



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



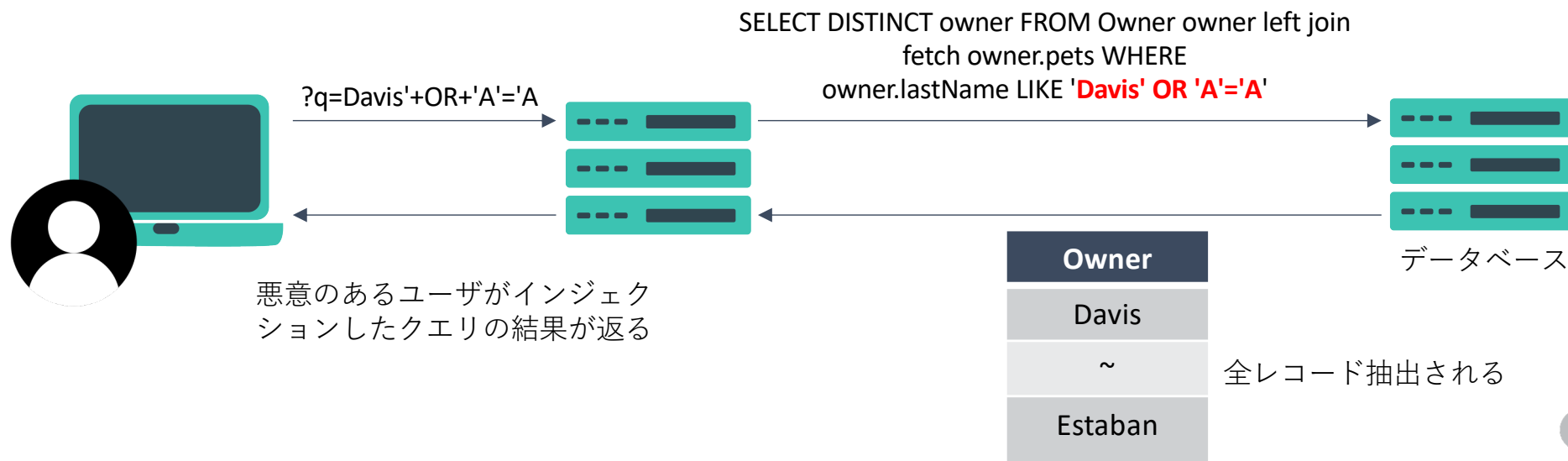
SQL/HQLインジェクションの原因



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)

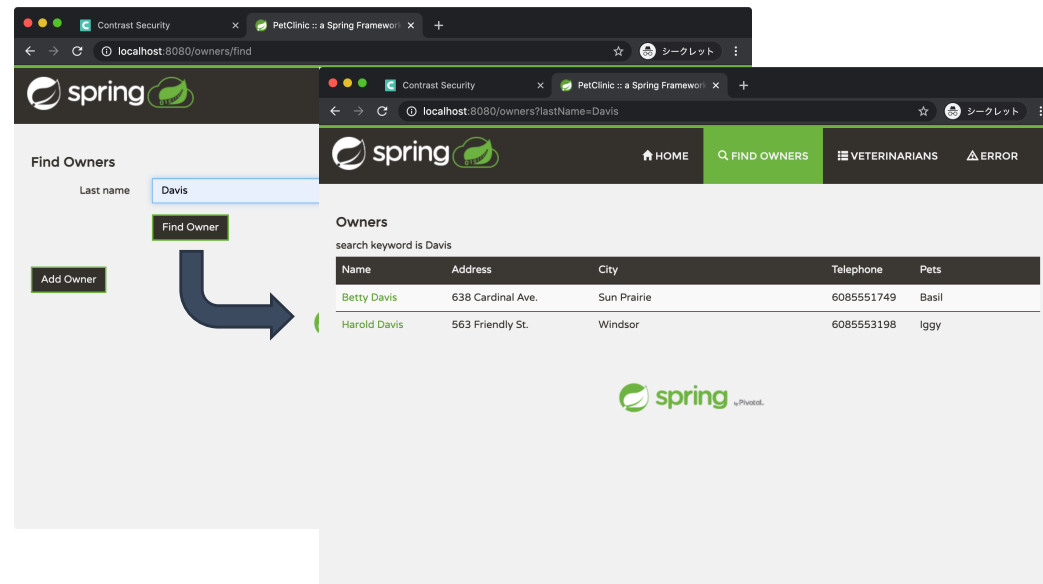
実際にContrast Assessを用いて検知

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。

The screenshot displays the Contrast Security web interface. On the left, a table lists detected vulnerabilities. The top entry is highlighted with a red box and labeled '検索機能で検知されたHQLインジェクション' (HQL injection detected by search function). The table columns include severity, description, last scan time, and status.

深刻度	脆弱性	最後の検出	ステータス
重大	HQLインジェクション: 「owners」ページ...	6分前	報告済
高	クロスサイトスクリプティング	最初の検出 6分前	
中	「SHA-1」ハッシュアルゴリズムを使用: Sec...	11分前 最初の検出 20分前	報告済
中	ハードコードされたパスワードを使用: Sprin...	9分前 最初の検出 17分前	報告済

On the right, the details for the HQL injection vulnerability are shown. A red box highlights the following information:

- 何が起ったか? **脆弱性を検知するまでにエージェントが観測したデータの流れ**
- 「lastName」パラメータの次のデータを追跡しました:
- GET /owners?lastName=**contrast-redacted-name**
- このデータは、次のコード内でアクセスされました:
- org.hibernate.internal.AbstractSharedSessionContract#createQuery(), line 739
- そして、最終的に次のデータベースクエリで使用されました:
- SELECT DISTINCT owner FROM Owner owner left join fetch owner.pets WHERE owner.lastName LIKE 'Davis'



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	<ol style="list-style-type: none">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	<ol style="list-style-type: none">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)

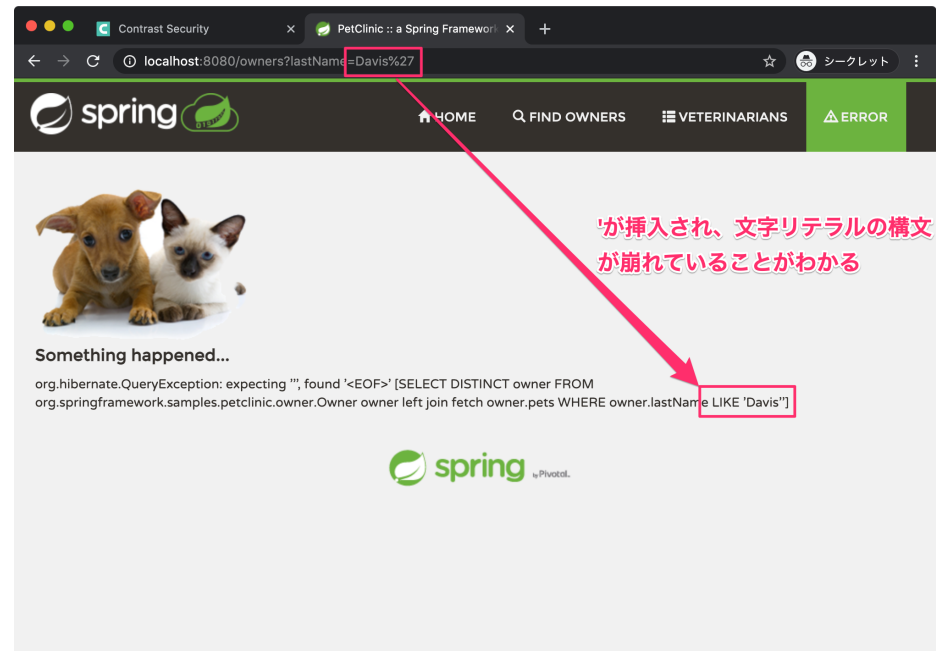
文字リテラルが崩れるパターンで確認

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'
```

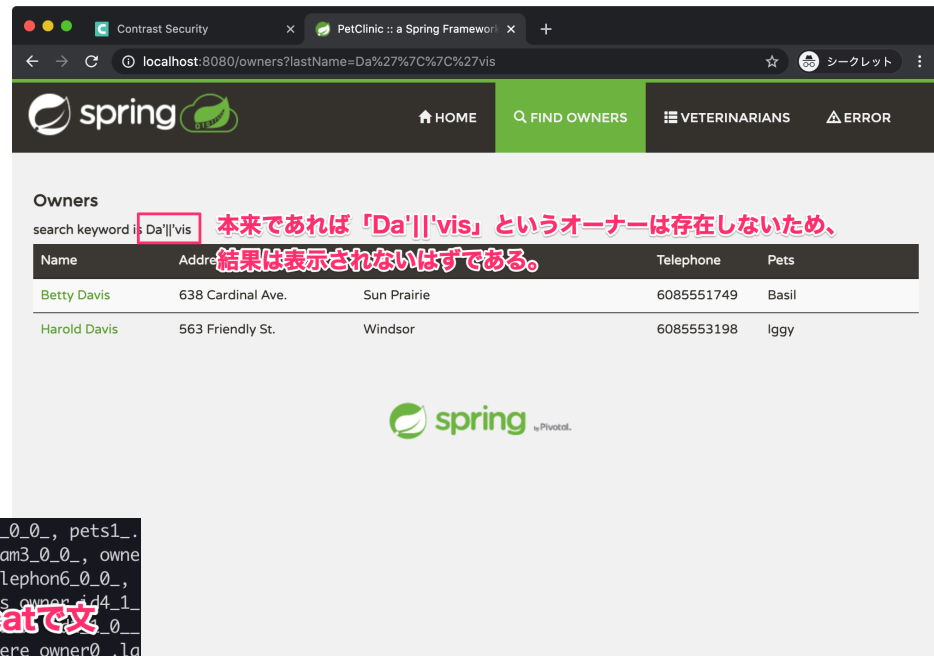
「Da'||'vis」が文字列結合により「Davis」と評価され、下記のWHERE句と等価になります。

```
WHERE owner.lastName LIKE 'Davis'
```

以下は実行されたクエリのログです。

```
2021-02-06T02:41:39.126048Z 228 Query select distinct owner0_.id as id1_0_0_, pets1_.id as id1_1_1_, owner0_.first_name as first_na2_0_0_, owner0_.last_name as last_nam3_0_0_, owner0_.address as address4_0_0_, owner0_.city as city5_0_0_, owner0_.telephone as telephon6_0_0_, pets1_.name as name2_1_1_, pets1_.birth_date as birth_da7_1_1_, pets1_.owner_id as owner_id4_1_1_, pets1_.type_id as type_id5_1_1_ from owners owner0_ left outer join pets pets1_ on owner0_.id=pets1_.owner_id where owner0_.last_name like concat('Da', 'vis')
```

実際に発行されるクエリでもconcatで文字列結合されていることがわかる



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
```

The screenshot shows the Contrast Security interface for a vulnerability report. The report is titled 'spring-petclinic-yuya' and is categorized as 'URL: / | 言語: Java | 重要性: 中'. The report details the following flow:

- HTTPのパラメータを取得**: A parameter `lastName = Davis` is extracted from the request.
- 文字列操作が発生**: The value `Davis` is used in a dynamic SQL query construction. The SQL query is: `SELECT DISTINCT owner FROM FR...E owner.lastName LIKE 'Davis'`.
- ルール違反の検出**: The system detects a rule violation based on the constructed query.

Red annotations highlight the dynamic SQL construction and the use of the user input 'Davis' in the query.

HTTPのパラメータとして取得されたDavisが文字列結合され、SELECT文を構築している

最後に、上記で構築したSELECT文が使用されている



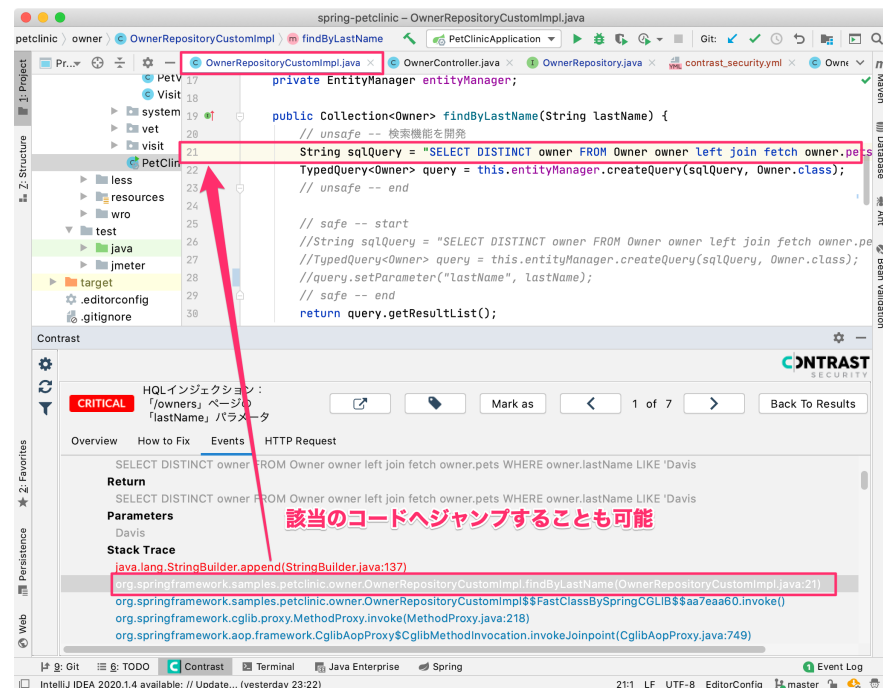
HQLインジェクションを修正(2/4)

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、該当のコードヘジャンプも可能です。



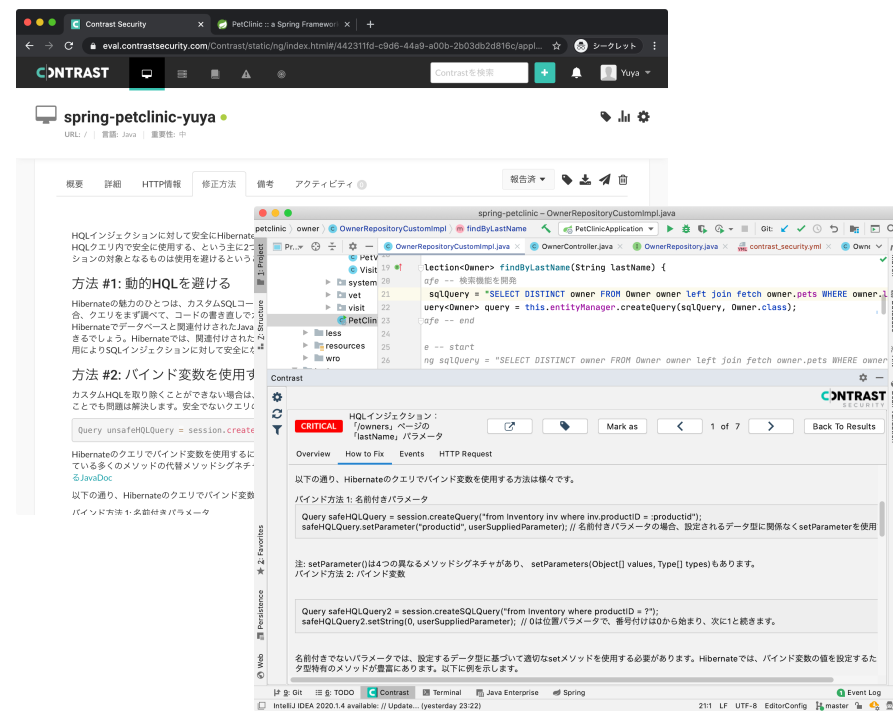
HQLインジェクションを修正(3/4)

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

```
12 @Transactional
13 @Component("ownerRepositoryImpl")
14 public class OwnerRepositoryCustomImpl implements OwnerRepository {
15
16     @PersistenceContext
17     private EntityManager entityManager;
18
19     public Collection<Owner> findByLastName(String lastName) {
20         // unsafe -- 検索機能を閉鎖
21         String sqlQuery = "SELECT DISTINCT owner FROM Owner owner left join fetch owner.pets WHERE owner.lastName LIKE ' + lastName + ''";
22         TypedQuery<Owner> query = this.entityManager.createQuery(sqlQuery, Owner.class);
23         // unsafe -- end
24
25         // safe -- start
26         String sqlQuery = "SELECT DISTINCT owner FROM Owner owner left join fetch owner.pets WHERE owner.lastName LIKE :lastName;";
27         TypedQuery<Owner> query = this.entityManager.createQuery(sqlQuery, Owner.class);
28         query.setParameter("lastName", lastName);
29         // safe -- end
30         return query.getResultList();
31     }
32
33     @Override
34     // ...
35 }
```

名前付きバインド変数に修正

lastNameに入力値 lastNameを割り当てる

CRITICAL HQLインジェクション: 「owners」ページの「lastName」パラメータ

Parameters
Stack Trace
java.lang.StringBuilder.toString(StringBuilder.java:407)
org.springframework.samples.petclinic.owner.OwnerRepositoryCustomImpl.findByLastName(OwnerRepositoryCustomImpl.java:21)
org.springframework.samples.petclinic.owner.OwnerRepositoryCustomImpl\$\$FastClassBySpringCGLIB\$\$5aa7aaa60.invoke()
org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218)

Contrast Securityで修正を確認



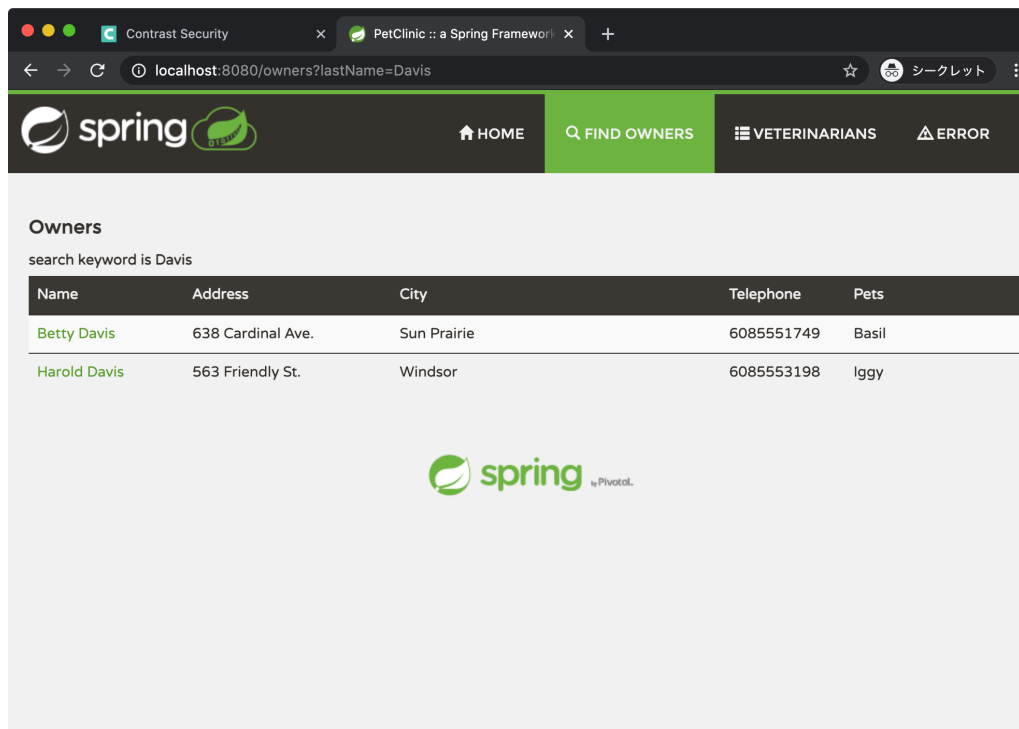
Contrastで修正を確認(1/2)

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「Davis」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。



The screenshot shows a web browser window with the URL `localhost:8080/owners?lastName=Davis`. The page displays a search result for 'Owners' with the search keyword 'Davis'. The results are shown in a table with the following data:

Name	Address	City	Telephone	Pets
Betty Davis	638 Cardinal Ave.	Sun Prairie	6085551749	Basil
Harold Davis	563 Friendly St.	Windsor	6085553198	Iggy

The page also features a navigation bar with 'HOME', 'FIND OWNERS', 'VETERINARIANS', and 'ERROR' links, and a footer with the 'spring by Pivotal' logo.



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。

The screenshot shows the Contrast Security dashboard for 'spring-petclinic-yuya'. The '脆弱性' (Vulnerabilities) tab is active, displaying a table of open vulnerabilities. The first row is highlighted with a red box:

深刻度	脆弱性	最後の検出	ステータス
重大	HQLインジェクション: 「/owners」ページの「lastName」パラ...	3時間前	報告済
高	クロスサイトスクリプティング: 「/owners」ページの「lastNam...	3分前	報告済
中	「SHA-1」ハッシュアルゴリズムを使用: Security.java	6分前	報告済
中	ハードコードされたパスワードを使用: SpringInputPasswordField...	3分前	報告済

A red annotation next to the first row states: **修正した脆弱性は「最後の検出」が更新されていない** (The updated vulnerability is not updated in the 'Last Detected' column).



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。

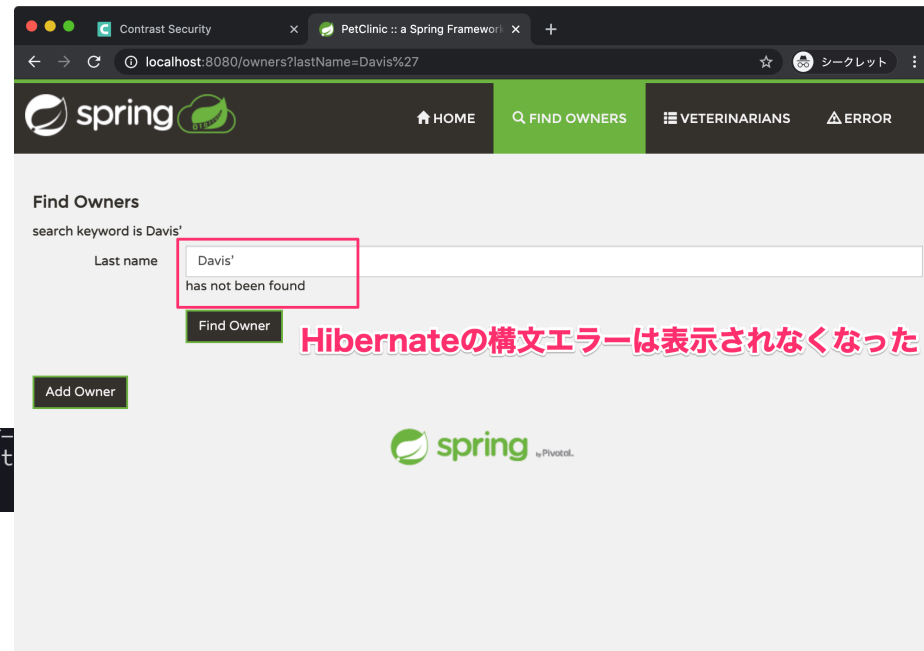
修正前に表示されていた構文エラーは表示されなくなりました。

以下は実行されたクエリのログです。

```
1, pets1.type_id as type_id_1_1, pets1.owner_id as owner_id_4  
from owners owner0 left outer join pets pets1 on owner0.id=pet  
st_name like 'Davis\''
```

「\」がエスケープされている

「\」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

