機能紹介資料

Contrast Assess
セッションメタデータ機能

Contrast Security Japan 合同会社

本資料について

- この資料は、Contrast Assessの「セッションメタデータ」機能の利用目的や設定方法、ユースケースについて概要を説明します。

Contrast SECURITY

機能概要

セッションメタデータ機能の概要

- **セッションメタデータ** - Assessエージェントによる検査時に、その検査セッション（アプリの起動～エージェントによる検査～アプリの終了の一連の流れ）に関して、下記のような付加情報を追加することで、セッションごとの検査結果を区別できるようにする仕組み
  - ビルド番号
  - ブランチ名
  - アプリのバージョン

  … etc.

- 脆弱性やルートカバレッジを閲覧する際に、メタデータの値を元にしたフィルタリングなどが可能

- 主な目的
  - 脆弱性が修正されたかどうかの確認を容易にする
  - 複数の開発ブランチが並存しているような環境で、どのブランチのビルドがどの脆弱性を持っているか、区別できるようにする

セッションメタデータの設定

Contrastエージェントの動作パラメータ `session_metadata` に、キー/値ペア形式で渡します。

指定できるメタデータの一覧

Name	Value
Commit Hash	`commitHash`
Committer	`committer`
Branch Name	`branchName`
Git Tag	`gitTag`
Repository	`repository`
Test Run	`testRun`
Version	`version`
Build Number	`buildNumber`

設定例

Javaシステムプロパティとして指定

```
-Dcontrast.application.session_metadata="branchName=feature/some-new-thing,
committer=Jane,repository=Contrast-Java"
```

Contrastエージェントのyaml設定ファイルで指定

```
application:
  session_metadata: branchName=feature/some-new-thing,committer=Jane,
repository=Contrast-Ruby
```

CIシステム (Jenkinsなど) の環境変数をビルドスクリプト内で渡す

```
-Dcontrast.application.session_metadata="branchName=$GIT_BRANCH,committer=$
GIT_COMMITTER_NAME,commitHash=$GIT_COMMIT_HASH,repository=$GIT_URL,buildNum
ber=$BUILD_NUMBER"
```

製品ドキュメント (https://docs.contrastsecurity.com/en/configure-session-metadata.html) より

Contrast SECURITY

脆弱性一覧画面でのセッションメタデータの表示

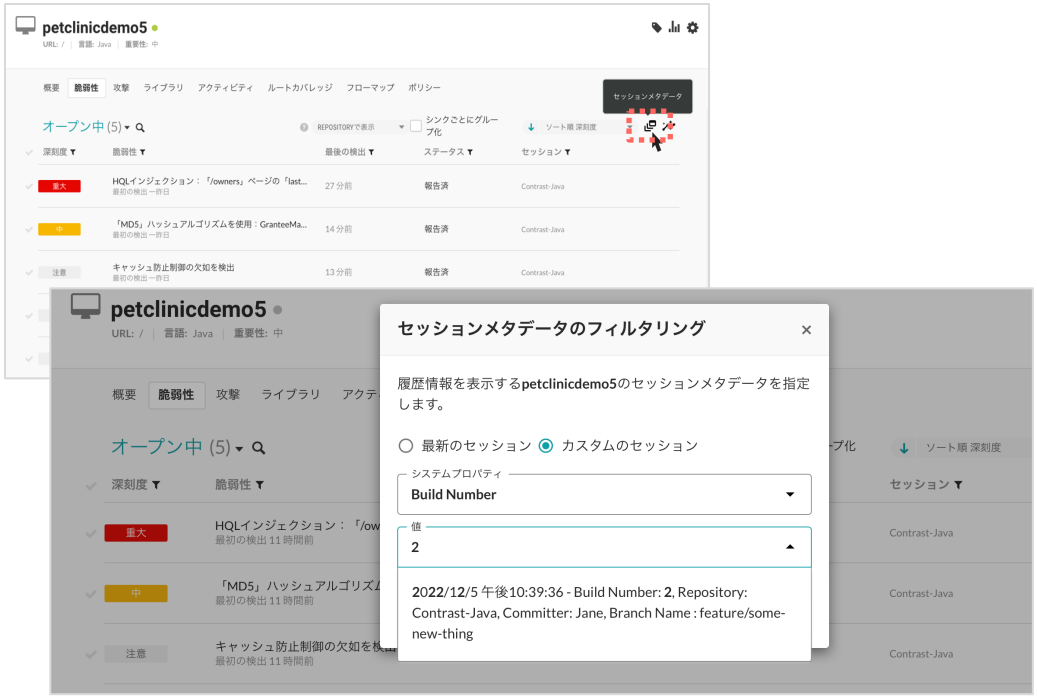
1. 上部のドロップダウンから、表示するメタデータを選択します。

2. 「セッション」列に、選択したメタデータの値が表示されます。

Contrast SECURITY

セッションメタデータを使った脆弱性のフィルタリング

1. 右上の四角が重なったアイコンをクリックして、
   フィルタに使うプロパティ名と値を指定します。



2. 該当するセッションからの検出結果のみが表示されます。

「脆弱性」画面のほか、「ルートカバレッジ」画面でも同じ要領でセッションを特定して確認ができます。

Contrast SECURITY

特定検査セッションに遡った脆弱性詳細情報（脆弱性インスタンス）の確認

脆弱性詳細情報を特定セッション時点に遡って確認したい場合、脆弱性ページの「備考」タブから「脆弱性インスタンス」を指定します。

1.

2.

Contrast SECURITY

利用シナリオ例

利用シナリオ例 – 脆弱性修正の追跡 (1/3)

① 脆弱性のあるコードをテスト

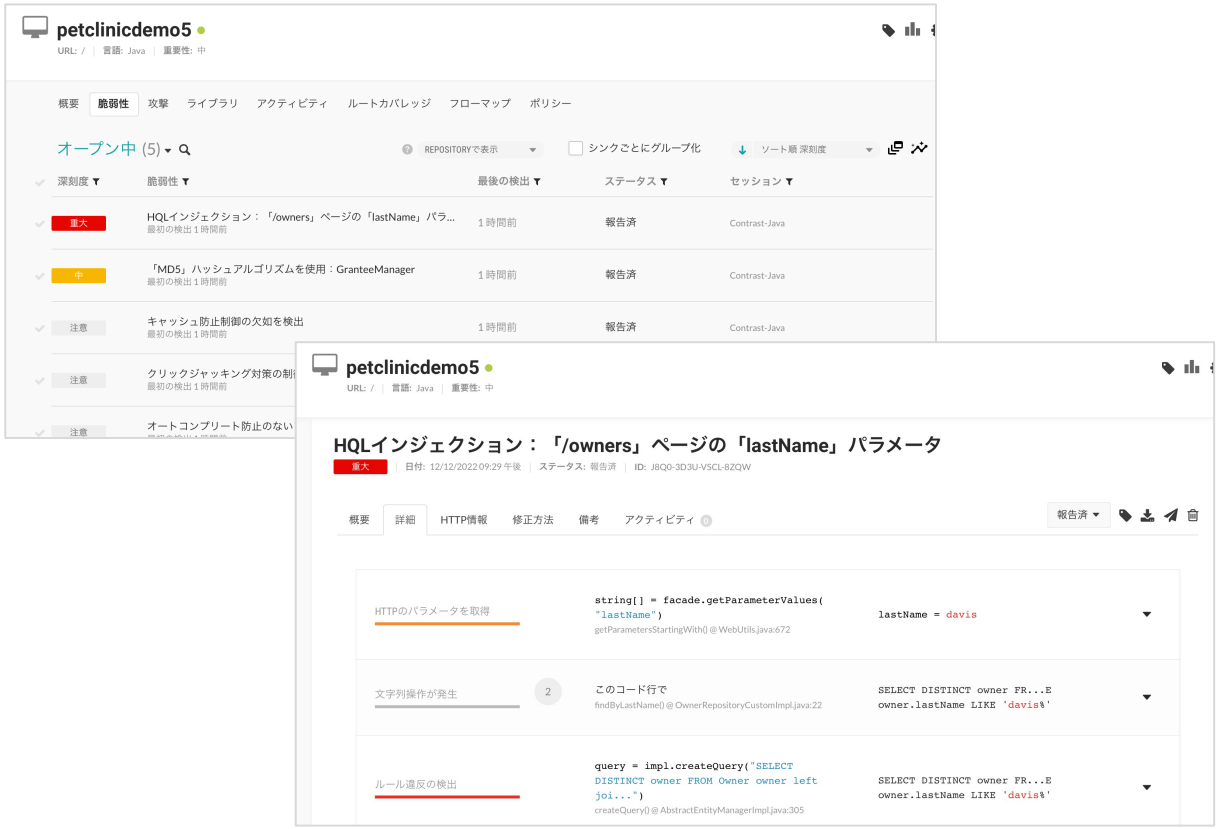
1. 脆弱性のあるコード

```
19      public Collection<Owner> findByLastName(String lastName) {
20              System.out.println("Vulnerable method 1");
21              // 脆弱なクエリ構築 -- ここから
22              String sqlQuery = "SELECT DISTINCT owner FROM Owner owner left
                    join fetch owner.pets WHERE owner.lastName LIKE '" + lastName +
                     "%'";
23              TypedQuery<Owner> query = this.entityManager.createQuery(sqlQuery,
                    Owner.class);
24              // 脆弱なクエリ構築 -- ここまで
25              return query.getResultList();
26          }
```

2. エージェントオプションで、セッションメタデータ Build Number = 1 を指定して起動・検査

```
java -javaagent:contrast.jar  \
    -Dcontrast.config.path=contrast_security.yaml \
    -Dcontrast.application.session_metadata="branchName=feature/
        some-new-thing,committer=Jane,repository=Contrast-Java,buildNumber=1" \
    -jar ./target/spring-petclinic-1.5.1.jar
```

3. 検出結果の表示



Contrast SECURITY

利用シナリオ例 – 脆弱性修正の追跡 (2/3)

② 脆弱なコードを修正し、再テスト

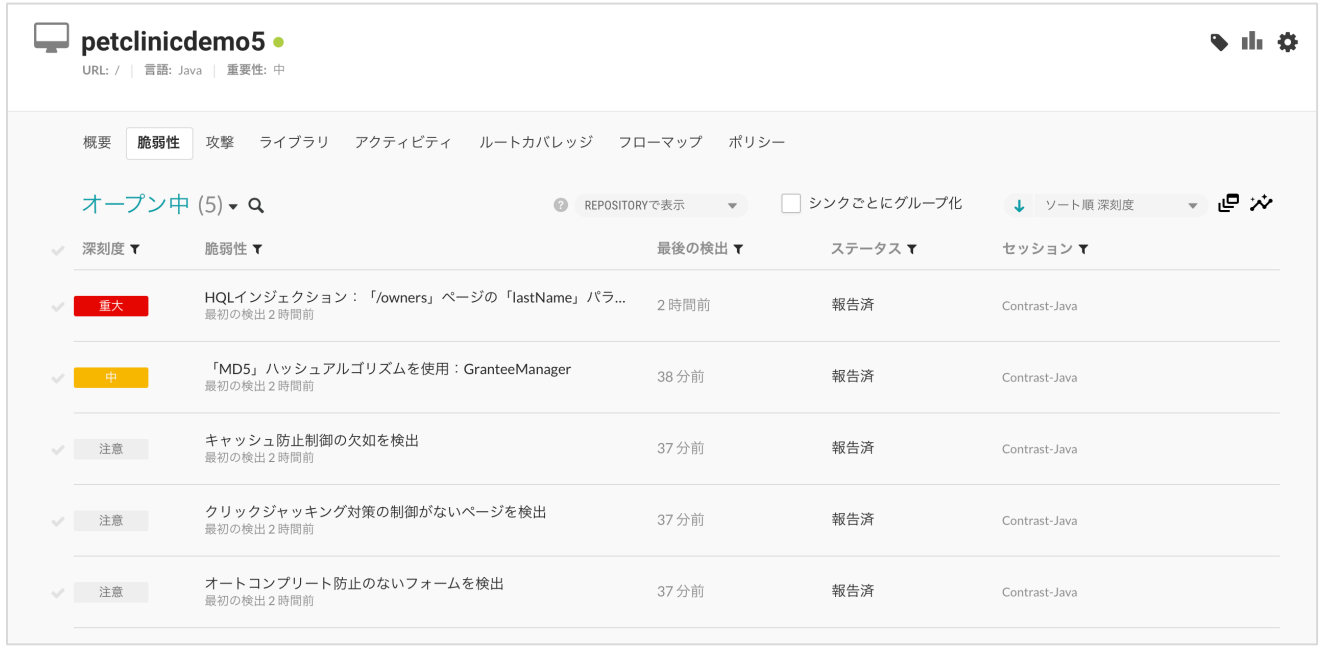
1. 修正されたコード

```
19      public Collection<Owner> findByLastName(String lastName) {
20              System.out.println("Vulnerable method 1");
21
22              // 脆弱なクエリ構築 -- ここから
23              //String sqlQuery = "SELECT DISTINCT owner FROM Owner owner left
                join fetch owner.pets WHERE owner.lastName LIKE '" + lastName +
                "%'";
24              //TypedQuery<Owner> query = this.entityManager.createQuery(
                sqlQuery, Owner.class);
25              // 脆弱なクエリ構築 -- ここまで
26
27              // 修正済み -- ここから
28              String sqlQuery = "SELECT DISTINCT owner FROM Owner owner left
                    join fetch owner.pets WHERE owner.lastName LIKE :lastName";
29              TypedQuery<Owner> query = this.entityManager.createQuery(sqlQuery,
                    Owner.class);
30              query.setParameter("lastName", lastName + "%");
31              // 修正済み -- ここまで
32
33              return query.getResultList();
34          }
```

2. エージェントオプションで、セッションメタデータ Build Number = 2を指定して起動・検査

```
java -javaagent:contrast.jar  \
    -Dcontrast.config.path=contrast_security.yaml \
    -Dcontrast.application.session_metadata="branchName=feature/
        some-new-thing,committer=Jane,repository=Contrast-Java,buildNumber=2" \
    -jar ./target/spring-petclinic-1.5.1.jar
```

2. Contrast Serverで脆弱性一覧画面にナビゲート。(一見変化がない。。)



Contrast SECURITY

利用シナリオ例 – 脆弱性修正の追跡 (3/3)

③ セッションメタデータを使って脆弱性が修正されたことを確認

1. 「セッション」列にBuild Numberを表示。HQLインジェクション脆弱性は修正後の「2」には該当しないことが確認できる。

2. フィルタ(Build Number = 2)を適用。HQLインジェクション脆弱性は表示されなくなっている。

Contrast SECURITY

補足

補足 - セッションメタデータ機能を使わない場合

セッションメタデータを使って特定の脆弱性インスタンスに遡る機能は、修正確認が簡単ではないケース、特に、同じ実行ルートに複数の脆弱性があるようなケースで有用です。

ここでは、そのような状況でセッションメタデータを使わないとどのようになるかを取り上げます。

1. 同じメソッドに脆弱なHQL呼び出しが２件ある状態でテスト（セッションメタデータ指定なし）

```
public Collection<Owner> findByLastName(String lastName) {
        System.out.println("Vulnerable method 1 - Two vulnerable queries");

        String lastName1 = lastName;
        String lastName2 = lastName;

        // クエリ1(脆弱) -- start
        String sqlQuery1 = "SELECT DISTINCT owner FROM Owner owner left join fetch owner.pets
            WHERE owner.lastName LIKE '" + lastName1 + "%'";
        TypedQuery<Owner> query1 = this.entityManager.createQuery(sqlQuery1, Owner.class);
        // クエリ1(脆弱) -- end

        // クエリ2(脆弱) -- start
        String sqlQuery2 = "SELECT DISTINCT owner FROM Owner owner left join fetch owner.pets
            WHERE owner.lastName LIKE '" + lastName2 + "%'";
        TypedQuery<Owner> query2 = this.entityManager.createQuery(sqlQuery2, Owner.class);
        // クエリ2(脆弱) -- end

        Collection result1 = query1.getResultList();
        Collection result2 = query2.getResultList();

        return result1;
    }
```

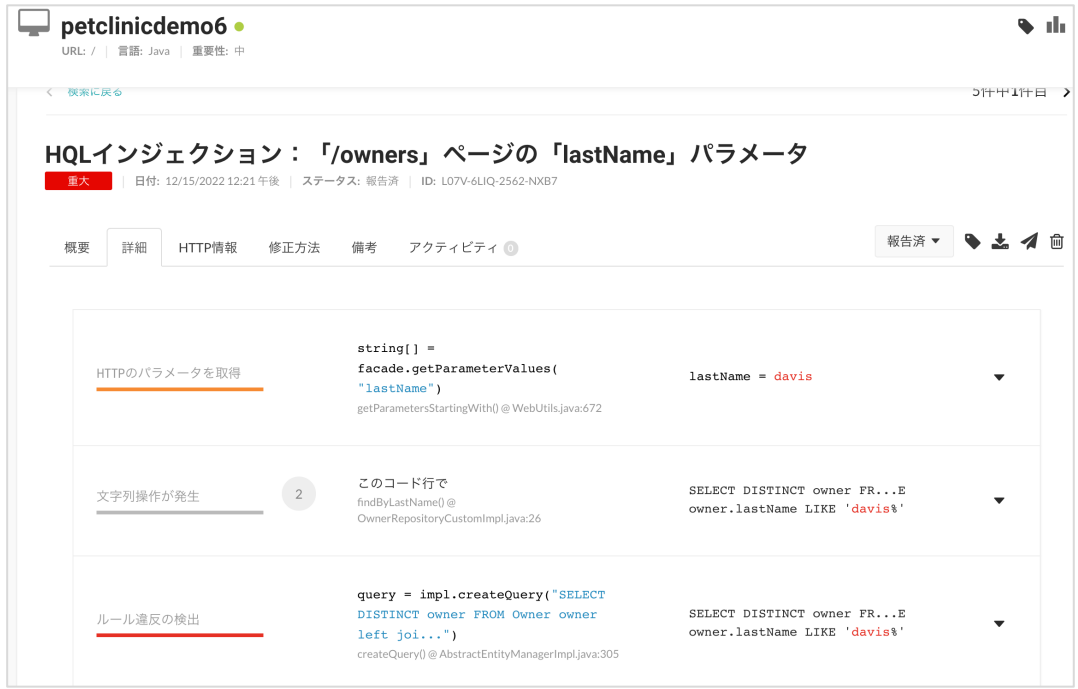
2. ２つの脆弱性のうち１つのみ修正してテスト（セッションメタデータ指定なし）

```
public Collection<Owner> findByLastName(String lastName) {
        System.out.println("Vulnerable method 1 - Two vulnerable queries");

        String lastName1 = lastName;
        String lastName2 = lastName;

        // クエリ1(安全) -- start
        String sqlQuery1 = "SELECT DISTINCT owner FROM Owner owner left join fetch owner.pets
            WHERE owner.lastName LIKE :lastName";
        TypedQuery<Owner> query1 = this.entityManager.createQuery(sqlQuery1, Owner.class);
        query1.setParameter("lastName", lastName1 + "%");
        // クエリ1(安全) -- end

        // クエリ2(脆弱) -- start
        String sqlQuery2 = "SELECT DISTINCT owner FROM Owner owner left join fetch owner.pets
            WHERE owner.lastName LIKE '" + lastName2 + "%'";
        TypedQuery<Owner> query2 = this.entityManager.createQuery(sqlQuery2, Owner.class);
        // クエリ2(脆弱) -- end

        Collection result1 = query1.getResultList();
        Collection result2 = query2.getResultList();

        return result1;
    }
```


Contrast SECURITY

補足 - セッションメタデータ機能を使わない場合

1. 脆弱性の詳細は変化せず (コード行の表示は変わらず)



2. 脆弱性インスタンスも１つのみ
（時点を遡った詳細比較ができない）

このように、セッションメタデータを使わないと、施した修正がうまくいっていないのか、それとも今回の修正自体はうまくいったものの他にも脆弱性がある状況なのか、切り分けがしづらくなります。

さまざまな状況で円滑に脆弱性対応ができるよう、セッションメタデータ機能を活用することが推奨されます。

Contrast SECURITY