

# Lab notebook turnin 1 - Andrew Stevenson

## Lab 1.2

- [5. Run sequential program](#)
- [7. Run multiprocessing program](#)
- [9. Run matplotlib program on repl.it](#)
- [12. Run asyncio program on repl.it](#)

## Lab 1.3

- [2. Login form](#)
- [3. authentication/password-based \(1\)](#)
- [4. authentication/password-based \(2\)](#)
- [5. authentication/password-based \(3\)](#)

## Lab 1.5

- [1. file-path-traversal \(1\)](#)
- [2. file-path-traversal \(2\)](#)
- [3. access-control \(1\)](#)
- [4. access-control \(2\)](#)
- [5. access-control \(3\)](#)
- [6. access-control \(4\)](#)
- [7. access-control \(5\)](#)
- [8. access-control \(6\)](#)
- [9. access-control \(7\)](#)
- [10. information-disclosure](#)
- [11. WFP1: File upload](#)

## Lab 1.6

- [1. ssrf \(1\)](#)
- [2. ssrf \(2\)](#)
- [3. ssrf \(3\)](#)
- [4. ssrf \(4\)](#)
- [5. ssrf/blind](#)

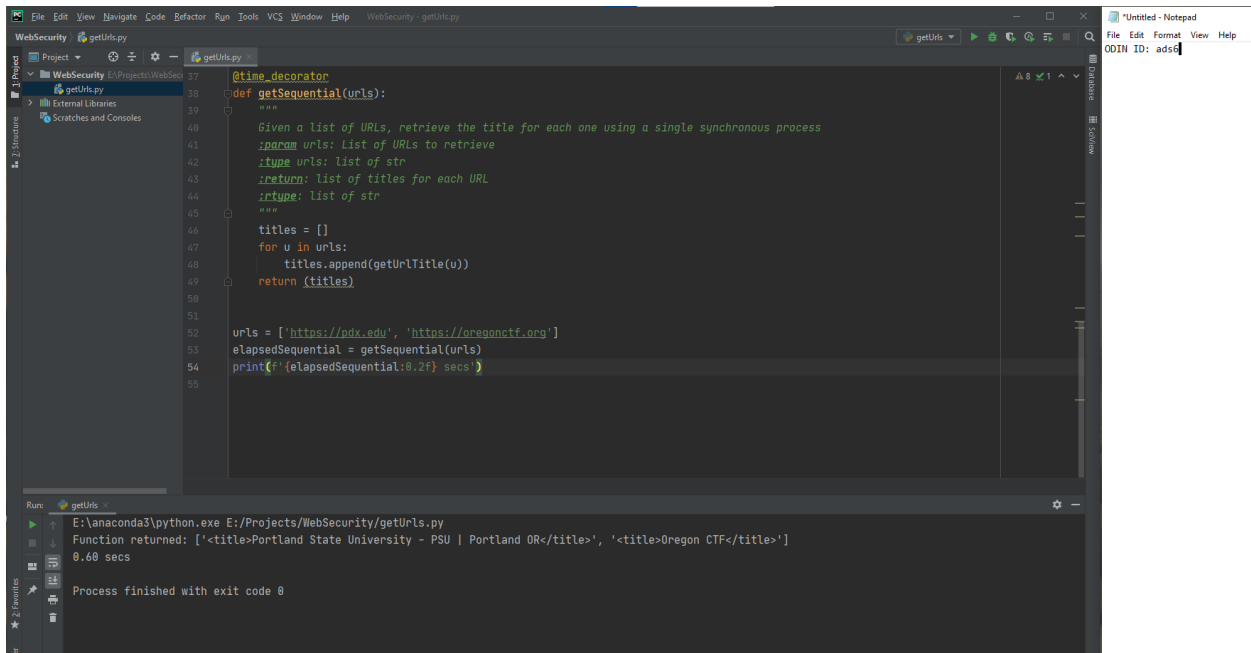
## Lab 1.7

- [1. xxe \(1\)](#)
- [2. xxe \(2\)](#)
- [3. xxe \(3\)](#)
- [4. xxe \(4\)](#)
- [5. -](#)

# Lab 1.2

## 5. Run sequential program

- Take a screenshot of the output of your program's execution that includes your OdinId for your lab notebook.



```
WebSecurity getUrls.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
WebSecurity getUrls.py
Project
WebSecurity E:\Projects\WebSecurity
getUrls.py
External Libraries
Scratches and Consoles
37 @time_decorator
38 def getSequential(urls):
39     """
40     Given a list of URLs, retrieve the title for each one using a single synchronous process
41     :param urls: list of URLs to retrieve
42     :type urls: list of str
43     :return: list of titles for each URL
44     :rtype: list of str
45     """
46     titles = []
47     for u in urls:
48         titles.append(getUrlTitle(u))
49     return (titles)
50
51
52 urls = ['https://pdx.edu', 'https://oregonctf.org']
53 elapsedSequential = getSequential(urls)
54 print(f'{elapsedSequential:0.2f} secs')
55
Run: getUrls
E:\anaconda3\python.exe E:\Projects\WebSecurity\getUrls.py
Function returned: ['<title>Portland State University - PSU | Portland OR</title>', '<title>Oregon CTF</title>']
0.68 secs
Process finished with exit code 0
```

## 7. Run multiprocessing program

- Take a screenshot of the output of your program's execution that includes your OdinId for your lab notebook.

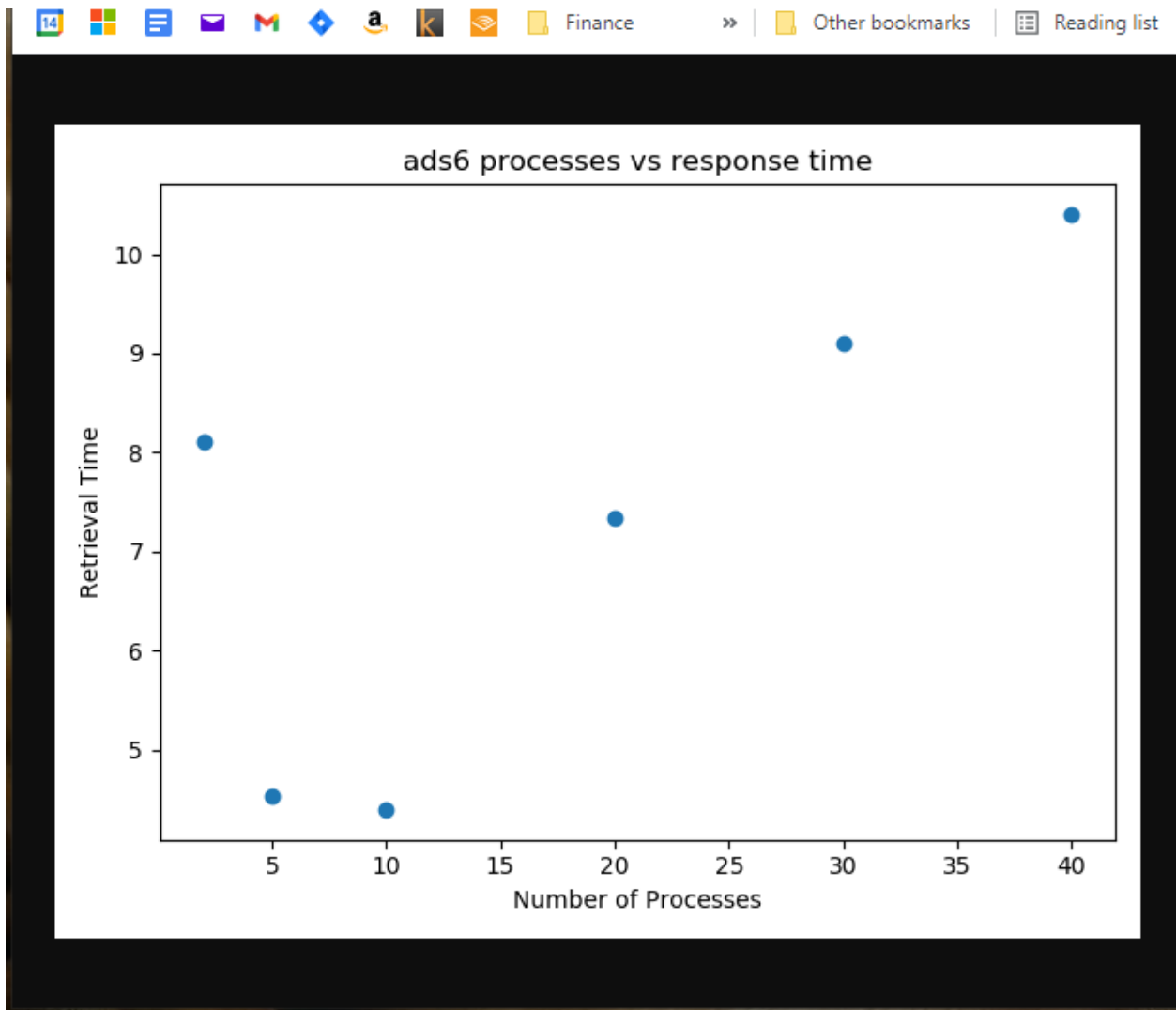
```
File Edit View Navigate Code Refactor Run Tools VCS Window Help WebSecurity - getUrls.py
WebSecurity E:\Projects\WebSecurity
getUrls.py
69
70
71 if __name__ == '__main__':
72     multiprocessing.freeze_support()
73
74     urls = ['https://pdx.edu', 'https://oregonctf.org', 'https://google.com', 'https://facebook.com', 'https://repl.it',
75            'http://whitehouse.gov', 'http://oxy.edu', 'http://yahoo.com', 'http://amazon.com', 'http://youtube.com']
76     # elapsedSequential = getSequential(urls)
77     # print(f'{elapsedSequential:0.2f} secs')
78
79     if __name__ == '__main__':
80         ...

Run: getUrls -
E:\anaconda3\python.exe E:\Projects\WebSecurity\getUrls.py
Function returned: ['<title>Portland State University - PSU | Portland OR</title>', '<title>Oregon CTF</title>', '<title>Google</title>', '<title
id="pageTitle">Facebook - Log In or Sign Up</title>', '<title>The collaborative browser based IDE - Replit</title>', '<title>The White House</title>',
'<title>Occidental College | The Liberal Arts College in Los Angeles</title>', '<title>Yahoo</title>', '<title>Sorry! Something went wrong!</title>',
'<title>YouTube</title>']
2 4.69
Function returned: ['<title>Portland State University - PSU | Portland OR</title>', '<title>Oregon CTF</title>', '<title>Google</title>', '<title
id="pageTitle">Facebook - Log In or Sign Up</title>', '<title>The collaborative browser based IDE - Replit</title>', '<title>The White House</title>',
'<title>Occidental College | The Liberal Arts College in Los Angeles</title>', '<title>Yahoo</title>', '<title>Sorry! Something went wrong!</title>',
'<title>YouTube</title>']
5 5.14
Function returned: ['<title>Portland State University - PSU | Portland OR</title>', '<title>Oregon CTF</title>', '<title>Google</title>', '<title
id="pageTitle">Facebook - Log In or Sign Up</title>', '<title>The collaborative browser based IDE - Replit</title>', '<title>The White House</title>',
'<title>Occidental College | The Liberal Arts College in Los Angeles</title>', '<title>Yahoo</title>', '<title>Sorry! Something went wrong!</title>',
'<title>YouTube</title>']
10 2.45

Process finished with exit code 0
```

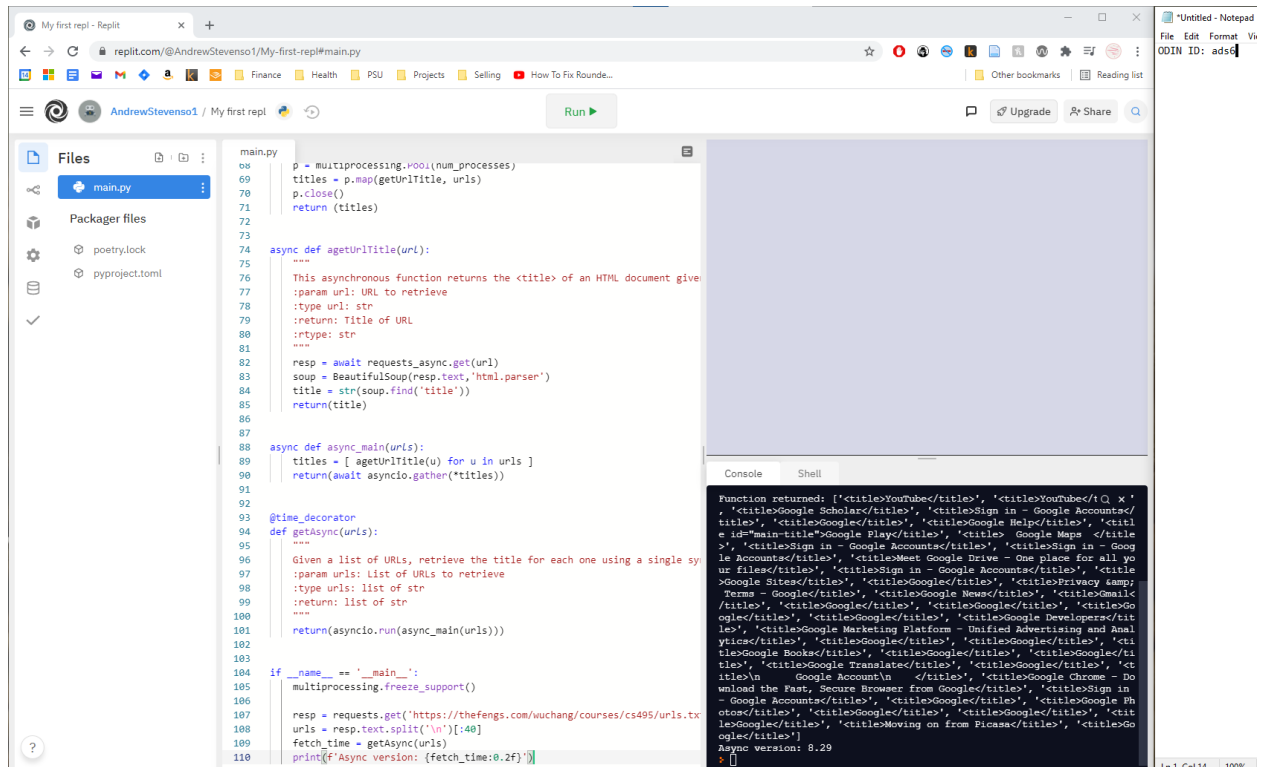
## 9. Run matplotlib program on repl.it

- Take a screenshot of the plot for your lab notebook



## 12. Run asyncio program on repl.it

- Take a screenshot of the output of your program's execution that includes your OdinId for your lab notebook.



```
main.py
68 p = multiprocessing.Pool(num_processes)
69 titles = p.map(get_url_title, urls)
70 p.close()
71 return (titles)
72
73
74 async def aget_url_title(url):
75     """
76     This asynchronous function returns the <title> of an HTML document given
77     :param url: URL to retrieve
78     :type url: str
79     :return: Title of URL
80     :rtype: str
81     """
82     resp = await requests_async.get(url)
83     soup = BeautifulSoup(resp.text, 'html.parser')
84     title = str(soup.find('title'))
85     return(title)
86
87
88 async def async_main(urls):
89     titles = [ aget_url_title(u) for u in urls ]
90     return(await asyncio.gather(*titles))
91
92
93 @time_decorator
94 def get_async(urls):
95     """
96     Given a list of URLs, retrieve the title for each one using a single sy
97     :param urls: list of URLs to retrieve
98     :type urls: list of str
99     :return: list of str
100     """
101     return(asyncio.run(async_main(urls)))
102
103
104 if __name__ == '__main__':
105     multiprocessing.freeze_support()
106
107     resp = requests.get('https://thefengs.com/wuchang/courses/cs495/urls.txt')
108     urls = resp.text.split('\n')[1:40]
109     fetch_time = get_async(urls)
110     print(f'Async version: {fetch_time:0.2f}')
```

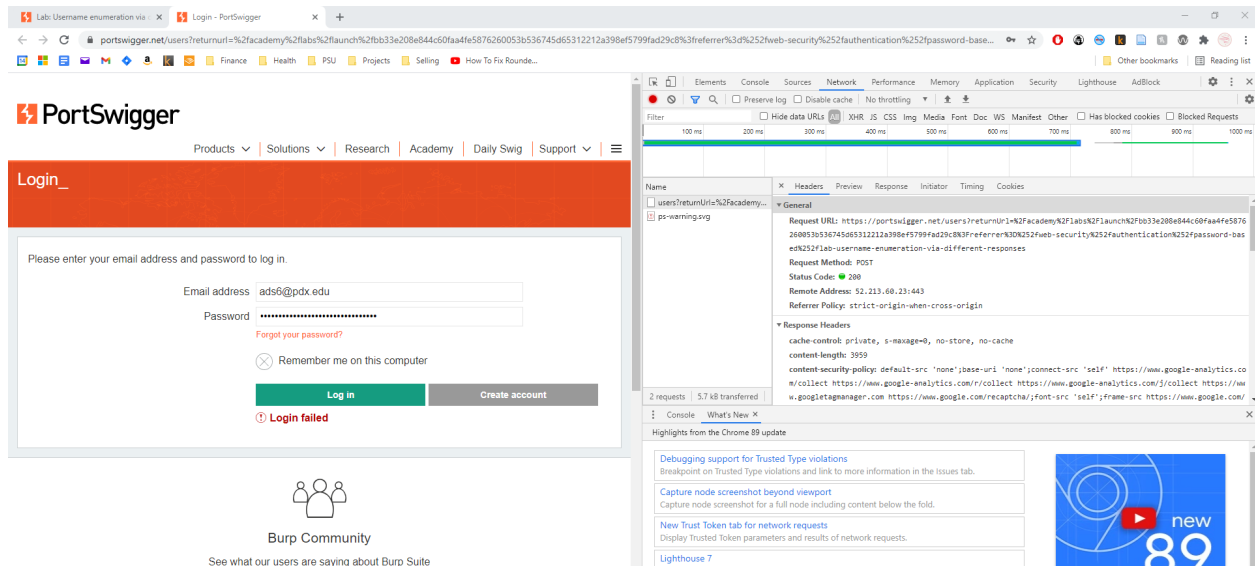
Function returned: ['<title>YouTube/</title>', '<title>YouTube/I.O.W.</title>', '<title>Google Scholar/</title>', '<title>Sign in - Google Accounts/</title>', '<title>Google/</title>', '<title>Google Help/</title>', '<title>id="main">Google Play/</title>', '<title> Google Maps </title>', '<title>Sign in - Google Accounts/</title>', '<title>Sign in - Google Accounts/</title>', '<title>Meet Google Drive - One place for all your files/</title>', '<title>Sign in - Google Accounts/</title>', '<title>Google Sites/</title>', '<title>Google/</title>', '<title>Privacy & Terms - Google/</title>', '<title>Google News/</title>', '<title>Gmail/</title>', '<title>Google/</title>', '<title>Google/</title>', '<title>Google/</title>', '<title>Google/</title>', '<title>Google Developer/</title>', '<title>Google Marketing Platform - Unified Advertising and Analytics/</title>', '<title>Google/</title>', '<title>Google/</title>', '<title>Google Books/</title>', '<title>Google/</title>', '<title>Google/</title>', '<title>Google Translate/</title>', '<title>Google/</title>', '<title>Google Account/</title>', '<title>Google Chrome - Do what you love, Secure Browser from Google/</title>', '<title>Sign in - Google Accounts/</title>', '<title>Google/</title>', '<title>Google Photos/</title>', '<title>Google/</title>', '<title>Google/</title>', '<title>Google/</title>', '<title>Moving on from Picasa/</title>', '<title>Google/</title>']

Async version: 8.29

# Lab 1.3

## 2. Login form

- Take a screenshot of this for your lab notebook

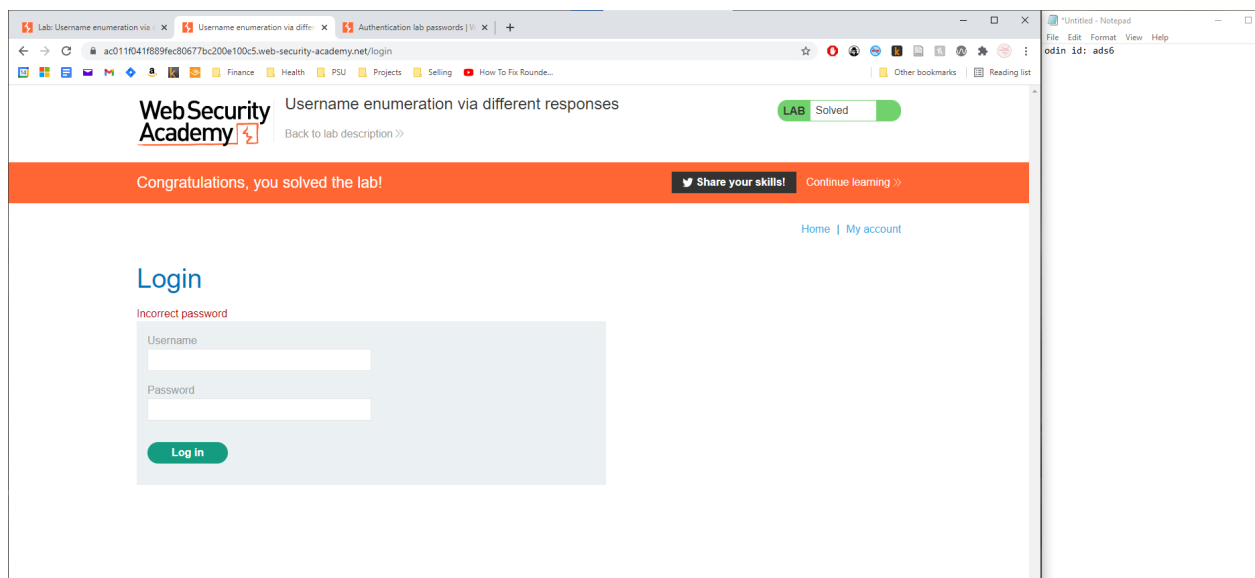


### 3. authentication/password-based (1)

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

**Vulnerability:** The login form returns information about whether the user id was valid, allowing me to brute force a valid user id with ease.

**Remediation:** Do not tell users whether the user name is valid when authentication fails.

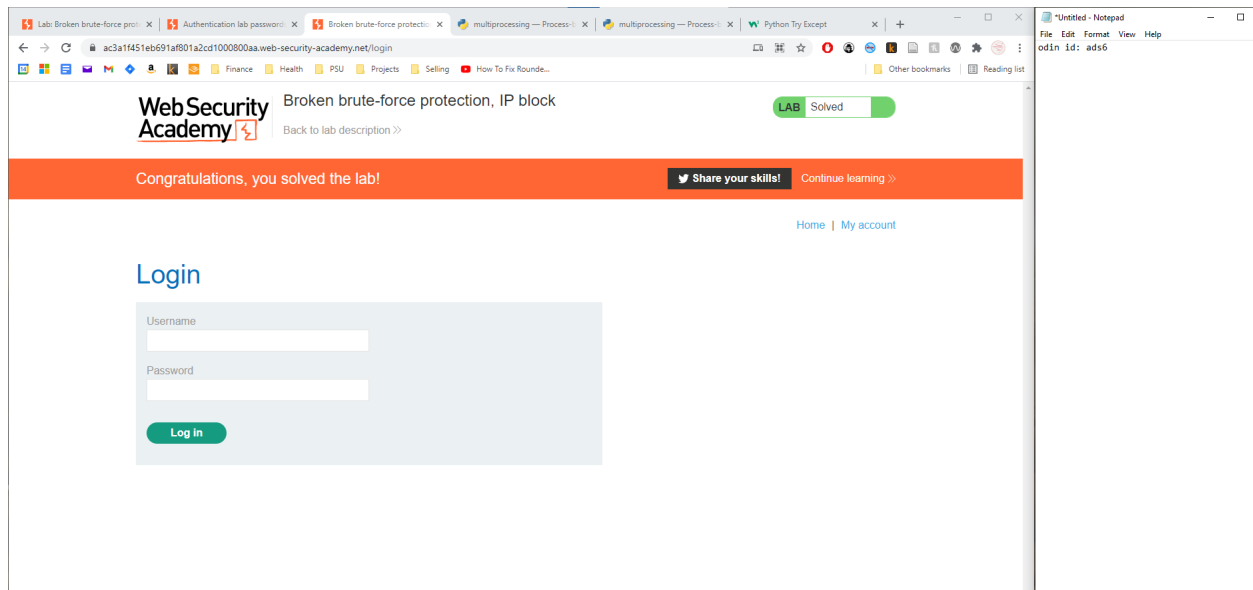


## 4. authentication/password-based (2)

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

**Vulnerability:** The ip lockout due to too many failed login attempts can be avoided by resetting the counter through logging in with a valid account, permitting anyone with valid credentials to brute force another user's password.

**Remediation:** Keep the counter per user account, not per ip address.

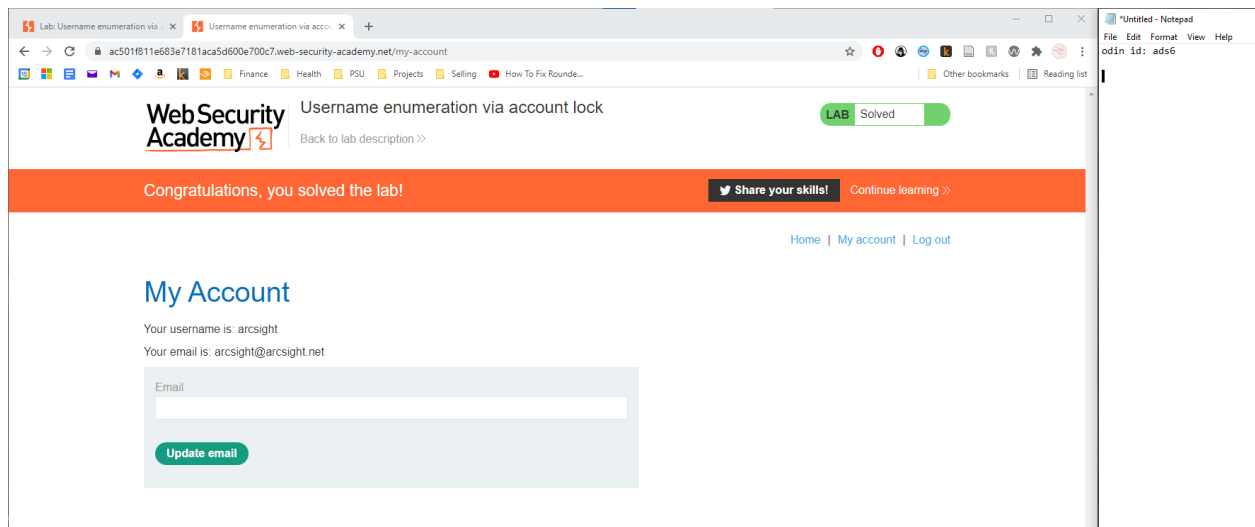


## 5. authentication/password-based (3)

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

**Vulnerability:** By locking out only valid user ids, the form allows us to create an oracle telling us whether a user id is valid by testing whether that user id is locked out when too many invalid login attempts are made.

**Remediation:** Extend the lockout mechanism to all accounts, whether they exist in the system or not.



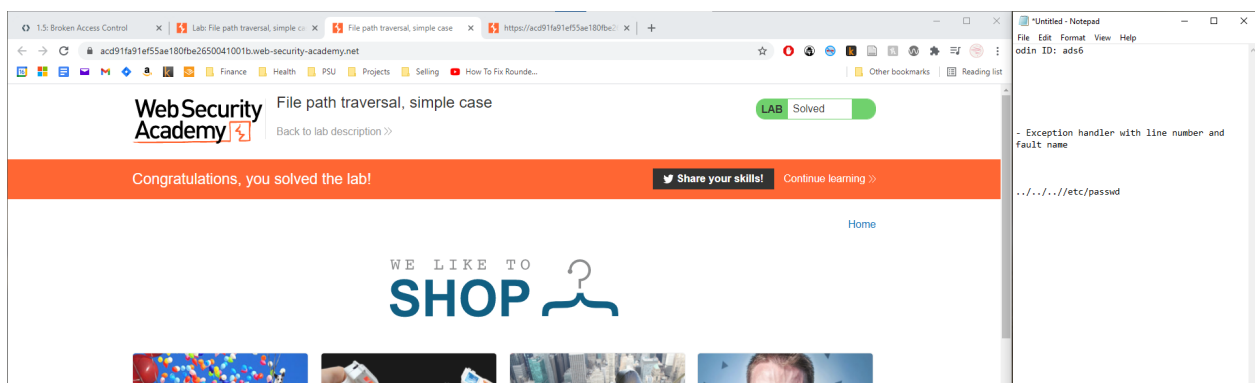
## Lab 1.5

### 1. file-path-traversal (1)

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

**Vulnerability:** By allowing the query string to specify an image path, the adversary (or anyone who's a bit clever, really) to request any file in the filesystem via unix pathing.

**Remediation:** Strip out all pathing during server-side processing.





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

**Vulnerability:** Absolute paths were not filtered out, allowing anyone with basic knowledge of the unix filesystem to request all sorts of sensitive files.

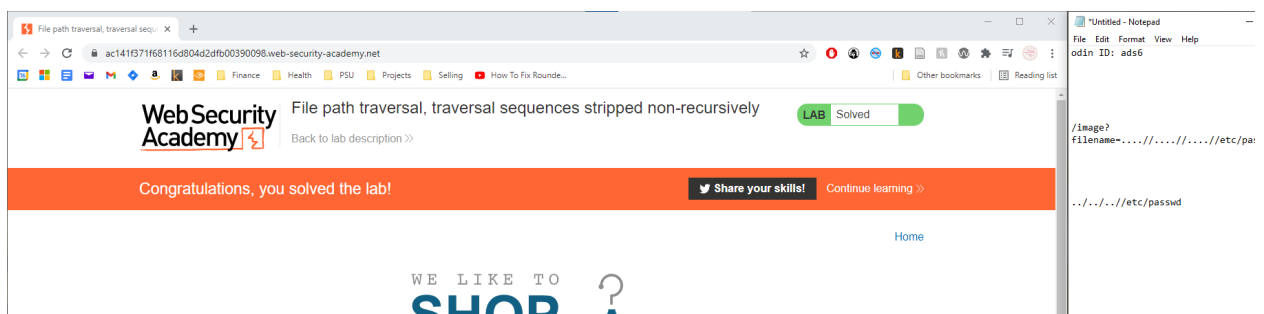
**Remediation:** Filter out all paths from the image request system.



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

**Vulnerability:** A simple search and replace has been implemented, instead of a properly recursive strip method. Thus ....// gets processed as ../ and the attacker has a way in as with the first problem in this set.

**Remediation:** Implement a fully recursive strip function.

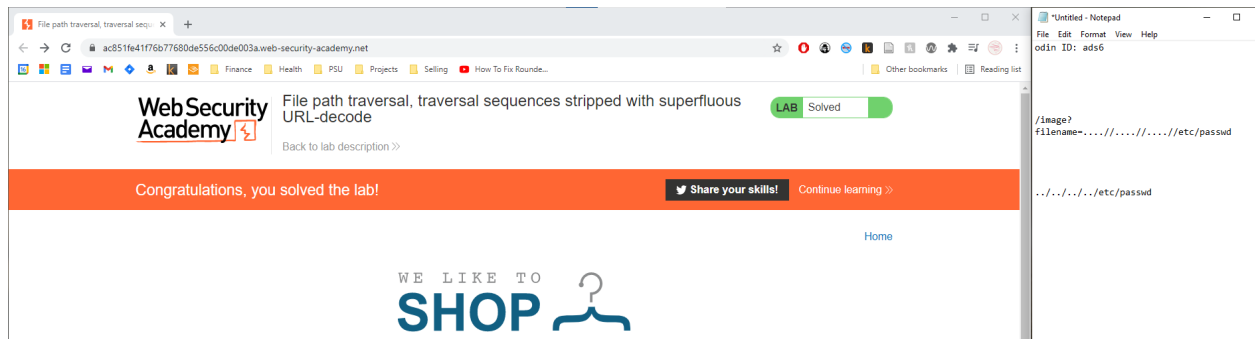


## 2. file-path-traversal (2)

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

**Vulnerability:** The program unencodes the query string in two different places, creating an opportunity to sneak in a path argument right through the filter. (This is very common, IME--web programmers like to think it “never hurts” to do it twice!)

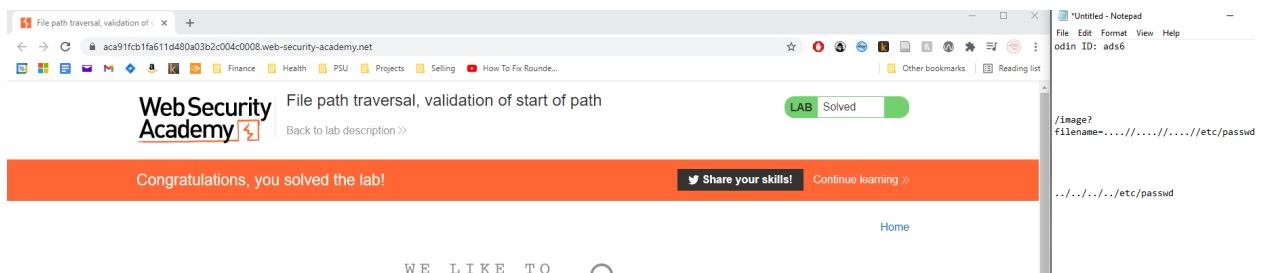
**Remediation:** Only decode the url query string once.



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

**Vulnerability:** Despite forcing image strings to conform to a certain url pattern, no filter is applied so it is a simple matter to add ../ directives after the required path.

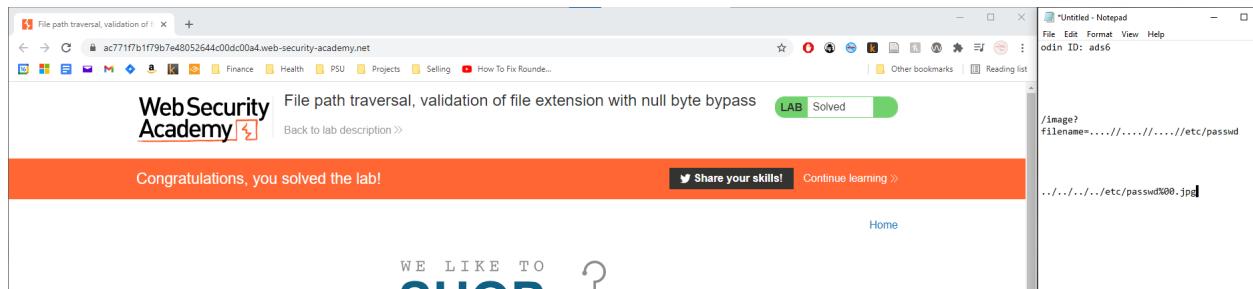
**Remediation:** Filter out those pathing portions of the string.



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

**Vulnerability:** Despite forcing the image paths to end with an image file extension, it's simple to insert an url-encoded null (%00) to make sure the string actually terminates before that extension is read.

**Remediation:** Be sure to strip out all null characters from that query string argument during pre-processing.

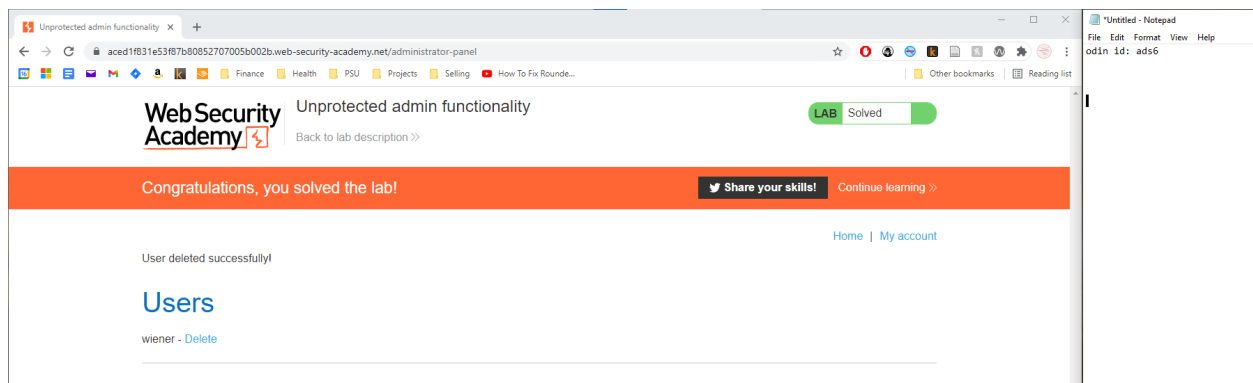


### 3. access-control (1)

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

**Vulnerability:** The admin panel is unprotected, allowing anyone with the right uri to navigate directly to it. Worse, the location of the panel is revealed in robots.txt.

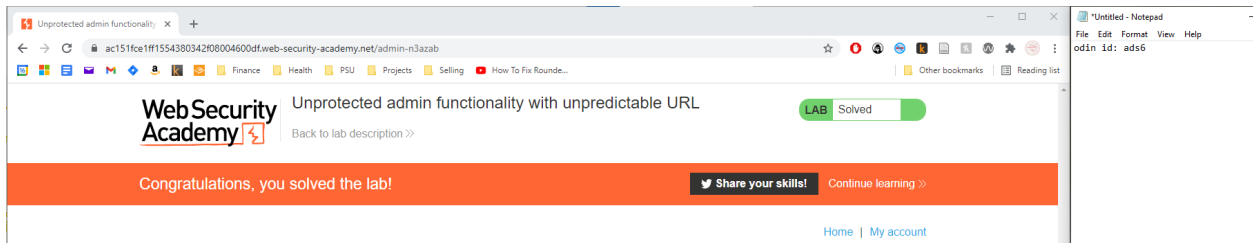
**Remediation:** Put the admin panel behind a login form.



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

**Vulnerability:** The url of the unpredictable admin panel location is revealed in a javascript block within the publicly visible html.

**Remediation:** Put the admin panel behind a login form.

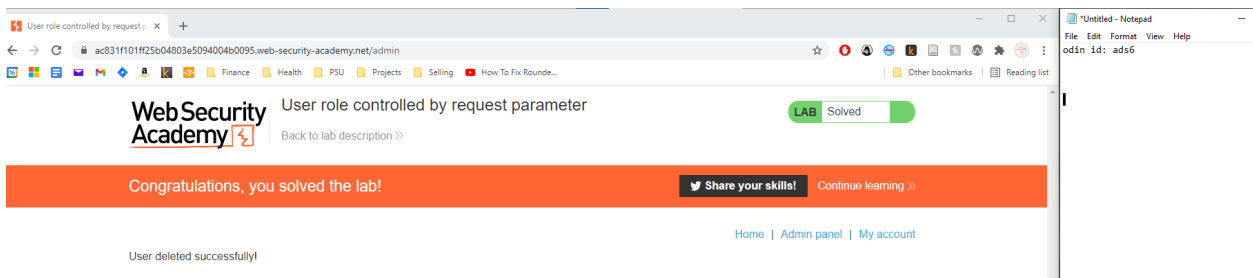


## 4. access-control (2)

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

**Vulnerability:** The admin flag is stored in a cookie, which can be changed by the client.

**Remediation:** Use a server state variable instead of a cookie to store the admin flag.

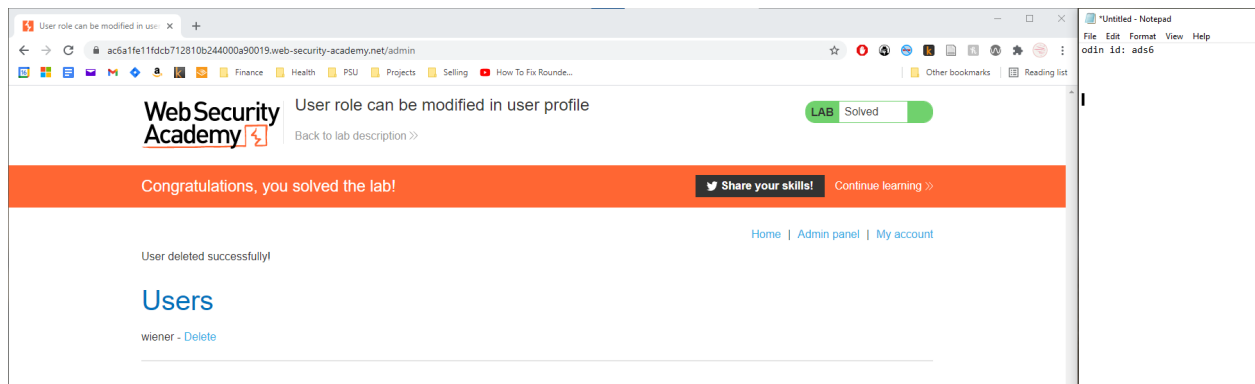


## 5. access-control (3)

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

**Vulnerability:** The server blindly incorporates the json string that had been sent to the client, back into its user record data. Since we can edit that object when it is on our side, we do so in order to achieve role escalation.

**Remediation:** Don't use the json sent back from the client to determine user role.

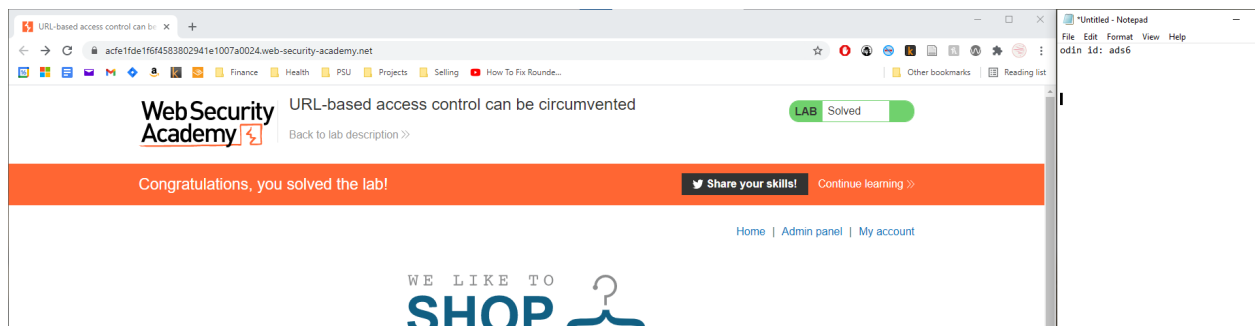


## 6. access-control (4)

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

**Vulnerability:** The front-end machine blocks requests for /admin -- but it doesn't block the contents of the X-Original-URL header. Since that header is under the client's control, we can easily request the admin page via that route.

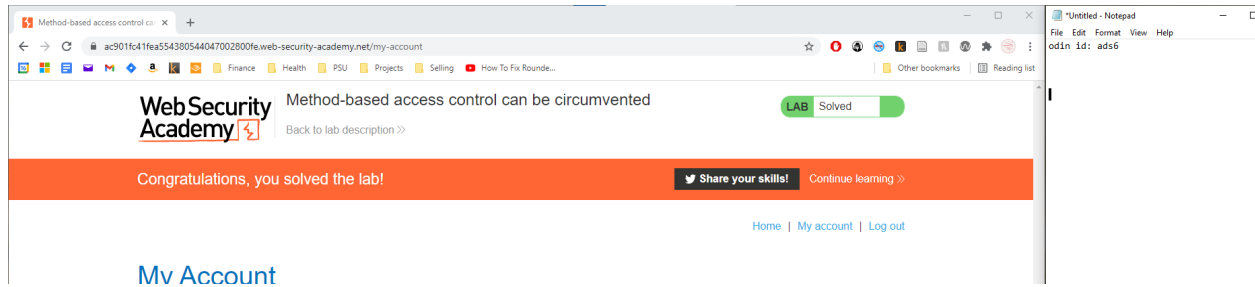
**Remediation:** Implement front-end logic blocking /admin requests through the X-Original-URL header (or maybe just all headers...)



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

**Vulnerability:** While the POST method is blocked from submitting to the privilege escalation form, the GET method is not blocked. Thus, an enterprising adversary can supply the necessary codes to receive full admin privileges.

**Remediation:** Block the GET method too.

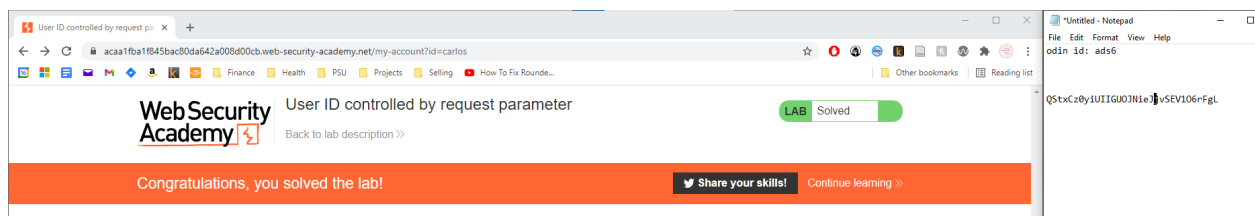


## 7. access-control (5)

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

**Vulnerability:** The querystring contains the user id; one can change that at will to become any user in the system.

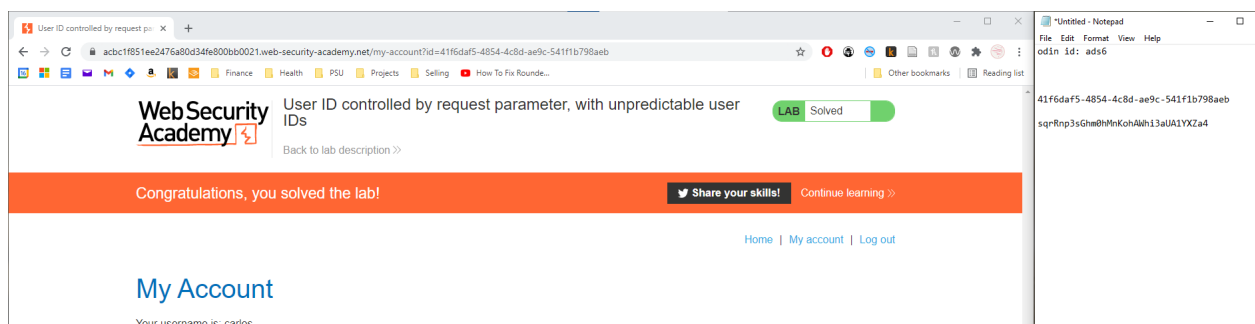
**Remediation:** Store user state in a server-side variable, not in the query string.



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

**Vulnerability:** The querystring still determines user state; only now it has been changed to a random user key. One has only to visit another page authored by the targeted user to retrieve their user key value and substitute for one's own on the account page.

**Remediation:** Store user state in a server-side variable, not in the query string.

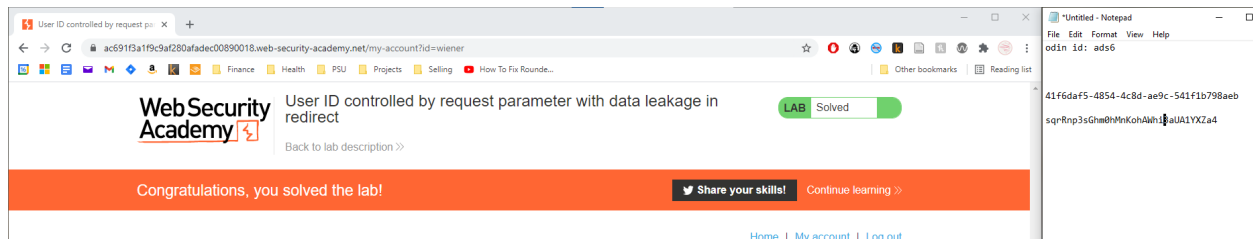


## 8. access-control (6)

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

**Vulnerability:** Despite redirecting users whose session user name doesn't match the query string, an adversary is able to view the page since it is up to the client whether they want to honor a redirect. Since the server fails to exit the script after issuing the redirect, it inappropriately spits out the page requested in the querystring.

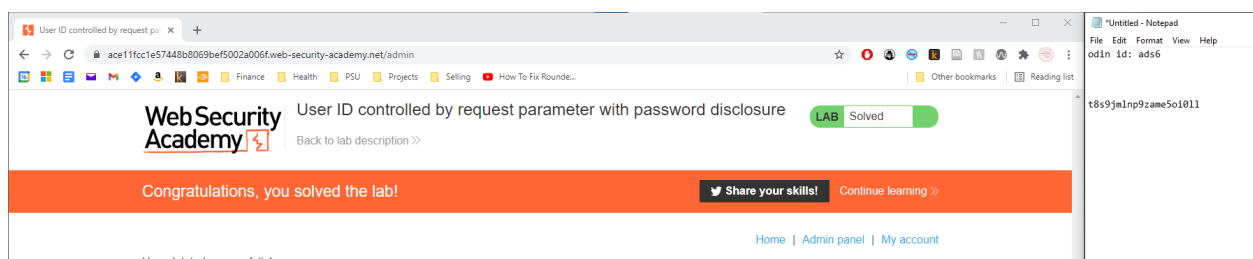
**Remediation:** Exit the server script after issuing the redirect.



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

**Vulnerability:** The password field comes prefilled, and it is possible to view another user's account page by merely changing the request in the query string. Together, these flaws mean the adversary has only to inspect the page code in order to find any other user's password, allowing for easy privilege escalation.

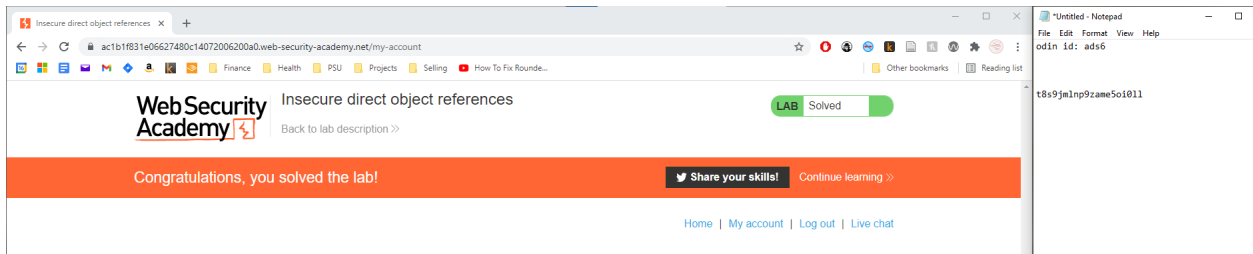
**Remediation:** Don't prefill the password field--there's no reason to do so.



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

**Vulnerability:** Other user's chat logs can be downloaded because the logs are given easily-guessable uris. Thus an adversary can easily snoop the logs to find the place where carlos told someone his password.

**Remediation:** Use long random strings for the log file names. Also, don't reveal your password in chat!

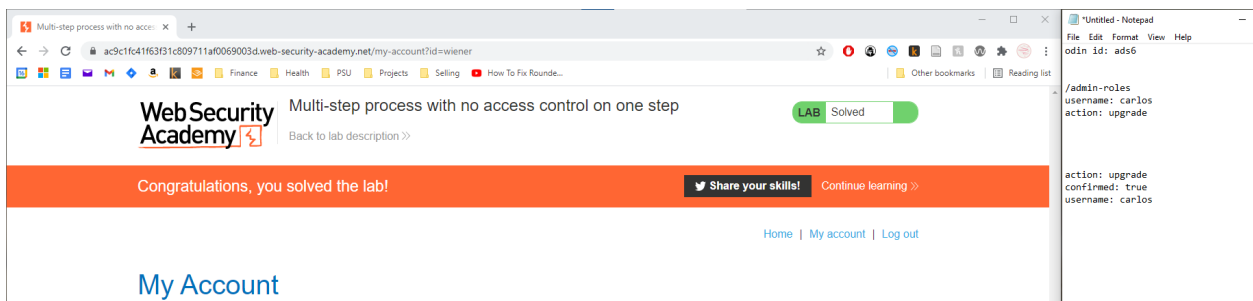


## 9. access-control (7)

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

**Vulnerability:** While the escalation forms themselves are protected, the confirmation process is not protected. Thus, the adversary may submit a POST request with the requisite confirmation form data appended in order to escalate their privileges to admin.

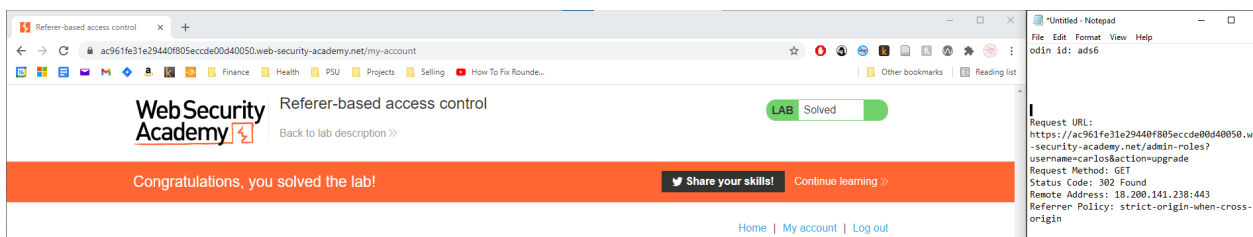
**Remediation:** Put the confirmation form behind a login requirement as well!



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

**Vulnerability:** The server uses the referrer header to decide whether the privilege escalation request should be acted on. But the client controls this data, allowing the adversary to easily forge a referrer header and escalate their privileges.

**Remediation:** Don't use the referrer header; instead check server-side login state to determine whether to honor the request.



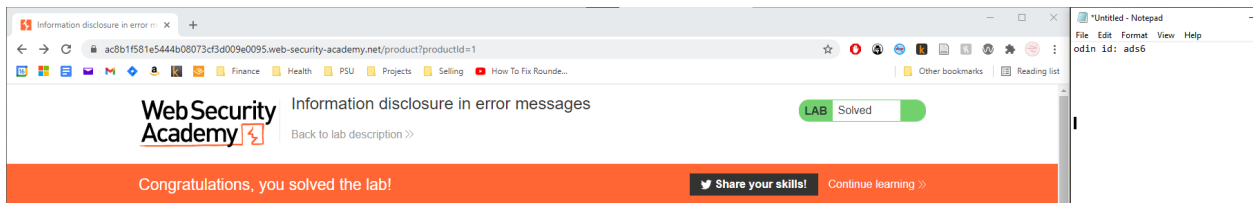


## 10. information-disclosure

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

**Vulnerability:** It's super easy to trigger an error message by requesting a productid that doesn't exist via the url query string, allowing the adversary to quickly discover what kind of software the server is running.

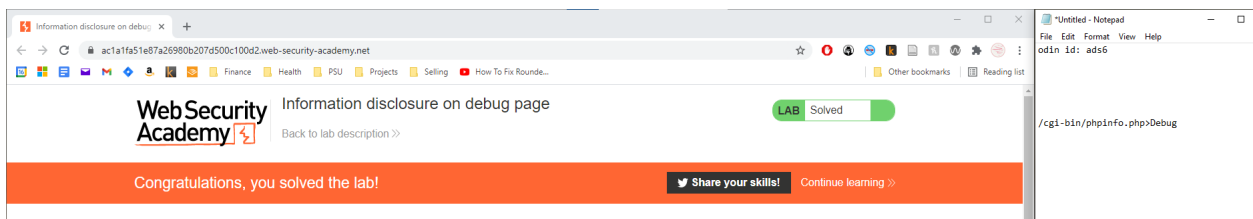
**Remediation:** Use server-level custom error handling to sanitize those error message (and also provide a better end-user experience!)



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

**Vulnerability:** The phpinfo page is publicly available, and even worse, its uri is revealed in a comment in the client-facing html.

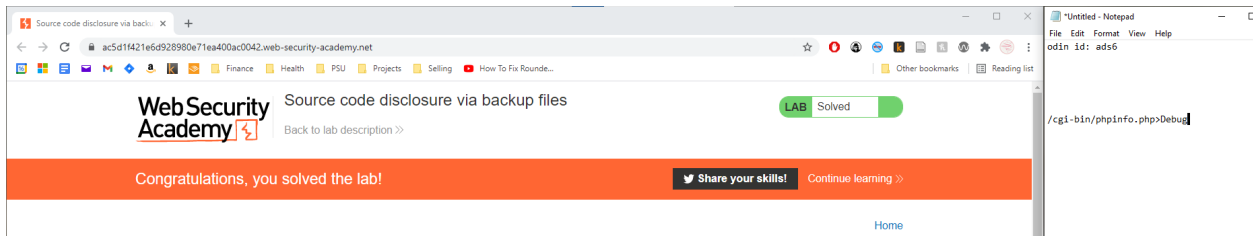
**Remediation:** Don't make this page available to clients at all, and definitely don't put that uri in the html.



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

**Vulnerability:** The backup directory with source files is accessible to the public. Worse, the location of these files is publicised in the robots.txt file.

**Remediation:** Put the backup files in a directory that isn't being published on the web.

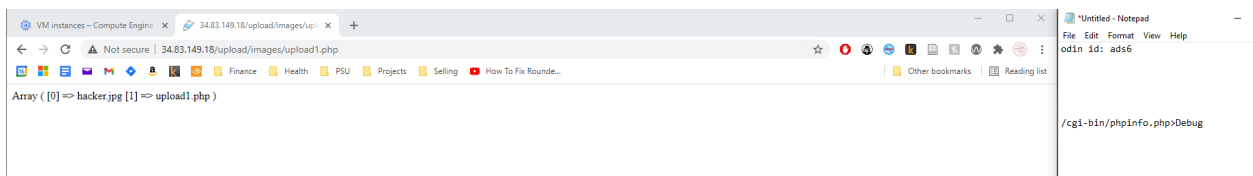


## 11. WFP1: File upload

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

**Vulnerability:** Client is allowed to upload any file type, permitting the adversary to put malicious files directly on the server.

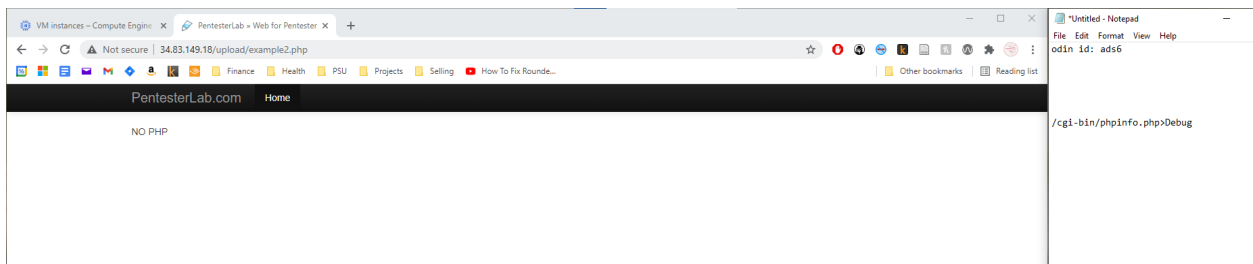
**Remediation:** Only allow files of a specific type (say, .bmp files) to be uploaded.

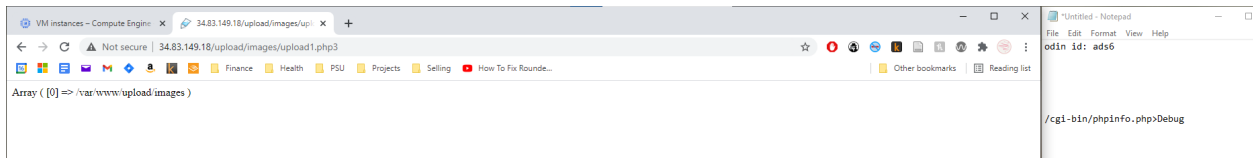


- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** PHP file extensions are blocked, but not php3, php4, or php5. PHP3 files will execute.

**Remediation:** Filter all valid php extensions.

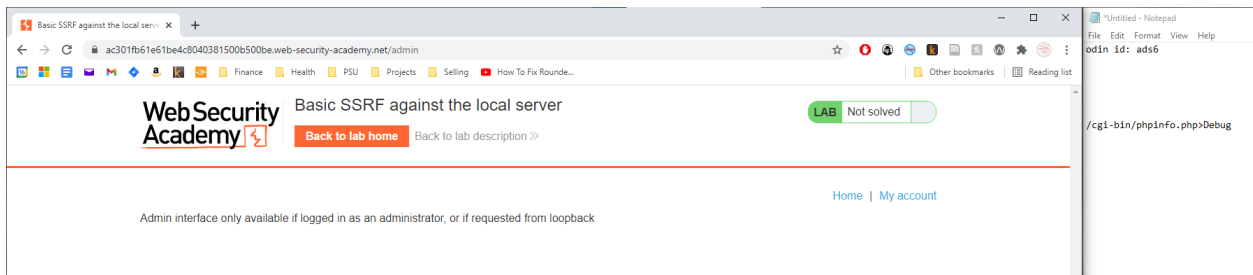




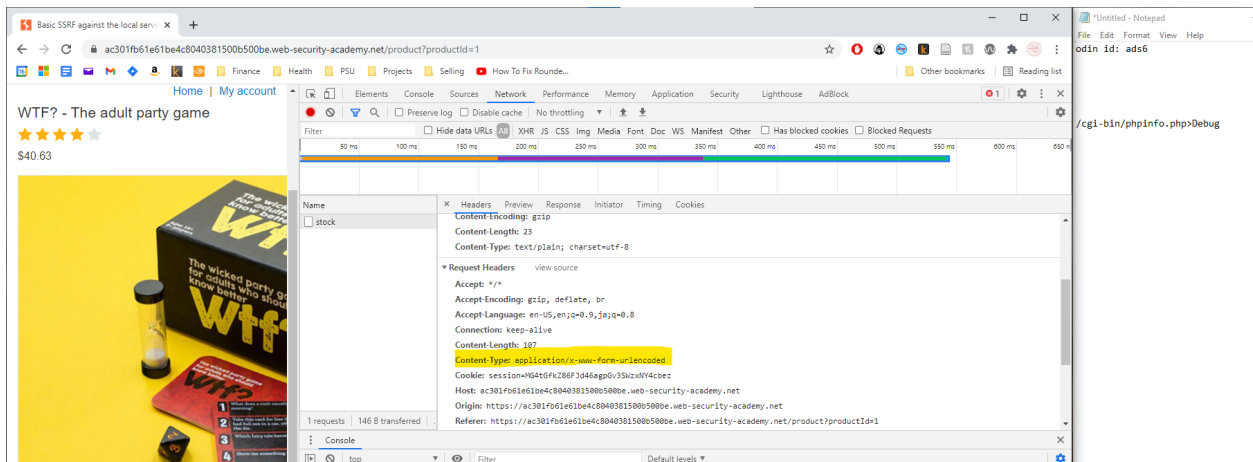
## Lab 1.6

### 1. ssrf (1)

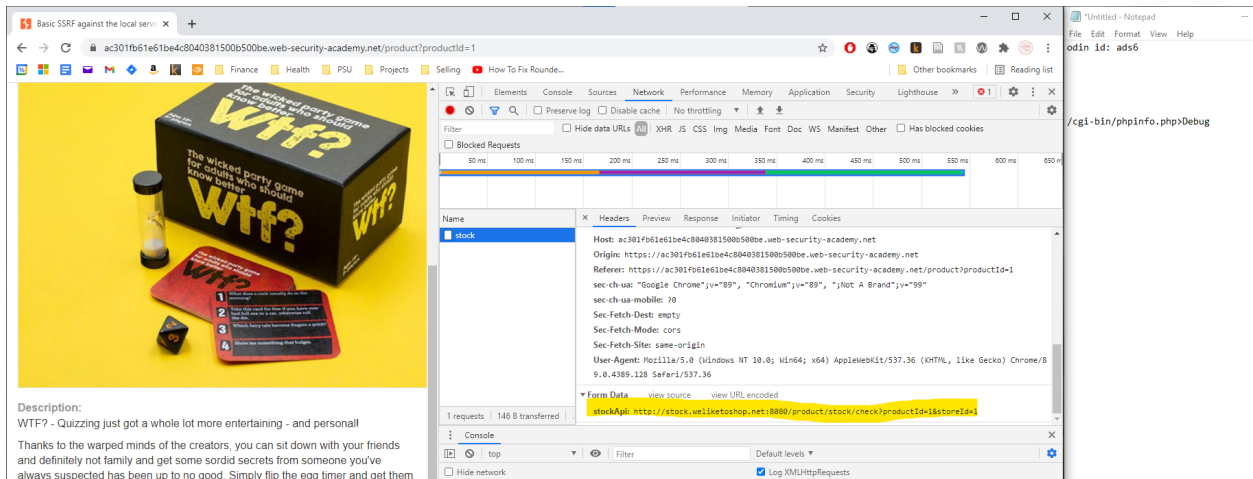
- Take a screenshot of the message that is returned when you attempt to visit the `/admin` path of your site.



- Take a screenshot that shows the Content-Type: request header that specifies the format of the form submission



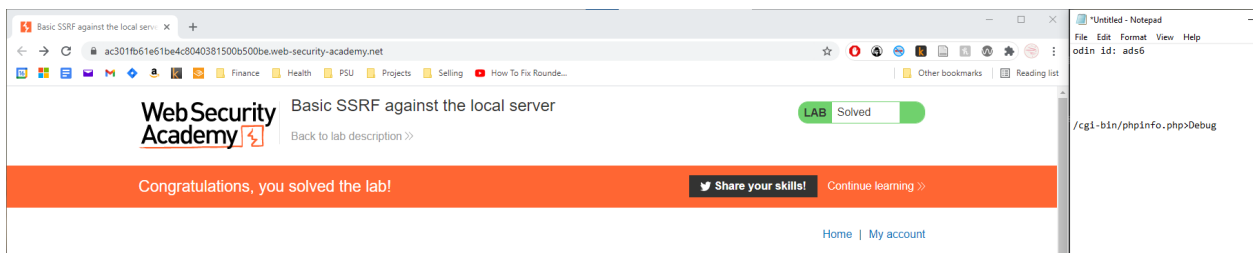
- Then, take a screenshot of the form submission payload.



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

**Vulnerability:** Through a proxy server, the adversary may request any page on the internal network, including the app admin page. (Though this would probably be the least of the host's worries...)

**Remediation:** Don't pass the ticker uri from the client.

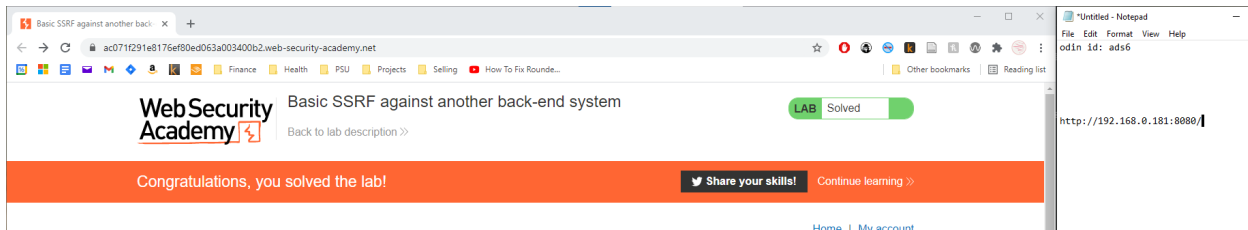


## 2. ssrf (2)

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

**Vulnerability:** The backend admin panel can be accessed through the proxy mechanism explored above. Although its identity is obfuscated by the fact we don't know the ip address, it is simple enough to scan the whole range to find the one hosting the panel and then access it freely.

**Remediation:** Put a password on the admin panel.

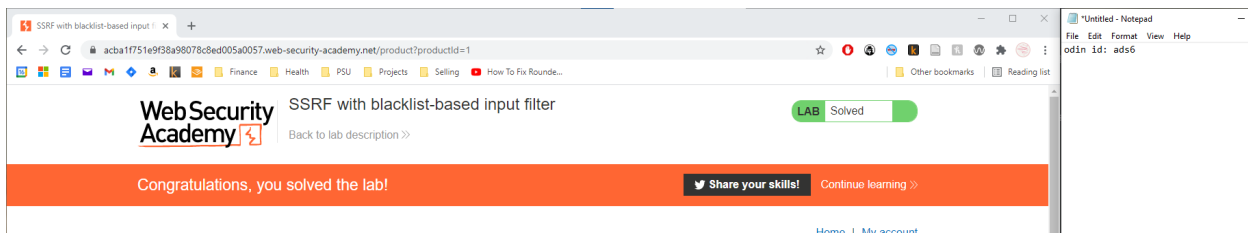


### 3. ssrf (3)

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

**Vulnerability:** Two security precautions impede the adversary from repeating the proxy attack on the admin web form: (1) localhost requests are filtered out, and (2) the admin string is filtered out. A clever adversary can bypass these by (1) using a localhost alias such as 2130706433, and (2) url-encoding the admin portion of the uri, which is subject to a superfluous url-decode.

**Remediation:** Replace the blacklist with a whitelist to disallow creative attacks such as the above.



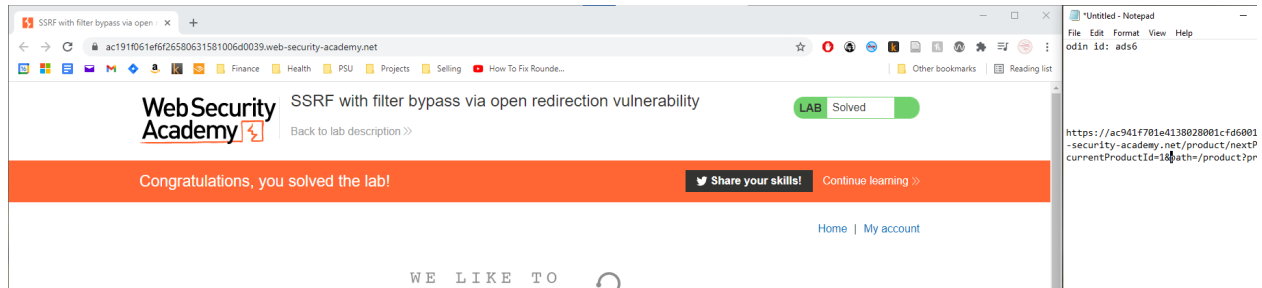
### 4. ssrf (4)

- What is the `page` that implements the redirect?  
product/nextProduct
- What `parameter` specifies the URI that the page uses to redirect the browser to?  
path
- Take a screenshot showing completion of the level that includes your OdinId

**Vulnerability:** The product navigation url includes an unchecked redirect parameter that can be used to bypass the server-side redirect filters from the ticker link we explored in previous

problems. By sending this redirect to the server we can thus force it to open up the admin console which would normally be available internally only.

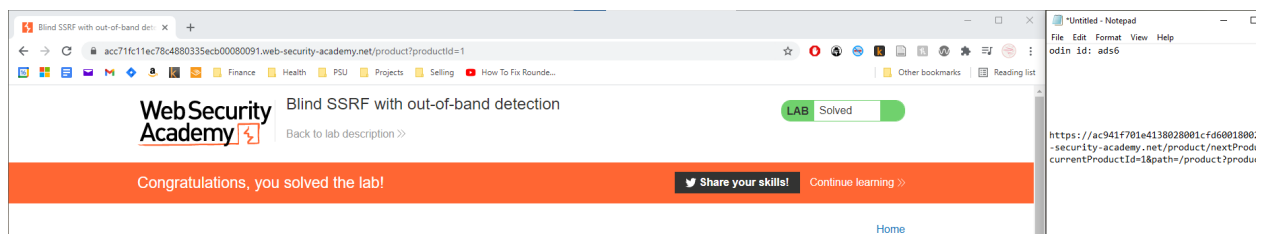
**Remediation:** Subject the path= parameter to the same checks as the SSR stockApi url.



## 5. ssrf/blind

**Vulnerability:** This site blindly fetches any url specified in the referrer header when loading a product page. All the adversary must do is specify that header to initiate the SSR attack.

**Remediation:** The referrer header must be validated before being processed.



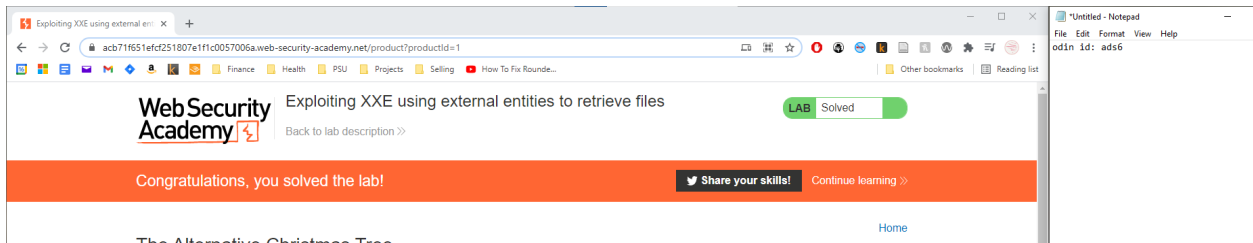
## Lab 1.7

### 1. xxe (1)

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

**Vulnerability:** The xml payload is not validated before execution, meaning that the adversary can request all kinds of sensitive files as includes.

**Remediation:** Don't allow includes in the payload.

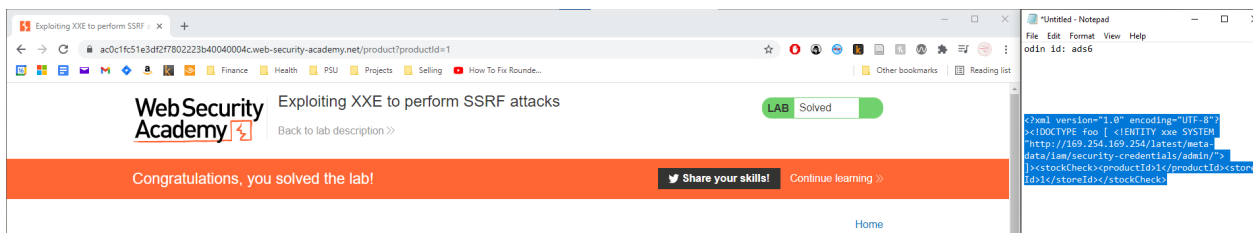


## 2. xxe (2)

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

**Vulnerability:** The XXE includes allow the adversary to request potentially any file hosted internally which is visible to the server. In this case, we retrieve sensitive amazon aws credentials.

**Remediation:** Strip out the DTD blocks before proceeding with xxe processing.

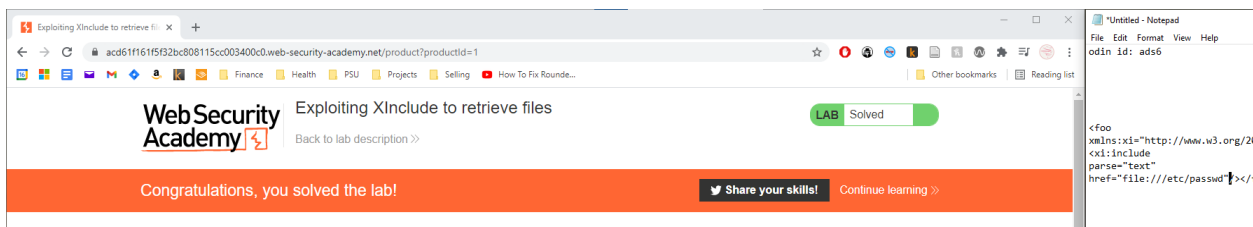


## 3. xxe (3)

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

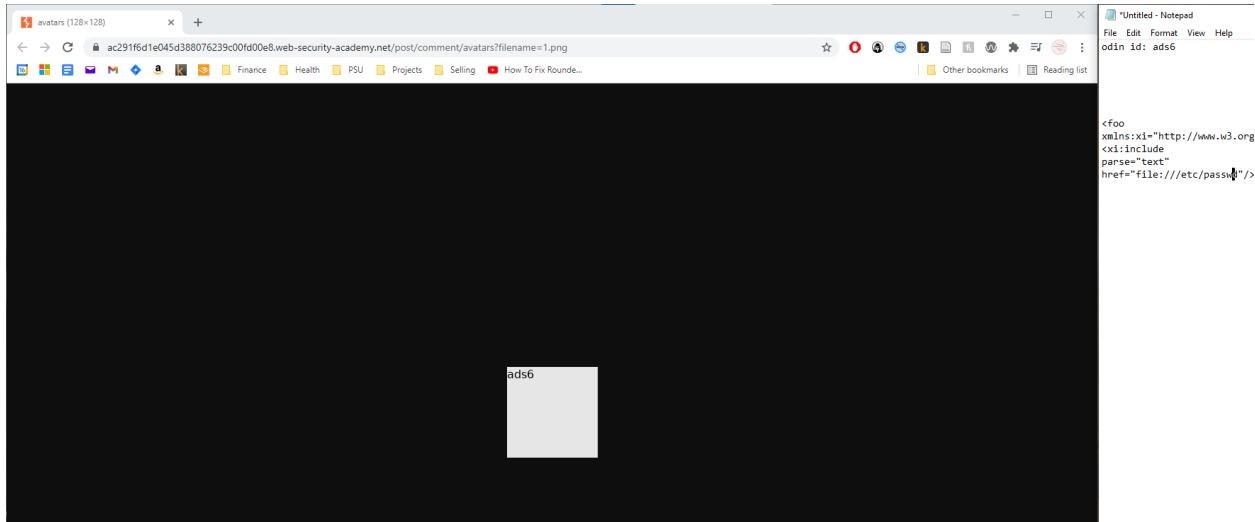
**Vulnerability:** Since the unfiltered productId data is incorporated into an XML document during server side processing, the adversary can trigger an unsanitized error message from the parser, which will spill all sorts of sensitive data.

**Remediation:** Check the user supplied data before incorporating into an xml document.



## 4. xxe (4)

- Take a screenshot of the image URL and the image for your lab notebook



## 5. -

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

**Vulnerability:** User uploaded svg files are processed as xml, allowing the adversary to insert a DTD requesting potentially any data in the system, which can subsequently be exfiltrated by simply viewing the image.

**Remediation:** Strip out DTDs from the svg uploads before processing.

