルトでは、**vsftpd** パッケージをインストールすると、**/var/ftp/** ディレクトリーと、ディレクトリーに対する読み取り専用権限を持つ匿名ユーザー用のディレクトリーツリーが作成されます。匿名ユーザーはデータにアクセスできるため、これらのディレクトリーに機密データを保存しないでください。

システムのセキュリティーを強化するために、匿名ユーザーが特定のディレクトリーにファイルをアップロードできるが、データは読み取れないように、FTP サーバーを設定できます。次の手順では、匿名ユーザーが **root** ユーザー所有のディレクトリーにファイルをアップロードできるが変更できないようにする必要があります。

手順

- **/var/ftp/pub/** ディレクトリーに書き込み専用ディレクトリーを作成します。

```
# mkdir /var/ftp/pub/upload
# chmod 730 /var/ftp/pub/upload
# ls -ld /var/ftp/pub/upload
drwx-wx---. 2 root ftp 4096 Nov 14 22:57 /var/ftp/pub/upload
```

- **/etc/vsftpd/vsftpd.conf** ファイルに以下の行を追加します。

```
anon_upload_enable=YES
anonymous_enable=YES
```

- オプション: システムで SELinux が有効で Enforcing に設定されている場合には、SELinux ブール属性 **allow_ftpd_anon_write** および **allow_ftpd_full_access** を有効にします。

警告

匿名ユーザーによるディレクトリーの読み取りと書き込みを許可すると、盗まれたソフトウェアのリポジトリーになってしまう可能性があります。

6.4.3. FTP のユーザーアカウントのセキュリティー保護

FTP は、認証のために安全でないネットワークを介して暗号化されていないユーザー名とパスワードを送信します。システムユーザーが自分のユーザーアカウントからサーバーにアクセスできないようにして、FTP のセキュリティーを向上させることができます。

以下の手順のうち、お使いの設定に該当するものをできるだけ多く実行してください。

手順

- **/etc/vsftpd/vsftpd.conf** ファイルに次の行を追加して、**vsftpd** サーバーのすべてのユーザーアカウントを無効にします。

```
local_enable=NO
```

- **/etc/pam.d/vsftpd** PAM 設定ファイルにユーザー名を追加して、特定のアカウントまたは特定のアカウントグループ (**root** ユーザーや **sudo** 権限を持つユーザーなど) の FTP アクセスを無効にします。
- **/etc/vsftpd/ftpusers** ファイルにユーザー名を追加して、ユーザーアカウントを無効にします。

6.4.4. 関連情報

- **ftpd_selinux(8)** の man ページ

6.5. HTTP サーバーのセキュリティー保護

6.5.1. httpd.conf のセキュリティー強化

/etc/httpd/conf/httpd.conf ファイルでセキュリティーオプションを設定して、Apache HTTP のセキュリティーを強化できます。

システムで実行されているすべてのスクリプトが正しく機能することを常に確認してから、本番環境に移行してください。

root ユーザーのみが、スクリプトまたは Common Gateway Interface (CGI) を含むディレクトリーへの書き込み権限を持っていることを確認してください。ディレクトリーの所有権を、書き込み権限を持つ **root** に変更するには、次のコマンドを入力します。

```
# chown root <directory_name>
# chmod 755 <directory_name>
```

/etc/httpd/conf/httpd.conf ファイルでは、次のオプションを設定できます。

FollowSymLinks

このディレクティブはデフォルトで有効になっており、ディレクトリー内のシンボリックリンクをたどります。

Indexes

このディレクティブはデフォルトで有効になっています。訪問者がサーバー上のファイルを閲覧できないようにするには、このディレクティブを削除してください。

UserDir

このディレクティブは、システム上にユーザーアカウントが存在することを確認できるため、デフォルトでは無効になっています。**/root/** 以外のすべてのユーザーディレクトリーのユーザーディレクトリーブラウジングをアクティブにするには、**UserDir enabled** と **UserDir disabled** の root ディレクティブを使用します。無効化されたアカウントのリストにユーザーを追加するには、**UserDir disabled** 行にスペースで区切られたユーザーのリストを追加します。

ServerTokens

このディレクティブは、クライアントに送り返されるサーバー応答ヘッダーフィールドを制御します。以下のパラメーターを使用するとログの出力をカスタマイズできます。

ServerTokens Full

以下のように、Web サーバーのバージョン番号、サーバーのオペレーティングシステムの詳細、インストールされている Apache モジュールなど、利用可能なすべての情報を指定します。

```
Apache/2.4.37 (Red Hat Enterprise Linux) MyMod/1.2
```

ServerTokens Full-Release

以下のように、利用可能なすべての情報をリリースバージョンとともに指定します。

```
Apache/2.4.37 (Red Hat Enterprise Linux) (Release 41.module+el8.5.0+11772+c8e0c271)
```

ServerTokens Prod / ServerTokens ProductOnly

以下のように、Web サーバー名を指定します。

```
Apache
```

ServerTokens Major

以下のように、Web サーバーのメジャーリリースバージョンを指定します。

```
Apache/2
```

ServerTokens Minor

以下のように、Web サーバーのマイナーリリースバージョンを指定します。

```
Apache/2.4
```

ServerTokens Min / ServerTokens Minimal

以下のように、Web サーバーの最小リリースバージョンを指定します。

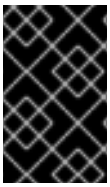
```
Apache/2.4.37
```

ServerTokens OS

以下のように、Web サーバーのリリースバージョンとオペレーティングシステムを指定します。

```
Apache/2.4.37 (Red Hat Enterprise Linux)
```

ServerTokens Prod オプションを使用して、攻撃者がシステムに関する貴重な情報を入手するリスクを軽減します。



重要

IncludesNoExec ディレクティブを削除しないでください。デフォルトでは、Server-Side Includes (SSI) モジュールは、コマンドを実行できません。これを変更すると、攻撃者がシステムにコマンドを入力できるようになる可能性があります。

httpd モジュールの削除

httpd モジュールを削除して、HTTP サーバーの機能を制限できます。これを行うには、**/etc/httpd/conf.modules.d/** または **/etc/httpd/conf.d/** ディレクトリーの設定ファイルを編集します。たとえば、プロキシーモジュールを削除するためには、以下のコマンドを実行します。

```
echo '# All proxy modules disabled' > /etc/httpd/conf.modules.d/00-proxy.conf
```

関連情報

- Apache HTTP サーバー
- Apache HTTP サーバーの SELinux ポリシーのカスタマイズ

6.5.2. Nginx サーバー設定のセキュリティー保護

Nginx は、高性能の HTTP およびプロキシーサーバーです。次の設定オプションを使用して、Nginx 設定を強化できます。

手順

- バージョン文字列を無効にするには、**server_tokens** 設定オプションを変更します。

  ```
  server_tokens off;
  ```

  このオプションは、サーバーのバージョン番号などの追加の情報表示を停止します。以下のようにこの設定では、Nginx によって処理されるすべての要求のサーバー名のみが表示されます。

  ```
  $ curl -sI http://localhost | grep Server
  Server: nginx
  ```

- 特定の **/etc/nginx/** conf ファイルに、特定の既知の Web アプリケーションの脆弱性を軽減するセキュリティーヘッダーを追加します。
  - たとえば、**X-Frame-Options** ヘッダーオプションは、Nginx が提供するコンテンツのフレーム化がされないように、ドメイン外のページを拒否して、クリックジャッキング攻撃を軽減します。

```
add_header X-Frame-Options "SAMEORIGIN";
```

- たとえば、**x-content-type** ヘッダーは、特定の古いブラウザーでの MIME タイプのスニッフィングを防ぎます。

```
add_header X-Content-Type-Options nosniff;
```

- また、**X-XSS-Protection** ヘッダーは、クロスサイトスクリプティング (XSS) フィルタリングを有効にし、Nginx での応答に含まれる可能性がある、悪意のあるコンテンツをブラウザーがレンダリングしないようにします。

```
add_header X-XSS-Protection "1; mode=block";
```

- たとえば、一般に公開されるサービスを制限し、訪問者からのサービスと受け入れを制限できます。

```
limit_except GET {
    allow 192.168.1.0/32;
    deny  all;
}
```

スニペットは、**GET** と **HEAD** を除くすべてのメソッドへのアクセスを制限します。

- 以下のように、HTTP メソッドを無効にできます。

```
# Allow GET, PUT, POST; return "405 Method Not Allowed" for all others.
if ( $request_method !~ ^(GET|PUT|POST)$ ) {
    return 405;
}
```

- Nginx Web サーバーによって提供されるデータを保護するように SSL を設定できます。これは、HTTPS 経由でのみ提供することを検討してください。さらに、Mozilla SSL Configuration Generator を使用して、Nginx サーバーで SSL を有効にするための安全な設定プロファイルを生成できます。生成された設定により、既知の脆弱なプロトコル (SSLv2 や SSLv3 など)、暗号、ハッシュアルゴリズム (3DES や MD5 など) が確実に無効化されます。また、SSL サーバーテストを使用して、設定した内容が最新のセキュリティー要件を満たしていることを確認できます。

関連情報

- Mozilla SSL Configuration Generator
- SSL Server Test

6.6. 認証されたローカルユーザーへのアクセスを制限することによる POSTGRESQL のセキュリティー保護

PostgreSQL は、オブジェクトリレーショナルデータベース管理システム (DBMS) です。Red Hat Enterprise Linux では、PostgreSQL は **postgresql-server** パッケージによって提供されます。

クライアント認証を設定して、攻撃のリスクを減らすことができます。データベースクラスターのデータディレクトリーに保存されている **pg_hba.conf** 設定ファイルは、クライアント認証を制御します。手順に従って、ホストベースの認証用に PostgreSQL を設定します。

手順

1. PostgreSQL をインストールします。

```
# yum install postgresql-server
```

2. 次のいずれかのオプションを使用して、データベースストレージ領域を初期化します。

   a. **initdb** ユーティリティーの使用:

```
$ initdb -D /home/postgresql/db1/
```

   -D オプションを指定した **initdb** コマンドを実行すると、指定したディレクトリーがまだ存在しない場合は作成します (例: **/home/postgresql/db1/**)。このディレクトリーには、データベースに保存されているすべてのデータと、クライアント認証設定ファイルが含まれています。

   b. **postgresql-setup** スクリプトの使用:

```
$ postgresql-setup --initdb
```

   デフォルトでは、スクリプトは **/var/lib/pgsql/data/** ディレクトリーを使用します。このスクリプトは、基本的なデータベースクラスター管理でシステム管理者を支援します。

3. 認証されたローカルユーザーが自分のユーザー名でデータベースにアクセスできるようにするには、**pg_hba.conf** ファイルの以下の行を変更します。

```
local   all             all                                     trust
```

   これは、データベースユーザーを作成し、ローカルユーザーを作成しないレイヤー型アプリケーションを使用する場合に、問題となることがあります。システム上のすべてのユーザー名を明示的に制御しない場合は、**pg_hba.conf** ファイルから **local** の行を削除してください。

4. データベースを再起動して、変更を適用します。

```
# systemctl restart postgresql
```

   前のコマンドはデータベースを更新し、設定ファイルの構文も検証します。

6.7. MEMCACHED サービスのセキュリティー保護

Memcached は、オープンソースの高性能分散メモリーオブジェクトキャッシングシステムです。データベースの負荷を軽減して、動的 Web アプリケーションのパフォーマンスを向上させることができます。

Memcached は、データベース呼び出し、API 呼び出し、またはページレンダリングの結果から、文字列やオブジェクトなどの任意のデータの小さなチャンクを格納するメモリー内のキーと値のストアです。Memcached を使用すると、十分に活用されていない領域から、より多くのメモリーを必要とするアプリケーションにメモリーを割り当てることができます。

2018 年に、パブリックインターネットに公開されている Memcached サーバーを悪用することによる DDoS 増幅攻撃の脆弱性が発見されました。これらの攻撃は、トランスポートに UDP プロトコルを使用する Memcached 通信を利用します。この攻撃は増幅率が高いため、効果的です。数百バイトのサイズの要求は、数メガバイトまたは数百メガバイトのサイズの応答を生成することができます。

ほとんどの場合、**memcached** サービスはパブリックインターネットに公開する必要はありません。このような弱点には、リモートの攻撃者が memcached に保存されている情報を漏洩または変更できるなど、独自のセキュリティー問題があります。

6.7.1. DDoS に対する Memcached の強化

セキュリティーリスクを軽減するために、以下の手順のうち、お使いの設定に該当するものをできるだけ多く実行してください。

手順

- LAN にファイアウォールを設定してください。Memcached サーバーにローカルネットワークだけでアクセスできるようにする必要がある場合は、**memcached** サービスで使用されるポートに外部トラフィックをルーティングしないでください。たとえば、許可されたポートのリストからデフォルトのポート **11211** を削除します。

```
# firewall-cmd --remove-port=11211/udp
# firewall-cmd --runtime-to-permanent
```

- アプリケーションと同じマシンで単一の memcached サーバーを使用する場合、ローカルホストトラフィックのみをリッスンするように **memcached** を設定します。**/etc/sysconfig/memcached** ファイルの **OPTIONS** 値を変更します。

```
OPTIONS="-l 127.0.0.1,::1"
```

- Simple Authentication and Security Layer (SASL) 認証を有効にします。

  1. **/etc/sasl2/memcached.conf** ファイルで、以下のように修正または追加します。

```
sasldb_path: /path.to/memcached.sasldb
```

  2. SASL データベースにアカウントを追加します。

```
# saslpasswd2 -a memcached -c cacheuser -f /path.to/memcached.sasldb
```

  3. **memcached** のユーザーとグループがデータベースにアクセスできることを確認します。

```
# chown memcached:memcached /path.to/memcached.sasldb
```

  4. **/etc/sysconfig/memcached** ファイルの **OPTIONS** パラメーターに **-S** 値を追加して、Memcached で SASL サポートを有効にします。

```
OPTIONS="-S"
```

  5. Memcached サーバーを再起動して、変更を適用します。

```
# systemctl restart memcached
```

  6. SASL データベースで作成したユーザー名とパスワードを、お使いのアプリケーションの Memcached クライアント設定に追加します。

- memcached クライアントとサーバー間の通信を TLS で暗号化します。

  1. **/etc/sysconfig/memcached** ファイルの **OPTIONS** パラメーターに **-Z** 値を追加して、TLS

1. **/etc/sysconfig/memcached** ファイルの **OPTIONS** パラメーターに **-Z** 値を追加して、TLS を使用した Memcached クライアントとサーバー間の暗号化通信を有効にします。

```
OPTIONS="-Z"
```

2. **-o ssl_chain_cert** オプションを使用して、証明書チェーンファイルパスを PEM 形式で追加します。

3. **-o ssl_key** オプションを使用して、秘密鍵ファイルのパスを追加します。

第7章 MACSEC を使用した同じ物理ネットワーク内のレイヤー 2 トラフィックの暗号化

MACsec を使用して、2 つのデバイス間の通信を (ポイントツーポイントで) セキュリティー保護できます。たとえば、ブランチオフィスがメトロイーサネット接続を介してセントラルオフィスに接続されている場合、オフィスを接続する 2 つのホストで MACsec を設定して、セキュリティーを強化できます。

Media Access Control Security (MACsec) は、イーサーネットリンクで異なるトラフィックタイプを保護するレイヤー 2 プロトコルです。これには以下が含まれます。

- DHCP (Dynamic Host Configuration Protocol)
- アドレス解決プロトコル (ARP)
- インターネットプロトコルのバージョン 4 / 6 (**IPv4** / **IPv6**)
- TCP や UDP などの IP 経由のトラフィック

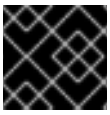
MACsec はデフォルトで、LAN 内のすべてのトラフィックを GCM-AES-128 アルゴリズムで暗号化および認証し、事前共有キーを使用して参加者ホスト間の接続を確立します。共有前の鍵を変更する場合は、MACsec を使用するネットワーク内のすべてのホストで NM 設定を更新する必要があります。

MACsec 接続は、親としてイーサネットネットワークカード、VLAN、トンネルデバイスなどのイーサネットデバイスを使用します。暗号化した接続のみを使用して他のホストと通信するように、MACsec デバイスでのみ IP 設定を指定するか、親デバイスに IP 設定を指定することもできます。後者の場合、親デバイスを使用して、暗号化されていない接続と暗号化された接続用の MACsec デバイスで他のホストと通信できます。

MACsec には特別なハードウェアは必要ありません。たとえば、ホストとスイッチの間のトラフィックのみを暗号化する場合を除き、任意のスイッチを使用できます。このシナリオでは、スイッチが MACsec もサポートする必要があります。

つまり、MACsec を設定する方法は 2 つあります。

- ホスト対ホスト
- 他のホストに切り替えるホスト



重要

MACsec は、同じ (物理または仮想) LAN のホスト間でのみ使用することができます。

7.1. NMCLI を使用した MACSEC 接続の設定

nmcli ツールを使用して、MACsec を使用するようにイーサーネットインターフェイスを設定できます。たとえば、イーサネット経由で接続された 2 つのホスト間に MACsec 接続を作成できます。

手順

1. MACsec を設定する最初のホストで:
   - 事前共有鍵の接続アソシエーション鍵 (CAK) と接続アソシエーション鍵名 (CKN) を作成します。

a. 16 バイトの 16 進 CAK を作成します。

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

b. 32 バイトの 16 進 CKN を作成します。

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

2. 両方のホストで、MACsec 接続を介して接続します。

3. MACsec 接続を作成します。

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

前の手順で生成された CAK および CKN を **macsec.mka-cak** および **macsec.mka-ckn** パラメーターで使用します。この値は、MACsec で保護されるネットワーク内のすべてのホストで同じである必要があります。

4. MACsec 接続で IP を設定します。

a. **IPv4** 設定を指定します。たとえば、静的 **IPv4** アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **macsec0** 接続に設定するには、以下のコマンドを実行します。

```
# 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. **IPv6** 設定を指定しますたとえば、静的 **IPv6** アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **macsec0** 接続に設定するには、以下のコマンドを実行します。

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

5. 接続をアクティベートします。

```
# nmcli connection up macsec0
```

検証

1. トラフィックが暗号化されていることを確認します。

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

2. オプション: 暗号化されていないトラフィックを表示します。

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

3. MACsec の統計を表示します。

```
# ip macsec show
```

4. integrity-only (encrypt off) および encryption (encrypt on) の各タイプの保護に対して個々のカウンターを表示します。

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

7.2. NMSTATECTL を使用した MACSEC 接続の設定

nmstatectl ユーティリティーを宣言的に使用して、イーサネットインターフェイスが MACsec を使用するように設定できます。たとえば、YAML ファイルでは、ネットワークの望ましい状態を記述します。ネットワークでは、イーサネット経由で接続された 2 つのホスト間に MACsec 接続があることが想定されます。**nmstatectl** ユーティリティーは、YAML ファイルを解釈し、ホスト間に永続的かつ一貫したネットワーク設定をデプロイします。

リンク層 (Open Systems Interconnection (OSI) モデルのレイヤー 2 とも呼ばれます) での通信を保護するために MACsec セキュリティー標準を使用すると、主に次のような利点が得られます。

- レイヤー 2 で暗号化することで、レイヤー 7 で個々のサービスを暗号化する必要がなくなります。これにより、各ホストの各エンドポイントで多数の証明書を管理することに関連するオーバーヘッドが削減されます。
- ルーターやスイッチなどの直接接続されたネットワークデバイス間のポイントツーポイントセキュリティー。
- アプリケーションや上位レイヤープロトコルに変更を加える必要がなくなります。

前提条件

- 物理または仮想イーサネットネットワークインターフェイスコントローラー (NIC) がサーバーに設定されている。
- **nmstate** パッケージがインストールされている。

手順

1. MACsec を設定する最初のホストで、事前共有キー用の接続関連キー (CAK: connectivity association key) および接続関連キー名 (CKN: connectivity-association key name) を作成します。

   a. 16 バイトの 16 進 CAK を作成します。

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

   b. 32 バイトの 16 進 CKN を作成します。

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

2. MACsec 接続を介して接続するホストの両方で、次の手順を実行します。

a. 次の設定を含む YAML ファイル (例: **create-macsec-connection.yml**) を作成します。

```
---
routes:
  config:
  - destination: 0.0.0.0/0
    next-hop-interface: macsec0
    next-hop-address: 192.0.2.2
    table-id: 254
  - destination: 192.0.2.2/32
    next-hop-interface: macsec0
    next-hop-address: 0.0.0.0
    table-id: 254
dns-resolver:
  config:
    search:
    - example.com
    server:
    - 192.0.2.200
    - 2001:db8:1::ffbb
interfaces:
- name: macsec0
  type: macsec
  state: up
  ipv4:
    enabled: true
    address:
    - ip: 192.0.2.1
      prefix-length: 32
  ipv6:
    enabled: true
    address:
    - ip: 2001:db8:1::1
      prefix-length: 64
  macsec:
    encrypt: true
    base-iface: enp0s1
    mka-cak: 50b71a8ef0bd5751ea76de6d6c98c03a
    mka-ckn: f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
    port: 0
    validation: strict
    send-sci: true
```

b. 前の手順で生成された CAK および CKN を **mka-cak** および **mka-ckn** パラメーターで使用します。この値は、MACsec で保護されるネットワーク内のすべてのホストで同じである必要があります。

c. オプション: 同じ YAML 設定ファイルで、次の設定も指定できます。

- 静的 IPv4 アドレス: **192.0.2.1** (サブネットマスクが **/32**)
- 静的 IPv6 アドレス: **2001:db8:1::1** (サブネットマスクが **/64**)
- IPv4 デフォルトゲートウェイ - **192.0.2.2**
- IPv4 DNS サーバー - **192.0.2.200**

- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**

3. 設定をシステムに適用します。

```
# nmstatectl apply create-macsec-connection.yml
```

検証

1. 現在の状態を YAML 形式で表示します。

```
# nmstatectl show macsec0
```

2. トラフィックが暗号化されていることを確認します。

```
# tcpdump -nn -i enp0s1
```

3. オプション: 暗号化されていないトラフィックを表示します。

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

4. MACsec の統計を表示します。

```
# ip macsec show
```

5. integrity-only (encrypt off) および encryption (encrypt on) の各タイプの保護に対して個々のカウンターを表示します。

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

関連情報

- MACsec: a different solution to encrypt network traffic

7.3. 関連情報

- MACsec: a different solution to encrypt network traffic ブログ

第8章 POSTFIX サービスを保護する

Postfix は、SMTP (Simple Mail Transfer Protocol) を使用して他の MTA 間で電子メッセージを配信したり、クライアントや配信エージェントに電子メールを送信したりするメール転送エージェント (MTA) です。MTA は相互間のトラフィックを暗号化できますが、デフォルトではそうしない場合があります。設定をより安全な値に変更することで、さまざまな攻撃に対するリスクを軽減することもできます。

8.1. POSTFIX ネットワーク関連のセキュリティーリスクの軽減

攻撃者がネットワーク経由でシステムに侵入するリスクを軽減するには、次のタスクをできるだけ多く実行してください。

- ネットワークファイルシステム (NFS) 共有ボリュームで **/var/spool/postfix/** メールスプールディレクトリーを共有しないでください。NFSv2 と NFSv3 は、ユーザー ID とグループ ID に対する制御を維持しません。したがって、2 人以上のユーザーが同じ UID を持っていると、互いのメールを受信して読むことができ、セキュリティー上のリスクが生じます。

注記

SECRPC_GSS カーネルモジュールは UID ベースの認証を使用しないため、この規則は Kerberos を使用する NFSv4 には適用されません。ただし、セキュリティーリスクを軽減するために、メールスプールディレクトリーを NFS 共有ボリュームに配置しないでください。

- Postfix サーバーの悪用の可能性を減らすために、メールユーザーは電子メールプログラムを使用して Postfix サーバーにアクセスする必要があります。メールサーバーでシェルアカウントを許可せず、**/etc/passwd** ファイル内のすべてのユーザーシェルを **/sbin/nologin** に設定します (**root** ユーザーは例外の可能性があります)。

- Postfix をネットワーク攻撃から保護するために、デフォルトではローカルループバックアドレスのみをリッスンするように設定されています。これは、**/etc/postfix/main.cf** ファイルの **inet_interfaces = localhost** 行を表示することで確認できます。これにより、Postfix はネットワークからではなく、ローカルシステムからのメールメッセージ (**cron** ジョブのレポートなど) のみを受け入れるようになります。これはデフォルトの設定で、Postfix をネットワーク攻撃から保護します。localhost の制限を取り除き、Postfix がすべてのインターフェイスでリッスンできるようにするには、**/etc/postfix/main.cf** で **inet_interfaces** パラメーターを **all** に設定します。

8.2. DOS 攻撃を制限するための POSTFIX 設定オプション

攻撃者は、トラフィックでサーバーをあふれさせたり、クラッシュを引き起こす情報を送信したりして、サービス拒否 (DoS) 攻撃を引き起こす可能性があります。**/etc/postfix/main.cf** ファイルで制限を設定することにより、このような攻撃のリスクを軽減するようにシステムを設定できます。既存のディレクティブの値を変更するか、**<directive> = <value>** 形式のカスタム値で新しいディレクティブを追加できます。

DoS 攻撃を制限するには、次のディレクティブリストを使用します。

smtpd_client_connection_rate_limit

クライアントがこのサービスに対して時間単位あたりに実行できる接続試行の最大数を制限します。デフォルト値は **0** です。これは、クライアントが時間単位で Postfix が受け入れることができる数と同じ数の接続を行うことができることを意味します。デフォルトでは、ディレクティブは信頼できるネットワークのクライアントを除外します。

anvil_rate_time_unit

レート制限を計算するための時間単位を定義します。デフォルト値は **60** 秒です。

smtpd_client_event_limit_exceptions

接続およびレート制限コマンドからクライアントを除外します。デフォルトでは、ディレクティブは信頼できるネットワークのクライアントを除外します。

smtpd_client_message_rate_limit

時間単位あたりのクライアントからの要求に対するメッセージ配信の最大数を定義します (Postfix が実際にそれらのメッセージを受け入れるかどうかは関係ありません)。

default_process_limit

特定のサービスを提供する Postfix 子プロセスのデフォルトの最大数を定義します。**master.cf** ファイル内の特定のサービスについては、このルールを無視できます。デフォルトでは、値は **100** です。

queue_minfree

キューファイルシステムでメールを受信するために必要な最小の空き容量を定義します。このディレクティブは現在、Postfix SMTP サーバーがメールを受け入れるかどうかを決定するために使用されています。デフォルトでは、Postfix SMTP サーバーは、空き容量が **message_size_limit** の 1.5 倍未満の場合に、**MAIL FROM** コマンドを拒否します。空き容量の最小値をこれよりも高く指定するには、**message_size_limit** の 1.5 倍以上の **queue_minfree** 値を指定します。デフォルトの **queue_minfree** 値は **0** です。

header_size_limit

メッセージヘッダーを格納するためのメモリーの最大量をバイト単位で定義します。ヘッダーが大きい場合、余分なヘッダーは破棄されます。デフォルトでは、値は **102400** バイトです。

message_size_limit

エンベロープ情報を含むメッセージの最大サイズをバイト単位で定義します。デフォルトでは、値は **10240000** バイトです。

8.3. POSTFIX が SASL を使用する設定

Postfix は Simple Authentication and Security Layer (SASL) ベースの SMTP 認証 (AUTH) をサポートしています。SMTP AUTH は Simple Mail Transfer Protocol の拡張です。現在、Postfix SMTP サーバーは次の方法で SASL 実装をサポートしています:

Dovecot SASL

Postfix SMTP サーバーは、UNIX ドメインソケットまたは TCP ソケットのいずれかを使用して、Dovecot SASL 実装と通信できます。Postfix と Dovecot アプリケーションが別のマシンで実行している場合は、この方法を使用します。

Cyrus SASL

有効にすると、SMTP クライアントは、サーバーとクライアントの両方でサポートおよび受け入れられる認証方法を使用して、SMTP サーバーで認証する必要があります。

前提条件

- **dovecot** パッケージがシステムにインストールされている

手順

1. Dovecot をセットアップします。

   a. **/etc/dovecot/conf.d/10-master.conf** ファイルに次の行を含めます。

```
service auth {
  unix_listener /var/spool/postfix/private/auth {
    mode = 0660
    user = postfix
    group = postfix
  }
}
```

前の例では、Postfix と Dovecot の間の通信に UNIX ドメインソケットを使用しています。また、**/var/spool/postfix/** ディレクトリーにあるメールキュー、および **postfix** ユーザーとグループの下で実行しているアプリケーションを含む Postfix SMTP サーバーのデフォルト設定を想定しています。

b. オプション: TCP 経由で Postfix 認証リクエストをリッスンするように Dovecot をセットアップします。

```
service auth {
  inet_listener {
      port = port-number
  }
}
```

c. **/etc/dovecot/conf.d/10-auth.conf** ファイルの **auth_mechanisms** パラメーターを編集して、電子メールクライアントが Dovecot での認証に使用する方法を指定します。

```
auth_mechanisms = plain login
```

auth_mechanisms パラメーターは、さまざまなプレーンテキストおよび非プレーンテキストの認証方法をサポートしています。

2. **/etc/postfix/main.cf** ファイルを変更して Postfix をセットアップします。

a. Postfix SMTP サーバーで SMTP 認証を有効にします。

```
smtpd_sasl_auth_enable = yes
```

b. SMTP 認証用の Dovecot SASL 実装の使用を有効にします。

```
smtpd_sasl_type = dovecot
```

c. Postfix キューディレクトリーに相対的な認証パスを指定します。相対パスを使用すると、Postfix サーバーが **chroot** で実行しているかどうかに関係なく、設定が確実に機能することに注意してください。

```
smtpd_sasl_path = private/auth
```

この手順では、Postfix と Dovecot の間の通信に UNIX ドメインソケットを使用します。

通信に TCP ソケットを使用する場合に、別のマシンで Dovecot を探すように Postfix を設定するには、次のような設定値を使用します。

```
smtpd_sasl_path = inet: ip-address : port-number
```

前の例で、**ip-address** を Dovecot マシンの IP アドレスに置き換え、 **port-number** を Dovecot の **/etc/dovecot/conf.d/10-master.conf** ファイルで指定されたポート番号に置き換えます。

d. Postfix SMTP サーバーがクライアントに提供する SASL メカニズムを指定します。暗号化されたセッションと暗号化されていないセッションに異なるメカニズムを指定できることに注意してください。

```
smtpd_sasl_security_options = noanonymous, noplaintext
smtpd_sasl_tls_security_options = noanonymous
```

前のディレクティブは、暗号化されていないセッションでは匿名認証が許可されず、暗号化されていないユーザー名またはパスワードを送信するメカニズムが許可されていないことを指定しています。暗号化セッション (TLS を使用) の場合、非匿名認証メカニズムのみが許可されます。

関連情報

- Postfix SMTP server policy - SASL mechanism properties
- Postfix and Dovecot SASL
- Postfix SMTP サーバーで SASL 認証を設定する