

## Lab 3 -- Andrew Stevenson

### Lab 3.1

- [1. cross-site-scripting/reflected \(1\)](#)
- [2. cross-site-scripting/reflected \(2\)](#)
- [6. cross-site-scripting/reflected \(3\)](#)
- [7. cross-site-scripting/reflected \(4\)](#)
- [8. cross-site-scripting/reflected \(5\)](#)
- [9. cross-site-scripting/reflected \(6\)](#)
- [10. cross-site-scripting/reflected \(7\)](#)
- [11. cross-site-scripting/reflected \(8\)](#)
- [12. cross-site-scripting/dom-based \(1\)](#)
- [13. cross-site-scripting/dom-based \(2\)](#)
- [14. cross-site-scripting/dom-based \(3\)](#)
- [15. cross-site-scripting/dom-based \(4\)](#)
- [16. cross-site-scripting/dom-based \(5\)](#)
- [18. cross-site-scripting/stored \(1\)](#)
- [19. cross-site-scripting/stored \(2\)](#)
- [20. cross-site-scripting/stored \(3\)](#)
- [23. cross-site-scripting/dom-based \(6\)](#)
- [25. cross-site-scripting/exploiting \(1\)](#)
- [28. cross-site-scripting/exploiting \(2\)](#)

### Lab 3.2

- [1. cors \(1\)](#)
- [8. Content-Security-Policy examples](#)

[Example #1](#)

[Take a screenshot of the page result that includes the URL in the browser](#)

[Example #2](#)

[Example #3](#)

[Example #4](#)

### Lab 3.3

- [1. csrf \(1\)](#)
- [3. csrf \(2\)](#)
- [4. csrf \(3\)](#)
- [5. csrf \(4\)](#)
- [9. csrf \(5\)](#)
- [11. cross-site-scripting/exploiting](#)

### Lab 3.4

- [1. clickjacking \(1\)](#)

- [4. clickjacking \(2\)](#)
- [5. clickjacking \(3\)](#)
- [8. Prevention \(X-Frame-Options\)](#)

[Lab 3.5](#)

[Lab 3.6](#)

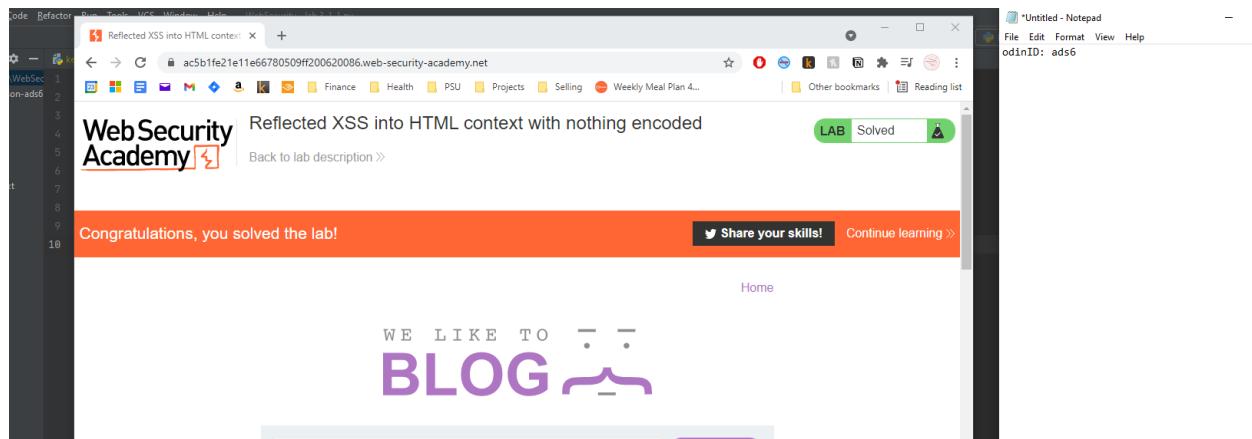
## Lab 3.1

### 1. cross-site-scripting/reflected (1)

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** Unsanitized use of the query string within the HTML allows the adversary to inject arbitrary scripts into the page for the client to unwittingly execute.

**Remediation:** Sanitize the querystring by html encoding it before using it within the HTML.

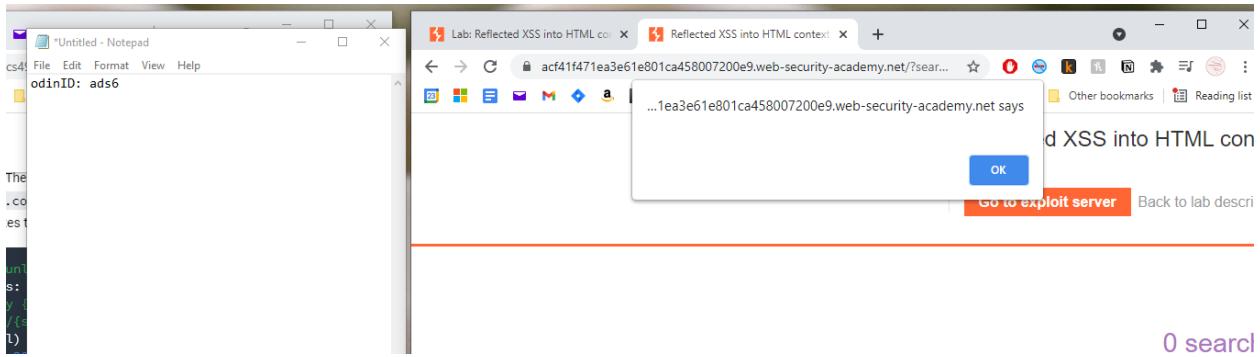


### 2. cross-site-scripting/reflected (2)

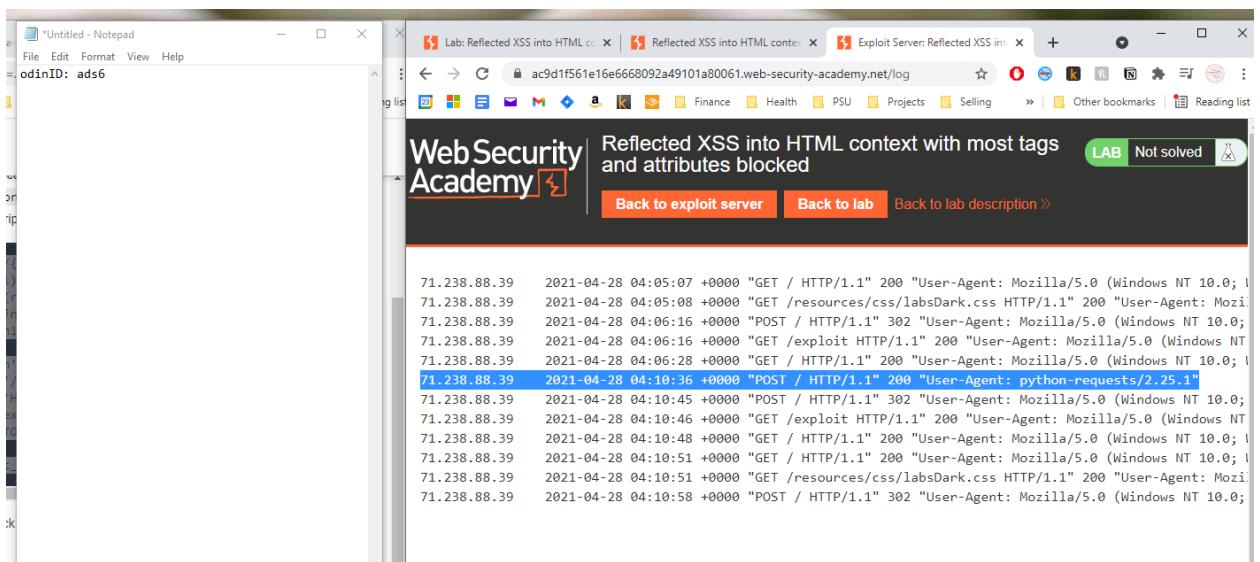
- List which window event attributes are allowed for your lab notebook

Onresize, onstorage

- Include a screenshot of the pop-up



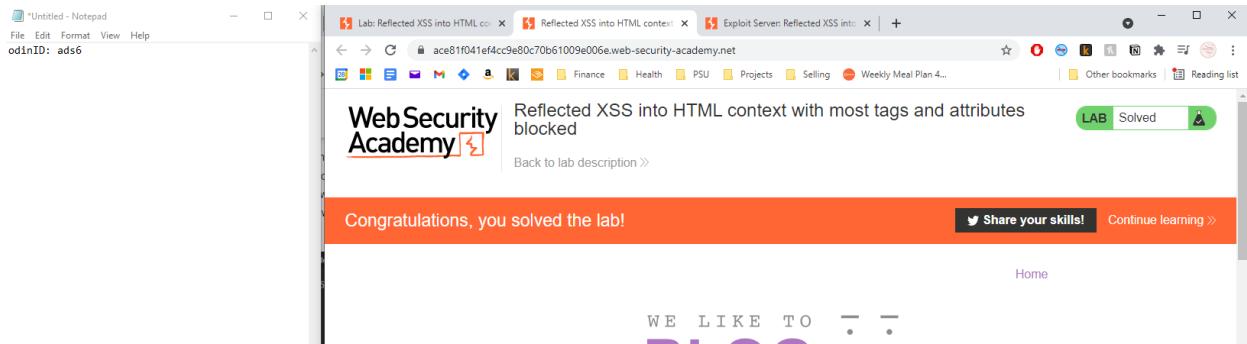
- Take a screenshot of the above log entry and include it in your lab notebook.



- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** Although the WAF blocks many scripting tags from being incorporated into the HTML, a few are allowed through, including the `onresize=` tag. Thus, the adversary may force execution of their attack page via an iframe inserted into the url that automatically resizes the client window, leading to execution of the `onresize=` payload.

**Remediation:** Properly html encode the query string before incorporating it into the HTML.



## 6. cross-site-scripting/reflected (3)

- Show the HTTP status code and response text obtained

```

import ...
site = 'acfaf1f511e9ee5e9888626778025009e.web-security-academy.net'
s = requests.Session()
odin_id = 'ads6'

site_url = f'{https://[site]}/'
tag = 'script'
search_url = f'{https://[site]}/?search=<{tag}>'

resp = s.get(search_url)
print(f'Status code is {resp.status_code} with response text {resp.text}')

```

- Show the list of tags that are not filtered

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and WebSecurity - lab 3-1-6.py. The left sidebar displays the project structure under 'WebSecurity E:\Projects\WebSec' with files like 'lab 3-1-6.py', 'tags.txt', and 'README.md'. The main code editor window contains the following Python script:

```
import ...
site = 'acfa1f511e9ee5e9888626770025009e.web-security-academy.net'
s = requests.Session()

site_url = f'https:///{site}/'
my_file = open("tags.txt", "r")

tags = my_file.readlines()
for tag in tags:
    search_url = f'{site_url}?search=<{tag.rstrip()}>'
    resp = s.get(search_url)
    if resp.status_code == 200:
        print(f'Tag \'{tag.rstrip()}\' was accepted.')

for tag in tags:
    search_url = f'{site_url}?search=<{tag}>'
    resp = s.get(search_url)
    if resp.status_code == 200:
        print(f'Tag \'{tag}\' was accepted.')

for tag in tags:
    search_url = f'{site_url}?search=<{tag}>&transform=animate'
    resp = s.get(search_url)
    if resp.status_code == 200:
        print(f'Tag \'image\' was accepted.')
    if 'svg' in resp.text:
        print(f'Tag \'svg\' was accepted.')
    if 'title' in resp.text:
        print(f'Tag \'title\' was accepted.')

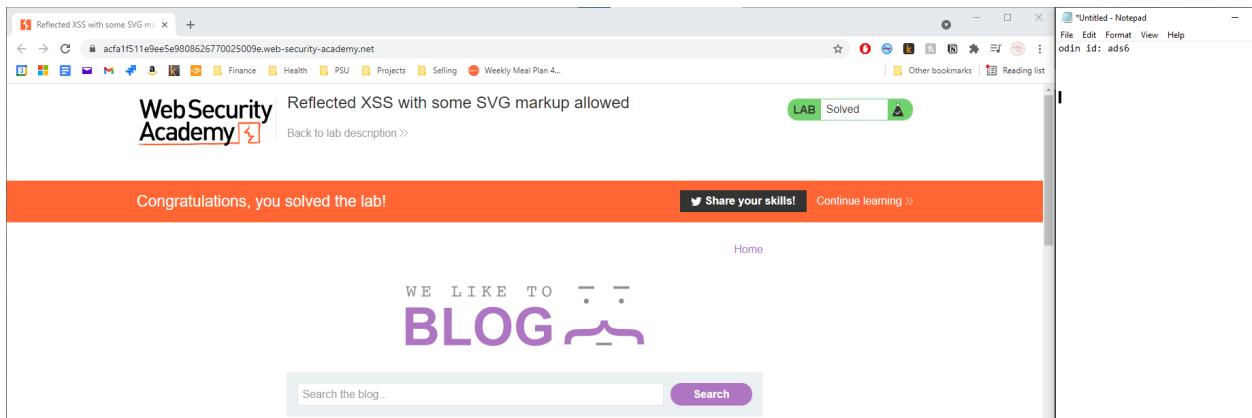
Process finished with exit code 0
```

The terminal window at the bottom shows the output of the script execution, indicating that various tags were accepted, including 'image', 'svg', and 'title'.

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** Not all tags are being filtered out from the querystring that gets converted to HTML, allowing the adversary to run an XSS attack using the <svg> and <animatetransform> tags.

**Remediation:** Filter out all tags, or --better yet--html encode the querystring contents before converting them to html.



## 7. cross-site-scripting/reflected (4)

- Show the part of the source demonstrating that the content has been HTML-encoded in the HTML context

```
view-source:https://ac7e1fd71e... + └─ Untitled - Notepad  
File Edit Format View Help  
odin id: ads6  
  
<link href="/resources/css/labBlog.css" rel="stylesheet">  
<title>Reflected XSS into attribute with angle brackets HTML-encoded</title>  
<body>  
    <script src="/resources/labHeader/js/labHeader.js"></script>  
  
    <div id="academyTableHeader">  
        <section class="academyLabBanner">  
            <div class="container">  
                <div class="logo"></div>  
                <div class="title-container">  
                    <h2>Reflected XSS into attribute with angle brackets HTML-encoded</h2>  
                    <a href="https://portswigger.net/web-security/cross-site-scripting/contextes/lab-attribute-angle-brackets-html-encoded">  
                        Back<br/>lab<br/>description<br/><svg version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px" viewBo  
<g>  
    <polygon points="14,0 0,12 6,15 0,28 8,14,30 15,1,15"></polygon>  
    <polygon points="14,3,0 12,9,1,2 25,6,15 12,9,28,8 14,3,30 28,15"></polygon>  
</g>  
</svg>  
        </div>  
        <div class="widgetcontainer-lab-status is-notsolved">  
            <span>LAB</span>  
            <p>not solved!</p>  
            <span class="lab-status-icon"></span>  
        </div>  
    </div>  
    </section>  
    </div>  
  
<div theme="blog">  
    <section class="mainContainer">  
        <div class="container is-page">  
            <header class="navigation-header">  
                <section class="top-links">  
                    <a href="#">User</p></p>  
                </section>  
            </header>  
            <header class="notification-header">  
                <header>  
                    <section class="blog-header">  
                        <h1>search results for 'alt:<#39;>'</h1>  
                    </section>  
                </header>  
            </header>  
        </div>  
    </section>
```

- Show the part of the source demonstrating that the reflected content also appears within an HTML tag's context

```
view-source:https://ac7e1fd71e9... +   
+ https://view-source:https://ac7e1fd71e9606cd80440b1b00110093.web-security-academy.net/?search=<ads6>  
  
<link href="/resources/css/labBlog.css" rel="stylesheet">  
<title>Reflected XSS into attribute with angle brackets HTML-encoded</title>  
</head>  
<body>  
    <script src="/resources/labHeader/js/labHeader.js"></script>  
  
    <div id="academyLabHeader">  
        <section class="academyLabBanner">  
            <div class="container">  
                <div class="logo"></div>  
                <div class="titleContainer">  
                    <h2>Reflected XSS into attribute with angle brackets HTML-encoded</h2>  
                    <a class="link-back" href="https://portswigger.net/web-security/cross-site-scripting/contextes/lab-attribute-angle-brackets-html-encoded">  
                        Back<br/><ads6>  
                    </a>  
                </div>  
                <div>  
                    <p>Not solved</p>  
                    <span class="lab-status-icon"></span>  
                </div>  
            </div>  
        </section>  
        <div theme="Blog">  
            <section class="maincontainer">  
                <div class="container is-page">  
                    <header class="navigation-header">  
                        <section class="top-links">  
                            <a href="/">Home</a>  
                        </section>  
                    </header>  
                    <header class="notification-header">  
                    </header>  
                    <section class="blog-header">  
                        <h1>search results for '<ads6>'</h1>  
                        <hr>  
                    </section>  
                    <section class="search">  
                        <form action="#" method="GET">  
                            <input type="text" placeholder="Search the blog..." name="search" value="<ads6>">  
                            <button type="submit" class="button">Search</button>  
                        </form>  
                    </section>  
                    <section class="blog-list">  
                        <div class="is-linkback">  
                            <a href="/>Back to Blog/></a>  
                        </div>  
                    </section>  
                </div>  
            </section>  
        </div>  
    </div>
```

- Take a screenshot of the output for your lab notebook

```

File Edit View Navigate Code Defactor Run Tools VCS Window Help WebSecurity - lab 3-1-7.py
File Edit Format View Help
odin id: ads6
Lab 3-1-7
File Structure
Project
WebSecurity E:\Projects\WebSecurity
  - c5d5-andrew-stevenson-ads6
    - hw1
    - hw2
    - notebooks
    - .gitignore
    - README.md
  - lab archive
    - lab 3-1-1.py
    - lab 3-1-2.py
    - lab 3-1-6.py
    - lab 3-1-7.py
    - tags.txt
  - External Libraries
  - Scratches and Consoles
Lab 3-1-7.py
1 import ...
2
3 site = 'ac7e1fd71e9606cd00440b1b00110093.web-security-academy.net'
4 s = requests.Session()
5
6 site_url = f'https://{site}/'
7 odin_id = 'ads6'
8 search_term = f'{odin_id}" foo="bar"'
9 search_url = f'{site_url}?search={search_term}'
10 resp = s.get(search_url)
11
12 for line in resp.text.split('\n'):
13     if 'input' in line:
14         print(line)

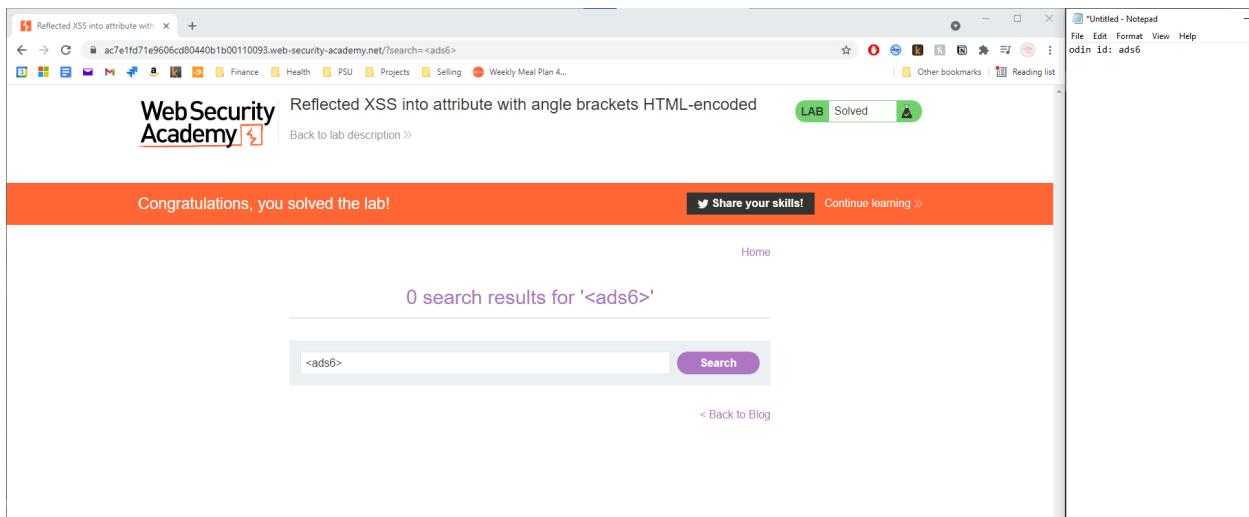
Run: keep-alive > Lab 3-1-7
C:/Users/ddste/miniconda3/envs/Snap2Route/python.exe "E:/Projects/WebSecurity/lab 3-1-7.py"
<input type="text" placeholder="Search the blog..." name=search value="ads6" foo="bar">
Process finished with exit code 0

```

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The adversary may bypass the html tag encoding by including a close quote mark in the querystring, allowing them to add arbitrary attributes to the input tag including an onmouseover attribute that will execute a cross site attack.

**Remediation:** HTML encode single and double quotation marks too.



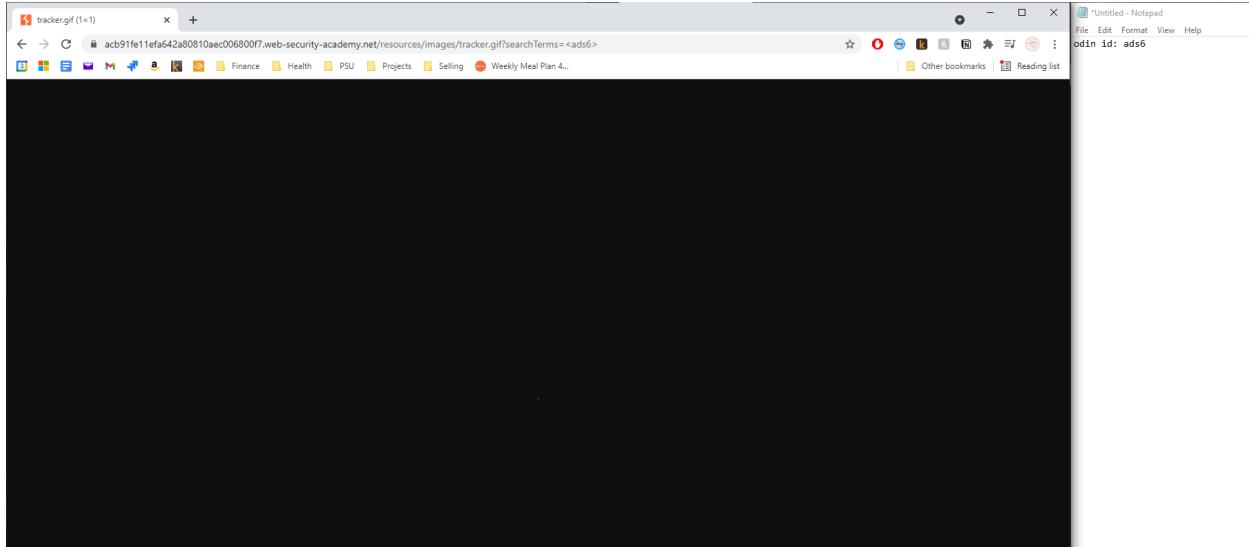
## 8. cross-site-scripting/reflected (5)

- Within the source, show the two contexts that the search term appears in.

```

 9 <script src="/resources/labHeader/js/labHeader.js"></script>
10 
11 <div id="academyLabHeader">
12   <section class="academyLabBanner">
13     <div class="container">
14       <div class="logo"></div>
15       <div class="title">Lab</div>
16       <div class="subTitle">XSS Lab</div>
17       <div class="link-back"><a href="https://portswigger.net/web-security/cross-site-scripting/context/lab-javascript-string-single-quote-backslash-escaped">Back</a><br>to lab</div>
18       <div class="description"><img alt="Layer 1 icon" data-bbox="198 188 218 208" style="vertical-align: middle;"/> Layer 1</div>
19       <div class="xss-line"><img alt="XSS Line icon" data-bbox="228 188 248 208" style="vertical-align: middle;"/> XSS</div>
20       <div class="xss-line"><img alt="XSS Line icon" data-bbox="258 188 278 208" style="vertical-align: middle;"/> XSS</div>
21       <div class="xss-line"><img alt="XSS Line icon" data-bbox="288 188 308 208" style="vertical-align: middle;"/> XSS</div>
22     </div>
23   </section>
24 </div>
25 
26 <div class="contentContainer-lab-status is-notsolved">
27   <div class="notSolved">
28     <p>Not solved!</p>
29     <span class="lab-status-icon"></span>
30   </div>
31 </div>
32 </div>
33 </div>
34 
35 <div theme="blog">
36   <section class="mainContainer">
37     <div class="container is-page">
38       <header class="notificationHeader">
39         <div class="links">
40           <a href="/">Home</a></p>
41         </div>
42       </header>
43       <header class="notificationHeader">
44         <div class="links">
45           <a href="#">Search results for &lt;ads6&gt;</a>
46         </div>
47       </header>
48       <form action="/" method="GET">
49         <input type="text" placeholder="Search the blog..." name=search>
50         <button type="submit" class="buttonSearch">Search</button>
51       </form>
52     </div>
53   </section>
54   <script>
55     var searchTerms = '<ads6>';
56     document.write('
59     <div class="is-linkback">
60       <a href="#">Back to Blog</a>
61     </div>
62   </section>
63 </div>
64 </div>
65 </div>
66 </body>
67 
```

- Open the tracker image in a new tab and show its URL that contains the search term

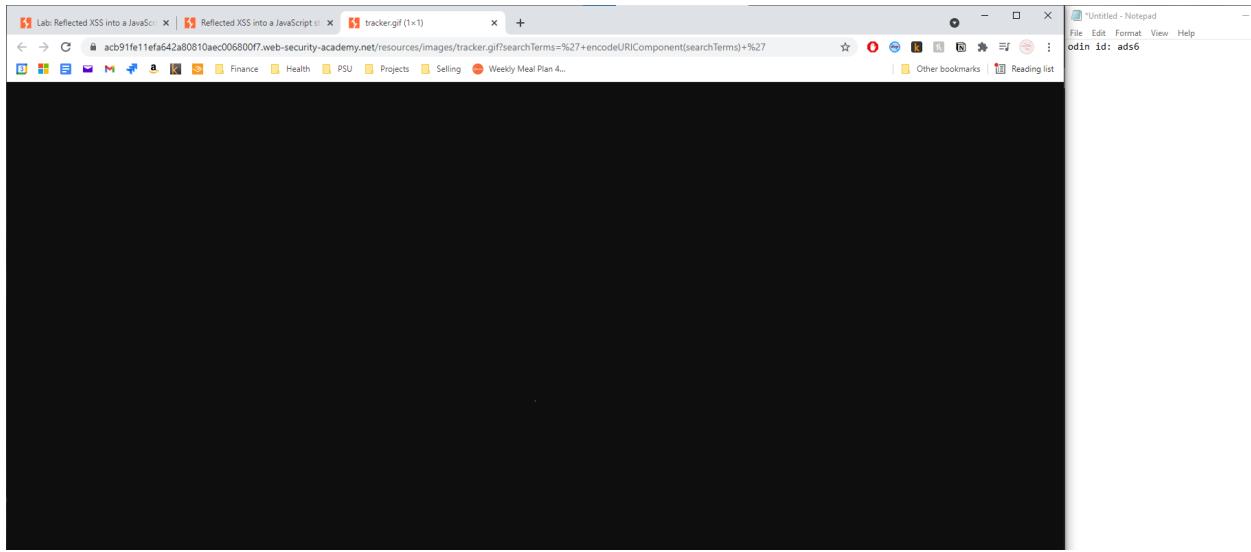


- Perform the search for `</script>` and explain why the string below the search form appears and where it came from

The '`;` `document.write(' ')`; below the search bar occurs because the inclusion of a `</script>` tag ends the javascript block and forces the remaining javascript to be included as content under the parent `<div>` tag.

- Show its URL and explain why it no longer contains the search term

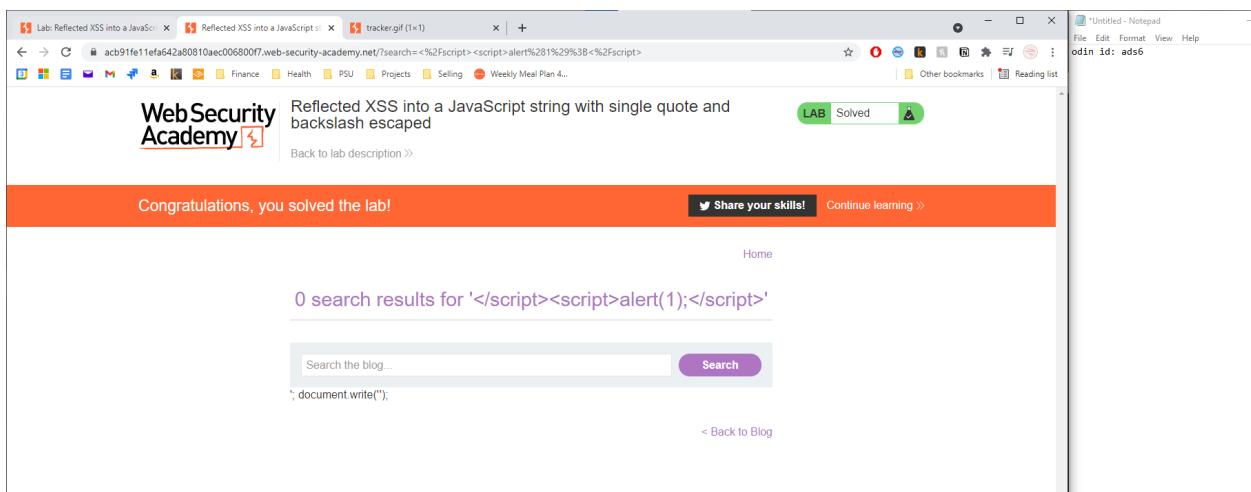
The url no longer contains the search term because the provided term "</script>" was interpreted as the ending of the <script> block and thus the actual <img> tag, including the image link, is nothing more than the literal javascript being parsed as HTML.



- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The search string appears unencoded within the <script> tag, allowing the adversary to terminate the script block by including a </script> string within the search string, followed by any arbitrary javascript encased within a new <script> block.

**Remediation:** HTML encode the search string before including it in the <script> block.



## 9. cross-site-scripting/reflected (6)

- Take a screenshot of the search term as it is reflected back in the Javascript code. What has been done to the search term as it appears in the Javascript code?

It appears the search terms are being html encoded.

The screenshot shows a browser window with the following details:

- URL:** ac511f591f56f9a8803a0ff1f0067006e.web-security-academy.net/?search=<%2Fscript><script>alert(1);</script>
- Title:** Reflected XSS into a JavaScript string with angle brackets HTML encoded
- Content:** "0 search results for '</script><script>alert(1);</script>'"
- Developer Tools:** The browser's developer tools are open, specifically the Elements and Scripts tabs. In the Scripts tab, the reflected search term is visible in the page source as part of a script block: `var searchTerms = '<%2Fscript><script>alert(1);</script>';`.

- Show the error that is returned and the line number it occurs on.

The screenshot shows a browser window with the following details:

- URL:** ac511f591f56f9a8803a0ff1f0067006e.web-security-academy.net/?search=%27
- Title:** Reflected XSS into a JavaScript string with angle brackets HTML encoded
- Content:** "5 search results for '''"
- Developer Tools:** The browser's developer tools are open, showing the console tab with an error message: "Uncaught SyntaxError: Invalid or unexpected token". The error occurred at line 30 of the page source.

- Why does the search term not appear in the <img> tag?

Because the javascript engine inserting the tag believes there is only an empty string, followed by the terminal semicolon and then a comment.

- What does // do in the Javascript code?

It is the comment marker.

- Take a screenshot of its URL demonstrating successful injection of the second Javascript assignment.

The screenshot shows a browser window with the following details:

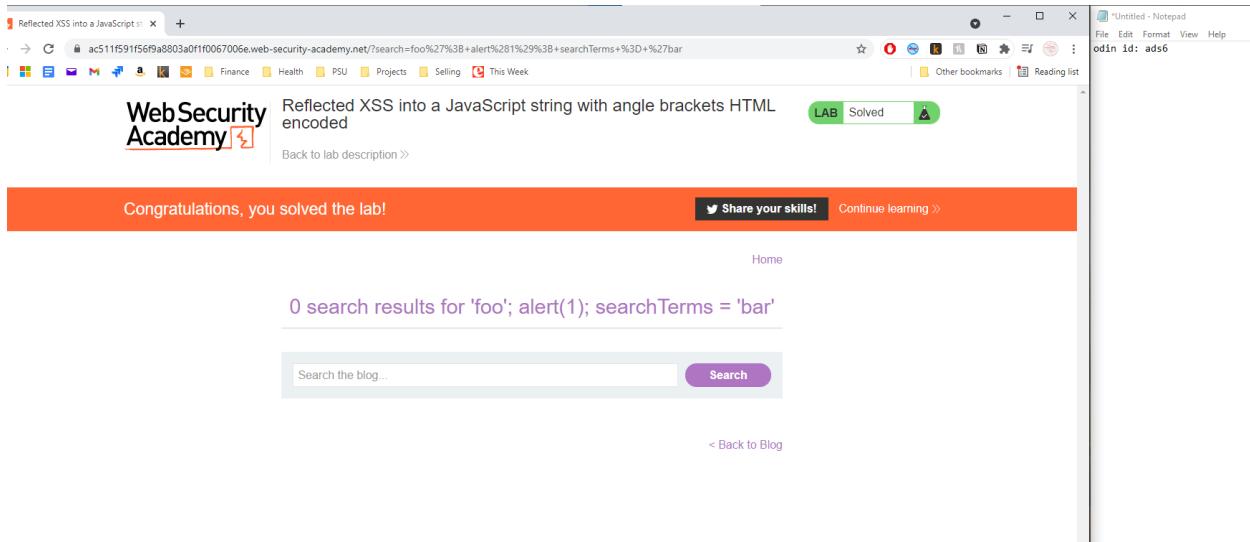
- Title Bar:** Reflected XSS into a JavaScript string with angle brackets HTML encoded
- URL:** ac511f591f569a8803a0f1f0067006e.web-security-academy.net/?search=foo%27%3B+searchTerms=%3D%27bar
- Content Area:** Displays "0 search results for 'foo'; searchTerms = 'bar'"
- Developer Tools:** The Elements tab is open, showing the DOM structure. A red circle highlights the injected script tag:

```
<script> == 50
    var searchTerms = 'foo'; searchTerms = 'bar';
    document.write('ing
        src="resources/images/tracker.gif?&searchTerms=' + encodeURIComponent(searchTerms) + '"';
    </script>
```
- Console Tab:** Shows the injected script being evaluated.
- Styles Tab:** Shows CSS rules from labsBlog.css, including a style for the .header class.

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The html encoding of the search terms that are subsequently included in the javascript do not properly escape the string terminating character () allowing the adversary to exploit the single quote to escape out of the string and thereby insert arbitrary javascript into the client code.

**Remediation:** Escape the string terminating character before including it in the javascript.



## 10. cross-site-scripting/reflected (7)

- Explain why this error has happened and what needs to be done to fix it

The encoding function blindly escaped the input with a single backslash, yielding the javascript code

```
var searchTerms = '\\\";
```

Which is the interpreter sees as:

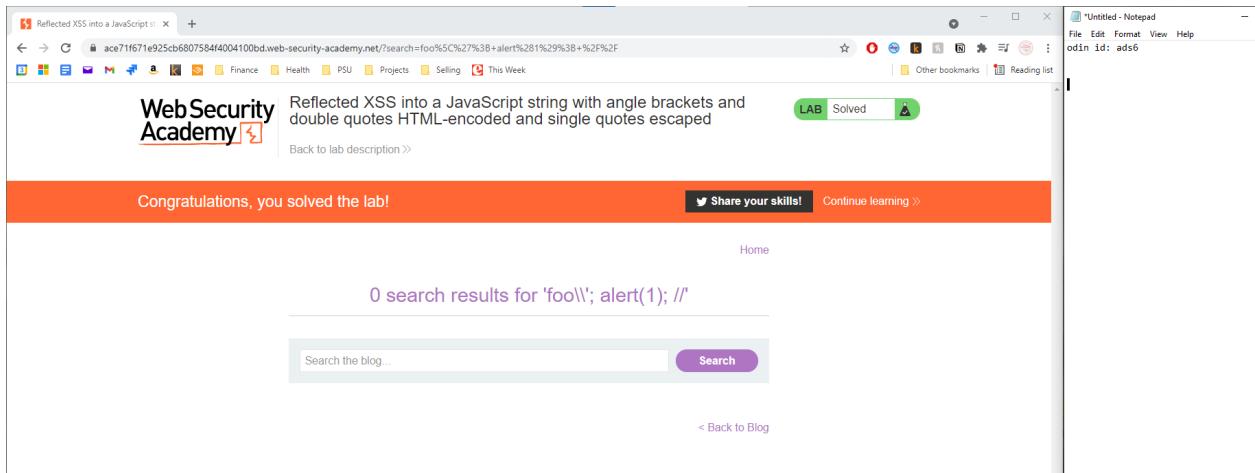
```
var searchTerms = '\\\'      ' // The final single quote is extra
```

Because the \\ is just the escaped code for \. To fix it, the escape char \ must also be escaped.

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The single-quote escape function does not account for the presence of the escape character, allowing the adversary to escape from the string using the \' combination similarly to before.

**Remediation:** Escape the backslash too.



## 11. cross-site-scripting/reflected (8)

- Show this line of Javascript

```
document.getElementById('searchMessage').innerText = message;
```

- Show the line that defines the template literal.

```
var message = `0 search results for
'ads6'`;
```

- Explain the results

The line:

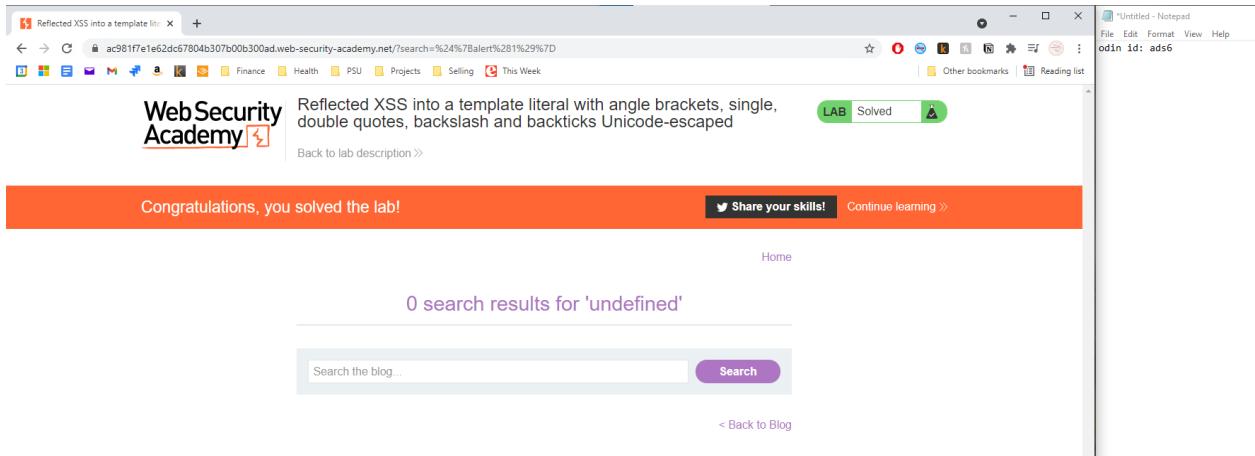
```
var message = `0 search results for 'cs${490+5}'`;
```

Tells the interpreter to resolve the expression  $490 + 5$  and put that value into the string before proceeding to set the text element's value.

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The unfortunate use of a template literal, combined with the failure to escape the characters \$, {, and } permits the adversary to inject executable code within the template literal itself.

**Remediation:** Don't use a template literal in this case; and, more generally, escape the above-mentioned characters.



## 12. cross-site-scripting/dom-based (1)

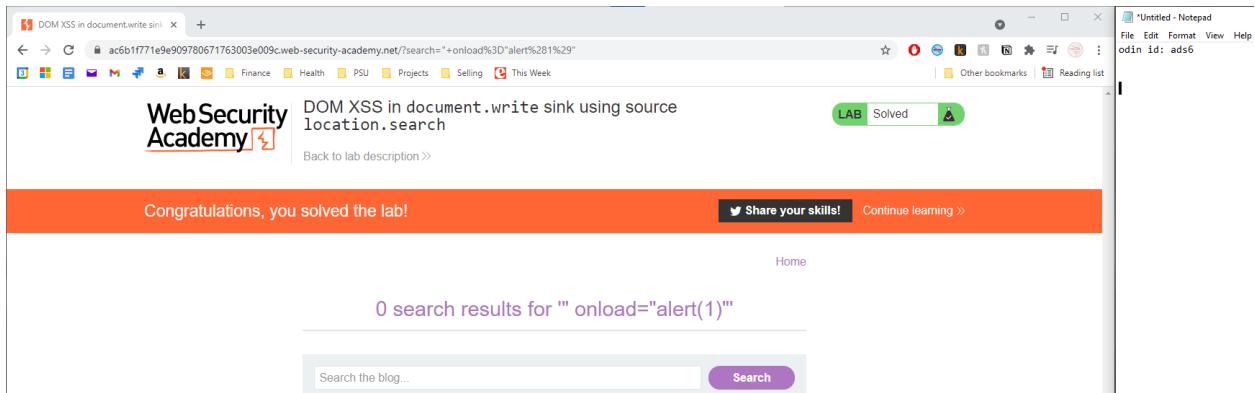
- Take a screenshot of the `<img>` tag in Developer Tools and show that its syntax has been broken.

```
▶ <script>..</script>
  
▶ <section class="blog-list">...</section>
```

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The failure to html encode the " character allows the adversary to escape the string within the DOM and insert arbitrary html code (including javascript.)

**Remediation:** HTML encode the " character.



## 13. cross-site-scripting/dom-based (2)

- How does the `store` variable get set? Can one always assume that it is one of the values in the `stores` list?

```
var store = (new URLSearchParams(window.location.search)).get('storeId');
```

Since the store value is coming from the url search string, one cannot assume it will be one of the values in the stores list.

- What is the purpose of the first `if` statement? What is the purpose of the second `if` statement?

The purpose of the first if is to make sure the default option is the same as what was specified in the search string. The purpose of the second if statement is to make sure that default choice does not appear twice in the select menu.

- Take a screenshot showing that you have successfully injected a bogus store that is not one of the initial ones in the stores list.

Get this ultra-fun pair of guns today and have hours of fun with your friends.

ads6

**Check stock**

< Return to list

New Trust Token pane

New Trust Token pane in the App

Emulate the CSS color-gamut

Emulate color-gamut to test diffe

Format strings as (valid) JavaSc

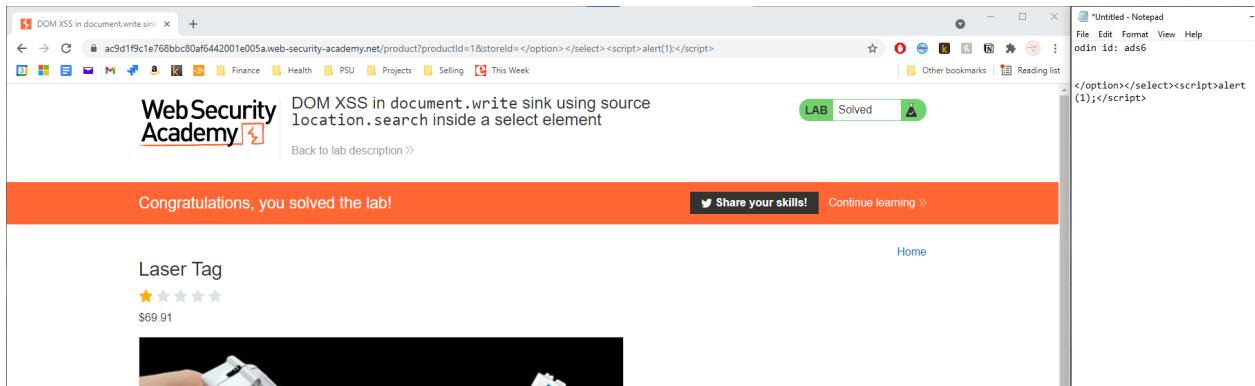
Display string with escaped doub

Improved PWA tooling

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The search string is not html encoded before inserting it into the `<option>` block, allowing the adversary to insert arbitrary html (and by extension, javascript) into the page.

**Remediation:** HTML encode the search string before using it in the HTML.

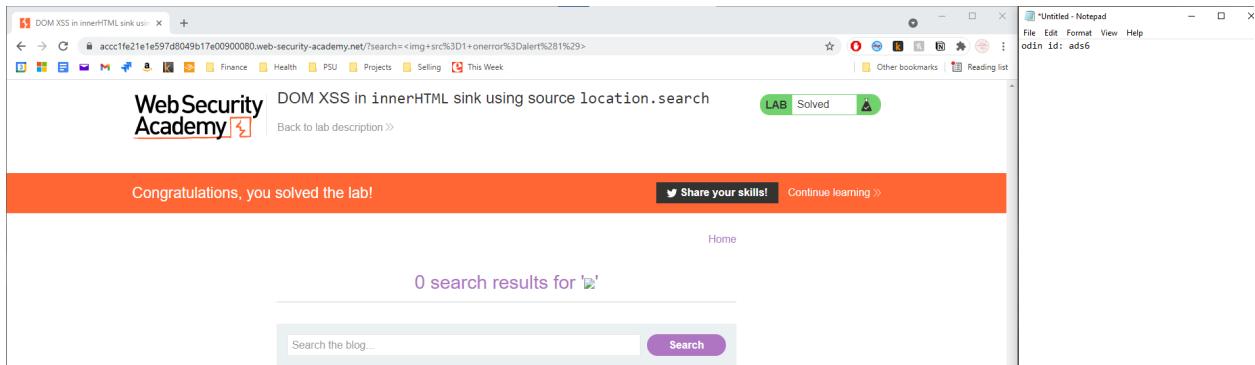


## 14. cross-site-scripting/dom-based (3)

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The search string is not html encoded before inserting it via the innerHTML call, allowing the adversary to insert arbitrary html (and by extension, javascript) into the page.

**Remediation:** HTML encode the search string before using it in the HTML.



## 15. cross-site-scripting/dom-based (4)

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** Because the contents of the querystring are used unsanitized within the backlink, the adversary may inject arbitrary javascript into the link by changing the querystring.

**Remediation:** HTML encode the search string before using it in the backlink.

A screenshot of a web browser window showing the 'Web Security Academy' lab titled 'DOM XSS in jQuery anchor href attribute sink using location.search source'. The status bar indicates 'LAB Solved'. Below the title, it says 'Congratulations, you solved the lab!' and has 'Share your skills!' and 'Continue learning >' buttons. A Notepad window on the right shows the ID 'odin id: ads6'.

## 16. cross-site-scripting/dom-based (5)

- Show a screenshot of the URI path of the XHR request

A screenshot of a browser window showing a reflected DOM XSS attack. The URL is https://ac3b1ffd1f68c36580b31d/web-security-academy.net/search-results?search=Lies%2C+Lies+%26+More+Lies. The page content includes JSON data: {"searchTerm": "Lies, Lies & More Lies", "results": [{"id": 1, "title": "Lies, Lies & More Lies", "image": "content/blog/posts/13.jpg", "summary": "I remember the first time I told a lie. That's not to say I didn't do it before then, I just don't remember. I was nine years old and at my third school already. Fitting into already established friendship groups..."}]}

- Show a screenshot of the JSON response that echoes the search term

A screenshot of a browser window showing a reflected DOM XSS attack. The URL is https://ac3b1ffd1f68c36580b31d/web-security-academy.net/search-results?search=Lies%2C+Lies+%26+More+Lies. The JSON response is highlighted in yellow: {"searchTerm": "Lies, Lies & More Lies", "results": [{"id": 1, "title": "Lies, Lies & More Lies", "image": "content/blog/posts/13.jpg", "summary": "I remember the first time I told a lie. That's not to say I didn't do it before then, I just don't remember. I was nine years old and at my third school already. Fitting into already established friendship groups..."}]}

- Take a screenshot of the vulnerable line of code

```

Reflected DOM XSS
ac3b1ff1f68c36580b318d40002007e.web-security-academy.net/?search=Lies%2C+Lies+%26+More+Lies
Elements Console Sources Network Performance Memory Application Security Lighthouse AdBlock
Reflected DOM XSS
Back to lab description >
1 search result
Search the blog...
Lies, Lies & More Lies

```

```

function search(results) {
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var searchResultsObj = JSON.parse(this.responseText);
      displaySearchResults(searchResultsObj);
    }
  };
  xhr.open("GET", path + window.location.search);
  xhr.send();
}

function displaySearchResults(searchResults) {
  var blogHeader = document.getElementsByClassName("blog-header")[0];
  var blogList = document.getElementsByClassName("blog-list")[0];
  var searchResults = searchResults["results"];
  var searchResultsObj = searchResultsObj["results"];

  var h1 = document.createElement("h1");
  h1.innerHTML = searchResults.length + " search results for '" + searchTerm + "'";
  blogHeader.appendChild(h1);
  var ul = document.createElement("ul");
  for (var i = 0; i < searchResults.length; ++i) {
    var searchResult = searchResults[i];
    if (searchResult["id"]) {
      var li = document.createElement("li");
      li.innerHTML = searchResult["id"] + " - " + searchResult["title"];
      ul.appendChild(li);
    }
  }
  blogList.appendChild(ul);
}

```

- What has been done to the delimiter to prevent syntax from being broken?

The double quote has been escaped with a backslash.

- Take a screenshot of the error that has been produced in the console. Examine the JSON response to verify that you have broken syntax.

```

Reflected DOM XSS [LAB Not solved]
ac3b1ff1f68c36580b318d40002007e.web-security-academy.net/?search=%5C%
Elements Console Sources Network Performance Memory Application Security Lighthouse AdBlock
Reflected DOM XSS
Back to lab description >
Search the blog...

```

Name	Status	Type	Initiator	Size	Time	Waterfall
academyLabHeader	101	websocket	labHeader.js:2	0 B	Pending	
search: %5C%22	200	document	Other	1.0 kB	573 ms	
tableHeader.js	200	script	[search]*	(memory cache)	0 ms	
searchResults.js	200	script	[search]*	(memory cache)	0 ms	
academyLabHeader.css	200	stylesheet	[search]*	(disk cache)	3 ms	
labBlog.css	200	stylesheet	[search]*	(disk cache)	3 ms	
search-results?search=%5C%22	200	xhr	searchResults.js:10	203 B	558 ms	
logoAcadem.svg	200	svg+xml	academ/labHeader.css	(memory cache)	0 ms	
ps-lab-notsolved.svg	200	svg+xml	academ/labHeader.css	(memory cache)	0 ms	

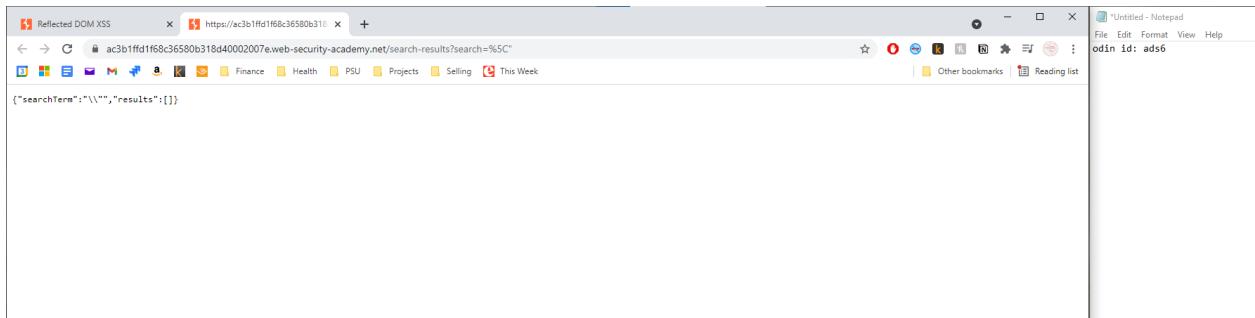
9 requests | 1.2 kB transferred | 46.4 kB resources | Finish: 1.25 s | DOMContentLoaded: 696 ms | Load: 702 ms

```

Console What's New
Default levels ▾ No Issues
 Hide network
 Preserve log
 Selected context only
 Group similar messages in console
 Log XMLHttpRequests
 Eager evaluation
 Autocomplete from history
 Evaluate triggers user activation

Navigated to https://ac3b1ff1f68c36580b318d40002007e.web-security-academy.net/?search=Lies%2C+Lies+%26+More+Lies
> XHR Finished loading: GET https://ac3b1ff1f68c36580b318d40002007e.web-security-academy.net/search-results?search=Lies%2C+Lies+%26+More+Lies
Navigated to https://ac3b1ff1f68c36580b318d40002007e.web-security-academy.net/?search=%22+hi+there
> XHR Finished loading: GET https://ac3b1ff1f68c36580b318d40002007e.web-security-academy.net/search-results?search=%22+hi+there
Navigated to https://ac3b1ff1f68c36580b318d40002007e.web-security-academy.net/?search=SC%22
 Unexpected token: ' at XMLHttpRequest.onreadystatechange (searchResults.js:1)
> XHR Finished loading: GET https://ac3b1ff1f68c36580b318d40002007e.web-security-academy.net/search-results?search=SC%22
searchResults.js:1:10

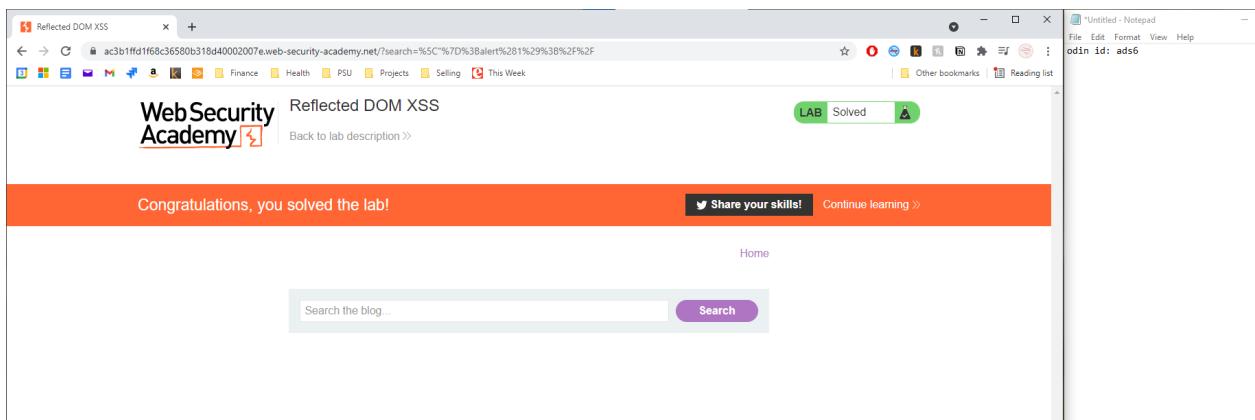
```



- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** Because the query string contents are not adequately sanitized before incorporating them into the json document, the adversary may force the client to execute arbitrary javascript by incorporating it into the search string.

**Remediation:** Properly escape backslashes as well as double quotes.

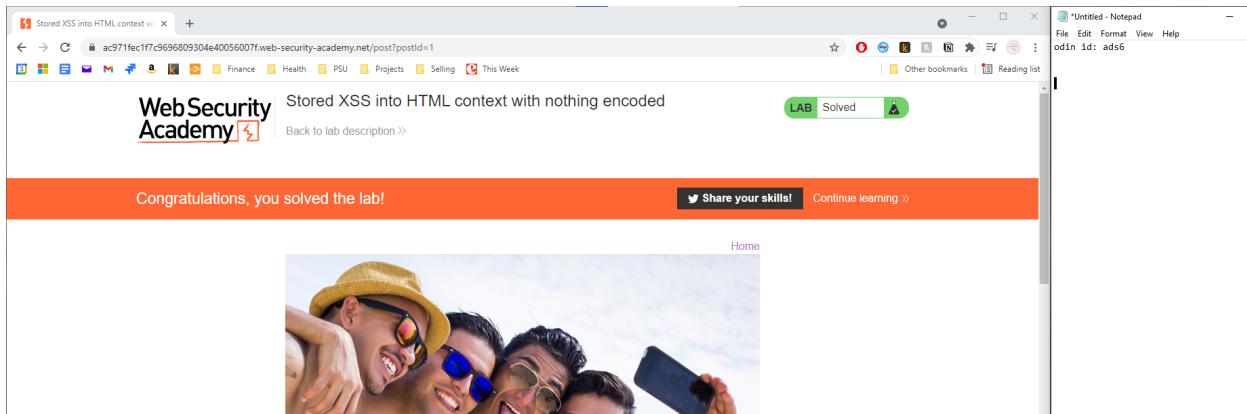


## 18. cross-site-scripting/stored (1)

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** Because the comment is not sanitized before incorporating into the page's html, the adversary may inject arbitrary javascript into the page via the comment function.

**Remediation:** HTML encode the comment before incorporating it into the page.



## 19. cross-site-scripting/stored (2)

- Take a screenshot showing that you have successfully added the `OdinId` attribute to the author's website link.

- Show the stored `<a>` tag you have used that pops up this `alert()`

The screenshot shows a web browser window with a comment section from a blog post. A user named 'badguy' has posted a comment containing a URL: <https://pdx.edu>. The browser's developer tools are open, specifically the Elements and Styles panels. In the Styles panel, a CSS rule for '.comment' is visible, which includes a 'background-image' property set to 'url(/resources/images/avatarDefault.svg)'. The injected URL is part of the href attribute of this element, demonstrating a stored XSS vulnerability.

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** Because the double quotes are not properly incorporated before incorporating the website field into the <a> tag in the HTML, the adversary may break out of the context of the a-tag and inject arbitrary javascript which any browser loading the page thereafter will execute.

**Remediation:** HTML encode the url before incorporating it into the page.

The screenshot shows the WebSecurityAcademy platform after solving a lab. The message 'Stored XSS into anchor href attribute with double quotes HTML-encoded' is displayed. The browser's developer tools are open, showing the DOM structure and the CSS styles applied to the elements, including the injected script.

## 20. cross-site-scripting/stored (3)

- Replace the URL with `https://pdx.edu'` (e.g. the original URL with a single-quote added to break Javascript syntax). Execute the code and show a screenshot of the error that is returned.

The screenshot shows a Microsoft Edge browser window with the following details:

- Title Bar:** Lab: Stored XSS into onclick event | Stored XSS into onclick event with...
- Address Bar:** ac561fc81ec47a9880d6457100c00031.web-security-academy.net/post?postId=1
- Page Content:** A list of comments from users Anna Nutherford, Chris Mass, John Top, Nick O'Time, and innocuous.
- Developer Tools (Console Tab):** Shows the exploit code being injected and the resulting error message: "Uncaught SyntaxError: missing ) after argument list".

- Explain what happens when the URL is replaced with `https://pdx.edu');//`

The subsequent javascript will be ignored, as it is all considered to be on a single line and is preceded by the comment marker.

- Finally, using this URL, insert an `alert(1);` into it and take a screenshot of the results in the console including the pop-up

The screenshot shows a browser window with two tabs open. The active tab is titled "Stored XSS into onclick event" and contains the URL <http://ac561fc81ec47a9880d6457100c00031.web-security-academy.net/post?postId=1>. The page content includes several user comments and a message box from the browser developer tools.

Comments:

- Anna Nutherford | 19 April 2021  
I wantd to pas this of as my own but frends poynuted I cant spel?
- Chris Mass | 20 April 2021  
This has been copied straight from my blog. Reported!
- John Top | 27 April 2021  
Does your Mother know you are out?
- Nick O'Time | 02 May 2021  
I cannot believe how many people disagree on the internet!
- innocuous | 05 May 2021  
I'm your huckleberry

Message box from developer tools:

```
...1ec47a9880d6457100c00031.web-security-academy.net says
1
OK
```

Developer tools console output:

```
Selected context only
Group similar messages in console
Navigated to https://ac561fc81ec47a9880d6457100c00031.web-security-academy.net/post?postId=2
> var tracker=[track()];tracker.track('https://pox.edu/');
< undefined
> var tracker=[track()];tracker.track('https://pox.edu/');
Uncaught SyntaxError: missing ) after argument list
> var tracker=[track()];tracker.track('https://pox.edu/'; alert(1); '');
Uncaught SyntaxError: missing ) after argument list
> var tracker=[track()];tracker.track('https://pox.edu/'); alert(1); '';
>
```

- Explain why they differ

The double quote is not being escaped with a backslash.

- Are the results the same?

Yes.

- Take a screenshot of the error message in the console. Is it similar to one that you have seen when crafting an exploit?

Yes, it appears to be identical.

```

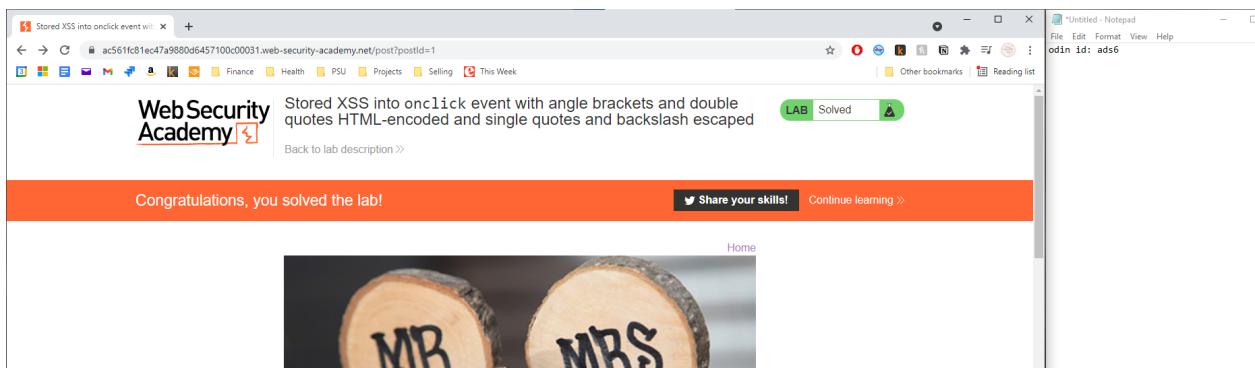
var tracker=track();};tracker.track('https://pdx.edu/');alert(1); //';

```

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The contents of the website field are HTML decoded after the javascript encoding, allowing the adversary to slip in an HTML encoded single quote that will get through the filter and permit them to escape from the string within the context of the javascript. Thus, the adversary may inject arbitrary code to run as part of the web page.

**Remediation:** HTML decode the field before escaping it for the javascript.



## 23. cross-site-scripting/dom-based (6)

- Take a screenshot of the result and explain the issue. What other [built-in String method](#) could be used instead to fix this particular issue?

The issue is that the replace() method only replaces the first instance of the value (unless the value is a regex.) The (more) correct method would have been replaceAll().

They do tell us once you have chosen your preferred fragrance you won't be able to change the settings, so it is essential you buy in bulk to ensure you don't ever run out. A little flick the head and the aroma will be detected by the inbuilt sensors, and you are up and running.

We tried this in our office to see how efficient this new smart technology really is. There were some glitches which the company informs us they are working on. It turns out that most companies favor one mainstream brand, and could unlock most of the cells in the room. We also found we had to do a bit more than 'just a little flick of the head' to release enough scent for the pick up. This, in turn, gave most of us a banging headache for the rest of the day.

We agreed we couldn't kick this off, a bit of a novelty that would soon wear off when the frustration kicks in. If we think it's bad enough walking into a room full of people looking down at their phones, imagine a room full of headbangers. Nice try guys, keep up good work, innovation is always refreshing.

**Comments**

Carrie On | 12-04-2021  
An enjoyable read while waiting for the police to arrive. Tell me, what's the WiFi like in pr...

```
<ads6><img src=1 onerror=alert(1).replace('<','&lt;').replace('>','&gt;');
```

- Which three fields of JSON are vulnerable to a cross-site scripting attack?

Website, body, author

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The inappropriate use of the javascript replace() function, when replaceAll() would have been more correct, allows the adversary to bypass the escapeHTML function by prepending their attack string with a simple <>.

**Remediation:** Use replaceAll() instead of replace().

Congratulations, you solved the lab!

Share your skills! Continue learning >

Home

```
<ads6><img src=1 onerror=alert(1).replace('<','&lt;').replace('>','&gt;');
```

## 25. cross-site-scripting/exploiting (1)

- Take a screenshot of the headers showing all of the form data for your POST including your own exfiltrated cookie sent as a comment.

The screenshot shows the Network tab of the Chrome DevTools. A single request is listed:

- Name:** comment
- Headers:**
  - Content-Length: 0
  - Location: /post/comment/confirmation?postId=1
  - X-XSS-Protection: 0
- Request Headers:**
  - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9
  - Accept-Encoding: gzip, deflate, br
  - Accept-Language: en-US,en;q=0.9,ja;q=0.8
  - Cache-Control: max-age=0
  - Connection: keep-alive
  - Content-Length: 160
  - Content-Type: application/x-www-form-urlencoded
  - Cookie: session=01xRkXlg1fg613n2n1FeEtKxx@tt
  - Host: ac9a1f041e2ede56804a0a5c00bc001f.web-security-academy.net
  - Origin: https://ac9a1f041e2ede56804a0a5c00bc001f.web-security-academy.net
  - Referer: https://ac9a1f041e2ede56804a0a5c00bc001f.web-security-academy.net/post/postId=1
  - sec-ch-ua: " Not A Brand";v="99", "Chromium";v="98", "Google Chrome";v="98"
  - sec-ch-ua-mobile: ?0
  - Sec-Fetch-Dest: document
  - Sec-Fetch-Mode: navigate
  - Sec-Fetch-Site: same-origin
  - Upgrade-Insecure-Requests: 1
  - User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4430.93 Safari/537.36
- Form Data:**
  - comment: I am a hacker
  - postId: 1

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The above vulnerability in comments submission feature allows the adversary to exfiltrate the secret cookie from other users, either by using another site to "silently" register the user's private data, or (less subtly) to automatically add other user's cookies to the HTML.

**Remediation:** HTML encode all the comment fields before incorporating them into the HTML.

The screenshot shows the browser after solving the lab. The page content includes:

- A success message: "Congratulations, you solved the lab!"
- A "Share your skills!" button.
- A "Continue learning >>" link.
- The status bar at the top right indicates "LAB Solved".

## 28. cross-site-scripting/exploiting (2)

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The failure to properly sanitize the comments before adding them to the HTML allows the adversary to inject a form into the site that all users will load; since the administrator automatically fills out this form, the adversary may also automate the posting of these credentials as a second comment via the onchange event handler.

**Remediation:** HTML encode all the comment fields before incorporating them into the HTML.

## Lab 3.2

### 1. cors (1)

- Show a screenshot of the CORS header that enables credentials to be sent to get the key.

- Show a screenshot of the headers in the response to see that the site simply reflects the header in Access-Control-Allow-Origin::

A screenshot of a terminal window titled "WebSecurity - Lab 3-2-1". It shows a Python script named "lab 3-2-1.py" being run. The script contains code to log in to a service using credentials ("wiener", "peter") and then extract session information. The output shows the session details and an API key ("shV4yzUL5IW1NrHBYlvbj8x76rzrbu5").

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help WebSecurity - lab 3-2-1.py
WebSecurity E:\Projects\WebSecurity\lab 3-2-1.py
Project
  WebSecurity
    -> c459-andrew-stevenson-ads6
      -> hw1
      -> hw2
      -> notebooks
        -> ptignore
          README.md
    -> lab archive
      -> lab 3-1.py
      -> lab 3-1-1.py
      -> lab 3-1-6.py
      -> lab 3-1-7.py
    Run: keep-alive x lab 3-2-1 x
C:\Users\ddste\miniconda3\envs\Snap2Route\python.exe "E:/Projects/WebSecurity/Lab 3-2-1.py"
{'Access-Control-Allow-Origin': 'https://ads6.com', 'Access-Control-Allow-Credentials': 'true', 'Content-Type': 'application/json; charset=utf-8', 'X-XSS-Protection': '0', 'Content-Encoding': 'gzip', 'Connection': 'close', 'Content-Length': '181'}
{
  "username": "wiener",
  "email": "",
  "apikey": "shV4yzUL5IW1NrHBYlvbj8x76rzrbu5",
  "sessions": [
    "FOEGZZ8XR3VS1dhwwkp0XRFhWh6VN6b",
    "gS4jcg230bLEs03pbfiyAp006sCfXPiq"
  ]
}
Process finished with exit code 0
```

\* Take a screenshot of the API key and include it in your lab notebook.

A screenshot of a browser window showing the "My Account" page of the WebSecurity Academy website. The page displays the user's username ("wiener") and API key ("shV4yzUL5IW1NrHBYlvbj8x76rzrbu5"). There is a form for updating the email address, with a placeholder "Email" and a "Update email" button.

WebSecurity Academy

CORS vulnerability with basic origin reflection

Back to lab home | Go to exploit server | Submit solution | Back to lab description >

Your username is: wiener

Your API Key is: shV4yzUL5IW1NrHBYlvbj8x76rzrbu5

Email

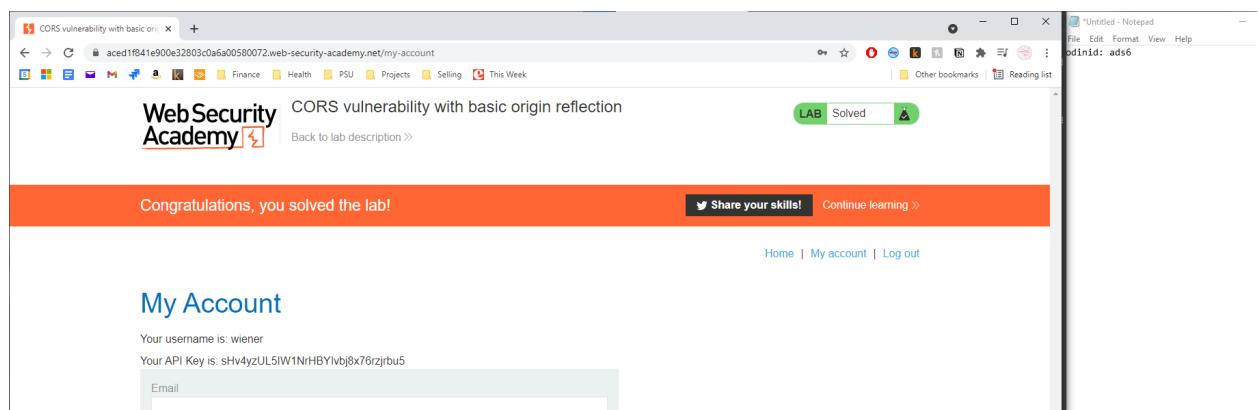
Update email

- Take a screenshot of the API key as it appears in the browser window.

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The server reflects back the request's Origin header in its Access-Control-Allow-Origin response header. Thus, the adversary may collect stored user information when the victim visits a site the adversary controls after logging in to the target site.

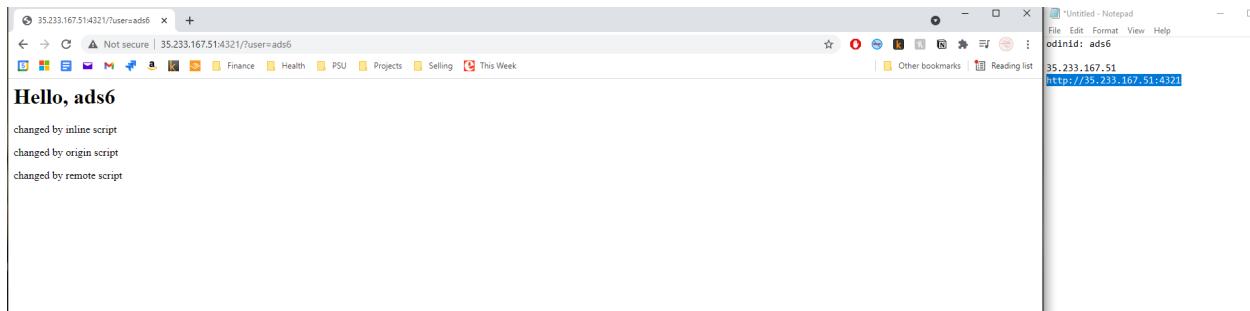
**Remediation:** Properly set the Access-Control-Allow-Origin header to restrict which sites may view the user information.



## 8. Content-Security-Policy examples

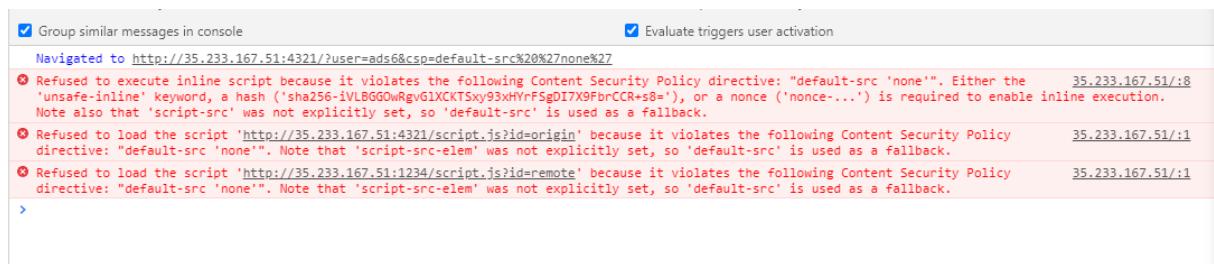
## Example #1

- Take a screenshot of the page result that includes the URL in the browser

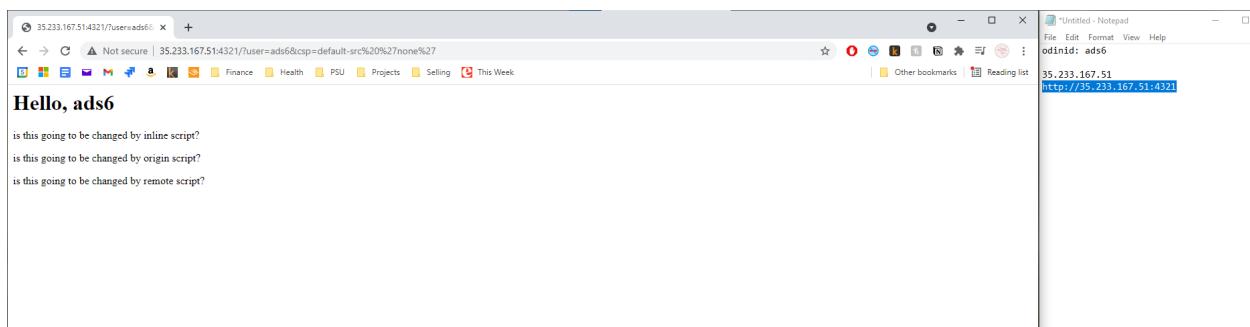


## Example #2

- Take a screenshot of the console output showing all scripts blocked



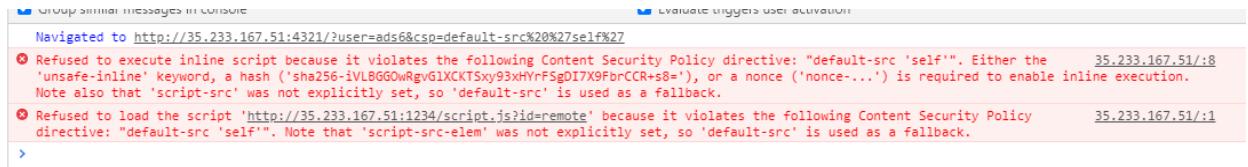
- Take a screenshot of the page result that includes the URL in the browser



## Example #3

- Take a screenshot of the console output showing the scripts that have been blocked. Explain why these results differ from the previous example.

The Content-Security-Policy header was set to self, so scripts from the same location as the file are allowed.

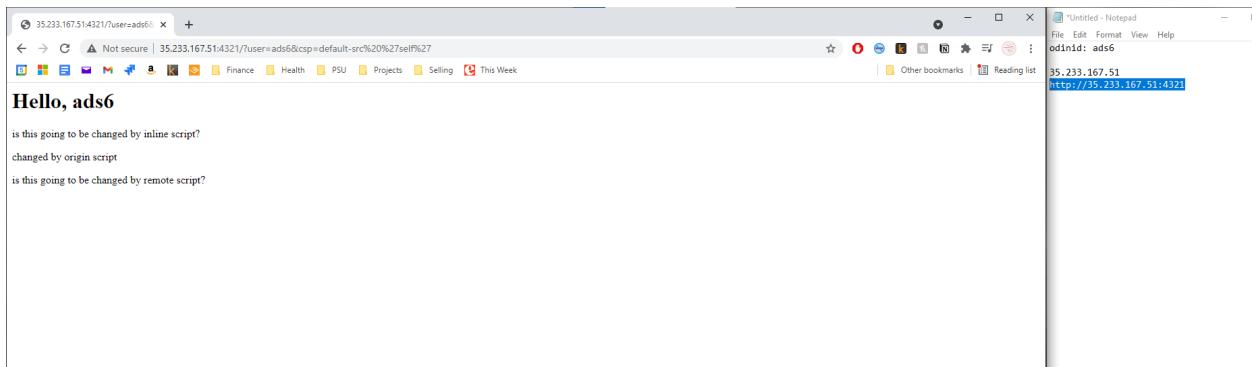


Navigated to <http://35.233.167.51:4321/?user=ads6&csp=default-src%20%7self%27>

Refused to execute inline script because it violates the following Content Security Policy directive: "default-src 'self'". Either the 'unsafe-inline' keyword, a hash ('sha256-iVLBGG0wRgvG1XCKTSxy93xHyrfSgDI7x9FbrCCR+s8=')<sup>35.233.167.51/:8</sup>, or a nonce ('nonce-...') is required to enable inline execution.  
Note also that 'script-src' was not explicitly set, so 'default-src' is used as a fallback.

Refused to load the script '<http://35.233.167.51:1234/script.js?id=remote>' because it violates the following Content Security Policy directive: "default-src 'self'". Note that 'script-src-elem' was not explicitly set, so 'default-src' is used as a fallback.<sup>35.233.167.51/:1</sup>

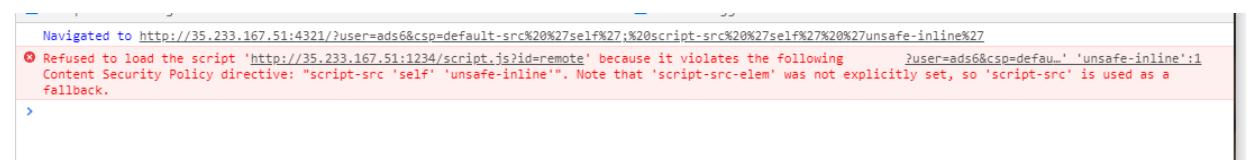
- Take a screenshot of the page result that includes the URL in the browser



## Example #4

- Take a screenshot of the console output showing the scripts that have been blocked. Explain why these results differ from the previous example based on the additional options given.

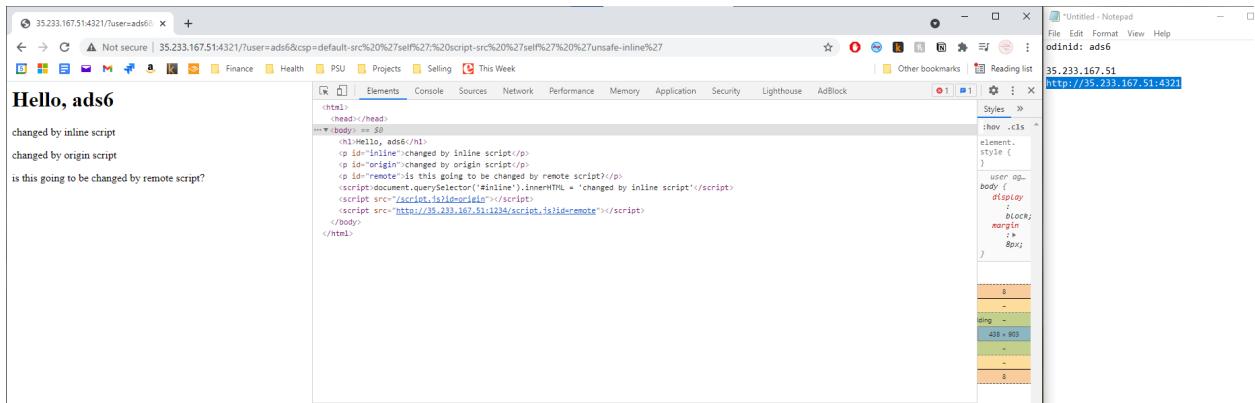
The inclusion of the unsafe-inline for script-src suffices to permit the inline script to execute, as was hinted by the error message for the inline node in the above examples.



Navigated to <http://35.233.167.51:4321/?user=ads6&csp=default-src%20%7self%27;%20script-src%20%7self%27%20%27unsafe-inline%27>

Refused to load the script '<http://35.233.167.51:1234/script.js?id=remote>' because it violates the following Content Security Policy directive: "script-src 'self' 'unsafe-inline'". Note that 'script-src-elem' was not explicitly set, so 'script-src' is used as a fallback.<sup>?user=ads6&csp=default-src%20%7self%27;%20script-src%20%7self%27%20%27unsafe-inline%27:1</sup>

- Take a screenshot of the page result that includes the URL in the browser



## Lab 3.3

### 1. csrf (1)

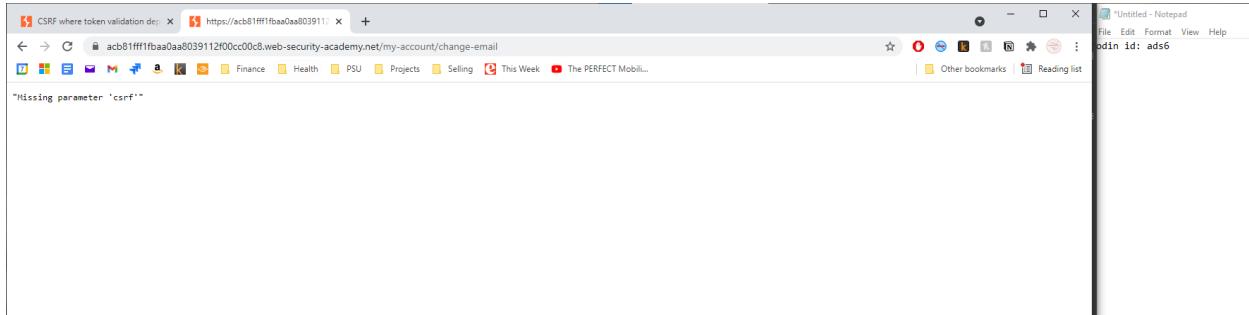
- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The submission url takes no precautions against unauthorized requests, allowing the adversary to submit any email change requests they desire.

**Remediation:** Use a CSRF token to ensure the request has come from the associated, validated, user.

### 3. csrf (2)

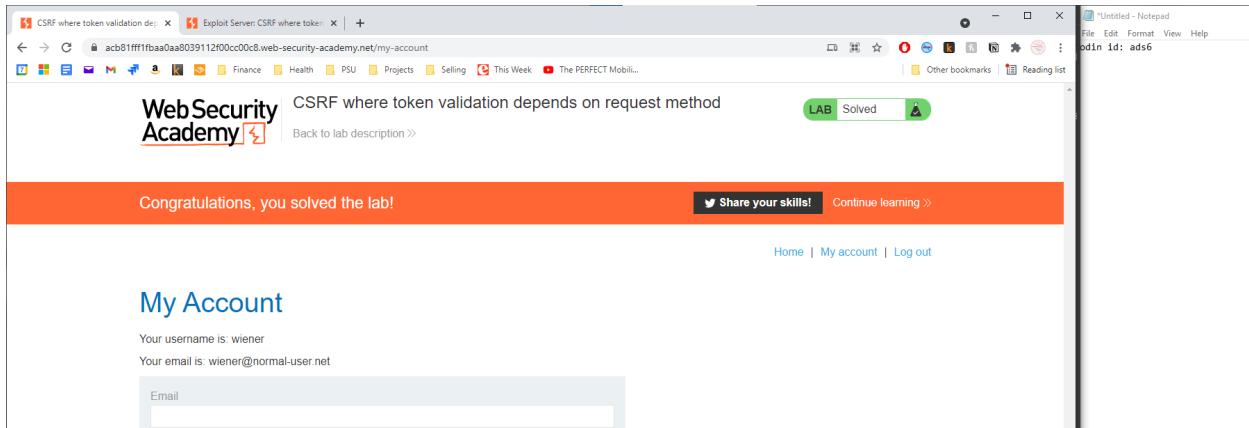
- Take a screenshot of the result showing that the exploit has failed



- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** Although a CSRF token is now in place, the GET route is not protected by the token, permitting the adversary to make the same attack as before.

**Remediation:** Protect the GET route with a CSRF token.

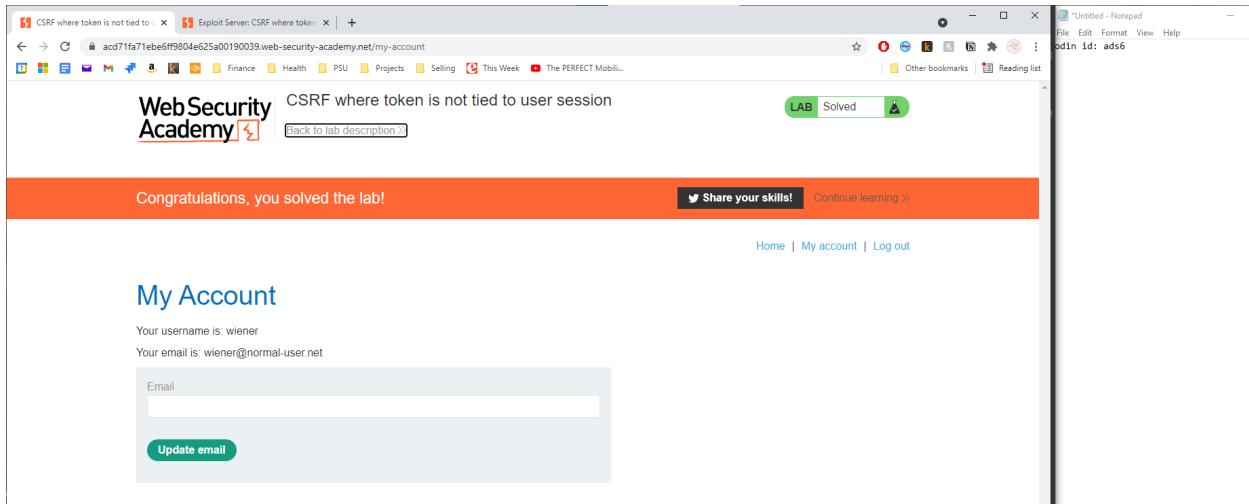


## 4. csrf (3)

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** Although a CSRF token is now in place, it is not tied to a user session, allowing the adversary to retrieve a token and embed it into their attack site form—which enables them to repeat the same attack as above.

**Remediation:** Ensure that the CSRF token is associated with a user session, and verify that connection before accepting the form submission.



## 5. csrf (4)

- Take a screenshot of the entire HTTP response header that includes your OdinId

Request URL: https://ac8c1f271e4ef00e8026217b003d0094.web-security-academy.net/?search=ads6  
Request Method: GET  
Status Code: 200 OK  
Remote Address: 18.200.141.238:443  
Referrer Policy: strict-origin-when-cross-origin

Response Headers View parsed

HTTP/1.1 200 OK  
Set-Cookie: LastSearchTerm=ads6; Secure; HttpOnly  
Content-Type: text/html; charset=utf-8  
X-XSS-Protection: 0  
Content-Encoding: gzip  
Connection: close  
Content-Length: 1015

Request Headers View source

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange

- How many cookies are returned? What are their names?

One Cookie: LastSearchTerm.

- How have foo and bar been interpreted?

As a new line in the http response header.

- How many cookies have been returned?

Two Cookies.

- Explain the results. Give one reason why a developer would choose to implement CSRF protection in this manner

The server simply compares the csrf in the form to the csrf in the cookie and checks if they're the same. Probably implemented this way because it's (1) easier to program, (2) allows for multiple simultaneous tabs with the same user; and (3) requires no back-end storing of csrf.

- What status code is returned? What is the value of the csrf field in the HTML form that is given back as a response?

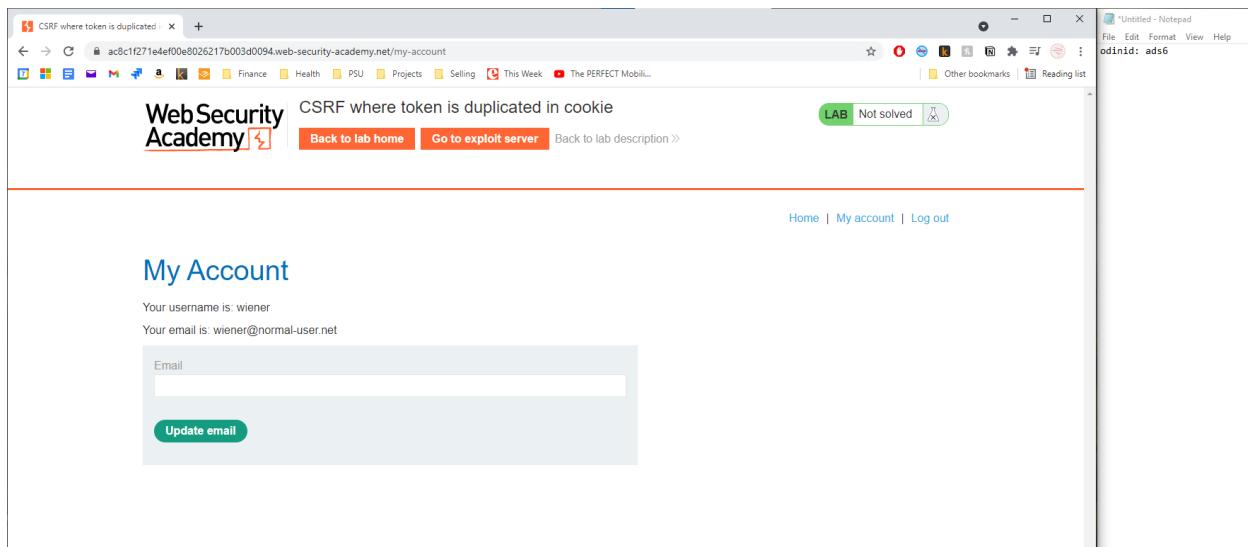
HTTP status code 200

CSRF token in HTML response is ads6

- How might a developer use a keyed hash function on the server to prevent this request from succeeding without being forced to store each token?

Make the csrf a hash of the session id and the time?

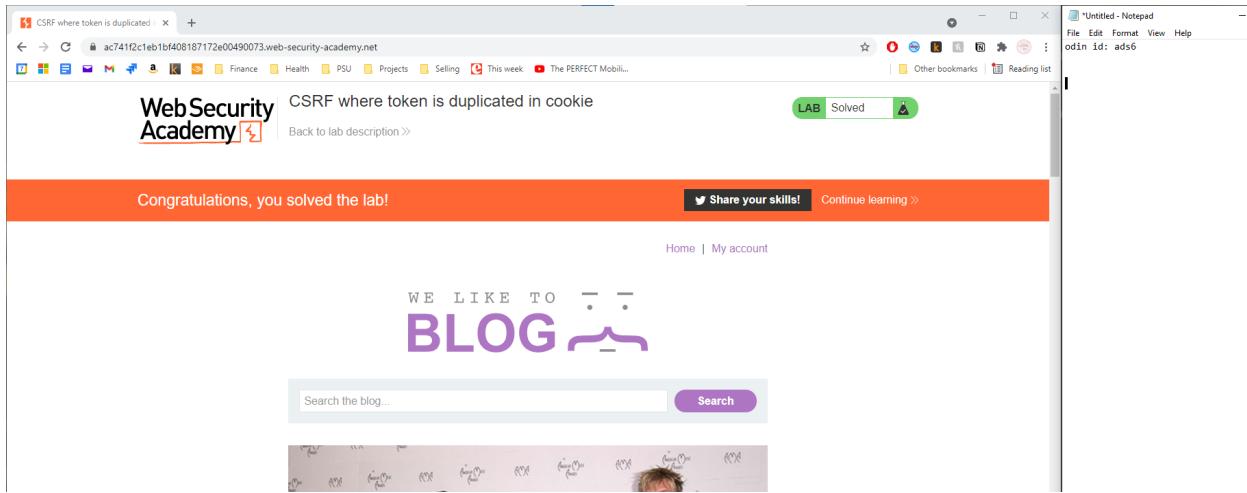
- Take a screenshot of the page that you are sent to for your lab notebook



- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The csrf uses the double submit method, which the adversary may bypass by choosing any string for both the form csrf and the cookie csrf. This, combined with the improper use of unsanitized user input in the http header, permits the adversary to execute a XSS attack when the authenticated user visits a site controlled by them.

**Remediation:** Authenticate the csrf against something that isn't client-supplied; and sanitize the user input before incorporating it in the http header.



The screenshot shows a browser window with the URL <https://ac741f2c1eb1bf408187172e00490073.web-security-academy.net>. The page is titled "CSRF where token is duplicated in cookie". A green button at the top right indicates the task is "Solved". A banner at the top says "Congratulations, you solved the lab!" with links to "Share your skills!" and "Continue learning >". The main content area features a logo with the text "WE LIKE TO BLOG" and a search bar. A small image of a person's face is visible in the background.

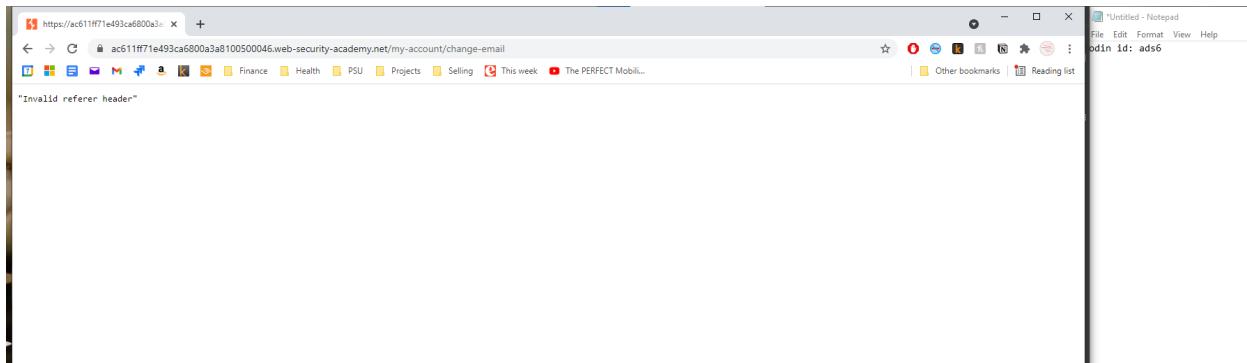
## 9. csrf (5)

- What status code and response text is returned?

HTTP status code: 400 with response text "Invalid referer header"

200

- Take a screenshot of the page that is returned including its URL

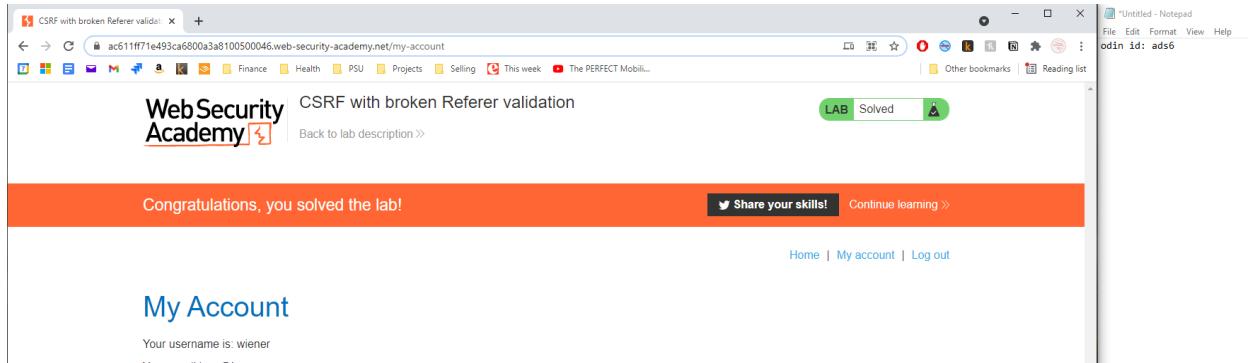


The screenshot shows a browser window with the URL <https://ac511ff71e493ca6800a3a8100500046.web-security-academy.net/my-account/change-email>. The page displays the error message "Invalid referer header".

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The server tries to prevent csrf attacks using the referrer header for validation; unfortunately, the adversary may bypass this check by simply adding the site to the client's history via the `history.pushState()` command.

**Remediation:** Use a server-side hashing or storage check to validate the csrf; don't rely on client-supplied input.

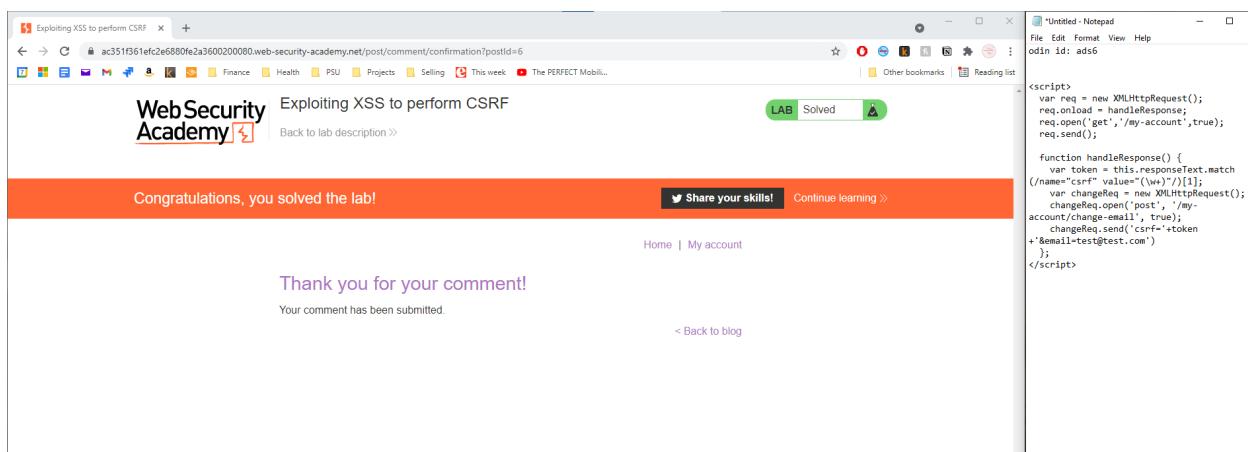


## 11. cross-site-scripting/exploiting

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The csrf is adequately checked; but a XSS vulnerability in the comments feature allows the adversary to embed their attack page inside the site itself.

**Remediation:** Sanitize user comments before incorporating them into the HTML.



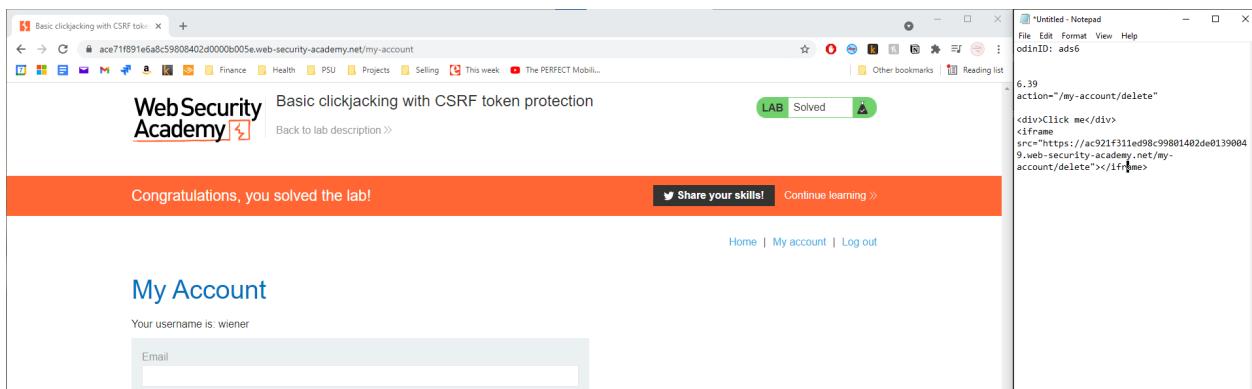
## Lab 3.4

## 1. clickjacking (1)

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The adversary has embedded the target site in an iframe on a site they control, allowing them full access to the form contents before they are submitted.

**Remediation:** Employ the X-Frame-Options: deny option in the response header.



## 4. clickjacking (2)

- In the form's HTML, how many fields are present? Which one has been prefilled with a value?

Two fields are present; one (CSRF) has been pre-filled.

- Take a screenshot showing that the form in the transparent <iframe> has been prefilled similar to below

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** Similar to above, but this time the adversary also exploits the inclusion of the email= query string parameter to pre-fill the victim's form before click-jacking the submission.

**Remediation:** Employ the X-Frame-Options: deny option in the response header.

## 5. clickjacking (3)

- What is the name of the HTML element that has been updated after the form has been submitted?

feedbackResult

- What is the name of the function that is registered as an event listener for when the page is loaded?

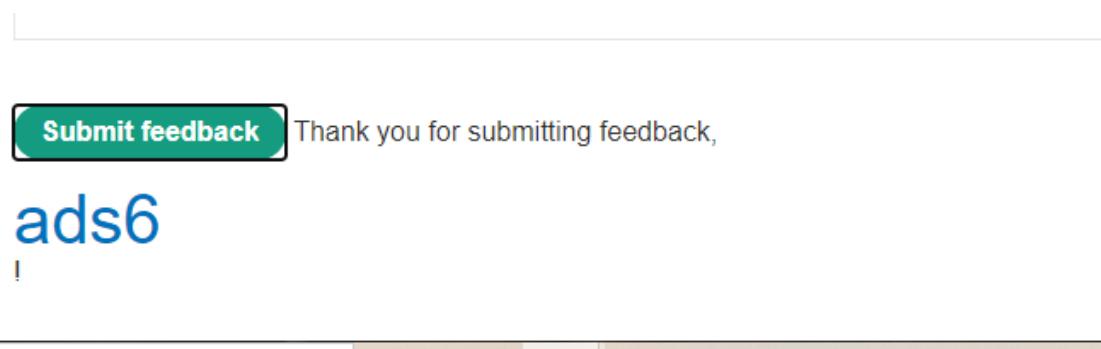
displayFeedbackMessage()

- Show the vulnerable line of code in this function and explain why it is subject to an XSS attack.

This line of code uses unsanitized client input from the form. (I could also show the place where this function is invoked: there are two places where the client input might have been sanitized, but was not.)

```
: (this.status === 200) {  
    feedbackResult.innerHTML = "Thank you for submitting feedback" + (name ? ", " + name : "") + "!";  
    feedbackForm.reset();
```

- Take a screenshot that includes the URL demonstrating successful injection



- Did the payload get successfully returned into the page?

yes

- Did the payload execute? If not, what might be the reason?

The script was added via the DOM /after/ the page was loaded...

- Take a screenshot showing the location of the "Click me" element on the underlying feedback page.

```

exploit_html = f'''<style>
iframe {{
    position: relative;
    width: 1400px;
    height: 1400px;
    opacity: 0.3;
    z-index: 2;
}}
div {{
    position: absolute;
    top: 830px;
    left: 150px;
    z-index: 1;
}}
</style>
<div>Click me</div>
<iframe src="https://({site}).name<img%20src=1%20onerror=alert(document.cookie)1%20>">...
...

```

- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** Similar to above, but this time the adversary injects a cross-site script through unsanitized client input vulnerability in the comment function, allowing them to execute arbitrary code on the client when the user clicks.

**Remediation:** Employ the X-Frame-Options: deny option in the response header. And html-encode the user input before incorporating it into the DOM.

```

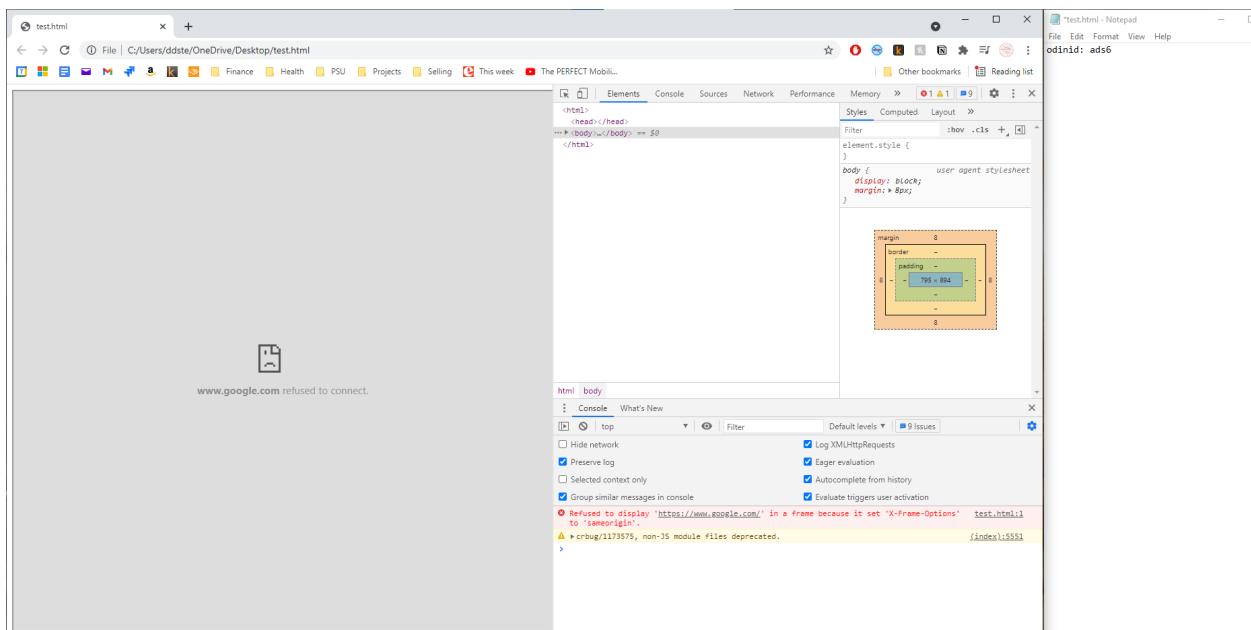
exploit_html = f'''<style>
iframe {{
    position: relative;
    width: 1400px;
    height: 1400px;
    opacity: 0.3;
    z-index: 2;
}}
div {{
    position: absolute;
    top: 830px;
    left: 150px;
    z-index: 1;
}}
</style>
<div>Click me</div>
<iframe src="https://({site}).feedback?name=<img%20src=1%20onerror=alert(document.cookie)>&email=a@b.com&subject=a&m...">...
...

```

## 8. Prevention (X-Frame-Options)

- Show a screenshot of what Google passes back with in its X-Frame-Options: header.

- If not, show the error in the console and explain the results.



- Show a screenshot of what OregonCTF passes back. Is there a X-Frame-Options: header?

No, it does not.

```

ads6 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal ~
May 10 21:46
ads6@ads6-VirtualBox:~$ curl -C oregonctf.org 80
HTTP/1.1 400 Bad Request
Server: nginx
Date: Tue, 11 May 2021 04:40:38 GMT
Content-Type: text/html
Content-Length: 150
Connection: close

<html>
<head><title>400 Bad Request</title></head>
<body>
<center><h1>400 Bad Request</h1></center>
<hr><center>nginx</center>
</body>
</html>
^[[A

ads6@ads6-VirtualBox:~$ nc -C oregonctf.org 80
HTTP/1.1 400 Bad Request
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 11 May 2021 04:42:23 GMT
Content-Type: text/html
Content-Length: 166
Connection: close

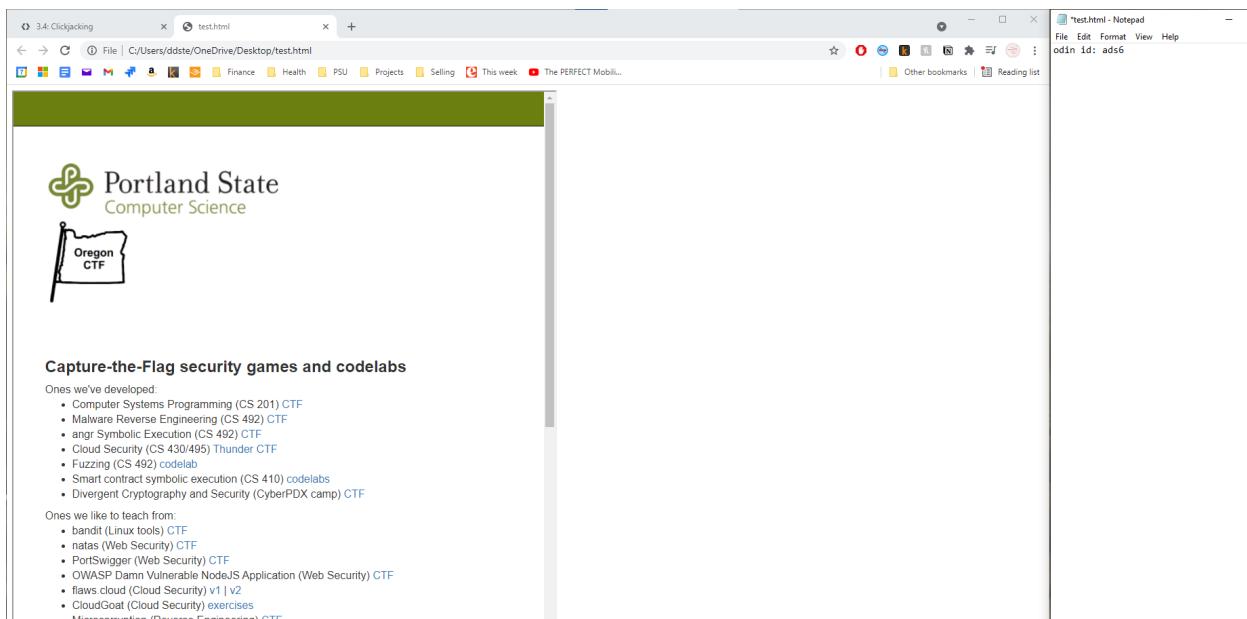
<html>
<head><title>400 Bad Request</title></head>
<body>
<center><h1>400 Bad Request</h1></center>
<hr><center>nginx/1.18.0 (Ubuntu)</center>
</body>
</html>
^[[A

ads6@ads6-VirtualBox:~$ nc -C oregonctf.org 80
HEAD / HTTP/1.0
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 11 May 2021 04:46:19 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Thu, 24 Dec 2020 16:24:05 GMT
Connection: close
ETag: "5fe4c0a5-264"
Accept-Ranges: bytes
^[[A

```

- If not, show the error in the console and explain the results.

Yep, it loads...



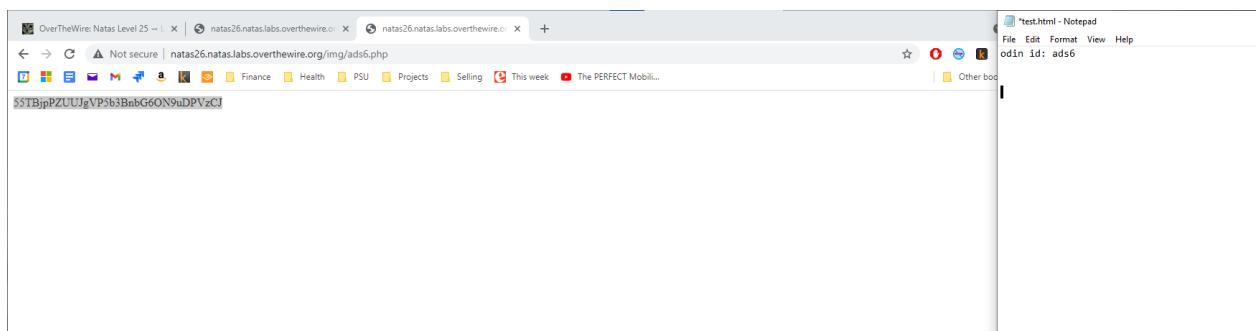
# Lab 3.5

- Highlight the coordinates you entered previously via the UI in the decoded output for your lab notebook

(highlighted in blue)

```
root@ads6:~/catBox/webSecurity# base64 -d 100.txt
a:2:{l:0;a:4:{s:2:"x1";s:1:"1";s:2:"y1";s:1:"1";s:2:"x2";s:2:"78";s:2:"y2";s:2:"78";}i:1;a:4:{s:2:"x1";s:1:"1";s:2:"y1";s:1:"1";s:2:"x2";s:2:"98";s:2:"y2";s:2:"9
0";}base64: invalid input
```

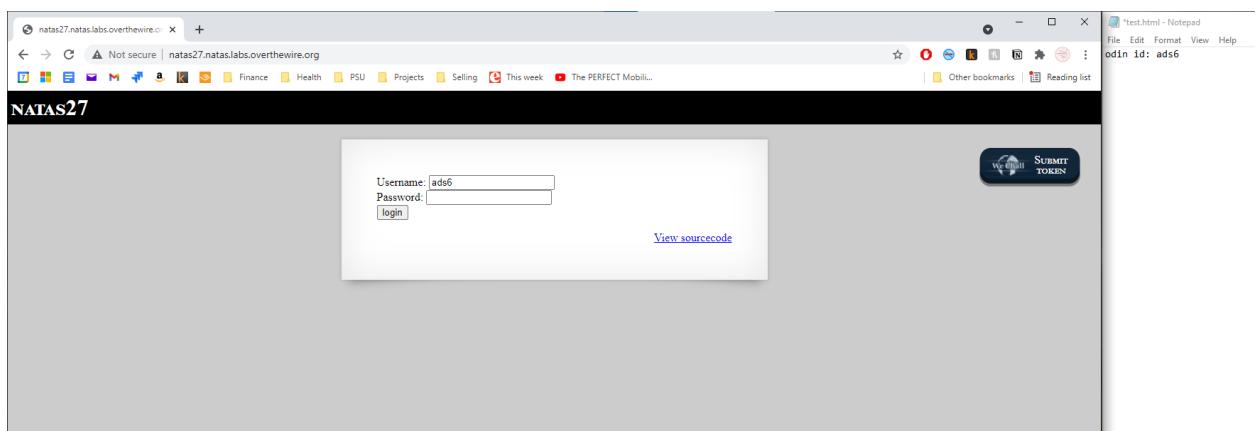
- Take a screenshot of it with your OdinID to include in your lab notebook.



- Take another screenshot with your OdinID in the Username field for your lab notebook.

**Vulnerability:** The cookie is deserialized before being validated, allowing the adversary to inject arbitrary php code into the flow of program execution. In this case, we demonstrate that vulnerability by forcing the OS to create a file that will print out a secret code to anyone who accesses it.

**Remediation:** Properly validate the contents of the cookie before deserializing it.



## Lab 3.6