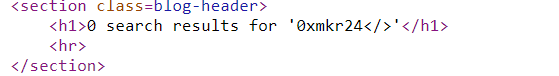
# ****Lab 1# Reflected XSS into HTML context with nothing encoded****

We notice that there is a search box so let's try to test it, If it’s vulnerable to xss or not.

* I’ll try my payload 0xmkr24</> and see if it is reflected in the page source.



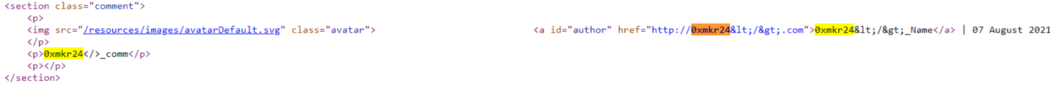
* there is no filter so try our payload <script>alert(1)</script> and congrats lab has been solved.

# ****Lab 2# Stored XSS into HTML context with nothing encoded****

in this lab, we’ll notice that its a blog

open any post and you’ll see that there is a form

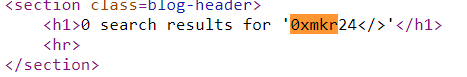
fill this form and see if your data is reflected in page source



I notice there’s no filter in the comment box so let’s try our payload in it <script>alert(1)</script> and congrats lab has been solved.

# ****Lab 3: Reflected XSS into HTML context with most tags and attributes blocked****

we have a search box so let's try to inject any payload like 0xmkr</>



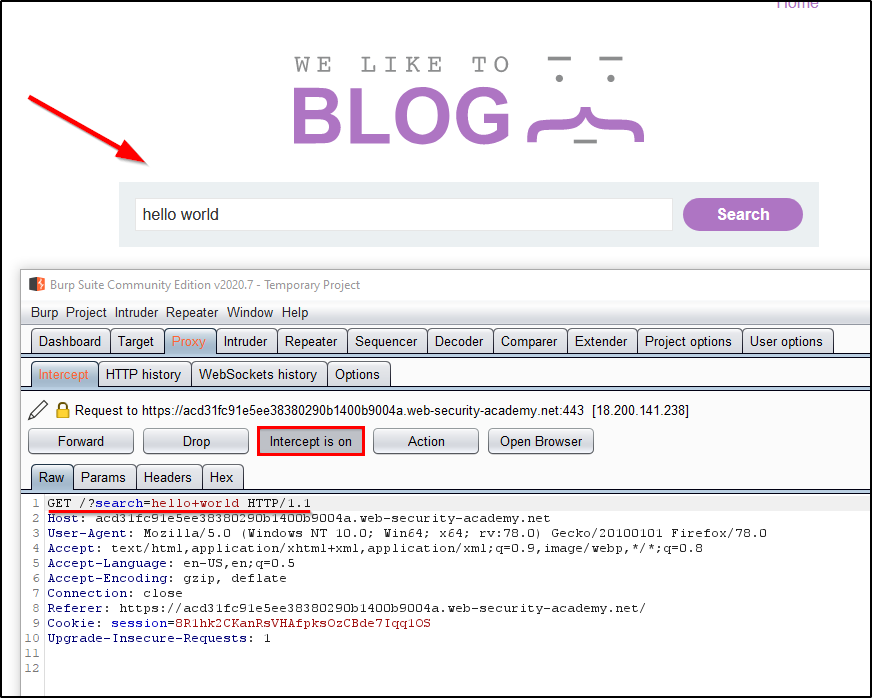
we notice that <,/> not filtered so try <script>alert(1)</script>

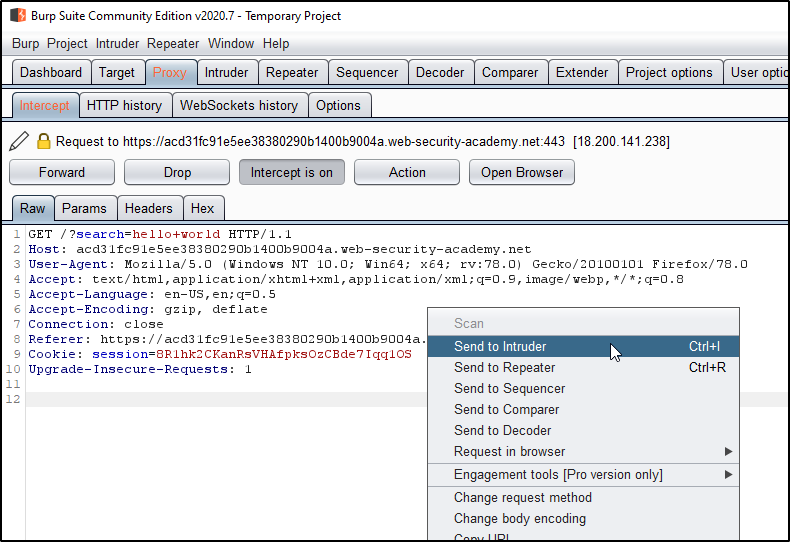
IMG_260

Unfortunately <script> is filtered so let's try another payload like <img src=1 onerror=print()> but it’s the same output, I’ll use Burp Intruder to test which tags and attributes are being blocked.

Now open burp, send a request from the search box and intercept it from burp, then send this request to an intruder

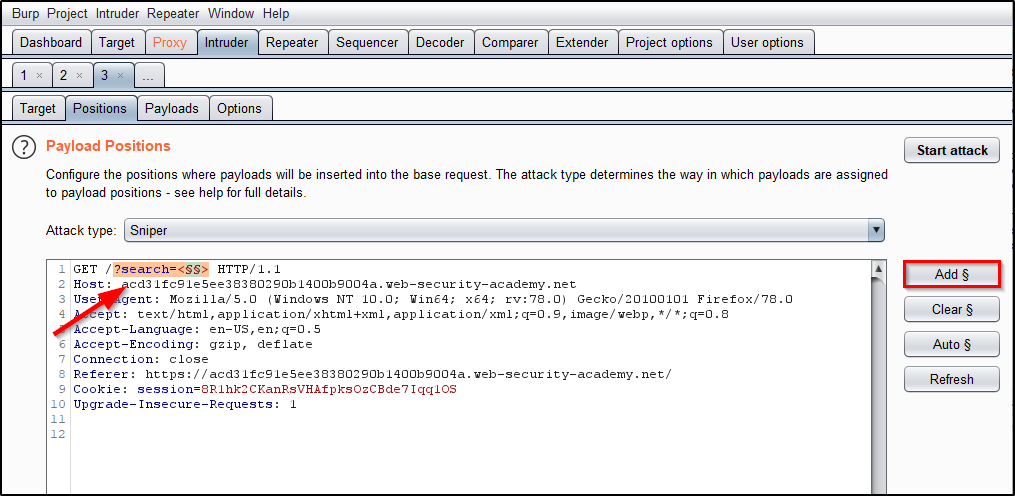
I have got these screenshots from the internet because I’m lazy to take them, forgive me that :)



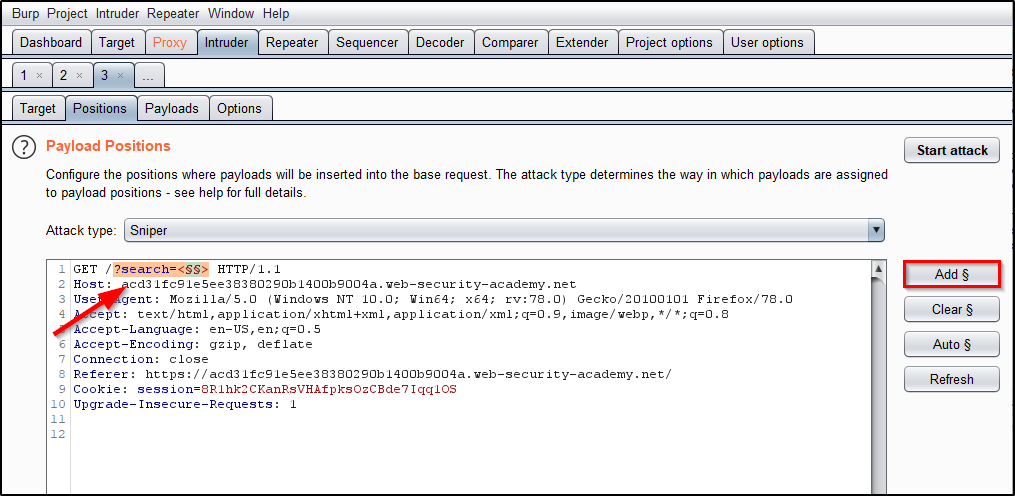


Now follow instructions from the Solution section

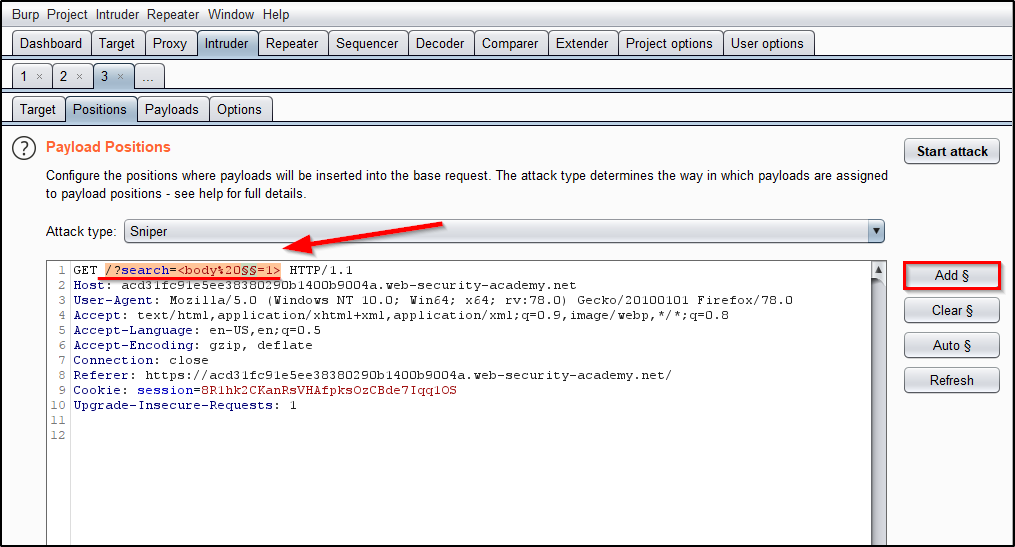
* In Burp Intruder, in the Positions tab, click “Clear §”. Replace the value of the search term with: <>
* Place the cursor between the angle brackets and click “Add §” twice, to create a payload position. The value of the search term should now look like this: <§§>



* Visit the [XSS cheat sheet](https://portswigger.net/web-security/cross-site-scripting/cheat-sheet" \t "https://0xmkr24.medium.com/_blank) 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, This indicates that the WAF allows this tag, and perhaps we can use it for our exploitation process.
* Go back to the Positions tab in Burp Intruder and replace your search term with: <body%20=1> , Now we know we can use the body tag, we need to know which events we can use.
* 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 this: <body%20§§=1>
* Visit the [XSS cheat sheet](https://portswigger.net/web-security/cross-site-scripting/cheat-sheet" \t "https://0xmkr24.medium.com/_blank) 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 the victim”.

But before that let’s break down this payload

* ifram tag embeds another HTML page into a current page.
* onresize attribute is an event attribute that executes a javascript when you resize the browser window.
* onload event attribute executes the javascript when the webpage is loaded.

What’s happening now? when you visit the website “[https:/](https://ac511f2e1ec1f6c880c02798019100fc.web-security-academy.net/exploit" \t "https://0xmkr24.medium.com/_blank)/your-lab-id.web-security-academy.net/exploit”, you will see a small white box on the web page.

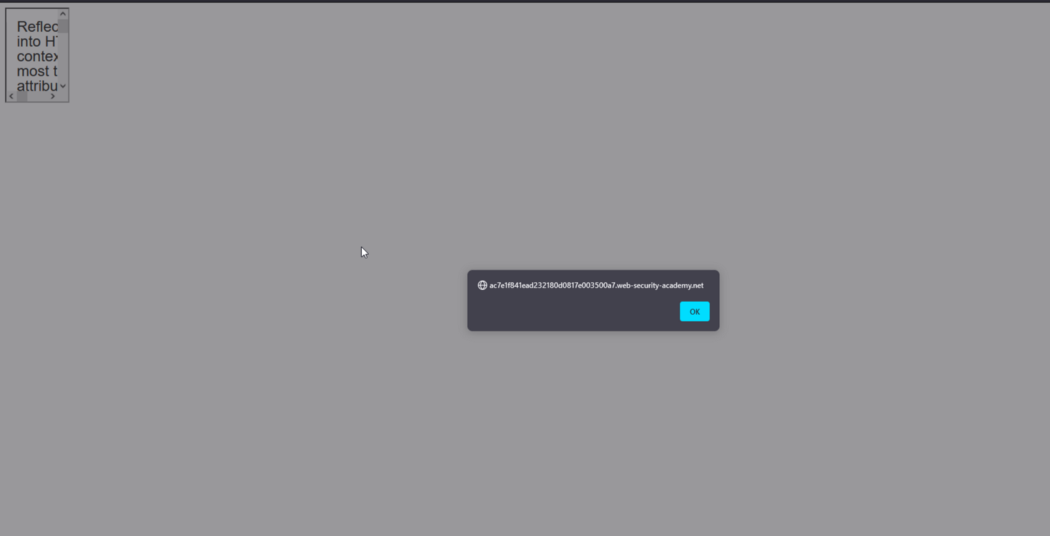
When the web page is loaded, this triggers the onload event to execute the javascript: this.style.width=‘100px’, meaning the width of the small white box you see will be adjusted to 100 pixels which is small.

When the width is adjusted, this triggers the next event which is the onresize event. This will execute the javascript: alert(document.cookie).

The alert([document.cookie](https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie" \l ":~:text=The Document property cookie lets,actual values of the cookies.)" \t "https://0xmkr24.medium.com/_blank)) allows you to read the cookies associated with the document on that website.

Next, click view exploit to go to the exploit link.

Once you are at the web page, you will see a white box created using the iframe tag.



Immediately, it will change to a smaller width size (100px) and this will activate the “onresize ”event, which will execute the alert function, showing a pop-up message as shown below.

There isn’t any message shown as there is no cookie associated with the document on that webpage.

# ****Lab 4: Reflected XSS into HTML context with all tags blocked except custom ones****

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>   
# decode   
<xss id=x onfocus=alert(domain.cookie) tabindex=1>

* *<xss id=x*to create a custom tag in HTML with id=x , if you want to know more about custom tags read this [blog](https://www.html5rocks.com/en/tutorials/webcomponents/customelements/" \l ":~:text=Custom Elements allow web developers to define new types of HTML elements.&text=Define new HTML/DOM elements,functionality into a single tag" \t "https://0xmkr24.medium.com/_blank).
* onfocus you can use any Event.
* tabindex=1 indicates that it’s element can be focused — [tabindex](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/tabindex" \t "https://0xmkr24.medium.com/_blank)
* #x because of onfocus event, This will make the focus on x.

# **Lab 5: Reflected XSS with event handlers and href attributes blocked**

i tried many payloads but its tag was blocked so I check Solution and used its payload, now let’s break down it

?search=%3Csvg%3E%3Ca%3E%3Canimate+attributeName%3Dhref+values%3Djavascript%3Aalert(1)+%2F%3E%3Ctext+x%3D20+y%3D20%3EClick%20me%3C%2Ftext%3E%3C%2Fa%3E

Decode

<svg><a><animate attributeName=href values=javascript:alert(1) /><text x=20 y=20>Click me</text></a>

* <a> creates a hyperlink to web pages
* The SVG <animate> element provides a way to animate an attribute of an element over time.
* <attributeName> The attributeName attribute indicates the name of the CSS property or attribute of the target element that is going to be changed during an animation.

****Example of svg animate****

<svg viewBox="0 0 10 10" xmlns="http://www.w3.org/2000/svg">  
 <rect width="10" height="10">  
 <animate attributeName="rx" values="0;5;0" dur="10s" repeatCount="indefinite" />   
 </rect>   
</svg>

# **Lab 6: Reflected XSS with some SVG markup allowed**

we will make the same steps in lab 3, we will find that <svg>, <animatetransform>, <title>, and <image> tags, which received a 200 response.

then go back to the Positions tab in Burp Intruder and replace your search term with: <svg><animatetransform%20=1>

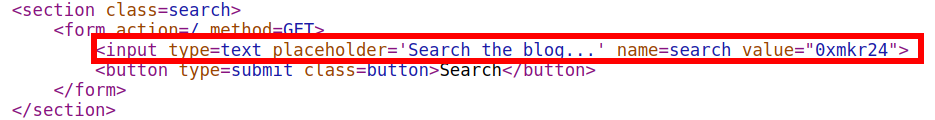
Place the cursor before the = character and click "Add §" twice to create a payload position. The value of the search term should now be: <svg><animatetransform%20§§=1>

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.  
https://your-lab-id.web-security-academy.net/?search=%22%3E%3Csvg%3E%3Canimatetransform%20onbegin=alert(1)%3E

# **Lab 7# Reflected XSS into attribute with angle brackets HTML-encoded**

try to submit any random input like 0xmkr24 and open page source to find how this input was reflected



we need to close double-quotes value “ and after that, we can use any event we need like onmouseover , onclick etc, you can found this event in the [Cross-site scripting (XSS) cheat sheet](https://portswigger.net/web-security/cross-site-scripting/cheat-sheet" \t "https://0xmkr24.medium.com/_blank).

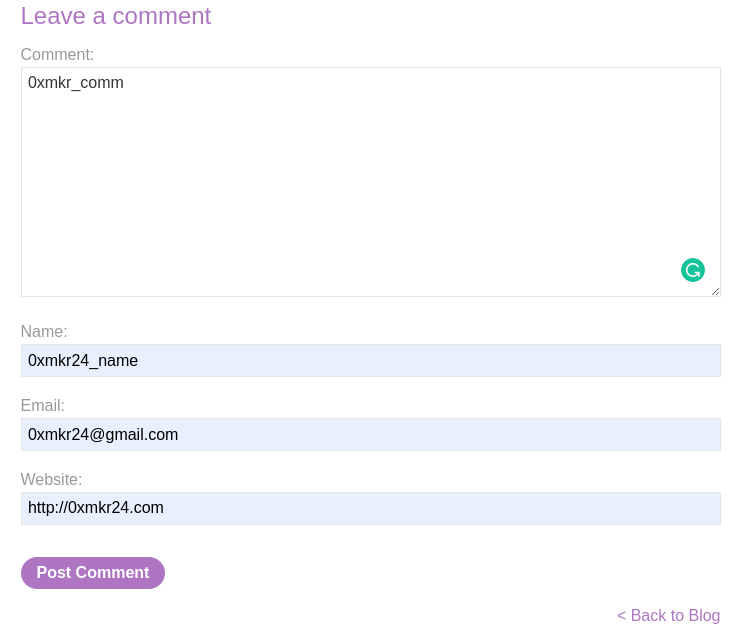
I’ll use the second double quote to don’t break

our payload will be like this value=”event=”alert(1)”>

and now I’ll use this payload “onclick=”alert(1) , now when you click on the search box, “1” will popup, and lab will be solved

# **Lab 8# Stored XSS into anchor href attribute with double quotes HTML-encoded**

there is a blog when opening it, I found a form section, try to fill its fields with random values

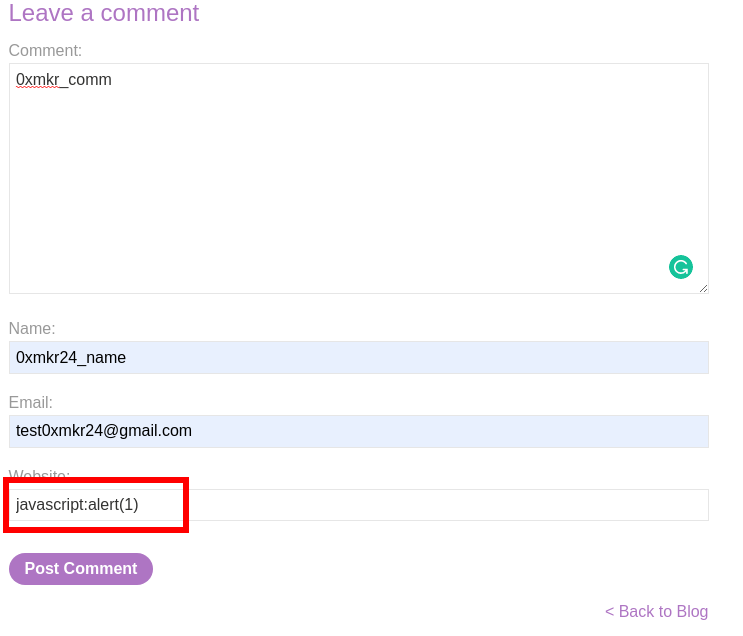


and post it, then return to this page, you will find your comment is reflected

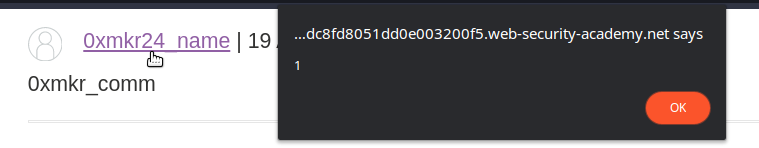
open page source and I notice that the data in website input has been reflected inside an anchor href attribute.



I will try javascript:alert(1) in website



and post it then return to a page again, when I click on website URL the alert happens

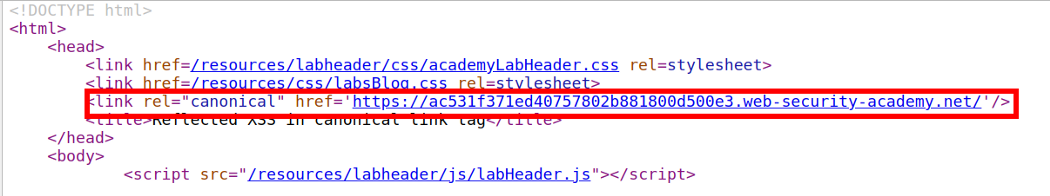


open page source to see how it reflected

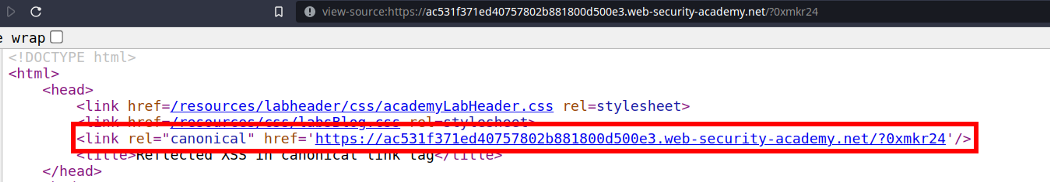


# **Lab 9# Reflected XSS in canonical link tag**

nothing on the home page so let’s see the page source, I notice the URL of the website is reflected in the page source



I try to add ? in URL and wow it’s reflected too



I try some payloads but it didn’t work with me if you find any payload work, tell me.

so I use the payload in Solution, let’s break down it

You can exploit this behavior using access keys and user interaction on Chrome

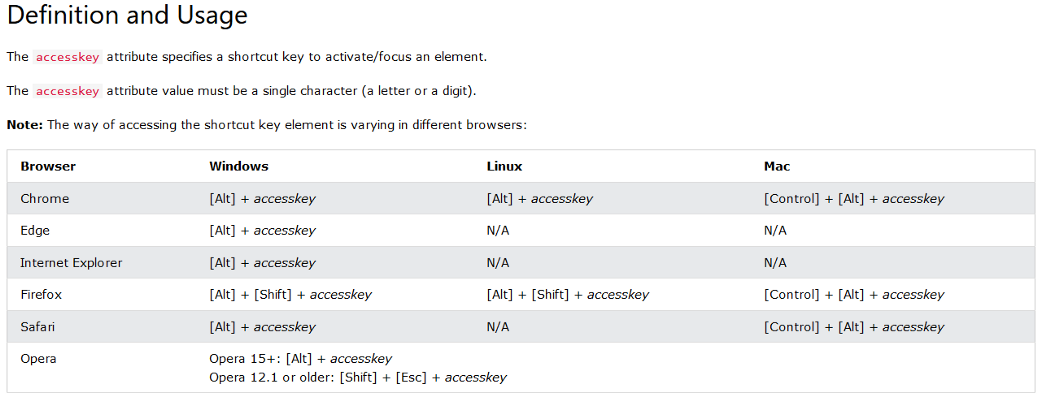
**Access keys** allow you to provide keyboard shortcuts that reference a specific element.

**The access key attribute** allows you to define a letter that, when pressed in combination with other keys (these vary across different platforms), will cause events to fire

## **[HTML accesskey Attribute](https://www.w3schools.com/tags/att_global_accesskey.asp" \t "https://0xmkr24.medium.com/_blank)**

### [The accesskey attribute specifies a shortcut key to activate/focus an element. The accesskey attribute value must be a…](https://www.w3schools.com/tags/att_global_accesskey.asp" \t "https://0xmkr24.medium.com/_blank)

[www.w3schools.com](https://www.w3schools.com/tags/att_global_accesskey.asp" \t "https://0xmkr24.medium.com/_blank)



?%27accesskey=%27x%27onclick=%27alert(1)

Decode

?'accesskey='x'onclick='alert(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