**Sri Lanka Institute of Information Technology**

B.Sc. (Hons) Information Technology- Cyber security

# 

Year 02 Semester 02

Web Security - IE2062

**Cross-site Scripting Lab Submission**

**IT 23 1843 12**

**AMANTH M A**

Date submitted-15/02/2025

**Contents**

[1](#_Toc190545463)

[**1.** **Reflected XSS into HTML context with nothing encoded** 3](#_Toc190545464)

[**2.** **Stored XSS into HTML context with nothing encoded** 5](#_Toc190545465)

[**3.** **DOM XSS in *document.writ*e sink using source *location.search*** 8](#_Toc190545466)

[**4.** **DOM XSS in *innerHTML* sink using source *location.search*** 11](#_Toc190545467)

[**5.** **DOM XSS in jQuery anchor href attribute sink using *location.search* source** 14](#_Toc190545468)

[**6.** **DOM XSS in jQuery selector sink using a hashchange event** 17](#_Toc190545469)

[**7.** **Reflected XSS into attribute with angle brackets HTML-encoded** 19](#_Toc190545470)

[**8.** **Stored XSS into anchor href attribute with double quotes HTML-encoded** 22](#_Toc190545471)

[**9.** **Reflected XSS into a JavaScript string with angle brackets HTML encoded** 24](#_Toc190545472)

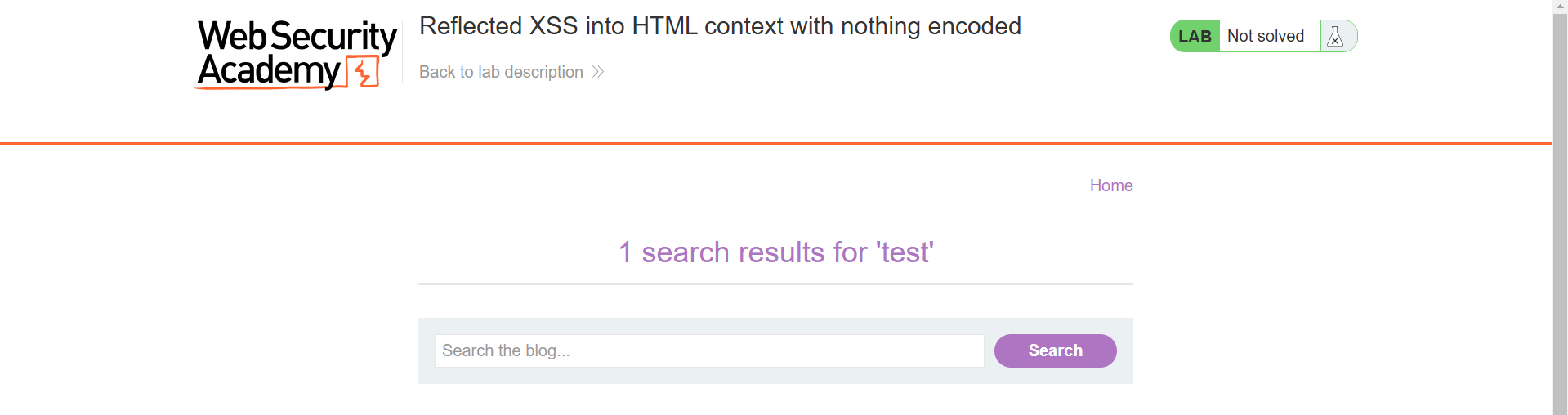
# **Reflected XSS into HTML context with nothing encoded**

**LAB**

This lab contains a simple reflected cross-site scripting vulnerability in the search functionality. To solve the lab, perform a cross-site scripting attack that calls the alert function.

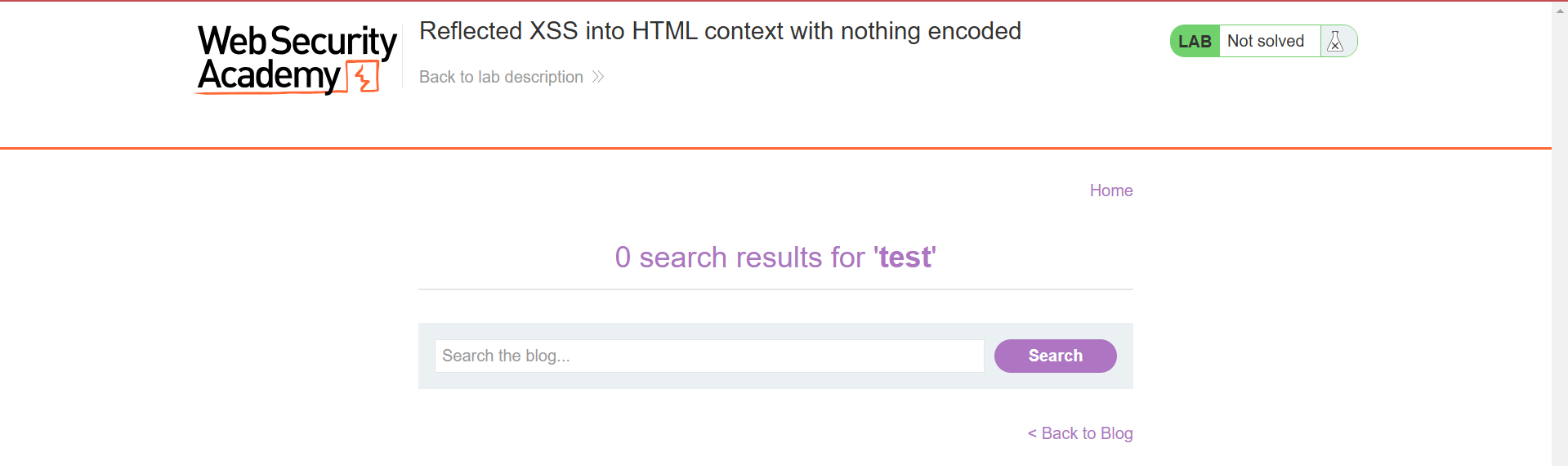
**SOLUTION**

**STEP 01:** Open the Lab and Identify the input field. It’s usually a Search Box. Test a Basic Input Reflection like a simple string “test”.



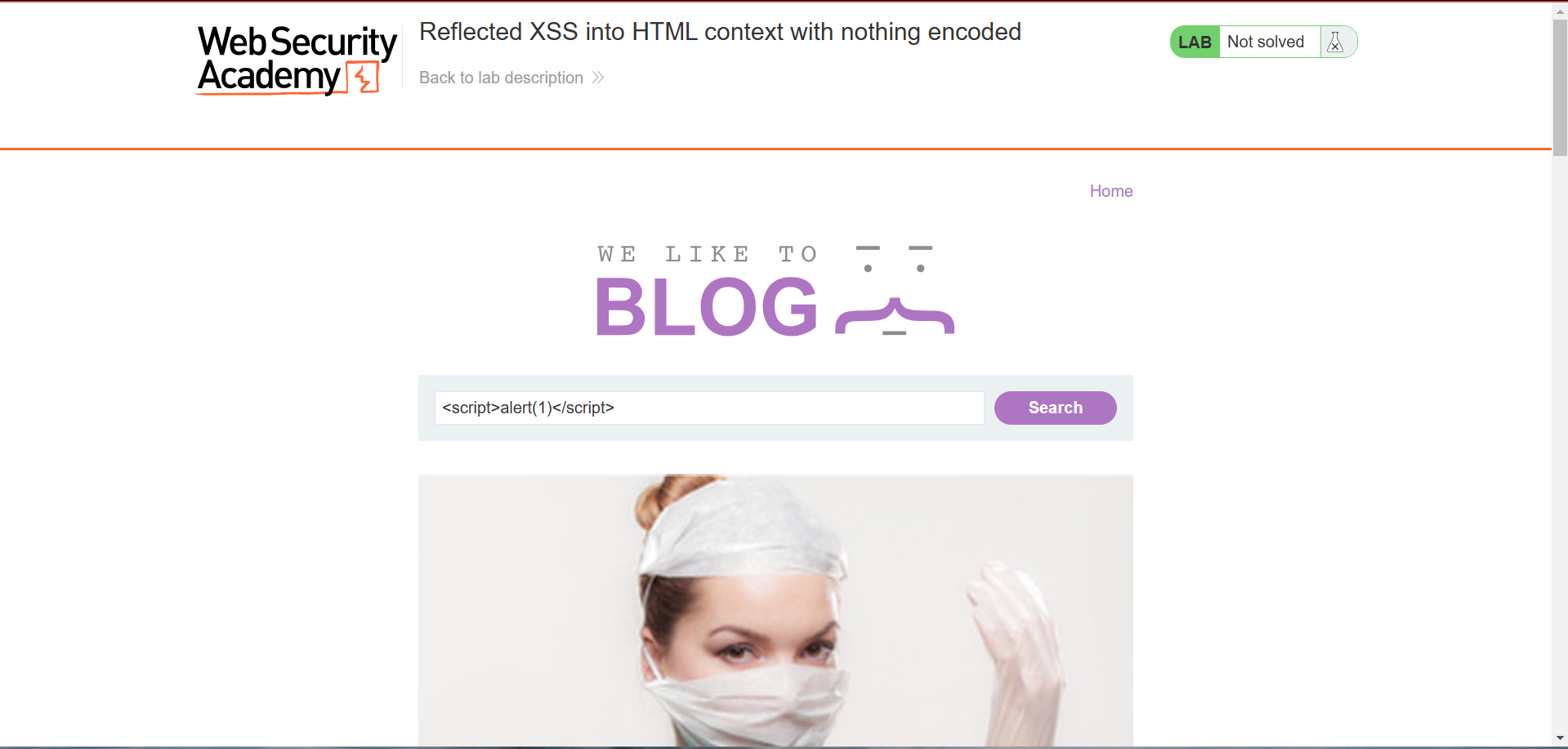
**STEP 02:** Check if appears in the response. If it appears try injecting a HTML tag.

Ex: <b> test </b>

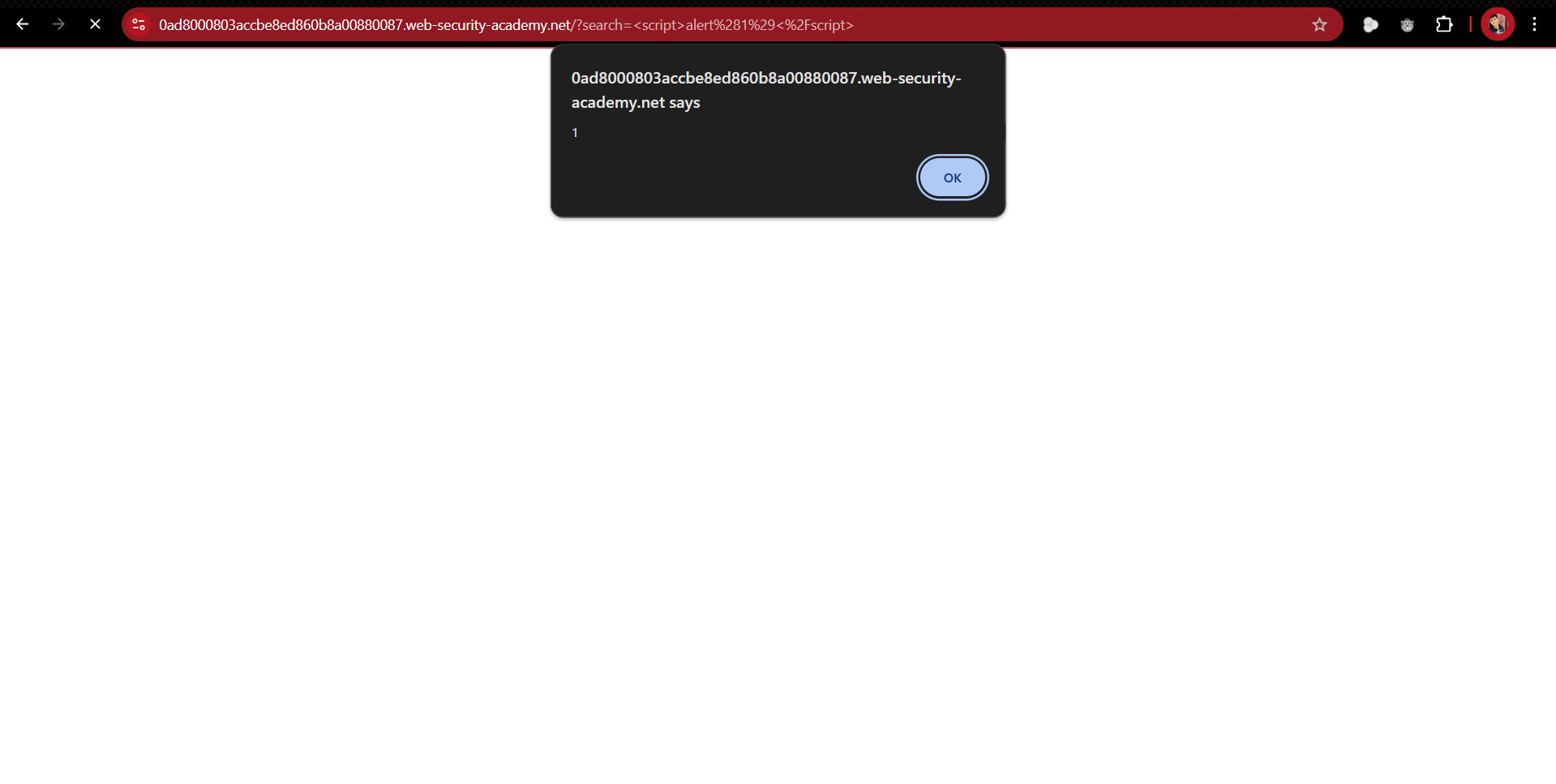
****

If the text appears bold, it means the input is directly rendered as HTML, making it vulnerable to XSS.

**STEP 03:** Inject a malicious Script.

****

If the page executes this script and shows an alert box with "1", the site is vulnerable.



**EXPLANATION**

In this lab, user input is reflected/printed to an HTML page **without any encoding or sanitization.**

**Vulnerability:** Reflected Cross-Site Scripting (XSS)

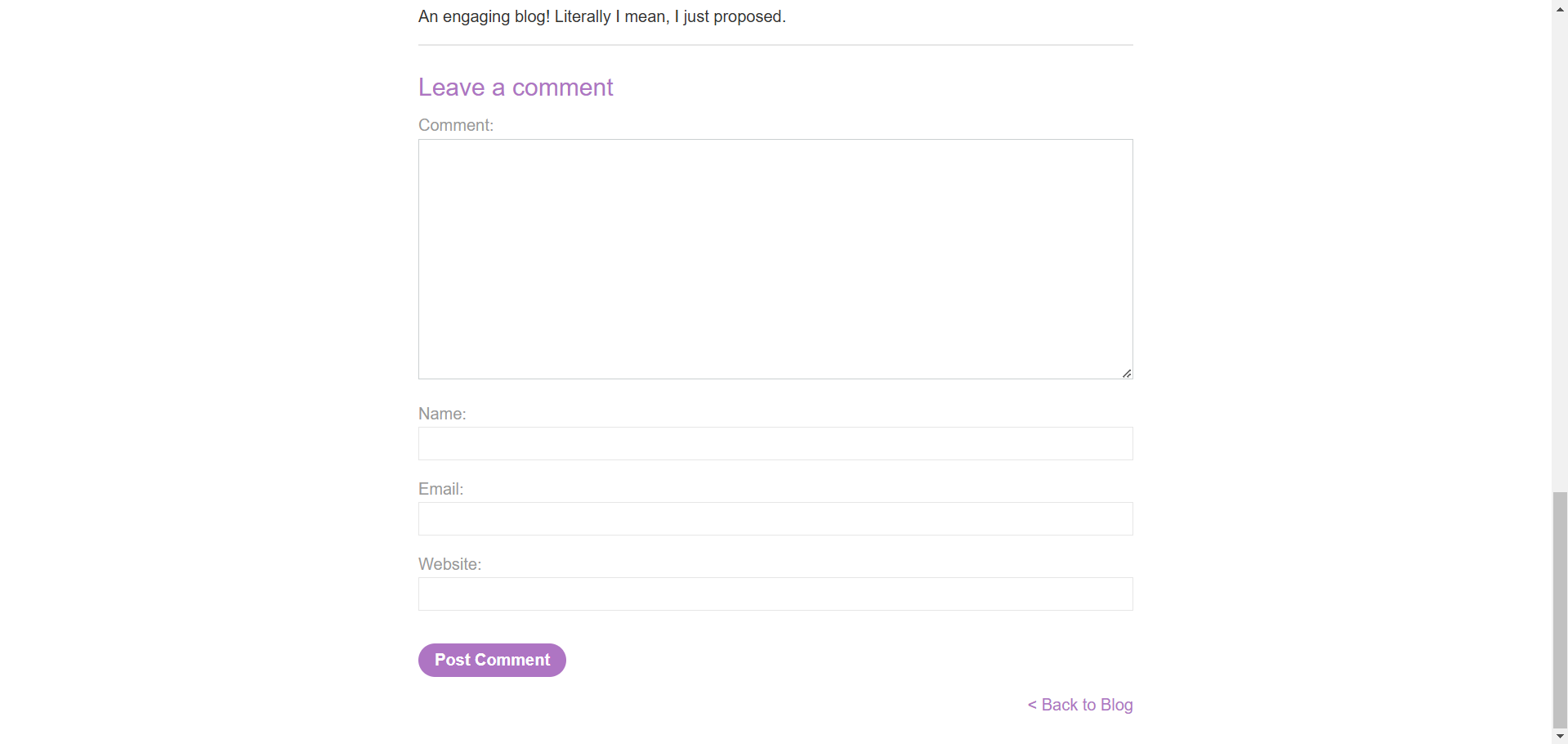
# **Stored XSS into HTML context with nothing encoded**

**LAB**

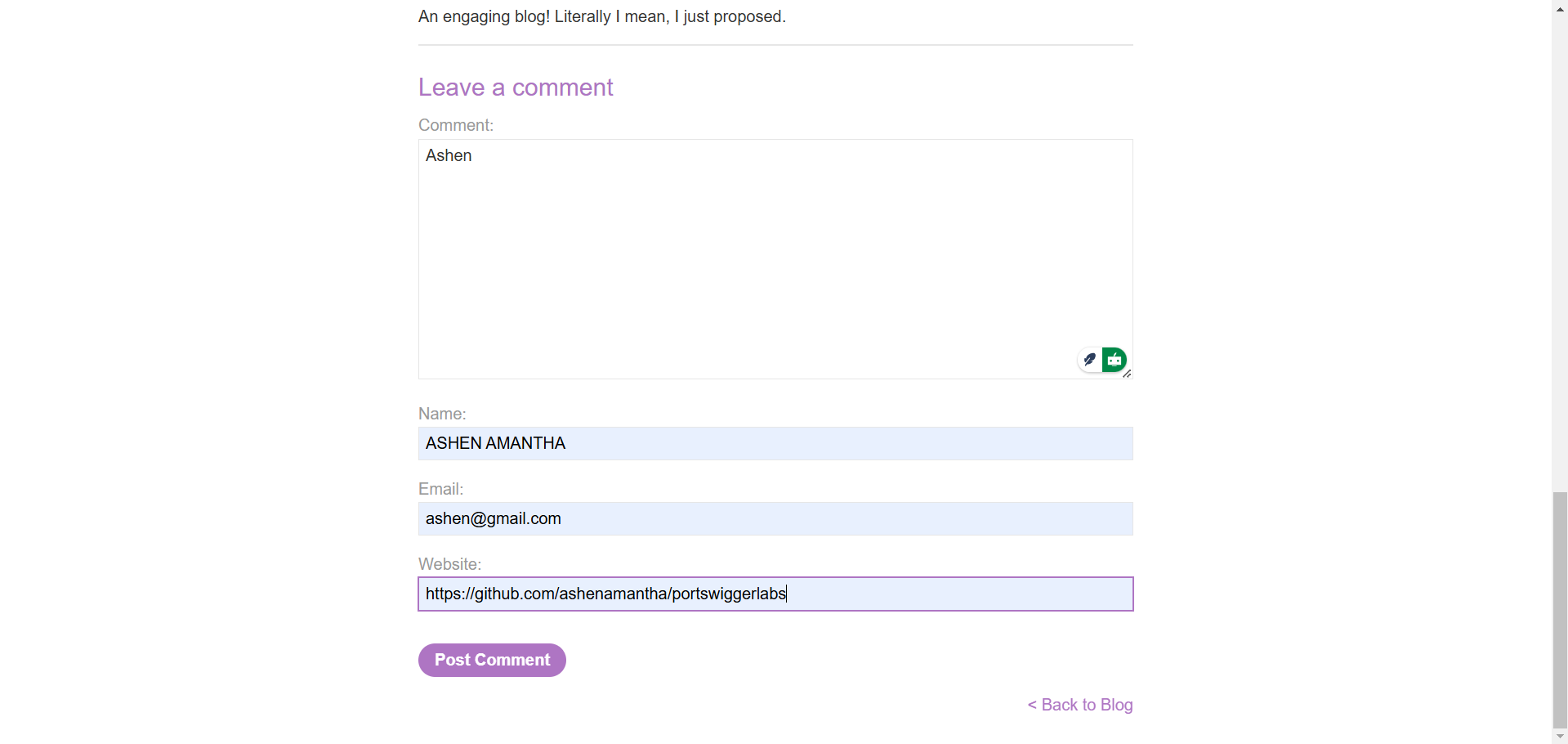
This lab contains a stored cross-site scripting vulnerability in the comment functionality. To solve this lab, submit a comment that calls the alert function when the blog post is viewed.

**SOLUTION**

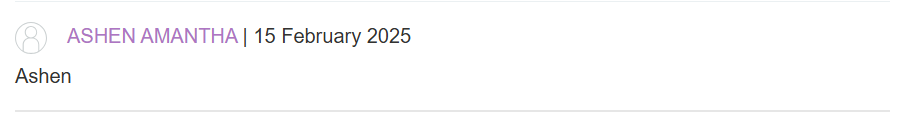
**STEP 01:** Open the Lab and Identify the input field. Look for a comment section, guestbook, or review submission form. When you submit a comment, it is stored and displayed on the page.



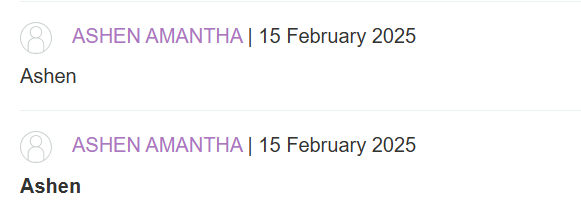
**STEP 02:** Test a basic input reflection



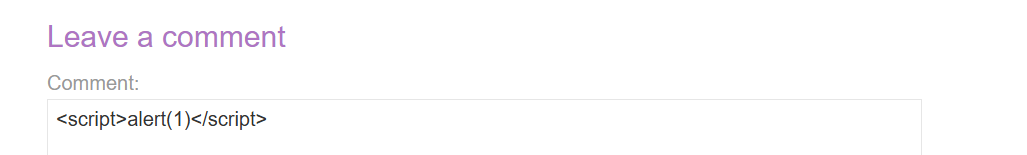
**STEP 03:** If "Ashen" appears in the comments, try submitting an HTML tag



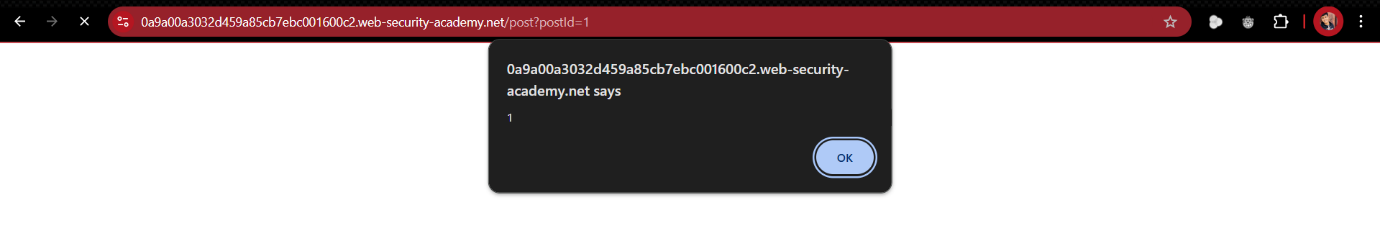
If the text appears bold, it means the input is directly rendered as HTML, making it vulnerable to XSS.



**STEP 04:** Inject a malicious Script.



If the page executes this script and shows an alert box with "1", the site is vulnerable.



**EXPLANATION**

In this lab, user input is stored on the server and then displayed on a web page without encoding.

Unlike reflected XSS, stored XSS is more dangerous because the payload persists and executes for multiple users.

**Vulnerability:** Stored Cross-Site Scripting (XSS)

# **DOM XSS in *document.writ*e sink using source *location.search***

**LAB**

This lab contains a DOM-based cross-site scripting vulnerability in the search query tracking functionality. It uses the JavaScript ***document.write*** function, which writes data out to the page. The ***document.write*** function is called with data from ***location.search***, which you can control using the website URL.

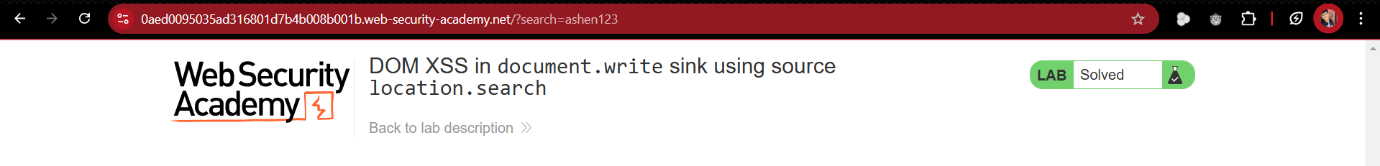
To solve this lab, perform a cross-site scripting attack that calls the alert function.

**SOLUTION**

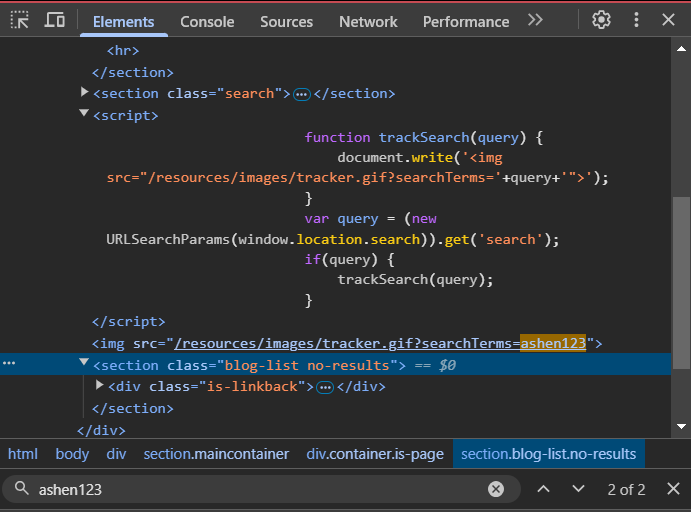
**STEP 01:** Open the Lab and Identify the input field. Test with a random string and look at the URL format when you perform the search.



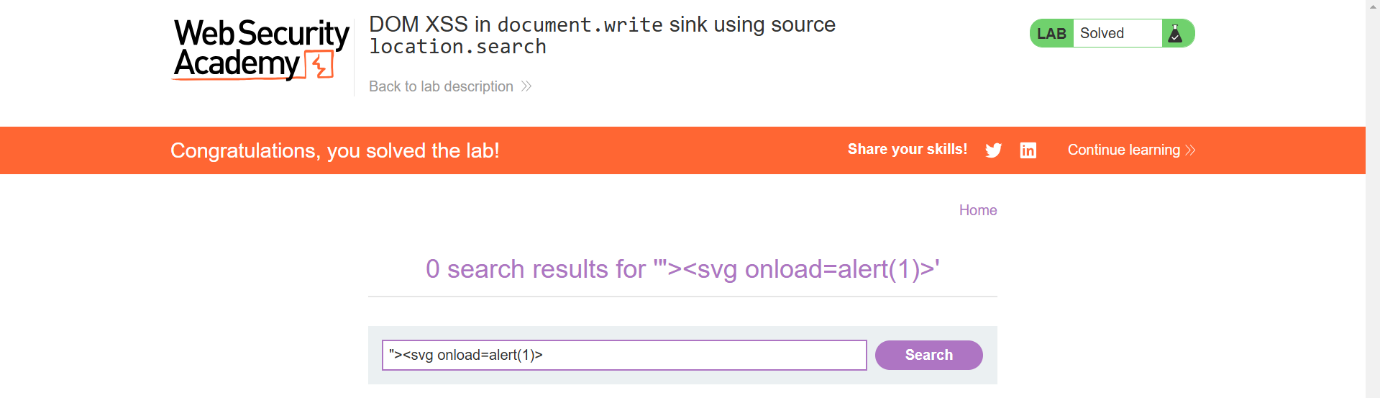
Look at the URL format .



**STEP 02:** Open the Inspect by simply right clicking. Search for the entered random string.

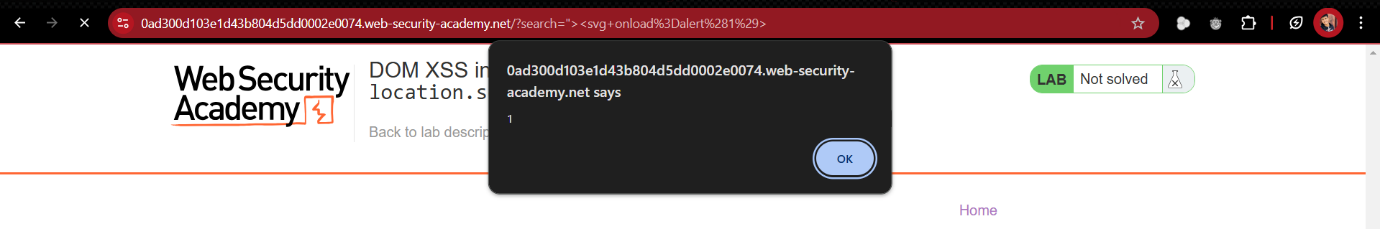


**STEP 03:** After Identifying the document.write() vulnerability Inject the malicious Script.



When <script> tags are **blocked** by filters, attackers often use alternative HTML elements to execute JavaScript. The <svg> (Scalable Vector Graphics) tag is one such element that supports JavaScript execution via **event handlers** like onload, onmouseover, or onclick.

If the page executes this script and shows an alert box with "1", the site is vulnerable.



**EXPLANATION**

document.write() is a JavaScript function that writes directly into the webpage.

If used insecurely, it can introduce **DOM-based XSS vulnerabilities**.

**Vulnerability:** DOM XSS in document.write()

# **DOM XSS in *innerHTML* sink using source *location.search***

**LAB**

This lab contains a DOM-based cross-site scripting vulnerability in the search blog functionality. It uses an innerHTML assignment, which changes the HTML contents of a div element, using data from location.search.

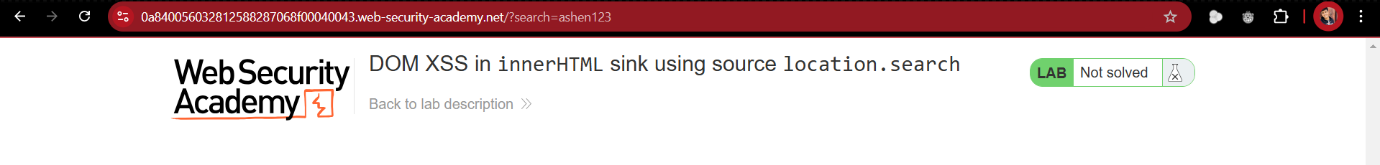
To solve this lab, perform a cross-site scripting attack that calls the alert function.

**SOLUTION**

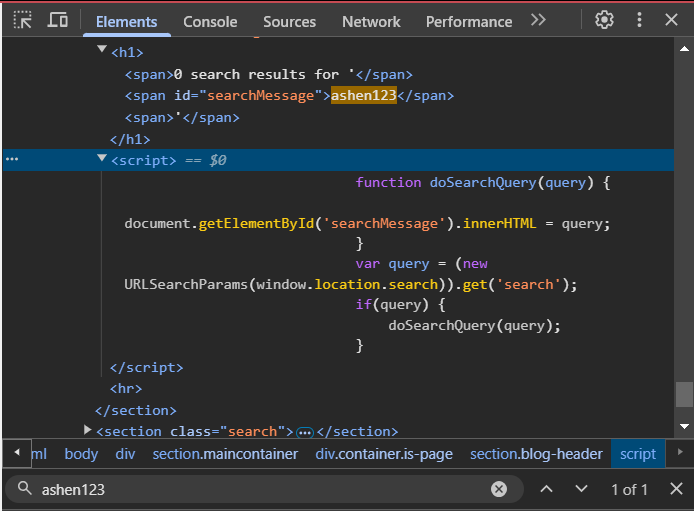
**STEP 01:** Open the Lab and Identify the input field. Test with a random string and look at the URL format when you perform the search.



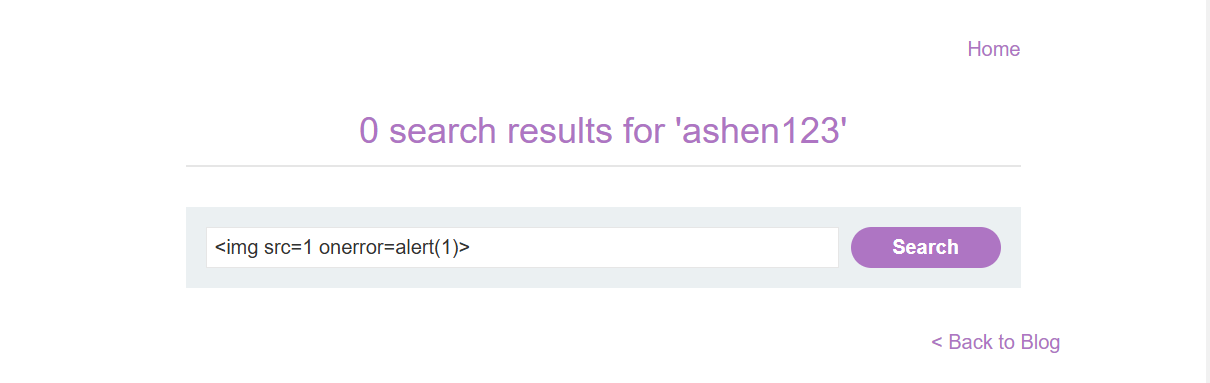
look at the URL format when you perform the search.



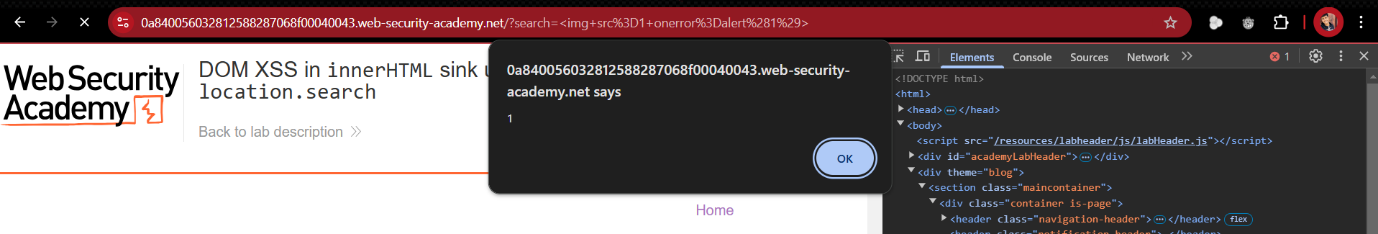
**STEP 02:** Open the Inspect by simply right clicking. Search for the entered random string. Check the behavior and identify the vulnerability.



**STEP 03:**  Inject the malicious Script.

If <script> is blocked, try alternative payloads like **<svg onload=alert(1)>** or **<img src=1 onerror=alert(1)>** 

If the page executes this script and shows an alert box with "1", the site is vulnerable.



**EXPLANATION**

This lab is vulnerable because it **directly assigns** user input from location.search to innerHTML, leading to DOM-based XSS.

**Vulnerability**: location.search is inserted into innerHTML without sanitization.

# **DOM XSS in jQuery anchor href attribute sink using *location.search* source**

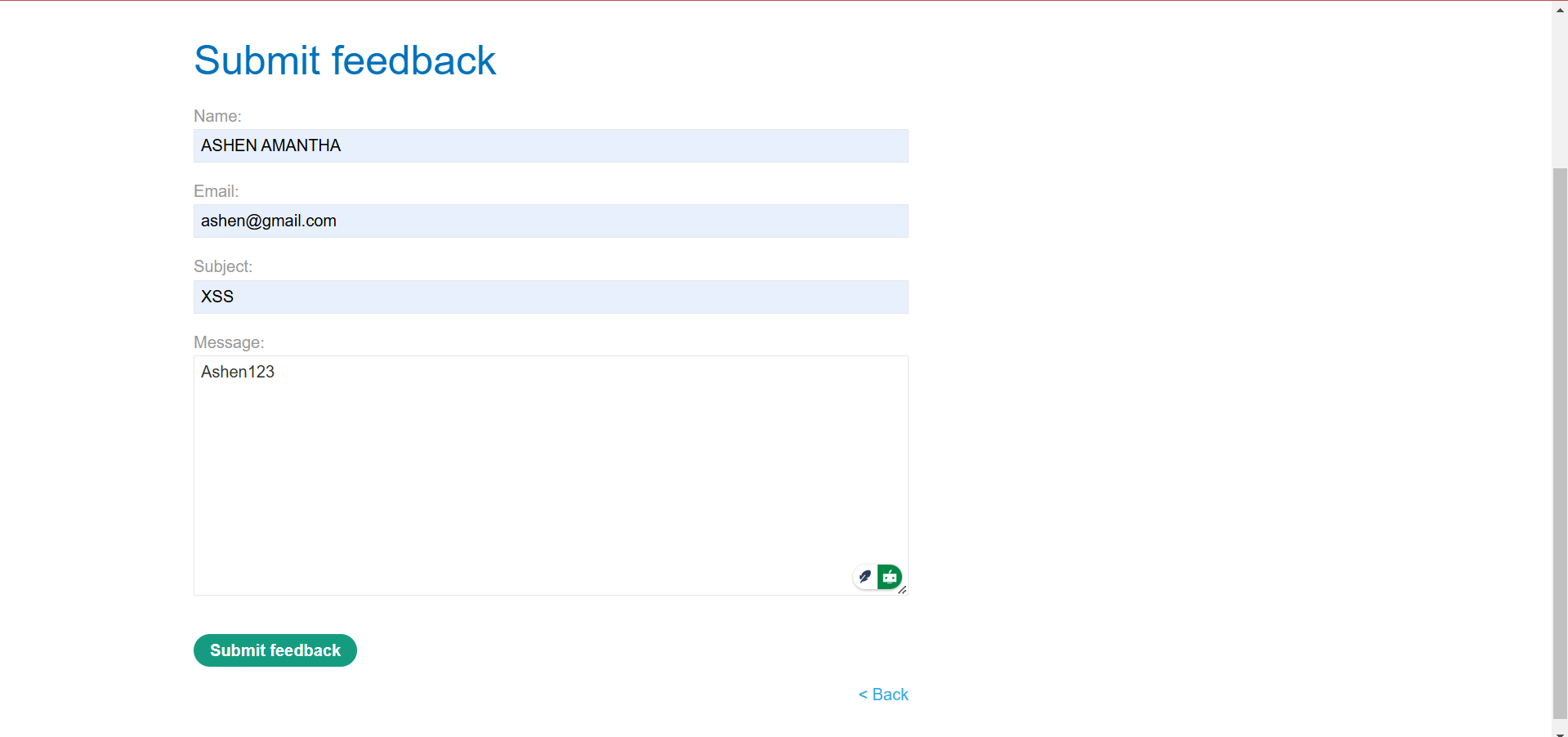
**LAB**

This lab contains a DOM-based cross-site scripting vulnerability in the submit feedback page. It uses the jQuery library's $ selector function to find an anchor element, and changes its **href** attribute using data from **location.search**.

To solve this lab, make the "**back**" link alert **document.cookie**.

**SOLUTION**

**STEP 01:** Open the Lab and Identify the input field. Test with a random strings and look at the URL format when you perform the Submit feedback.

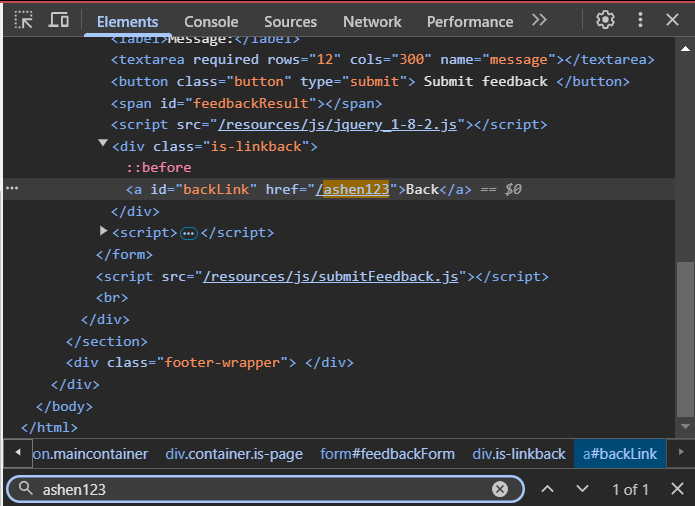


****

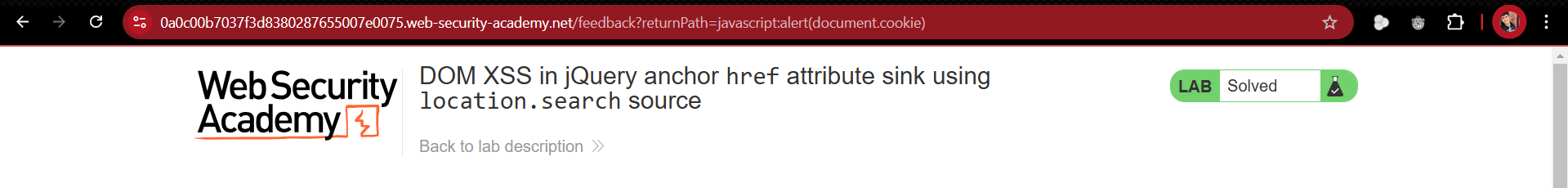
**STEP 02:** Change the URL. Modify the query parameter returnPath to include / followed by a random string.



**STEP 03:** Inspect the Elements by simply right clicking. You should see the random string (/ashen123) inside the href attribute.

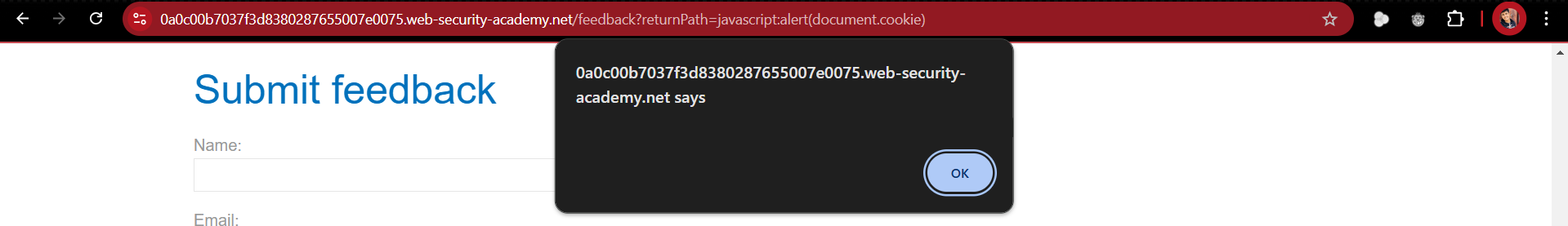


**STEP 04:** Inject a malicious Payload. Change the returnpath to **javascript:alert(document.cookie).**



After modifying the URL with the malicious payload:

Hit Enter to load the page with the modified URL. Click the "back" link or similar navigation element. When you click this link, The href value (javascript:alert(document.cookie)) will be executed as JavaScript, and it will alert the document's cookies.



**EXPLANATION**

When the returnPath parameter is inserted directly into the href attribute without proper sanitization, the malicious javascript: payload is treated as executable JavaScript.

How document.cookie Works: document.cookie returns the cookies associated with the current domain. By triggering alert(document.cookie), you can view the cookies.

**Vulnerability**: DOM XSS with javascript: in href

# **DOM XSS in jQuery selector sink using a hashchange event**

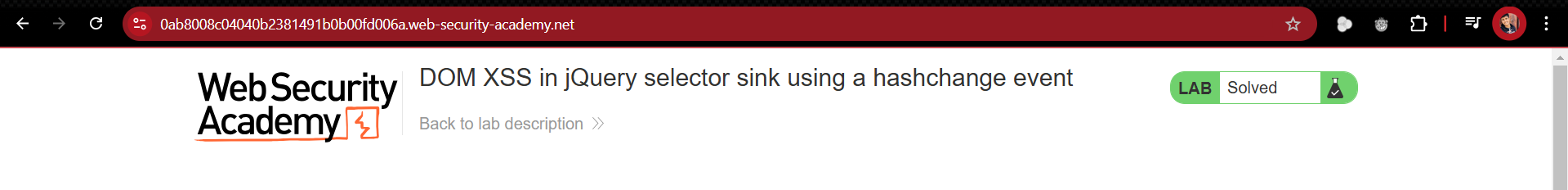
**LAB**

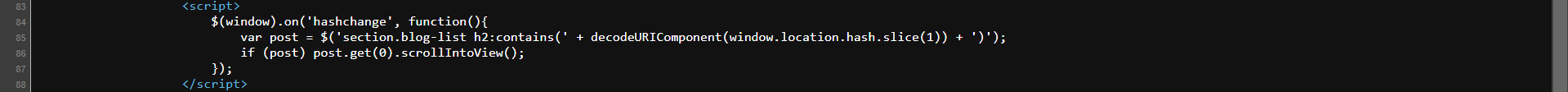
This lab contains a DOM-based cross-site scripting vulnerability on the home page. It uses jQuery's $() selector function to auto-scroll to a given post, whose title is passed via the location.hash property.

To solve the lab, deliver an exploit to the victim that calls the print() function in their browser.

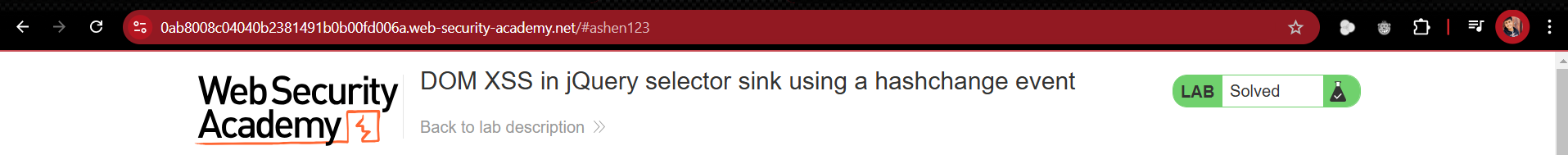
**SOLUTION**

**STEP 01:** Open the Lab and check the source code for the jQuery’s Selector function.



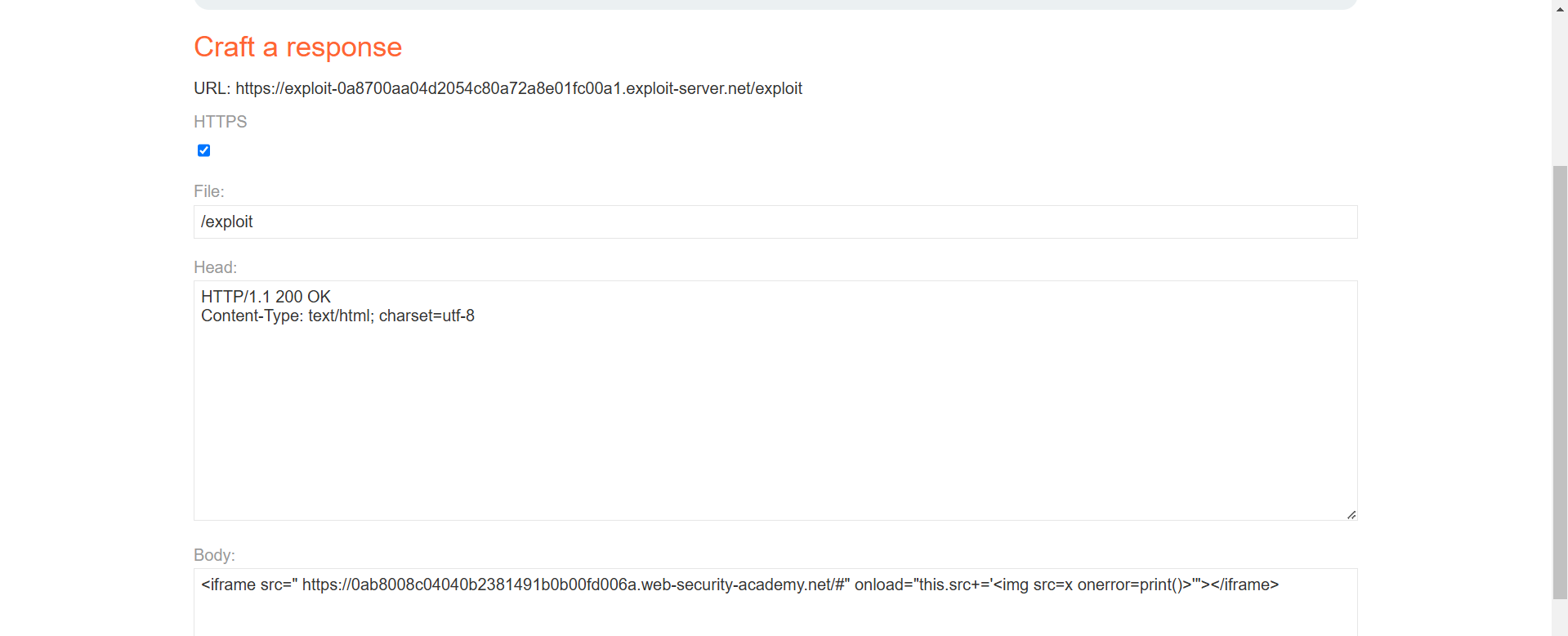
****

**STEP 02:**  As in the code when ever the hash changes the function called and the **selector** auto-scroll to a post by taking its title from ***location.hash.***

****

**STEP 03:** Modify the URL with the payload.

<iframe src=" https://0ab8008c04040b2381491b0b00fd006a.web-security-academy.net/#" onload="this.src+='<img src=x onerror=print()>'"></iframe>

**STEP 04:** Go the Exploit server and deliver it to the victim.

**EXPLANATION**

The vulnerability exists because jQuery’s $() selector directly uses location.hash from the URL.We inject JavaScript via location.hash, which is processed as HTML, leading to DOM-based XSS.Instead of a simple payload, we use an iframe to automate execution when the victim visits our exploit page.

**<iframe>** can embed any webpage inside another.

Attacker’s use **<iframe>** to load a vulnerable site without user interaction.

The onload event allows us to modify the page dynamically, injecting malicious payloads.

This technique is useful for silent XSS attacks where victims don’t have to click anything.

**Vulnerability**: DOM XSS with jQuery in location.hash

# **Reflected XSS into attribute with angle brackets HTML-encoded**

**LAB**

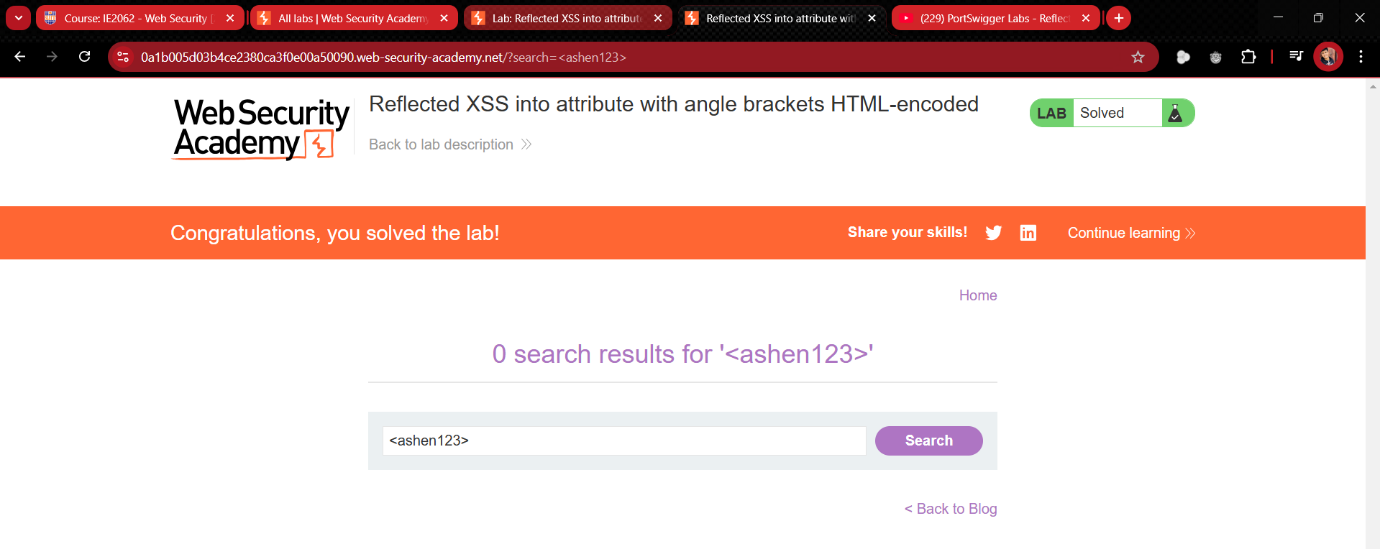
This lab contains a reflected cross-site scripting vulnerability in the search blog functionality where angle brackets are HTML-encoded. To solve this lab, perform a cross-site scripting attack that injects an attribute and calls the alert function.

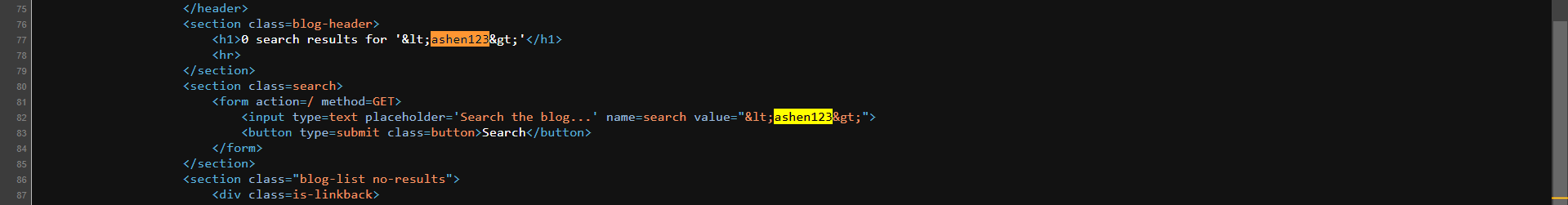
**HINT**

Just because you're able to trigger the alert() yourself doesn't mean that this will work on the victim. You may need to try injecting your proof-of-concept payload with a variety of different attributes before you find one that successfully executes in the victim's browser.

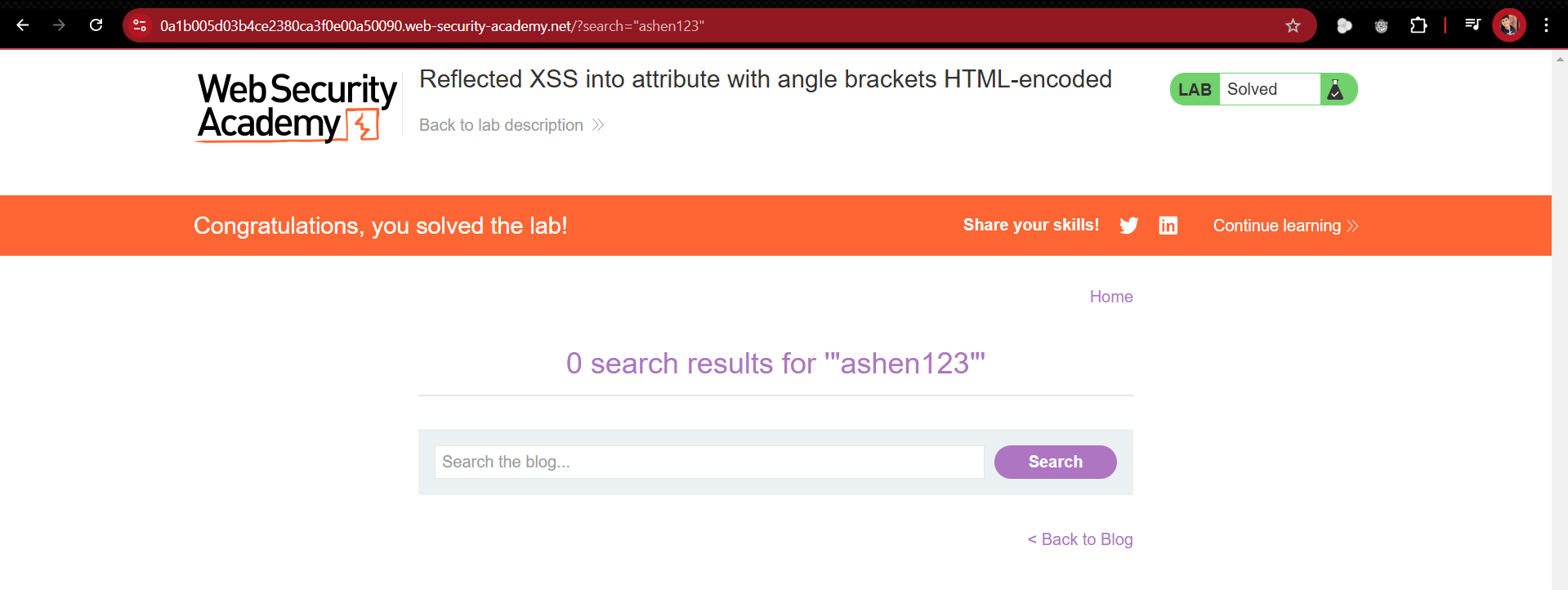
**SOLUTION**

**STEP 01:** Identify the input field and test with a random string with angle brackets.

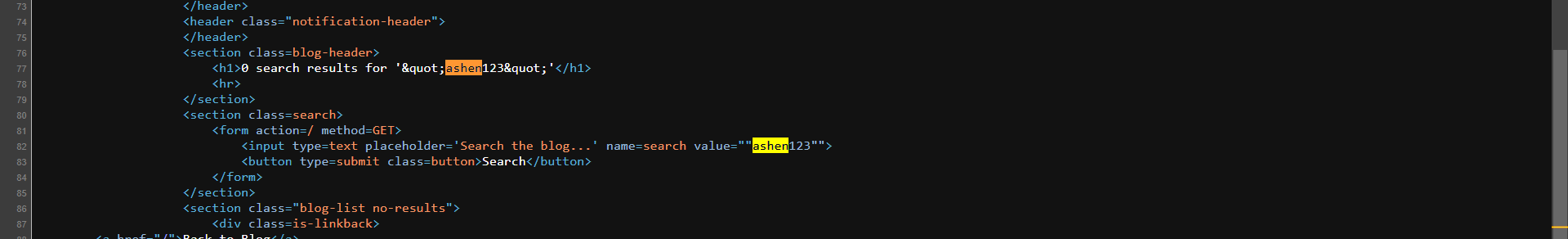
Open the Source page by clicking ctrl + u. Check for the entered random string. We can see that the angle brackets are encoded.



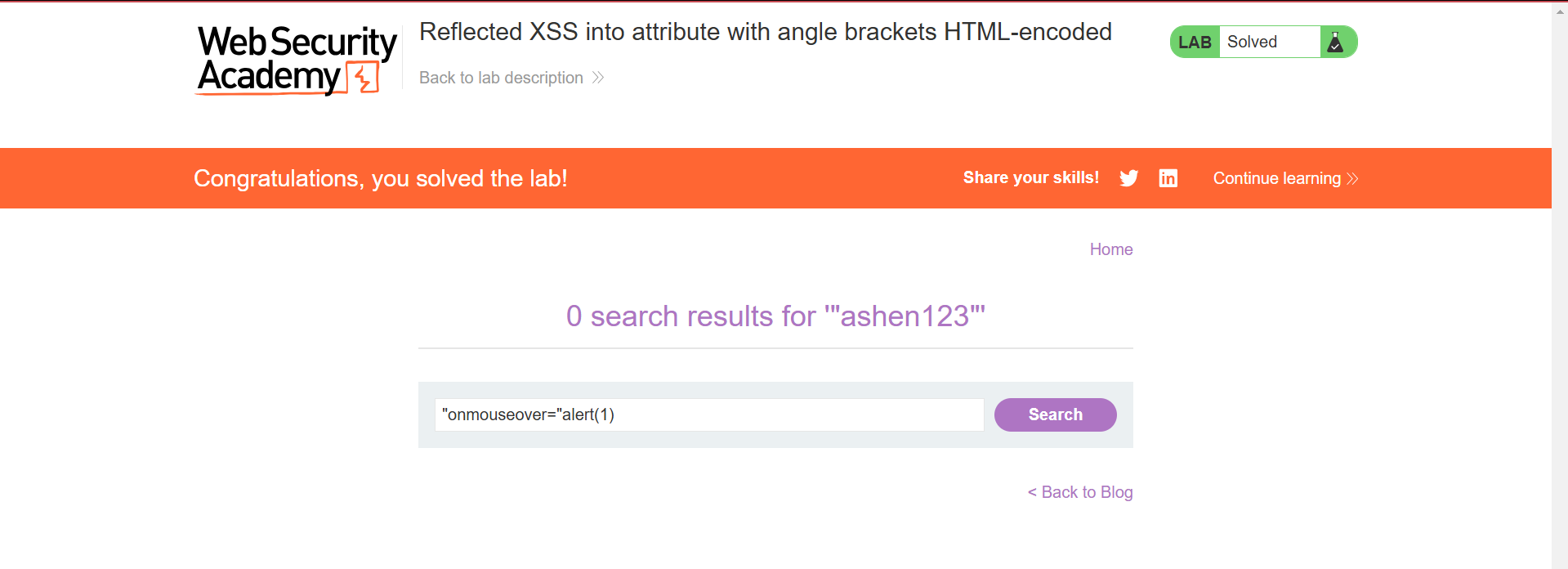
**STEP 02:** Test again entering a random string with double quotation marks “”.

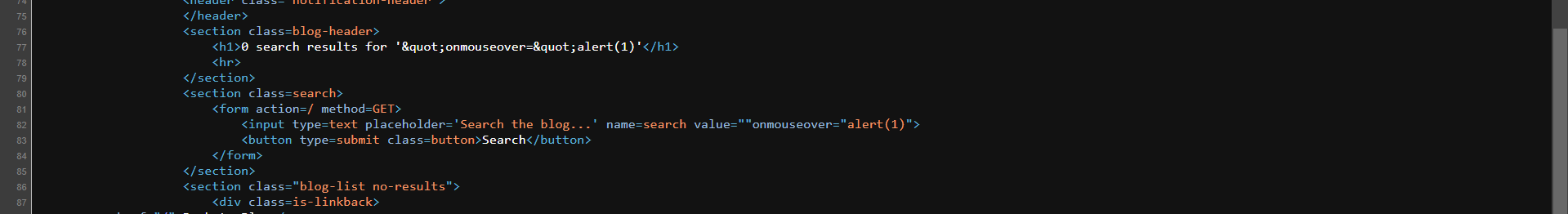


Open the Source page by clicking ctrl + u. Check for the entered random string. We can see that the double quotation marks are not encoded.

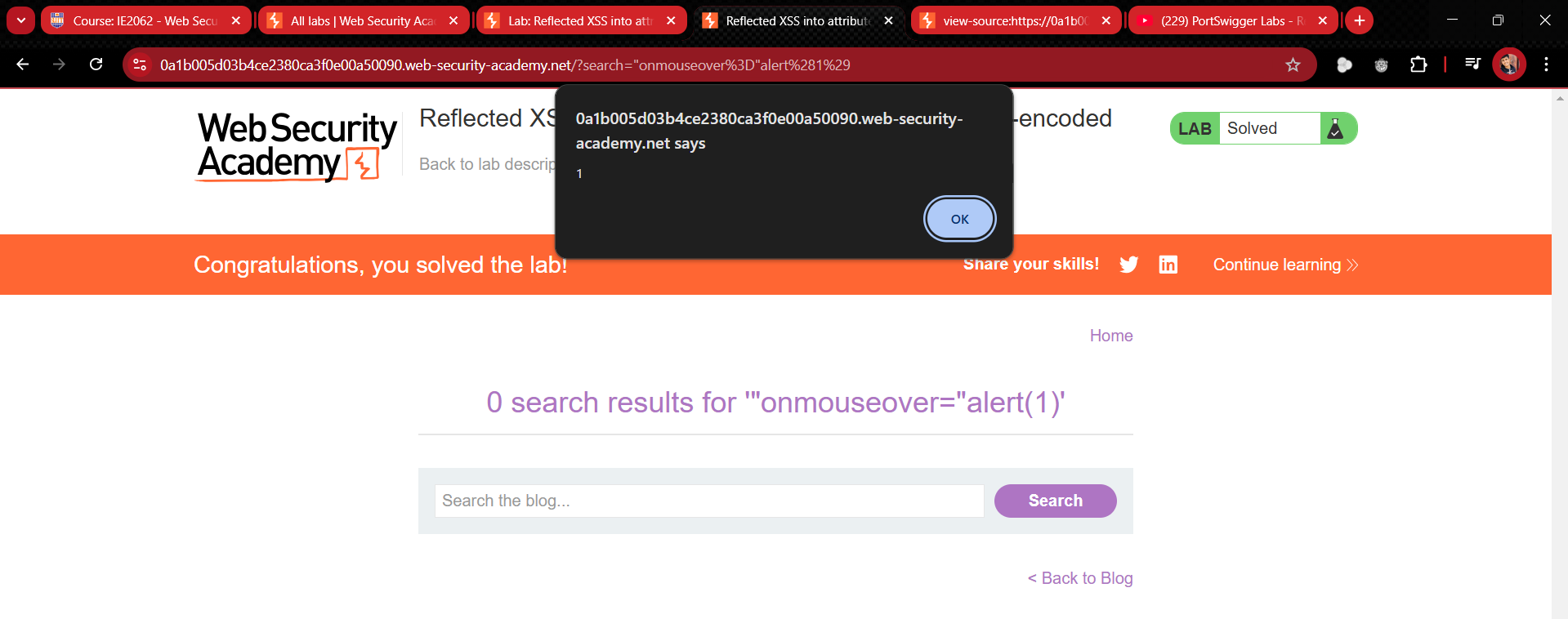


**STEP 03:** Insert the malicious payload. Here we used onmouseover attribute so the alert will pop up even the mouse goes through the search bar.





If the page executes this payload and shows an alert box with "1", the site is vulnerable.



**EXPLANATION**

**Reflected XSS vulnerability** where **angle brackets (< >) are encoded**, but other special characters are not. Inject a **malicious payload** (event handlers are not blocked)into an HTML attribute and execute JavaScript.

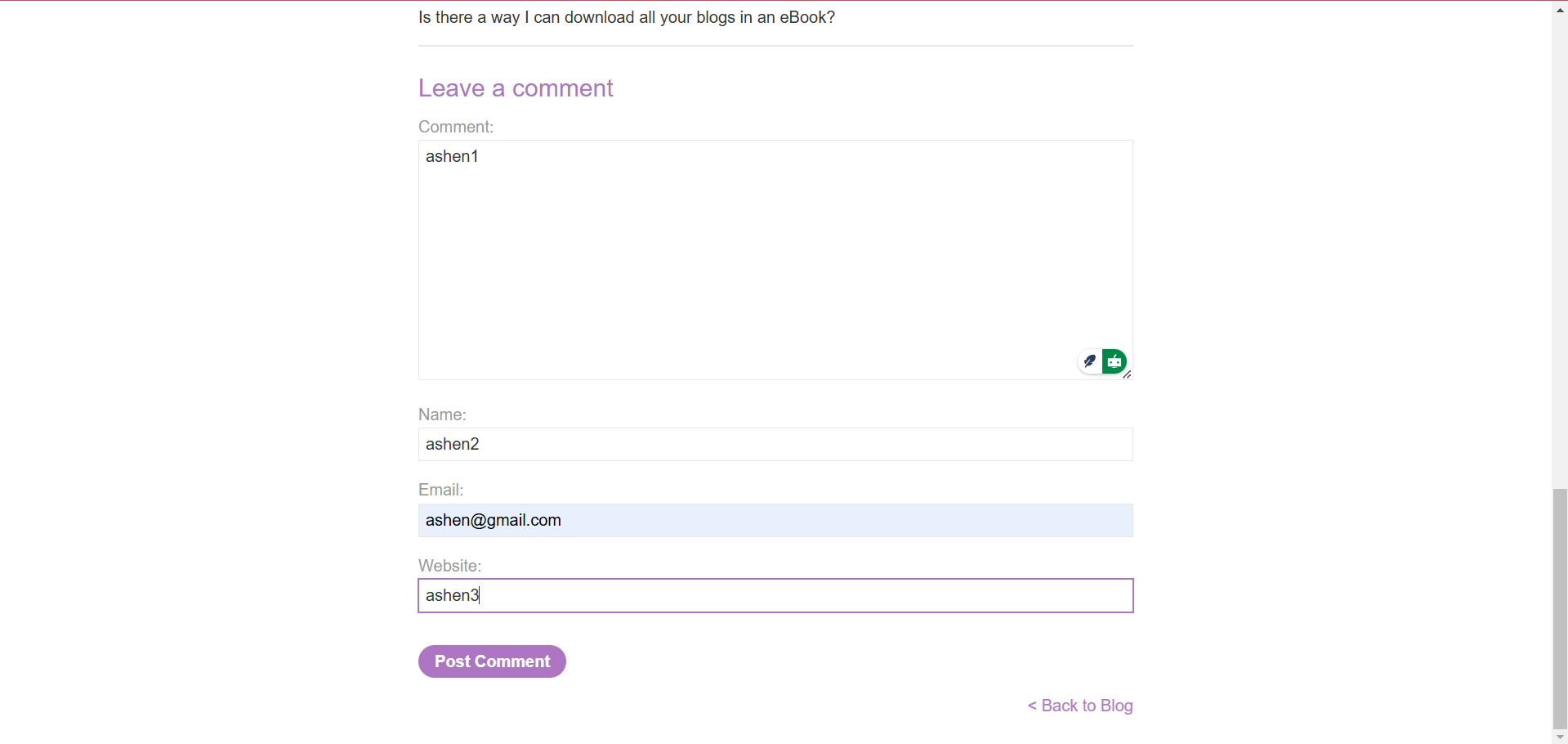
# **Stored XSS into anchor href attribute with double quotes HTML-encoded**

**LAB**

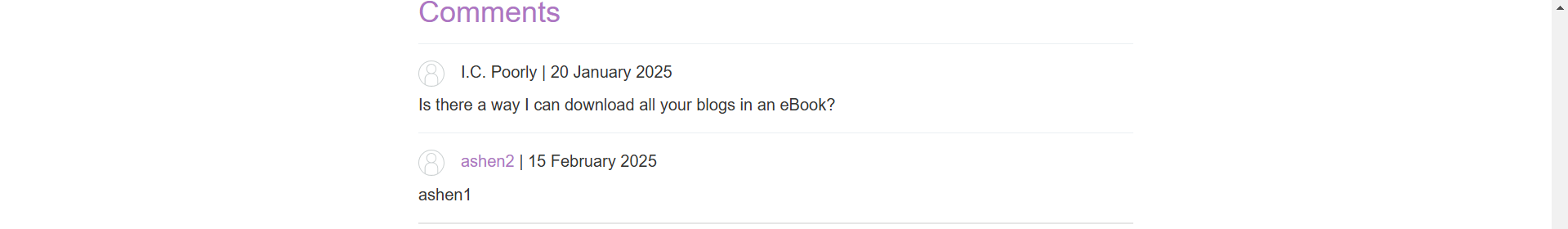
This lab contains a stored cross-site scripting vulnerability in the comment functionality. To solve this lab, submit a comment that calls the alert function when the comment author name is clicked.

**SOLUTION**

**STEP 02:**Open the lab, open a post and scroll down to a comment page where we can find an input field.



**STEP 02:** Insert some test random strings to check how it behaves.

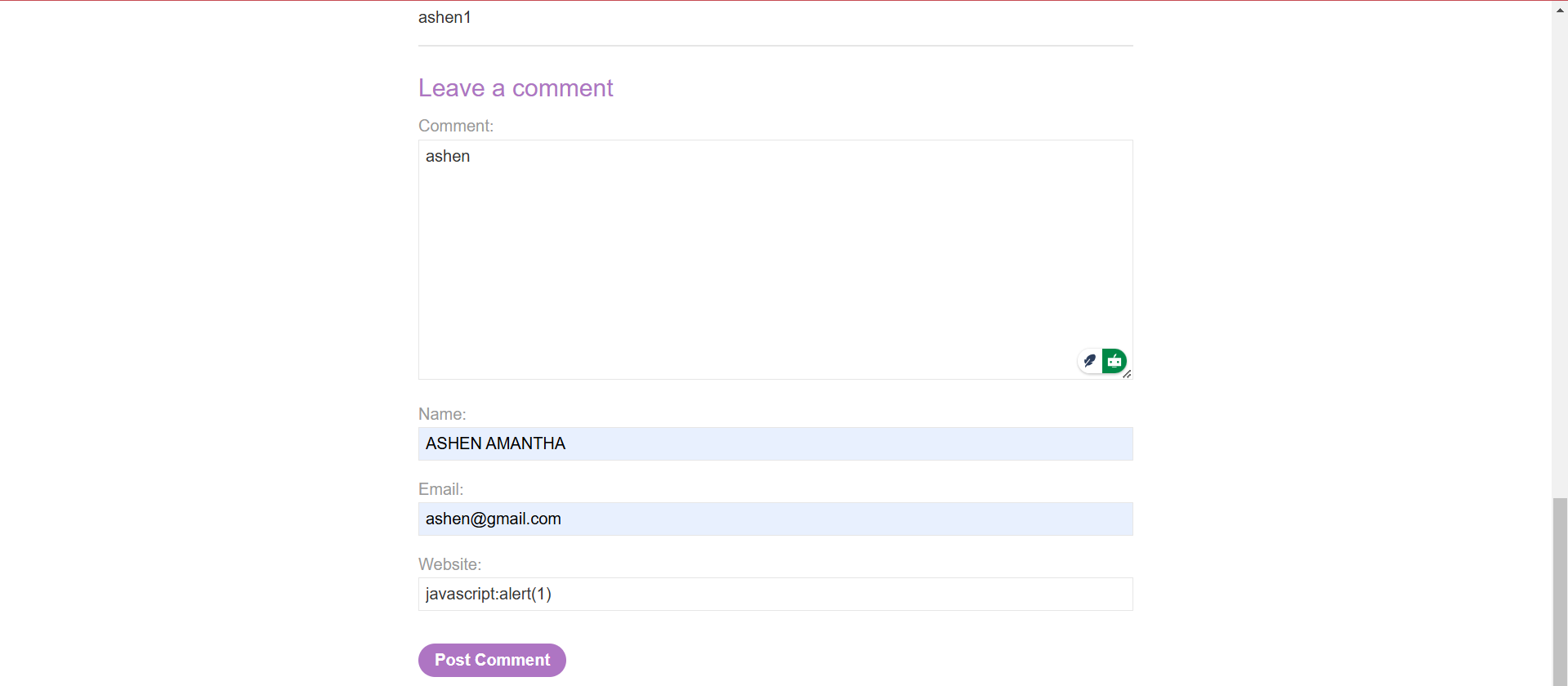


After submitting open the page source and search for the inserted random strings.



As you can see “ashen3” is inside anchor tag, so we can use it as the vulnerability to exploit.

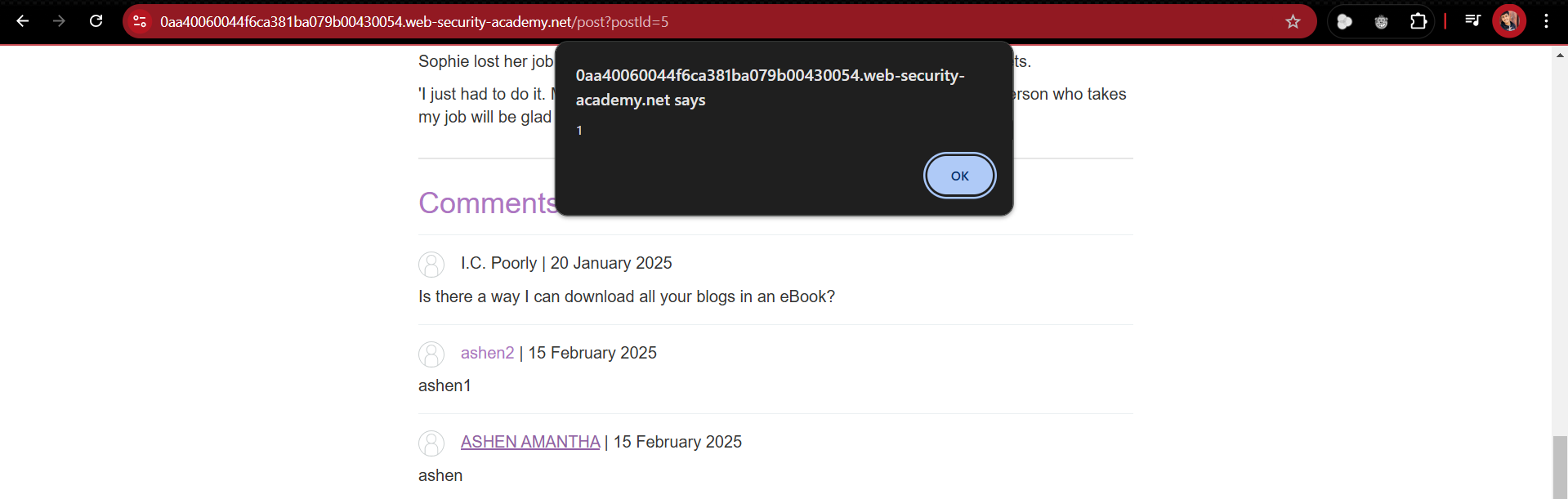
**STEP 03:** Inject the payload to the website field.



Confirm by checking it in the source page.



When the victim click on the name, the script will execute and popup an alert.



**EXPLANATION**

**Vulnerability**: User input is stored inside an **anchor tag’s href**, but **double quotes are encoded**, allowing javascript: injection.

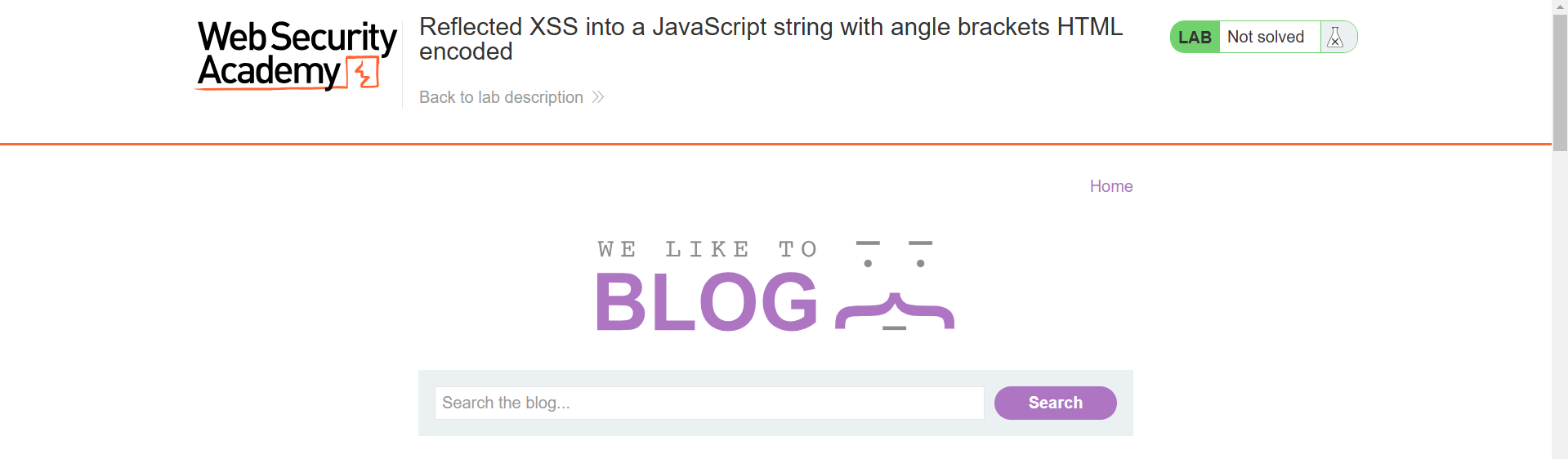
# **Reflected XSS into a JavaScript string with angle brackets HTML encoded**

**LAB**

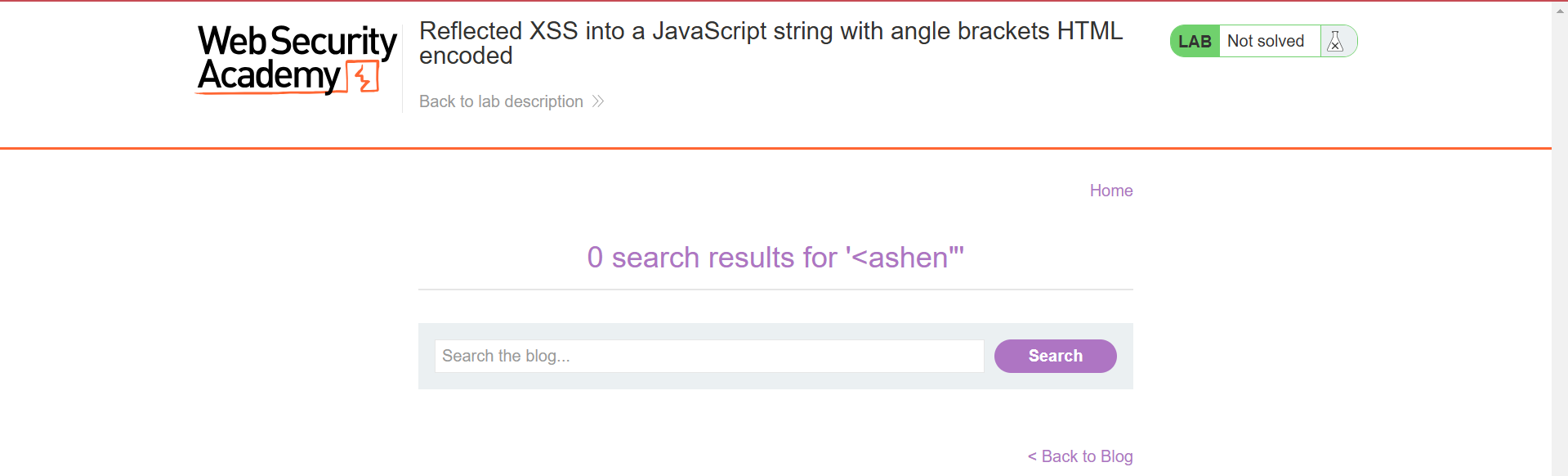
This lab contains a reflected cross-site scripting vulnerability in the search query tracking functionality where angle brackets are encoded. The reflection occurs inside a JavaScript string. To solve this lab, perform a cross-site scripting attack that breaks out of the JavaScript string and calls the alert function.

**SOLUTION**

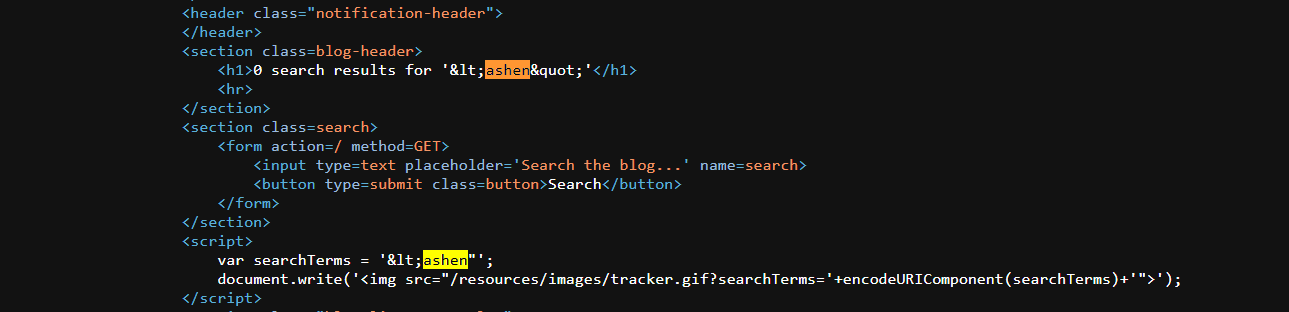
**STEP 01:** Open the Lab and Identify the input field.



**STEP 02:** Test a basic input such as a random string, here the angle brackets are encoded.



Search the random string in the page source.



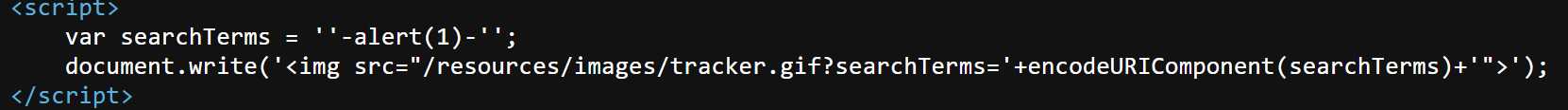
**STEP 03:** As the angle brackets are encoded but single quotes are not encoded. So we can use single quotes to break out the string.

**'-alert(1)-'**

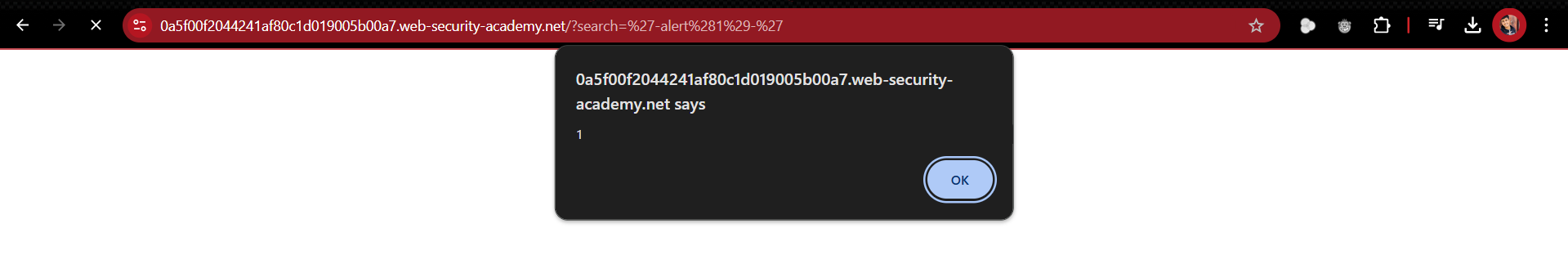
**'** closes the existing string.

**-alert(1)-** adds JavaScript code.

**'** prevents syntax errors by restoring string format.

****

If the page executes this script and shows an alert box with "1", the site is vulnerable.



**EXPLANATION**

**Vulnerability:** User input is reflected inside a JavaScript string, with angle brackets encoded but quotes unprotected.