CONTRAST SECURITY

SQL/HQLインジェクションの解説と対策

この動画で説明すること

SQLインジェクションの説明と脆弱性の修正

1. SQL/HQLインジェクションが起こる原因
2. Contrast Securityで検知
3. 検査パターンでHQLインジェクションを確認
4. HQLインジェクションを修正
5. Contrast Securityで修正を確認
6. 検査パターンでHQLインジェクションの修正を確認

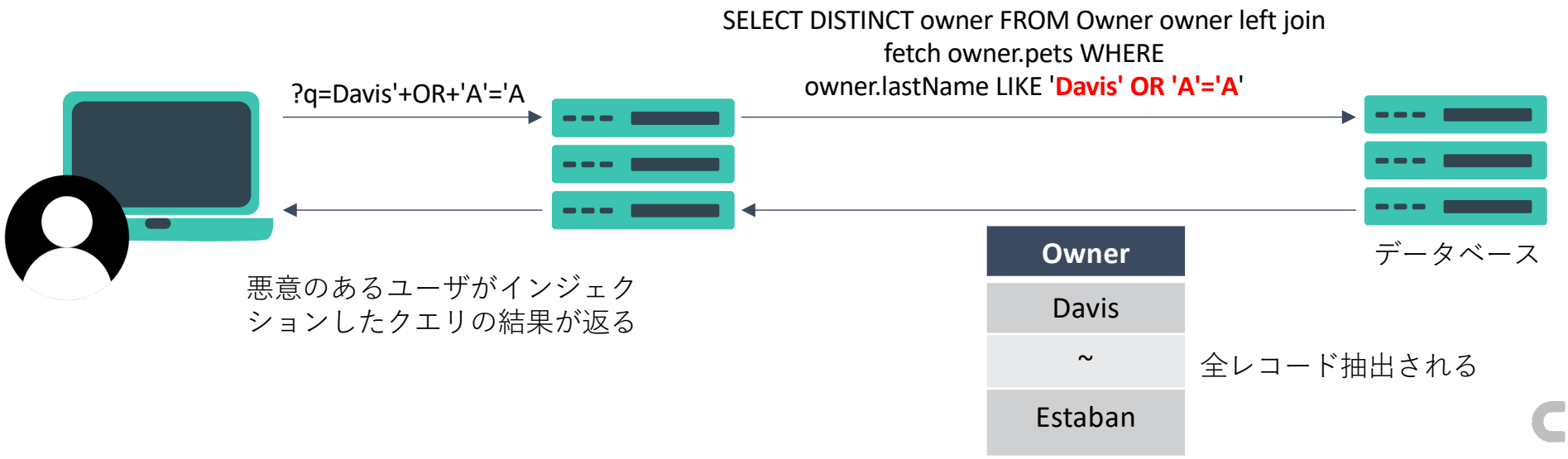
3,6については任意です


SQL/HQLインジェクションの原因

SQL/HQLインジェクションの原因 (1/2)

SQLインジェクションの概要

SQL/HQLインジェクションは外部からの入力値を使用してSQL文を構築していることが原因で発生します。



SQL/HQLインジェクションの原因 (2/2)

原因は文字列結合を行い動的にSQL文を構築しているため

以下のコードはSQLインジェクションの脆弱性があります。

```
String user = request.getParameter( "user" );
String query = "SELECT user_id FROM user_data WHERE user_name = '" + user + "'";
Statement statement = connection.createStatement( );
ResultSet results = statement.executeQuery( query );
```

queryはuserの値に応じて動的にSQLを作り出す仕組みになっています。
下記はuserの値と生成される実行するqueryの値です。

変数userの値	変数queryの値
Davis	SELECT user_id FROM user_data WHERE user_name = '**Davis**';
Davis' OR 'A'='A	SELECT user_id FROM user_data WHERE user_name = '**Davis' OR 'A' = 'A**';

上記のようにシングルクォート(')がuserに含まれるとWHERE句の文字列リテラルが強制的に閉じられ、後続に任意のSQLを挿入できることがわかります。


Contrast Assessでの検知

HQLインジェクションの検知(1/2)

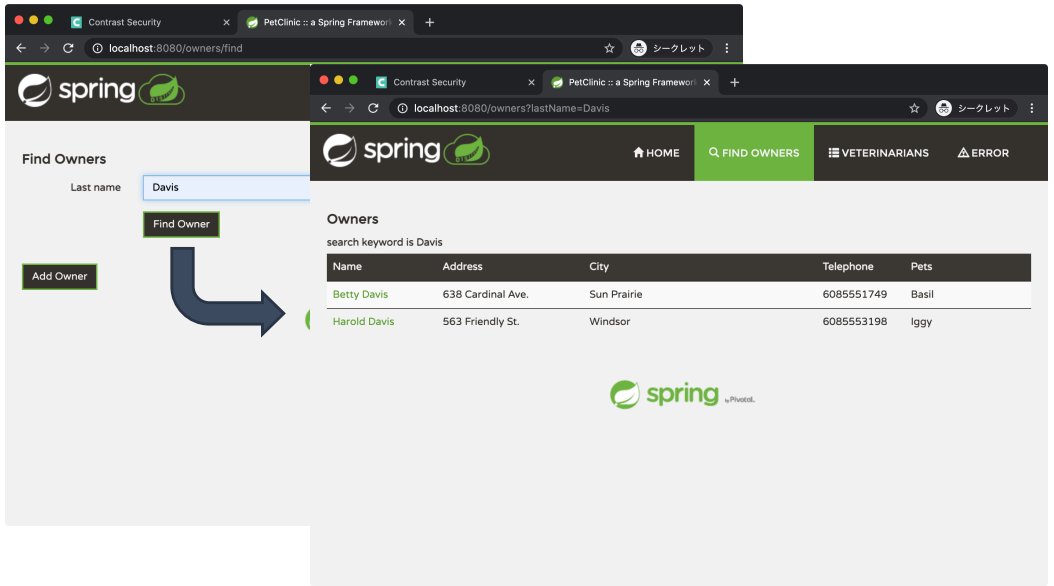
実際にContrast Assessを用いて検知

脆弱性のあるアプリケーションにContrastエージェントを組み込んで、アプリケーションを起動します。

その後、アプリケーションを操作し脆弱性を検知します。

検索機能をテストします

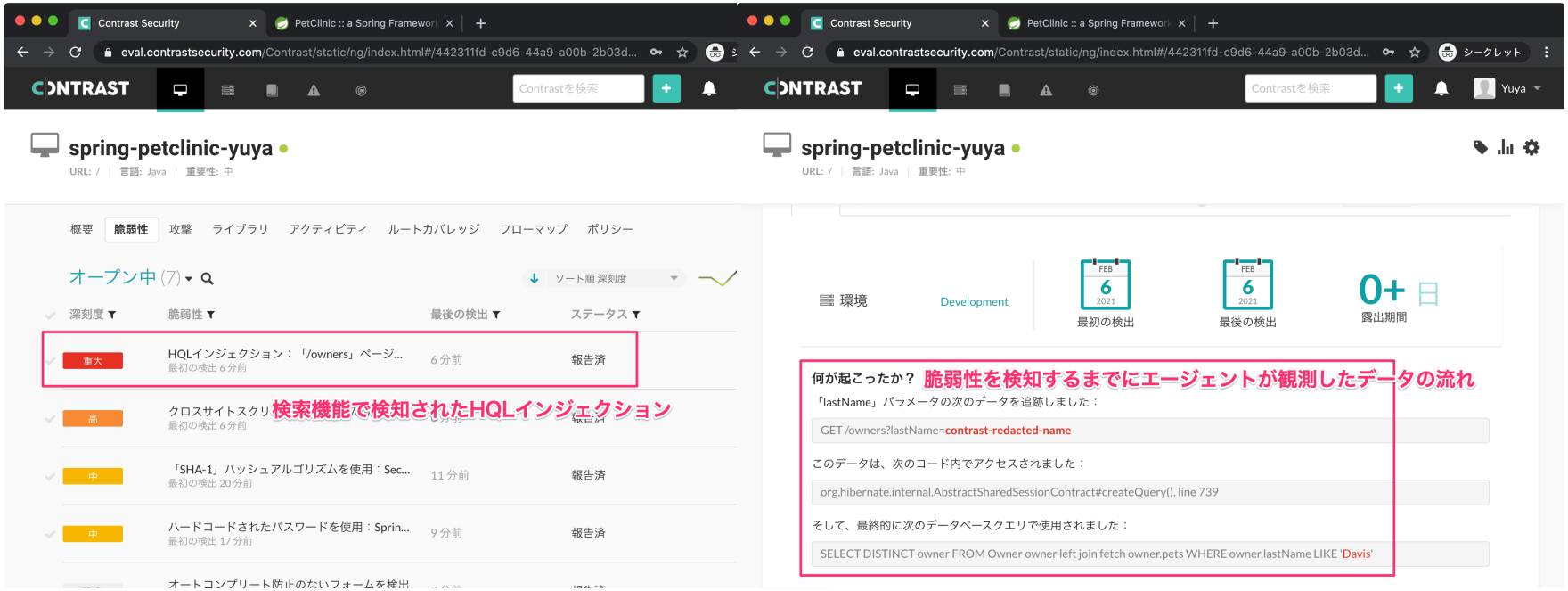
DASTなどで使用される攻撃文字列を使用する必要はありません。



HQLインジェクションの検知(2/2)

Contrast Assessで検知した結果

Contrast UIにアクセスし検知した脆弱性を確認します。




アプリケーションでの確認

検査パターンで確認(1/3)

実際にExploitableか確認

検査パターンベースでアプリケーションにSQL/HQLインジェクションが存在するかは下記の観点で確認します。

観点	入力値
SQL構文が崩れるパターンと崩れないパターンを入力して結果の差分を確認	1. 構文が崩れる Davis' 2. 構文が崩れない Da'||'vis
SQLの機能が有効かどうかを確認	1. sleep関数を使用する '+(select*from(select(sleep(5)))c)+'

検査パターンで確認(2/3)

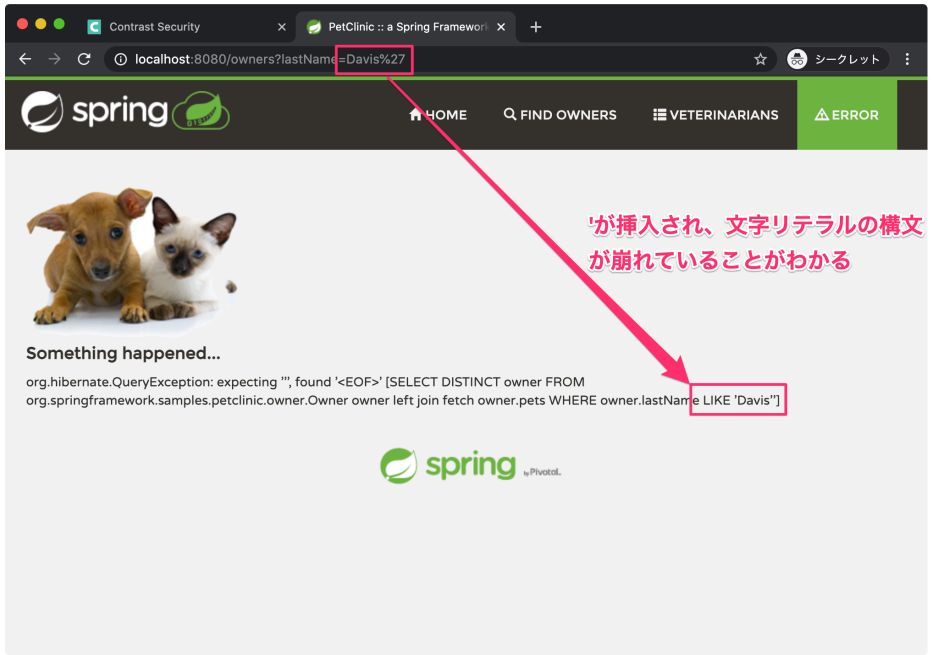
文字リテラルが崩れるパターンで確認

Contrast AssessがHQLインジェクションを検知した機能を前ページの観点で確認してみます。

Davis'を入力した場合、構築されるSELECT文のWHERE句は下記のとおりです。

HERE owner.lastName LIKE '**Davis**''

「'」が一つ余り、文字リテラルとして正しい構成ではないことによるエラーが発生していることがわかります。



検査パターンで確認(3/3)

文字リテラルが崩れないパターンで確認

引き続き、Contrast AssessがHQLインジェクションを検知した機能を前々ページの観点で確認してみます。

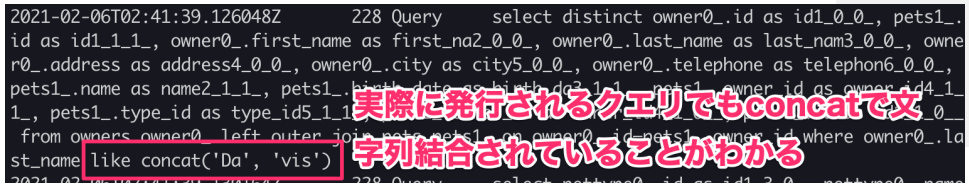
[Da'||'vis」を入力した場合、構築されるSELECT文のWHERE句は下記のとおりです。

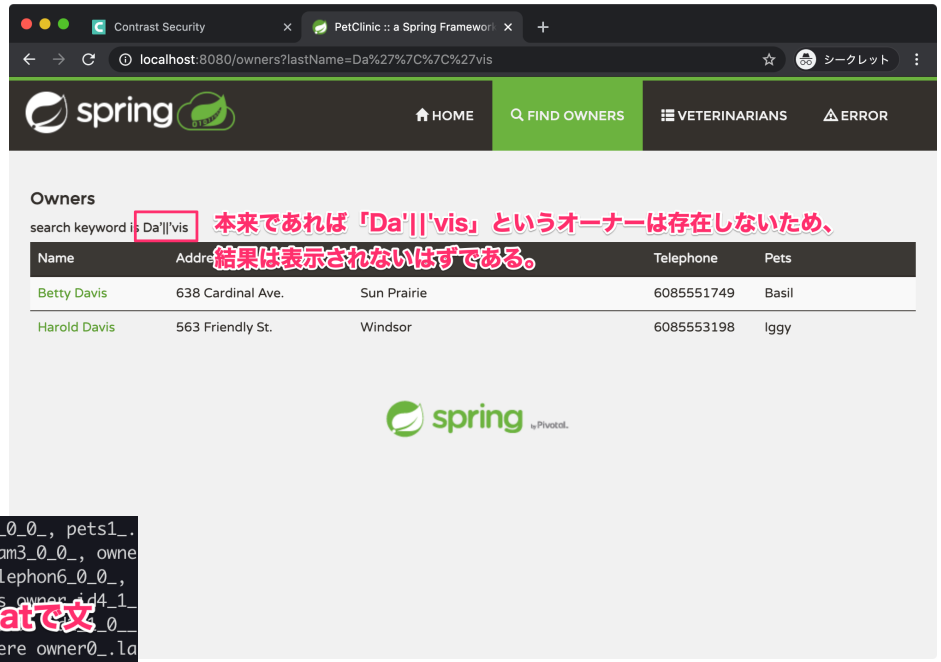
WHERE owner.lastName LIKE '**Da'||'vis**'

「Da'||'vis」が文字列結合により「Davis」と評価され、下記のWHERE句と等価になります。

WHERE owner.lastName LIKE 'Davis'

以下は実行されたクエリのログです。






HQLインジェクションを修正

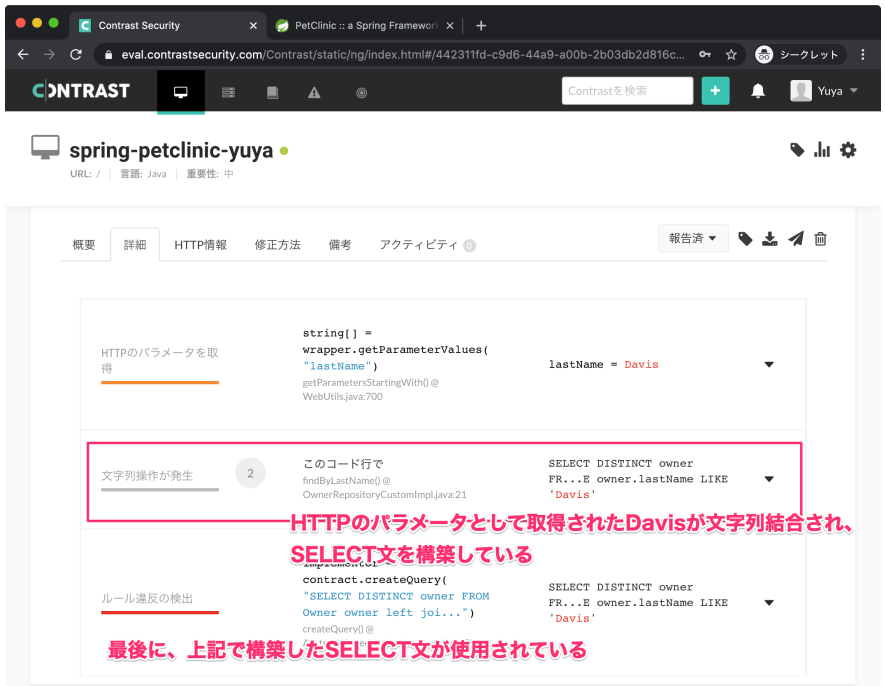
HQLインジェクションを修正(1/4)

Contrast UIを確認し修正箇所を特定

脆弱性の詳細タブを確認すると、Contrast Assessのエージェントが観測したイベントを確認できます。

右の画面の赤枠内で示している箇所で、ユーザの入力値を使用して動的にSQL文を構築していることがわかります。

このコード行で
findByLastName() @
OwnerRepositoryCustomImpl.java:21



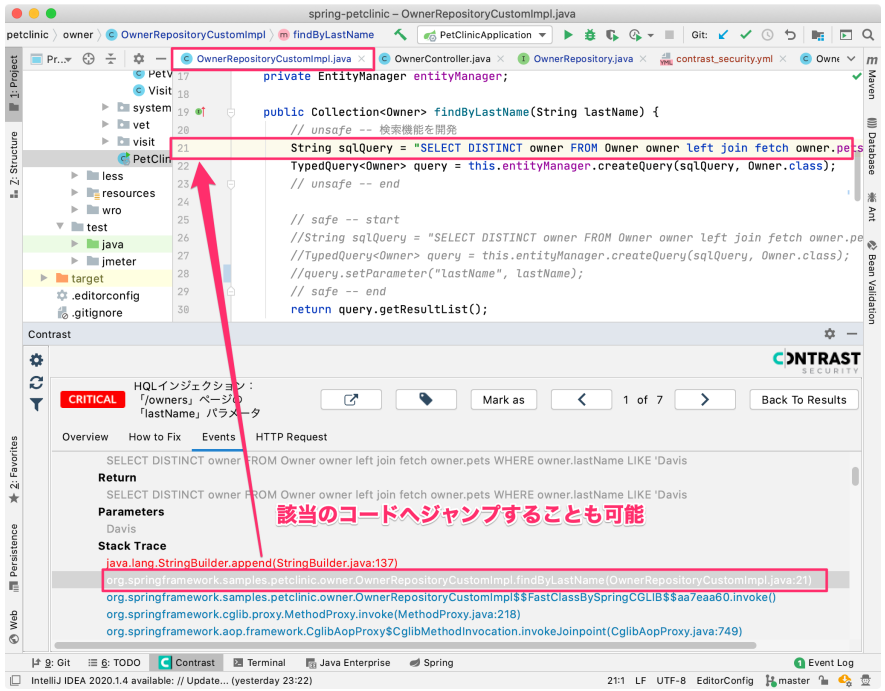
HQLインジェクションを修正(2/4)

IDE連携で修正箇所を特定

IDEからContrastのプラグインを使用することで、Contrast UIと同様に脆弱性の情報を確認できます。

Contrast プラグインでは、Eventsタブから Contrast Assessのエージェントが観測した情報を確認できます。

また、カスタムコードをダブルクリックすることで、該当のコードへジャンプも可能です。



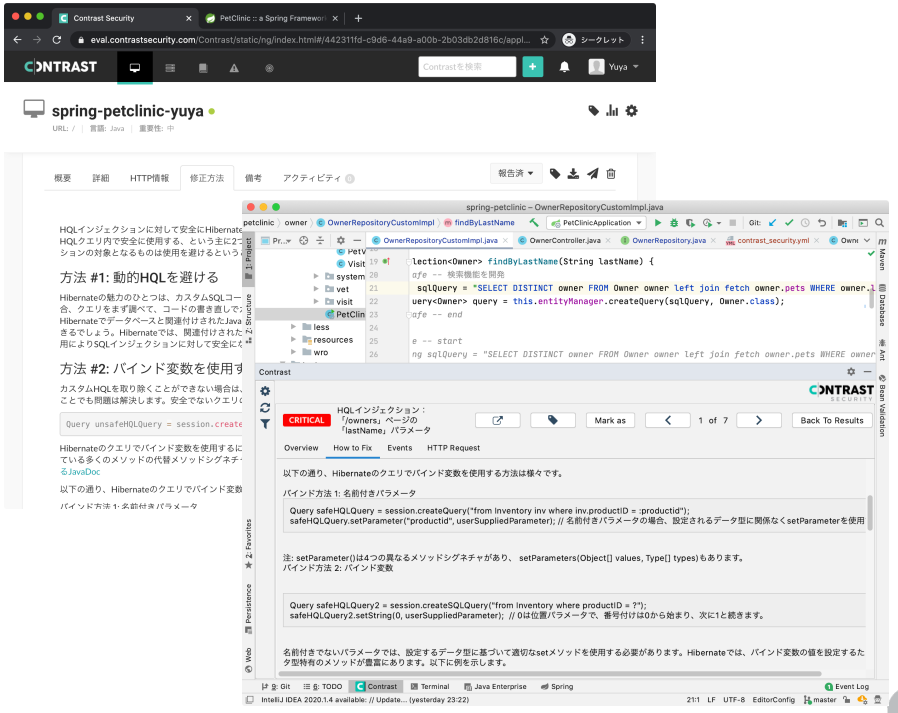
HQLインジェクションを修正(3/4)

修正方法を確認

修正箇所の特定ができたため、続けて修正方法を確認します。

プラグインの「How To Fix」タブもしくは、Contrast UIの「修正方法」タブに修正方法のサンプルが記載されています。

今回は、方法 #2: バインド変数を使用するのバインド方法1: 名前付きパラメータを使用して修正します。



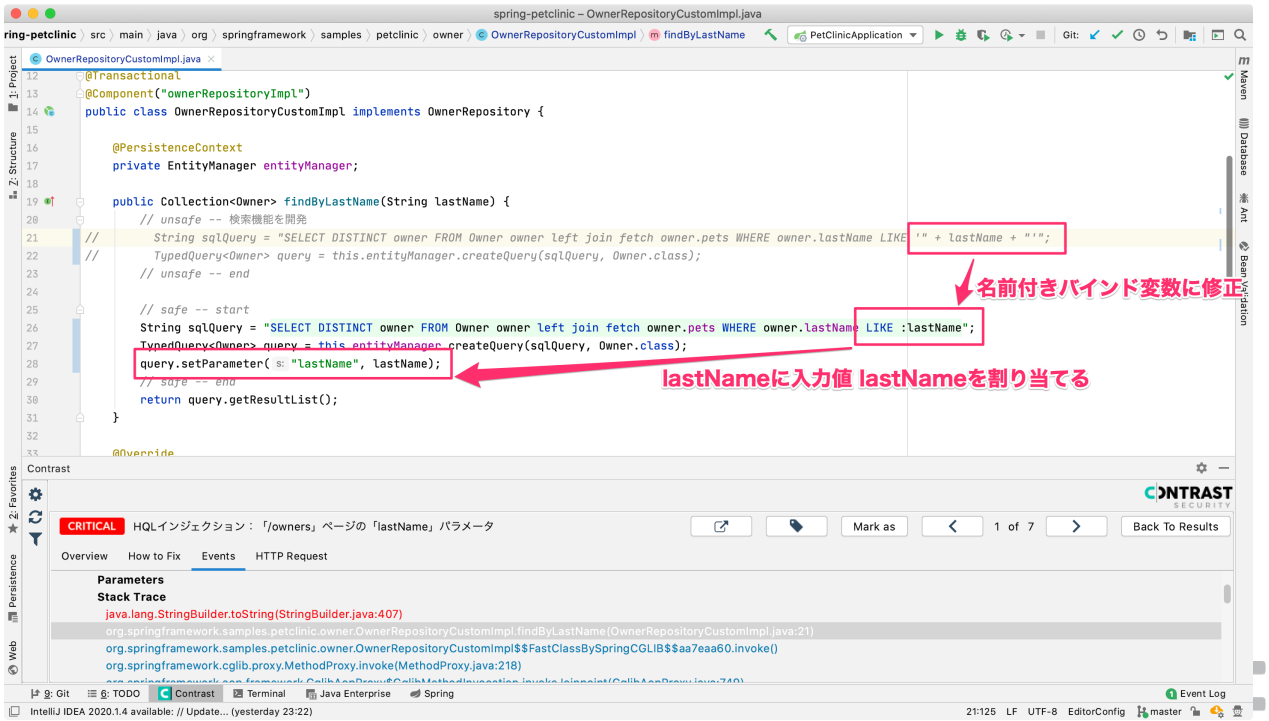
HQLインジェクションを修正(4/4)

コードを修正

確認した修正方法でHQLインジェクションを修正していきます。

21行目で文字列結合して構築していたSELECT文を名前付きバインドに修正します。

その後、28行目でfindByLastNameメソッドの実引数 lastNameをバインド変数のlastNameに割り当てる




Contrast Securityで修正を確認

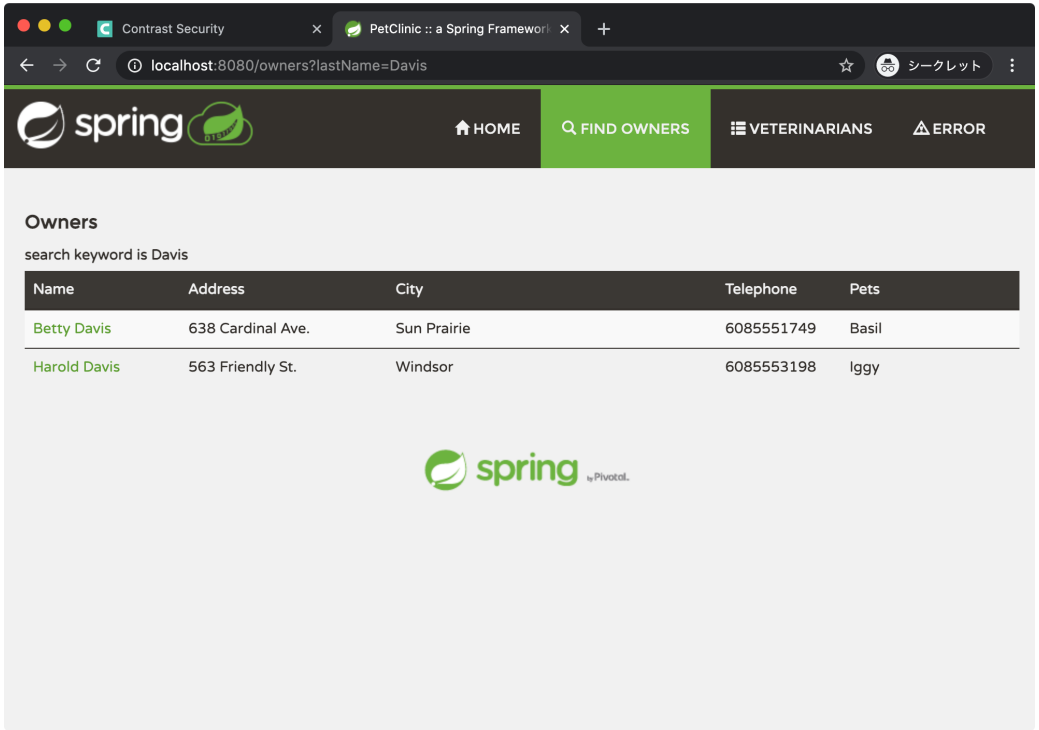
Contrastで修正を確認(1/2)

脆弱性が修正されているかを確認

脆弱性を修正後に再ビルドを行い、エージェントを組み込みアプリケーションを起動します。

脆弱性検出時と同じキーワード「Daivs」で検索を行います。

リクエストを再現する場合、HTTP情報タブにある「リクエストを再生」機能からも実施可能です。



Contrastで修正を確認(2/2)

脆弱性が修正されているかを確認

脆弱性を修正後に再ビルドを行い、エージェントを組み込みアプリケーションを起動します。

HQLインジェクションの「最後の検出」が更新されていないことがわかります。

脆弱性が修正されたかどうかを確認する方法はいくつかあります。

- session_metadataを使用する
- アプリケーションのバージョンを指定する

などの方法があります。


検査パターンで修正を確認

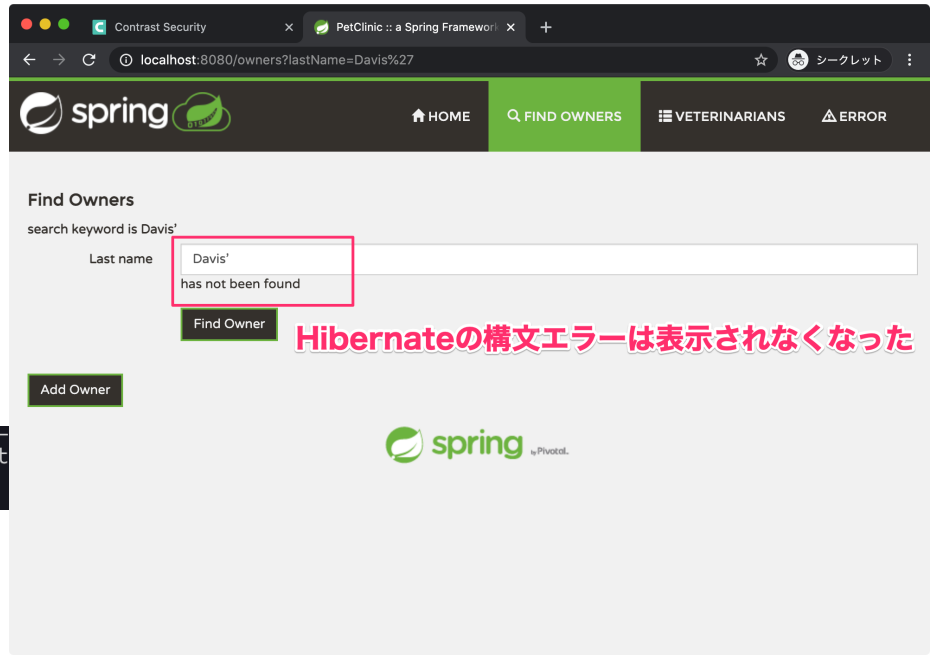
検査パターンで修正を確認(1/1)

文字リテラルが崩れるパターンで確認

検索機能に存在していたHQLインジェクションが修正されているかを検査パターンベースで確認します。

検出時と同様に「Davis'」を入力し検索します。

修正前に表示されていた構文エラーは表示されなくなりました。



以下は実行されたクエリのログです。

「'」がエスケープされ、文字リテラルが崩せないよう修正されていることがわかります。


まとめ

まとめ

SQL/HQLインジェクションの原因と対策

1. SQL/HQLインジェクションが起こる原因

   ユーザの入力を使用してSQL/HQLを構築しているため脆弱性が埋め込まれます。

2. 脆弱性の検出

   Contrast Assessを使用することで、UIを操作するだけで脆弱性を検出できます。

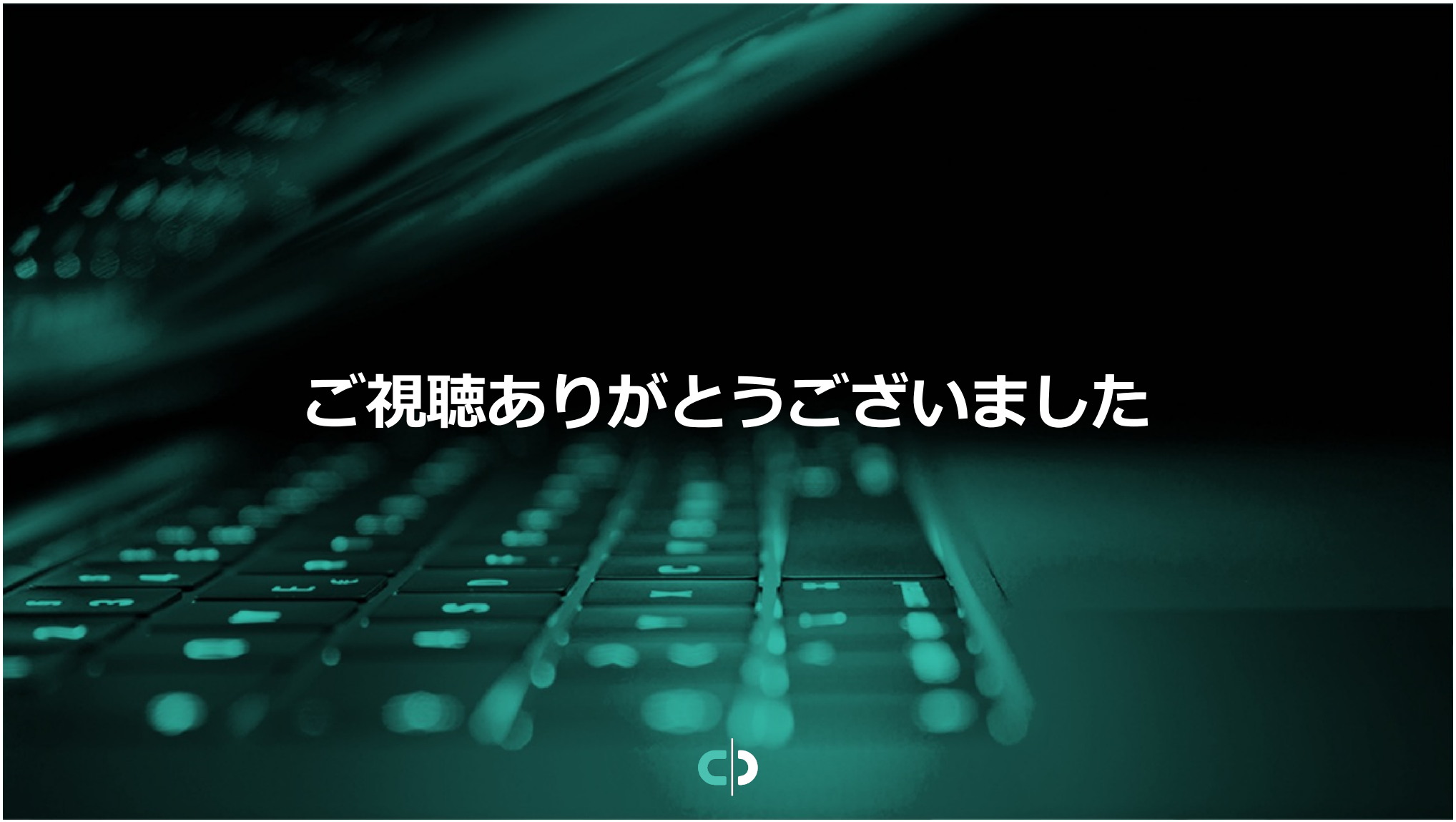
3. 脆弱性の修正

   Contrastが観測した情報、修正方法のサンプルコードをもとに脆弱性を修正し、検出時と同じ操作を実施することで脆弱性の対策が実施されていることを確認できます。

この動画で使用した素材

アプリケーションは下記のソースコードに脆弱性を追加したものを使用しております。本来のコードに脆弱性は含まれておりません。

https://github.com/spring-projects/spring-petclinic


ご視聴ありがとうございました