SCENARIO

The application contains a simple DOM based cross site scripting vulnerability in the stock checker functionality which uses **document.write** function to write data out to the page and that function is called with data from **location.search** which we can control using the website URL but there exists a select element in which the data is enclosed in, so we will try to exploit it by injecting some payload containing malicious script in the comment box.

**PROCEDURE**

1. Go to the vulnerable web application.
2. Open the element inspector tab and got to debugger tab in that.
3. Now, being in the debugger tab we can see some files there, then select the file named **product?productId=x** and in that file look for script tags.
4. Now in the script tags we can see a line of code like:

**var store = (new URLSearchParams(window.location.search)).get('storeId');**

1. By looking at the string we can easily make out that the application is getting the **storeId** parameter from the URL.
2. So now, we will try to inject a payload into that field but first we need to get out of those tags in order to force the application to execute whatever we need.
3. We closed the select tag first and then we will put an image tag to use onerror function and we’ll get the alert we wanted.

**PAYOAD**

/product?productId=1&storeId="></select><img%20src=1%20onerror=alert(“Hey!”)>

**REMEDIATION**

1. **Input Validation:** Adopt strict input validation and sanitization measures. Ensure that data passed to the document.write function is sanitized, preferably by using a well-maintained library that's explicitly designed for sanitizing user-controlled input against XSS. Data from location.search should never be directly passed without validation.
2. **Avoid Unsafe Functions:** Refrain from using functions like document.write that can introduce vulnerabilities when combined with user-controlled data. Instead, consider utilizing safer DOM manipulation methods, such as textContent or createElement, which do not interpret their input as HTML.
3. **Content Security Policy (CSP):** Implement a strong Content Security Policy (CSP) which restricts the use of inline scripts. This would help in mitigating the impact of any potential DOM-based XSS attacks as any inline script would not be executed by the browser.