



Red Hat Enterprise Linux 9

セキュリティの強化

Red Hat Enterprise Linux 9 システムのセキュリティ強化

Red Hat Enterprise Linux 9 セキュリティーの強化

Red Hat Enterprise Linux 9 システムのセキュリティー強化

Legal Notice

Copyright © 2025 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

Red Hat Enterprise Linux サーバーとワークステーションをローカルおよびリモートの侵入、悪用、および悪意のある活動から保護するためのプロセスと実践を学びます。これらのアプローチとツールを使用することで、データセンター、職場、および家庭用のより安全なコンピューティング環境を作成できます。

Table of Contents

RED HAT ドキュメントへのフィードバック (英語のみ)	6
第1章 インストール中およびインストール直後の RHEL の保護	7
1.1. ディスクパーティション設定	7
1.2. インストールプロセス時のネットワーク接続の制限	7
1.3. 必要なパッケージの最小限のインストール	8
1.4. インストール後の手順	8
1.5. WEB コンソールを使用して CPU のセキュリティの問題を防ぐために SMT を無効化する手順	8
第2章 FIPS モードへの RHEL の切り替え	10
2.1. FEDERAL INFORMATION PROCESSING STANDARDS 140 および FIPS モード	10
2.2. FIPS モードが有効なシステムのインストール	11
2.3. FIPS モードへのシステムの切り替え	12
2.4. コンテナでの FIPS モードの有効化	13
2.5. FIPS 140-3 に準拠していない暗号化を使用している RHEL アプリケーションのリスト	14
第3章 システム全体の暗号化ポリシーの使用	17
3.1. システム全体の暗号化ポリシー	17
3.2. システム全体の暗号化ポリシーの変更	19
3.3. システム全体の暗号化ポリシーを以前のリリースと互換性のあるモードに切り替える	20
3.4. SHA-1 を再度有効に	20
3.5. WEB コンソールでシステム全体の暗号化ポリシーを設定する	21
3.6. システム全体の暗号化ポリシーからアプリケーションを除外する	23
3.7. 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	25
3.8. システム全体のカスタム暗号化ポリシーの作成および設定	26
3.9. CRYPTO_POLICIES RHEL システムロールを使用した FUTURE 暗号化ポリシーによるセキュリティの強化	27
第4章 PKCS #11 で暗号化ハードウェアを使用するようにアプリケーションを設定	31
4.1. PKCS #11 による暗号化ハードウェアへの対応	31
4.2. スマートカードに保存した SSH 鍵による認証	31
4.3. スマートカード上の証明書を使用して認証するアプリケーションの設定	33
4.4. APACHE で秘密鍵を保護する HSM の使用	34
4.5. NGINX で秘密鍵を保護する HSM の使用	34
4.6. 関連情報	34
第5章 POLKIT を使用したスマートカードへのアクセスの制御	35
5.1. POLKIT を介したスマートカードアクセス制御	35
5.2. PC/SC および POLKIT に関連する問題のトラブルシューティング	35
5.3. PC/SC への POLKIT 認可の詳細情報の表示	37
5.4. 関連情報	38
第6章 設定コンプライアンスおよび脆弱性スキャンの開始	39
6.1. RHEL における設定コンプライアンスツール	39
6.2. 脆弱性スキャン	40
6.3. 設定コンプライアンススキャン	42
6.4. 特定のベースラインに合わせたシステムの修正	46
6.5. SSG ANSIBLE PLAYBOOK を使用した特定のベースラインに合わせたシステムの修正	47
6.6. システムを特定のベースラインに合わせるための修復用 ANSIBLE PLAYBOOK の作成	48
6.7. 後でアプリケーションを修復するための BASH スクリプトの作成	50
6.8. SCAP WORKBENCH を使用したカスタムプロファイルでシステムのスキャン	51
6.9. インストール直後にセキュリティプロファイルに準拠するシステムのデプロイメント	55
6.10. コンテナおよびコンテナイメージの脆弱性スキャン	58

6.11. 特定のベースラインを使用したコンテナまたはコンテナイメージのセキュリティーコンプライアンスの評価	59
6.12. RHEL 9 でサポートされる SCAP SECURITY GUIDE プロファイル	60
6.13. 関連情報	72
第7章 KEYLIME でシステムの整合性を確保する	74
7.1. KEYLIME の仕組み	74
7.2. パッケージから KEYLIME VERIFIER をデプロイする	76
7.3. KEYLIME VERIFIER をコンテナとしてデプロイする	79
7.4. パッケージから KEYLIME REGISTRAR をデプロイする	82
7.5. KEYLIME REGISTRAR をコンテナとしてデプロイする	84
7.6. RHEL システムロールを使用して KEYLIME サーバーをデプロイする	87
7.7. KEYLIME_SERVER RHEL システムロールの変数	89
7.8. パッケージから KEYLIME テナントをデプロイする	90
7.9. パッケージから KEYLIME エージェントをデプロイする	92
7.10. ランタイム監視用に KEYLIME を設定する	96
7.11. ブート測定のアテステーション用に KEYLIME を設定する	99
7.12. KEYLIME の環境変数	101
第8章 AIDE で整合性の確認	114
8.1. AIDE のインストール	114
8.2. AIDE を使用した整合性チェックの実行	115
8.3. AIDE データベースの更新	115
8.4. ファイル整合性ツール:AIDE および IMA	116
8.5. AIDE RHEL システムロールを使用したファイル整合性チェックの設定	116
8.6. 関連情報	118
第9章 LUKS を使用したブロックデバイスの暗号化	119
9.1. LUKS ディスクの暗号化	119
9.2. RHEL の LUKS バージョン	120
9.3. LUKS2 再暗号化中のデータ保護のオプション	121
9.4. LUKS2 を使用したブロックデバイスの既存データの暗号化	121
9.5. 独立したヘッダーがある LUKS2 を使用してブロックデバイスの既存データの暗号化	124
9.6. LUKS2 を使用した空のブロックデバイスの暗号化	127
9.7. WEB コンソールでの LUKS パスフレーズの設定	128
9.8. WEB コンソールで LUKS パスフレーズの変更	129
9.9. コマンドラインを使用した LUKS パスフレーズの変更	130
9.10. STORAGE RHEL システムロールを使用して LUKS2 暗号化ボリュームを作成する	130
第10章 ポリシーベースの復号を使用した暗号化ボリュームの自動ロック解除の設定	134
10.1. NETWORK-BOUND DISK ENCRYPTION	134
10.2. ENFORCING モードの SELINUX を使用して TANG サーバーをデプロイする	136
10.3. TANG サーバーの鍵のローテーションおよびクライアントでのバインディングの更新	137
10.4. WEB コンソールで TANG キーを使用して自動ロック解除を設定する	139
10.5. 基本的な NBDE および TPM2 暗号化クライアント操作	142
10.6. LUKS 暗号化ボリュームのロックを自動解除するように NBDE クライアントを設定する	144
10.7. 静的な IP 設定を持つ NBDE クライアントの設定	146
10.8. TPM 2.0 ポリシーを使用して LUKS 暗号化ボリュームの手動登録を設定する	147
10.9. PKCS #11 ピンを使用して LUKS 暗号化ボリュームのロック解除を設定する	149
10.10. LUKS で暗号化したボリュームからの CLEVIS ピンの手動削除	151
10.11. キックスタートを使用して LUKS 暗号化ボリュームの自動登録を設定する	152
10.12. LUKS で暗号化されたリムーバブルストレージデバイスの自動ロック解除を設定する	154
10.13. 高可用性 NBDE システムをデプロイする	154
10.14. NBDE ネットワークで仮想マシンをデプロイする	156

10.15. NBDE を使用してクラウド環境用の自動登録可能な仮想マシンイメージをビルドする	156
10.16. コンテナとしての TANG のデプロイ	157
10.17. RHEL システムロールを使用した NBDE の設定	158
第11章 システムの監査	165
11.1. LINUX の AUDIT	165
11.2. AUDIT システムのアーキテクチャー	166
11.3. 環境を保護するための AUDITD の設定	167
11.4. AUDITD の開始および制御	168
11.5. AUDIT ログファイルについて	169
11.6. AUDITCTL で AUDIT ルールを定義および実行	173
11.7. 永続的な AUDIT ルールの定義	174
11.8. 標準に準拠するための事前設定された監査ルールファイル	175
11.9. 永続ルールを定義する AUGENRULES の使用	175
11.10. AUGENRULES の無効化	176
11.11. ソフトウェアの更新を監視するための AUDIT の設定	177
11.12. AUDIT によるユーザーログイン時刻の監視	179
11.13. 関連情報	180
第12章 FAPOLICYD を使用したアプリケーションの拒否および許可	181
12.1. FAPOLICYD の概要	181
12.2. FAPOLICYD のデプロイ	182
12.3. 追加の信頼ソースを使用してファイルを信頼できるものとしてマークする	184
12.4. FAPOLICYD のカスタムの許可および拒否ルールの追加	185
12.5. FAPOLICYD 整合性チェックの有効化	188
12.6. FAPOLICYD に関連する問題のトラブルシューティング	189
12.7. FAPOLICYD RHEL システムロールを使用してユーザーによる信頼できないコードの実行を防止する	191
12.8. 関連情報	193
第13章 侵入型 USB デバイスに対するシステムの保護	194
13.1. USBGUARD	194
13.2. USBGUARD のインストール	194
13.3. CLI を使用した USB デバイスのブロックと許可	195
13.4. USB デバイスの永続的なブロックおよび許可	196
13.5. USB デバイス用のカスタムポリシーの作成	198
13.6. USB デバイス用の構造化されたカスタムポリシーの作成	199
13.7. USBGUARD IPC インターフェイスを使用するユーザーおよびグループの許可	200
13.8. LINUX 監査ログへの USBGUARD 許可イベントの記録	201
13.9. 関連情報	202
第14章 リモートロギングソリューションの設定	203
14.1. RSYSLOG ロギングサービス	203
14.2. RSYSLOG ドキュメントのインストール	203
14.3. TCP でのリモートロギング用のサーバーの設定	204
14.4. TCP 経由のサーバーへのリモートロギングの設定	206
14.5. TLS 暗号化リモートロギングの設定	207
14.6. UDP でリモートロギング情報を受信するためのサーバー設定	212
14.7. UDP 経由のサーバーへのリモートロギングの設定	214
14.8. RSYSLOG の負荷分散ヘルパー	215
14.9. 信頼できるリモートロギングの設定	216
14.10. サポート対象の RSYSLOG モジュール	217
14.11. カーネルメッセージをリモートホストに記録するように NETCONSOLE サービスを設定	218
14.12. 関連情報	218

第15章 LOGGING システムロールの使用	220
15.1. LOGGING RHEL システムロールを使用したローカルログメッセージのフィルタリング	220
15.2. LOGGING RHEL システムロールを使用したリモートロギングソリューションの適用	222
15.3. TLS を使用した LOGGING RHEL システムロールの使用	225
15.4. RELP で LOGGING RHEL システムロールの使用	231
15.5. 関連情報	236

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 インストール中およびインストール直後の RHEL の保護

セキュリティへの対応は、Red Hat Enterprise Linux をインストールする前にすでに始まっています。最初からシステムのセキュリティを設定することで、追加のセキュリティ設定を実装することがより簡単になります。

1.1. ディスクパーティション設定

ディスクパーティション設定の推奨プラクティスは、ベアメタルマシンへのインストールと、インストール済みオペレーティングシステムを含む仮想ディスクハードウェアおよびファイルシステムの調整をサポートする仮想化環境またはクラウド環境とでは異なります。

ベアメタルインストール でデータの分離と保護を確実に行うには、**/boot**、**/**、**/home**、**/tmp**、**/var/tmp**/ディレクトリー用に個別のパーティションを作成します。

/boot

このパーティションは、システムの起動時にシステムが最初に読み込むパーティションです。RHEL 9 でシステムを起動するのに使用するブートローダーとカーネルイメージはこのパーティションに保存されます。このパーティションは暗号化しないでください。このパーティションが **/** に含まれており、そのパーティションが暗号化されているなどの理由で利用できなくなると、システムを起動できなくなります。

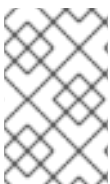
/home

ユーザーデータ (**/home**) が別のパーティションではなく **/** に保存されていると、このパーティションが満杯になり、オペレーティングシステムが不安定になる可能性があります。また、システムを、RHEL 9 の次のバージョンにアップグレードする際に、**/home** パーティションにデータを保存できると、このデータはインストール時に上書きされないため、アップグレードが非常に簡単になります。root パーティション (**/**) が破損すると、データが完全に失われます。したがって、パーティションを分けることが、データ損失に対する保護につながります。また、このパーティションを、頻繁にバックアップを作成する対象にすることも可能です。

/tmp および **/var/tmp**

/tmp ディレクトリーおよび **/var/tmp** ディレクトリーは、どちらも長期保存の必要がないデータを保存するために使用されます。しかし、このいずれかのディレクトリーでデータがあふれると、ストレージ領域がすべて使用されてしまう可能性があります。このディレクトリーは **/** に置かれているため、こうした状態が発生すると、システムが不安定になり、クラッシュする可能性があります。そのため、このディレクトリーは個別のパーティションに移動することが推奨されます。

仮想マシンまたはクラウドインスタンスの場合、**/boot**、**/home**、**/tmp**、および **/var/tmp** パーティションの分離は任意です。**/** パーティションがいっぱいになり始めたら、仮想ディスクのサイズとそのパーティションを拡張できるためです。**/** パーティションがいっぱいにならないように、その使用状況を定期的にチェックするモニタリングを設定してから、仮想ディスクのサイズを適宜拡張してください。



注記

インストールプロセス時に、パーティションを暗号化するオプションがあります。パスフレーズを入力する必要があります。これは、パーティションのデータを保護するのに使用されるバルク暗号鍵を解除する鍵として使用されます。

1.2. インストールプロセス時のネットワーク接続の制限

RHEL9 をインストールする際に使用するインストールメディアは、特定のタイミングで作成されたスナップショットです。そのため、セキュリティ修正が最新ののではなく、このインストールメディアで設定するシステムが公開されてから修正された特定の問題に対して安全性に欠ける場合があります。

す。

脆弱性が含まれる可能性のあるオペレーティングシステムをインストールする場合には、必ず、公開レベルを、必要最小限のネットワークゾーンに限定してください。最も安全な選択肢は、インストールプロセス時にマシンをネットワークから切断した状態にする“ネットワークなし”ゾーンです。インターネット接続からのリスクが最も高く、一方で LAN またはイントラネット接続で十分な場合もあります。セキュリティのベストプラクティスに従い、ネットワークから RHEL 9 をインストールする場合は、お使いのリポジトリに最も近いゾーンを選択してください。

1.3. 必要なパッケージの最小限のインストール

コンピューターの各ソフトウェアには脆弱性が潜んでいる可能性があるため、実際に使用するパッケージのみをインストールすることがベストプラクティスになります。インストールを DVD から行う場合は、インストールしたいパッケージのみを選択するようにします。その他のパッケージが必要になる場合は、後でいつでもシステムに追加できます。

1.4. インストール後の手順

以下は、RHEL 9 のインストール直後に実行する必要があるセキュリティ関連の手順です。

- システムを更新します。root で以下のコマンドを実行します。

```
# dnf update
```

- ファイアウォールサービスの **firewalld** は、Red Hat Enterprise Linux のインストールによって自動的に有効になりますが、キックスタート設定などで明示的に無効になっている場合があります。このような場合は、ファイアウォールを再度有効にしてください。

firewalld を開始するには、root で次のコマンドを実行します。

```
# systemctl start firewalld
# systemctl enable firewalld
```

- セキュリティを強化するために、不要なサービスは無効にしてください。たとえば、コンピューターにプリンターがインストールされていない場合は、次のコマンドを使用して、**cups** サービスを無効にします。

```
# systemctl disable cups
```

アクティブなサービスを確認するには、次のコマンドを実行します。

```
$ systemctl list-units | grep service
```

1.5. WEB コンソールを使用して CPU のセキュリティの問題を防ぐために SMT を無効化する手順

CPU SMT (Simultaneous Multi Threading) を悪用する攻撃が発生した場合に SMT を無効にします。SMT を無効にすると、L1TF や MDS などのセキュリティ脆弱性を軽減できます。



重要

SMT を無効にすると、システムパフォーマンスが低下する可能性があります。

前提条件

- RHEL 9 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。

手順

1. RHEL 9 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. **Overview** タブで、**System information** フィールドを見つけて、**View hardware details** をクリックします。
3. **CPU Security** 行で、**Mitigations** をクリックします。
このリンクがない場合は、システムが SMT に対応していないため、攻撃を受けません。
4. **CPU Security Toggles** テーブルで、**Disable simultaneous multithreading (nosmt)** オプションに切り替えます。
5. **Save and reboot** ボタンをクリックします。

システムの再起動後、CPU は SMT を使用しなくなりました。

関連情報

- [L1 Terminal Fault \(L1TF\) を使用したカーネルのサイドチャネル攻撃: CVE-2018-3620 & CVE-2018-3646](#)
- [MDS - マイクロアーキテクチャーデータサンプリング - CVE-2018-12130、CVE-2018-12126、CVE-2018-12127、および CVE-2019-11091](#)

第2章 FIPS モードへの RHEL の切り替え

連邦情報処理標準 (FIPS) 140-3 で義務付けられている暗号化モジュールの自己チェックを有効にするには、FIPS モードで RHEL 9 を操作する必要があります。FIPS 準拠を目指す場合は、FIPS モードでインストールを開始することを推奨します。

暗号化モジュールの検証ステータスは、Red Hat カスタマーポータルの [Product compliance](#) ページの [FIPS - Federal Information Processing Standards](#) セクションで確認できます。

2.1. FEDERAL INFORMATION PROCESSING STANDARDS 140 および FIPS モード

Federal Information Processing Standards (FIPS) Publication 140 は、暗号化モジュールの品質を確保するために National Institute of Standards and Technology (NIST) によって開発された一連のコンピュータセキュリティ標準です。FIPS 140 標準は、暗号化ツールがアルゴリズムを正しく実装できるようにします。ランタイム暗号アルゴリズムと整合性セルフテストは、システムが標準の要件を満たす暗号を使用していることを確認するためのメカニズムの一部です。

FIPS モードの RHEL

RHEL システムで FIPS 承認のアルゴリズムだけを使用してすべての暗号鍵を生成および使用するには、RHEL を FIPS モードに切り替える必要があります。

次のいずれかの方法を使用して、FIPS モードを有効にできます。

- FIPS モードでのインストールの開始
- インストール後に FIPS モードにシステムを切り替えます。

FIPS 準拠を目指す場合は、FIPS モードでインストールを開始してください。これにより、デプロイ済みのシステムの変換に伴う暗号鍵マテリアルの再生成と、変換後のシステムのコンプライアンス再評価を行う必要がなくなります。

FIPS 準拠のシステムを運用するには、すべての暗号化キーマテリアルを FIPS モードで作成します。さらに、暗号鍵マテリアルは、セキュアにラップされていない限り、絶対に FIPS 環境の外部に出さないでください。また、絶対に FIPS 以外の環境でラップを解除しないでください。

Red Hat カスタマーポータルの [Product compliance](#) ページの [FIPS - Federal Information Processing Standards](#) セクションで、選択した RHEL マイナーリリースの暗号化モジュール検証ステータスの概要を確認できます。

インストール後に FIPS モードに切り替える

fips-mode-setup ツールを使用してシステムを FIPS モードに切り替えても、FIPS 140 標準への準拠は保証されません。システムを FIPS モードに設定した後にすべての暗号キーを再生成することは、可能でない場合があります。たとえば、ユーザーの暗号キーを含む既存の IdM レalm の場合、すべてのキーを再生成することはできません。FIPS モードでインストールを開始できない場合は、インストール後の設定手順を実行したり、ワークロードをインストールしたりする前に、インストール後の最初の手順として常に FIPS モードを有効にしてください。

fips-mode-setup ツールも内部的に **FIPS** システム全体の暗号化ポリシーを使用します。ただし、**update-crypto-policies --set FIPS** コマンドが実行する内容に加え、**fips-mode-setup** は、**fips-finish-install** ツールを使用して FIPS dracut モジュールを確実にインストールします。また、**fips=1** ブートオプションをカーネルコマンドラインに追加し、初期 RAM ディスクを再生成します。

さらに、FIPS モードで必要な制限の適用は、**/proc/sys/crypto/fips_enabled** ファイルの内容によって異なります。ファイルに **1** が含まれている場合、RHEL コア暗号化コンポーネントは、FIPS 承認の暗

号化アルゴリズムの実装のみを使用するモードに切り替わります。`/proc/sys/crypto/fips_enabled` に 0 が含まれている場合、暗号化コンポーネントは FIPS モードを有効にしません。

暗号化ポリシーにおける FIPS

FIPS システム全体の暗号化ポリシーは、より高いレベルの制限を設定するのに役立ちます。したがって、暗号化の俊敏性をサポートする通信プロトコルは、選択時にシステムが拒否する暗号をアナウンスしません。たとえば、ChaCha20 アルゴリズムは FIPS によって承認されておらず、**FIPS** 暗号化ポリシーは、TLS サーバーおよびクライアントが

TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 TLS 暗号スイートをアナウンスしないようにします。これは、そのような暗号を使用しようとすると失敗するためです。

RHEL を FIPS モードで操作し、独自の FIPS モード関連の設定オプションを提供するアプリケーションを使用する場合は、これらのオプションと対応するアプリケーションのガイダンスを無視してください。FIPS モードで実行されているシステムとシステム全体の暗号化ポリシーは、FIPS 準拠の暗号化のみを適用します。たとえば、システムが FIPS モードで実行されている場合、Node.js 設定オプション `--enable-fips` は無視されます。FIPS モードで実行されていないシステムで `--enable-fips` オプションを使用すると、FIPS-140 準拠の要件を満たせなくなります。



警告

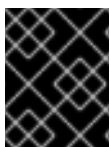
FIPS モードで実行されている RHEL 9.2 以降のシステムでは、FIPS 140-3 標準の要件に従って、TLS 1.2 接続で Extended Master Secret (EMS) 拡張機能 (RFC 7627) を使用する必要があります。したがって、EMS または TLS 1.3 をサポートしていないレガシークライアントは、FIPS モードで実行されている RHEL 9 サーバーに接続できません。FIPS モードの RHEL 9 クライアントは、EMS なしで TLS 1.2 のみをサポートするサーバーに接続できません。詳細は、Red Hat ナレッジベースのソリューション [TLS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

関連情報

- Red Hat カスタマーポータルの [Product compliance](#) ページの [FIPS - Federal Information Processing Standards](#) セクション
- [RHEL のシステム全体の暗号化ポリシー](#)
- [FIPS publications at NIST Computer Security Resource Center](#)
- [Federal Information Processing Standards Publication: FIPS 140-3](#)

2.2. FIPS モードが有効なシステムのインストール

連邦情報処理規格 (FIPS) 140 で義務付けられている暗号化モジュールの自己チェックを有効にするには、システムのインストール時に FIPS モードを有効にします。



重要

RHEL のインストール中に FIPS モードを有効にするだけで、システムは FIPS で承認されるアルゴリズムと継続的な監視テストですべての鍵を生成するようになります。

**警告**

FIPS モードのセットアップを完了した後、FIPS モードをオフにすると、システムが必ず不整合な状態になります。このような変更が必要な場合、システムを完全に再インストールするのが唯一の正しい方法です。

手順

1. システムのインストール時に **fips=1** オプションをカーネルコマンドラインに追加します。
2. ソフトウェアの選択段階で、サードパーティーのソフトウェアをインストールしないでください。
3. インストール後に、システムは FIPS モードで自動的に起動します。

検証

- システムが起動したら、FIPS モードが有効になっていることを確認します。

```
$ fips-mode-setup --check
FIPS mode is enabled.
```

関連情報

- [起動オプションの編集](#)

2.3. FIPS モードへのシステムの切り替え

システム全体の暗号化ポリシーには、連邦情報処理標準 (FIPS) Publication 140 の要件に従って暗号化アルゴリズムを有効にするポリシーレベルが含まれています。FIPS モードを有効または無効にする **fips-mode-setup** ツールは、内部的に **FIPS** のシステム全体の暗号化ポリシーを使用します。

FIPS システム全体の暗号化ポリシーを使用してシステムを FIPS モードに切り替えても、FIPS 140 標準への準拠は保証されません。システムを FIPS モードに設定した後にすべての暗号キーを再生成することは、可能でない場合があります。たとえば、ユーザーの暗号キーを含む既存の IdM レルムの場合、すべてのキーを再生成することはできません。

**重要**

RHEL のインストール中に FIPS モード を有効にするだけで、システムは FIPS で承認されるアルゴリズムと継続的な監視テストですべてのキーを生成するようになります。

fips-mode-setup ツールは、**FIPS** ポリシーを内部的に使用します。ただし、**--set FIPS** オプションを指定した **update-crypto-policies** コマンドが実行する内容に加え、**fips-mode-setup** は、**fips-finish-install** ツールを使用して FIPS dracut モジュールを確実にインストールします。また、**fips=1** ブートオプションをカーネルコマンドラインに追加し、初期 RAM ディスクを再生成します。



警告

FIPS モードのセットアップを完了した後、FIPS モードをオフにすると、システムが必ず不整合な状態になります。このような変更が必要な場合、システムを完全に再インストールするのが唯一の正しい方法です。

手順

1. システムを FIPS モードに切り替えるには、以下のコマンドを実行します。

```
# fips-mode-setup --enable
Kernel initramdisks are being regenerated. This might take some time.
Setting system policy to FIPS
Note: System-wide crypto policies are applied on application start-up.
It is recommended to restart the system for the change of policies
to fully take place.
FIPS mode will be enabled.
Please reboot the system for the setting to take effect.
```

2. 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

- システムが再起動したら、FIPS モードの現在の状態を確認できます。

```
# fips-mode-setup --check
FIPS mode is enabled.
```

関連情報

- システム上の **fips-mode-setup(8)** man ページ
- Red Hat カスタマーポータルの [Product compliance](#) ページの [FIPS - Federal Information Processing Standards](#) セクション
- [How to enable FIPS on instances using Red Hat Unified Kernel Images](#) (Red Hat ナレッジベース)
- NIST (National Institute of Standards and Technology) の Web サイトの [Security Requirements for Cryptographic Modules](#)。

2.4. コンテナでの FIPS モードの有効化

Federal Information Processing Standard Publication 140-2 (FIPS モード) で義務付けられている暗号化モジュールのセルフチェックの完全なセットを有効にするには、ホストシステムのカーネルが FIPS モードで実行されている必要があります。**podman** ユーティリティは、サポートされているコンテナで FIPS モードを自動的に有効にします。

コンテナで **fips-mode-setup** コマンドが正しく機能せず、このシナリオでこのコマンドを使用して FIPS モードを有効にしたり確認することができません。

前提条件

- ホストシステムが FIPS モードである必要があります。

手順

- FIPS モードが有効になっているシステムでは、**podman** ユーティリティーはサポートされているコンテナで FIPS モードを自動的に有効にします。

関連情報

- [FIPS モードへのシステムの切り替え](#)
- [FIPS モードでのシステムのインストール](#)
- Red Hat カスタマーポータルの [Product compliance](#) ページの [FIPS - Federal Information Processing Standards](#) セクション

2.5. FIPS 140-3 に準拠していない暗号化を使用している RHEL アプリケーションのリスト

FIPS 140-3 などの関連するすべての暗号化認定に合格するには、コア暗号化コンポーネントセットのライブラリーを使用します。これらのライブラリーは、**libgcrypt** を除き、RHEL システム全体の暗号化ポリシーに従います。

コア暗号化コンポーネントの概要、そのコンポーネントの選択方法、オペレーティングシステムへの統合方法、ハードウェアセキュリティモジュールおよびスマートカードのサポート方法、暗号化による認定の適用方法の概要は、Red Hat ナレッジベースの記事 [RHEL core cryptographic components](#) を参照してください。

FIPS 140-3 に準拠していない暗号化を使用している RHEL 9 アプリケーションのリスト

Bacula

CRAM-MD5 認証プロトコルを実装します。

Cyrus SASL

SCRAM-SHA-1 認証方式を使用します。

Dovecot

SCRAM-SHA-1 を使用します。

Emacs

SCRAM-SHA-1 を使用します。

FreeRADIUS

認証プロトコルに MD5 および SHA-1 を使用します。

Ghostscript

ドキュメントを暗号化および復号化するためのカスタムの cryptography 実装 (MD5、RC4、SHA-2、AES)

GRUB

SHA-1 を必要とするレガシーファームウェアプロトコルをサポートし、**libgcrypt** ライブラリーを含みます。

iPXE

TLS スタックを実装します。

Kerberos

SHA-1 (Windows との相互運用性) のサポートを維持します。

Lasso

lasso_wsse_username_token_derive_key () 鍵導出関数 (KDF) は SHA-1 を使用します。

MariaDB、MariaDB コネクター

mysql_native_password 認証プラグインは SHA-1 を使用します。

MySQL

mysql_native_password は SHA-1 を使用します。

OpenIPMI

RAKP-HMAC-MD5 認証方式は、FIPS の使用が承認されておらず、FIPS モードでは機能しません。

OVMF (UEFI ファームウェア)、Edk2、shim

完全な暗号スタック (OpenSSL ライブラリーの埋め込みコピー)。

Perl

HMAC、HMAC-SHA1、HMAC-MD5、SHA-1、SHA-224 などを使用します。

Pidgin

DES および RC4 暗号を実装します。

PKCS #12 ファイル処理 (OpenSSL、GnuTLS、NSS、Firefox、Java)

ファイル全体の HMAC の計算に使用されるキー派生関数 (KDF) が FIPS で承認されていないため、PKCS #12 のすべての使用は FIPS に準拠していません。そのため、PKCS #12 ファイルは、FIPS 準拠のためにプレーンテキストと見なされます。鍵転送の目的で、FIPS 承認の暗号化方式を使用して PKCS #12 (.p12) ファイルをラップします。

Poppler

元の PDF (MD5、RC4、SHA-1 など) に存在する場合は、許可されていないアルゴリズムに基づいて署名、パスワード、および暗号化を使用して PDF を保存できます。

PostgreSQL

Blowfish、DES、MD5 を実装します。KDF は SHA-1 を使用します。

QAT エンジン

暗号化プリミティブのハードウェアおよびソフトウェア実装 (RSA、EC、DH、AES、...)

Ruby

安全でないライブラリー関数 MD5 および SHA-1 を提供します。

Samba

RC4 および DES (Windows との相互運用性) のサポートを維持します。

Syslinux

BIOS パスワードは SHA-1 を使用します。

SWTPM

OpenSSL の使用時に FIPS モードを明示的に無効にします。

Unbound

DNS 仕様では、DNSSEC リゾルバーが検証のために DNSKEY レコードで SHA-1 ベースのアルゴリズムを使用する必要があります。

Valgrind

AES、SHA ハッシュ。 [1]

zip

パスワードを使用してアーカイブを暗号化および復号化するためのカスタム暗号化実装 (セキュアでない PKWARE 暗号化アルゴリズム)。

関連情報

- Red Hat カスタマーポータルの [Product compliance](#) ページの [FIPS - Federal Information Processing Standards](#) セクション
- [RHEL core cryptographic components](#) (Red Hat ナレッジベース)

[1] ARM 上の AES-NI、SHA-1 および SHA-2 などのソフトウェアハードウェアオフロード操作を再実装します。

第3章 システム全体の暗号化ポリシーの使用

システム全体の暗号化ポリシーは、コア暗号化サブシステムを設定するシステムコンポーネントで、TLS、IPsec、SSH、DNSSec、および Kerberos の各プロトコルに対応します。これにより、管理者が選択できる小規模セットのポリシーを提供します。

3.1. システム全体の暗号化ポリシー

システム全体のポリシーを設定すると、RHEL のアプリケーションはそのポリシーに従い、ポリシーを満たしていないアルゴリズムやプロトコルを使用するように明示的に要求されない限り、その使用を拒否します。つまり、システムが提供した設定で実行する際に、デフォルトのアプリケーションの挙動にポリシーを適用しますが、必要な場合は上書きできます。

RHEL 9 には、以下の定義済みポリシーが含まれています。

DEFAULT

デフォルトのシステム全体の暗号化ポリシーレベルで、現在の脅威モデルに対して安全なものです。TLS プロトコルの 1.2 と 1.3、IKEv2 プロトコル、および SSH2 プロトコルが使用できます。RSA 鍵と Diffie-Hellman パラメーターは長さが 2048 ビット以上であれば許容されます。

LEGACY

Red Hat Enterprise Linux 6 以前との互換性を最大限に確保します。攻撃対象領域が増えるため、セキュリティが低下します。SHA-1 は、TLS ハッシュ、署名、およびアルゴリズムとして使用できます。CBC モードの暗号は、SSH と併用できます。GnuTLS を使用するアプリケーションは、SHA-1 で署名した証明書を許可します。TLS プロトコルの 1.2 と 1.3、IKEv2 プロトコル、および SSH2 プロトコルが使用できます。RSA 鍵と Diffie-Hellman パラメーターは長さが 2048 ビット以上であれば許容されます。

FUTURE

将来の潜在的なポリシーをテストすることを目的とした、より厳格な将来を見据えたセキュリティレベル。このポリシーでは、DNSSec または HMAC としての SHA-1 の使用は許可されません。SHA2-224 および SHA3-224 ハッシュは拒否されます。128 ビット暗号は無効になります。CBC モードの暗号は、Kerberos を除き無効になります。TLS プロトコルの 1.2 と 1.3、IKEv2 プロトコル、および SSH2 プロトコルが使用できます。RSA 鍵と Diffie-Hellman パラメーターは、ビット長が 3072 以上だと許可されます。システムが公共のインターネット上で通信する場合、相互運用性の問題が発生する可能性があります。



重要

カスタマーポータル API の証明書が使用する暗号化鍵は **FUTURE** のシステム全体の暗号化ポリシーが定義する要件を満たさないため、現時点で **redhat-support-tool** ユーティリティは、このポリシーレベルでは機能しません。

この問題を回避するには、カスタマーポータル API への接続中に **DEFAULT** 暗号化ポリシーを使用します。

FIPS

FIPS 140 要件に準拠します。RHEL システムを FIPS モードに切り替える **fips-mode-setup** ツールは、このポリシーを内部的に使用します。**FIPS** ポリシーに切り替えても、FIPS 140 標準への準拠は保証されません。また、システムを FIPS モードに設定した後、すべての暗号キーを再生成する必要があります。多くのシナリオでは、これは不可能です。

また、RHEL はシステム全体のサブポリシー **FIPS:OSPP** を提供します。これには、Common Criteria (CC) 認証に必要な暗号化アルゴリズムに関する追加の制限が含まれています。このサブポ

リシーを設定すると、システムの相互運用性が低下します。たとえば、3072 ビットより短い RSA 鍵と DH 鍵、追加の SSH アルゴリズム、および複数の TLS グループを使用できません。また、**FIPS:OSPP** を設定すると、Red Hat コンテンツ配信ネットワーク (CDN) 構造への接続が防止されます。さらに、**FIPS:OSPP** を使用する IdM デプロイメントには Active Directory (AD) を統合できません。**FIPS:OSPP** を使用する RHEL ホストと AD ドメイン間の通信が機能しないか、一部の AD アカウントが認証できない可能性があります。



注記

FIPS:OSPP 暗号化サブポリシーを設定すると、システムが CC 非準拠になります。RHEL システムを CC 標準に準拠させる唯一の正しい方法は、**cc-config** パッケージで提供されているガイダンスに従うことです。認定済み RHEL バージョン、検証レポート、CC ガイドへのリンクのリストについては、Red Hat カスタマーポータル [Product compliance](#) ページの [Common Criteria](#) セクションを参照してください。

Red Hat は、**LEGACY** ポリシーを使用する場合を除き、すべてのライブラリーがセキュアなデフォルト値を提供するように、すべてのポリシーレベルを継続的に調整します。**LEGACY** プロファイルはセキュアなデフォルト値を提供しませんが、このプロファイルには、簡単に悪用できるアルゴリズムは含まれていません。このため、提供されたポリシーで有効なアルゴリズムのセットまたは許容可能な鍵サイズは、Red Hat Enterprise Linux の存続期間中に変更する可能性があります。

このような変更は、新しいセキュリティ標準や新しいセキュリティ調査を反映しています。Red Hat Enterprise Linux のライフサイクル全体にわたって特定のシステムとの相互運用性を確保する必要がある場合は、そのシステムと対話するコンポーネントのシステム全体の暗号化ポリシーからオプトアウトするか、カスタム暗号化ポリシーを使用して特定のアルゴリズムを再度有効にする必要があります。

ポリシーレベルで許可されていると記載されている特定のアルゴリズムと暗号は、アプリケーションがそれらをサポートしている場合にのみ使用できます。

表3.1 暗号化ポリシーで有効になっている暗号スイートとプロトコル

	LEGACY	DEFAULT	FIPS	FUTURE
IKEv1	いいえ	いいえ	いいえ	いいえ
3DES	いいえ	いいえ	いいえ	いいえ
RC4	いいえ	いいえ	いいえ	いいえ
DH	最低 2048 ビット	最低 2048 ビット	最低 2048 ビット	最低 3072 ビット
RSA	最低 2048 ビット	最低 2048 ビット	最低 2048 ビット	最低 3072 ビット
DSA	いいえ	いいえ	いいえ	いいえ
TLS v1.1 以前	いいえ	いいえ	いいえ	いいえ
TLS v1.2 以降	はい	はい	はい	はい

	LEGACY	DEFAULT	FIPS	FUTURE
デジタル署名および証明書の SHA-1	はい	いいえ	いいえ	いいえ
CBC モード暗号	はい	いいえ ^[a]	いいえ ^[b]	いいえ ^[c]
256 ビットより小さい鍵を持つ対称暗号	はい	はい	はい	いいえ
<p>[a] CBC 暗号は SSH で無効になります。</p> <p>[b] CBC 暗号は、Kerberos を除くすべてのプロトコルで無効になります。</p> <p>[c] CBC 暗号は、Kerberos を除くすべてのプロトコルで無効になります。</p>				

関連情報

- システム上の **crypto-policies(7)** および **update-crypto-policies(8)** man ページ
- [Product compliance](#) (Red Hat カスタマーポータル)

3.2. システム全体の暗号化ポリシーの変更

update-crypto-policies ツールを使用してシステムを再起動すると、システム全体の暗号化ポリシーを変更できます。

前提条件

- システムの root 権限がある。

手順

- オプション: 現在の暗号化ポリシーを表示します。

```
$ update-crypto-policies --show
DEFAULT
```

- 新しい暗号化ポリシーを設定します。

```
# update-crypto-policies --set <POLICY>
<POLICY>
```

<POLICY> は、設定するポリシーまたはサブポリシー (**FUTURE**、**LEGACY**、**FIPS:OSPP** など) に置き換えます。

- システムを再起動します。

```
# reboot
```

検証

- 現在の暗号化ポリシーを表示します。

```
$ update-crypto-policies --show  
<POLICY>
```

関連情報

- システム全体の暗号化ポリシーの詳細は、[システム全体の暗号化ポリシー](#) を参照してください。

3.3. システム全体の暗号化ポリシーを以前のリリースと互換性のあるモードに切り替える

Red Hat Enterprise Linux 9 におけるデフォルトのシステム全体の暗号化ポリシーでは、現在は古くて安全ではないプロトコルは許可されません。Red Hat Enterprise Linux 6 およびそれ以前のリリースとの互換性が必要な場合には、安全でない **LEGACY** ポリシーレベルを利用できます。



警告

LEGACY ポリシーレベルに設定すると、システムおよびアプリケーションの安全性が低下します。

手順

- システム全体の暗号化ポリシーを **LEGACY** レベルに切り替えるには、**root** で以下のコマンドを実行します。

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

関連情報

- 利用可能な暗号化ポリシーレベルのリストは、システム上の **update-crypto-policies(8)** man ページを参照してください。
- カスタム暗号化ポリシーの定義は、システム上の **update-crypto-policies(8)** man ページの **Custom Policies** セクションと、**crypto-policies(7)** man ページの **Crypto Policy Definition Format** セクションを参照してください。

3.4. SHA-1 を再度有効に

署名を作成および検証するための SHA-1 アルゴリズムの使用は、**DEFAULT** 暗号化ポリシーで制限されています。シナリオで既存またはサードパーティーの暗号署名を検証するために SHA-1 を使用する必要がある場合は、RHEL 9 がデフォルトで提供する **SHA1** サブポリシーを適用することで有効にできます。システムのセキュリティが弱くなることに注意してください。

前提条件

- このシステムは、**DEFAULT** システム全体の暗号化ポリシーを使用します。

手順

- SHA1** サブポリシーを **DEFAULT** 暗号化ポリシーに適用します。

```
# update-crypto-policies --set DEFAULT:SHA1
Setting system policy to DEFAULT:SHA1
Note: System-wide crypto policies are applied on application start-up.
It is recommended to restart the system for the change of policies
to fully take place.
```

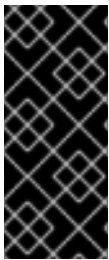
- システムを再起動します。

```
# reboot
```

検証

- 現在の暗号化ポリシーを表示します。

```
# update-crypto-policies --show
DEFAULT:SHA1
```



重要

update-crypto-policies --set LEGACY コマンドを使用して **LEGACY** 暗号化ポリシーに切り替えると、署名に対して SHA-1 も有効になります。ただし、**LEGACY** 暗号化ポリシーは、他の弱い暗号化アルゴリズムも有効にすることで、システムをはるかに脆弱にします。この回避策は、SHA-1 署名以外のレガシー暗号化アルゴリズムを有効にする必要があるシナリオでのみ使用してください。

関連情報

- [SSH from RHEL 9 to RHEL 6 systems does not work](#) (Red Hat ナレッジベース)
- [Packages signed with SHA-1 cannot be installed or upgraded](#) (Red Hat ナレッジベース)

3.5. WEB コンソールでシステム全体の暗号化ポリシーを設定する

RHEL Web コンソールインターフェイスで、システム全体の暗号化ポリシーとサブポリシーのいずれかを直接設定できます。4つの事前定義されたシステム全体の暗号化ポリシーに加え、グラフィカルインターフェイスを介して、次のポリシーとサブポリシーの組み合わせを適用することもできます。

DEFAULT:SHA1

SHA-1 アルゴリズムが有効になっている **DEFAULT** ポリシー。

LEGACY:AD-SUPPORT

Active Directory サービスの相互運用性を向上させる、セキュリティの低い設定を含む **LEGACY** ポリシー。

FIPS:OSPP

Common Criteria for Information Technology Security Evaluation 標準によって要求される追加の制限を含む **FIPS** ポリシー。



警告

システム全体のサブポリシー **FIPS:OSPP** には、Common Criteria (CC) 認定に必要な暗号化アルゴリズムに関する追加の制限が含まれています。そのため、このサブポリシーを設定すると、システムの相互運用性が低下します。たとえば、3072 ビットより短い RSA 鍵と DH 鍵、追加の SSH アルゴリズム、および複数の TLS グループを使用できません。また、**FIPS:OSPP** を設定すると、Red Hat コンテンツ配信ネットワーク (CDN) 構造への接続が防止されます。さらに、**FIPS:OSPP** を使用する IdM デプロイメントには Active Directory (AD) を統合できません。**FIPS:OSPP** を使用する RHEL ホストと AD ドメイン間の通信が機能しないか、一部の AD アカウントが認証できない可能性があります。

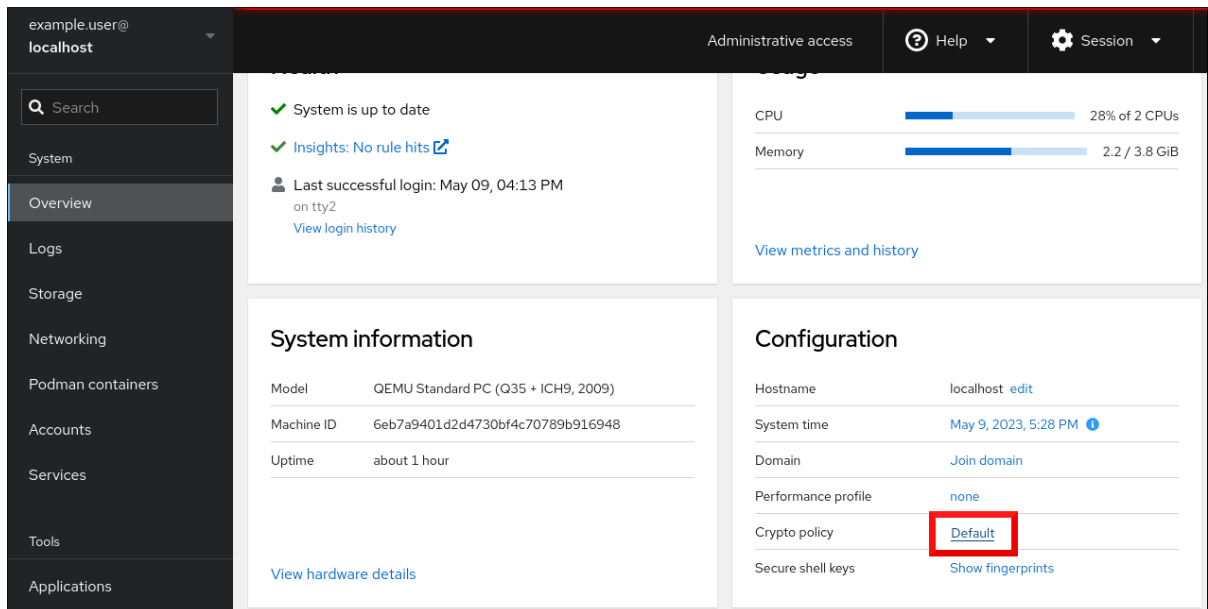
FIPS:OSPP 暗号化サブポリシーを設定すると、システムが **CC 非準拠**になることに注意してください。RHEL システムを CC 標準に準拠させる唯一の正しい方法は、**cc-config** パッケージで提供されているガイダンスに従うことです。認定された RHEL バージョン、検証レポート、および [National Information Assurance Partnership \(NIAP\)](#) の Web サイトでホストされる CC ガイドへのリンクのリストは、Red Hat カスタマーポータルページの [Product compliance](#) の [Common Criteria](#) セクションを参照してください。

前提条件

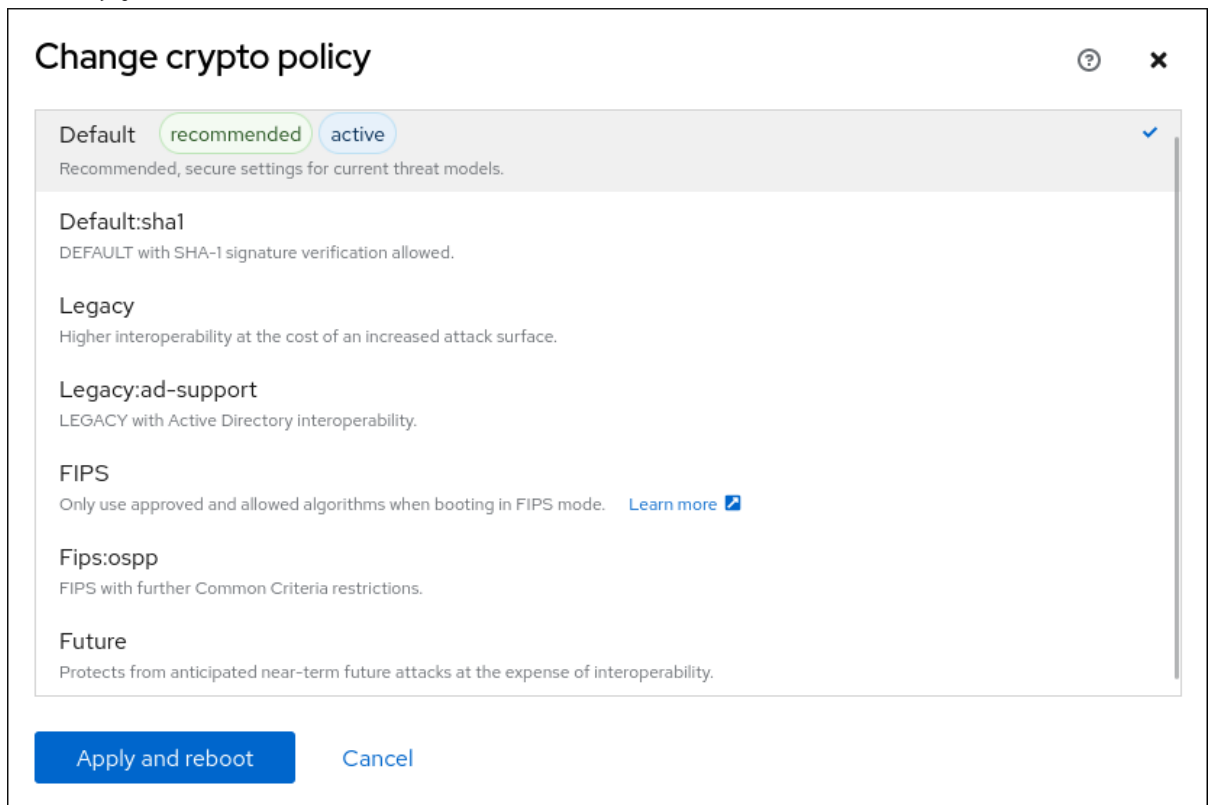
- RHEL 9 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- **root** 特権、または **sudo** を使用して管理コマンドを入力する権限がある。

手順

1. RHEL 9 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. **Overview** ページの **Configuration** カードで、**Crypto policy** の横にある現在のポリシー値をクリックします。



3. **Change crypto policy** ダイアログウィンドウで、システムで使用を開始するポリシーをクリックします。



4. **Apply and reboot** ボタンをクリックします。

検証

- 再起動後、Web コンソールに再度ログインし、**暗号化ポリシー** の値が選択したものと一致していることを確認します。
あるいは、**update-crypto-policies --show** コマンドを入力して、現在のシステム全体の暗号化ポリシーをターミナルに表示することもできます。

3.6. システム全体の暗号化ポリシーからアプリケーションを除外する

アプリケーションで使用される暗号化関連の設定をカスタマイズする必要がある場合は、サポートされる暗号スイートとプロトコルをアプリケーションで直接設定することが推奨されます。

/etc/crypto-policies/back-ends ディレクトリーからアプリケーション関連のシンボリックリンクを削除することもできます。カスタマイズした暗号化設定に置き換えることもできます。この設定により、除外されたバックエンドを使用するアプリケーションに対するシステム全体の暗号化ポリシーが使用できなくなります。この修正は、Red Hat ではサポートされていません。

3.6.1. システム全体の暗号化ポリシーをオプトアウトする例

wget

wget ネットワークダウンローダーで使用される暗号化設定をカスタマイズするには、**--secure-protocol** オプションおよび **--ciphers** オプションを使用します。以下に例を示します。

```
$ wget --secure-protocol=TLSv1_1 --ciphers="SECURE128" https://example.com
```

詳細は、**wget(1)** man ページの HTTPS (SSL/TLS) Options のセクションを参照してください。

curl

curl ツールで使用する暗号を指定するには、**--ciphers** オプションを使用して、その値に、コロンで区切った暗号化のリストを指定します。以下に例を示します。

```
$ curl https://example.com --ciphers '@SECLEVEL=0:DES-CBC3-SHA:RSA-DES-CBC3-SHA'
```

詳細は、**curl(1)** の man ページを参照してください。

Firefox

Web ブラウザーの **Firefox** でシステム全体の暗号化ポリシーをオプトアウトできない場合でも、Firefox の設定エディターで、対応している暗号と TLS バージョンをさらに詳細に制限できます。アドレスバーに **about:config** と入力し、必要に応じて **security.tls.version.min** の値を変更します。たとえば、**security.tls.version.min** を **1** に設定すると、最低でも TLS 1.0 が必要になり、**security.tls.version.min 2** が TLS 1.1 になります。

OpenSSH

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**)。

詳細は、**ssh_config(5)** の man ページを参照してください。

Libreswan

詳細は、[Securing networks](#) の [Configuring IPsec connections that opt out of the system-wide crypto policies](#) を参照してください。

関連情報

- システム上の **update-crypto-policies(8)** man ページ

3.7. 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

この手順を使用して、有効な暗号化アルゴリズムまたはプロトコルのセットを調整します。

既存のシステム全体の暗号化ポリシーの上にカスタムサブポリシーを適用するか、そのようなポリシーを最初から定義することができます。

スコープが設定されたポリシーの概念により、バックエンドごとに異なるアルゴリズムセットを有効にできます。各設定ディレクティブは、特定のプロトコル、ライブラリー、またはサービスに限定できます。

また、ディレクティブでは、ワイルドカードを使用して複数の値を指定する場合にアスタリスクを使用できます。

/etc/crypto-policies/state/CURRENT.pol ファイルには、ワイルドカードデプロイメント後に現在適用されているシステム全体の暗号化ポリシーのすべての設定がリスト表示されます。暗号化ポリシーをより厳密にするには、**/usr/share/crypto-policies/policies/FUTURE.pol** ファイルにリストされている値を使用することを検討してください。

サブポリシーの例は、**/usr/share/crypto-policies/policies/modules/** ディレクトリーにあります。このディレクトリーのサブポリシーファイルには、コメントアウトされた行に説明が含まれています。

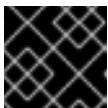
手順

1. **/etc/crypto-policies/policies/modules/** ディレクトリーをチェックアウトします。

```
# cd /etc/crypto-policies/policies/modules/
```

2. 調整用のサブポリシーを作成します。次に例を示します。

```
# touch MYCRYPTO-1.pmod
# touch SCOPES-AND-WILDCARDS.pmod
```



重要

ポリシーモジュールのファイル名には大文字を使用します。

3. 任意のテキストエディターでポリシーモジュールを開き、システム全体の暗号化ポリシーを変更するオプションを挿入します。次に例を示します。

```
# vi MYCRYPTO-1.pmod
```

```
min_rsa_size = 3072
hash = SHA2-384 SHA2-512 SHA3-384 SHA3-512
```

```
# vi SCOPES-AND-WILDCARDS.pmod
```

```
# Disable the AES-128 cipher, all modes
cipher = -AES-128-*
```

```
# Disable CHACHA20-POLY1305 for the TLS protocol (OpenSSL, GnuTLS, NSS, and
OpenJDK)
cipher@TLS = -CHACHA20-POLY1305
```

```
# Allow using the FFDHE-1024 group with the SSH protocol (libssh and OpenSSH)
group@SSH = FFDHE-1024+
```

```
# Disable all CBC mode ciphers for the SSH protocol (libssh and OpenSSH)
cipher@SSH = -*-CBC
```

```
# Allow the AES-256-CBC cipher in applications using libssh
cipher@libssh = AES-256-CBC+
```

4. 変更をモジュールファイルに保存します。
5. ポリシーの調整を、システム全体の暗号化ポリシーレベル **DEFAULT** に適用します。

```
# update-crypto-policies --set DEFAULT:MYCRYPTO-1:SCOPES-AND-WILDCARDS
```

6. 暗号化設定を実行中のサービスやアプリケーションで有効にするには、システムを再起動します。

```
# reboot
```

検証

- **/etc/crypto-policies/state/CURRENT.pol** ファイルに変更が含まれていることを確認します。以下に例を示します。

```
$ cat /etc/crypto-policies/state/CURRENT.pol | grep rsa_size
min_rsa_size = 3072
```

関連情報

- システム上の **update-crypto-policies(8)** man ページの **Custom Policies** セクション
- システム上の **crypto-policies(7)** man ページの **Crypto Policy Definition Format** セクション
- Red Hat ブログ記事 [How to customize crypto policies in RHEL 8.2](#)

3.8. システム全体のカスタム暗号化ポリシーの作成および設定

完全なポリシーファイルを作成して使用することで、システム全体の暗号化ポリシーを特定の状況向けにカスタマイズできます。

手順

1. カスタマイズのポリシーファイルを作成します。

```
# cd /etc/crypto-policies/policies/
# touch MYPOLICY.pol
```

または、定義されている 4 つのポリシーレベルのいずれかをコピーします。

```
# cp /usr/share/crypto-policies/policies/DEFAULT.pol /etc/crypto-
policies/policies/MYPOLICY.pol
```

2. 必要に応じて、テキストエディターでファイルを編集します。以下のようにしてカスタム暗号化ポリシーを使用します。

```
# vi /etc/crypto-policies/policies/MYPOLICY.pol
```

3. システム全体の暗号化ポリシーをカスタムレベルに切り替えます。

```
# update-crypto-policies --set MYPOLICY
```

4. 暗号化設定を実行中のサービスやアプリケーションで有効にするには、システムを再起動します。

```
# reboot
```

関連情報

- システム上の **update-crypto-policies(8)** man ページの **Custom Policies** セクションと **crypto-policies(7)** man ページの **Crypto Policy Definition Format** セクション
- Red Hat ブログ記事 [How to customize crypto policies in RHEL](#)

3.9. CRYPTO_POLICIES RHEL システムロールを使用した FUTURE 暗号化ポリシーによるセキュリティの強化

crypto_policies RHEL システムロールを使用して、管理対象ノードで **FUTURE** ポリシーを設定できます。このポリシーは、たとえば次のことを実現するのに役立ちます。

- 将来の新たな脅威への対応: 計算能力の向上を予測します。
- セキュリティの強化: 強力な暗号化標準により、より長い鍵長とよりセキュアなアルゴリズムを必須にします。
- 高度なセキュリティ標準への準拠: 医療、通信、金融などの分野ではデータの機密性が高く、強力な暗号化を利用できることが重要です。

通常、**FUTURE** は、機密性の高いデータを扱う環境、将来の規制に備える環境、長期的なセキュリティストラテジーを採用する環境に適しています。



警告

レガシーのシステムやソフトウェアでは、**FUTURE** ポリシーによって強制される、より最新しく厳格なアルゴリズムやプロトコルをサポートする必要はありません。たとえば、古いシステムでは TLS 1.3 以上の鍵サイズがサポートされていない可能性があります。これにより互換性の問題が発生する可能性があります。

また、強力なアルゴリズムを使用すると、通常、計算負荷が増加し、システムのパフォーマンスに悪影響が及ぶ可能性があります。

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: Configure cryptographic policies
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure the FUTURE cryptographic security policy on the managed node
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.crypto_policies
      vars:
        - crypto_policies_policy: FUTURE
        - crypto_policies_reboot_ok: true
```

サンプル Playbook で指定されている設定は次のとおりです。

crypto_policies_policy: FUTURE

管理対象ノードで必要な暗号化ポリシー (**FUTURE**) を設定します。これは、基本ポリシー、またはいくつかのサブポリシーを含む基本ポリシーのどちらかです。指定した基本ポリシーとサブポリシーが、管理対象ノードで使用可能である必要があります。デフォルト値は **null** です。つまり、設定は変更されず、**crypto_policies** RHEL システムロールは Ansible fact のみを収集します。

crypto_policies_reboot_ok: true

すべてのサービスとアプリケーションが新しい設定ファイルを読み取るように、暗号化ポリシーの変更後にシステムを再起動します。デフォルト値は **false** です。

Playbook で使用されるすべての変数の詳細は、コントロールノードの `/usr/share/ansible/roles/rhel-system-roles.crypto_policies/README.md` ファイルを参照してください。

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
```



警告

システム全体のサブポリシー **FIPS:OSPP** には、Common Criteria (CC) 認定に必要な暗号化アルゴリズムに関する追加の制限が含まれています。そのため、このサブポリシーを設定すると、システムの相互運用性が低下します。たとえば、3072 ビットより短い RSA 鍵と DH 鍵、追加の SSH アルゴリズム、および複数の TLS グループを使用できません。また、**FIPS:OSPP** を設定すると、Red Hat コンテンツ配信ネットワーク (CDN) 構造への接続が防止されます。さらに、**FIPS:OSPP** を使用する IdM デプロイメントには Active Directory (AD) を統合できません。**FIPS:OSPP** を使用する RHEL ホストと AD ドメイン間の通信が機能しないか、一部の AD アカウントが認証できない可能性があります。

FIPS:OSPP 暗号化サブポリシーを設定すると、システムが **CC 非準拠**になることに注意してください。RHEL システムを CC 標準に準拠させる唯一の正しい方法は、**cc-config** パッケージで提供されているガイダンスに従うことです。認定された RHEL バージョン、検証レポート、および [National Information Assurance Partnership \(NIAP\)](#) の Web サイトでホストされる CC ガイドへのリンクのリストは、Red Hat カスタマーポータルページの [Product compliance](#) の [Common Criteria](#) セクションを参照してください。

検証

1. コントロールノードで、たとえば **verify_playbook.yml** という名前の別の Playbook を作成します。

```
---
- name: Verification
  hosts: managed-node-01.example.com
  tasks:
    - name: Verify active cryptographic policy
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.crypto_policies
    - name: Display the currently active cryptographic policy
      ansible.builtin.debug:
        var: crypto_policies_active
```

サンプル Playbook で指定されている設定は次のとおりです。

crypto_policies_active

crypto_policies_policy 変数で受け入れられる形式の現在アクティブなポリシー名が含まれているエクスポートされた Ansible fact。

2. Playbook の構文を検証します。

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

3. Playbook を実行します。

```
$ ansible-playbook ~/verify_playbook.yml
TASK [debug] *****
ok: [host] => {
  "crypto_policies_active": "FUTURE"
}
```

crypto_policies_active 変数は、管理対象ノード上のアクティブなポリシーを示します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.crypto_policies/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/crypto_policies/** ディレクトリー
- **update-crypto-policies(8)** および **crypto-policies(7)** man ページ

第4章 PKCS #11 で暗号化ハードウェアを使用するようにアプリケーションを設定

スマートカードや、エンドユーザー認証用の暗号化トークン、サーバーアプリケーション用のハードウェアセキュリティモジュール (HSM) など、専用の暗号化デバイスで秘密情報の一部を分離することで、セキュリティ層が追加されます。RHEL では、PKCS #11 API を使用した暗号化ハードウェアへの対応がアプリケーション間で統一され、暗号ハードウェアでの秘密の分離が複雑なタスクではなくなりました。

4.1. PKCS #11 による暗号化ハードウェアへの対応

Public-Key Cryptography Standard (PKCS) #11 は、暗号化情報を保持し、暗号化機能を実行する暗号化デバイスへのアプリケーションプログラミングインターフェイス (API) を定義します。

PKCS #11 では、各ハードウェアまたはソフトウェアデバイスを統一された方法でアプリケーションに提示するオブジェクトである **暗号化トークン** が導入されています。したがって、アプリケーションは、通常はユーザーによって使用されるスマートカードなどのデバイスや、通常はコンピューターによって使用されるハードウェアセキュリティモジュールを PKCS #11 暗号化トークンとして認識します。

PKCS #11 トークンには、証明書、データオブジェクト、公開鍵、秘密鍵、または秘密鍵を含むさまざまなオブジェクトタイプを保存できます。これらのオブジェクトは、PKCS #11 Uniform Resource Identifier (URI) スキームを通じて一意に識別できます。

PKCS #11 の URI は、オブジェクト属性に従って、PKCS #11 モジュールで特定のオブジェクトを識別する標準的な方法です。これにより、URI の形式で、すべてのライブラリーとアプリケーションを同じ設定文字列で設定できます。

RHEL では、デフォルトでスマートカード用に OpenSC PKCS #11 ドライバーが提供されています。ただし、ハードウェアトークンと HSM には、システムにカウンターパートを持たない独自の PKCS #11 モジュールがあります。この PKCS #11 モジュールは **p11-kit** ツールで登録できます。これは、システムの登録済みスマートカードドライバにおけるラッパーとして機能します。

システムで独自の PKCS #11 モジュールを有効にするには、新しいテキストファイルを `/etc/pkcs11/modules/` ディレクトリーに追加します。

`/etc/pkcs11/modules/` ディレクトリーに新しいテキストファイルを作成すると、独自の PKCS #11 モジュールをシステムに追加できます。たとえば、**p11-kit** の OpenSC 設定ファイルは、以下のようになります。

```
$ cat /usr/share/p11-kit/modules/opensc.module  
module: opensc-pkcs11.so
```

関連情報

- [PKCS #11 の URI スキーム](#)
- [Controlling access to smart cards](#)

4.2. 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

- 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
```

- 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>
```

- 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] $
```

- オプション: **~/.ssh/config** ファイルで同じ URI 文字列を使用して、設定を永続的にすることができます。

```
$ 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

4.3. スマートカード上の証明書を使用して認証するアプリケーションの設定

アプリケーションでスマートカードを使用して認証することにより、セキュリティが強化され、自動化が簡素化される場合があります。次の方法を使用して、Public Key Cryptography Standard (PKCS) #11 URI をアプリケーションに統合できます。

- Firefox** Web ブラウザーは、**p11-kit-proxy** PKCS #11 モジュールを自動的にロードします。つまり、システムで対応しているすべてのスマートカードが自動的に検出されます。TLS クライアント認証を使用する場合、追加のセットアップは必要ありません。サーバーが要求したときにスマートカードの鍵と証明書が自動的に使用されます。
- アプリケーションが **GnuTLS** または **NSS** ライブラリーを使用している場合、PKCS #11 URI はすでにサポートされています。また、**OpenSSL** ライブラリーに依存するアプリケーションは、**openssl-pkcs11** パッケージによって提供される **pkcs11** エンジンを通じて、スマートカードを含む暗号化ハードウェアモジュールにアクセスできます。
- スマートカード上の秘密鍵を操作する必要があり、**NSS**、**GnuTLS**、**OpenSSL** を使用しないアプリケーションは、特定の PKCS #11 モジュールの PKCS #11 API を使用するのではなく、**p11-kit** API を直接使用して、スマートカードを含む暗号化ハードウェアモジュールを操作できます。
- wget** ネットワークダウンローダーを使用すると、ローカルに保存された秘密鍵と証明書へのパスの代わりに PKCS #11 URI を指定できます。これにより、安全に保管された秘密鍵と証明書を必要とするタスクのスクリプトの作成が簡素化される可能性があります。以下に例を示します。

```
$ wget --private-key 'pkcs11:token=softhsm;id=%01;type=private?pin-value=111111' --certificate 'pkcs11:token=softhsm;id=%01;type=cert' https://example.com/
```

- また、**curl** ツールを使用する場合は、PKCS #11 URI を指定することもできます。

```
$ curl --key 'pkcs11:token=softhsm;id=%01;type=private?pin-value=111111' --cert 'pkcs11:token=softhsm;id=%01;type=cert' https://example.com/
```



注記

PIN は、スマートカードに保存されている鍵へのアクセスを制御するセキュリティ対策であり、設定ファイルにはプレーンテキスト形式の PIN が含まれているため、攻撃者が PIN を読み取れないように追加の保護を検討してください。たとえば、**pin-source** 属性を使用して、ファイルから PIN を読み取るための **file:** URI を指定できます。詳細は、[RFC 7512: PKCS #11 URI Scheme Query Attribute Semantics](#) を参照してください。コマンドパスを **pin-source** 属性の値として使用することには対応していないことに注意してください。

関連情報

- システム上の **curl(1)**、**wget(1)**、および **p11-kit(8)** man ページ

4.4. APACHE で秘密鍵を保護する HSM の使用

Apache HTTP サーバーは、ハードウェアセキュリティモジュール (HSM) に保存されている秘密鍵と連携できます。これにより、鍵の漏えいや中間者攻撃を防ぐことができます。通常、これを行うには、ビジーなサーバーに高パフォーマンスの HSM が必要になります。

HTTPS プロトコルの形式でセキュアな通信を行うために、**Apache** HTTP サーバー (**httpd**) は OpenSSL ライブラリーを使用します。OpenSSL は、PKCS #11 にネイティブに対応しません。HSM を使用するには、エンジンインターフェイスを介して PKCS #11 モジュールへのアクセスを提供する **openssl-pkcs11** パッケージをインストールする必要があります。通常のファイル名ではなく PKCS #11 の URI を使用すると、**/etc/httpd/conf.d/ssl.conf** 設定ファイルでサーバーの鍵と証明書を指定できます。以下に例を示します。

```
SSLCertificateFile "pkcs11:id=%01;token=softhsm;type=cert"
SSLCertificateKeyFile "pkcs11:id=%01;token=softhsm;type=private?pin-value=111111"
```

httpd-manual パッケージをインストールして、TLS 設定を含む **Apache** HTTP サーバー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。

4.5. NGINX で秘密鍵を保護する HSM の使用

Nginx HTTP サーバーは、ハードウェアセキュリティモジュール (HSM) に保存されている秘密鍵と連携できます。これにより、鍵の漏えいや中間者攻撃を防ぐことができます。通常、これを行うには、ビジーなサーバーに高パフォーマンスの HSM が必要になります。

Nginx は暗号化操作に OpenSSL を使用するため、PKCS #11 への対応は **openssl-pkcs11** エンジンを通じて行う必要があります。**Nginx** は現在、HSM からの秘密鍵の読み込みのみに対応します。また、証明書は通常のファイルとして個別に提供する必要があります。**/etc/nginx/nginx.conf** 設定ファイルの **server** セクションで **ssl_certificate** オプションおよび **ssl_certificate_key** オプションを変更します。

```
ssl_certificate    /path/to/cert.pem
ssl_certificate_key "engine:pkcs11:pkcs11:token=softhsm;id=%01;type=private?pin-value=111111";
```

Nginx 設定ファイルの PKCS #11 URI に接頭辞 **engine:pkcs11:** が必要なことに注意してください。これは、他の **pkcs11** 接頭辞がエンジン名を参照するためです。

4.6. 関連情報

- システム上の **pkcs11.conf(5)** man ページ

第5章 POLKIT を使用したスマートカードへのアクセスの制御

PIN、PIN パッド、バイオメトリックなどのスマートカードに組み込まれたメカニズムでは防ぐことができない脅威に対処するため、およびより詳細な制御のために、RHEL は **polkit** フレームワークを使用してスマートカードへのアクセス制御を制御します。

システム管理者は、非特権ユーザーや非ローカルユーザー、サービスに対するスマートカードアクセスなど、特定のシナリオに合わせて **polkit** を設定できます。

5.1. POLKIT を介したスマートカードアクセス制御

PC/SC (Personal Computer/Smart Card) プロトコルは、スマートカードとそのリーダーをコンピューティングシステムに統合するための標準を指定します。RHEL では、**pcsc-lite** パッケージが、PC/SC の API を使用するスマートカードにアクセスするミドルウェアを提供します。このパッケージの一部である **pcscd** (PC/SC スマートカード) デーモンにより、システムが PC/SC プロトコルを使用してスマートカードにアクセスできるようになります。

PIN、PIN パッド、バイオメトリックなどのスマートカードに組み込まれたアクセス制御メカニズムは、考えられるすべての脅威をカバーするものではないため、RHEL は、より強力なアクセス制御に **polkit** フレームワークを使用します。**polkit** 認可マネージャーは、特権操作へのアクセスを許可できます。ディスクへのアクセスを許可することに加えて、**polkit** を使用して、スマートカードのセキュリティを保護するポリシーを指定することもできます。たとえば、スマートカードで操作を実行できるユーザーを定義できます。

pcsc-lite パッケージをインストールし、**pcscd** デーモンを起動すると、システムは、**/usr/share/polkit-1/actions/** ディレクトリーで定義されているポリシーを強制します。システム全体のデフォルトのポリシーは、**/usr/share/polkit-1/actions/org.debian.pcsc-lite.policy** ファイルにあります。Polkit ポリシーファイルは XML 形式を使用します。構文は、システム上の **polkit(8)** man ページで説明されています。

polkitd は、**/etc/polkit-1/rules.d/** ディレクトリーおよび **/usr/share/polkit-1/rules.d/** ディレクトリーで、これらのディレクトリーに保存されているルールファイルの変更を監視します。ファイルには、JavaScript 形式の認可ルールが含まれています。システム管理者は、両方のディレクトリーにカスタムルールファイルを追加し、**polkitd** がファイル名に基づいてアルファベット順に読み込むことができます。2つのファイルが同じ名前である場合は、最初に **/etc/polkit-1/rules.d/** 内のファイルが読み込まれます。

System Security Services Daemon (SSSD) が **root** として実行されていないときにスマートカードのサポートを有効にする必要がある場合は、**sssd-polkit-rules** パッケージをインストールする必要があります。このパッケージは、SSSD と **polkit** の統合を提供します。

関連情報

- システム上の **polkit(8)**、**polkitd(8)**、**pcscd(8)** man ページ

5.2. PC/SC および POLKIT に関連する問題のトラブルシューティング

pcsc-lite パッケージをインストールし、**pcscd** デーモンを起動した後に自動的に強制される Polkit ポリシーは、ユーザーがスマートカードと直接対話しない場合でも、ユーザーのセッションで認証を求められることがあります。GNOME では、以下のエラーメッセージが表示されます。

Authentication is required to access the PC/SC daemon

opensc などのスマートカードに関連する他のパッケージをインストールする場合は、システムが依存関係として **pcsc-lite** パッケージをインストールすることに注意してください。

スマートカードとの相互作用が必要なく、PC/SC デモンの認可要求が表示されないようにする場合は、**pcsc-lite** パッケージを削除できます。必要なパッケージを最小限にとどめることが、セキュリティー上の推奨事項です。

スマートカードを使用する場合は、**/usr/share/polkit-1/actions/org.debian.pcsc-lite.policy** 時に、システムが提供するポリシーファイルのルールを確認して、トラブルシューティングを開始します。カスタムルールファイルは、**/etc/polkit-1/rules.d/** ディレクトリーのポリシー (**03-allow-pcscd.rules** など) に追加できます。ルールファイルは JavaScript 構文を使用し、ポリシーファイルは XML 形式であることに注意してください。

システムに表示される認可要求を理解するには、以下の例のように Journal ログを確認します。

```
$ journalctl -b | grep pcsc
...
Process 3087 (user: 1001) is NOT authorized for action: access_pcsc
...
```

以前のログエントリーは、ユーザーがポリシーによるアクションを実行する権限を持っていないことを示しています。この拒否を解決するには、対応するルールを **/etc/polkit-1/rules.d/** に追加します。

polkitd ユニットに関連するログエントリーも検索できます。以下に例を示します。

```
$ journalctl -u polkit
...
polkitd[NNN]: Error compiling script /etc/polkit-1/rules.d/00-debug-pcscd.rules
...
polkitd[NNN]: Operator of unix-session:c2 FAILED to authenticate to gain authorization for action
org.debian.pcsc-lite.access_pcsc for unix-process:4800:14441 [/usr/libexec/gsd-smartcard] (owned
by unix-user:group)
...
```

この出力では、最初のエントリーがルールファイルに構文エラーが含まれていることを示しています。2 番目のエントリーは、ユーザーが **pcscd** へのアクセスを取得できなかったことを示しています。

また、短いスクリプトを使用して、PC/SC プロトコルを使用するすべてのアプリケーションをリスト表示することもできます。**pcsc-apps.sh** などの実行ファイルを作成し、以下のコードを挿入します。

```
#!/bin/bash

cd /proc

for p in [0-9]*
do
  if grep libpcsc-lite.so.1.0.0 $p/maps &> /dev/null
  then
    echo -n "process: "
    cat $p/cmdline
    echo " ($p)"
  fi
done
```

root でスクリプトを実行します。


```
# ./pcsc-apps.sh
process: /usr/libexec/gsd-smartcard (3048)
enable-sync --auto-ssl-client-auth --enable-crashpad (4828)
...
```

関連情報

- man ページの **journalctl**、**polkit(8)**、**polkitd(8)**、および **pcscd(8)**

5.3. PC/SC への POLKIT 認可の詳細情報の表示

デフォルト設定では、**polkit** 認可フレームワークは、限られた情報のみをジャーナルログに送信します。新しいルールを追加することで、PC/SC プロトコル関連の **polkit** ログエントリーを拡張できます。

前提条件

- システムに **pcsc-lite** パッケージをインストールしている。
- pcscd** デーモンが実行中である。

手順

1. **/etc/polkit-1/rules.d/** ディレクトリーに新規ファイルを作成します。

```
# touch /etc/polkit-1/rules.d/00-test.rules
```

2. 選択したエディターでファイルを編集します。以下に例を示します。

```
# vi /etc/polkit-1/rules.d/00-test.rules
```

3. 以下の行を挿入します。

```
polkit.addRule(function(action, subject) {
  if (action.id == "org.debian.pcsc-lite.access_pcsc" ||
      action.id == "org.debian.pcsc-lite.access_card") {
    polkit.log("action=" + action);
    polkit.log("subject=" + subject);
  }
});
```

ファイルを保存して、エディターを終了します。

4. **pcscd** サービスおよび **polkit** サービスを再起動します。

```
# systemctl restart pcscd.service pcscd.socket polkit.service
```

検証

1. **pcscd** の認可リクエストを作成します。たとえば、Firefox の Web ブラウザーを開くか、**opensc** が提供する **pkcs11-tool -L** を使用します。

2. 拡張ログエントリーを表示します。以下に例を示します。

```
# journalctl -u polkit --since "1 hour ago"
polkitd[1224]: <no filename>:4: action=[Action id='org.debian.pcsc-lite.access_pcsc']
polkitd[1224]: <no filename>:5: subject=[Subject pid=2020481 user=user'
groups=user,wheel,mock,wireshark seat=null session=null local=true active=true]
```

関連情報

- [man ページの **polkit\(8\)** および **polkitd\(8\)**](#)

5.4. 関連情報

- [スマートカードへのアクセスの制御](#) Red Hat ブログの記事

第6章 設定コンプライアンスおよび脆弱性スキャンの開始

コンプライアンス監査は、指定したオブジェクトが、コンプライアンスポリシーに指定されているすべてのルールに従っているかどうかを判断するプロセスです。コンプライアンスポリシーは、コンピューティング環境で使用される必要な設定を指定するセキュリティ専門家が定義します。これは多くの場合は、チェックリストの形式を取ります。

コンプライアンスポリシーは組織により大幅に異なることがあり、同一組織内でもシステムが異なるとポリシーが異なる可能性があります。ポリシーは、各システムの目的や、組織におけるシステム重要性により異なります。カスタマイズしたソフトウェア設定や導入の特徴によっても、カスタマイズしたポリシーのチェックリストが必要になってきます。

6.1. RHEL における設定コンプライアンスツール

次の設定コンプライアンスツールを使用すると、Red Hat Enterprise Linux で完全に自動化されたコンプライアンス監査を実行できます。このツールは SCAP (Security Content Automation Protocol) 規格に基づいており、コンプライアンスポリシーの自動化に合わせるように設計されています。

SCAP Workbench

scap-workbench グラフィカルユーティリティーは、単一のローカルシステムまたはリモートシステム上で設定および脆弱性スキャンを実行するように設計されています。これらのスキャンと評価に基づくセキュリティレポートの生成にも使用できます。

OpenSCAP

OpenSCAP ライブラリーは、付随する **oscap** コマンドラインユーティリティーとともに、ローカルシステムで設定スキャンと脆弱性スキャンを実行するように設計されています。これにより、設定コンプライアンスのコンテンツを検証し、スキャンおよび評価に基づいてレポートおよびガイドを生成します。



重要

OpenSCAP の使用中にメモリー消費の問題が発生する可能性があります。これにより、プログラムが途中で停止し、結果ファイルが生成されない可能性があります。詳細は、ナレッジベース記事 [OpenSCAP のメモリー消費の問題](#) を参照してください。

SCAP Security Guide (SSG)

scap-security-guide パッケージは、Linux システム用のセキュリティポリシーのコレクションを提供します。このガイダンスは、セキュリティ強化に関する実践的なアドバイスのカタログで構成されています (該当する場合は、法規制要件へのリンクが含まれます)。このプロジェクトは、一般的なポリシー要件と特定の実装ガイドラインとの間にあるギャップを埋めることを目的としています。

Script Check Engine (SCE)

SCAP プロトコルの拡張機能である SCE を使用すると、管理者は Bash、Python、Ruby などのスクリプト言語を使用してセキュリティコンテンツを作成できます。SCE 拡張機能は、**openscap-engine-sce** パッケージで提供されます。SCE 自体は SCAP 標準規格の一部ではありません。

複数のリモートシステムで自動コンプライアンス監査を実行する必要がある場合は、Red Hat Satellite 用の OpenSCAP ソリューションを利用できます。

関連情報

- システム上の **oscap(8)**、**scap-workbench(8)**、**scap-security-guide(8)** man ページ

- [Red Hat Security Demos: Creating Customized Security Policy Content to Automate Security Compliance](#)
- [Red Hat Security Demos: Defend Yourself with RHEL Security Technologies](#)
- [Red Hat Satellite のセキュリティコンプライアンスの管理](#)

6.2. 脆弱性スキャン

6.2.1. Red Hat Security Advisories OVAL フィード

Red Hat Enterprise Linux のセキュリティ監査機能は、標準規格セキュリティ設定共通化手順 (Security Content Automation Protocol (SCAP)) を基にしています。SCAP は、自動化された設定、脆弱性およびパッチの確認、技術的な制御コンプライアンスアクティビティー、およびセキュリティの測定に対応している多目的な仕様のフレームワークです。

SCAP 仕様は、スキャナーまたはポリシーエディターの実装が義務付けられていなくても、セキュリティコンテンツの形式がよく知られて標準化されているエコシステムを作成します。これにより、組織は、採用しているセキュリティベンダーの数に関係なく、セキュリティポリシー (SCAP コンテンツ) を構築するのは一度で済みます。

セキュリティ検査言語 OVAL (Open Vulnerability Assessment Language) は、SCAP に不可欠で最も古いコンポーネントです。その他のツールやカスタマイズされたスクリプトとは異なり、OVAL は、宣言型でリソースが必要な状態を記述します。OVAL コードは、スキャナーと呼ばれる OVAL インタープリターツールを使用して直接実行されることは決してありません。OVAL が宣言型であるため、評価されるシステムの状態が偶然修正されることはありません。

他のすべての SCAP コンポーネントと同様に、OVAL は XML に基づいています。SCAP 標準規格は、いくつかのドキュメント形式を定義します。この形式にはそれぞれ異なる種類の情報が記載され、異なる目的に使用されます。

[Red Hat 製品セキュリティ](#) を使用すると、Red Hat 製品をお使いのお客様に影響を及ぼすセキュリティ問題をすべて追跡して調査します。Red Hat カスタマーポータルで簡潔なパッチやセキュリティアドバイザリーを適時提供します。Red Hat は OVAL パッチ定義を作成してサポートし、マシンが判読可能なセキュリティアドバイザリーを提供します。

プラットフォーム、バージョン、およびその他の要因が異なるため、Red Hat 製品セキュリティによる脆弱性の重大度定性評価は、サードパーティーが提供する Common Vulnerability Scoring System (CVSS) のベースライン評価と完全に一致していません。したがって、サードパーティーが提供する定義ではなく、RHSA OVAL 定義を使用することが推奨されます。

各 [RHSA OVAL 定義](#) は完全なパッケージとして利用でき、新しいセキュリティアドバイザリーが Red Hat カスタマーポータルで利用可能になってから 1 時間以内に更新されます。

各 OVAL パッチ定義は、Red Hat セキュリティアドバイザリー (RHSA) と 1 対 1 にマッピングしています。RHSA には複数の脆弱性に対する修正が含まれるため、各脆弱性は、共通脆弱性識別子 (Common Vulnerabilities and Exposures (CVE)) 名ごとに表示され、公開バグデータベースの該当箇所へのリンクが示されます。

RHSA OVAL 定義は、システムにインストールされている RPM パッケージで脆弱なバージョンを確認するように設計されています。この定義は拡張でき、パッケージが脆弱な設定で使用されているかどうかを見つけるなど、さらに確認できるようにすることができます。この定義は、Red Hat が提供するソフトウェアおよび更新に対応するように設計されています。サードパーティーソフトウェアのパッチ状態を検出するには、追加の定義が必要です。



注記

[Red Hat Insights for Red Hat Enterprise Linux コンプライアンスサービス](#) は、IT セキュリティーおよびコンプライアンス管理者が Red Hat Enterprise Linux システムのセキュリティポリシーのコンプライアンスを評価、監視、およびレポートするのに役立ちます。また、コンプライアンスサービス UI 内で完全に SCAP セキュリティーポリシーを作成および管理することもできます。

関連情報

- [Red Hat and OVAL compatibility](#)
- [Red Hat and CVE compatibility](#)
- [製品セキュリティの概要](#) の [通知およびアドバイザリー](#)
- [Security Data Metrics](#)

6.2.2. システムの脆弱性のスキャン

oscap コマンドラインユーティリティーを使用すると、ローカルシステムのスキャン、設定コンプライアンスコンテンツの確認、ならびにスキャンおよび評価を基にしたレポートとガイドの生成が可能です。このユーティリティーは、OpenSCAP ライブラリーのフロントエンドとしてサービスを提供し、その機能を処理する SCAP コンテンツのタイプに基づいてモジュール (サブコマンド) にグループ化します。

前提条件

- **openscap-scanner** および **bzip2** パッケージがインストールされます。

手順

1. システムに最新 RHSA OVAL 定義をダウンロードします。

```
# wget -O - https://www.redhat.com/security/data/oval/v2/RHEL9/rhel-9.oval.xml.bz2 | bzip2 -  
-decompress > rhel-9.oval.xml
```

2. システムの脆弱性をスキャンし、**vulnerability.html** ファイルに結果を保存します。

```
# oscap oval eval --report vulnerability.html rhel-9.oval.xml
```

検証

- 結果をブラウザで確認します。以下に例を示します。

```
$ firefox vulnerability.html &
```

関連情報

- システム上の **oscap(8)** man ページ
- [Red Hat OVAL 定義](#)
- [OpenSCAP のメモリー消費の問題](#)

6.2.3. リモートシステムの脆弱性のスキャン

SSH プロトコル経由で **oscap-ssh** ツールを使用して、OpenSCAP スキャナーでリモートシステムの脆弱性をチェックできます。

前提条件

- **openscap-utils** および **bzip2** パッケージは、スキャンに使用するシステムにインストールされます。
- リモートシステムに **openscap-scanner** パッケージがインストールされている。
- リモートシステムで SSH サーバーが実行している。

手順

1. システムに最新 RHSA OVAL 定義をダウンロードします。

```
# wget -O - https://www.redhat.com/security/data/oval/v2/RHEL9/rhel-9.oval.xml.bz2 | bzip2 -  
-decompress > rhel-9.oval.xml
```

2. リモートシステムの脆弱性をスキャンし、結果をファイルに保存します。

```
# oscap-ssh <username>@<hostname> <port> oval eval --report <scan-report.html> rhel-  
9.oval.xml
```

以下のように置き換えます。

- **<username>@<hostname>** は、リモートシステムのユーザー名とホスト名に置き換えます。
- **<port>** は、リモートシステムにアクセスできるポート番号 (例: **22**) です。
- **<scan-report.html>** は、**oscap** がスキャン結果を保存するファイル名です。

関連情報

- **oscap-ssh(8)**
- [Red Hat OVAL 定義](#)
- [OpenSCAP のメモリー消費の問題](#)

6.3. 設定コンプライアンススキャン

6.3.1. RHEL の設定コンプライアンス

設定コンプライアンススキャンを使用して、特定の組織で定義されているベースラインに準拠できます。たとえば、米国政府と協力している場合は、システムを Operating System Protection Profile (OSPP) に準拠させ、支払い処理業者の場合は、システムを Payment Card Industry Data Security Standard (PCI-DSS) に準拠させなければならない場合があります。設定コンプライアンススキャンを実行して、システムセキュリティを強化することもできます。

Red Hat は、対象コンポーネント向けの Red Hat のベストプラクティスに従っているため、SCAP Security Guide パッケージで提供される Security Content Automation Protocol (SCAP) コンテンツに従うことを推奨します。

SCAP Security Guide パッケージは、SCAP 1.2 および SCAP 1.3 標準規格に準拠するコンテンツを提供します。**openscap scanner** ユーティリティーは、SCAP Security Guide パッケージで提供される SCAP 1.2 および SCAP 1.3 コンテンツの両方と互換性があります。

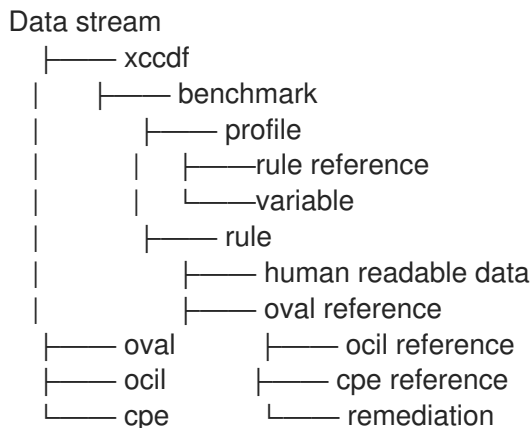


重要

設定コンプライアンススキャンを実行しても、システムが準拠しているとは限りません。

SCAP Security Guide スイートは、データストリームドキュメントの形式で、複数のプラットフォームのプロファイルを提供します。データストリームは、定義、ベンチマーク、プロファイル、および個々のルールが含まれるファイルです。各ルールでは、コンプライアンスの適用性と要件を指定します。RHEL は、セキュリティポリシーを扱う複数のプロファイルを提供します。Red Hat データストリームには、業界標準の他に、失敗したルールの修正に関する情報も含まれます。

コンプライアンススキャンリソースの構造



プロファイルは、OSPP、PCI-DSS、Health Insurance Portability and Accountability Act (HIPAA) などのセキュリティポリシーに基づく一連のルールです。これにより、セキュリティ標準規格に準拠するために、システムを自動で監査できます。

プロファイルを変更 (調整) して、パスワードの長さなどの特定のルールをカスタマイズできます。プロファイルの調整の詳細は、[SCAP Workbench を使用したセキュリティプロファイルのカスタマイズ](#)を参照してください。

6.3.2. OpenSCAP スキャン結果の例

OpenSCAP スキャンに適用されるデータストリームとプロファイル、およびシステムのさまざまなプロパティに応じて、各ルールから特定の結果が生成される場合があります。以下に考えられる結果とその意味の簡単な説明を示します。

Pass

スキャンでは、このルールとの競合が見つかりませんでした。

Fail

スキャンで、このルールとの競合が検出されました。

Not checked

OpenSCAP はこのルールの自動評価を実行しません。システムがこのルールに手動で準拠しているかどうかを確認してください。

Not applicable

このルールは、現在の設定には適用されません。

Not selected

このルールはプロファイルには含まれません。OpenSCAP はこのルールを評価せず、結果にこのようなルールは表示されません。

Error

スキャンでエラーが発生しました。詳細は、**--verbose DEVEL** オプションを指定して **oscap** コマンドで確認できます。[Red Hat カスタマーポータル](#) でサポートケースを作成するか、[Red Hat Jira の RHEL プロジェクト](#) でチケットを作成します。

Unknown

スキャンで予期しない状況が発生しました。詳細は、**--verbose DEVEL** オプションを指定して **oscap** コマンドを入力できます。[Red Hat カスタマーポータル](#) でサポートケースを作成するか、[Red Hat Jira の RHEL プロジェクト](#) でチケットを作成します。

6.3.3. 設定コンプライアンスのプロファイルの表示

スキャンまたは修復にプロファイルを使用することを決定する前に、**oscap info** サブコマンドを使用して、プロファイルを一覧表示し、詳細な説明を確認できます。

前提条件

- **openscap-scanner** パッケージおよび **scap-security-guide** パッケージがインストールされている。

手順

1. SCAP Security Guide プロジェクトが提供するセキュリティコンプライアンスプロファイルで利用可能なファイルをすべて表示します。

```
$ ls /usr/share/xml/scap/ssg/content/
ssg-rhel9-ds.xml
```

2. **oscap info** サブコマンドを使用して、選択したデータストリームに関する詳細情報を表示します。データストリームを含む XML ファイルは、名前に **-ds** 文字列で示されます。**Profiles** セクションでは、利用可能なプロファイルと、その ID のリストを確認できます。

```
$ oscap info /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
Profiles:
...
Title: Australian Cyber Security Centre (ACSC) Essential Eight
Id: xccdf_org.ssgproject.content_profile_e8
Title: Health Insurance Portability and Accountability Act (HIPAA)
Id: xccdf_org.ssgproject.content_profile_hipaa
Title: PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 9
Id: xccdf_org.ssgproject.content_profile_pci-dss
...
```

3. データストリームファイルからプロファイルを選択し、選択したプロファイルに関する追加情報を表示します。そのためには、**oscap info** に **--profile** オプションを指定した後に、直前のコ

マンドの出力で表示された ID の最後のセクションを指定します。たとえば、HIPAA プロファイルの ID は `xccdf_org.ssgproject.content_profile_hipaa` で、`--profile` オプションの値は `hipaa` です。

```
$ oscap info --profile hipaa /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
...
Profile
Title: Health Insurance Portability and Accountability Act (HIPAA)

Description: The HIPAA Security Rule establishes U.S. national standards to protect
individuals' electronic personal health information that is created, received, used, or
maintained by a covered entity.
...
```

関連情報

- システム上の **scap-security-guide(8)** man ページ
- [OpenSCAP のメモリー消費の問題](#)

6.3.4. 特定のベースラインによる設定コンプライアンスの評価

oscap コマンドラインツールを使用して、システムまたはリモートシステムが特定のベースラインに準拠しているかどうかを判断し、結果をレポートに保存できます。

前提条件

- **openscap-scanner** パッケージおよび **scap-security-guide** パッケージがインストールされている。
- システムが準拠する必要があるベースライン内のプロファイルの ID を知っている必要があります。ID を見つけるには、[設定コンプライアンスのプロファイルの表示](#) セクションを参照してください。

手順

1. ローカルシステムをスキャンして、選択したプロファイルへの準拠を評価し、スキャン結果をファイルに保存します。

```
$ oscap xccdf eval --report <scan-report.html> --profile <profileID>
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

以下のように置き換えます。

- **<scan-report.html>** は、**oscap** がスキャン結果を保存するファイル名です。
 - **<profileID>** は、システムが準拠する必要があるプロファイル ID (例: **hipaa**) です。
2. オプション: リモートシステムをスキャンして、選択したプロファイルへの準拠を評価し、スキャン結果をファイルに保存します。

```
$ oscap-ssh <username>@<hostname> <port> xccdf eval --report <scan-report.html> --
profile <profileID> /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

以下のように置き換えます。

- `<username>@<hostname>` は、リモートシステムのユーザー名とホスト名に置き換えます。
- `<port>` は、リモートシステムにアクセスできるポート番号です。
- `<scan-report.html>` は、**oscap** がスキャン結果を保存するファイル名です。
- `<profileID>` は、システムが準拠する必要があるプロファイル ID (例: **hipaa**) です。

関連情報

- システム上の **scap-security-guide(8)** man ページ
- `/usr/share/doc/scap-security-guide/` ディレクトリーにある **SCAP Security Guide** ドキュメント
- `/usr/share/doc/scap-security-guide/guides/ssg-rhel9-guide-index.html - scap-security-guide-doc` パッケージでインストールされた Red Hat Enterprise Linux 9 のセキュアな設定ガイド
- [OpenSCAP のメモリー消費の問題](#)

6.4. 特定のベースラインに合わせたシステムの修正

特定のベースラインに合わせて RHEL システムを修正できます。SCAP Security Guide で提供されるプロファイルに合わせてシステムを修正できます。使用可能なプロファイルのリストの詳細は、[設定コンプライアンスのプロファイルの表示](#) を参照してください。



警告

修正 オプションが有効な状態でのシステム評価は、慎重に行わないとシステムが機能不全に陥る場合があります。Red Hat は、セキュリティーを強化した修正で加えられた変更を元に戻す自動手段は提供していません。修復は、デフォルト設定の RHEL システムで対応しています。インストール後にシステムが変更した場合は、修正を実行しても、必要なセキュリティープロファイルに準拠しない場合があります。

前提条件

- **scap-security-guide** パッケージがインストールされている。

手順

1. **--remediate** オプションを指定した **oscap** コマンドを使用してシステムを修復します。

```
# oscap xccdf eval --profile <profileID> --remediate /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

<profileID> は、システムが準拠する必要があるプロファイル ID (例: **hipaa**) に置き換えます。

2. システムを再起動します。

検証

1. システムがプロファイルに準拠しているかどうかを評価し、スキャン結果をファイルに保存します。

```
$ oscap xccdf eval --report <scan-report.html> --profile <profileID>
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

以下のように置き換えます。

- **<scan-report.html>** は、**oscap** がスキャン結果を保存するファイル名です。
- **<profileID>** は、システムが準拠する必要があるプロファイル ID (例: **hipaa**) です。

関連情報

- システム上の **scap-security-guide(8)** および **oscap(8)** の man ページ

6.5. SSG ANSIBLE PLAYBOOK を使用した特定のベースラインに合わせたシステムの修正

SCAP Security Guide プロジェクトの Ansible Playbook ファイルを使用して、特定のベースラインに合わせてシステムを修正できます。SCAP Security Guide によって提供されるすべてのプロファイルに合わせて修正できます。



警告

Remediate オプションが有効な状態でのシステム評価は、慎重に行わないとシステムが機能不全に陥る場合があります。Red Hat は、セキュリティを強化した修正で加えられた変更を元に戻す自動手段は提供していません。修復は、デフォルト設定の RHEL システムで対応しています。インストール後にシステムが変更した場合は、修正を実行しても、必要なセキュリティプロファイルに準拠しない場合があります。

前提条件

- **scap-security-guide** パッケージがインストールされている。
- **ansible-core** パッケージがインストールされている。詳細は、[Ansible インストールガイド](#) を参照してください。
- **rhc-worker-playbook** パッケージがインストールされている。
- システムの修正に使用するプロファイルの ID がわかっている。詳細は、[設定コンプライアンスのプロファイルの表示](#) を参照してください。

手順

1. Ansible を使用して、選択したプロファイルに準拠するようにシステムを修正します。

```
# ANSIBLE_COLLECTIONS_PATH=/usr/share/rhc-worker-  
playbook/ansible/collections/ansible_collections/ ansible-playbook -i "localhost," -c local  
/usr/share/scap-security-guide/ansible/rhel9-playbook-<profileID>.yaml
```

このコマンドで Playbook を実行するには、**ANSIBLE_COLLECTIONS_PATH** 環境変数が必要です。

<profileID> は、選択したプロファイルのプロファイル ID に置き換えます。

2. システムを再起動します。

検証

- システムが選択したプロファイルに準拠しているかどうかを評価し、スキャン結果をファイルに保存します。

```
# oscap xccdf eval --profile <profileID> --report <scan-report.html>  
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

<scan-report.html> は、**oscap** がスキャン結果を保存するファイル名に置き換えます。

関連情報

- システム上の **scap-security-guide(8)** および **oscap(8)** の man ページ
- [Ansible Documentation](#)

6.6. システムを特定のベースラインに合わせるための修復用 ANSIBLE PLAYBOOK の作成

システムを特定のベースラインに合わせるために必要な修復のみを含む Ansible Playbook を作成できます。この Playbook は、すでに満たされている要件を含んでいないため、小型です。Playbook を作成しても、システムは一切変更されません。ここでは、後で適用するためのファイルを準備するだけです。



注記

RHEL 9 では、Ansible Engine は、組み込みモジュールのみを含む **ansible-core** パッケージに置き換えられました。Ansible 修復の多くは、community コレクションおよび Portable Operating System Interface (POSIX) コレクションのモジュールを使用することに注意してください。これは組み込みモジュールには含まれていません。この場合は、Bash 修復を Ansible 修復の代わりに使用できます。RHEL 9.0 の Red Hat Connector には、修復 Playbook が Ansible Core で機能するために必要な Ansible モジュールが含まれています。

前提条件

- **scap-security-guide** パッケージがインストールされている。
- **ansible-core** パッケージがインストールされている。詳細は、[Ansible インストールガイド](#) を参照してください。

- **rhc-worker-playbook** パッケージがインストールされている。
- システムの修正に使用するプロファイルの ID がわかっている。詳細は、[設定コンプライアンスのプロファイルの表示](#) を参照してください。

手順

1. システムをスキャンして結果を保存します。

```
# oscap xccdf eval --profile <profileID> --results <profile-results.xml>
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

2. 結果が含まれるファイルで、結果 ID の値を見つけます。

```
# oscap info <profile-results.xml>
```

3. ステップ 1 で生成されたファイルに基づいて Ansible Playbook を生成します。

```
# oscap xccdf generate fix --fix-type ansible --result-id xccdf_org.open-
scap_testresult_xccdf_org.ssgproject.content_profile_<profileID> --output <profile-
remediations.yml> <profile-results.xml>
```

4. 生成された **<profile-remediations.yml>** ファイルに、ステップ 1 で実行したスキャンで失敗したルールに対する Ansible 修復が含まれていることを確認します。
5. Ansible を使用して、選択したプロファイルに準拠するようにシステムを修正します。

```
# ANSIBLE_COLLECTIONS_PATH=/usr/share/rhc-worker-
playbook/ansible/collections/ansible_collections/ ansible-playbook -i "localhost," -c local
<profile-remediations.yml>
```

このコマンドで Playbook を実行するには、**ANSIBLE_COLLECTIONS_PATH** 環境変数が必要です。



警告

Remediate オプションが有効な状態でのシステム評価は、慎重に行わないとシステムが機能不全に陥る場合があります。Red Hat は、セキュリティーハードニング関連の修復によって行われた変更を自動的に元に戻す方法は提供していません。修復は、デフォルト設定の RHEL システムで対応しています。インストール後にシステムが変更した場合は、修正を実行しても、必要なセキュリティープロファイルに準拠しない場合があります。

検証

- システムが選択したプロファイルに準拠しているかどうかを評価し、スキャン結果をファイルに保存します。

```
# oscap xccdf eval --profile <profileID> --report <scan-report.html>
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

<scan-report.html> は、**oscap** がスキャン結果を保存するファイル名に置き換えます。

関連情報

- システム上の **scap-security-guide(8)** および **oscap(8)** の man ページ
- [Ansible Documentation](#)

6.7. 後でアプリケーションを修復するための BASH スクリプトの作成

この手順を使用して、システムを HIPAA などのセキュリティープロファイルと調整する修正を含む Bash スクリプトを作成します。次の手順では、システムに変更を加えることなく、後のアプリケーション用にファイルを準備する方法を説明します。

前提条件

- RHEL システムに、**scap-security-guide** パッケージがインストールされている。

手順

- oscap** コマンドを使用してシステムをスキャンし、結果を XML ファイルに保存します。以下の例では、**oscap** は **hipaa** プロファイルに対してシステムを評価します。

```
# oscap xccdf eval --profile hipaa --results <hipaa-results.xml>
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

- 結果が含まれるファイルで、結果 ID の値を見つけます。

```
# oscap info <hipaa-results.xml>
```

- 手順1で生成された結果ファイルに基づいて Bash スクリプトを生成します。

```
# oscap xccdf generate fix --fix-type bash --result-id <xccdf_org.open-
scap_testresult_xccdf_org.ssgproject.content_profile_hipaa> --output <hipaa-
remediations.sh> <hipaa-results.xml>
```

- <hipaa-remediations.sh> ファイルには、手順1で実行されたスキャン中に失敗したルールの修復が含まれます。この生成されたファイルを確認したら、このファイルと同じディレクトリ内で、./<hipaa-remediations.sh> コマンドを使用してファイルを適用できます。

検証

- お使いのテキストエディターで、手順1で実行したスキャンで失敗したルールが <hipaa-remediations.sh> ファイルに含まれていることを確認します。

関連情報

- システム上の **scap-security-guide(8)**、**oscap(8)**、**bash(1)** man ページ

6.8. SCAP WORKBENCH を使用したカスタムプロファイルでシステムのスキャン

SCAP Workbench (`scap-workbench`) パッケージはグラフィカルユーティリティで、1 台のローカルシステムまたはリモートシステムで設定スキャンと脆弱性スキャンを実行し、システムの修復を実行して、スキャン評価に基づくレポートを生成します。**oscap** コマンドラインユーティリティとの比較は、**SCAP Workbench** には限定的な機能しかないことに注意してください。**SCAP Workbench** は、データストリームファイルの形式でセキュリティーコンテンツを処理します。

6.8.1. SCAP Workbench を使用したシステムのスキャンおよび修復

選択したセキュリティーポリシーに対してシステムを評価するには、以下の手順に従います。

前提条件

- `scap-workbench` 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. **GNOME Classic** デスクトップ環境から **SCAP Workbench** を実行するには、**Super** キーを押して **アクティビティーの概要** を開き、`scap-workbench` と入力して **Enter** を押します。または、次のコマンドを実行します。

```
$ scap-workbench &
```

2. 以下のオプションのいずれかを使用してセキュリティーポリシーを選択します。

- 開始ウィンドウの **Load Content** ボタン
- **Open content from SCAP Security Guide**
- **File** メニューの **Open Other Content** で、XCCDF、SCAP RPM、またはデータストリームファイルの各ファイルを検索します。



3. **Remediate** チェックボックスを選択して、システム設定の自動修正を行うことができます。このオプションを有効にすると、**SCAP Workbench** は、ポリシーにより適用されるセキュリティールールに従ってシステム設定の変更を試みます。このプロセスは、システムスキャン時に失敗した関連チェックを修正する必要があります。



警告

修正 オプションが有効な状態でのシステム評価は、慎重に行わないとシステムが機能不全に陥る場合があります。Red Hat は、セキュリティーを強化した修正で加えられた変更を元に戻す自動手段は提供していません。修復は、デフォルト設定の RHEL システムで対応しています。インストール後にシステムが変更した場合は、修正を実行しても、必要なセキュリティープロファイルに準拠しない場合があります。

4. **Scan** ボタンをクリックし、選択したプロファイルでシステムをスキャンします。

The screenshot shows the SCAP Workbench window titled 'ssg-rhel9-ds.xml - SCAP Workbench'. The interface includes a menu bar with 'File' and 'Help'. The main area displays the following configuration and results:

- Title:** Guide to the Secure Configuration of Red Hat Enterprise Linux 9
- Customization:** None selected
- Profile:** PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 9 (121)
- Target:** Local Machine (selected), Remote Machine (over SSH) (unselected)
- Rules:** A list of 12 rules, all with a 'pass' status. An 'Expand all' button is visible.
- Progress:** 100% (121 results, 121 rules selected)
- Buttons:** Clear, Save Results, Generate remediation role, Show Report
- Status:** Processing has been finished!

Rule	Status
Verify Group Who Owns shadow File	pass
Verify User Who Owns group File	pass
Verify User Who Owns passwd File	pass
Verify User Who Owns shadow File	pass
Verify Permissions on group File	pass
Verify Permissions on passwd File	pass
Verify Permissions on shadow File	pass
The Chronyd service is enabled	pass
A remote time server for Chrony is configured	pass
Distribute the SSH Server configuration to multiple files in a config directory.	pass
Enable Smartcards in SSSD	pass

5. スキャン結果を XCCDF ファイル、ARF ファイル、または HTML ファイルの形式で保存するには、**Save Results** コンボボックスをクリックします。**HTML Report** オプションを選択して、スキャンレポートを、人間が判読できる形式で生成します。XCCDF 形式および ARF (データストリーム) 形式は、追加の自動処理に適しています。3 つのオプションはすべて繰り返し選択できます。
6. 結果ベースの修復をファイルにエクスポートするには、ポップアップメニューの **Generate remediation role** を使用します。

6.8.2. SCAP Workbench を使用したセキュリティープロファイルのカスタマイズ

セキュリティープロファイルのカスタマイズするには、特定のルール (パスワードの最小長など) のパラメーターを変更し、別の方法で対象とするルールを削除し、追加のルールを選択して内部ポリシーを実装できます。プロファイルのカスタマイズして新しいルールの定義はできません。

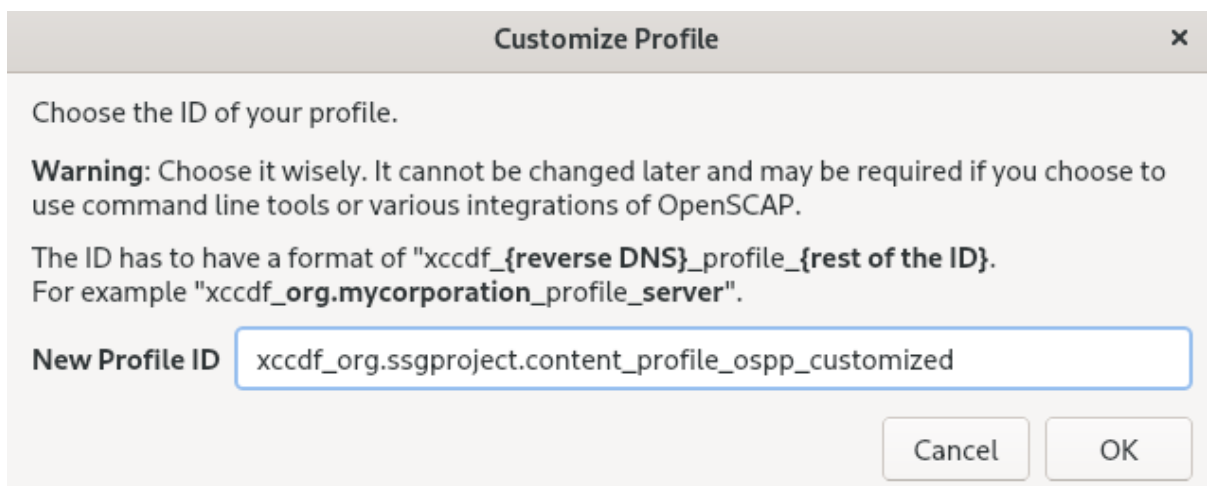
以下の手順では、**SCAP Workbench** を使用してプロファイルのカスタマイズ (調整) します。**oscap** コマンドラインユーティリティーで使用するようカスタマイズしたプロファイルを保存することもできます。

前提条件

- **scap-workbench** 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. **SCAP Workbench** を実行し、**Open content from SCAP Security Guide** または **File** メニューの **Open Other Content** を使用してカスタマイズするプロファイルを選択します。
2. 選択したセキュリティープロファイルを必要に応じて調整するには、**Customize** ボタンをクリックします。
これにより、元のデータストリームファイルを変更せずに現在選択されているプロファイルを変更できる新しいカスタマイズウィンドウが開きます。新しいプロファイル ID を選択します。



Customize Profile

Choose the ID of your profile.

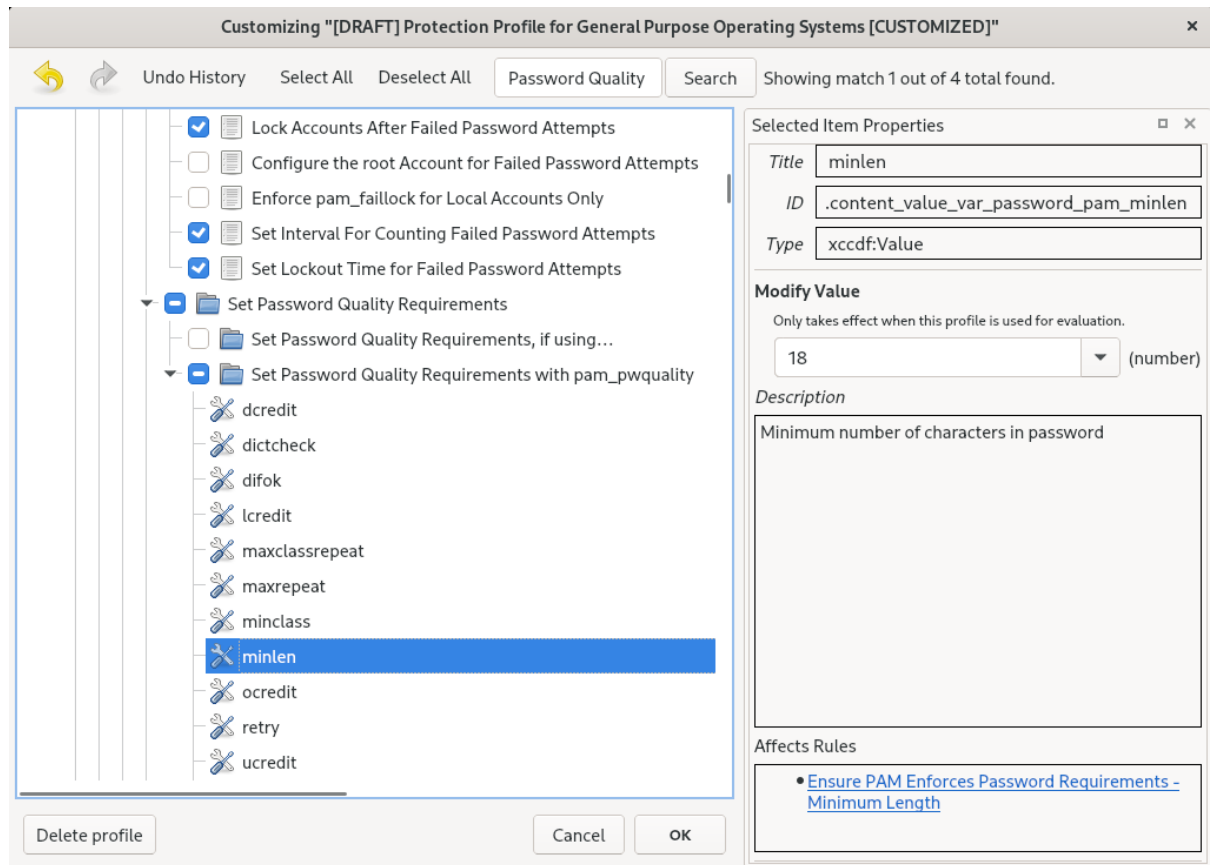
Warning: Choose it wisely. It cannot be changed later and may be required if you choose to use command line tools or various integrations of OpenSCAP.

The ID has to have a format of "xccdf_{reverse DNS}_profile_{rest of the ID}."
For example "xccdf_org.mycorporation_profile_server".

New Profile ID

Cancel OK

3. 論理グループに分けられたルールを持つツリー構造を使用するか、**Search** フィールドを使用して変更するルールを検索します。
4. ツリー構造のチェックボックスを使用した include ルールまたは exclude ルール、または必要に応じてルールの値を変更します。



5. **OK** ボタンをクリックして変更を確認します。
6. 変更内容を永続的に保存するには、以下のいずれかのオプションを使用します。
 - **File** メニューの **Save Customization Only** を使用して、カスタマイズファイルを別途保存します。
 - **File** メニュー **Save All** を選択して、すべてのセキュリティーコンテンツを一度に保存します。
Into a directory オプションを選択すると、**SCAP Workbench** は、データストリームファイルおよびカスタマイズファイルの両方を、指定した場所に保存します。これはバックアップソリューションとして使用できます。
 - **As RPM** オプションを選択すると、**SCAP Workbench** に、データストリームファイル、ならびにカスタマイズファイルを含む RPM パッケージの作成を指示できます。これは、リモートでスキャンできないシステムにセキュリティーコンテンツを配布したり、詳細な処理のためにコンテンツを配信するのに便利です。
7. **SCAP Workbench** は、カスタマイズしたプロファイル向けの結果ベースの修正に対応していないため、**oscap** コマンドラインユーティリティーでエクスポートした修正を使用します。

6.8.3. 関連情報

- システム上の **scap-workbench(8)** man ページ
- **scap-workbench** パッケージで提供される **/usr/share/doc/scap-workbench/user_manual.html** ファイル
- [Deploy customized SCAP policies with Satellite 6.x](#) (Red Hat ナレッジベース)

6.9. インストール直後にセキュリティープロファイルに準拠するシステムのデプロイメント

OpenSCAP スイートを使用して、インストールプロセスの直後に、OSPP や PCI-DSS、HIPAA プロファイルなどのセキュリティープロファイルに準拠する RHEL システムをデプロイできます。このデプロイメント方法を使用すると、修正スクリプトを使用して後で適用できない特定のルール (パスワードの強度とパーティション化のルールなど) を適用できます。

6.9.1. Server with GUI と互換性のないプロファイル

SCAP Security Guide の一部として提供される一部のセキュリティープロファイルは、**Server with GUI** ベース環境に含まれる拡張パッケージセットと互換性がありません。したがって、次のいずれかのプロファイルに準拠するシステムをインストールする場合は、**Server with GUI** を選択しないでください。

表6.1 Server with GUI と互換性のないプロファイル

プロファイル名	プロファイル ID	理由	注記
[ドラフト] CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Server	xccdf_org.ssgproject.content_profile_cis	パッケージ xorg-x11-server-Xorg 、 xorg-x11-server-common 、 xorg-x11-server-utils 、 xorg-x11-server-Xwayland は、 Server with GUI パッケージセットの一部ですが、ポリシーにより削除する必要があります。	
[ドラフト] CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server	xccdf_org.ssgproject.content_profile_cis_server_l1	パッケージ xorg-x11-server-Xorg 、 xorg-x11-server-common 、 xorg-x11-server-utils 、 xorg-x11-server-Xwayland は、 Server with GUI パッケージセットの一部ですが、ポリシーにより削除する必要があります。	

プロファイル名	プロファイル ID	理由	注記
DISA STIG for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig	パッケージ xorg-x11-server-Xorg 、 xorg-x11-server-common 、 xorg-x11-server-utils 、 xorg-x11-server-Xwayland は、 Server with GUI パッケージセットの一部ですが、ポリシーにより削除する必要があります。	RHEL システムを DISA STIG に準拠した Server with GUI としてインストールするには、DISA STIG with GUI プロファイルを使用できます (BZ#1648162)

6.9.2. グラフィカルインストールを使用したベースライン準拠の RHEL システムのデプロイメント

この手順を使用して、特定のベースラインに合わせた RHEL システムをデプロイします。この例では、OSPP (Protection Profile for General Purpose Operating System) を使用します。



警告

SCAP Security Guide の一部として提供される一部のセキュリティープロファイルは、**Server with GUI** ベース環境に含まれる拡張パッケージセットと互換性がありません。詳細は、[GUI サーバーと互換性のないプロファイル](#) を参照してください。

前提条件

- **グラフィカル** インストールプログラムでシステムを起動している。**OSCAP Anaconda アドオン** はインタラクティブなテキストのみのインストールをサポートしていないことに注意してください。
- **インストール概要** 画面を開いている。

手順

1. **インストール概要** 画面で、**ソフトウェアの選択** をクリックします。**ソフトウェアの選択** 画面が開きます。
2. **ベース環境** ペインで、**サーバー** 環境を選択します。ベース環境は、1つだけ選択できます。
3. **完了** をクリックして設定を適用し、**インストール概要** 画面に戻ります。

4. OSPP には、準拠する必要がある厳密なパーティション分割要件があるため、**/boot**、**/home**、**/var**、**/tmp**、**/var/log**、**/var/tmp**、および **/var/log/audit** にそれぞれパーティションを作成します。
5. **セキュリティポリシー** をクリックします。**セキュリティポリシー** 画面が開きます。
6. システムでセキュリティポリシーを有効にするには、**セキュリティポリシーの適用** を **ON** に切り替えます。
7. プロファイルペインで **Protection Profile for General Purpose Operating Systems** プロファイルを選択します。
8. **プロファイルの選択** をクリックして選択を確定します。
9. 画面下部に表示される **Changes that were done or need to be done** の変更を確定します。残りの手動変更を完了します。
10. グラフィカルインストールプロセスを完了します。



注記

グラフィカルインストールプログラムは、インストールに成功すると、対応するキックスタートファイルを自動的に作成します。**/root/anaconda-ks.cfg** ファイルを使用して、OSPP 準拠のシステムを自動的にインストールできます。

検証

- インストール完了後にシステムの現在のステータスを確認するには、システムを再起動して新しいスキャンを開始します。

```
# oscap xccdf eval --profile ospp --report eval_postinstall_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

関連情報

- [手動パーティションの設定](#)

6.9.3. キックスタートを使用したベースライン準拠の RHEL システムのデプロイメント

特定のベースラインに準拠した RHEL システムをデプロイできます。この例では、OSPP (Protection Profile for General Purpose Operating System) を使用します。

前提条件

- RHEL 9 システムに、**scap-security-guide** パッケージがインストールされている。

手順

1. キックスタートファイル **/usr/share/scap-security-guide/kickstart/ssg-rhel9-ospp-ks.cfg** を、選択したエディターで開きます。
2. 設定要件を満たすように、パーティション設定スキームを更新します。OSPP に準拠するには、**/boot**、**/home**、**/var**、**/tmp**、**/var/log**、**/var/tmp**、および **/var/log/audit** の個別のパーティションを保持する必要があります。パーティションのサイズのみ変更することができます。

3. [キックスタートを使用した自動インストールの実行](#) の説明に従って、キックスタートインストールを開始します。



重要

キックスタートファイルのパスワードでは、OSPP の要件が確認されていません。

検証

- インストール完了後にシステムの現在のステータスを確認するには、システムを再起動して新しいスキャンを開始します。

```
# oscap xccdf eval --profile ospp --report eval_postinstall_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

関連情報

- [OSCAP Anaconda Add-on](#)
- [キックスタートのコマンドおよびオプションのリファレンス: %addon org_fedora_oscap](#)

6.10. コンテナおよびコンテナイメージの脆弱性スキャン

以下の手順を使用して、コンテナまたはコンテナイメージのセキュリティ脆弱性を検索します。

前提条件

- **openscap-utils** および **bzip2** パッケージがインストールされます。

手順

1. システムに最新 RHSA OVAL 定義をダウンロードします。

```
# wget -O - https://www.redhat.com/security/data/oval/v2/RHEL9/rhel-9.oval.xml.bz2 | bzip2 -
-decompress > rhel-9.oval.xml
```

2. コンテナまたはコンテナイメージの ID を取得します。以下に例を示します。

```
# podman images
REPOSITORY          TAG    IMAGE ID    CREATED    SIZE
registry.access.redhat.com/ubi9/ubi latest 096cae65a207 7 weeks ago 239 MB
```

3. コンテナまたはコンテナイメージで脆弱性をスキャンし、結果を **vulnerability.html** ファイルに保存します。

```
# oscap-podman 096cae65a207 oval eval --report vulnerability.html rhel-9.oval.xml
```

oscap-podman コマンドには root 権限が必要で、コンテナの ID は最初の引数であることに注意してください。

検証

- 結果をブラウザで確認します。以下に例を示します。

```
$ firefox vulnerability.html &
```

関連情報

- 詳細は、**oscap-podman(8)** および **oscap(8)** の man ページを参照してください。

6.11. 特定のベースラインを使用したコンテナまたはコンテナイメージのセキュリティーコンプライアンスの評価

コンテナまたはコンテナイメージが、Operating System Protection Profile (OSPP)、Payment Card Industry Data Security Standard (PCI-DSS)、Health Insurance Portability and Accountability Act (HIPAA) などの特定のセキュリティーベースラインに準拠しているかどうかを評価できます。

前提条件

- **openscap-utils** パッケージおよび **scap-security-guide** パッケージがインストールされている。
- システムへの root アクセス権があります。

手順

1. コンテナまたはコンテナイメージの ID を見つけます。
 - a. コンテナの ID を見つけるには、**podman ps -a** コマンドを入力します。
 - b. コンテナイメージの ID を見つけるには、**podman images** コマンドを入力します。
2. コンテナまたはコンテナイメージがプロファイルに準拠しているかどうかを評価し、スキャン結果をファイルに保存します。

```
# oscap-podman <ID> xccdf eval --report <scan-report.html> --profile <profileID>
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

以下のように置き換えます。

- **<ID>** は、コンテナまたはコンテナイメージの ID です。
- **<scan-report.html>** は、**oscap** がスキャン結果を保存するファイル名です。
- **<profileID>** は、システムが準拠する必要があるプロファイル ID (例: **hipaa**、**ospp**、**pci-dss**) です。

検証

- 結果をブラウザで確認します。以下に例を示します。

```
$ firefox <scan-report.html> &
```

**注記**

notapplicable とマークされたルールは、ベアメタルおよび仮想化システムにのみ適用され、コンテナまたはコンテナイメージには適用されません。

関連情報

- **oscap-podman(8)** および **scap-security-guide(8)** の man ページ。
- `/usr/share/doc/scap-security-guide/` ディレクトリー。

6.12. RHEL 9 でサポートされる SCAP SECURITY GUIDE プロファイル

RHEL の特定のマイナーリリースで提供される SCAP コンテンツだけを使用してください。これは、ハードニングに關与するコンポーネントが新しい機能で更新されることがあるためです。SCAP コンテンツは、この更新を反映するように変更されますが、以前のバージョンと互換性があるとは限りません。

**注記**

oscap info コマンドを使用すると、システムにインストールされている **scap-security-guide** RPM のバージョンに關連する情報を取得できます。詳細は、[設定コンプライアンスのプロファイルの表示](#) を参照してください。

以下の表では、RHEL 9 で提供されるプロファイルと、プロファイルが適合するポリシーのバージョンを紹介しています。

表6.2 RHEL 10.0 でサポートされている SCAP Security Guide プロファイル

プロファイル名	プロファイル ID	ポリシーバージョン
Security of Information Systems (ANSSI) BP-028 Enhanced Level	xccdf_org.ssgproject.content_profile_anssi_bp28_enhanced	2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 High Level	xccdf_org.ssgproject.content_profile_anssi_bp28_high	2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Intermediary Level	xccdf_org.ssgproject.content_profile_anssi_bp28_intermediary	2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Minimal Level	xccdf_org.ssgproject.content_profile_anssi_bp28_minimal	2.0
CCN Red Hat Enterprise Linux 9 - Advanced	xccdf_org.ssgproject.content_profile_ccn_advanced	2022-10

プロファイル名	プロファイル ID	ポリシーバージョン
CCN Red Hat Enterprise Linux 9 - Basic	xccdf_org.ssgproject.content_profile_ccn_basic	2022-10
CCN Red Hat Enterprise Linux 9 - Intermediate	xccdf_org.ssgproject.content_profile_ccn_intermediate	2022-10
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Server	xccdf_org.ssgproject.content_profile_cis	2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server	xccdf_org.ssgproject.content_profile_cis_server_l1	2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l1	2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l2	2.0.0
[ドラフト] Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	xccdf_org.ssgproject.content_profile_cui	r2
Australian Cyber Security Centre (ACSC) Essential Eight	xccdf_org.ssgproject.content_profile_e8	バージョン付けなし
Health Insurance Portability and Accountability Act (HIPAA)	xccdf_org.ssgproject.content_profile_hipaa	バージョン付けなし
Australian Cyber Security Centre (ACSC) ISM Official	xccdf_org.ssgproject.content_profile_ism_o	バージョン付けなし
Protection Profile for General Purpose Operating Systems	xccdf_org.ssgproject.content_profile_ospp	4.3
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_pci-dss	4.0.1
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig	V2R3

プロファイル名	プロファイル ID	ポリシーバージョン
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig_gui	V2R3

表6.3 RHEL 9.5 でサポートされる SCAP Security Guide プロファイル

プロファイル名	プロファイル ID	ポリシーバージョン
Security of Information Systems (ANSSI) BP-028 Enhanced Level	xccdf_org.ssgproject.content_profile_anssi_bp28_enhanced	2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 High Level	xccdf_org.ssgproject.content_profile_anssi_bp28_high	2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Intermediary Level	xccdf_org.ssgproject.content_profile_anssi_bp28_intermediary	2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Minimal Level	xccdf_org.ssgproject.content_profile_anssi_bp28_minimal	2.0
CCN Red Hat Enterprise Linux 9 - Advanced	xccdf_org.ssgproject.content_profile_ccn_advanced	2022-10
CCN Red Hat Enterprise Linux 9 - Basic	xccdf_org.ssgproject.content_profile_ccn_basic	2022-10
CCN Red Hat Enterprise Linux 9 - Intermediate	xccdf_org.ssgproject.content_profile_ccn_intermediate	2022-10
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Server	xccdf_org.ssgproject.content_profile_cis	2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server	xccdf_org.ssgproject.content_profile_cis_server_l1	2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l1	2.0.0

プロファイル名	プロファイル ID	ポリシーバージョン
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l2	2.0.0
[ドラフト] Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	xccdf_org.ssgproject.content_profile_cui	r2
Australian Cyber Security Centre (ACSC) Essential Eight	xccdf_org.ssgproject.content_profile_e8	バージョン付けなし
Health Insurance Portability and Accountability Act (HIPAA)	xccdf_org.ssgproject.content_profile_hipaa	バージョン付けなし
Australian Cyber Security Centre (ACSC) ISM Official	xccdf_org.ssgproject.content_profile_ism_o	バージョン付けなし
Protection Profile for General Purpose Operating Systems	xccdf_org.ssgproject.content_profile_ospp	4.3
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_pci-dss	RHEL9.5.0: 4.0 RHEL 9.5.1 以降: 4.0.1
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig	V2R3
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig_gui	V2R3

表6.4 RHEL 9.4 でサポートされる SCAP Security Guide プロファイル

プロファイル名	プロファイル ID	ポリシーバージョン
Security of Information Systems (ANSSI) BP-028 Enhanced Level	xccdf_org.ssgproject.content_profile_anssi_bp28_enhanced	2.0

プロファイル名	プロファイル ID	ポリシーバージョン
French National Agency for the Security of Information Systems (ANSSI) BP-028 High Level	xccdf_org.ssgproject.content_profile_anssi_bp28_high	2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Intermediary Level	xccdf_org.ssgproject.content_profile_anssi_bp28_intermediary	2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Minimal Level	xccdf_org.ssgproject.content_profile_anssi_bp28_minimal	2.0
CCN Red Hat Enterprise Linux 9 - Advanced	xccdf_org.ssgproject.content_profile_ccn_advanced	2022-10
CCN Red Hat Enterprise Linux 9 - Basic	xccdf_org.ssgproject.content_profile_ccn_basic	2022-10
CCN Red Hat Enterprise Linux 9 - Intermediate	xccdf_org.ssgproject.content_profile_ccn_intermediate	2022-10
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Server	xccdf_org.ssgproject.content_profile_cis	RHEL 9.4.0 から RHEL 9.4.2: 1.0.0 RHEL 9.4.3 以降: 2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server	xccdf_org.ssgproject.content_profile_cis_server_l1	RHEL 9.4.0 から RHEL 9.4.2: 1.0.0 RHEL 9.4.3 以降: 2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l1	RHEL 9.4.0 から RHEL 9.4.2: 1.0.0 RHEL 9.4.3 以降: 2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l2	RHEL 9.4.0 から RHEL 9.4.2: 1.0.0 RHEL 9.4.3 以降: 2.0.0
[ドラフト] Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	xccdf_org.ssgproject.content_profile_cui	r2
Australian Cyber Security Centre (ACSC) Essential Eight	xccdf_org.ssgproject.content_profile_e8	バージョン付けなし

プロファイル名	プロファイル ID	ポリシーバージョン
Health Insurance Portability and Accountability Act (HIPAA)	xccdf_org.ssgproject.content_profile_hipaa	バージョン付けなし
Australian Cyber Security Centre (ACSC) ISM Official	xccdf_org.ssgproject.content_profile_ism_o	バージョン付けなし
Protection Profile for General Purpose Operating Systems	xccdf_org.ssgproject.content_profile_ospp	4.3
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_pci-dss	RHEL 9.4.0 から RHEL 9.4.4: 4.0 RHEL9.4.5: 4.0.1
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig	V2R3
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig_gui	V2R3

表6.5 RHEL 9.3 でサポートされる SCAP Security Guide プロファイル

プロファイル名	プロファイル ID	ポリシーバージョン
Security of Information Systems (ANSSI) BP-028 Enhanced Level	xccdf_org.ssgproject.content_profile_anssi_bp28_enhanced	2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 High Level	xccdf_org.ssgproject.content_profile_anssi_bp28_high	2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Intermediary Level	xccdf_org.ssgproject.content_profile_anssi_bp28_intermediary	2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Minimal Level	xccdf_org.ssgproject.content_profile_anssi_bp28_minimal	2.0
CCN Red Hat Enterprise Linux 9 - Advanced	xccdf_org.ssgproject.content_profile_ccn_advanced	2022-10

プロファイル名	プロファイル ID	ポリシーバージョン
CCN Red Hat Enterprise Linux 9 - Basic	xccdf_org.ssgproject.content_profile_ccn_basic	2022-10
CCN Red Hat Enterprise Linux 9 - Intermediate	xccdf_org.ssgproject.content_profile_ccn_intermediate	2022-10
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Server	xccdf_org.ssgproject.content_profile_cis	1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server	xccdf_org.ssgproject.content_profile_cis_server_l1	1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l1	1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l2	1.0.0
[ドラフト] Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	xccdf_org.ssgproject.content_profile_cui	r2
Australian Cyber Security Centre (ACSC) Essential Eight	xccdf_org.ssgproject.content_profile_e8	バージョン付けなし
Health Insurance Portability and Accountability Act (HIPAA)	xccdf_org.ssgproject.content_profile_hipaa	バージョン付けなし
Australian Cyber Security Centre (ACSC) ISM Official	xccdf_org.ssgproject.content_profile_ism_o	バージョン付けなし
Protection Profile for General Purpose Operating Systems	xccdf_org.ssgproject.content_profile_ospp	4.3
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_pci-dss	RHEL 9.3.0 から RHEL 9.3.2:3.2.1 RHEL 9.3.3:4.0
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig	RHEL 9.3.0: ドラフト ^[a] RHEL 9.3.2: V1R1 RHEL 9.3.3 以降: V1R2

プロファイル名	プロファイル ID	ポリシーバージョン
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig_gui	RHEL 9.3.0: ドラフト [a] RHEL 9.3.2: VIR1 RHEL 9.3.3 以降: VIR2
[a] DISA は RHEL 9 の公式ベンチマークを公開していません。		

表6.6 RHEL 9.2 でサポートされる SCAP Security Guide プロファイル

プロファイル名	プロファイル ID	ポリシーバージョン
Security of Information Systems (ANSSI) BP-028 Enhanced Level	xccdf_org.ssgproject.content_profile_anssi_bp28_enhanced	RHEL 9.2.0 から RHEL 9.2.2: 1.2 RHEL 9.2.3 以降: 2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 High Level	xccdf_org.ssgproject.content_profile_anssi_bp28_high	RHEL 9.2.0 から RHEL 9.2.2: 1.2 RHEL 9.2.3 以降: 2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Intermediary Level	xccdf_org.ssgproject.content_profile_anssi_bp28_intermediary	RHEL 9.2.0 から RHEL 9.2.2: 1.2 RHEL 9.2.3 以降: 2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Minimal Level	xccdf_org.ssgproject.content_profile_anssi_bp28_minimal	RHEL 9.2.0 から RHEL 9.2.2: 1.2 RHEL 9.2.3 以降: 2.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Server	xccdf_org.ssgproject.content_profile_cis	RHEL 9.2.0 から RHEL 9.2.10: 1.0.0 RHEL 9.2.11 以降: 2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server	xccdf_org.ssgproject.content_profile_cis_server_l1	RHEL 9.2.0 から RHEL 9.2.10: 1.0.0 RHEL 9.2.11 以降: 2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l1	RHEL 9.2.0 から RHEL 9.2.10: 1.0.0 RHEL 9.2.11 以降: 2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l2	RHEL 9.2.0 から RHEL 9.2.10: 1.0.0 RHEL 9.2.11 以降: 2.0.0

プロファイル名	プロファイル ID	ポリシーバージョン
[ドラフト] Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	xccdf_org.ssgproject.content_profile_cui	r2
Australian Cyber Security Centre (ACSC) Essential Eight	xccdf_org.ssgproject.content_profile_e8	バージョン付けなし
Health Insurance Portability and Accountability Act (HIPAA)	xccdf_org.ssgproject.content_profile_hipaa	バージョン付けなし
Australian Cyber Security Centre (ACSC) ISM Official	xccdf_org.ssgproject.content_profile_ism_o	バージョン付けなし
Protection Profile for General Purpose Operating Systems	xccdf_org.ssgproject.content_profile_ospp	4.2.1
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_pci-dss	RHEL 9.2.0 から RHEL 9.2.5: 3.2.1 RHEL 9.2.6 から RHEL 9.2.12: 4.0 RHEL 9.2.13 以降: 4.0.1
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig	V2R3
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig_gui	V2R3
CCN Red Hat Enterprise Linux 9 - Basic	xccdf_org.ssgproject.content_profile_ccn_basic	2022-10
CCN Red Hat Enterprise Linux 9 - Intermediate	xccdf_org.ssgproject.content_profile_ccn_intermediate	2022-10
CCN Red Hat Enterprise Linux 9 - Advanced	xccdf_org.ssgproject.content_profile_ccn_advanced	2022-10

表6.7 RHEL 9.1でサポートされる SCAP Security Guide プロファイル

プロファイル名	プロファイル ID	ポリシーバージョン
Security of Information Systems (ANSSI) BP-028 Enhanced Level	xccdf_org.ssgproject.content_profile_anssi_bp28_enhanced	1.2
French National Agency for the Security of Information Systems (ANSSI) BP-028 High Level	xccdf_org.ssgproject.content_profile_anssi_bp28_high	1.2
French National Agency for the Security of Information Systems (ANSSI) BP-028 Intermediary Level	xccdf_org.ssgproject.content_profile_anssi_bp28_intermediary	1.2
French National Agency for the Security of Information Systems (ANSSI) BP-028 Minimal Level	xccdf_org.ssgproject.content_profile_anssi_bp28_minimal	1.2
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Server	xccdf_org.ssgproject.content_profile_cis	RHEL 9.1.0 および RHEL 9.1.1: ドラフト [a] RHEL 9.1.2 以降: 1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server	xccdf_org.ssgproject.content_profile_cis_server_l1	RHEL 9.1.0 および RHEL 9.1.1: ドラフト [a] RHEL 9.1.2 以降: 1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l1	RHEL 9.1.0 および RHEL 9.1.1: ドラフト [a] RHEL 9.1.2 以降: 1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l2	RHEL 9.1.0 および RHEL 9.1.1: ドラフト [a] RHEL 9.1.2 以降: 1.0.0
[ドラフト] Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	xccdf_org.ssgproject.content_profile_cui	r2
Australian Cyber Security Centre (ACSC) Essential Eight	xccdf_org.ssgproject.content_profile_e8	バージョン付けなし
Health Insurance Portability and Accountability Act (HIPAA)	xccdf_org.ssgproject.content_profile_hipaa	バージョン付けなし
Australian Cyber Security Centre (ACSC) ISM Official	xccdf_org.ssgproject.content_profile_ism_o	バージョン付けなし

プロファイル名	プロファイル ID	ポリシーバージョン
Protection Profile for General Purpose Operating Systems	xccdf_org.ssgproject.content_profile_ospp	4.2.1
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_pci-dss	3.2.1
[ドラフト] The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig	ドラフト [a]
[ドラフト] The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig_gui	ドラフト [a]
[a] CIS は RHEL 9 の公式ベンチマークを公開していません。		

表6.8 RHEL 9.0 でサポートされる SCAP Security Guide プロファイル

プロファイル名	プロファイル ID	ポリシーバージョン
Security of Information Systems (ANSSI) BP-028 Enhanced Level	xccdf_org.ssgproject.content_profile_anssi_bp28_enhanced	RHEL 9.0.0 から RHEL 9.0.10:1.2 RHEL 9.0.11 以降:2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 High Level	xccdf_org.ssgproject.content_profile_anssi_bp28_high	RHEL 9.0.0 から RHEL 9.0.10:1.2 RHEL 9.0.11 以降:2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Intermediary Level	xccdf_org.ssgproject.content_profile_anssi_bp28_intermediary	RHEL 9.0.0 から RHEL 9.0.10:1.2 RHEL 9.0.11 以降:2.0
French National Agency for the Security of Information Systems (ANSSI) BP-028 Minimal Level	xccdf_org.ssgproject.content_profile_anssi_bp28_minimal	RHEL 9.0.0 から RHEL 9.0.10:1.2 RHEL 9.0.11 以降:2.0

プロファイル名	プロファイル ID	ポリシーバージョン
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Server	xccdf_org.ssgproject.content_profile_cis	RHEL 9.0.0 から RHEL 9.0.6: DRAFT [a] RHEL 9.0.7 to RHEL 9.0.19:1.0.0 RHEL 9.0.20 以降: 2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server	xccdf_org.ssgproject.content_profile_cis_server_l1	RHEL 9.0.0 から RHEL 9.0.6: DRAFT [a] RHEL 9.0.7 to RHEL 9.0.19:1.0.0 RHEL 9.0.20 以降: 2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l1	RHEL 9.0.0 から RHEL 9.0.6: DRAFT [a] RHEL 9.0.7 to RHEL 9.0.19:1.0.0 RHEL 9.0.20 以降: 2.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l2	RHEL 9.0.0 から RHEL 9.0.6: DRAFT [a] RHEL 9.0.7 to RHEL 9.0.19:1.0.0 RHEL 9.0.20 以降: 2.0.0
[ドラフト] Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	xccdf_org.ssgproject.content_profile_cui	r2
Australian Cyber Security Centre (ACSC) Essential Eight	xccdf_org.ssgproject.content_profile_e8	バージョン付けなし
Health Insurance Portability and Accountability Act (HIPAA)	xccdf_org.ssgproject.content_profile_hipaa	バージョン付けなし
Australian Cyber Security Centre (ACSC) ISM Official	xccdf_org.ssgproject.content_profile_ism_o	バージョン付けなし
Protection Profile for General Purpose Operating Systems	xccdf_org.ssgproject.content_profile_ospp	RHEL 9.0.0 から RHEL 9.0.2: ドラフト RHEL 9.0.3 以降: 4.2.1
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_pci-dss	RHEL 9.0.0 から RHEL 9.0.14: 3.2.1 RHEL 9.0.15 から RHEL 9.0.21: 4.0 RHEL 9.0.22 以降: 4.0.1
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig	V2R3

プロファイル名	プロファイル ID	ポリシーバージョン
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig_gui	V2R3
CCN Red Hat Enterprise Linux 9 - Basic	xccdf_org.ssgproject.content_profile_ccn_basic	RHEL 9.0.11 以降:2022-10
CCN Red Hat Enterprise Linux 9 - Intermediate	xccdf_org.ssgproject.content_profile_ccn_intermediate	RHEL 9.0.11 以降:2022-10
CCN Red Hat Enterprise Linux 9 - Advanced	xccdf_org.ssgproject.content_profile_ccn_advanced	RHEL 9.0.11 以降:2022-10

6.13. 関連情報

- [Supported versions of the SCAP Security Guide in RHEL](#)
- [OpenSCAP プロジェクトページ](#) では、**oscap** ユーティリティー、その他のコンポーネント、および SCAP に関連するプロジェクトに関する詳細情報を提供しています。
- [SCAP Workbench プロジェクトページ](#) は、**scap-workbench** アプリケーションに関する詳細情報を提供します。
- [SCAP Security Guide \(SSG\) プロジェクトページ](#) は、Red Hat Enterprise Linux の最新のセキュリティコンテンツを提供します。
- [セキュリティコンプライアンスと脆弱性スキャンに OpenSCAP を使用する](#): RHEL でのコンプライアンスと脆弱性スキャンのための Security Content Automation Protocol (SCAP) 標準に基づくツールの実行に関するハンズオンラボ。
- [Red Hat Security Demos: Creating Customized Security Policy Content to Automate Security Compliance](#) - RHEL に含まれるツールを使用してセキュリティコンプライアンスを自動化し、業界標準のセキュリティポリシーとカスタムセキュリティポリシーの両方に準拠するためのハンズオンラボ。トレーニングや、このラボ演習へのアクセスを希望する場合は、Red Hat アカウントチームにお問い合わせください。
- [Red Hat Security Demos: Defend Yourself with RHEL Security Technologies](#) - OpenSCAP を含む RHEL で利用可能な主要なセキュリティ技術を使用して、RHEL システムの全レベルでセキュリティを実装する方法を学ぶためのハンズオンラボ。トレーニングや、このラボ演習へのアクセスを希望する場合は、Red Hat アカウントチームにお問い合わせください。
- [National Institute of Standards and Technology \(NIST\) SCAP のページ](#) では、SCAP の出版物、仕様、SCAP 検出プログラムなどの SCAP 関連の資料が多数提供されます。
- [National Vulnerability Database \(NVD\)](#) は、SCAP コンテンツおよびその他の SCAP 標準ベースの脆弱性管理データに関する最大のリポジトリです。

- [Red Hat OVAL コンテンツリポジトリ](#) には、RHEL システムの脆弱性に対する OVAL 定義が含まれています。このページは、脆弱性の情報を得るために確認が推奨されるページです。
- [MITRE CVE](#) - これは、MITRE corporation が提供する既知のセキュリティー脆弱性のデータベースです。RHEL の場合は、Red Hat が提供する OVAL CVE コンテンツを使用することが推奨されます。
- [MITRE OVAL](#) - このページでは、MITRE corporation が提供する OVAL 関連のプロジェクトが紹介されています。OVAL の関連情報、たとえば OVAL 言語、数千にもなる OVAL 定義が用意された OVAL コンテンツのリポジトリがあります。RHEL のスキャンには、Red Hat が提供する OVAL CVE コンテンツを使用することが推奨されます。
- [Red Hat Satellite のセキュリティーコンプライアンスの管理](#) - この一連のガイドでは、OpenSCAP を使用して複数のシステムでシステムセキュリティーを維持する維持する方法などが説明されています。

第7章 KEYLIME でシステムの整合性を確保する

Keylime を使用すると、リモートシステムの整合性を継続的に監視し、起動時にシステムの状態を確認できます。また、暗号化されたファイルを監視対象システムに送信し、監視対象システムが整合性テストに失敗するたびにトリガーされる自動アクションを指定することもできます。

7.1. KEYLIME の仕組み

Keylime エージェントを設定すると、次の操作を1つ以上実行できます。

ランタイム整合性監視

Keylime のランタイム整合性監視では、エージェントがデプロイされているシステムを継続的に監視し、許可リストに含まれるファイルと除外リストに含まれないファイルの整合性を評価します。

ブート測定

Keylime のブート測定では、起動時にシステムの状態を検証します。

Keylime の信頼の概念は、Trusted Platform Module (TPM) テクノロジーに基づいています。TPM は、暗号化鍵が統合されたハードウェア、ファームウェア、または仮想コンポーネントです。TPM クォートをポーリングし、オブジェクトのハッシュを比較することで、Keylime はリモートシステムの初期監視とランタイム監視を提供します。



重要

Keylime を仮想マシン内で実行するか、仮想 TPM を使用するかは、基盤となるホストの整合性によって異なります。仮想環境での Keylime 測定を利用する前に、必ずホスト環境を信頼してください。

Keylime は、次の3つの主要コンポーネントで構成されています。

verifier

エージェントを実行するシステムの整合性を最初から継続的に検証します。verifier は、パッケージからデプロイすることも、コンテナとしてデプロイすることも、**keylime_server** RHEL システムロールを使用してデプロイすることもできます。

registrar

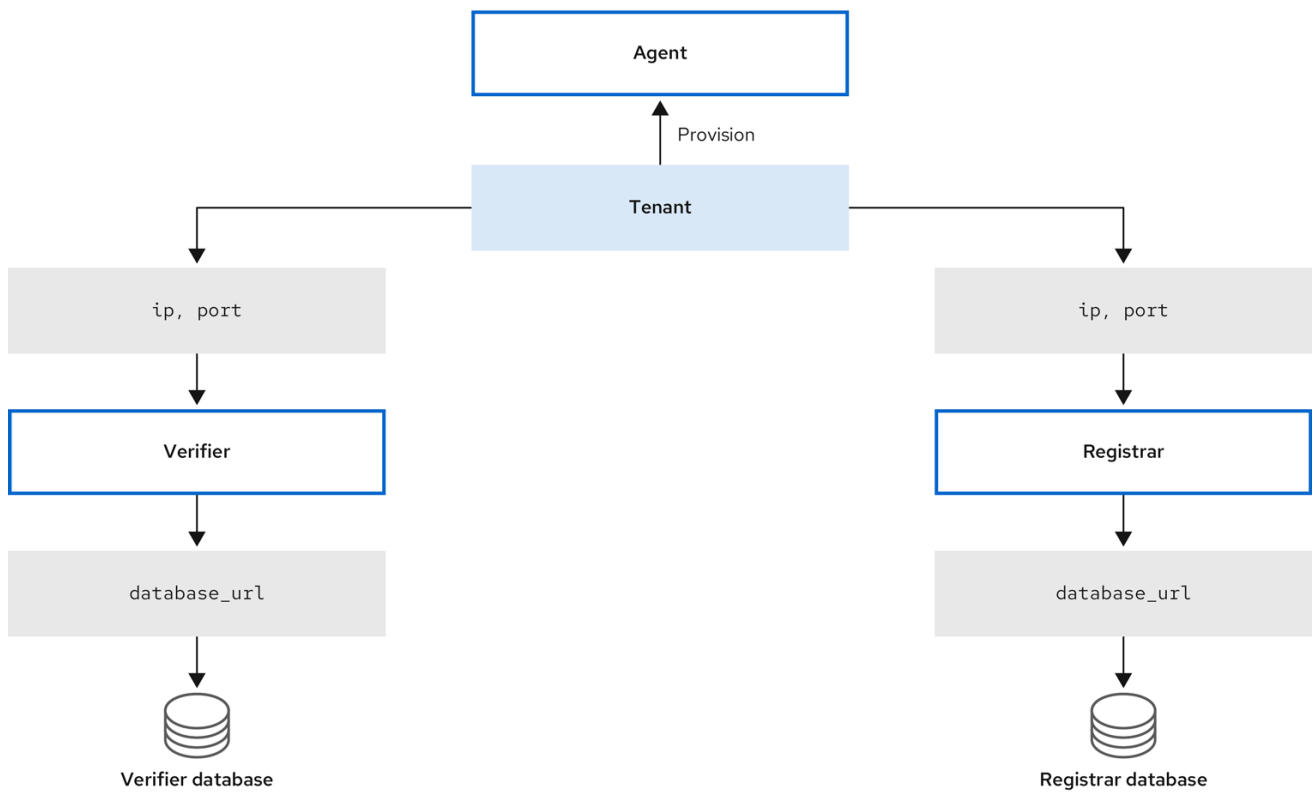
すべてのエージェントのデータベースを含んでおり、TPM ベンダーの公開鍵をホストします。registrar は、パッケージからデプロイすることも、コンテナとしてデプロイすることも、**keylime_server** RHEL システムロールを使用してデプロイすることもできます。

エージェント

verifier によって測定されるリモートシステムにデプロイされます。

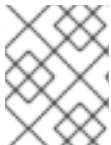
さらに、Keylime は、ターゲットシステムでのエージェントのプロビジョニングを含む多くの機能に **keylime_tenant** ユーティリティーを使用します。

図7.1 設定による Keylime コンポーネント間の接続



379_RHEL_0923

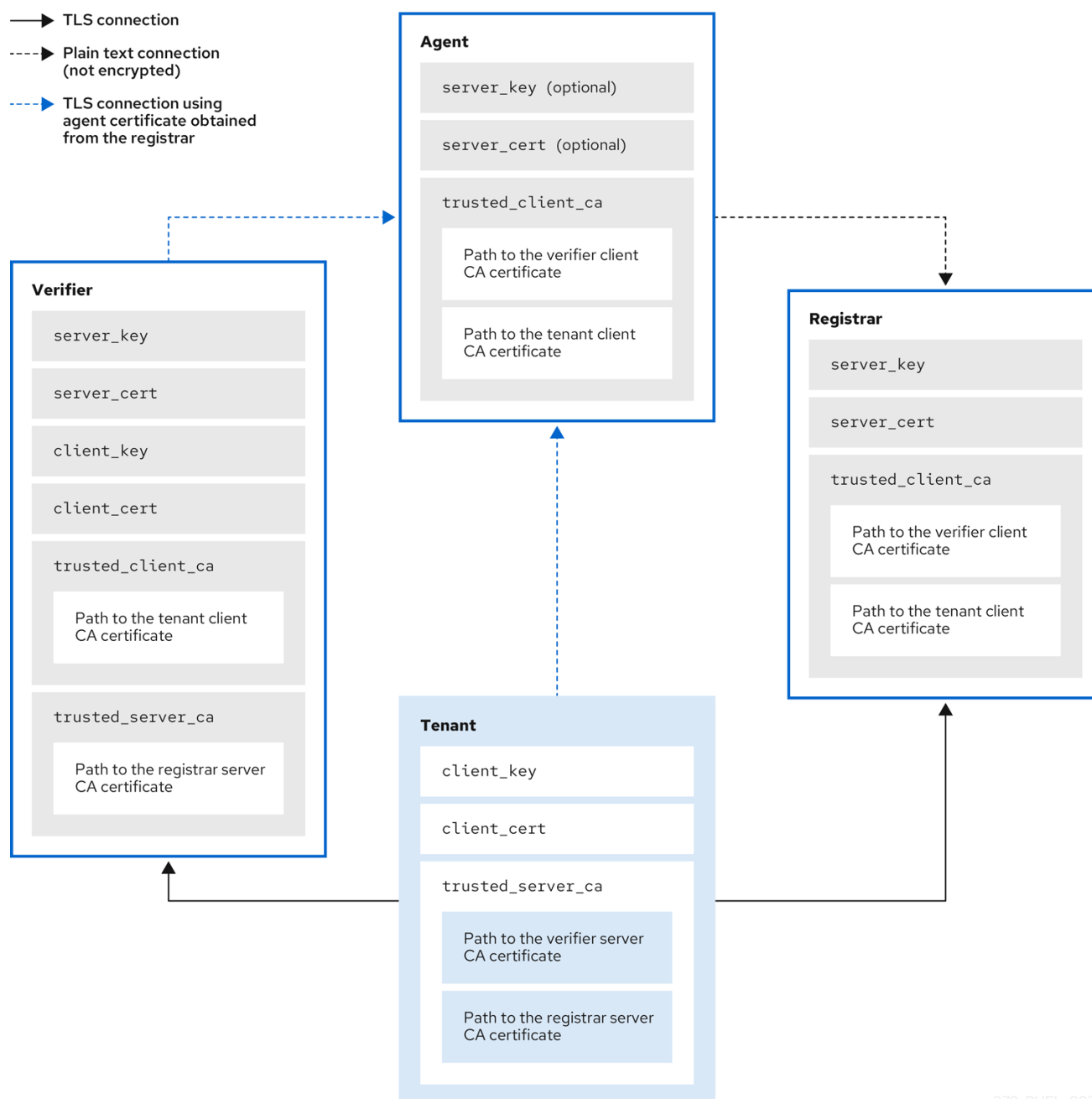
Keylime は、コンポーネントとテナントの間で交換される鍵と証明書を使用して、信頼の連鎖で監視対象システムの整合性を保証します。このチェーンの安全な基盤として、信頼できる認証局 (CA) を使用してください。



注記

エージェントが鍵と証明書を受け取らない場合は、CA の関与なしに鍵と自己署名証明書を生成します。

図7.2 Keylime コンポーネントの証明書と鍵の間の接続



379_RHEL_0923

7.2. パッケージから KEYLIME VERIFIER をデプロイする

verifier は、Keylime で最も重要なコンポーネントです。システム整合性の初期および定期的なチェックを行い、エージェントを使用して暗号化鍵を安全にブートストラップすることをサポートします。verifier は、制御インターフェイスに相互 TLS 暗号化を使用します。



重要

信頼の連鎖を維持するには、verifier を実行するシステムをセキュアに管理してください。

要件に応じて、verifier を別のシステムにインストールすることも、Keylime registrar と同じシステムにインストールすることもできます。verifier と registrar を別々のシステムで実行すると、パフォーマンスが向上します。



注記

設定ファイルをドロップインディレクトリー内に整理するには、`/etc/keylime/verifier.conf.d/00-verifier-ip.conf` のように、2桁の数字の接頭辞を付けたファイル名を使用します。設定処理は、ドロップインディレクトリー内のファイルを辞書順で読み取り、各オプションを最後に読み取った値に設定します。

前提条件

- **root** 権限と、Keylime コンポーネントをインストールするシステムへのネットワーク接続がある。
- 認証局からの有効な鍵と証明書がある。
- オプション: Keylime が verifier からのデータを保存するデータベースにアクセスできます。次のデータベース管理システムのいずれかを使用できます。
 - SQLite (デフォルト)
 - PostgreSQL
 - MySQL
 - MariaDB

手順

1. Keylime verifier をインストールします。

```
# dnf install keylime-verifier
```

2. `/etc/keylime/verifier.conf.d/` ディレクトリーに、次の内容の新しい **.conf** ファイル (`/etc/keylime/verifier.conf.d/00-verifier-ip.conf` など) を作成して、verifier の IP アドレスとポートを定義します。

```
[verifier]
ip = <verifier_IP_address>
```

- **<verifier_IP_address>** は、verifier の IP アドレスに置き換えます。あるいは、**ip = *** または **ip = 0.0.0.0** を使用して、使用可能なすべての IP アドレスに verifier をバインドします。
 - 必要に応じて、**port** オプションを使用して、verifier のポートをデフォルト値 **8881** から変更することもできます。
3. オプション: エージェントのリスト用に verifier のデータベースを設定します。デフォルトの設定では、verifier の `/var/lib/keylime/cv_data.sqlite` ディレクトリーにある SQLite データベースを使用します。`/etc/keylime/verifier.conf.d/` ディレクトリーに次の内容の新しい **.conf** ファイル (`/etc/keylime/verifier.conf.d/00-db-url.conf` など) を作成することで、別のデータベースを定義できます。

```
[verifier]
database_url = <protocol>://<name>:
<password>@<ip_address_or_hostname>/<properties>
```

`<protocol>://<name>:<password>@<ip_address_or_hostname>/<properties>` は、データベースの URL (例: `postgresql://verifier:UQ?nRNY9g7GZzN7@198.51.100.1/verifierdb`) に置き換えます。

使用する認証情報が、Keylime にデータベース構造を作成するための権限を提供していることを確認してください。

4. verifier に証明書と鍵を追加します。Keylime にそれらを生成させることも、既存の鍵と証明書を使用することもできます。

- デフォルトの `tls_dir =generate` オプションを使用すると、Keylime は verifier、registrar、およびテナントの新しい証明書を `/var/lib/keylime/cv_ca/` ディレクトリーに生成します。
- 既存の鍵と証明書を設定にロードするには、verifier 設定でそれらの場所を定義します。Keylime サービスの実行者である **keylime** ユーザーが証明書にアクセスできる必要があります。

`/etc/keylime/verifier.conf.d/` ディレクトリーに、次の内容の新しい `.conf` ファイル (`/etc/keylime/verifier.conf.d/00-keys-and-certs.conf` など) を作成します。

```
[verifier]
tls_dir = /var/lib/keylime/cv_ca
server_key = </path/to/server_key>
server_key_password = <passphrase1>
server_cert = </path/to/server_cert>
trusted_client_ca = ['</path/to/ca/cert1>', '</path/to/ca/cert2>']
client_key = </path/to/client_key>
client_key_password = <passphrase2>
client_cert = </path/to/client_cert>
trusted_server_ca = ['</path/to/ca/cert3>', '</path/to/ca/cert4>']
```



注記

絶対パスを使用して、鍵と証明書の場所を定義します。また、相対パスは `tls_dir` オプションで定義されたディレクトリーから解決されます。

5. ファイアウォールでポートを開きます。

```
# firewall-cmd --add-port 8881/tcp
# firewall-cmd --runtime-to-permanent
```

別のポートを使用する場合は、**8881** を `.conf` ファイルで定義されているポート番号に置き換えます。

6. verifier サービスを開始します。

```
# systemctl enable --now keylime_verifier
```



注記

デフォルト設定では、verifier が他の Keylime コンポーネントの CA と証明書を作成するため、**keylime_registrar** サービスを起動する前に **keylime_verifier** を起動します。カスタム証明書を使用する場合、この順序で起動する必要はありません。

検証

- **keylime_verifier** サービスがアクティブで実行中であることを確認します。

```
# systemctl status keylime_verifier
● keylime_verifier.service - The Keylime verifier
   Loaded: loaded (/usr/lib/systemd/system/keylime_verifier.service; disabled; vendor preset:
   disabled)
   Active: active (running) since Wed 2022-11-09 10:10:08 EST; 1min 45s ago
```

次のステップ

- 「[パッケージから Keylime registrar をデプロイする](#)」。

7.3. KEYLIME VERIFIER をコンテナとしてデプロイする

Keylime verifier は、システム整合性の初期チェックと定期チェックを実行し、エージェントを使用した暗号化鍵のセキュアなブートストラップをサポートします。Keylime verifier は、ホスト上にバイナリやパッケージがなくても、RPM 方式ではなくコンテナとして設定できます。コンテナとしてデプロイすることにより、Keylime コンポーネントの分離性、モジュール性、再現性が向上します。

コンテナを起動すると、Keylime verifier がデフォルトの設定ファイルとともにデプロイされます。次の1つ以上の方法を使用して設定をカスタマイズできます。

- 設定ファイルを含むホストのディレクトリーをコンテナにマウントします。これは、RHEL 9 のすべてのバージョンで使用できます。
- コンテナで環境変数を直接変更します。これは、RHEL 9.3 以降のバージョンで使用できます。環境変数を変更すると、設定ファイルの値がオーバーライドされます。

前提条件

- **podman** パッケージとその依存関係がシステムにインストールされている。
- オプション: Keylime が verifier からのデータを保存するデータベースにアクセスできます。次のデータベース管理システムのいずれかを使用できます。
 - SQLite (デフォルト)
 - PostgreSQL
 - MySQL
 - MariaDB
- 認証局からの有効な鍵と証明書がある。

手順

1. オプション: 設定ファイルにアクセスするには、**keylime-verifier** パッケージをインストールします。このパッケージがなくてもコンテナを設定することはできますが、パッケージに付属する設定ファイルを変更する方が簡単な場合があります。

```
# dnf install keylime-verifier
```

2. `/etc/keylime/verifier.conf.d/` ディレクトリーに新しい `.conf` ファイル (例: `/etc/keylime/verifier.conf.d/00-verifier-ip.conf`) を作成し、次の内容を記述して、`verifier` をすべての使用可能な IP アドレスにバインドします。

```
[verifier]
ip = *
```

- 必要に応じて、`port` オプションを使用して、`verifier` のポートをデフォルト値 **8881** から変更することもできます。

3. オプション: エージェントのリスト用に `verifier` のデータベースを設定します。デフォルトの設定では、`verifier` の `/var/lib/keylime/cv_data.sqlite` ディレクトリーにある SQLite データベースを使用します。`/etc/keylime/verifier.conf.d/` ディレクトリーに次の内容の新しい `.conf` ファイル (`/etc/keylime/verifier.conf.d/00-db-url.conf` など) を作成することで、別のデータベースを定義できます。

```
[verifier]
database_url = <protocol>://<name>:
<password>@<ip_address_or_hostname>/<properties>
```

`<protocol>://<name>:<password>@<ip_address_or_hostname>/<properties>` は、データベースの URL (例: `postgresql://verifier:UQ?nRNY9g7GZzN7@198.51.100.1/verifierdb`) に置き換えます。

使用する認証情報に、Keylime がデータベース構造を作成するための権限があることを確認してください。

4. `verifier` に証明書と鍵を追加します。Keylime にそれらを生成させることも、既存の鍵と証明書を使用することもできます。
 - デフォルトの `tls_dir =generate` オプションを使用すると、Keylime は `verifier`、`registrar`、およびテナントの新しい証明書を `/var/lib/keylime/cv_ca/` ディレクトリーに生成します。
 - 既存の鍵と証明書を設定にロードするには、`verifier` 設定でそれらの場所を定義します。Keylime プロセスの実行者である **keylime** ユーザーが証明書にアクセスできる必要があります。
`/etc/keylime/verifier.conf.d/` ディレクトリーに、次の内容の新しい `.conf` ファイル (`/etc/keylime/verifier.conf.d/00-keys-and-certs.conf` など) を作成します。

```
[verifier]
tls_dir = /var/lib/keylime/cv_ca
server_key = </path/to/server_key>
server_cert = </path/to/server_cert>
trusted_client_ca = ['</path/to/ca/cert1>', '</path/to/ca/cert2>']
client_key = </path/to/client_key>
client_cert = </path/to/client_cert>
trusted_server_ca = ['</path/to/ca/cert3>', '</path/to/ca/cert4>']
```



注記

絶対パスを使用して、鍵と証明書の場所を定義します。また、相対パスは `tls_dir` オプションで定義されたディレクトリーから解決されます。

5. ファイアウォールでポートを開きます。

```
# firewall-cmd --add-port 8881/tcp
# firewall-cmd --runtime-to-permanent
```

別のポートを使用する場合は、**8881** を **.conf** ファイルで定義されているポート番号に置き換えます。

6. コンテナを実行します。

```
$ podman run --name keylime-verifier \
-p 8881:8881 \
-v /etc/keylime/verifier.conf.d:/etc/keylime/verifier.conf.d:Z \
-v /var/lib/keylime/cv_ca:/var/lib/keylime/cv_ca:Z \
-d \
-e KEYLIME_VERIFIER_SERVER_KEY_PASSWORD=<passphrase1> \
-e KEYLIME_VERIFIER_CLIENT_KEY_PASSWORD=<passphrase2> \
registry.access.redhat.com/rhel9/keylime-verifier
```

- **-p** オプションは、ホスト上とコンテナ上のデフォルトポート **8881** を開きます。
- **-v** オプションは、コンテナへのディレクトリーのバインドマウントを作成します。
 - **Z** オプションを指定すると、Podman がコンテンツにプライベート非共有ラベルを付けます。つまり、現在のコンテナだけがプライベートボリュームを使用できます。
- **-d** オプションは、コンテナをデタッチしてバックグラウンドで実行します。
- オプション **-e KEYLIME_VERIFIER_SERVER_KEY_PASSWORD=<passphrase1>** は、サーバーの鍵のパスフレーズを定義します。
- オプション **-e KEYLIME_VERIFIER_CLIENT_KEY_PASSWORD=<passphrase2>** は、クライアントの鍵のパスフレーズを定義します。
- オプション **-e KEYLIME_VERIFIER_<ENVIRONMENT_VARIABLE>=<value>** を指定すると、環境変数で設定オプションをオーバーライドできます。複数のオプションを変更するには、環境変数ごとに **-e** オプションを個別に挿入します。環境変数とそのデフォルト値の完全なリストは、[Keylime の環境変数](#) を参照してください。

検証

- コンテナが実行されていることを確認します。

```
$ podman ps -a
CONTAINER ID  IMAGE                                COMMAND                                CREATED
STATUS       PORTS                                NAMES
80b6b9dbf57c registry.access.redhat.com/rhel9/keylime-verifier:latest keylime_verifier 14
seconds ago  Up 14 seconds  0.0.0.0:8881->8881/tcp  keylime-verifier
```

次のステップ

- [Keylime registrar](#) をコンテナとして設定する

関連情報

- Keylime コンポーネントの詳細は、[Keylime の仕組み](#) を参照してください。

- Keylime 検証ツールの設定の詳細は、[Keylime verifier の設定](#) を参照してください。
- **podman run** コマンドの詳細は、システム上の **podman-run(1)** man ページを参照してください。

7.4. パッケージから KEYLIME REGISTRAR をデプロイする

registrar は、すべてのエージェントのデータベースを含む Keylime コンポーネントであり、TPM ベンダーの公開鍵をホストします。registrar の HTTPS サービスは、Trusted Platform Module (TPM) 公開鍵を受け入れると、クォートを確認するためにこれらの公開鍵を取得するインターフェイスを提供します。



重要

信頼の連鎖を維持するには、registrar を実行するシステムをセキュアに管理してください。

要件に応じて、registrar を別のシステムにインストールすることも、Keylime verifier と同じシステムにインストールすることもできます。verifier と registrar を別々のシステムで実行すると、パフォーマンスが向上します。



注記

設定ファイルをドロップインディレクトリー内に整理するには、**/etc/keylime/registrar.conf.d/00-registrar-ip.conf** のように、2桁の数字の接頭辞を付けたファイル名を使用します。設定処理は、ドロップインディレクトリー内のファイルを辞書順で読み取り、各オプションを最後に読み取った値に設定します。

前提条件

- Keylime verifier がインストールされ実行されているシステムへのネットワークアクセスがある。詳細は、「[パッケージから Keylime verifier をデプロイする](#)」を参照してください。
- **root** 権限と、Keylime コンポーネントをインストールするシステムへのネットワーク接続がある。
- Keylime が registrar からのデータを保存するデータベースにアクセスできる。次のデータベース管理システムのいずれかを使用できます。
 - SQLite (デフォルト)
 - PostgreSQL
 - MySQL
 - MariaDB
- 認証局からの有効な鍵と証明書がある。

手順

1. Keylime registrar をインストールします。

```
# dnf install keylime-registrar
```

2. `/etc/keylime/registrar.conf.d/` ディレクトリーに、次の内容の新しい `.conf` ファイル (`/etc/keylime/registrar.conf.d/00-registrar-ip.conf` など) を作成して、registrar の IP アドレスとポートを定義します。

```
[registrar]
ip = <registrar_IP_address>
```

- `<registrar_IP_address>` を registrar の IP アドレスに置き換えます。あるいは、`ip = *` または `ip = 0.0.0.0` を使用して、使用可能なすべての IP アドレスに registrar をバインドします。
 - 必要に応じて、`port` オプションを使用して、Keylime エージェントが接続するポートを変更します。デフォルト値は **8890** です。
 - 必要に応じて、`tls_port` オプションを使用して、Keylime verifier とテナントが接続する TLS ポートを変更します。デフォルト値は **8891** です。
3. オプション: エージェントのリスト用に registrar のデータベースを設定します。デフォルト設定では、registrar の `/var/lib/keylime/reg_data.sqlite` ディレクトリーにある SQLite データベースを使用します。`/etc/keylime/registrar.conf.d/` ディレクトリーに、次の内容の新しい `.conf` ファイル (`/etc/keylime/registrar.conf.d/00-db-url.conf` など) を作成できます。

```
[registrar]
database_url = <protocol>://<name>:
<password>@<ip_address_or_hostname>/<properties>
```

`<protocol>://<name>:<password>@<ip_address_or_hostname>/<properties>` は、データベースの URL (例: `postgresql://registrar:EKYYX-bqY2?#raXm@198.51.100.1/registrardb`) に置き換えます。

使用する認証情報に、Keylime がデータベース構造を作成するための権限があることを確認してください。

4. registrar に証明書と鍵を追加します。
 - デフォルトの設定を使用して、鍵と証明書を `/var/lib/keylime/reg_ca` ディレクトリーにロードできます。
 - または、設定で鍵と証明書の場所を定義することもできます。`/etc/keylime/registrar.conf.d/` ディレクトリーに新しい `.conf` ファイルを作成します (例: `/etc/keylime/registrar.conf.d/00-keys-and-certs.conf` の内容は次のとおりです)。

```
[registrar]
tls_dir = /var/lib/keylime/reg_ca
server_key = </path/to/server_key>
server_key_password = <passphrase1>
server_cert = </path/to/server_cert>
trusted_client_ca = ['</path/to/ca/cert1>', '</path/to/ca/cert2>']
```



注記

絶対パスを使用して、鍵と証明書の場所を定義します。または、`tls_dir` オプションでディレクトリーを定義し、そのディレクトリーからの相対パスを使用することもできます。

- ファイアウォールでポートを開きます。

```
# firewall-cmd --add-port 8890/tcp --add-port 8891/tcp
# firewall-cmd --runtime-to-permanent
```

別のポートを使用する場合は、**8890** または **8891** を **.conf** ファイルで定義されているポート番号に置き換えます。

- keylime_registrar** サービスを起動します。

```
# systemctl enable --now keylime_registrar
```



注記

デフォルト設定では、verifier が他の Keylime コンポーネントの CA と証明書を作成するため、**keylime_registrar** サービスを起動する前に **keylime_verifier** を起動します。カスタム証明書を使用する場合、この順序で起動する必要はありません。

検証

- keylime_registrar** サービスがアクティブで実行中であることを確認します。

```
# systemctl status keylime_registrar
● keylime_registrar.service - The Keylime registrar service
   Loaded: loaded (/usr/lib/systemd/system/keylime_registrar.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2022-11-09 10:10:17 EST; 1min 42s ago
   ...
```

次のステップ

- [「パッケージから Keylime テナントをデプロイする」](#)

7.5. KEYLIME REGISTRAR をコンテナとしてデプロイする

registrar は、すべてのエージェントのデータベースを格納する Keylime コンポーネントであり、Trusted Platform Module (TPM) ベンダーの公開鍵をホストします。registrar の HTTPS サービスは、TPM 公開鍵を受け入れると、クォートをチェックするために、この公開鍵を取得するためのインターフェイスを提供します。Keylime registrar は、ホスト上にバイナリーやパッケージがなくても、RPM 方式ではなくコンテナとして設定できます。コンテナとしてデプロイすることにより、Keylime コンポーネントの分離性、モジュール性、再現性が向上します。

コンテナを起動すると、Keylime registrar がデフォルトの設定ファイルとともにデプロイされます。次の1つ以上の方法を使用して設定をカスタマイズできます。

- 設定ファイルを含むホストのディレクトリーをコンテナにマウントします。これは、RHEL 9 のすべてのバージョンで使用できます。
- コンテナで環境変数を直接変更します。これは、RHEL 9.3 以降のバージョンで使用できます。環境変数を変更すると、設定ファイルの値がオーバーライドされます。

前提条件

- **podman** パッケージとその依存関係がシステムにインストールされている。
- オプション: Keylime が registrar からデータを保存するデータベースにアクセスできます。次のデータベース管理システムのいずれかを使用できます。
 - SQLite (デフォルト)
 - PostgreSQL
 - MySQL
 - MariaDB
- 認証局からの有効な鍵と証明書がある。

手順

1. オプション: 設定ファイルにアクセスするには、**keylime-registrar** パッケージをインストールします。このパッケージがなくてもコンテナを設定することはできますが、パッケージに付属する設定ファイルを変更する方が簡単な場合があります。

```
# dnf install keylime-registrar
```

2. **/etc/keylime/registrar.conf.d/** ディレクトリーに新しい **.conf** ファイル (例: **/etc/keylime/registrar.conf.d/00-registrar-ip.conf**) を作成し、次の内容を記述して、registrar を使用可能なすべての IP アドレスにバインドします。

```
[registrar]
ip = *
```

- 必要に応じて、**port** オプションを使用して、Keylime エージェントが接続するポートを変更します。デフォルト値は **8890** です。
 - 必要に応じて、**tls_port** オプションを使用して、Keylime テナントが接続する TLS ポートを変更します。デフォルト値は **8891** です。
3. オプション: エージェントのリスト用に registrar のデータベースを設定します。デフォルト設定では、registrar の **/var/lib/keylime/reg_data.sqlite** ディレクトリーにある SQLite データベースを使用します。**/etc/keylime/registrar.conf.d/** ディレクトリーに、次の内容の新しい **.conf** ファイル (**/etc/keylime/registrar.conf.d/00-db-url.conf** など) を作成できます。

```
[registrar]
database_url = <protocol>://<name>:
<password>@<ip_address_or_hostname>/<properties>
```

<protocol>://<name>:<password>@<ip_address_or_hostname>/<properties> は、データベースの URL (例: **postgresql://registrar:EKYYX-bqY2?#raXm@198.51.100.1/registrardb**) に置き換えます。

使用する認証情報に、Keylime がデータベース構造を作成するための権限があることを確認してください。

4. registrar に証明書と鍵を追加します。
 - デフォルトの設定を使用して、鍵と証明書を **/var/lib/keylime/reg_ca** ディレクトリーに

ポートごさます。

- または、設定で鍵と証明書の場所を定義することもできます。
/etc/keylime/registrar.conf.d/ ディレクトリーに新しい .conf ファイルを作成します
(例: /etc/keylime/registrar.conf.d/00-keys-and-certs.conf の内容は次のとおりです)。

```
[registrar]
tls_dir = /var/lib/keylime/reg_ca
server_key = </path/to/server_key>
server_cert = </path/to/server_cert>
trusted_client_ca = ['</path/to/ca/cert1>', '</path/to/ca/cert2>']
```



注記

絶対パスを使用して、鍵と証明書の場所を定義します。または、**tls_dir** オプションでディレクトリーを定義し、そのディレクトリーからの相対パスを使用することもできます。

5. ファイアウォールでポートを開きます。

```
# firewall-cmd --add-port 8890/tcp --add-port 8891/tcp
# firewall-cmd --runtime-to-permanent
```

別のポートを使用する場合は、**8890** または **8891** を .conf ファイルで定義されているポート番号に置き換えます。

6. コンテナを実行します。

```
$ podman run --name keylime-registrar \
-p 8890:8890 \
-p 8891:8891 \
-v /etc/keylime/registrar.conf.d:/etc/keylime/registrar.conf.d:Z \
-v /var/lib/keylime/reg_ca:/var/lib/keylime/reg_ca:Z \
-d \
-e KEYLIME_REGISTRAR_SERVER_KEY_PASSWORD=<passphrase1> \
registry.access.redhat.com/rhel9/keylime-registrar
```

- **-p** オプションは、ホスト上とコンテナ上のデフォルトポート **8890** および **8881** を開きます。
- **-v** オプションは、コンテナへのディレクトリーのバインドマウントを作成します。
 - **Z** オプションを指定すると、Podman がコンテンツにプライベート非共有ラベルを付けます。つまり、現在のコンテナだけがプライベートボリュームを使用できます。
- **-d** オプションは、コンテナをデタッチしてバックグラウンドで実行します。
- オプション **-e KEYLIME_REGISTRAR_SERVER_KEY_PASSWORD=<passphrase1>** は、サーバーの鍵のパスフレーズを定義します。
- オプション **-e KEYLIME_REGISTRAR_<ENVIRONMENT_VARIABLE>=<value>** を指定すると、環境変数で設定オプションをオーバーライドできます。複数のオプションを変更するには、環境変数ごとに **-e** オプションを個別に挿入します。環境変数とそのデフォルト値の完全なリストは、「[Keylime の環境変数](#)」を参照してください。

検証

- コンテナが実行されていることを確認します。

```
$ podman ps -a
CONTAINER ID  IMAGE                                COMMAND                                CREATED      STATUS
PORTS        NAMES
07d4b4bff1b6  localhost/keylime-registrar:latest  keylime_registrar  12 seconds ago  Up 12
seconds      0.0.0.0:8881->8881/tcp, 0.0.0.0:8891->8891/tcp  keylime-registrar
```

次のステップ

- [「パッケージから Keylime テナントをデプロイする」](#)。

関連情報

- Keylime コンポーネントの詳細は、[「Keylime の仕組み」](#) を参照してください。
- Keylime registrar の設定の詳細は、[「パッケージから Keylime registrar をデプロイする」](#) を参照してください。
- **podman run** コマンドの詳細は、システム上の **podman-run(1)** man ページを参照してください。

7.6. RHEL システムロールを使用して KEYLIME サーバーをデプロイする

keylime_server RHEL システムロールを使用して、Keylime サーバーのコンポーネントである verifier と registrar をセットアップできます。**keylime_server** ロールは、verifier コンポーネントと registrar コンポーネントの両方を各ノードに共にインストールして設定します。

Ansible コントロールノードで以下の手順を実行します。

Keylime の詳細は、[8.1. Keylime の仕組み](#) を参照してください。

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。
- この Playbook を実行する管理対象ノードまたは管理対象ノードのグループが、Ansible インベントリファイルにリストされている。

手順

1. 必要なロールを定義する Playbook を作成します。
 - a. 新しい YAML ファイルを作成し、これをテキストエディターで開きます。以下に例を示します。

```
# vi keylime-playbook.yml
```

- b. 以下の内容を入力します。

```
---
- name: Manage keylime servers
  hosts: all
  vars:
    keylime_server_verifier_ip: "{{ ansible_host }}"
    keylime_server_registrar_ip: "{{ ansible_host }}"
    keylime_server_verifier_tls_dir: <ver_tls_directory>
    keylime_server_verifier_server_cert: <ver_server_certfile>
    keylime_server_verifier_server_key: <ver_server_key>
    keylime_server_verifier_server_key_passphrase: <ver_server_key_passphrase>
    keylime_server_verifier_trusted_client_ca: <ver_trusted_client_ca_list>
    keylime_server_verifier_client_cert: <ver_client_certfile>
    keylime_server_verifier_client_key: <ver_client_key>
    keylime_server_verifier_client_key_passphrase: <ver_client_key_passphrase>
    keylime_server_verifier_trusted_server_ca: <ver_trusted_server_ca_list>
    keylime_server_registrar_tls_dir: <reg_tls_directory>
    keylime_server_registrar_server_cert: <reg_server_certfile>
    keylime_server_registrar_server_key: <reg_server_key>
    keylime_server_registrar_server_key_passphrase: <reg_server_key_passphrase>
    keylime_server_registrar_trusted_client_ca: <reg_trusted_client_ca_list>
  roles:
    - rhel-system-roles.keylime_server
```

変数の詳細は、[keylime_server RHEL システムロールの変数](#) を参照してください。

2. Playbook を実行します。

```
$ ansible-playbook <keylime-playbook.yml>
```

検証

1. **keylime_verifier** サービスがアクティブであり、管理対象ホスト上で実行されていることを確認します。

```
# systemctl status keylime_verifier
● keylime_verifier.service - The Keylime verifier
   Loaded: loaded (/usr/lib/systemd/system/keylime_verifier.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2022-11-09 10:10:08 EST; 1min 45s ago
```

2. **keylime_registrar** サービスがアクティブで実行中であることを確認します。

```
# systemctl status keylime_registrar
● keylime_registrar.service - The Keylime registrar service
   Loaded: loaded (/usr/lib/systemd/system/keylime_registrar.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2022-11-09 10:10:17 EST; 1min 42s ago
...
```

次のステップ

[「パッケージから Keylime テナントをデプロイする」](#)

7.7. KEYLIME_SERVER RHEL システムロールの変数

keylime_server RHEL システムロールを使用して Keylime サーバーをセットアップする場合、registrar と verifier の次の変数をカスタマイズできます。

Keylime verifier を設定するための **keylime_server** RHEL システムロール変数のリスト

keylime_server_verifier_ip

verifier の IP アドレスを定義します。

keylime_server_verifier_tls_dir

キーと証明書が保存されるディレクトリーを指定します。デフォルトに設定されている場合、verifier は **/var/lib/keylime/cv_ca** ディレクトリーを使用します。

keylime_server_verifier_server_key_passphrase

サーバーの秘密鍵を復号化するためのパスフレーズを指定します。値が空の場合、秘密鍵は暗号化されません。

keylime_server_verifier_server_cert: Keylime verifier サーバー証明書ファイルを指定します。

keylime_server_verifier_trusted_client_ca

信頼できるクライアント CA 証明書のリストを定義します。ファイルは、**keylime_server_verifier_tls_dir** オプションで設定されたディレクトリーに保存する必要があります。

keylime_server_verifier_client_key

Keylime verifier のクライアントの秘密鍵を含むファイルを定義します。

keylime_server_verifier_client_key_passphrase

クライアントの秘密鍵ファイルを復号化するためのパスフレーズを定義します。値が空の場合、秘密鍵は暗号化されません。

keylime_server_verifier_client_cert

Keylime verifier クライアント証明書ファイルを定義します。

keylime_server_verifier_trusted_server_ca

信頼できるサーバー CA 証明書のリストを定義します。ファイルは、**keylime_server_verifier_tls_dir** オプションで設定されたディレクトリーに保存する必要があります。

keylime_server RHEL システムロールをセットアップするための registrar 変数のリスト

keylime_server_registrar_ip

registrar の IP アドレスを定義します。

keylime_server_registrar_tls_dir

registrar のキーと証明書を保存するディレクトリーを指定します。デフォルトに設定すると、registrar は **/var/lib/keylime/reg_ca** ディレクトリーを使用します。

keylime_server_registrar_server_key

Keylime registrar のプライベートサーバーキーファイルを定義します。

keylime_server_registrar_server_key_passphrase

registrar のサーバー秘密鍵を復号化するためのパスフレーズを指定します。値が空の場合、秘密鍵は暗号化されません。

keylime_server_registrar_server_cert

Keylime registrar サーバー証明書ファイルを指定します。

keylime_server_registrar_trusted_client_ca

信頼できるクライアント CA 証明書のリストを定義します。ファイルは、**keylime_server_registrar_tls_dir** オプションで設定されたディレクトリーに保存する必要があります。

7.8. パッケージから KEYLIME テナントをデプロイする

Keylime は、ターゲットシステム上でのエージェントのプロビジョニングなど、多くの機能に **keylime_tenant** ユーティリティーを使用します。**keylime_tenant** は、要件に応じて、他の Keylime コンポーネントを実行するシステムを含む任意のシステムにインストールすることも、別のシステムにインストールすることもできます。

前提条件

- **root** 権限と、Keylime コンポーネントをインストールするシステムへのネットワーク接続がある。
- 他の Keylime コンポーネントが設定されているシステムへのネットワークアクセスがある。

verifier

詳細は、[「パッケージから Keylime verifier をデプロイする」](#) を参照してください。

registrar

詳細は、[「パッケージから Keylime registrar をデプロイする」](#) を参照してください。

手順

1. Keylime テナントをインストールします。

```
# dnf install keylime-tenant
```

2. **/etc/keylime/tenant.conf.d/00-verifier-ip.conf** ファイルを編集して、Keylime verifier へのテナントの接続を定義します。

```
[tenant]
verifier_ip = <verifier_ip>
```

- **<verifier_ip>** は、verifier のシステムの IP アドレスに置き換えます。
- verifier がデフォルト値 **8881** とは異なるポートを使用する場合は、**verifier_port = <verifier_port>** 設定を追加します。

3. **/etc/keylime/tenant.conf.d/00-registrar-ip.conf** ファイルを編集して、Keylime registrar へのテナントの接続を定義します。

```
[tenant]
registrar_ip = <registrar_ip>
```

- **<registrar_ip>** は、registrar のシステムの IP アドレスに置き換えます。

- registrar がデフォルト値 **8891** とは異なるポートを使用する場合は、**registrar_port = <registrar_port>** 設定を追加します。

4. テナントに証明書と鍵を追加します。

- デフォルトの設定を使用して、鍵と証明書を `/var/lib/keylime/cv_ca` ディレクトリーにロードできます。
- または、設定で鍵と証明書の場所を定義することもできます。`/etc/keylime/tenant.conf.d/` ディレクトリーに、次の内容の新しい `.conf` ファイル (`/etc/keylime/tenant.conf.d/00-keys-and-certs.conf` など) を作成します。

```
[tenant]
tls_dir = /var/lib/keylime/cv_ca
client_key = tenant-key.pem
client_key_password = <passphrase1>
client_cert = tenant-cert.pem
trusted_server_ca = ['</path/to/ca/cert>']
```

trusted_server_ca パラメーターは、verifier および registrar サーバーの CA 証明書へのパスを受け入れます。verifier と registrar が異なる CA を使用する場合などに、複数のコンマ区切りのパスを指定できます。



注記

絶対パスを使用して、鍵と証明書の場所を定義します。または、**tls_dir** オプションでディレクトリーを定義し、そのディレクトリーからの相対パスを使用することもできます。

- オプション: `/var/lib/keylime/tpm_cert_store` ディレクトリー内の証明書を使用して Trusted Platform Module (TPM) の保証鍵 (EK) を検証できない場合は、証明書をそのディレクトリーに追加します。これは、エミュレートされた TPM を備えた仮想マシンを使用する場合に特に発生する可能性があります。

検証

- verifier のステータスを確認します。

```
# keylime_tenant -c cvstatus
Reading configuration from ['/etc/keylime/logging.conf']
2022-10-14 12:56:08.155 - keylime.tpm - INFO - TPM2-TOOLS Version: 5.2
Reading configuration from ['/etc/keylime/tenant.conf']
2022-10-14 12:56:08.157 - keylime.tenant - INFO - Setting up client TLS...
2022-10-14 12:56:08.158 - keylime.tenant - INFO - Using default client_cert option for tenant
2022-10-14 12:56:08.158 - keylime.tenant - INFO - Using default client_key option for tenant
2022-10-14 12:56:08.178 - keylime.tenant - INFO - TLS is enabled.
2022-10-14 12:56:08.178 - keylime.tenant - WARNING - Using default UUID d432fbb3-d2f1-4a97-9ef7-75bd81c00000
2022-10-14 12:56:08.221 - keylime.tenant - INFO - Verifier at 127.0.0.1 with Port 8881 does not have agent d432fbb3-d2f1-4a97-9ef7-75bd81c00000.
```

正しくセットアップされていて、エージェントが設定されていない場合、verifier は、デフォルトのエージェント UUID を認識しないと応答します。

- registrar のステータスを確認します。

```
# keylime_tenant -c regstatus
Reading configuration from ['/etc/keylime/logging.conf']
2022-10-14 12:56:02.114 - keylime.tpm - INFO - TPM2-TOOLS Version: 5.2
Reading configuration from ['/etc/keylime/tenant.conf']
2022-10-14 12:56:02.116 - keylime.tenant - INFO - Setting up client TLS...
2022-10-14 12:56:02.116 - keylime.tenant - INFO - Using default client_cert option for tenant
2022-10-14 12:56:02.116 - keylime.tenant - INFO - Using default client_key option for tenant
2022-10-14 12:56:02.137 - keylime.tenant - INFO - TLS is enabled.
2022-10-14 12:56:02.137 - keylime.tenant - WARNING - Using default UUID d432fbb3-d2f1-4a97-9ef7-75bd81c00000
2022-10-14 12:56:02.171 - keylime.registrar_client - CRITICAL - Error: could not get agent d432fbb3-d2f1-4a97-9ef7-75bd81c00000 data from Registrar Server: 404
2022-10-14 12:56:02.172 - keylime.registrar_client - CRITICAL - Response code 404: agent d432fbb3-d2f1-4a97-9ef7-75bd81c00000 not found
2022-10-14 12:56:02.172 - keylime.tenant - INFO - Agent d432fbb3-d2f1-4a97-9ef7-75bd81c00000 does not exist on the registrar. Please register the agent with the registrar.
2022-10-14 12:56:02.172 - keylime.tenant - INFO - {"code": 404, "status": "Agent d432fbb3-d2f1-4a97-9ef7-75bd81c00000 does not exist on registrar 127.0.0.1 port 8891.", "results": {}}
```

正しくセットアップされていて、エージェントが設定されていない場合、registrar は、デフォルトのエージェント UUID を認識しないと応答します。

関連情報

- **keylime_tenant** ユーティリティーの追加の詳細オプションは、**keylime_tenant -h** コマンドを入力します。

7.9. パッケージから KEYLIME エージェントをデプロイする

Keylime エージェントは、Keylime によって監視されるすべてのシステムにデプロイされるコンポーネントです。

デフォルトでは、Keylime エージェントはすべてのデータを監視対象システムの `/var/lib/keylime/` ディレクトリーに保存します。



注記

設定ファイルをドロップインディレクトリー内で整理するには、`/etc/keylime/agent.conf.d/00-registrar-ip.conf` のように、2 桁の数字の接頭辞を付けたファイル名を使用します。設定処理は、ドロップインディレクトリー内のファイルを辞書順で読み取り、各オプションを最後に読み取った値に設定します。

前提条件

- 監視対象システムに対する **root** 権限がある。
- 監視対象システムに Trusted Platform Module (TPM) が搭載されている。確認するには、**tpm2_pcrread** コマンドを入力します。出力が複数のハッシュを返す場合は、TPM が使用可能です。
- 他の Keylime コンポーネントが設定されているシステムへのネットワークアクセスがある。

verifier

詳細は、[Keylime verifier の設定](#) を参照してください。

registrar

詳細は、[Keylime registrar の設定](#) を参照してください。

テナント

詳細は、[Keylime テナントの設定](#) を参照してください。

- 監視対象システムで Integrity Measurement Architecture (IMA) が有効になっている。詳細は、[整合性測定アーキテクチャーと拡張検証モジュールの有効化](#) を参照してください。

手順

1. Keylime エージェントをインストールします。

```
# dnf install keylime-agent
```

このコマンドは、**keylime-agent-rust** パッケージをインストールします。

2. 設定ファイルでエージェントの IP アドレスとポートを定義します。`/etc/keylime/agent.conf.d/` ディレクトリーに、次の内容の新しい `.conf` ファイル (`/etc/keylime/agent.conf.d/00-agent-ip.conf` など) を作成します。

```
[agent]
ip = '<agent_ip>'
```



注記

Keylime エージェントの設定では TOML 形式が使用されます。これは、他のコンポーネントの設定に使用される INI 形式とは異なります。したがって、値は有効な TOML 構文で入力してください。たとえば、パスは一重引用符で囲み、複数のパスの配列は角括弧で囲みます。

- `<agent_ip_address>` は、エージェントの IP アドレスに置き換えます。あるいは、`ip = '*'` または `ip = '0.0.0.0'` を使用して、使用可能なすべての IP アドレスにエージェントをバインドします。
 - 必要に応じて、`port = '<agent_port>'` オプションを使用して、エージェントのポートをデフォルト値 **9002** から変更することもできます。
3. 設定ファイルで registrar の IP アドレスとポートを定義します。`/etc/keylime/agent.conf.d/` ディレクトリーに、次の内容の新しい `.conf` ファイル (`/etc/keylime/agent.conf.d/00-registrar-ip.conf` など) を作成します。

```
[agent]
registrar_ip = '<registrar_ip_address>'
```

- `<registrar_ip_address>` を registrar の IP アドレスに置き換えます。
 - 必要に応じて、`registrar_port = '<registrar_port>'` オプションを使用して、registrar のポートをデフォルト値 **8890** から変更することもできます。
4. オプション: エージェントの汎用一意識別子 (UUID) を定義します。定義されていない場合は、デフォルトの UUID が使用されます。`/etc/keylime/agent.conf.d/` ディレクトリーに、次の内容の新しい `.conf` ファイル (`/etc/keylime/agent.conf.d/00-agent-uuid.conf` など) を作成します。

```
[agent]
uuid = '<agent_UUID>'
```

- **<agent_UUID>** は、エージェントの UUID に置き換えます (例: **d432fbb3-d2f1-4a97-9ef7-abcdef012345**)。 **uuidgen** ユーティリティーを使用して UUID を生成できます。

5. オプション: エージェントの既存のキーと証明書を読み込みます。エージェントが **server_key** と **server_cert** を受信しない場合、エージェントは独自の鍵と自己署名証明書を生成します。設定で鍵と証明書の場所を定義します。 **/etc/keylime/agent.conf.d/** ディレクトリーに、次の内容の新しい **.conf** ファイル (**/etc/keylime/agent.conf.d/00-keys-and-certs.conf** など) を作成します。

```
[agent]
server_key = '</path/to/server_key>'
server_key_password = '<passphrase1>'
server_cert = '</path/to/server_cert>'
trusted_client_ca = ['</path/to/ca/cert3>', '</path/to/ca/cert4>']
```



注記

絶対パスを使用して、鍵と証明書の場所を定義します。Keylime エージェントは相対パスを受け入れません。

6. ファイアウォールでポートを開きます。

```
# firewall-cmd --add-port 9002/tcp
# firewall-cmd --runtime-to-permanent
```

別のポートを使用する場合は、**9002** を **.conf** ファイルで定義されているポート番号に置き換えます。

7. **keylime_agent** サービスを有効にして起動します。

```
# systemctl enable --now keylime_agent
```

8. オプション: Keylime テナントが設定されているシステムから、エージェントが正しく設定されており、registrar に接続できることを確認します。

```
# keylime_tenant -c regstatus --uuid <agent_uuid>
Reading configuration from ['/etc/keylime/logging.conf']
...
==\n-----END CERTIFICATE-----\n", "ip": "127.0.0.1", "port": 9002, "regcount": 1,
"operational_state": "Registered"]}]}
```

- **<agent_uuid>** は、エージェントの UUID に置き換えます。
registrar と agent が正しく設定されている場合、出力にはエージェントの IP アドレスとポートが表示され、その後に **"operational_state": "Registered"** が表示されます。

9. **/etc/ima/ima-policy** ファイルに次の内容を入力して、新しい IMA ポリシーを作成します。

```
# PROC_SUPER_MAGIC = 0x9fa0
dont_measure fsmagic=0x9fa0
# SYSFS_MAGIC = 0x62656572
```

```

dont_measure fsmagic=0x62656572
# DEBUGFS_MAGIC = 0x64626720
dont_measure fsmagic=0x64626720
# TMPFS_MAGIC = 0x01021994
dont_measure fsmagic=0x1021994
# RAMFS_MAGIC
dont_measure fsmagic=0x858458f6
# DEVPTS_SUPER_MAGIC=0x1cd1
dont_measure fsmagic=0x1cd1
# BINMTFS_MAGIC=0x42494e4d
dont_measure fsmagic=0x42494e4d
# SECURITYFS_MAGIC=0x73636673
dont_measure fsmagic=0x73636673
# SELINUX_MAGIC=0xf97cff8c
dont_measure fsmagic=0xf97cff8c
# SMACK_MAGIC=0x43415d53
dont_measure fsmagic=0x43415d53
# NSFS_MAGIC=0x6e736673
dont_measure fsmagic=0x6e736673
# EFIVARFS_MAGIC
dont_measure fsmagic=0xde5e81e4
# CGROUP_SUPER_MAGIC=0x27e0eb
dont_measure fsmagic=0x27e0eb
# CGROUP2_SUPER_MAGIC=0x63677270
dont_measure fsmagic=0x63677270
# OVERLAYFS_MAGIC
# when containers are used we almost always want to ignore them
dont_measure fsmagic=0x794c7630
# MEASUREMENTS
measure func=BPRM_CHECK
measure func=FILE_MMAP mask=MAY_EXEC
measure func=MODULE_CHECK uid=0

```

このポリシーは、実行されたアプリケーションのランタイム監視をターゲットとしています。このポリシーは状況に応じて調整できます。MAGIC 定数は、システム上の **stats(2)** man ページを参照してください。

- カーネルパラメーターを更新します。

```

# grubby --update-kernel DEFAULT --args 'ima_appraise=fix ima_canonical_fmt
ima_policy=tcb ima_template=ima-ng'

```

- システムを再起動して、新しい IMA ポリシーを適用します。

検証

- エージェントが実行されていることを確認します。

```

# systemctl status keylime_agent
● keylime_agent.service - The Keylime compute agent
   Loaded: loaded (/usr/lib/systemd/system/keylime_agent.service; enabled; preset:
disabled)
   Active: active (running) since ...

```

次のステップ

監視対象のすべてのシステムでエージェントを設定したら、Keylime をデプロイして、次の機能のいずれかまたは両方を実行できます。

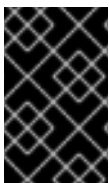
- [ランタイム監視用に Keylime をデプロイする](#)
- [ブート測定のアテストーション用に Keylime をデプロイする](#)

関連情報

- [Integrity Measurement Architecture \(IMA\) Wiki](#)

7.10. ランタイム監視用に KEYLIME を設定する

監視対象システムの状態が正しいことを確認するには、Keylime エージェントが監視対象システム上で実行されている必要があります。



重要

Keylime のランタイム監視は、Integrity Measurement Architecture (IMA) を使用して多数のファイルを測定するため、システムのパフォーマンスに重大な影響を与える可能性があります。

エージェントをプロビジョニングするときに、Keylime が監視対象システムに送信するファイルを定義することもできます。Keylime はエージェントに送信されたファイルを暗号化し、エージェントのシステムが TPM ポリシーと IMA 許可リストに準拠している場合にのみ復号化します。

Keylime の除外リストを設定することで、Keylime が特定のファイルまたは特定のディレクトリー内の変更を無視するようにできます。ファイルを除外しても、そのファイルは引き続き IMA によって測定されます。

RHEL 9.3 で提供される Keylime バージョン 7.3.0 以降、許可リストと除外リストは Keylime ランタイムポリシーに統合されます。

前提条件

- Keylime コンポーネントが設定されているシステムへのネットワークアクセスがある。

verifier

詳細は、[「パッケージから Keylime verifier をデプロイする」](#) を参照してください。

registrar

詳細は、[「パッケージから Keylime registrar をデプロイする」](#) を参照してください。

テナント

詳細は、[「パッケージから Keylime テナントをデプロイする」](#) を参照してください。

エージェント

詳細は、[「パッケージから Keylime エージェントをデプロイする」](#) を参照してください。

手順

1. Keylime エージェントが設定され実行されている監視対象システムに、**keylime-policy** ツールを含む **python3-keylime** パッケージをインストールします。

```
# dnf -y install python3-keylime
```

2. エージェントシステムの現在の状態からランタイムポリシーを作成します。

```
# keylime-policy create runtime --ima-measurement --rootfs '/' --ramdisk-dir '/boot' --output
<policy.json>
```

このコマンドの詳細は次のとおりです。

- **<policy.json>** はランタイムポリシーのファイル名に置き換えます。
- 次のディレクトリーは測定から自動的に除外されます。
 - **/sys**
 - **/run**
 - **/proc**
 - **/lost+found**
 - **/dev**
 - **/media**
 - **/snap**
 - **/mnt**
 - **/var**
 - **/tmp**
- 必要に応じて、**--excludelist <excludelist.txt>** オプションを追加して、追加の特定のパスを測定から除外することもできます。除外リストでは、1行に1つの Python 正規表現を使用できます。特殊文字の完全なリストは、[docs.python.org の Regular expression operations](https://docs.python.org/3/library/re.html) を参照してください。

3. 生成されたランタイムポリシーを、**keylime_tenant** ユーティリティーが設定されているシステムにコピーします。次に例を示します。

```
# scp <policy.json> root@<tenant.ip>:/root/<policy.json>
```

4. Keylime テナントが設定されているシステムで、**keylime_tenant** ユーティリティーを使用してエージェントをプロビジョニングします。

```
# keylime_tenant --command add --targethost <agent_ip> --uuid <agent_uuid> --runtime-policy
<policy.json> --cert default
```

- **<agent_ip>** は、エージェントの IP アドレスに置き換えます。
- **<agent_uuid>** は、エージェントの UUID に置き換えます。
- **<policy.json>** を Keylime ランタイムポリシーファイルへのパスに置き換えます。
- **--cert** オプションを使用すると、テナントは、指定されたディレクトリーまたはデフォルトの **/var/lib/keylime/ca/** ディレクトリーにある CA 証明書と鍵を使用して、エージェントの

証明書を作成し、署名します。ディレクトリーに CA 証明書と鍵が含まれていない場合、テナントは **/etc/keylime/ca.conf** ファイルの設定に従ってそれらを自動的に生成し、指定されたディレクトリーに保存します。その後、テナントはこれらの鍵と証明書をエージェントに送信します。

CA 証明書または署名エージェント証明書を生成するときに、CA 秘密鍵にアクセスするためのパスワードの入力を求められる (**Please enter the password to decrypt your keystore:**) 場合があります。



注記

Keylime はエージェントに送信されたファイルを暗号化し、エージェントのシステムが TPM ポリシーと IMA 許可リストに準拠している場合にのみ復号化します。デフォルトでは、Keylime は送信された **.zip** ファイルを展開します。

たとえば、次のコマンドを使用すると、**keylime_tenant** は UUID **d432fbb3-d2f1-4a97-9ef7-75bd81c00000** を持つ新しい Keylime エージェントを **127.0.0.1** にプロビジョニングし、**policy.json** というランタイムポリシーをロードします。また、デフォルトのディレクトリーに証明書を生成し、その証明書ファイルをエージェントに送信します。Keylime は、**/etc/keylime/verifier.conf** で設定された TPM ポリシーが満たされている場合にのみ、ファイルを復号化します。

```
# keylime_tenant --command add --targethost 127.0.0.1 --uuid d432fbb3-d2f1-4a97-9ef7-75bd81c00000 --runtime-policy policy.json --cert default
```



注記

keylime_tenant --command delete --uuid <agent_uuid> コマンドを使用して、Keylime によるノードの監視を停止できます。

keylime_tenant --command update コマンドを使用して、すでに登録されているエージェントの設定を変更できます。

検証

- オプション: 監視対象システムを再起動して、設定が永続的であることを確認します。
- エージェントのアテステーションが成功することを確認します。

```
# keylime_tenant --command cvstatus --uuid <agent.uuid>
...
{"<agent.uuid>": {"operational_state": "Get Quote"... "attestation_count": 5
...
```

<agent.uuid> をエージェントの UUID に置き換えます。

operational_state の値が **Get Quote** で、**attestation_count** が 0 以外の場合、このエージェントのアテステーションは成功しています。

Operational_state の値が **Invalid Quote** か **Failed** の場合、アテステーションは失敗し、次のようなコマンド出力が表示されます。

```
{<agent.uuid>: {"operational_state": "Invalid Quote", ... "ima.validation.ima-
ng.not_in_allowlist", "attestation_count": 5, "last_received_quote": 1684150329,
"last_successful_attestation": 1684150327}}
```

3. アテステーションが失敗した場合は、verifier ログで詳細を表示します。

```
# journalctl --unit keylime_verifier
keylime.tpm - INFO - Checking IMA measurement list...
keylime.ima - WARNING - File not found in allowlist: /root/bad-script.sh
keylime.ima - ERROR - IMA ERRORS: template-hash 0 fnf 1 hash 0 good 781
keylime.cloudverifier - WARNING - agent D432FBB3-D2F1-4A97-9EF7-75BD81C00000
failed, stopping polling
```

関連情報

- IMA の詳細は、[カーネル整合性サブシステムによるセキュリティーの強化](#) を参照してください。

7.11. ブート測定のアテステーション用に KEYLIME を設定する

Keylime をブート測定のアテステーション用に設定すると、Keylime は、測定対象システム上のブートプロセスが、定義した状態と一致しているかどうかを確認します。

前提条件

- Keylime コンポーネントが設定されているシステムへのネットワークアクセスがある。

verifier

詳細は、[「パッケージから Keylime verifier をデプロイする」](#) を参照してください。

registrar

詳細は、[「パッケージから Keylime registrar をデプロイする」](#) を参照してください。

テナント

詳細は、[「パッケージから Keylime テナントをデプロイする」](#) を参照してください。

エージェント

詳細は、[「パッケージから Keylime エージェントをデプロイする」](#) を参照してください。

- Unified Extensible Firmware Interface (UEFI) がエージェントシステムで有効になっている。

手順

1. Keylime エージェントが設定され実行されている監視対象システムに、**keylime-policy** ツールを含む **python3-keylime** パッケージをインストールします。

```
# dnf -y install python3-keylime
```

2. 監視対象システムで、**keylime-policy** ツールを使用して、システムの現在の状態を測定したブートログからポリシーを生成します。

```
# keylime-policy create measured-boot --eventlog-file
/sys/kernel/security/tpm0/binary_bios_measurements --output
<./measured_boot_reference_state.json>
```

- `<./measured_boot_reference_state.json>` は、**keylime-policy** によって生成されるポリシーが保存されるパスに置き換えます。
- UEFI システムでセキュアブートが有効になっていない場合は、**--without-secureboot** 引数を渡します。



重要

keylime-policy によって生成されるポリシーは、システムの現在の状態に基づいており、非常に厳格です。カーネルの更新やシステムの更新を含め、システムに変更を加えると、ブートプロセスが変更され、システムはアテステーションに失敗します。

3. 生成されたポリシーを、**keylime_tenant** ユーティリティーが設定されているシステムにコピーします。次に例を示します。

```
# scp root@<agent_ip>:./measured_boot_reference_state.json
./measured_boot_reference_state.json
```

4. Keylime テナントが設定されているシステムで、**keylime_tenant** ユーティリティーを使用してエージェントをプロビジョニングします。

```
# keylime_tenant --command add --targethost <agent_ip> --uuid <agent_uuid> --
mb_refstate <./measured_boot_reference_state.json> --cert default
```

- **<agent_ip>** は、エージェントの IP アドレスに置き換えます。
- **<agent_uuid>** は、エージェントの UUID に置き換えます。
- **<./measured_boot_reference_state.json>** を、ブート測定ポリシーへのパスに置き換えます。

ランタイム監視と一緒にブート測定を設定する場合は、**keylime_tenant --command add** コマンドを入力するときに両方のユースケースのオプションをすべて指定します。



注記

keylime_tenant --command delete --targethost <agent_ip> --uuid <agent_uuid> コマンドを使用して、Keylime によるノードの監視を停止できます。

keylime_tenant --command update コマンドを使用して、すでに登録されているエージェントの設定を変更できます。

検証

1. 監視対象システムを再起動し、エージェントのアテステーションが成功することを確認します。

```
# keylime_tenant --command cvstatus --uuid <agent_uuid>
...
{"<agent.uuid>": {"operational_state": "Get Quote"... "attestation_count": 5
...

```


<agent_uuid> は、エージェントの UUID に置き換えます。

operational_state の値が **Get Quote** で、**attestation_count** が 0 以外の場合、このエージェントのアテステーションは成功しています。

Operational_state の値が **Invalid Quote** か **Failed** の場合、アテステーションは失敗し、次のようなコマンド出力が表示されます。

```
{"<agent.uuid>": {"operational_state": "Invalid Quote", ... "ima.validation.ima-
ng.not_in_allowlist", "attestation_count": 5, "last_received_quote": 1684150329,
"last_successful_attestation": 1684150327}}
```

2. アテステーションが失敗した場合は、verifier ログで詳細を表示します。

```
# journalctl -u keylime_verifier
{"d432fbb3-d2f1-4a97-9ef7-75bd81c00000": {"operational_state": "Tenant Quote Failed", ...
"last_event_id": "measured_boot.invalid_pcr_0", "attestation_count": 0,
"last_received_quote": 1684487093, "last_successful_attestation": 0}}
```

7.12. KEYLIME の環境変数

podman run コマンドで **-e** オプションを使用してコンテナを起動するときなどに、Keylime の環境変数を設定すると、設定ファイルの値をオーバーライドできます。

環境変数の構文は次のとおりです。

```
KEYLIME_<SECTION>_<ENVIRONMENT_VARIABLE>=<value>
```

ここでは、以下のようになります。

- **<SECTION>** は Keylime 設定ファイルのセクションです。
- **<ENVIRONMENT_VARIABLE>** は環境変数です。
- **<value>** は環境変数に設定する値です。

たとえば、**-e KEYLIME_VERIFIER_MAX_RETRIES=6** は、**[verifier]** セクションの **max_retries** 設定オプションを **6** に設定します。

verifier の設定

表7.1[verifier] セクション

設定オプション	環境変数	デフォルト値
auto_migrate_db	KEYLIME_VERIFIER_AUTO_MIGRATE_DB	True
client_cert	KEYLIME_VERIFIER_CLIENT_CERT	default

設定オプション	環境変数	デフォルト値
client_key_password	KEYLIME_VERIFIER_CLIENT_KEY_PASSWORD	
client_key	KEYLIME_VERIFIER_CLIENT_KEY	default
database_pool_sz_ovfl	KEYLIME_VERIFIER_DATABASE_POOL_SZ_OVFL	5,10
database_url	KEYLIME_VERIFIER_DATABASE_URL	sqlite
durable_attestation_import	KEYLIME_VERIFIER_DURABLE_ATTESTATION_IMPORT	
enable_agent_mtls	KEYLIME_VERIFIER_ENABLE_AGENT_MTLS	True
exponential_backoff	KEYLIME_VERIFIER_EXPONENTIAL_BACKOFF	True
ignore_tomtom_errors	KEYLIME_VERIFIER_IGNORE_TOMTOM_ERRORS	False
ip	KEYLIME_VERIFIER_IP	127.0.0.1
max_retries	KEYLIME_VERIFIER_MAX_RETRIES	5
max_upload_size	KEYLIME_VERIFIER_MAX_UPLOAD_SIZE	104857600
measured_boot_evaluate	KEYLIME_VERIFIER_MEASURED_BOOT_EVALUATE	once
measured_boot_imports	KEYLIME_VERIFIER_MEASURED_BOOT_IMPORTS	[]
measured_boot_policy_name	KEYLIME_VERIFIER_MEASURED_BOOT_POLICY_NAME	accept-all
num_workers	KEYLIME_VERIFIER_NUM_WORKERS	0

設定オプション	環境変数	デフォルト値
persistent_store_encoding	KEYLIME_VERIFIER_PERSISTENT_STORE_ENCODING	
persistent_store_format	KEYLIME_VERIFIER_PERSISTENT_STORE_FORMAT	json
persistent_store_url	KEYLIME_VERIFIER_PERSISTENT_STORE_URL	
port	KEYLIME_VERIFIER_PORT	8881
quote_interval	KEYLIME_VERIFIER_QUOTE_INTERVAL	2
registrar_ip	KEYLIME_VERIFIER_REGISTRAR_IP	127.0.0.1
registrar_port	KEYLIME_VERIFIER_REGISTRAR_PORT	8891
request_timeout	KEYLIME_VERIFIER_REQUEST_TIMEOUT	60.0
require_allow_list_signatures	KEYLIME_VERIFIER_REQUIRE_ALLOW_LIST_SIGNATURES	True
retry_interval	KEYLIME_VERIFIER_RETRY_INTERVAL	2
server_cert	KEYLIME_VERIFIER_SERVER_CERT	default
server_key_password	KEYLIME_VERIFIER_SERVER_KEY_PASSWORD	default
server_key	KEYLIME_VERIFIER_SERVER_KEY	default
severity_labels	KEYLIME_VERIFIER_SEVERITY_LABELS	["info", "notice", "warning", "error", "critical", "alert", "emergency"]

設定オプション	環境変数	デフォルト値
severity_policy	KEYLIME_VERIFIER_SEVERITY_POLICY	[{"event_id": ".*", "severity_label": "emergency"}]
signed_attributes	KEYLIME_VERIFIER_SIGNED_ATTRIBUTES	
time_stamp_authority_certs_path	KEYLIME_VERIFIER_TIME_STAMP_AUTHORITY_CERTS_PATH	
time_stamp_authority_url	KEYLIME_VERIFIER_TIME_STAMP_AUTHORITY_URL	
tls_dir	KEYLIME_VERIFIER_TLS_DIR	generate
transparency_log_sign_algo	KEYLIME_VERIFIER_TRANSPARENCY_LOG_SIGN_ALGO	sha256
transparency_log_url	KEYLIME_VERIFIER_TRANSPARENCY_LOG_URL	
trusted_client_ca	KEYLIME_VERIFIER_TRUSTED_CLIENT_CA	default
trusted_server_ca	KEYLIME_VERIFIER_TRUSTED_SERVER_CA	default
uuid	KEYLIME_VERIFIER_UUID	default
version	KEYLIME_VERIFIER_VERSION	2.0

表7.2 [revocations] セクション

設定オプション	環境変数	デフォルト値
enabled_revocation_notifications	KEYLIME_VERIFIER_REVOCATIONS_ENABLED_REVOCATION_NOTIFICATIONS	[agent]

webhook_url	KEYLIME_VERIFIER_REVOCATIONS_WEBHOOK_URL	
--------------------	---	--

registrar の設定

表7.3 [registrar] セクション

設定オプション	環境変数	デフォルト値
auto_migrate_db	KEYLIME_REGISTRAR_AUTO_MIGRATE_DB	True
database_pool_sz_ovfl	KEYLIME_REGISTRAR_DATABASE_POOL_SZ_OVFL	5,10
database_url	KEYLIME_REGISTRAR_DATABASE_URL	sqlite
durable_attestation_import	KEYLIME_REGISTRAR_DURABLE_ATTESTATION_IMPORT	
ip	KEYLIME_REGISTRAR_IP	127.0.0.1
persistent_store_encoding	KEYLIME_REGISTRAR_PERSISTENT_STORE_ENCODING	
persistent_store_format	KEYLIME_REGISTRAR_PERSISTENT_STORE_FORMAT	json
persistent_store_url	KEYLIME_REGISTRAR_PERSISTENT_STORE_URL	
port	KEYLIME_REGISTRAR_PORT	8890
prov_db_filename	KEYLIME_REGISTRAR_PROVIDER_DB_FILENAME	provider_reg_data.sqlite
server_cert	KEYLIME_REGISTRAR_SERVER_CERT	default
server_key_password	KEYLIME_REGISTRAR_SERVER_KEY_PASSWORD	default

server_key	KEYLIME_REGISTRAR_SERVER_KEY	default
signed_attributes	KEYLIME_REGISTRAR_SIGNED_ATTRIBUTES	ek_tpm,aik_tpm,ekcert
time_stamp_authority_certs_path	KEYLIME_REGISTRAR_TIME_STAMP_AUTHORITY_CERTS_PATH	
time_stamp_authority_url	KEYLIME_REGISTRAR_TIME_STAMP_AUTHORITY_URL	
tls_dir	KEYLIME_REGISTRAR_TLS_DIR	default
tls_port	KEYLIME_REGISTRAR_TLS_PORT	8891
transparency_log_sign_algo	KEYLIME_REGISTRAR_TRANSPARENCY_LOG_SIGN_ALGO	sha256
transparency_log_url	KEYLIME_REGISTRAR_TRANSPARENCY_LOG_URL	
trusted_client_ca	KEYLIME_REGISTRAR_TRUSTED_CLIENT_CA	default
version	KEYLIME_REGISTRAR_VERSION	2.0

テナント設定

表7.4 [tenant] セクション

設定オプション	環境変数	デフォルト値
accept_tpm_encryption_algs	KEYLIME_TENANT_ACCEPT_TPM_ENCRYPTION_ALGS	ecc, rsa
accept_tpm_hash_algs	KEYLIME_TENANT_ACCEPT_TPM_HASH_ALGS	sha512, sha384, sha256
accept_tpm_signing_algs	KEYLIME_TENANT_ACCEPT_TPM_SIGNING_ALGS	ecschnorr, rsassa

client_cert	KEYLIME_TENANT_CLIENT_CERT	default
client_key_password	KEYLIME_TENANT_CLIENT_KEY_PASSWORD	
client_key	KEYLIME_TENANT_CLIENT_KEY	default
ek_check_script	KEYLIME_TENANT_EK_CHECK_SCRIPT	
enable_agent_mtls	KEYLIME_TENANT_ENABLE_AGENT_MTLS	True
exponential_backoff	KEYLIME_TENANT_EXPONENTIAL_BACKOFF	True
max_payload_size	KEYLIME_TENANT_MAX_PAYLOAD_SIZE	1048576
max_retries	KEYLIME_TENANT_MAX_RETRIES	5
mb_refstate	KEYLIME_TENANT_MB_REFSTATE	
registrar_ip	KEYLIME_TENANT_REGISTRAR_IP	127.0.0.1
registrar_port	KEYLIME_TENANT_REGISTRAR_PORT	8891
request_timeout	KEYLIME_TENANT_REQUEST_TIMEOUT	60
require_ek_cert	KEYLIME_TENANT_REQUIRE_EK_CERT	True
retry_interval	KEYLIME_TENANT_RETRY_INTERVAL	2
tls_dir	KEYLIME_TENANT_TLS_DIR	default
tpm_cert_store	KEYLIME_TENANT_TPM_CERT_STORE	/var/lib/keylime/tpm_cert_store

trusted_server_ca	KEYLIME_TENANT_TRUSTED_SERVER_CA	default
verifier_ip	KEYLIME_TENANT_VERIFIER_IP	127.0.0.1
verifier_port	KEYLIME_TENANT_VERIFIER_PORT	8881
version	KEYLIME_TENANT_VERSION	2.0

CA 設定

表7.5 [ca] セクション

設定オプション	環境変数	デフォルト値
cert_bits	KEYLIME_CA_CERT_BITS	2048
cert_ca_lifetime	KEYLIME_CA_CERT_CA_LIFETIME	3650
cert_ca_name	KEYLIME_CA_CERT_CA_NAME	Keylime Certificate Authority
cert_country	KEYLIME_CA_CERT_COUNTRY	US
cert_crl_dist	KEYLIME_CA_CERT_CRL_DIST	http://localhost:38080/crl
cert_lifetime	KEYLIME_CA_CERT_LIFETIME	365
cert_locality	KEYLIME_CA_CERT_LOCALITY	Lexington
cert_org_unit	KEYLIME_CA_CERT_ORG_UNIT	53
cert_organization	KEYLIME_CA_CERT_ORGANIZATION	MITLL
cert_state	KEYLIME_CA_CERT_STATE	MA
password	KEYLIME_CA_PASSWORD	default

version	KEYLIME_CA_VERSION	2.0
---------	--------------------	-----

エージェント設定

表7.6 [agent] セクション

設定オプション	環境変数	デフォルト値
contact_ip	KEYLIME_AGENT_CONTACT_IP	127.0.0.1
contact_port	KEYLIME_AGENT_CONTACT_PORT	9002
dec_payload_file	KEYLIME_AGENT_DEC_PAYLOAD_FILE	decrypted_payload
ek_handle	KEYLIME_AGENT_EK_HANDLE	generate
enable_agent_mtls	KEYLIME_AGENT_ENABLE_AGENT_MTLS	true
enable_insecure_payload	KEYLIME_AGENT_ENABLE_INSECURE_PAYLOAD	false
enable_revocation_notifications	KEYLIME_AGENT_ENABLE_REVOCATION_NOTIFICATIONS	true
enc_keyname	KEYLIME_AGENT_ENC_KEYNAME	derived_tci_key
exponential_backoff	KEYLIME_AGENT_EXPONENTIAL_BACKOFF	true
extract_payload_zip	KEYLIME_AGENT_EXTRACT_PAYLOAD_ZIP	true
ip	KEYLIME_AGENT_IP	127.0.0.1
max_retries	KEYLIME_AGENT_MAX_RETRIES	4
measure_payload_pcr	KEYLIME_AGENT_MEASURE_PAYLOAD_PCR	-1

設定オプション	環境変数	デフォルト値
payload_script	KEYLIME_AGENT_PAYLOAD_SCRIPT	autorun.sh
port	KEYLIME_AGENT_PORT	9002
registrar_ip	KEYLIME_AGENT_REGISTRAR_IP	127.0.0.1
registrar_port	KEYLIME_AGENT_REGISTRAR_PORT	8890
retry_interval	KEYLIME_AGENT_RETRY_INTERVAL	2
revocation_actions	KEYLIME_AGENT_REVOCATION_ACTIONS	[]
revocation_cert	KEYLIME_AGENT_REVOCATION_CERT	default
revocation_notification_ip	KEYLIME_AGENT_REVOCATION_NOTIFICATION_IP	127.0.0.1
revocation_notification_port	KEYLIME_AGENT_REVOCATION_NOTIFICATION_PORT	8992
run_as	KEYLIME_AGENT_RUN_AS	keylime:tss
secure_size	KEYLIME_AGENT_SECURE_SIZE	1m
server_cert	KEYLIME_AGENT_SERVER_CERT	default
server_key_password	KEYLIME_AGENT_SERVER_KEY_PASSWORD	
server_key	KEYLIME_AGENT_SERVER_KEY	default
tls_dir	KEYLIME_AGENT_TLS_DIR	default
tpm_encryption_alg	KEYLIME_AGENT_TPM_ENCRYPTION_ALG	rsa

設定オプション	環境変数	デフォルト値
tpm_hash_alg	KEYLIME_AGENT_TPM_HASH_ALG	sha256
tpm_ownerpassword	KEYLIME_AGENT_TPM_OWNERPASSWORD	
tpm_signing_alg	KEYLIME_AGENT_TPM_SIGNING_ALG	rsassa
trusted_client_ca	KEYLIME_AGENT_TRUSTED_CLIENT_CA	default
uuid	KEYLIME_AGENT_UUID	d432fbb3-d2f1-4a97-9ef7-75bd81c00000
version	KEYLIME_AGENT_VERSION	2.0

ロギング設定

表7.7 [logging] セクション

設定オプション	環境変数	デフォルト値
version	KEYLIME_LOGGING_VERSION	2.0

表7.8 [loggers] セクション

設定オプション	環境変数	デフォルト値
keys	KEYLIME_LOGGING_LOGGERS_KEYS	root,keylime

表7.9 [handlers] セクション

設定オプション	環境変数	デフォルト値
keys	KEYLIME_LOGGING_HANDLERS_KEYS	consoleHandler

表7.10 [formatters] セクション

設定オプション	環境変数	デフォルト値
keys	KEYLIME_LOGGING_FORM ATTERS_KEYS	formatter

表7.11 [formatter_formatter] セクション

設定オプション	環境変数	デフォルト値
datefmt	KEYLIME_LOGGING_FORM ATTER_FORMATTER_DATE FMT	%Y-%m-%d %H:%M:%S
format	KEYLIME_LOGGING_FORM ATTER_FORMATTER_FORM AT	%(asctime)s.%(msecs)03d - %(name)s - %(levelname)s - %(message)s

表7.12 [logger_root] セクション

設定オプション	環境変数	デフォルト値
handlers	KEYLIME_LOGGING_LOGG ER_ROOT_HANDLERS	consoleHandler
level	KEYLIME_LOGGING_LOGG ER_ROOT_LEVEL	INFO

表7.13 [handler_consoleHandler] セクション

設定オプション	環境変数	デフォルト値
args	KEYLIME_LOGGING_HANDL ER_CONSOLEHANDLER_AR GS	(sys.stdout,)
class	KEYLIME_LOGGING_HANDL ER_CONSOLEHANDLER_CL ASS	StreamHandler
formatter	KEYLIME_LOGGING_HANDL ER_CONSOLEHANDLER_FO RMATTER	formatter
level	KEYLIME_LOGGING_HANDL ER_CONSOLEHANDLER_LE VEL	INFO

表7.14 [logger_keylime] セクション

設定オプション	環境変数	デフォルト値
handlers	KEYLIME_LOGGING_LOGGER_KEYLIME_HANDLERS	
level	KEYLIME_LOGGING_LOGGER_KEYLIME_LEVEL	INFO
qualname	KEYLIME_LOGGING_LOGGER_KEYLIME_QUALNAME	keylime

第8章 AIDE で整合性の確認

Advanced Intrusion Detection Environment (AIDE) は、システムにファイルのデータベースを作成し、そのデータベースを使用してファイルの整合性を確保し、システムの侵入を検出するユーティリティーです。

8.1. AIDE のインストール

AIDE を使用してファイル整合性チェックを開始するには、対応するパッケージをインストールし、AIDE データベースを初期化する必要があります。

前提条件

- **AppStream** リポジトリが有効になっている。

手順

1. **aide** パッケージをインストールします。

```
# dnf install aide
```

2. 初期データベースを生成します。

```
# aide --init
Start timestamp: 2024-07-08 10:39:23 -0400 (AIDE 0.16)
AIDE initialized database at /var/lib/aide/aide.db.new.gz

Number of entries: 55856

-----
The attributes of the (uncompressed) database(s):
-----

/var/lib/aide/aide.db.new.gz
...
SHA512 : mZaWoGzL2m6ZcyyZ/AXTlowliEXWSZqx
        IFYImY4f7id4u+Bq8WeuSE2jasZur/A4
        FPBFaBkoCFHdoE/FW/V94Q==
```

3. オプション: デフォルト設定では、**aide --init** コマンドは、**/etc/aide.conf** ファイルで定義するディレクトリーとファイルのセットのみを確認します。ディレクトリーまたはファイルを AIDE データベースに追加し、監視パラメーターを変更するには、**/etc/aide.conf** を変更します。
4. データベースの使用を開始するには、初期データベースのファイル名から末尾の **.new** を削除します。

```
# mv /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
```

5. オプション: AIDE データベースの場所を変更するには、**/etc/aide.conf** ファイルを編集し、**DBDIR** 値を変更します。追加のセキュリティーのデータベース、設定、**/usr/sbin/aide** バイナリーファイルを、読み取り専用メディアなどの安全な場所に保存します。

8.2. AIDE を使用した整合性チェックの実行

crond サービスを使用すると、AIDE で定期的なファイル整合性チェックをスケジュールできます。

前提条件

- AIDE が適切にインストールされ、データベースが初期化されている。[AIDE のインストール](#) を参照してください。

手順

1. 手動でチェックを開始するには、以下を行います。

```
# aide --check
Start timestamp: 2024-07-08 10:43:46 -0400 (AIDE 0.16)
AIDE found differences between database and filesystem!!

Summary:
Total number of entries: 55856
Added entries: 0
Removed entries: 0
Changed entries: 1

-----
Changed entries:
-----

f ... ..S : /root/.viminfo

-----
Detailed information about changes:
-----

File: /root/.viminfo
SELinux : system_u:object_r:admin_home_t:s | unconfined_u:object_r:admin_home
          0                               |_t:s0
...
```

2. 少なくとも、AIDE を毎週実行するようにシステムを設定します。最適な設定としては、AIDE を毎日実行します。たとえば、**cron** コマンドを使用して毎日午前 **04:05** に AIDE を実行するようにスケジュールするには、**/etc/crontab** ファイルに次の行を追加します。

```
05 4 * * * root /usr/sbin/aide --check
```

関連情報

- システム上の **cron(8)** man ページ

8.3. AIDE データベースの更新

パッケージの更新や設定ファイルの調整など、システムの変更が確認できたら、基本となる AIDE データベースも更新します。

前提条件

- AIDE が適切にインストールされ、データベースが初期化されている。[AIDE のインストール](#) を参照してください。

手順

1. 基本となる AIDE データベースを更新します。

```
# aide --update
```

aide --update コマンドは、**/var/lib/aide/aide.db.new.gz** データベースファイルを作成します。

2. 整合性チェックで更新したデータベースを使用するには、ファイル名から末尾の **.new** を削除します。

8.4. ファイル整合性ツール:AIDE および IMA

Red Hat Enterprise Linux は、システム上のファイルとディレクトリーの整合性をチェックおよび保持するためのさまざまなツールを提供します。次の表は、シナリオに適したツールを決定するのに役立ちます。

表8.1 AIDE と IMA の比較

比較項目	Advanced Intrusion Detection Environment (AIDE)	Integrity Measurement Architecture (IMA)
確認対象	AIDE は、システム上のファイルとディレクトリーのデータベースを作成するユーティリティです。このデータベースは、ファイルの整合性をチェックし、侵入を検出するのに役立ちます。	IMA は、以前に保存された拡張属性と比較してファイル測定値 (ハッシュ値) をチェックすることにより、ファイルが変更されているかどうかを検出します。
確認方法	AIDE はルールを使用して、ファイルとディレクトリーの整合性状態を比較します。	IMA は、ファイルハッシュ値を使用して侵入を検出します。
理由	検出 - AIDE は、ルールを検証することにより、ファイルが変更されているかどうかを検出します。	検出と防止 - IMA は、ファイルの拡張属性を置き換えることにより、攻撃を検出および防止します。
使用方法	AIDE は、ファイルまたはディレクトリーが変更されたときに脅威を検出します。	誰かがファイル全体の変更を試みた時に、IMA は脅威を検出します。
範囲	AIDE は、ローカルシステム上のファイルとディレクトリーの整合性をチェックします。	IMA は、ローカルシステムとリモートシステムのセキュリティを確保します。

8.5. AIDE RHEL システムロールを使用したファイル整合性チェックの設定

aide RHEL システムロールを使用すると、複数のシステムにわたりに貫して Advanced Intrusion Detection Environment (AIDE) を設定できます。このロールは、すべての管理対象ノードに **aide** パッケージを自動的にインストールし、設定に応じて次のアクションを実行できます。

- AIDE データベースを初期化し、コントロールノードに保存する
- 管理対象ノードで AIDE 整合性チェックを実行する
- AIDE データベースを更新し、コントロールノードに保存する

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: Configure system integrity
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure file integrity checks with AIDE
      ansible.builtin.include_role:
        name: rhel-system-roles.aide.aide
      vars:
        aide_db_fetch_dir: files
        aide_init: true
        aide_check: false
        aide_update: false
        aide_cron_check: true
        aide_cron_interval: 0 12 * * *
```

サンプル Playbook で指定されている設定は次のとおりです。

aide_db_fetch_dir: files

リモートノードから取得した AIDE データベースを保存するための Ansible コントロールノード (ACN) 上のディレクトリを指定します。デフォルトの **files** 値では、Playbook と同じディレクトリにデータベースが保存されます。データベースファイルを別の場所に保存するには、別のパスを指定します。

aide_check: false

リモートノードで整合性チェックを実行します。

aide_update: false

AIDE データベースを更新し、コントロールノードに保存します。

aide_cron_check: true

管理対象ノードで AIDE 整合性チェックをアクティブ化する定期的な **cron** ジョブを設定します。

aide_cron_interval: 0 12 * * *

cron ジョブの間隔を **<minute> <hour> <day_of_month> <month> <day of week>** という形式で設定します。値を **0 12 * * *** にすると、毎日正午にジョブを実行するように設定されます。

Playbook で使用されるすべての変数の詳細は、コントロールノードの **/usr/share/ansible/roles/rhel-system-roles.aide/README.md** ファイルを参照してください。

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

8.6. 関連情報

- システム上の **aide(1)** man ページ
- [Kernel integrity subsystem](#)

第9章 LUKS を使用したブロックデバイスの暗号化

ディスク暗号化を使用すると、ブロックデバイス上のデータを暗号化して保護できます。デバイスの復号化されたコンテンツにアクセスするには、認証としてパスフレーズまたは鍵を入力します。これは、デバイスがシステムから物理的に取り外された場合でも、デバイスのコンテンツを保護するのに役立つため、モバイルコンピューターやリムーバブルメディアにとって重要です。LUKS 形式は、Red Hat Enterprise Linux におけるブロックデバイスの暗号化のデフォルト実装です。

9.1. LUKS ディスクの暗号化

Linux Unified Key Setup-on-disk-format (LUKS) は、暗号化されたデバイスの管理を簡素化するツールセットを提供します。LUKS を使用すると、ブロックデバイスを暗号化し、複数のユーザーキーでマスターキーを復号化できるようになります。パーティションの一括暗号化には、このマスターキーを使用します。

Red Hat Enterprise Linux は、LUKS を使用してブロックデバイスの暗号化を実行します。デフォルトではインストール時に、ブロックデバイスを暗号化するオプションが指定されていません。ディスクを暗号化するオプションを選択すると、コンピューターを起動するたびにパスフレーズの入力が必要です。このパスフレーズは、パーティションを復号化するバルク暗号鍵のロックを解除します。デフォルトのパーティションテーブルを変更する場合は、暗号化するパーティションを選択できます。この設定は、パーティションテーブル設定で行われます。

Ciphers

LUKS に使用されるデフォルトの暗号は **aes-xts-plain64** です。LUKS のデフォルトの鍵サイズは 512 ビットです。**Anaconda** XTS モードを使用した LUKS のデフォルトの鍵サイズは 512 ビットです。使用可能な暗号は次のとおりです。

- 高度暗号化標準 (Advanced Encryption Standard, AES)
- Twofish
- Serpent

LUKS によって実行される操作

- LUKS は、ブロックデバイス全体を暗号化するため、脱着可能なストレージメディアやラップトップのディスクドライブといった、モバイルデバイスのコンテンツを保護するのに適しています。
- 暗号化されたブロックデバイスの基本的な内容は任意であり、スワップデバイスの暗号化に役立ちます。また、とりわけデータストレージ用にフォーマットしたブロックデバイスを使用する特定のデータベースに関しても有用です。
- LUKS は、既存のデバイスマッパーのカーネルサブシステムを使用します。
- LUKS はパスフレーズのセキュリティを強化し、辞書攻撃から保護します。
- LUKS デバイスには複数のキースロットが含まれているため、バックアップキーやパスフレーズを追加できます。



重要

LUKS は次のシナリオには推奨されません。

- LUKS などのディスク暗号化ソリューションは、システムの停止時にしかデータを保護しません。システムの電源がオンになり、LUKS がディスクを復号化すると、そのディスクのファイルは、そのファイルにアクセスできるすべてのユーザーが使用できます。
- 同じデバイスに対する個別のアクセスキーを複数のユーザーが持つ必要があるシナリオ。LUKS1 形式はキースロットを 8 個提供し、LUKS2 形式はキースロットを最大 32 個提供します。
- ファイルレベルの暗号化を必要とするアプリケーション。

関連情報

- [LUKS プロジェクトのホームページ](#)
- [LUKS オンディスクフォーマットの仕様](#)
- [FIPS 197: Advanced Encryption Standard \(AES\)](#)

9.2. RHEL の LUKS バージョン

Red Hat Enterprise Linux では、LUKS 暗号化のデフォルト形式は LUKS2 です。古い LUKS1 形式は引き続き完全にサポートされており、以前の Red Hat Enterprise Linux リリースと互換性のある形式で提供されます。LUKS2 再暗号化は、LUKS1 再暗号化と比較して、より堅牢で安全に使用できる形式と考えられています。

LUKS2 形式を使用すると、バイナリー構造を変更することなく、さまざまな部分を後に更新できます。LUKS2 は、内部的にメタデータに JSON テキスト形式を使用し、メタデータの冗長性を提供し、メタデータの破損を検出し、メタデータのコピーから自動的に修復します。



重要

LUKS1 のみをサポートするシステムでは LUKS2 を使用しないでください。

Red Hat Enterprise Linux 9.2 以降では、両方の LUKS バージョンで **cryptsetup reencrypt** コマンドを使用してディスクを暗号化できます。

オンラインの再暗号化

LUKS2 形式は、デバイスが使用中の間に、暗号化したデバイスの再暗号化に対応します。たとえば、以下のタスクを実行するにあたり、デバイスでファイルシステムをアンマウントする必要はありません。

- ボリュームキーの変更
- 暗号化アルゴリズムの変更
暗号化されていないデバイスを暗号化する場合は、ファイルシステムのマウントを解除する必要があります。暗号化の短い初期化後にファイルシステムを再マウントできます。

LUKS1 形式は、オンライン再暗号化に対応していません。

変換

特定の状況では、LUKS1 を LUKS2 に変換できます。具体的には、以下のシナリオでは変換ができません。

- LUKS1 デバイスが、Policy-Based Decryption (PBD) Clevis ソリューションにより使用されているとマークされている。**cryptsetup** ツールは、**luksmeta** メタデータが検出されると、そのデバイスを変換することを拒否します。
- デバイスがアクティブになっている。デバイスが非アクティブ状態でなければ、変換することはできません。

9.3. LUKS2 再暗号化中のデータ保護のオプション

LUKS2 では、再暗号化プロセスで、パフォーマンスやデータ保護の優先度を設定する複数のオプションを選択できます。**resilience** オプションには次のモードが用意されています。**cryptsetup reencrypt --resilience resilience-mode /dev/<device_ID>** コマンドを使用すると、これらのモードのいずれかを選択できます。**<device_ID>** は、デバイスの ID に置き換えてください。

checksum

デフォルトのモード。データ保護とパフォーマンスのバランスを取ります。

このモードでは、再暗号化領域内のセクターのチェックサムが個別に保存されます。チェックサムは、LUKS2 によって再暗号化されたセクターについて、復旧プロセスで検出できます。このモードでは、ブロックデバイスセクターの書き込みがアトミックである必要があります。

journal

最も安全なモードですが、最も遅いモードでもあります。このモードでは、再暗号化領域をバイナリ領域にジャーナル化するため、LUKS2 はデータを 2 回書き込みます。

none

none モードではパフォーマンスが優先され、データ保護は提供されません。**SIGTERM** シグナルやユーザーによる **Ctrl+C** キーの押下など、安全なプロセス終了からのみデータを保護します。予期しないシステム障害やアプリケーション障害が発生すると、データが破損する可能性があります。

LUKS2 の再暗号化プロセスが強制的に突然終了した場合、LUKS2 は以下のいずれかの方法で復旧を実行できます。

自動

次のいずれかのアクションを実行すると、次回の LUKS2 デバイスを開くアクション中に自動復旧アクションがトリガーされます。

- **cryptsetup open** コマンドを実行する。
- **systemd-cryptsetup** コマンドを使用してデバイスを接続する。

手動

LUKS2 デバイスで **cryptsetup repair /dev/<device_ID>** コマンドを使用します。

関連情報

- システム上の **cryptsetup-reencrypt(8)** および **cryptsetup-repair(8)** man ページ

9.4. LUKS2 を使用したブロックデバイスの既存データの暗号化

LUKS2 形式を使用して、まだ暗号化されていないデバイスの既存のデータを暗号化できます。新しい LUKS ヘッダーは、デバイスのヘッドに保存されます。

前提条件

- ブロックデバイスにファイルシステムがある。
- データのバックアップを作成している。



警告

ハードウェア、カーネル、または人的ミスにより、暗号化プロセス時にデータが失われる場合があります。データの暗号化を開始する前に、信頼性の高いバックアップを作成してください。

手順

1. 暗号化するデバイスにあるファイルシステムのマウントをすべて解除します。次に例を示します。

```
# umount /dev/mapper/vg00-lv00
```

2. LUKS ヘッダーを保存するための空き容量を確認します。シナリオに合わせて、次のいずれかのオプションを使用します。

- 論理ボリュームを暗号化する場合は、以下のように、ファイルシステムのサイズを変更せずに、論理ボリュームを拡張できます。以下に例を示します。

```
# lvextend -L+32M /dev/mapper/vg00-lv00
```

- **parted** などのパーティション管理ツールを使用してパーティションを拡張します。
- このデバイスのファイルシステムを縮小します。ext2、ext3、または ext4 のファイルシステムには **resize2fs** ユーティリティーを使用できます。XFS ファイルシステムは縮小できないことに注意してください。

3. 暗号化を初期化します。

```
# cryptsetup reencrypt --encrypt --init-only --reduce-device-size 32M /dev/mapper/vg00-lv00  
lv00_encrypted
```

```
/dev/mapper/lv00_encrypted is now active and ready for online encryption.
```

4. デバイスをマウントします。

```
# mount /dev/mapper/lv00_encrypted /mnt/lv00_encrypted
```

5. 永続的なマッピングのエントリーを **/etc/crypttab** ファイルに追加します。

- a. **luksUUID** を見つけます。

```
# cryptsetup luksUUID /dev/mapper/vg00-lv00
```

```
a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325
```

- b. 任意のテキストエディターで **/etc/crypttab** を開き、このファイルにデバイスを追加します。

```
$ vi /etc/crypttab
```

```
lv00_encrypted UUID=a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325 none
```

a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325 は、デバイスの **luksUUID** に置き換えます。

- c. **dracut** で **initramfs** を更新します。

```
$ dracut -f --regenerate-all
```

6. **/etc/fstab** ファイルに永続的なマウントのエントリーを追加します。

- a. アクティブな LUKS ブロックデバイスのファイルシステムの UUID を見つけます。

```
$ blkid -p /dev/mapper/lv00_encrypted
```

```
/dev/mapper/lv00-encrypted: UUID="37bc2492-d8fa-4969-9d9b-bb64d3685aa9"  
BLOCK_SIZE="4096" TYPE="xfs" USAGE="filesystem"
```

- b. 任意のテキストエディターで **/etc/fstab** を開き、このファイルにデバイスを追加します。次に例を示します。

```
$ vi /etc/fstab
```

```
UUID=37bc2492-d8fa-4969-9d9b-bb64d3685aa9 /home auto rw,user,auto 0
```

37bc2492-d8fa-4969-9d9b-bb64d3685aa9 は、ファイルシステムの UUID に置き換えます。

7. オンライン暗号化を再開します。

```
# cryptsetup reencrypt --resume-only /dev/mapper/vg00-lv00
```

```
Enter passphrase for /dev/mapper/vg00-lv00:
```

```
Auto-detected active dm device 'lv00_encrypted' for data device /dev/mapper/vg00-lv00.
```

```
Finished, time 00:31.130, 10272 MiB written, speed 330.0 MiB/s
```

検証

1. 既存のデータが暗号化されているかどうかを確認します。

```
# cryptsetup luksDump /dev/mapper/vg00-lv00
```

```
LUKS header information
```

```
Version: 2
```

```
Epoch: 4
```

```

Metadata area: 16384 [bytes]
Keyslots area: 16744448 [bytes]
UUID: a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325
Label: (no label)
Subsystem: (no subsystem)
Flags: (no flags)

Data segments:
  0: crypt
  offset: 33554432 [bytes]
  length: (whole device)
  cipher: aes-xts-plain64
  [...]

```

2. 暗号化された空のブロックデバイスのステータスを表示します。

```

# cryptsetup status lv00_encrypted

/dev/mapper/lv00_encrypted is active and is in use.
type:    LUKS2
cipher:  aes-xts-plain64
keysize: 512 bits
key location: keyring
device:  /dev/mapper/vg00-lv00

```

関連情報

- システム上の **cryptsetup(8)**、**cryptsetup-reencrypt(8)**、**lvextend(8)**、**resize2fs(8)**、および **parted(8)** [man ページ](#)

9.5. 独立したヘッダーがある LUKS2 を使用してブロックデバイスの既存データの暗号化

LUKS ヘッダーを保存するための空き領域を作成せずに、ブロックデバイスの既存のデータを暗号化できます。ヘッダーは、追加のセキュリティー層としても使用できる、独立した場所に保存されます。この手順では、LUKS2 暗号化形式を使用します。

前提条件

- ブロックデバイスにファイルシステムがある。
- データがバックアップ済みである。



警告

ハードウェア、カーネル、または人的ミスにより、暗号化プロセス時にデータが失われる場合があります。データの暗号化を開始する前に、信頼性の高いバックアップを作成してください。

手順

1. 以下のように、そのデバイスのファイルシステムをすべてアンマウントします。

```
# umount /dev/<nvme0n1p1>
```

<nvme0n1p1> は、アンマウントするパーティションに対応するデバイス識別子に置き換えます。

2. 暗号化を初期化します。

```
# cryptsetup reencrypt --encrypt --init-only --header </home/header> /dev/<nvme0n1p1>
<nvme_encrypted>
```

WARNING!

=====

Header file does not exist, do you want to create it?

Are you sure? (Type 'yes' in capital letters): YES

Enter passphrase for </home/header>:

Verify passphrase:

/dev/mapper/<nvme_encrypted> is now active and ready for online encryption.

以下のように置き換えます。

- **</home/header>** には、独立した LUKS ヘッダーを含むファイルへのパスを指定します。後で暗号化したデバイスのロックを解除するために、独立した LUKS ヘッダーにアクセスする必要があります。
- **<nvme_encrypted>** は、暗号化後に作成されるデバイスマッパーの名前に置き換えます。

3. デバイスをマウントします。

```
# mount /dev/mapper/<nvme_encrypted> /mnt/<nvme_encrypted>
```

4. 永続的なマッピングのエントリーを **/etc/crypttab** ファイルに追加します。

```
# <nvme_encrypted> /dev/disk/by-id/<nvme-partition-id> none header=</home/header>
```

<nvme-partition-id> は、NVMe パーティションの識別子に置き換えます。

5. **dracut** を使用して **initramfs** を再生成します。

```
# dracut -f --regenerate-all -v
```

6. **/etc/fstab** ファイルに永続的なマウントのエントリーを追加します。

- a. アクティブな LUKS ブロックデバイスのファイルシステムの UUID を見つけます。

```
$ blkid -p /dev/mapper/<nvme_encrypted>
```

```
/dev/mapper/<nvme_encrypted>: UUID="37bc2492-d8fa-4969-9d9b-bb64d3685aa9"
BLOCK_SIZE="4096" TYPE="xfs" USAGE="filesystem"
```

- b. テキストエディターで **/etc/fstab** を開き、このファイルにデバイスを追加します。次に例を示します。

```
UUID=<file_system_UUID> /home auto rw,user,auto 0
```

<file_system_UUID> は、直前の手順で見つかったファイルシステムの UUID に置き換えます。

7. オンライン暗号化を再開します。

```
# cryptsetup reencrypt --resume-only --header </home/header> /dev/<nvme0n1p1>
```

```
Enter passphrase for /dev/<nvme0n1p1>:
```

```
Auto-detected active dm device '<nvme_encrypted>' for data device /dev/<nvme0n1p1>.
```

```
Finished, time 00m51s, 10 GiB written, speed 198.2 MiB/s
```

検証

1. 独立したヘッダーがある LUKS2 を使用するブロックデバイスの既存のデータが暗号化されているかどうかを確認します。

```
# cryptsetup luksDump </home/header>
```

```
LUKS header information
```

```
Version:      2
```

```
Epoch:       88
```

```
Metadata area: 16384 [bytes]
```

```
Keyslots area: 16744448 [bytes]
```

```
UUID:        c4f5d274-f4c0-41e3-ac36-22a917ab0386
```

```
Label:        (no label)
```

```
Subsystem:    (no subsystem)
```

```
Flags:        (no flags)
```

```
Data segments:
```

```
0: crypt
```

```
offset: 0 [bytes]
```

```
length: (whole device)
```

```
cipher: aes-xts-plain64
```

```
sector: 512 [bytes]
```

```
[...]
```

2. 暗号化された空のブロックデバイスのステータスを表示します。

```
# cryptsetup status <nvme_encrypted>
```

```
/dev/mapper/<nvme_encrypted> is active and is in use.
```

```
type: LUKS2
```

```
cipher: aes-xts-plain64
```

```
keysize: 512 bits
```

```
key location: keyring
```

```
device: /dev/<nvme0n1p1>
```

関連情報

- システム上の **cryptsetup(8)** および **cryptsetup-reencrypt(8)** man ページ

9.6. LUKS2 を使用した空のブロックデバイスの暗号化

LUKS2 形式を使用して、空のブロックデバイスを暗号化して、暗号化ストレージとして使用できます。

前提条件

- 空のブロックデバイス。**lsblk** などのコマンドを使用して、そのデバイス上に実際のデータ（ファイルシステムなど）がないかどうかを確認できます。

手順

1. 暗号化した LUKS パーティションとしてパーティションを設定します。

```
# cryptsetup luksFormat /dev/nvme0n1p1

WARNING!
=====
This will overwrite data on /dev/nvme0n1p1 irrevocably.
Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/nvme0n1p1:
Verify passphrase:
```

2. 暗号化した LUKS パーティションを開きます。

```
# cryptsetup open /dev/nvme0n1p1 nvme0n1p1_encrypted

Enter passphrase for /dev/nvme0n1p1:
```

これにより、パーティションのロックが解除され、デバイスマッパーを使用してパーティションが新しいデバイスにマッピングされます。暗号化されたデータを上書きしないように、このコマンドは、デバイスが暗号化されたデバイスであり、**/dev/mapper/device_mapped_name** パスを使用して LUKS を通じてアドレス指定されることをカーネルに警告します。

3. 暗号化されたデータをパーティションに書き込むためのファイルシステムを作成します。このパーティションには、デバイスマップ名を介してアクセスする必要があります。

```
# mkfs -t ext4 /dev/mapper/nvme0n1p1_encrypted
```

4. デバイスをマウントします。

```
# mount /dev/mapper/nvme0n1p1_encrypted mount-point
```

検証

1. 空のブロックデバイスが暗号化されているかどうかを確認します。

```
# cryptsetup luksDump /dev/nvme0n1p1

LUKS header information
Version:      2
```

```

Epoch:      3
Metadata area: 16384 [bytes]
Keyslots area: 16744448 [bytes]
UUID:       34ce4870-ffdf-467c-9a9e-345a53ed8a25
Label:      (no label)
Subsystem:  (no subsystem)
Flags:      (no flags)

Data segments:
  0: crypt
  offset: 16777216 [bytes]
  length: (whole device)
  cipher: aes-xts-plain64
  sector: 512 [bytes]
  [...]

```

2. 暗号化された空のブロックデバイスのステータスを表示します。

```

# cryptsetup status nvme0n1p1_encrypted

/dev/mapper/nvme0n1p1_encrypted is active and is in use.
type: LUKS2
cipher: aes-xts-plain64
keysize: 512 bits
key location: keyring
device: /dev/nvme0n1p1
sector size: 512
offset: 32768 sectors
size: 20938752 sectors
mode: read/write

```

関連情報

- システム上の **cryptsetup(8)**、**cryptsetup-open(8)**、および **cryptsetup-luksFormat(8)** man ページ

9.7. WEB コンソールでの LUKS パスフレーズの設定

システムの既存の論理ボリュームに暗号化を追加する場合は、ボリュームをフォーマットすることでは実行できません。

前提条件

- RHEL 9 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- **cockpit-storaged** パッケージがシステムにインストールされている。
- 暗号化なしで、既存の論理ボリュームを利用できます。

手順

1. RHEL 9 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. パネルで、**Storage** をクリックします。
3. **Storage** テーブルで、暗号化するストレージデバイスのメニューボタン **⋮** をクリックし、**Format** をクリックします。
4. **Encryption field** で、暗号化仕様 **LUKS1** または **LUKS2** を選択します。
5. 新しいパスフレーズを設定し、確認します。
6. オプション: さらなる暗号化オプションを変更します。
7. フォーマット設定の最終処理
8. **Format** をクリックします。

9.8. WEB コンソールで LUKS パスフレーズの変更

Web コンソールで、暗号化されたディスクまたはパーティションで LUKS パスフレーズを変更します。

前提条件

- RHEL 9 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- **cockpit-storaged** 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. RHEL 9 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. パネルで、**Storage** をクリックします。
3. **Storage** テーブルで、暗号化されたデータを含むディスクを選択します。
4. ディスクページで、**Keys** セクションまでスクロールし、**編集ボタン** をクリックします。
5. **パスフレーズの変更** ダイアログウィンドウで、以下を行います。
 - a. 現在のパスフレーズを入力します。
 - b. 新しいパスフレーズを入力します。
 - c. 新しいパスフレーズを確認します。
6. **Save** をクリックします。

9.9. コマンドラインを使用した LUKS パスフレーズの変更

コマンドラインを使用して、暗号化されたディスクまたはパーティションの LUKS パスフレーズを変更します。**cryptsetup** ユーティリティを使用すると、さまざまな設定オプションと機能を使用して暗号化プロセスを制御し、既存の自動化ワークフローにプロセスを統合できます。

前提条件

- **root** 特権、または **sudo** を使用して管理コマンドを入力する権限がある。

手順

1. LUKS 暗号化デバイスの既存のパスフレーズを変更します。

```
# cryptsetup luksChangeKey /dev/<device_ID>
```

<device_ID> は、デバイス指定子 (例: **sda**) に置き換えます。

複数のキースロットが設定されている場合は、使用するスロットを指定できます。

```
# cryptsetup luksChangeKey /dev/<device_ID> --key-slot <slot_number>
```

<slot_number> は、変更するキースロットの番号に置き換えます。

2. 現在のパスフレーズと新しいパスフレーズを入力します。

```
Enter passphrase to be changed:
Enter new passphrase:
Verify passphrase:
```

3. 新しいパスフレーズを検証します。

```
# cryptsetup --verbose open --test-passphrase /dev/<device_ID>
```

検証

1. 新しいパスフレーズでデバイスのロックを解除できることを確認します。

```
Enter passphrase for /dev/<device_ID>:
Key slot <slot_number> unlocked.
Command successful.
```

9.10. STORAGE RHEL システムロールを使用して LUKS2 暗号化ボリュームを作成する

storage ロールを使用し、Ansible Playbook を実行して、LUKS で暗号化されたボリュームを作成および設定できます。

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. 機密性の高い変数を暗号化されたファイルに保存します。

- a. vault を作成します。

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

- b. **ansible-vault create** コマンドでエディターが開いたら、機密データを **<key>: <value>** 形式で入力します。

```
luks_password: <password>
```

- c. 変更を保存して、エディターを閉じます。Ansible は vault 内のデータを暗号化します。

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

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create and configure a volume encrypted with LUKS
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_volumes:
          - name: barefs
            type: disk
            disks:
              - sdb
            fs_type: xfs
            fs_label: <label>
            mount_point: /mnt/data
            encryption: true
            encryption_password: "{{ luks_password }}"
```

Playbook で使用されるすべての変数の詳細は、コントロールノードの **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** ファイルを参照してください。

3. Playbook の構文を検証します。

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

検証

1. LUKS 暗号化ボリュームの **luksUUID** 値を見つけます。

```
# ansible managed-node-01.example.com -m command -a 'cryptsetup luksUUID /dev/sdb'

4e4e7970-1822-470e-b55a-e91efe5d0f5c
```

2. ボリュームの暗号化ステータスを表示します。

```
# ansible managed-node-01.example.com -m command -a 'cryptsetup status luks-4e4e7970-1822-470e-b55a-e91efe5d0f5c'

/dev/mapper/luks-4e4e7970-1822-470e-b55a-e91efe5d0f5c is active and is in use.
type: LUKS2
cipher: aes-xts-plain64
keysize: 512 bits
key location: keyring
device: /dev/sdb
...
```

3. 作成された LUKS 暗号化ボリュームを確認します。

```
# ansible managed-node-01.example.com -m command -a 'cryptsetup luksDump /dev/sdb'

LUKS header information
Version:      2
Epoch:       3
Metadata area: 16384 [bytes]
Keyslots area: 16744448 [bytes]
UUID:         4e4e7970-1822-470e-b55a-e91efe5d0f5c
Label:        (no label)
Subsystem:    (no subsystem)
Flags:        (no flags)

Data segments:
0: crypt
  offset: 16777216 [bytes]
  length: (whole device)
  cipher: aes-xts-plain64
  sector: 512 [bytes]
...
```

関連情報

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

- [LUKS を使用したブロックデバイスの暗号化](#)
- [Ansible vault](#)

第10章 ポリシーベースの復号を使用した暗号化ボリュームの自動ロック解除の設定

ポリシーベースの復号 (PBD) は、物理マシンおよび仮想マシンにおいて、ハードドライブで暗号化した root ボリュームおよびセカンダリーボリュームのロックを解除できるようにする一連の技術です。PBD は、ユーザーパスワード、TPM (Trusted Platform Module) デバイス、システムに接続する PKCS #11 デバイス (たとえばスマートカード) などのさまざまなロックの解除方法、もしくは特殊なネットワークサーバーを使用します。

PBD を使用すると、ポリシーにさまざまなロックの解除方法を組み合わせて、さまざまな方法で同じボリュームのロックを解除できるようにすることができます。RHEL における PBD の現在の実装は、Clevis フレームワークと、ピンと呼ばれるプラグインで構成されます。ピンはそれぞれ個別のロック解除機能を提供します。現在利用できるピンは以下のとおりです。

tang

ネットワークサーバーを使用してボリュームのロックを解除できます。

tpm2

TPM2 ポリシーを使用してボリュームのロックを解除できます。

pkcs11

PKCS #11 URI を使用してボリュームのロックを解除できます。

sss

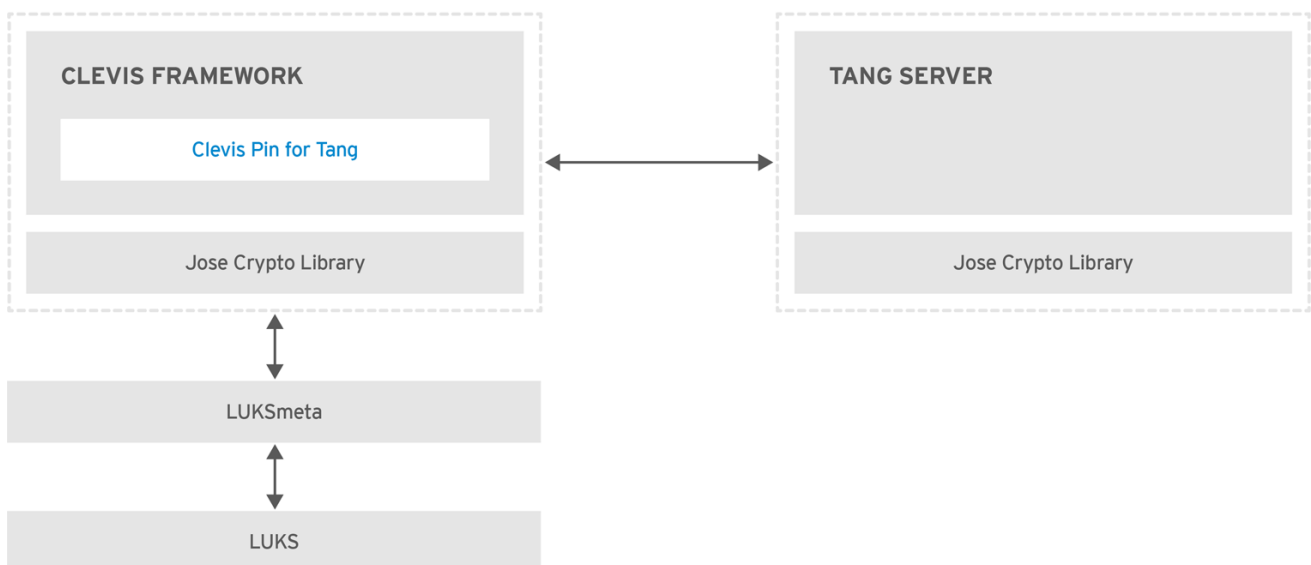
Shamir's Secret Sharing (SSS) 暗号化スキームを使用して高可用性システムをデプロイできます。

10.1. NETWORK-BOUND DISK ENCRYPTION

Network Bound Disc Encryption (NBDE) は、ポリシーベースの復号 (PBD) のサブカテゴリーであり、暗号化されたボリュームを特別なネットワークサーバーにバインドできるようにします。NBDE の現在の実装には、Tang サーバー自体と、Tang サーバー用の Clevis ピンが含まれます。

RHEL では、NBDE は次のコンポーネントとテクノロジーによって実装されます。

図10.1 LUKS1で暗号化したボリュームを使用する場合の NBDE スキーム(luksmeta パッケージは、LUKS2 ボリュームには使用されません)



RHEL_453350_0717

Tang は、ネットワークのプレゼンスにデータをバインドするためのサーバーです。セキュリティが保護された特定のネットワークにシステムをバインドする際に利用可能なデータを含めるようにします。Tang はステートレスで、TLS または認証は必要ありません。エスクローベースのソリューション (サーバーが暗号鍵をすべて保存し、使用されたことがあるすべての鍵に関する知識を有する) とは異なり、Tang はクライアントの鍵と相互作用することはないため、クライアントから識別情報を得ることがありません。

Clevis は、自動化された復号用のプラグイン可能なフレームワークです。NBDE では、Clevis は、LUKS ボリュームの自動アンロックを提供します。**clevis** パッケージは、クライアントで使用される機能を提供します。

Clevis ピン は、Clevis フレームワークへのプラグインです。このようなピンの1つに、Tang があります。これは NBDE サーバーとのやり取りを実装するプラグインです。

Clevis および Tang は、一般的なクライアントおよびサーバーのコンポーネントで、ネットワークがバインドされた暗号化を提供します。RHEL では、LUKS と組み合わせて使用され、ルートおよび非ルートストレージボリュームを暗号化および復号して、ネットワークにバインドされたディスク暗号化を実現します。

クライアントおよびサーバーのコンポーネントはともに **José** ライブラリーを使用して、暗号化および復号の操作を実行します。

NBDE のプロビジョニングを開始すると、Tang サーバーの Clevis ピンは、Tang サーバーの、アドバタイズされている非対称鍵のリストを取得します。もしくは、鍵が非対称であるため、Tang の公開鍵のリストを帯域外に配布して、クライアントが Tang サーバーにアクセスしなくても動作できるようにします。このモードは **オフラインプロビジョニング** と呼ばれます。

Tang 用の Clevis ピンは、公開鍵のいずれかを使用して、固有で、暗号論的に強力な暗号鍵を生成します。この鍵を使用してデータを暗号化すると、この鍵は破棄されます。Clevis クライアントは、使いやすい場所に、このプロビジョニング操作で生成した状態を保存する必要があります。データを暗号化するこのプロセスは **プロビジョニング手順** と呼ばれています。

LUKS バージョン 2 (LUKS2) は、RHEL のデフォルトのディスク暗号化形式であるため、NBDE のプロビジョニング状態は、LUKS2 ヘッダーにトークンとして保存されます。**luksmeta** パッケージによる NBDE のプロビジョニング状態は、LUKS1 で暗号化したボリュームにのみ使用されます。

Tang 用の Clevis ピンは、規格を必要とせずに LUKS1 と LUKS2 の両方をサポートします。Clevis はプレーンテキストファイルを暗号化できますが、ブロックデバイスの暗号化には **cryptsetup** ツールを使用する必要があります。詳細は、[LUKS を使用したブロックデバイスの暗号化](#) を参照してください。

クライアントがそのデータにアクセスする準備ができると、プロビジョニング手順で生成したメタデータを読み込み、応答して暗号鍵を戻します。このプロセスは **復旧手順** と呼ばれます。

Clevis は、NBDE ではピンを使用して LUKS ボリュームをバインドしているため、自動的にロックが解除されます。バインドプロセスが正常に終了すると、提供されている Dracut アンロックを使用してディスクをアンロックできます。



注記

kdump カーネルクラッシュのダンプメカニズムが、システムメモリーのコンテンツを LUKS で暗号化したデバイスに保存するように設定されている場合には、2 番目のカーネル起動時にパスワードを入力するように求められます。

関連情報

- ナレッジベース記事 [NBDE \(Network-Bound Disk Encryption\) Technology](#)

- システム上の **tang(8)**、**clevis(1)**、**jose(1)**、および **clevis-luks-unlockers(7)** man ページ
- ナレッジベースの記事 [How to set up Network-Bound Disk Encryption with multiple LUKS devices\(Clevis + Tang unlocking\)](#)

10.2. ENFORCING モードの SELINUX を使用して TANG サーバーをデプロイする

Tang サーバーを使用して、Clevis 対応クライアント上の LUKS 暗号化ボリュームのロックを自動的に解除できます。最小限のシナリオでは、**tang** パッケージをインストールし、**systemctl enable tangd.socket --now** コマンドを入力することにより、ポート 80 に Tang サーバーをデプロイします。次の手順の例では、SELinux 強制モードの限定サービスとしてカスタムポートで実行されている Tang サーバーのデプロイメントを示しています。

前提条件

- **polycoreutils-python-utils** パッケージおよび依存関係がインストールされている。
- **firewalld** サービスが実行中である。

手順

1. **tang** パッケージとその依存関係をインストールするには、**root** で以下のコマンドを実行します。

```
# dnf install tang
```

2. **7500/tcp** などの不要なポートを選択し、**tangd** サービスがそのポートにバインドできるようにします。

```
# semanage port -a -t tangd_port_t -p tcp 7500
```

ポートは1つのサービスのみで一度に使用できるため、すでに使用しているポートを使用しようとすると、**ValueError: Port already defined** エラーが発生します。

3. ファイアウォールのポートを開きます。

```
# firewall-cmd --add-port=7500/tcp
# firewall-cmd --runtime-to-permanent
```

4. **tangd** サービスを有効にします。

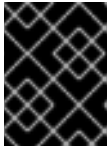
```
# systemctl enable tangd.socket
```

5. オーバーライドファイルを作成します。

```
# systemctl edit tangd.socket
```

6. 以下のエディター画面で、**/etc/systemd/system/tangd.socket.d/** ディレクトリーにある空の **override.conf** ファイルを開き、次の行を追加して、Tang サーバーのデフォルトのポートを、80 から、以前取得した番号に変更します。

```
[Socket]
ListenStream=
ListenStream=7500
```



重要

Anything between here と **# Lines below this** で始まる行の間に以前のコードスニペットを挿入します。挿入しない場合、システムは変更を破棄します。

7. 変更を保存し、エディターを終了します。デフォルトの **vi** エディターでこれを実行するには、**Esc** キーを押してコマンドモードに切り替え、**:wq** と入力して **Enter** キーを押します。
8. 変更した設定を再読み込みします。

```
# systemctl daemon-reload
```

9. 設定が機能していることを確認します。

```
# systemctl show tangd.socket -p Listen
Listen=[::]:7500 (Stream)
```

10. **tangd** サービスを開始します。

```
# systemctl restart tangd.socket
```

tangd が、**systemd** のソケットアクティベーションメカニズムを使用しているため、最初に接続するとすぐにサーバーが起動します。最初の起動時に、一組の暗号鍵が自動的に生成されます。鍵の手動生成などの暗号化操作を実行するには、**jose** ユーティリティーを使用します。

検証

- NBDE クライアントで、次のコマンドを使用して、Tang サーバーが正しく動作していることを確認します。このコマンドにより、暗号化と復号化に渡すものと同じメッセージが返される必要があります。

```
# echo test | clevis encrypt tang '{"url":"<tang.server.example.com:7500>"}' -y | clevis
decrypt
test
```

関連情報

- システム上の **tang(8)**、**semanage(8)**、**firewall-cmd(1)**、**jose(1)**、**systemd.unit(5)** および **systemd.socket(5)** man ページ

10.3. TANG サーバーの鍵のローテーションおよびクライアントでのバインディングの更新

セキュリティ上の理由から、Tang サーバーの鍵をローテーションし、クライアント上の既存のバインディングを定期的に更新してください。鍵をローテートするのに適した間隔は、アプリケーション、鍵のサイズ、および組織のポリシーにより異なります。

したがって、**nbde_server** RHEL システムロールを使用して、Tang 鍵をローテーションできます。詳細は [複数の Tang サーバー設定での nbde_server システムロールの使用](#) を参照してください。

前提条件

- Tang サーバーが実行している。
- **clevis** パッケージおよび **clevis-luks** パッケージがクライアントにインストールされている。

手順

1. **/var/db/tang** 鍵データベースディレクトリーのすべての鍵の名前の前に **.** を指定して、アドバタイズメントに対して非表示にします。以下の例のファイル名は、Tang サーバーの鍵データベースディレクトリーにある一意のファイル名とは異なります。

```
# cd /var/db/tang
# ls -l
-rw-r--r--. 1 root root 349 Feb  7 14:55 UV6dqXSwe1bRKG3KbJmdiR020hY.jwk
-rw-r--r--. 1 root root 354 Feb  7 14:55 y9hxLTQSiSB5jSEGWnjhY8fDTJU.jwk
# mv UV6dqXSwe1bRKG3KbJmdiR020hY.jwk .UV6dqXSwe1bRKG3KbJmdiR020hY.jwk
# mv y9hxLTQSiSB5jSEGWnjhY8fDTJU.jwk .y9hxLTQSiSB5jSEGWnjhY8fDTJU.jwk
```

2. 名前が変更され、Tang サーバーのアドバタイズに対してすべての鍵が非表示になっていることを確認します。

```
# ls -l
total 0
```

3. Tang サーバーの **/var/db/tang** で **/usr/libexec/tangd-keygen** コマンドを使用して新しい鍵を生成します。

```
# /usr/libexec/tangd-keygen /var/db/tang
# ls /var/db/tang
3ZWS6-cDrCG61UPJS2BMmPU4I54.jwk zyLuX6hijUy_PSeUEFDi7hi38.jwk
```

4. Tang サーバーが、以下のように新規キーペアから署名キーを公開していることを確認します。

```
# tang-show-keys 7500
3ZWS6-cDrCG61UPJS2BMmPU4I54
```

5. NBDE クライアントで **clevis luks report** コマンドを使用して、Tang サーバーでアドバタイズされた鍵が同じままかどうかを確認します。**clevis luks list** コマンドを使用すると、関連するバインドイングのあるスロットを特定できます。以下に例を示します。

```
# clevis luks list -d /dev/sda2
1: tang '{"url":"http://tang.srv"}'
# clevis luks report -d /dev/sda2 -s 1
...
Report detected that some keys were rotated.
Do you want to regenerate luks metadata with "clevis luks regen -d /dev/sda2 -s 1"? [ynYN]
```

6. 新しい鍵の LUKS メタデータを再生成するには、直前のコマンドプロンプトで **y** を押すか、**clevis luks regen** コマンドを使用します。

```
# clevis luks regen -d /dev/sda2 -s 1
```

- すべての古いクライアントが新しい鍵を使用することを確認したら、Tang サーバーから古い鍵を削除できます。次に例を示します。

```
# cd /var/db/tang
# rm *.jwk
```



警告

クライアントが使用している最中に古い鍵を削除すると、データが失われる場合があります。このような鍵を誤って削除した場合は、クライアントで **clevis luks regen** コマンドを実行し、LUKS パスワードを手動で提供します。

関連情報

- システム上の **tang-show-keys(1)**、**clevis-luks-list(1)**、**clevis-luks-report(1)**、および **clevis-luks-regen(1)** man ページ

10.4. WEB コンソールで TANG キーを使用して自動ロック解除を設定する

Tang サーバーが提供する鍵を使用して、LUKS で暗号化したストレージデバイスの自動ロック解除を設定できます。

前提条件

- RHEL 9 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- cockpit-storaged** と **clevis-luks** パッケージがシステムにインストールされている。
- cockpit.socket** サービスがポート 9090 で実行されている。
- Tang サーバーを利用できる。詳細は、[Deploying a Tang server with SELinux in enforcing mode](#) を参照してください。
- root** 特権、または **sudo** を使用して管理コマンドを入力する権限がある。

手順

- RHEL 9 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。

2. 管理者アクセスに切り替え、認証情報を入力して、**Storage** をクリックします。**Storage** テーブルで、自動的にロックを解除するために追加する予定の暗号化ボリュームが含まれるディスクをクリックします。
3. 次のページに選択したディスクの詳細が表示されたら、**Keys** セクションの **+** をクリックして Tang 鍵を追加します。

The screenshot shows the configuration page for a storage device. At the top, the breadcrumb navigation is "Storage > vda - VirtIO Disk > vda2". Below this is a table with the following details:

Name	-
UUID	44d29c6b-02
Type	Linux filesystem data edit
Size	15.0 GB

Below the table is the "Encryption" section with the following details:

Encryption type	LUKS2
Cleartext device	/dev/mapper/luks-37128c9a-70a2-483f-8d64-9f00acf80449
Stored passphrase	none edit
Options	discard edit

Below the encryption section is the "Keys" section. It has a "+" button to add a new key. Below that, there is a table with one row:

Passphrase	Slot 0
------------	--------

To the right of the "Passphrase" column is an edit icon (pencil) and a "-" button to remove the key.

4. **Key source** として **Tang keyserver** を選択し、Tang サーバーのアドレスと、LUKS で暗号化されたデバイスのロックを解除するパスワードを入力します。**Add** をクリックして確定します。

The screenshot shows the "Add key" dialog box. It has the following fields and options:

- Key source:** Two radio buttons are present: "Passphrase" (unselected) and "Tang keyserver" (selected).
- Keyserver address:** A text input field containing "tang1.example.com:7500".
- Disk passphrase:** A text input field filled with dots, representing a password. To the right of the field is an eye icon to toggle visibility.

Below the input fields is a message: "Saving a new passphrase requires unlocking the disk. Please provide a current disk passphrase."

At the bottom of the dialog are two buttons: "Add" (in blue) and "Cancel" (in light blue).

以下のダイアログウィンドウは、鍵ハッシュが一致することを確認するコマンドを提供します。

5. Tang サーバーのターミナルで、**tang-show-keys** コマンドを使用して、比較のためにキーハッシュを表示します。この例では、Tang サーバーはポート 7500 で実行されています。

```
# tang-show-keys 7500
x100_1k6GPiDOaMIL3WbpCjHOy9ul1bSfdhI3M08wO0
```

6. Web コンソールと前述のコマンドの出力のキーハッシュが同じ場合は、**Trust key** をクリックします。

Verify key

Check the key hash with the Tang server.

How to check

In a terminal, run: `ssh tang1. com tang-show-keys`

Check that the SHA-256 or SHA-1 hash from the command matches this dialog.

SHA-256

x100_1k6GPiDOaMIL3WbpCjHOy9ul1bSfdhI3M08wO0

SHA-1

hmINhleYB000ddFszgICjqJizFI

Trust key

Cancel

7. RHEL 9.2 以降では、暗号化されたルートファイルシステムと Tang サーバーを選択した後、カーネルコマンドラインへの **rd.neednet=1** パラメーターの追加、**clevis-dracut** パッケージのインストール、および初期 RAM ディスクイメージ (**initrd**) の再生成をスキップできます。非ルートファイルシステムの場合、Web コンソールは、**remote-cryptsetup.target** および **clevis-luks-akspass.path systemd** ユニットを有効にし、**clevis-systemd** パッケージをインストールし、**_netdev** パラメーターを **fstab** および **crypttab** 設定ファイルに追加するようになりました。

検証

1. 新規に追加された Tang キーが **Keyserver** タイプの **Keys** セクションにリスト表示されていることを確認します。



Encryption

Encryption type	LUKS2
Cleartext device	/dev/mapper/luks-37128c9a-70a2-483f-8d64-9f00acf80449
Stored passphrase	none edit
Options	discard edit

Keys

Passphrase



Slot 0

Keyserver

http://tang1. com/

Slot 1

2. バインディングが初期ブートで使用できることを確認します。次に例を示します。

```
# lsinitrd | grep clevis-luks
lrwxrwxrwx 1 root root 48 Jan 4 02:56
etc/systemd/system/cryptsetup.target.wants/clevis-luks-askpass.path ->
/usr/lib/systemd/system/clevis-luks-askpass.path
...
```

関連情報

- [RHEL Web コンソールの使用](#)

10.5. 基本的な NBDE および TPM2 暗号化クライアント操作

Clevis フレームワークは、プレーンテキストファイルを暗号化し、JSON Web Encryption (JWE) 形式の暗号化テキストと LUKS 暗号化ブロックデバイスの両方を復号できます。Clevis クライアントは、暗号化操作に Tang ネットワークサーバーまたは Trusted Platform Module 2.0 (TPM 2.0) チップのいずれかを使用できます。

次のコマンドは、プレーンテキストファイルが含まれる例で Clevis が提供する基本的な機能を示しています。また、NBDE または Clevis + TPM のデプロイメントのトラブルシューティングにも使用できます。

Tang サーバーにバインドされた暗号化クライアント

- Clevis 暗号化クライアントが Tang サーバーにバインドされることを確認するには、**clevis encrypt tang** サブコマンドを使用します。

```
$ clevis encrypt tang '{"url":"http://tang.srv:port"}' < input-plain.txt > secret.jwe
The advertisement contains the following signing keys:

_OsIk0T-E2l6qjfdDiwVmidoZjA

Do you wish to trust these keys? [ynYN] y
```

上記の例の **http://tang.srv:port** URL は、**tang** がインストールされているサーバーの URL と一致するように変更します。**secret.jwe** 出力ファイルには、JWE 形式で暗号化した暗号文が含まれます。この暗号文は **input-plain.txt** 入力ファイルから読み込まれます。

また、設定に SSH アクセスなしで Tang サーバーとの非対話型の通信が必要な場合は、アドバタイズメントをダウンロードしてファイルに保存できます。

```
$ curl -sfg http://tang.srv:port/adv -o adv.jws
```

adv.jws ファイル内のアドバタイズメントは、ファイルやメッセージの暗号化など、後続のタスクに使用します。

```
$ echo 'hello' | clevis encrypt tang '{"url":"http://tang.srv:port","adv":"adv.jws"}
```

- データを復号するには、**clevis decrypt** コマンドを実行して、暗号文 (JWE) を提供します。

```
$ clevis decrypt < secret.jwe > output-plain.txt
```

TPM 2.0 を使用する暗号化クライアント

- TPM 2.0 チップを使用して暗号化するには、JSON 設定オブジェクト形式の引数のみが使用されている **clevis encrypt tpm2** サブコマンドを使用します。

```
$ clevis encrypt tpm2 '{}' < input-plain.txt > secret.jwe
```

別の階層、ハッシュ、および鍵アルゴリズムを選択するには、以下のように、設定プロパティを指定します。

```
$ clevis encrypt tpm2 '{"hash":"sha256","key":"rsa"}' < input-plain.txt > secret.jwe
```

- データを復号するには、JSON Web Encryption (JWE) 形式の暗号文を提供します。

```
$ clevis decrypt < secret.jwe > output-plain.txt
```

ピンは、PCR (Platform Configuration Registers) 状態へのデータのシーリングにも対応します。このように、PCR ハッシュ値が、シーリング時に使用したポリシーと一致する場合にのみ、データのシーリングを解除できます。

たとえば、SHA-256 バンクに対して、インデックス 0 および 7 の PCR にデータをシールするには、以下を行います。

```
$ clevis encrypt tpm2 '{"pcr_bank":"sha256","pcr_ids":"0,7"}' < input-plain.txt > secret.jwe
```

**警告**

PCR のハッシュは書き換えることができ、暗号化されたボリュームのロックを解除することはできなくなりました。このため、PCR の値が変更された場合でも、暗号化されたボリュームのロックを手動で解除できる強力なパスワードを追加します。

shim-x64 パッケージのアップグレード後にシステムが暗号化されたボリュームのロックを自動的に解除できない場合は、Red Hat ナレッジベースソリューション [Clevis TPM2 no longer decrypts LUKS devices after a restart](#) を参照してください。

関連情報

- システム上の **clevis-encrypt-tang(1)**、**clevis-luks-unlockers(7)**、**clevis(1)**、および **clevis-encrypt-tpm2(1)** man ページ
- 以下のように引数指定せずに **clevis**、**clevis decrypt** および **clevis encrypt tang** コマンドを入力したときに表示される組み込み CLI。

```
$ clevis encrypt tang
Usage: clevis encrypt tang CONFIG < PLAINTEXT > JWE
...
```

10.6. LUKS 暗号化ボリュームのロックを自動解除するように NBDE クライアントを設定する

Clevis フレームワークを使用すると、選択した Tang サーバーが使用可能な場合に、LUKS 暗号化ボリュームのロックを自動解除するようにクライアントを設定できます。これにより、Network-Bound Disk Encryption (NBDE) デプロイメントが作成されます。

前提条件

- Tang サーバーが実行されていて、使用できるようにしてある。

手順

- 既存の LUKS 暗号化ボリュームのロックを自動的に解除するには、**clevis-luks** サブパッケージをインストールします。

```
# dnf install clevis-luks
```

- PBD 用 LUKS 暗号化ボリュームを特定します。次の例では、ブロックデバイスは **/dev/sda2** と呼ばれています。

```
# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda                                8:0  0  12G  0 disk
├─sda1                            8:1  0   1G  0 part  /boot
└─sda2                            8:2  0  11G  0 part
```

```
└─luks-40e20552-2ade-4954-9d56-565aa7994fb6 253:0 0 11G 0 crypt
  └─rhel-root 253:0 0 9.8G 0 lvm /
    └─rhel-swap 253:1 0 1.2G 0 lvm [SWAP]
```

3. **clevis luks bind** コマンドを使用して、ボリュームを Tang サーバーにバインドします。

```
# clevis luks bind -d /dev/sda2 tang '{"url":"http://tang.srv"}'
The advertisement contains the following signing keys:

_OsIk0T-E2l6qjfdDiwVmidoZjA

Do you wish to trust these keys? [ynYN] y
You are about to initialize a LUKS device for metadata storage.
Attempting to initialize it may result in data loss if data was
already written into the LUKS header gap in a different format.
A backup is advised before initialization is performed.

Do you wish to initialize /dev/sda2? [yn] y
Enter existing LUKS password:
```

このコマンドは、以下の 4 つの手順を実行します。

- LUKS マスター鍵と同じエントロピーを使用して、新しい鍵を作成します。
- Clevis で新しい鍵を暗号化します。
- LUKS2 ヘッダートークンに Clevis JWE オブジェクトを保存するか、デフォルト以外の LUKS1 ヘッダーが使用されている場合は LUKSMeta を使用します。
- LUKS を使用する新しい鍵を有効にします。



注記

バインド手順では、空き LUKS パスワードスロットが少なくとも 1 つあることが前提となっています。そのスロットの 1 つを **clevis luks bind** コマンドが使用します。

ボリュームは、現在、既存のパスワードと Clevis ポリシーを使用してロックを解除できます。

- システムの起動プロセスの初期段階でディスクバインディングを処理するには、インストール済みのシステムで **dracut** ツールを使用します。RHEL では、Clevis はホスト固有の設定オプションを指定せずに汎用 **initrd** (初期 RAM ディスク) を生成し、カーネルコマンドラインに **rd.neednet=1** などのパラメーターを自動的に追加しません。初期の起動時にネットワークを必要とする Tang ピンを使用する場合は、**--hostonly-cmdline** 引数を使用し、**dracut** が Tang バインディングを検出すると **rd.neednet=1** を追加します。
 - clevis-dracut** パッケージをインストールします。

```
# dnf install clevis-dracut
```

- 初期 RAM ディスクを再生成します。

```
# dracut -fv --regenerate-all --hostonly-cmdline
```

- c. または、`/etc/dracut.conf.d/` ディレクトリーに `.conf` ファイルを作成し、そのファイルに **hostonly_cmdline=yes** オプションを追加します。すると、**--hostonly-cmdline** なしで **dracut** を使用できます。次に例を示します。

```
# echo "hostonly_cmdline=yes" > /etc/dracut.conf.d/clevis.conf
# dracut -fv --regenerate-all
```

- d. Clevis がインストールされているシステムで **grubby** ツールを使用して、システム起動時の早い段階で Tang ピンのネットワークを利用できるようにすることができます。

```
# grubby --update-kernel=ALL --args="rd.neednet=1"
```

検証

1. Clevis JWE オブジェクトが LUKS ヘッダーに正常に配置されていることを確認します。 **clevis luks list** コマンドを使用します。

```
# clevis luks list -d /dev/sda2
1: tang '{"url":"http://tang.srv:port"}'
```

2. バインディングが初期ブートで使用できることを確認します。次に例を示します。

```
# lsinitrd | grep clevis-luks
lrwxrwxrwx 1 root root 48 Jan 4 02:56
etc/systemd/system/cryptsetup.target.wants/clevis-luks-askpass.path ->
/usr/lib/systemd/system/clevis-luks-askpass.path
...
```

関連情報

- システム上の **clevis-luks-bind(1)** および **dracut.cmdline(7)** man ページ
- [ネットワークのブートオプション](#)
- [Looking forward to Linux network configuration in the initial ramdisk \(initrd\)](#) (Red Hat Enable Sysadmin)

10.7. 静的な IP 設定を持つ NBDE クライアントの設定

(DHCP を使用しない) 静的な IP 設定を持つクライアントに NBDE を使用するには、ネットワーク設定を **dracut** ツールに手動で渡す必要があります。

前提条件

- Tang サーバーが実行されていて、使用できるようにしてある。
- Tang サーバーによって暗号化されたボリュームのロックを自動解除するように NBDE クライアントが設定されている。
詳細は、[LUKS 暗号化ボリュームのロックを自動解除するように NBDE クライアントを設定する](#) を参照してください。

手順

1. 静的ネットワーク設定を、**dracut** コマンドの **kernel-cmdline** オプションの値として指定できます。次に例を示します。

```
# dracut -fv --regenerate-all --kernel-cmdline
"ip=192.0.2.10::192.0.2.1:255.255.255.0::ens3:none nameserver=192.0.2.100"
```

2. または、静的ネットワーク情報を含む .conf ファイルを **/etc/dracut.conf.d/** ディレクトリーに作成し、初期 RAM ディスクイメージを再生成します。

```
# cat /etc/dracut.conf.d/static_ip.conf
kernel_cmdline="ip=192.0.2.10::192.0.2.1:255.255.255.0::ens3:none
nameserver=192.0.2.100"
# dracut -fv --regenerate-all
```

10.8. TPM 2.0 ポリシーを使用して LUKS 暗号化ボリュームの手動登録を設定する

Trusted Platform Module 2.0 (TPM 2.0) ポリシーを使用して、LUKS 暗号化ボリュームのロック解除を設定できます。

前提条件

- アクセス可能な TPM2.0 互換デバイス。
- システムが 64 ビット Intel アーキテクチャー、または 64 ビット AMD アーキテクチャーである。

手順

1. 既存の LUKS 暗号化ボリュームのロックを自動的に解除するには、**clevis-luks** サブパッケージをインストールします。

```
# dnf install clevis-luks
```

2. PBD 用 LUKS 暗号化ボリュームを特定します。次の例では、ブロックデバイスは **/dev/sda2** と呼ばれています。

```
# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda                                8:0  0   12G  0 disk
├─sda1                            8:1  0    1G  0 part  /boot
└─sda2                            8:2  0   11G  0 part
   └─luks-40e20552-2ade-4954-9d56-565aa7994fb6 253:0  0   11G  0 crypt
      ├─rhel-root                    253:0  0   9.8G  0 lvm   /
      └─rhel-swap                    253:1  0   1.2G  0 lvm   [SWAP]
```

3. **clevis luks bind** コマンドを使用して、ボリュームを TPM 2.0 デバイスにバインドします。以下に例を示します。

```
# clevis luks bind -d /dev/sda2 tpm2 '{"hash":"sha256","key":"rsa"}'
...
Do you wish to initialize /dev/sda2? [yn] y
Enter existing LUKS password:
```

-

このコマンドは、以下の 4 つの手順を実行します。

- a. LUKS マスター鍵と同じエントロピーを使用して、新しい鍵を作成します。
- b. Clevis で新しい鍵を暗号化します。
- c. LUKS2 ヘッダートークンに Clevis JWE オブジェクトを保存するか、デフォルト以外の LUKS1 ヘッダーが使用されている場合は LUKSMeta を使用します。
- d. LUKS を使用する新しい鍵を有効にします。



注記

バインド手順では、空き LUKS パスワードスロットが少なくとも 1 つあることが前提となっています。そのスロットの 1 つを **clevis luks bind** コマンドが使用します。

あるいは、特定の Platform Configuration Registers (PCR) の状態にデータをシールする場合は、**clevis luks bind** コマンドに **pcr_bank** と **pcr_ids** 値を追加します。以下に例を示します。

```
# clevis luks bind -d /dev/sda2 tpm2
'{"hash":"sha256","key":"rsa","pcr_bank":"sha256","pcr_ids":["0,1"]}'
```



重要

PCR ハッシュ値がシール時に使用されるポリシーと一致し、ハッシュを書き換えることができる場合にのみ、データをアンシールできるため、PCR の値が変更された場合、暗号化されたボリュームのロックを手動で解除できる強力なパズルを追加します。

shim-x64 パッケージをアップグレードした後、システムが暗号化されたボリュームのロックを自動的に解除できない場合は、Red Hat ナレッジベースソリューション [Clevis TPM2 no longer decrypts LUKS devices after a restart](#) を参照してください。

4. ボリュームは、現在、既存のパスワードと Clevis ポリシーを使用してロックを解除できます。
5. システムの起動プロセスの初期段階でディスクバイndingを処理するには、インストール済みのシステムで **dracut** ツールを使用します。

```
# dnf install clevis-dracut
# dracut -fv --regenerate-all
```

検証

1. Clevis JWE オブジェクトが LUKS ヘッダーに適切に置かれていることを確認するには、**clevis luks list** コマンドを使用します。

```
# clevis luks list -d /dev/sda2
1: tpm2 '{"hash":"sha256","key":"rsa"}
```


関連情報

- システム上の **clevis-luks-bind(1)**、**clevis-encrypt-tpm2(1)**、**dracut.cmdline(7)** man ページ

10.9. PKCS #11 ピンを使用して LUKS 暗号化ボリュームのロック解除を設定する

PKCS #11 と互換性のあるデバイス (スマートカードまたはハードウェアセキュリティーモジュール (HSM)) を使用して、LUKS で暗号化されたボリュームのロック解除を設定できます。

Clevis PKCS #11 ピンを使用して暗号化されたボリュームを自動的にロック解除するには、**/etc/crypttab** ファイルの変更も必要です。この変更により、コンソールでユーザーにプロンプトを表示する代わりに、**AF_UNIX** ソケットを使用して、ボリュームのロックを解除するためのキーフレーズを待機するように **systemd** マネージャーを設定します。

Clevis PKCS #11 のユニットファイルは、ディスクのロック解除に関する情報を送受信するためのソケットを **/run/systemd/clevis-pkcs11.sock** ファイル内に設定します。Clevis PKCS #11 ピンによってロック解除されるディスクの場合は、ソケットファイルをキーファイルとして設定する必要があります。

前提条件

- PKCS #11 デバイスがすでに設定されており、アクセス可能である。
- clevis-pin-pkcs11** パッケージがインストールされている。
- clevis luks bind** コマンド用に、LUKS パスワードの空きスロットが少なくとも1つある。

手順

- PBD 用 LUKS 暗号化ボリュームを特定します。次の例では、ブロックデバイスは **/dev/sda2** と呼ばれています。

```
# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda                                8:0  0  12G  0 disk
├─sda1                            8:1  0   1G  0 part  /boot
├─sda2                            8:2  0  11G  0 part
│   └─luks-40e20552-2ade-4954-9d56-565aa7994fb6 253:0  0  11G  0 crypt
│       ├─rhel-root                253:0  0   9.8G  0 lvm  /
│       └─rhel-swap                253:1  0   1.2G  0 lvm  [SWAP]
```

- ボリュームのロック解除に使用する PKCS #11 デバイスの URI を特定します。次に例を示します。

```
$ pkcs11-tool -L | grep uri
uri          :
pkcs11:model=PKCS%2315%20emulated;manufacturer=piv_II;serial=42facd1f749ece7f;token=
clevis
uri          :
pkcs11:model=PKCS%2315%20emulated;manufacturer=OpenPGP%20project;serial=000f060
80f4f;token=OpenPGP%20card%20%28User%20PIN%29
```

- clevis luks bind** コマンドを使用してボリュームを PKCS #11 デバイスにバインドします。次に例を示します。

```
# clevis luks bind -d /dev/sda2 pkcs11
'{"uri":"pkcs11:model=PKCS%2315%20emulated;manufacturer=OpenPGP%20project;serial=00f06080f4f;token=OpenPGP%20card%20%28User%20PIN%29;id=%03;object=Authentication%20key;type=public"}'
...
Do you wish to initialize /dev/sda2? [yn] y
Enter existing LUKS password:
```

このコマンドは、次の手順を実行します。

- a. LUKS マスター鍵と同じエントロピーを使用して、新しい鍵を作成します。
 - b. Clevis で新しい鍵を暗号化します。
 - c. LUKS2 ヘッダートークンに Clevis JWE オブジェクトを保存するか、デフォルト以外の LUKS1 ヘッダーが使用されている場合は LUKSMeta を使用します。
 - d. LUKS を使用する新しい鍵を有効にします。
4. オプション: 使用するモジュールを指定する必要がある場合は、`module-path` URI パラメーターを追加します。

```
# clevis luks bind -d /dev/sda2 pkcs11 '{"uri":"pkcs11:module-path=/usr/lib64/libykcs11.so.2";model=PKCS%2315%20emulated;manufacturer=OpenPGP%20project;serial=00f06080f4f;token=OpenPGP%20card%20%28User%20PIN%29;id=%03;object=Authentication%20key;type=public}'
```

5. **clevis-luks-pkcs11-askpass.socket** ユニットを有効にします。

```
# systemctl enable --now clevis-luks-pkcs11-askpass.socket
```

6. テキストエディターで `/etc/crypttab` ファイルを開き、PKCS #11 ピンでロックを解除する LUKS 暗号化ボリュームを含む行を特定します。次に例を示します。

```
luks-6e38d5e1-7f83-43cc-819a-7416bcbf9f84 UUID=6e38d5e1-7f83-43cc-819a-7416bcbf9f84 - -
```

7. ダッシュを `/run/systemd/clevis-pkcs11.sock` ファイルパスと **keyfile-timeout** オプションに置き換えます。

```
luks-6e38d5e1-7f83-43cc-819a-7416bcbf9f84 UUID=6e38d5e1-7f83-43cc-819a-7416bcbf9f84 /run/systemd/clevis-pkcs11.sock keyfile-timeout=30s
```

keyfile-timeout オプションは、ロック解除エラーが発生し、システムがコンソールからパズルを手動で入力する必要がある場合に、次の処理に移行する仕組みを提供します。

8. 変更を保存し、エディターを終了します。
9. システムの起動プロセスの初期段階で、ルートファイルシステムのロックを解除するために必要なディスクバインディングを処理するには、インストール済みのシステムで **dracut** ツールを使用します。

```
# dracut -fv --regenerate-all
```

10. システムを再起動します。

次のブートプロセス中に、PKCS #11 デバイス PIN の入力が必要です。正しい PIN を入力した場合にのみ、対応する設定済みの暗号化ディスクが復号化されます。

検証

- ブートプロセスを手動でテストする代わりに、次のコマンドを使用してテキストメッセージを暗号化および復号化できます。

```
# echo "top secret" | clevis encrypt pkcs11 '{"uri":"pkcs11:module-path=/usr/lib64/libykcs11.so.2?pin-value=<PIN>"}' | clevis decrypt
```

<PIN> は PIN 値に置き換えます。メッセージを復号化するには、この PIN 値を入力する必要があります。

- Clevis JWE オブジェクトが LUKS ヘッダーに正常に配置されていることを確認するには、**clevis luks list** コマンドを使用します。次に例を示します。

```
# clevis luks list -d /dev/sda2
1: pkcs11 '{"uri": "pkcs11:model=PKCS%2315%20emulated;manufacturer=piv_II;serial=0a35ba26b062b9c5;token=clevis;id=%02;object=Encryption%20Key?module-path=/usr/lib64/libykcs11.so.2"}'
```

関連情報

- システム上の **clevis-luks-bind(1)**、**clevis-encrypt-pkcs11(1)**、および **dracut.cmdline(7)** man ページ

10.10. LUKS で暗号化したボリュームからの CLEVIS ピンの手動削除

clevis luks bind コマンドで作成されたメタデータを手動で削除する場合や、Clevis が追加したパズルズを含む鍵スロットを一掃するには、以下の手順を行います。

重要

LUKS で暗号化したボリュームから Clevis ピンを削除する場合は、**clevis luks unbind** コマンドを使用することが推奨されます。**clevis luks unbind** を使用した削除手順は、1 回のステップで構成され、LUKS1 ボリュームおよび LUKS2 ボリュームの両方で機能します。次のコマンド例は、バインド手順で作成されたメタデータを削除し、**/dev/sda2** デバイスの鍵スロット **1** を削除します。

```
# clevis luks unbind -d /dev/sda2 -s 1
```

前提条件

- Clevis バインディングを使用した LUKS 暗号化ボリューム。

手順

- /dev/sda2** などのボリュームがどの LUKS バージョンであるかを確認し、Clevis にバインドされているスロットおよびトークンを特定します。

```
# cryptsetup luksDump /dev/sda2
LUKS header information
Version:      2
...
Keyslots:
  0: luks2
...
  1: luks2
    Key:      512 bits
    Priority:  normal
    Cipher:   aes-xts-plain64
...
  Tokens:
    0: clevis
      Keyslot: 1
...
```

上記の例では、Clevis トークンは **0** で識別され、関連付けられたキースロットは **1** です。

2. LUKS2 暗号化の場合は、トークンを削除します。

```
# cryptsetup token remove --token-id 0 /dev/sda2
```

3. デバイスを LUKS1 で暗号化し、**cryptsetup luksDump** コマンドの出力に **Version: 1** 文字列が表示されている場合は、**luksmeta wipe** コマンドでこの追加手順を行います。

```
# luksmeta wipe -d /dev/sda2 -s 1
```

4. Clevis パスフレーズを含む鍵スロットを削除します。

```
# cryptsetup luksKillSlot /dev/sda2 1
```

関連情報

- システム上の **clevis-luks-unbind(1)**、**cryptsetup(8)**、**luksmeta(8)** man ページ

10.11. キックスタートを使用して LUKS 暗号化ボリュームの自動登録を設定する

この手順に従って、LUKS で暗号化されたボリュームの登録に Clevis を使用する自動インストールプロセスを設定します。

手順

1. 一時パスワードを使用して、LUKS 暗号化が有効になっているディスクを、**/boot** 以外のすべてのマウントポイントで分割するように、キックスタートに指示します。パスワードは、登録プロセスの手順に使用するための一時的なものです。

```
part /boot --fstype="xfs" --ondisk=vda --size=256
part / --fstype="xfs" --ondisk=vda --grow --encrypted --passphrase=temppass
```

OSPP 準拠のシステムには、より複雑な設定が必要であることに注意してください。次に例を示します。

```
part /boot --fstype="xfs" --ondisk=vda --size=256
part / --fstype="xfs" --ondisk=vda --size=2048 --encrypted --passphrase=temppass
part /var --fstype="xfs" --ondisk=vda --size=1024 --encrypted --passphrase=temppass
part /tmp --fstype="xfs" --ondisk=vda --size=1024 --encrypted --passphrase=temppass
part /home --fstype="xfs" --ondisk=vda --size=2048 --grow --encrypted --
passphrase=temppass
part /var/log --fstype="xfs" --ondisk=vda --size=1024 --encrypted --passphrase=temppass
part /var/log/audit --fstype="xfs" --ondisk=vda --size=1024 --encrypted --
passphrase=temppass
```

2. 関連する Clevis パッケージを **%packages** セクションに追加して、インストールします。

```
%packages
clevis-dracut
clevis-luks
clevis-systemd
%end
```

3. オプション: 必要に応じて暗号化されたボリュームのロックを手動で解除できるようにするには、一時パズフレーズを削除する前に強力なパズフレーズを追加します。詳細は、Red Hat ナレッジベースソリューション [How to add a passphrase, key, or keyfile to an existing LUKS device](#) を参照してください。
4. **clevis luks bind** を呼び出して、**%post** セクションのバインディングを実行します。その後、一時パスワードを削除します。

```
%post
clevis luks bind -y -k - -d /dev/vda2 \
tang '{"url":"http://tang.srv"}' <<< "temppass"
cryptsetup luksRemoveKey /dev/vda2 <<< "temppass"
dracut -fv --regenerate-all
%end
```

設定が起動初期にネットワークを必要とする Tang ピンに依存している場合、または静的 IP 設定の NBDE クライアントを使用している場合は、[Configuring manual enrollment of LUKS-encrypted volumes](#)に従って **dracut** コマンドを変更する必要があります。

clevis luks bind コマンドの **-y** オプションは、RHEL 8.3 から使用できることに注意してください。RHEL 8.2 以前では、**clevis luks bind** コマンドで **-y** を **-f** に置き換え、Tang サーバーからアドバタイズメントをダウンロードします。

```
%post
curl -sfg http://tang.srv/adv -o adv.jws
clevis luks bind -f -k - -d /dev/vda2 \
tang '{"url":"http://tang.srv","adv":"adv.jws"}' <<< "temppass"
cryptsetup luksRemoveKey /dev/vda2 <<< "temppass"
dracut -fv --regenerate-all
%end
```



警告

cryptsetup luksRemoveKey コマンドは、それを適用する LUKS2 デバイスがそれ以上に管理されるのを防ぎます。LUKS1 デバイスに対してのみ **dmsetup** コマンドを使用して、削除されたマスターキーを回復できます。

Tang サーバーの代わりに TPM 2.0 ポリシーを使用する場合は、同様の手順を使用できます。

関連情報

- システム上の **clevis(1)**、**clevis-luks-bind(1)**、**cryptsetup(8)**、および **dmsetup(8)** man ページ
- [RHEL の自動インストール](#)

10.12. LUKS で暗号化されたリムーバブルストレージデバイスの自動ロック解除を設定する

LUKS 暗号化 USB ストレージデバイスの自動ロック解除プロセスを設定できます。

手順

1. USB ドライブなど、LUKS で暗号化したリムーバブルストレージデバイスを自動的にアンロックするには、**clevis-udisks2** パッケージをインストールします。

```
# dnf install clevis-udisks2
```

2. システムを再起動し、[LUKS で暗号化したボリュームの手動登録の設定](#) に従って、**clevis luks bind** コマンドを使用したバインディング手順を実行します。以下に例を示します。

```
# clevis luks bind -d /dev/sdb1 tang '{"url":"http://tang.srv"}'
```

3. これで、LUKS で暗号化されたリムーバブルデバイスを、GNOME デスクトップセッションで自動的にロック解除できるようになりました。Clevis ポリシーにバインドするデバイスは、**clevis luks unlock** コマンドでアンロックできます。

```
# clevis luks unlock -d /dev/sdb1
```

Tang サーバーの代わりに TPM 2.0 ポリシーを使用する場合は、同様の手順を使用できます。

関連情報

- システム上の **clevis-luks-unlockers(7)** man ページ

10.13. 高可用性 NBDE システムをデプロイする

Tang は、高可用性デプロイメントを構築する方法を 2 つ提供します。

クライアントの冗長性 (推奨)

クライアントは、複数の Tang サーバーにバインドする機能を使用して設定する必要があります。この設定では、各 Tang サーバーに独自の鍵があり、クライアントは、このサーバーのサブセットに接続することで復号できます。Clevis はすでに、**sss** プラグインを使用してこのワークフローに対応しています。Red Hat は、高可用性のデプロイメントにこの方法を推奨します。

鍵の共有

冗長性を確保するために、Tang のインスタンスは複数デプロイできます。2 つ目以降のインスタンスを設定するには、**tang** パッケージをインストールし、**SSH** 経由で **rsync** を使用してその鍵ディレクトリーを新規ホストにコピーします。鍵を共有すると鍵への不正アクセスのリスクが高まり、追加の自動化インフラストラクチャーが必要になるため、Red Hat はこの方法を推奨していません。

シャミアの秘密分散を使用した高可用性 NBDE

シャミアの秘密分散 (SSS) は、秘密を複数の固有のパーツに分割する暗号スキームです。秘密を再構築するには、いくつかのパーツが必要になります。数値はしきい値と呼ばれ、SSS はしきい値スキームとも呼ばれます。

Clevis は、SSS の実装を提供します。鍵を作成し、これをいくつかのパーツに分割します。各パーツは、SSS も再帰的に含む別のピンを使用して暗号化されます。また、しきい値 **t** も定義します。NBDE デプロイメントで少なくとも **t** の部分を復号すると、暗号化鍵が復元され、復号プロセスが成功します。Clevis がしきい値で指定されている数よりも小さい部分を検出すると、エラーメッセージが出力されます。

例 1: 2 台の Tang サーバーを使用した冗長性

次のコマンドは、2 台の Tang サーバーのうち少なくとも 1 台が使用可能な場合に、LUKS で暗号化されたデバイスを復号します。

```
# clevis luks bind -d /dev/sda1 sss '{"t":1,"pins":{"tang":{"url":"http://tang1.srv"},
{"url":"http://tang2.srv"}}}'
```

上記のコマンドでは、以下の設定スキームを使用していました。

```
{
  "t":1,
  "pins":{
    "tang":[
      {
        "url":"http://tang1.srv"
      },
      {
        "url":"http://tang2.srv"
      }
    ]
  }
}
```

この設定では、リストに記載されている 2 台の **tang** サーバーのうち少なくとも 1 つが利用可能であれば、SSS しきい値 **t** が **1** に設定され、**clevis luks bind** コマンドが秘密を正常に再構築します。

例 2: Tang サーバーと TPM デバイスで共有している秘密

次のコマンドは、**tang** サーバーと **tpm2** デバイスの両方が利用可能な場合に、LUKS で暗号化したデバイスを正常に復号します。

```
# clevis luks bind -d /dev/sda1 sss '{"t":2,"pins":{"tang":{"url":"http://tang1.srv"}}, "tpm2":
{"pcr_ids":"0,7"}'}
```

SSS しきい値 't' が '2' に設定されている設定スキームは以下のようになります。

```
{
  "t":2,
  "pins":{
    "tang":[
      {
        "url":"http://tang1.srv"
      }
    ],
    "tpm2":{
      "pcr_ids":"0,7"
    }
  }
}
```

関連情報

- システム上の **tang(8)** (High Availability セクション)、**clevis(1)** (Shamir's Secret Sharing セクション)、および **clevis-encrypt-sss(1)** man ページ

10.14. NBDE ネットワークで仮想マシンをデプロイする

clevis luks bind コマンドは、LUKS マスター鍵を変更しません。これは、仮想マシンまたはクラウド環境で使用する、LUKS で暗号化したイメージを作成する場合に、このイメージを実行するすべてのインスタンスがマスター鍵を共有することを意味します。これにはセキュリティの観点で大きな問題があるため、常に回避する必要があります。

これは、Clevis の制限ではなく、LUKS の設計原理です。シナリオでクラウド内のルートボリュームを暗号化する必要がある場合は、クラウド内の Red Hat Enterprise Linux の各インスタンスに対しても (通常はキックスタートを使用して) インストールプロセスを実行します。このイメージは、LUKS マスター鍵を共有しなければ共有できません。

仮想化環境で自動ロック解除をデプロイメントするには、**lorax** や **virt-install** などのシステムとキックスタートファイル ([キックスタートを使用した LUKS 暗号化ボリュームの自動登録の設定参照](#)) またはその他の自動プロビジョニングツールを使用して、各暗号化 VM に固有のマスターキーを確実に付与します。

関連情報

- システム上の **clevis-luks-bind(1)** man ページ

10.15. NBDE を使用してクラウド環境用の自動登録可能な仮想マシンイメージをビルドする

自動登録可能な暗号化イメージをクラウド環境にデプロイすると、特有の課題が発生する可能性があります。他の仮想化環境と同様に、LUKS マスター鍵を共有しないように、1つのイメージから起動するインスタンス数を減らすことが推奨されます。

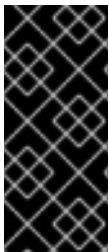
したがって、ベストプラクティスは、どのパブリックリポジトリでも共有されず、限られたインスタ

ンスのデプロイメントのベースを提供するように、イメージをカスタマイズすることです。作成するインスタンスの数は、デプロイメントのセキュリティポリシーで定義する必要があります。また、LUKS マスター鍵の攻撃ベクトルに関連するリスク許容度に基づいて決定する必要があります。

LUKS に対応する自動デプロイメントを構築するには、Lorax、virt-install などのシステムとキックスタートファイルを一緒に使用し、イメージ構築プロセス中にマスター鍵の一意性を確保する必要があります。

クラウド環境では、ここで検討する 2 つの Tang サーバーデプロイメントオプションが利用できます。まず、クラウド環境そのものに Tang サーバーをデプロイできます。もしくは、2 つのインフラストラクチャー間で VPN リンクを使用した独立したインフラストラクチャーで、クラウドの外に Tang サーバーをデプロイできます。

クラウドに Tang をネイティブにデプロイすると、簡単にデプロイできます。ただし、別のシステムの暗号文のデータ永続化層でインフラストラクチャーを共有します。Tang サーバーの秘密鍵および Clevis メタデータは、同じ物理ディスクに保存できる場合があります。この物理ディスクでは、暗号文データへのいかなる不正アクセスが可能になります。



重要

データを保存する場所と Tang を実行するシステムを、常に物理的に分離してください。クラウドと Tang サーバーを分離することで、Tang サーバーの秘密鍵が、Clevis メタデータと誤って結合することがないようにします。さらに、これにより、クラウドインフラストラクチャーが危険にさらされている場合に、Tang サーバーのローカル制御を提供します。

10.16. コンテナとしての TANG のデプロイ

tang コンテナイメージは、OpenShift Container Platform (OCP) クラスターまたは別の仮想マシンで実行する Clevis クライアントの Tang-server 復号化機能を提供します。

前提条件

- **podman** パッケージとその依存関係がシステムにインストールされている。
- **podman login registry.redhat.io** コマンドを使用して **registry.redhat.io** コンテナカタログにログインしている。詳細は、[Red Hat コンテナレジストリーの認証](#) を参照してください。
- Clevis クライアントは、Tang サーバーを使用して、自動的にアンロックする LUKS で暗号化したボリュームを含むシステムにインストールされている。

手順

1. **registry.redhat.io** レジストリーから **tang** コンテナイメージをプルします。

```
# podman pull registry.redhat.io/rhel9/tang
```

2. コンテナを実行し、そのポートを指定して Tang 鍵へのパスを指定します。上記の例では、**tang** コンテナを実行し、ポート 7500 を指定し、**/var/db/tang** ディレクトリーの Tang 鍵へのパスを示します。

```
# podman run -d -p 7500:7500 -v tang-keys:/var/db/tang --name tang
registry.redhat.io/rhel9/tang
```

Tang はデフォルトでポート 80 を使用しますが、Apache HTTP サーバーなどの他のサービスと共存する可能性があることに注意してください。

- オプション: セキュリティーを強化する場合は、Tang 鍵を定期的にローテーションします。 **tangd-rotate-keys** スクリプトを使用できます。以下に例を示します。

```
# podman run --rm -v tang-keys:/var/db/tang registry.redhat.io/rhel9/tang tangd-rotate-keys -
v -d /var/db/tang
Rotated key 'rZAMKAseaXBe0rcKXL1hCClq-DY.jwk' -> '.rZAMKAseaXBe0rcKXL1hCClq-
DY.jwk'
Rotated key 'x1Alpc6WmnCU-CabD8_4q18vDuw.jwk' -> '.x1Alpc6WmnCU-
CabD8_4q18vDuw.jwk'
Created new key GrMMX_WfdqomIU_4RyjpcdlXb0E.jwk
Created new key _dTTFn17sZZqVAp80u3ygFDHtjk.jwk
Keys rotated successfully.
```

検証

- Tang サーバーが存在しているために自動アンロック用に LUKS で暗号化したボリュームが含まれているシステムで、Clevis クライアントが Tang を使用してプレーンテキストのメッセージを暗号化および復号化できることを確認します。

```
# echo test | clevis encrypt tang '{"url":"http://localhost:7500"}' | clevis decrypt
The advertisement contains the following signing keys:

x1Alpc6WmnCU-CabD8_4q18vDuw

Do you wish to trust these keys? [ynYN] y
test
```

上記のコマンド例は、**localhost** URL で Tang サーバーが利用できる場合にその出力の最後に **テスト** 文字列を示し、ポート **7500** 経由で通信します。

関連情報

- システム上の **podman(1)**、**clevis(1)**、および **tang(8)** man ページ

10.17. RHEL システムロールを使用した NBDE の設定

nbde_client および **nbde_server** RHEL システムロールを使用すると、Clevis および Tang を使用した Policy-Based Decryption (PBD) ソリューションを自動でデプロイできます。**rhel-system-roles** パッケージには、これらのシステムロール、関連する例、リファレンスドキュメントが含まれます。

10.17.1. 複数の Tang サーバーのセットアップに **nbde_server** RHEL システムロールを使用する

nbde_server システムロールを使用すると、自動ディスク暗号化ソリューションの一部として、Tang サーバーをデプロイして管理できます。このロールは以下の機能をサポートします。

- Tang 鍵のローテーション
- Tang 鍵のデプロイおよびバックアップ

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: Deploy a Tang server
  hosts: tang.server.example.com
  tasks:
    - name: Install and configure periodic key rotation
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.nbde_server
  vars:
    nbde_server_rotate_keys: yes
    nbde_server_manage_firewall: true
    nbde_server_manage_selinux: true
```

このサンプル Playbook により、Tang サーバーのデプロイと鍵のローテーションが実行されます。

サンプル Playbook で指定されている設定は次のとおりです。

nbde_server_manage_firewall: true

firewall システムロールを使用して、**nbde_server** ロールで使用されるポートを管理します。

nbde_server_manage_selinux: true

selinux システムロールを使用して、**nbde_server** ロールで使用されるポートを管理します。

Playbook で使用されるすべての変数の詳細は、コントロールノードの

`/usr/share/ansible/roles/rhel-system-roles.nbde_server/README.md` ファイルを参照してください。

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

- NBDE クライアントで、次のコマンドを使用して、Tang サーバーが正しく動作していることを確認します。このコマンドにより、暗号化と復号化に渡すものと同じメッセージが返される必要があります。

```
# ansible managed-node-01.example.com -m command -a 'echo test | clevis encrypt tang
{"url":"<tang.server.example.com>"}' -y | clevis decrypt
test
```

関連情報

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

10.17.2. nbde_client RHEL システムロールを使用して DHCP を使用する Clevis クライアントを設定する

nbde_client システムロールにより、複数の Clevis クライアントを自動的にデプロイできます。

このロールを使用すると、LUKS で暗号化されたボリュームを 1 つ以上の Network-Bound (NBDE) サーバー (Tang サーバー) にバインドすることが可能です。パスフレーズを使用して既存のボリュームの暗号化を保持するか、削除できます。パスフレーズを削除したら、NBDE だけを使用してボリュームのロックを解除できます。これは、システムのプロビジョニング後に削除する必要がある一時鍵またはパスワードを使用して、ボリュームが最初に暗号化されている場合に役立ちます。

パスフレーズと鍵ファイルの両方を指定する場合には、ロールは最初に指定した内容を使用します。有効なバインディングが見つからない場合は、既存のバインディングからパスフレーズの取得を試みます。

Policy-Based Decryption (PBD) では、デバイスとスロットのマッピングの形でバインディングを定義します。そのため、同じデバイスに対して複数のバインドを設定できます。デフォルトのスロットは 1 です。



注記

nbde_client システムロールは、Tang バインディングのみをサポートします。したがって、TPM2 バインディングには使用できません。

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。
- LUKS を使用してすでに暗号化されているボリューム。

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: Configure clients for unlocking of encrypted volumes by Tang servers
  hosts: managed-node-01.example.com
  tasks:
    - name: Create NBDE client bindings
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.nbde_client
      vars:
        nbde_client_bindings:
          - device: /dev/rhel/root
            encryption_key_src: /etc/luks/keyfile
            nbde_client_early_boot: true
            state: present
            servers:
              - http://server1.example.com
              - http://server2.example.com
          - device: /dev/rhel/swap
            encryption_key_src: /etc/luks/keyfile
            servers:
              - http://server1.example.com
              - http://server2.example.com

```

このサンプル Playbook は、2 台の Tang サーバーのうち少なくとも 1 台が利用可能な場合に、LUKS で暗号化した 2 つのボリュームを自動的にアンロックするように Clevis クライアントを設定します。

サンプル Playbook で指定されている設定は次のとおりです。

state: present

state の値は、Playbook を実行した後の設定を示します。新しいバインディングを作成するか、既存のバインディングを更新する場合は、**present** を使用します。**clevis luks bind** とは異なり、**state: present** を使用してデバイススロットにある既存のバインディングを上書きすることもできます。**absent** に設定すると、指定したバインディングが削除されます。

nbde_client_early_boot: true

nbde_client ロールは、デフォルトで、起動初期段階で Tang ピン用のネットワークを利用可能にします。この機能を無効にする必要がある場合は、Playbook に **nbde_client_early_boot: false** 変数を追加します。

Playbook で使用されるすべての変数の詳細は、コントロールノードの **/usr/share/ansible/roles/rhel-system-roles.nbde_client/README.md** ファイルを参照してください。

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. NBDE クライアントで、Tang サーバーによって自動的にロック解除される暗号化ボリュームの LUKS ピンに、対応する情報が含まれていることを確認します。

```
# ansible managed-node-01.example.com -m command -a 'clevis luks list -d /dev/rhel/root'
1: tang '{"url":"<http://server1.example.com/>"}'
2: tang '{"url":"<http://server2.example.com/>"}'
```

2. **nbde_client_early_boot: false** 変数を使用しない場合は、起動初期にバインディングが使用できることを確認します。次に例を示します。

```
# ansible managed-node-01.example.com -m command -a 'lsinitrd | grep clevis-luks'
lrwxrwxrwx 1 root root 48 Jan 4 02:56
etc/systemd/system/cryptsetup.target.wants/clevis-luks-askpass.path ->
/usr/lib/systemd/system/clevis-luks-askpass.path
...
```

関連情報

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

10.17.3. nbde_client RHEL システムロールを使用して静的 IP Clevis クライアントを設定する

nbde_client RHEL システムロールは、Dynamic Host Configuration Protocol (DHCP) を使用する環境にのみ対応しています。静的 IP 設定の NBDE クライアントでは、ネットワーク設定をカーネルブートパラメーターとして渡す必要があります。

通常、管理者は Playbook を再利用します。Ansible が起動初期段階で静的 IP アドレスを割り当てるホストごとに、個別の Playbook を管理することはありません。そうすることにより、Playbook 内の変数を使用し、外部ファイルで設定を提供できます。そのため、必要なのは Playbook 1 つと設定を含むファイル 1 つだけです。

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。
- LUKS を使用してすでに暗号化されているボリューム。

手順

1. ホストのネットワーク設定を含むファイル (例: **static-ip-settings-clients.yml**) を作成し、ホストに動的に割り当てる値を追加します。

```
clients:
  managed-node-01.example.com:
    ip_v4: 192.0.2.1
    gateway_v4: 192.0.2.254
```

```

netmask_v4: 255.255.255.0
interface: enp1s0
managed-node-02.example.com:
ip_v4: 192.0.2.2
gateway_v4: 192.0.2.254
netmask_v4: 255.255.255.0
interface: enp1s0

```

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

```

- name: Configure clients for unlocking of encrypted volumes by Tang servers
  hosts: managed-node-01.example.com,managed-node-02.example.com
  vars_files:
    - ~/static-ip-settings-clients.yml
  tasks:
    - name: Create NBDE client bindings
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.network
      vars:
        nbde_client_bindings:
          - device: /dev/rhel/root
            encryption_key_src: /etc/luks/keyfile
            servers:
              - http://server1.example.com
              - http://server2.example.com
          - device: /dev/rhel/swap
            encryption_key_src: /etc/luks/keyfile
            servers:
              - http://server1.example.com
              - http://server2.example.com

    - name: Configure a Clevis client with static IP address during early boot
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.bootloader
      vars:
        bootloader_settings:
          - kernel: ALL
            options:
              - name: ip
                value: "{{ clients[inventory_hostname]['ip_v4'] }}::{{ clients[inventory_hostname]
['gateway_v4'] }}::{{ clients[inventory_hostname]['netmask_v4'] }}::{{
clients[inventory_hostname]['interface'] }}:none"

```

この Playbook は、`~/static-ip-settings-clients.yml` ファイルにリストされている各ホストの特定の値を動的に読み取ります。

Playbook で使用されるすべての変数の詳細は、コントロールノードの `/usr/share/ansible/roles/rhel-system-roles.network/README.md` ファイルを参照してください。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.nbde_client/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/nbde_client/** ディレクトリー
- [Looking forward to Linux network configuration in the initial ramdisk \(initrd\)](#) (Red Hat Enable Sysadmin)

第11章 システムの監査

監査はシステムのセキュリティを強化するものではありませんが、システム上のセキュリティポリシー違反を検出するために使用できます。検出後、SELinux などの追加のセキュリティ対策を設定することで、同じ違反の再発を防ぐことができます。

11.1. LINUX の AUDIT

Linux の Audit システムは、システムのセキュリティ関連情報を追跡する方法を提供します。事前設定されたルールに基づき、Audit は、ログエントリを生成し、システムで発生しているイベントに関する情報をできるだけ多く記録します。この情報は、ミッションクリティカルな環境でセキュリティポリシーの違反者と、違反者によるアクションを判断する上で必須のものです。

以下は、Audit がログファイルに記録できる情報の概要です。

- イベントの日時、タイプ、結果
- サブジェクトとオブジェクトの機密性のラベル
- イベントを開始したユーザーの ID とイベントの関連性
- Audit 設定の全修正および Audit ログファイルへのアクセス試行
- SSH、Kerberos などの認証メカニズムのすべての使用
- 信頼できるデータベース (`/etc/passwd` など) への変更
- システムからの情報のインポート、およびシステムへの情報のエクスポートの試行
- ユーザー ID、サブジェクトとオブジェクトのラベルなどの属性に基づくイベントの追加と除外

Audit システムの使用は、多くのセキュリティ関連の認定における要件でもあります。Audit は、以下の認定またはコンプライアンスガイドの要件に合致するか、それを超えるように設計されています。

- Controlled Access Protection Profile (CAPP)
- Labeled Security Protection Profile (LSPP)
- Rule Set Base Access Control (RSBAC)
- NISPOM (National Industrial Security Program Operating Manual)
- Federal Information Security Management Act (FISMA)
- Payment Card Industry - Data Security Standard (PCI-DSS)
- Security Technical Implementation Guides (STIG)

監査は、National Information Assurance Partnership (NIAP) および Best Security Industries (BSI) によっても評価されています。

ユースケース

ファイルアクセスの監視

Audit は、ファイルやディレクトリーがアクセス、修正、または実行されたか、もしくはファイル属性が変更されたかを追跡できます。これはたとえば、重要なファイルへのアクセスを検出し、これらのファイルが破損した場合に監査証跡を入手可能とする際に役に立ちます。

システムコールの監視

Audit は、一部のシステムコールが使用されるたびにログエントリーを生成するように設定できます。これを使用すると、**settimeofday** や **clock_adjtime**、その他の時間関連のシステムコールを監視することで、システム時間への変更を追跡できます。

ユーザーが実行したコマンドの記録

Audit はファイルが実行されたかどうかを追跡できるため、特定のコマンドの実行を毎回記録するようにルールを定義できます。たとえば、**/bin** ディレクトリー内のすべての実行可能ファイルにルールを定義できます。これにより作成されるログエントリーをユーザー ID で検索すると、ユーザーごとに実行されたコマンドの監査証跡を生成できます。

システムのパス名の実行の記録

ルールの呼び出し時にパスを inode に変換するファイルアクセスをウォッチする以外に、ルールの呼び出し時に存在しない場合や、ルールの呼び出し後にファイルが置き換えられた場合でも、Audit がパスの実行をウォッチできるようになりました。これにより、ルールは、プログラム実行ファイルをアップグレードした後、またはインストールされる前にも機能を継続できます。

セキュリティーイベントの記録

pam_faillock 認証モジュールは、失敗したログイン試行を記録できます。Audit で失敗したログイン試行も記録するように設定すると、ログインを試みたユーザーに関する追加情報が提供されます。

イベントの検索

Audit は **ausearch** ユーティリティーを提供します。これを使用すると、ログエントリーをフィルターにかけ、いくつかの条件に基づく完全な監査証跡を提供できます。

サマリーレポートの実行

aureport ユーティリティーを使用すると、記録されたイベントのデイリーレポートを生成できます。システム管理者は、このレポートを分析し、疑わしいアクティビティーをさらに調べることができます。

ネットワークアクセスの監視

nftables、**iptables**、および **ebtables** ユーティリティーは、Audit イベントを発生するように設定できるため、システム管理者がネットワークアクセスを監視できるようになります。



注記

システムのパフォーマンスは、Audit が収集する情報量によって影響される可能性があります。

11.2. AUDIT システムのアーキテクチャー

Audit システムは、ユーザー空間アプリケーションおよびユーティリティーと、カーネル側のシステムコール処理という 2 つの主要部分で構成されます。カーネルコンポーネントは、ユーザー空間アプリケーションからシステムコールを受け、これを **user**、**task**、**fstype**、または **exit** のいずれかのフィルターで振り分けます。

システムコールが **exclude** フィルターを通過すると、前述のフィルターのいずれかに送られます。このフィルターにより、Audit ルール設定に基づいてシステムコールが Audit デーモンに送信され、さらに処理されます。

ユーザー空間の Audit デーモンは、カーネルから情報を収集し、ログファイルのエントリーを作成します。他のユーザー空間ユーティリティーは、Audit デーモン、カーネルの Audit コンポーネント、または Audit ログファイルと相互作用します。

- **auditctl** Audit 制御ユーティリティーはカーネル Audit コンポーネントと相互作用し、ルールを管理するだけでなくイベント生成プロセスの多くの設定やパラメーターも制御します。
- 残りの Audit ユーティリティーは、Audit ログファイルのコンテンツを入力として受け取り、ユーザーの要件に基づいて出力を生成します。たとえば、**aureport** ユーティリティーは、記録された全イベントのレポートを生成します。

RHEL 9 では、Audit dispatcher デーモン (**audisp**) 機能は、Audit デーモン (**auditd**) に統合されています。監査イベントと、リアルタイムの分析プログラムの相互作用に使用されるプラグイン設定ファイルは、デフォルトで **/etc/audit/plugins.d/** ディレクトリーに保存されます。

11.3. 環境を保護するための AUDITD の設定

デフォルトの **auditd** 設定は、ほとんどの環境に適しています。ただし、環境が厳格なセキュリティポリシーを満たす必要がある場合は、**/etc/audit/auditd.conf** ファイル内の Audit デーモン設定の次の設定を変更できます。

log_file

Audit ログファイル (通常は **/var/log/audit/**) を保持するディレクトリーは、別のマウントポイントにマウントされている必要があります。これにより、その他のプロセスがこのディレクトリー内の領域を使用しないようにし、Audit デーモンの残りの領域を正確に検出します。

max_log_file

1つの Audit ログファイルの最大サイズを指定します。Audit ログファイルを保持するパーティションで利用可能な領域をすべて使用するように設定する必要があります。**max_log_file** パラメーターは、最大ファイルサイズをメガバイト単位で指定します。指定する値は、数値にする必要があります。

max_log_file_action

max_log_file に設定した制限に達すると実行するアクションを指定します。Audit ログファイルが上書きされないように **keep_logs** に設定する必要があります。

space_left

space_left_action パラメーターに設定したアクションが発生するディスクの空き容量を指定します。管理者は、ディスクの領域を反映して解放するのに十分な時間を設定する必要があります。**space_left** の値は、Audit ログファイルが生成される速度によって異なります。**space_left** の値が整数として指定されている場合は、メガバイト (MiB) 単位の絶対サイズとして解釈されます。値が 1～99 の数値の後にパーセント記号を付けて指定されている場合 (5% など)、Audit デーモンは、**log_file** を含むファイルシステムのサイズに基づいて、メガバイト単位で絶対サイズを計算します。

space_left_action

space_left_action パラメーターは、適切な通知方法 (**email** または **exec**) に設定することが推奨されます。

admin_space_left

admin_space_left_action パラメーターに設定したアクションが発生するディスクの空き領域の最小サイズ。管理者が実行するアクションのログを記録するのに十分なサイズを残す必要があります。このパラメーターの数値は、**space_left** の数値より小さくする必要があります。また、数値にパーセント記号を追加 (1% など) して、Audit デーモンが、ディスクパーティションサイズに基づいて、数値を計算するようにすることもできます。

admin_space_left_action

single を、システムをシングルユーザーモードにし、管理者がディスク領域を解放できるようにします。

disk_full_action

Audit ログファイルが含まれるパーティションに空き領域がない場合に発生するアクションを指定します (**halt** または **single** に設定する必要があります)。これにより、Audit がイベントをログに記録できなくなると、システムは、シングルユーザーモードでシャットダウンまたは動作します。

disk_error_action

Audit ログファイルが含まれるパーティションでエラーが検出された場合に発生するアクションを指定します。このパラメーターは、ハードウェアの機能不全処理に関するローカルのセキュリティポリシーに基づいて、**syslog**、**single**、**halt** のいずれかに設定する必要があります。

flush

incremental_async に設定する必要があります。これは **freq** パラメーターと組み合わせて機能します。これは、ハードドライブとのハード同期を強制する前にディスクに送信できるレコードの数を指定します。**freq** パラメーターは **100** に設定する必要があります。このパラメーターにより、アクティビティーが集中した際に高いパフォーマンスを保ちつつ、Audit イベントデータがディスクのログファイルと確実に同期されるようになります。

残りの設定オプションは、ローカルのセキュリティポリシーに合わせて設定します。

11.4. AUDITD の開始および制御

auditd が設定されたら、サービスを起動して Audit 情報を収集し、ログファイルに保存します。root ユーザーで次のコマンドを実行し、**auditd** を起動します。

```
# service auditd start
```

システムの起動時に **auditd** が起動するように設定するには、次のコマンドを実行します。

```
# systemctl enable auditd
```

auditctl -e 0 で **auditd** を一時的に無効にし、# **auditctl -e 1** で再度有効にできます。

service auditd <action> コマンドを使用すると、**auditd** で他のアクションを実行できます。<action> は次のいずれかです。

stop

auditd を停止します。

restart

auditd を再起動します。

reload または force-reload

/etc/audit/auditd.conf ファイルから **auditd** の設定を再ロードします。

rotate

/var/log/audit/ ディレクトリーのログファイルをローテーションします。

resume

Audit イベントのログが一旦停止した後、再開します。たとえば、Audit ログファイルが含まれるディスクパーティションの未使用領域が不足している場合などです。

condrestart または try-restart

auditd がすでに起動している場合にのみ、これを再起動します。

status

auditd の稼働状況を表示します。

**注記**

service コマンドは、**auditd** デーモンと正しく相互作用する唯一の方法です。**audit** 値が適切に記録されるように、**service** コマンドを使用する必要があります。**systemctl** コマンドは、2つのアクション (**enable** および **status**) にのみ使用できます。

11.5. AUDIT ログファイルについて

デフォルトでは、Audit システムはログエントリを **/var/log/audit/audit.log** ファイルに保存します。ログローテーションが有効になっていれば、ローテーションされた **audit.log** ファイルは同じディレクトリに保存されます。

下記の Audit ルールを追加して、**/etc/ssh/sshd_config** ファイルの読み取りまたは修正の試行をすべてログに記録します。

```
# auditctl -w /etc/ssh/sshd_config -p warx -k sshd_config
```

auditd デーモンが実行している場合は、たとえば次のコマンドを使用して、Audit ログファイルに新しいイベントを作成します。

```
$ cat /etc/ssh/sshd_config
```

このイベントは、**audit.log** ファイルでは以下のようになります。

```
type=SYSCALL msg=audit(1364481363.243:24287): arch=c000003e syscall=2 success=no exit=-13
a0=7fffd19c5592 a1=0 a2=7fffd19c4b50 a3=a items=1 ppid=2686 pid=3538 audit=1000 uid=1000
gid=1000 euid=1000 suid=1000 fsuid=1000 egid=1000 sgid=1000 fsgid=1000 tty=pts0 ses=1
comm="cat" exe="/bin/cat" subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
key="sshd_config"
type=CWD msg=audit(1364481363.243:24287): cwd="/home/shadowman"
type=PATH msg=audit(1364481363.243:24287): item=0 name="/etc/ssh/sshd_config" inode=409248
dev=fd:00 mode=0100600 ouid=0 ogid=0 rdev=00:00 obj=system_u:object_r:etc_t:s0
nametype=NORMAL cap_fp=none cap_fi=none cap_fe=0 cap_fver=0
type=PROCTITLE msg=audit(1364481363.243:24287) :
proctitle=636174002F6574632F7373682F737368645F636F6E666967
```

上記のイベントは4つのレコードで構成されており、タイムスタンプとシリアル番号を共有します。レコードは、常に **type=** で始まります。各レコードは、スペースまたはコンマで区切られた複数の名前と値のペア (**name=value**) で構成されます。上記のイベントの詳細な分析は以下のようになります。

1つ目のレコード**type=SYSCALL**

type フィールドには、レコードのタイプが記載されます。この例の **SYSCALL** 値は、カーネルへのシステムコールによりこれが記録されたことを示しています。

msg=audit(1364481363.243:24287):

msg フィールドには以下が記録されます。

- **audit(time_stamp:ID)** 形式のレコードのタイムスタンプおよび一意の ID。複数のレコードが同じ Audit イベントの一部として生成されている場合は、同じタイムスタンプおよび ID を共有できます。タイムスタンプは Unix の時間形式です (1970 年 1 月 1 日 00:00:00 UTC からの秒数)。
- カーネル空間およびユーザー空間のアプリケーションが提供するさまざまなイベント固有の **name=value** ペア。

arch=c000003e

arch フィールドには、システムの CPU アーキテクチャーに関する情報が含まれます。**c000003e** の値は 16 進数表記で記録されます。**ausearch** コマンドで Audit レコードを検索する場合は、**-i** オプションまたは **--interpret** オプションを使用して、16 進数の値を人間が判読できる値に自動的に変換します。**c000003e** 値は **x86_64** として解釈されます。

syscall=2

syscall フィールドは、カーネルに送信されたシステムコールのタイプを記録します。値が **2** の場合は、**/usr/include/asm/unistd_64.h** ファイルに、人間が判読できる値を指定できます。この場合の **2** は、オープン なシステムコールです。**ausyscall** ユーティリティーでは、システムコール番号を、人間が判読できる値に変換できます。**ausyscall --dump** コマンドを使用して、システムコールのリストとその数字を表示します。詳細は、**ausyscall(8)** の man ページを参照してください。

success=no

success フィールドは、その特定のイベントで記録されたシステムコールが成功したかどうかを記録します。この例では、呼び出しが成功しませんでした。

exit=-13

exit フィールドには、システムコールが返した終了コードを指定する値が含まれます。この値は、システムコールにより異なります。次のコマンドを実行すると、この値を人間が判読可能なものに変換できます。

```
# ausearch --interpret --exit -13
```

この例では、監査ログに、終了コード **-13** で失敗したイベントが含まれていることが前提となります。

a0=7fffd19c5592, a1=0, a2=7fffd19c5592, a3=a

a0 から **a3** までのフィールドは、このイベントにおけるシステムコールの最初の 4 つの引数を、16 進法で記録します。この引数は、使用されるシステムコールにより異なります。**ausearch** ユーティリティーで解釈できます。

items=1

items フィールドには、システムコールのレコードに続く PATH 補助レコードの数が含まれます。

ppid=2686

ppid フィールドは、親プロセス ID (PPID) を記録します。この例では、**2686** は、**bash** などの親プロセスの PPID です。

pid=3538

pid フィールドは、プロセス ID (PID) を記録します。この例の **3538** は **cat** プロセスの PID です。

audit=1000

audit フィールドには、loginuid である Audit ユーザー ID が記録されます。この ID は、ログイン時にユーザーに割り当てられ、ユーザーの ID が変更した後もすべてのプロセスに引き継がれます（たとえば、**su - john** コマンドでユーザーアカウントを切り替えた場合）。

uid=1000

uid フィールドは、解析しているプロセスを開始したユーザーのユーザー ID を記録します。ユーザー ID は、**ausearch -i --uid UID** のコマンドを使用するとユーザー名に変換されます。

gid=1000

gid フィールドは、解析しているプロセスを開始したユーザーのグループ ID を記録します。

euid=1000

euid フィールドは、解析しているプロセスを開始したユーザーの実効ユーザー ID を記録します。

suid=1000

suid フィールドは、解析しているプロセスを開始したユーザーのセットユーザー ID を記録します。

fsuid=1000

fsuid フィールドは、解析しているプロセスを開始したユーザーのファイルシステムユーザー ID を記録します。

egid=1000

egid フィールドは、解析しているプロセスを開始したユーザーの実効グループ ID を記録します。

sgid=1000

sgid フィールドは、解析しているプロセスを開始したユーザーのセットグループ ID を記録します。

fsgid=1000

fsgid フィールドは、解析しているプロセスを開始したユーザーのファイルシステムグループ ID を記録します。

tty=pts0

tty フィールドは、解析しているプロセスが開始したターミナルを記録します。

ses=1

ses フィールドは、解析しているプロセスが開始したセッションのセッション ID を記録します。

comm="cat"

comm フィールドは、解析しているプロセスを開始するために使用したコマンドのコマンドライン名を記録します。この例では、この Audit イベントが発生するのに、**cat** コマンドが使用されました。

exe="/bin/cat"

exe フィールドは、解析しているプロセスを開始するために使用した実行可能ファイルへのパスを記録します。

subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023

subj フィールドは、解析しているプロセスの実行時にラベル付けされた SELinux コンテンツを記録します。

key="sshd_config"

key フィールドは、Audit ログでこのイベントを生成したルールに関連付けられている管理者による定義の文字列を記録します。

2 つ目のレコード**type=CWD**

2 つ目のレコードの **type** フィールドの値は、**CWD** (現在の作業ディレクトリー) です。このタイプは、最初のレコードで指定されたシステムコールを開始したプロセスの作業ディレクトリーを記録するために使用されます。

この記録の目的は、相対パスが関連する PATH 記録に保存された場合に、現行プロセスの位置を記録することにあります。これにより、絶対パスを再構築できます。

msg=audit(1364481363.243:24287)

msg フィールドは、最初のレコードと同じタイムスタンプと ID の値を保持します。タイムスタンプは Unix の時間形式です (1970 年 1 月 1 日 00:00:00 UTC からの秒数)。

cwd="/home/user_name"

cwd フィールドは、システムコールが開始したディレクトリーのパスになります。

3 つ目のレコード

type=PATH

3 つ目のレコードでは、**type** フィールドの値は **PATH** です。Audit イベントには、システムコールに引数として渡されたすべてのパスに **PATH** タイプのレコードが含まれます。この Audit イベントでは、1 つのパス (**/etc/ssh/sshd_config**) のみが引数として使用されます。

msg=audit(1364481363.243:24287):

msg フィールドは、1 つ目と 2 つ目のレコードと同じタイムスタンプと ID になります。

item=0

item フィールドは、**SYSCALL** タイプレコードで参照されているアイテムの合計数のうち、現在のレコードがどのアイテムであるかを示します。この数はゼロベースで、**0** は最初の項目であることを示します。

name="/etc/ssh/sshd_config"

name フィールドは、システムコールに引数として渡されたファイルまたはディレクトリーのパスを記録します。この場合、これは **/etc/ssh/sshd_config** ファイルです。

inode=409248

inode フィールドには、このイベントで記録されたファイルまたはディレクトリーに関連する inode 番号が含まれます。以下のコマンドは、inode 番号 **409248** に関連するファイルまたはディレクトリーを表示します。

```
# find / -inum 409248 -print
/etc/ssh/sshd_config
```

dev=fd:00

dev フィールドは、このイベントで記録されたファイルまたはディレクトリーを含むデバイスのマイナーおよびメジャーの ID を指定します。ここでは、値が **/dev/fd/0** デバイスを示しています。

mode=0100600

mode フィールドは、ファイルまたはディレクトリーのパーミッションを、**st_mode** フィールドの **stat** コマンドが返す数字表記で記録します。詳細は、**stat(2)** の man ページを参照してください。この場合、**0100600** は **-rw-----** として解釈されます。つまり、root ユーザーにのみ、**/etc/ssh/sshd_config** ファイルに読み取りおよび書き込みのパーミッションが付与されます。

oid=0

oid フィールドは、オブジェクトの所有者のユーザー ID を記録します。

ogid=0

ogid フィールドは、オブジェクトの所有者のグループ ID を記録します。

rdev=00:00

rdev フィールドには、特定ファイルにのみ記録されたデバイス識別子が含まれます。ここでは、記録されたファイルは通常のファイルであるため、このフィールドは使用されません。

obj=system_u:object_r:etc_t:s0

obj フィールドは、実行時に、記録されているファイルまたはディレクトリーにラベル付けする SELinux コンテキストを記録します。

nametype=NORMAL

nametype フィールドは、指定したシステムコールのコンテキストで各パスのレコード操作の目的を記録します。

cap_fp=none

cap_fp フィールドは、ファイルまたはディレクトリーオブジェクトで許可されたファイルシステムベースの機能の設定に関連するデータを記録します。

cap_fi=none

cap_fi フィールドは、ファイルまたはディレクトリーオブジェクトの継承されたファイルシステムベースの機能の設定に関するデータを記録します。

cap_fe=0

cap_fe フィールドは、ファイルまたはディレクトリーオブジェクトのファイルシステムベースの機能の有効ビットの設定を記録します。

cap_fver=0

cap_fver フィールドは、ファイルまたはディレクトリーオブジェクトのファイルシステムベースの機能のバージョンを記録します。

4 つ目のレコード

type=PROCTITLE

type フィールドには、レコードのタイプが記載されます。この例の **PROCTITLE** 値は、このレコードにより、カーネルへのシステムコールにより発生するこの監査イベントを発生させた完全なコマンドラインを提供することが指定されることを示しています。

proctitle=636174002F6574632F7373682F737368645F636F6E666967

proctitle フィールドは、解析しているプロセスを開始するために使用したコマンドのコマンドラインを記録します。このフィールドは 16 進数の表記で記録され、Audit ログパーサーに影響が及ばないようにします。このテキストは、この Audit イベントを開始したコマンドに復号します。**ausearch** コマンドで Audit レコードを検索する場合は、**-i** オプションまたは **--interpret** オプションを使用して、16 進数の値を人間が判読できる値に自動的に変換します。**636174002F6574632F7373682F737368645F636F6E666967** 値は、**cat /etc/ssh/sshd_config** として解釈されます。

11.6. AUDITCTL で AUDIT ルールを定義および実行

Audit システムは、ログファイルで取得するものを定義する一連のルールで動作します。Audit ルールは、**auditctl** ユーティリティを使用してコマンドラインで設定するか、**/etc/audit/rules.d/** ディレクトリーで設定できます。

auditctl コマンドを使用すると、Audit システムの基本的な機能を制御し、どの Audit イベントをログに記録するかを指定するルールを定義できます。

ファイルシステムのルールの例

1. すべての書き込みアクセスと **/etc/passwd** ファイルのすべての属性変更をログに記録するルールを定義するには、次のコマンドを実行します。

```
# auditctl -w /etc/passwd -p wa -k passwd_changes
```

2. すべての書き込みアクセスと、**/etc/selinux/** ディレクトリー内の全ファイルへのアクセスと、その属性変更をすべてログに記録するルールを定義するには、次のコマンドを実行します。

```
# auditctl -w /etc/selinux/ -p wa -k selinux_changes
```

システムロールのルールの例

1. システムで 64 ビットアーキテクチャーが使用され、システムコールの **adjtimex** または **settimeofday** がプログラムにより使用されるたびにログエントリーを作成するルールを定義するには、次のコマンドを実行します。

```
# auditctl -a always,exit -F arch=b64 -S adjtimex -S settimeofday -k time_change
```

2. ユーザー ID が 1000 以上のシステムユーザーがファイルを削除したりファイル名を変更するたびに、ログエントリーを作成するルールを定義するには、次のコマンドを実行します。

```
# auditctl -a always,exit -S unlink -S unlinkat -S rename -S renameat -F auid>=1000 -F auid!=4294967295 -k delete
```

-F auid!=4294967295 オプションが、ログイン UID が設定されていないユーザーを除外するために使用されています。

実行可能なファイルルール

/bin/id プログラムのすべての実行をログに取得するルールを定義するには、次のコマンドを実行します。

```
# auditctl -a always,exit -F exe=/bin/id -F arch=b64 -S execve -k execution_bin_id
```

関連情報

- システム上の **auditctl(8)** man ページ

11.7. 永続的な AUDIT ルールの定義

再起動後も持続するように Audit ルールを定義するには、**/etc/audit/rules.d/audit.rules** ファイルに直接追加するか、**/etc/audit/rules.d/** ディレクトリーにあるルールを読み込む **augenrules** プログラムを使用する必要があります。

auditd サービスを開始すると、**/etc/audit/audit.rules** ファイルが生成されることに注意してください。**/etc/audit/rules.d/** のファイルは、同じ **auditctl** コマンドライン構文を使用してルールを指定します。ハッシュ記号 (#) に続く空の行とテキストは無視されます。

また、**auditctl** コマンドは、以下のように **-R** オプションを使用して指定したファイルからルールを読み込むのに使用することもできます。

■

```
# auditctl -R /usr/share/audit/sample-rules/30-stig.rules
```

11.8. 標準に準拠するための事前設定された監査ルールファイル

特定の認定標準 (OSPP、PCI DSS、STIG など) に準拠するように Audit を設定するには、**audit** パッケージとともにインストールされる事前設定済みルールファイルのセットを出発点として使用できます。サンプルルールは、**/usr/share/audit/sample-rules** ディレクトリーにあります。



警告

セキュリティ標準は動的であり、変更される可能性があるため、**sample-rules** ディレクトリー内の Audit サンプルルールは網羅的なものではなく、最新のものではありません。これらのルールは、Audit ルールがどのように構造化および記述されるかを示すためにのみ提供されています。これらは、最新のセキュリティ標準に即時に準拠することを保証するものではありません。特定のセキュリティガイドラインに従ってシステムを最新のセキュリティ標準に準拠させるには、[SCAP ベースのセキュリティコンプライアンスツール](#) を使用してください。

30-nispom.rules

「National Industrial Security Program Operating Manual」の「Information System Security」の章で指定されている要件を満たす監査ルール設定

30-ospp-v42*.rules

OSPP (Protection Profile for General Purpose Operating Systems) プロファイルバージョン 4.2 に定義されている要件を満たす監査ルール設定

30-pci-dss-v31.rules

PCI DSS (Payment Card Industry Data Security Standard) v3.1 に設定されている要件を満たす監査ルール設定

30-stig.rules

Security Technical Implementation Guides (STIG) で設定されている要件を満たす監査ルール設定

上記の設定ファイルを使用するには、**/etc/audit/rules.d/** ディレクトリーにコピーして、以下のように **augenrules --load** コマンドを使用します。

```
# cd /usr/share/audit/sample-rules/
# cp 10-base-config.rules 30-stig.rules 31-privileged.rules 99-finalize.rules /etc/audit/rules.d/
# augenrules --load
```

番号指定スキームを使用して監査ルールを順序付けできます。詳細は、**/usr/share/audit/sample-rules/README-rules** ファイルを参照してください。

関連情報

- システム上の **audit.rules(7)** man ページ

11.9. 永続ルールを定義する AUGENRULES の使用

augenrules スクリプトは、`/etc/audit/rules.d/` ディレクトリーにあるルールを読み込み、**audit.rules** ファイルにコンパイルします。このスクリプトは、自然なソート順序の特定の順番で、**.rules** で終わるすべてのファイル进行处理します。このディレクトリーのファイルは、以下の意味を持つグループに分類されます。

10

カーネルと `auditctl` の設定

20

一般的なルールと一致する可能性があるが、別の一致が必要なルール

30

主なルール

40

オプションのルール

50

サーバー固有のルール

70

システムのローカルルール

90

ファイナライズ (イミュータブル)

ルールは、すべてを一度に使用することは意図されていません。ルールは考慮すべきポリシーの一部であり、個々のファイルは `/etc/audit/rules.d/` にコピーされます。たとえば、STIG 設定でシステムを設定し、**10-base-config**、**30-stig**、**31-privileged**、**99-finalize** の各ルールをコピーします。

`/etc/audit/rules.d/` ディレクトリーにルールを置いたら、`--load` ディレクティブで **augenrules** スクリプトを実行することでそれを読み込みます。

```
# augenrules --load
/sbin/augenrules: No change
No rules
enabled 1
failure 1
pid 742
rate_limit 0
...
```

関連情報

- システム上の **audit.rules(8)** および **augenrules(8)** man ページ

11.10. AUGENRULES の無効化

augenrules ユーティリティーを無効にするには、以下の手順に従います。これにより、Audit が `/etc/audit/audit.rules` ファイルで定義されたルールを使用するように切り替えます。

手順

1. `/usr/lib/systemd/system/auditd.service` ファイルを `/etc/systemd/system/` ディレクトリーにコピーします。

—

```
# cp -f /usr/lib/systemd/system/auditd.service /etc/systemd/system/
```

2. 任意のテキストエディターで **/etc/systemd/system/auditd.service** ファイルを編集します。以下に例を示します。

```
# vi /etc/systemd/system/auditd.service
```

3. **augenrules** を含む行をコメントアウトし、**auditctl -R** コマンドを含む行のコメント設定を解除します。

```
#ExecStartPost=-/sbin/augenrules --load
ExecStartPost=-/sbin/auditctl -R /etc/audit/audit.rules
```

4. **systemd** デーモンを再読み込みして、**auditd.service** ファイルの変更を取得します。

```
# systemctl daemon-reload
```

5. **auditd** サービスを再起動します。

```
# service auditd restart
```

関連情報

- システム上の **augenrules(8)** および **audit.rules(8)** man ページ
- [Auditd service restart overrides changes made to /etc/audit/audit.rules](#) (Red Hat ナレッジベース)

11.11. ソフトウェアの更新を監視するための AUDIT の設定

事前設定されたルール **44-installers.rules** を使用して、ソフトウェアをインストールする次のユーティリティーを監視するように Audit を設定できます。

- **dnf** [2]
- **yum**
- **pip**
- **npm**
- **cpan**
- **gem**
- **luarocks**

rpm ユーティリティーを監視するには、**rpm-plugin-audit** パッケージをインストールします。その後、Audit は、パッケージをインストールまたは更新するときに **SOFTWARE_UPDATE** イベントを生成します。これらのイベントをリスト表示するには、コマンドラインで **ausearch -m SOFTWARE_UPDATE** と入力します。



注記

事前設定されたルールファイルは、**ppc64le** および **aarch64** アーキテクチャーを備えたシステムでは使用できません。

前提条件

- **auditd** が、[環境を保護するための auditd の設定](#) で提供される設定に従って定義されている。

手順

1. 事前設定されたルールファイル **44-installers.rules** を **/usr/share/audit/sample-rules/** ディレクトリーから **/etc/audit/rules.d/** ディレクトリーにコピーします。

```
# cp /usr/share/audit/sample-rules/44-installers.rules /etc/audit/rules.d/
```

2. 監査ルールを読み込みます。

```
# augenrules --load
```

検証

1. 読み込まれたルールをリスト表示します。

```
# auditctl -l
-p x-w /usr/bin/dnf-3 -k software-installer
-p x-w /usr/bin/yum -k software-installer
-p x-w /usr/bin/pip -k software-installer
-p x-w /usr/bin/npm -k software-installer
-p x-w /usr/bin/cpan -k software-installer
-p x-w /usr/bin/gem -k software-installer
-p x-w /usr/bin/luarocks -k software-installer
```

2. インストールを実行します。以下に例を示します

```
# dnf reinstall -y vim-enhanced
```

3. Audit ログで最近のインストールイベントを検索します。次に例を示します。

```
# ausearch -ts recent -k software-installer
_____
time->Thu Dec 16 10:33:46 2021
type=PROCTITLE msg=audit(1639668826.074:298):
proctitle=2F7573722F6C6962657865632F706C6174666F726D2D707974686F6E002F75737
22F62696E2F646E66007265696E7374616C6C002D790076696D2D656E68616E636564
type=PATH msg=audit(1639668826.074:298): item=2 name="/lib64/ld-linux-x86-64.so.2"
inode=10092 dev=fd:01 mode=0100755 ouid=0 ogid=0 rdev=00:00
obj=system_u:object_r:ld_so_t:s0 nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0
cap_fver=0 cap_frootid=0
type=PATH msg=audit(1639668826.074:298): item=1 name="/usr/libexec/platform-python"
inode=4618433 dev=fd:01 mode=0100755 ouid=0 ogid=0 rdev=00:00
obj=system_u:object_r:bin_t:s0 nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0
cap_fver=0 cap_frootid=0
type=PATH msg=audit(1639668826.074:298): item=0 name="/usr/bin/dnf" inode=6886099
```

```
dev=fd:01 mode=0100755 ouid=0 ogid=0 rdev=00:00 obj=system_u:object_r:rpm_exec_t:s0
nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0
type=CWD msg=audit(1639668826.074:298): cwd="/root"
type=EXECVE msg=audit(1639668826.074:298): argc=5 a0="/usr/libexec/platform-python"
a1="/usr/bin/dnf" a2="reinstall" a3="-y" a4="vim-enhanced"
type=SYSCALL msg=audit(1639668826.074:298): arch=c000003e syscall=59 success=yes
exit=0 a0=55c437f22b20 a1=55c437f2c9d0 a2=55c437f2aeb0 a3=8 items=3 ppid=5256
pid=5375 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=3
comm="dnf" exe="/usr/libexec/platform-python3.6"
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 key="software-installer"
```

11.12. AUDIT によるユーザーログイン時刻の監視

特定の時刻にログインしたユーザーを監視するために、特別な方法で Audit を設定する必要はありません。同じ情報を表示する異なる方法を提供する **ausearch** または **aureport** ツールを使用できます。

前提条件

- **auditd** が、[環境を保護するための auditd の設定](#) で提供される設定に従って定義されている。

手順

ユーザーのログイン時刻を表示するには、次のいずれかのコマンドを使用します。

- 監査ログで **USER_LOGIN** メッセージタイプを検索します。

```
# ausearch -m USER_LOGIN -ts '12/02/2020' '18:00:00' -sv no
time->Mon Nov 22 07:33:22 2021
type=USER_LOGIN msg=audit(1637584402.416:92): pid=1939 uid=0 auid=4294967295
ses=4294967295 subj=system_u:system_r:sshd_t:s0-s0:c0.c1023 msg='op=login acct="
(unknown)" exe="/usr/sbin/sshd" hostname=? addr=10.37.128.108 terminal=ssh res=failed'
```

- **-ts** オプションを使用して日付と時刻を指定できます。このオプションを使用しない場合、**ausearch** は今日の結果を提供し、時刻を省略すると、**ausearch** は午前 0 時からの結果を提供します。
- 成功したログイン試行を除外するには **-sv yes** オプションを、失敗したログイン試行を除外するには **-sv no** を、それぞれ使用することができます。
- **ausearch** コマンドの生の出力を **aulast** ユーティリティにパイプで渡します。このユーティリティは、**last** コマンドの出力と同様の形式で出力を表示します。以下に例を示します。

```
# ausearch --raw | aulast --stdin
root  ssh      10.37.128.108  Mon Nov 22 07:33 - 07:33 (00:00)
root  ssh      10.37.128.108  Mon Nov 22 07:33 - 07:33 (00:00)
root  ssh      10.22.16.106  Mon Nov 22 07:40 - 07:40 (00:00)
reboot system boot 4.18.0-348.6.el8 Mon Nov 22 07:33
```

- **--login -i** オプションを指定して **aureport** コマンドを使用し、ログインイベントのリストを表示します。

```
# aureport --login -i

Login Report
```

```
=====
# date time auid host term exe success event
=====
1. 11/16/2021 13:11:30 root 10.40.192.190 ssh /usr/sbin/sshd yes 6920
2. 11/16/2021 13:11:31 root 10.40.192.190 ssh /usr/sbin/sshd yes 6925
3. 11/16/2021 13:11:31 root 10.40.192.190 ssh /usr/sbin/sshd yes 6930
4. 11/16/2021 13:11:31 root 10.40.192.190 ssh /usr/sbin/sshd yes 6935
5. 11/16/2021 13:11:33 root 10.40.192.190 ssh /usr/sbin/sshd yes 6940
6. 11/16/2021 13:11:33 root 10.40.192.190 /dev/pts/0 /usr/sbin/sshd yes 6945
```

関連情報

- システム上の **ausearch(8)**、**aulast(8)**、**aureport(8)** man ページ

11.13. 関連情報

- [RHEL Audit System Reference](#) (Red Hat ナレッジベース)
- [Auditd execution options in a container](#) (Red Hat ナレッジベース)
- [Linux Audit Documentation Project のページ](#) (Github.com)
- **audit** パッケージによって提供される **/usr/share/doc/audit/** ディレクトリー内のドキュメント
- システム上の **auditd(8)**、**auditctl(8)**、**ausearch(8)**、**audit.rules(7)**、**audispd.conf(5)**、**audispd(8)**、**auditd.conf(5)**、**ausearch-expression(5)**、**aulast(8)**、**aulastlog(8)**、**aureport(8)**、**ausyscall(8)**、**autrace(8)**、および **auvirt(8)** man ページ

[2] **dnf** は RHEL ではシンボリックリンクであるため、**dnf** Audit ルールのパスにはシンボリックリンクのターゲットが含まれている必要があります。正しい Audit イベントを受信するには、**path=/usr/bin/dnf** パスを **/usr/bin/dnf-3** に変更して、**44-installers.rules** ファイルを変更します。

第12章 FAPOLICYD を使用したアプリケーションの拒否および許可

ルールセットに基づいてアプリケーションの実行を許可または拒否するポリシーを設定して有効にすることで、効率的に悪意のある一般的に知られていないソフトウェアや、害を及ぼす可能性のあるソフトウェアの実行を回避できます。

12.1. FAPOLICYD の概要

fapolicyd ソフトウェアフレームワークは、ユーザー定義のポリシーに基づいてアプリケーションの実行を制御します。このフレームワークは、最適な方法で、システム上で信頼されていないアプリケーションや悪意のあるアプリケーションを実行されないようにします。

fapolicyd フレームワークは、以下のコンテンツを提供します。

- **fapolicyd** サービス
- **fapolicyd** コマンドラインユーティリティ
- **fapolicyd** RPM プラグイン
- **fapolicyd** ルール言語
- **fagenrules** スクリプト

管理者は、パス、ハッシュ、MIME タイプ、信頼に基づいて、すべてのアプリケーションに実行ルール **allow** および **deny** の両方を監査する定義できます。

fapolicyd フレームワークにより、信頼の概念が導入されます。アプリケーションは、システムパッケージマネージャーによって適切にインストールされると信頼されるため、システムの RPM データベースに登録されます。**fapolicyd** デーモンは、RPM データベースを信頼できるバイナリーとスクリプトのリストとして使用します。**fapolicyd** RPM プラグインは、DNF Package Manager または RPM Package Manager のいずれかで処理されるシステム更新をすべて登録するようになりました。プラグインは、このデータベースの変更を **fapolicyd** デーモンに通知します。アプリケーションを追加する他の方法では、カスタムルールを作成し、**fapolicyd** サービスを再起動する必要があります。

fapolicyd サービス設定は、次の構造を持つ **/etc/fapolicyd/** ディレクトリーにあります。

- **/etc/fapolicyd/fapolicyd.trust** ファイルには、信頼できるファイルのリストが含まれています。**/etc/fapolicyd/trust.d/** ディレクトリーで複数の信頼ファイルを使用することもできます。
- **allow** および **deny** の実行ルールを含むファイルの **/etc/fapolicyd/rules.d/** ディレクトリー。**fagenrules** スクリプトは、これらのコンポーネントルールファイルを **/etc/fapolicyd/compiled.rules** ファイルにマージします。
- **fapolicyd.conf** ファイルには、デーモンの設定オプションが含まれています。このファイルは、主にパフォーマンス調整の目的で役に立ちます。

/etc/fapolicyd/rules.d/ のルールは、それぞれ異なるポリシーゴールを表す複数のファイルで整理されます。対応するファイル名の先頭の数字によって、**/etc/fapolicyd/compiled.rules** での順序が決まります。

20

dracut 関連のルール

21

アップデーターのルール

30

パターン

40

ELF ルール

41

共有オブジェクトルール

42

信頼された ELF ルール

70

信頼された言語ルール

72

シェルルール

90

実行拒否ルール

95

オープン許可ルール

fapolicyd 整合性チェックには、次のいずれかの方法を使用できます。

- File-size チェック
- SHA-256 ハッシュの比較
- Integrity Measurement Architecture (IMA) サブシステム

デフォルトでは、**fapolicyd** は整合性チェックを行いません。ファイルサイズに基づいた整合性チェックは高速ですが、攻撃者はファイルの内容を置き換え、そのバイトサイズを保持することができます。SHA-256 チェックサムの計算とチェックがより安全ですが、システムのパフォーマンスに影響します。**fapolicyd.conf** の **integrity = ima** オプションでは、実行可能ファイルを含むすべてのファイルシステムでファイル拡張属性 (**xattr** と呼ばれます) のサポートが必要です。

関連情報

- **fapolicyd(8)**、**fapolicyd.rules(5)**、**fapolicyd.conf(5)**、**fapolicyd.trust(13)**、**fagenrules(8)**、および **fapolicyd-cli(1)** man ページ
- [カーネルの管理、監視、および更新](#) ドキュメントの [カーネル整合性サブシステムによるセキュリティの強化](#) の章
- **fapolicyd** パッケージとともに **/usr/share/doc/fapolicyd/** ディレクトリーにインストールされるドキュメントと **/usr/share/fapolicyd/sample-rules/README-rules** ファイル

12.2. FAPOLICYD のデプロイ

fapolicyd アプリケーションの許可リストフレームワークをデプロイする場合は、最初に permissive モードで設定を最初に試すか、デフォルト設定でサービスを直接有効にできます。

手順

1. **fapolicyd** パッケージをインストールします。

```
# dnf install fapolicyd
```

2. オプション: 最初に設定を試すには、モードを permissive に変更します。

- a. 任意のテキストエディターで **/etc/fapolicyd/fapolicyd.conf** ファイルを開きます。以下に例を示します。

```
# vi /etc/fapolicyd/fapolicyd.conf
```

- b. **permissive** オプションの値を **0** から **1** に変更し、ファイルを保存してエディターを終了します。

```
permissive = 1
```

または、サービスを開始する前に **fapolicyd --debug-deny --permissive** コマンドを使用して設定をデバッグできます。詳細は、[fapolicyd に関連する問題のトラブルシューティング](#) セクションを参照してください。

3. **fapolicyd** サービスを有効にして開始します。

```
# systemctl enable --now fapolicyd
```

4. **/etc/fapolicyd/fapolicyd.conf** を使用して permissive モードを有効にした場合:

- a. **fapolicyd** イベントを記録する Audit サービスを設定します。

```
# auditctl -w /etc/fapolicyd/ -p wa -k fapolicyd_changes
# service try-restart auditd
```

- b. アプリケーションを使用します。

- c. 監査ログで **fanotify** 拒否を確認します。以下に例を示します。

```
# ausearch -ts recent -m fanotify
```

- d. デバッグしたら、対応する値を **permissive = 0** に戻して permissive モードを無効にし、サービスを再起動します。

```
# systemctl restart fapolicyd
```

検証

1. **fapolicyd** サービスが正しく実行されていることを確認します。

```
# systemctl status fapolicyd
● fapolicyd.service - File Access Policy Daemon
```

```
Loaded: loaded (/usr/lib/systemd/system/fapolicyd.service; enabled; preset: disabled)
Active: active (running) since Tue 2024-10-08 05:53:50 EDT; 11s ago
...
Oct 08 05:53:51 machine1.example.com fapolicyd[4974]: Loading trust data from rpmdb
backend
Oct 08 05:53:51 machine1.example.com fapolicyd[4974]: Loading trust data from file
backend
Oct 08 05:53:51 machine1.example.com fapolicyd[4974]: Starting to listen for events
```

2. root 権限のないユーザーとしてログインし、以下のように **fapolicyd** が機能していることを確認します。

```
$ cp /bin/ls /tmp
$ /tmp/ls
bash: /tmp/ls: Operation not permitted
```

12.3. 追加の信頼ソースを使用してファイルを信頼できるものとしてマークする

fapolicyd フレームワークは、RPM データベースに含まれるファイルを信頼します。対応するエントリーを **/etc/fapolicyd/fapolicyd.trust** プレーンテキストファイルまたは **/etc/fapolicyd/trust.d/** ディレクトリーに追加することにより、追加のファイルを信頼済みとしてマークできます。**fapolicyd.trust** または **/etc/fapolicyd/trust.d** 内のファイルは、テキストエディターを直接使用するか、**fapolicyd-cli** コマンドを使用して変更できます。



注記

fapolicyd.trust または **trust.d/** を使用してファイルを信頼済みとしてマークすることは、パフォーマンス上の理由から、カスタムの **fapolicyd** ルールを記述するよりも優れています。

前提条件

- **fapolicyd** フレームワークがシステムにデプロイされます。

手順

1. カスタムバイナリーを必要なディレクトリーにコピーします。以下に例を示します。

```
$ cp /bin/ls /tmp
$ /tmp/ls
bash: /tmp/ls: Operation not permitted
```

2. カスタムバイナリーを信頼済みとしてマークし、対応するエントリーを **/etc/fapolicyd/trust.d/** の **myapp** ファイルに保存します。

```
# fapolicyd-cli --file add /tmp/ls --trust-file myapp
```

- **--trust-file** オプションをスキップすると、前のコマンドは対応する行を **/etc/fapolicyd/fapolicyd.trust** に追加します。

- ディレクトリー内のすべての既存ファイルを信頼済みとしてマークするには、**--file** オプションの引数としてディレクトリーパスを指定します。たとえば、**fapolicyd-cli --file add/tmp/my_bin_dir/--trust-file myapp** です。

3. fapolicyd データベースを更新します。

```
# fapolicyd-cli --update
```

注記

信頼されたファイルまたはディレクトリーの内容を変更すると、それらのチェックサムが変更されるため、**fapolicyd** はそれらを信頼済みと見なしなくなります。

新しいコンテンツを再び信頼できるようにするには、**fapolicyd-cli --file update** コマンドを使用してファイル信頼データベースを更新します。引数を何も指定しない場合、データベース全体が更新されます。または、特定のファイルまたはディレクトリーへのパスを指定できます。次に、**fapolicyd-cli --update** を使用してデータベースを更新します。

検証

- たとえば、カスタムバイナリーが実行できることを確認します。

```
$ /tmp/ls
ls
```

関連情報

- fapolicyd.trust(13)** man ページ

12.4. FAPOLICYD のカスタムの許可および拒否ルールの追加

fapolicyd パッケージのデフォルトのルールセットは、システム機能に影響しません。バイナリーやスクリプトを標準以外のディレクトリーに保存する、または **dnf** または **rpm** インストーラーを使用せずにアプリケーションを追加するなどのカスタムシナリオでは、追加のファイルを信頼済みとしてマークするか、新しいカスタムルールを追加する必要があります。

基本的なシナリオでは、[信頼の追加ソースを使用してファイルを信頼済みとしてマークする](#) ことを推奨します。特定のユーザーおよびグループ ID に対してのみカスタムバイナリーの実行を許可するなど、より高度なシナリオでは、新しいカスタムルールを **/etc/fapolicyd/rules.d/** ディレクトリーに追加します。

次の手順は、新しいルールを追加してカスタムバイナリーを許可する方法を示しています。

前提条件

- fapolicyd** フレームワークがシステムにデプロイされます。

手順

- カスタムバイナリーを必要なディレクトリーにコピーします。以下に例を示します。

```
$ cp /bin/ls /tmp
$ /tmp/ls
bash: /tmp/ls: Operation not permitted
```

2. **fapolicyd** サービスを停止します。

```
# systemctl stop fapolicyd
```

3. デバッグモードを使用して、対応するルールを識別します。**fapolicyd --debug** コマンドの出力は冗長で、**Ctrl+C** を押すか、対応するプロセスを強制終了するだけで停止できるため、エラー出力をファイルにリダイレクトします。この場合、**--debug** の代わりに **--debug-deny** オプションを使用して、アクセス拒否のみに出力を制限できます。

```
# fapolicyd --debug-deny 2> fapolicy.output &
[1] 51341
```

または、別の端末で **fapolicyd** デバッグモードを実行できます。

4. **fapolicyd** が拒否したコマンドを繰り返します。

```
$ /tmp/ls
bash: /tmp/ls: Operation not permitted
```

5. デバッグモードをフォアグラウンドで再開し、**Ctrl+C** を押して停止します。

```
# fg
fapolicyd --debug 2> fapolicy.output
^C
...
```

または、**fapolicyd** デバッグモードのプロセスを強制終了します。

```
# kill 51341
```

6. アプリケーションの実行を拒否するルールを見つけます。

```
# cat fapolicy.output | grep 'deny_audit'
...
rule=13 dec=deny_audit perm=execute auid=0 pid=6855 exe=/usr/bin/bash : path=/tmp/ls
ftype=application/x-executable trust=0
```

7. カスタムバイナリーの実行を妨げたルールを含むファイルを見つけます。この場合、**deny_audit perm=execute** ルールは **90-deny-execute.rules** ファイルに属します。

```
# ls /etc/fapolicyd/rules.d/
10-languages.rules 40-bad-elf.rules 72-shell.rules
20-dracut.rules 41-shared-obj.rules 90-deny-execute.rules
21-updaters.rules 42-trusted-elf.rules 95-allow-open.rules
30-patterns.rules 70-trusted-lang.rules

# cat /etc/fapolicyd/rules.d/90-deny-execute.rules
# Deny execution for anything untrusted
```

```
deny_audit perm=execute all : all
```

8. **/etc/fapolicyd/rules.d/** ディレクトリー内のカスタムバイナリーの実行を拒否するルールを含むルールファイルよりも辞書順で **前にある** ファイルに、新しい **allow** ルールを追加します。

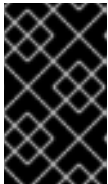
```
# touch /etc/fapolicyd/rules.d/80-myapps.rules
# vi /etc/fapolicyd/rules.d/80-myapps.rules
```

以下のルールを **80-myapps.rules** ファイルに挿入します。

```
allow perm=execute exe=/usr/bin/bash trust=1 : path=/tmp/ls ftype=application/x-executable
trust=0
```

または、**/etc/fapolicyd/rules.d/** のルールファイルに次のルールを追加して、**/tmp** ディレクトリー内のすべてのバイナリーの実行を許可することもできます。

```
allow perm=execute exe=/usr/bin/bash trust=1 : dir=/tmp/ trust=0
```



重要

指定したディレクトリーの下にあるすべてのディレクトリーに対してルールを再帰的に有効にするには、ルール内の **dir=** パラメーターの値に末尾のスラッシュを追加します (上記の例の **/tmp/**)。

9. カスタムバイナリーのコンテンツの変更を防ぐには、SHA-256 チェックサムを使用して必要なルールを定義します。

```
$ sha256sum /tmp/ls
780b75c90b2d41ea41679fcb358c892b1251b68d1927c80fbc0d9d148b25e836 ls
```

ルールを以下の定義に変更します。

```
allow perm=execute exe=/usr/bin/bash trust=1 :
sha256hash=780b75c90b2d41ea41679fcb358c892b1251b68d1927c80fbc0d9d148b25e836
```

10. コンパイル済みのリストが **/etc/fapolicyd/rules.d/** に設定されているルールと異なることを確認し、**/etc/fapolicyd/compiled.rules** ファイルに保存されているリストを更新します。

```
# fagenrules --check
/usr/sbin/fagenrules: Rules have changed and should be updated
# fagenrules --load
```

11. カスタムルールが、実行を妨げたルールの前に **fapolicyd** ルールのリストにあることを確認します。

```
# fapolicyd-cli --list
...
13. allow perm=execute exe=/usr/bin/bash trust=1 : path=/tmp/ls ftype=application/x-
```

```
executable trust=0
14. deny_audit perm=execute all : all
...
```

12. **fapolicyd** サービスを開始します。

```
# systemctl start fapolicyd
```

検証

1. たとえば、カスタムバイナリーが実行できることを確認します。

```
$ /tmp/ls
ls
```

関連情報

- システム上の **fapolicyd.rules(5)** および **fapolicyd-cli(1)** man ページ
- **/usr/share/fapolicyd/sample-rules/README-rules** ファイルの **fapolicyd** パッケージでインストールされるドキュメント。

12.5. FAPOLICYD 整合性チェックの有効化

デフォルトでは、**fapolicyd** は整合性チェックを実行しません。ファイルサイズまたは SHA-256 ハッシュのいずれかを比較して整合性チェックを実行するように **fapolicyd** を設定できます。Integrity Measurement Architecture (IMA) サブシステムを使用して整合性チェックを設定することもできます。

前提条件

- **fapolicyd** フレームワークがシステムにデプロイされます。

手順

1. 任意のテキストエディターで **/etc/fapolicyd/fapolicyd.conf** ファイルを開きます。以下に例を示します。

```
# vi /etc/fapolicyd/fapolicyd.conf
```

2. **整合性** オプションの値を **none** から **sha256** に変更し、ファイルを保存してエディターを終了します。

```
integrity = sha256
```

3. **fapolicyd** サービスを再起動します。

```
# systemctl restart fapolicyd
```

検証

1. 検証に使用するファイルのバックアップを作成します。

-


```
# cp /bin/more /bin/more.bak
```

2. **/bin/more** バイナリーの内容を変更します。

```
# cat /bin/less > /bin/more
```

3. 変更したバイナリーを一般ユーザーとして使用します。

```
# su example.user
$ /bin/more /etc/redhat-release
bash: /bin/more: Operation not permitted
```

4. 変更を元に戻します。

```
# mv -f /bin/more.bak /bin/more
```

12.6. FAPOLICYD に関連する問題のトラブルシューティング

fapolicyd アプリケーションフレームワークは、最も一般的な問題のトラブルシューティングを行うツールを提供します。また、**rpm** コマンドでインストールされたアプリケーションを信頼データベースに追加することもできます。

rpm を使用したアプリケーションのインストール

- **rpm** コマンドを使用してアプリケーションをインストールする場合は、**fapolicyd** RPM データベースを手動で更新する必要があります。

1. **アプリケーション** をインストールします。

```
# rpm -i application.rpm
```

2. データベースを更新します。

```
# fapolicyd-cli --update
```

この手順を飛ばすとシステムがフリーズする可能性があるため、再起動する必要があります。

サービスのステータス

- **fapolicyd** が正しく機能しない場合は、サービスステータスを確認します。

```
# systemctl status fapolicyd
```

fapolicyd-cli チェックとリスト

- **--check-config**、**--check-watch_fs**、および **--check-trustdb** オプションは、構文エラー、まだ監視されていないファイルシステム、およびファイルの不一致を見つけるのに役立ちます。次に例を示します。

```
# fapolicyd-cli --check-config
Daemon config is OK
```

```
# fapolicyd-cli --check-trustdb
/etc/selinux/targeted/contexts/files/file_contexts miscompares: size sha256
/etc/selinux/targeted/policy/policy.31 miscompares: size sha256
```

- **--list** オプションを使用して、ルール現在のリストとその順序を確認します。

```
# fapolicyd-cli --list
...
9. allow perm=execute all : trust=1
10. allow perm=open all : ftype=%languages trust=1
11. deny_audit perm=any all : ftype=%languages
12. allow perm=any all : ftype=text/x-shellscrip
13. deny_audit perm=execute all : all
...
```

デバッグモード

- デバッグモードは、一致したルール、データベースステータスなどに関する詳細情報を提供します。**fapolicyd** をデバッグモードに切り替えるには、以下を行います。

1. **fapolicyd** サービスを停止します。

```
# systemctl stop fapolicyd
```

2. デバッグモードを使用して、対応するルールを識別します。

```
# fapolicyd --debug
```

fapolicyd --debug コマンドの出力は冗長であるため、エラー出力をファイルにリダイレクトできます。

```
# fapolicyd --debug 2> fapolicy.output
```

または、**fapolicyd** がアクセスを拒否した場合にのみ出力をエントリーに制限するには、**--debug-deny** オプションを使用します。

```
# fapolicyd --debug-deny
```

また、アプリケーションの実行を防止する Permissive モードを使用することもできますが、一致した **fapolicyd** ルールのみが記録されます。

```
# fapolicyd --debug-deny --permissive
```

fapolicyd データベースの削除

- **fapolicyd** データベースに関連する問題を解決するには、データベースファイルを削除してください。

```
# systemctl stop fapolicyd
# fapolicyd-cli --delete-db
```



警告

/var/lib/fapolicyd/ ディレクトリーを削除しないでください。**fapolicyd** フレームワークは、このディレクトリー内のデータベースファイルのみを自動的に復元します。

fapolicyd データベースのダンプ

- **fapolicyd** には、有効なすべての信頼ソースからのエントリーが含まれます。データベースをダンプした後にエントリーを確認できます。

```
# fapolicyd-cli --dump-db
```

アプリケーションパイプ

- まれに、**fapolicyd** パイプファイルを削除するとロックアップが解決する場合があります。

```
# rm -f /var/run/fapolicyd/fapolicyd.fifo
```

関連情報

- システム上の **fapolicyd-cli(1)** man ページ

12.7. FAPOLICYD RHEL システムロールを使用してユーザーによる信頼できないコードの実行を防止する

fapolicyd RHEL システムロールを使用すると、**fapolicyd** サービスのインストールと設定を自動化できます。このロールを使用すると、RPM データベースや許可リストに指定されているアプリケーションなど、信頼できるアプリケーションのみをユーザーが実行できるようにサービスをリモートで設定できます。さらに、サービスは許可されたアプリケーションを実行する前に整合性チェックを実行できます。

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: Configuring fapolicyd
  hosts: managed-node-01.example.com
```

```

tasks:
  - name: Allow only executables installed from RPM database and specific files
    ansible.builtin.include_role:
      name: redhat.rhel_system_roles.fapolicyd
  vars:
    fapolicyd_setup_permissive: false
    fapolicyd_setup_integrity: sha256
    fapolicyd_setup_trust: rpmdb,file
    fapolicyd_add_trusted_file:
      - <path_to_allowed_command>
      - <path_to_allowed_service>

```

サンプル Playbook で指定されている設定は次のとおりです。

fapolicyd_setup_permissive: <true|false>

ポリシーの決定をカーネルに送信して適用することを有効または無効にします。デバッグおよびテスト目的の場合、この変数を **false** に設定します。

fapolicyd_setup_integrity: <type_type>

整合性チェック方法を定義します。次のいずれかの値を設定できます。

- **none** (デフォルト): 整合性チェックを無効にします。
- **size**: サービスにより、許可されたアプリケーションのファイルサイズのみを比較します。
- **ima**: サービスにより、カーネルの Integrity Measurement Architecture (IMA) がファイルの拡張属性に保存した SHA-256 ハッシュをチェックします。さらに、サイズチェックも実行します。ローカルは IMA カーネルサブシステムを設定しないことに注意してください。このオプションを使用するには、IMA サブシステムを手動で設定する必要があります。
- **sha256**: サービスにより、許可されたアプリケーションの SHA-256 ハッシュを比較します。

fapolicyd_setup_trust: <trust_backends>

信頼バックエンドのリストを定義します。**file** バックエンドを含める場合は、**fapolicyd_add_trusted_file** リストで許可されている実行可能ファイルを指定します。

Playbook で使用されるすべての変数の詳細は、コントロールノードの `/usr/share/ansible/roles/rhel-system-roles.fapolicyd.README.md` ファイルを参照してください。

2. Playbook の構文を検証します。

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

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

- 許可リストにないバイナリーアプリケーションをユーザーとして実行します。

```
$ ansible managed-node-01.example.com -m command -a 'su -c  
"/bin/not_authorized_application " <user_name>'  
bash: line 1: /bin/not_authorized_application: Operation not permitted non-zero return code
```

関連情報

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

12.8. 関連情報

- システム上の **fapolicyd** 関連の man ページ (`man -k fapolicyd` コマンドでリスト表示されるもの)
- [FOSDEM 2020 fapolicyd プレゼンテーション](#)

第13章 侵入型 USB デバイスに対するシステムの保護

USB デバイスには、スパイウェア、マルウェア、またはトロイの木馬が読み込まれ、データを盗んだり、システムを損傷する可能性があります。Red Hat Enterprise Linux 管理者は、**USBGuard** でこのような USB 攻撃を防ぐことができます。

13.1. USBGUARD

USBGuard ソフトウェアフレームワークを使用すると、カーネルの USB デバイス許可機能に基づいて許可されたデバイスおよび禁止されているデバイスの基本リストを使用して、侵入型 USB デバイスからシステムを保護できます。

USBGuard フレームワークは、次を提供します。

- 動的対話およびポリシー強制向けの IPC (inter-process communication) インターフェイスを使用したシステムサービスコンポーネント
- 実行中の **usbguard** システムサービスと対話するコマンドライン
- USB デバイス許可ポリシーを記述するルール言語
- 共有ライブラリーに実装されたシステムサービスコンポーネントと対話する C++ API

usbguard システムサービス設定ファイル (**/etc/usbguard/usbguard-daemon.conf**) には、IPC インターフェイスを使用するためのユーザーおよびグループを認可するオプションが含まれます。

重要

システムサービスは、USBGuard パブリック IPC インターフェイスを提供します。Red Hat Enterprise Linux では、このインターフェイスへのアクセスはデフォルトで root ユーザーに限定されています。

IPC インターフェイスへのアクセスを制限するには、**IPCAccessControlFiles** オプション (推奨)、**IPCAAllowedUsers** オプション、および **IPCAAllowedGroups** オプションを設定することを検討してください。

アクセス制御リスト (ACL) を未設定のままにしないでください。設定しないと、すべてのローカルユーザーに IPC インターフェイスが公開され、USB デバイスの許可状態を操作して USBGuard ポリシーを変更できるようになります。

13.2. USBGUARD のインストール

この手順を使用して、USBGuard フレームワークをインストールして開始します。

手順

1. **usbguard** パッケージをインストールします。

```
# dnf install usbguard
```

2. 初期ルールセットを作成します。

```
# usbguard generate-policy > /etc/usbguard/rules.conf
```

3. **usbguard** デーモンを起動し、システムの起動時に自動的に起動することを確認します。

```
# systemctl enable --now usbguard
```

検証

1. **usbguard** サービスが実行中であることを確認します。

```
# systemctl status usbguard
● usbguard.service - USBGuard daemon
   Loaded: loaded (/usr/lib/systemd/system/usbguard.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2019-11-07 09:44:07 CET; 3min 16s ago
     Docs: man:usbguard-daemon(8)
    Main PID: 6122 (usbguard-daemon)
      Tasks: 3 (limit: 11493)
     Memory: 1.2M
    CGroup: /system.slice/usbguard.service
            └─6122 /usr/sbin/usbguard-daemon -f -s -c /etc/usbguard/usbguard-daemon.conf

Nov 07 09:44:06 localhost.localdomain systemd[1]: Starting USBGuard daemon...
Nov 07 09:44:07 localhost.localdomain systemd[1]: Started USBGuard daemon.
```

2. USBGuard が認識する USB デバイスのリストを表示します。

```
# usbguard list-devices
4: allow id 1d6b:0002 serial "0000:02:00.0" name "xHCI Host Controller" hash...
```

関連情報

- **usbguard(1)** および **usbguard-daemon.conf(5)** の man ページ

13.3. CLI を使用した USB デバイスのブロックと許可

ターミナルで **usbguard** コマンドを使用すると、USB デバイスを許可およびブロックするように USBGuard を設定できます。

前提条件

- **usbguard** サービスがインストールされ、実行中である。

手順

1. USBGuard が認識する USB デバイスのリストを表示します。以下に例を示します。

```
# usbguard list-devices
1: allow id 1d6b:0002 serial "0000:00:06.7" name "EHCI Host Controller" hash
   "JDOb0BiktYs2ct3mSQKopnOOV2h9MGYADwhT+oUtF2s=" parent-hash
   "4PHGcaDKWtPjKDwYpIRG722cB9SIGz9I9lea93+Gt9c=" via-port "usb1" with-interface
   09:00:00
...
6: block id 1b1c:1ab1 serial "000024937962" name "Voyager" hash
```

```
"CrXgiaWlf2bZAU+5WkzOE7y0rdSO82XMzubn7HD95Q=" parent-hash
"JDOb0BiktYs2ct3mSQKopnOOV2h9MGYADwhT+oUtF2s=" via-port "1-3" with-interface
08:06:50
```

2. デバイス <6> がシステムと対話することを許可します。

```
# usbguard allow-device <6>
```

3. デバイス <6> の許可を解除し、デバイスを削除します。

```
# usbguard reject-device <6>
```

4. デバイス <6> の許可を解除し、デバイスを保持します。

```
# usbguard block-device <6>
```

注記

USBGuard では、**block** および **reject** は以下の意味で使用されます。

block

今はこのデバイスとやり取りしません。

reject

このデバイスは存在しないものとして無視します。

関連情報

- システム上の **usbguard(1)** man ページ
- **usbguard --help** コマンド

13.4. USB デバイスの永続的なブロックおよび許可

-p オプションを使用すると、USB デバイスを永続的にブロックおよび許可できます。これにより、デバイス固有のルールが現在のポリシーに追加されます。

前提条件

- **usbguard** サービスがインストールされ、実行中である。

手順

1. **usbguard** デーモンがルールの書き込みを許可するように SELinux を設定します。
 - a. **usbguard** に関連する **semanage** ブール値を表示します。

```
# semanage boolean -l | grep usbguard
usbguard_daemon_write_conf    (off , off) Allow usbguard to daemon write conf
usbguard_daemon_write_rules   (on , on) Allow usbguard to daemon write rules
```


- b. 必要に応じて、**usbguard_daemon_write_rules** のブール値が無効になっている場合は、有効にします。

```
# semanage boolean -m --on usbguard_daemon_write_rules
```

2. USBGuard が認識する USB デバイスのリストを表示します。

```
# usbguard list-devices
1: allow id 1d6b:0002 serial "0000:00:06.7" name "EHCI Host Controller" hash
"JDOb0BiktYs2ct3mSQKopnOOV2h9MGYADwhT+oUtF2s=" parent-hash
"4PHGcaDKWtPjKDwYpIRG722cB9SIGz9l9lea93+Gt9c=" via-port "usb1" with-interface
09:00:00
...
6: block id 1b1c:1ab1 serial "000024937962" name "Voyager" hash
"CrXgiaWlf2bZAU+5WkzOE7y0rdSO82XMzubn7HDb95Q=" parent-hash
"JDOb0BiktYs2ct3mSQKopnOOV2h9MGYADwhT+oUtF2s=" via-port "1-3" with-interface
08:06:50
```

3. デバイス **6** がシステムと対話することを永続的に許可します。

```
# usbguard allow-device 6 -p
```

4. デバイス **6** の許可を永続的に解除し、デバイスを削除します。

```
# usbguard reject-device 6 -p
```

5. デバイス **6** の許可を永続的に解除し、デバイスを保持します。

```
# usbguard block-device 6 -p
```

注記

USBGuard では、**block** および **reject** は以下の意味で使用されます。

block

今はこのデバイスとやり取りしません。

reject

このデバイスは存在しないものとして無視します。

検証

1. USBGuard ルールに加えた変更が含まれていることを確認します。

```
# usbguard list-rules
```

関連情報

- システム上の **usbguard(1)** man ページ
- usbguard --help** コマンドを使用してリスト表示される組み込みヘルプ。

13.5. USB デバイス用のカスタムポリシーの作成

以下の手順では、シナリオの要件を反映する USB デバイス用のルールセットを作成する手順を説明します。

前提条件

- **usbguard** サービスがインストールされ、実行中である。
- **/etc/usbguard/rules.conf** ファイルに、**usbguard generate-policy** コマンドによって生成された初期ルールセットが含まれている。

手順

1. 現在接続している USB デバイスを許可するポリシーを作成し、生成されたルールを **rules.conf** ファイルに保存します。

```
# usbguard generate-policy --no-hashes > ./rules.conf
```

--no-hashes オプションは、デバイスのハッシュ属性を生成しません。設定のハッシュ属性は永続的ではない可能性があるため、回避してください。

2. 選択したテキストエディターで **rules.conf** ファイルを編集します。次に例を示します。

```
# vi ./rules.conf
```

3. 必要に応じて、ルールを追加、削除、または編集します。たとえば、以下のルールを使用すると、大容量ストレージインターフェイスが1つあるデバイスのみがシステムと対話できます。

```
allow with-interface equals { 08:*:* }
```

詳細なルール言語の説明とその他の例は、**usbguard-rules.conf(5)** の man ページを参照してください。

4. 更新したポリシーをインストールします。

```
# install -m 0600 -o root -g root rules.conf /etc/usbguard/rules.conf
```

5. **usbguard** デーモンを再起動して、変更を適用します。

```
# systemctl restart usbguard
```

検証

1. カスタムルールがアクティブポリシーにあることを確認します。以下に例を示します。

```
# usbguard list-rules
...
4: allow with-interface 08:*:*
...
```

関連情報

- システム上の **usbguard-rules.conf(5)** man ページ

13.6. USB デバイス用の構造化されたカスタムポリシーの作成

/etc/usbguard/rules.d/ ディレクトリー内の複数の **.conf** ファイルで、カスタムの USBGuard ポリシーを整理できます。**usbguard-daemon** により、メインの **rules.conf** ファイルとディレクトリー内の **.conf** ファイルがアルファベット順に結合されます。

前提条件

- **usbguard** サービスがインストールされ、実行中である。

手順

1. 現在接続している USB デバイスを許可するポリシーを作成し、生成されたルールを、新しい **.conf** ファイル (例: **policy.conf**) ファイルに保存します。

```
# usbguard generate-policy --no-hashes > ./policy.conf
```

--no-hashes オプションは、デバイスのハッシュ属性を生成しません。設定のハッシュ属性は永続的ではない可能性があるため、回避してください。

2. 任意のテキストエディターで **policy.conf** ファイルを編集します。次に例を示します。

```
# vi ./policy.conf
...
allow id 04f2:0833 serial "" name "USB Keyboard" via-port "7-2" with-interface { 03:01:01
03:00:00 } with-connect-type "unknown"
...
```

3. 選択した行を別の **.conf** ファイルに移動します。



注記

ファイル名の先頭にある 2 つの数字は、デーモンが設定ファイルを読み込む順序を指定します。

たとえば、キーボードのルールを新規 **.conf** ファイルにコピーします。

```
# grep "USB Keyboard" ./policy.conf > ./10keyboards.conf
```

4. 新しいポリシーを **/etc/usbguard/rules.d/** ディレクトリーにインストールします。

```
# install -m 0600 -o root -g root 10keyboards.conf /etc/usbguard/rules.d/10keyboards.conf
```

5. 残りの行をメインの **rules.conf** ファイルに移動します。

```
# grep -v "USB Keyboard" ./policy.conf > ./rules.conf
```

6. 残りのルールをインストールします。

```
# install -m 0600 -o root -g root rules.conf /etc/usbguard/rules.conf
```

7. **usbguard** デーモンを再起動して、変更を適用します。

```
# systemctl restart usbguard
```

検証

1. アクティブな USBGuard ルールをすべて表示します。

```
# usbguard list-rules
...
15: allow id 04f2:0833 serial "" name "USB Keyboard" hash
"KxM/iddRe/WSCocgiuQIVs6Dn0VEza7KiHoDeTz0fyg=" parent-hash
"2i6ZBJfTI5BakXF7Gba84/Cp1gslnNc1DM6vWQpie3s=" via-port "7-2" with-interface {
03:01:01 03:00:00 } with-connect-type "unknown"
...
```

2. **rules.conf** ファイルと、**/etc/usbguard/rules.d/** ディレクトリー内の **.conf** ファイルの内容をすべて表示します。

```
# cat /etc/usbguard/rules.conf /etc/usbguard/rules.d/*.conf
```

3. アクティブなルールに、ファイルのすべてのルールが正しく、正しい順序で含まれていることを確認します。

関連情報

- システム上の **usbguard-rules.conf(5)** man ページ

13.7. USBGUARD IPC インターフェイスを使用するユーザーおよびグループの許可

この手順を使用して、特定のユーザーまたはグループが USBGuard のパブリック IPC インターフェイスを使用するように認可します。デフォルトでは、root ユーザーだけがこのインターフェイスを使用できます。

前提条件

- **usbguard** サービスがインストールされ、実行中である。
- **/etc/usbguard/rules.conf** ファイルに、**usbguard generate-policy** コマンドによって生成された初期ルールセットが含まれている。

手順

1. 任意のテキストエディターで **/etc/usbguard/usbguard-daemon.conf** ファイルを編集します。

```
# vi /etc/usbguard/usbguard-daemon.conf
```

2. たとえば、**wheel** グループの全ユーザーが IPC インターフェイスを使用できるように、ルールがある行を追加して、ファイルを保存します。

```
IPCAllowGroups=wheel
```

3. **usbguard** コマンドで、ユーザーまたはグループを追加することもできます。たとえば、次のコマンドを使用すると、**joesec** ユーザーが **Devices** セクションおよび **Exceptions** セクションに完全アクセスできます。さらに、**joesec** は現行ポリシーのリスト表示および変更を行うことができます。

```
# usbguard add-user joesec --devices ALL --policy modify,list --exceptions ALL
```

joesec ユーザーに付与されたパーミッションを削除するには、**usbguard remove-user joesec** コマンドを使用します。

4. **usbguard** デーモンを再起動して、変更を適用します。

```
# systemctl restart usbguard
```

関連情報

- **usbguard(1)** および **usbguard-rules.conf(5)** の man ページ

13.8. LINUX 監査ログへの USBGUARD 許可イベントの記録

以下の手順に従って、USBguard 許可イベントのログと標準の Linux 監査ログを1つにまとめます。デフォルトでは、**usbguard** デーモンは **/var/log/usbguard/usbguard-audit.log** ファイルにイベントを記録します。

前提条件

- **usbguard** サービスがインストールされ、実行中である。
- **auditd** サービスが実行中である。

手順

1. **usbguard-daemon.conf** ファイルを、選択したテキストエディターで編集します。

```
# vi /etc/usbguard/usbguard-daemon.conf
```

2. **AuditBackend** オプションを、**FileAudit** から **LinuxAudit** に変更します。

```
AuditBackend=LinuxAudit
```

3. **usbguard** デーモンを再起動して、設定変更を適用します。

```
# systemctl restart usbguard
```

検証

1. **audit** デーモンログで USB 許可イベントを照会します。次に例を示します。

```
# ausearch -ts recent -m USER_DEVICE
```

関連情報

- システム上の **usbguard-daemon.conf(5)** man ページ

13.9. 関連情報

- **usbguard(1)**、**usbguard-rules.conf(5)**、**usbguard-daemon(8)**、および **usbguard-daemon.conf(5)** の man ページ
- [USBGuard ホームページ](#)

第14章 リモートロギングソリューションの設定

環境内の各種マシンからのログをロギングサーバーに集中的に記録するために、クライアントシステムからサーバーに特定の基準に合致するログを記録するように **Rsyslog** アプリケーションを設定できます。

14.1. RSYSLOG ロギングサービス

Rsyslog アプリケーションは、**systemd-journald** サービスと組み合わせて、Red Hat Enterprise Linux でローカルおよびリモートのロギングサポートを提供します。**rsyslogd** デーモンは、ジャーナルから **systemd-journald** サービスが受信した **syslog** メッセージを継続的に読み取ります。**rsyslogd** は、**syslog** イベントをフィルタリングして処理し、**rsyslog** ログファイルに記録するか、設定に応じて他のサービスに転送します。

rsyslogd デーモンは、拡張されたフィルタリング、暗号化で保護されたメッセージのリレー、入出力モジュール、TCP プロトコルおよび UDP プロトコルを使用した転送のサポートも提供します。

rsyslog の主な設定ファイルである `/etc/rsyslog.conf` では、どの **rsyslogd** がメッセージを処理するかに応じてルールを指定できます。通常は、ソースおよびトピック (ファシリティ) 別および緊急度 (優先度) 別にメッセージを分類し、メッセージがその基準に合致したときに実行するアクションを割り当てることができます。

`/etc/rsyslog.conf` では、**rsyslogd** が維持するログファイルのリストも確認できます。ほとんどのログファイルは `/var/log/` ディレクトリーにあります。**httpd**、**samba** などの一部のアプリケーションは、ログファイルを `/var/log/` 内のサブディレクトリーに保存します。

関連情報

- システム上の **rsyslogd(8)** および **rsyslog.conf(5)** man ページ
- `/usr/share/doc/rsyslog/html/index.html` ファイルに **rsyslog-doc** パッケージでインストールされたドキュメント。

14.2. RSYSLOG ドキュメントのインストール

Rsyslog アプリケーションには、<https://www.rsyslog.com/doc/> で利用可能な詳細なオンラインドキュメントがありますが、**rsyslog-doc** ドキュメントパッケージをローカルにインストールすることもできます。

前提条件

- システムで **AppStream** リポジトリをアクティベートしている。
- sudo** を使用して新規パッケージをインストールする権限がある。

手順

- rsyslog-doc** パッケージをインストールします。

```
# dnf install rsyslog-doc
```

検証

- 任意のブラウザで `/usr/share/doc/rsyslog/html/index.html` ファイルを開きます。次に例を示します。

```
$ firefox /usr/share/doc/rsyslog/html/index.html &
```

14.3. TCP でのリモートロギング用のサーバーの設定

Rsyslog アプリケーションを使用すると、ロギングサーバーを実行し、個別のシステムがログファイルをロギングサーバーに送信するように設定できます。TCP 経由でリモートロギングを使用するには、サーバーとクライアントの両方を設定します。サーバーは、クライアントシステムにより送信されたログを収集し、分析します。

Rsyslog アプリケーションを使用すると、ログメッセージがネットワークを介してサーバーに転送される中央ロギングシステムを維持できます。サーバーが利用できない場合にメッセージが失われないようにするには、転送アクションのアクションキューを設定します。これにより、送信に失敗したメッセージは、サーバーが再度到達可能になるまでローカルに保存されます。このようなキューは、UDP プロトコルを使用する接続用に設定できないことに注意してください。

omfwd プラグインは、UDP または TCP による転送を提供します。デフォルトのプロトコルは UDP です。このプラグインは組み込まれているため、読み込む必要はありません。

デフォルトでは、**rsyslog** はポート **514** で TCP を使用します。

前提条件

- rsyslog がサーバーシステムにインストールされている。
- サーバーに **root** としてログインしている。
- **policycoreutils-python-utils** パッケージは、**semanage** コマンドを使用して任意の手順でインストールします。
- **firewalld** サービスが実行中である。

手順

1. 必要に応じて、**rsyslog** トラフィックに別のポートを使用するには、SELinux タイプ **syslogd_port_t** をポートに追加します。たとえば、ポート **30514** を有効にします。

```
# semanage port -a -t syslogd_port_t -p tcp 30514
```

2. 必要に応じて、**rsyslog** トラフィックに別のポートを使用するには、**firewalld** がそのポートでの着信 **rsyslog** トラフィックを許可するように設定します。たとえば、ポート **30514** で TCP トラフィックを許可します。

```
# firewall-cmd --zone=<zone-name> --permanent --add-port=30514/tcp
success
# firewall-cmd --reload
```

3. **/etc/rsyslog.d/** ディレクトリーに新規ファイル (例: **remotelog.conf**) を作成し、以下のコンテンツを挿入します。

```
# Define templates before the rules that use them
```



```
# Per-Host templates for remote systems
template(name="TmplAuthpriv" type="list") {
    constant(value="/var/log/remote/auth/")
    property(name="hostname")
    constant(value="/")
    property(name="programname" SecurePath="replace")
    constant(value=".log")
}

template(name="TmplMsg" type="list") {
    constant(value="/var/log/remote/msg/")
    property(name="hostname")
    constant(value="/")
    property(name="programname" SecurePath="replace")
    constant(value=".log")
}

# Provides TCP syslog reception
module(load="imtcp")

# Adding this ruleset to process remote messages
ruleset(name="remote1"){
    authpriv.* action(type="omfile" DynaFile="TmplAuthpriv")
    *.info;mail.none;authpriv.none;cron.none
    action(type="omfile" DynaFile="TmplMsg")
}

input(type="imtcp" port="30514" ruleset="remote1")
```

4. **/etc/rsyslog.d/remotelog.conf** ファイルへの変更を保存します。

5. **/etc/rsyslog.conf** ファイルの構文をテストします。

```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-2.el8, config validation run...
rsyslogd: End of config validation run. Bye.
```

6. **Rsyslog** サービスがロギングサーバーで実行中で、有効になっていることを確認します。

```
# systemctl status rsyslog
```

7. **rsyslog** サービスを再起動します。

```
# systemctl restart rsyslog
```

8. 必要に応じて、**rsyslog** が有効になっていない場合は、再起動後に **rsyslog** サービスが自動的に起動するようにします。

```
# systemctl enable rsyslog
```

環境内の他のシステムからログファイルを受け取り、保存するように、ログサーバーが設定されています。

- システム上の **rsyslogd(8)**、**rsyslog.conf(5)**、**semanage(8)**、および **firewall-cmd(1)** man ページ
- `/usr/share/doc/rsyslog/html/index.html` ファイルに **rsyslog-doc** パッケージでインストールされたドキュメント。

14.4. TCP 経由のサーバーへのリモートロギングの設定

TCP プロトコル経由でログメッセージをサーバーに転送するようにシステムを設定できます。**omfwd** プラグインは、UDP または TCP による転送を提供します。デフォルトのプロトコルは UDP です。プラグインは組み込まれているのでロードする必要はありません。

前提条件

- **rsyslog** パッケージが、サーバーに報告する必要があるクライアントシステムにインストールされている。
- リモートロギング用にサーバーを設定している。
- 指定したポートが SELinux で許可され、ファイアウォールで開いている。
- システムには、**polycoreutils-python-utils** パッケージが含まれています。このパッケージは、標準以外のポートを SELinux 設定に追加するための **semanage** コマンドを提供します。

手順

1. `/etc/rsyslog.d/` ディレクトリーに新規ファイル (例: **10-remotelog.conf**) を作成し、以下のコンテンツを挿入します。

```
*.* action(type="omfwd"
    queue.type="linkedlist"
    queue.filename="example_fwd"
    action.resumeRetryCount="-1"
    queue.saveOnShutdown="on"
    target="example.com" port="30514" protocol="tcp"
)
```

ここでは、以下のようになります。

- **queue.type="linkedlist"** 設定は、LinkedList インメモリーキューを有効にします。
- **queue.filename** 設定は、ディスクストレージを定義します。バックアップファイルは、前のグローバルの **workDirectory** ディレクティブで指定された作業ディレクトリーに **example_fwd** 接頭辞を付けて作成されます。
- **action.resumeRetryCount -1** 設定は、サーバーが応答しない場合に接続を再試行するときに **rsyslog** がメッセージを破棄しないようにします。
- **queue.saveOnShutdown="on"** 設定は、**rsyslog** がシャットダウンした場合にインメモリーデータを保存します。
- 最後の行は、受信したすべてのメッセージをロギングサーバーに転送します。ポートの指定は任意です。
この設定では、**rsyslog** はメッセージをサーバーに送信しますが、リモートサーバーに到達

できない場合には、メッセージをメモリーに保持します。ディスク上にあるファイルは、設定されたメモリーキュー領域が **rsyslog** で不足するか、シャットダウンする必要がある場合にのみ作成されます。これにより、システムパフォーマンスが向上します。



注記

Rsyslog は設定ファイル **/etc/rsyslog.d/** を字句順に処理します。

2. **rsyslog** サービスを再起動します。

```
# systemctl restart rsyslog
```

検証

クライアントシステムがサーバーにメッセージを送信することを確認するには、以下の手順に従います。

1. クライアントシステムで、テストメッセージを送信します。

```
# logger test
```

2. サーバーシステムで、**/var/log/messages** ログを表示します。以下に例を示します。

```
# cat /var/log/remote/msg/hostname/root.log
Feb 25 03:53:17 hostname root[6064]: test
```

hostname はクライアントシステムのホスト名です。ログには、**logger** コマンドを入力したユーザーのユーザー名 (この場合は **root**) が含まれていることに注意してください。

関連情報

- システム上の **rsyslogd(8)** および **rsyslog.conf(5)** man ページ
- **/usr/share/doc/rsyslog/html/index.html** ファイルに **rsyslog-doc** パッケージでインストールされたドキュメント。

14.5. TLS 暗号化リモートロギングの設定

デフォルトでは、Rsyslog はプレーンテキスト形式でリモートロギング通信を送信します。シナリオでこの通信チャネルのセキュリティを確保する必要がある場合は、TLS を使用して暗号化できます。

TLS を介した暗号化されたトランスポートを使用するには、サーバーとクライアントの両方を設定します。サーバーは、クライアントシステムにより送信されたログを収集し、分析します。

ossl ネットワークストリームドライバー (OpenSSL) または **gtls** ストリームドライバー (GnuTLS) のいずれかを使用できます。



注記

ネットワークに接続されていない、厳格な認可を受けているなど、セキュリティが強化された別のシステムがある場合は、その別のシステムを認証局 (CA) として使用します。

サーバー側では **global**、**module**、**input** レベルで、クライアント側では **global** および **action** レベルで、ストリームドライバーを使用して接続設定をカスタマイズできます。より具体的な設定は、より一般的な設定よりも優先されます。そのため、たとえば、ほとんどの接続のグローバル設定で **ossl** を使用し、特定の接続の入力とアクション設定で **gtls** を使用することができます。

前提条件

- クライアントシステムとサーバーシステムの両方に **root** にアクセスできる。
- サーバーおよびクライアントシステムに次のパッケージがインストールされている。
 - **rsyslog** パッケージ
 - **ossl** ネットワークストリームドライバー用の **rsyslog-openssl** パッケージ
 - **gtls** ネットワークストリームドライバー用の **rsyslog-gnutls** パッケージ
 - **certtool** コマンドを使用して証明書を生成するための **gnutls-utils** パッケージ
- ログサーバーの **/etc/pki/ca-trust/source/anchors/** ディレクトリーには、次の証明書があり、**update-ca-trust** コマンドを使用してシステム設定を更新します。
 - **ca-cert.pem** - ログサーバーとクライアントで鍵と証明書を検証できる CA 証明書。
 - **server-cert.pem** - ロギングサーバーの公開鍵。
 - **server-key.pem** - ロギングサーバーの秘密鍵。
- ログクライアントでは、次の証明書が **/etc/pki/ca-trust/source/anchors/** ディレクトリーにあり、**update-ca-trust** を使用してシステム設定を更新します。
 - **ca-cert.pem** - ログサーバーとクライアントで鍵と証明書を検証できる CA 証明書。
 - **client-cert.pem** - クライアントの公開鍵。
 - **client-key.pem** - クライアントの秘密鍵。
 - サーバーが RHEL 9.2 以降を実行し、FIPS モードが有効になっている場合、クライアントが Extended Master Secret (EMS) 拡張機能をサポートしているか、TLS 1.3 を使用している必要があります。EMS を使用しない TLS 1.2 接続は失敗します。詳細は、Red Hat ナレッジベースソリューション [TLS extension "Extended Master Secret" enforced](#) を参照してください。

手順

1. クライアントシステムから暗号化したログを受信するようにサーバーを設定します。
 - a. **/etc/rsyslog.d/** ディレクトリーに、新規ファイル (**securelogser.conf** など) を作成します。
 - b. 通信を暗号化するには、設定ファイルに、サーバーの証明書ファイルへのパス、選択した認証方法、および TLS 暗号化に対応するストリームドライバーが含まれている必要があります。**/etc/rsyslog.d/securelogser.conf** に以下の行を追加します。

```
# Set certificate files
global(
```

```

DefaultNetstreamDriverCAFile="/etc/pki/ca-trust/source/anchors/ca-cert.pem"
DefaultNetstreamDriverCertFile="/etc/pki/ca-trust/source/anchors/server-cert.pem"
DefaultNetstreamDriverKeyFile="/etc/pki/ca-trust/source/anchors/server-key.pem"
)

# TCP listener
module(
    load="imtcp"
    PermittedPeer=["client1.example.com", "client2.example.com"]
    StreamDriver.AuthMode="x509/name"
    StreamDriver.Mode="1"
    StreamDriver.Name="sssl"
)

# Start up listener at port 514
input(
    type="imtcp"
    port="514"
)

```



注記

GnuTLS ドライバーが必要な場合は、**StreamDriver.Name="gtls"** 設定オプションを使用します。**x509/name** よりも厳密ではない認証モードの詳細は、**rsyslog-doc** にインストールされているドキュメントを参照してください。

- c. オプション: RHEL 9.4 で提供される Rsyslog バージョン 8.2310 からは、接続設定をカスタマイズできます。これを行うには、**input** セクションを以下に置き換えます。

```

input(
    type="imtcp"
    Port="50515"
    StreamDriver.Name="<driver>"
    streamdriver.CAFile="/etc/rsyslog.d/<ca1>.pem"
    streamdriver.CertFile="/etc/rsyslog.d/<server1-cert>.pem"
    streamdriver.KeyFile="/etc/rsyslog.d/<server1-key>.pem"
)

```

- 使用するドライバーに応じて、**<driver>** を **sssl** または **gtls** に置き換えます。
- **<ca1>** はカスタマイズする接続の CA 証明書に、**<server1-cert>** は証明書に、**<server1-key>** は鍵に置き換えます。

- d. 変更を **/etc/rsyslog.d/securelogser.conf** ファイルに保存します。
- e. **/etc/rsyslog.conf** ファイルの構文と **/etc/rsyslog.d/** ディレクトリー内のすべてのファイルを確認します。

```

# rsyslogd -N 1
rsyslogd: version 8.1911.0-2.el8, config validation run (level 1)...
rsyslogd: End of config validation run. Bye.

```

- f. **Rsyslog** サービスがログインサーバーで実行中で、有効になっていることを確認します。

```
# systemctl status rsyslog
```

- g. **rsyslog** サービスを再起動します。

```
# systemctl restart rsyslog
```

- h. オプション: Rsyslog が有効になっていない場合は、再起動後に **rsyslog** サービスが自動的に起動されることを確認してください。

```
# systemctl enable rsyslog
```

2. 暗号化したログをサーバーに送信するようにクライアントを設定するには、以下のコマンドを実行します。

- a. クライアントシステムで、**/etc/rsyslog.d/** ディレクトリーに、新規ファイル (**securelogcli.conf** など) を作成します。
- b. **/etc/rsyslog.d/securelogcli.conf** に以下の行を追加します。

```
# Set certificate files
global(
    DefaultNetstreamDriverCAFile="/etc/pki/ca-trust/source/anchors/ca-cert.pem"
    DefaultNetstreamDriverCertFile="/etc/pki/ca-trust/source/anchors/client-cert.pem"
    DefaultNetstreamDriverKeyFile="/etc/pki/ca-trust/source/anchors/client-key.pem"
)

# Set up the action for all messages
*. * action(
    type="omfwd"
    StreamDriver="ossf"
    StreamDriverMode="1"
    StreamDriverPermittedPeers="server.example.com"
    StreamDriverAuthMode="x509/name"
    target="server.example.com" port="514" protocol="tcp"
)
```



注記

GnuTLS ドライバーが必要な場合は、**StreamDriver.Name="gtls"** 設定オプションを使用します。

- c. オプション: RHEL 9.4 で提供される Rsyslog バージョン 8.2310 からは、接続設定をカスタマイズできます。これを行うには、**action** セクションを以下に置き換えます。

```
local1.* action(
    type="omfwd"
    StreamDriver="<driver>"
    StreamDriverMode="1"
    StreamDriverAuthMode="x509/certvalid"
    streamDriver.CAFile="/etc/rsyslog.d/<ca1>.pem"
    streamDriver.CertFile="/etc/rsyslog.d/<client1-cert>.pem"
```

```
streamDriver.KeyFile="/etc/rsyslog.d/<client1-key>.pem"
target="server.example.com" port="514" protocol="tcp"
)
```

- 使用するドライバーに応じて、**<driver>** を **ossl** または **gtls** に置き換えます。
- **<ca1>** はカスタマイズする接続の CA 証明書に、**<client1-cert>** は証明書に、**<client1-key>** は鍵に置き換えます。

d. 変更を **/etc/rsyslog.d/securelogcli.conf** ファイルに保存します。

e. **/etc/rsyslog.conf** ファイルの構文と **/etc/rsyslog.d/** ディレクトリー内のその他のファイルを確認します。

```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-2.el8, config validation run (level 1)...
rsyslogd: End of config validation run. Bye.
```

f. **Rsyslog** サービスがロギングサーバーで実行中で、有効になっていることを確認します。

```
# systemctl status rsyslog
```

g. **rsyslog** サービスを再起動します。

```
# systemctl restart rsyslog
```

h. オプション: Rsyslog が有効になっていない場合は、再起動後に **rsyslog** サービスが自動的に起動されることを確認してください。

```
# systemctl enable rsyslog
```

検証

クライアントシステムがサーバーにメッセージを送信することを確認するには、以下の手順に従います。

1. クライアントシステムで、テストメッセージを送信します。

```
# logger test
```

2. サーバーシステムで、**/var/log/messages** ログを表示します。以下に例を示します。

```
# cat /var/log/remote/msg/<hostname>/root.log
Feb 25 03:53:17 <hostname> root[6064]: test
```

<hostname> はクライアントシステムのホスト名です。ログには、logger コマンドを入力したユーザーのユーザー名 (この場合は **root**) が含まれていることに注意してください。

関連情報

- システム上の **certtool(1)**、**openssl(1)**、**update-ca-trust(8)**、**rsyslogd(8)**、および **rsyslog.conf(5)** man ページ

- `/usr/share/doc/rsyslog/html/index.html` に **rsyslog-doc** パッケージでインストールされたドキュメント。
- [TLS での logging システムロールの使用](#)

14.6. UDP でリモートロギング情報を受信するためのサーバー設定

Rsyslog アプリケーションを使用すると、リモートシステムからロギング情報を受信するようにシステムを設定できます。UDP 経由でリモートロギングを使用するには、サーバーとクライアントの両方を設定します。受信サーバーは、クライアントシステムが送信したログの収集および分析を行います。デフォルトでは、**rsyslog** はポート **514** で UDP を使用して、リモートシステムからログ情報を受信します。

以下の手順に従って、UDP プロトコルでクライアントシステムが送信したログの収集および分析を行うサーバーを設定します。

前提条件

- **rsyslog** がサーバーシステムにインストールされている。
- サーバーに **root** としてログインしている。
- **polycoreutils-python-utils** パッケージは、**semanage** コマンドを使用して任意の手順でインストールします。
- **firewalld** サービスが実行中である。

手順

1. 必要に応じて、デフォルトのポート **514** 以外の **rsyslog** トラフィックに別のポートを使用するには、次のコマンドを実行します。
 - a. SELinux ポリシー設定に **syslogd_port_t** SELinux タイプを追加し、**portno** は **rsyslog** で使用するポート番号に置き換えます。

```
# semanage port -a -t syslogd_port_t -p udp portno
```

- b. **rsyslog** の受信トラフィックを許可するように **firewalld** を設定します。**portno** はポート番号に、**zone** は **rsyslog** が使用するゾーンに置き換えます。

```
# firewall-cmd --zone=zone --permanent --add-port=portno/udp
success
# firewall-cmd --reload
```

- c. ファイアウォールルールを再読み込みします。

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

2. `/etc/rsyslog.d/` ディレクトリーに **.conf** の新規ファイル (例: **remotelogserv.conf**) を作成し、以下のコンテンツを挿入します。

```
# Define templates before the rules that use them
# Per-Host templates for remote systems
template(name="TmplAuthpriv" type="list") {
```



```

    constant(value="/var/log/remote/auth/")
    property(name="hostname")
    constant(value="/")
    property(name="programname" SecurePath="replace")
    constant(value=".log")
}

template(name="TplMsg" type="list") {
    constant(value="/var/log/remote/msg/")
    property(name="hostname")
    constant(value="/")
    property(name="programname" SecurePath="replace")
    constant(value=".log")
}

# Provides UDP syslog reception
module(load="imudp")

# This ruleset processes remote messages
ruleset(name="remote1"){
    authpriv.* action(type="omfile" DynaFile="TplAuthpriv")
    *.info;mail.none;authpriv.none;cron.none
    action(type="omfile" DynaFile="TplMsg")
}

input(type="imudp" port="514" ruleset="remote1")

```

514 は、**rsyslog** がデフォルトで使用するポート番号です。代わりに別のポートを指定できます。

3. **/etc/rsyslog.conf** ファイルの構文と **/etc/rsyslog.d/** ディレクトリー内の全 **.conf** ファイルを確認します。

```

# rsyslogd -N 1
rsyslogd: version 8.1911.0-2.el8, config validation run...

```

4. **rsyslog** サービスを再起動します。

```

# systemctl restart rsyslog

```

5. 必要に応じて、**rsyslog** が有効になっていない場合は、再起動後に **rsyslog** サービスが自動的に起動するようにします。

```

# systemctl enable rsyslog

```

関連情報

- システム上の **rsyslogd(8)**、**rsyslog.conf(5)**、**semanage(8)**、および **firewall-cmd(1)** man ページ
- **/usr/share/doc/rsyslog/html/index.html** ファイルに **rsyslog-doc** パッケージでインストールされたドキュメント。

14.7. UDP 経由のサーバーへのリモートロギングの設定

UDP プロトコル経由でログメッセージをサーバーに転送するようにシステムを設定できます。**omfwd** プラグインは、UDP または TCP による転送を提供します。デフォルトのプロトコルは UDP です。プラグインは組み込まれているのでロードする必要はありません。

前提条件

- **rsyslog** パッケージが、サーバーに報告する必要のあるクライアントシステムにインストールされている。
- [UDP でリモートロギング情報を受信するためのサーバー設定](#) で説明されているように、リモートロギング用にサーバーを設定している。

手順

1. **/etc/rsyslog.d/** ディレクトリーに **.conf** の新規ファイル (例: **10-remotelogcli.conf**) を作成し、以下のコンテンツを挿入します。

```
*.* action(type="omfwd"
    queue.type="linkedlist"
    queue.filename="example_fwd"
    action.resumeRetryCount="-1"
    queue.saveOnShutdown="on"
    target="example.com" port="portno" protocol="udp"
)
```

ここでは、以下ようになります。

- **queue.type="linkedlist"** 設定は、LinkedList インメモリーキューを有効にします。
- **queue.filename** 設定は、ディスクストレージを定義します。バックアップファイルは、前のグローバルの **workDirectory** ディレクティブで指定された作業ディレクトリーに **example_fwd** 接頭辞を付けて作成されます。
- **action.resumeRetryCount -1** 設定は、サーバーが応答しない場合に接続を再試行するときに **rsyslog** がメッセージを破棄しないようにします。
- **queue.saveOnShutdown="on"** 設定を有効にすると、**rsyslog** がシャットダウンした場合にインメモリーデータが保存されます。
- **portno** 値は、**rsyslog** で使用するポート番号です。デフォルト値は **514** です。
- 最後の行は受信メッセージをすべてロギングサーバーに転送します。ポートの指定は任意です。
この設定では、**rsyslog** はメッセージをサーバーに送信しますが、リモートサーバーに到達できない場合には、メッセージをメモリーに保持します。ディスク上にあるファイルは、設定されたメモリーキュー領域が **rsyslog** で不足するか、シャットダウンする必要がある場合にのみ作成されます。これにより、システムパフォーマンスが向上します。



注記

Rsyslog は設定ファイル **/etc/rsyslog.d/** を字句順に処理します。

2. **rsyslog** サービスを再起動します。

```
# systemctl restart rsyslog
```

3. 必要に応じて、**rsyslog** が有効になっていない場合は、再起動後に **rsyslog** サービスが自動的に起動するようにします。

```
# systemctl enable rsyslog
```

検証

クライアントシステムがサーバーにメッセージを送信することを確認するには、以下の手順に従います。

1. クライアントシステムで、テストメッセージを送信します。

```
# logger test
```

2. サーバースystemで、**/var/log/remote/msg/hostname/root.log** ログを表示します。以下に例を示します。

```
# cat /var/log/remote/msg/hostname/root.log
Feb 25 03:53:17 hostname root[6064]: test
```

hostname はクライアントシステムのホスト名です。ログには、logger コマンドを入力したユーザーのユーザー名 (この場合は **root**) が含まれていることに注意してください。

関連情報

- システム上の **rsyslogd(8)** および **rsyslog.conf(5)** man ページ
- **rsyslog-doc** パッケージとともに **/usr/share/doc/rsyslog/html/index.html** にインストールされるドキュメント

14.8. RSYSLOG の負荷分散ヘルパー

Rsyslog をクラスターで使用する場合、**RebindInterval** 設定を変更することで Rsyslog の負荷分散を改善できます。

RebindInterval では、現在の接続を切断して再確立する間隔を指定します。この設定は、TCP、UDP、および RELP のトラフィックに適用されます。ロードバランサーはこれを新しい接続と認識し、メッセージを別の物理ターゲットシステムに転送します。

RebindInterval は、ターゲットシステムの IP アドレスが変更された場合に役立ちます。Rsyslog アプリケーションは、接続の確立時に IP アドレスをキャッシュするため、メッセージは同じサーバーに送信されます。IP アドレスが変更されると、Rsyslog サービスが再起動するまで UDP パケットが失われます。接続を再確立すると、IP が DNS により再度解決されます。

TCP、UDP、および RELP トラフィックに対する RebindInterval の使用例

```
action(type="omfwd" protocol="tcp" RebindInterval="250" target="example.com" port="514" ...)
```

```
action(type="omfwd" protocol="udp" RebindInterval="250" target="example.com" port="514" ...)
```

```
action(type="omrelp" RebindInterval="250" target="example.com" port="6514" ...)
```

14.9. 信頼できるリモートロギングの設定

Reliable Event Logging Protocol (RELP) を使用すると、メッセージ損失のリスクを大幅に軽減して TCP で **syslog** メッセージを送受信できます。RELP は、信頼できるイベントメッセージを配信するので、メッセージ損失が許されない環境で有用です。RELP を使用するには、**imrelp** の入力モジュール (サーバー上での実行とログの受信) と **omrelp** 出力モジュール (クライアント上での実行とロギングサーバーへのログの送信) を設定します。

前提条件

- **rsyslog** パッケージ、**librelp** パッケージ、および **rsyslog-relp** パッケージをサーバーおよびクライアントシステムにインストールしている。
- 指定したポートが SELinux で許可され、ファイアウォール設定で開放されている。

手順

1. 信頼できるリモートロギング用にクライアントシステムを設定します。

- a. クライアントシステムの **/etc/rsyslog.d/** ディレクトリーに、**relpclient.conf** などと名前を指定して新しい **.conf** ファイルを作成し、以下のコンテンツを挿入します。

```
module(load="omrelp")
*. * action(type="omrelp" target="_target_IP_" port="_target_port_")
```

ここでは、以下のようになります。

- **target_IP** は、ロギングサーバーの IP アドレスに置き換えます。
- **target_port** はロギングサーバーのポートに置き換えます。

- b. 変更を **/etc/rsyslog.d/relpclient.conf** ファイルに保存します。

- c. **rsyslog** サービスを再起動します。

```
# systemctl restart rsyslog
```

- d. 必要に応じて、**rsyslog** が有効になっていない場合は、再起動後に **rsyslog** サービスが自動的に起動するようにします。

```
# systemctl enable rsyslog
```

2. 信頼できるリモートロギング用にサーバーシステムを設定します。

- a. サーバーシステムの **/etc/rsyslog.d/** ディレクトリーに、**relpserv.conf** などと名前を指定して新しい **.conf** ファイルを作成し、以下のコンテンツを挿入します。

```
ruleset(name="relp"){
*. * action(type="omfile" file="_log_path_")
}
```

```
module(load="imrelp")
input(type="imrelp" port="_target_port_" ruleset="relp")
```

ここでは、以下のようになります。

- **log_path** は、メッセージを保存するパスを指定します。
 - **target_port** はロギングサーバーのポートに置き換えます。クライアント設定ファイルと同じ値を使用します。
- b. **/etc/rsyslog.d/relpserv.conf** ファイルへの変更を保存します。
- c. **rsyslog** サービスを再起動します。

```
# systemctl restart rsyslog
```

- d. 必要に応じて、**rsyslog** が有効になっていない場合は、再起動後に **rsyslog** サービスが自動的に起動するようにします。

```
# systemctl enable rsyslog
```

検証

クライアントシステムがサーバーにメッセージを送信することを確認するには、以下の手順に従います。

1. クライアントシステムで、テストメッセージを送信します。

```
# logger test
```

2. サーバーシステムで、指定された **log_path** でログを表示します。以下に例を示します。

```
# cat /var/log/remote/msg/hostname/root.log
Feb 25 03:53:17 hostname root[6064]: test
```

hostname はクライアントシステムのホスト名です。ログには、logger コマンドを入力したユーザーのユーザー名 (この場合は **root**) が含まれていることに注意してください。

関連情報

- システム上の **rsyslogd(8)** および **rsyslog.conf(5)** man ページ
- **/usr/share/doc/rsyslog/html/index.html** ファイルに **rsyslog-doc** パッケージでインストールされたドキュメント。

14.10. サポート対象の RSYSLOG モジュール

Rsyslog アプリケーションの機能を拡張するために、特定のモジュールを使用できます。モジュールは、追加の入力 (入力モジュール)、出力 (出力モジュール)、およびその他の機能を提供します。モジュールは、モジュールの読み込み後に利用可能な設定ディレクティブを追加で提供することも可能です。

以下のコマンドを使用して、システムにインストールされている入出力モジュールをリスト表示できます。

```
# ls /usr/lib64/rsyslog/{i,o}m*
```

rsyslog-doc パッケージをインストールした

後、**/usr/share/doc/rsyslog/html/configuration/modules/idx_output.html** ファイルで使用可能なすべての **rsyslog** モジュールのリストを表示できます。

14.11. カーネルメッセージをリモートホストに記録するように NETCONSOLE サービスを設定

ディスクへのログインやシリアルコンソールの使用ができない場合は、**netconsole** カーネルモジュールおよび同じ名前のサービスを使用して、ネットワーク経由でカーネルメッセージをリモートの **rsyslog** サービスに記録できます。

前提条件

- **rsyslog** などのシステムログサービスがリモートホストにインストールされている。
- リモートシステムログサービスは、このホストから受信ログエントリを受け取るように設定されています。

手順

1. **netconsole-service** パッケージをインストールします。

```
# dnf install netconsole-service
```

2. **/etc/sysconfig/netconsole** ファイルを編集し、**SYSLOGADDR** パラメーターをリモートホストの IP アドレスに設定します。

```
# SYSLOGADDR=192.0.2.1
```

3. **netconsole** サービスを有効にして起動します。

```
# systemctl enable --now netconsole
```

検証

- リモートシステムログサーバーの **/var/log/messages** ファイルを表示します。

14.12. 関連情報

- **/usr/share/doc/rsyslog/html/index.html** ファイルに **rsyslog-doc** パッケージでインストールされたドキュメント。
- システム上の **rsyslog.conf(5)** および **rsyslogd(8)** man ページ
- ナレッジベースの記事 [Configuring system logging without journald](#)

- ナレッジベースの記事 [Negative effects of the RHEL default logging setup on performance and their mitigations](#)
- [logging システムロールの使用](#) の章

第15章 LOGGING システムロールの使用

システム管理者は、**logging** システムロールを使用して、Red Hat Enterprise Linux ホストをロギングサーバーとして設定し、多数のクライアントシステムからログを収集できます。

15.1. LOGGING RHEL システムロールを使用したローカルログメッセージのフィルタリング

logging RHEL システムロールのプロパティーベースのフィルターを使用すると、さまざまな条件に基づいてローカルログメッセージをフィルタリングできます。その結果、たとえば以下を実現できます。

- 明確なログ: トラフィック量の多い環境では、ログが急増することがあります。エラーなどの特定のメッセージに注目することで、問題をより早く特定できるようになります。
- システムパフォーマンスの最適化: ログの量が多すぎると、通常、システムパフォーマンスが低下します。重要なイベントのみを選択的にログに記録することで、リソースの枯渇を防ぎ、システムをより効率的に運用できます。
- セキュリティーの強化: システムエラーやログイン失敗などのセキュリティメッセージを効率的にフィルタリングすることで、関連するログのみを取得できます。これは、違反を検出し、コンプライアンス基準を満たすために重要です。

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: Deploy the logging solution
  hosts: managed-node-01.example.com
  tasks:
    - name: Filter logs based on a specific value they contain
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.logging
      vars:
        logging_inputs:
          - name: files_input
            type: basics
        logging_outputs:
          - name: files_output0
            type: files
            property: msg
            property_op: contains
            property_value: error
            path: /var/log/errors.log
          - name: files_output1
```



```

type: files
property: msg
property_op: "!contains"
property_value: error
path: /var/log/others.log
logging_flows:
- name: flow0
  inputs: [files_input]
  outputs: [files_output0, files_output1]

```

サンプル Playbook で指定されている設定は次のとおりです。

logging_inputs

ロギングの入力ディクショナリーのリストを定義します。**type: basics** オプションを指定すると、**systemd** ジャーナルまたは Unix ソケットからの入力対象になります。

logging_outputs

ロギングの出力ディクショナリーのリストを定義します。**type: files** オプションにより、ローカルファイル (通常は **/var/log/** ディレクトリー内) にログを保存できます。**property: msg**、**property: contains**、および **property_value: error** オプションを指定すると、**error** 文字列を含むすべてのログが **/var/log/errors.log** ファイルに保存されます。**property: msg**、**property: !contains**、および **property_value: error** オプションを指定すると、他のすべてのログが **/var/log/others.log** ファイルに保存されます。**error** 値は、フィルタリングする必要がある文字列に置き換えることができます。

logging_flows

ロギングのフローディクショナリーのリストを定義して、**logging_inputs** と **logging_outputs** の関係を指定します。**inputs: [files_input]** オプションは、ログの処理を開始する入力のリストを指定します。**outputs: [files_output0, files_output1]** オプションは、ログ送信先の出力のリストを指定します。

Playbook で使用されるすべての変数の詳細は、コントロールノードの **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** ファイルを参照してください。

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. 管理対象ノードで、**/etc/rsyslog.conf** ファイルの構文をテストします。

```

# rsyslogd -N 1
rsyslogd: version 8.1911.0-6.el8, config validation run...
rsyslogd: End of config validation run. Bye.

```

2. 管理対象ノードで、システムが **error** 文字列を含むメッセージをログに送信することを確認します。
 - a. テストメッセージを送信します。

```
# logger error
```

- b. 以下のように `/var/log/errors.log` ログを表示します。

```
# cat /var/log/errors.log
Aug  5 13:48:31 hostname root[6778]: error
```

hostname はクライアントシステムのホスト名に置き換えます。ログには、`logger` コマンドを入力したユーザーのユーザー名 (この場合は **root**) が含まれていることに注意してください。

関連情報

- `/usr/share/ansible/roles/rhel-system-roles/logging/README.md` ファイル
- `/usr/share/doc/rhel-system-roles/logging/` ディレクトリー
- システム上の `rsyslog.conf(5)` および `syslog(3)` の man ページ

15.2. LOGGING RHEL システムロールを使用したリモートロギングソリューションの適用

logging RHEL システムロールを使用して、1つ以上のクライアントで **systemd-journal** サービスからログを取得し、リモートサーバーに転送するリモートロギングソリューションを設定できます。このサーバーは、**remote_rsyslog** および **remote_files** 設定からリモート入力を受け取り、リモートホスト名によって指定されたディレクトリー内のローカルファイルにログを出力します。

その結果、たとえば次のようなユースケースに対応できます。

- ログの集中管理: 複数のマシンのログメッセージを1つのストレージポイントから収集、アクセス、管理することで、日々の監視とトラブルシューティングのタスクが簡素化されます。また、このユースケースでは、ログメッセージを確認するために個々のマシンにログインする必要性が軽減されます。
- セキュリティーの強化: ログメッセージを1カ所に集中して保存すると、セキュアで改ざん不可能な環境にログを保存しやすくなります。このような環境により、セキュリティーインシデントをより効果的に検出して対応し、監査要件を満たすことが容易になります。
- ログ分析の効率向上: 複数のマシンまたはサービスにまたがる複雑な問題を迅速にトラブルシューティングするには、複数のシステムからのログメッセージを相関させることが重要です。これにより、さまざまなソースからのイベントをすばやく分析し、相互参照することができます。

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。
- サーバーまたはクライアントシステムの SELinux ポリシーでポートを定義し、それらのポートのファイアウォールを開く。デフォルトの SELinux ポリシーには、ポート 601、514、6514、10514、および 20514 が含まれます。別のポートを使用するには、[クライアントおよびサーバーシステムの SELinux ポリシーの変更](#) を参照してください。

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: Deploy the logging solution
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure the server to receive remote input
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.logging
      vars:
        logging_inputs:
          - name: remote_udp_input
            type: remote
            udp_ports: [ 601 ]
          - name: remote_tcp_input
            type: remote
            tcp_ports: [ 601 ]
        logging_outputs:
          - name: remote_files_output
            type: remote_files
        logging_flows:
          - name: flow_0
            inputs: [remote_udp_input, remote_tcp_input]
            outputs: [remote_files_output]

- name: Deploy the logging solution
  hosts: managed-node-02.example.com
  tasks:
    - name: Configure the server to output the logs to local files in directories named by
      remote host names
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.logging
      vars:
        logging_inputs:
          - name: basic_input
            type: basics
        logging_outputs:
          - name: forward_output0
            type: forwards
            severity: info
            target: <host1.example.com>
            udp_port: 601
          - name: forward_output1
            type: forwards
            facility: mail
            target: <host1.example.com>
            tcp_port: 601
```

```
logging_flows:
  - name: flows0
    inputs: [basic_input]
    outputs: [forward_output0, forward_output1]
```

サンプル Playbook の最初のプレイで指定されている設定は次のとおりです。

logging_inputs

ロギングの入力ディクショナリーのリストを定義します。**type: remote** オプションを指定すると、ネットワークを介した他のロギングシステムからのリモート入力対象になります。**udp_ports: [601]** オプションは、監視する UDP ポート番号のリストを定義します。**tcp_ports: [601]** オプションは、監視する TCP ポート番号のリストを定義します。**udp_ports** と **tcp_ports** の両方が設定されている場合、**udp_ports** が使用され、**tcp_ports** は削除されます。

logging_outputs

ロギングの出力ディクショナリーのリストを定義します。**type: remote_files** オプションを指定すると、ログの送信元であるリモートホストとプログラム名ごとに、出力がローカルファイルに保存されます。

logging_flows

ロギングのフローディクショナリーのリストを定義して、**logging_inputs** と **logging_outputs** の関係を指定します。**inputs: [remote_udp_input, remote_tcp_input]** オプションは、ログの処理を開始する入力のリストを指定します。**outputs: [remote_files_output]** オプションは、ログ送信先の出力のリストを指定します。

サンプル Playbook の 2 番目のプレイで指定されている設定は次のとおりです。

logging_inputs

ロギングの入力ディクショナリーのリストを定義します。**type: basics** オプションを指定すると、**systemd** ジャーナルまたは Unix ソケットからの入力対象になります。

logging_outputs

ロギングの出力ディクショナリーのリストを定義します。**type: forwards** オプションにより、ネットワーク経由でリモートログサーバーにログを送信できます。**severity: info** オプションは、重大度が情報のログメッセージを示します。**facility: mail** オプションは、ログメッセージを生成するシステムプログラムの種類を示します。**target: <host1.example.com>** オプションは、リモートログサーバーのホスト名を指定します。**udp_port: 601/tcp_port: 601** オプションは、リモートログサーバーがリッスンする UDP/TCP ポートを定義します。

logging_flows

ロギングのフローディクショナリーのリストを定義して、**logging_inputs** と **logging_outputs** の関係を指定します。**inputs: [basic_input]** オプションは、ログの処理を開始する入力のリストを指定します。**outputs: [forward_output0, forward_output1]** オプションは、ログ送信先の出力のリストを指定します。

Playbook で使用されるすべての変数の詳細は、コントロールノードの `/usr/share/ansible/roles/rhel-system-roles.logging/README.md` ファイルを参照してください。

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. クライアントとサーバーシステムの両方で、**/etc/rsyslog.conf** ファイルの構文をテストします。

```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-6.el8, config validation run (level 1), master config
/etc/rsyslog.conf
rsyslogd: End of config validation run. Bye.
```

2. クライアントシステムがサーバーにメッセージを送信することを確認します。

- a. クライアントシステムで、テストメッセージを送信します。

```
# logger test
```

- b. サーバーシステムで、**/var/log/<host2.example.com>/messages** ログを表示します。次に例を示します。

```
# cat /var/log/<host2.example.com>/messages
Aug  5 13:48:31 <host2.example.com> root[6778]: test
```

<host2.example.com> は、クライアントシステムのホスト名に置き換えます。ログには、**logger** コマンドを入力したユーザーのユーザー名 (この場合は **root**) が含まれていることに注意してください。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.logging/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/logging/** ディレクトリー
- **rsyslog.conf(5)** および **syslog(3)** man ページ

15.3. TLS を使用した LOGGING RHEL システムロールの使用

Transport Layer Security (TLS) は、コンピューターネットワーク上でセキュアな通信を可能にするために設計された暗号化プロトコルです。

RHEL システムロールの **logging** を使用すると、ログメッセージのセキュアな転送を設定して、1つ以上のクライアントで **systemd-journal** サービスからログを取得し、TLS を使用してリモートサーバーに転送できます。

通常、リモートロギングソリューションでログを転送するための TLS は、インターネットなどの信頼性の低いネットワークやパブリックネットワーク経由で機密データを送信する場合に使用されます。また、TLS で証明書を使用することで、クライアントから正しい信頼できるサーバーにログを確実に転送

できます。これにより、"中間者攻撃"などの攻撃を防ぐことができます。

15.3.1. TLS を使用したクライアントロギングの設定

logging RHEL システムロールを使用すると、RHEL クライアントでのロギングを設定し、TLS 暗号化を使用してログをリモートロギングシステムに転送できます。

この手順では、秘密鍵と証明書を作成します。その後、Ansible インベントリーのクライアントグループ内の全ホストに TLS を設定します。TLS プロトコルは、メッセージ送信を暗号化し、ネットワーク経由でログを安全に転送します。



注記

証明書を作成するために、Playbook で **certificate** RHEL システムロールを呼び出す必要はありません。このロールは、**logging_certificates** 変数が設定されている場合、**logging** RHEL システムロールによって自動的に呼び出されます。

CA が作成された証明書に署名できるようにするには、管理対象ノードが IdM ドメインに登録されている必要があります。

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。
- 管理対象ノードが IdM ドメインに登録されている。
- 管理対象ノード上で設定するロギングサーバーが RHEL 9.2 以降を実行し、FIPS モードが有効になっている場合、クライアントが Extended Master Secret (EMS) 拡張機能をサポートしているか、TLS 1.3 を使用している。EMS を使用しない TLS 1.2 接続は失敗します。詳細は、Red Hat ナレッジベースソリューション [TLS extension "Extended Master Secret" enforced](#) を参照してください。

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: Configure remote logging solution using TLS for secure transfer of logs
  hosts: managed-node-01.example.com
  tasks:
    - name: Deploying files input and forwards output with certs
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.logging
      vars:
        logging_certificates:
          - name: logging_cert
            dns: ['www.example.com']
            ca: ipa
            principal: "logging/{{ inventory_hostname }}@IDM.EXAMPLE.COM"
        logging_pki_files:
```

```

- ca_cert: /local/path/to/ca_cert.pem
  cert: /local/path/to/logging_cert.pem
  private_key: /local/path/to/logging_cert.pem
logging_inputs:
- name: input_name
  type: files
  input_log_path: /var/log/containers/*.log
logging_outputs:
- name: output_name
  type: forwards
  target: your_target_host
  tcp_port: 514
  tls: true
  pki_authmode: x509/name
  permitted_server: 'server.example.com'
logging_flows:
- name: flow_name
  inputs: [input_name]
  outputs: [output_name]

```

サンプル Playbook で指定されている設定は次のとおりです。

logging_certificates

このパラメーターの値は、**certificate** RHEL システムロールの **certificate_requests** に渡され、秘密鍵と証明書の作成に使用されます。

logging_pki_files

このパラメーターを使用すると、TLS に使用する CA、証明書、および鍵ファイルを検索するためにロギングで使用するパスとその他の設定 (サブパラメーター **ca_cert**、**ca_cert_src**、**cert**、**cert_src**、**private_key**、**private_key_src**、および **tls** で指定) を設定できます。



注記

logging_certificates を使用して管理対象ノードにファイルを作成する場合は、**ca_cert_src**、**cert_src**、および **private_key_src** を使用しないでください。これらは、**logging_certificates** によって作成されていないファイルのコピーに使用されます。

ca_cert

管理対象ノード上の CA 証明書ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/ca.pem** で、ファイル名はユーザーが設定します。

cert

管理対象ノード上の証明書ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/server-cert.pem** で、ファイル名はユーザーが設定します。

private_key

管理対象ノード上の秘密鍵ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/private/server-key.pem** で、ファイル名はユーザーが設定します。

ca_cert_src

ターゲットホストの **ca_cert** で指定された場所にコピーされる、コントロールノード上の CA 証明書ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

cert_src

ターゲットホストの **cert** で指定された場所にコピーされる、コントロールノード上の証明書ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

private_key_src

ターゲットホストの **private_key** で指定された場所にコピーされる、コントロールノード上の秘密鍵ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

tls

このパラメーターを **true** に設定すると、ネットワーク上でログがセキュアに転送されます。セキュアなラッパーが必要ない場合は、**tls: false** に設定します。

Playbook で使用されるすべての変数の詳細は、コントロールノードの `/usr/share/ansible/roles/rhel-system-roles.logging/README.md` ファイルを参照してください。

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.logging/README.md` ファイル
- `/usr/share/doc/rhel-system-roles/logging/` ディレクトリー
- `/usr/share/ansible/roles/rhel-system-roles.certificate/README.md` ファイル
- `/usr/share/doc/rhel-system-roles/certificate/` ディレクトリー
- [RHEL システムロールを使用した証明書の要求](#)
- `rsyslog.conf(5)` および `syslog(3)` man ページ

15.3.2. TLS を使用したサーバーロギングの設定

logging RHEL システムロールを使用すると、RHEL サーバーでのロギングを設定し、TLS 暗号化を使用してリモートロギングシステムからログを受信するようにサーバーを設定できます。

この手順では、秘密鍵と証明書を作成します。その後、Ansible インベントリーのサーバーグループ内の全ホストに TLS を設定します。



注記

証明書を作成するために、Playbook で **certificate** RHEL システムロールを呼び出す必要はありません。**logging** RHEL システムロールがこのロールを自動的に呼び出します。

CA が作成された証明書に署名できるようにするには、管理対象ノードが IdM ドメインに登録されている必要があります。

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。
- 管理対象ノードが IdM ドメインに登録されている。
- 管理対象ノード上で設定するロギングサーバーが RHEL 9.2 以降を実行し、FIPS モードが有効になっている場合、クライアントが Extended Master Secret (EMS) 拡張機能をサポートしているか、TLS 1.3 を使用している。EMS を使用しない TLS 1.2 接続は失敗します。詳細は、Red Hat ナレッジベースソリューション [TLS extension "Extended Master Secret" enforced](#) を参照してください。

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: Configure remote logging solution using TLS for secure transfer of logs
  hosts: managed-node-01.example.com
  tasks:
    - name: Deploying remote input and remote_files output with certs
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.logging
      vars:
        logging_certificates:
          - name: logging_cert
            dns: ['www.example.com']
            ca: ipa
            principal: "logging/{{ inventory_hostname }}@IDM.EXAMPLE.COM"
        logging_pki_files:
          - ca_cert: /local/path/to/ca_cert.pem
            cert: /local/path/to/logging_cert.pem
            private_key: /local/path/to/logging_cert.pem
        logging_inputs:
          - name: input_name
            type: remote
            tcp_ports: [514]
            tls: true
            permitted_clients: ['clients.example.com']
        logging_outputs:
          - name: output_name
            type: remote_files
            remote_log_path: /var/log/remote/%FROMHOST%/PROGRAMNAME:::secpath-
```

```
replace%.log
  async_writing: true
  client_count: 20
  io_buffer_size: 8192
  logging_flows:
    - name: flow_name
      inputs: [input_name]
      outputs: [output_name]
```

サンプル Playbook で指定されている設定は次のとおりです。

logging_certificates

このパラメーターの値は、**certificate** RHEL システムロールの **certificate_requests** に渡され、秘密鍵と証明書の作成に使用されます。

logging_pki_files

このパラメーターを使用すると、TLS に使用する CA、証明書、および鍵ファイルを検索するためにロギングで使用するパスとその他の設定 (サブパラメーター

ca_cert、**ca_cert_src**、**cert**、**cert_src**、**private_key**、**private_key_src**、および **tls** で指定) を設定できます。



注記

logging_certificates を使用して管理対象ノードにファイルを作成する場合は、**ca_cert_src**、**cert_src**、および **private_key_src** を使用しないでください。これらは、**logging_certificates** によって作成されていないファイルのコピーに使用されます。

ca_cert

管理対象ノード上の CA 証明書ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/ca.pem** で、ファイル名はユーザーが設定します。

cert

管理対象ノード上の証明書ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/server-cert.pem** で、ファイル名はユーザーが設定します。

private_key

管理対象ノード上の秘密鍵ファイルへのパスを表します。デフォルトのパスは **/etc/pki/tls/private/server-key.pem** で、ファイル名はユーザーが設定します。

ca_cert_src

ターゲットホストの **ca_cert** で指定された場所にコピーされる、コントロールノード上の CA 証明書ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

cert_src

ターゲットホストの **cert** で指定された場所にコピーされる、コントロールノード上の証明書ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

private_key_src

ターゲットホストの **private_key** で指定された場所にコピーされる、コントロールノード上の秘密鍵ファイルへのパスを表します。**logging_certificates** を使用する場合は、これを使用しないでください。

tls

このパラメーターを **true** に設定すると、ネットワーク上でログがセキュアに転送されます。セキュアなラッパーが必要ない場合は、**tls: false** に設定します。

Playbook で使用されるすべての変数の詳細は、コントロールノードの `/usr/share/ansible/roles/rhel-system-roles.logging/README.md` ファイルを参照してください。

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.logging/README.md` ファイル
- `/usr/share/doc/rhel-system-roles/logging/` ディレクトリー
- [RHEL システムロールを使用した証明書の要求](#)
- `rsyslog.conf(5)` および `syslog(3)` man ページ

15.4. RELP で LOGGING RHEL システムロールの使用

Reliable Event Logging Protocol (RELP) とは、TCP ネットワークを使用する、データとメッセージロギング用のネットワーキングプロトコルのことです。イベントメッセージを確実に配信するので、メッセージの損失が許されない環境で使用できます。

RELP の送信側はコマンド形式でログエントリーを転送し、受信側は処理後に確認応答します。RELP は、一貫性を保つために、転送されたコマンドごとにトランザクション番号を保存し、各種メッセージの復旧します。

RELP Client と RELP Server の間に、リモートロギングシステムを検討することができます。RELP Client はリモートロギングシステムにログを転送し、RELP Server はリモートロギングシステムから送信されたすべてのログを受け取ります。このユースケースを実現するには、**logging** RHEL システムロールを使用して、ログエントリーが確実に送受信されるようにロギングシステムを設定できます。

15.4.1. RELP を使用したクライアントロギングの設定

logging RHEL システムロールを使用すると、ローカルに保存されたログメッセージを RELP を使用してリモートロギングシステムに転送するように設定できます。

この手順では、Ansible インベントリーの **clients** グループ内の全ホストに RELP を設定します。RELP 設定は Transport Layer Security (TLS) を使用して、メッセージ送信を暗号化し、ネットワーク経由でログを安全に転送します。

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: Configure client-side of the remote logging solution using RELP
  hosts: managed-node-01.example.com
  tasks:
    - name: Deploy basic input and RELP output
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.logging
  vars:
    logging_inputs:
      - name: basic_input
        type: basics
    logging_outputs:
      - name: relp_client
        type: relp
        target: logging.server.com
        port: 20514
        tls: true
        ca_cert: /etc/pki/tls/certs/ca.pem
        cert: /etc/pki/tls/certs/client-cert.pem
        private_key: /etc/pki/tls/private/client-key.pem
        pki_authmode: name
        permitted_servers:
          - '*.server.example.com'
    logging_flows:
      - name: example_flow
        inputs: [basic_input]
        outputs: [relp_client]
```

サンプル Playbook で指定されている設定は次のとおりです。

target

リモートロギングシステムが稼働しているホスト名を指定する必須パラメーターです。

port

リモートロギングシステムがリスンしているポート番号です。

tls

ネットワーク上でログをセキュアに転送します。セキュアなラッパーが必要ない場合は、**tls** 変数を **false** に設定します。デフォルトでは **tls** パラメーターは **true** に設定されますが、RELP を使用する場合には鍵/証明書およびトリプレット **{ca_cert、cert、private_key}** や **{ca_cert_src、cert_src、private_key_src}** が必要です。

- **{ca_cert_src、cert_src、private_key_src}** のトリプレットを設定すると、デフォルト

の場所 (`/etc/pki/tls/certs` と `/etc/pki/tls/private`) が、コントロールノードからファイルを転送するため、管理対象ノードの宛先として使用されます。この場合、ファイル名はトリプレットの元の名前と同じです。

- `{ca_cert, cert, private_key}` トリプレットが設定されている場合は、ファイルはロギング設定の前にデフォルトのパスに配置されている必要があります。
- トリプレットの両方が設定されている場合には、ファイルはコントロールノードのローカルのパスから管理対象ノードの特定のパスへ転送されます。

ca_cert

CA 証明書へのパスを表します。デフォルトのパスは `/etc/pki/tls/certs/ca.pem` で、ファイル名はユーザーが設定します。

cert

証明書へのパスを表します。デフォルトのパスは `/etc/pki/tls/certs/server-cert.pem` で、ファイル名はユーザーが設定します。

private_key

秘密鍵へのパスを表します。デフォルトのパスは `/etc/pki/tls/private/server-key.pem` で、ファイル名はユーザーが設定します。

ca_cert_src

ローカルの CA 証明書ファイルパスを表します。これは管理対象ノードにコピーされます。**ca_cert** を指定している場合は、その場所にコピーされます。

cert_src

ローカルの証明書ファイルパスを表します。これは管理対象ノードにコピーされます。**cert** を指定している場合には、その証明書が場所にコピーされます。

private_key_src

ローカルキーファイルのパスを表します。これは管理対象ノードにコピーされます。**private_key** を指定している場合は、その場所にコピーされます。

pki_authmode

name または **fingerprint** の認証モードを使用できます。

permitted_servers

ロギングクライアントが、TLS 経由での接続およびログ送信を許可するサーバーのリスト。

inputs

ロギング入力ディクショナリーのリスト。

outputs

ロギング出力ディクショナリーのリスト。

Playbook で使用されるすべての変数の詳細は、コントロールノードの `/usr/share/ansible/roles/rhel-system-roles.logging/README.md` ファイルを参照してください。

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/logging/README.md` ファイル
- `/usr/share/doc/rhel-system-roles/logging/` ディレクトリー
- `rsyslog.conf(5)` および `syslog(3)` man ページ

15.4.2. RELP を使用したサーバーログの設定

logging RHEL システムロールを使用すると、ログメッセージを RELP を使用してリモートログインシステムから受信するようにサーバーを設定できます。

以下の手順では、Ansible インベントリーの **server** グループ内の全ホストに RELP を設定します。RELP 設定は TLS を使用して、メッセージ送信を暗号化し、ネットワーク経由でログを安全に転送します。

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: Configure server-side of the remote logging solution using RELP
  hosts: managed-node-01.example.com
  tasks:
    - name: Deploying remote input and remote_files output
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.logging
      vars:
        logging_inputs:
          - name: relp_server
            type: relp
            port: 20514
            tls: true
            ca_cert: /etc/pki/tls/certs/ca.pem
            cert: /etc/pki/tls/certs/server-cert.pem
            private_key: /etc/pki/tls/private/server-key.pem
            pki_authmode: name
            permitted_clients:
              - '*client.example.com'
        logging_outputs:
```

```

- name: remote_files_output
  type: remote_files
logging_flows:
- name: example_flow
  inputs: [relp_server]
  outputs: [remote_files_output]

```

サンプル Playbook で指定されている設定は次のとおりです。

port

リモートロギングシステムがリスンしているポート番号です。

tls

ネットワーク上でログをセキュアに転送します。セキュアなラッパーが必要ない場合は、**tls** 変数を **false** に設定します。デフォルトでは **tls** パラメーターは **true** に設定されますが、RELП を使用する場合には鍵/証明書およびトリプレット **{ca_cert、cert、private_key}** や **{ca_cert_src、cert_src、private_key_src}** が必要です。

- **{ca_cert_src、cert_src、private_key_src}** のトリプレットを設定すると、デフォルトの場所 (**/etc/pki/tls/certs** と **/etc/pki/tls/private**) が、コントロールノードからファイルを転送するため、管理対象ノードの宛先として使用されます。この場合、ファイル名はトリプレットの元の名前と同じです。
- **{ca_cert、cert、private_key}** トリプレットが設定されている場合は、ファイルはロギング設定の前にデフォルトのパスに配置されている必要があります。
- トリプレットの両方が設定されている場合には、ファイルはコントロールノードのローカルのパスから管理対象ノードの特定のパスへ転送されます。

ca_cert

CA 証明書へのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/ca.pem** で、ファイル名はユーザーが設定します。

cert

証明書へのパスを表します。デフォルトのパスは **/etc/pki/tls/certs/server-cert.pem** で、ファイル名はユーザーが設定します。

private_key

秘密鍵へのパスを表します。デフォルトのパスは **/etc/pki/tls/private/server-key.pem** で、ファイル名はユーザーが設定します。

ca_cert_src

ローカルの CA 証明書ファイルパスを表します。これは管理対象ノードにコピーされます。**ca_cert** を指定している場合は、その場所にコピーされます。

cert_src

ローカルの証明書ファイルパスを表します。これは管理対象ノードにコピーされます。**cert** を指定している場合には、その証明書が場所にコピーされます。

private_key_src

ローカルキーファイルのパスを表します。これは管理対象ノードにコピーされます。**private_key** を指定している場合は、その場所にコピーされます。

pki_authmode

name または **fingerprint** の認証モードを使用できます。

permitted_clients

ロギングサーバーが TLS 経由での接続およびログ送信を許可するクライアントのリスト。

inputs

ロギング入力ディクショナリーのリスト。

outputs

ロギング出力ディクショナリーのリスト。

Playbook で使用されるすべての変数の詳細は、コントロールノードの `/usr/share/ansible/roles/rhel-system-roles.logging/README.md` ファイルを参照してください。

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.logging/README.md` ファイル
- `/usr/share/doc/rhel-system-roles/logging/` ディレクトリー
- `rsyslog.conf(5)` および `syslog(3)` man ページ

15.5. 関連情報

- [RHEL システムロールを使用するためのコントロールノードと管理対象ノードの準備](#)
- `rhel-system-roles` パッケージでインストールされたドキュメントは、`/usr/share/ansible/roles/rhel-system-roles.logging/README.html` にあります。
- [RHEL システムロール](#)
- システムの `ansible-playbook(1)` の man ページ