

クロスサイトスクリプティングの 解説と対策

この動画で説明すること

クロスサイトスクリプティングの説明と脆弱性の修正

1. クロスサイトスクリプティング(XSS)が起こる原因
2. Contrast Securityで検知
3. 検査パターンでXSSを確認
4. XSSを修正
5. Contrast Securityで修正を確認
6. 検査パターンでXSSの修正を確認

3,6については任意です



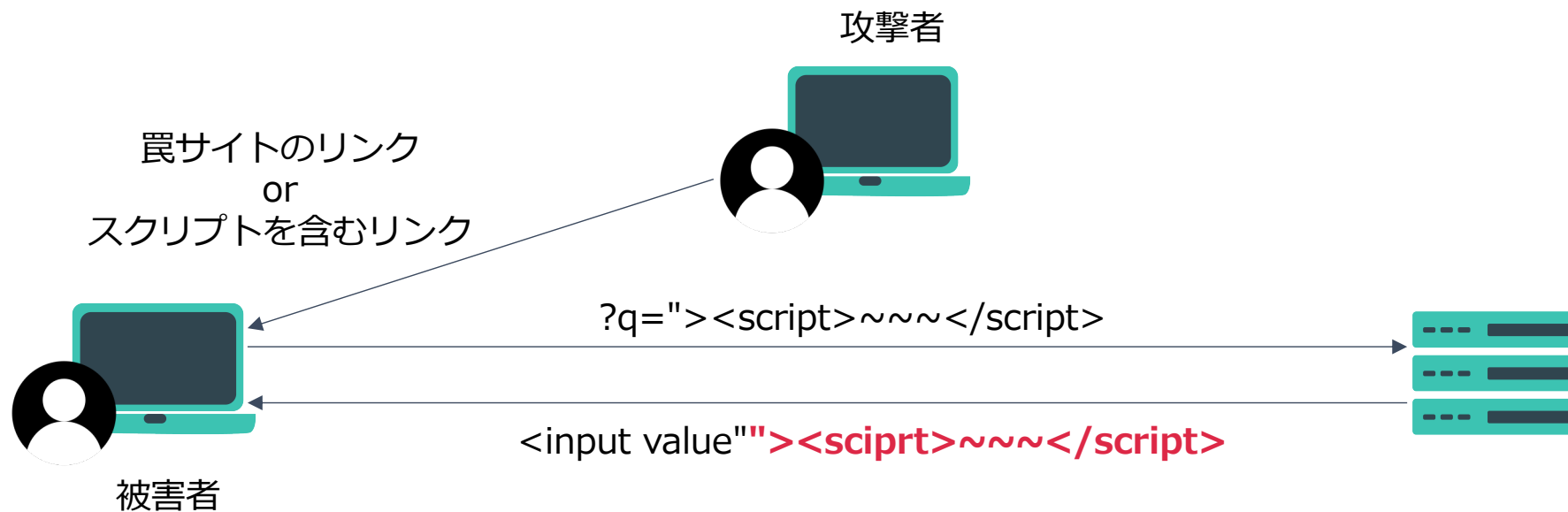
XSSの原因



XSSの原因 (1/3)

XSSの概要

XSSは主に外部からの入力値に対して、HTMLにおける特殊文字をコンテキストに応じてエスケープ処理せずにレスポンスに出力していることが原因で発生します。



XSSの原因 (2/3)

原因はHTMLにおける特殊文字をエスケープしていないため

以下のJSPのコードはXSSの脆弱性があります。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head><title>Title</title></head>
<body>
<p>user input: <%= request.getParameter("q") %></p>
<form action="/xss">
<input type="text" name="q" value="<%= request.getParameter("q") %>">
<button type="submit">send</button>
</form>
</body>
</html>
```



XSSの原因 (3/3)

原因はHTMLにおける特殊文字をエスケープしていないため

JSP標準のスキプト式を使用した出力ではHTMLの特殊文字がエスケープされません。結果として、HTMLタグ、属性へ干渉できる特殊文字が入力された場合、下記のようなHTMLが生成されます。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head><title>Title</title></head>
<body>
<p>user input: "><script>~~~</script></p>
<form action="/xss">
<input type="text" name="q" value="""><script>~~~</script>">
<button type="submit">send</button>
</form>
</body>
</html>
```

HTMLのタグや属性の文字リテラルを強制的に終了させ、任意のHTMLタグを挿入可能なことがわかります。



Contrast Assessでの検知



XSSの検知(1/2)

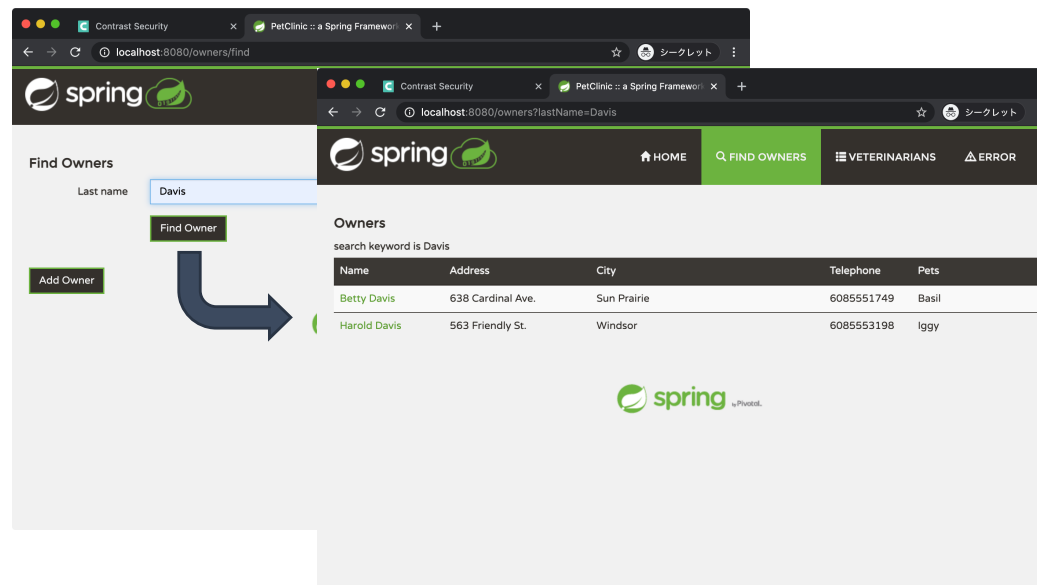
実際にContrast Assessを用いて検知

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。



XSS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 shows the Contrast Security interface. On the left, a table lists detected vulnerabilities:

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...	8日前	報告済
高	クロスサイトスクリプティング: 「/owne...	17分前	報告済
高	OSコマンドインジェクション: 「/rest/e...	6日前	報告済

A red box highlights the XSS entry, with the text "検索機能で検出したXSS" (XSS detected by search function) overlaid. The right side of the image shows a detailed view of the XSS vulnerability:

- Initial detection: 2月6日 2021
- Last detection: 2月14日 2021
- Disclosure period: 9+日
- What happened? **脆弱性を検出するまでにエージェントが観測したデータの流れ** (Data flow observed by the agent before the vulnerability was detected)
- Request: GET /owners?lastName=**contrast-redacted-name**
- Code snippet: org.thymeleaf.engine.AbstractTextualTemplateEvent#writeContent(), line 218
- Response: Davis

Below the detailed view, the question "どんなリスクであるか?" (What is the risk?) is visible.



アプリケーションでの確認



検査パターンで確認(1/2)

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XSSが存在するかは主に下記の観点で確認します。

観点	入力値
HTML構文に干渉する文字列がエスケープされて出力されるか	<ol style="list-style-type: none">1. <code>"><script>alert(1)</script></code>2. <code>'><script>alert(1)</script></code>3. <code>[space]onclick=alert(1)</code>4. <code>");alert(1)//</code>5. <code>');alert(1)//</code>
URI属性に出力時javascriptスキームとして有効にならないよう制御されているか	<ol style="list-style-type: none">1. <code>javascript:alert(1)</code>



検査パターンで確認(2/2)

文字リテラルが崩れるパターンで確認

Contrast AssessがXSSを検知した機能を前ページの観点で確認してみます。

`<script>alert(1)</script>`を入力した場合、検索結果画面でJavaScriptのalert関数が動作します。

localhost:8080の内容
1
OK

アプリケーション上で実装していないスクリプトが動作

```
<h2>Find Owners</h2>
<div>
  <label>search keyword is <span><script>alert(1)</script></span></label>
</div>
<form action="/owners" method="get"
  class="form-horizontal" id="search-owner-form">
  <div class="form-group">
    <div class="control-group" id="lastNameGroup">
      <label class="col-sm-2 control-label">Last name </label>
      <div class="col-sm-10">
        <input class="form-control" size="30"
          maxlength="80" id="lastName" name="lastName" value="&lt;script&gt;alert(1)&lt;/script&gt;" />
        <p>has not been found</p>
      </div></span>
    </div>
  </div>
</form>
```

エスケープされていない

エスケープされている

localhost を待機しています...

実際に生成されたHTMLを確認します。

1箇所目の出力ではタグ文字がエスケープされておらず、前述のスクリプトが動作したことがわかります。



XSSを修正



XSSを修正(1/3)

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、ユーザの入力値をレスポンスに出力していることがわかります。エージェントは出力までに特殊文字をエスケープするメソッドを観測していないこともわかります。

The screenshot shows a web browser window with the Contrast Security interface. The main content is a report titled "クロスサイトスクリプティング: 「/owners」ページの「lastName」パラメータ". The report details a high-severity vulnerability. A red box highlights a code snippet where the user input "Davis" is written to the response without being escaped.

HTTPのパラメータを取得	string[] = wrapper.getParameterValues("lastName") getParametersStartingWith() @WebUtils.java:700	lastName = contrast-redacted-name
HTTPレスポンスで信頼できないデータを送信	writer.write("Davis") writeContent() @ AbstractTextualTemplateEvent.java:218	Davis

レスポンスに出力するまでにエスケープする処理がない



XSSを修正(2/3)

修正方法を確認

通常、テンプレートエンジン側で出力時にHTMLにおける特殊文字をエスケープして出力する機構を持っています。

Contrast UIの修正タブにある適切なエンコード方法を確認し、使用しているテンプレートエンジンで適切な出力方法へ修正します。



spring-petclinic-yuya

URL: / | 言語: Java | 重要性: 中

報告済

概要 詳細 HTTP情報 修正方法 備考 アクティビティ

パラメータの入力または出力を削除できる場合は、削除してください。それ以外の場合は、パラメータがページのどこでレンダリングされるかに基づいて、適切なエンコード方式を使用してパラメータをエンコードしてください。

コンテキスト	例	危険な文字	エンコード	注記
HTMLエンティティ	<code><div>{untrusted}</div></code>	<code>&</code> <code><>'"/</code>	<code>&#xHH;</code>	
HTML属性	<code><input value="{untrusted}"></code>	英数字以外	<code>&#xHH;</code>	これは、 <code>href</code> 、 <code>src</code> 、 <code>style</code> などの複雑な属性や、 <code>onclick</code> などのイベントハンドラには安全ではありません。 <code>javascript:</code> や <code>data:</code> などの安全ではないURLやCSS式を回避するために、ホワイトリストの十分な検証が必要です。
URLパラメータ	<code></code>	英数字以外	<code>%HH</code>	
CSS	<code>p { color : {untrusted} ; }</code>	英数字以外	<code>\HH</code>	これは、 <code>url</code> 、 <code>behavior</code> 、 <code>-moz-binding</code> などの複雑なプロパティには安全ではありません。JavaScriptのURLやCSS式を回避するために、ホワイトリストの十分な検証が必要です。
JavaScript	<code>var name = '{untrusted}';</code>	英数字以外	<code>\xHH;</code>	一部のJavaScript関数では、ホワイトリストによる検証が行われな い限り、信頼できないデータを入力として安全に使用することは できません。



XSSを修正(3/3)

コードを修正

確認した修正方法でXSSを修正していきます。

8行目でThymeleafのth:utextを使用して入力値を出力していることがわかります。th:utextは出力時にHTMLの特殊文字をエスケープしません。

そのため、この出力箇所をth:utextからth:textへ修正します。

参考

https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf_ja.html

```
1 <html xmlns:th="https://www.thymeleaf.org"
2   th:replace="@{fragments/layout :: layout (#{::body}, 'owners')}">
3
4 <body>
5
6 <h2>Find Owners</h2>
7
8 <div th:unless="${keyword == null}"
9   <label>search keyword is <span th:utext="${keyword}" /></label>
10 </div>
11
12 <form th:object="${owner}" th:action="@{/owners}" method="get"
13   class="form-horizontal" id="search-owner-form">
14 <div class="form-group">
15 <div class="control-group" id="lastNameGroup">
16 <label class="col-sm-2 control-label">Last name </label>
17 <div class="col-sm-10">
18 <input class="form-control" th:field="*{lastName}" size="30"
19   maxLength="80" /> <span class="help-inline"><div
20   th:if="${#fields.hasAnyErrors()}">
21 <p th:each="err : ${#fields.allErrors()}" th:text="${err}">Error</p>
22 </div></span>
23 </div>
24 </div>
25 </div>
26 </div>
27 </div>
28 </div>
29 </div>
30 </div>
31 </div>
32 </div>
33 </div>
34 </div>
35 </div>
36 </div>
37 </div>
38 </div>
39 </div>
40 </div>
41 </div>
42 </div>
43 </div>
44 </div>
45 </div>
46 </div>
47 </div>
48 </div>
49 </div>
50 </div>
51 </div>
52 </div>
53 </div>
54 </div>
55 </div>
56 </div>
57 </div>
58 </div>
59 </div>
60 </div>
61 </div>
62 </div>
63 </div>
64 </div>
65 </div>
66 </div>
67 </div>
68 </div>
69 </div>
70 </div>
71 </div>
72 </div>
73 </div>
74 </div>
75 </div>
76 </div>
77 </div>
78 </div>
79 </div>
80 </div>
81 </div>
82 </div>
83 </div>
84 </div>
85 </div>
86 </div>
87 </div>
88 </div>
89 </div>
90 </div>
91 </div>
92 </div>
93 </div>
94 </div>
95 </div>
96 </div>
97 </div>
98 </div>
99 </div>
100 </div>
```



Contrast Securityで修正を確認



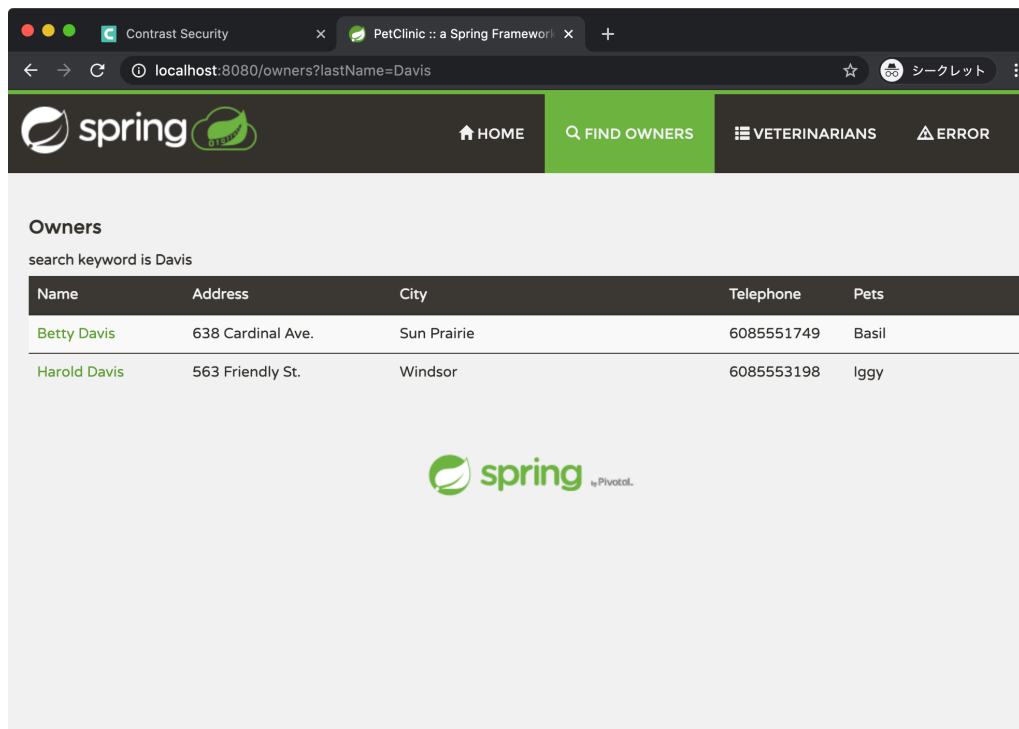
Contrastで修正を確認(1/2)

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。



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

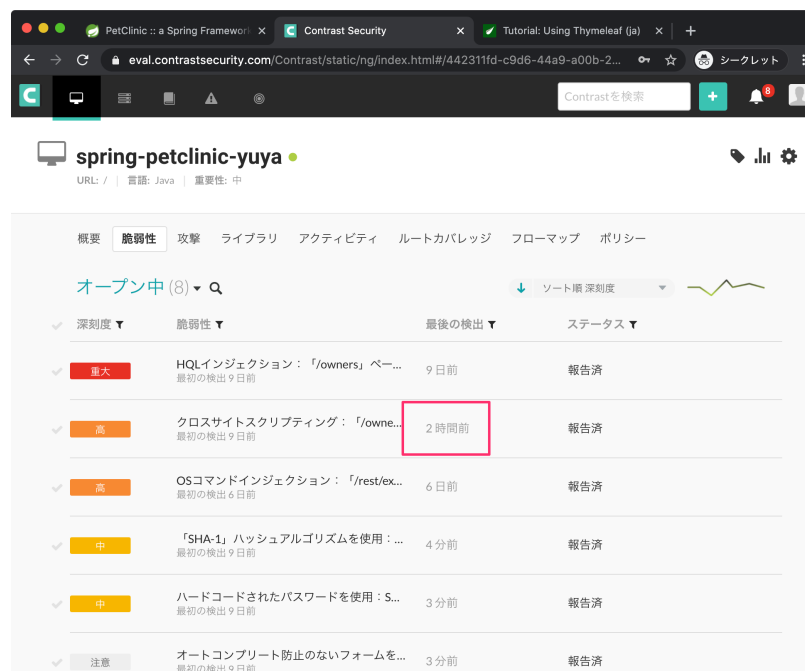
脆弱性を修正後に再ビルドを行い、エージェントを組み込みアプリケーションを起動します。

クロスサイトスクリプティングの「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 a Spring PetClinic application. The '脆弱性' (Vulnerabilities) tab is active, displaying a table of detected issues. The table has columns for severity, vulnerability name, last detected time, and status. The entry for 'クロスサイトスクリプティング' (Cross-site scripting) is highlighted with a red box, indicating it was last detected 2 hours ago.

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...	9日前	報告済
高	クロスサイトスクリプティング: 「/owne...	2時間前	報告済
高	OSコマンドインジェクション: 「/rest/ex...	6日前	報告済
中	「SHA-1」ハッシュアルゴリズムを使用: ...	4分前	報告済
中	ハードコードされたパスワードを使用: S...	3分前	報告済
注意	オートコンプリート防止のないフォーム...	3分前	報告済



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)

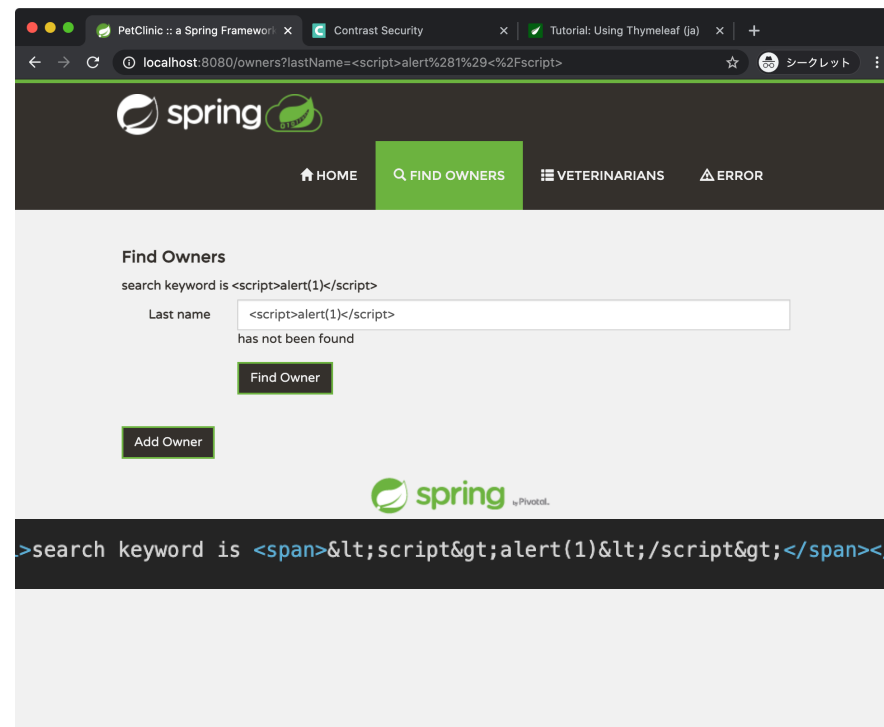
文字リテラルが崩れるパターンで確認

検索機能に存在していたクロスサイトスクリプティング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「`<script>alert(1)</script>`」を入力し検索します。

修正前に動作していたalert関数は表示されなくなりました。

HTMLの特殊文字もエスケープされていることが確認できます。



まとめ



まとめ

XSSの原因と対策

1. XSSが起こる原因

ユーザーの入力をエスケープせずに出力しているため。

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

