

VulnWebApp (VWA) Security Report

Code Revision: 1.0.0.0

Company: Acme Inc.

Report: VWA240601

Author: [Shahsank Shekhar]

Date: [2024/05/23]

VWA Security Report

VWA240601## - XSS - Critical	3
VWA240601## - SQL Injection - Critical	5
VWA240601## - Brute Force Login - High	7
VWA240601## - Weak base64 Encryption - High	10
VWA240601## - Use of MD5 Hashing - High	12
VWA240601## - Cookie Manipulation - High	15
VWA240601## - Data Exposure - HIGH	18
VWA240601## - Broken Access For Customer's Data - High	20
VWA240601## - Broken Access For User's Data - High	23

Cookie manipulation typically falls under the OWASP Top Ten 2017 category of A6:2017 - Security Misconfiguration. 25

Security misconfiguration includes issues where security settings are not properly configured, which can lead to various vulnerabilities. Cookie manipulation can occur when cookies are not properly secured, allowing attackers to manipulate or steal them for malicious purposes such as session hijacking, privilege escalation, or identity theft.

By ensuring proper configuration and security measures for handling cookies, such as using secure flags, HttpOnly flags, and secure transport protocols (like HTTPS), developers can mitigate the risk of cookie manipulation and improve the overall security of their applications.

VWA Security Report

VWA240601## - XSS - Critical

Vulnerability Exploited: A07: 2017 Cross-Site Scripting XSS

Severity: Critical

System: VWA Web Application

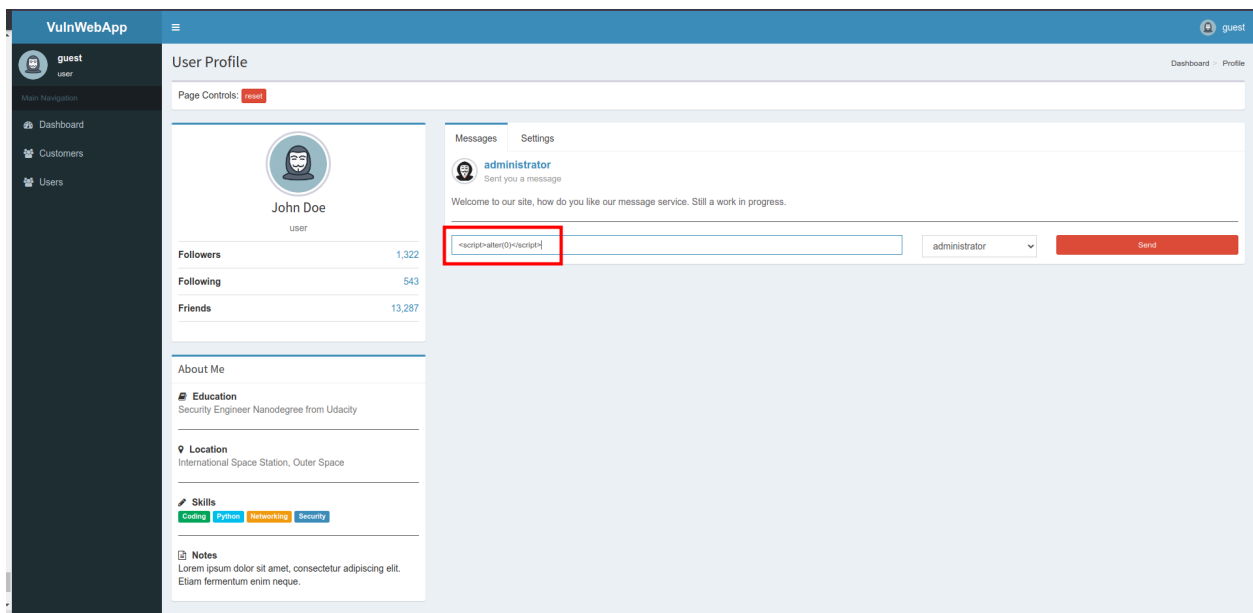
Vulnerability Explanation:

Cross-Site Scripting (XSS) is a common web security vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. These scripts can steal sensitive information, hijack user sessions, or deface websites.

In the internal chat section, the system is not doing any sanitization and allowing scripts, tags with javascript code on the client side. I tested by sending `<script>alert(1)</script>` in the internal chat box.

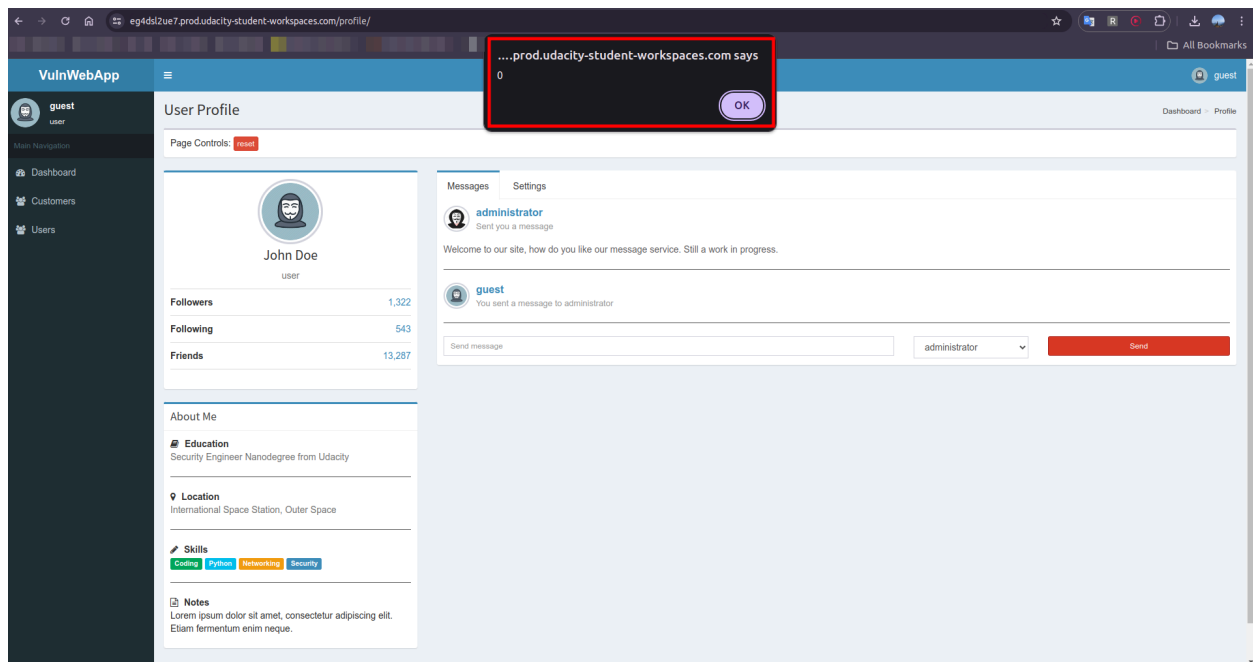
Vulnerability Walk-thru:

1. Login normally to the web app.
2. Goto user profile section and insert a script say `<script>alert(0)</script>` in the chat section.



VWA Security Report

3. With this we are able to inject javascript in the chat section.



Recommendations:

1. Preventing XSS requires separation of untrusted data from active browser content. This can be achieved by:
2. Using frameworks that 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.
3. Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities. The [OWASP Cheat Sheet 'XSS Prevention'](#) has details on the required data escaping techniques.
4. Applying context-sensitive encoding when modifying the browser document on the client side acts against DOM XSS. When this cannot be avoided, similar context sensitive escaping techniques can be applied to browser APIs as described in the [OWASP Cheat Sheet 'DOM based XSS Prevention'](#).
5. [OWASP Proactive Controls: Encode and Escape Data](#)

VWA Security Report

VWA240601## - SQL Injection - Critical

Vulnerability Exploited: A01: 2017 Injection

Severity: Critical

System: VWA Web Application

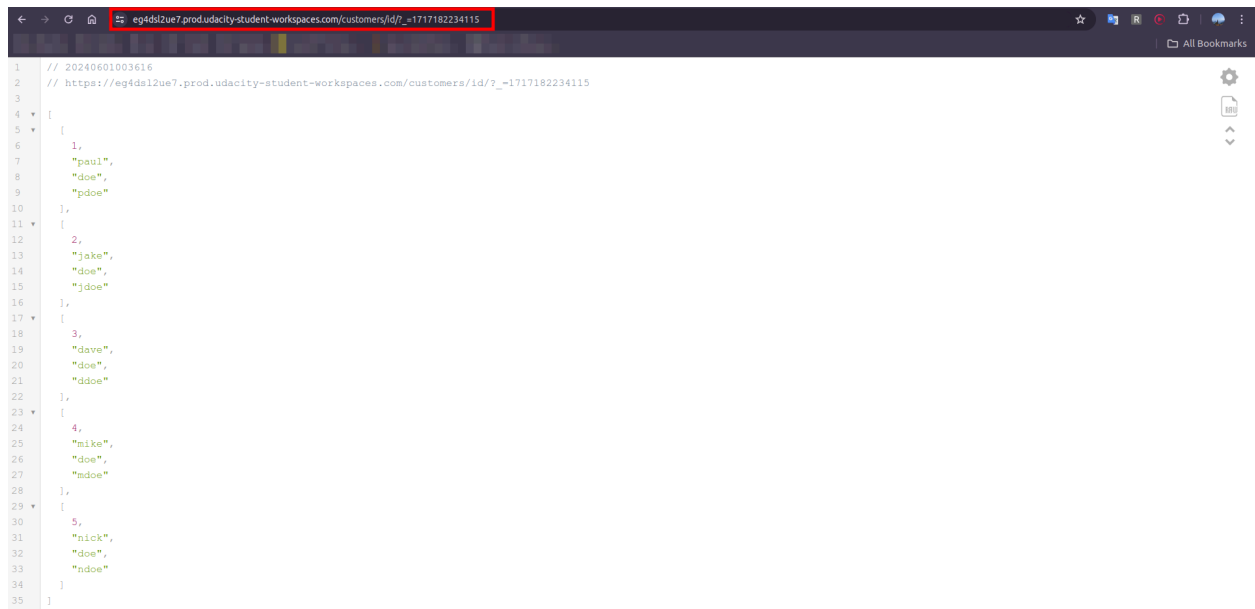
Vulnerability Explanation:

Inserting `' or 1 = '1` at the end of the url reveals all the user sensitive information. Ability to display the customer credentials with proper access/login using the sql injection in the url.

SQL injection (SQLi) is a type of security vulnerability that occurs when an attacker can manipulate SQL queries sent to a database through a web application. This manipulation can lead to unauthorized access to sensitive data, data manipulation, and in some cases, complete compromise of the underlying server.

Vulnerability Walk-thru:

1. Login to the web app
2. Gain admin privileges via cookie manipulation.
3. Go to customers tab



```
1 // 20240601003616
2 // https://eg4ds12ue7.prod.udacity-student-workspaces.com/customers/id/?_='1717182234115
3
4 {
5   [
6     1,
7     "paul",
8     "doe",
9     "pdoe"
10    ],
11    [
12      2,
13      "jake",
14      "doe",
15      "jdoe"
16    ],
17    [
18      3,
19      "dave",
20      "doe",
21      "ddoe"
22    ],
23    [
24      4,
25      "mike",
26      "doe",
27      "mdoe"
28    ],
29    [
30      5,
31      "nick",
32      "doe",
33      "ndoe"
34    ]
35  ]
36 }
```

VWA Security Report

4. Edit the URL and add ` or 1=' 1 towards the end of the url.

```
1 // 20240601003531
2 // https://eg4dd12ue7.prod.udacity-student-workspaces.com/customers/id/1%20or%201%20%201
3
4 {
5   [
6     1,
7     "paul",
8     "doe",
9     "d8578edf8458ce06fbc5bb76a58c5ca4"
10    ],
11   [
12     2,
13     "jake",
14     "doe",
15     "i doe",
16     "5f4dc3b5aa765d61d8327deb882cf99"
17    ],
18   [
19     3,
20     "dave",
21     "doe",
22     "ddoe",
23     "e807f1fcf82d132f9bb018ca6738a19f"
24    ],
25   [
26     4,
27     "mike",
28     "doe",
29     "mdoe",
30     "0621ffdbc5698829397d97767ac13db3"
31    ],
32   [
33     5,
34     "nick",
35     "doe",
36     "ndoe",
37     "df53ca268240ca76670cc856ee54568a"
38    ]
39  ]
40 }
```

5. Use hashid.py and checkhash.py to crack the hashes

```
tools root@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@root$ python checkhash.py -t md5 d8578edf8458ce06fbc5bb76a58c5ca4
Hashed value passed = d8578edf8458ce06fbc5bb76a58c5ca4
Hash type passed = md5
Result = query
Actual Hash Type should be = md5
tools root@root$ python checkhash.py -t md5 5f4dc3b5aa765d61d8327deb882cf99
Hashed value passed = 5f4dc3b5aa765d61d8327deb882cf99
Hash type passed = md5
Result = password
Actual Hash Type should be = md5
tools root@root$ python checkhash.py -t md5 e807f1fcf82d132f9bb018ca6738a19f
Hashed value passed = e807f1fcf82d132f9bb018ca6738a19f
Hash type passed = md5
Result = 124597890
Actual Hash Type should be = md5
tools root@root$ python checkhash.py -t md5 0621ffdbc5698829397d97767ac13db3
Hashed value passed = 0621ffdbc5698829397d97767ac13db3
Hash type passed = md5
Result = dragon
Actual Hash Type should be = md5
tools root@root$ python checkhash.py -t md5 df53ca268240ca76670cc856ee54568a
Hashed value passed = df53ca268240ca76670cc856ee54568a
Hash type passed = md5
Result = computer
Actual Hash Type should be = md5
tools root@root$
```

Recommendations:

1. The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized

VWA Security Report

interface, or migrate to use Object Relational Mapping Tools (ORMs).

2. Note: Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().
3. Use positive or "whitelist" server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
4. For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.
5. [OWASP Proactive Controls: Secure Database Access](#)
6. https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html

VWA240601## - Brute Force Login - High

Vulnerability Exploited: A02: 2017 Broken Authentication

Severity: High

System: VWA Web Application

Vulnerability Explanation:

Successful in performing brute force attack on the system.

This kind of vulnerability can occur when an attacker attacks an online application using a collection of common/harvest usernames and passwords using a brute force approach in the hope of discovering an account that is using a combination from the list.

Vulnerability Walk-thru:

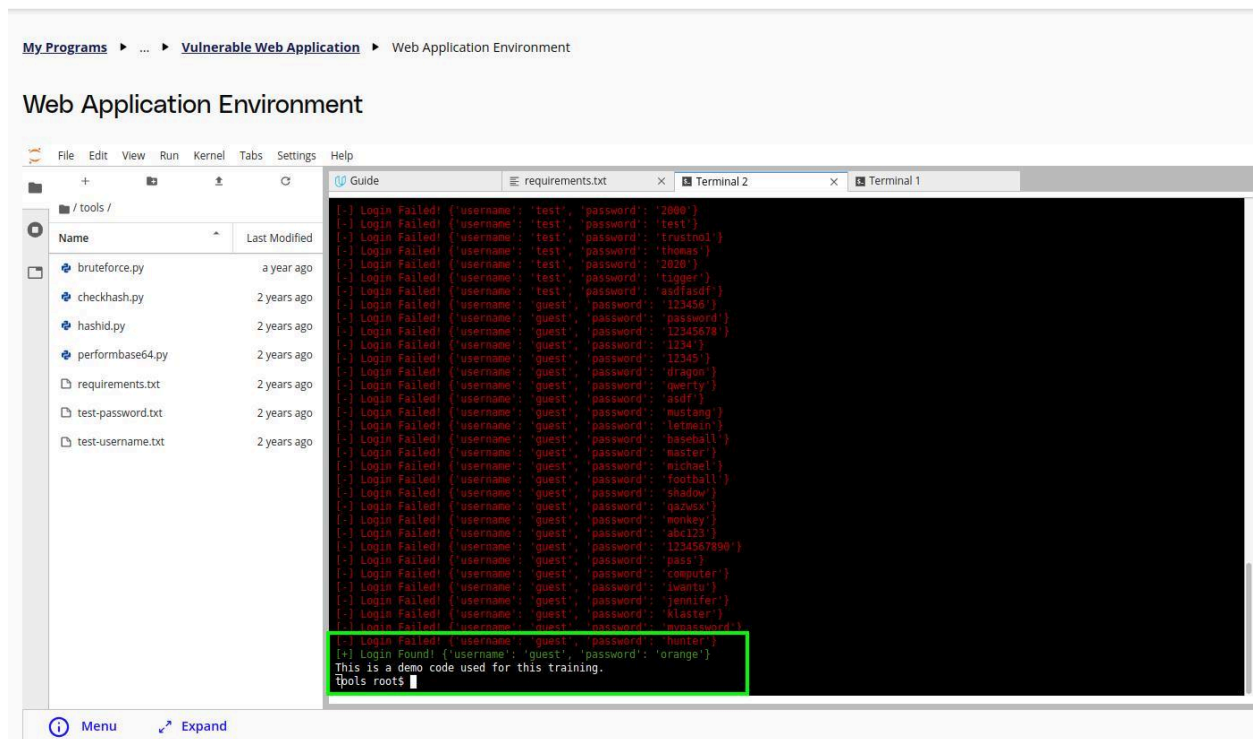
Use the bruteforce.py python tool which uses a collection of usernames and passwords to brute force attack.

VWA Security Report

I have used the below command in the terminal to do the attack:

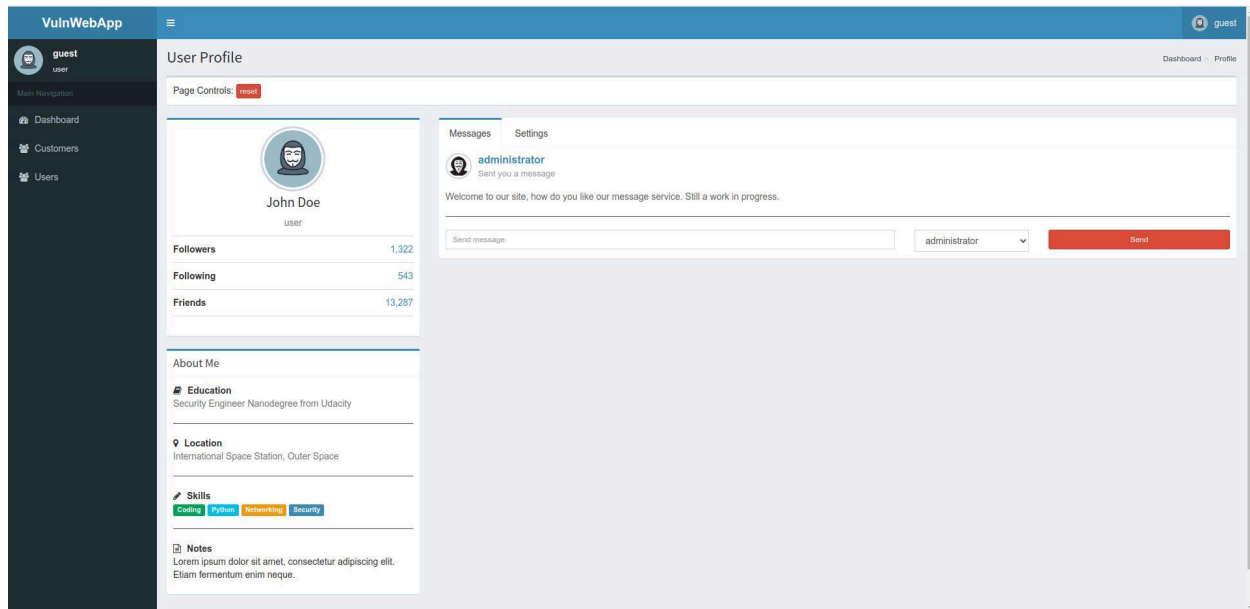
```
python bruteforce.py -U test-username.txt -P test-password.txt  
-d username=^USR^:password=^PWD^ -m POST -f "Login Failed"
```

<http://localhost:3000/login>



Running this command gave the username and password which exists for that app to log in.

VWA Security Report



Recommendations :

1. Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.
2. Do not ship or deploy with any default credentials, particularly for admin users.
3. Implement weak-password checks, such as testing new or changed passwords against a list of the [top 10000 worst passwords](#).
4. Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
5. Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.
6. [OWASP Proactive Controls: Implement Digital Identity](#)

VWA Security Report

VWA240601## - Use of MD5 Hashing - High

Vulnerability Exploited: A03: 2017 Sensitive Data Exposure

Severity: High

System: VWA Web Application

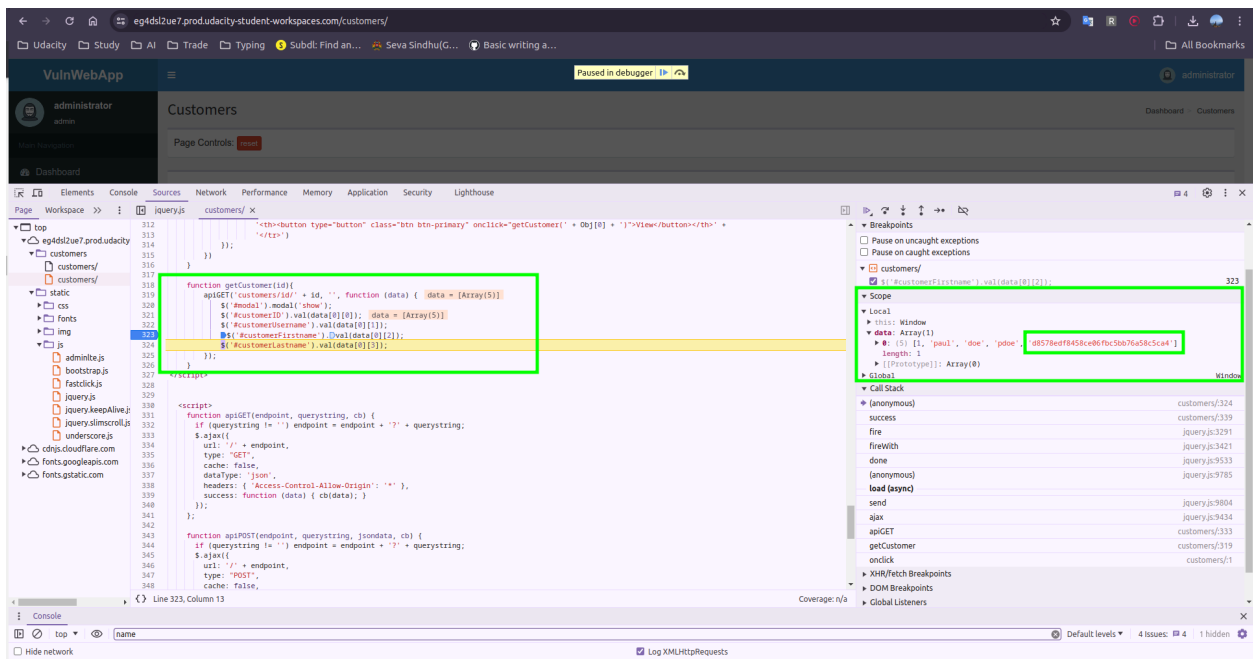
Vulnerability Explanation:

The app seems to be using a weak MD5 hashing to mask sensitive data what looks like password or something.

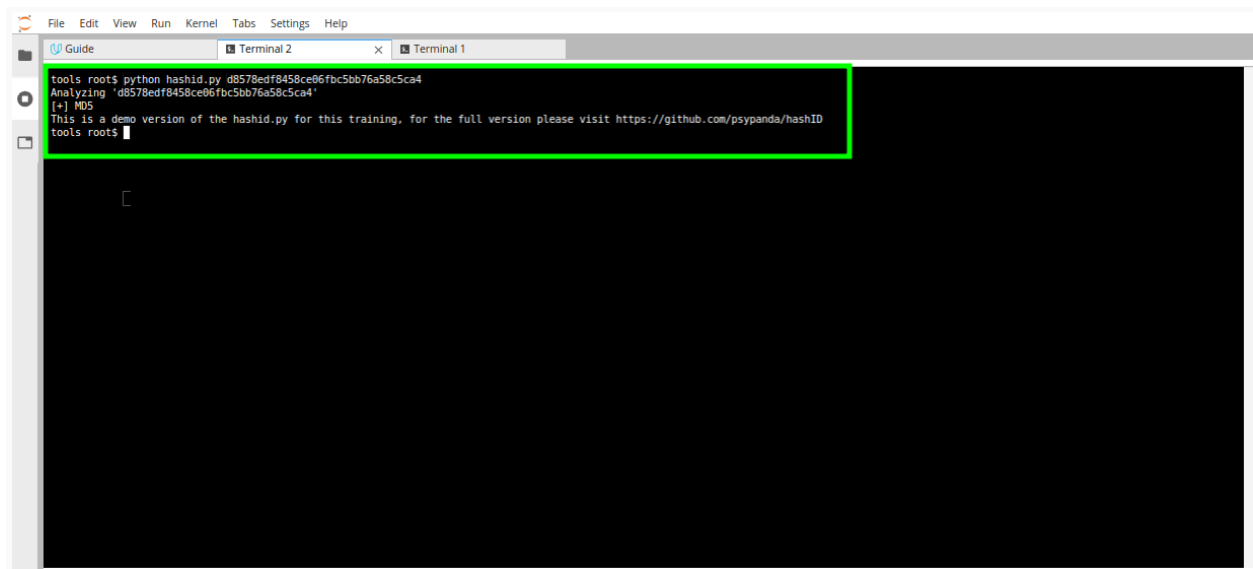
Vulnerability Walk-thru:

- 1.Login Normally
- 2.Gain Admin privilege through cookie manipulation as mentioned in the report.
- 3.Open developer tools using F12.
- 4.Go to sources tab and open customers file.
- 5.Add breakpoint at line 323.
- 6.View any of the customer data and use jump button to step to the breakpoint.
- 7.In the array select the last item which looks like some encrypted data.

VWA Security Report

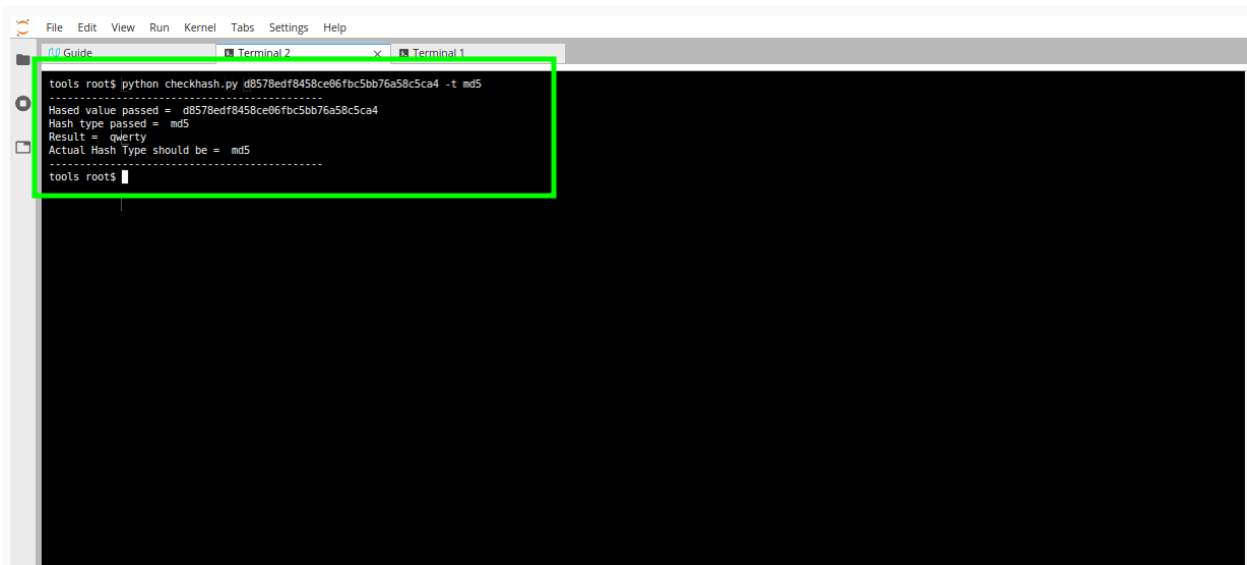


8. Use the hashid.py to know the hash type.



9. Use checkhash.py to crack the hash

VWA Security Report



The screenshot shows a terminal window with a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and two tabs (Terminal 2, Terminal 1). The terminal content is as follows:

```
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$
```

Recommendations:

1. Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as [Argon2](#), [scrypt](#), [bcrypt](#) or [PBKDF2](#).
2. Verify independently the effectiveness of configuration and settings.
3. Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
4. https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

VWA Security Report

VWA240601## - Cookie Manipulation - High

Vulnerability Exploited: A06: 2017 Security Misconfiguration

Severity: High

System: VWA Web Application

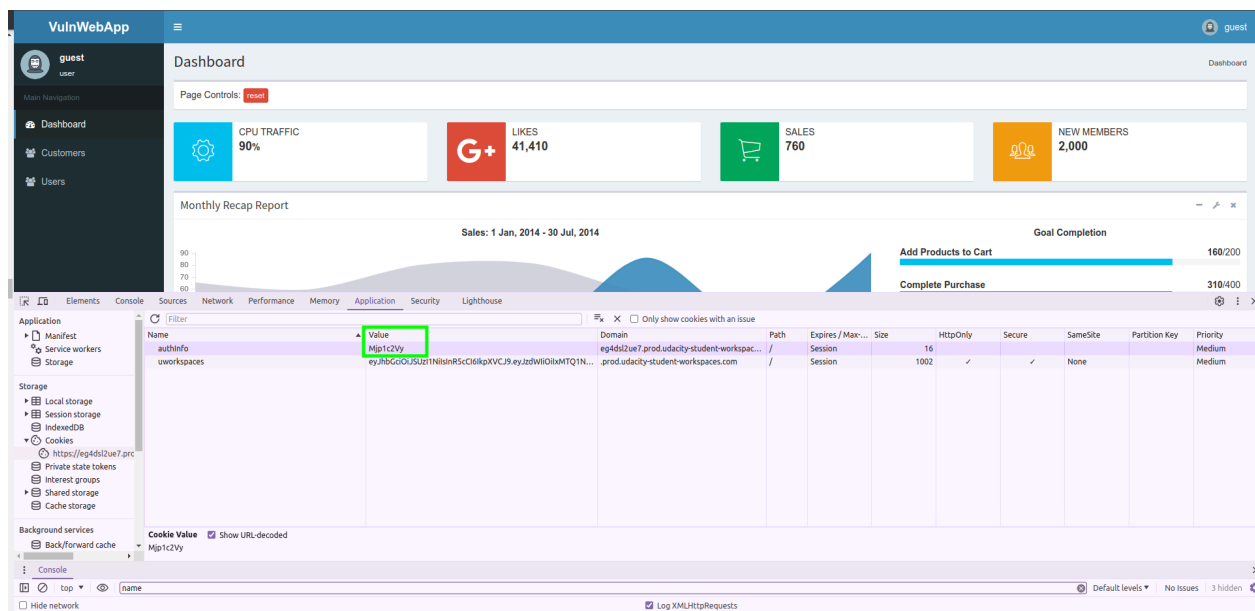
Vulnerability Explanation:

The cookie can be manipulated to gain the admin right in the web app.

I have modified the cookie values in the web page and I was able to gain admin access to the web app.

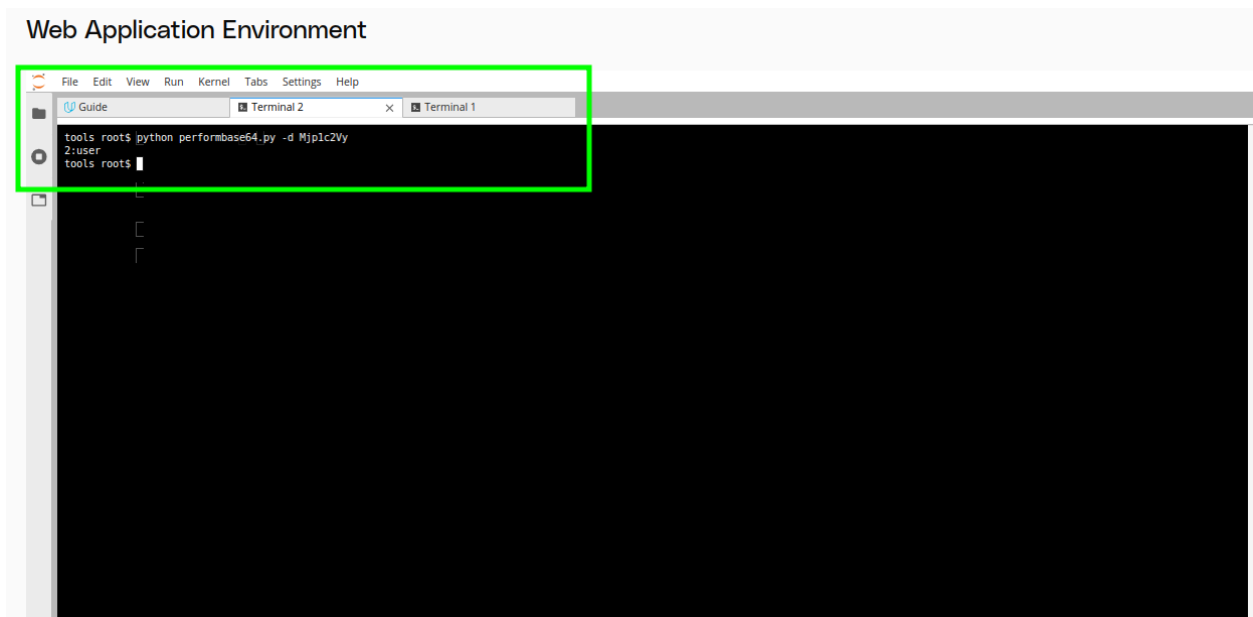
Vulnerability Walk-thru:

1. Login Normally
2. Open developer tool using F12
3. Go to the cookies tab

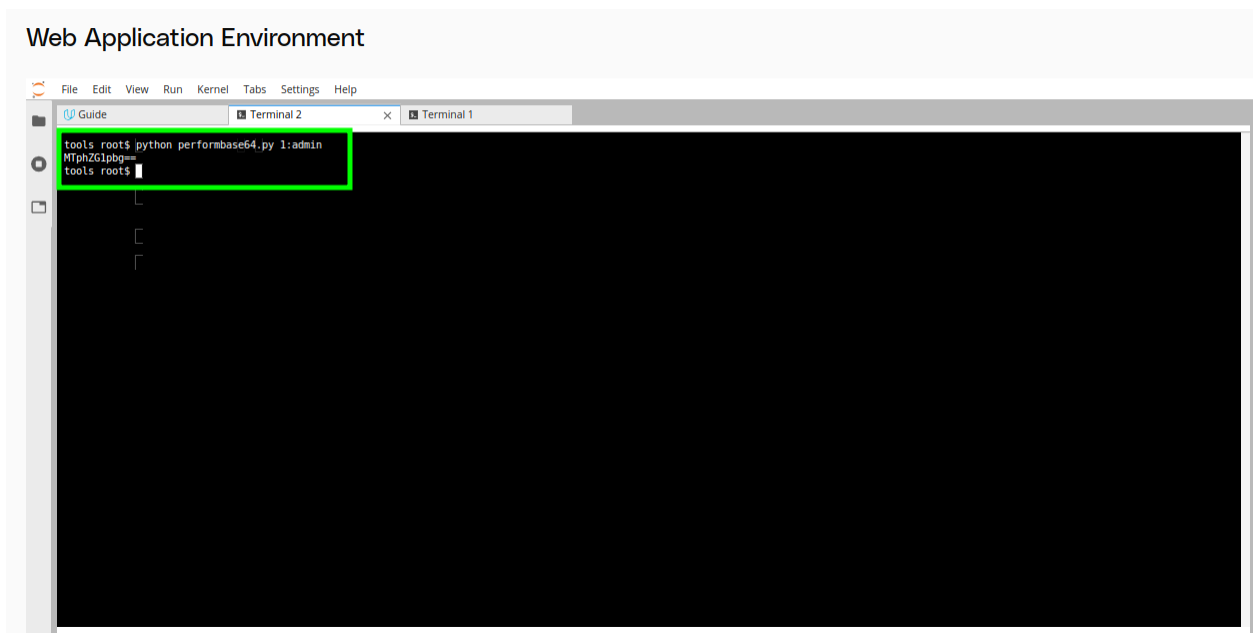


4. Perform base64 decoding of the cookie value which is returned for the user.

VWA Security Report



5. Encode **1:admin** via base64 to get the value to be replaced.



6. Replace the original cookie with the new encoded one and refresh the page.

The screenshot shows the VulnWebApp dashboard. In the top left, the 'administrator' profile is highlighted with a red box. The dashboard includes a 'Monthly Recap Report' for sales from Jan 1, 2014, to Jul 30, 2014, with a line graph and progress bars for 'Add Products to Cart' (160/200) and 'Complete Purchase' (310/400). The bottom section shows a list of cookies, with the 'Value' column highlighted by a red box. The cookie table has the following data:

Name	Value	Domain	Path	Expires / Max...	Size	HttpOnly	Secure	SameSite	Partition Key	Priority
authInfo	MTpbZG1pbpg==	eg4ds1zue7.prod.udacity-student-workspaces.com	/	Session	20					Medium
workspaces	eyJhBgGOLjU2ZT1Ni0uRSC6bikpXVCJ9.eyJzdWl0aW0MTQ1N...	.prod.udacity-student-workspaces.com	/	Session	1002			None		Medium

In the top right, the 'administrator' profile is again highlighted with a red box, showing 'Profile' and 'Signout' buttons.

Access control is only effective if enforced in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

- VWA240524 - This document is confidential and for internal use only.

VWA Security Report

VWA240601## - Data Exposure - HIGH

Vulnerability Exploited: A06: 2017 Security Misconfiguration

Severity: High

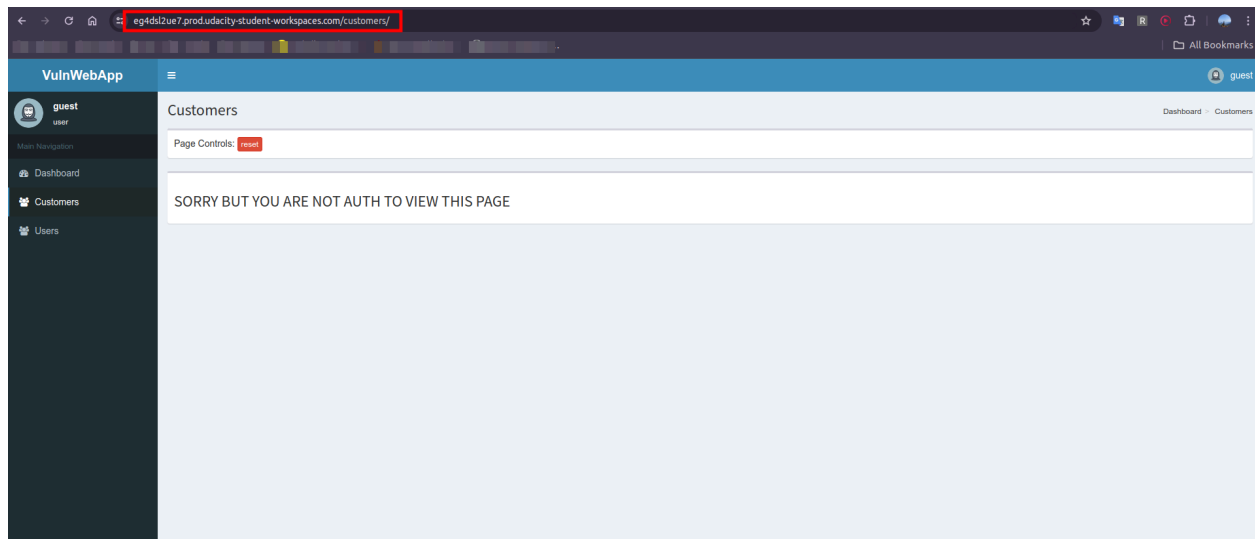
System: VWA Web Application

Vulnerability Explanation:

Attacker can access the data that is not authorized to be accessed. By adding /id/ towards the end of the url the data which is not supposed to be visible can be seen.

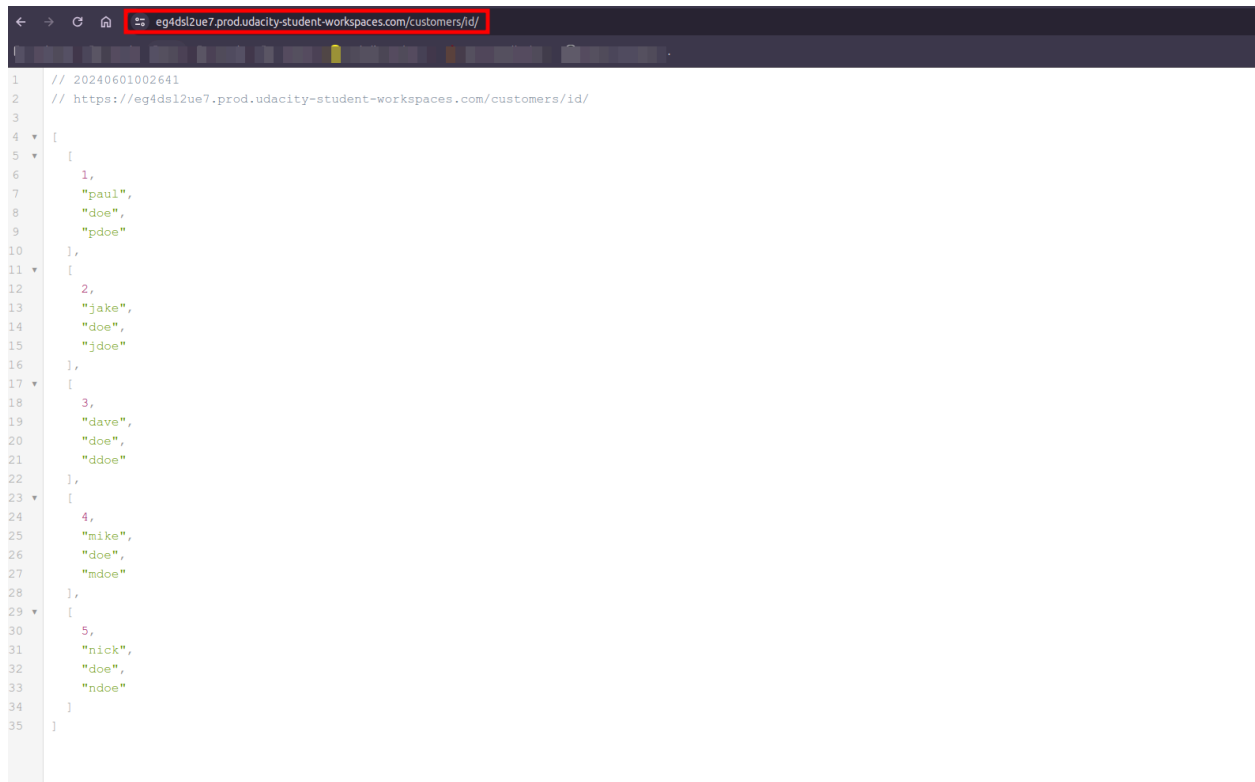
Vulnerability Walk-thru:

1. Login to the web app normally
2. Navigate to customer tab



3. Add /id/ to the end of the URL

VWA Security Report



```
1 // 20240601002641
2 // https://eg4ds12ue7.prod.udacity-student-workspaces.com/customers/id/
3
4 [
5   [
6     1,
7     "paul",
8     "doe",
9     "pdoe"
10  ],
11  [
12    2,
13    "jake",
14    "doe",
15    "jdoe"
16  ],
17  [
18    3,
19    "dave",
20    "doe",
21    "ddoe"
22  ],
23  [
24    4,
25    "mike",
26    "doe",
27    "mdoe"
28  ],
29  [
30    5,
31    "nick",
32    "doe",
33    "ndoe"
34  ]
35 ]
```

Recommendations:

1. A task to review and update the configurations appropriate to all security notes, updates and patches as part of the patch management process (see [A9:2017-Using Components with Known Vulnerabilities](#)). In particular, review cloud storage permissions (e.g. S3 bucket permissions).
2. A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
3. An automated process to verify the effectiveness of the configurations and settings in all environments.
4. [OWASP Testing Guide: Configuration Management](#)
5. https://owasp.org/www-project-web-security-testingguide/latest/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/README

VWA Security Report

VWA240601## - Broken Access For Customer's Data - High

Vulnerability Exploited: A05: 2017 Broken Access Control

Severity: High

System: VWA Web Application

Vulnerability Explanation:

A05:2017-Broken Access Control

One of the two request objects contains an async all to the server to retrieve the customer's current information.

By altering the initial request, we can use the API to retrieve the data of the other customer without any valid authentication.

Vulnerability Walk-thru:

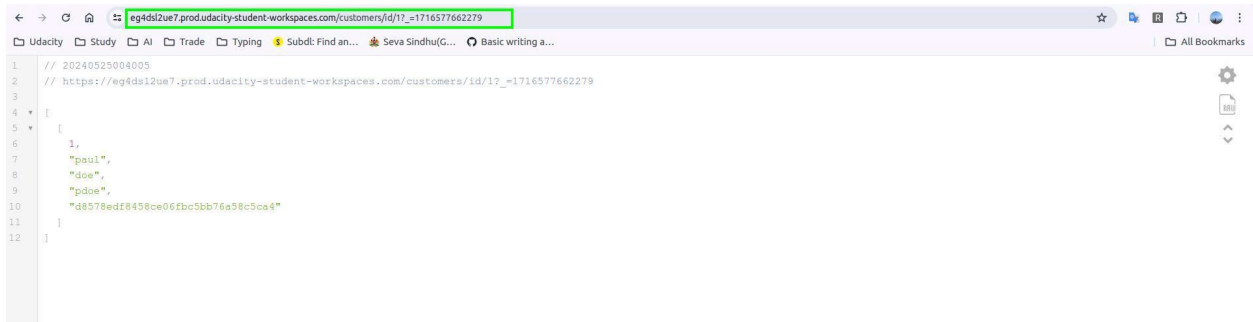
Once we get the admin access by manipulating the cookie:

1. Login Normally.
2. Gain admin access through cookie manipulation.
3. Go to the customer page.

The screenshot displays the VWA application interface. The sidebar on the left contains navigation links: Main Navigation, Dashboard, Customers, and Users. The main content area shows the 'Customers List' page. The table lists five customers with their IDs, first names, last names, and usernames. The 'Options' column for each customer has a 'View' button. A green box highlights the 'View' button for the first customer (paul), and a green arrow points to it from the right. Below the table, there is a 'Request Cookies' section showing details for the 'authInfo' and 'workspaces' cookies. The 'authInfo' cookie has a value 'MTphZG1pbG=='. The 'workspaces' cookie has a value 'eyJhGCo0JS01T1NlbnR5C0lkaXVCJ9.eYJ2bWl0aWMTQ1NDI3MDUyNC01Ym...'. The bottom of the screenshot shows the browser's developer tools with the 'Console' tab open, displaying 25 requests and 6.7 KB transferred.

VWA Security Report

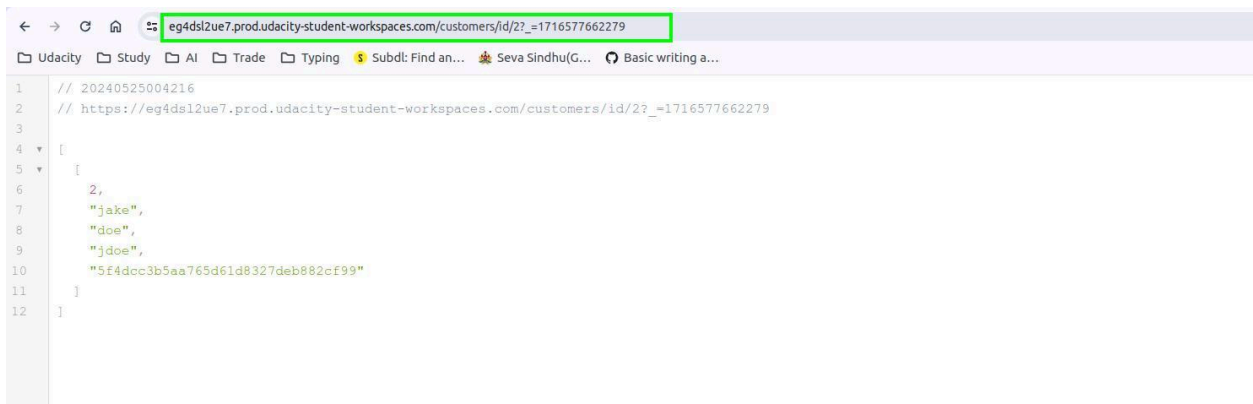
4. Hit the view button.
5. Go to the xhr file, then go to the response tab, and then open a new window by double-clicking it.



The screenshot shows a web browser with a REST client interface. The address bar contains the URL `eg4dsl2ue7.prod.udacity-student-workspaces.com/customers/id/17_1716577662279`. The response tab is selected, displaying a JSON array with three objects. The first object is highlighted in green.

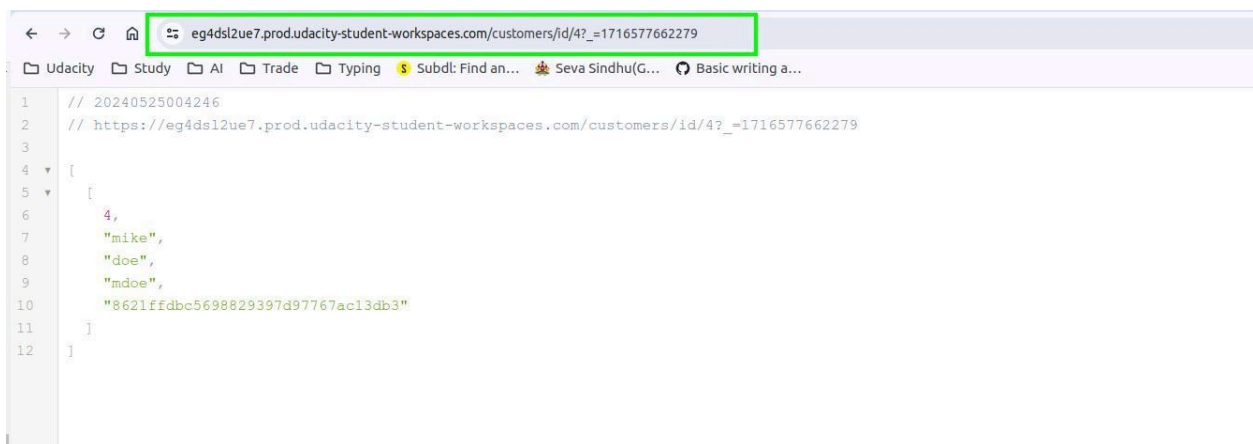
```
1 // 20240525004005
2 // https://eg4dsl2ue7.prod.udacity-student-workspaces.com/customers/id/17_1716577662279
3
4 [
5   {
6     "paul",
7     "doe",
8     "pdoe",
9     "d8578edf8458ce06fbc5bb76a58c5ca4"
10  }
11 ]
12
```

6. Here in the address bar customer ID can be modified to retrieve the information of other users.



The screenshot shows the same web browser with the REST client interface. The address bar now contains the URL `eg4dsl2ue7.prod.udacity-student-workspaces.com/customers/id/2?_1716577662279`. The response tab is selected, displaying a JSON array with three objects. The first object is highlighted in green.

```
1 // 20240525004216
2 // https://eg4dsl2ue7.prod.udacity-student-workspaces.com/customers/id/2?_1716577662279
3
4 [
5   {
6     "jake",
7     "doe",
8     "jdoe",
9     "5f4dcc3b5aa765d61d8327deb882cf99"
10  }
11 ]
12
```



The screenshot shows the same web browser with the REST client interface. The address bar now contains the URL `eg4dsl2ue7.prod.udacity-student-workspaces.com/customers/id/4?_1716577662279`. The response tab is selected, displaying a JSON array with three objects. The first object is highlighted in green.

```
1 // 20240525004246
2 // https://eg4dsl2ue7.prod.udacity-student-workspaces.com/customers/id/4?_1716577662279
3
4 [
5   {
6     "mike",
7     "doe",
8     "mdoe",
9     "8621ffdbc5698829397d97767ac13db3"
10  }
11 ]
12
```

Recommendations :

VWA Security Report

1. Prevent users from scrapping all the data from the web application by rate limiting the access to the data on the site.
2. Refuse access to the non-public pages and validation to access these pages.
3. [OWASP Proactive Controls: Enforce Access Controls](#)

VWA Security Report

VWA240601## - Broken Access For User's Data - High

Vulnerability Exploited: A05: Broken Access Control

Severity: High

System: VWA Web Application

Vulnerability Explanation:

A05:2017-Broken Access Control

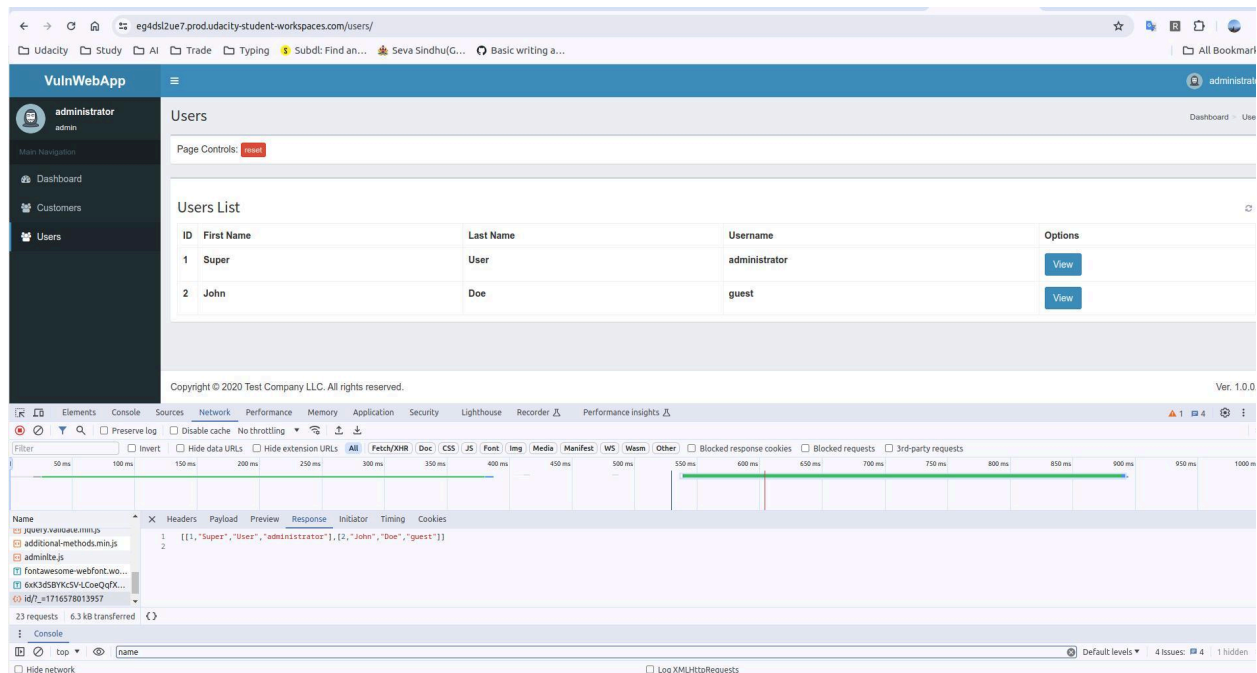
One of the two request objects contains an async all to the server to retrieve the user's current information.

By altering the initial request, we can use the API to retrieve the data of the other users without any valid authentication.

Vulnerability Walk-thru:

Once we get the admin access by manipulating the cookie:

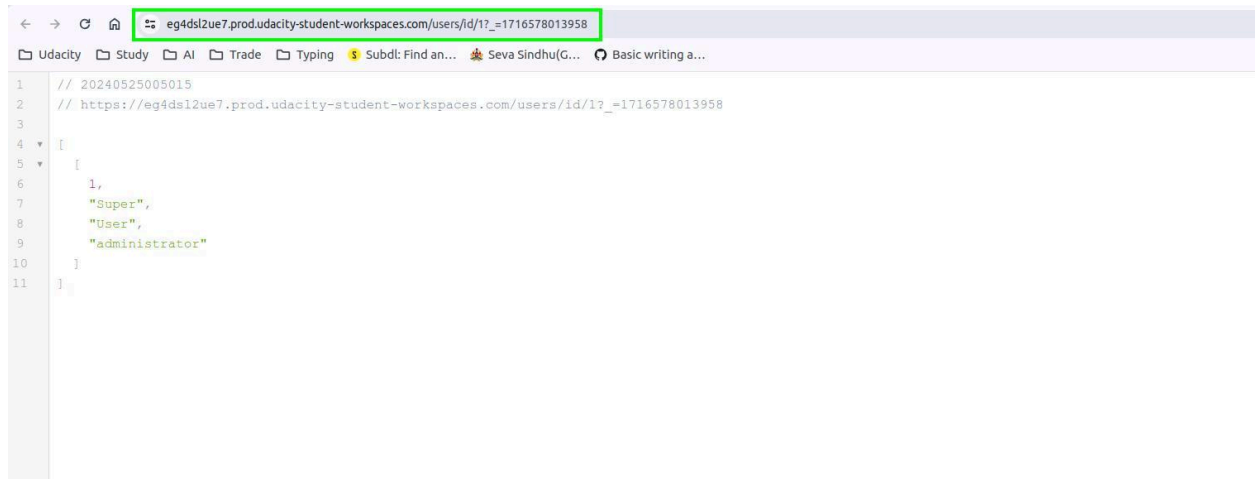
1. Login normally to the web app.
2. Gain admin right using cookie manipulation.
3. Go to the user page.



4. Hit the view button.

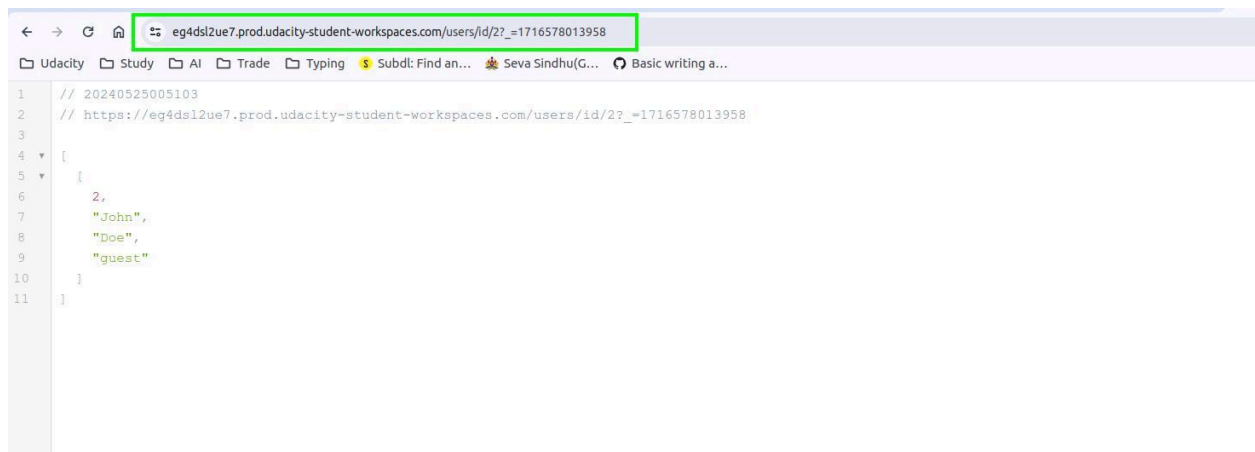
VWA Security Report

5. Go to the xhr file, then go to the response tab, and then open a new window by double-clicking it.



```
1 // 20240525005015
2 // https://eg4dsl2ue7.prod.udacity-student-workspaces.com/users/id/1?_=1716578013958
3
4 [
5   {
6     "Super",
7     "User",
8     "administrator"
9   }
10 ]
11
```

6. Here in the address bar user ID can be modified to retrieve the information of other users.



```
1 // 20240525005103
2 // https://eg4dsl2ue7.prod.udacity-student-workspaces.com/users/id/2?_=1716578013958
3
4 [
5   {
6     "John",
7     "Doe",
8     "guest"
9   }
10 ]
11
```

Recommendations:

1. Prevent users from scrapping all the data from the web application by rate limiting the access to the data on the site.
2. Refuse access to the non-public pages and validation to access these pages.
3. [OWASP Cheat Sheet: Access Control](#)

VWA Security Report

Cookie manipulation typically falls under the OWASP Top Ten 2017 category of A6:2017 - Security Misconfiguration.

Security misconfiguration includes issues where security settings are not properly configured, which can lead to various vulnerabilities. Cookie manipulation can occur when cookies are not properly secured, allowing attackers to manipulate or steal them for malicious purposes such as session hijacking, privilege escalation, or identity theft.

By ensuring proper configuration and security measures for handling cookies, such as using secure flags, HttpOnly flags, and secure transport protocols (like HTTPS), developers can mitigate the risk of cookie manipulation and improve the overall security of their applications.
