Red Hat Quay 3.8

Red Hat Quay の Clair による脆弱性レポート

Red Hat Quay の Clair による脆弱性レポート

Last Updated: 2024-01-16

Red Hat Quay 3.8 Red Hat Quay の Clair による脆弱性レポート

Red Hat Quay の Clair による脆弱性レポート

法律上の通知

概要

Red Hat Quay を使い始める

目次

はじめに

このガイドでは、Red Hat Quay の Clair の概要、スタンドアロンの Red Hat Quay および Operator デプロイメントでの Clair の実行、および高度な Clair 設定を説明します。

パート I. RED HAT QUAY の概要に関する CLAIR による脆弱性レポート

このガイドでは、Red Hat Quay での Clair の主な目的と概念を説明します。また、Clair のリリースと公式の Clair コンテナーの場所に関する情報も含まれます。

第1章 RED HAT QUAY の CLAIR

Clair v4 (Clair) は、静的コード分析を活用してイメージコンテンツを解析し、コンテンツに影響を与える脆弱性を報告するオープンソースアプリケーションです。Clair は Red Hat Quay にパッケージ化されており、スタンドアロンと Operator デプロイメントの両方で使用できます。エンタープライズ環境に合わせてコンポーネントを個別にスケーリングできる、非常にスケーラブルな設定で実行できます。

1.1. CLAIR 脆弱性データベース

Clair は、次の脆弱性データベースを使用して、イメージの問題を報告します。

- Ubuntu Oval データベース
- Debian Oval データベース
- Red Hat Enterprise Linux (RHEL) Oval データベース
- SUSE Oval データベース
- Oracle Oval データベース
- アルパイン SecDB データベース
- VMWare Photon OS データベース
- Amazon Web Services (AWS) UpdateInfo
- Pyup.io (Python) データベース

Clair がさまざまなデータベースでセキュリティーマッピングを行う方法は、ClairCore Severity Mapping を参照してください。

第2章 CLAIR の概念

次のセクションでは、Clair がどのように機能するかの概念的な概要を示します。

2.1. CLAIR の実践

Clair の分析は、インデックス作成、マッチング、通知の 3 つの部分に分類されます。

2.1.1. インデックス化

Clair のインデクサーサービスは、マニフェストのインデックス作成を実行します。Clair では、マニフェストはコンテナーイメージの表現です。インデクサーサービスは、Clair がレイヤーのコンテンツを理解するために使用するコンポーネントです。Clair は、重複作業を減らすために、Open Container Initiative (OCI) マニフェストとレイヤーがコンテンツアドレス可能であるという事実を利用しています。

インデックス作成には、コンテナーイメージを表すマニフェストの取得と、その設定要素の計算が含まれます。インデクサーは、イメージ内に存在するパッケージ、イメージが派生したディストリビューション、およびイメージ内で使用されているパッケージリポジトリーを検出しようとします。この情報が計算されると、**IndexReport** に永続化されます。

IndexReport は Clair のデータベースに保存されます。 **matcher** ノードにフィードして、脆弱性レポートを計算できます。

2.1.1.1. コンテンツアドレス可能性

Clair は、すべてのマニフェストとレイヤーを **コンテンツアドレス可能** として処理します。Clair のコンテキストでは、コンテンツアドレス可能とは、特定のマニフェストがインデックス化されると、必要でないかぎり、再度インデックス化されないことを意味します。これは個々のレイヤーでも同じです。

たとえば、レジストリー内で **ubuntu:artful** をベースレイヤーとして使用するイメージの数を考えてみましょう。開発者がイメージを Ubuntu に基づいて作成する場合は、それがイメージの大部分になる可能性があります。レイヤーとマニフェストをコンテンツアドレス可能として処理するということは、Clair がベースレイヤーを 1 回だけフェッチして分析することを意味します。

場合によっては、Clair がマニフェストのインデックスを再作成する必要があります。たとえば、パッケージスキャナーなどの内部コンポーネントが更新されると、Clair は新しいパッケージスキャナーで分析を実行します。Clair は、コンポーネントが変更され、2 回目は **IndexReport** が異なる可能性があることを判断するのに十分な情報を持っているため、マニフェストのインデックスを再作成します。

クライアントは、Clair の **index_state** エンドポイントを追跡して、内部コンポーネントがいつ変更されたかを把握し、その後再インデックスを発行できます。Clair の API 仕様を表示する方法については、Clair API ガイドを参照してください。

2.1.2. マッチング

Clair では、マッチャーノードが提供された **IndexReport** と脆弱性をマッチングします。

マッチャーは脆弱性のデータベースを最新の状態に維持します。通常、マッチャーは一連のアップデーターを実行し、定期的にデータソースをプローブして新しいコンテンツを探します。新しい脆弱性は、発見されると、データベースに保存されます。

マッチャー API は、頻繁に使用されるように設計されています。クエリーを実行すると、常に最新の **VulnerabilityReport** が提供されるように設計されています。**VulnerabilityReport** は、マニフェストのコンテンツと、コンテンツに影響を与える脆弱性の両方をまとめたものです。

2.1.2.1. リモートマッチング

リモートマッチャーはマッチャーと同様に機能しますが、リモートマッチャーは API 呼び出しを使用して、提供された **IndexReport** の脆弱性データをフェッチします。リモートマッチャーは、特定のソースからデータベースにデータを永続化できない場合に役立ちます。

CRDA リモートマッチャーは、Red Hat Code Ready Dependency Analytics (CRDA) から脆弱性をフェッチします。デフォルトでは、このマッチャーは 1 分あたり 100 リクエストを処理します。レート制限は、API キーリクエストフォーム を送信して専用の API キーをリクエストすることで解除できます。

CRDA リモートマッチングを有効にするには、「Clair の CRDA を有効にする」を参照してください。

2.1.3. 通知

Clair は、新しいセキュリティーデータベースの更新を追跡し、新しい脆弱性または削除された脆弱性がインデックス付きマニフェストに影響を与えるかどうかをユーザーに通知するノーティファイアーサービスを使用します。

ノーティファイアーは、以前にインデックス付けされたマニフェストに影響を与える新しい脆弱性を認識すると、**config.yaml** ファイルで設定されたメソッドを使用して、新しい変更に関する通知を発行します。返された通知は、変更により発見された最も深刻な脆弱性を表しています。これにより、同じセキュリティーデータベースの更新に対して過剰な通知が作成されるのを回避できます。

ユーザーが通知を受け取ると、マッチャーに対して新しいリクエストを発行して、最新の脆弱性レポートを受け取ります。

通知スキーマは、以下の型を JSON でマーシャリングしたものであす。

```
// Reason indicates the catalyst for a notification
type Reason string
const (
    Added   Reason = "added"
    Removed Reason = "removed"
    Changed Reason = "changed"
)
type Notification struct {
    ID           uuid.UUID        `json:"id"`
    Manifest     claircore.Digest `json:"manifest"`
    Reason       Reason           `json:"reason"`
    Vulnerability VulnSummary     `json:"vulnerability"`
}
type VulnSummary struct {
    Name           string                `json:"name"`
    Description    string                `json:"description"`
    Package        *claircore.Package    `json:"package,omitempty"`
    Distribution   *claircore.Distribution `json:"distribution,omitempty"`
    Repo           *claircore.Repository  `json:"repo,omitempty"`
    Severity       string                `json:"severity"`
    FixedInVersion string                `json:"fixed_in_version"`
    Links          string                `json:"links"`
}
```

次の方法で通知に登録できます。

- Webhook 配信

- AMQP 配信
- STOMP 配信

ノーティファイアーの設定は、Clair YAML 設定ファイルを介して行われます。

2.1.3.1. Webhook 配信

Webhook 配信のノーティファイアーを設定する場合は、サービスに次の情報を提供します。

- Webhook が起動するターゲット URL。
- API パスを含む、ノーティファイアーに到達する可能性があるコールバック URL。たとえば、**http://clair-notifier/notifier/api/v1/notifications** です。

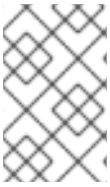
ノーティファイアーは、更新されたセキュリティーデータベースが変更され、インデックス付きマニフェストの影響を受けるステータスが変更されたと判断すると、次の JSON 本文を設定済みのターゲットに配信します。

```
{
  "notifiction_id": {uuid_string},
  "callback": {url_to_notifications}
}
```

受信すると、サーバーはコールバックフィールドに指定された URL を参照できます。

2.1.3.2. AMQP 配信

Clair ノーティファイアーは、AMQP ブローカーへの通知の配信もサポートしています。AMQP 配信を使用すると、コールバックをブローカーに配信するか、通知をキューに直接配信するかを制御できます。これにより、AMQP コンシューマーの開発者は、通知処理のロジックを決定できます。



注記

AMQP 配信は、AMQP 0.x プロトコル (RabbitMQ など) のみをサポートします。AMQP 1.x メッセージキュー (ActiveMQ など) に通知を発行する必要がある場合は、STOMP 配信を使用できます。

2.1.3.2.1. AMQP 直接配信

Clair ノーティファイアーの設定で AMQP 配信に **direct: true** が指定されている場合、通知は設定された交換に直接配信されます。

direct が設定されている場合、**rollup** プロパティが設定され、1 回の AMQP で最大数の通知を送信するように、ノーティファイアーに指示することができます。これにより、メッセージのサイズとキューに配信されるメッセージの数のバランスが取れます。

2.1.3.3. ノーティファイアーのテストおよび開発モード

ノーティファイアーには、**NOTIFIER_TEST_MODE** パラメーターで有効にできるテストおよび開発モードがあります。このパラメーターは、任意の値に設定できます。

NOTIFIER_TEST_MODE パラメーターが設定されている場合、ノーティファイアーは、**poll_interval** 間隔ごとに、設定された配信機能にフェイクの通知を送信し始めます。これにより、新規または既存のデリバラーを簡単に実装およびテストできます。

ノーティファイアーは、環境変数がクリアされてサービスが再起動されるまで、**NOTIFIER_TEST_MODE** で実行されます。

2.1.3.4. 通知の削除

通知を削除するには、**DELETE** API 呼び出しを使用できます。通知 ID を手動で削除すると、ノーティファイアー内のリソースがクリーンアップされます。**DELETE** API 呼び出しを使用しない場合、ノーティファイアーは、配信された通知をデータベースから消去する前に、所定の時間待機します。

2.2. CLAIR 認証

現在のイテレーションでは、Clair v4 (Clair) は認証を内部で処理します。

注記

Clair の以前のバージョンでは、JWT Proxy を使用して認証を制御していました。

認証は、設定の **auth** キーの下に設定オブジェクトを指定することによって設定されます。複数の認証設定が存在する場合がありますが、次の順序で優先的に使用されます。

1. PSK。この認証設定により、Clair は事前共有キーを使用して JWT ベースの認証を実装します。
2. 設定以下に例を示します。

```
auth:
  psk:
    key: >-
      MDQ4ODBlNDAtNDc0ZC00MWUxLThhMzAtOTk0MzEwMGQwYTMxCg==
    iss: 'issuer'
```

この設定では、**auth** フィールドに 2 つのパラメーターが必要です。**iss** は、すべての受信リクエストを検証する発行者であり、**key** は、リクエストを検証するための base64 コード化された対称鍵です。

2.3. CLAIR アップデーター

Clair は、さまざまな脆弱性データベースを取得して解析するロジックを含む、**アップデーター** と呼ばれる **Go** パッケージを使用しています。

通常、アップデーターはマッチャーとペアになって、脆弱性がパッケージに関連しているかどうか、およびどのように関連しているかを解釈します。管理者は、脆弱性データベースの更新頻度を減らす場合も、使用されないことがわかっているデータベースから脆弱性をインポートしないようにする場合もあります。

2.3.1. アップデーターの設定

アップデーターは、設定の上部にある **updaters** キーによって設定できます。アップデーターがマッチャープロセス内で自動的に実行されている場合 (既定の設定)、アップデーターを実行する期間はマッチャーの設定フィールドで設定されます。

2.3.1.1. アップデーターセット

次のセットは、Clair アップデーターで設定できます。

- **alpine**
- **aws**
- **debian**
- **enricher/cvss**
- **libvuln/driver**
- **oracle**
- **photon**
- **pyupio**
- **rhel**
- **rhel/rhcc**
- **suse**
- **ubuntu**
- **updater**

2.3.1.2. アップデーターセットの選択

アップデーターの特定のセットは、**sets** リストで選択できます。以下に例を示します。

```
updaters:
  sets:
    - rhel
```

sets フィールドが入力されていない場合、デフォルトですべてのセットが使用されます。

2.3.1.3. アップデーターセットのフィルタリング

セット全体を無効にすることなくアップデーターの実行を拒否するには、**filter** オプションを使用できます。

次の例では、文字列は Go **regexp** パッケージとして解釈されます。これにより、名前がマッチングしないアップデーターは拒否されます。

注記

これは、空の文字列が任意の文字列にマッチングすることを意味します。文字列にマッチングしないという意味ではありません。

```
updaters:
  filter: '^$'
```

2.3.1.4. 特定のアップデーターの設定

特定のアップデーターの設定は、**updaters** オブジェクトの **config** パラメーターの下にキーを配置することで渡すことができます。アップデーターの名前は動的に作成される場合があるため、ユーザーはログを調べて、アップデーター名が正確であることを確認する必要があります。アップデーターが期待する特定のオブジェクトについては、アップデーターのドキュメントで説明する必要があります。

次の例では、**rhel** アップデーターは別の場所からマニフェストをフェッチします。

```
updaters:
  config:
    rhel:
      url: https://example.com/mirror/oval/PULP_MANIFEST
```

2.3.1.5. Clair アップデーターコンポーネントの無効化

一部のシナリオでは、Clair アップデーターコンポーネントを無効にする場合があります。切断された環境で Red Hat Quay を実行する場合は、アップデーターを無効にする必要があります。

次の例では、Clair アップデーターが無効になっています。

```
matcher:
  disable_updaters: true
```

2.3.2. Clair アップデーター URL

以下は、Clair がデフォルト設定で対話を試みる HTTP ホストおよびパスです。このリストは網羅的なものではありません。一部のサーバーはリダイレクトを発行し、一部のリクエスト URL は動的に構築されます。

- https://secdb.alpinelinux.org/
- http://repo.us-west-2.amazonaws.com/2018.03/updates/x86_64/mirror.list
- https://cdn.amazonlinux.com/2/core/latest/x86_64/mirror.list
- https://www.debian.org/security/oval/
- https://linux.oracle.com/security/oval/
- https://packages.vmware.com/photon/photon_oval_definitions/
- https://github.com/pyupio/safety-db/archive/
- https://catalog.redhat.com/api/containers/
- https://www.redhat.com/security/data/
- https://support.novell.com/security/oval/
- https://people.canonical.com/~ubuntu-security/oval/

第3章 CLAIR について

このセクションのコンテンツでは、Clair のリリース、公式の Clair コンテナー、および CVSS エンリッチメントデータに関する情報に焦点を当てています。

3.1. CLAIR のリリース

Clair の新しいバージョンは定期的にリリースされます。Clair のビルドに必要なソースコードは、アーカイブとしてパッケージ化され、各リリースに添付されています。Clair のリリースは、Clair のリリース にあります。

リリースアーティファクトには、オープンホストを使用してインターネットからアップデーターデータを取得する **clairctl** コマンドラインインターフェイスツールも含まれています。

3.2. CLAIR がサポートする言語

Clair は * Python * Java (CRDA を有効にする必要があります) をサポートしています。

3.3. CLAIR コンテナー

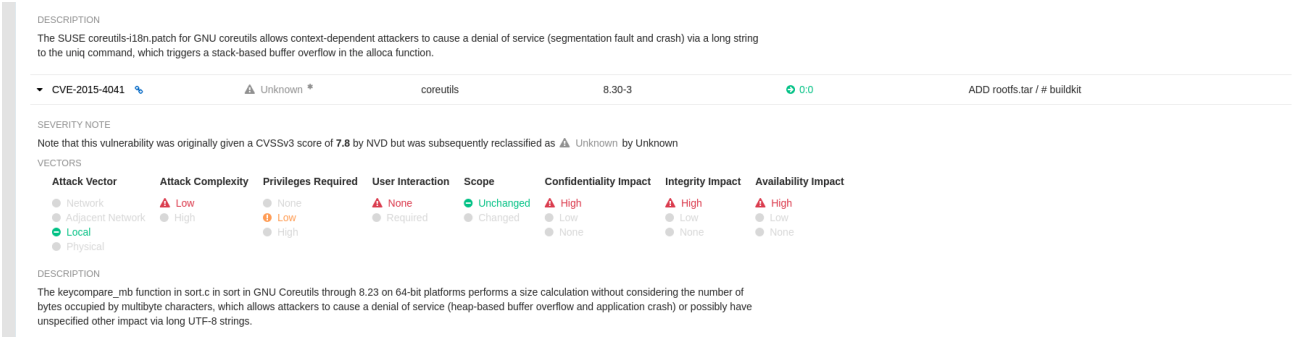
Red Hat Quay にバンドルされている公式のダウンストリーム Clair コンテナーは、Red Hat Ecosystem Catalog にあります。

公式のアップストリームコンテナーはパッケージ化され、Quay.io/projectquay/clair でコンテナーとしてリリースされます。最新のタグは、Git 開発ブランチを追跡します。バージョンタグは、対応するリリースから構築されます。

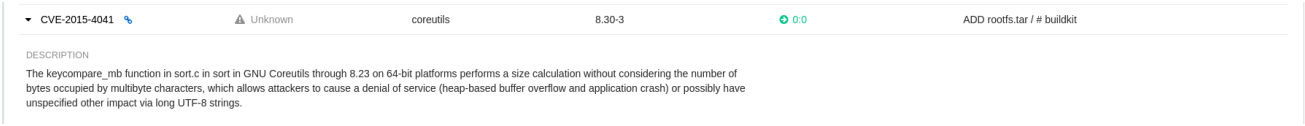
3.4. 脆弱性情報データベース (NVD: NATIONAL VULNERABILITY DATABASE) の CVE 評価

Clair v4.2 の時点で、Common Vulnerability Scoring System (CVSS) 強化データが Red Hat Quay UI で表示できるようになりました。さらに、Clair v4.2 は、検出された脆弱性について National Vulnerability Database から CVSS スコアを追加します。

今回の変更により、脆弱性の CVSS スコアがディストリビューションスコアの 2 レベル以内である場合、Red Hat Quay UI はデフォルトでディストリビューションのスコアを提示します。以下に例を示します。



これは以前のインターフェイスとは異なり、以下の情報のみを表示します。



3.5. 連邦情報処理標準 (FIPS) の準備と準拠

米国国立標準技術研究所 (NIST) によって開発された連邦情報処理標準 (FIPS) は、特に銀行、医療、公共部門などの高度に規制された分野で、機密データを保護および暗号化するために高く評価されていると見なされています。Red Hat Enterprise Linux (RHEL) および OpenShift Container Platform は FIPS **モード** を提供することで FIPS 標準をサポートします。このモードでは、システムは **openssl** などの特定の FIPS 検証済み暗号モジュールの使用のみを許可します。これにより、FIPS への準拠が保証されます。

Red Hat Quay は、Red Hat Quay バージョン 3.5.0 からの FIPS 対応の RHEL および OpenShift Container Platform 環境での実行をサポートします。

パート II. RED HAT QUAY の CLAIR

このガイドには、スタンドアロンおよび OpenShift Container Platform Operator デプロイメントの両方で Red Hat Quay で Clair を実行するための手順が含まれています。

第4章 スタンドアロンの RED HAT QUAY デプロイメントでの CLAIR のセットアップ

スタンドアロンの Red Hat Quay デプロイメントの場合、Clair を手動でセットアップできます。

手順

1. Red Hat Quay インストールディレクトリーに、Clair データベースデータ用の新しいディレクトリーを作成します。

```
$ mkdir /home/<user-name>/quay-poc/postgres-clairv4
```

2. 次のコマンドを入力して、**postgres-clairv4** ファイルに適切な権限を設定します。

```
$ setfacl -m u:26:-wx /home/<user-name>/quay-poc/postgres-clairv4
```

3. 次のコマンドを入力して、Clair Postgres データベースをデプロイします。

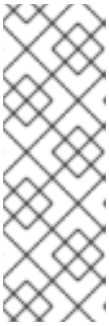
```
$ sudo podman run -d --name postgresql-clairv4 \
  -e POSTGRESQL_USER=clairuser \
  -e POSTGRESQL_PASSWORD=clairpass \
  -e POSTGRESQL_DATABASE=clair \
  -e POSTGRESQL_ADMIN_PASSWORD=adminpass \
  -p 5433:5433 \
  -v /home/<user-name>/quay-poc/postgres-clairv4:/var/lib/pgsql/data:Z \
  {postgresimage}
```

4. Clair デプロイメント用に Postgres **uuid-ossp** モジュールをインストールします。

```
$ podman exec -it postgresql-clairv4 /bin/bash -c 'echo "CREATE EXTENSION IF NOT EXISTS \"uuid-ossp\"" | psql -d clair -U postgres'
```

出力例

```
CREATE EXTENSION
```



注記

Clair では、**uuid-ossp** 拡張機能を Postgres データベースに追加する必要があります。適切な権限を持つユーザーの場合は、拡張機能を作成すると、Clair によって自動的に追加されます。ユーザーが適切な権限を持っていない場合は、Clair を開始する前に拡張機能を追加する必要があります。

拡張機能が存在しない場合は、Clair が起動しようとすると、**ERROR: Please load the "uuid-ossp" extension.(SQLSTATE 42501)** エラーが発生します。

5. 実行中の場合は、**Quay** コンテナーを停止し、設定モードで再始動して、既存の設定をボリュームとしてロードします。

```
$ sudo podman run --rm -it --name quay_config \
  -p 80:8080 -p 443:8443 \
  -v $QUAY/config:/conf/stack:Z \
```

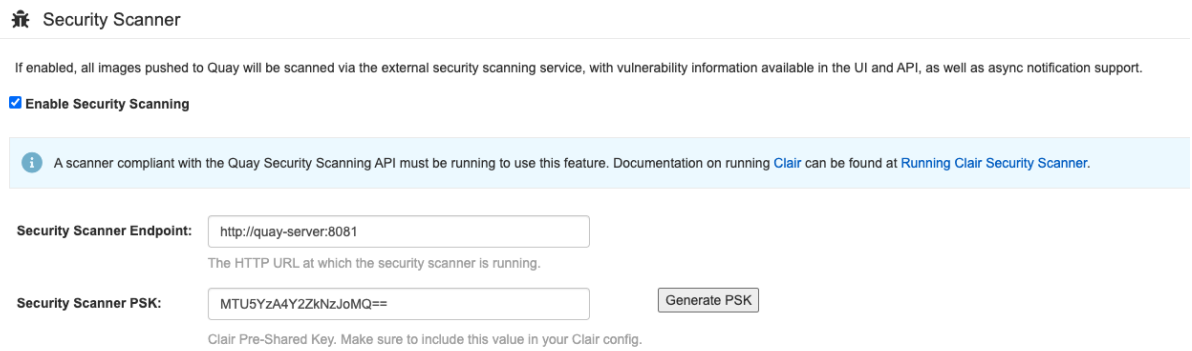
```
{productrepo}/{quayimage}:{productminv} config secret
```

6. 設定ツールにログインし、UI の **Security Scanner** セクションで **Enable Security Scanning** をクリックします。

7. **quay-server** システムでまだ使用されていないポート (**8081** など) を使用して、Clair の HTTP エンドポイントを設定します。

8. **Generate PSK** ボタンを使用して、事前共有キー (PSK) を作成します。

   セキュリティースキャナー **UI**



9. Red Hat Quay の **config.yaml** ファイルを検証してダウンロードし、設定エディターを実行している **Quay** コンテナーを停止します。

10. 新しい設定バンドルを Red Hat Quay インストールディレクトリーに展開します。次に例を示します。

```
$ tar xvf quay-config.tar.gz -d /home/<user-name>/quay-poc/
```

11. Clair 設定ファイル用のフォルダーを作成します。次に例を示します。

```
$ mkdir /etc/opt/clairv4/config/
```

12. Clair 設定フォルダーに移動します。

```
$ cd /etc/opt/clairv4/config/
```

13. 以下のように、Clair 設定ファイルを作成します。

```
http_listen_addr: :8081
introspection_addr: :8088
log_level: debug
indexer:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
password=clairpass sslmode=disable
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
matcher:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
password=clairpass sslmode=disable
  max_conn_pool: 100
  run: ""
```

```
  migrations: true
  indexer_addr: clair-indexer
notifier:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
password=clairpass sslmode=disable
  delivery_interval: 1m
  poll_interval: 5m
  migrations: true
auth:
  psk:
    key: "MTU5YzA4Y2ZkNzJoMQ=="
    iss: ["quay"]
# tracing and metrics
trace:
  name: "jaeger"
  probability: 1
  jaeger:
    agent_endpoint: "localhost:6831"
    service_name: "clair"
metrics:
  name: "prometheus"
```

Clair の設定形式の詳細は、Clair 設定リファレンス を参照してください。

14. コンテナーイメージを使用して Clair を起動し、作成したファイルから設定にマウントします。

```
$ sudo podman run -d --name clairv4 \
-p 8081:8081 -p 8088:8088 \
-e CLAIR_CONF=/clair/config.yaml \
-e CLAIR_MODE=combo \
-v /etc/opt/clairv4/config:/clair:Z \
registry.redhat.io/quay/clair-rhel8:v3.8.15
```

注記

複数の Clair コンテナーを実行することもできますが、単一のコンテナーを超えるデプロイシナリオでは、Kubernetes や OpenShift Container Platform などのコンテナーオーケストレーターを使用することが強く推奨されます。

第5章 OPENSHIFT CONTAINER PLATFORM の CLAIR

OpenShift Container Platform 上の Red Hat Quay デプロイメントで Clair v4 (Clair) をセットアップするには、Red Hat Quay Operator を使用することが推奨されます。デフォルトでは、Red Hat Quay Operator は、Clair デプロイメントを Red Hat Quay デプロイメントとともにインストールまたはアップグレードし、Clair を自動的に設定します。

第6章 CLAIR のテスト

以下の手順を使用して、スタンドアロンの Red Hat Quay デプロイメントまたは OpenShift Container Platform Operator ベースのデプロイメントで Clair をテストします。

前提条件

- Clair コンテナーイメージをデプロイしました。

手順

1. 次のコマンドを入力して、サンプルイメージをプルします。

```
$ podman pull ubuntu:20.04
```

2. 次のコマンドを入力して、レジストリーにイメージをタグ付けします。

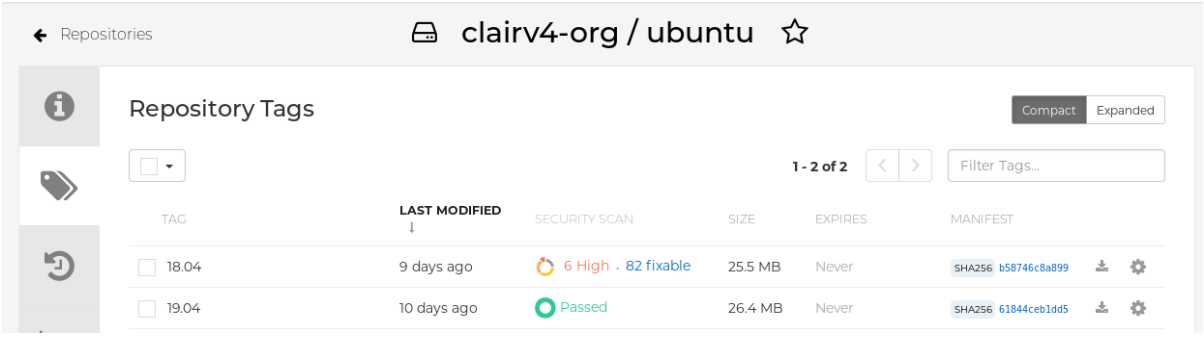
```
$ sudo podman tag docker.io/library/ubuntu:20.04 <quay-server.example.com>/<user-name>/ubuntu:20.04
```

3. 以下のコマンドを入力して、イメージを Red Hat Quay レジストリーにプッシュします。

```
$ sudo podman push --tls-verify=false quay-server.example.com/quayadmin/ubuntu:20.04
```

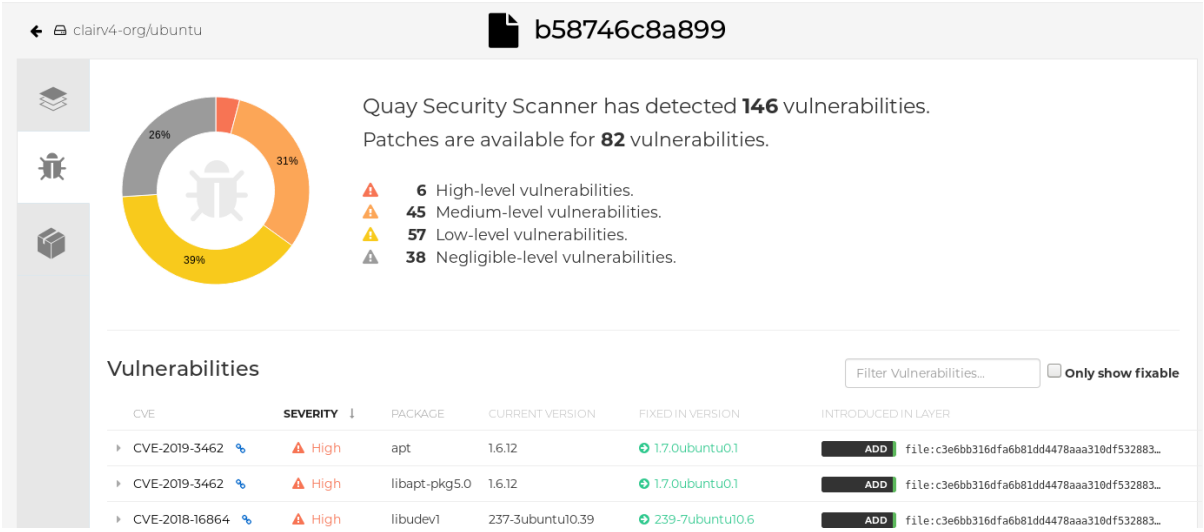
4. UI から Red Hat Quay デプロイメントにログインします。
5. リポジトリー名 (**quayadmin/ubuntu** など) をクリックします。
6. ナビゲーションウィンドウで、**タグ** をクリックします。

   レポートの概要



7. イメージレポート (例: **45 medium**) をクリックして、より詳細なレポートを表示します。

   レポートの詳細


clairv4-org/ubuntu
b58746c8a899
26%
31%
39%
Quay Security Scanner has detected 146 vulnerabilities.
Patches are available for 82 vulnerabilities.
6 High-level vulnerabilities.
45 Medium-level vulnerabilities.
57 Low-level vulnerabilities.
38 Negligible-level vulnerabilities.
Vulnerabilities
Filter Vulnerabilities...
Only show fixable
CVE
SEVERITY
PACKAGE
CURRENT VERSION
FIXED IN VERSION
INTRODUCED IN LAYER
CVE-2019-3462
High
apt
1.6.12
1.7.0ubuntu0.1
ADD
file:c3e6bb316dfa6b81dd4478aaa310df532883...
CVE-2019-3462
High
libapt-pkg5.0
1.6.12
1.7.0ubuntu0.1
ADD
file:c3e6bb316dfa6b81dd4478aaa310df532883...
CVE-2018-16864
High
libudev1
237-3ubuntu10.39
239-7ubuntu10.6
ADD
file:c3e6bb316dfa6b81dd4478aaa310df532883...

パート III. 高度な CLAIR 設定

このセクションを使用して、高度な Clair 機能を設定します。

第7章 アンマネージド CLAIR 設定

Red Hat Quay ユーザーは、Red Hat Quay OpenShift Container Platform Operator を使用してアンマネージド Clair 設定を実行できます。この機能により、ユーザーはアンマネージド Clair データベースを作成したり、アンマネージドデータベースなしでカスタム Clair 設定を実行したりできます。

アンマネージド Clair データベースにより、Red Hat Quay オペレーターは、Operator の複数のインスタンスが同じデータベースと通信する必要がある地理的に複製された環境で作業できます。アンマネージド Clair データベースは、ユーザーがクラスターの外部に存在する高可用性 (HA) Clair データベースを必要とする場合にも使用できます。

7.1. アンマネージド CLAIR データベースを使用したカスタム CLAIR 設定の実行

次の手順を使用して、Clair データベースをアンマネージドに設定します。

手順

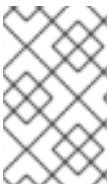
- Quay Operator で、**QuayRegistry** カスタムリソースの **clairpostgres** コンポーネントを **managed: false** に設定します。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: quay370
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: clairpostgres
      managed: false
```

7.2. アンマネージド CLAIR データベースを使用したカスタム CLAIR データベースの設定

Red Hat Quay Operator for OpenShift Container Platform を使用すると、ユーザーは独自の Clair データベースを提供できます。

次の手順を使用して、カスタム Clair データベースを作成します。



注記

次の手順では、SSL/TLS 証明書を使用して Clair をセットアップします。SSL/TSL 証明書を使用して Clair をセットアップしない同様の手順を表示するには、マネージド Clair 設定を使用したカスタム Clair データベースの設定を参照してください。

手順

1. 次のコマンドを入力して、**clair-config.yaml** を含む Quay 設定バンドルシークレットを作成します。

```
$ oc create secret generic --from-file config.yaml=./config.yaml --from-file extra_ca_cert_rds-ca-2019-root.pem=./rds-ca-2019-root.pem --from-file clair-config.yaml=./clair-config.yaml --from-file ssl.cert=./ssl.cert --from-file ssl.key=./ssl.key config-bundle-secret
```

Clair config.yaml ファイルの例

```
indexer:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
    layer_scan_concurrency: 6
    migrations: true
    scanlock_retry: 11
log_level: debug
matcher:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
    migrations: true
metrics:
    name: prometheus
notifier:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
    migrations: true
```

注記

- データベース証明書は、**clair-config.yaml** の Clair アプリケーション Pod の **/run/certs/rds-ca-2019-root.pem** の下にマウントされます。**clair-config.yaml** を設定するときに指定する必要があります。
- **clair-config.yaml** の例は、OpenShift 設定の Clair にあります。

2. **clair-config.yaml** ファイルをバンドルシークレットに追加します。次に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: <base64 encoded Quay config>
  clair-config.yaml: <base64 encoded Clair config>
  extra_ca_cert_<name>: <base64 encoded ca cert>
  ssl.crt: <base64 encoded SSL certificate>
  ssl.key: <base64 encoded SSL private key>
```

注記

更新すると、提供された **clair-config.yaml** ファイルが Clair Pod にマウントされます。提供されていないフィールドには、Clair 設定モジュールを使用してデフォルトが自動的に入力されます。

3. **Build History** ページでコミットをクリックするか、**oc get pods -n <namespace>** を実行して、Clair Pod のステータスを確認できます。以下に例を示します。

```
$ oc get pods -n <namespace>
```

出力例

```
NAME                                                  READY   STATUS    RESTARTS   AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2   1/1     Running   0          7s
```

第8章 マネージド CLAIR データベースを使用したカスタム CLAIR 設定の実行

場合によっては、マネージド Clair データベースを使用してカスタム Clair 設定を実行します。これは、以下のシナリオで役に立ちます。

- ユーザーが特定のアップデーターリソースを無効にする場合。
- ユーザーが切断された環境で Red Hat Quay を実行している場合。切断された環境での Clair の実行の詳細は、エアギャップ OpenShift クラスターでの Clair データベースへのアクセスの設定 を参照してください。

注記

- 切断された環境で Red Hat Quay を実行している場合は、**clair-config.yaml** の **airgap** パラメーターを **true** に設定する必要があります。
- 切断された環境で Red Hat Quay を実行している場合は、すべてのアップデーターコンポーネントを無効にする必要があります。

8.1. CLAIR データベースをマネージドに設定する

次の手順を使用して、Clair データベースをマネージドに設定します。

手順

- Quay Operator で、**QuayRegistry** カスタムリソースの **clairpostgres** コンポーネントを **managed: true** に設定します。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: quay370
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: clairpostgres
      managed: true
```

8.2. マネージド CLAIR 設定を使用したカスタム CLAIR データベースの設定

Red Hat Quay Operator for OpenShift Container Platform を使用すると、ユーザーは独自の Clair データベースを提供できます。

次の手順を使用して、カスタム Clair データベースを作成します。

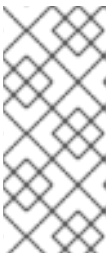
手順

1. 次のコマンドを入力して、**clair-config.yaml** を含む Quay 設定バンドルシークレットを作成します。

```
$ oc create secret generic --from-file config.yaml=./config.yaml --from-file extra_ca_cert_rds-ca-2019-root.pem=./rds-ca-2019-root.pem --from-file clair-config.yaml=./clair-config.yaml config-bundle-secret
```

Clair config.yaml ファイルの例

```
indexer:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb password=quayrdsdb sslmode=disable
    layer_scan_concurrency: 6
    migrations: true
    scanlock_retry: 11
log_level: debug
matcher:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb password=quayrdsdb sslmode=disable
    migrations: true
metrics:
    name: prometheus
notifier:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb password=quayrdsdb sslmode=disable
    migrations: true
```

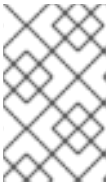


注記

- データベース証明書は、**clair-config.yaml** の Clair アプリケーション Pod の **/run/certs/rds-ca-2019-root.pem** の下にマウントされます。**clair-config.yaml** を設定するときに指定する必要があります。
- **clair-config.yaml** の例は、OpenShift 設定の Clair にあります。

2. **clair-config.yaml** ファイルをバンドルシークレットに追加します。次に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: <base64 encoded Quay config>
  clair-config.yaml: <base64 encoded Clair config>
```



注記

- 更新すると、提供された **clair-config.yaml** ファイルが Clair Pod にマウントされます。提供されていないフィールドには、Clair 設定モジュールを使用してデフォルトが自動的に入力されます。

3. **Build History** ページでコミットをクリックするか、**oc get pods -n <namespace>** を実行し

3. **Build History** ページでコミットをクリックするか、**oc get pods -n <namespace>** を実行して、Clair Pod のステータスを確認できます。以下に例を示します。

```
$ oc get pods -n <namespace>
```

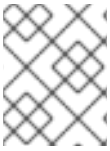
出力例

```
NAME                                                READY   STATUS    RESTARTS   AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2    1/1     Running   0          7s
```

第9章 切断された環境での CLAIR

Clair は、**アップデーター** と呼ばれる一連のコンポーネントを使用して、さまざまな脆弱性データベースからのデータのフェッチと解析を処理します。アップデーターはデフォルトで、脆弱性データをインターネットから直接プルし、すぐに使用できるように設定されています。ただし、一部のユーザーは、切断された環境またはインターネットに直接アクセスできない環境で Red Hat Quay を実行する必要がある場合があります。Clair は、ネットワーク分離を考慮したさまざまな種類の更新ワークフローを使用することで、切断された環境をサポートします。これは **clairctl** コマンドラインインターフェイスツールを使用して機能します。このツールは、オープンホストを使用してインターネットからアップデーターデータを取得し、そのデータを隔離されたホストにセキュアに転送してから、隔離されたホスト上のアップデーターデータを Clair にインポートします。

このガイドを使用して、切断された環境で Clair をデプロイします。



注記

現在、Clair エンリッチメントデータは CVSS データです。エンリッチメントデータは現在、オフライン環境ではサポートされていません。

Clair アップデーターの詳細は、Clair アップデーターを参照してください。

9.1. 切断された OPENSHIFT CONTAINER PLATFORM クラスターでの CLAIR のセットアップ

以下の手順を使用して、切断された OpenShift Container Platform クラスターに OpenShift Container Platform でプロビジョニングされた Clair Pod をセットアップします。

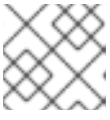
9.1.1. OpenShift Container Platform デプロイメント用の clairctl コマンドラインユーティリティーツールのインストール

以下の手順を使用して、OpenShift Container Platform デプロイメント用の **clairctl** CLI ツールをインストールします。

手順

1. 以下のコマンドを入力して、Clair デプロイメント用の **clairctl** プログラムを OpenShift Container Platform クラスターにインストールします。

```
$ oc -n quay-enterprise exec example-registry-clair-app-64dd48f866-6ptgw -- cat
/usr/bin/clairctl > clairctl
```



注記

非公式ですが、**clairctl** ツールをダウンロードできます。

2. **clairctl** ファイルの権限を設定して、ユーザーが実行できるようにします。次に例を示します。

```
$ chmod u+x ./clairctl
```

9.1.2. OpenShift Container Platform での Clair デプロイメントの Clair 設定シークレットの取得とデコード

以下の手順を使用して、OpenShift Container Platform 上の OpenShift Container Platform でプロビジョニングされた Clair インスタンスの設定シークレットを取得してデコードします。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。

手順

1. 次のコマンドを入力して、設定シークレットを取得してデコードし、それを Clair 設定 YAML に保存します。

```
$ oc get secret -n quay-enterprise example-registry-clair-config-secret  -o "jsonpath=
{$.data['config\.yaml']}" | base64 -d > clair-config.yaml
```

2. **disable_updaters** および **airgap** パラメーターが **true** に設定されるように、**clair-config.yaml** ファイルを更新します。次に例を示します。

```
---
indexer:
  airgap: true
---
matcher:
  disable_updaters: true
---
```

9.1.3. 接続された Clair インスタンスからアップデータバンドルをエクスポートする

次の手順を使用して、インターネットにアクセスできる Clair インスタンスから更新プログラムバンドルをエクスポートします。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair 設定シークレットを取得してデコードし、Clair の **config.yaml** ファイルに保存している。
- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。

手順

- インターネットにアクセスできる Clair インスタンスから、設定ファイルで **clairctl** CLI ツールを使用して、アップデーターバンドルをエクスポートします。以下に例を示します。

```
$ ./clairctl --config ./config.yaml export-updaters updates.gz
```

9.1.4. 切断された OpenShift Container Platform クラスターでの Clair データベースへのアクセスの設定

以下の手順を使用して、切断された OpenShift Container Platform クラスター内の Clair データベースへのアクセスを設定します。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair 設定シークレットを取得してデコードし、Clair の **config.yaml** ファイルに保存している。
- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。
- インターネットにアクセスできる Clair インスタンスからアップデーターバンドルをエクスポートしている。

手順

1. CLI ツール **oc** を使用して、Clair データベースサービスを特定します。次に例を示します。

```
$ oc get svc -n quay-enterprise
```

出力例

```
NAME                            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
example-registry-clair-app          ClusterIP     172.30.224.93    <none>
80/TCP,8089/TCP                 4d21h
example-registry-clair-postgres     ClusterIP     172.30.246.88    <none>        5432/TCP
4d21h
...
```

2. Clair データベースポートを転送して、ローカルマシンからアクセスできるようにします。以下に例を示します。

```
$ oc port-forward -n quay-enterprise service/example-registry-clair-postgres 5432:5432
```

3. Clair の **config.yaml** ファイルを更新します。次に例を示します。

```
indexer:
  connstring: host=localhost port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable (1)
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
  scanner:
    repo:
      rhel-repository-scanner: (2)
        repo2cpe_mapping_file: /data/cpe-map.json
    package:
      rhel_containerscanner: (3)
        name2repos_mapping_file: /data/repo-map.json
```

(1) 複数の **connstring** フィールドの **host** の値を **localhost** に置き換えます。

(2) **rhel-repository-scanner** パラメーターの詳細は、「共通製品列挙情報へのリポジトリーのマッピング」を参照してください。

3 **rhel_containersscanner** パラメーターの詳細は、「共通製品列挙情報へのリポジトリーのマッピング」を参照してください。

9.1.5. 切断された OpenShift Container Platform クラスターへのアップデーターバンドルのインポート

以下の手順を使用して、アップデーターバンドルを切断された OpenShift Container Platform クラスターにインポートします。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair 設定シークレットを取得してデコードし、Clair の **config.yaml** ファイルに保存している。
- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。
- インターネットにアクセスできる Clair インスタンスからアップデーターバンドルをエクスポートしている。
- アップデーターバンドルを切断された環境に転送している。

手順

- CLI ツール **clairctl** を使用して、アップデーターバンドルを OpenShift Container Platform によってデプロイされた Clair データベースにインポートします。以下に例を示します。

```
$ ./clairctl --config ./clair-config.yaml import-updaters updates.gz
```

9.2. 切断された OPENSHIFT CONTAINER PLATFORM クラスター用の CLAIR の自己管理デプロイメントのセットアップ

以下の手順を使用して、切断された OpenShift Container Platform クラスター用に Clair の自己管理デプロイメントをセットアップします。

9.2.1. OpenShift Container Platform で自己管理 Clair デプロイメント用の clairctl コマンドラインユーティリティーツールをインストールする

以下の手順を使用して、OpenShift Container Platform に自己管理 Clair デプロイメント用の **clairctl** CLI ツールをインストールします。

手順

1. **podman cp** コマンドを使用して、自己管理の Clair デプロイメント用の **clairctl** プログラムをインストールします。以下に例を示します。

```
$ sudo podman cp clairv4:/usr/bin/clairctl ./clairctl
```

2. **clairctl** ファイルの権限を設定して、ユーザーが実行できるようにします。次に例を示します。

```
$ chmod u+x ./clairctl
```

9.2.2. 切断された OpenShift Container Platform クラスター用の自己管理 Clair コンテナーのデプロイ

以下の手順を使用して、切断された OpenShift Container Platform クラスター用の自己管理 Clair コンテナーをデプロイします。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。

手順

1. Clair 設定ファイル用のフォルダーを作成します。次に例を示します。

```
$ mkdir /etc/clairv4/config/
```

2. **disable_updaters** パラメーターを **true** に設定して Clair 設定ファイルを作成します。次に例を示します。

```
---
indexer:
  airgap: true
---
matcher:
  disable_updaters: true
---
```

3. コンテナーイメージを使用して Clair を起動し、作成したファイルから設定にマウントします。

```
$ sudo podman run -it --rm --name clairv4 \
-p 8081:8081 -p 8088:8088 \
-e CLAIR_CONF=/clair/config.yaml \
-e CLAIR_MODE=combo \
-v /etc/clairv4/config:/clair:Z \
registry.redhat.io/quay/clair-rhel8:v3.8.15
```

9.2.3. 接続された Clair インスタンスからアップデータバンドルをエクスポートする

次の手順を使用して、インターネットにアクセスできる Clair インスタンスから更新プログラムバンドルをエクスポートします。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair をデプロイしている。
- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。

手順

- インターネットにアクセスできる Clair インスタンスから、設定ファイルで **clairctl** CLI ツールを使用して、アップデーターバンドルをエクスポートします。以下に例を示します。

```
$ ./clairctl --config ./config.yaml export-updaters updates.gz
```

9.2.4. 切断された OpenShift Container Platform クラスターでの Clair データベースへのアクセスの設定

以下の手順を使用して、切断された OpenShift Container Platform クラスター内の Clair データベースへのアクセスを設定します。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair をデプロイしている。
- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。
- インターネットにアクセスできる Clair インスタンスからアップデーターバンドルをエクスポートしている。

手順

1. CLI ツール **oc** を使用して、Clair データベースサービスを特定します。次に例を示します。

```
$ oc get svc -n quay-enterprise
```

出力例

```
NAME                              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
example-registry-clair-app        ClusterIP   172.30.224.93    <none>
80/TCP,8089/TCP                   4d21h
example-registry-clair-postgres   ClusterIP   172.30.246.88    <none>        5432/TCP
4d21h
...
```

2. Clair データベースポートを転送して、ローカルマシンからアクセスできるようにします。以下に例を示します。

```
$ oc port-forward -n quay-enterprise service/example-registry-clair-postgres 5432:5432
```

3. Clair の **config.yaml** ファイルを更新します。次に例を示します。

```
indexer:
  connstring: host=localhost port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable (1)
  scanlock_retry: 10
  layer_scan_concurrency: 5
```

```
    migrations: true
    scanner:
      repo:
        rhel-repository-scanner: 2
          repo2cpe_mapping_file: /data/cpe-map.json
      package:
        rhel_containerscanner: 3
          name2repos_mapping_file: /data/repo-map.json
```

1. 複数の **connstring** フィールドの **host** の値を **localhost** に置き換えます。
2. **rhel-repository-scanner** パラメーターの詳細は、「共通製品列挙情報へのリポジトリーのマッピング」を参照してください。
3. **rhel_containersscanner** パラメーターの詳細は、「共通製品列挙情報へのリポジトリーのマッピング」を参照してください。

9.2.5. 切断された OpenShift Container Platform クラスターへのアップデーターバンドルのインポート

以下の手順を使用して、アップデーターバンドルを切断された OpenShift Container Platform クラスターにインポートします。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair をデプロイしている。
- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。
- インターネットにアクセスできる Clair インスタンスからアップデーターバンドルをエクスポートしている。
- アップデーターバンドルを切断された環境に転送している。

手順

- CLI ツール **clairctl** を使用して、アップデーターバンドルを OpenShift Container Platform によってデプロイされた Clair データベースにインポートします。

```
$ ./clairctl --config ./clair-config.yaml import-updaters updates.gz
```

9.3. CLAIR CRDA の有効化

Java スキャンは、Code Ready Dependency Analytics (CRDA) と呼ばれる Red Hat が提供する公開 API サービスに依存しています。CRDA はインターネットアクセスでのみ使用でき、デフォルトでは有効になっていません。

CRDA サービスをカスタム API キーと統合し、CRDA for Java および Python スキャンを有効にするには、次の手順に従います。

前提条件

前提条件

- Red Hat Quay 3.7 以降

手順

1. API キーリクエストフォーム を送信して、Quay 固有の CRDA リモートマッチャーを取得します。
2. **clair-config.yaml** ファイルで CRDA 設定を設定します。

```
matchers:
  config:
    crda:
      url: https://gw.api.openshift.io/api/v2/
      key: <CRDA_API_KEY> 1
      source: <QUAY_SERVER_HOSTNAME> 2
```

1. この API キーリクエストフォーム から Quay 固有の CRDA リモートマッチャーを挿入します。
2. Quay サーバーのホスト名。

9.4. 共通製品列挙情報へのリポジトリーのマッピング

Clair の Red Hat Enterprise Linux (RHEL) スキャナーは、Common Product Enumeration (CPE) ファイルに依存して、RPM パッケージを対応するセキュリティーデータにマッピングし、マッチングする結果を生成します。これらのファイルは製品セキュリティーによって所有され、毎日更新されます。

スキャナーが RPM を適切に処理するには、CPE ファイルが存在するか、ファイルへのアクセスが許可されている必要があります。ファイルが存在しないと、コンテナーイメージにインストールされている RPM パッケージはスキャンされません。

表9.1 Clair CPE マッピングファイル

CPE	JSON マッピングファイルへのリンク
repos2cpe	Red Hat Repository-to-CPE JSON
names2repos	Red Hat Name-to-Repos JSON

切断された Clair インストール用のデータベースに CVE 情報をアップロードするだけでなく、マッピングファイルをローカルで使用できるようにする必要もあります。

- スタンドアロン Red Hat Quay および Clair デプロイメントの場合は、マッピングファイルを Clair Pod に読み込む必要があります。
- OpenShift Container Platform および Clair デプロイメントでの Red Hat Quay Operator デプロイメントの場合は、Clair コンポーネントを **unmanged** に設定する必要があります。次に、Clair を手動でデプロイメントし、マッピングファイルのローカルコピーを読み込むように設定する必要があります。

9.4.1. Common Product Enumeration サンプル設定へのリポジトリーのマッピング

Clair 設定の **repo2cpe_mapping_file** フィールドと **name2repos_mapping_file** フィールドを使用して、CPE JSON マッピングファイルを含めます。以下に例を示します。

```
indexer:
 scanner:
   repo:
     rhel-repository-scanner:
       repo2cpe_mapping_file: /data/cpe-map.json
   package:
     rhel_containerscanner:
       name2repos_mapping_file: /data/repo-map.json
```

詳細は、OVAL セキュリティーデータをインストール済みの RPM と正確にマッチングする方法 を参照してください。

第10章 CLAIR 設定の概要

Clair は、構造化された YAML ファイルによって設定されます。各 Clair ノードは、CLI フラグまたは環境変数を使用して、実行するモードと設定ファイルへのパスを指定する必要があります。以下に例を示します。

```
$ clair -conf ./path/to/config.yaml -mode indexer
```

または

```
$ clair -conf ./path/to/config.yaml -mode matcher
```

前述のコマンドはそれぞれ、同じ設定ファイルを使用して 2 つの Clair ノードを開始します。1 つはインデックス作成機能を実行し、もう 1 つはマッチング機能を実行します。

Go 標準ライブラリーが尊重する環境変数は、必要に応じて指定できます。次に例を示します。

- **HTTP_PROXY**
- **HTTPS_PROXY**
- **SSL_CERT_DIR**

Clair を **combo** モードで実行している場合は、設定でインデクサー、マッチャー、通知設定ブロックを指定する必要があります。

10.1. CLAIR 設定リファレンス

次の YAML は、Clair 設定の例を示しています。

```
http_listen_addr: ""
introspection_addr: ""
log_level: ""
tls: {}
indexer:
    connstring: ""
    scanlock_retry: 0
    layer_scan_concurrency: 0
    migrations: false
    scanner: {}
    airgap: false
matcher:
    connstring: ""
    indexer_addr: ""
    migrations: false
    period: ""
    disable_updaters: false
    update_retention: 2
matchers:
    names: nil
    config: nil
updaters:
    sets: nil
    config: nil
```

```
notifier:
    connstring: ""
    migrations: false
    indexer_addr: ""
    matcher_addr: ""
    poll_interval: ""
    delivery_interval: ""
    disable_summary: false
    webhook: null
    amqp: null
    stomp: null
auth:
  psk: nil
trace:
    name: ""
    probability: null
    jaeger:
        agent:
            endpoint: ""
        collector:
            endpoint: ""
            username: null
            password: null
        service_name: ""
        tags: nil
        buffer_max: 0
metrics:
    name: ""
    prometheus:
        endpoint: null
    dogstatsd:
        url: ""
```

注記

上記の YAML ファイルには、完全を期すためにすべてのキーがリストされています。この設定ファイルをそのまま使用すると、一部のオプションがデフォルトで正常に設定されない場合があります。

10.2. CLAIR の一般的なフィールド

次のセクションでは、Clair デプロイメントで使用できる一般的な設定フィールドを説明します。

フィールド	Typhttp_listen_ae	説明
http_listen_addr	String	HTTP API が公開される場所を設定します。 デフォルト: **:6060**

フィールド	Typhttp_listen_ae	説明
introspection_addr	String	Clair のメトリックと正常性エンドポイントが公開される場所を設定します。
log_level	String	ログレベルを設定します。文字列 **debug-color**、**debug**、**info**、**warn**、**error**、**fatal**、**panic** のいずれかが必要です。
tls	String	TLS/SSL および HTTP/2 の HTTP API を提供するための設定を含むマップ。
.cert	String	使用する TLS 証明書。フルチェーン証明書である必要があります。

10.3. CLAIR インデクサー設定フィールド

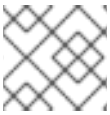
Clair では、次のインデクサー設定フィールドを使用できます。

フィールド	タイプ	説明
indexer	Object	Clair インデクサーノード設定を提供します。
.airgap	Boolean	インデクサーとフェッチャーのインターネットへの HTTP アクセスを無効にします。プライベート IPv4 および IPv6 アドレスが許可されます。データベース接続は影響を受けません。
.connstring	String	Postgres 接続文字列。URL または libpq 接続文字列として形式を受け入れます。

フィールド	タイプ	説明
.index_report_request_concurrency	Integer	レートは、インデックスレポート作成リクエストの数を制限します。これを **0** に設定すると、この値の自動サイズ調整が試みられます。負の値を設定すると、無制限になります。自動サイジングは、使用可能なコア数の倍数です。 同時実行数を超えた場合、API はステータスコード **429** を返します。
.scanlock_retry	Integer	秒を表す正の整数。並行インデクサーは、マニフェストスキャンをロックして、上書きを回避します。この値は、待機中のインデクサーがロックをポーリングする頻度をチューニングします。
.layer_scan_concurrency	Integer	レイヤーの同時スキャン数を制限する正の整数。インデクサーは、マニフェストのレイヤーを同時にマッチングします。この値は、インデクサーが並行してスキャンするレイヤーの数をチューニングします。
.migrations	Boolean	インデクサーノードがデータベースへの移行を処理するかどうか。
.scanner	String	インデクサー設定。 Scanner を使用すると、設定オプションをレイヤースキャナーに渡すことができます。スキャナーは、そのように設計されていると、構築時にこの設定を渡します。
.scanner.dist	String	特定のスキャナーの名前と値として任意の YAML を持つマップ。
.scanner.package	String	特定のスキャナーの名前と値として任意の YAML を持つマップ。
.scanner.repo	String	特定のスキャナーの名前と値として任意の YAML を持つマップ。

10.4. CLAIR マッチャー設定フィールド

Clair では、次のマッチャー設定フィールドを使用できます。



注記

matchers 設定フィールドとは異なります。

フィールド	タイプ	説明
matcher	Object	Clair マッチャーノード設定を提供します。
.cache_age	String	レスポンスをキャッシュするように、ユーザーに通知する期間を制御します。
.connstring	String	Postgres 接続文字列。URL または libpq 接続文字列として形式を受け入れます。
.max_conn_pool	Integer	データベース接続プールのサイズを制限します。 Clair では、カスタムの接続プールサイズを使用できます。この数は、同時に許可されるアクティブなデータベース接続の数を直接設定します。 このパラメーターは、将来のバージョンでは無視されます。ユーザーは、接続文字列を使用して、これを設定する必要があります。
.indexer_addr	String	マッチャーはインデクサーに接続して **VulnerabilityReport** を作成します。このインデクサーの場所は必須です。 デフォルトは **30m** です。
.migrations	Boolean	マッチャーノードがデータベースへの移行を処理するかどうか。
.period	String	新しいセキュリティーアドバイザリーの更新頻度を決定します。 デフォルトは **30m** です。
.disable_updaters	Boolean	バックグラウンド更新を実行するかどうか。

フィールド	タイプ	説明
.update_retention	Integer	ガベージコレクションサイクル間で保持する更新操作の数を設定します。これは、データベースサイズの制約に基づいて安全な MAX 値に設定する必要があります。 デフォルトは **10m** です。 **0** 未満の値を指定すると、ガベージコレクションが無効になります。**2** は、更新を通知と比較できるようにするための最小値です。

10.5. CLAIR マッチャー設定フィールド

Clair では、次のマッチャー設定フィールドを使用できます。

注記

matcher 設定フィールドとは異なります。

フィールド	タイプ	説明
matchers	文字列の配列。	in-tree **matchers** および **remotematchers** の設定を提供します。
.names	String	有効なマッチャーについてマッチャーファクトリーに通知する文字列値のリスト。値が **null** に設定されている場合、マッチャーのデフォルトのリストは、alpine、aws、debian、oracle、photon、python、python、rhel、suse、ubuntu、crda を実行します。

フィールド	タイプ	説明

.config	String	特定のマッチャーに設定を提供します。
		
		マッチャーファクトリーコンストラクターに提供されるサブオブジェクトを含むマッチャーの名前をキーとするマップ。以下に例を示します。
		
		`config:`
		`  python:`
		`  ignore_vulns:`
		`  - CVE-XYZ`
		`  - CVE-ABC`

10.6. CLAIR アップデーター設定フィールド

Clair では、次のアップデーター設定フィールドを使用できます。

フィールド	タイプ	説明
updaters	Object	マッチャーの更新マネージャーの設定を提供します。
.sets	String	どのアップデーターを実行するかを更新マネージャーに通知する値のリスト。 値が **null** に設定されている場合、アップデーターのデフォルトのセット は、**alpine**、**aws**、**debian**、**oracle**、**photon**、**pyupio**、**rhel**、**suse**、**ubuntu** を実行します。 空白のままにすると、更新プログラムは実行されません。

フィールド	タイプ	説明

.config	String	特定のアップデーターセットに設定を提供します。
		
		アップデーターセットのコンストラクターに提供されるサブオブジェクトを含むアップデーターセットの名前をキーとするマップ。以下に例を示します。
		
		`config:`
		`  ubuntu:`
		`  security_tracker_url: http://security.url`
		`  ignore_distributions:`
		`  - cosmic`

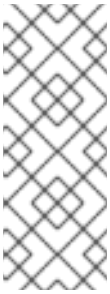
10.7. CLAIR ノーティファイアー設定フィールド

Clair では、次のノーティファイアー設定フィールドを使用できます。

フィールド	タイプ	説明
notifier	Object	Clair ノーティファイアーノード設定を提供します。
.connstring	String	Postgres 接続文字列。形式を URL または libpq 接続文字列として受け入れます。
.migrations	Boolean	ノーティファイアーノードがデータベースへの移行を処理するかどうか。
.indexer_addr	String	ノーティファイアーはインデクサーに連絡して、脆弱性の影響を受けるマニフェストを作成または取得します。このインデクサーの場所は必須です。
.matcher_addr	String	ノーティファイアーはマッチャーに連絡して、更新操作をリストし、差分を取得します。このマッチャーの場所は必須です。

フィールド	タイプ	説明
.poll_interval	String	ノーティファイアーがマッチャーに更新操作をクエリーする頻度。
.delivery_interval	String	ノーティファイアーが、作成された通知または以前に失敗した通知の配信を試行する頻度。
.disable_summary	Boolean	通知をマニフェストごとに 1 つに要約するかどうかを制御します。
.webhook	Object	Webhook 配信のノーティファイアーを設定します。
.webhook.target	String	Webhook が配信される URL。
.webhook.callback	String	通知を取得できるコールバック URL。この URL に通知 ID が追加されます。 これは通常、Clair ノーティファイアーがホスティングされている場所です。
.webhook.headers	String	ヘッダー名を値のリストに関連付けるマップ。
.amqp	Object	AMQP 配信のノーティファイアーを設定します。  注記 Clair は、独自に AMQP コンポーネントを宣言しません。交換またはキューを使用しようとする試みはすべて受動的であり、失敗します。ブローカー管理者は、事前に交換とキューをセットアップする必要があります。
.amqp.direct	Boolean	**true** の場合、ノーティファイアーは設定された AMQP ブローカーに個別の通知 (コールバックではない) を配信します。

フィールド	タイプ	説明
.amqp.rollup	Integer	**amqp.direct** が **true** に設定されている場合、この値は直接配信で送信する通知の数をノーティファイアーに通知します。たとえば、**direct** が **true** に設定され、**amqp.rollup** が **5** に設定されている場合、ノーティファイアーは単一の JSON ペイロードで 5 つ以下の通知をブローカーに配信します。値を **0** に設定すると、実質的に **1** に設定されます。
.amqp.exchange	Object	接続先の AMQP 交換。
.amqp.exchange.name	String	接続先の交換の名前。
.amqp.exchange.type	String	交換のタイプ。通常は、**direct**、**fanout**、**topic**、**headers** のいずれかです。
.amqp.exchange.durability	Boolean	設定されたキューが永続的かどうか。
.amqp.exchange.auto_delete	Boolean	設定されたキューが **auto_delete_policy** を使用するかどうか。
.amqp.routing_key	String	各通知が送信されるルーティングキーの名前。
.amqp.callback	String	**amqp.direct** が **false** に設定されている場合、この URL はブローカーに送信される通知コールバックで提供されます。この URL は、Clair の通知 API エンドポイントを指している必要があります。
.amqp.uris	String	接続先の 1 つ以上の AMQP ブローカーのリスト (優先順位順)。
.amqp.tls	Object	AMQP ブローカーへの TLS/SSL 接続を設定します。
.amqp.tls.root_ca	String	ルート CA を読み取ることができるファイルシステムパス。

フィールド	タイプ	説明
.amqp.tls.cert	String	TLS/SSL 証明書を読み取ることができるファイルシステムパス。  注記 Go **crypto/x509** パッケージで文書化されているように、Clair は **SSL_CERT_DIR** も許可します。
.amqp.tls.key	String	TLS/SSL 秘密鍵を読み取ることができるファイルシステムパス。
.stomp	Object	STOMP 配信のノーティファイアーを設定します。
.stomp.direct	Boolean	**true** の場合、ノーティファイアーは個別の通知 (コールバックではない) を設定済みの STOMP ブローカーに配信します。
.stomp.rollup	Integer	**stomp.direct** が **true** に設定されている場合、この値は、1回の直接配信で送信される通知の数を制限します。たとえば、**direct** が **true** に設定され、**rollup** が **5** に設定されている場合、ノーティファイアーは単一の JSON ペイロードで 5 つ以下の通知をブローカーに配信します。値を **0** に設定すると、実質的に **1** に設定されます。
.stomp.callback	String	**stomp.callback** が **false** に設定されている場合は、通知コールバックで指定された URL がブローカーに送信されます。この URL は、Clair の通知 API エンドポイントを指している必要があります。
.stomp.destination	String	通知を配信する STOMP の宛先。
.stomp.uris	String	接続先の 1 つ以上の STOMP ブローカーのリスト (優先順位順)。

フィールド	タイプ	説明
.stomp.tls	Object	STOMP ブローカーへの TLS/SSL 接続を設定しました。
.stomp.tls.root_ca	String	ルート CA を読み取ることができるファイルシステムパス。  注記 Go **crypto/x509** パッケージで文書化されているように、Clair は **SSL_CERT_DIR** も尊重します。
.stomp.tls.cert	String	TLS/SSL 証明書を読み取ることができるファイルシステムパス。
.stomp.tls.key	String	TLS/SSL 秘密鍵を読み取ることができるファイルシステムパス。
.stomp.user	String	STOMP ブローカーのログインの詳細を設定します。
.stomp.user.login	String	接続に使用する STOMP ログイン。
.stomp.user.passcode	String	接続に使用する STOMP パスコード。

10.8. CLAIR 承認設定フィールド

Clair では、次の承認設定フィールドを使用できます。

フィールド	タイプ	説明
auth	Object	Clair の外部およびサービス内 JWT ベースの認証を定義します。複数の **auth** 機能が定義されている場合、Clair は 1 つを選択します。現在、複数の機能はサポートされていません。
.psk	String	事前共有キー認証を定義します。

フィールド	タイプ	説明
.psk.key	String	JWT の署名と検証を行うすべての当事者間で配布される、base64 でエンコードされた共有キー。
.psk.iss	String	確認する JWT 発行者のリスト。空のリストは、JWT クレームで任意の発行者を受け入れます。

10.9. CLAIR トレース設定フィールド

Clair では、次のトレース設定フィールドを使用できます。

フィールド	タイプ	説明
trace	Object	OpenTelemetry に基づいて分散トレース設定を定義します。
.name	String	トレースが属するアプリケーションの名前。
.probability	Integer	トレースが発生する確率。
.jaeger	Object	Jaeger トレースの値を定義します。
.jaeger.agent	Object	Jaeger エージェントへの配信を設定するための値を定義します。
.jaeger.agent.endpoint	String	トレースを送信できる **<host>:<post>** 構文のアドレス。
.jaeger.collector	Object	Jaeger コレクターへの配信を設定するための値を定義します。
.jaeger.collector.endpoint	String	トレースを送信できる **<host>:<post>** 構文のアドレス。
.jaeger.collector.username	String	Jaeger ユーザー名。
.jaeger.collector.password	String	Jaeger パスワード。
.jaeger.service_name	String	Jaeger に登録されているサービス名。

フィールド	タイプ	説明
.jaeger.tags	String	追加のメタデータを提供するキーと値のペア。
.jaeger.buffer_max	Integer	保管および分析のために Jaeger バックエンドに送信される前にメモリーにバッファーできるスパンの最大数。

10.10. CLAIR メトリック設定フィールド

Clair では、次のメトリック設定フィールドを使用できます。

フィールド	タイプ	説明
metrics	Object	OpenTelemetry に基づいて分散トレース設定を定義します。
.name	String	使用中のメトリックの名前。
.prometheus	String	Prometheus メトリックエクスポーターの設定。
.prometheus.endpoint	String	メトリックが提供されるパスを定義します。