SCENARIO

The application contains a simple DOM based cross site scripting vulnerability in the feedback submission 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 jQuesry library’s $ selector to find an anchor element to modify HTML’s href attribute using data from **location.search**, so we will try to exploit it by injecting some payload containing malicious script in the feedback box.

**PROCEDURE**

1. Go to the vulnerable web application and navigate to submit feedback page.
2. Open the element inspector tab and got to debugger tab in that.
3. Enter any random string into the URL for **returnPath** and we can see that our string became a href attribute.
4. Now, being in the debugger tab we can see some files there, then select the file named **feedback?returnPath=/string** and in that file look for script tags.
5. Now in the script tags we can see a line of code like:

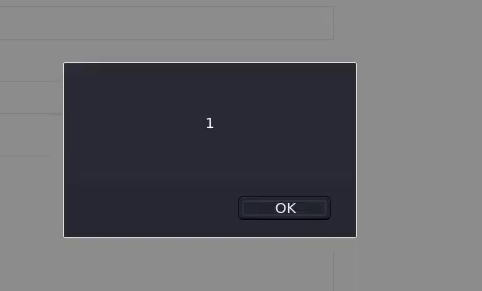
**$('#backLink').attr("href", (newURLSearchParams(window.location.search)).get('returnPath'));**

By looking at the string we can easily make out that the application is getting the **returnPath** 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.

**PAYLOAD**

1. /feedback?returnPath=/abcde
2. javascript:alert(document.cookie)

**PROOF OF CONCEPT**

**REMEDIATION**

1. **Avoid Direct Injection of User Input:** Never directly insert untrusted input into the DOM without proper validation and sanitization. This includes using user data to manipulate attributes or content on a page. If you have to use dynamic data, always validate and sanitize it before insertion.
2. **Use Safe APIs:** Prefer safe APIs like Element.textContent (rather than Element.innerHTML) or use frameworks like React that automatically escape values to prevent XSS attacks. When dealing with jQuery, it's important to be aware of its potential risks. Instead of using jQuery's .attr() to set href attributes from untrusted input, validate the URL to ensure it begins with a known safe protocol such as http:// or https://.
3. **Content Security Policy (CSP):** Implement a strong CSP header in your application to limit the execution of JavaScript. This will not only help to prevent XSS attacks but also restrict the capabilities of any successful XSS injection.