

Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is a type of **web security vulnerability** that allows an attacker to inject **malicious scripts (usually JavaScript)** into content that is delivered to users by a trusted website. These scripts then run in the browsers of users who visit the infected page, leading to various security risks.

1. Lab: Reflected XSS into HTML context with nothing encoded

Vulnerability Context: Reflected XSS in HTML Context

In this lab:

- Reflected: The input from the URL (search parameter) is immediately reflected into the HTML response page.
- HTML context with nothing encoded: The input is directly inserted into the body of the HTML without escaping special characters, such as `<` and `>`.

This means that if the attacker includes a `<script>` tag in the search parameter, the browser will interpret it as an actual script and execute it.

Steps to Solve the Lab

- a. Identify the vulnerable input field

Visit the lab page and look for the search box or search URL.

Perform a search like: `?search=test`

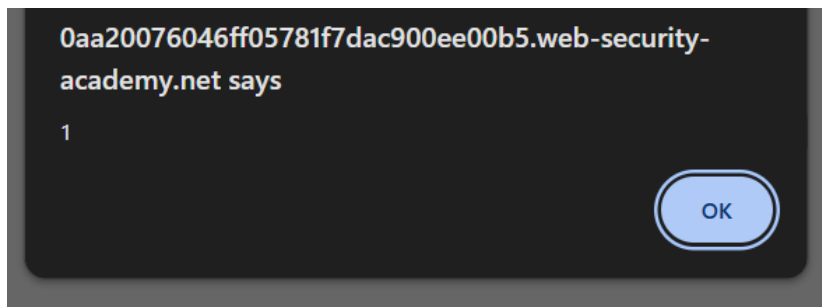
Inspect how the input is reflected on the page

- b. Craft the XSS Payload

To confirm the vulnerability, inject a simple payload like: `<script>alert(1)</script>`

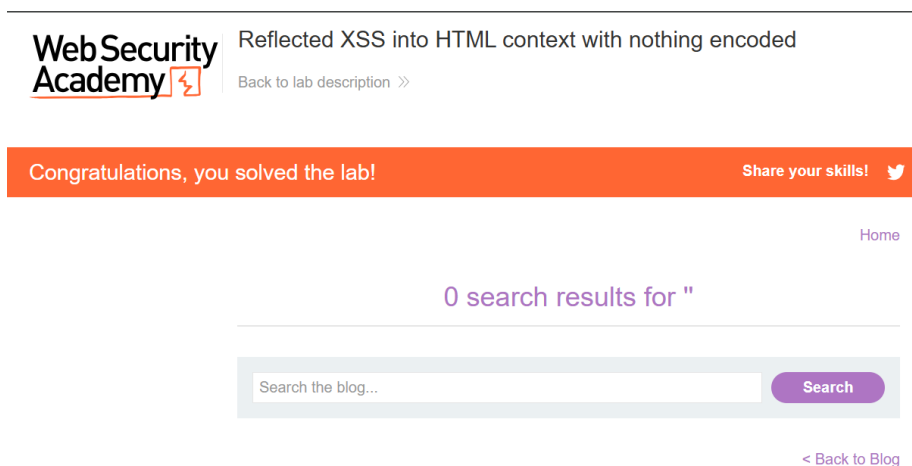


If the page renders this without encoding the characters the browser will execute the JavaScript, and a popup alert with 1 will appear.



c. Verify Successful Exploitation

When the browser displays a JavaScript alert box with the value 1, it confirms that the script has executed successfully. This satisfies the lab requirement of demonstrating the XSS vulnerability.



2. Lab: Stored XSS into HTML context with nothing encoded

- Stored XSS, also known as persistent XSS, occurs when malicious input is stored on the server and later served to other users without being sanitized.
- In this lab, the attacker submits a comment that is later displayed as part of a blog post. If the application does not encode or sanitize HTML, any `<script>` code will be executed in the victim's browser.

Steps to Solve the Lab

- a. Visit the Blog Post Page
- b. Navigate to the blog post that allows users to submit comments.
- c. Look for the comment form



It's All in the Game - Football for Dummies

There are two types of people in the world; those who watch soccer, and those who watch people watching soccer. I fall into the latter category. If only they'd leave me in peace to drink my beer and zone out....

[View post](#)

- d. Craft a Malicious XSS Payload
- e. A typical stored XSS test payload is: `<script>alert(1)</script>`

Leave a comment

Comment:

`<script>alert(1)</script>`
Hello!!

Name:

Emma

Email:

emma11@gmail.com

Website:

[Post Comment](#)[< Back to Blog](#)

- f. Submit the form.

WebSecurity Academy

Stored XSS into HTML context with nothing encoded

[Back to lab description >>](#)

LAB Solved

Congratulations, you solved the lab!

Share your skills! [Twitter](#) [LinkedIn](#) [Continue learning >>](#)

[Home](#)

Thank you for your comment!

Your comment has been submitted.

[< Back to blog](#)

3. Lab: DOM XSS in document.write sink using source_location.search

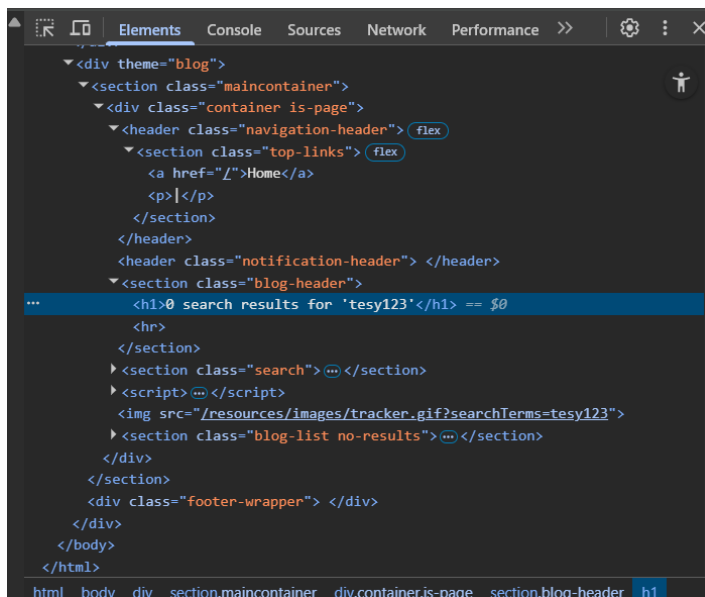
This lab demonstrates a DOM-based cross-site scripting (XSS) vulnerability caused by injecting unsanitized user input into the `src` attribute of an `` tag, using JavaScript and DOM manipulation.

Steps to Solve the Lab

- Type a random alphanumeric string in the search box

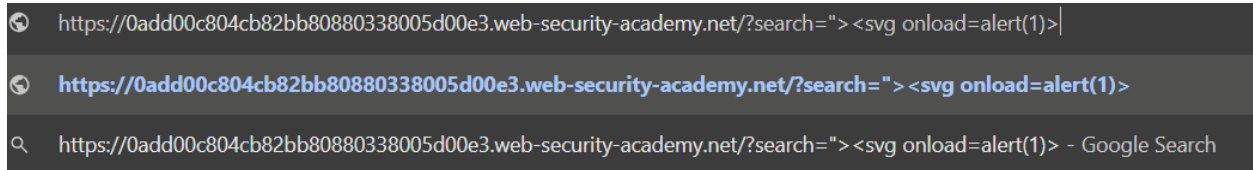


- Right-click on the page and select Inspect.
- Then I saw my input appears in an element like this: ``



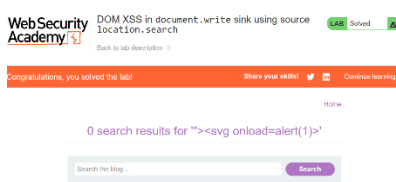
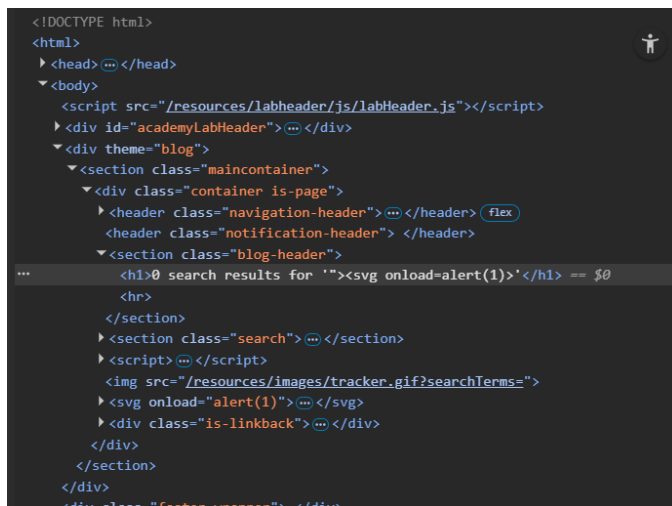
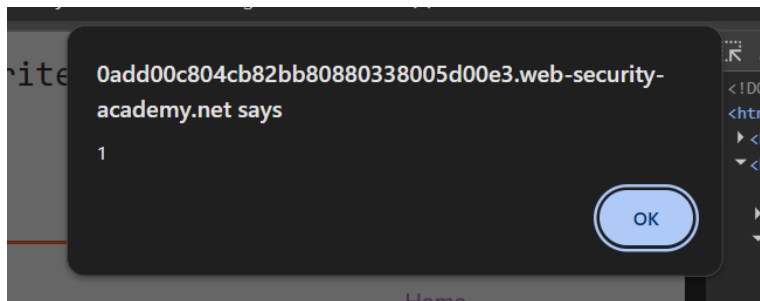
This confirms that my input is injected directly into the `src` attribute of an image tag without encoding.

I break out of the `src` attribute and inject a new tag with a script payload. : "><svg onload=alert(1)>



Once the payload is injected:

- The page renders: `<svg onload=alert(1)>`
- The onload event fires.
- An alert box appears, confirming successful exploitation.

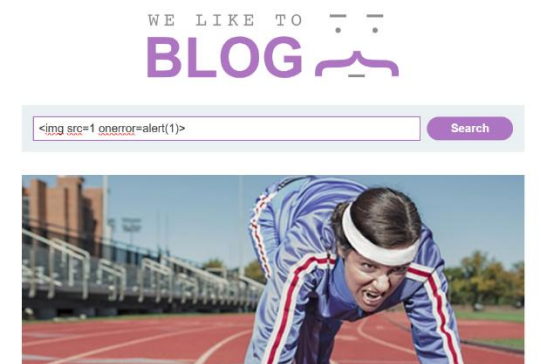


4. Lab: DOM XSS in innerHTML sink using source_location.search

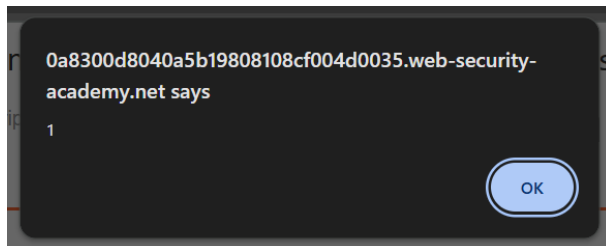
Exploit a DOM-based cross-site scripting vulnerability by injecting a payload into the search functionality, triggering the JavaScript alert() function.

Steps to Solve the Lab

1. Go to the lab site.
2. In the search box type : ``

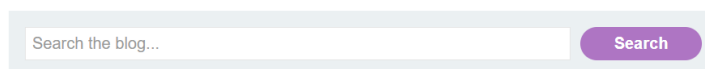


3. Click Search.
4. If successful, the page will show a popup with 1, confirming that payload was executed



[Home](#)

0 search results for '🚩'



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

- The application reads the value of the URL parameter called returnPath from location.search.
- It then uses jQuery to update the href attribute of a <a> tag on the page
- Because there is no sanitization or validation, any value passed to returnPath gets inserted directly into the href attribute.
- This creates an opportunity to inject JavaScript using a javascript: URI, which will execute when the link is clicked

Steps to Solve the Lab

1. Go to submit feedback page
2. Change the returnPath parameter
3. In the URL bar, change the value of returnPath to: javascript:alert(document.cookie)



0ae200d6044f2072b31c492700d9001f.web-security-academy.net/feedback?returnPath=javascript:alert(document.cookie)

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

LAB Solved

[Back to lab description >>](#)

4. Press Enter to load the modified URL
5. The page reloads with your new URL.
6. Click the "Back" link on the page
7. This link now has the injected href="javascript:alert(document.cookie)".
8. When you click it, the JavaScript will execute.

Congratulations, you solved the lab!

Share your skills!



Continue learning >

[Home](#) | [Submit feedback](#)

6. Lab: DOM XSS in jQuery selector sink using a hashchange event

- The lab uses jQuery's `$()` selector on `location.hash` input without sanitization.
- This causes a DOM-based XSS vulnerability triggered via hashchange.
- The goal is to exploit the vulnerability and call `print()` in the victim's browser.

Steps to Solve the Lab

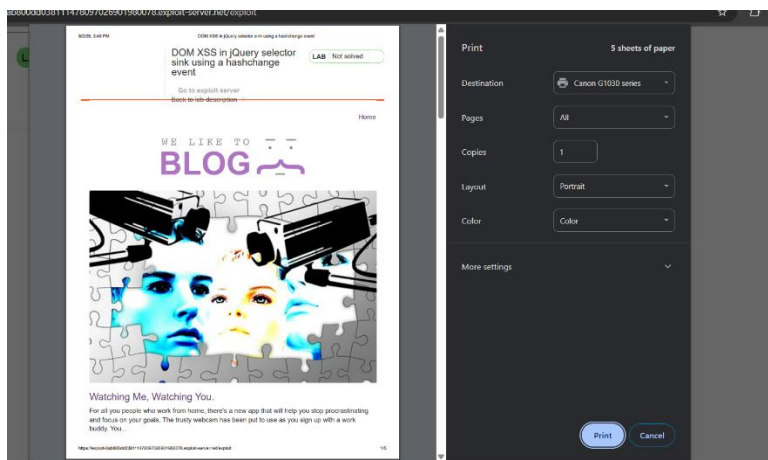
1. Open the Lab
2. Open the Exploit Server
3. Craft the Payload
4. In the “Body” field of the exploit server. Type :

Body:

```
<iframe src="https://0aa4004603f711cb80ad038e004600a0.web-security-academy.net/##" onload="this.src+='%3Cimg src=x onerror=print()%3E'></iframe>
```



[Store](#) [View exploit](#) [Deliver exploit to victim](#) [Access log](#)

5. Store and Test the Exploit
6. Click “Store”.
7. Then click “View exploit”.
8. I saw `print()` function triggered in my browser



9. Deliver to Victim

Congratulations, you solved the lab!

Share your skills!   Continue learning >>

This is your server. You can use the form below to save an exploit, and send it to the victim.

Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Browser or Chrome.

Craft a response

URL: <https://exploit-0ab800dd038111478097026901980078.exploit-server.net/exploit>

HTTPS ☒

7. Lab: Reflected XSS into attribute with angle brackets HTML-encoded

Steps to Solve the Lab

Go to the search function in the lab.

Type any random string like: test123

Home

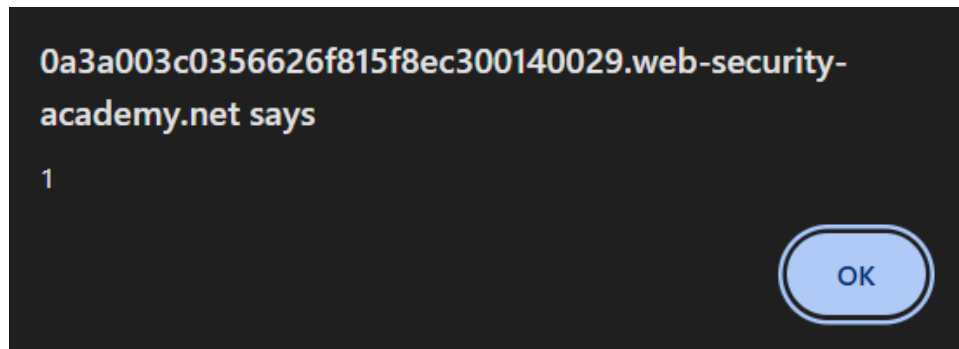
WE LIKE TO BLOG 

```
<section class="academyLabBanner"></section> (flex)
</div>
<div theme="blog">
  <section class="maincontainer">
    <div class="container is-page">
      <header class="navigation-header"> (flex)
        <section class="top-links"> (flex)
          <a href="/">Home</a>
        <p></p>
      </section>
    </header>
    <header class="notification-header"> </header>
    <section class="blog-header">
      <h1>0 search results for 'test123'</h1>
      <hr>
    </section>
    <section class="search">
      <form action="/" method="GET"> (flex)
        <input type="text" placeholder="Search the blog..." name="search"
          value="test123">
        <button type="submit" class="button">Search</button>
      </form>
```

Intercept the request using Burp Suite.

I saw something like: `<input value="test123">`

Replace input with this payload: `" onmouseover="alert(1)`



Congratulations, you solved the lab!

Share your skills!



Continue learning >>

[Home](#)

0 search results for "'onmouseover="alert(1)'

[< Back to Blog >>](#)

8. Lab: Stored XSS into anchor `href` attribute with double quotes HTML-encoded

Steps to Solve the Lab

- Post a comment on the target blog/forum with a random string in the "Website" input field.
- Use Burp Suite to intercept the HTTP request when you post the comment.

Leave a comment

Comment:

Hello!!!!

Name:

Emma

Email:

emma11@gmail.com

Website:

web123

Post Comment

- Send the intercepted request to Burp Repeater.
- Make a second request in the browser to view the post
- Use Burp Suite again to intercept the page request showing the comment, and send that response to another Burp Repeater tab.
- In the second Repeater tab, locate the reflected random string inside the href attribute.
- modify the comment submission in Burp Repeater by replacing the "Website" field value with: javascript:alert(1)
- Resend the modified comment submission request.
- View the post again in the browser.
- Right-click the author's name and paste that URL into the browser address bar.
- Clicking the author name trigger an alert popup.

Congratulations, you solved the lab!

Share your skills! [Twitter](#) [LinkedIn](#) [Continue learning >](#)

[Home](#)

Thank you for your comment!

Your comment has been submitted.

[< Back to blog](#)

9. Lab: Reflected XSS into a JavaScript string with angle brackets HTML encoded

Steps to Solve the Lab

Submit a random string in the search box

- View the page source

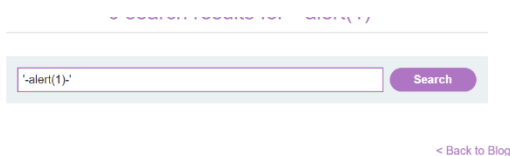
```
<section class="maincontainer">
  <div class="container is-page">
    <header class="navigation-header"> flex
    <section class="top-links"> flex
      <a href="/">Home</a>
      <p>|</p>
    </section>
  </header>
  <header class="notification-header"> </header>
  <section class="blog-header"> == $0
    <h1>0 search results for 'tts123'</h1>
    <hr>
  </section>
  <section class="search"> ... </section>
  <script> ... </script>
  
  <section class="blog-list no-results"> ... </section>
</div>
</section>
<div class="footer-wrapper"> </div>
</div>
</body>
```

- Search for test

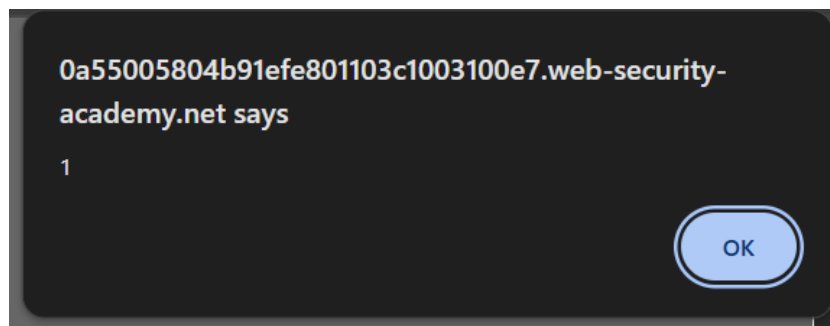
Construct the attack URL

Then inject JavaScript directly in the search parameter.

payload: '-alert(1)-'



This will break out of the string and run JavaScript.



[Home](#)

0 search results for "-alert(1)-"

[< Back to Blog](#)

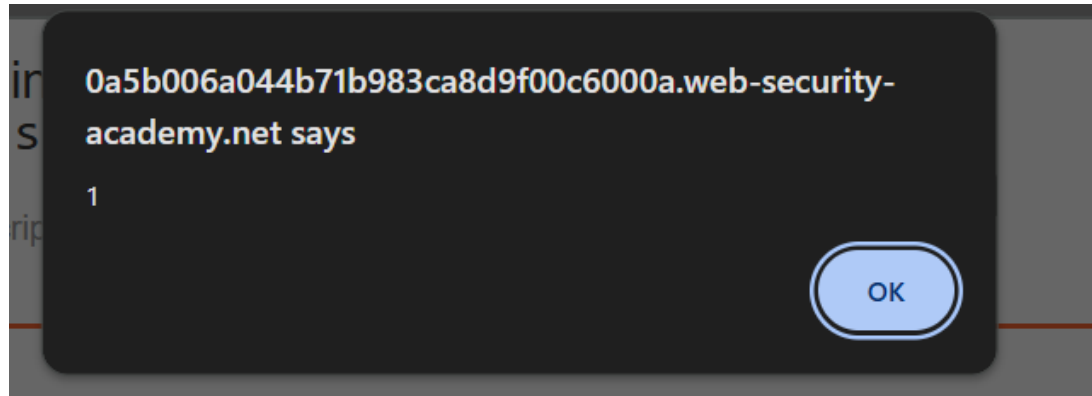
10. Lab: DOM XSS in `document.write` sink using `source_location.search` inside a select element

Steps to Solve the Lab

- pen the lab.
- Modify the URL in your browser's address bar



- Enter.
- If successful, you'll see a JavaScript alert(1) popup.
- The lab will then be marked solved



Congratulations, you solved the lab!

Share your skills!



Continue learning >>

Cheshire Cat Grin



507 22

[Home](#)