SCENARIO

The application contains a simple DOM based cross site scripting vulnerability in the blog search box 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 the application uses innerHTML to modify HTML’s inner content’s div tag, 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  **(index)/?search=sa** and in that file look for script tags.
4. Now in the script tags we can see a line of code like:

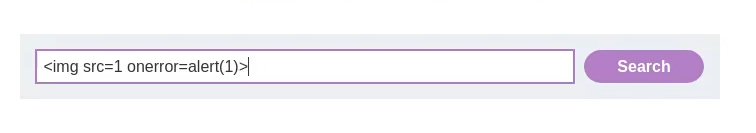
**var query = (newURLSearchParams(window.location.search)).get('search');**

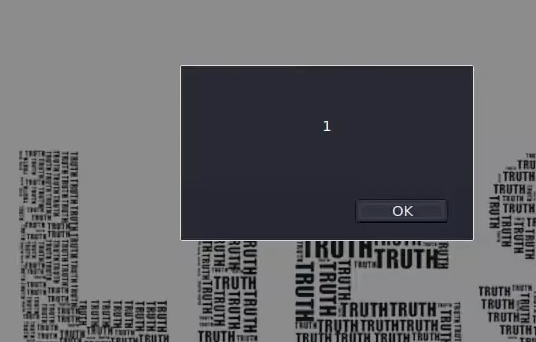
By looking at the string we can easily make out that the application is getting the **search** parameter from the URL.

1. 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.
2. We closed the colon and bracket first and then we will put an image tag to use onerror function and we’ll get the alert we wanted.

**PAYOAD**

/?search=');<img src=1 onerror=alert("Hey!")>

**PROOF OF CONCEPT**



**REMEDIATION**

1. **Avoid Using innerHTML and document.write:** One of the most effective ways to prevent DOM-based XSS attacks is to avoid using methods that can interpret strings as HTML or JavaScript. Use textContent instead of innerHTML to insert text into the DOM. This ensures that any text is treated as data and not executable code.
2. **Use Trusted Libraries for DOM Manipulation:** Use libraries or frameworks that have been designed to safely create and manipulate DOM elements. Libraries such as React or Angular automatically escape values to prevent XSS attacks.
3. **Implement Proper Output Encoding:** Ensure that dynamic data is properly encoded when added to HTML. For example, use entity encoding to ensure characters like < and > are rendered as text and not treated as HTML tags.