

# VulnWebApp (VWA) Security Report

Code Revision: 1.0.0.0  
Company: Acme Inc.  
Report: VWAYMMDD  
Author: [Shamima Sultana]  
Date: [1/11/21]

# VWA Security Report

VWA21-1-14-1 - A2:2017 - Broken Authentication - MEDIUM.....	3
VWA21-1-14-2 - A7:2017 - Cross-Site Scripting (XSS) - MEDIUM.....	7
VWA21-1-14-3 - A8:2017 - Insecure Deserialization - LOW.....	11
VWA21-1-14-4 - A5:2017 - Broken Access Control - MEDIUM.....	14
VWA21-1-14-5 - A5:2017 - Broken Access Control - LOW.....	18
VWA21-1-14-6 - A5:2017 - Broken Access Control - MEDIUM.....	21
VWA21-1-14-7 - A6:2017 - Security Misconfiguration - HIGH.....	24
VWA21-1-14-8 - A3:2017 - Sensitive Data Exposure - MEDIUM.....	26
VWA21-1-14-9 - A1:2017 - Injection - CRITICAL.....	33
VWA21-1-14-10 - A1:2017 - Injection - CRITICAL.....	37
VWA21-1-14-11 - A1:2017 - Injection - HIGH.....	41

# VWA Security Report

**VWA21-1-14-1 - A2:2017 - Broken Authentication - MEDIUM**

**Vulnerability Exploited:** A2-Broken Authentication

**Severity:** [Medium]

**System:** VWA Web Application

## Vulnerability Explanation:

Here brute force method is used in broken authentication. By using large number of common username and password through brute-force login tools against a web application an attacker tries to find the combination from the list.

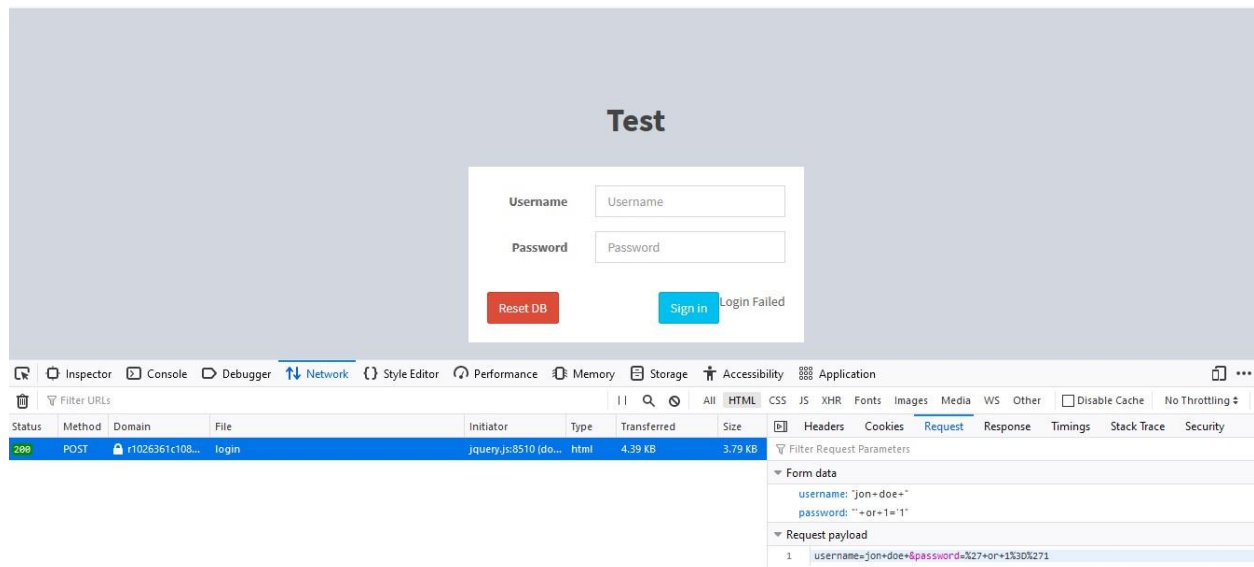
## Vulnerability Walk-thru:

A detailed walk-thru on how to reproduce this vulnerability with screenshots.

1. For login go to the url:

<https://r1026361c1084570xjupyterl1tj21rh6-3000.udacity-student-workspaces.com/login>

2. Credentials for login is unknown and SQL injection is failed



3. From tools option in workspace using bruteforce.py to get match credentials for login.

# VWA Security Report

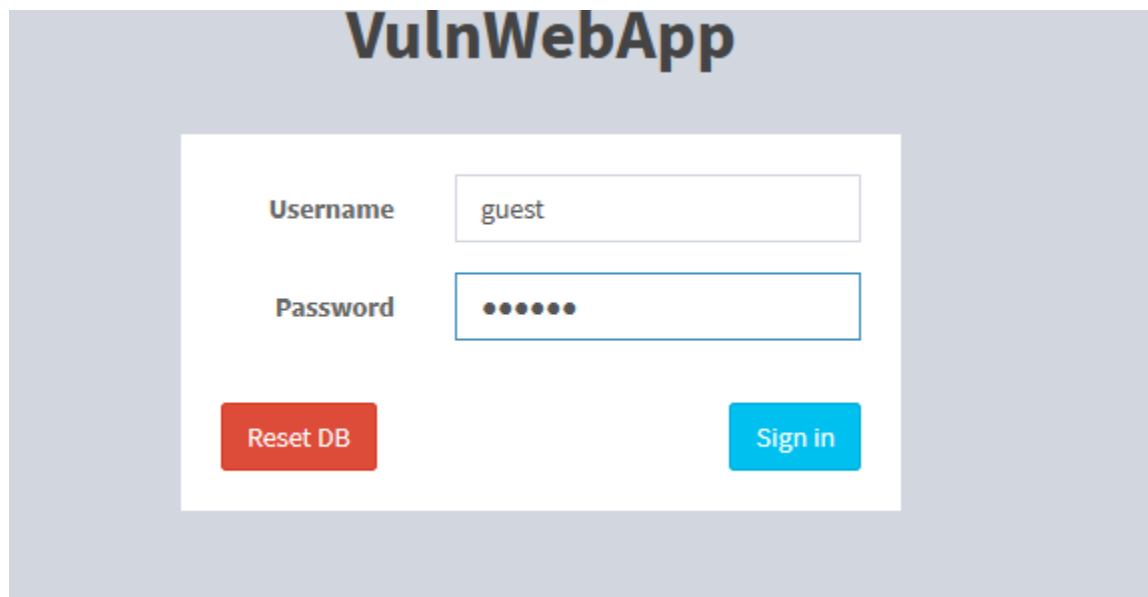
4. `python bruteforce.py -U top-usernames-shortlist.txt -P top-passwords-shortlist.txt -d username=^USR^:password=^PWD^ -m 'post' -f 'Login Failed'`

<https://r1026361c1084570xjupyterl1tj21rh6-3000.udacity-student-workspaces.com/login>



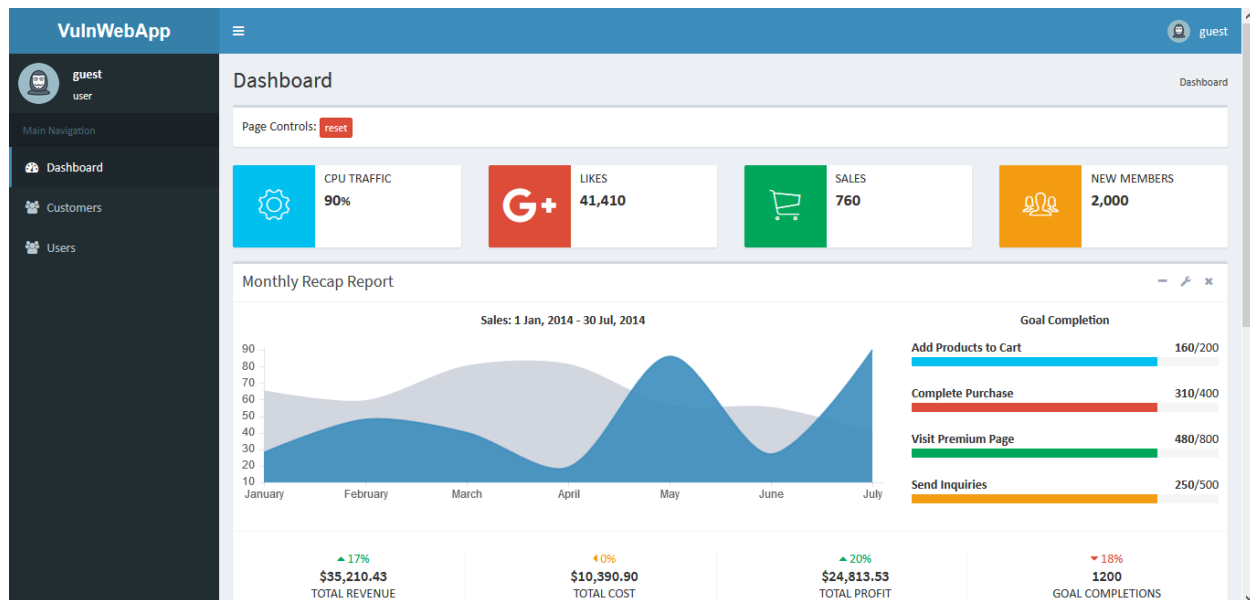
```
[-] Login Failed! ['username': 'guest', 'password': 'pepper']
[-] Login Failed! ['username': 'guest', 'password': 'l2ll']
[-] Login Failed! ['username': 'guest', 'password': 'astin']
[-] Login Failed! ['username': 'guest', 'password': 'william']
[-] Login Failed! ['username': 'guest', 'password': 'daniel']
[-] Login Failed! ['username': 'guest', 'password': 'wilfer']
[-] Login Failed! ['username': 'guest', 'password': 'summer']
[-] Login Failed! ['username': 'guest', 'password': 'heather']
[-] Login Failed! ['username': 'guest', 'password': 'summer']
[-] Login Failed! ['username': 'guest', 'password': 'joshua']
[-] Login Failed! ['username': 'guest', 'password': 'apple']
[-] Login Failed! ['username': 'guest', 'password': 'anna']
[-] Login Failed! ['username': 'guest', 'password': 'ashley']
[-] Login Failed! ['username': 'guest', 'password': 'thunder']
[-] Login Failed! ['username': 'guest', 'password': 'cody']
[-] Login Failed! ['username': 'guest', 'password': 'silver']
[-] Login Found! ['username': 'guest', 'password': 'orange']
This is a demo code used for this training.
root@591b91c509c2:/home/workspace/tools#
```

5. Using bruteforce.py tool got the matched credentials for login where user: guest password: orange



6. Got the access of test page.

# VWA Security Report



## Recommendations:

- ✓ [https://cheatsheetseries.owasp.org/cheatsheets/Credential Stuffing Prevention Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Credential_Stuffing_Prevention_Cheat_Sheet.html)
- ✓ <https://github.com/danielmiessler/SecLists/tree/master/Passwords>

## Best Practice:

- **Multi-Factor Authentication (MFA)** – MFA prevents automated attacks like credential stuffing, brute force, and stolen credential re-use attacks. So, using MFA would be the best methods for protecting accounts, even if a user credentials are exposed to the internet either by data breaches or stolen using some type of exploit the user account is still safe because the attacker will not be able to login without the MFA.
- **Captcha** – Using a captcha is a good way to slow down or even stop attackers from using some sort of automation on your web application.

# VWA Security Report

- **Weak-password check** - Attempting weak-password check by using top 1000 worst passwords against new or changed password.
- **Failed Login Attempts** - By using failed login attempts you can auto lock accounts or even block IP address that appears to be using some sort of brute forcing against a user account.

# VWA Security Report

## VWA21-1-14-2 - A7:2017 - Cross-Site Scripting (XSS) - MEDIUM

Vulnerability Exploited: A7-Cross-Site Scripting (XSS)

Severity: [ Medium]

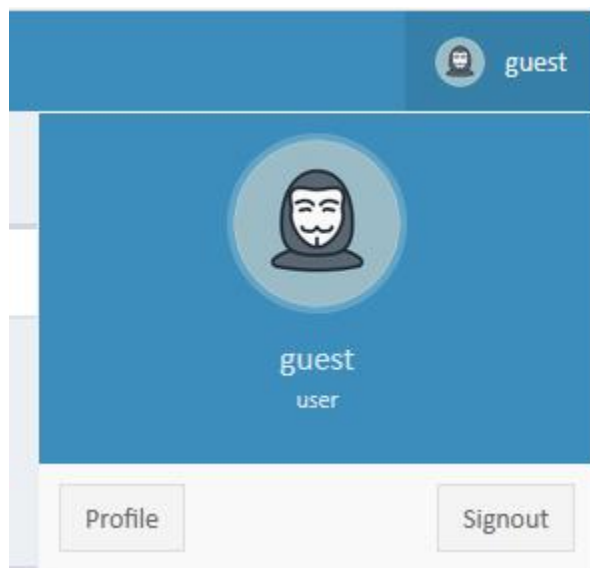
**System:** VWA Web Application

### Vulnerability Explanation:

Cross-Site Scripting (XSS) is used to run unintended code on a victim's local machine it permits attackers to perform scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites

### Vulnerability Walk-thru:

1. Go to the profile page

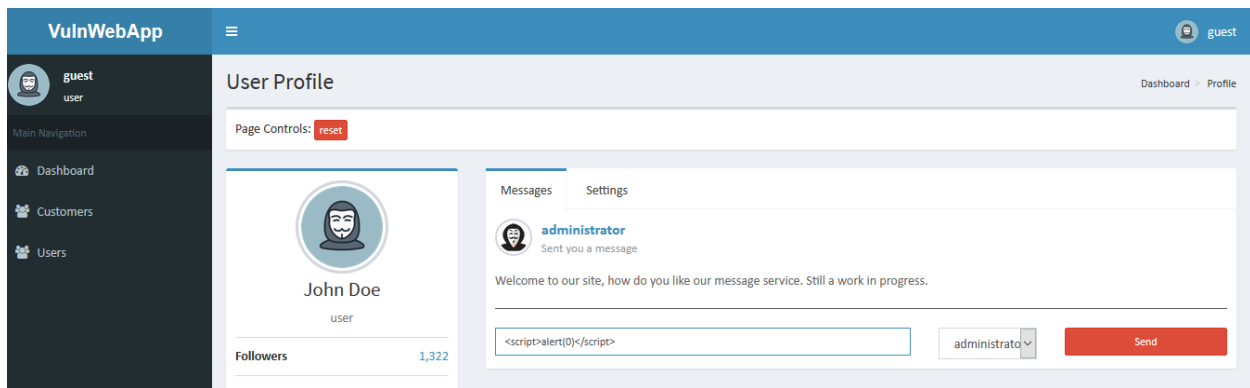


# VWA Security Report

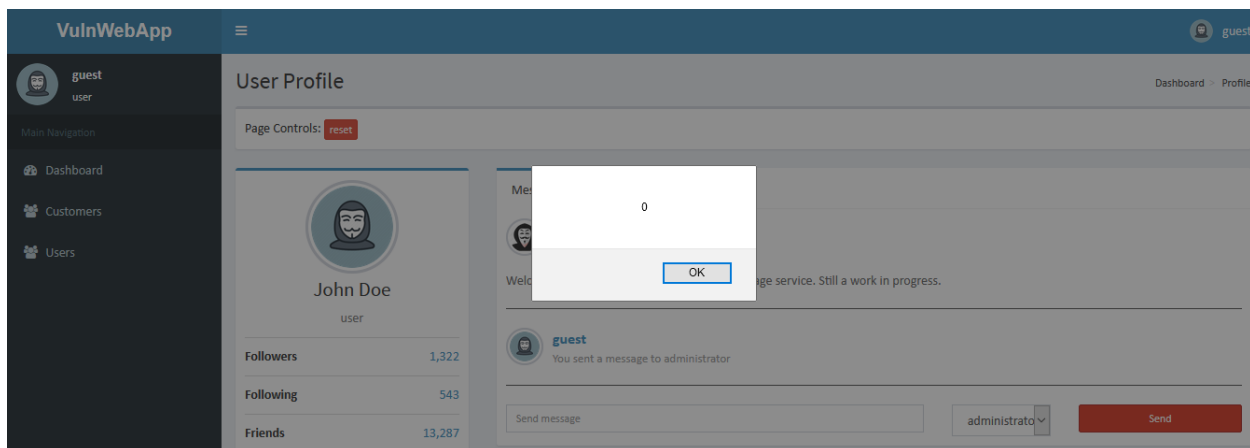
## Stored XSS

In this type of attack, a payload is sent to the web application where it is saved in the DB and then when called from the database it is executed on the target local machine

2. Injecting `<script>alert(0)</script>` in send message option



3. Showing the executed alert in profile page



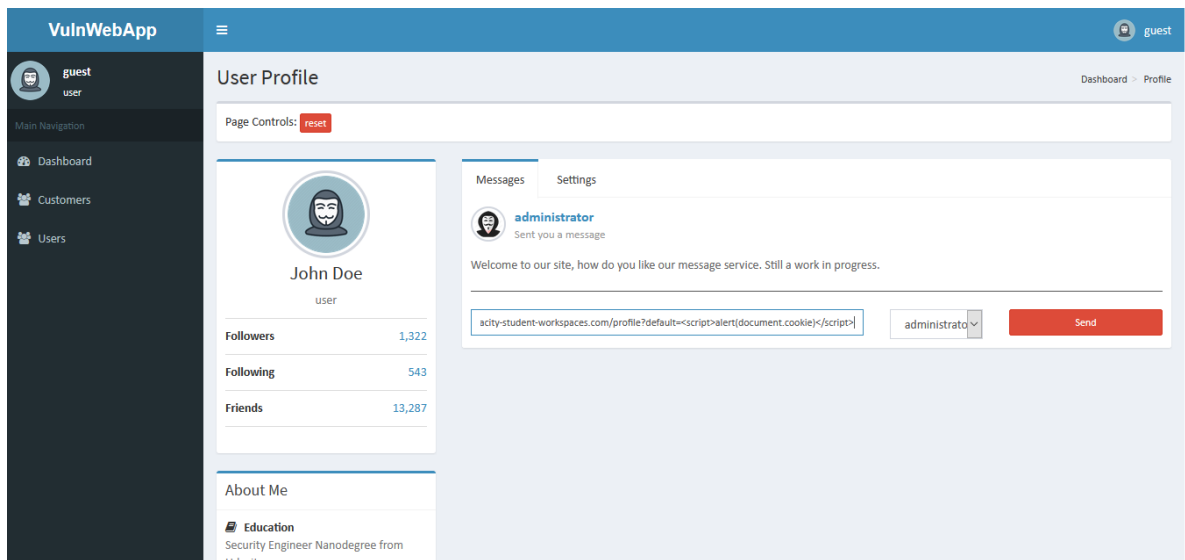


# VWA Security Report

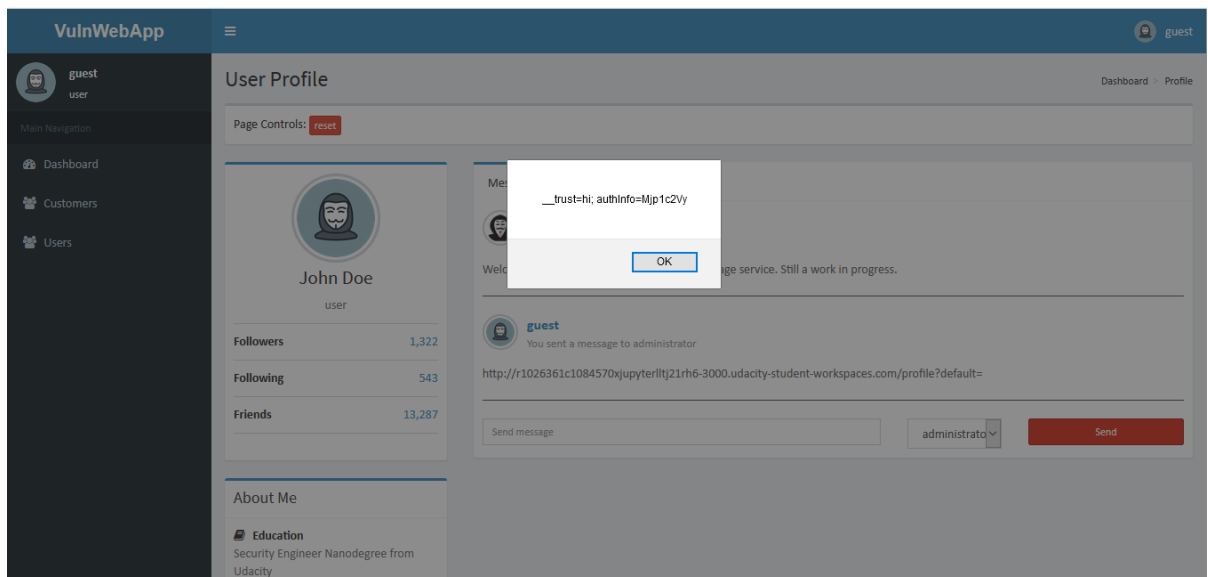
## DOM XSS:

This type of attack is considered a more advanced type of XSS and is when the vulnerability appears up in the Document Object Model (DOM) rather than in the html page.

4. Injecting [http://r1026361c1084570xjupyterl1tj21rh6-3000.udacity-student-workspaces.com/profile?default=<script>alert\(document.cookie\)</script>](http://r1026361c1084570xjupyterl1tj21rh6-3000.udacity-student-workspaces.com/profile?default=<script>alert(document.cookie)</script>)



5. Vulnerability appearing up in document object model.



# VWA Security Report

## Recommendations:

- ✓ [https://cheatsheetseries.owasp.org/cheatsheets/Cross Site Scripting Prevention Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross%20Site%20Scripting%20Prevention%20Cheat%20Sheet.html)
- ✓ [https://cheatsheetseries.owasp.org/cheatsheets/DOM based XSS Prevention Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/DOM%20based%20XSS%20Prevention%20Cheat%20Sheet.html)

## Best Practice:

- **Using Frameworks** - It automatically escape XSS by design, such as the latest Ruby on Rails, React JS. Learn the limitations of each framework's XSS protection and appropriately handle the use cases which are not covered
- **Sanitizing all inputs** - All input that is entered by the user should be precisely validated, because the user's input may find its way to the output.
- **Filtering inputs** - The idea of the filtering is to search for risky keywords in the user's input and remove them or replace them by empty strings.
- **Characters escaping** - You can use appropriate characters to change them by special codes. There exist appropriate libraries to escape the characters.

# VWA Security Report

**VWA21-1-14-3 - A8:2017 - Insecure  
Deserialization - LOW**

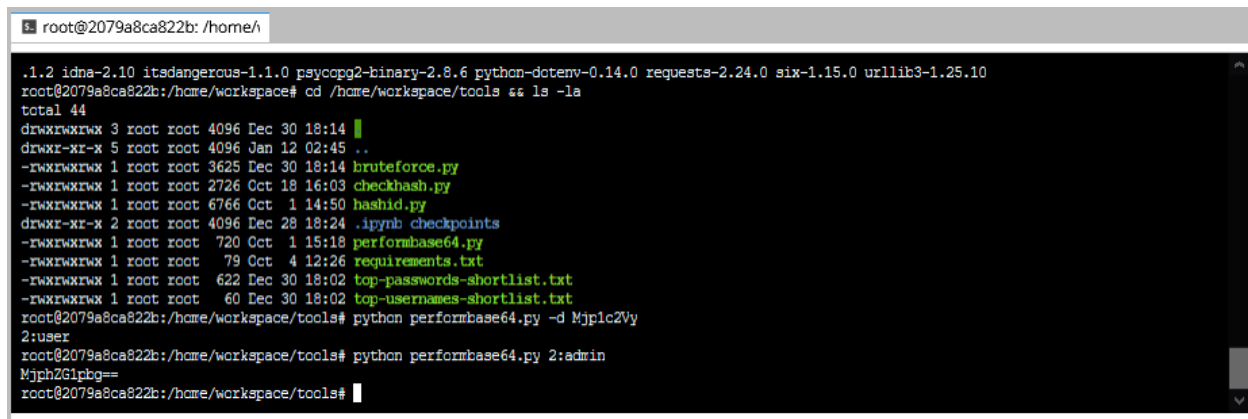
**Vulnerability Exploited:** A8 - Insecure Deserialization  
**Severity:** [Low]

**System:** VWA Web Application

## Vulnerability Explanation:

Insecure deserialization often leads to remote code execution by trusting a data source without validating. An attacker can modify their access level and view area of the site they were not intended to see.

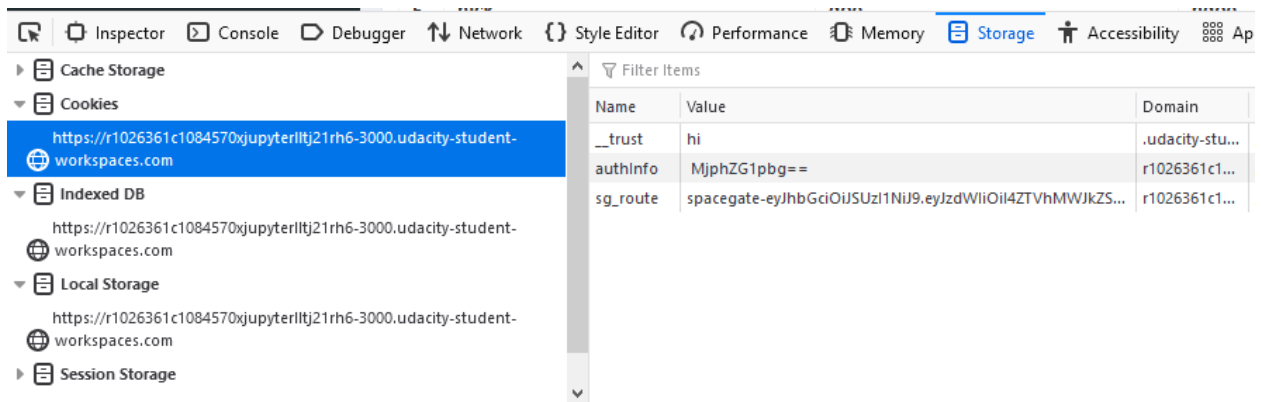
## Vulnerability Walk-thru:



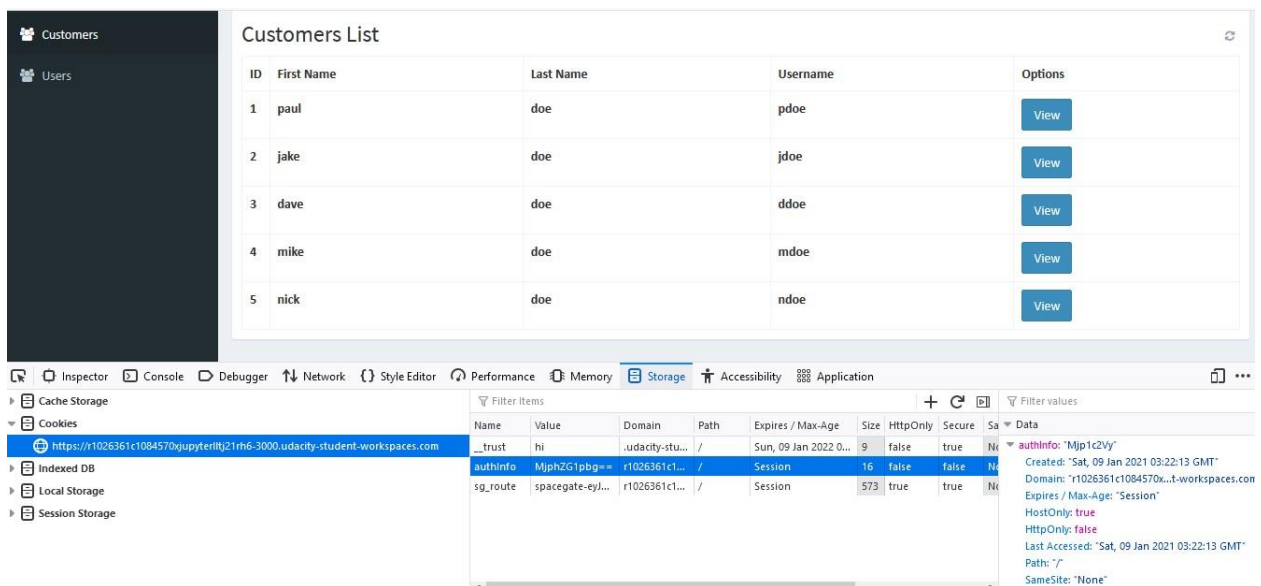
```
root@2079a8ca822b: /home/
.1.2 idna-2.10 itsdangerous-1.1.0 psycpg2-binary-2.8.6 python-dotenv-0.14.0 requests-2.24.0 six-1.15.0 urllib3-1.25.10
root@2079a8ca822b:/home/workspace# cd /home/workspace/tools && ls -la
total 44
drwxrwxrwx 3 root root 4096 Dec 30 18:14 .
drwxr-xr-x 5 root root 4096 Jan 12 02:45 ..
-rwxrwxrwx 1 root root 3625 Dec 30 18:14 bruteforce.py
-rwxrwxrwx 1 root root 2726 Oct 18 16:03 checkhash.py
-rwxrwxrwx 1 root root 6766 Oct 1 14:50 hashid.py
drwxr-xr-x 2 root root 4096 Dec 28 18:24 .ipynb_checkpoints
-rwxrwxrwx 1 root root 720 Oct 1 15:18 performbase64.py
-rwxrwxrwx 1 root root 79 Oct 4 12:26 requirements.txt
-rwxrwxrwx 1 root root 622 Dec 30 18:02 top-passwords-shortlist.txt
-rwxrwxrwx 1 root root 60 Dec 30 18:02 top-usernames-shortlist.txt
root@2079a8ca822b:/home/workspace/tools# python performbase64.py -d Mjplc2Vy
2:user
root@2079a8ca822b:/home/workspace/tools# python performbase64.py 2:admin
MjphZG1pbG==
root@2079a8ca822b:/home/workspace/tools#
```

4. Placed the encoded 2:admin value in string MjphZG1pbG== in authinfo value.

# VWA Security Report



5. Got the access as admin to view the customers and users page.



# VWA Security Report

The screenshot shows a web application interface with a 'Test' header and a 'Users' page. The 'Users' page has a 'Page Controls' section with a 'reset' button and a 'Users List' table. The 'Users List' table has columns for ID, First Name, Last Name, Username, and Options. The table contains two rows: one for 'Super' (User: administrator) and one for 'John' (User: guest). The 'Options' column has 'View' buttons for each user.

Below the 'Users' page, the 'Storage' tab is active, showing a table of cookies and session storage data. The table has columns for Name, Value, Domain, Path, Expires / Max-Age, Size, HttpOnly, Secure, and SameSite. The 'authinfo' cookie is highlighted, showing its details in the 'Data' column.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite
__trust	hi	.udacity-stu...	/	Mon, 10 Jan 2022 ...	9	false	true	None
authinfo	MjphZG1pbGw=	r1026361c1...	/	Session	20	false	false	None
sg_route	spacegate-eyl...	r1026361c1...	/	Session	568	true	true	None

The 'Data' column for the 'authinfo' cookie shows the following details:

- Created: "Sun, 10 Jan 2021 03:58:33 GMT"
- Domain: "r1026361c1084570x...t-workspaces.com"
- Expires / Max-Age: "Session"
- HostOnly: true
- HttpOnly: false
- Last Accessed: "Sun, 10 Jan 2021 03:59:07 GMT"
- Path: "/"
- SameSite: "None"

## Recommendations:

- ✓ <https://cheatsheetseries.owasp.org/cheatsheets/Deserialization Cheat Sheet.html>
- ✓ <https://owasp.org/www-project-proactive-controls/v3/en/c5-validate-inputs>

## Best Practice

- **Use Serialization Mediums** - The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types
- **Integrity Checks** - By using a hashing function you can create a digital signature that then can be used later on to verify that the data has not been altered
- **Strict Data Type** - By using strict data types, this can help protect your data from expected data types.

# VWA Security Report

**VWA21-1-14-4 - A5:2017 - Broken Access Control - MEDIUM**

**Vulnerability Exploited:** A5 - Broken Access Control (customer)

**Severity:** [Medium]

**System:** VWA Web Application

## Vulnerability Explanation:

Broken access controls often fail to enforce restrictions on authenticated users. An attacker takes this advantage to get unauthorized access to restricted areas of the application such as accessing other users' accounts, viewing sensitive files, modifying other users' data, changing access rights, etc.

## Vulnerability Walk-thru:

1. I explained in VWAYMMDD-3 how I got the access as admin to view customers list.
2. There are 5 customers in customers list with respected id and value

The screenshot displays a web application interface with a sidebar containing 'Customers' and 'Users' links. The main content area shows 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>

Below the table, a network inspector shows the details of a GET request to the endpoint `/customers/id?_id=1610341421508`. The response is a JSON array containing the details of the customer with ID 1 (paul doe).

```
[{"id": "1", "first_name": "paul", "last_name": "doe", "username": "pdoe", "password": "d8578edf8458ce06fbc5bb76a58c5ca4"}]
```

# VWA Security Report

3. Information of customer where id=1



4. Information of customer where id=2 is showing by changing the id number in url instead of viewing it from customer page.



5. Information of customer where id=3 is showing by changing the id number in url instead of viewing it from customer page.

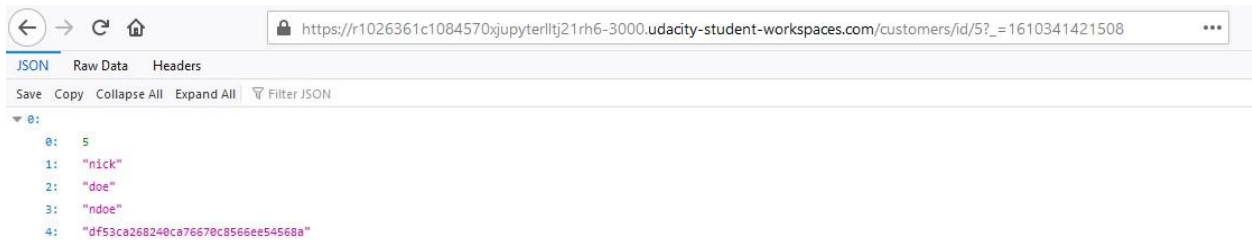


6. Information of customer where id=4 is showing by changing the id number in url instead of viewing it from customer page.

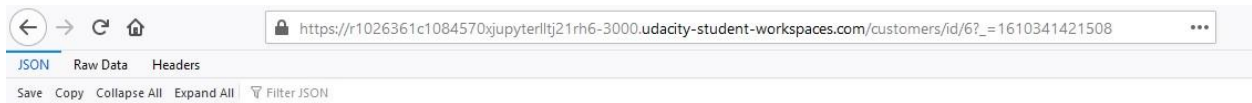
# VWA Security Report



7. Information of customer where id=5 is showing by changing the id number in url instead of viewing it from customer page.



8. As there are 5 customers so there exist no information for id=6





# VWA Security Report

## Recommendations:

- ✓ [https://cheatsheetseries.owasp.org/cheatsheets/Access Control Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html)
- ✓ <https://owasp.org/www-project-application-security-verification-standard/>

## Best Practice:

- **Rate Limit Data** - by rate limiting the user access to data on the site, you can slow down their ability to scrape all data from your web application.
- **Re-validate on all secure pages** - make sure you have proper testing around all secure pages and endpoints to validate access control are working as expected
- **Deny Access for non-public pages by default** - you should have general rules that auto deny access to non-public pages and require validation to access these pages
- **Log Access failures** - you should not only log all access failures, but also create automation process that alert you of IP/Users that have high level of failures.

# VWA Security Report

**VWA21-1-14-5 - A5:2017 - Broken Access Control - LOW**

**Vulnerability Exploited: A5 - Broken Access Control (user)**

**Severity: [ Low]**

**System:** VWA Web Application

## Vulnerability Explanation:

Broken access controls often fail to enforce restrictions on authenticated users. An attacker takes this advantage to get unauthorized access to restricted areas of the application such as accessing other users' accounts, viewing sensitive files, modifying other users' data, changing access rights, etc.

## Vulnerability Walk-thru:

1. I explained in VWAYMMDD-3 how I got the access as admin to view customers list.
2. There are 2 users in the users list with respected id and value.

The screenshot shows the VWA application interface. On the left is a dark sidebar with a user profile (guest admin) and navigation links for Dashboard, Customers, and Users. The main content area is titled 'Users' and contains a 'Page Controls' section with a 'reset' button. Below this is a 'Users List' table with the following data:

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

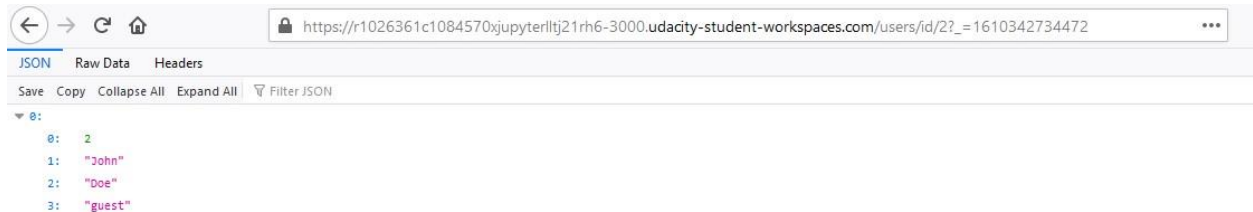
Below the table, the browser's developer tools are open to the Network tab. It shows a list of requests. The selected request is a GET request to the URL `/users/1?_id=1610342734472` with a status of 200. The response is a JSON object: `{ "0": [ "Super", "User", "administrator" ] }`.

# VWA Security Report

3.Information of user where id=1



4. Information of user where id=2 is showing by changing the id in url instead of viewing it from user page.



5. As there are 2 users in userlist so no information is available for id=3.



# VWA Security Report

## Recommendations:

- ✓ [https://cheatsheetseries.owasp.org/cheatsheets/Access Contr ol Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Access_Contr ol Cheat Sheet.html)
- ✓ <https://owasp.org/www-project-application-security-verification-standard/>

## Best Practice:

- **Rate Limit Data** - by rate limiting the user access to data on the site, you can slow down their ability to scrape all data from your web application.
- **Re-validate on all secure pages** - make sure you have proper testing around all secure pages and endpoints to validate access control are working as expected
- **Deny Access for non-public pages by default** - you should have general rules that auto deny access to non-public pages and require validation to access these pages
- **Log Access failures** - you should not only log all access failures, but also create automation process that alert you of IP/Users that have high level of failures.

# VWA Security Report

## VWA21-1-14-6 - A5:2017 - Broken Access Control - MEDIUM

Vulnerability Exploited: A5 - Broken Access Control (profile)

Severity: [Medium]

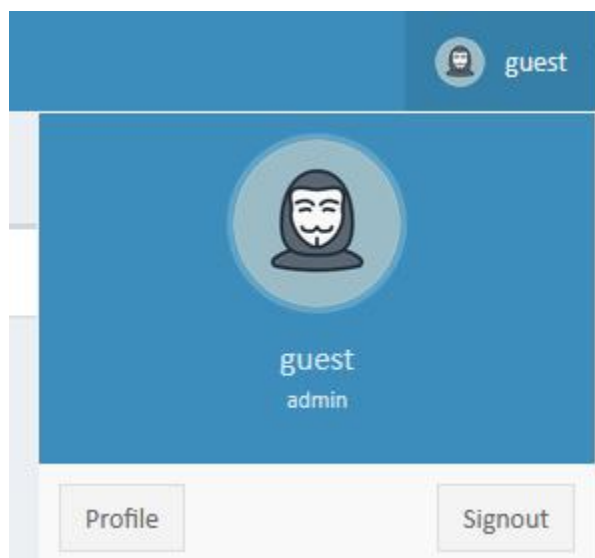
**System:** VWA Web Application

### Vulnerability Explanation:

Broken access controls often fail to enforce restrictions on authenticated users. An attacker takes this advantage to get unauthorized access to restricted areas of the application such as accessing other users' accounts, viewing sensitive files, modifying other users' data, changing access rights, etc.

### Vulnerability Walk-thru:

1. Go to profile as admin



# VWA Security Report

2. Open the inspect go to network tab. There are 2 userlists in profile one is administration another is guest.

The screenshot shows a web application interface with a sidebar on the left containing 'Main Navigation' with links to 'Dashboard', 'Customers', and 'Users'. The main content area is titled 'User Profile' and shows a profile for 'John Doe' (admin) with 1,322 followers and 543 following. Below the profile, there are tabs for 'Messages' and 'Settings'. The 'Messages' tab is active, showing a message from 'administrator' with the text 'Welcome to our site, how do you like our message service. Still a work in progress.' and a 'Send message' button. The bottom of the screen shows a network inspector with the 'Network' tab selected. It displays a list of requests, with the last one highlighted: a GET request to 'https://r1026361c1084570xjupyterl1tj21rh6-3000.udacity-student-workspaces.com/profile/userlist/2?\_=1610342916827' with a status of 200. The response is a JSON array containing one object: {id: 1, name: 'administrator'}. The 'Response' tab is selected, showing the JSON data.

3. Open the userlist in new tab where id =2 is showing the information of administration

The screenshot shows a web browser window with the address bar displaying the URL: 'https://r1026361c1084570xjupyterl1tj21rh6-3000.udacity-student-workspaces.com/profile/userlist/2?\_=1610342916827'. The page content is a JSON array containing one object: {id: 2, name: 'administrator'}. The 'JSON' tab is selected, showing the JSON data.

4. Information of userlist where id=1 is showing the information by changing the id number in url instead of viewing it from profile page.

The screenshot shows a web browser window with the address bar displaying the URL: 'https://r1026361c1084570xjupyterl1tj21rh6-3000.udacity-student-workspaces.com/profile/userlist/1?\_=1610342916827'. The page content is a JSON array containing one object: {id: 1, name: 'guest'}. The 'JSON' tab is selected, showing the JSON data.

# VWA Security Report

5. As there are only two userlist where id=3 is showing both guest and administration. Whatever we use except id=1 and id=2 it shows both guest and administration information.



## Recommendations:

- ✓ [https://cheatsheetseries.owasp.org/cheatsheets/Access Contr ol Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Access_Contr ol Cheat Sheet.html)
- ✓ <https://owasp.org/www-project-application-security-verification-standard/>

## Best Practice:

- **Rate Limit Data** - by rate limiting the user access to data on the site, you can slow down their ability to scrape all data from your web application.
- **Re-validate on all secure pages** - make sure you have proper testing around all secure pages and endpoints to validate access control are working as expected
- **Deny Access for non-public pages by default** - you should have general rules that auto deny access to non-public pages and require validation to access these pages
- **Log Access failures** - you should not only log all access failures, but also create automation process that alert you of IP/Users that have high level of failures.

# VWA Security Report

**VWA21-1-14-7 - A6:2017 - Security**

**Misconfiguration - HIGH**

**Vulnerability Exploited: A6 - Security Misconfiguration**

**Severity: [High]**

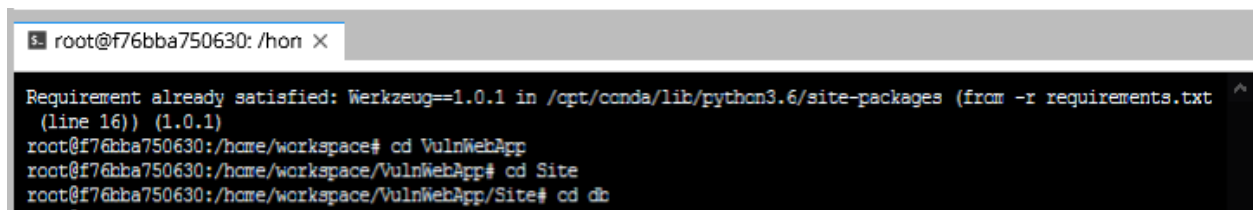
**System:** VWA Web Application

## **Vulnerability Explanation:**

Security Misconfiguration is basically distinct as deteriorating to implement all the security controls for a server or web application. It is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information.

## **Vulnerability Walk-thru:**

By scanning VulnWebApp/Site/db/more \_\_init\_\_.py file one security misconfiguration found regarding password issue.

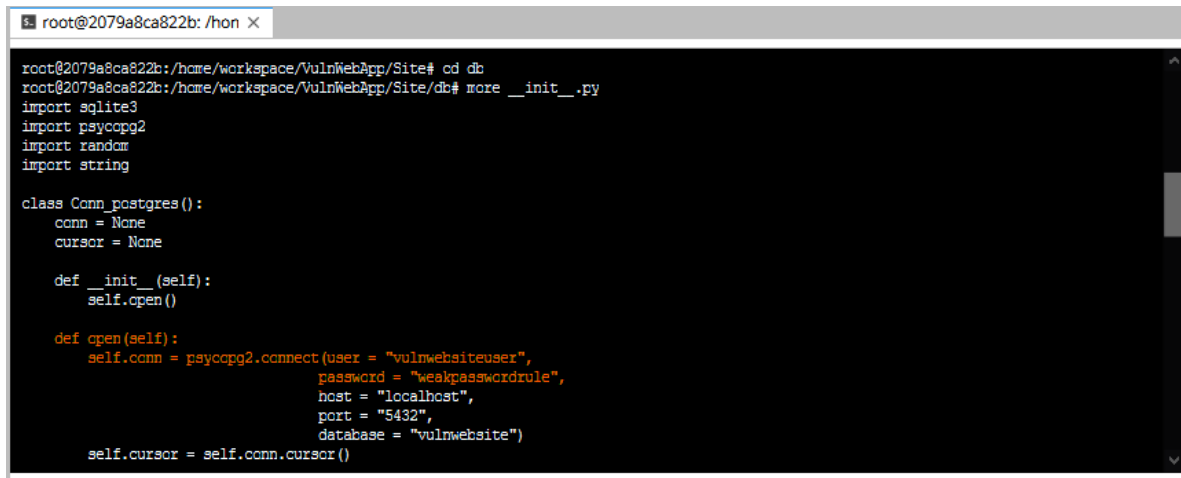


```
root@f76bba750630: /home/x
Requirement already satisfied: Werkzeug==1.0.1 in /opt/conda/lib/python3.6/site-packages (from -r requirements.txt
(line 16)) (1.0.1)
root@f76bba750630:/home/workspace# cd VulnWebApp
root@f76bba750630:/home/workspace/VulnWebApp# cd Site
root@f76bba750630:/home/workspace/VulnWebApp/Site# cd db
```



# VWA Security Report

Issue: Possible hardcoded password: 'weakpasswordrule'

A terminal window with a dark background and light-colored text. The title bar at the top reads 'root@2079a8ca822b: /hon x'. The terminal shows a user navigating to a directory and viewing a Python file. The code defines a PostgreSQL connection class with a hardcoded password.

```
root@2079a8ca822b:/home/workspace/VulnWebApp/Site# cd db
root@2079a8ca822b:/home/workspace/VulnWebApp/Site/db# more __init__.py
import sqlite3
import psycopg2
import random
import string

class Conn_postgres():
    conn = None
    cursor = None

    def __init__(self):
        self.open()

    def open(self):
        self.conn = psycopg2.connect(user = "vulnwebsiteuser",
                                     password = "weakpasswordrule",
                                     host = "localhost",
                                     port = "5432",
                                     database = "vulnwebsite")

        self.cursor = self.conn.cursor()
```

## Recommendations:

- ✓ [https://owasp.org/www-community/vulnerabilities/Use of hard-coded password](https://owasp.org/www-community/vulnerabilities/Use%20of%20hard-coded%20password)

## Best Practice:

- **All environment should be the same** - By keeping all of your environments the same you will be able to find any issues before they make it out to production.
- **Limit components** - You should push to limit the different components used to only what is really needed, then this will reduce your risk and amount of work to make sure everything is updated and working correctly.
- **Automate Environment creation and validation** - Using automation to create all environments will help reduce user errors and make sure all environments are built exactly the same, next you should automate the validation of the environment to help you understand the health and state of your environments.

# VWA Security Report

## VWA21-1-14-8 - A3:2017 - Sensitive Data Exposure - MEDIUM

**Vulnerability Exploited:** A3-Sensitive Data Exposure  
**Severity:** [Medium]

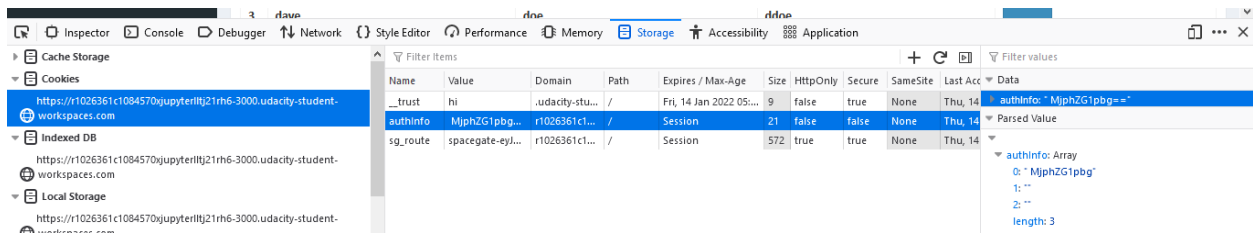
**System:** VWA Web Application

### Vulnerability Explanation:

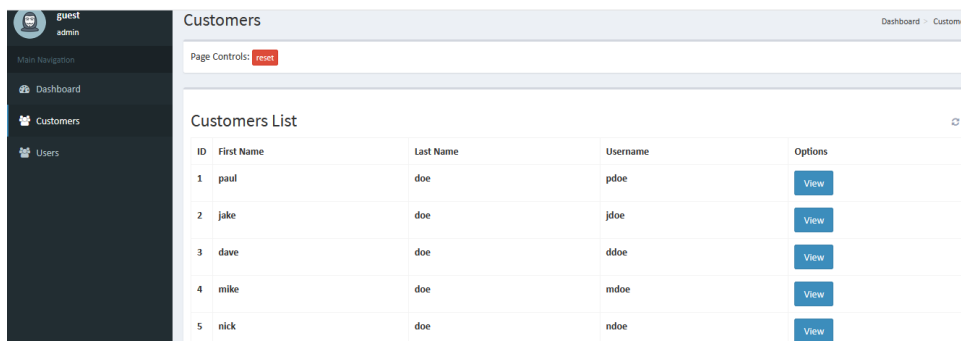
Sensitive data exposure arises when an application, company, or other entity unintentionally exposes personal data. Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. An attacker targets those data to steal or modify to conduct mischief like credit card fraud, identity theft, or other crimes.

### Vulnerability Walk-thru:

1. I explained it in VWA21-1-14-3 how I got the string value of 2:admin to get access as admin.



2. After changing the authinfo value refresh the customer page.



# VWA Security Report

3. From customer page by viewing the information in developer tools network tab we can see the information of id=1

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>

Network Tab: GET /customers/id/?\_id=1610292649783 (jQuery.js:9837 (xhr) json 291 B 61 B)

JSON Response:

```
{ 0: [ 1, "paul", "doe", "pdoe", "d8578edf8458ce06fbc5bb76a58c5ca4" ]}
```

4. From customer page by viewing the information in developer tools network tab we can see the information of id=2

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>

Network Tab: GET /customers/id/?\_id=1610292649784 (jQuery.js:9837 (xhr) json 291 B 61 B)

JSON Response:

```
{ 0: [ 2, "jake", "doe", "jdoe", "5f4dcc3b5aa765d61d8327deb882cf99" ]}
```

# VWA Security Report

5. From customer page by viewing the information in developer tools network tab and opening in it in new tab we can see the information of id=3



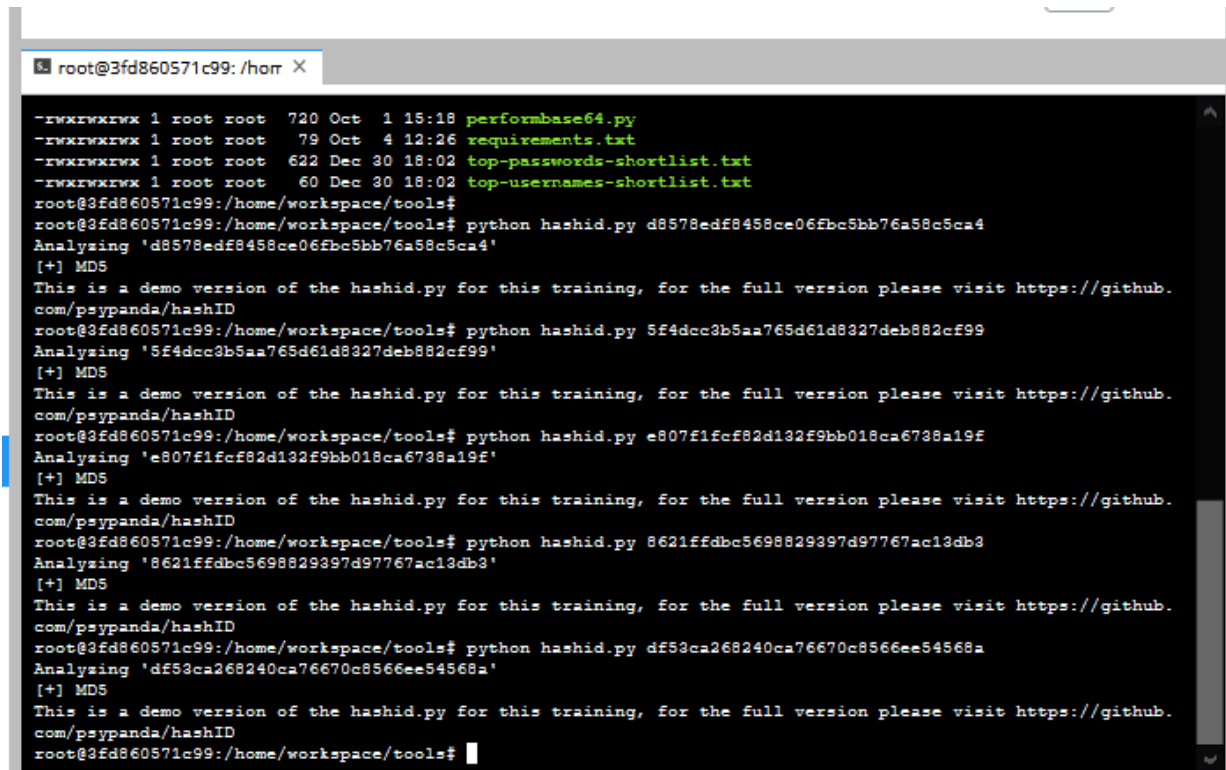
6. From customer page by viewing the information in developer tools network tab and opening in it in new tab we can see the information of id=4



# VWA Security Report

## 7. Cheking the hash type using hashid.py in workspace

```
python hashid.py d8578edf8458ce06fbc5bb76a58c5ca4
python hashid.py 5f4dcc3b5aa765d61d8327deb882cf99
python hashid.py e807f1fcf82d132f9bb018ca6738a19f
python hashid.py 8621ffdbc5698829397d97767ac13db3
python hashid.py df53ca268240ca76670c8566ee54568a
```



```
root@3fd860571c99: /horr X
-rwxrwxrwx 1 root root 720 Oct 1 15:18 performbase64.py
-rwxrwxrwx 1 root root 79 Oct 4 12:26 requirements.txt
-rwxrwxrwx 1 root root 622 Dec 30 18:02 top-passwords-shortlist.txt
-rwxrwxrwx 1 root root 60 Dec 30 18:02 top-usernames-shortlist.txt
root@3fd860571c99:/home/workspace/tools:#
root@3fd860571c99:/home/workspace/tools:# 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/psyypanda/hashID
root@3fd860571c99:/home/workspace/tools:# 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/psyypanda/hashID
root@3fd860571c99:/home/workspace/tools:# python hashid.py e807f1fcf82d132f9bb018ca6738a19f
Analyzing 'e807f1fcf82d132f9bb018ca6738a19f'
[+] MD5
This is a demo version of the hashid.py for this training, for the full version please visit https://github.com/psyypanda/hashID
root@3fd860571c99:/home/workspace/tools:# python hashid.py 8621ffdbc5698829397d97767ac13db3
Analyzing '8621ffdbc5698829397d97767ac13db3'
[+] MD5
This is a demo version of the hashid.py for this training, for the full version please visit https://github.com/psyypanda/hashID
root@3fd860571c99:/home/workspace/tools:# python hashid.py df53ca268240ca76670c8566ee54568a
Analyzing 'df53ca268240ca76670c8566ee54568a'
[+] MD5
This is a demo version of the hashid.py for this training, for the full version please visit https://github.com/psyypanda/hashID
root@3fd860571c99:/home/workspace/tools:#
```

# VWA Security Report

8. Using hash cracking from lesson exercise got the passwords.

1. If we give a close look, we can see this is the first 5 letter from of first row. This type of password is easy to guess need to avoid using this.

### Hash Cracking

**Hash**

**Type**

---

Algorithm: MD5  
Plaintext: qwerty

This data is provided by hashes.org

2. Commonly used password to attack.

### Hash Cracking

**Hash**

**Type**

---

Algorithm: MD5  
Plaintext: password

This data is provided by hashes.org

# VWA Security Report

3. This password is first 10 digit from number row. Can be cracked by easy guess or trial and error.

Hash Cracking

Hash

e807f1fcf82d132f9bb018ca6738a19f

Type

MD5

Process

Algorithm: MD5

Plaintext: 1234567890

This data is provided by hashes.org

4. Brute force may crack this password using common password.

Hash Cracking

Hash

8621ffdbc5698829397d97767ac13db3

Type

MD5

Process

Algorithm: MD5

Plaintext: dragon

This data is provided by hashes.org

Hash Cracking

Hash

df53ca268240ca76670c8566ee54568a

Type

MD5

Process

Algorithm: MD5

Plaintext: computer

This data is provided by hashes.org

# VWA Security Report

## Recommendations:

- ✓ <https://cheatsheetseries.owasp.org/cheatsheets/Password Storage Cheat Sheet.html>
- ✓ <https://cheatsheetseries.owasp.org/cheatsheets/User Privacy Protection Cheat Sheet.html>

## Best Practice:

- **Classify data** - Classify data processed, stored or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs. Apply controls as per the classification.
- **Encryption:** Make sure to encrypt all sensitive data at rest. Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS)
- **Strengthen Password** - Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt or PBKDF2. Salt and Pepper will increase the strength of your user passwords and make it very difficult to crack.



# VWA Security Report

## VWA21-1-14-9 - A1:2017 - Injection - CRITICAL

Vulnerability Exploited: A1 - Injection (customer)

Severity: [Critical]

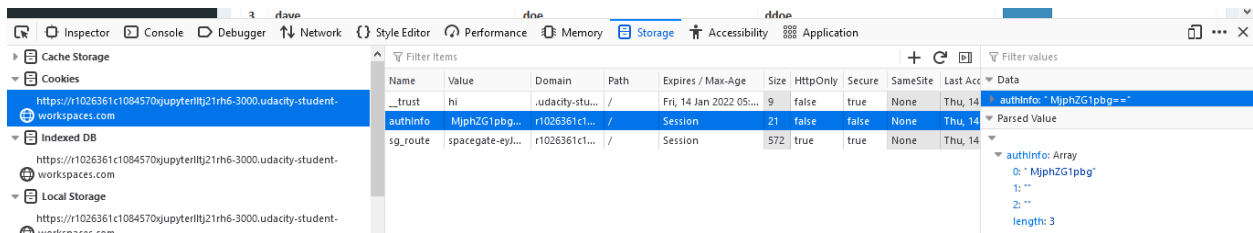
System: VWA Web Application

### Vulnerability Explanation:

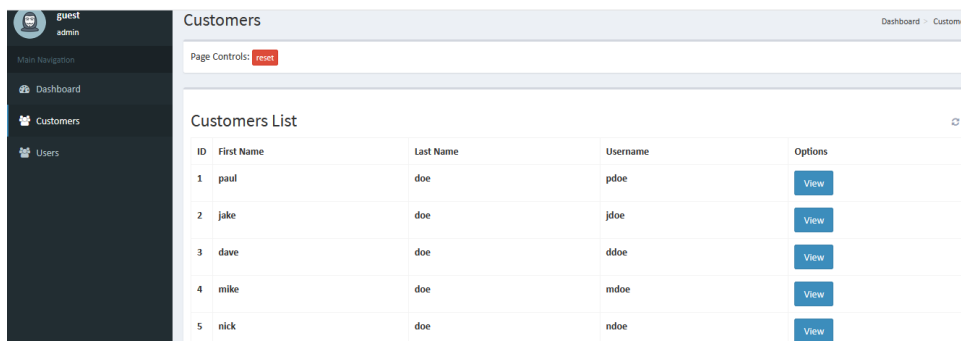
A1-injection occur when untrusted data is sent to an interpreter as part of a command or query. Injection flaws are SQL, NoSQL, OS, and LDAP. We found out SQL injection in VulWebApp. An attacker can exploit the application by using one of the most common SQL Injection commands. An attacker must first breakout of the current SQL query which will then trick the Database into executing the malicious code.

### Vulnerability Walk-thru:

1. I explained it in VWA21-1-14-3 how I got the string value of 2:admin to get access as admin.

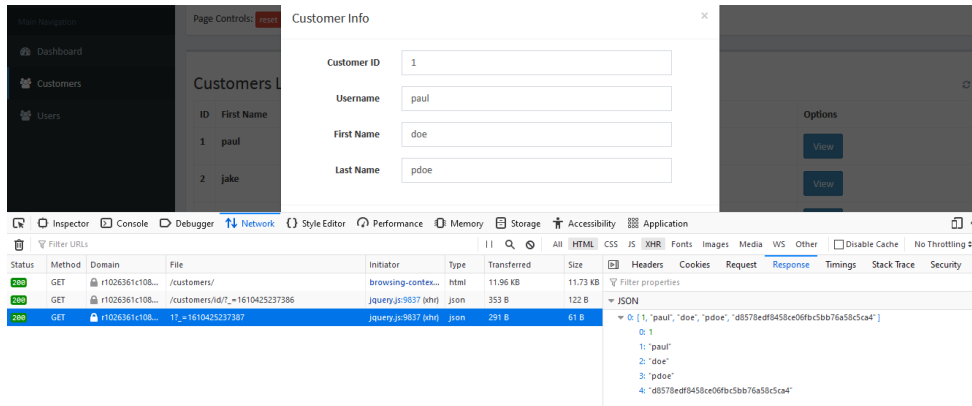


2. After changing the authinfo value refresh the customer page.

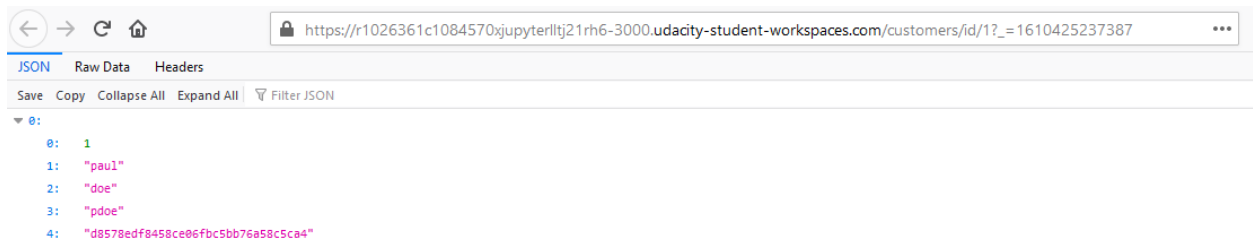


# VWA Security Report

3. Open inspect go to network tab. Click view to see information of customers.

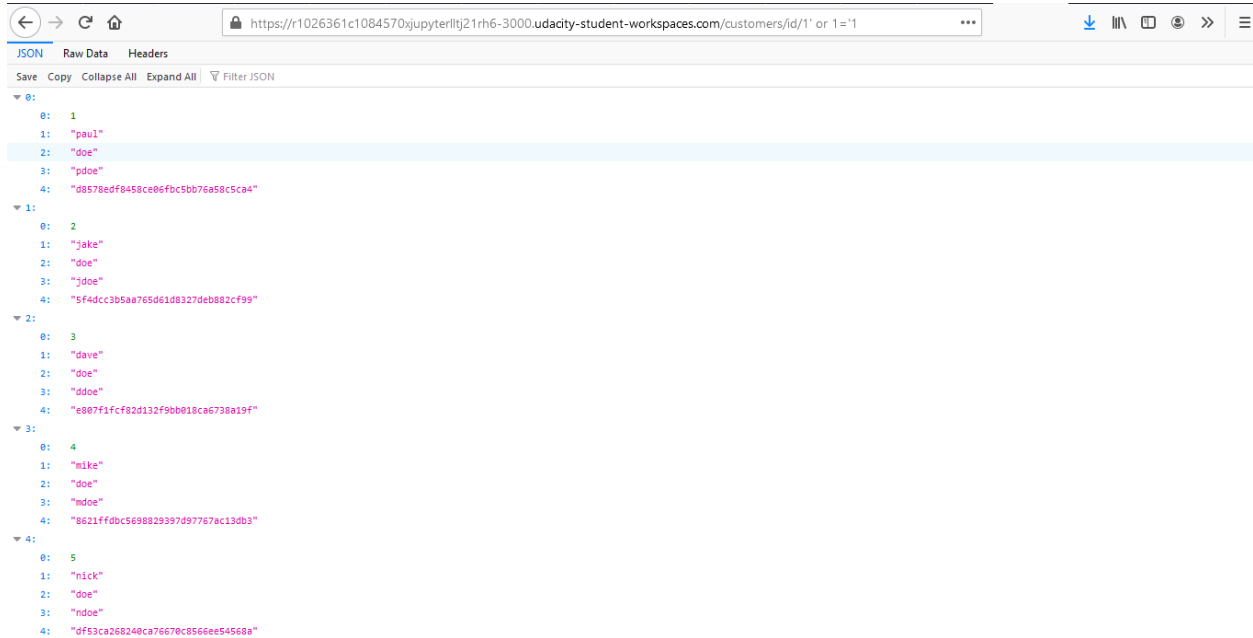


4. Open the customer id=1 information from network in new tab.



# VWA Security Report

5. Using the most common SQL `` or 1='1" injection in one customer url found the information of all customer.



```
0: 1
  1: "paul"
  2: "doe"
  3: "pdoe"
  4: "d8578edf8458ce06fbc5bb76a58c5ca4"
1: 2
  1: "jake"
  2: "doe"
  3: "jdoe"
  4: "5f4dcc3b5aa765d61d8327deb882cf99"
2: 3
  1: "dave"
  2: "doe"
  3: "ddoe"
  4: "e807f1fcf82d132f9bb018ca6738a19f"
3: 4
  1: "mike"
  2: "doe"
  3: "mdoe"
  4: "8e21ffdb0c569829397d9776ac13db3"
4: 5
  1: "nick"
  2: "doe"
  3: "ndoe"
  4: "df53ca268240ca76670c8566ee54568a"
```

# VWA Security Report

## Recommendations:

<https://cheatsheetseries.owasp.org/cheatsheets/SQL Injection Prevention Cheat Sheet.html>

## Best Practice:

- **Use Parameterized Queries** - This is the best method in preventing SQL Injection, because all variables are limited to the data type which will prevent malicious code from breaking out of the SQL code and preventing the malicious code from running.
- **Sanitize all inputs** - I recommend that you always sanitize all inputs before using them, this will prevent malicious code from running not only in SQL Queries but also in other parts of your code.

# VWA Security Report

**VWA21-1-14-10 - A1:2017 - Injection - CRITICAL**

**Vulnerability Exploited:** A1 - Injection Advance SQL (profile)

**Severity:** [Critical]

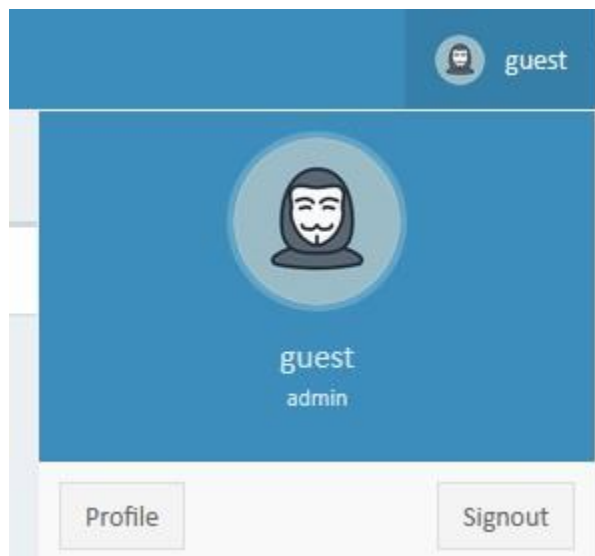
**System:** VWA Web Application

## **Vulnerability Explanation:**

A1-injection occur when untrusted data is sent to an interpreter as part of a command or query. Injection flaws are SQL, NoSQL, OS, and LDAP. We found out SQL injection in VulWebApp. An attacker can exploit the application by using one of the most common SQL Injection commands. An attacker must first breakout of the current SQL query which will then trick the Database into executing the malicious code.

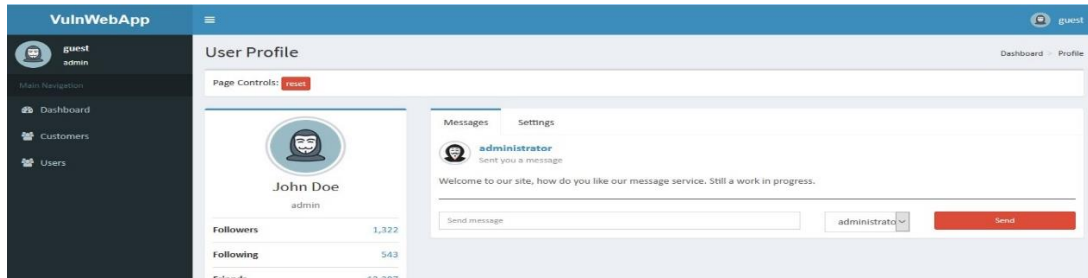
## **Vulnerability Walk-thru:**

1. Go to the profile page as admin.

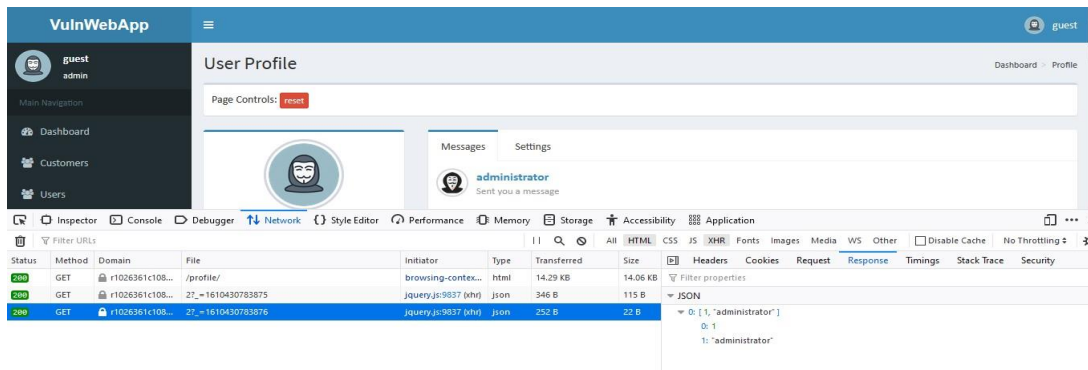


# VWA Security Report

## 2. Profile page of admin



## 3. Open the inspect and reload the network tab to see userlist.



## 4. Open then selected inspect in new tab.



# VWA Security Report

5. To see the number of columns inject the SQL injection  
"-2' ORDER BY 2--"



6. Again inject the same SQL increasing the value from 2 to 3  
"-2' ORDER BY 3--". Here error is showing which means there is only two user value in userlist.

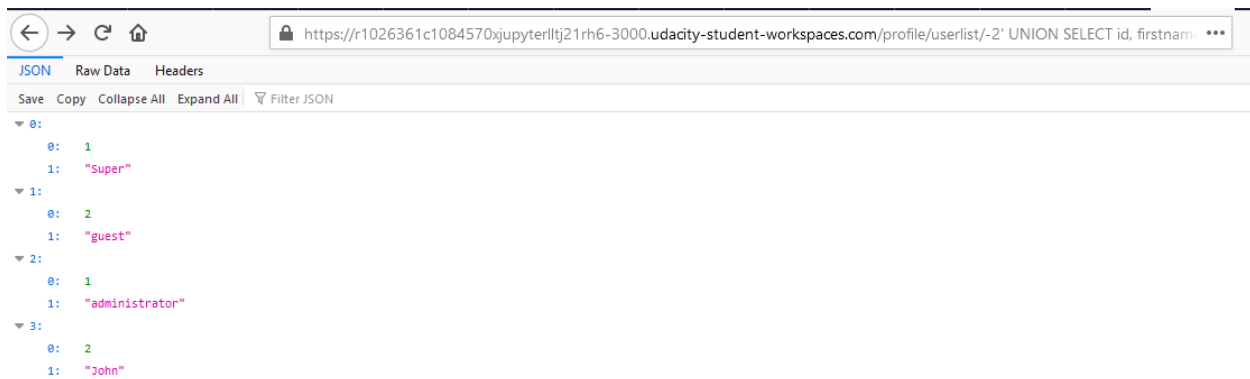


## Internal Server Error

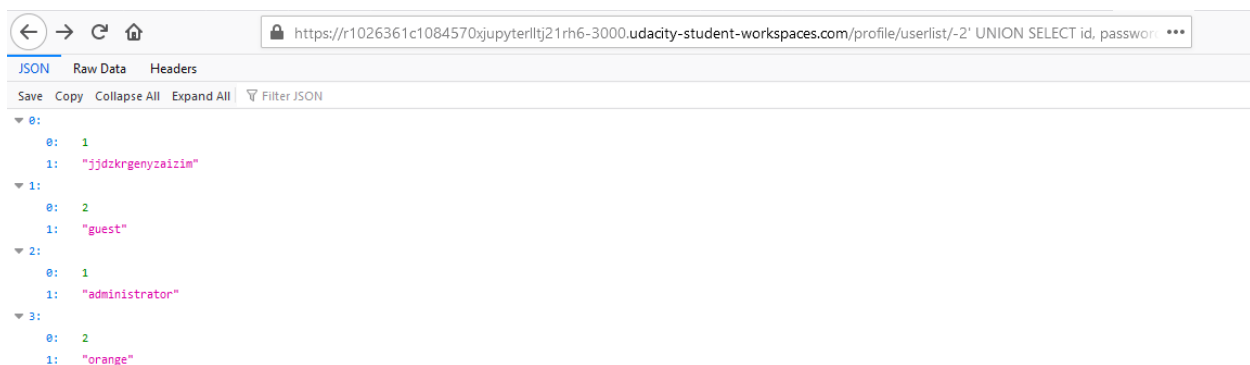
The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

7. To get the details by changing parameters we will inject -2' UNION SELECT id, Firstname FROM users WHERE username!='0. Here username!='0 means need to show both users details . All details of both guest and administrator will be changed from Firstname parameter to Lastname and username parameter.

# VWA Security Report



8. To get the password from the details we will put password in firstname place. inject `-2' UNION SELECT id, password FROM users WHERE username!='0`. For both guest and administrator, we got the password parameter.



## Recommendations:

<https://cheatsheetseries.owasp.org/cheatsheets/SQL Injection Prevention Cheat Sheet.html>

## Best Practice:

- **Use Parameterized Queries** - This is the best method in preventing SQL Injection, because all variables are limited to the data type which will prevent malicious code from



# VWA Security Report

breaking out of the SQL code and preventing the malicious code from running.

- **Sanitize all inputs** - I recommend that you always sanitize all inputs before using them, this will prevent malicious code from running not only in SQL Queries but also in other parts of your code.

**VWA21-1-14-11 - A1:2017 - Injection - HIGH**

**Vulnerability Exploited:** A1 - Injection SQL (profile)

**Severity:** [High]

**System:** VWA Web Application

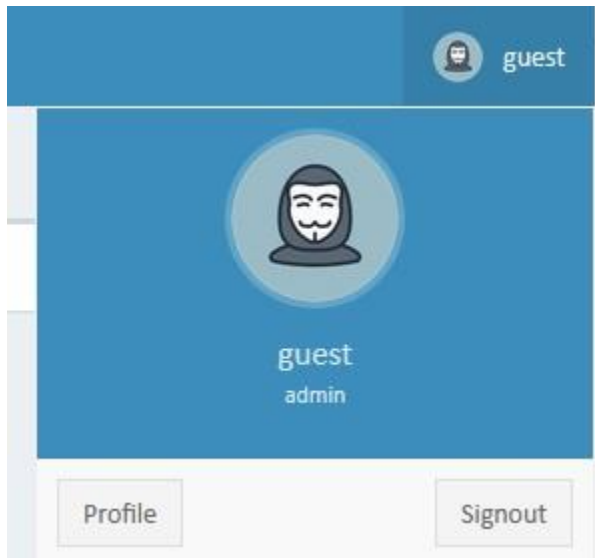
## **Vulnerability Explanation:**

A1-injection occur when untrusted data is sent to an interpreter as part of a command or query. Injection flaws are SQL, NoSQL, OS, and LDAP. We found out SQL injection in VulWebApp. An attacker can exploit the application by using one of the most common SQL Injection commands. An attacker must first breakout of the current SQL query which will then trick the Database into executing the malicious code.

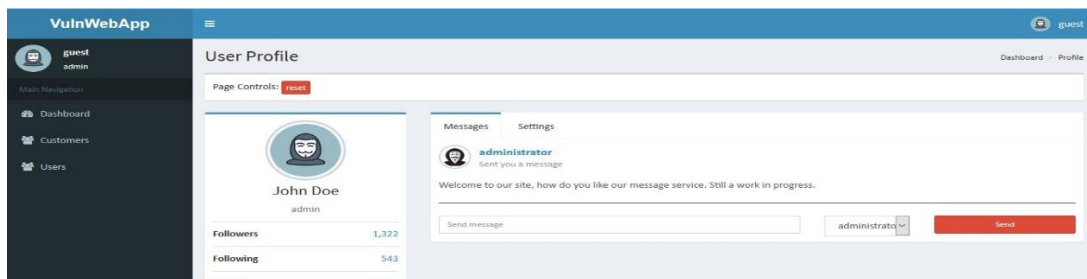
## **Vulnerability Walk-thru:**

1. Go to the profile page as admin.

# VWA Security Report

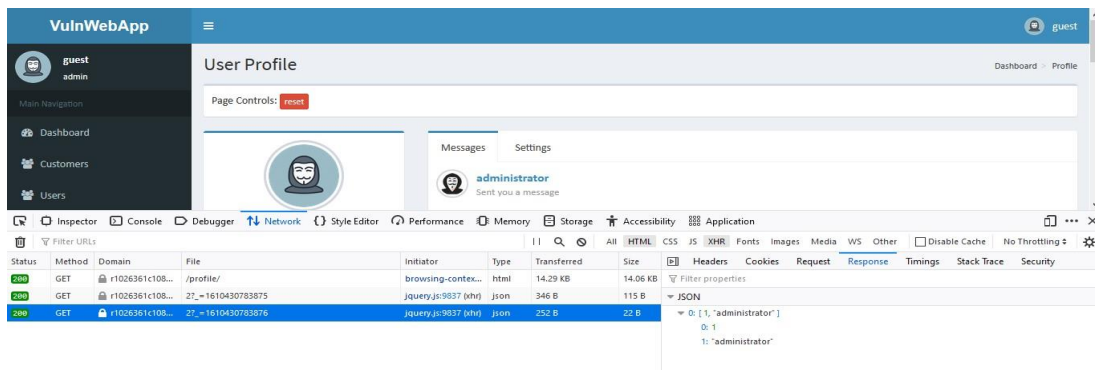


## 2. Profile page of admin



## 3. Open the inspect and reload the network tab to see userlist.

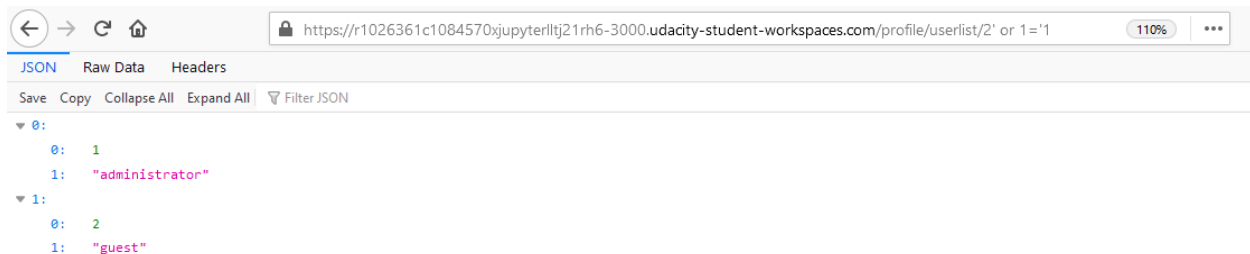
# VWA Security Report



4. Open then selected inspect in new tab.



5. Using the most common SQL `` or 1='1' injection in profile userlist url found the information of all userlist of profile.



## Recommendations:

VWAYMMDD - This document is confidential and for internal use only.

# VWA Security Report

<https://cheatsheetseries.owasp.org/cheatsheets/SQL Injection Prevention Cheat Sheet.html>

## Best Practice:

- **Use Parameterized Queries** - This is the best method in preventing SQL Injection, because all variables are limited to the data type which will prevent malicious code from breaking out of the SQL code and preventing the malicious code from running.
- **Sanitize all inputs** - I recommend that you always sanitize all inputs before using them, this will prevent malicious code from running not only in SQL Queries but also in other parts of your code.