Red Hat Enterprise Linux 8

セキュリティーの強化

Red Hat Enterprise Linux 8 システムのセキュリティーの強化

Last Updated: 2024-06-29

Red Hat Enterprise Linux 8 セキュリティーの強化

Red Hat Enterprise Linux 8 システムのセキュリティーの強化

法律上の通知

Copyright © 2024 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.

概要

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について学びます。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。

目次

RED HAT ドキュメントへのフィードバック (英語のみ)

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

Jira からのフィードバック送信 (アカウントが必要)

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

第1章 インストール時の RHEL の保護

セキュリティーへの対応は、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ーを設定することで、追加のセキュリティー設定を実装することがより簡単になります。

1.1. BIOS および UEFI のセキュリティー

BIOS (もしくは BIOS に相当するもの) およびブートローダーをパスワードで保護することで、システムに物理的にアクセス可能な未承認ユーザーがリムーバブルメディアを使用して起動したり、シングルユーザーモードで root 権限を取得することを防ぐことができます。このような攻撃から保護するためのセキュリティー対策は、ワークステーションに関する情報の機密性とマシンの場所の両方によって異なります。

たとえば、見本市で使用されていて機密情報を含んでいないマシンでは、このような攻撃を防ぐことが重要ではないかもしれません。しかし、同じ見本市で、企業ネットワークに対して暗号化されていない SSH 秘密鍵のある従業員のノートパソコンが、誰の監視下にもなく置かれていた場合は、重大なセキュリティー侵害につながり、その影響は企業全体に及ぶ可能性があります。

一方で、ワークステーションが権限のあるユーザーもしくは信頼できるユーザーのみがアクセスできる場所に置かれてるのであれば、BIOS もしくはブートローダーの安全確保は必要ない可能性もあります。

1.1.1. BIOS パスワード

コンピューターの BIOS をパスワードで保護する主な 2 つの理由を以下に示します。[1]:

BIOS 設定の変更の防止

侵入者が BIOS にアクセスできる場合、CD-ROM またはフラッシュドライブから起動するように BIOS を設定できます。このようにすると、侵入者がレスキューモードやシングルユーザーモードに入ることが可能になり、システムで任意のプロセスを開始したり、機密性の高いデータをコピーできるようになってしまいます。

システムの起動の防止

一部の BIOS では、ブートプロセスをパスワードで保護できます。これを有効にすると、攻撃者は BIOS がブートローダーを開始する前にパスワード入力を求められます。

BIOS パスワードの設定方法はコンピューターメーカーで異なるため、具体的な方法はコンピューターのマニュアルを参照してください。

BIOS パスワードを忘れた場合は、マザーボードのジャンパーでリセットするか、CMOS バッテリーを外します。このため、可能な場合はコンピューターのケースをロックすることが推奨されます。ただし、CMOS バッテリーを外す前にコンピューターもしくはマザーボードのマニュアルを参照してください。

1.1.2. 非 BIOS ベースシステムのセキュリティー

その他のシステムやアーキテクチャーでは、異なるプログラムを使用して x86 システムの BIOS とほぼ同等の低レベルのタスクを実行します。**UEFI** (**Unified Extensible Firmware Interface**) シェルなどがこの例になります。

BIOS のようなプログラムをパスワード保護する方法は、メーカーにお問い合わせください。

1.2. 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。Red Hat Enterprise Linux 8 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、Red Hat Enterprise Linux 8 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。

1.3. インストールプロセス時のネットワーク接続の制限

Red Hat Enterprise Linux 8 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、セキュリティー修正が最新のものではなく、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。

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のゾーンです。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。セキュリティーのベストプラクティスに従い、ネットワークから Red Hat Enterprise Linux 8 をインストールする場合は、お使いのリポジトリーに最も近いゾーンを選択するようにしてください。

1.4. 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。

1.5. インストール後の手順

以下は、Red Hat Enterprise Linux 8 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ー関連の手順です。

- 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。

```
# yum 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
```

- セキュリティーを強化するために、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] システム BIOS はメーカーによって異なるため、いずれかのタイプのパスワード保護のみをサポートするものもあれば、いずれのタイプのパスワード保護もサポートしないものもあります。

第2章 FIPS モードでのシステムのインストール

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 8 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。

2.1. 連邦情報処理標準 140 および FIPS モード

連邦情報処理標準 (FIPS) Publication 140 は、暗号モジュールの品質を確保するために米国立標準技術研究所 (NIST) によって開発された一連のコンピューターセキュリティー標準です。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。

RHEL システムが FIPS 承認アルゴリズムのみを使用してすべての暗号キーを生成および使用するようにするには、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、暗号鍵マテリアルの再生成と、すでにデプロイメントされているシステムの変換に関連する、結果として得られるシステムのコンプライアンスの再評価が回避されます。

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、安全にラップされ、非 FIPS 環境でラップ解除されない限り、FIPS 環境から決して出てはなりません。

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。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。

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。

注記

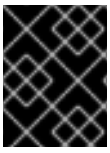
FIPS モードの RHEL 8.6 (およびそれ以降) カーネルは、FIPS 140-3 に準拠するように設計されていますが、まだ米国国立標準技術研究所 (NIST) 暗号モジュール検証プログラム (CMVP) によって認定されていません。最新の認定カーネルモジュールは、RHSA-2021:4356 アドバイザリー更新後の更新された RHEL 8.5 カーネルです。この認定は FIPS 140-2 標準に適用されます。暗号化モジュールが FIPS 140-2 または 140-3 のどちらに準拠するかを選択することはできません。Compliance Activities and Government Standards ナレッジベース記事の FIPS 140-2 and FIPS 140-3 セクションでは、選択した RHEL マイナーリリースにおける暗号化モジュールの検証ステータスの概要が説明されています。

関連情報

- ナレッジベース記事 Compliance Activities and Government Standards の FIPS 140-2 and FIPS 140-3 セクション
- RHEL system-wide cryptographic policies
- 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

- 高度な RHEL インストールの実行ドキュメントの 起動オプションの編集 セクション

2.3. 関連情報

- FIPS モードへのシステムの切り替え
- コンテナーでの FIPS モードの有効化
- List of RHEL 8 applications using cryptography that is not compliant with FIPS 140-2

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 8 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 5 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、セキュリティーが低下します。**DEFAULT** レベルでのアルゴリズムとプロトコルに加えて、TLS プロトコル 1.0 および 1.1 を許可します。アルゴリズム DSA、3DES、および RC4 が許可され、RSA 鍵と Diffie-Hellman パラメーターの長さが 1023 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ーレベル。このポリシーでは、署名アルゴリズムでの SHA-1 の使用は許可されません。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。

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。

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** パッケージをインストールすることです。認定済みの RHEL バージョン、検証レポート、および CC ガイドへのリンクのリストについては、ナレッジベース記事「Compliance Activities and Government Standards」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ー標準や新しいセキュリティー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。

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ので、現時点で **redhat-support-tool** ユーティリティーは、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。

注記

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。

暗号化ポリシーを管理するためのツール

現在のシステム全体の暗号化ポリシーを表示または変更するには、**update-crypto-policies** ツールを使用します。以下に例を示します。

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

暗号化ポリシーの変更を確実に適用するには、システムを再起動します。

安全でない暗号スイートとプロトコルを削除することによる強力な暗号デフォルト

以下の一覧には、RHEL のコア暗号化ライブラリーから削除された暗号スイートおよびプロトコルが含まれます。この暗号スイートおよびプロトコルは、ソースに存在しないか、ビルド時にそのサポートが無効になっているため、アプリケーションはこれらを使用できません。

- DES (RHEL 7 以降)
- すべてのエクスポートグレードの暗号化スイート (RHEL 7 以降)
- 署名内の MD5 (RHEL 7 以降)
- SSLv2 (RHEL 7 以降)
- SSLv3 (RHEL 8 以降)
- 224 ビットより小さいすべての ECC 曲線 (RHEL 6 以降)
- すべてのバイナリーフィールドの ECC 曲線 (RHEL 6 以降)

すべてのポリシーレベルで無効になっている暗号スイートおよびプロトコル

次の暗号スイートとプロトコルは、すべての暗号化ポリシーで無効になっています。これは、各アプリケーションで明示的に有効にした場合に限り利用可能にできます。

- パラメーターが 1024 ビットより小さい DH
- 鍵のサイズが 1024 ビットより小さい RSA
- Camellia
- ARIA
- SEED
- IDEA
- 完全性のみの暗号スイート
- SHA-384 HMAC を使用した TLS CBC モード暗号化スイート
- AES-CCM8
- TLS 1.3 と互換性がないすべての ECC 曲線 (secp256k1 を含む)
- IKEv1 (RHEL 8 以降)

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
各暗号化ポリシーでは、特定の暗号スイートとプロトコルが有効になっています。

	LEGACY	DEFAULT	FIPS	FUTURE
IKEv1	いいえ	いいえ	いいえ	いいえ
3DES	はい	いいえ	いいえ	いいえ
RC4	はい	いいえ	いいえ	いいえ
DH	最低 1024 ビット	最低 2048 ビット	最低 2048 ビット[a]	最低 3072 ビット
RSA	最低 1024 ビット	最低 2048 ビット	最低 2048 ビット	最低 3072 ビット
DSA	はい	いいえ	いいえ	いいえ
TLS v1.0	はい	いいえ	いいえ	いいえ
TLS v1.1	はい	いいえ	いいえ	いいえ
デジタル署名における SHA-1	はい	はい	いいえ	いいえ
CBC モード暗号	はい	はい	はい	いいえ[b]

	LEGACY	DEFAULT	FIPS	FUTURE
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
証明書における SHA-1 および SHA-224 の署名	はい	はい	はい	いいえ

[a] RFC 7919 および RFC 3526 で定義されている Diffie-Hellman グループのみを使用できます。

[b] TLS の CBC 暗号が無効になっています。TLS 以外のシナリオでは、**AES-128-CBC** は無効になっていますが、**AES-256-CBC** が有効になります。**AES-256-CBC** も無効にするには、カスタムのサブポリシーを適用します。

関連情報

- **update-crypto-policies(8)** の man ページ

3.2. 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

Red Hat Enterprise Linux 8 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

1. 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のレベルは、**update-crypto-policies(8)** の 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については、man ページの **update-crypto-policies (8)** の **Custom Policies** セクションと、**crypto-policies(7)** の **Crypto Policy Definition Format** セクションを参照してください。

3.3. 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、セキュリティーの低い設定を含む **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。

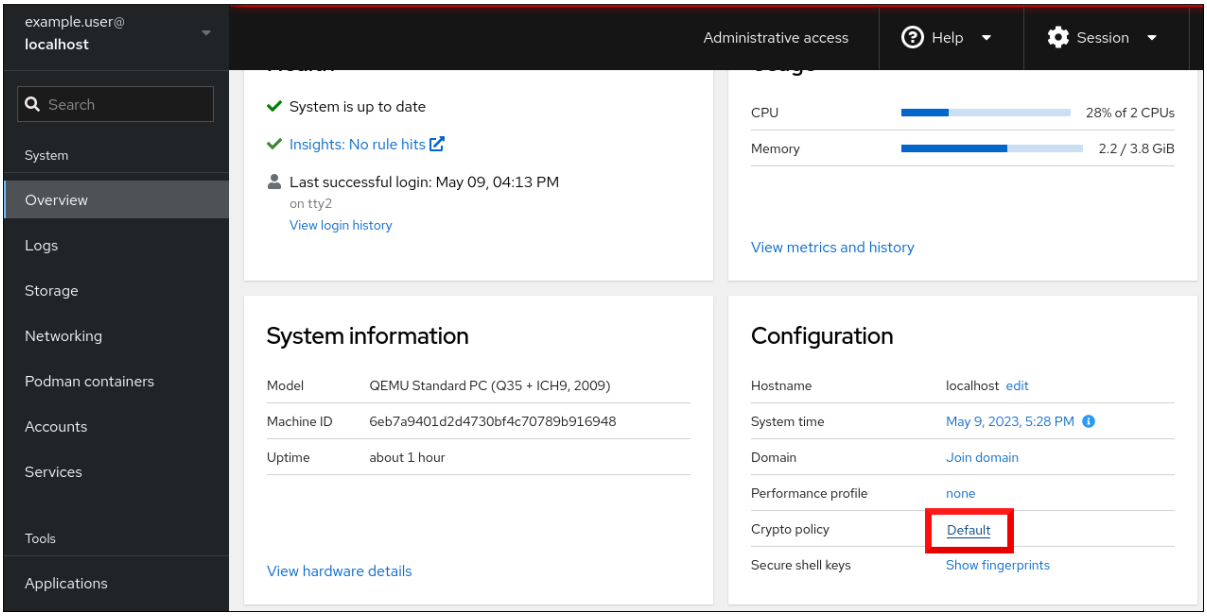
FIPS:OSPP 暗号化サブポリシーを設定すると、**システムが CC 非準拠になる** ことに注意してください。RHEL システムを CC 標準に準拠させる唯一の正しい方法は、**cc-config** パッケージをインストールすることです。認定済みの RHEL バージョン、検証レポート、および National Information Assurance Partnership (NIAP) で提供されている CC ガイドへのリンクのリストについては、ナレッジベース記事「Compliance Activities and Government Standards」の Common Criteria セクションを参照してください。

前提条件

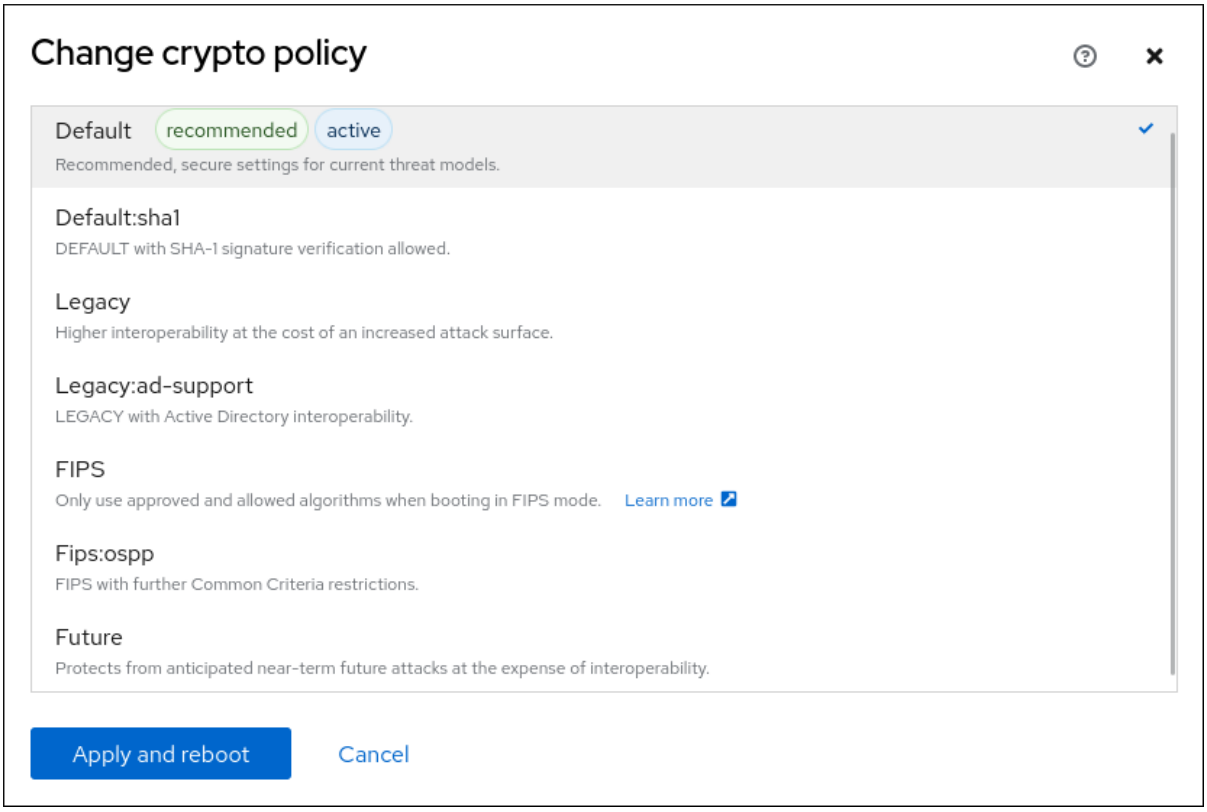
- RHEL 8 Web コンソールがインストールされている。詳細は、Web コンソールのインストールおよび有効化 を参照してください。
- **sudo** を使用して管理コマンドを入力するため の **root** 権限または権限がある。

手順

1. 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。

3.4. 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。

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。

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。

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

- **FIPS-mode-setup(8)** の man ページ
- FIPS モードが有効な RHEL 8 システムのインストール
- List of RHEL applications using cryptography that is not compliant with FIPS 140-2
- NIST (National Institute of Standards and Technology) の Web サイトの Security Requirements for Cryptographic Modules。

3.5. コンテナーでの 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。ホストシステムのバージョンに応じて、コンテナーで FIPS モードを有効にすることが完全に自動で行われるか、コマンドが 1 つだけ必要になります。

注記

コンテナーで **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

- **RHEL 8.1 および 8.2 を実行しているホストの場合**: 次のコマンドを使用してコンテナー内の FIPS 暗号化ポリシーレベルを設定し、**fips-mode-setup** コマンドを使用するというアドバイスは無視します。

```
$ update-crypto-policies --set FIPS
```

- **RHEL 8.4 以降を実行しているホスト**: 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ーで 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 モードが有効な RHEL 8 システムのインストール

3.6. LIST OF RHEL APPLICATIONS USING CRYPTOGRAPHY THAT IS NOT COMPLIANT WITH FIPS 140-2

FIPS 140 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ー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、RHEL コア crypto コンポーネント の記事を参照してください。

以下の表に加えて、一部の RHEL 8 Z-stream リリース (例: 8.1.1) では Firefox ブラウザーパッケージが更新され、別の NSS 暗号化ライブラリーが含まれています。このように、Red Hat では、パッチリリースでこのような詳細レベルのコンポーネントをリベースするなどの混乱を回避できればと考えています。そのため、この Firefox パッケージは FIPS 140-2 検証モジュールを使用しません。

FIPS 140-2 に準拠していない暗号化を使用する RHEL 8 アプリケーションのリスト

FreeRADIUS

MD5 を使用する RADIUS プロトコル。

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 cryptogtaphy 実装 (MD5、RC4、SHA-2、AES)

iPXE

TLS の暗号スタックはコンパイルされていますが、使用されません。

Libica

CPACF 命令から RSA や ECDH などのさまざまなアルゴリズムのソフトウェアフォールバック。

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

Pidgin

DES と RC4 を実装します。

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 など)。

Samba [2]

AES、DES、および RC4 を実装します。

Valgrind

AES、ハッシュ [3]

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

- ナレッジベース記事 コンプライアンス活動と政府標準 の FIPS 140-2 and FIPS 140-3 セクション
- ナレッジベース記事 RHEL core cryptographic components

3.7. システム全体の暗号化ポリシーに従わないようにアプリケーションを除外

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。

3.7.1. システム全体の暗号化ポリシーを除外する例

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 サーバーに対するシステム全体の暗号化ポリシーを除外するには、**/etc/sysconfig/sshd** ファイルの **CRYPTO_POLICY=** 変数行のコメントを除外します。この変更後、**/etc/ssh/sshd_config** ファイルの **Ciphers** セクション、**MACs** セクション、**KexAlgoritms** セクション、および **GSSAPIKexAlgorithms** セクションで指定した値は上書きされません。

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 **05-redhat.conf** ファイルよりも前に来るように、5 未満の 2 桁の接頭辞と、**.conf** という接尾辞を付けます (例: **04-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、ネットワークのセキュリティー保護 の システム全体の暗号化ポリシーをオプトアウトする IPsec 接続の設定 を参照してください。

関連情報

- **update-crypto-policies(8)** の man ページ

3.8. 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。

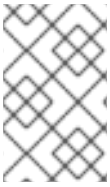
既存のシステム全体の暗号化ポリシーの上にカスタムサブポリシーを適用するか、そのようなポリシーを最初から定義することができます。

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。



注記

システム全体の暗号化ポリシーのカスタマイズは、RHEL 8.2 から利用できます。スコープ指定ポリシーの概念と、RHEL 8.5 以降でワイルドカードを使用するオプションを使用できます。

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

- **update-crypto-policies(8)** man ページの **Custom Policies** セクション
- **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

3.9. システム全体の暗号化ポリシーをカスタマイズして SHA-1 を無効化

SHA-1 ハッシュ関数は本質的に設計面で弱く、暗号解析機能によって攻撃を受けやすいため、RHEL 8 ではデフォルトで SHA-1 を使用していません。ただし、一部のサードパーティーアプリケーション (公

開署名など) は SHA-1 を使用します。システムの署名アルゴリズムで SHA-1 の使用を無効にするには、**NO-SHA1** ポリシーモジュールを使用できます。

重要

NO-SHA1 ポリシーモジュールが無効にするのは、署名の SHA-1 ハッシュ関数だけでそれ以外は無効になりません。特に、**NO-SHA1** モジュールでも、ハッシュベースのメッセージ認証コード (HMAC) とともに SHA-1 を使用できます。HMAC セキュリティープロパティーは、対応するハッシュ関数の衝突耐性に依存しないので、HMAC に SHA-1 を使用している場合に最近の SHA-1 への攻撃による影響は大幅に低くなっています。

特定の鍵交換 (KEX) アルゴリズムの組み合わせ (例: **diffie-hellman-group-exchange-sha1**) を無効にする必要があるものの、関連する KEX とアルゴリズムの両方を継続して使用する場合は、SSH のシステム全体の暗号化ポリシーをオプトアウトして SSH を直接設定する手順について、SSH で diffie-hellman-group1-sha1 アルゴリズムを無効にする手順 を参照してください。

注記

SHA-1 を無効にするモジュールは、RHEL 8.3 で利用できます。システム全体の暗号化ポリシーのカスタマイズは、RHEL 8.2 から利用できます。

手順

1. 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:NO-SHA1
```

2. 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

- **update-crypto-policies(8)** man ページの **Custom Policies** セクション
- **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

3.10. 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

以下の手順は、完全なポリシーファイルでシステム全体の暗号化ポリシーをカスタマイズする方法を示しています。

注記

システム全体の暗号化ポリシーのカスタマイズは、RHEL 8.2 から利用できます。

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

- **update-crypto-policies (8)** の man ページの **Custom Policies** セクションと、**crypto-policies(7)** の **Crypto Policy Definition Format** セクションを参照してください。
- Red Hat ブログ記事 How to customize crypto policies in RHEL

3.11. 関連情報

- ナレッジベースの記事 System-wide crypto policies in RHEL 8 および Strong crypto defaults in RHEL 8 and deprecation of weak crypto algorithms

[2] RHEL 8.3 以降、samba は FIPS 準拠の暗号を使用します。

[3] AES-NI などのソフトウェア/ハードウェアオフロード操作に再実装します。

第4章 RHEL システムロールを使用してカスタム暗号化ポリシーを設定する

管理者は、**crypto_policies** RHEL システムロールを使用して、Ansible Core パッケージを使用し、さまざまなシステムのカスタム暗号化ポリシーを迅速かつ一貫して設定できます。

4.1. CRYPTO_POLICIES RHEL システムロールを使用したカスタム暗号化ポリシーの設定

crypto_policies システムロールを使用して、単一のコントロールノードから多数の管理対象ノードを一貫して設定できます。

前提条件

- 制御ノードと管理ノードを準備している
- 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 crypto policies
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure crypto policies
      ansible.builtin.include_role:
        name: rhel-system-roles.crypto_policies
      vars:
        - crypto_policies_policy: FUTURE
        - crypto_policies_reboot_ok: true
```

FUTURE の値は、任意の暗号化ポリシー (例: **DEFAULT**、**LEGACY**、および **FIPS:OSPP**) に置き換えることができます。

crypto_policies_reboot_ok: true を設定すると、システムロールが暗号化ポリシーを変更した後にシステムが再起動します。

2. Playbook の構文を検証します。

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

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

3. Playbook を実行します。

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

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** パッケージをインストールすることです。認定済みの RHEL バージョン、検証レポート、および National Information Assurance Partnership (NIAP) で提供されている CC ガイドへのリンクのリストについては、ナレッジベース記事「Compliance Activities and Government Standards」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 crypto policy
      ansible.builtin.include_role:
        name: rhel-system-roles.crypto_policies
    - debug:
        var: crypto_policies_active
```

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、管理対象ノードで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

第5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ーモジュール (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、セキュリティー層が追加されます。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。

5.1. 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ーモジュールを 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ー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

- Consistent PKCS #11 support in Red Hat Enterprise Linux 8
- The PKCS #11 URI Scheme
- Controlling access to smart cards

5.2. スマートカードに保存された SSH 鍵の使用

Red Hat Enterprise Linux では、OpenSSH クライアントでスマートカードに保存されている RSA 鍵および ECDSA 鍵を使用できるようになりました。この手順に従って、パスワードの代わりにスマートカードを使用した認証を有効にします。

前提条件

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

手順

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

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

2. リモートサーバー (**example.com**) でスマートカードを使用した認証を有効にするには、公開鍵をリモートサーバーに転送します。前の手順で作成された **keys.pub** で **ssh-copy-id** コマンドを使用します。

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

3. 手順 1 の **ssh-keygen -D** コマンドの出力にある ECDSA 鍵を使用して **example.com** に接続するには、鍵を一意に参照する URI のサブセットのみを使用できます。以下に例を示します。

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

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

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

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

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

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

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

関連情報

- Fedora 28: Better smart card support in OpenSSH
- **p11-kit(8)**、**opensc.conf(5)**、**pcscd(8)**、**ssh(1)**、および **ssh-keygen(1)** の man ページ

5.3. 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、セキュリティーが強化され、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/
  ```

関連情報

- **curl(1)**、**wget(1)**、および **p11-kit(8)** man ページ

5.4. 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ーモジュール (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。

5.5. 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ーモジュール (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** エンジンを介し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。

5.6. 関連情報

- **pkcs11.conf(5)** の man ページ

第6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。

6.1. 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ー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、その構文は man ページの **polkit(8)** 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。

関連情報

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

6.2. 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でき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 デー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 libpcsclite.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)**

6.3. 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)**

6.4. 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7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ー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。

7.1. 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ー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。詳細については、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ー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、セキュリティー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ー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

- **oscap(8)**、**scap-workbench(8)**、および **scap-security-guide(8)** の 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 の管理ガイドのセキュリティーコンプライアンスの管理

7.2. 脆弱性スキャン

7.2.1. Red Hat Security Advisories OVAL フィード

Red Hat Enterprise Linux のセキュリティー監査機能は、標準規格セキュリティー設定共通化手順 (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ー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、セキュリティー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ーベンダーの数に関係なく、セキュリティーポリシー (SCAP コンテンツ) を構築するのは一度で済みます。

セキュリティー検査言語 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。

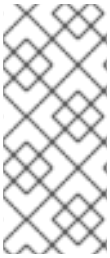
Red Hat 製品セキュリティー を使用すると、Red Hat 製品をお使いのお客様に影響を及ぼすセキュリティー問題をすべて追跡して調査します。Red Hat カスタマーポータルで簡潔なパッチやセキュリティーアドバイザリーを適時提供します。Red Hat は OVAL パッチ定義を作成してサポートし、マシンが判読可能なセキュリティーアドバイザリーを提供します。

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 製品セキュリティー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 (CHC) のベースライン評価と完全に一致しているわけではありません。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ーアドバイザリーが 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 セキュリティーアドバイザリー (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ー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
- 製品セキュリティーの概要 の 通知およびアドバイザリー
- Security Data Metrics

7.2.2. 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/RHEL8/rhel-8.oval.xml.bz2 | bzip2 --decompress > rhel-8.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-8.oval.xml
```

検証

- 結果をブラウザーで確認します。以下に例を示します。

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

関連情報

- **oscap(8)** の man ページ
- Red Hat OVAL 定義
- OpenSCAP のメモリー消費の問題

7.2.3. 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確認することもできます。

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/RHEL8/rhel-8.oval.xml.bz2 | bzip2 --decompress > rhel-8.oval.xml
```

2. 脆弱性に対して、ホスト名 **machine1**、ポート 22 で実行する SSH、およびユーザー名 **joesec** でリモートシステムをスキャンし、結果を **remote-vulnerability.html** ファイルに保存します。

```
# oscap-ssh joesec@machine1 22 oval eval --report remote-vulnerability.html rhel-8.oval.xml
```

関連情報

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

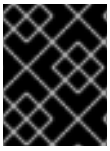
7.3. 設定コンプライアンススキャン

7.3.1. 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ー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 セキュリティーガイドスイートは、データストリームのドキュメント形式で、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、セキュリティー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

```
Data stream
   ├── xccdf
   |     ├── benchmark
   |           ├── profile
   |           |    ├──rule reference
   |           |    └──variable
   |           ├── rule
   |                ├── human readable data
   |                ├── oval reference
   ├── oval         ├── ocil reference
   ├── ocil         ├── cpe reference
   └── cpe          └── remediation
```

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ーポリシーに基づく一連のルールです。これにより、セキュリティー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ープロファイルのカスタマイズを参照してください。

7.3.2. 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ーに応じて、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。バグレポート を作成することを検討してください。

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。バグレポート を作成することを検討してください。

7.3.3. 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ー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-firefox-cpe-dictionary.xml  ssg-rhel6-ocil.xml
ssg-firefox-cpe-oval.xml        ssg-rhel6-oval.xml
...
ssg-rhel6-ds-1.2.xml          ssg-rhel8-oval.xml
ssg-rhel8-ds.xml              ssg-rhel8-xccdf.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-rhel8-ds.xml
Profiles:
...
  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 8
    Id: xccdf_org.ssgproject.content_profile_pci-dss
  Title: OSPP - Protection Profile for General Purpose Operating Systems
    Id: xccdf_org.ssgproject.content_profile_ospp
...
```

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、HIPPA 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-rhel8-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

- **scap-security-guide (8)** の man ページ
- OpenSCAP のメモリー消費の問題

7.3.4. 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

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確認するには、次の手順に従います。

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 を参照してください。

手順

1. 選択したプロファイルでそのシステムがどのように複雑であるかを評価し、スキャン内容を保存すると、以下のように HTML ファイル (**report.html**) に結果が表示されます。

```
$ oscap xccdf eval --report report.html --profile hipaa /usr/share/xml/scap/ssg/content/ssg-
rhel8-ds.xml
```

2. 必要に応じて、コンプライアンスに対して、ホスト名 **machine1**、ポート **22** で実行する SSH、およびユーザー名 **joesec** でリモートシステムをスキャンし、結果を **remote-vulnerability.html** ファイルに保存します。

```
$ oscap-ssh joesec@machine1 22 xccdf eval --report remote_report.html --profile hipaa
/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

関連情報

- **scap-security-guide (8)** の 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-rhel8-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 8 のセキュアな設定ガイド
- OpenSCAP のメモリー消費の問題

7.4. 特定のベースラインに合わせたシステムの修復

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。この例では Health Insurance Portability and Accountability Act (HIPAA) プロファイルを使用していますが、SCAP Security Guide で提供されている他のプロファイルに合わせて修正することもできます。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

- 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

1. **oscap** コマンドに **--remediate** オプションを指定して使用します。

```
# oscap xccdf eval --profile hipaa --remediate /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

2. システムを再起動します。

検証

1. システムの OSPP プロファイルへのコンプライアンスを評価し、スキャン結果を **ospp_report.html** ファイルに保存します。

```
$ oscap xccdf eval --report hipaa_report.html --profile hipaa /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

関連情報

- **scap-security-guide(8)** および **oscap(8)** の man ページ
- ナレッジベースの記事 Complementing the DISA benchmark using the SSG content

7.5. SSG ANSIBLE PLAYBOOK 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する

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。この例では Health Insurance Portability and Accountability Act

(HIPAA) プロファイルを使用していますが、SCAP Security Guide で提供されている他のプロファイルに合わせて修正することもできます。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。
- **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。

注記

RHEL 8.6 以降では、Ansible Engine は、ビルトイン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、Ansible 修復の代わりに Bash 修復を使用できます。RHEL 8 の Red Hat Connector には、Ansible Core で修復 Playbook を機能させるために必要な Ansible モジュールが含まれています。

手順

1. Ansible を使用して OSPP に合わせてシステムを修正します。

```
# ansible-playbook -i localhost, -c local /usr/share/scap-security-guide/ansible/rhel8-playbook-hipaa.yml
```

2. システムを再起動します。

検証

1. システムの OSPP プロファイルへのコンプライアンスを評価し、スキャン結果を **ospp_report.html** ファイルに保存します。

```
# oscap xccdf eval --profile hipaa --report hipaa_report.html /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

関連情報

- **scap-security-guide(8)** および **oscap(8)** の man ページ
- Ansible ドキュメント

7.6. 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

以下の手順に従って、必要な修復のみを含む Ansible Playbook を作成し、システムを特定のベースラインに合わせます。この例では、Health Insurance Portability and Accountability Act (HIPAA) プロファイルを使用します。この手順では、要件を満たしていない小規模の Playbook を作成します。以下の手順に従うと、システムを変更せずに、後のアプリケーション用にファイルの準備を行うだけです。

注記

RHEL 8.6 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 8.6 の Red Hat Connector には、Ansible Core で修復 Playbook を機能させるために必要な 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。

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 hipaa --results <hipaa-results.xml>
/usr/share/xml/scap/ssg/content/ssg-rhel8-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 <hipaa-results.xml>
```

3. 手順 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_hipaa> --output <hipaa-
remediations.yml> <hipaa-results.xml>
```

4. 生成されたファイルを確認します。これには、手順 1 で実行されたスキャン中に失敗したルールの Ansible 修復が含まれています。この生成されたファイルを確認した後、**ansible-playbook <hipaa-remediations.yml>** コマンドを使用して適用できます。

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、生成した **<hipaa-remediations.yml>** ファイルに含まれていることを確認します。

関連情報

- **scap-security-guide(8)** および **oscap(8)** の man ページ
- Ansible ドキュメント

7.7. 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

1. **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-rhel8-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 <hipaa-results.xml>
```

3. 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>
```

4. **<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ー内で、**./<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

- **scap-security-guide(8)**、**oscap(8)**、および **bash(1)** の man ページ

7.8. 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ーで、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ーとの比較は、**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。

7.8.1. 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

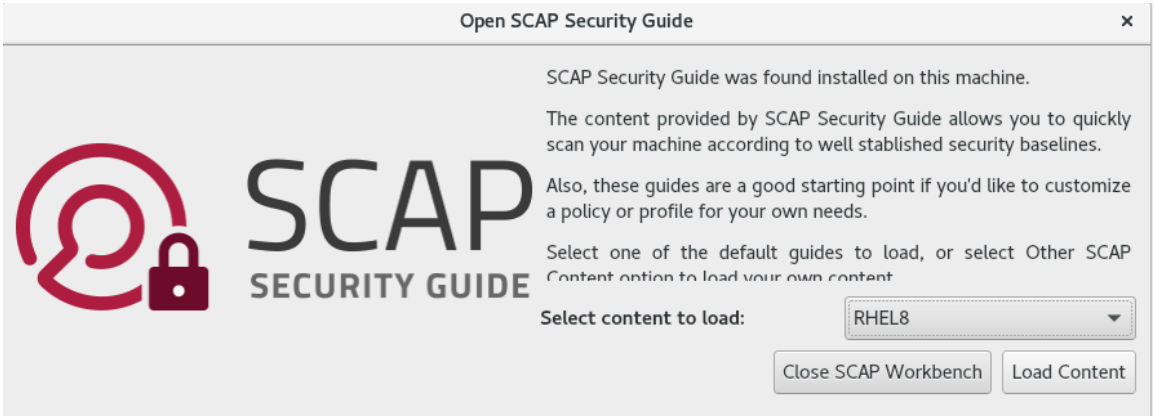
- **scap-workbench** パッケージがシステムにインストールされている。

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。

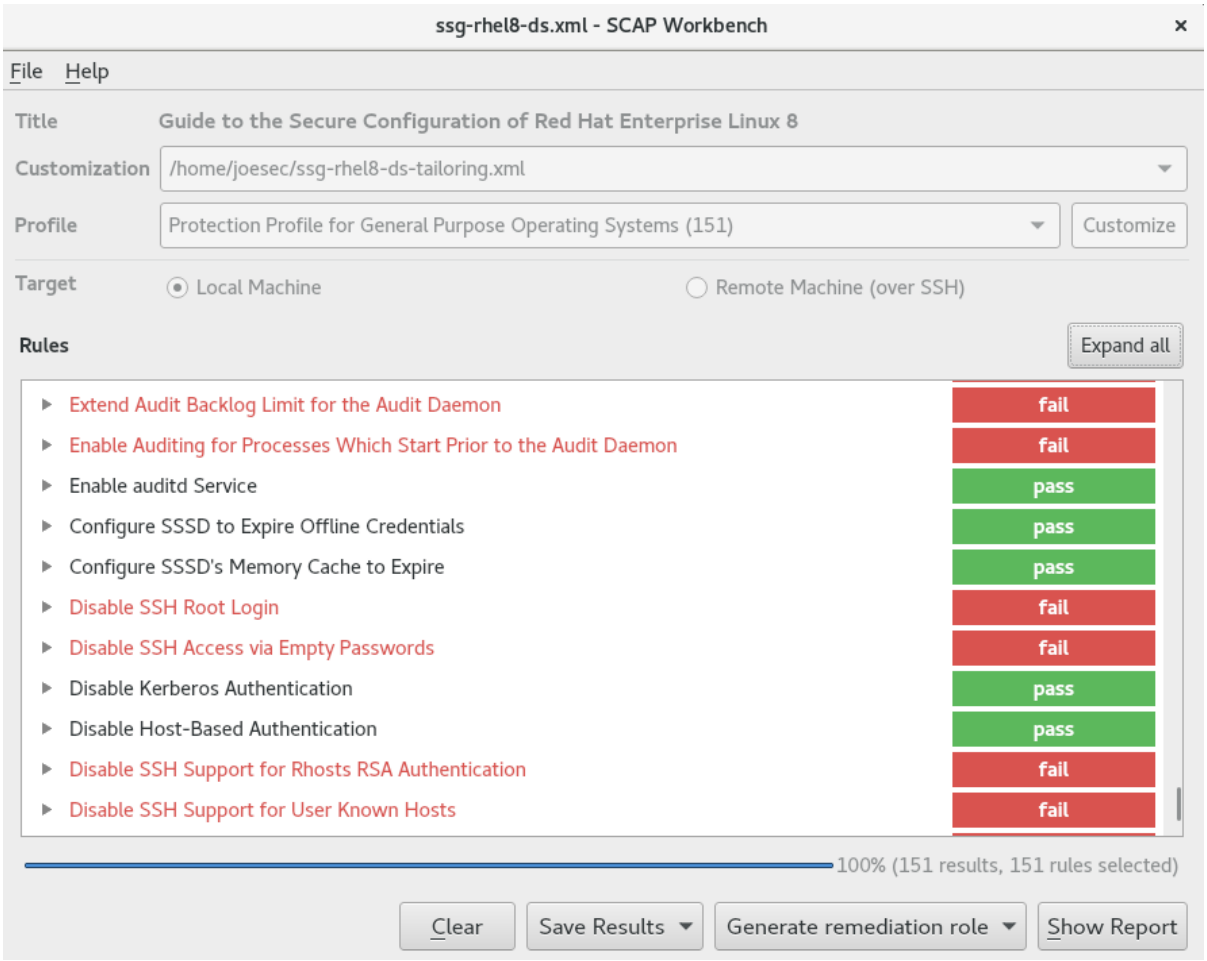


3. **Remediate** チェックボックスを選択して、システム設定の自動修正を行うことができます。このオプションを有効にすると、**SCAP Workbench** は、ポリシーにより適用されるセキュリティールールに従ってシステム設定の変更を試みます。このプロセスは、システムスキャン時に失敗した関連チェックを修正する必要があります。

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。



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。

7.8.2. 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を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。プロファイルを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をカスタマイズ (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に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。

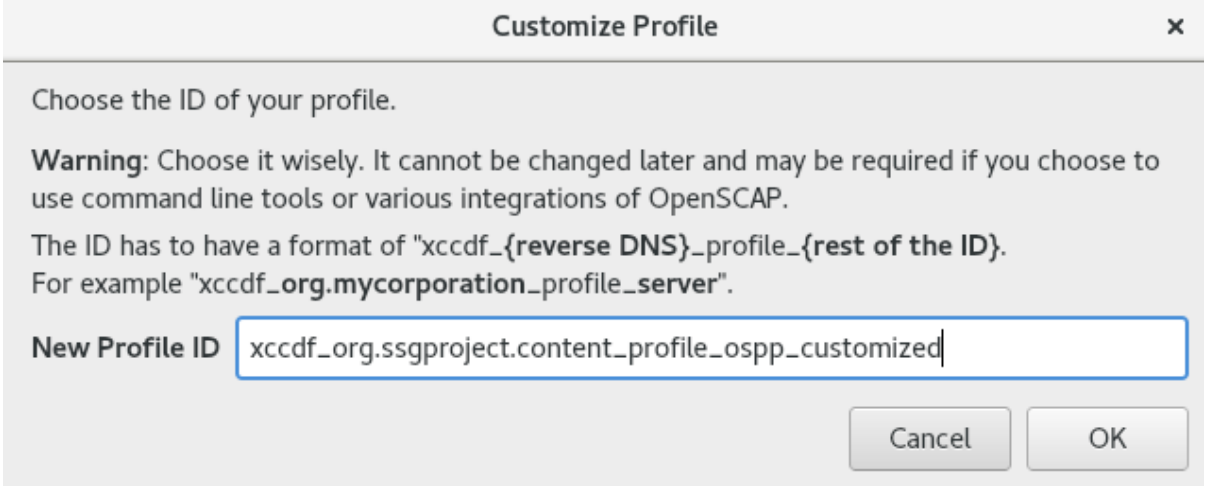
前提条件

- **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

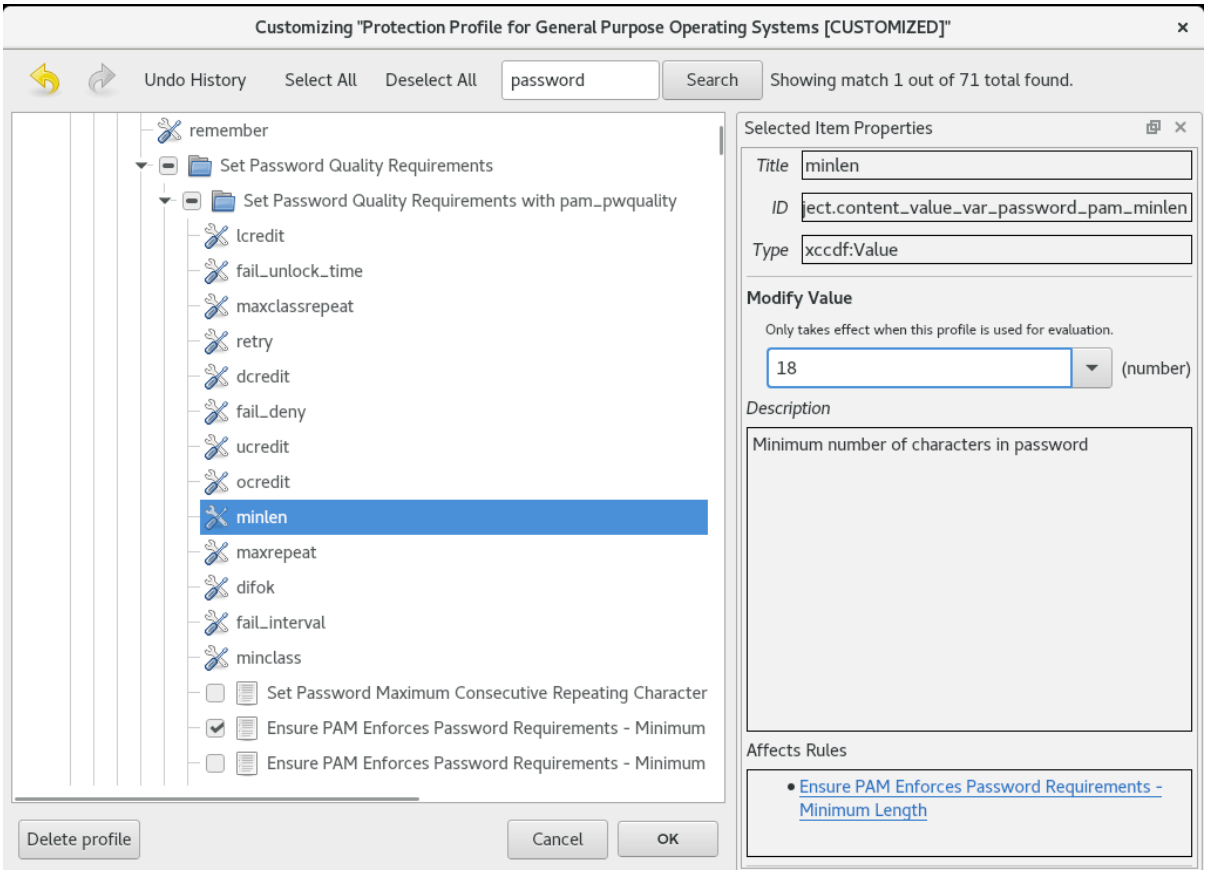
1. **SCAP Workbench** を実行し、**Open content from SCAP Security Guide** または **File** メニューの **Open Other Content** を使用してカスタマイズするプロファイルを選択します。

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。



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。

注記

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。

7.8.3. 関連情報

- **scap-workbench (8)** の 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
- カスタマイズされた SCAP ポリシーを Satellite 6.x KCS でデプロイする 記事

7.9. 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。

7.9.1. GUI を備えたサーバーと互換性のないプロファイル

SCAP セキュリティーガイド 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** ベースの環境の拡張パッケージセットと互換性がない場合があります。したがって、次のプロファイルのいずれか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7.1 GUI を備えたサーバー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	備考

プロファイル名	プロファイル ID	理由	備考
CIS Red Hat Enterprise Linux 8 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**、および **xorg-x11-server-Xwayland** は、Server with GUIパッケージセットの一部ですが、ポリシーではそれらを削除する必要があります。	
CIS Red Hat Enterprise Linux 8 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**、および **xorg-x11-server-Xwayland** は、Server with GUIパッケージセットの一部ですが、ポリシーではそれらを削除する必要があります。	
Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	**xccdf_org.ssgproject.content_profile_cui**	**nfs-utils** パッケージはServer with GUIパッケージセットの一部ですが、ポリシーではその削除が必要です。	
Protection Profile for General Purpose Operating Systems	**xccdf_org.ssgproject.content_profile_ospp**	**nfs-utils** パッケージはServer with GUIパッケージセットの一部ですが、ポリシーではその削除が必要です。	
DISA STIG for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	パッケージ **xorg-x11-server-Xorg**、**xorg-x11-server-common**、**xorg-x11-server-utils**、および **xorg-x11-server-Xwayland** は、Server with GUIパッケージセットの一部ですが、ポリシーではそれらを削除する必要があります。	RHEL バージョン 8.4 以降で、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。

7.9.2. 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

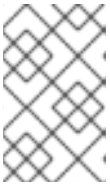
SCAP セキュリティーガイド の一部として提供される一部のセキュリティープロファイルは、**Server with GUI** ベースの環境の拡張パッケージセットと互換性がない場合があります。詳細は、GUI サーバーと互換性のないプロファイル を参照してください。

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. **セキュリティーポリシー** をクリックします。**セキュリティーポリシー** 画面が開きます。
6. システムでセキュリティーポリシーを有効にするには、**セキュリティーポリシーの適用** 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 **Protection Profile for General Purpose Operating Systems** 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-rhel8-ds.xml
```

関連情報

- 手動パーティションの設定

7.9.3. 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

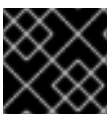
この手順を使用して、特定のベースラインに合わせた RHEL システムをデプロイします。この例では、OSPP (Protection Profile for General Purpose Operating System) を使用します。

前提条件

- RHEL 8 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-rhel8-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. キックスタートインストールを開始する方法は、キックスタートインストールの開始を参照してください。



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

1. 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。

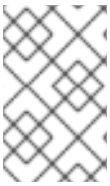
```
# oscap xccdf eval --profile ospp --report eval_postinstall_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

関連情報

- OSCAP Anaconda Add-on

7.10. コンテナーおよびコンテナーイメージの脆弱性スキャン

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



注記

oscap-podman コマンドは、RHEL 8.2 で利用できます。RHEL 8.1 および 8.0 の場合は、ナレッジベースの記事 Using OpenSCAP for scanning containers in RHEL 8 で説明されている回避策を利用します。

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/RHEL8/rhel-8.oval.xml.bz2 | bzip2 --decompress > rhel-8.oval.xml
```

2. コンテナーまたはコンテナーイメージの 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/ubi8/ubi     latest   096cae65a207   7 weeks ago   239 MB
```

3. コンテナーまたはコンテナー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-8.oval.xml
```

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

検証

- 結果をブラウザーで確認します。以下に例を示します。

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

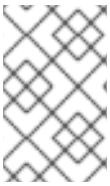
関連情報

- 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。

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

以下の手順に従い、OSPP (Operating System Protection Profile) や PCI-DSS (Payment Card Industry Data Security Standard)、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のあるコンテナーまたはコンテナーイメージのコンプライアンスを評価します。



注記

oscap-podman コマンドは、RHEL 8.2 で利用できます。RHEL 8.1 および 8.0 の場合は、ナレッジベースの記事 Using OpenSCAP for scanning containers in RHEL 8 で説明されている回避策を利用します。

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。

手順

1. コンテナーまたはコンテナーイメージの 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/ubi8/ubi   latest   096cae65a207   7 weeks ago   239 MB
```

2. HIPAA プロファイルでコンテナーイメージのコンプライアンスを評価し、スキャン結果を **report.html** ファイルに保存します。

```
# oscap-podman 096cae65a207 xccdf eval --report report.html --profile hipaa
/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

OSPP または PCI-DSS ベースラインでセキュリティーコンプライアンスを評価する場合は、**096cae65a207** をコンテナーイメージの ID に、**hipaa** の値を **ospp** または **pci-dss** に置き換えます。**oscap-podman** コマンドには、root 権限が必要なことに注意してください。

検証

- 結果をブラウザーで確認します。以下に例を示します。

```
$ firefox report.html &
```

注記

notapplicable が付いているルールは、コンテナー化されたシステムには適用されないルールです。これらのルールは、ベアメタルおよび仮想化システムにのみ適用されます。

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。

7.12. RHEL 8 で対応している SCAP セキュリティーガイドプロファイル

RHEL の特定のマイナーリリースで提供される SCAP コンテンツのみを使用します。これは、ハードニングに参加するコンポーネントが新機能で更新されるためです。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、常に後方互換性があるわけではありません。

以下の表は、RHEL の各マイナーバージョン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検出できます。

表7.2 RHEL 9.0 でサポートされる SCAP セキュリティーガイドプロファイル

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
CIS Red Hat Enterprise Linux 8 Benchmark for Level 2 - Server	**xccdf_org.ssgproject.content_profile_cis**	3.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 1 - Server	**xccdf_org.ssgproject.content_profile_cis_server_l1**	3.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 1 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l1**	3.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 2 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l2**	3.0.0
Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	**xccdf_org.ssgproject.content_profile_cui**	r1
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

プロファイル名	プロファイル ID	ポリシーバージョン
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 8	**xccdf_org.ssgproject.content_profile_pci-dss**	4.0
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	RHEL 8.10.0:V1R13 RHEL 8.10.1 以降:V1R14
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig_gui**	RHEL 8.10.0:V1R13 RHEL 8.10.1 以降:V1R14

表7.3 RHEL 9.0 でサポートされる SCAP セキュリティーガイドプロファイル

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
CIS Red Hat Enterprise Linux 8 Benchmark for Level 2 - Server	**xccdf_org.ssgproject.content_profile_cis**	RHEL 8.9.0 および RHEL 8.9.2:2.0.0 RHEL 8.9.3:3.0.0

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 8 Benchmark for Level 1 - Server	**xccdf_org.ssgproject.content_profile_cis_server_l1**	RHEL 8.9.0 および RHEL 8.9.2:2.0.0 RHEL 8.9.3:3.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 1 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l1**	RHEL 8.9.0 および RHEL 8.9.2:2.0.0 RHEL 8.9.3:3.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 2 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l2**	RHEL 8.9.0 および RHEL 8.9.2:2.0.0 RHEL 8.9.3:3.0.0
Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	**xccdf_org.ssgproject.content_profile_cui**	r1
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 8	**xccdf_org.ssgproject.content_profile_pci-dss**	RHEL 8.9.0 および RHEL 8.9.2:3.2.1 RHEL 8.9.3:4.0
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	RHEL 8.9.0 および RHEL 8.9.2:V1R11 RHEL 8.9.3:V1R13
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig_gui**	RHEL 8.9.0 および RHEL 8.9.2:V1R11 RHEL 8.9.3:V1R13

表7.4 RHEL 8.8 で対応する SCAP セキュリティーガイドプロファイル

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
CIS Red Hat Enterprise Linux 8 Benchmark for Level 2 - Server	**xccdf_org.ssgproject.content_profile_cis**	RHEL 8.8.0 および RHEL 8.8.5:2.0.0 RHEL 8.8.6:3.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 1 - Server	**xccdf_org.ssgproject.content_profile_cis_server_l1**	RHEL 8.8.0 および RHEL 8.8.5:2.0.0 RHEL 8.8.6:3.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 1 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l1**	RHEL 8.8.0 および RHEL 8.8.5:2.0.0 RHEL 8.8.6:3.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 2 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l2**	RHEL 8.8.0 および RHEL 8.8.5:2.0.0 RHEL 8.8.6:3.0.0
Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	**xccdf_org.ssgproject.content_profile_cui**	r1
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

プロファイル名	プロファイル ID	ポリシーバージョン
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_pci-dss**	RHEL 8.8.0 および RHEL 8.8.5:3.2.1 RHEL 8.8.6:4.0
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	RHEL 8.8.0 および RHEL 8.8.5:V1R9 RHEL 8.8.6 および RHEL 8.8.7:V1R13 RHEL 8.8.8 以降:V1R14
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig_gui**	RHEL 8.8.0 および RHEL 8.8.5:V1R9 RHEL 8.8.6 および RHEL 8.8.7:V1R13 RHEL 8.8.8 以降:V1R14

表7.5 RHEL 8.7 で対応する SCAP セキュリティーガイドプロファイル

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 8 Benchmark for Level 2 - Server	**xccdf_org.ssgproject.content_profile_cis**	2.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 1 - Server	**xccdf_org.ssgproject.content_profile_cis_server_l1**	2.0.0
CIS Red Hat Enterprise Linux 8 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 8 Benchmark for Level 2 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l2**	2.0.0
Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	**xccdf_org.ssgproject.content_profile_cui**	r1
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 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	RHEL 8.7.0 および RHEL 8.7.1: V1R7 RHEL 8.7.2 以降: V1R9
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig_gui**	RHEL 8.7.0 および RHEL 8.7.1: V1R7 RHEL 8.7.2 以降: V1R9

表7.6 RHEL 8.6 で対応する SCAP セキュリティーガイドプロファイル

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 8.6.0 から 8.6.10: 1.2 RHEL 8.6.11 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 8.6.0 から 8.6.10: 1.2 RHEL 8.6.11 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
French National Agency for the Security of Information Systems (ANSSI) BP-028 Intermediary Level	**xccdf_org.ssgproject.content_profile_anssi_bp28_intermediary**	RHEL 8.6.0 から 8.6.10: 1.2 RHEL 8.6.11 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 8.6.0 から 8.6.10: 1.2 RHEL 8.6.11 以降: 2.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 2 - Server	**xccdf_org.ssgproject.content_profile_cis**	RHEL 8.6.0 から RHEL 8.6.2:1.0.0 RHEL 8.6.3 から RHEL 8.6.15:2.0.0 RHEL 8.6.16 以降:3.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 1 - Server	**xccdf_org.ssgproject.content_profile_cis_server_l1**	RHEL 8.6.0 から RHEL 8.6.2:1.0.0 RHEL 8.6.3 から RHEL 8.6.15:2.0.0 RHEL 8.6.16 以降:3.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 1 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l1**	RHEL 8.6.0 から RHEL 8.6.2:1.0.0 RHEL 8.6.3 から RHEL 8.6.15:2.0.0 RHEL 8.6.16 以降:3.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 2 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l2**	RHEL 8.6.0 から RHEL 8.6.2:1.0.0 RHEL 8.6.3 から RHEL 8.6.15:2.0.0 RHEL 8.6.16 以降:3.0.0
Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	**xccdf_org.ssgproject.content_profile_cui**	r1
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 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1

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) for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	RHEL 8.6.0:V1R5 RHEL 8.6.1 および RHEL 8.6.2:V1R6 RHEL 8.6.3 から RHEL 8.6.6:V1R7 RHEL 8.6.7 から RHEL 8.6.10:V1R9 RHEL 8.6.11 から RHEL 8.6.15:V1R11 へ RHEL 8.6.16 および RHEL 8.6.17:V1R13 RHEL 8.6.18 以降:V1R14
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig_gui**	RHEL 8.6.0:V1R5 RHEL 8.6.1 および RHEL 8.6.2:V1R6 RHEL 8.6.3 から RHEL 8.6.6:V1R7 RHEL 8.6.7 から RHEL 8.6.10:V1R9 RHEL 8.6.11 から RHEL 8.6.15:V1R11 へ RHEL 8.6.16 および RHEL 8.6.17:V1R13 RHEL 8.6.18 以降:V1R14

表7.7 RHEL 8.5 で対応する SCAP セキュリティーガイドプロファイル

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 8 Benchmark for Level 2 - Server	**xccdf_org.ssgproject.content_profile_cis**	1.0.0

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 8 Benchmark for Level 1 - Server	**xccdf_org.ssgproject.content_profile_cis_server_l1**	1.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 1 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l1**	1.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 2 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l2**	1.0.0
Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	**xccdf_org.ssgproject.content_profile_cui**	r1
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 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	RHEL 8.5.0 から RHEL 8.5.3: V1R3 RHEL 8.5.4 以降: V1R5
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig_gui**	RHEL 8.5.0 から RHEL 8.5.3: V1R3 RHEL 8.5.4 以降: V1R5

表7.8 RHEL 8.4 での SCAP セキュリティーガイドプロファイル

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**	RHEL 8.4.4 以降: 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 8 Benchmark for Level 2 - Server	**xccdf_org.ssgproject.content_profile_cis**	RHEL 8.4.3 以前:1.0.0 RHEL 8.4.4 ～ RHEL 8.4.10:1.0.1 RHEL 8.4.11 以降:2.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 1 - Server	**xccdf_org.ssgproject.content_profile_cis_server_l1**	RHEL 8.4.4 ～ RHEL 8.4.10:1.0.1 RHEL 8.4.11 以降:2.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 1 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l1**	RHEL 8.4.4 ～ RHEL 8.4.10:1.0.1 RHEL 8.4.11 以降:2.0.0
CIS Red Hat Enterprise Linux 8 Benchmark for Level 2 - Workstation	**xccdf_org.ssgproject.content_profile_cis_workstation_l2**	RHEL 8.4.4 ～ RHEL 8.4.10:1.0.1 RHEL 8.4.11 以降:2.0.0
Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	**xccdf_org.ssgproject.content_profile_cui**	r1
Australian Cyber Security Centre (ACSC) Essential Eight	**xccdf_org.ssgproject.content_profile_e8**	バージョン付けなし
Australian Cyber Security Centre (ACSC) ISM Official	**xccdf_org.ssgproject.content_profile_ism_o**	RHEL 8.4.4 以降: 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
Protection Profile for General Purpose Operating Systems	**xccdf_org.ssgproject.content_profile_ospp**	4.2.1

プロファイル名	プロファイル ID	ポリシーバージョン
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	RHEL 8.4.3 以前:V1R1 RHEL 8.4.4 ～ RHEL 8.4.7:V1R3 RHEL 8.4.8:V1R5 RHEL 8.4.9 ～ RHEL 8.4.10:V1R6 RHEL 8.4.11 ～ RHEL 8.4.14:V1R7 RHEL 8.4.15 以降:V1R9
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) with GUI for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig_gui**	RHEL 8.4.4 ～ RHEL 8.4.7:V1R3 RHEL 8.4.8:V1R5 RHEL 8.4.9 ～ RHEL 8.4.10:V1R6 RHEL 8.4.11 ～ RHEL 8.4.14:V1R7 RHEL 8.4.15 以降:V1R9

表7.9 RHEL 8.3 で SCAP セキュリティーガイドプロファイル

プロファイル名	プロファイル ID	ポリシーバージョン
CIS Red Hat Enterprise Linux 8 Benchmark	**xccdf_org.ssgproject.content_profile_cis**	1.0.0
Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	**xccdf_org.ssgproject.content_profile_cui**	r1
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
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 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1
[DRAFT] The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	ドラフト

表7.10 RHEL 8.2 で SCAP セキュリティーガイドプロファイル

プロファイル名	プロファイル ID	ポリシーバージョン
Australian Cyber Security Centre (ACSC) Essential Eight	**xccdf_org.ssgproject.content_profile_e8**	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 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1
[DRAFT] DISA STIG for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	ドラフト

表7.11 RHEL 8.1 で SCAP セキュリティーガイドプロファイル

プロファイル名	プロファイル ID	ポリシーバージョン
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 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1

表7.12 RHEL 8.0 で対応する SCAP セキュリティーガイドプロファイル

プロファイル名	プロファイル ID	ポリシーバージョン
OSPP: 汎用オペレーティングシステムの保護プロファイル	**xccdf_org.ssgproject.content_profile_ospp**	ドラフト
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1

7.13. 関連情報

- RHEL の SCAP セキュリティーガイドで対応しているバージョン
- 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ーコンテンツを提供します。

- セキュリティー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ーコンプライアンスを自動化し、業界標準のセキュリティーポリシーとカスタムセキュリティー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ー技術を使用して、RHEL システムの全レベルでセキュリティー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ーです。

- Red Hat OVAL コンテンツリポジトリー 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ーがあります。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 における セキュリティーコンプライアンスの管理 - このガイドセットでは、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方法などが説明されています。

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 をインストールし、そのデータベースを開始するには、次の手順が必要です。

前提条件

- **AppStream** リポジトリー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。

```
# yum install aide
```

2. 初期データベースを生成します。

```
# aide --init
```

注記

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。

3. 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
```

4. 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

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: 2018-07-11 12:41:20 +0200 (AIDE 0.16)
AIDE found differences between database and filesystem!!
...
[trimmed for clarity]
```

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
```

8.3. AIDE データベースの更新

Red Hat は、システムの変更 (パッケージの更新、設定ファイルの修正など) を確認してから、基本となる AIDE データベースを更新することを推奨します。

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ーです。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。

比較項目	Advanced Intrusion Detection Environment (AIDE)	Integrity Measurement Architecture (IMA)
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ーを確保します。

8.5. 関連情報

- **aide(1)** の man ページ
- Kernel integrity subsystem

第9章 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

カーネル整合性サブシステムのコンポーネントを使用して、システムの保護を強化できます。関連するコンポーネントとその設定の詳細をご覧ください。

9.1. カーネル整合性サブシステム

整合性サブシステムは、システムデータの全体的な整合性を維持するカーネルの一部です。このサブシステムは、システムの状態を構築時と同じ状態に保つのに役立ちます。このサブシステムを使用すると、特定のシステムファイルの望ましくない変更を防ぐことができます。

カーネル整合性サブシステムは、2 つの主要なコンポーネントで設定されています。

Integrity Measurement Architecture (IMA)

- IMA は、ファイルが実行されるか開かれるたびに、暗号学的にハッシュするか、暗号化キーで署名することにより、ファイルの内容を測定します。キーは、カーネルキーリングサブシステムに格納されます。
- IMA は、測定値をカーネルのメモリー空間内に格納します。これにより、システムのユーザーが測定値を変更できなくなります。
- IMA は、ローカルおよびリモートのユーザーが測定値を検証できるようにします。
- IMA は、カーネルメモリー内の測定リストに以前に格納された値に対して、ファイルの現在の内容をローカルで検証します。この拡張機能は、現在の測定と以前の測定が一致しない場合に、特定のファイルに対して操作を実行することを禁止します。

Extended Verification Module (EVM)

- EVM は、IMA 測定値や SELinux 属性など、システムセキュリティーに関連するファイルの拡張属性 (**xattr** とも呼ばれます) を保護します。EVM は、対応する値を暗号的にハッシュするか、暗号鍵で署名します。キーは、カーネルキーリングサブシステムに格納されます。

カーネル整合性サブシステムは、TPM (Trusted Platform Module) を使用して、システムセキュリティーをさらに強化できます。

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、重要な暗号化機能のために Trusted Computing Group (TCG) による TPM 仕様に従って構築されています。TPM は通常、プラットフォームのマザーボードに接続された専用ハードウェアとして構築されます。ハードウェアチップの保護された改ざん防止領域から暗号化機能を提供することにより、TPM はソフトウェアベースの攻撃から保護されます。TPM は次の機能を提供します。

- 乱数ジェネレーター
- 暗号化キーのジェネレーターと安全なストレージ
- ハッシュジェネレーター
- リモート認証

関連情報

- セキュリティーの強化

- SELinux (Security-Enhanced Linux) の基本設定および高度な設定

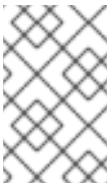
9.2. 信頼できる鍵および暗号化された鍵

信頼できる鍵 および **暗号化鍵** は、システムセキュリティーを強化する上で重要な要素です。

信頼できる鍵と暗号化された鍵は、カーネルキーリングサービスを使用するカーネルが生成する可変長の対称鍵です。キーの整合性を検証できます。つまり、実行中のシステムの整合性を検証および確認するために、Extended Verification Module (EVM) などでキーを使用できます。ユーザーレベルのプログラムがアクセス可能なのは、暗号化された **ブロブ** の形式での鍵のみです。

信頼できる鍵

信頼できる鍵は、鍵の作成と暗号化 (保護) の両方に使用される Trusted Platform Module (TPM) チップを必要とします。各 TPM には、ストレージルートキーと呼ばれるマスターラッピングキーがあります。これは TPM 自体に保存されます。



注記

Red Hat Enterprise Linux 8 は、TPM 1.2 と TPM 2.0 の両方をサポートしています。詳細は、Red Hat 製品では Trusted Platform Module (TPM) がサポートされますか? を参照してください。

次のコマンドを入力すると、TPM 2.0 チップが有効になっていることを確認できます。

```
$ cat /sys/class/tpm/tpm0/tpm_version_major
2
```

TPM 2.0 チップを有効にし、マシンのファームウェアの設定を通じて TPM 2.0 デバイスを管理することもできます。

さらに、TPM の **platform configuration register** (PCR) 値の特定セットを使用して、信頼できる鍵を保護できます。PCR には、ファームウェア、ブートローダー、およびオペレーティングシステムを反映する整合性管理値のセットが含まれます。つまり、PCR で保護された鍵は、暗号化を行った同じシステム上にある TPM でしか復号できません。ただし、PCR で保護された信頼できる鍵が読み込まれると (キーリングに追加されると)、新しいカーネルなどを起動できるように、関連する PCR 値が検証され、新しい (または今後の) PCR 値に更新されます。単一のキーを、それぞれ異なる PCR 値を持つ複数のブロブとして保存することもできます。

暗号化鍵

暗号化鍵はカーネル Advanced Encryption Standard (AES) を使用するため、TPM を必要としません。これにより、暗号化鍵は信頼できる鍵よりも高速になります。暗号化鍵は、カーネルが生成した乱数を使用して作成され、ユーザー空間のブロブへのエクスポート時に **マスターキー** により暗号化されます。

マスターキーは信頼できる鍵か、ユーザーキーのいずれかです。マスターキーが信頼されていない場合には、暗号化鍵のセキュリティーは、暗号化に使用されたユーザーキーと同じように保護されます。

9.3. 信頼できる鍵での作業

keyctl ユーティリティーを使用して、信頼できる鍵を作成、エクスポート、ロード、更新することにより、システムのセキュリティーを向上できます。

前提条件

- 64 ビット ARM アーキテクチャーおよび IBM Z の場合、**trusted** カーネルモジュールがロードされます。

```
# modprobe trusted
```

カーネルモジュールをロードする方法の詳細は、システムランタイム時のカーネルモジュールの読み込み を参照してください。

- Trusted Platform Module (TPM) が有効でアクティブである。カーネル整合性サブシステム および 信頼できる鍵および暗号化鍵 を参照してください。

注記

Red Hat Enterprise Linux 8 は、TPM 1.2 と TPM 2.0 の両方をサポートしています。TPM 1.2 を使用する場合は、手順 1 をスキップします。

手順

1. 次のユーティリティーのいずれかを使用して、永続ハンドル (たとえば、81000001) を持つ SHA-256 プライマリーストレージキーを使用して 2048 ビット RSA キーを作成します。

   a. **tss2** パッケージを使用する場合:

```
# TPM_DEVICE=/dev/tpm0 tsscreateprimary -hi o -st
Handle 80000000
# TPM_DEVICE=/dev/tpm0 tssevictcontrol -hi o -ho 80000000 -hp 81000001
```

   b. **tpm2-tools** パッケージを使用する場合:

```
# tpm2_createprimary --key-algorithm=rsa2048 --key-context=key.ctxt
name-alg:
  value: sha256
  raw: 0xb
…
sym-keybits: 128
rsa: xxxxxx…

# tpm2_evictcontrol -c key.ctxt 0x81000001
persistentHandle: 0x81000001
action: persisted
```

2. 信頼できるキーを作成します。

   a. **keyctl add trusted <NAME> "new <KEY_LENGTH> keyhandle=<PERSISTENT-HANDLE> [options]" <KEYRING>** の構文の TPM 2.0 を使用します。この例では、永続ハンドルは 81000001 です。

```
# keyctl add trusted kmk "new 32 keyhandle=0x81000001" @u
642500861
```

このコマンドは、**kmk** という名前の信頼できる鍵を **32** バイト (256 ビット) の長さで作成し、ユーザーキーリング (**@u**) に配置します。鍵の長さは 32 から 128 バイト (256 から 1024 ビット) です。

b. **keyctl add trusted <NAME> "new <KEY_LENGTH>" <KEYRING>** の構文の TPM 1.2 を使用します。

```
# keyctl add trusted kmk "new 32" @u
```

3. カーネルキーリングの現在の構造を一覧表示します。

```
# keyctl show
Session Keyring
       -3 --alswrv    500   500  keyring: ses 97833714 --alswrv 500 -1 \ keyring: uid.1000
642500861 --alswrv 500 500 \ trusted: kmk
```

4. 信頼できる鍵のシリアル番号を使用して、鍵をユーザー空間のブロブにエクスポートします。

```
# keyctl pipe 642500861 > kmk.blob
```

このコマンドは、**pipe** サブコマンドと **kmk** のシリアル番号を使用します。

5. ユーザー空間のブロブから信頼できる鍵をロードします。

```
# keyctl add trusted kmk "load `cat kmk.blob`" @u
268728824
```

6. TPM で保護された信頼できる鍵 (**kmk**) を使用するセキュアな暗号化鍵を作成します。**keyctl add encrypted <NAME> "new FORMAT <KEY_TYPE>:<PRIMARY_KEY_NAME> <KEY_LENGTH>" <KEYRING>** という構文に従います。

```
# keyctl add encrypted encr-key "new trusted:kmk 32" @u
159771175
```

関連情報

- **keyctl(1)** の man ページ
- 信頼できる鍵および暗号化された鍵
- Kernel Key Retention Service
- カーネル整合性サブシステム

9.4. 暗号化鍵での作業

暗号化鍵を管理することで、Trusted Platform Module (TPM) が使用できないシステムのシステムセキュリティーを向上できます。

前提条件

- 64 ビット ARM アーキテクチャーおよび IBM Z の場合、**encrypted-keys** カーネルモジュールがロードされます。

```
# modprobe encrypted-keys
```

カーネルモジュールをロードする方法の詳細は、システムランタイム時のカーネルモジュールの読み込み を参照してください。

手順

1. 無作為な数列を使用してユーザーキーを生成します。

```
# keyctl add user kmk-user "$(dd if=/dev/urandom bs=1 count=32 2>/dev/null)" @u
427069434
```

このコマンドは、**kmk-user** という名前のユーザーキーを生成し、**プライマリーキー** として動作させ、このユーザーキーを使用して実際の暗号化鍵を保護します。

2. 前の手順で取得したプライマリーキーを使用して、暗号化鍵を生成します。

```
# keyctl add encrypted encr-key "new user:kmk-user 32" @u
1012412758
```

3. 必要に応じて、指定したユーザーキーリングにあるすべての鍵をリスト表示します。

```
# keyctl list @u
2 keys in keyring:
427069434: --alswrv  1000  1000 user: kmk-user
1012412758: --alswrv  1000  1000 encrypted: encr-key
```

重要

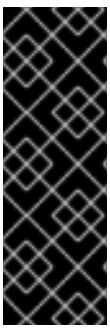
信頼できるプライマリーキーで保護されていない暗号化鍵では、暗号化に使用されたユーザーのプライマリーキー (乱数キー) と同程度のセキュリティーしか得られません。そのため、プライマリーユーザーキーはできるだけセキュアに、システムの起動プロセスの早い段階でロードしてください。

関連情報

- **keyctl(1)** の man ページ
- Kernel Key Retention Service

9.5. IMA と EVM の有効化

Integrity Measurement Architecture (IMA) と Extended Verification Module (EVM) を有効にして設定することで、オペレーティングシステムのセキュリティーを向上できます。



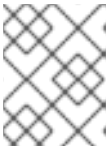
重要

必ず IMA と一緒に EVM を有効にしてください。

EVM を単独で有効にすることもできますが、EVM 評価は IMA 評価ルールによってのみトリガーされます。したがって、SELinux 属性などのファイルメタデータが EVM によって保護されません。ファイルメタデータがオフラインで改ざんされた場合、EVM はファイルメタデータの変更を防ぐことしかできません。ファイルの実行などのファイルアクセスは妨げません。

前提条件

- セキュアブートが一時的に無効になっている。



注記

セキュアブートが有効になっている場合、**ima_appraise=fix** カーネルコマンドラインパラメーターが機能しません。

- **securityfs** ファイルシステムが **/sys/kernel/security/** ディレクトリーにマウントされており、**/sys/kernel/security/integrity/ima/** ディレクトリーが存在している。**mount** コマンドを使用して、**securityfs** がマウントされている場所を確認できます。

```
# mount
...
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
...
```

- **systemd** サービスマネージャーに、ブート時に IMA と EVM をサポートするパッチが適用されている。次のコマンドを使用して確認します。

```
# grep <options> pattern <files>
```

以下に例を示します。

```
# dmesg | grep -i -e EVM -e IMA -w
[ 0.598533] ima: No TPM chip found, activating TPM-bypass!
[ 0.599435] ima: Allocated hash algorithm: sha256
[ 0.600266] ima: No architecture policies found
[ 0.600813] evm: Initialising EVM extended attributes:
[ 0.601581] evm: security.selinux
[ 0.601963] evm: security.ima
[ 0.602353] evm: security.capability
[ 0.602713] evm: HMAC attrs: 0x1
[ 1.455657] systemd[1]: systemd 239 (239-74.el8_8) running in system mode. (+PAM
+AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP
+GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN2 -
IDN +PCRE2 default-hierarchy=legacy)
[ 2.532639] systemd[1]: systemd 239 (239-74.el8_8) running in system mode. (+PAM
+AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP
+GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN2 -
IDN +PCRE2 default-hierarchy=legacy)
```

手順

1. 現在のブートエントリーの **fix** モードで IMA と EVM を有効にし、次のカーネルコマンドラインパラメーターを追加することで、ユーザーが IMA 測定値を収集および更新できるようにします。

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="ima_policy=appraise_tcb ima_appraise=fix evm=fix"
```

このコマンドは、現在のブートエントリーの **fix** モードで IMA および EVM を有効にしてユーザーが IMA 測定を収集し、更新できるようにします。

ima_policy=appraise_tcb カーネルコマンドラインパラメーターにより、カーネルは、デフォルトの TCB (Trusted Computing Base) 測定ポリシーと評価手順を使用するようになります。評価手順では、以前の測定と現在の測定が一致しないファイルへのアクセスを禁止します。

2. 再起動して変更を適用します。

3. オプション: 必要に応じて、パラメーターがカーネルコマンドラインに追加されていることを確認します。

```
# cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-167.el8.x86_64 root=/dev/mapper/rhel-root
ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet ima_policy=appraise_tcb ima_appraise=fix evm=fix
```

4. カーネルマスターキーを作成して、EVM 鍵を保護します。

```
# keyctl add user kmk "$(dd if=/dev/urandom bs=1 count=32 2> /dev/null)" @u
748544121
```

kmk は、すべてカーネル領域メモリー内に保持されます。**kmk** の 32 バイトの Long 値は、**/dev/urandom** ファイルの乱数バイト数から生成し、ユーザーの (**@u**) キーリングに配置します。鍵のシリアル番号は、前の出力の 1 行目にあります。

5. **kmk** に基づいて暗号化された EVM 鍵を作成します。

```
# keyctl add encrypted evm-key "new user:kmk 64" @u
641780271
```

kmk を使用して 64 バイトの long 型ユーザーキー (**evm-key**) を生成してユーザー (**@u**) のキーリングに配置します。鍵のシリアル番号は、前の出力の 1 行目にあります。

重要

ユーザーキーの名前は **evm-key** (EVM サブシステムが想定して使用している名前) にする必要があります。

6. エクスポートする鍵のディレクトリーを作成します。

```
# mkdir -p /etc/keys/
```

7. **kmk** を検索し、その暗号化されていない値を新しいディレクトリーにエクスポートします。

```
# keyctl pipe $(keyctl search @u user kmk) > /etc/keys/kmk
```

8. **evm-key** を検索し、その暗号化された値を新しいディレクトリーにエクスポートします。

```
# keyctl pipe $(keyctl search @u encrypted evm-key) > /etc/keys/evm-key
```

evm-key は、すでにカーネルのマスターキーにより暗号化されています。

9. オプション: 新しく作成されたキーを表示します。

```
# keyctl show
Session Keyring
974575405   --alswrv     0        0      keyring: ses 299489774 --alswrv 0 65534 \ keyring:
uid.0 748544121 --alswrv 0 0 \ user: kmk
641780271   --alswrv     0        0           \_ encrypted: evm-key

# ls -l /etc/keys/
total 8
-rw-r--r--. 1 root root 246 Jun 24 12:44 evm-key
-rw-r--r--. 1 root root  32 Jun 24 12:43 kmk
```

10. オプション: システムの再起動後など、鍵がキーリングから削除されている場合は、新しい **kmk** と **evm-key** を作成せずに、すでにエクスポートされているものをインポートできます。

    a. **kmk** をインポートします。

    ```
    # keyctl add user kmk "$(cat /etc/keys/kmk)" @u
    451342217
    ```

    b. **evm-key** をインポートします。

    ```
    # keyctl add encrypted evm-key "load $(cat /etc/keys/evm-key)" @u
    924537557
    ```

11. EVM をアクティブ化します。

    ```
    # echo 1 > /sys/kernel/security/evm
    ```

12. システム全体のラベルを付け直します。

    ```
    # find / -fstype xfs -type f -uid 0 -exec head -n 1 '{}' >/dev/null \;
    ```

警告

システムのラベルを変更せずに IMA と EVM を有効にすると、システム上のファイルの大部分にアクセスできなくなる可能性があります。

検証

- EVM が初期化されていることを確認します。

  ```
  # dmesg | tail -1
  […] evm: key initialized
  ```

関連情報

- grep(1) manpage

- カーネル整合性サブシステム
- 信頼できる鍵および暗号化された鍵

9.6. INTEGRITY MEASUREMENT ARCHITECTURE によるファイルのハッシュの収集

測定 フェーズでは、ファイルハッシュを作成し、そのファイルの拡張属性 (**xattrs**) としてファイルハッシュを保存できます。ファイルハッシュを使用すると、RSA ベースのデジタル署名またはハッシュベースのメッセージ認証コード (HMAC-SHA1) を生成できるため、拡張属性に対するオフライン改ざん攻撃を防ぐことができます。

前提条件

- IMA と EVM 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。
- 有効な信頼できる鍵または暗号化鍵が、カーネルキーリングに保存されている。
- **ima-evm-utils**、**attr**、および **keyutils** パッケージがインストールされている。

手順

1. テストファイルを作成します。

```
# echo <Test_text> > test_file
```

   IMA と EVM は、**test_file** サンプルファイルにハッシュ値が割り当てられ、その拡張属性として格納されていることを確認します。

2. ファイルの拡張属性を検査します。

```
# getfattr -m . -d test_file
# file: test_file
security.evm=0sAnDIy4VPA0HArpPO/EqiutnNyBql
security.ima=0sAQOEDeuUnWzwwKYk+n66h/vby3eD
```

   出力例には、IMA および EVM ハッシュ値と SELinux コンテキストを含む拡張属性が示されています。EVM は、他の属性に関連する **security.evm** 拡張属性を追加します。この時点で、**security.evm** で **evmctl** ユーティリティーを使用して、RSA ベースのデジタル署名またはハッシュベースのメッセージ認証コード (HMAC-SHA1) を生成できます。

関連情報

- セキュリティーの強化

第10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。

10.1. 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求められます。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ノート PC 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ーを強化し、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。

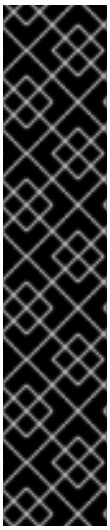
関連情報

- 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)

10.2. 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

LUKS2 と LUKS1 はディスクの暗号化に異なるコマンドを使用するため、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。LUKS バージョンに誤ったコマンドを使用すると、データが失われる可能性があります。

表10.1 LUKS バージョンに応じた暗号化コマンド

LUKS バージョン	暗号化コマンド
LUKS2	**cryptsetup reencrypt**
LUKS1	**cryptsetup-reencrypt**

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。

10.3. 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。LUKS2 は、次のモードの **resilience** オプションを備えています。**cryptsetup reencrypt --resilience resilience-mode /dev/sdx** 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。

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ー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/sdx** コマンドを使用する。

関連情報

- **cryptsetup-reencrypt(8)** および **cryptsetup-repair(8)** man ページ

10.4. 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

- **cryptsetup(8)**、**cryptsetup-reencrypt(8)**、**lvextend(8)**、**resize2fs(8)**、および **parted(8)** man ページ

10.5. 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. 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
```

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.
```

/home/header は、独立した LUKS ヘッダーを持つファイルへのパスに置き換えます。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できる必要があります。

3. デバイスをマウントします。

```
# mount /dev/mapper/nvme_encrypted /mnt/nvme_encrypted
```

4. 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

- **cryptsetup(8)** および **cryptsetup-reencrypt(8)** man ページ

10.6. 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

- **cryptsetup(8)**、**cryptsetup-open (8)**、および **cryptsetup-lusFormat(8)** man ページ

10.7. 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

- 制御ノードと管理ノードを準備している
- 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: Create and configure a volume encrypted with LUKS
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
        - sdb
        fs_type: xfs
        fs_label: label-name
        mount_point: /mnt/data
        encryption: true
        encryption_password: <password>
```

また、**encryption_key**、**encryption_cipher**、**encryption_key_size**、**encryption_luks** など、他の暗号化パラメーターを Playbook ファイルに追加することもできます。

2. Playbook の構文を検証します。

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

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

3. Playbook を実行します。

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

検証

1. 暗号化ステータスを表示します。

```
# cryptsetup status sdb

/dev/mapper/sdb is active and is in use.
type: LUKS2
cipher: aes-xts-plain64
keysize: 512 bits
key location: keyring
device: /dev/sdb
...
```

2. 作成された LUKS 暗号化ボリュームを確認します。

```
# cryptsetup luksDump /dev/sdb

Version:        2
Epoch:          6
Metadata area:  16384 [bytes]
Keyslots area:  33521664 [bytes]
UUID:           a4c6be82-7347-4a91-a8ad-9479b72c9426
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          allow-discards

Data segments:
  0: crypt
        offset: 33554432 [bytes]
        length: (whole device)
        cipher: aes-xts-plain64
        sector: 4096 [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

第11章 ポリシーベースの複号を使用した暗号化ボリュームの自動アンロックの設定

ポリシーベースの複号 (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、もくし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。各ピンは、個別のアンロック機能を提供します。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るようにします。

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るようにします。

sss

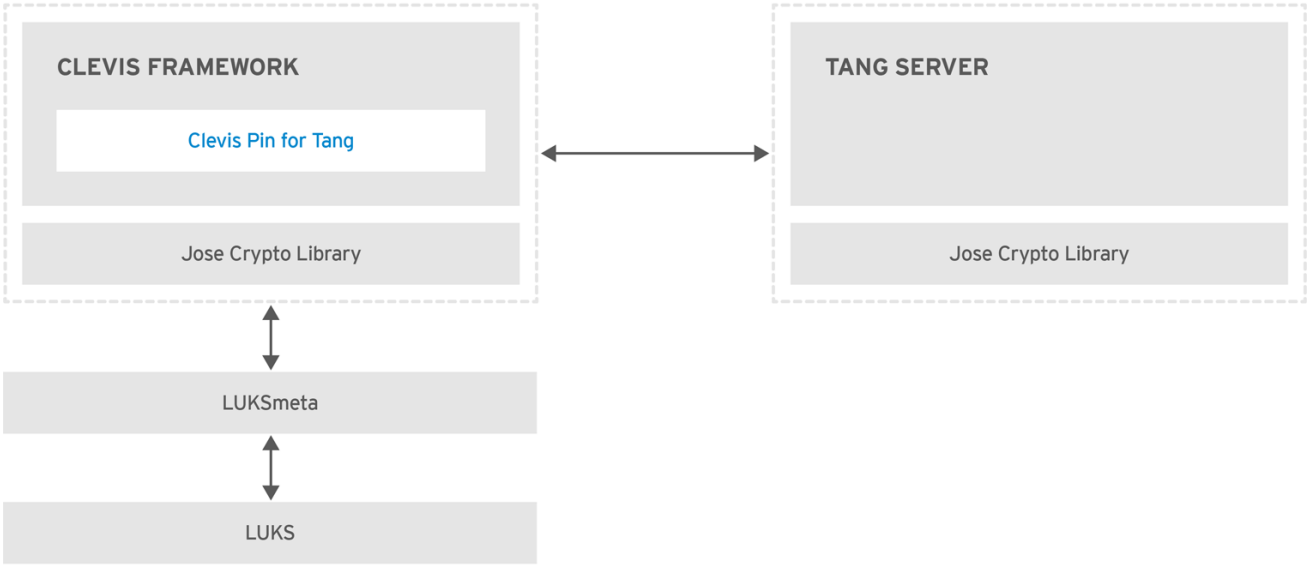
Shamir's Secret Sharing (SSS) 暗号化スキームを使用して高可用性システムをデプロイできます。

11.1. NBDE (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11.1 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。セキュリティー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 つは、NBDE サーバー (Tang) との相互作用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、暗号化および複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。

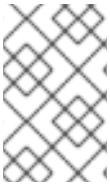
Tang 用の Clevis ピンは、公開鍵のいずれかを使用して、固有で、暗号論的に強力な暗号鍵を生成します。この鍵を使用してデータを暗号化すると、この鍵は破棄されます。Clevis クライアントは、使いやすい場所に、このプロビジョニング操作で生成した状態を保存する必要があります。データを暗号化するこのプロセスは **プロビジョニング手順** と呼ばれています。

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。詳細については、 Encrypting block devices using LUKS 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

- NBDE (Network-Bound Disk Encryption) テクノロジーの ナレッジベース記事
- **tang(8)**、**clevis(1)**、**jose(1)** および **clevis-luks-unlockers(7)** の 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)

11.2. 暗号化クライアント (CLEVIS) のインストール

この手順に従って、システムに Clevis プラグ可能フレームワークを使用してデプロイと起動を行います。

手順

1. 暗号化されたボリュームを持つシステムに Clevis とそのピンをインストールするには、次のコマンドを実行します。

```
# yum install clevis
```

2. データを複号するには、**clevis decrypt** コマンドを実行して、JWE (JSON Web Encryption) 形式で暗号文を指定します。以下に例を示します。

```
$ clevis decrypt < secret.jwe
```

関連情報

- **clevis(1)** の man ページ
- 引数を指定せずに **clevis** コマンドを実行した後の組み込み CLI ヘルプ

```
$ clevis
Usage: clevis COMMAND [OPTIONS]

clevis decrypt      Decrypts using the policy defined at encryption time
clevis encrypt sss  Encrypts using a Shamir's Secret Sharing policy
clevis encrypt tang Encrypts using a Tang binding server policy
clevis encrypt tpm2 Encrypts using a TPM2.0 chip binding policy
clevis luks bind    Binds a LUKS device using the specified policy
clevis luks edit    Edit a binding from a clevis-bound slot in a LUKS device
clevis luks list    Lists pins bound to a LUKSv1 or LUKSv2 device
clevis luks pass    Returns the LUKS passphrase used for binding a particular slot.
clevis luks regen   Regenerate clevis binding
clevis luks report  Report tang keys' rotations
clevis luks unbind  Unbinds a pin bound to a LUKS volume
clevis luks unlock  Unlocks a LUKS volume
```

11.3. 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

- **policycoreutils-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** サービスが実行している。

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。

```
# yum 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。

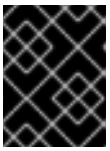
```
# systemctl enable tangd.socket
```

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. **Ctrl**+**O** と **Enter** を押し、変更を保存します。**Ctrl**+**X** を押してエディターを終了します。
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。

関連情報

- **tang(8)**、**semanage(8)**、**firewall-cmd(1)**、**jose(1)**、**systemd.unit(5)** および **systemd.socket(5)** の man ページ

11.4. 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

以下の手順に従って、Tang サーバーの鍵をローテーションし、クライアントの既存のバインディングを更新します。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。
- RHEL 8.2 に、**clevis luks list**、**clevis luks report**、および **clevis luks regen** が追加されていることに注意してください。

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、関連するバインディ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
```

7. 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

- **tang-show-keys(1)**、**clevis-luks-list(1)**、**clevis-luks-report(1)**、および **clevis-luks-regen(1)** の man ページ

11.5. 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 8 Web コンソールがインストールされている。詳細は、Web コンソールのインストール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 参照してください。

手順

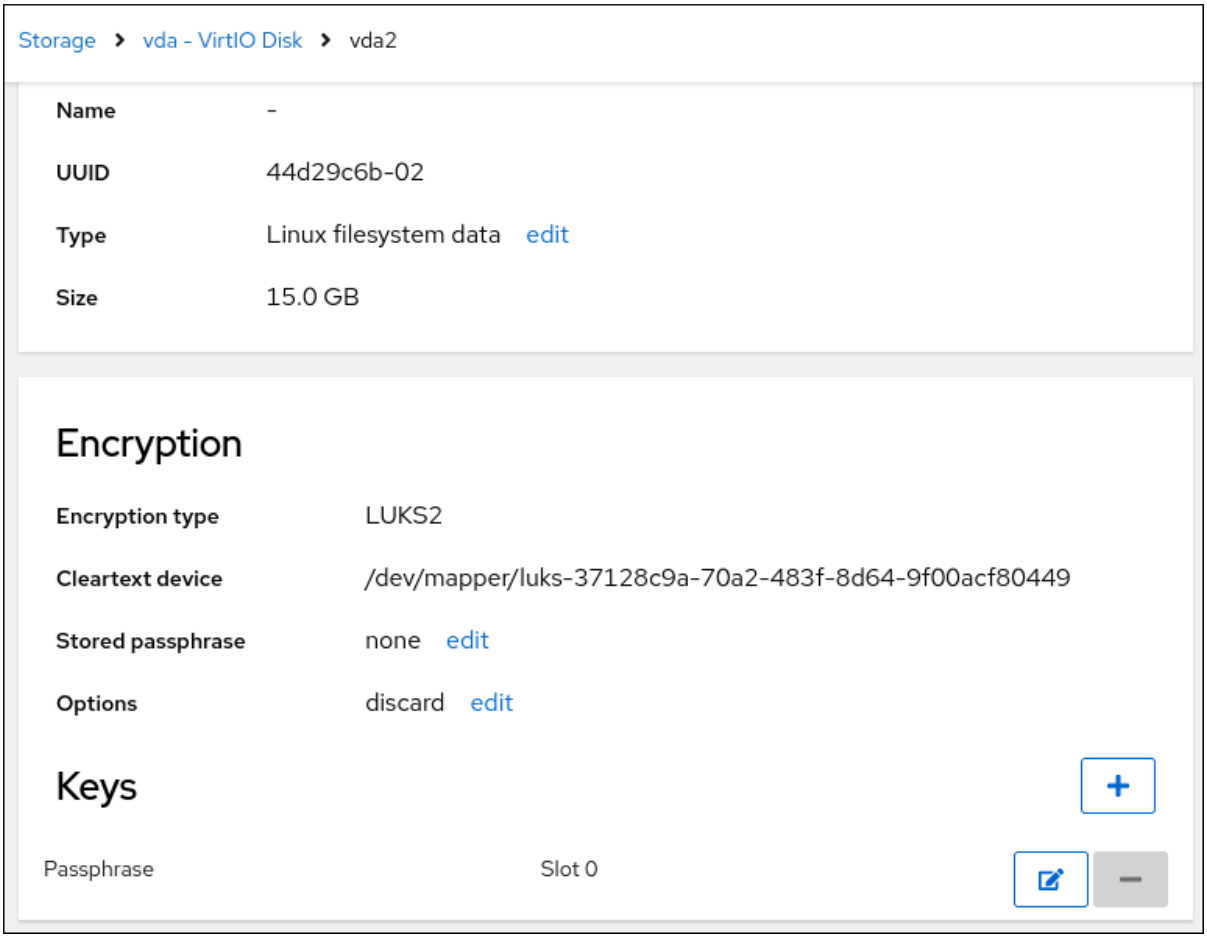
1. Web ブラウザーに以下のアドレスを入力して、RHEL Web コンソールを開きます。

   ```
   https://<localhost>:9090
   ```

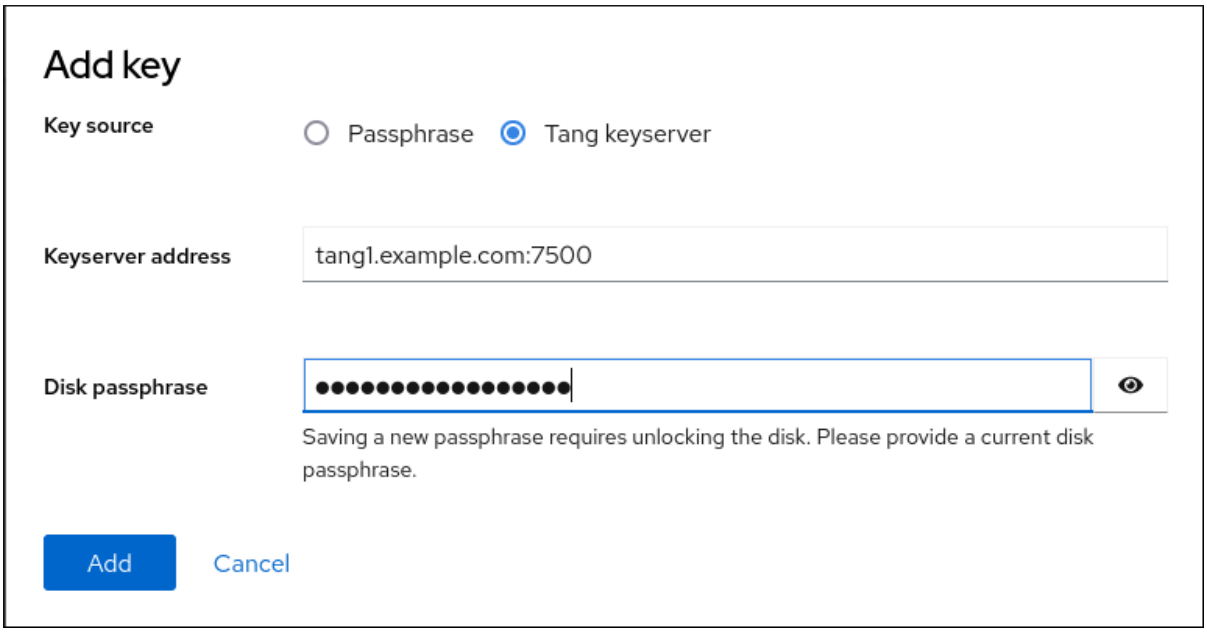
   リモートシステムに接続する場合は、**<localhost>** の部分をリモートサーバーのホスト名または IP アドレスに置き換えます。

2. 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。



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。



以下のダイアログウインドウは、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_1k6GPiDOaMlL3WbpCjHOy9ul1bSfdhI3M08wO0
```

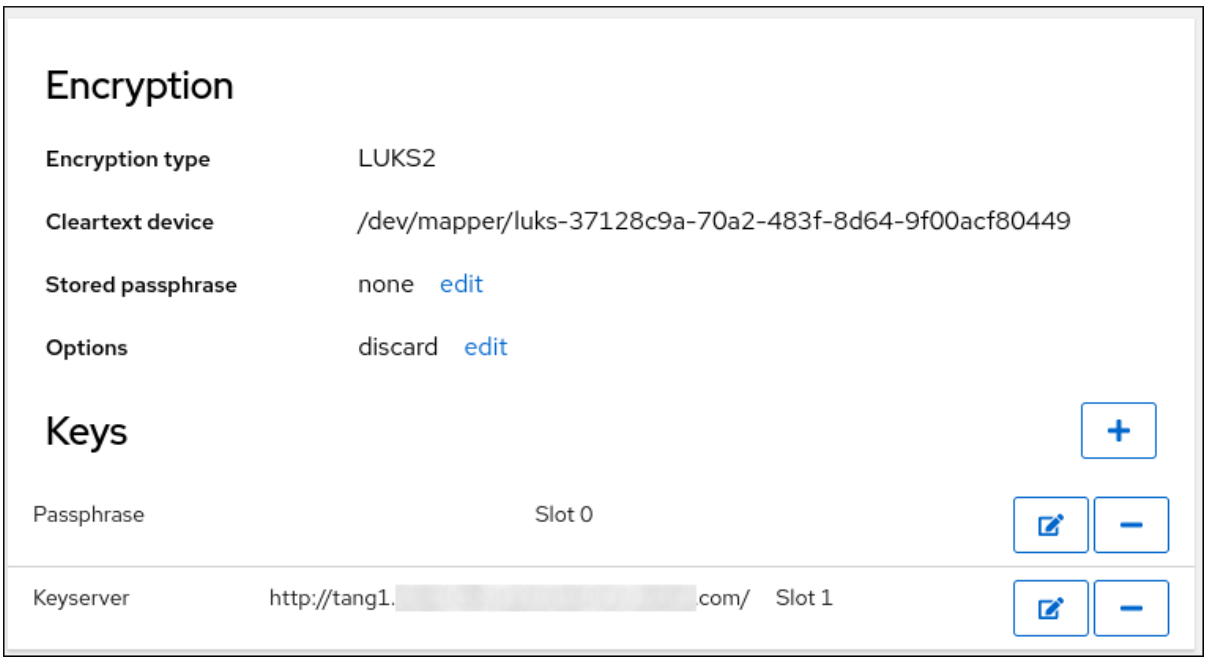
6. Web コンソールと前述のコマンドの出力のキーハッシュが同じ場合は、**Trust key** をクリックします。

7. RHEL 8.8 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 (**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。



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

11.6. 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サ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"}'
```

- データを複号するには、**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
```

TPM2.0 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、設定プロパティーを指定します。

```
$ 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、PCP 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フレーズを追加します。

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、KCS の記事Clevis TPM2 no longer decrypts LUKS devices after a restartの手順に従ってください。

関連情報

- **clevis-encrypt-tang(1)**、**clevis-luks-unlockers(7)**、**clevis(1)**、および **clevis-encrypt-tpm2(1)** の 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
...
```

11.7. LUKS で暗号化したボリュームの手動登録の設定

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

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。

```
# yum 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 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:
```

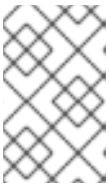
このコマンドは、以下の 4 つの手順を実行します。

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。

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。

4. 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ようにするには、インストール済みのシステムで **dracut** ツールを使用します。

```
# yum install clevis-dracut
```

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。

```
# dracut -fv --regenerate-all --hostonly-cmdline
```

または、**/etc/dracut.conf.d/** に .conf ファイルを作成し、以下のように **hostonly_cmdline=yes** オプションを追加します。

```
# echo "hostonly_cmdline=yes" > /etc/dracut.conf.d/clevis.conf
```

注記

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"
```

次に、**--hostonly-cmdline** なしで **dracut** を使用できます。

```
# 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: tang '{"url":"http://tang.srv:port"}'
```

重要

(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、以下のように、手動でネットワーク設定を **dracut** ツールに渡します。

```
# 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"
```

もしくは、静的ネットワーク情報を使用して **/etc/dracut.conf.d/** ディレクトリーに .conf ファイルを作成します。以下に例を示します。

```
# 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"
```

初期 RAM ディスクイメージを再生成します。

```
# dracut -fv --regenerate-all
```

関連情報

- **clevis-luks-bind(1)** および **dracut.cmdline(7)** の man ページ
- ネットワーク起動オプション
- Looking forward to Linux network configuration in the initial ramdisk (initrd)

11.8. 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

次の手順を使用して、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します。

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. LUKS で暗号化した既存のボリュームを自動的にアンロックするには、サブパッケージの **clevis-luks** をインストールします。

```
# yum 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"}'
```

警告

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スフレーズを追加します。

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、KCS の記事Clevis TPM2 no longer decrypts LUKS devices after a restartの手順に従ってください。

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ンディングを処理するようにするには、インストール済みのシステムで **dracut** ツールを使用します。

```
# yum 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

- **clevis-luks-bind(1)**、**clevis-encrypt-tpm2(1)**、および **dracut.cmdline(7)** の man ページ

11.9. 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 が追加したパスフレー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

1. **/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

- **clevis-luks-unbind(1)**、**cryptsetup(8)**、および **luksmeta(8)** の man ページ

11.10. 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. オプションで、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、一時パスフレーズを削除する前に強力なパスフレーズを追加します。詳細については、How to add a passphrase, key, or keyfile to an existing LUKS device の記事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

- **clevis(1)**、**clevis-luks-bind(1)**、**cryptsetup(8)**、および **dmsetup(8)** の man ページ
- キックスタートを使用して Red Hat Enterprise Linux 8 のインストール

11.11. 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アンロックの設定

この手順に従って、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します。

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。

```
# yum 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. LUKS で暗号化したリムーバブルデバイスは、GNOME デスクトップセッションで自動的にアンロック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

- **clevis-luks-unlockers(7)** man ページ

11.12. 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

- **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)** の man ページ

11.13. 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ー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

- **clevis-luks-bind(1)** man ページ

11.14. 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

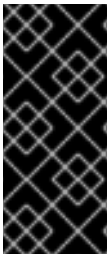
自動登録可能な暗号化イメージをクラウド環境にデプロイすると、特有の課題が発生する可能性があります。他の仮想化環境と同様に、LUKS マスター鍵を共有しないように、1 つのイメージから起動するインスタンス数を減らすことが推奨されます。

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ーでも共有されず、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ー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

このため、Red Hat は、データを保存する場所と、Tang が実行しているシステムを、物理的に分離させることを強く推奨します。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。

11.15. コンテナーとしての TANG のデプロイ

tang コンテナーイメージは、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** コンテナーカタログにログインしている。詳細は、Red Hat コンテナー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** コンテナーイメージをプルします。

```
# podman pull registry.redhat.io/rhel8/tang
```

2. コンテナーを実行し、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** コンテナーを実行し、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/rhel8/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。

3. (必要に応じて) 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/rhel8/tang tangd-rotate-keys -
v -d /var/db/tang
Rotated key 'rZAMKAseaXBe0rcKXL1hCCIq-DY.jwk' -> .'rZAMKAseaXBe0rcKXL1hCCIq-
DY.jwk'
Rotated key 'x1AIpc6WmnCU-CabD8_4q18vDuw.jwk' -> .'x1AIpc6WmnCU-
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:

x1AIpc6WmnCU-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

- **podman(1)**、**clevis(1)** および **tang(8)** の man ページ

11.16. NBDE_CLIENT および NBDE_SERVER RHEL システムロールの概要 (CLEVIS および TANG)

RHEL システムロールは、複数の RHEL システムをリモートで管理するための一貫した設定インターフェイスを提供する Ansible ロールおよびモジュールのコレクションです。

RHEL 8.3 では、Clevis および Tang を使用した PBD (Policy-Based Decryption) ソリューションの自動デプロイメント用 Ansible ロールが導入されました。**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。

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。**nbde_client** ロールは、Tang バインディングのみをサポートしており、現時点では TPM2 バインディングには使用できない点に留意してください。

nbde_client ロールには、LUKS を使用して暗号化済みのボリュームが必要です。このロールは、LUKS 暗号化ボリュームの 1 つ以上の Network-Bound (NBDE) サーバー (Tang サーバー) へのバインドに対応します。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。

PBD では、デバイスをスロットにマッピングするものとしてバインディングを定義します。つまり、同じデバイスに複数のバインディングを指定できます。デフォルトのスロットは 1 です。

nbde_client ロールでは、**state** 変数も指定できます。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 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

関連情報

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

11.17. 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

以下の手順に従って、Tang サーバー設定を含む Ansible Playbook を準備および適用します。

前提条件

- 制御ノードと管理ノードを準備している

- 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。

```
---
- hosts: managed-node-01.example.com
  roles:
    - 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。

nbde_server_manage_firewall と **nbde_server_manage_selinux** が両方とも **true** に設定されている場合、**nbde_server** ロールは **firewall** ロールと **selinux** ロールを使用して、**nbde_server** ロールが使用するポートを管理します。

2. Playbook の構文を検証します。

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

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

3. Playbook を実行します。

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

検証

- 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には、次のコマンドを実行します。

```
# grubby --update-kernel=ALL --args="rd.neednet=1"
```

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

11.18. NBDE_CLIENT RHEL システムロールを使用した複数の CLEVIS クライアントのセットアップ

nbde_client RHEL システムロールを使用すると、Clevis クライアント設定を含む Ansible Playbook を複数のシステムで準備および適用できます。

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

- 制御ノードと管理ノードを準備している
- 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。

```
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.nbde_client
  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
```

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。

nbde_client システムロールは、動的ホスト設定プロトコル (DHCP) を使用する場合のみをサポートします。静的 IP 設定のクライアントに NBDE を使用するには、次の Playbook を使用します。

```
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.nbde_client
  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
  tasks:
    - name: Configure a client with a static IP address during early boot
      ansible.builtin.command:
        cmd: grubby --update-kernel=ALL --args='GRUB_CMDLINE_LINUX_DEFAULT="ip={{
<ansible_default_ipv4.address> }}::{{ <ansible_default_ipv4.gateway> }}:{{
<ansible_default_ipv4.netmask> }}::{{ <ansible_default_ipv4.alias> }}:none"'
```

この Playbook の **<ansible_default_ipv4.*>** 文字列は、ネットワークの IP アドレス (**ip={{ 192.0.2.10 }}::{{ 192.0.2.1 }}:{{ 255.255.255.0 }}::{{ ens3 }}:none**) に置き換えます。

2. Playbook の構文を検証します。

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

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

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.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) article

第12章 システムの監査

Audit は、追加のセキュリティー機能をシステムに提供するのではありません。システムで使用されるセキュリティーポリシーの違反を発見するために使用できます。このような違反は、SELinux などの別のセキュリティー対策で防ぐことができます。

12.1. LINUX の AUDIT

Linux の Audit システムは、システムのセキュリティー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、ログエントリーを生成し、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ー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ー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)
- PCI DSS (Payment Card Industry Data Security Standard)
- セキュリティー技術実装ガイド (Security Technical Implementation Guide (STIG))

Audit は以下でも認定されています。

- National Information Assurance Partnership (NIAP) および Best Security Industries (BSI) による評価
- Red Hat Enterprise Linux 5 における LSPP/CAPP/RSBAC/EAL4 以降の認定
- Red Hat Enterprise Linux 6 における OSPP/EAL4 以降 (Operating System Protection Profile / Evaluation Assurance Level 4 以降) の認定

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。

12.2. 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 8 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。

12.3. 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ー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ー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ーポリシーに合わせて設定します。

12.4. 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。**auid** 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。

12.5. 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ーを **/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ーに保存されます。

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 auid=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。各レコードには、スペースまたはコンマで区切られた名前と値のペア (**name=value**) が複数使用されています。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。

auid=1000

auid 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 が変更した後で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 イベントを発生するのに、**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。

ouid=0

ouid 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。

12.6. 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** ユーティリティー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

- **auditctl(8)** man ページ

12.7. 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
```

12.8. 標準に準拠するための事前設定された AUDIT 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

セキュリティー標準は動的であり、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、最新のものでもありません。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ー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ーガイドラインに従ってシステムを最新のセキュリティー標準に準拠させるには、SCAP ベースのセキュリティーコンプライアンスツール を使用してください。

30-nispom.rules

NISPOM (National Industrial Security Program Operating Manual) の Information System Security の章で指定している要件を満たす Audit 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

セキュリティー技術実装ガイド (STIG: Security Technical Implementation Guide) で設定されている要件を満たす Audit 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

- **audit.rules(7)** man ページ

12.9. 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を処理します。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

- **audit.rules(8)** および **augenrules(8)** の man ページ

12.10. 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

- **augenrules(8)** および **audit.rules(8)** の man ページ
- auditd service restart overrides changes made to /etc/audit/audit.rules .

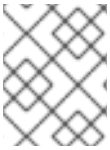
12.11. ソフトウェアの更新を監視するための AUDIT の設定

RHEL 8.6 以降のバージョンでは、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** [4]
- **yum**
- **pip**
- **npm**
- **cpan**
- **gem**
- **luarocks**

デフォルトでは、**rpm** はパッケージのインストールまたは更新時に監査 **SOFTWARE_UPDATE** イベントをすでに提供しています。これら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。

RHEL 8.5 以前のバージョンでは、手動でルールを追加して、**/etc/audit/rules.d/** ディレクトリーの **.rules** ファイルにソフトウェアをインストールするユーティリティーを監視できます。



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. RHEL 8.6 以降では、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/
```

RHEL 8.5 以前では、**/etc/audit/rules.d/** ディレクトリーに、**44-installers.rules** という名前の新規ファイルを作成し、以下のルールを挿入します。

```
-a always,exit -F perm=x -F path=/usr/bin/dnf-3 -F key=software-installer
-a always,exit -F perm=x -F path=/usr/bin/yum -F
```

同じ構文を使用して、**pip** や **npm** などのソフトウェアをインストールする他のユーティリティー用に、さらにルールを追加できます。

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

```
# yum 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"
```

12.12. 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** ユーティリティーにパイプで渡します。このユーティリティーは、**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

- **ausearch(8)** の man ページ。
- **aulast(8)** の man ページ。
- **aureport(8)** の man ページ。

12.13. 関連情報

- ナレッジベースのアーティクル記事 RHEL Audit System Reference
- ナレッジベースのアーティクル記事 Auditd execution options in a container
- Linux Audit ドキュメントのプロジェクトページ
- **audit** パッケージが提供するドキュメントは、**/usr/share/doc/audit/** ディレクトリーにあります。
- **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)** の man ページ。

[4] **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13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。

13.1. 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ー
- **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、YUM パッケージマネージャーまたは 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。

10

言語ルール

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** とも呼ばれます) 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。
- Managing, monitoring, and updating the kernel の Enhancing security with the kernel integrity subsystem の章を参照してください。
- **/usr/share/doc/fapolicyd/** ディレクトリーおよび **/usr/share/fapolicyd/sample-rules/README-rules** ファイルに **fapolicyd** パッケージとともにインストールされるドキュメント。

13.2. FAPOLICYD のデプロイ

RHEL に **fapolicyd** フレームワークをデプロイするには、以下を行います。

手順

1. **fapolicyd** パッケージをインストールします。

```
# yum install fapolicyd
```

2. **fapolicyd** サービスを有効にして開始します。

```
# systemctl enable --now 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; vendor p>
   Active: active (running) since Tue 2019-10-15 18:02:35 CEST; 55s ago
  Process: 8818 ExecStart=/usr/sbin/fapolicyd (code=exited, status=0/SUCCESS)
 Main PID: 8819 (fapolicyd)
    Tasks: 4 (limit: 11500)
   Memory: 78.2M
   CGroup: /system.slice/fapolicyd.service
           └─8819 /usr/sbin/fapolicyd

Oct 15 18:02:35 localhost.localdomain systemd[1]: Starting File Access Policy D>
Oct 15 18:02:35 localhost.localdomain fapolicyd[8819]: Initialization of the da>
Oct 15 18:02:35 localhost.localdomain fapolicyd[8819]: Reading RPMDB into memory
Oct 15 18:02:35 localhost.localdomain systemd[1]: Started File Access Policy Da>
Oct 15 18:02:36 localhost.localdomain fapolicyd[8819]: Creating database
```

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
```

13.3. 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

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

- **fapolicyd.trust (13)** man ページ。

13.4. 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 **yum** 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

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. **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の**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

- **fapolicyd.rules (5)** および **fapolicyd-cli (1)** の 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。

13.5. 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
```

13.6. FAPOLICYD に関連する問題のトラブルシューティング

次のセクションでは、**fapolicyd** アプリケーションフレームワークの基本的なトラブルシューティングのヒントと、**rpm** コマンドを使用してアプリケーションを追加するためのガイダンスを示します。

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、ルールの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-shellscript
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
```

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

- **fapolicyd-cli(1)** の man ページ

13.7. FAPOLICYD RHEL システムロールを使用して未知のコード実行に対する保護を設定

fapolicyd システムロールを使用すると、Ansible Playbook を実行して不明なコードの実行を防ぐことができます。

前提条件

- 制御ノードと管理ノードを準備している
- 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: Preventing execution of unknown code
  hosts: all
  vars:
    fapolicyd_setup_integrity: sha256
    fapolicyd_setup_trust: rpmdb,file
    fapolicyd_add_trusted_file:
      - </usr/bin/my-ls>
      - </opt/third-party/app1>
      - </opt/third-party/app2>
  roles:
    - rhel-system-roles.fapolicyd
```

linux-system-roles.fapolicyd RHEL システムロールの次の変数を使用すると、保護をさらにカスタマイズできます。

fapolicyd_setup_integrity

整合性のタイプとして、**none**、**sha256**、**size** のいずれかを設定できます。

fapolicyd_setup_trust

信頼ファイルのタイプ **file**、**rpmd**、**deb** を設定できます。

fapolicyd_add_trusted_file

信頼できる実行可能ファイル、および **fapolicyd** によって実行を防止しない実行可能ファイルのリストを指定できます。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.fapolicyd/README.md** ファイル

13.8. 関連情報

- **man -k fapolicyd** コマンドを使用してリスト表示される **fapolicyd** 関連の man ページ。
- FOSDEM 2020 fapolicyd プレゼンテーション。

第14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。

14.1. 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インターフェイス
- 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)、**IPCAllowedUsers** オプション、および **IPCAllowedGroups** 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。

14.2. 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。

```
# yum 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** サービスが実行してい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

14.3. 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ており、実行している。

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
"4PHGcaDKWtPjKDwYpIRG722cB9SlGz9l9lea93+Gt9c=" via-port "usb1" with-interface
09:00:00
...
6: block id 1b1c:1ab1 serial "000024937962" name "Voyager" hash
```

```
"CrXgiaWIf2bZAU+5WkzOE7y0rdSO82XMzubn7HDb95Q=" 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。

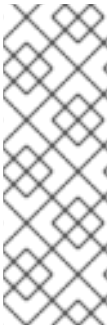
```
# usbguard allow-device <6>
```

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対話しません。

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

- **usbguard(1)** の man ページ
- **usbguard --help** コマンド

14.4. 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ており、実行している。

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
"4PHGcaDKWtPjKDwYpIRG722cB9SlGz9l9lea93+Gt9c=" via-port "usb1" with-interface
09:00:00
...
6: block id 1b1c:1ab1 serial "000024937962" name "Voyager" hash
"CrXgiaWIf2bZAU+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対話しません。

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

- **usbguard(1)** の 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。

14.5. 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ており、実行している。
- **/etc/usbguard/rules.conf** ファイルには、**usbguard generate-policy** コマンドで生成した初期ルールセットが含まれます。

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

- **usbguard-rules.conf(5)** man ページ

14.6. USB デバイス用に構造化されたカスタムポリシーの作成

カスタム USBGuard ポリシーは、**/etc/usbguard/rules.d/** ディレクトリー内の複数の **.conf** ファイルで整理できます。次に、**usbguard-daemon** は、メインの **rules.conf** ファイルを、ディレクトリー内の **.conf** ファイルをアルファベット順で組み合わせます。

前提条件

- **usbguard** サービスがインストールされており、実行している。

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、新しい **.conf** ファイル (例: **policy.conf**) ファイルに保存します。

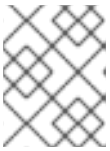
```
# usbguard generate-policy --no-hashes > ./policy.conf
```

--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 **rules.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/WSCocgiuQlVs6Dn0VEza7KiHoDeTz0fyg=" parent-hash
"2i6ZBJfTl5BakXF7Gba84/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

- **usbguard-rules.conf(5)** man ページ

14.7. 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認可

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ており、実行している。
- **/etc/usbguard/rules.conf** ファイルには、**usbguard generate-policy** コマンドで生成した初期ルールセットが含まれます。

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

14.8. 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ており、実行している。
- **auditd** サービスが実行している。

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** デーモンログを USB 認可イベントに対してクエリーします。次に例を示します。

```
# ausearch -ts recent -m USER_DEVICE
```

関連情報

- **usbguard-daemon.conf(5)** の man ページ

14.9. 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