



# Vulnerable Web Application



*Sushil*



# Table of Content

## Section 1 (Static code scan)

- Issue: B106
- Issue: B108
- Issue: B303
- Issue: B311
- Issue: B320
- Issue: B603
- Issue: B703

## Section 2

- Assess the Web Application

## Section 3 (Security Report)

- Broken Authentication
- Broken Access
- Sensitive Data Exposure
- Cryptographic failures
- HTML Injection
- Cross-Site Scripting (XSS)



# Section One:

# Static Code Scan

---



# Vulnerability Risk Matrix

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

- The above chart represents a Vulnerability Risk Matrix - -
- which helps assess the overall risk severity of a vulnerability by combining its impact and likelihood.
- Impact measures how severe the consequences would be if the vulnerability is exploited.
- Likelihood represents how probable it is that the vulnerability will be exploited.



# Issue: B106

```
>> Issue: [B106:hardcoded_password_funcarg] Possible hardcoded password: 'mysecurepassword'
```

```
Location: SampleCode/init_db.py:14
```

```
13         def open(self):
14             self.conn = psycopg2.connect(user = "webappuser",
15                                           password = "mysecurepassword",
16                                           host = "localhost",
17                                           port = "5432",
18                                           database = "website")
19             self.cursor = self.conn.cursor()
```

<b>Severity:</b>	<i>High</i>
<b>OWASP TOP 10 reference:</b>	<i>A07:2021-Identification and Authentication Failure</i>

## **Remediation recommendation**

*Keep your database login details out of the actual code. Instead, store them safely in environment variables –this is a common and secure practice used in the industry. It helps protect sensitive data and make you app easier to manage across different environments. This way, even if someone sees your code, your credentials stay hidden. It also mitigates risks related to CWE-259(use of hardcoded password ) and A07:2021-Identification and authentication failures .*



# Issue: B108

>> Issue: [B108:hardcoded\_tmp\_directory] Use of a hardcoded temporary directory.

Location: SampleCode/temp\_file.py:19

```
18 def createTempFile(data):
19     temp_path = "/tmp/tempfile.txt"
20     with open(temp_path, 'w') as temp_file:
21         temp_file.write(data)
22     return temp_path
```

<b>Severity:</b>	Medium
<b>OWASP TOP 10 reference:</b>	A01:2021 - Broken Access control A04:2021 - Insecure Design A05-2021 - Security Misconfiguration
<b>Remediation recommendation</b>	
<p>Use of a hard coded temporary directory can lead to several issues including Information disclosure, Data tampering, Denial of Service, This may compromise the application's confidentiality, integrity, or availability.</p> <pre>temp_path = "/tmp/tempfile.txt"</pre> <p>This make the temporary directory &amp; file static , which can be predicted or brute forced .Instead of this create function to create temp file with random name without a predefined directory or in restricted directory.</p>	



# Issue: B108

## **Remediation recommendation**

*Developer must attach to use secure coding practices. Temporary file name have to be generated randomly , making it difficult to predict.manage*

*temporary files in more privileged environment , this could involve creating the files in directory with restricted access.*

*Enforce strict security policies around file operations.*

*Temporary file have to deleted automatically after the process exit.*



## Issue: B303

>> Issue: [B303:blacklist] Use of insecure MD2, MD4, MD5, or SHA1 hash function.

Location: SampleCode/create\_customer.py:23

```
22         self.email = email
23         self.password = hashlib.md5(password.encode('utf-
24         self.banner = safestring.mark_safe(banner)
```

<b>Severity:</b>	<i>Medium</i>
<b>OWASP TOP 10 reference:</b>	<i>A02:2021- Cryptographic Failures A07:2021- Identification and Authentication Failures</i>

### **Remediation recommendation**

*The code uses the MD5 algorithm to store or process the passwords. MD5 is deprecated cryptographic function that is no longer secure due to known collision and preimage attacks. When used for password hashing, It allows attackers to quickly compete hash collisions or reverse hashes using rainbow tables and brute-force tools.*

*This flaw also increases the risk of broken authentication, since weak hashes make it trivial for attackers to recover passwords and access user accounts .*





# Issue: B311

>> Issue: [B311:blacklist] Standard pseudo-random generators are not suitable for security/cryptographic purposes.

Location: SampleCode/init\_db.py:40

```
39         letters = string.ascii_lowercase
40         result_str = ''.join(random.choice(letters) for i in
range(length))
41         return result_str
```

**Severity:**

Low

**OWASP TOP 10 reference:**

A02:2021-Cryptographic Failures

## **Remediation recommendation**

*The code's using Python's random.choice() which anyone can predict if they know the seed value, so hackers can basically guess what "random" values your system will generate next - not good for security stuff like passwords or tokens. Replace it with the secrets module which taps into your computer's most secure randomness source, making predictions impossible even for attackers with supercomputers. Just swap random.choice(letters) with secrets.choice(letters) in line 40 of init\_db.py - literally a one-word change but massively upgrades security. The secrets module is built specifically for creating passwords, OTPs, session tokens, and other security-critical stuff where true randomness matters. Bottom line: if it's security-related (authentication, credentials, tokens), always use secrets; if it's just for games or testing, random is fine write this more simple.*



# Issue: B320

>> Issue: [B320:blacklist] Using `lxml.etree.fromstring` to parse untrusted XML data is known to be vulnerable to XML attacks. Replace `lxml.etree.fromstring` with its `defusedxml` equivalent function.

Location: `SampleCode/fix_customer_orders.py:11`

```
10     def customerOrdersXML():
11         root = lxml.etree.fromstring(xmlString)
12         root = fromstring(xmlString)
```

<b>Severity:</b>	<i>High</i>
<b>OWASP TOP 10 reference:</b>	<i>A05:2021-Security Misconfiguration</i>
<b>Remediation recommendation</b>	
<i>"lxml.etree.fromstring(xmlString)"</i>	
<i>This allow to parse a string containing XML into an XML element tree without any input validation or proper sanitization. Improper use "lxml" lead to several XML vulnerabilities such as XXE (XML External Entity injection). Billion Laughs / Entity Expansion attacks .</i>	
<i>To avoid this implement "defusedxml.lxml" instead of "lxml" , and implement proper input sanitization. Implement the input size limit.</i>	



## Issue: B603

>> Issue: [B603:subprocess\_without\_shell\_equals\_true] subprocess call - check for execution of untrusted input.

Location: SampleCode/onLogin.py:8

```
7         def process(self, user, startupcmd):
8             p = subprocess.Popen([startupcmd],
9                                   stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
10            r = p.communicate()[0]
```

<b>Severity:</b>	Low (False Positive)
<b>OWASP TOP 10 reference:</b>	N/A
<b>Remediation recommendation</b>	
<p><i>In this parameter is set to "shell=false" . So it state that input parsing won't get execute in shell.</i></p> <p><i>if "startupcmd" argument is fully validated by the application not by the user , then it will repot as false positive</i></p>	



# Issue: B703

>> Issue: [B703:django\_mark\_safe] Potential XSS on mark\_safe function.

Location: SampleCode/create\_customer.py:24

```
23         self.password = hashlib.md5(password.encode('utf-8')).hexdigest()
24         self.banner = safestring.mark_safe(banner)
25
```

**Severity:**

*Medium*

**OWASP TOP 10 reference:**

A03:2021 - Injection (specifically Cross-Site Scripting - XSS)

## **Remediation recommendation**

The code uses Django's mark\_safe() function on line 23 of create\_customer.py to bypass Django's automatic HTML escaping for the password field.

When mark\_safe() is applied to user-controlled input, attackers can inject malicious JavaScript that executes in victims' browsers. XSS is one of the most common web application vulnerabilities, allowing attackers to inject client-side scripts into web pages viewed by other users.

Remove mark\_safe() *because* MD5 hashes are alphanumeric and don't need



# Section Two:

## Assess the Web Application

---

Integrated Application Security Services for ATCI

Course 3 Application Security

6 Mitigation and Verification

7 Vulnerable Web Application

7.1 Overview

7.2 Getting Started

7.3 Web Application Environment

7.4 Part 1 - Static Code Scan

7.5 Part 2 - Assess the Web Application

7.6 Part 3 - Create a Security Report

7.7 Project Rubric

7.8 Submit Project

Please do not upload sensitive information to workspaces.

workspace

VulnWebApp

tools

app.log

startup.sh

No Open Files

OPEN FILE

Terminal 1

```
workspace root$ ./startup.sh
```

Settings Usage Links Help

- Start the workspace. Then open the Terminal
- Then run command **./startup.sh**
- Then run **cd VulnWebApp && ./run\_site.sh**
- Then access the lab from links

Integrated Application Security Services for ATCI

Course 3 Application Security

6 Mitigation and Verification

7 Vulnerable Web Application

7.1 Overview

7.2 Getting Started

7.3 Web Application Environment

7.4 Part 1 - Static Code Scan

7.5 Part 2 - Assess the Web Application

7.6 Part 3 - Create a Security Report

7.7 Project Rubric

7.8 Submit Project

workspace

VulnWebApp

tools

app.log

startup.sh

No Open Files

OPEN FILE

Terminal 1

```
Uninstalling requests-2.25.1:
Successfully uninstalled requests-2.25.1
Successfully installed Flask-1.1.2 Flask-Breadcrumbs-0.5.1 Flask-Menu-0.7.2 Jinja2-2.11.2 MarkupSafe-1.1.1 Werkzeug-1.0.1 certifi-2020.6.20 click-7.1.2 idna-2.10 itdangerous-1.1.0 pycopg2-binary-2.8.6 python-dotenv-0.14.0 requests-2.24.0 six-1.15.0 urllib3-1.25.10
workspace root$ cd VulnWebApp/ && ./run_site.sh

* Serving Flask-SocketIO app "Site"
* Forcing debug mode off
* Serving Flask app "Site"
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:3000/ (Press CTRL+C to quit)
```

Settings Usage Links Help

← Previous Next →

Give Page Feedback



# Section Three:

# Security Report

---



# Broken Authentication

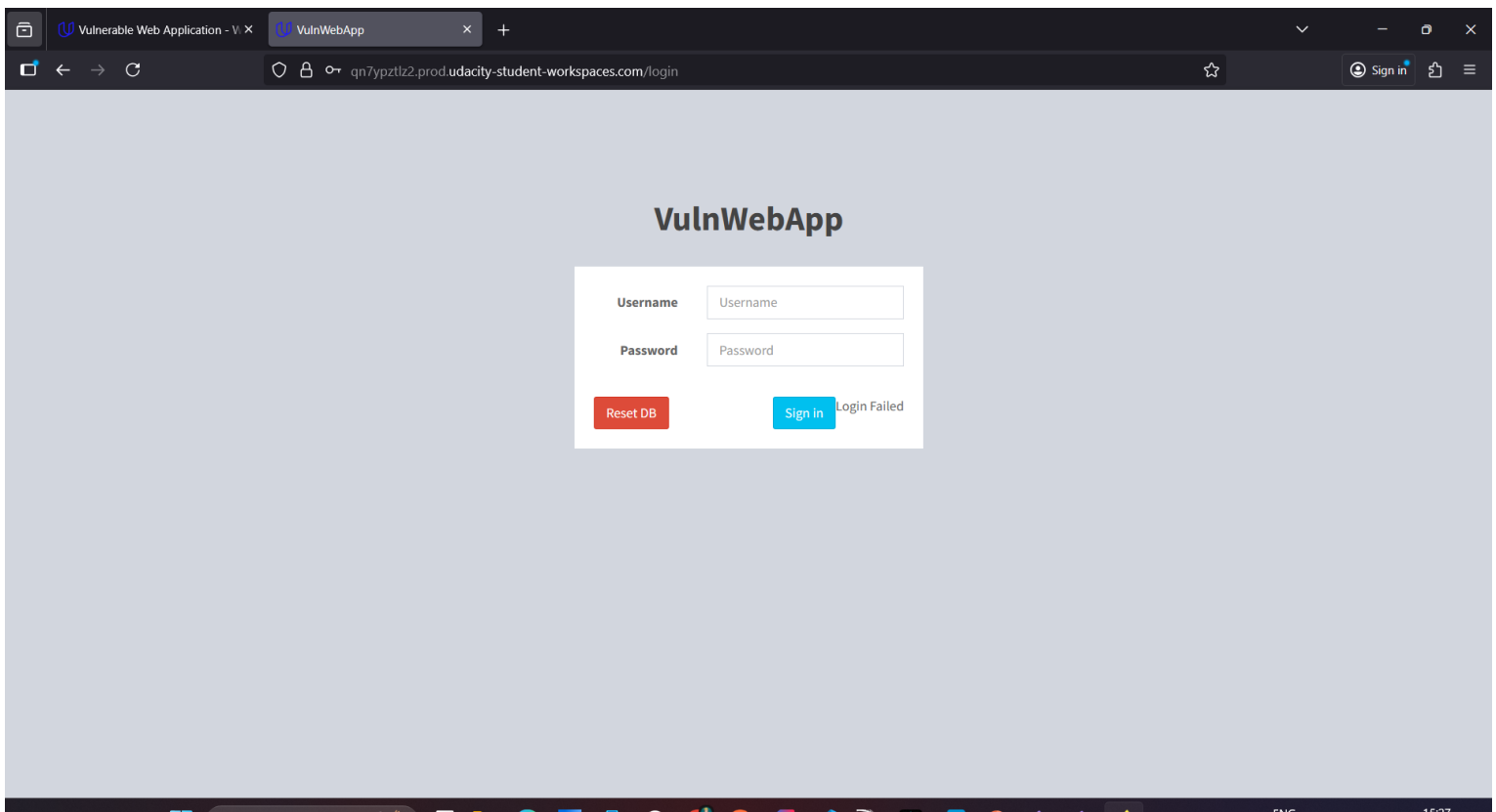
<b>Severity:</b>	<i>High</i>
<b>OWASP TOP 10 reference:</b>	A07:2021 – Identification and Authentication Failures
<b><i>Vulnerability Explanation</i></b>	
<p>The login functionality is vulnerable to brute-force attacks. An attacker can make unlimited authentication attempts without being blocked or delayed. This allows guessing of valid credentials and may lead to unauthorized access to user accounts or administrative systems.</p>	
<b><i>Remediation recommendation</i></b>	
<p><i>We can Enable MFA (Multi-Factor Authentication): Require OTP for login. We can also Rate Limiting for the restrict the number of login attempts as IP or account (e.g. 5 attempt in 1 hour) We can use Monitor and Alert log all failed login attempts and generate alert for suspicious patterns. We can use password policy for enforce strong password requirements and check against known breached passwords. We also add Implement Account Lock for temporarily lock or delay in login attempts after repeated failures.</i></p>	





# Broken Authentication

step-by-step walkthrough



- After visiting the website, it open a **login** page (/login)
- Press "**Reset**" before perform attack.

**VulnWebApp**

Username:

Password:

Reset DB Sign in Login Failed

Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms	2.56 s
200	POST	qn7ypztiz2.prod.udacity-studen...	login	jquery.js:8510 (document)	html	1.93 kB	3.90 kB	460 ms	
200	GET	qn7ypztiz2.prod.udacity-studen-w...	bootstrap.css	stylesheet	css	cached	-1 B	0 ms	
200	GET	qn7ypztiz2.prod.udacity-studen-w...	font-awesome.css	stylesheet	css	cached	-1 B	0 ms	
200	GET	qn7ypztiz2.prod.udacity-studen-w...	ionicons.css	stylesheet	css	cached	-1 B	0 ms	
200	GET	qn7ypztiz2.prod.udacity-studen-w...	adminlte.css	stylesheet	css	cached	-1 B	0 ms	
200	GET	qn7ypztiz2.prod.udacity-studen-w...	icheck.blue.css	stylesheet	css	cached	-1 B	0 ms	
200	GET	fonts.googleapis.com	css?family=Source+Sans+Pro:300,400,600,700,300italic,400italic,600italic	stylesheet	css	cached	-1 B	0 ms	
200	GET	qn7ypztiz2.prod.udacity-studen...	jquery.js	script	js	cached	0 B	0 ms	
200	GET	qn7ypztiz2.prod.udacity-studen...	jquery-ui.min.js	script	js	cached	0 B	0 ms	
200	GET	qn7ypztiz2.prod.udacity-studen...	bootstrap.js	script	js	cached	0 B	0 ms	

14 requests | 19.30 kB / 1.93 kB transferred | Finish: 2.03 s | DOMContentLoaded: 1.13 s | load: 1.13 s

- Open the developer tool & network tab to see the **post request** of the login

**VulnWebApp**

Username:

Password:

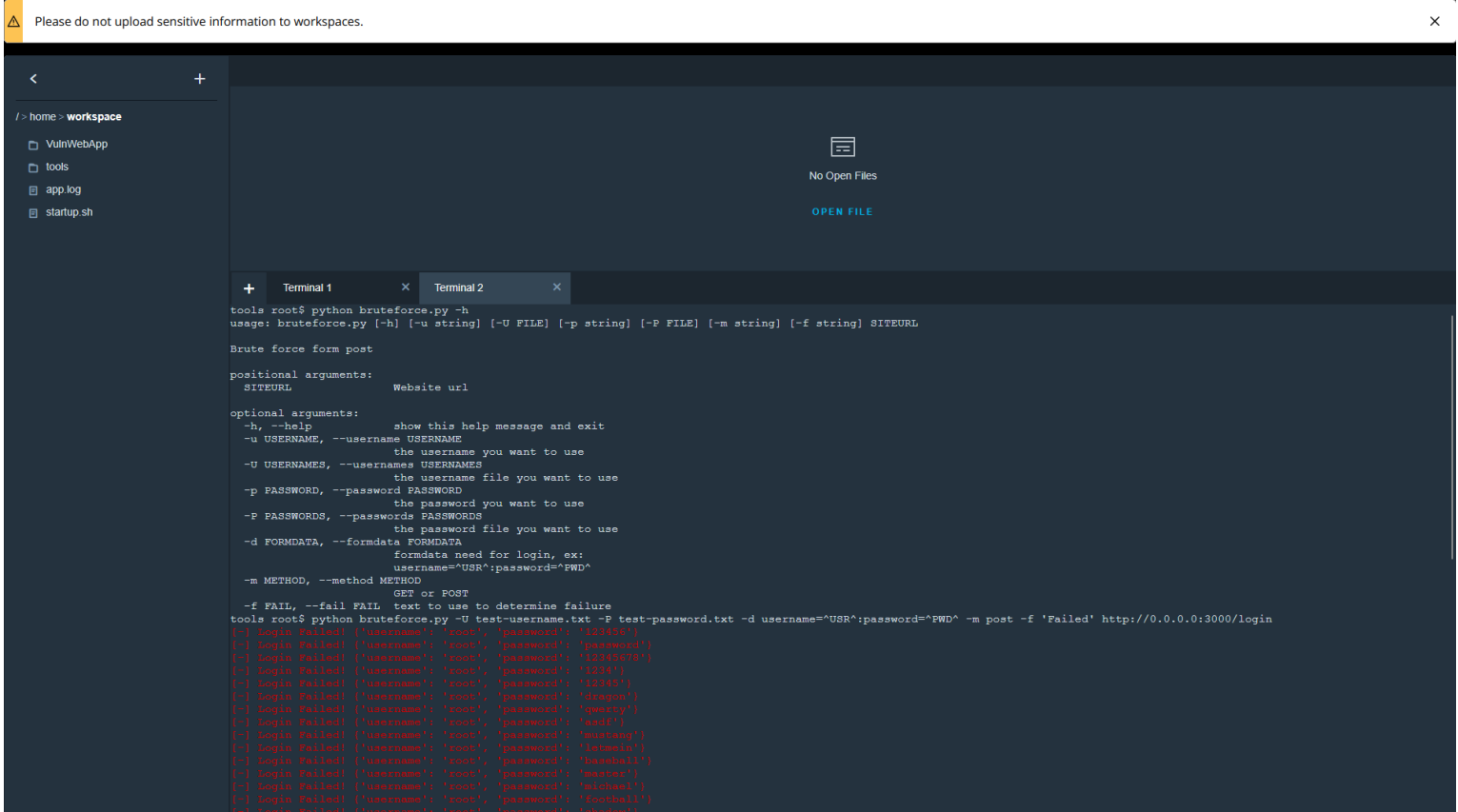
Reset DB Sign in Login Failed

Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms	2.56 s
200	POST	qn7ypztiz2.prod.ud...	login	jquery.js:8510 (docume...	html	1.93 kB	3.90 kB	460 ms	
200	GET	qn7ypztiz2.prod.udact...	bootstrap.css	stylesheet	css	cached	-1 B	0 ms	
200	GET	qn7ypztiz2.prod.udact...	font-awesome.css	stylesheet	css	cached	-1 B	0 ms	
200	GET	qn7ypztiz2.prod.udact...	ionicons.css	stylesheet	css	cached	-1 B	0 ms	
200	GET	qn7ypztiz2.prod.udact...	adminlte.css	stylesheet	css	cached	-1 B	0 ms	
200	GET	qn7ypztiz2.prod.udact...	icheck.blue.css	stylesheet	css	cached	-1 B	0 ms	
200	GET	fonts.googleapis.com	css?family=Source+Sans+Pro:300,400,600,700,300italic,400italic,600italic,600italic	stylesheet	css	cached	-1 B	0 ms	
200	GET	qn7ypztiz2.prod.ud...	jquery.js	script	js	cached	0 B	0 ms	
200	GET	qn7ypztiz2.prod.ud...	jquery-ui.min.js	script	js	cached	0 B	0 ms	
200	GET	qn7ypztiz2.prod.ud...	bootstrap.js	script	js	cached	0 B	0 ms	

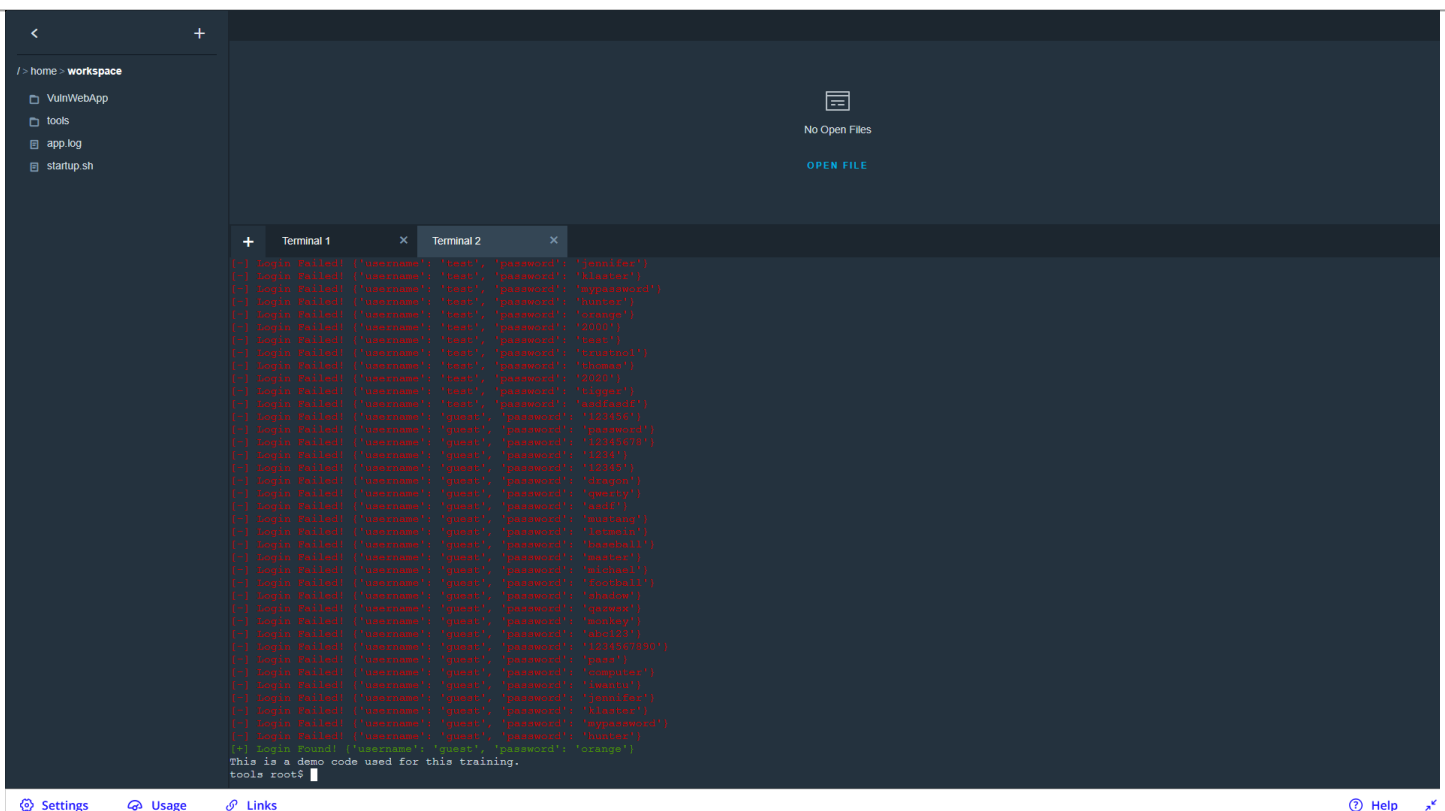
14 requests | 19.30 kB / 1.93 kB transferred | Finish: 2.03 s | DOMContentLoaded: 1.13 s | load: 1.13 s

Request payload: username=test&password=test

- It show the request payload “username=test&password=test” ,
- In these test credentials it gives message “**Login Failed**”



- By using bruteforce tool “bruteforce.py”, craft the command by login payload credentials got from previous screenshot.
- **python bruteforce.py -U test-username.txt -P test-password.txt -d username=^USR^:password=^PWD^ -m post -f 'Failed' <http://0.0.0.0:3000/login>**
- Use the given username and password file

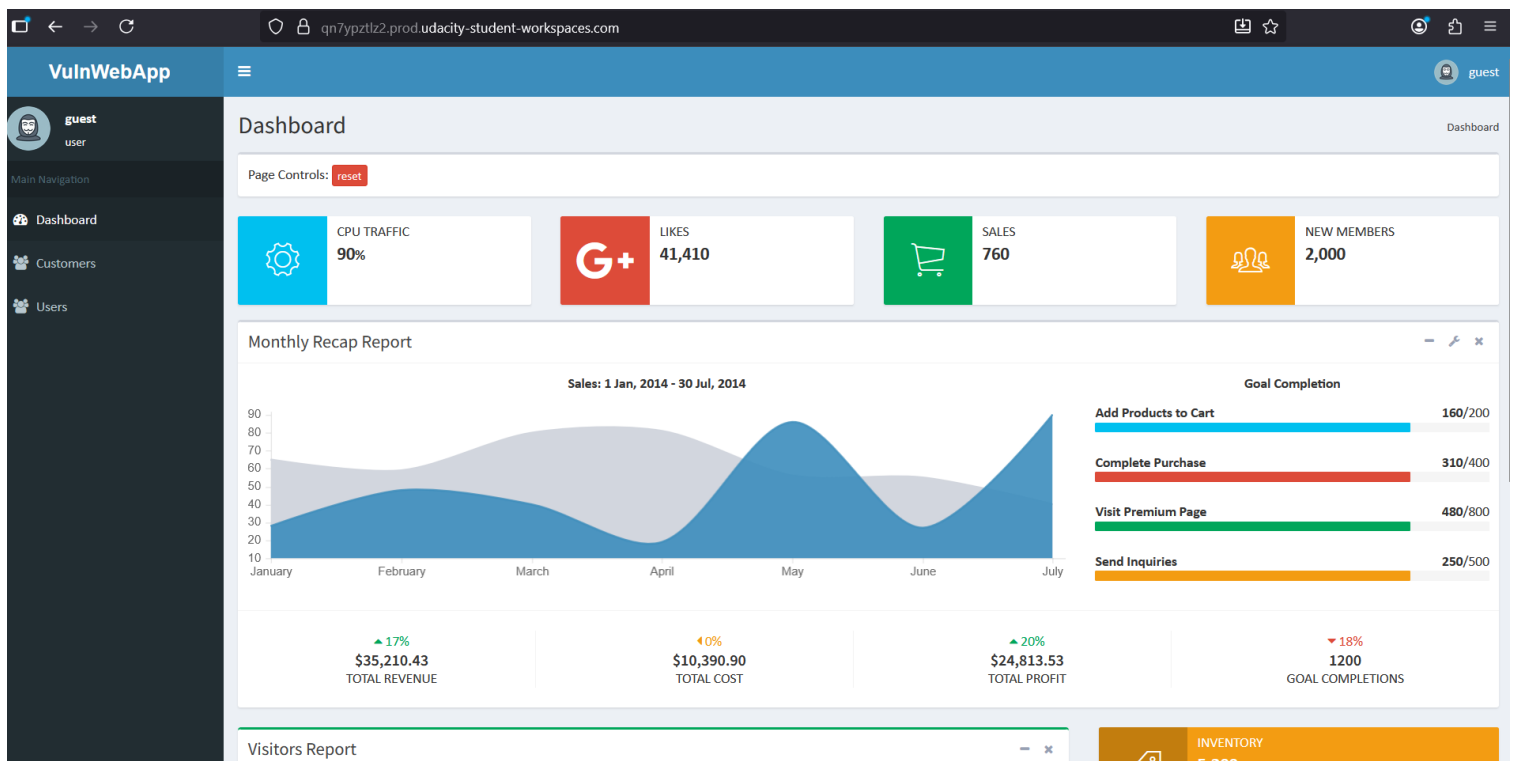




```
[~] Login Failed! ('username': 'guest', 'password': '1234567890')
[~] Login Failed! ('username': 'guest', 'password': 'pass')
[~] Login Failed! ('username': 'guest', 'password': 'computer')
[~] Login Failed! ('username': 'guest', 'password': 'iwantu')
[~] Login Failed! ('username': 'guest', 'password': 'jennifer')
[~] Login Failed! ('username': 'guest', 'password': 'klaster')
[~] Login Failed! ('username': 'guest', 'password': 'mypassword')
[~] Login Failed! ('username': 'guest', 'password': 'hunter')
[+] Login Found! ('username': 'guest', 'password': 'orange')
This is a demo code used for this training.
tools root$
```

[Usage](#) [Links](#)

- Brute force attack got successful and get the username and password
- **Username - Guest**
- **Password - orange**
- By these got login in website.





# Broken Access

<b>Severity:</b>	<i>Critical</i>
<b>OWASP TOP 10 reference:</b>	<i>A01:2021 – Broken Access Control</i>
<b>Vulnerability Explanation</b>	
<p><i>The application uses client-side cookies <b>authInfo</b> to store sensitive user data such as role or authorization level.</i></p> <p><i>If this value is not validated or signed securely on the server, an attacker can manipulate the cookie (e.g., change "role=user" to "role=admin") and gain unauthorized access to admin functionalities. This is a Broken Access Control issue because the system fails to enforce proper access restrictions on the server side</i></p>	
<b>Remediation recommendation</b>	
<p><i>Do not store roles or permissions in client-side cookies or tokens without proper integrity protection because Always validate roles on the server side. Implement server-side access control checks for every endpoint or action because Each request should verify the user's actual role and privileges from the server's session or database. Apply the Principle of Least Privilege because Grant only the minimum required permissions to each role. Add server-side validation and logging because Detect and block suspicious role changes.</i></p>	



# Broken Access

## Step-by-step walkthrough

- We are logged in as guest user.
- Go to profile page. One developer tool then network tab
- Refresh page.

The screenshot shows the VulnWebApp interface. The top navigation bar includes the app name 'VulnWebApp', a menu icon, and a user profile icon labeled 'guest'. The left sidebar contains navigation links: 'Dashboard', 'Customers', and 'Users'. The main content area displays the 'User Profile' for 'John Doe', a 'User' with 1,322 followers. It includes a 'Messages' section with a message from 'administrator' and a 'Send message' form. Below the profile, the 'Network' tab of the browser's developer tools is open, showing a list of requests. The selected request is a GET request to the URL 'https://qn7ypztiz2.prod.udacity-student-workspaces.com/profile/userlist/2?\_id=1760192429345', which returned a 200 status and a JSON response of 406 B. The response headers show 'alt-svc: h3=\":443\"; ma=86400', 'cf-cache-status: DYNAMIC', and 'cf-ray: 98ceff9ebcf6a79e-DEL'.

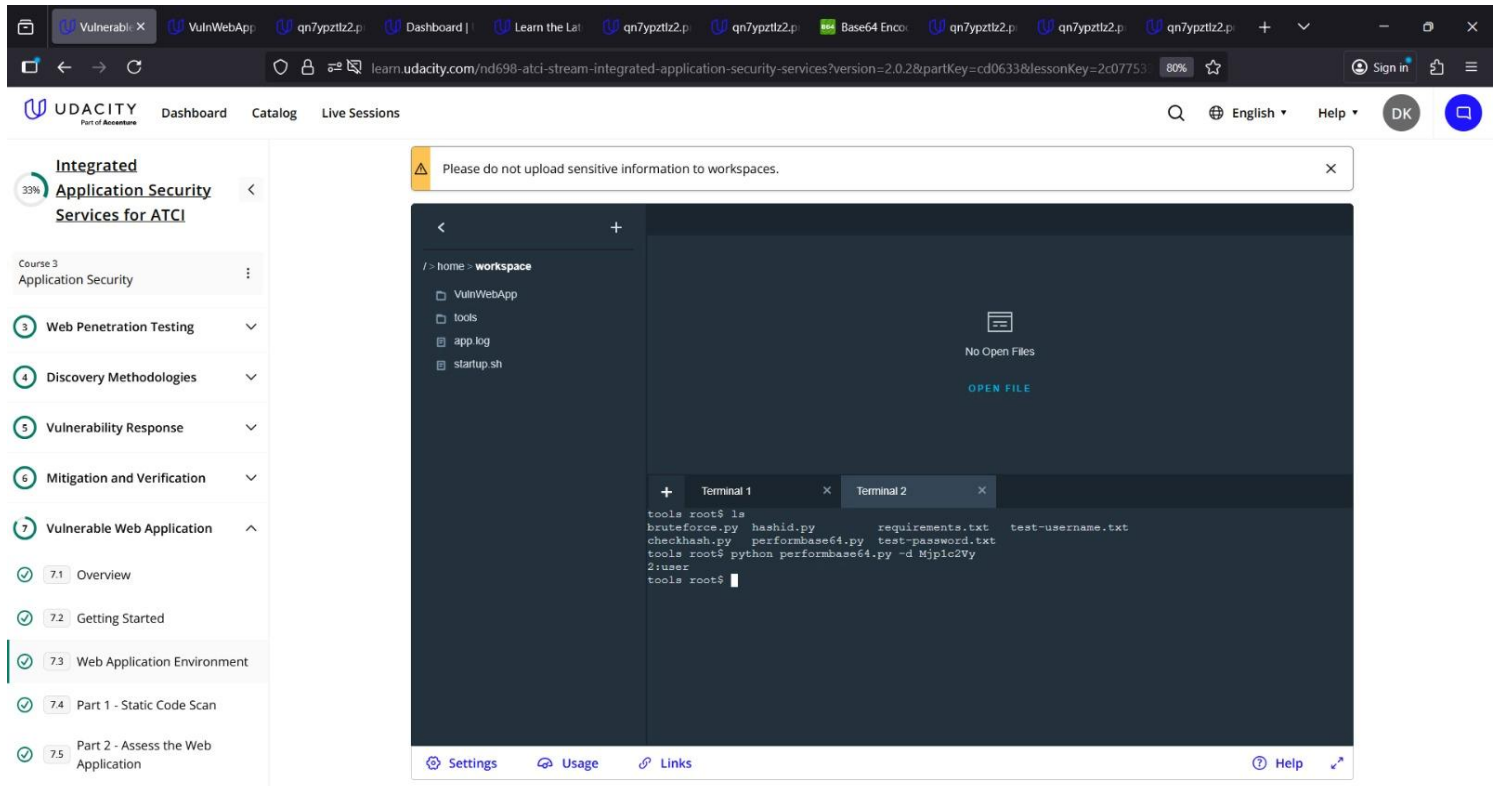
Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	qn7ypztiz2.prod.ud...	fastclick.js	script	js	cached	0 B
200	GET	qn7ypztiz2.prod.ud...	jquery.KeepAlive.js	script	js	cached	0 B
200	GET	qn7ypztiz2.prod.ud...	underscore.js	script	js	cached	0 B
200	GET	cdnjs.cloudflare.com	jquery.validate.min.js	script	js	cached	0 B
200	GET	cdnjs.cloudflare.com	additional-methods.min.js	script	js	cached	0 B
200	GET	qn7ypztiz2.prod.ud...	adminlte.js	script	js	cached	0 B
200	GET	qn7ypztiz2.prod.ud...	favicon.ico	FaviconLoader.async.js...	x-icon	cached	15.41 kB
200	GET	qn7ypztiz2.prod.ud...	2?_id=1760192429344	jquery.js:9837 (xhr)	json	510 B	115 B
200	GET	qn7ypztiz2.prod.ud...	2?_id=1760192429345	jquery.js:9837 (xhr)	json	406 B	22 B
200	GET	qn7ypztiz2.prod.ud...	user.png	img	png	cached	41.09 kB

- We can see a request below which get the api **"/profile/userlist"**
- Which show the data of user id

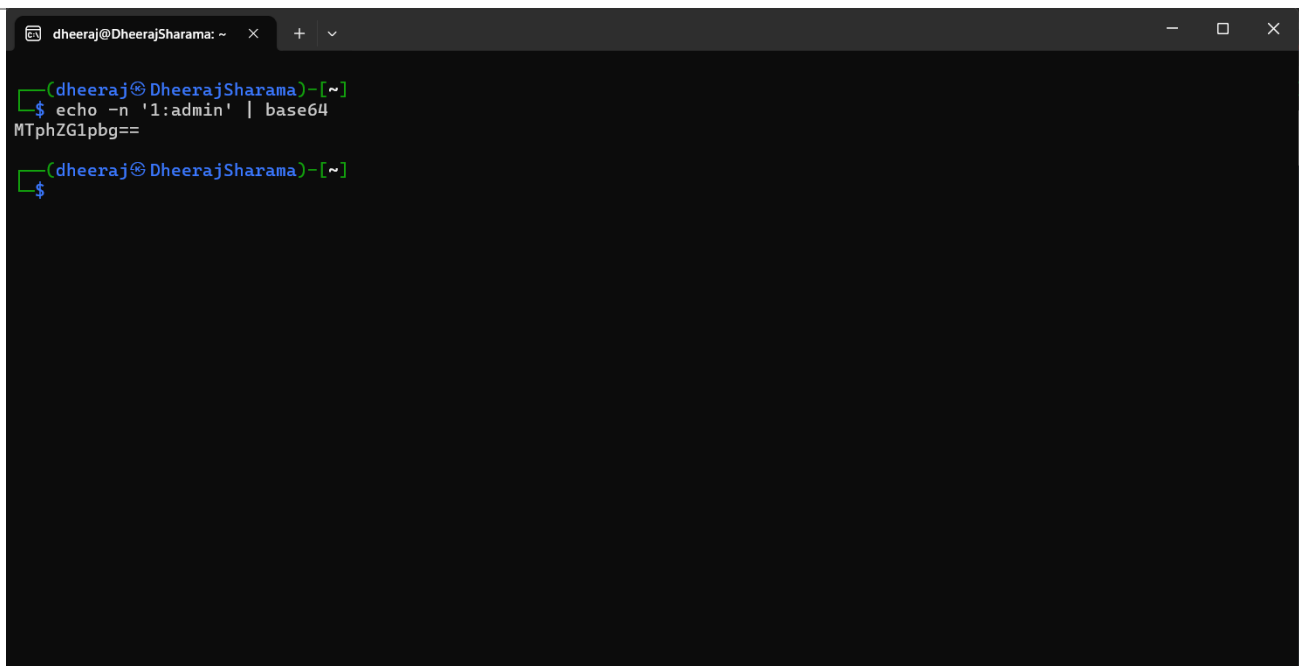




- Use the give tool **performbase64.py** to decrypt base64
- Its value is **2:user**



- Now make **authinfo** cookie for the admin
- Admin id we have founded in **profile/userlist/0**
- Admin cookie will be **1:admin**
- Then encode this to base64









# Sensitive Data Exposure

<b>Severity:</b>	High
<b>OWASP TOP 10 reference:</b>	A02:2021-Cryptographic Failures
<b>Vulnerability Explanation</b>	
<p>The API <code>/customers/id/1</code> is returning user details without proper authentication or access control. Even though the password seems hashed, exposing it in a public or client-facing API is still dangerous. Attackers can use brute-force or rainbow table attacks to recover the original password. It leaks identifiable customer data (names, usernames).</p>	
<b>Remediation recommendation</b>	
<p>Add authentication &amp; authorization because Require a valid user token or session before anyone can access <code>/customers/id/....</code>. Don't expose sensitive data in API responses because Only return the data your frontend actually needs (e.g., not password hashes, tokens, or internal IDs). Hash and salt passwords securely on the server (but never return them). Use proper access control because a user should only see their own data, not anyone else's. securing endpoints, limiting what's returned, and protecting stored data.</p>	



# Sensitive Data Exposure

step-by-step walkthrough

- We got the Admin access earlier
- Now we can see the Customers list

Customers

Page Controls [reset](#)

Customers List

ID	First Name	Last Name	Username	Options
1	paul	doe	pdoe	<a href="#">View</a>
2	jake	doe	jdoe	<a href="#">View</a>
3	dave	doe	ddoe	<a href="#">View</a>
4	mike	doe	mdoe	<a href="#">View</a>
5	nick	doe	ndoe	<a href="#">View</a>

Copyright © 2020 Test Company LLC. All rights reserved. Ver. 1.0.0.0

- We look the network request of this page
- It request to API **/customers/id**



ID	First Name	Last Name	Username	Options
1	paul	doe	pdoe	<a href="#">View</a>
2	jake	doe	jdoe	<a href="#">View</a>
3	dave	doe	ddoe	<a href="#">View</a>
4	mike	doe	mdoe	<a href="#">View</a>
5	nick	doe	ndoe	<a href="#">View</a>

- Opening the request to new tab will hit the Api **/customers/id**
- It will show the Json data of customers

```
{
  "id": 1,
  "first_name": "paul",
  "last_name": "doe",
  "username": "pdoe"
}
```



- Now go to specific customer id , which reveal the sensitive information
- **Hash of the Password**

A screenshot of a web browser window. The address bar shows the URL `https://qn7ypztlz2.prod.udacity-student-workspaces.com/customers/id/1`. The browser's developer tools are open, displaying a JSON response in the console. The response is a single object with the following fields: `id` (1), `name` ("jake"), `email` ("jdoe"), and `password_hash` ("d8578edf9459ce96fbc5b976a58c5ca4"). The console has tabs for "Raw" and "Parsed", with "Parsed" selected.

```
[{"id": 1, "name": "jake", "email": "jdoe", "password_hash": "d8578edf9459ce96fbc5b976a58c5ca4"}]
```

A screenshot of a web browser window. The address bar shows the URL `qn7ypztlz2.prod.udacity-student-workspaces.com/customers/id/2`. The browser's developer tools are open, displaying a JSON response in the console. The response is an array with one object containing the fields: `id` (2), `name` ("jake"), `email` ("jdoe"), and `password_hash` ("5f4dcc3b5aa765d61d8327deb882cf99"). The console has a "Pretty print" checkbox checked.

```
[{"id": 2, "name": "jake", "email": "jdoe", "password_hash": "5f4dcc3b5aa765d61d8327deb882cf99"}]
```

A screenshot of a web browser window. The address bar shows the URL `qn7ypztlz2.prod.udacity-student-workspaces.com/customers/id/4`. The browser's developer tools are open, displaying a JSON response in the console. The response is an array with one object containing the fields: `id` (4), `name` ("mike"), `email` ("mdoe"), and `password_hash` ("8621ffdbc5698829397d97767ac13db3"). The console has a "Pretty print" checkbox checked.

```
[{"id": 4, "name": "mike", "email": "mdoe", "password_hash": "8621ffdbc5698829397d97767ac13db3"}]
```



# Cryptographic failures

<b>Severity:</b>	<i>High</i>
<b>OWASP TOP 10 reference:</b>	<b>A02:2021 – Cryptographic Failures</b>
<b><i>Vulnerability Explanation</i></b>	
<p>The system is using weak hash like MD5 to store passwords It might look secure because the password turns into a long string like d8578edf8458ce06fbc5bb76a58c5ca4, but in reality, hackers can crack it in seconds using free online tools or pre-made hash databases. So even though it seems hidden, that string can be quickly turned back into “qwerty” — meaning your users’ passwords are practically lying in plain sight.</p>	
<b><i>Remediation recommendation</i></b>	
<p><i>Stop using MD5 or SHA1 because they are old and insecure.</i> <i>Enforce strong password rules for users — avoid weak passwords like “qwerty”.</i> <i>Add a unique salt to each password before hashing.</i> <i>Use strong, modern password hashing algorithms such as: bcrypt.</i> <i>Re-hash old passwords because when users log in next time, re-hash their password with a strong algorithm.</i></p>	



# Cryptographic failures

step-by-step walkthrough

- From specific request to customer ID like **/customer/id/1**
- It give us customer data with their Hash Password

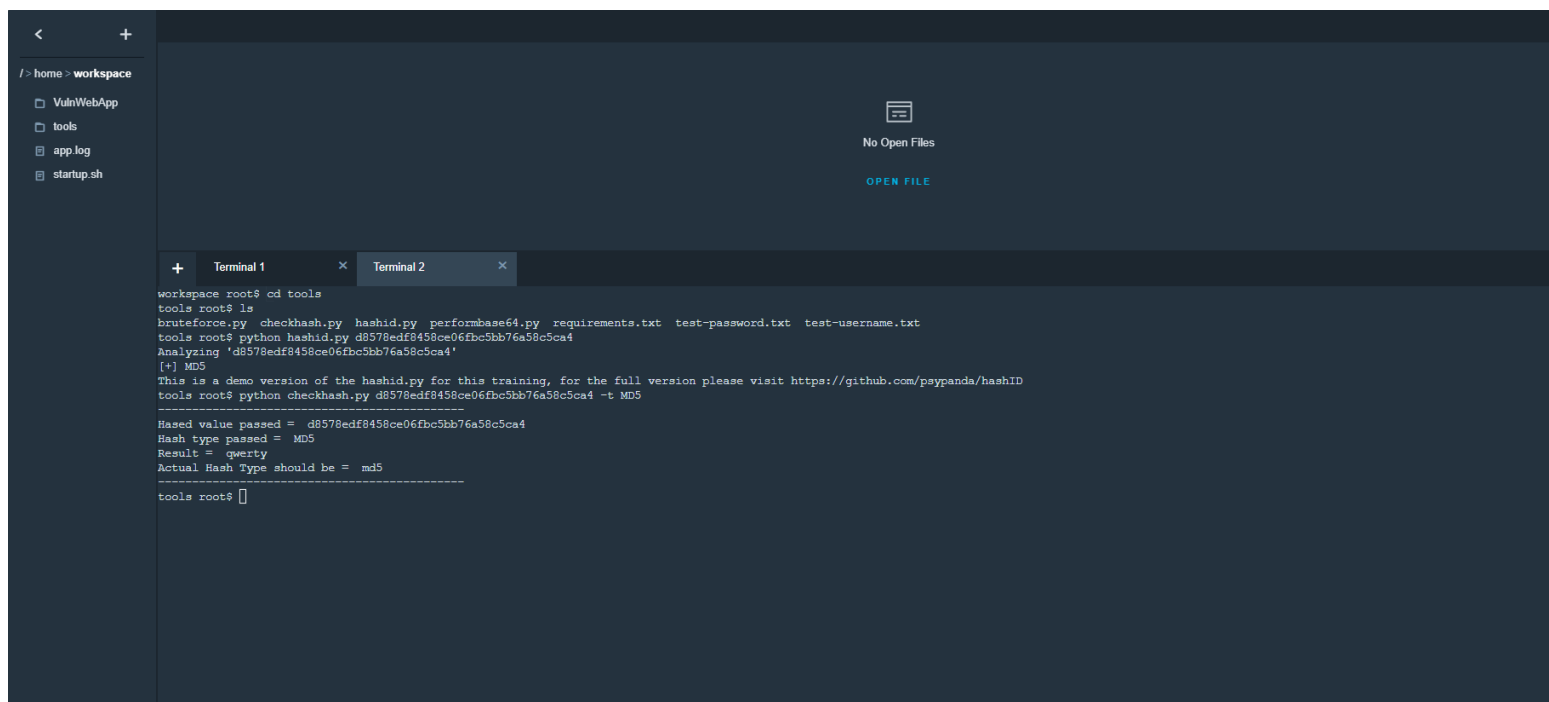
The screenshot shows a web browser window with the URL `https://qn7ypztz2.prod.udacity-student-workspaces.com/customers/id/1`. The browser's address bar and tabs are visible at the top. The main content area displays a JSON response in a dark-themed editor. The JSON object contains a numeric ID, a name, a last name, and a password hash. The password hash is a long hexadecimal string.

```
[
  {
    "id": 1,
    "name": "Paul",
    "lname": "Doe",
    "password": "d8578edf8458ce06fbc50076a58c5ca4"
  }
]
```

- Hashing used here is very weak & can be cracked by brute force
- Now use the given tool **hashid.py** to check which type of hash is this
- And try to crack the hash with given tool **checkhash.py**

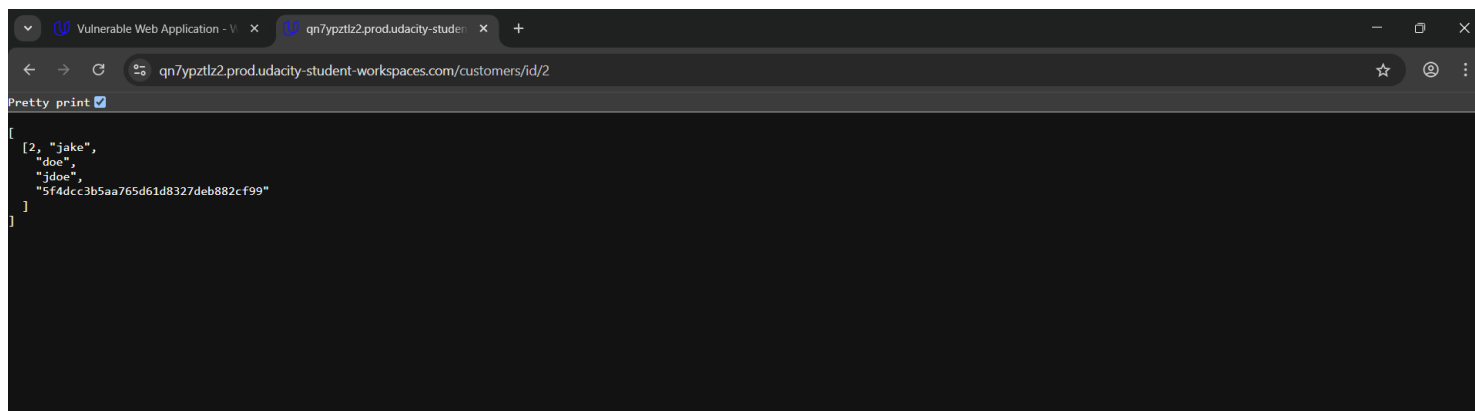


- **python hashid.py “hash here”**
- Use above command to get the type of hash
- We got the hash type is **MD5**
- Now use **python checkhash.py “hash here” -t “hash type”**
- it crack the hash and give the password of customer



```
workspace root$ cd tools
tools root$ ls
bruteforce.py  checkhash.py  hashid.py  performbase64.py  requirements.txt  test-password.txt  test-username.txt
tools root$ python hashid.py d8578edf8458ce06fbc5bb76a58c5ca4
Analyzing 'd8578edf8458ce06fbc5bb76a58c5ca4'
[+] MD5
This is a demo version of the hashid.py for this training, for the full version please visit https://github.com/psypanda/hashID
tools root$ python checkhash.py d8578edf8458ce06fbc5bb76a58c5ca4 -t MD5
-----
Hashed value passed = d8578edf8458ce06fbc5bb76a58c5ca4
Hash type passed = MD5
Result = qwerty
Actual Hash Type should be = md5
-----
tools root$
```

- Now let's crack another hash of customer id 2



```
q7ypztz2.prod.udacity-student-workspaces.com/customers/id/2
Pretty print
[
  [2, "jake",
    "doe",
    "jdoe",
    "5f4dcc3b5aa765d61d8327deb882cf99"]
]
```





- By using **hashid.py** we can see the hash type if **md5**
- By using **checkhash.py** and give type as md5 we can crack the hash
- And got the password for customer 2

```
tools root$ python hashid.py 5f4dcc3b5aa765d61d8327deb882cf99
Analyzing '5f4dcc3b5aa765d61d8327deb882cf99'
[+] MD5
This is a demo version of the hashid.py for this training, for the full version please visit https://github.com/psypanada/hashID
tools root$ python checkhash.py 5f4dcc3b5aa765d61d8327deb882cf99 -t md5
-----
Hased value passed = 5f4dcc3b5aa765d61d8327deb882cf99
Hash type passed = md5
Result = password
Actual Hash Type should be = md5
-----
tools root$
```

Usage Links ? Help



# HTML Injection

<b>Severity:</b>	High
<b>OWASP TOP 10 reference:</b>	A03:2021 – Injection
<b>Vulnerability Explanation</b>	
<p>The website is executing user input directly without cleaning them. If someone puts malicious code like <code>&lt;h1&gt;Udacity&lt;/h1&gt;</code> in a message, it will run on user's end as well as to whom message is sent. who opens that page. That means a bad guy can: Steal your login cookie Trick users into doing actions (like changing passwords or sending messages) Or even take over admin accounts</p>	
<b>Remediation recommendation</b>	
<p>Validate and filter inputs because it only allow safe characters or limited HTML if needed.</p> <p>Sanitize data on both input and output because always treat user input as unsafe until cleaned.</p> <p>Escape or sanitize all user input before displaying it because use built-in frameworks or libraries to escape HTML special characters (<code>&lt;</code>, <code>&gt;</code>, <code>"</code>) for example, show <code>&lt;b&gt;Hello&lt;/b&gt;</code> as text, not bold.</p> <p>Fix it by cleaning and escaping user input, so browser treats it as text — not code.</p> <p>Use Content Security Policy (CSP) because it set limits what scripts can run on your page, blocking many XSS attacks.</p>	



# HTML Injection

## step-by-step walkthrough

- In profile section there is a message box .
- Which is vulnerable to injection vulnerability
- We can inject html codes in it
- And it will execute them
- Here we have executed several codes
- Let's run **<h1>Udacity</h1>**
- The html code will make a header name Udacity

The screenshot shows a web application interface. On the left is a dark sidebar with navigation links: Dashboard, Customers, and Users. The main content area displays a user profile for 'Super User' (admin). The profile includes statistics: 1,322 Followers, 543 Following, and 13,287 Friends. Below this is an 'About Me' section with fields for Education (Security Engineer Nanodegree from Udacity), Location (International Space Station, Outer Space), Skills (Coding, Python, Networking, Security), and Notes (Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam fermentum enim neque.). On the right side of the profile, there is a 'Messages' tab. The message history shows three messages from 'administrator' to 'guest'. The latest message contains the HTML payload '<h1>Udacity</h1>'. Below the message history, there is a text input field containing the same payload, a dropdown menu set to 'guest', and a red 'Send' button. A small error message 'www.google.com refused to connect.' is visible above the input field.



Following543

Friends13,287

About Me

Education

Security Engineer Nanodegree from Udacity

Location

International Space Station, Outer Space

Skills

CodingPythonNetworkingSecurity

Notes

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam fermentum enim neque.

HTML

administrator

You sent a message to guest

administrator

You sent a message to guest

Udacity

Send message

guest

Send

Copyright © 2020 Test Company LLC. All rights reserved.

Ver. 1.0.0.0

- We can also execute other injection like displaying external website content
- **<iframe id="if1" src="https://www.google.com"></iframe>**
- it will make a frame with the content of external website

NetworkingSecurity

amet, consectetur adipiscing elit. Etiam fermentum enim neque.

administrator

You sent a message to guest

Udacity

administrator

You sent a message to guest

www.google.com refused to connect.

<iframe id="if1" src="https://www.google.com"></iframe>

guest

Send

company LLC. All rights reserved.

Ver. 1.0.0.0



# Cross-Site Scripting (Stored XSS)

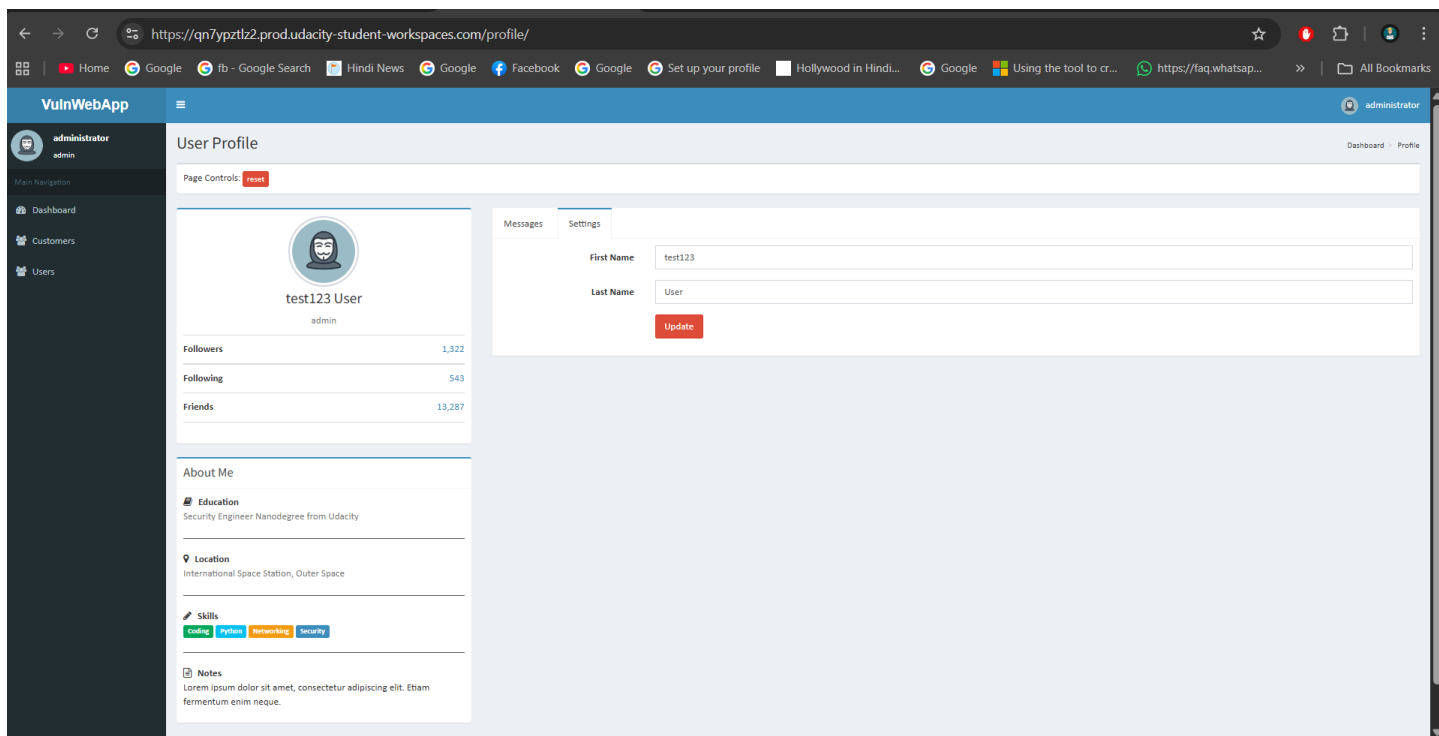
<b>Severity:</b>	High
<b>OWASP TOP 10 reference:</b>	A03:2021 – Injection (cross site scripting) (Stored)
<b>Vulnerability Explanation</b>	
<p>The username update filed is vulnerable to stored xss , if the any user update the username to any malicious code the system will update the username to that code on frontend but in backend on admin panel it execute the malicious code, If admin open the users page the code will got executed.</p>	
<b>Remediation recommendation</b>	
<p><i>Sanitize / escape output — show user input as text, not code and just like convert &lt; to &amp;lt;, &gt; to &amp;gt;.</i></p> <p><i>Validate inputs because it only allow safe characters or whitelisted HTML tags if really needed.</i></p> <p><i>Add a Content Security Policy (CSP) because it restrict what scripts can run.</i></p> <p><i>Use a safe templating engine frameworks like React, Angular, or Django auto-escape values.</i></p> <p><i>First test often try entering &lt;script&gt; in forms to confirm it's blocked.</i></p> <p><i>Clean the input and escape it before showing it, and boom no more XSS drama.</i></p>	



# Cross-Site Scripting (XSS)

step-by-step walkthrough

- We need admin access here , which we got earlier
- In Profile section there is field to update's first name and last name



- Try changing username



- If we change the username like **test123**
- it will update the username in user list page also

The screenshot shows the 'Users' page of the VulnWebApp. The page has a sidebar with navigation links: Dashboard, Customers, and Users. The main content area is titled 'Users' and contains a 'Users List' table. The table has columns for ID, First Name, Last Name, Username, and Options. The user 'test123' is listed with ID 1, First Name 'User', Last Name 'User', and Username 'administrator'. The 'Options' column has a 'View' button next to each user.

ID	First Name	Last Name	Username	Options
2	John	Doe	guest	<a href="#">View</a>
1	test123	User	administrator	<a href="#">View</a>

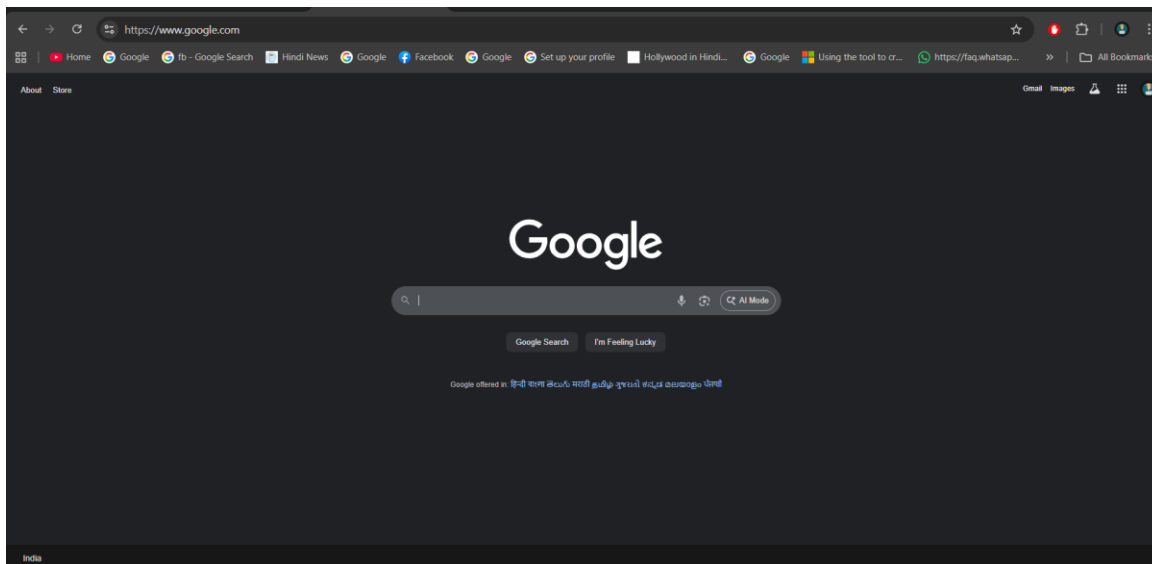
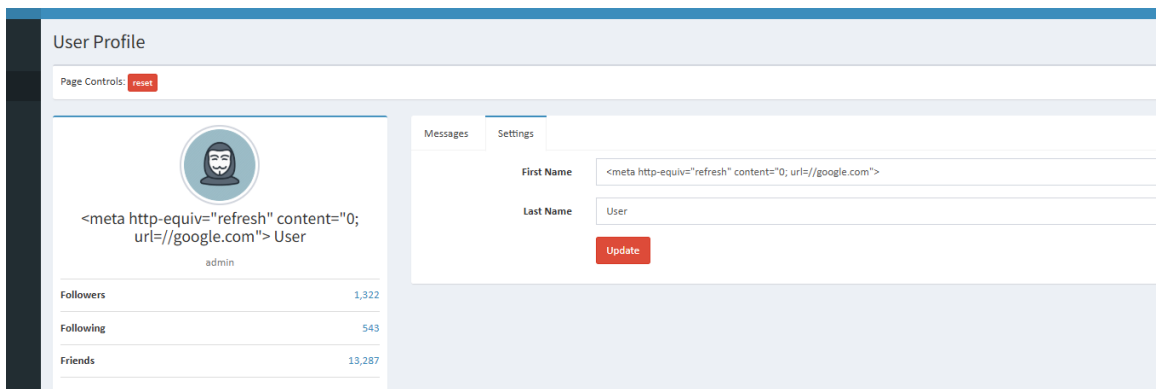
- If we injection any malicious code here it will change and display the username to that
- And upon opening user list it will execute the injected code

The screenshot shows the 'User Profile' page of the VulnWebApp. The page has a sidebar with navigation links: Dashboard, Customers, and Users. The main content area is titled 'User Profile' and contains a user profile for 'User'. The profile includes a bio, location, and skills. The 'First Name' field contains a malicious payload: `<meta http-equiv="refresh" content="0; url=//google.com">`. The 'Last Name' field contains 'User'. The 'Update' button is visible.

Field	Value
First Name	<code>&lt;meta http-equiv="refresh" content="0; url=//google.com"&gt;</code>
Last Name	User



- Here we injected the code which will redirect to external page
- **<meta http-equiv="refresh" content="0; url=//google.com">**
- Now username changed to this
- If we go to user list page it will execute the code & redirect us to google.com
- It show the stored XSS vulnerability





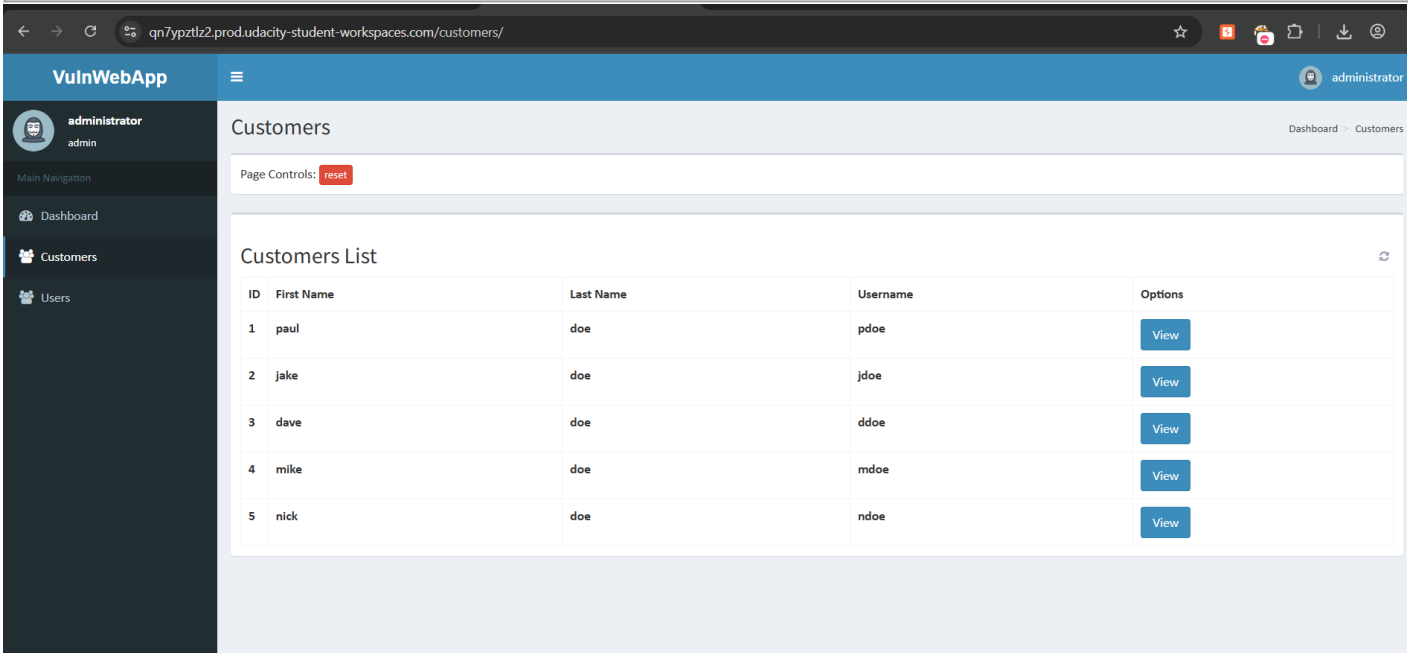


# SQLi

<b>Severity:</b>	<i>Critical</i>
<b>OWASP TOP 10 reference:</b>	<i>A03:2021 – Injection</i>
<b>Vulnerability Explanation</b>	
<i>In URL <b>/customers/id/1</b> , it is vulnerable to SQL injection . Attacker can make a payload which make all condition true to database even if the credential is wrong and it can fetch all the customers data including their personal information their password hashes</i>	
<b>Remediation recommendation</b>	
<i>To prevent from SQL injection Use parameterized queries. Validate and filter input from users avoid raw inputs. Apply the allowlist, whitelist. Use a strong and secure web application firewall. Always limit the database privileges. Don't display the errors on the user's side. Always monitor logs and deploy automated tools which can create alert when a malicious query is run.</i>	

## step-by-step walkthrough

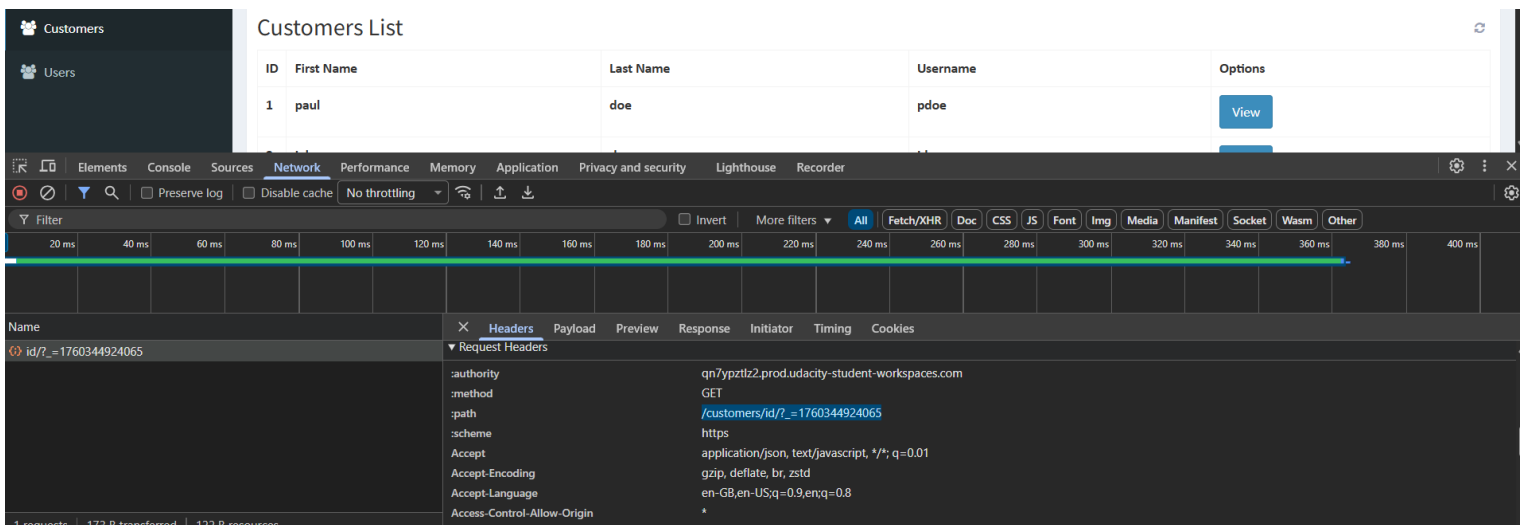
- If we go to page **/customers** , it show list of customers fetched from database



The screenshot shows the VulnWebApp interface. The browser address bar displays `qn7ypztlz2.prod.udacity-student-workspaces.com/customers/`. The application header includes the title "VulnWebApp" and a user profile for "administrator". A sidebar on the left contains navigation links for "Dashboard", "Customers", and "Users". The main content area, titled "Customers", features a "Page Controls" section with a "reset" button. Below this is a "Customers List" table with the following data:

ID	First Name	Last Name	Username	Options
1	paul	doe	pdoe	<a href="#">View</a>
2	jake	doe	jdoe	<a href="#">View</a>
3	dave	doe	ddoe	<a href="#">View</a>
4	mike	doe	mdoe	<a href="#">View</a>
5	nick	doe	ndoe	<a href="#">View</a>

- Now open the developer tool
- Then go to Network tab to see the endpoint request to database



The screenshot shows the browser's developer tools with the Network tab selected. The "Customers List" table is visible in the background. The Network tab displays a list of requests, with the first request selected. The request details are as follows:

Name	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
id/?_id=1760344924065	<ul style="list-style-type: none"><li>authority: qn7ypztlz2.prod.udacity-student-workspaces.com</li><li>method: GET</li><li>path: /customers/id/?_id=1760344924065</li><li>scheme: https</li></ul>						



- Now open the api request in new tab
- Which will show the data of customer
- If we hit with the **/id/1** it will show data of customer 1 , But

```
[[1, "paul", "doe", "pdoe", "d8578edf8458ce06fbc5bb76a58c5ca4"]]
```

- If we want to get data of all customers from database
- We have to build a **SQL injection payload**
- Payload worked here is **"1' OR '1"**
- By going to URL
- **/customers/id/1' or '1**
- It will trigger the database to a true statement and fetch all customers data

```
[
  [1, "paul",
    "doe",
    "pdoe",
    "d8578edf8458ce06fbc5bb76a58c5ca4"
  ],
  [2, "jake",
    "doe",
    "jdoe",
    "5f4dcc3b5aa765d61d8327deb882cf99"
  ],
  [3, "dave",
    "doe",
    "ddoe",
    "e807f1fcf82d132f9bb018ca6738a19f"
  ],
  [4, "mike",
    "doe",
    "mdoe",
    "8621ffdb5698829397d97767ac13db3"
  ],
  [5, "nick",
    "doe",
    "ndoe",
    "df53ca268240ca76670c8566ee54568a"
  ]
]
```