BSc (Hons) in Information Technology
Specializing in Cyber Security

Sri Lanka Institute of Information Technology

Web Security (IE2062)

Practical -01

MOHAMMED RUHAIM

IT23256446

Cross-site scripting

Cross-site scripting (XSS) is a web security vulnerability that allows attackers to inject malicious code (scripts) into a trusted website, which then gets executed by the user's browser, potentially stealing sensitive information or performing unauthorized actions on their behalf

How does XSS work?

Cross-Site Scripting (XSS) works by exploiting a vulnerability in a website that allows an attacker to inject malicious JavaScript code into a web page, which then executes in the victim's browser when they visit the site, enabling the attacker to steal sensitive information like cookies, session tokens, or even manipulate the website's content depending on the injected script; essentially, the attacker uses the victim's browser as a vehicle to run their malicious code on a trusted website without their knowledge.

Key points about XSS:

- **User Input:**

Most XSS attacks occur when a website allows user input without properly sanitizing it, meaning the attacker can embed malicious JavaScript code within the input that gets displayed on the page and executed by the victim's browser.

- **Injection Methods:**

Attackers can inject malicious code through various methods like links, form submissions, comments, or even through specially crafted HTTP headers.

- **Types of XSS:**
    - **Stored XSS:** When the malicious script is stored on the server-side, like in a database, and executed whenever a user accesses the page containing that stored data.
    - **Reflected XSS:** When the malicious script is reflected back to the user in the response of a web request, often through a URL parameter.
    - **DOM-Based XSS:** When the malicious script manipulates the Document Object Model (DOM) on the client-side, allowing for more complex attacks.

View all XSS labs

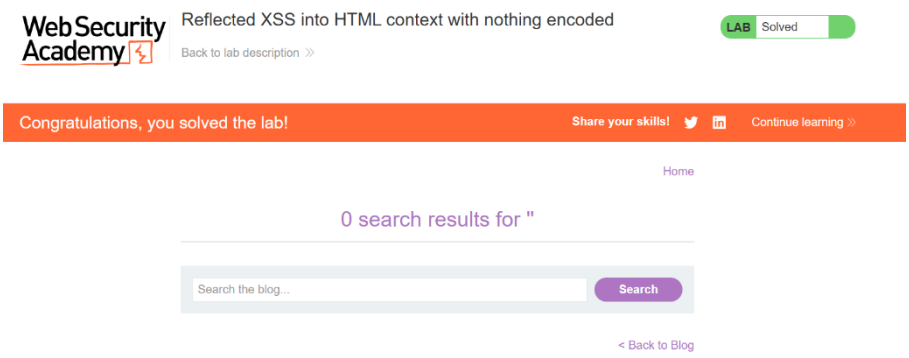
Lab 01: Reflected XSS into HTML context with nothing encoded

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.

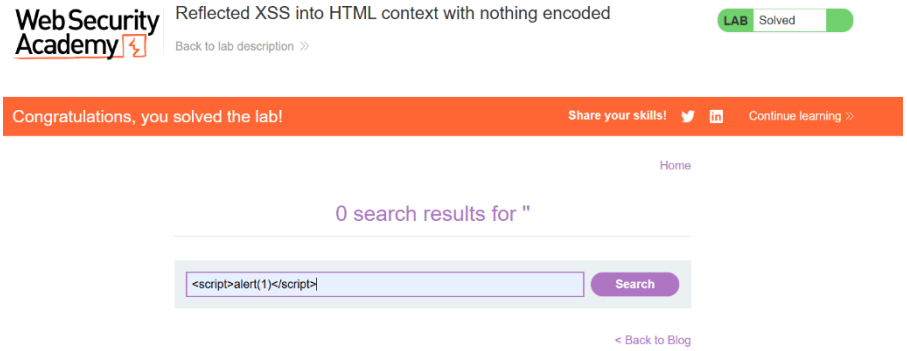
Steps to solve the problem

Type this script in search bar " <script>alert(1)</script>" and press the search button. When you press the search button, you got the alert message

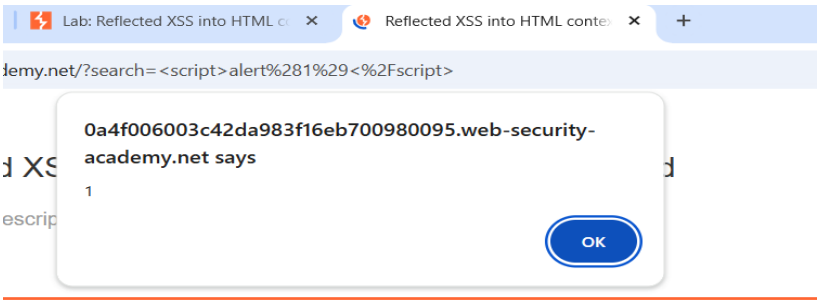
Step 01:



step 02:



step03:



Lab 02: Stored XSS into HTML context with nothing encoded

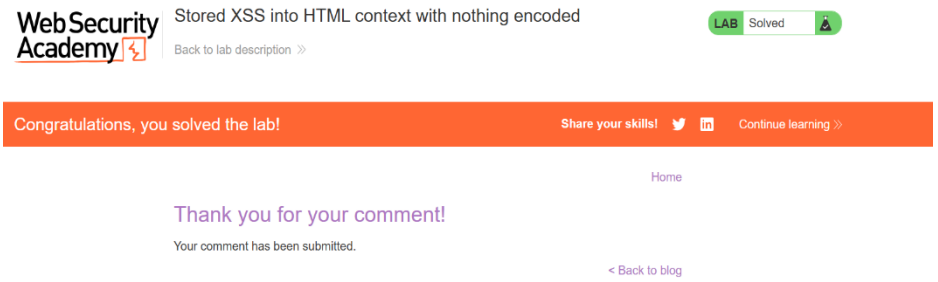
This lab contains a stored cross-site scripting vulnerability in the comment functionality. To solve this lab, submit a comment that calls the alert function when the blog post is viewed.

Steps to solve the problem

1. Enter the following into the comment box: <script>alert(1)</script>
2. Enter a name, email and website.
3. Click "Post comment".
4. Go back to the blog.

Step1:

Step2:



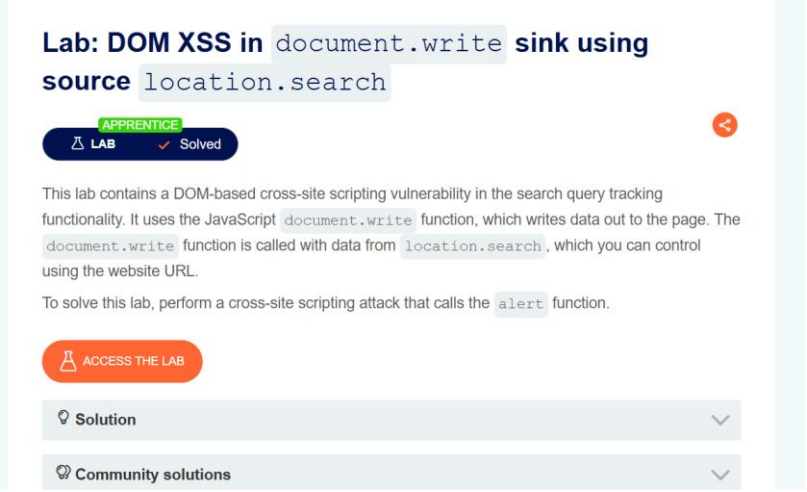
Lab 03: DOM XSS in document.write sink using source location.search

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.

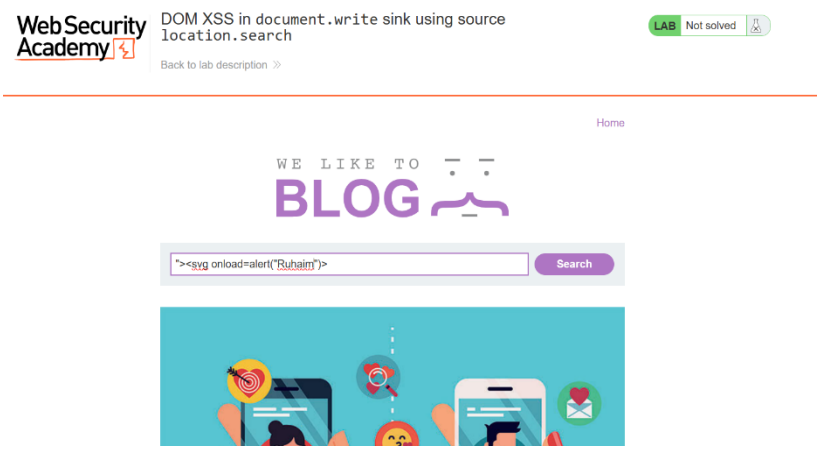
Steps to solve the problem

1. Enter a random alphanumeric string into the search box.
2. Right-click and inspect the element, and observe that your random string has been placed inside an img src attribute.
3. Break out of the img attribute by searching for:"><svg onload=alert(1)>

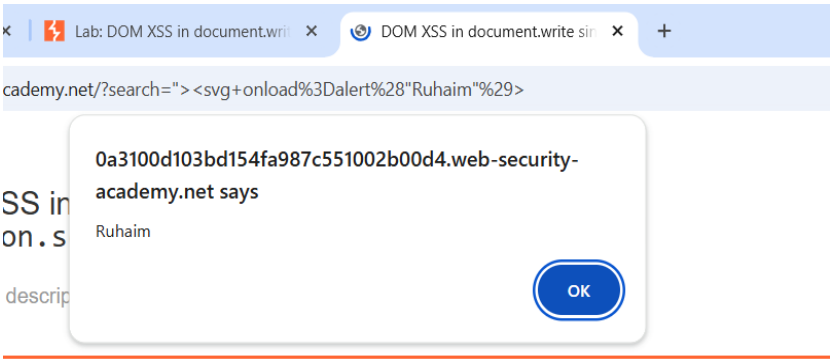
Step1



step2



step3:



Home

Lab04: DOM XSS in innerHTML sink using source location.search

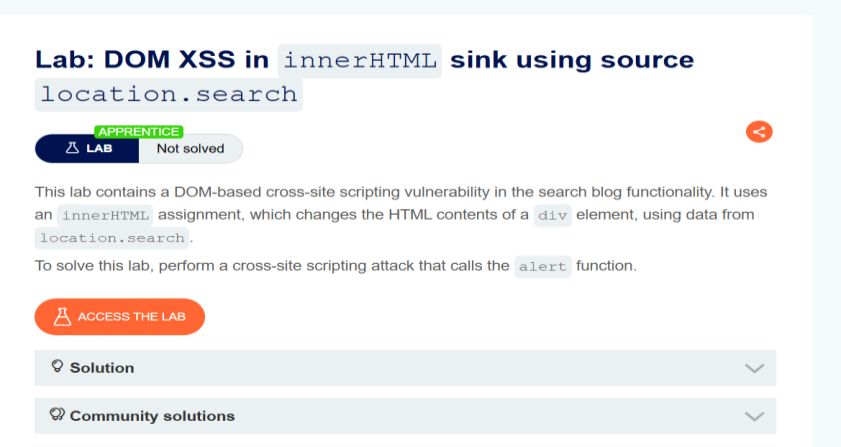
This lab contains a DOM-based cross-site scripting vulnerability in the search blog functionality. It uses an innerHTML assignment, which changes the HTML contents of a div element, using data from location.search. To solve this lab, perform a cross-site scripting attack that calls the alert function.

Steps to solve the problem

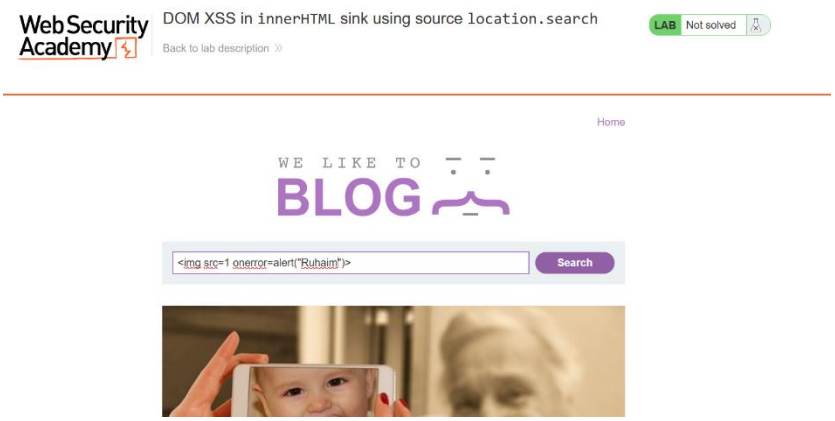
1. Enter the following into the into the search box: <img src=1 onerror=alert(1)>
2. Click "Search".

The value of the src attribute is invalid and throws an error. This triggers the onerror event handler, which then calls the alert() function. As a result, the payload is executed whenever the user's browser attempts to load the page containing your malicious post.

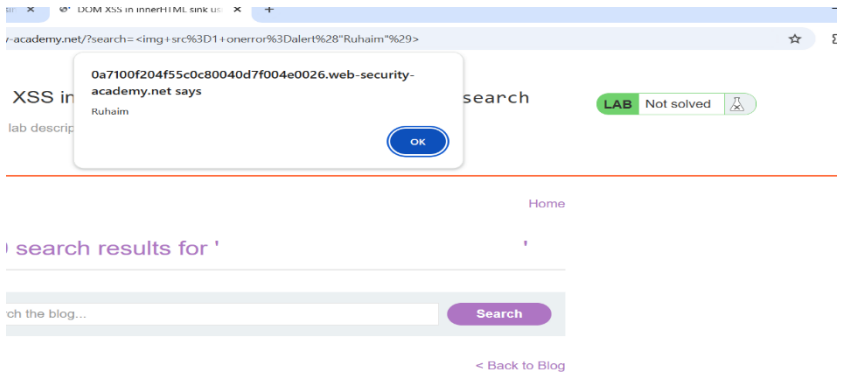
Step1:



step2:



step3:



Lab05:DOM XSS in jQuery anchor href attribute sink using location.search source

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.

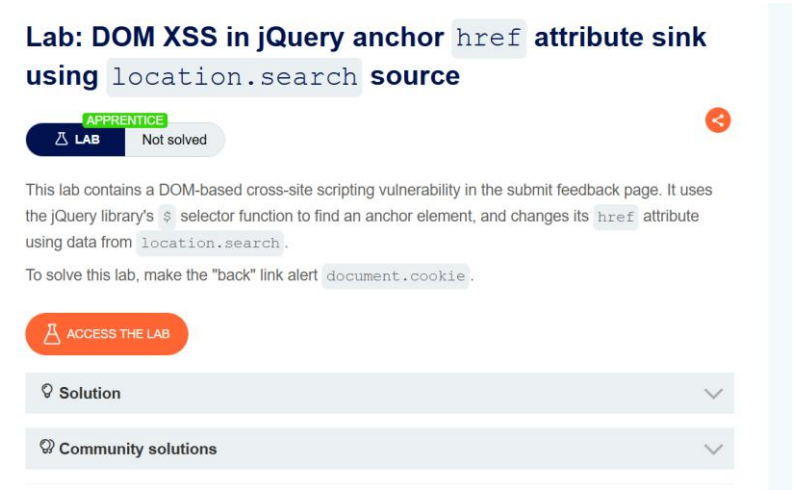
Steps to solve the problem

On the Submit feedback page, change the query parameter returnPath to / followed by a random alphanumeric string.

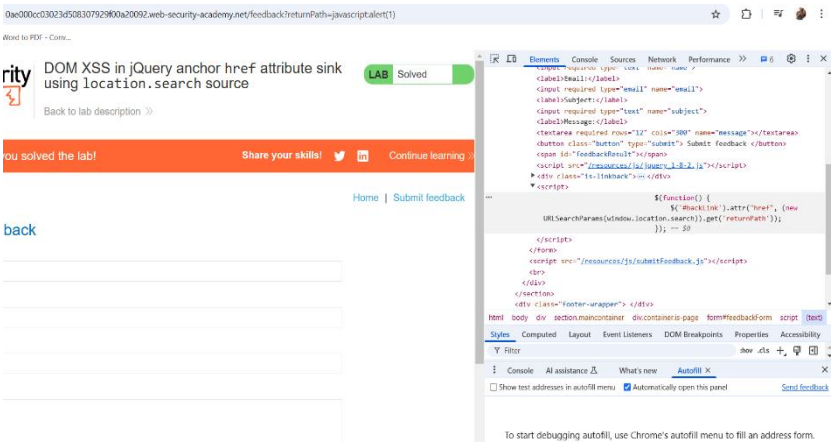
Right-click and inspect the element, and observe that your random string has been placed inside an a href attribute.

Change returnPath to: javascript:alert(document.cookie) Hit enter and click "back".

Step1:



step2:



Lab06: DOM XSS in jQuery selector sink using a hashchange event

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.

Steps to solve the problem

Notice the vulnerable code on the home page using Burp or the browser's DevTools.

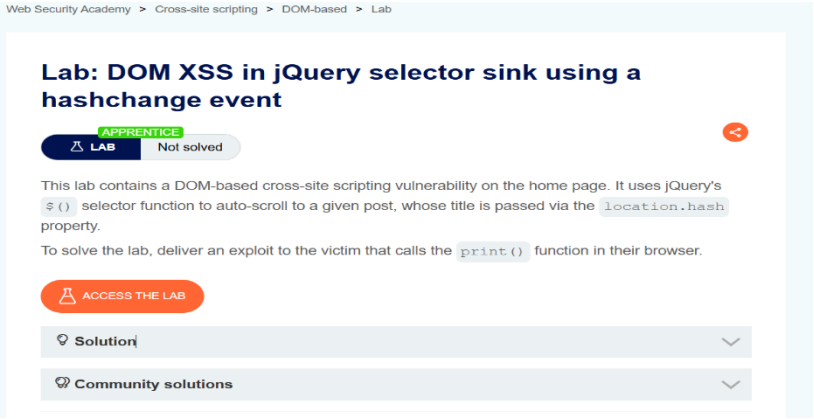
From the lab banner, open the exploit server. In the Body section, add the following malicious iframe

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

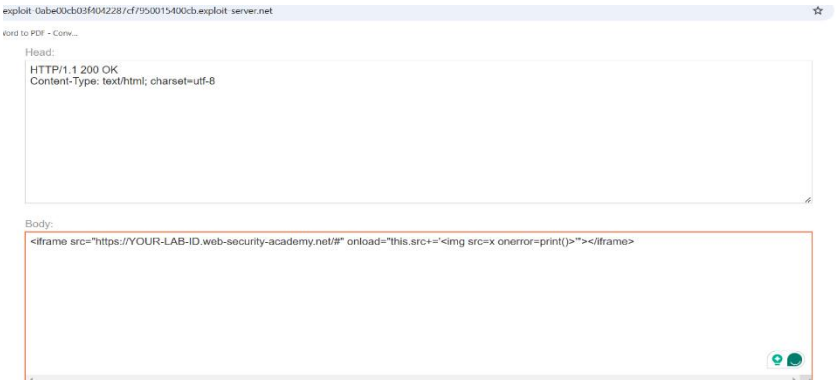
Store the exploit, then click View exploit to confirm that the print() function is called.

Go back to the exploit server and click Deliver to victim to solve the lab.

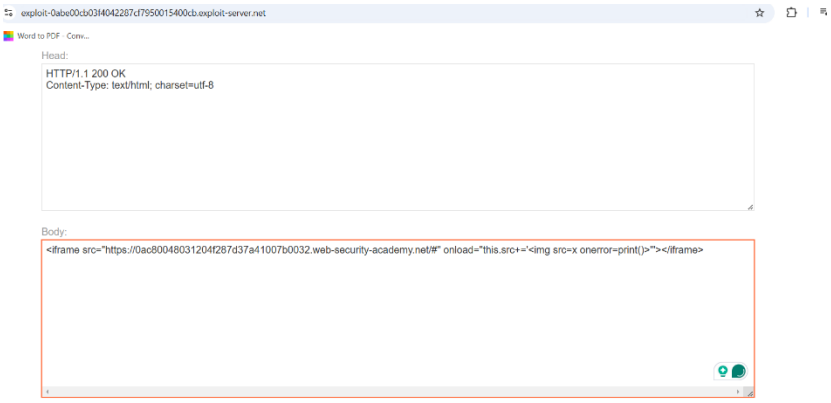
Step1:



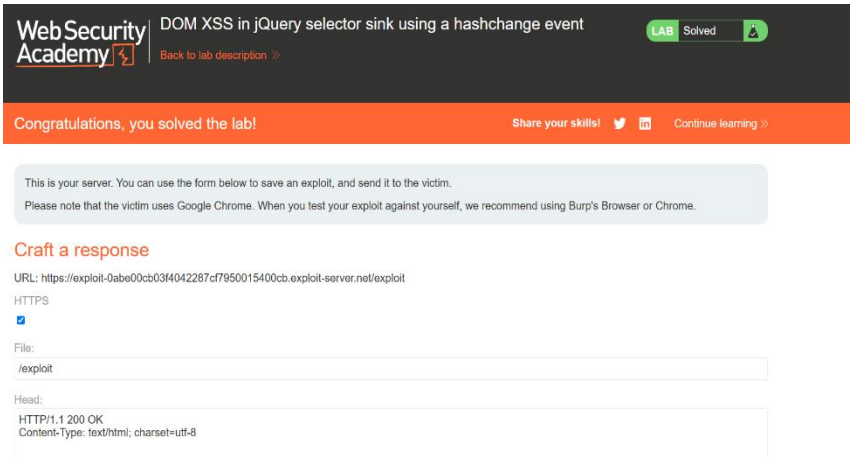
Step2:



step3:



step4:



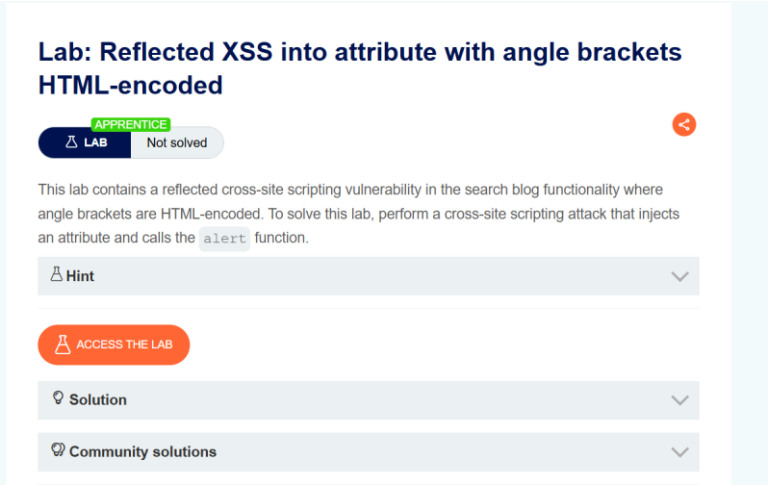
Lab07: Reflected XSS into attribute with angle brackets HTML-encoded

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.

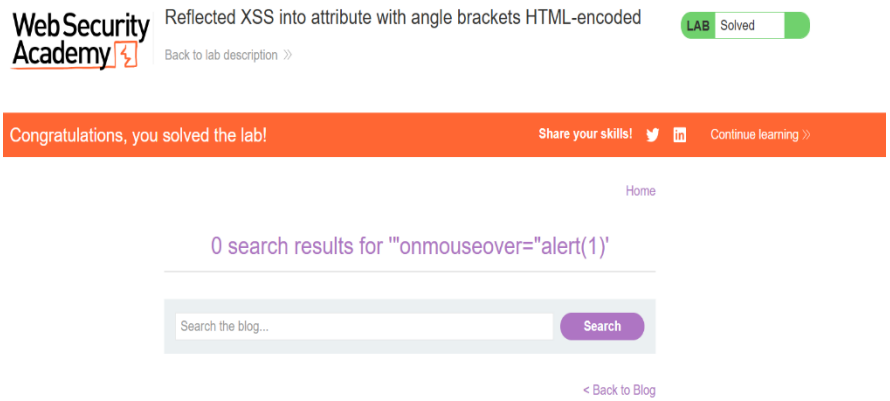
Steps to solve the problem

1. Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
2. Observe that the random string has been reflected inside a quoted attribute.
3. Replace your input with the following payload to escape the quoted attribute and inject an event handler:"onmouseover="alert(1)
4. Verify the technique worked by right-clicking, selecting "Copy URL", and pasting the URL in the browser. When you move the mouse over the injected element it should trigger an alert.

Step1:



step2:



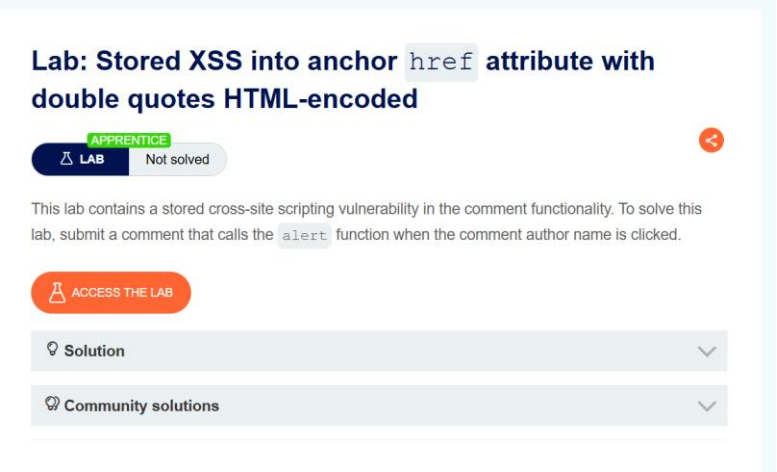
Lab08: Stored XSS into anchor href attribute with double quotes HTML-encoded

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.

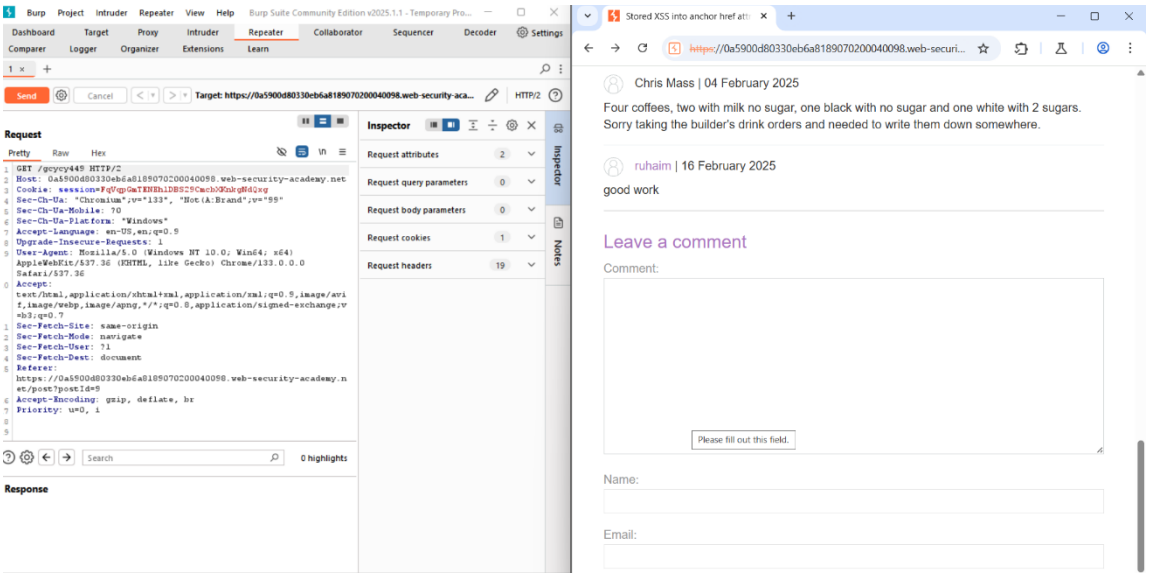
Steps to solve the problem

1. Post a comment with a random alphanumeric string in the "Website" input, then use Burp Suite to intercept the request and send it to Burp Repeater.
2. Make a second request in the browser to view the post and use Burp Suite to intercept the request and send it to Burp Repeater.
3. Observe that the random string in the second Repeater tab has been reflected inside an anchor href attribute.
4. Repeat the process again but this time replace your input with the following payload to inject a JavaScript URL that calls alert:javascript:alert(1)
5. Verify the technique worked by right-clicking, selecting "Copy URL", and pasting the URL in the browser. Clicking the name above your comment should trigger an alert.

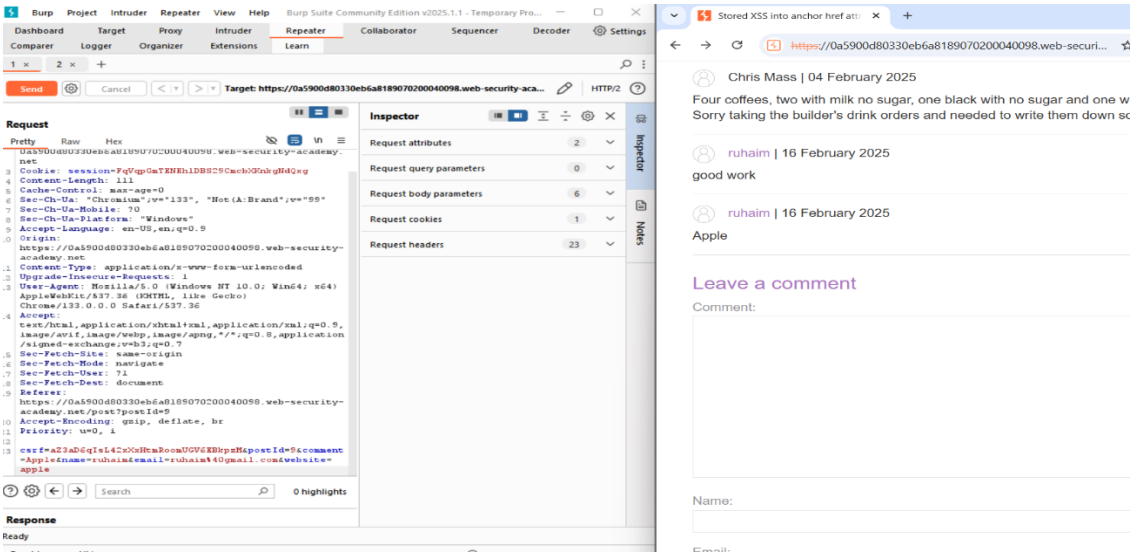
Step1:



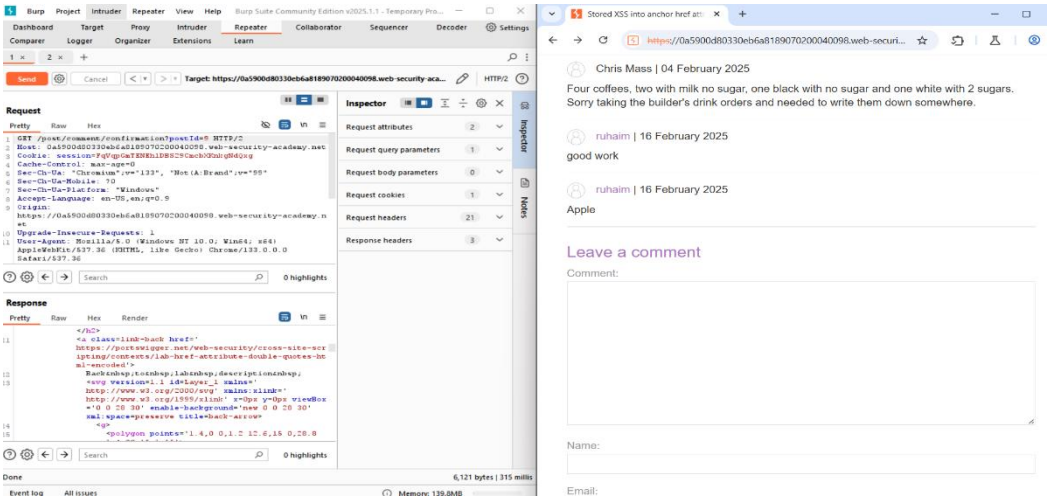
step2:



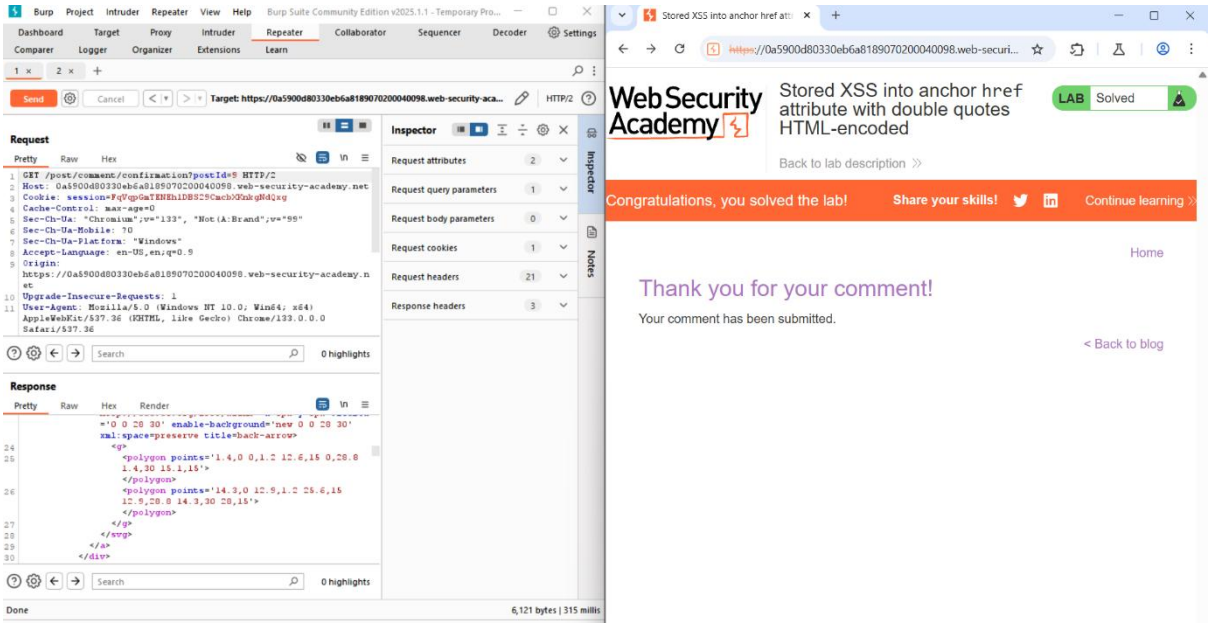
step3:



Step4:



step5:



Lab09: Reflected XSS into a JavaScript string with angle brackets HTML encoded

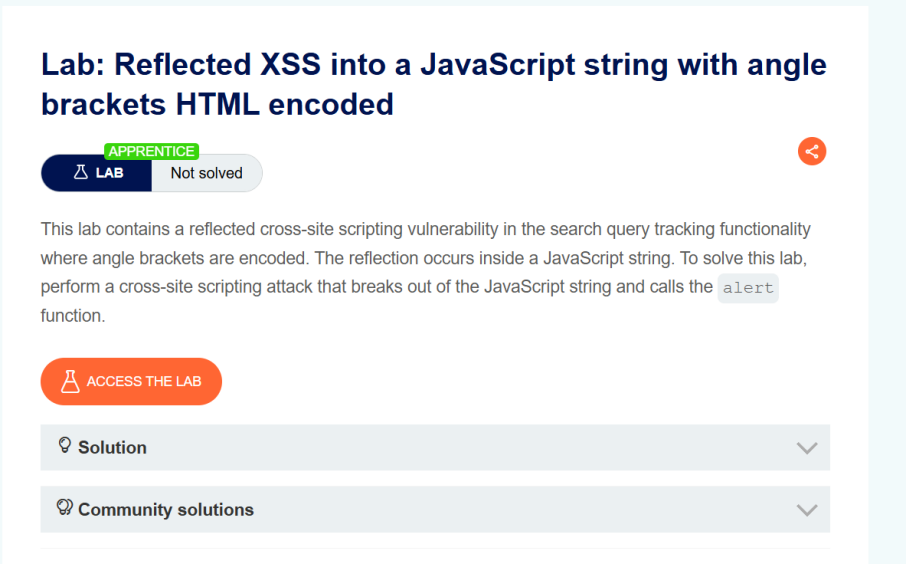
This lab contains a reflected cross-site scripting vulnerability in the search query tracking functionality where angle brackets are encoded. The reflection occurs inside a JavaScript string. To solve this lab, perform a cross-site scripting attack that breaks out of the JavaScript string and calls the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

Steps to solve the problem

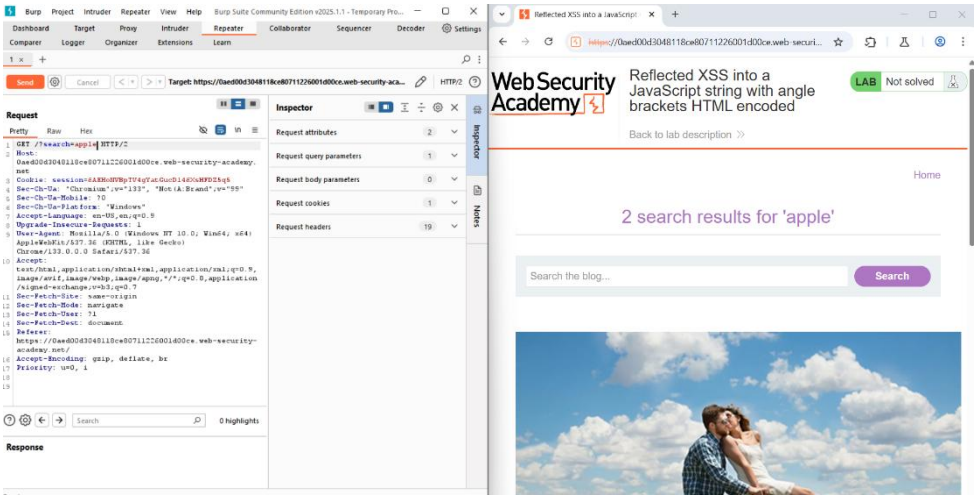
1. Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
2. Observe that the random string has been reflected inside a JavaScript string.
3. Replace your input with the following payload to break out of the JavaScript string and inject an alert:'-alert(1)-'

4. Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in the browser. When you load the page it should trigger an alert.

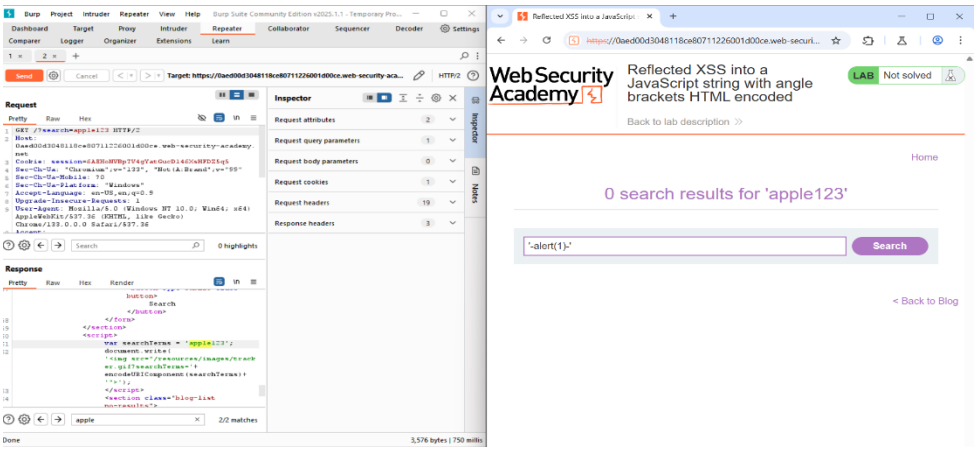
Step1:



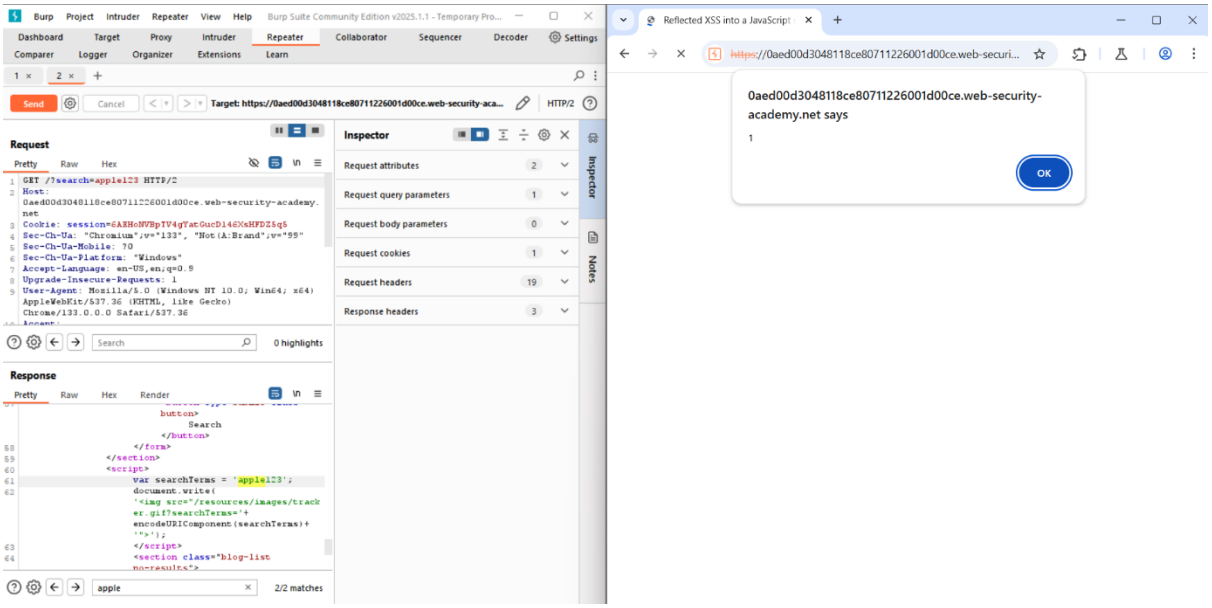
step2:



step3:



Step4:


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.1.1 - Temporary Pro...
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Settings
Comparer Logger Organizer Extensions Learn
Send Cancel
Target: https://0aed00d3048118ce80711226001d00ce.web-security-aca...
HTTP/2
Request
Pretty Raw Hex
GET /?search=apple123 HTTP/2
Host: 0aed00d3048118ce80711226001d00ce.web-security-academy.net
Cookie: session=6AEHoNVBpIV4gYatGucD14EXsHFDZ1q5
Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
Search
0 highlights
Response
Pretty Raw Hex Render
button>
Search
</button>
</form>
</section>
<script>
var searchTerms = 'apple123';
document.write('<img src="/resources/images/tracker.gif?searchTerms='+encodeURIComponent(searchTerms)+'">');
</script>
<section class="blog-list no-results">
apple
2/2 matches
Inspector
Request attributes 2
Request query parameters 1
Request body parameters 0
Request cookies 1
Request headers 19
Response headers 3
Inspector
Notes
Reflected XSS into a JavaScript
https://0aed00d3048118ce80711226001d00ce.web-securi...
0aed00d3048118ce80711226001d00ce.web-security-academy.net says
1
OK