**PortSwigger**

**Cross-site Scripting - XSS**

**Apprentice Lab 1 to Lab 9**

**Sachila Hirunjith**

Table of Contents

[1. Lab: Reflected XSS into HTML context with nothing encoded 3](#_Toc205054050)

[2. Lab: Stored XSS into HTML context with nothing encoded 4](#_Toc205054051)

[3. Lab: DOM XSS in document.write sink using source location.search 5](#_Toc205054052)

[4. Lab: DOM XSS in innerHTML sink using source location.search 6](#_Toc205054053)

[5. Lab: DOM XSS in jQuery anchor href attribute sink using location.search source 7](#_Toc205054054)

[6. Lab: DOM XSS in jQuery selector sink using a hashchange event 8](#_Toc205054055)

[7. Lab: Reflected XSS into attribute with angle brackets HTML-encoded 10](#_Toc205054056)

[8. Lab: Stored XSS into anchor href attribute with double quotes HTML-encoded 11](#_Toc205054057)

[9. Lab: Reflected XSS into a JavaScript string with angle brackets HTML encoded 13](#_Toc205054058)

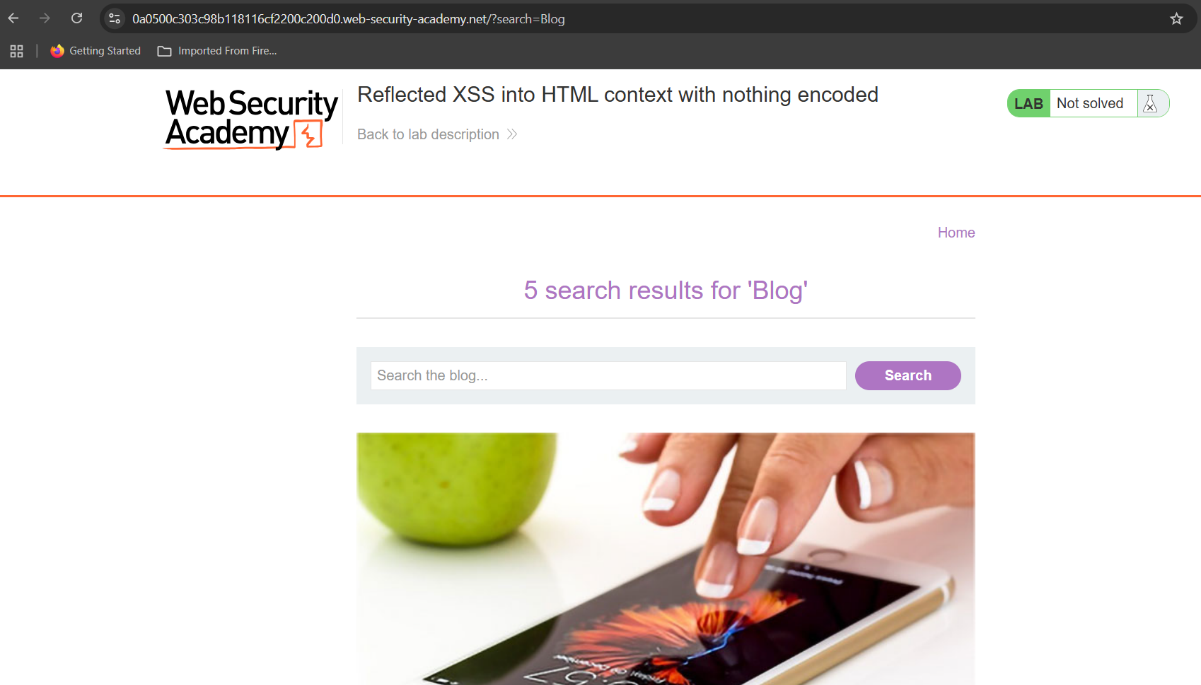
# Lab: Reflected XSS into HTML context with nothing encoded

In this lab nothing is encoded as mentioned in its topic and it has mentioned that the vulnerability is in the search function. So, after I accessed the lab just searched for a random word just to see how it works, in my case “Blog”.

It showed that there are 5 search results according to my search query and the query itself is clearly visible in the URL as well. The goal of this lab was to cross-site scripting attack that calls the alert function. So, I just entered a script in the search box of alert as **<script> alert(“vulnerable”) </script>**

Then the lab was completed but we can do the same thing in the URL as well. Since we can see the search query in the URL, for example **?search=Blog** in my case. So I replaced the blog with the script which the same as previous.

**?search=<script> alert(“vulnerable”) </script>**



A white and purple line

AI-generated content may be incorrect.

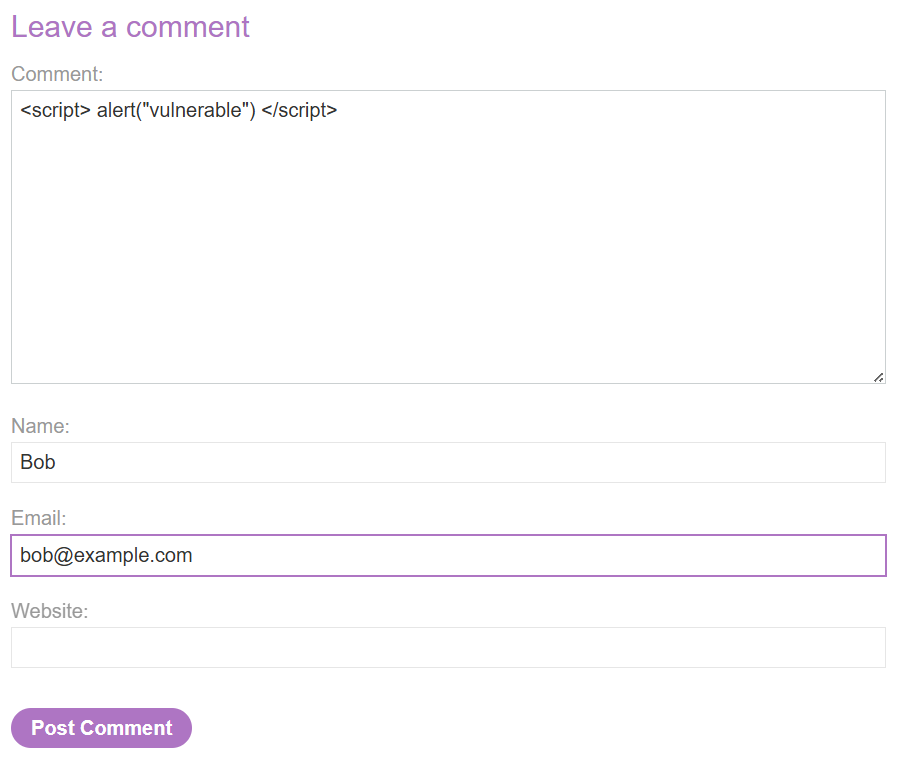
A screenshot of a computer

AI-generated content may be incorrect.

# Lab: Stored XSS into HTML context with nothing encoded

In the description it says in this lab contains a stored cross-site scripting vulnerability in the comment functionality. So, after accessing the lab we cannot see a search box like the previous one. So, under each post there is a “View post” button and I clicked on that as to find the comment section.

So, after scrolling to the comment section, I just added the script as a comment as in the below figure. Then after entering the name and the email I just posted the comment. Now the lab is solved.

Unlike in the previous challenge it wont display the alert as it is a stores script and it will execute when the next person views the same post but if you want to check the functionality you can just move back to the previous page and refresh the page then it will pop up the alert.

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

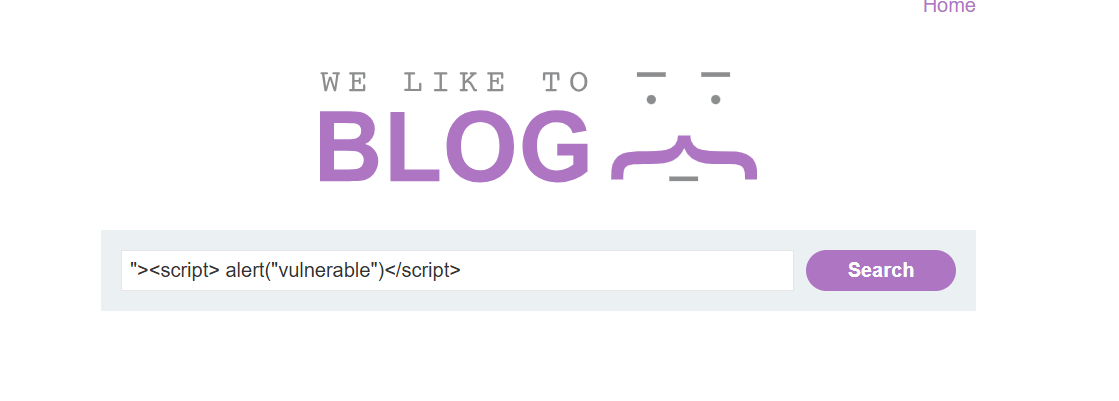
In this lab it has stated that the document.write function has called with data from location.search. Hence, the window.location.search extracts the part with and after ‘?’. So, when I searched for “Blog” the location.search will be “?search=Blog”.

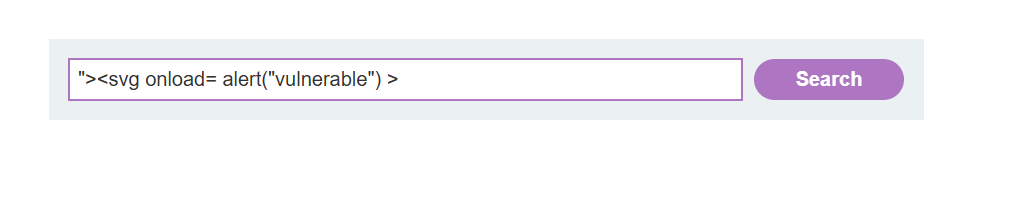
First, I searched for a random word which is ‘Blog’ as in the previous labs and it gave me the results as previous, but we just can’t use script within the search box to execute as it will be considered as just a string for an attribute. If you enter the script, it won’t do anything as its just a string not actually a script.

So, for this lab, I entered the command **”><script> alert(“vulnerable”) </script>.** Then I could complete the challenge. What the above command does is, ” will conclude the attribute value and ‘>’ closes the current tag then the script is injected separately which makes our job done.

But some web applications will filter/sanitize and even block scripts so in that case this will not work so in such a case we can use <svg>. So you can replace this with <svg> to execute JavaScript as well. So, the command will be **”><svg onload = alert(“vulnerable”)>**

You can do the exact thing in the URL as well so the final part of the URL should look like **?search=”<svg onload = alert(“vulnerable”)>**







# Lab: DOM XSS in innerHTML sink using source location.search

In this case it uses innerHTML to change the contents of a div element. So, whatever we enter in the search box will be displayed in the document through innerHTML and as it is not sanitized.

In innerHTML we can pass other tags within it. So we can take advantage of that to exploit this vulnerability. So, I used the img tag to include a image and pass my js as a attribute of it. Simply I entered **<img src=1 onerror = alert(“vulnerable”)>.** In here it will try to include a image but the path is invalid then onerror attribute will work alerting the text and we can exploit the vulnerability.

Apart from the above method we can get this done according to the previous method as well. So even if you include **<svg onload = alert(“vulnerable”)>** it will work as it quite similar to previous we are just passing js withing another tag.



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

In this section it says that the vulnerability is in the submit feedback page. So, I first moved to the particular page and in the URL, it is visible that “returnPath=/” which means when we hit the back button, it will be directing to the root page or home page.

And then I checked the elements tab to check whether that the return path is extracted from the URL, which is also mentioned in the lab description. Then, it was visible that the return path is extracted from the document.location(URL).

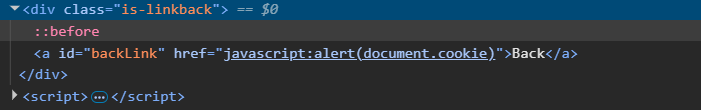
Then I tried entering a random path in the URL and hit enter to inspect whether the return path is visible in the elements section.



A screenshot of a computer

AI-generated content may be incorrect.

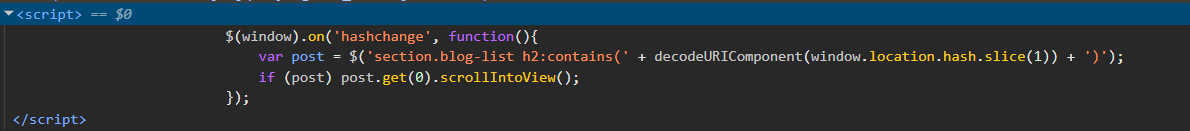
Then it is clear that the return path can be changed from the URL. So, as it asked in the lab to change the return path to “**javascript:alert(document.cookie)**”, I replaced the return path with that.



Then the lab was completed and if you hit back button you can see the alert as well.

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

In this lab it focuses on hash change event. Since the lab description has mentioned that this is about hash change event. I checked the javascript code by inspecting the elements section to check how hash function is used.



So, by viewing the above code section it is obvious that it extracts the value following the hash and if there is a match, it will scroll to that point. So first I just provided a random hash and checked how that’s going to work.



A screenshot of a computer

AI-generated content may be incorrect.

According to the above diagrams I just provided random hash and I could view the error in the console as well and the hash I was looking for is ‘xss’.

Then I tried a h2 which is there in the blog page to see if this work according to the logic I expected. So I tried providing “#Machine Parenting” in the URL and it worked as expected.

A screenshot of a computer

AI-generated content may be incorrect.So the next thing to check is whether we can provide tags as a hash which will be a vulnerability if we can. For that I tried using the same post variable and passing a h1 tag.

As in the above figure it is visible that it creates a object even though it is not visible in the blog page until you append it. So, the next thing I tried was to pass a alert function in the URL hash. For that I used an img tag with an invalid source and then the onerrror attribute to run the script.



When I tried the above hash, it generated the alert. So now we have identified the vulnerability. So now as the lab expects we need to call the print function and send that to a victim. So for that I clicked on the go to exploit server and added the payload in the body section as in the figure below.

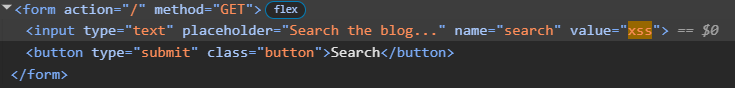


Then first I viewed the exploit and it executed successfully. Finally, to finish the lab I just clicked on the “Deliver exploit to victim”.



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

So, in this lab it has been stated that vulnerability is in the search functionality. So as we always do I just searched for a random string my case and I inspected the elements section to see where does my string is recorded in that section.

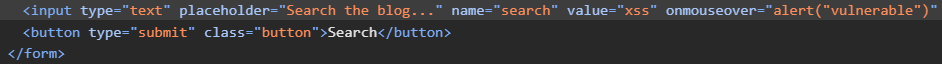
A screenshot of a search results box

AI-generated content may be incorrect.

So, in the above image as you can see, the string I searched for is added as the value of the input field. So, user can control what is within this input tag. For example I can add another attribute with this same tag, which is the vulnerability in this case.

Then I added an ‘onmouseover’ js part to create an alert which is expected in this case.

A screenshot of a computer

AI-generated content may be incorrect.

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

A screenshot of a computer

AI-generated content may be incorrect.In this lab it is said that vulnerability is in the comment section. So, first I just check the comment section by posting a normal comment and hence it is said that the issue is with the href functionality, it is needed to enter a website as well.

Above is how I entered the comment and below image shows how it is displayed when someone visits the post.

A white background with black text

AI-generated content may be incorrect.

So, you can see my name “Bob” in this case is in purple color which mean a link is attached so by clicking on it we will be directed to the link.

A black and white text

AI-generated content may be incorrect.

So, if we inspect the behavior of the source code, we can see that the link we provided as the website will be appended to the href attribute. So, we can add a JavaScript code to make an alert as expected in this lab.

So, in this case I added another comment as the username “Bob2” and added the alert function as my website.

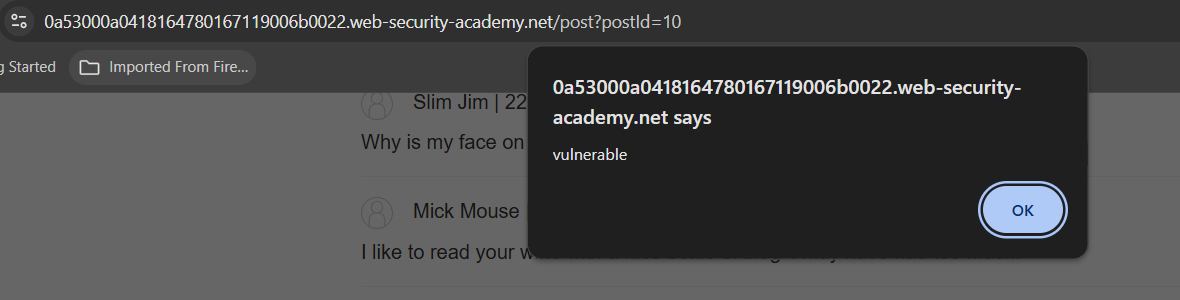
A white rectangular object with purple edges

AI-generated content may be incorrect.

Below is how my comment will display within the post’s comment section. So when someone click on the name “Bob2”, the JavaScript alert will trigger

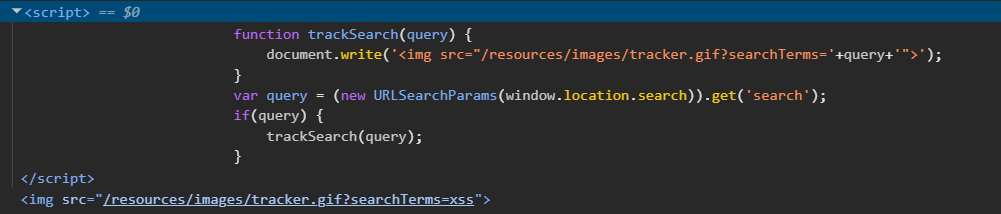
A white background with a black border

AI-generated content may be incorrect.



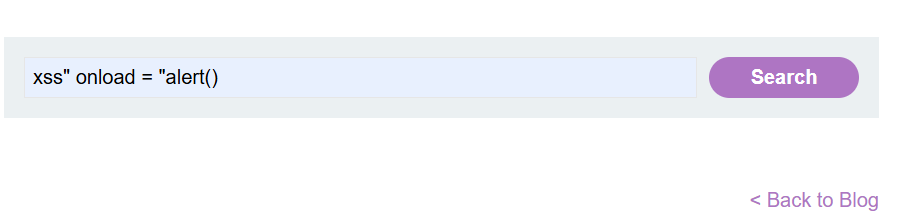
# Lab: Reflected XSS into a JavaScript string with angle brackets HTML encoded

In this lab we are expected to call an alert. So firstly as on every other challenge I searched for a random string in my case “xss”.Then I inspected the elements section and searched for that string and below is the code section of that.



We can see that the string we searched for is stored in a variable called query and then through the document.write function and image element is created with the src attribute by appending our searched string.

So here the issue is that we have control of what is appended in the image source and also we have the control of image tag. So now we can get out of the src attribute and add another attribute such as onload to call our alert function. Below image shows how to call the alert function.



So here as the previous search of “xss”, the src attribute value will end with the ” mark after xss and then the onload will call the alert function. Here after the alert ” are not closed as those will always add according to the trackSearch function as we can view in the screenshot of inspecting elements section. That will complete this lab as well