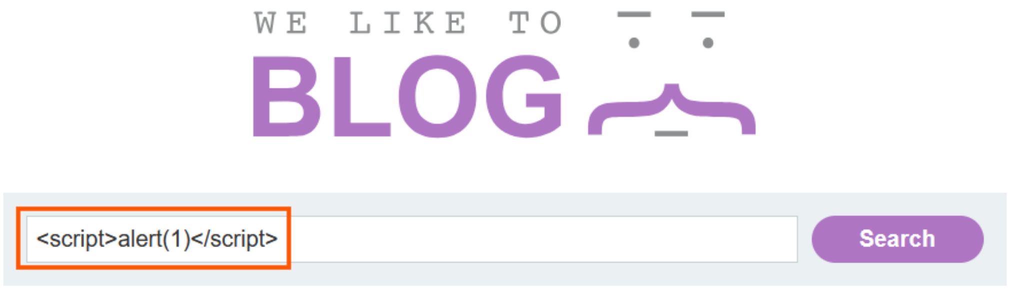
XSSi Write-Ups

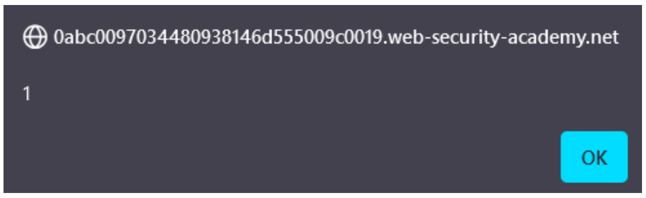
Lab 1 : Reflected XSS into HTML context with nothing encoded

In this lab, we have to perform reflected cross-site scripting attack by exploiting a XSS vulnerability in the search functionality, resulting into calling (or reflecting us back) the alert function in the website.



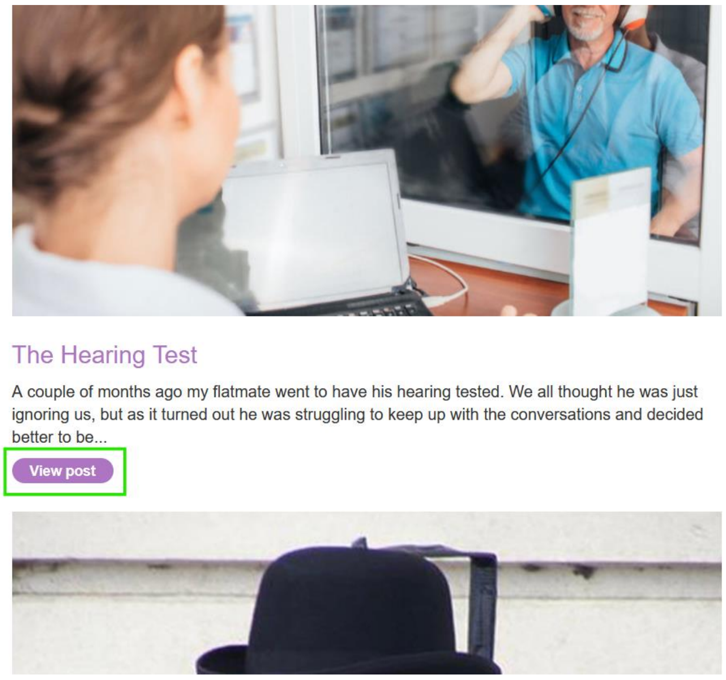
```
<script>alert(1)</script>
```

This XSS code in the search input field that executes on the website and calls the alert function with 1.



```
0abc0097034480938146d555009c0019.web-security-academy.net
1
```

Lab 2 : Stored XSS into HTML context with nothing encoded

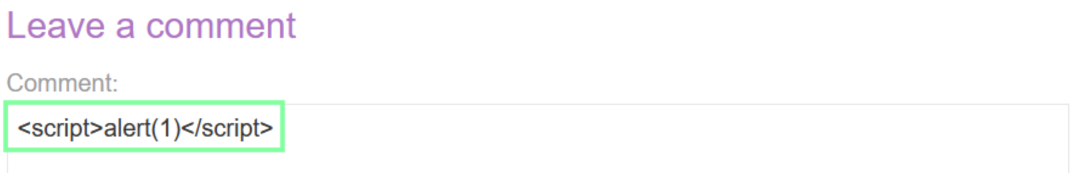


The Hearing Test

A couple of months ago my flatmate went to have his hearing tested. We all thought he was just ignoring us, but as it turned out he was struggling to keep up with the conversations and decided better to be...

In this lab, we have to perform XSS attack where we utilise the comment functionality of the website to our XSS code and execute it when blog post is viewed. So, let view any post on the website.

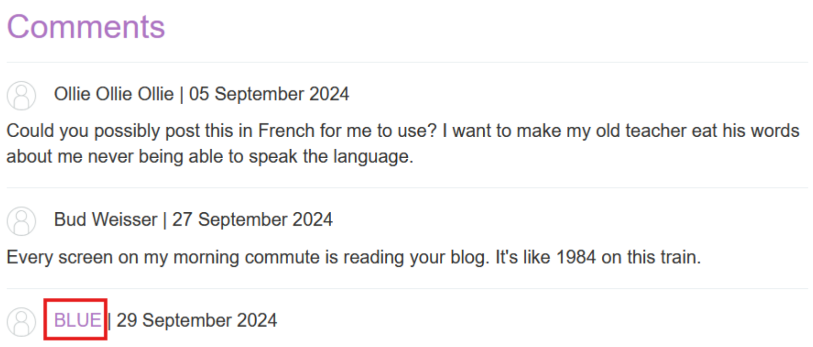
In order to inject our XSS code we have to find the comment section of that post by scrolling down and putting the exploit in the comment body filling other input fields as required and submit the comment.


Leave a comment

Comment:

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

When our comment got submitted, we visit that post page again and we can see an alert pops up with 1, every time we visit it. We can also execute our stored exploit by clicking on our comment's username.


Comments

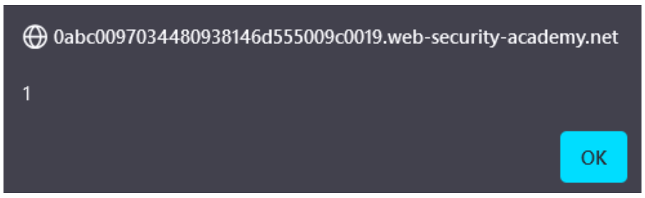
Ollie Ollie Ollie | 05 September 2024

Could you possibly post this in French for me to use? I want to make my old teacher eat his words about me never being able to speak the language.

Bud Weisser | 27 September 2024

Every screen on my morning commute is reading your blog. It's like 1984 on this train.

BLUE | 29 September 2024

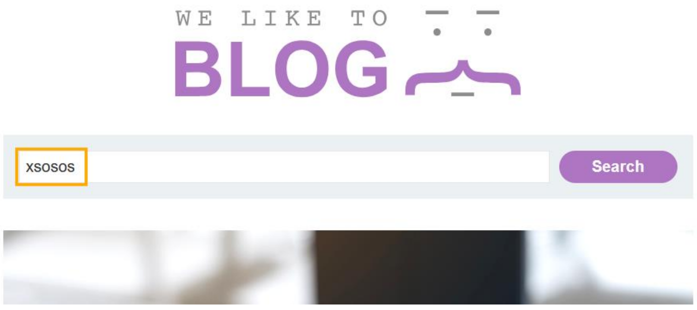

0abc0097034480938146d555009c0019.web-security-academy.net

1

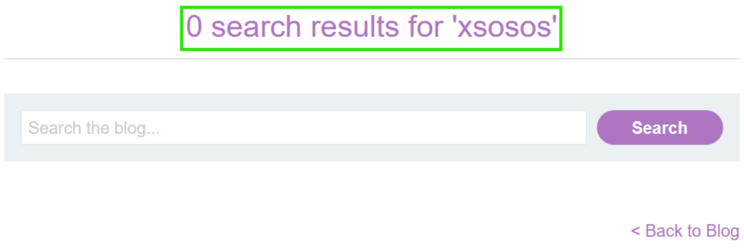
This also proves that our exploit is now stored on the server's database and will execute every time a user visits this post page. This completes our lab, let's move on to the next lab!

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

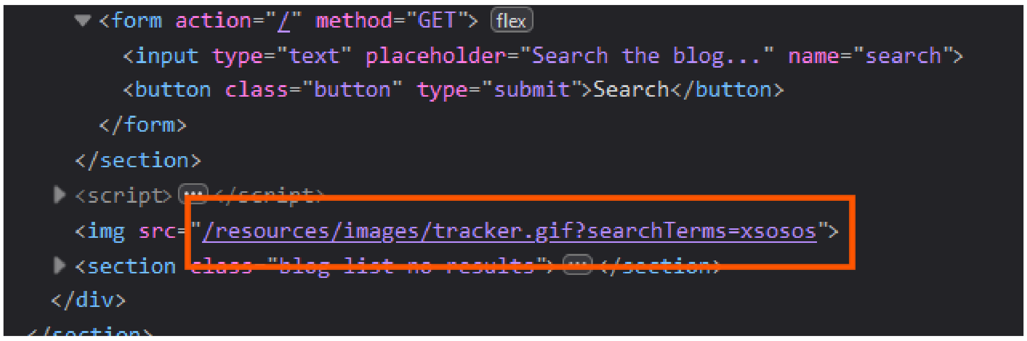
This lab requires us to use sink using source DOM-based attack in XSS in the search query tracking functionality which uses JavaScript to write data out to the page, with data from search input values, which we can control using website URL or search bar itself.



Similar to Lab 1, here we put some random string in the search bar and get back a response to analyze.

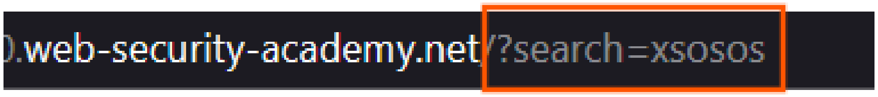


We can see there is no search with the search query but let's deep dive and inspect the page for something useful in the DOM of the website and for our lead.



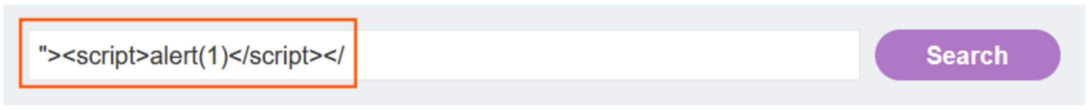
```
<form action="/" method="GET"> flex
  <input type="text" placeholder="Search the blog..." name="search">
  <button class="button" type="submit">Search</button>
</form>
</section>
<script> ··· </script>
<img src="/resources/images/tracker.gif?searchTerms=xsosos">
<section class="blog-list no-results"> ··· </section>
</div>
</section>
```

In the inspect tab we can see an "img" tag with a src to our searched random string, which is not validated because results are 0 but it's still showing, and if we open the script tag above it, we can see it is retrieving the params value from **window.location.search** using **URLSearchParams()** a function to get the values of URL params and then writing it into img src in the "**searchTerms**" using **document.write** function, there we can clearly see no validation provided to encode HTML or user inputs and is vulnerable to XSS.



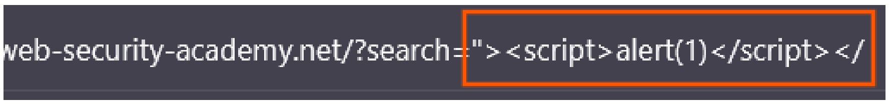
```
).web-security-academy.net/?search=xsosos
```

Our script looks something like this, <**script**>**alert(1)**</**script**> with other bypassing characters like (**"**) to get out of the string, (**>**) to close the img tag, etc, also tags like <**svg**>,<**body**> can be used.



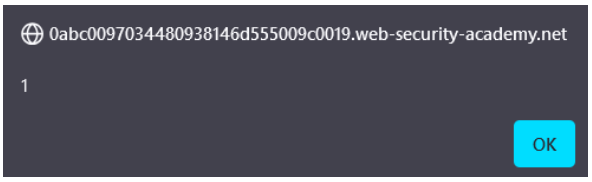
```
"><script>alert(1)</script></
```

OR



```
web-security-academy.net/?search="><script>alert(1)</script></
```

When we search using this XSS code, we get an alert popping up with 1 again. So many 1..s, although we have solved this lab completely exploiting the DOM, we can even see that inspecting the HTML.

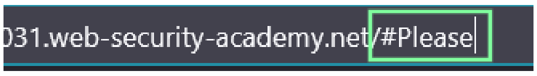


0abc0097034480938146d555009c0019.web-security-academy.net

1

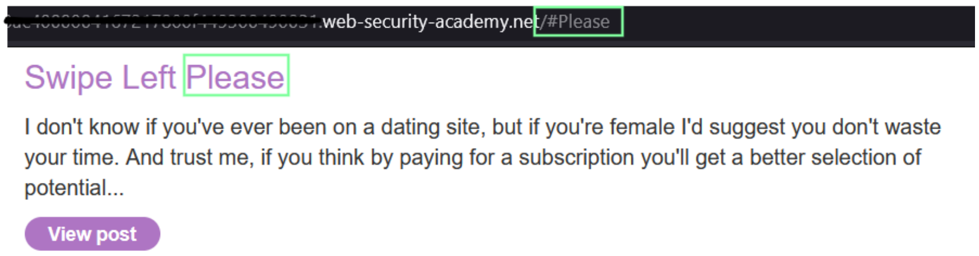
Lab 4 : DOM XSS in jQuery selector sink using a hashchange event

We have to perform a DOM based XSS where JQuery's selector function **$()** has been used for auto-scroll to a given post, whose title is passed in **location.hash** property. To solve this lab, we have to perform **print()** function on every click or reload of the page using our exploited link to the victim.



```
031.web-security-academy.net/#Please
```

We go visit the website's home page and put this inside URL with '**#**' which put the values after that into **location.hash** property on which the JQuery executes. We can also see if we hit enter it scrolls to the post automatically.

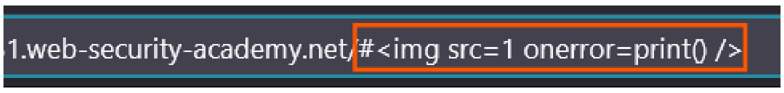


[redacted].web-security-academy.net/#Please

Swipe Left Please

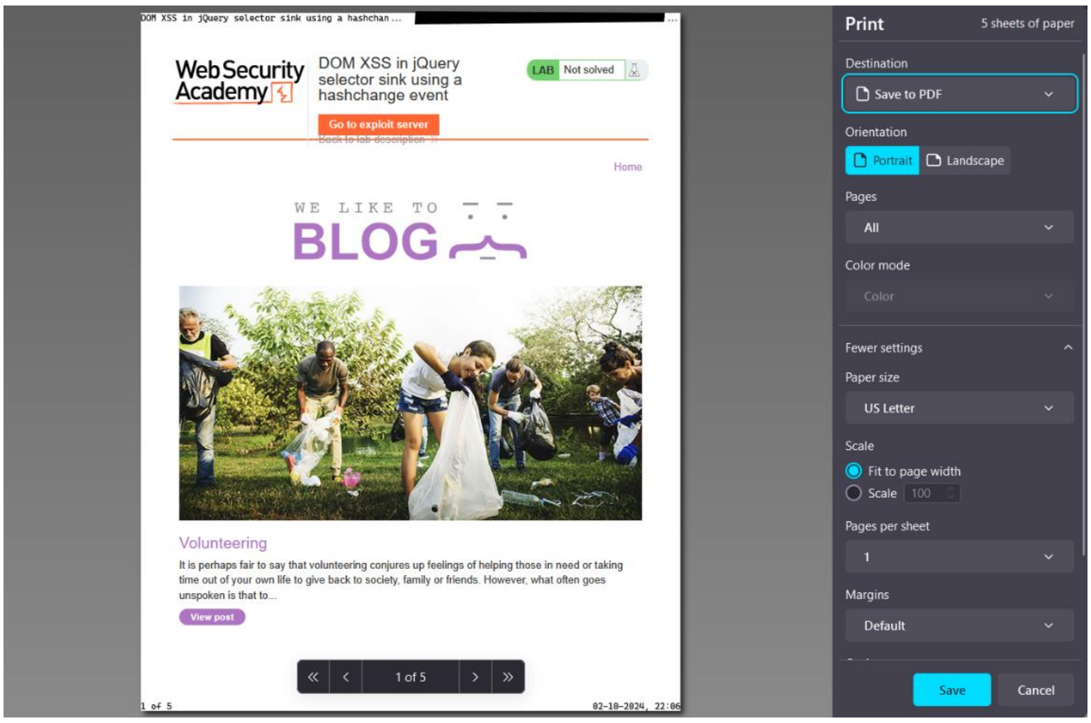
I don't know if you've ever been on a dating site, but if you're female I'd suggest you don't waste your time. And trust me, if you think by paying for a subscription you'll get a better selection of potential...

But if we search the same term again it won't work no matter what we do, so we need to have some way to perform it every time we search the term it executes. Before that lets put some exploit inside it to check if it works or not because it is the outdated version of JQuery that is running on this website which creates a new HTML element if we put HTML tags inside it, ops, my bad lets do it and see for ourselves if it happens.



```
1.web-security-academy.net/#<img src=1 onerror=print() />
```

We are going to put an img tag with src anything and perform **print()** on error event. Hit enter and let's see what happens.



DOM XSS in jQuery selector sink using a hashchan... [redacted]

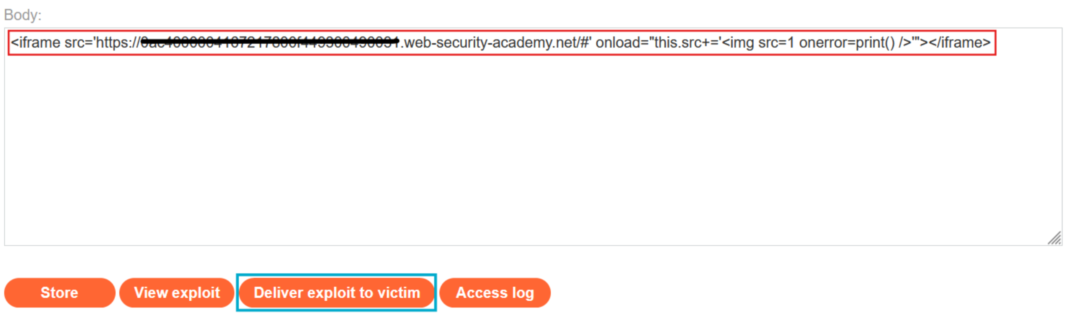
It is perhaps fair to say that volunteering conjures up feelings of helping those in need or taking time out of your own life to give back to society, family or friends. However, what often goes unspoken is that to...

1 of 5 02-10-2024, 22:06

Print 5 sheets of paper

Destination: Save to PDF
Orientation: Portrait
Pages: All
Color mode: Color
Fewer settings
Paper size: US Letter
Scale: Fit to page width
Pages per sheet: 1
Margins: Default

Okay, we made the website perform the print function and so we exploited the vulnerability that exists on this lab but to solve this lab we have to make it so that it works every time on victim's search that is through our exploited link we going to send to the victim. Let's go to our exploit server.



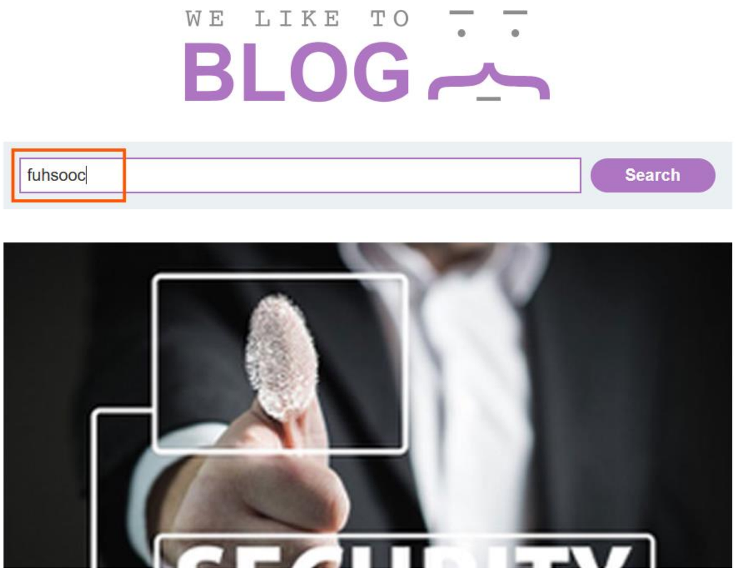
Body:

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

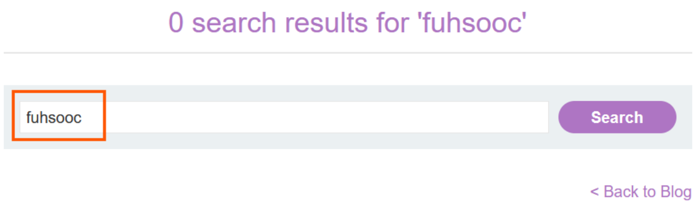
Here, we found one solution which is creating an iframe on every click or reload of the page with the original src that is our lab and what happens is when our iframe loads we use **onload** event to perform the same **print()** event inside img tag on error occurred by putting src to anything. We can view our exploit by clicking on View exploit and then if everything works correctly we can deliver the exploit to victim which solves the lab.

Lab 5 : <u>Reflected XSS into attribute with angle brackets HTML-encoded</u>

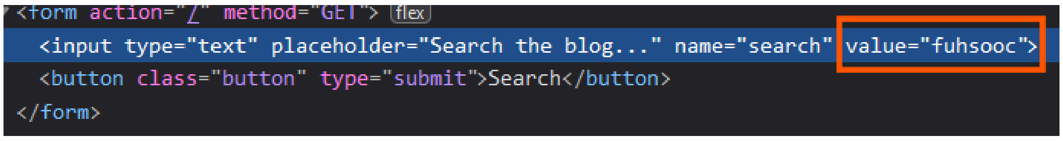
This lab contains a search functionality on home page where angle brackets are HTML-encoded but still there is a XSS vulnerability which we have to manipulate and perform an attribute injection to call alert function which solves the lab.



Let's search some random string in the search input bar, and observe the response we get back.

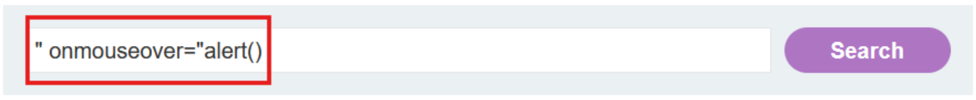


We can see our random string, searched with no results but inside search input it is still present, so let's inspect the website and see how it is being stored.



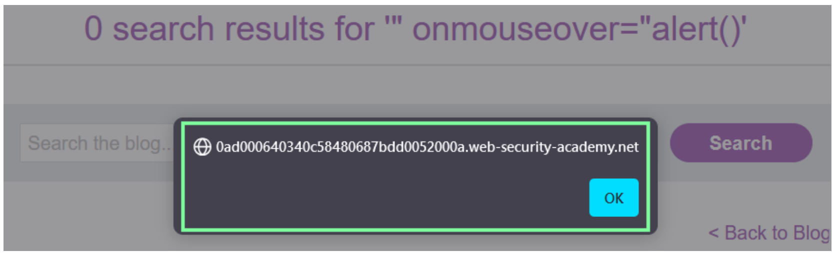
```
<form action="/" method="GET"> flex
  <input type="text" placeholder="Search the blog..." name="search" value="fuhsooc">
  <button class="button" type="submit">Search</button>
</form>
```

So, we can put any string and it stored inside the value attribute but can we break out of it? Let's find out by putting our exploit inside the search input.



```
" onmouseover="alert()
```

That is **onmouseover** event and frame the string such that it is in the correct format and hit enter. And there we go we solved our lab but wait we didn't saw any alert right? Let's try hovering over the search input.



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

0ad000640340c58480687bdd0052000a.web-security-academy.net

And here we have our alert every time we hover over the search input bar.