Secure Programming

Group Assignment 1

Students:

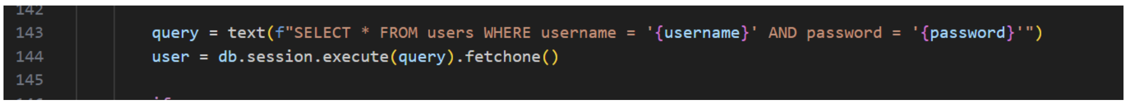
Kate Fitzpatrick B00154169, Paritchamon Barnwell B00160912, Evan Casey B00164583

Table of Contents

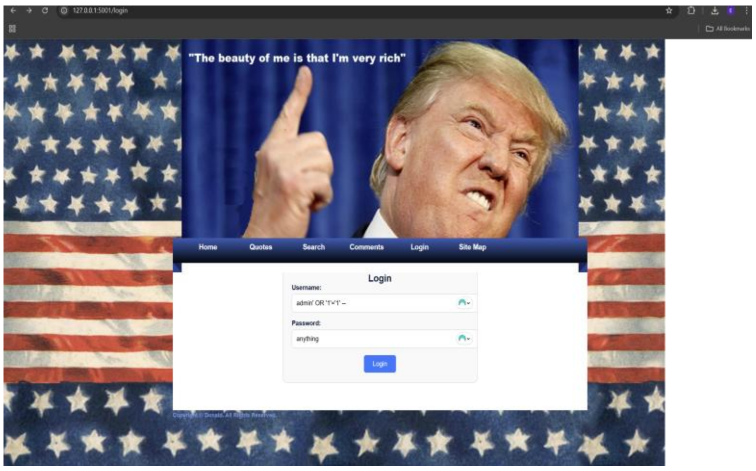
Risks and Vulnerabilities found

SQL injection

In regards to what I believe is the first vulnerability we detected here is that there is no sanitisation on the query or escape based on the user input the f string at the beginning essentially hurls user input into the query and no one is questioned everything is taken in to accept far too dangerous. As you can tell by the screenshot I input a suspect username (which is sql attack) and anything as a password what happens here is that the single quote kicks you out of the expected scope and allows you to do your own injection then we add a or condition and the - comments out the rest like the password check.



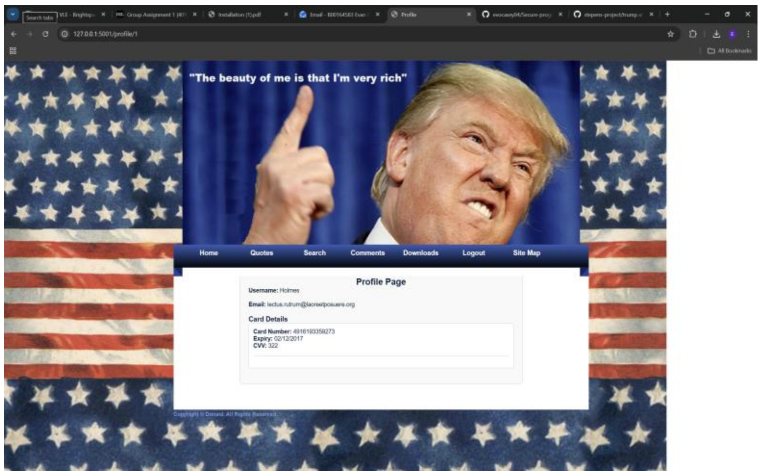
```
        query = text(f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'")
        user = db.session.execute(query).fetchone()
```



Login

Username: admin' OR '1'='1' --

Password: anything



Profile Page

Username: Holmes

Email: lectus.rutrum@laoreetposuere.org

Card Details

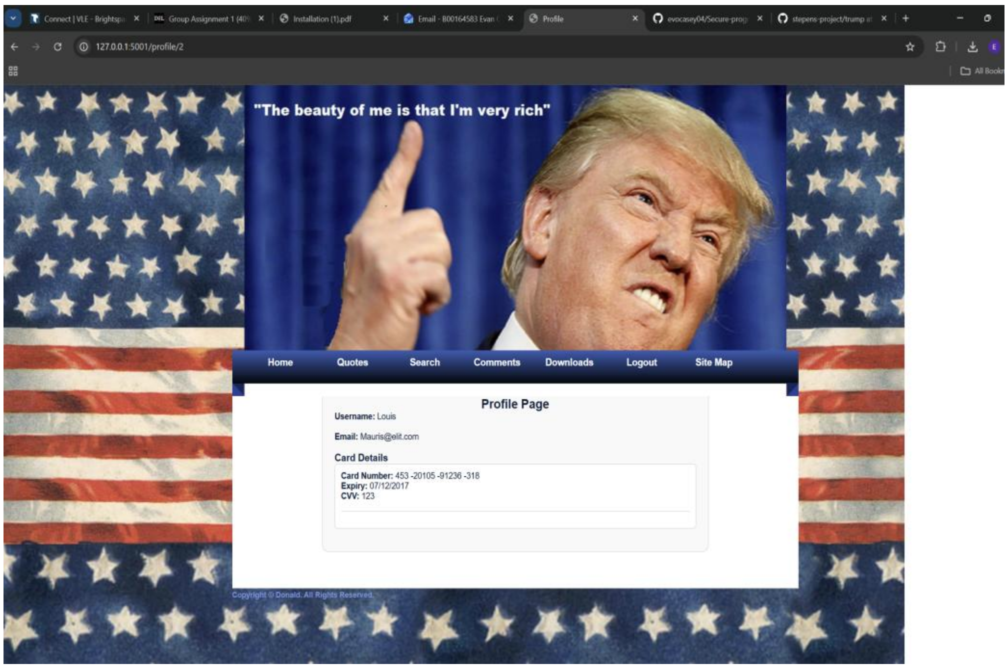
Card Number: 4916193359273
Expiry: 02/12/2017
CVV: 322

Card Details exposed

Data extraction

Once I fixed the SQL injection I then examined the URL, http://127.0.0.1:5001/profile/1 so I then thought perhaps changing the last number would give me different details which it did.

As you can see all we did was change it from a 1 to a 2; these are vulnerable because of how the query is created.


127.0.0.1:5001/profile/2
"The beauty of me is that I'm very rich"
Home
Quotes
Search
Comments
Downloads
Logout
Site Map
Copyright © Donald. All Rights Reserved.

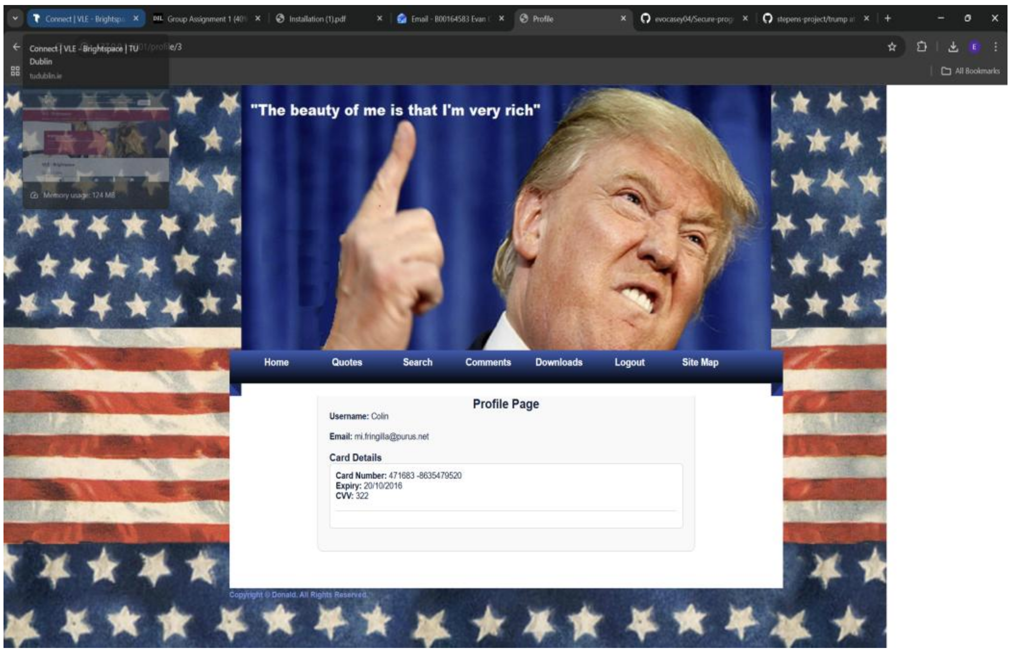
Profile Page

Username: Louis

Email: Mauris@elit.com

Card Details

Card Number: 453 -20105 -91236 -318
Expiry: 07/12/2017
CVV: 123


"The beauty of me is that I'm very rich"
Home
Quotes
Search
Comments
Downloads
Logout
Site Map
Copyright © Donald. All Rights Reserved.

Profile Page

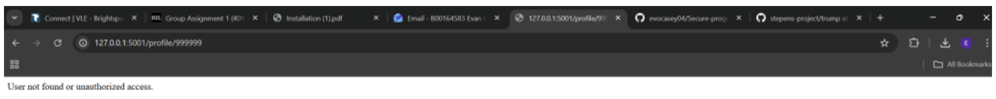
Username: Colin

Email: mi.fringilla@purus.net

Card Details

Card Number: 471683 -8635479520
Expiry: 20/10/2016
CVV: 322

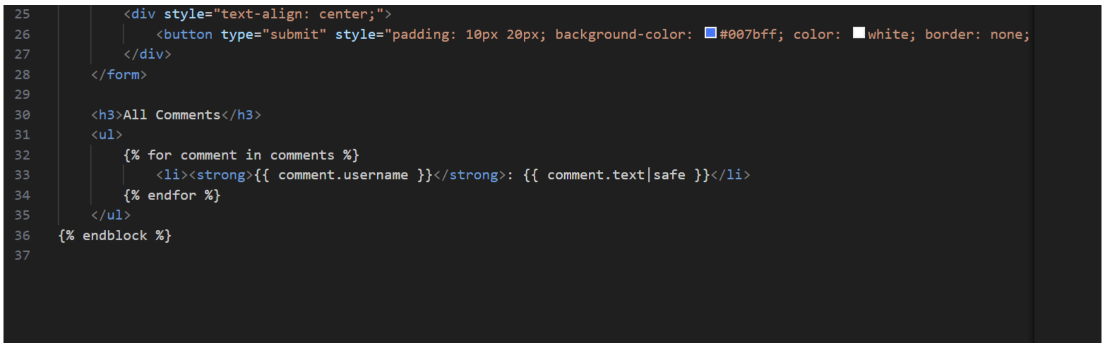
I am unsure if this one is anything at all, but it could be a hint at some sort of admin privileges account or something along those lines.



```
User not found or unauthorized access.
```

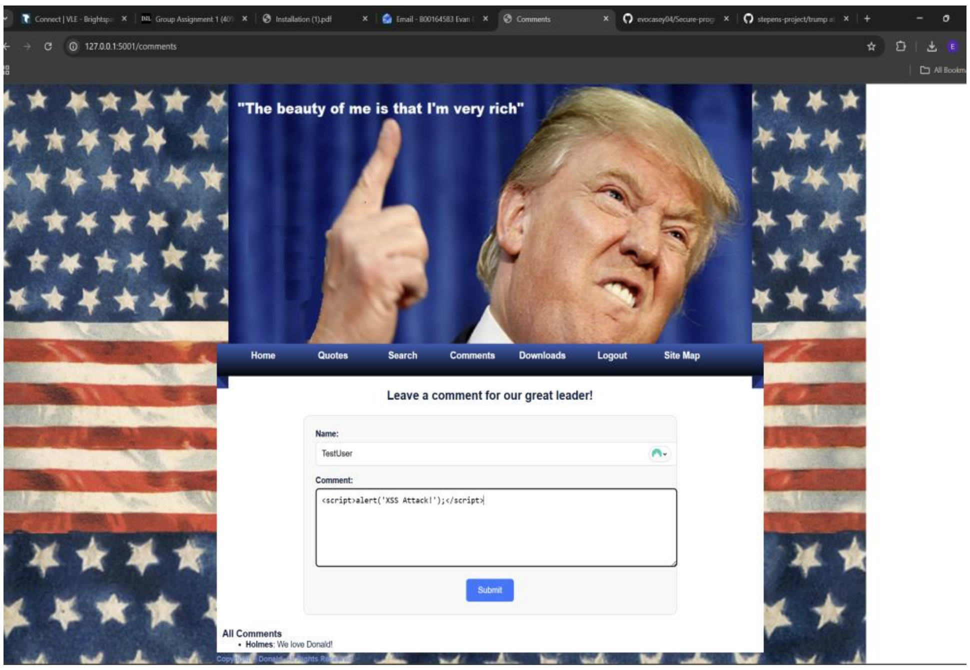
We tested with a very large number to see what would happen and was met with this unauthorized access which suggests a hint at the database structure.

Xss



```
25          <div style="text-align: center;">
26              <button type="submit" style="padding: 10px 20px; background-color: #007bff; color: white; border: none;
27          </div>
28      </form>
29
30      <h3>All Comments</h3>
31      <ul>
32          {% for comment in comments %}
33              <li><strong>{{ comment.username }}</strong>: {{ comment.text|safe }}</li>
34          {% endfor %}
35      </ul>
36  {% endblock %}
37
```

Line 33 is a security vulnerability the safe filters allow the template to display the page/render it without any escaping which means the JavaScript can run which means every single time anyone decides to go to the comment page now that I've rendered it, they'll be met with my message.

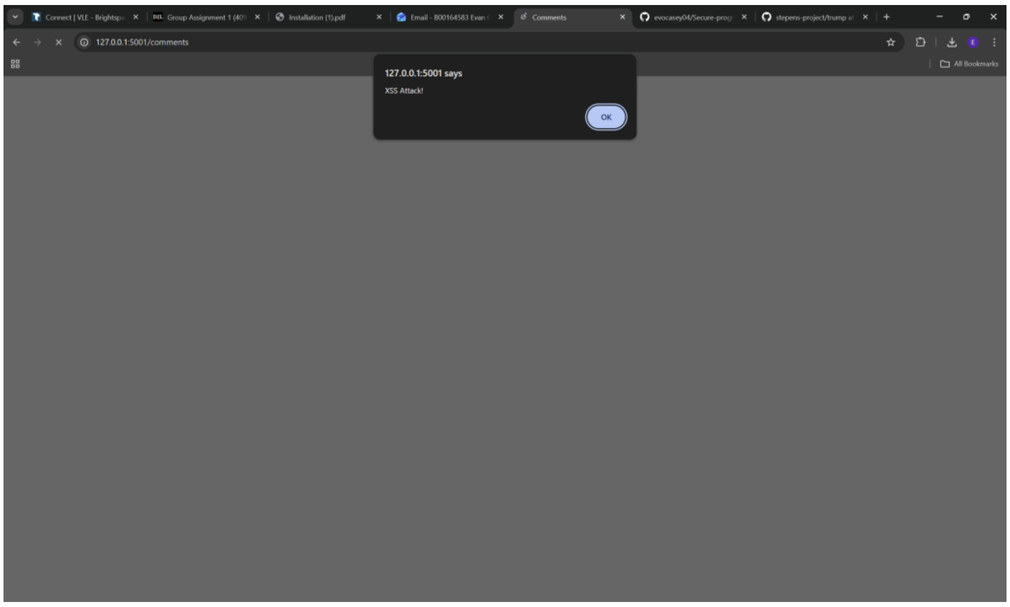


Leave a comment for our great leader!

Name:
TestUser

Comment:
```
<script>alert('XSS Attack!');</script>
```

All Comments
- **Holmes**: We love Donald!

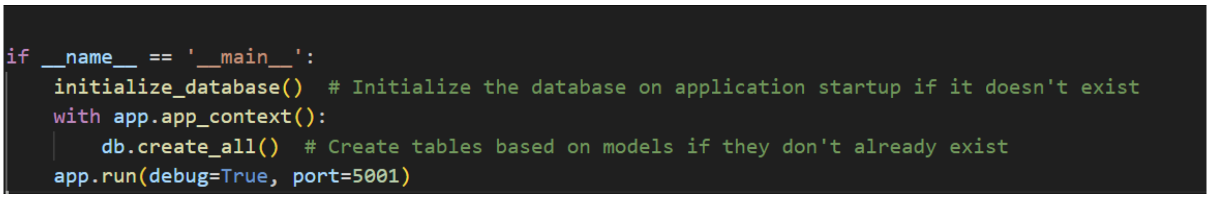


127.0.0.1:5001 says

XSS Attack!

Debug Mode

Debug mode is present in production which puts the site at risk to exposing sensitive information

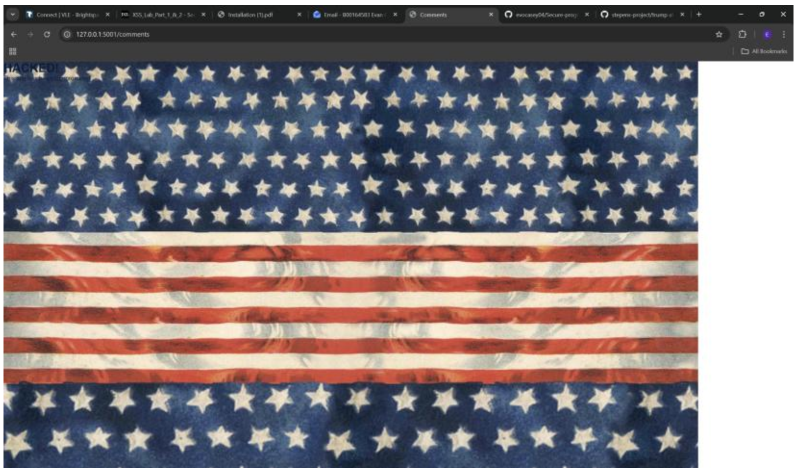


```python
if __name__ == '__main__':
    initialize_database()  # Initialize the database on application startup if it doesn't exist
    with app.app_context():
        db.create_all()  # Create tables based on models if they don't already exist
    app.run(debug=True, port=5001)
```

Defacement attack using xss

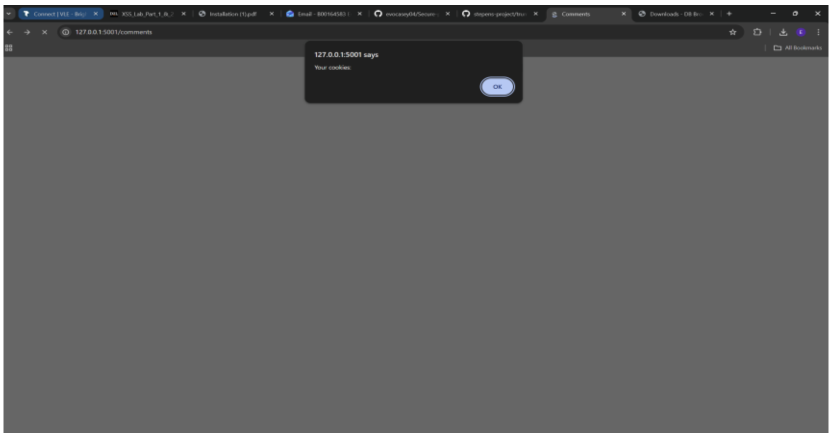
We completely changed the comments page using xss comments in this one.

```
<script>document.body.innerHTML = '<h1>HACKED!</h1><p>This site has been compromised</p>';</script>
```



```
<script>

if(document.cookie) {

   alert('Cookies found: ' + document.cookie);

} else {

   alert('No cookies - user not logged in');

}

</script>
```

you can use something like this to try and steal cookies usually, but we were unable to get it to work despite our best efforts



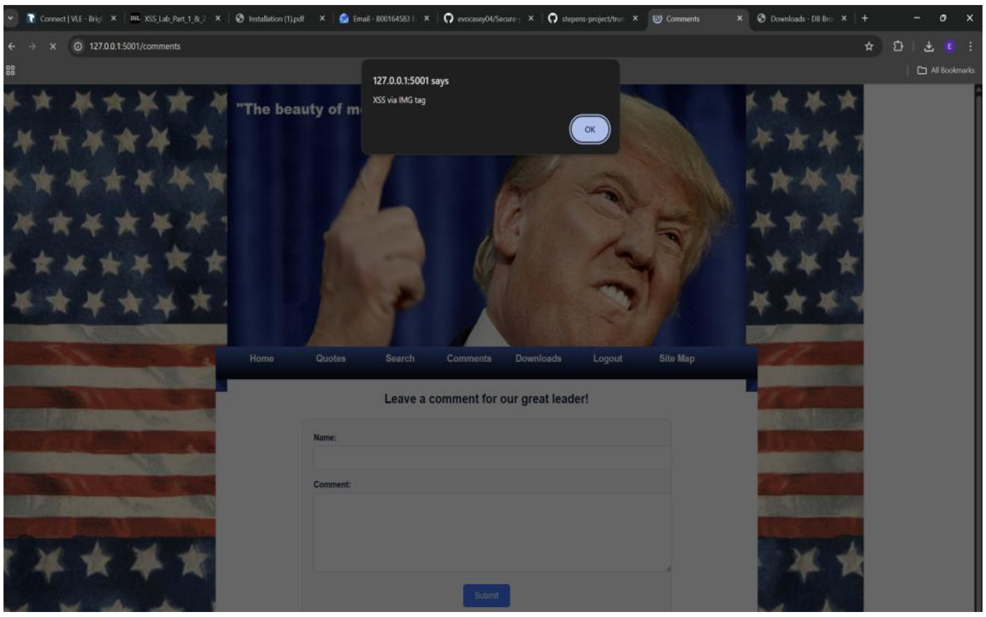
127.0.0.1:5001 says

Your cookies:

Xss via image tag

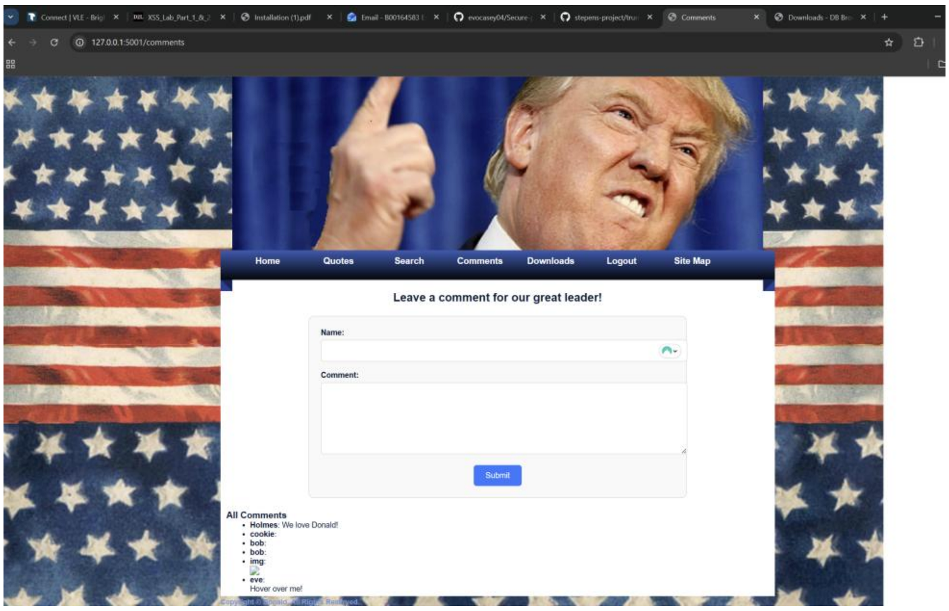
We did this just to show variation in how the site can be exposed to xss

<img src="x" onerror="alert('XSS via IMG tag')">



127.0.0.1:5001 says

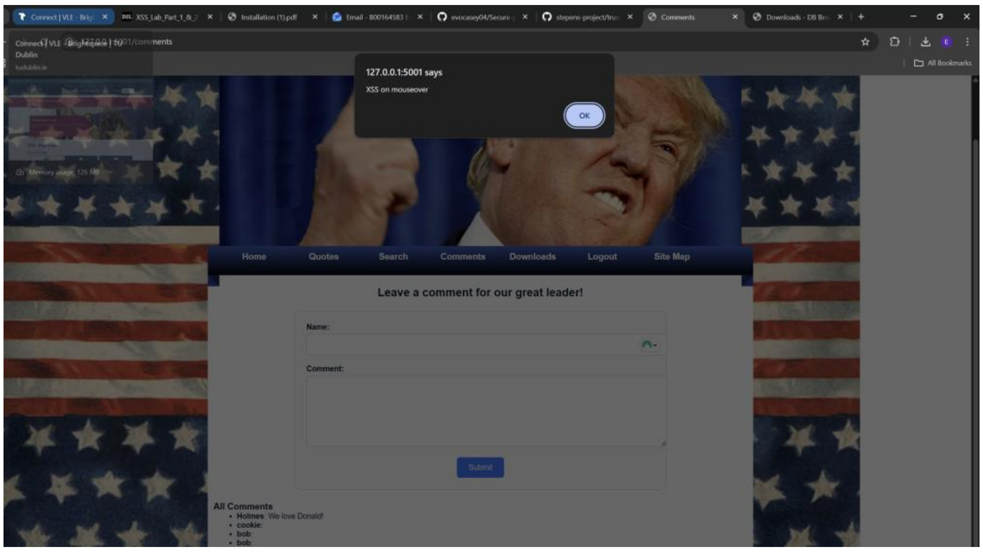
XSS via IMG tag



All Comments

- **Holmes**: We love Donald!
- **cookie**:
- **bob**:
- **bob**:
- **img**:
- **eve**:
  Hover over me!

<div onmouseover="alert('XSS on mouseover')">Hover over me!</div>


127.0.0.1:5001 says

XSS on mouseover

Leave a comment for our great leader!

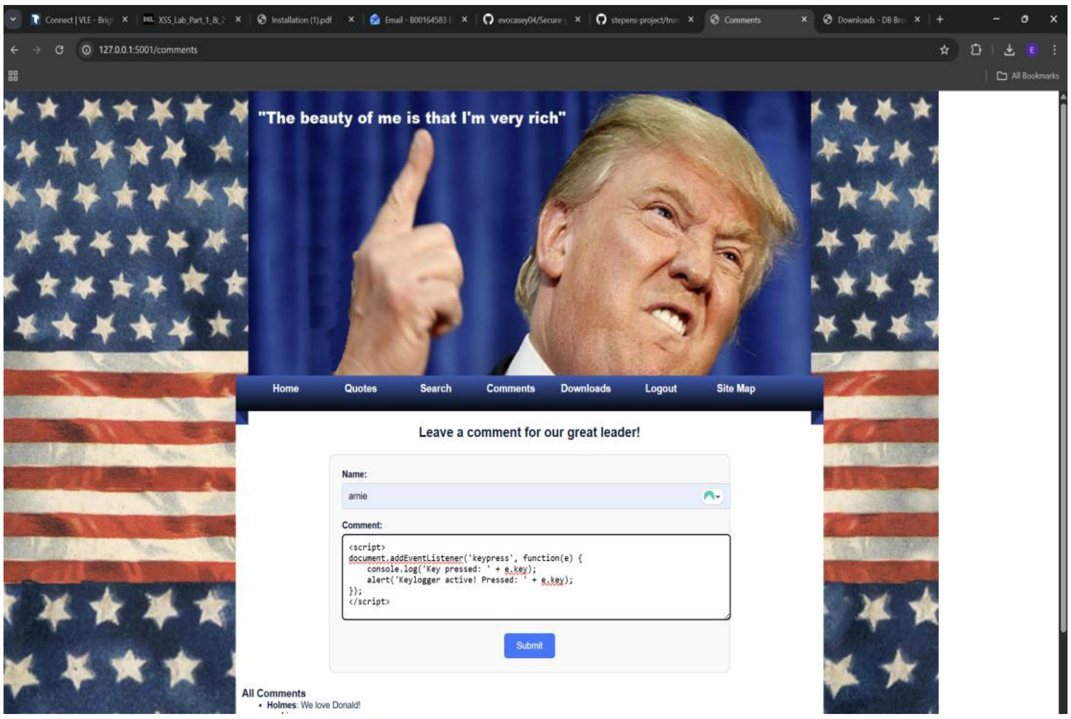
Name:

Comment:

All Comments
- Holmes: We love Donald!
- cookie:
- bob:
- bob:

This one's very interesting as it shows that xss can affect the user by how they are even interacting on the page.

The next xss logs the users' keystrokes


127.0.0.1:5001/comments

Leave a comment for our great leader!

Name: arnie

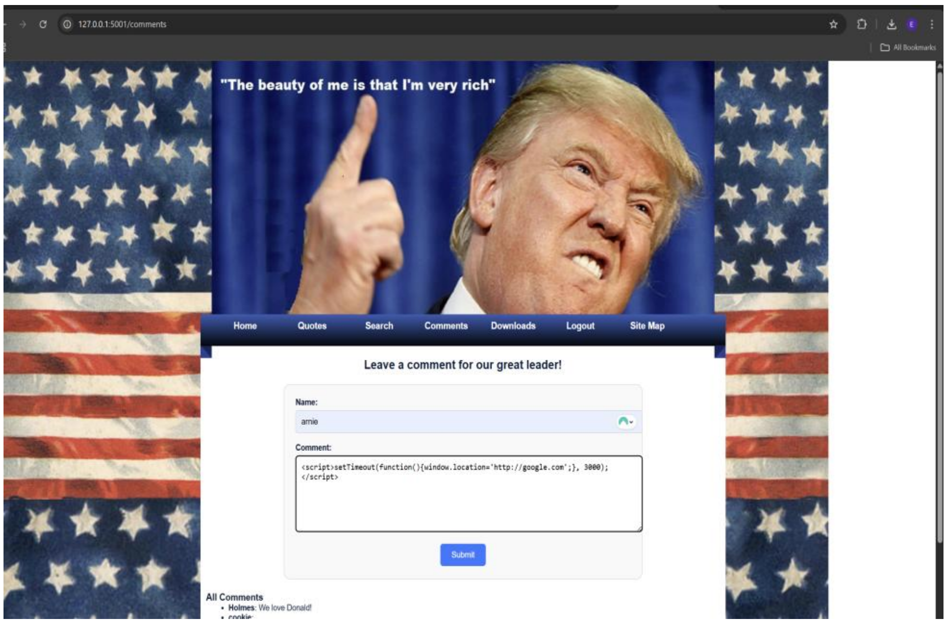
Comment:

```
<script>
document.addEventListener('keypress', function(e) {
    console.log('Key pressed: ' + e.key);
    alert('Keylogger active! Pressed: ' + e.key);
});
</script>
```

All Comments
- Holmes: We love Donald!

Redirect attack using xss

Intention of this is to show an attacker could use it to direct people to malicious sites



Leave a comment for our great leader!

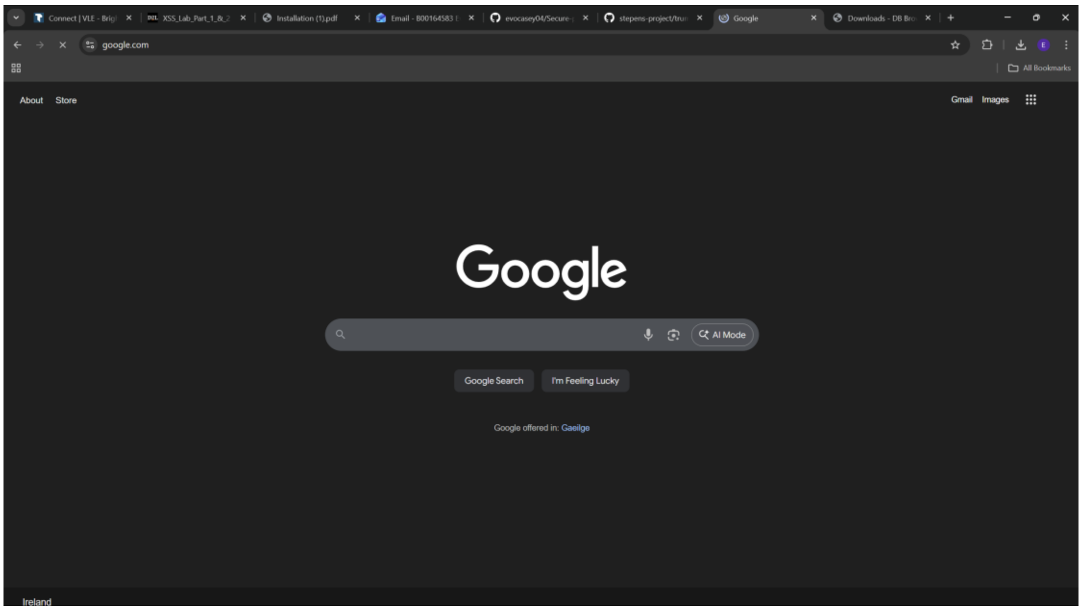
Name: amie

Comment:

```
<script>setTimeout(function(){window.location='http://google.com';}, 3000);
</script>
```

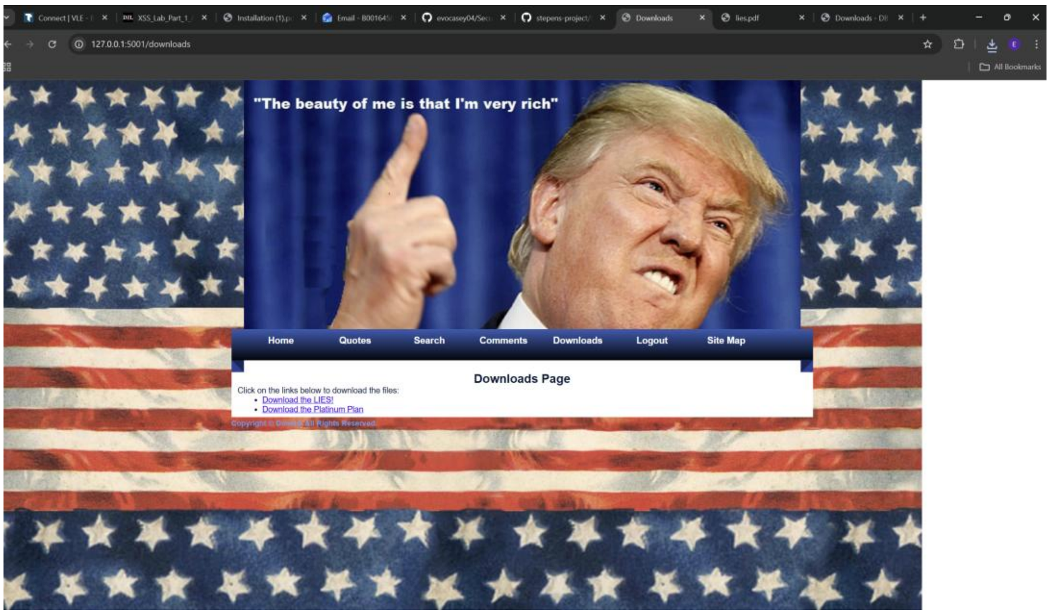
Submit

All Comments
- Holmes: We love Donald!
- cookie:



The script has been set so that after 3 seconds it redirects the user to google just as an example

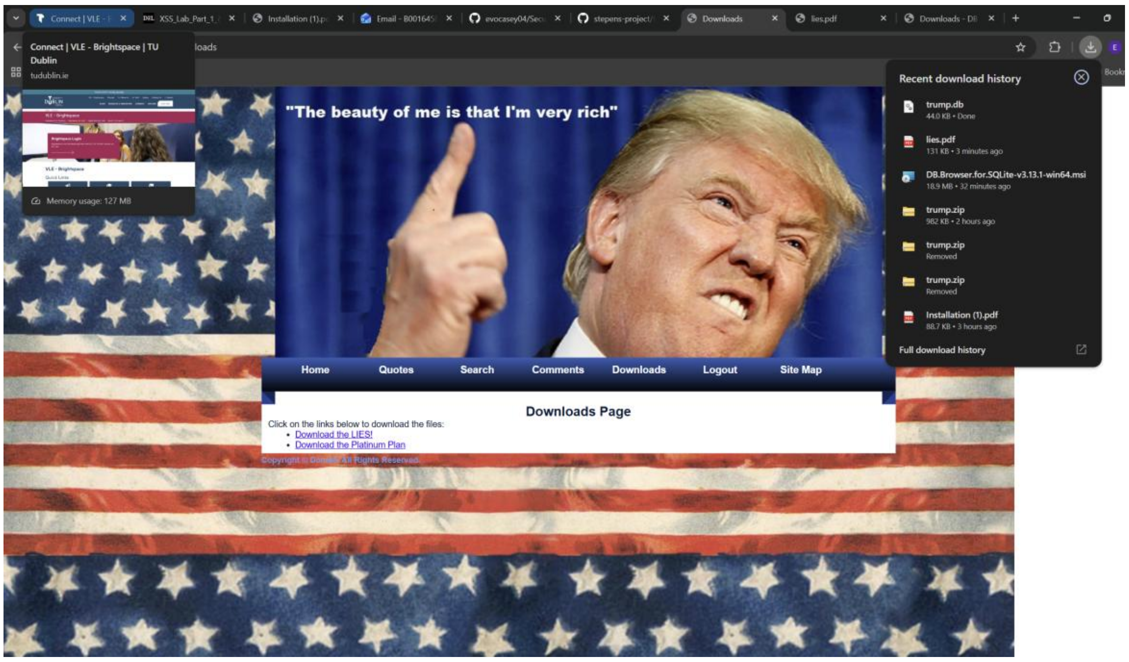
Path traversal exploit



Click on the links below to download the files:

- Download the LIES!
- Download the Platinum Plan

This is the downloads page, and we are going to try and do a path traversal exploit to download the entire database.



Recent download history

- trump.db 44.0 KB • Done
- lies.pdf 131 KB • 3 minutes ago
- DB.Browser.for.SQLite-v3.13.1-win64.msi 18.9 MB • 32 minutes ago
- trump.zip 982 KB • 2 hours ago
- trump.zip Removed
- trump.zip Removed
- Installation (1).pdf 88.7 KB • 3 hours ago

Click on the links below to download the files:

- Download the LIES!
- Download the Platinum Plan

SUCCESS

Using this URL, http://127.0.0.1:5001/download?file=../trump.db we were able to launch a path traversal attack and acquire the entire database for the site.

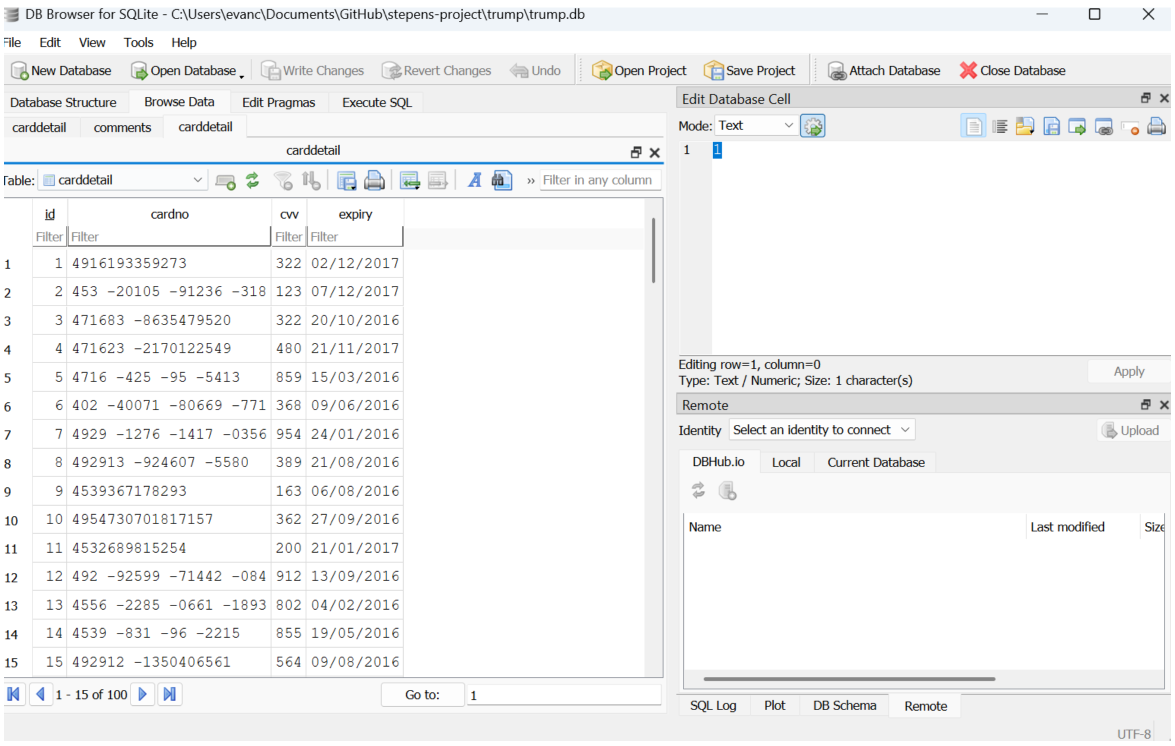

DB Browser for SQLite - C:\Users\evanc\Documents\GitHub\stepens-project\trump\trump.db

Table: carddetail

	id	cardno	cvv	expiry
1	1	4916193359273	322	02/12/2017
2	2	453 -20105 -91236 -318	123	07/12/2017
3	3	471683 -8635479520	322	20/10/2016
4	4	471623 -2170122549	480	21/11/2017
5	5	4716 -425 -95 -5413	859	15/03/2016
6	6	402 -40071 -80669 -771	368	09/06/2016
7	7	4929 -1276 -1417 -0356	954	24/01/2016
8	8	492913 -924607 -5580	389	21/08/2016
9	9	4539367178293	163	06/08/2016
10	10	4954730701817157	362	27/09/2016
11	11	4532689815254	200	21/01/2017
12	12	492 -92599 -71442 -084	912	13/09/2016
13	13	4556 -2285 -0661 -1893	802	04/02/2016
14	14	4539 -831 -96 -2215	855	19/05/2016
15	15	492912 -1350406561	564	09/08/2016

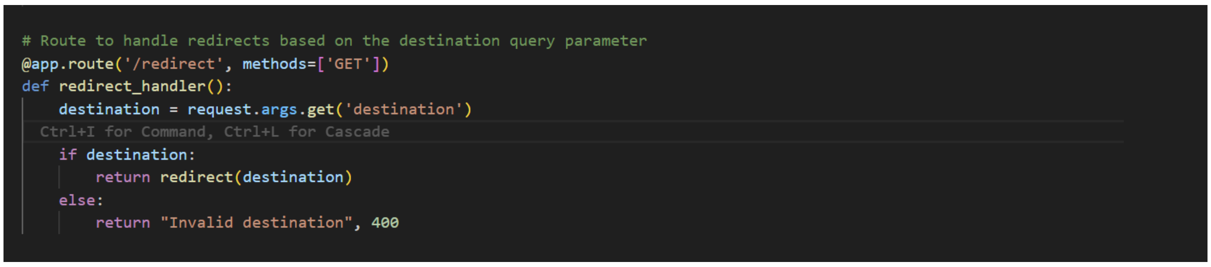
1 - 15 of 100

Editing row=1, column=0
Type: Text / Numeric; Size: 1 character(s)

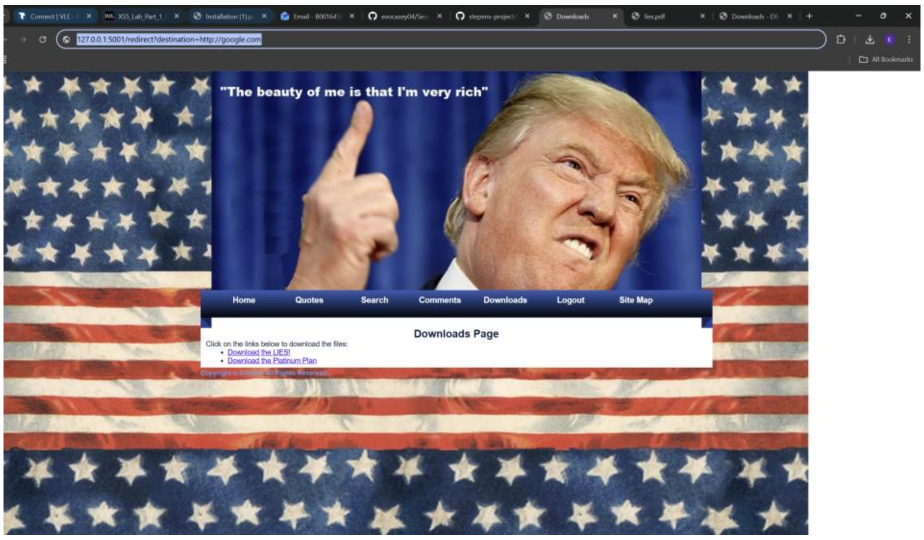
As you can see, we now have access to everyone's card details and more.

Open redirect

This code in app.py shows that the site vulnerable to redirects



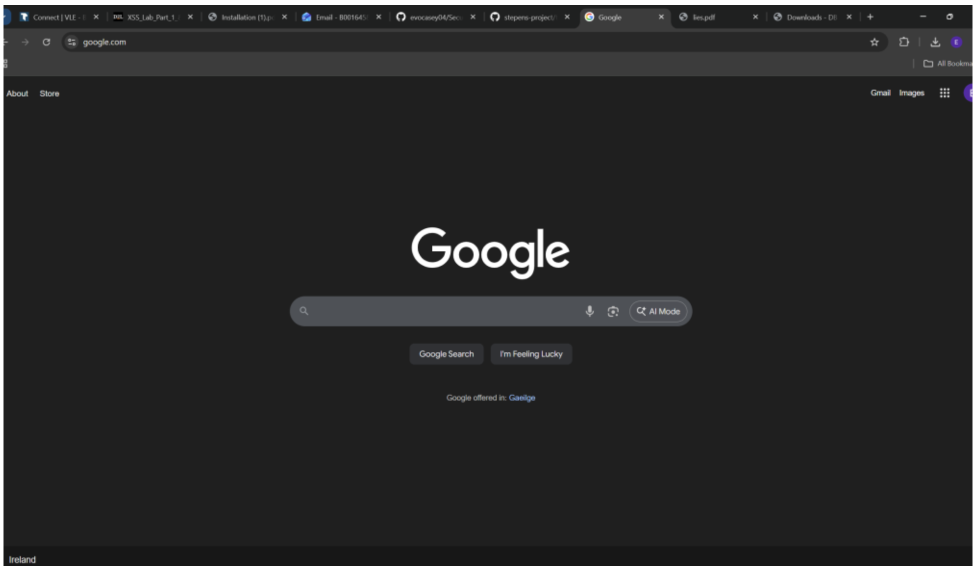
```python
# Route to handle redirects based on the destination query parameter
@app.route('/redirect', methods=['GET'])
def redirect_handler():
    destination = request.args.get('destination')
  Ctrl+I for Command, Ctrl+L for Cascade
    if destination:
        return redirect(destination)
    else:
        return "Invalid destination", 400
```



```
127.0.0.1:5001/redirect?destination=http://google.com
```

Downloads Page

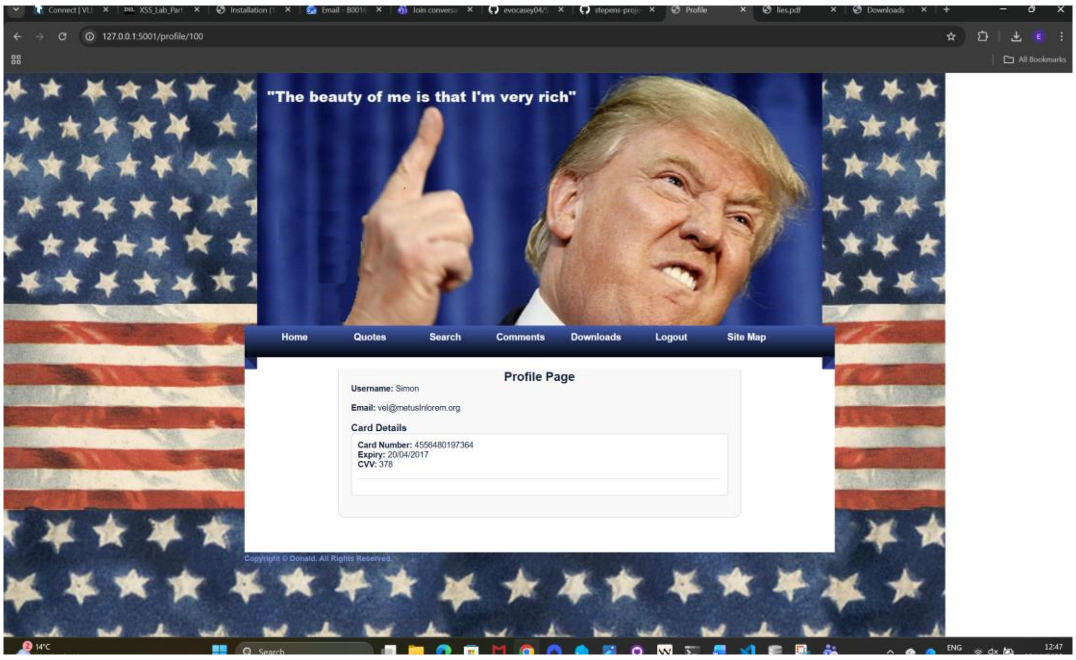
Click on the links below to download the files:
- Download the LIES!
- Download the Platinum Plan



```
google.com
```

IDOR

By systematically using IDOR we were able to get every user’s bank details example here all that had to be done was simply change the URL.


127.0.0.1:5001/profile/100
"The beauty of me is that I'm very rich"
Home
Quotes
Search
Comments
Downloads
Logout
Site Map
Copyright © Donald. All Rights Reserved.

Profile Page

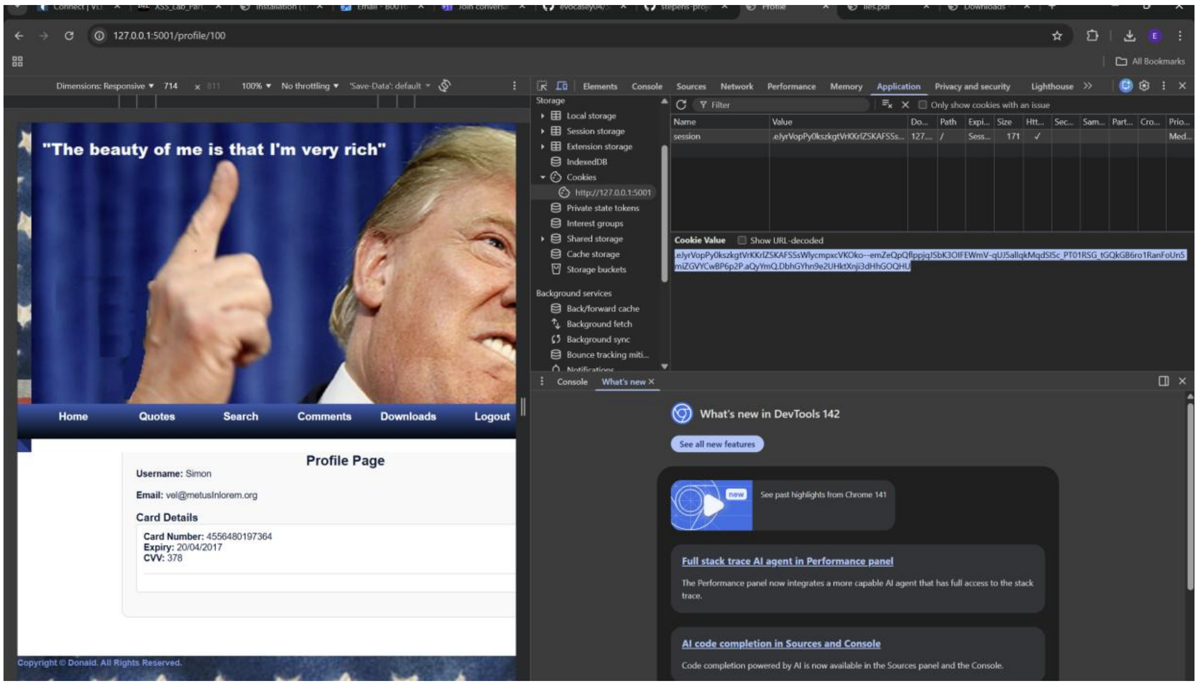
Username: Simon

Email: vel@metusInlorem.org

Card Details

Card Number: 4556480197364
Expiry: 20/04/2017
CVV: 378

Weak secret key


127.0.0.1:5001/profile/100

"The beauty of me is that I'm very rich"

Profile Page

Username: Simon

Email: vel@metusInlorem.org

Card Details

Card Number: 4556480197364
Expiry: 20/04/2017
CVV: 378

Copyright © Donald. All Rights Reserved.

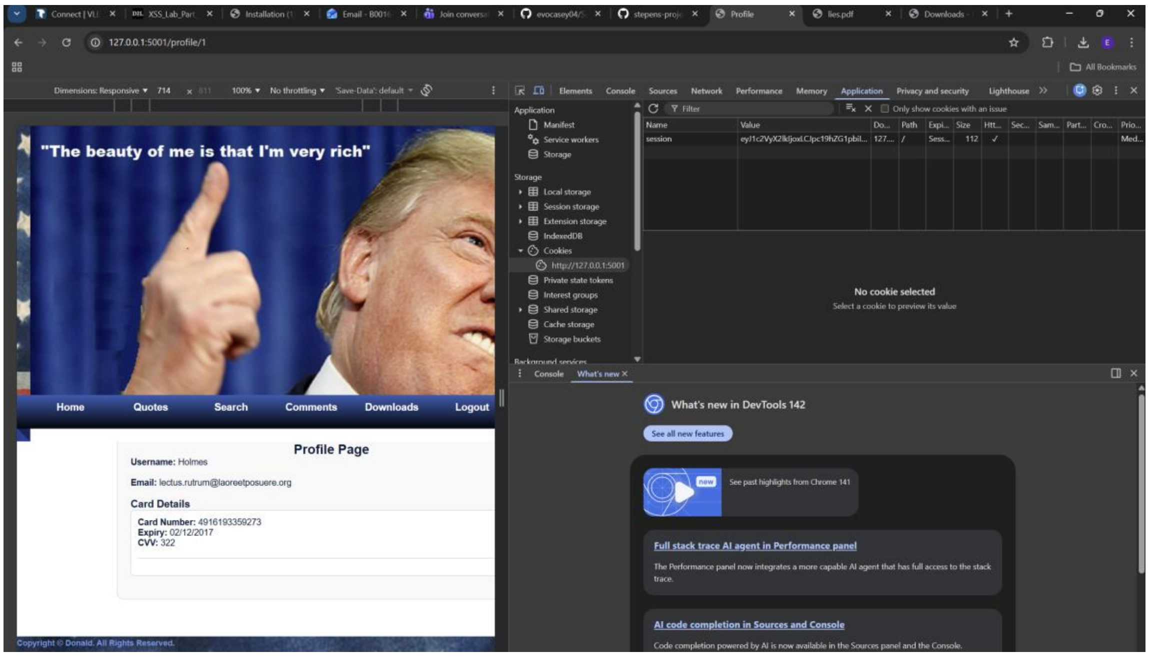
Name	Value	Do...	Path	Expi...	Size	Htt...	Sec...	Sam...	Part...	Cro...	Prio...
session	.eJyrVopPy0kszkgtVrKKrlZSKAFSSs...	127....	/	Sess...	171	✓					Med...

Cookie Value

```
.eJyrVopPy0kszkgtVrKKrlZSKAFSSsWlycmpxcVKOko--emZeQpQflppjqJSbK3OIFEWmV-qUJ5allqkMqdSISc_PT01RSG_tGQkG86ro1RanFoUn5
miZGVYCwBP6p2P.aQyYmQ.DbhGYhn9e2UHktXnji3dHhGOQHU
```

We took the cookie for the session.

And changed it we were able to decode them due to the weak secret key which is hard coded trump123.


127.0.0.1:5001/profile/1

"The beauty of me is that I'm very rich"

Profile Page

Username: Holmes

Email: lectus.rutrum@laoreetposuere.org

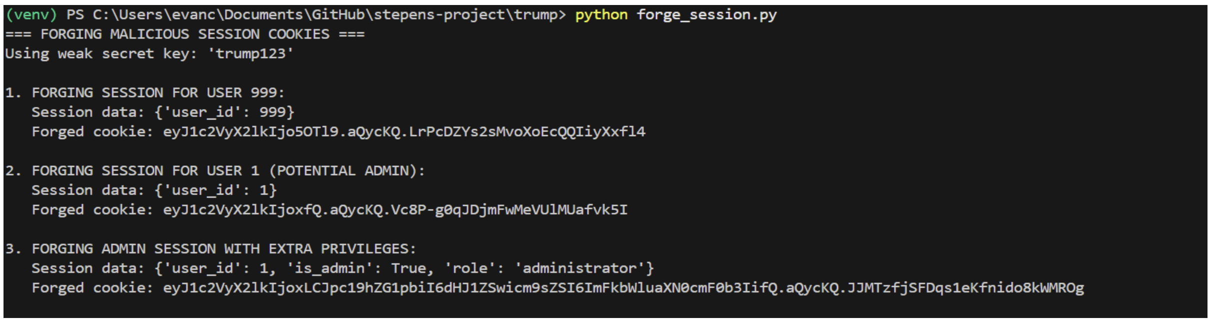
Card Details

Card Number: 4916193359273
Expiry: 02/12/2017
CVV: 322

Copyright © Donald. All Rights Reserved.

Name	Value	Do...	Path	Expi...	Size	Htt...	Sec...	Sam...	Part...	Cro...	Prio...
session	eyJ1c2VyX2lkIjoxLCJpc19hZG1pbil...	127....	/	Sess...	112	✓					Med...

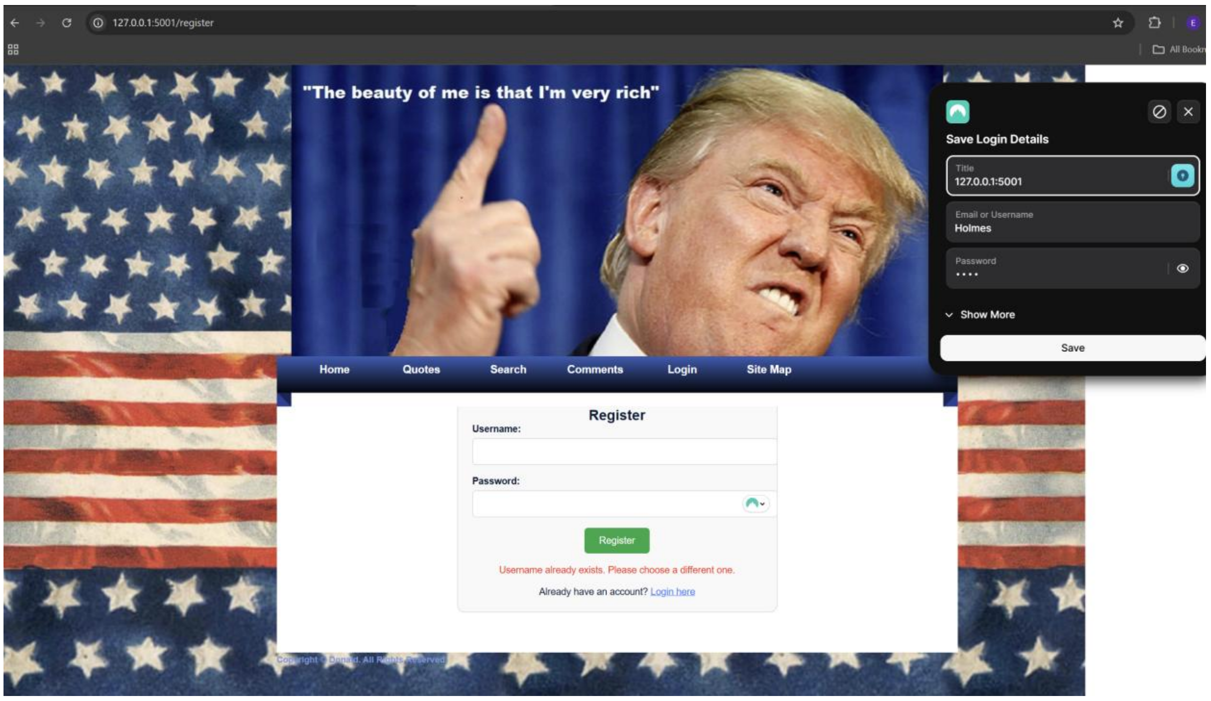
No cookie selected
Select a cookie to preview its value



```
(venv) PS C:\Users\evanc\Documents\GitHub\stepens-project\trump> python forge_session.py
=== FORGING MALICIOUS SESSION COOKIES ===
Using weak secret key: 'trump123'

1. FORGING SESSION FOR USER 999:
   Session data: {'user_id': 999}
   Forged cookie: eyJ1c2VyX2lkIjo5OTl9.aQycKQ.LrPcDZYs2sMvoXoEcQQIiyXxfl4

2. FORGING SESSION FOR USER 1 (POTENTIAL ADMIN):
   Session data: {'user_id': 1}
   Forged cookie: eyJ1c2VyX2lkIjoxfQ.aQycKQ.Vc8P-g0qJDjmFwMeVUlMUafvk5I

3. FORGING ADMIN SESSION WITH EXTRA PRIVILEGES:
   Session data: {'user_id': 1, 'is_admin': True, 'role': 'administrator'}
   Forged cookie: eyJ1c2VyX2lkIjoxLCJpc19hZG1pbiI6dHJ1ZSwicm9sZSI6ImFkbWluaXN0cmF0b3IifQ.aQycKQ.JJMTzfjSFDqs1eKfnido8kWMROg
```

Username Enumeration

This vulnerability shows that the username exists which allows for attackers to potentially do recon on who uses the site and then try to potentially brute force attack the password



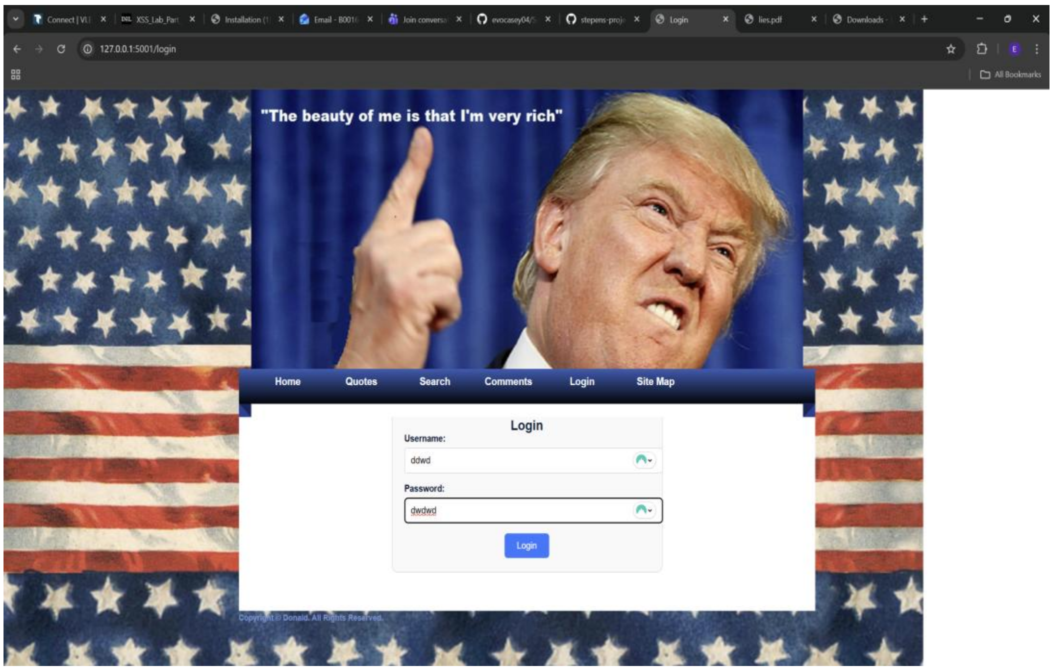
Title: 127.0.0.1:5001
Email or Username: Holmes
Password: ••••

Register
Username:
Password:

Username already exists. Please choose a different one.

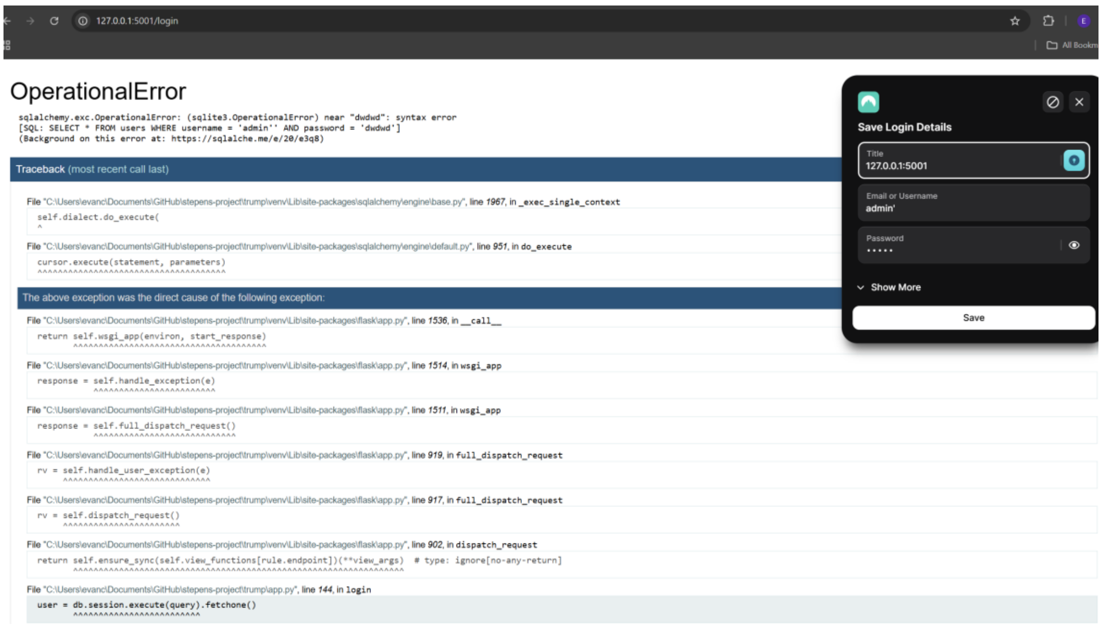
Already have an account? Login here

Password vulnerability



Password is displayed in plaintext which could cause risks with people shoulder surfing

Vulnerable code locations

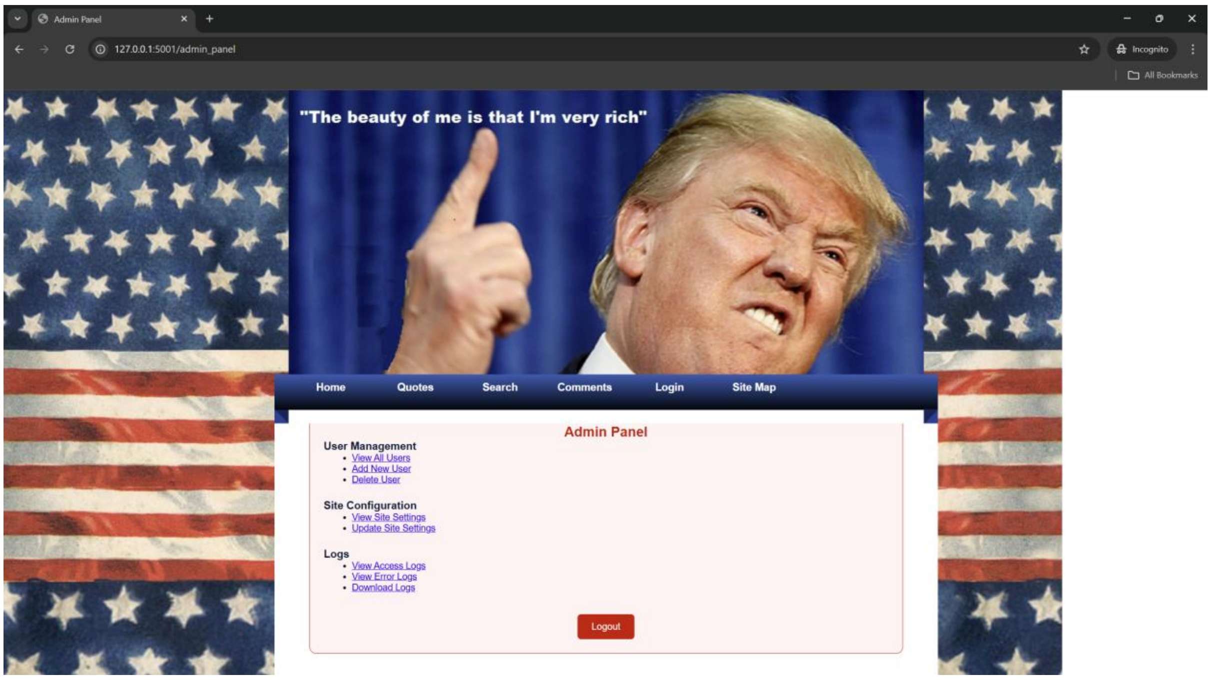


```
OperationalError

sqlalchemy.exc.OperationalError: (sqlite3.OperationalError) near "dwdwd": syntax error
[SQL: SELECT * FROM users WHERE username = 'admin'' AND password = 'dwdwd']
(Background on this error at: https://sqlalche.me/e/20/e3q8)

Traceback (most recent call last)

File "C:\Users\evanc\Documents\GitHub\stepens-project\trump\venv\Lib\site-packages\sqlalchemy\engine\base.py", line 1967, in _exec_single_context
  self.dialect.do_execute(
  ^
File "C:\Users\evanc\Documents\GitHub\stepens-project\trump\venv\Lib\site-packages\sqlalchemy\engine\default.py", line 951, in do_execute
  cursor.execute(statement, parameters)
  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

The above exception was the direct cause of the following exception:

File "C:\Users\evanc\Documents\GitHub\stepens-project\trump\venv\Lib\site-packages\flask\app.py", line 1536, in __call__
  return self.wsgi_app(environ, start_response)
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "C:\Users\evanc\Documents\GitHub\stepens-project\trump\venv\Lib\site-packages\flask\app.py", line 1514, in wsgi_app
  response = self.handle_exception(e)
             ^^^^^^^^^^^^^^^^^^^^^^^^
File "C:\Users\evanc\Documents\GitHub\stepens-project\trump\venv\Lib\site-packages\flask\app.py", line 1511, in wsgi_app
  response = self.full_dispatch_request()
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "C:\Users\evanc\Documents\GitHub\stepens-project\trump\venv\Lib\site-packages\flask\app.py", line 919, in full_dispatch_request
  rv = self.handle_user_exception(e)
       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "C:\Users\evanc\Documents\GitHub\stepens-project\trump\venv\Lib\site-packages\flask\app.py", line 917, in full_dispatch_request
  rv = self.dispatch_request()
       ^^^^^^^^^^^^^^^^^^^^^^^
File "C:\Users\evanc\Documents\GitHub\stepens-project\trump\venv\Lib\site-packages\flask\app.py", line 902, in dispatch_request
  return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)  # type: ignore[no-any-return]
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "C:\Users\evanc\Documents\GitHub\stepens-project\trump\app.py", line 144, in login
  user = db.session.execute(query).fetchone()
         ^^^^^^^^^^^^^^^^^^^^^^^^^
```

By logging in as admin’ and causing an error we were able to expose vulnerable code locations if someone wanted to use them.

Weak admin panel

Admin panel requires no password to access at all you must do is change the URL.



Admin Panel

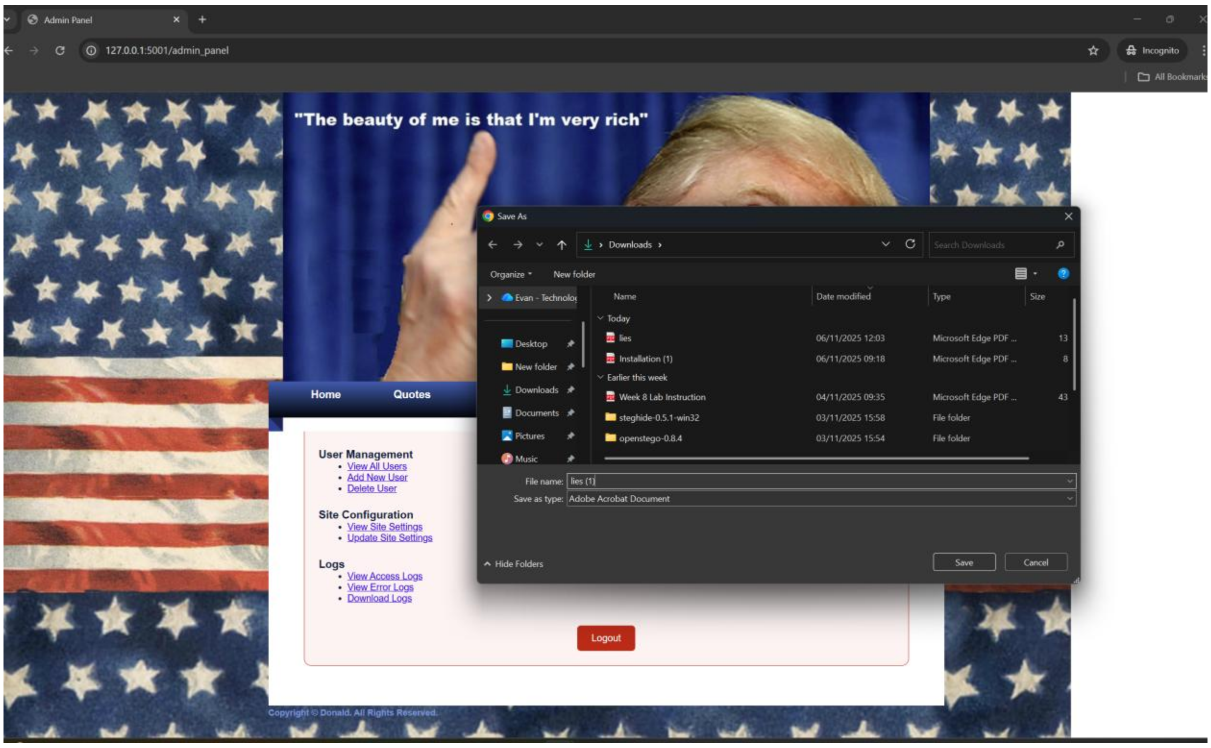
User Management
- View All Users
- Add New User
- Delete User

Site Configuration
- View Site Settings
- Update Site Settings

Logs
- View Access Logs
- View Error Logs
- Download Logs

No authentication on downloads

There is no authentication on the downloads



Name	Date modified	Type	Size
Today			
lies	06/11/2025 12:03	Microsoft Edge PDF ...	13
Installation (1)	06/11/2025 09:18	Microsoft Edge PDF ...	8
Earlier this week			
Week 8 Lab Instruction	04/11/2025 09:35	Microsoft Edge PDF ...	43
steghide-0.5.1-win32	03/11/2025 15:58	File folder	
openstego-0.8.4	03/11/2025 15:54	File folder	

File name: lies (1)

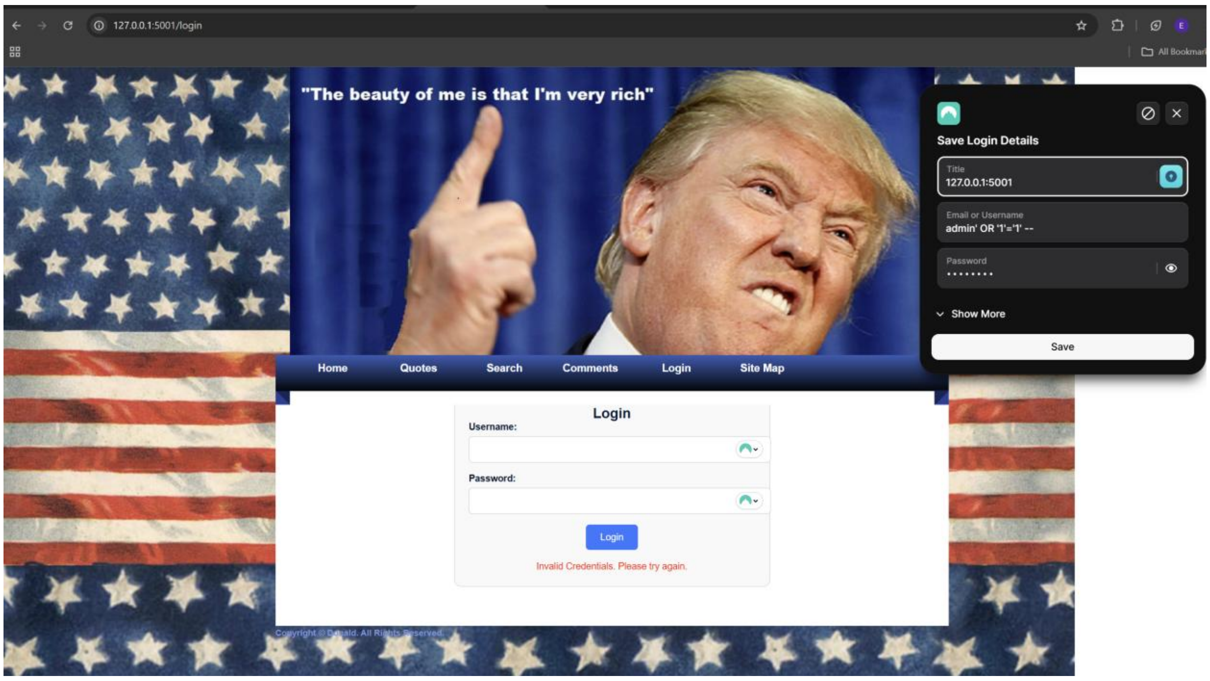
Save as type: Adobe Acrobat Document

Fixing Vulnerabilities

SQL Fix:

We removed the f string and put parameters on the queries so that people can no longer do SQL injects.

query = text("SELECT * FROM users WHERE username = :username AND password = :password")
user = db.session.execute(query, {'username': username, 'password': password}).fetchone()



Save Login Details

Title
127.0.0.1:5001

Email or Username
admin' OR '1'='1' --

Password
••••••••

Login

Username:

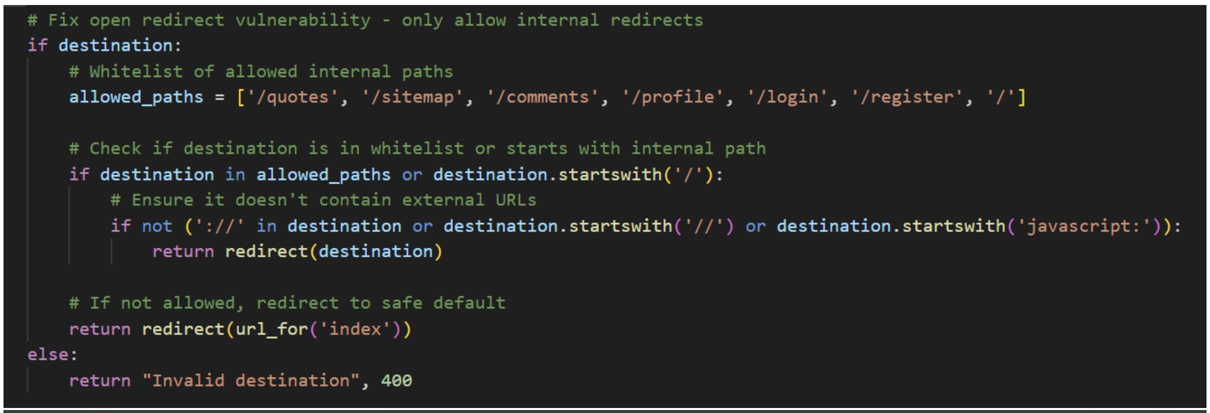
Password:

Login

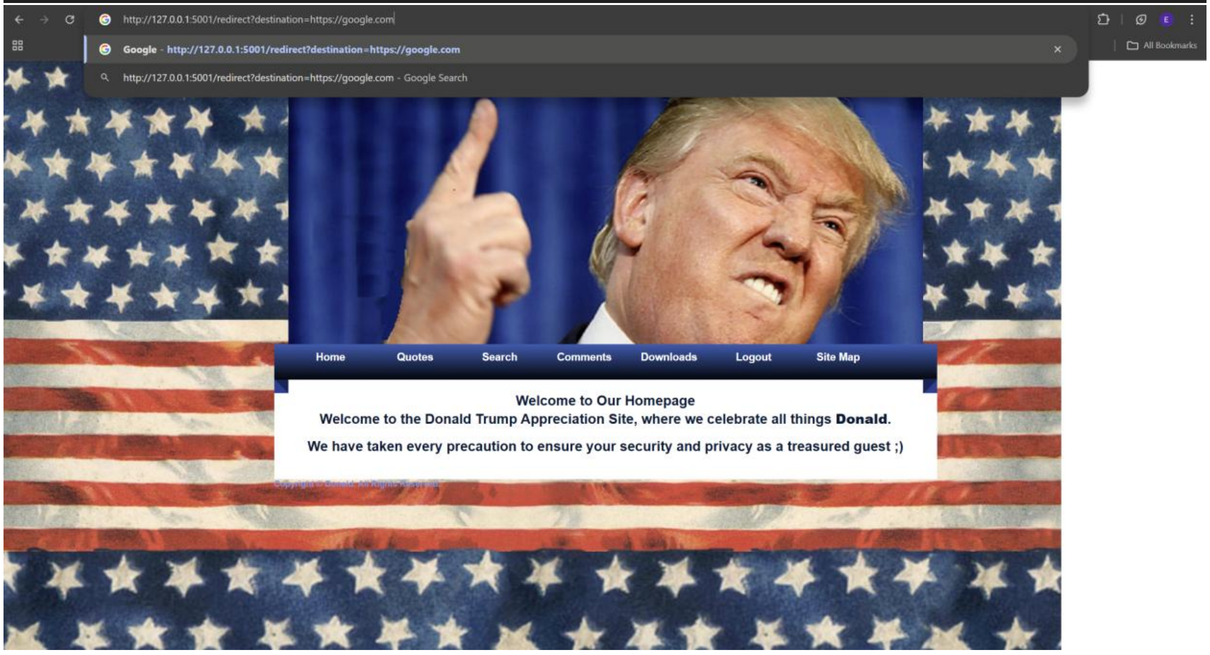
Invalid Credentials. Please try again.

Copyright © Donald. All Rights Reserved.

Open redirect fix



```python
# Fix open redirect vulnerability - only allow internal redirects
if destination:
    # Whitelist of allowed internal paths
    allowed_paths = ['/quotes', '/sitemap', '/comments', '/profile', '/login', '/register', '/']

    # Check if destination is in whitelist or starts with internal path
    if destination in allowed_paths or destination.startswith('/'):
        # Ensure it doesn't contain external URLs
        if not ('://' in destination or destination.startswith('//') or destination.startswith('javascript:')):
            return redirect(destination)

    # If not allowed, redirect to safe default
    return redirect(url_for('index'))
else:
    return "Invalid destination", 400
```



```
http://127.0.0.1:5001/redirect?destination=https://google.com
Google - http://127.0.0.1:5001/redirect?destination=https://google.com
http://127.0.0.1:5001/redirect?destination=https://google.com - Google Search
```

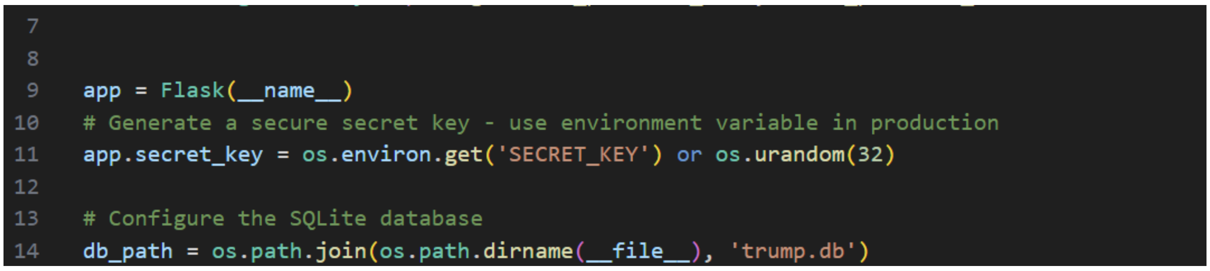
Welcome to Our Homepage
Welcome to the Donald Trump Appreciation Site, where we celebrate all things **Donald**.
We have taken every precaution to ensure your security and privacy as a treasured guest ;)

the redirect no longer works it simply sends the user who is trying the attack to the homepage now.

Secret Key Fix:

We took the hard coded secret key out of the code and updated it so that the key is generated that was if someone managed to get the source code, they can't decode the cookie ID.

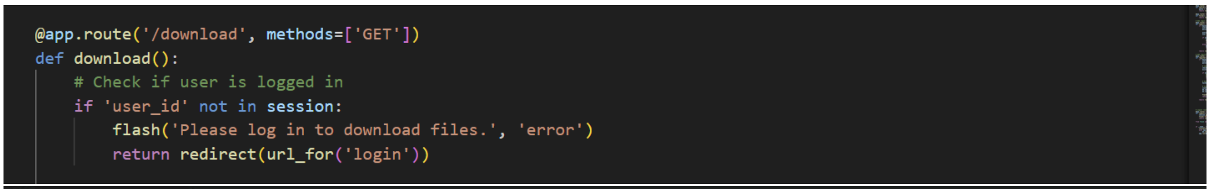
We use os.random to generate a random secret key



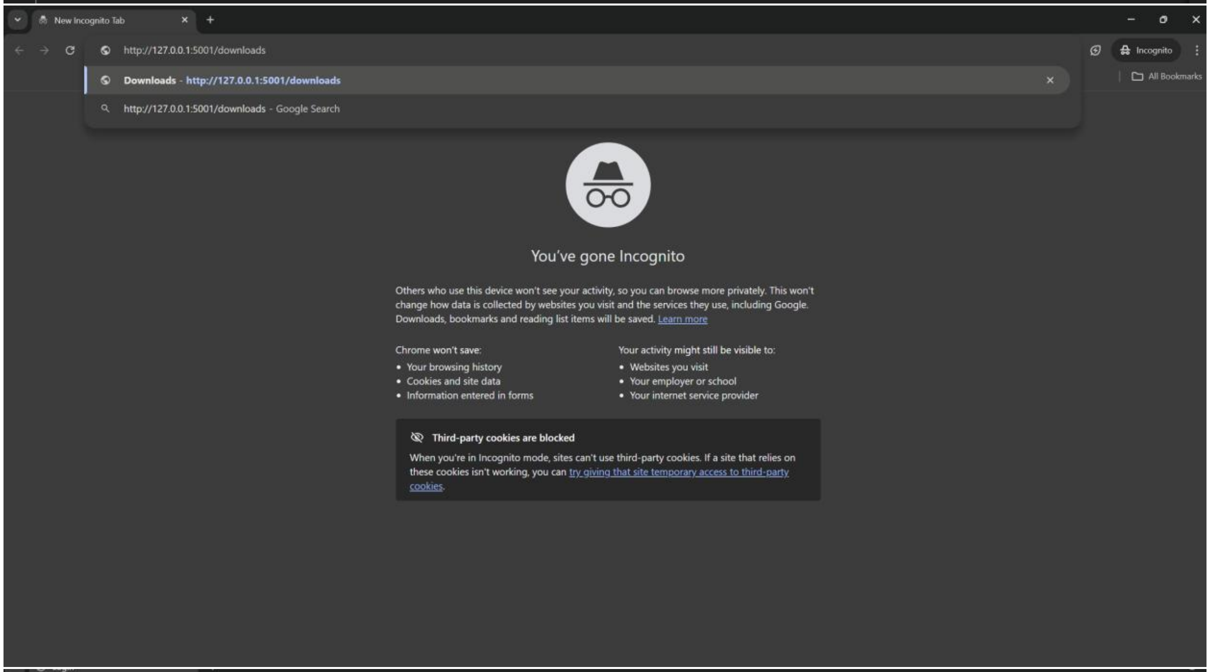
```python


app = Flask(__name__)
# Generate a secure secret key - use environment variable in production
app.secret_key = os.environ.get('SECRET_KEY') or os.urandom(32)

# Configure the SQLite database
db_path = os.path.join(os.path.dirname(__file__), 'trump.db')
```

No Authentication on Downloads fix

By doing this if the user isn't logged in and attempts to download it should redirect them



```
@app.route('/download', methods=['GET'])
def download():
    # Check if user is logged in
    if 'user_id' not in session:
        flash('Please log in to download files.', 'error')
        return redirect(url_for('login'))
```



New Incognito Tab

http://127.0.0.1:5001/downloads

Downloads - http://127.0.0.1:5001/downloads

http://127.0.0.1:5001/downloads - Google Search

You've gone Incognito

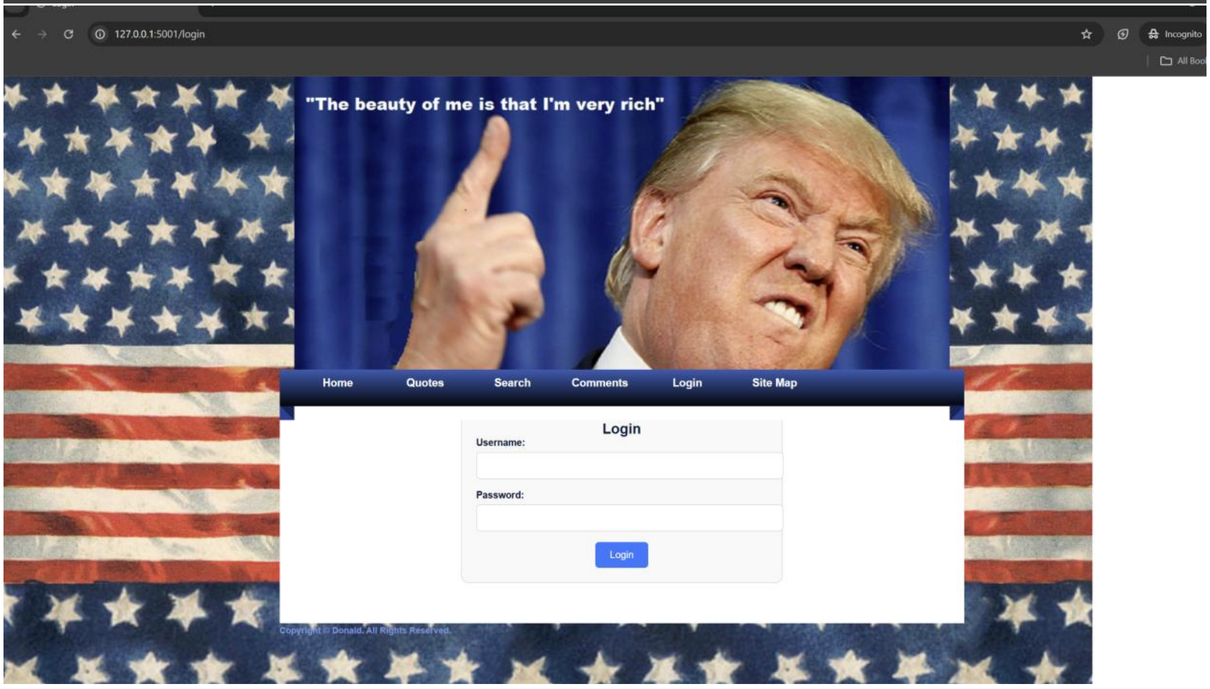
Others who use this device won't see your activity, so you can browse more privately. This won't change how data is collected by websites you visit and the services they use, including Google. Downloads, bookmarks and reading list items will be saved. Learn more

Chrome won't save:
- Your browsing history
- Cookies and site data
- Information entered in forms

Your activity might still be visible to:
- Websites you visit
- Your employer or school
- Your internet service provider

Third-party cookies are blocked

When you're in Incognito mode, sites can't use third-party cookies. If a site that relies on these cookies isn't working, you can try giving that site temporary access to third-party cookies.



127.0.0.1:5001/login

"The beauty of me is that I'm very rich"
Home
Quotes
Search
Comments
Login
Site Map

Login

Username:

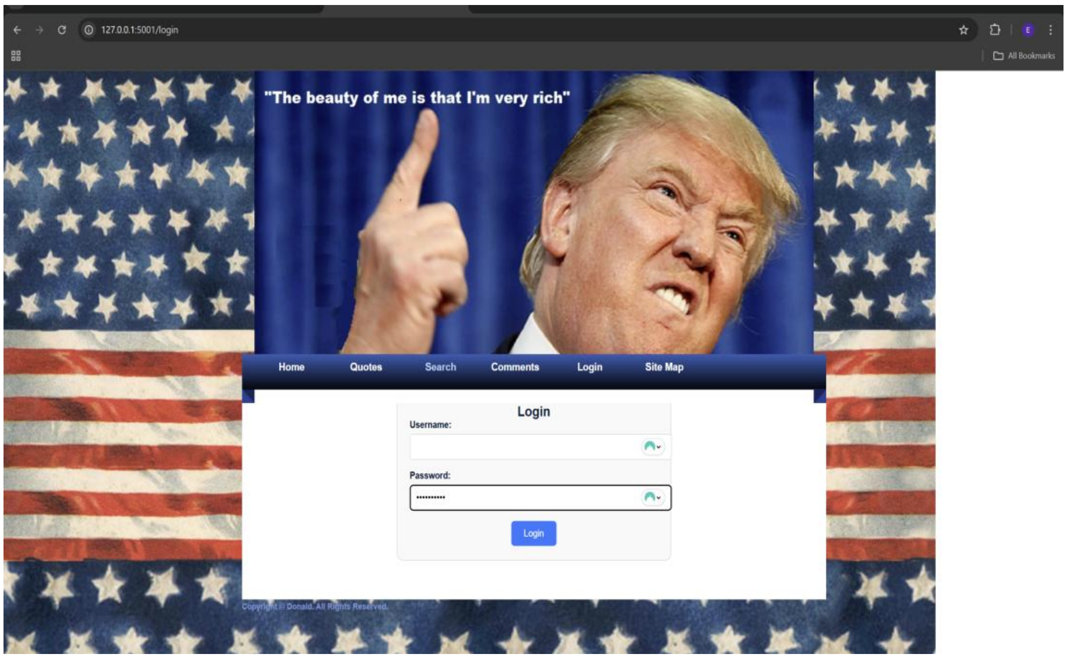
Password:

Login
Copyright © Donald. All Rights Reserved.

redirects to login page

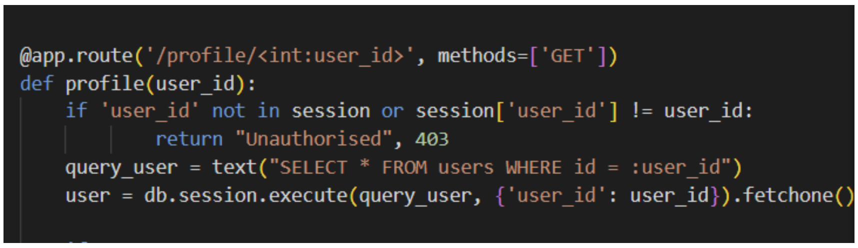
Plain Text Password Fix:

We used a built-in flask function called werkzeug.security to make sure it's not in plaintext.



Data Extraction Fix:

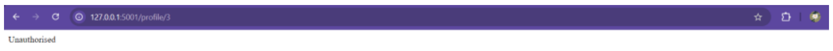
```
@app.route('/profile/<int:user_id>', methods=['GET'])

def profile(user_id):

  if 'user_id' not in session or session['user_id'] != user_id:

      return "Unauthorised", 403
```



```
@app.route('/profile/<int:user_id>', methods=['GET'])
def profile(user_id):
    if 'user_id' not in session or session['user_id'] != user_id:
            return "Unauthorised", 403
    query_user = text("SELECT * FROM users WHERE id = :user_id")
    user = db.session.execute(query_user, {'user_id': user_id}).fetchone()
```

Initially the /profile/<id> route allowed anyone to change the URL like we presented when we demonstrated the vulnerability, we implemented an authentication check into our code so now it checks if a user is logged in and if someone changed the URL theyd get an Unauthorised message, as shown below

.

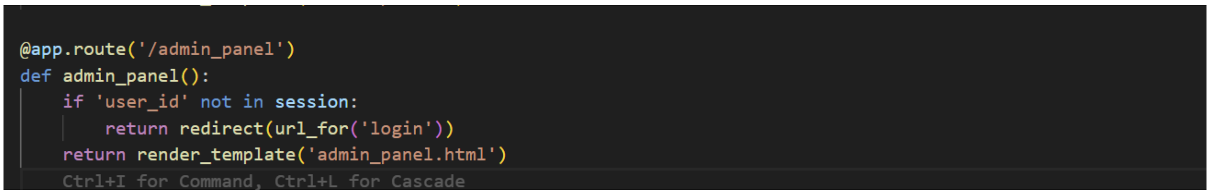


127.0.0.1:5001/profile/3

Unauthorised

Weak Admin Panel fix:

It now checks for user id in session and if there isnt it redirects the user to login page ensuring no anonymous people can access admin panel.



```
@app.route('/admin_panel')
def admin_panel():
    if 'user_id' not in session:
        return redirect(url_for('login'))
    return render_template('admin_panel.html')
    Ctrl+I for Command, Ctrl+L for Cascade
```

@app.route('/admin_panel')

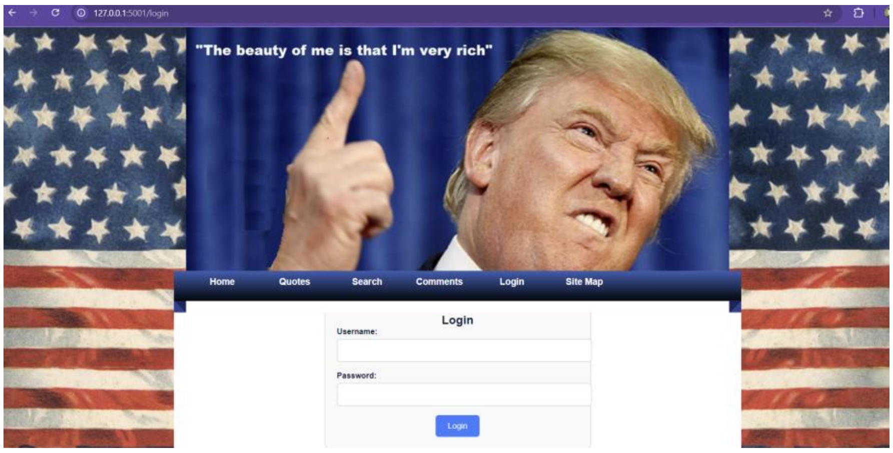
def admin_panel():

  if 'user_id' not in session:

    return redirect(url_for('login'))

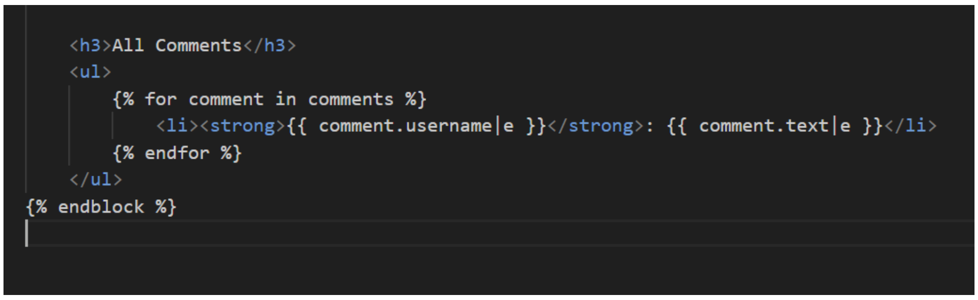
  return render_template('admin_panel.html')

This is what we are now given when we try to get into the admin panel simply by changing the URL



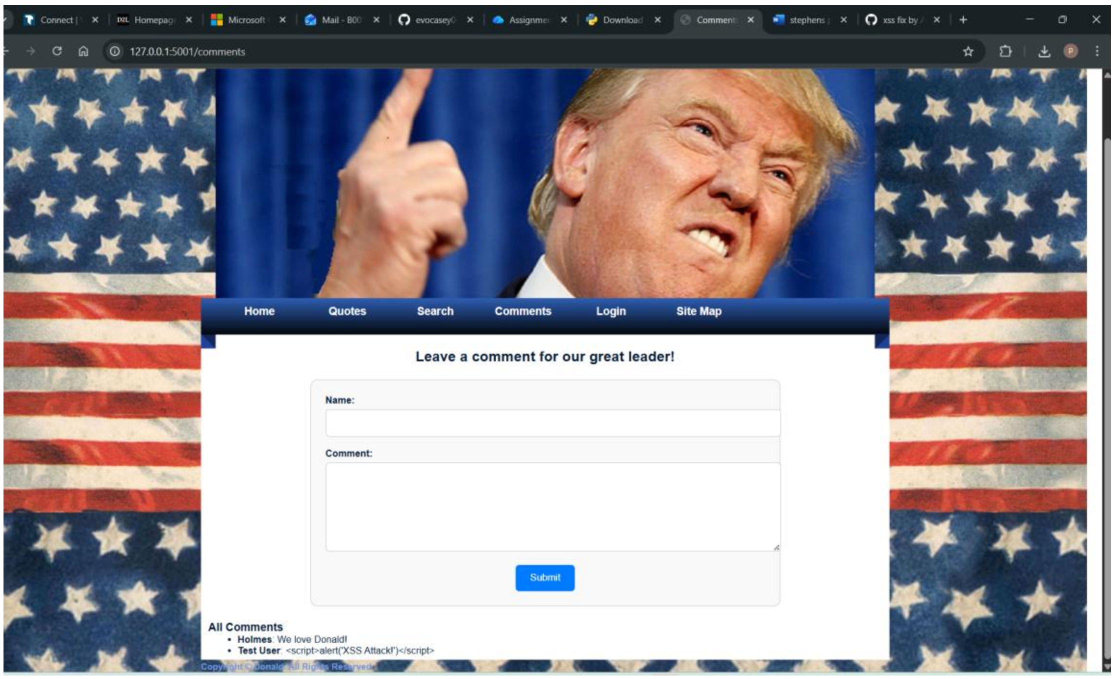
Xss Fix:

For the XSS vulnerability, the fix was simple. If I just take away the word safe from {{ comment.text|safe }}, it won't render the input as raw HTML and whatever is typed will show up as text instead of being executed. The |e converts dangerous characters.



```
    <h3>All Comments</h3>
    <ul>
        {% for comment in comments %}
            <li><strong>{{ comment.username|e }}</strong>: {{ comment.text|e }}</li>
        {% endfor %}
    </ul>
{% endblock %}
```

As you can see in image below comment is shown in plaintext and no popup appears which shows that the site is no longer vulnerable to xss attacks



Leave a comment for our great leader!

Name:

Comment:

All Comments

- **Holmes**: We love Donald!
- **Test User**: <script>alert('XSS Attack!')</script>

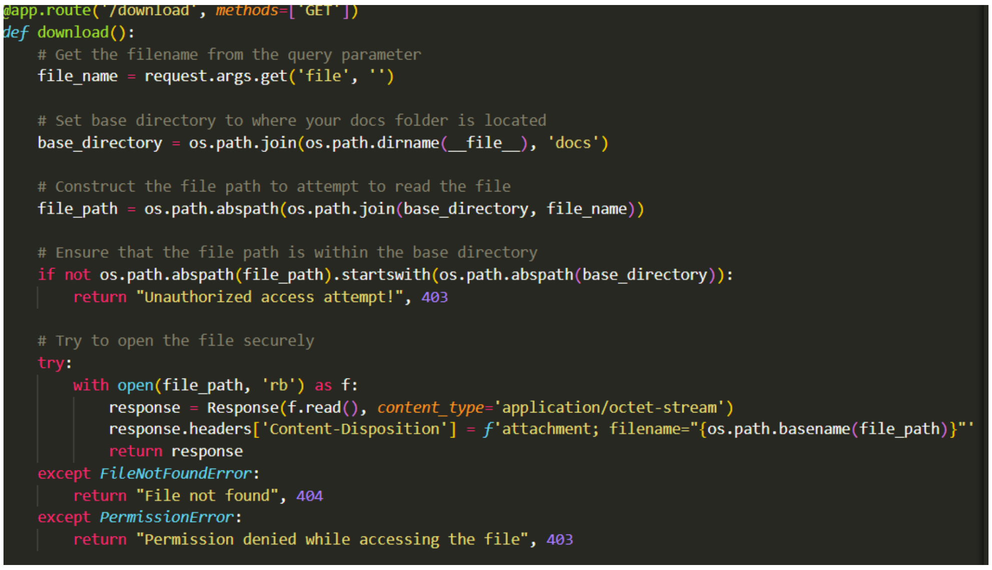
Copyright © Donald. All Rights Reserved.

Path traversal Fix:

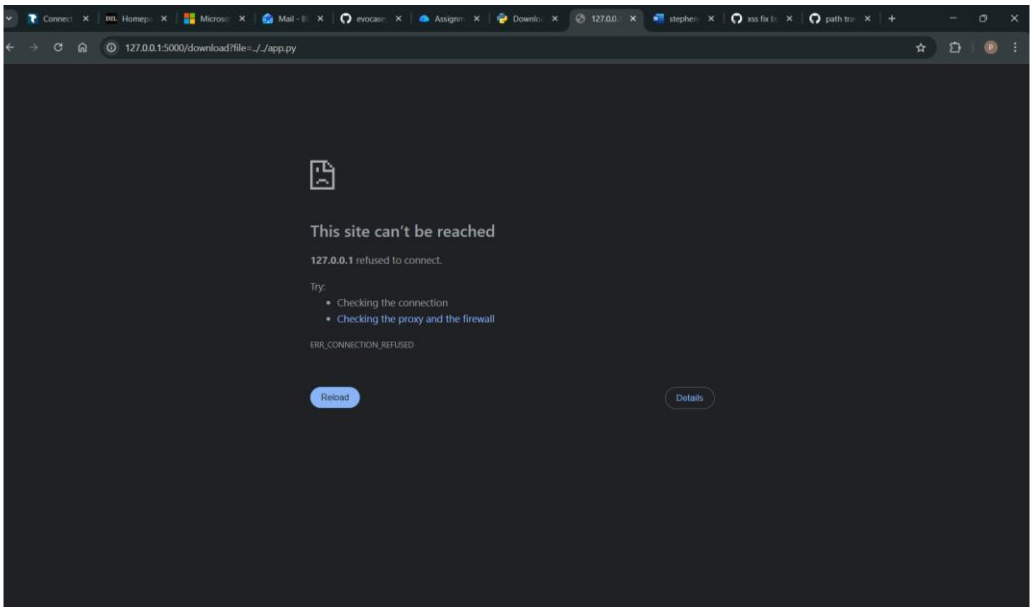
To fix the problem we added 2 lines of code seen below.

```python
# Ensure that the file path is within the base directory
if not os.path.abspath(file_path).startswith(os.path.abspath(base_directory)):
    return "Unauthorized access attempt!", 403
```

This was achieved by uncommenting the code above (it was commented out before) and by adding os.path.abspath() this made it an absolute path and hopefully prevented the attacker from doing ../



```
@app.route('/download', methods=['GET'])
def download():
    # Get the filename from the query parameter
    file_name = request.args.get('file', '')

    # Set base directory to where your docs folder is located
    base_directory = os.path.join(os.path.dirname(__file__), 'docs')

    # Construct the file path to attempt to read the file
    file_path = os.path.abspath(os.path.join(base_directory, file_name))

    # Ensure that the file path is within the base directory
    if not os.path.abspath(file_path).startswith(os.path.abspath(base_directory)):
        return "Unauthorized access attempt!", 403

    # Try to open the file securely
    try:
        with open(file_path, 'rb') as f:
            response = Response(f.read(), content_type='application/octet-stream')
            response.headers['Content-Disposition'] = f'attachment; filename="{os.path.basename(file_path)}"'
            return response
    except FileNotFoundError:
        return "File not found", 404
    except PermissionError:
        return "Permission denied while accessing the file", 403
```



127.0.0.1:5000/download?file=../../app.py

This site can't be reached

127.0.0.1 refused to connect.

Try:

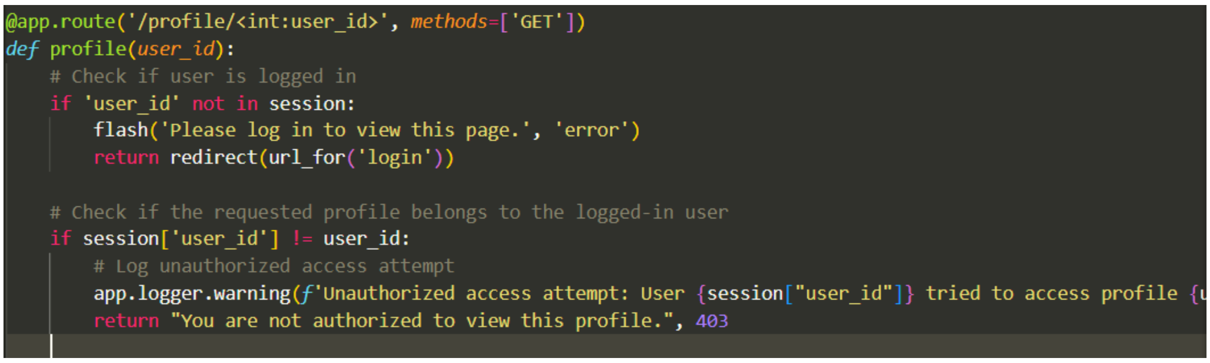
- Checking the connection
- Checking the proxy and the firewall

ERR_CONNECTION_REFUSED

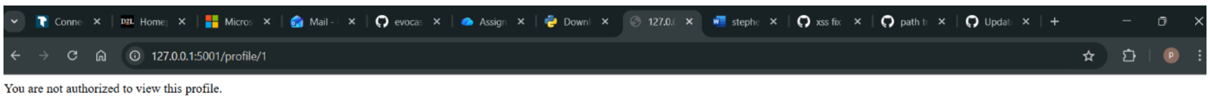
IDOR Fix:

The IDOR vulnerability was important as the endpoint response would expose another's profile if the user id value was manipulated in the respective GET request. Furthermore, the fact that it exposed usernames and credit card information due to not checking if the other user was authorized to view the information was poor because it could expose user usernames and other sensitive information like credit card info.

The IDOR fix was to integrate the following code. This checks for authentication meaning that if a user is not logged into his account, the user will be prompted to do so. Furthermore, viewing profiles is now limited to one's own, accessing others will log the user in, and appropriate error pop-ups will occur.



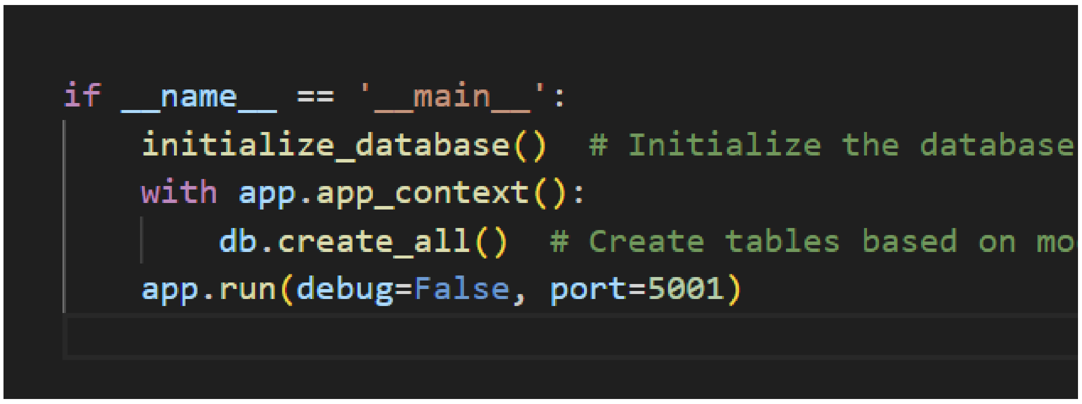
```
@app.route('/profile/<int:user_id>', methods=['GET'])
def profile(user_id):
    # Check if user is logged in
    if 'user_id' not in session:
        flash('Please log in to view this page.', 'error')
        return redirect(url_for('login'))

    # Check if the requested profile belongs to the logged-in user
    if session['user_id'] != user_id:
        # Log unauthorized access attempt
        app.logger.warning(f'Unauthorized access attempt: User {session["user_id"]} tried to access profile {u
        return "You are not authorized to view this profile.", 403
```



You are not authorized to view this profile.

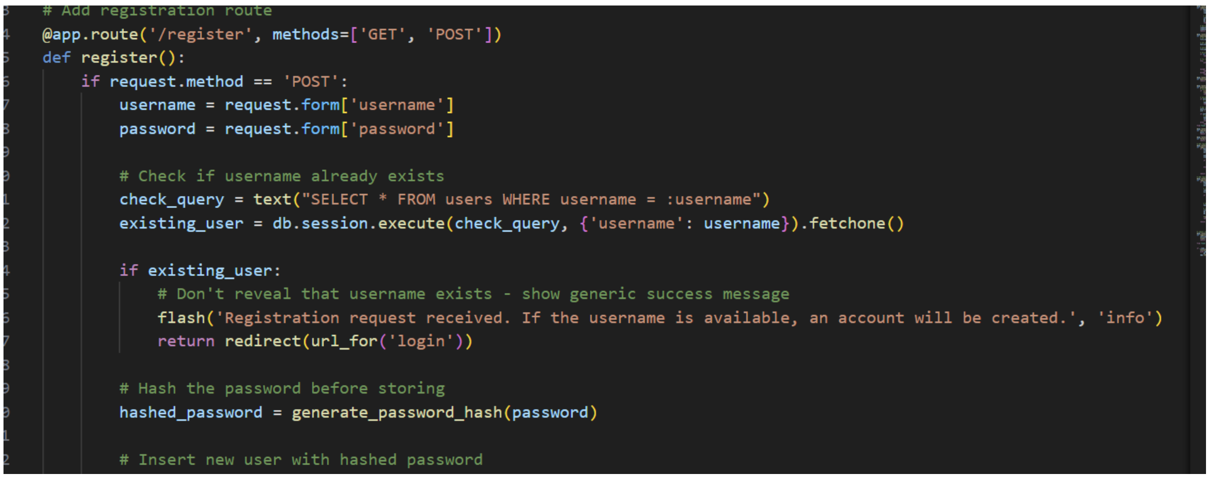
Debug mode fix

The fix for this is simple all we have to do is change it from true to false this eliminates the detailed error messages



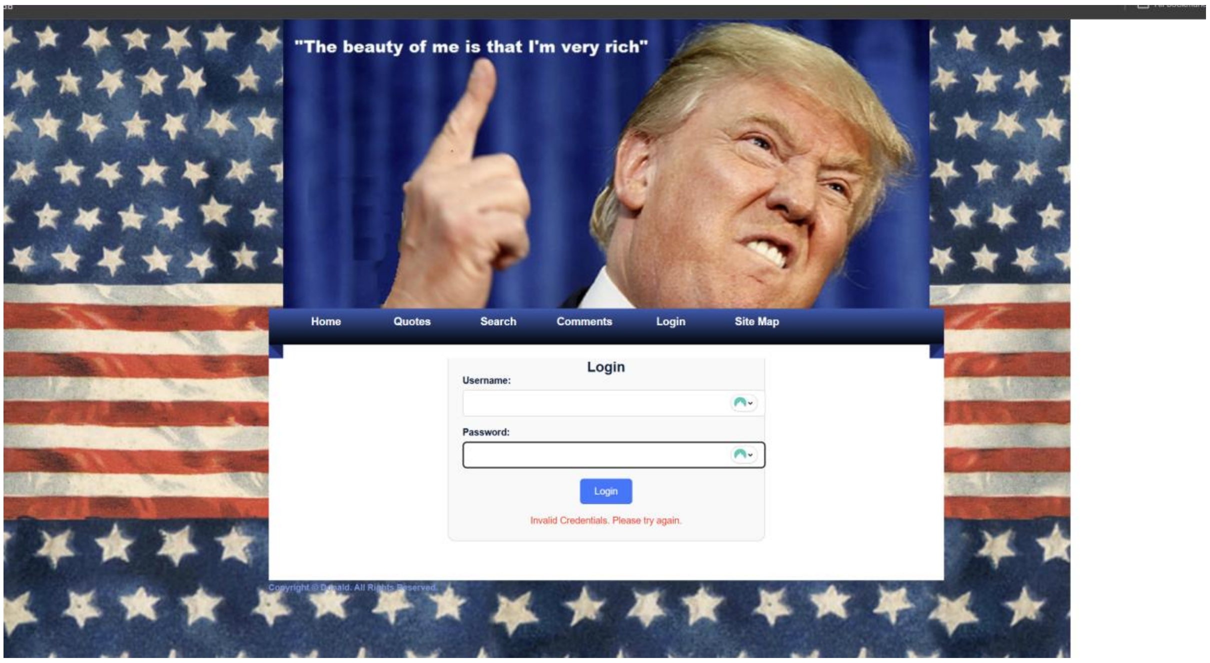
```
if __name__ == '__main__':
    initialize_database()  # Initialize the database
    with app.app_context():
        db.create_all()  # Create tables based on mo
    app.run(debug=False, port=5001)
```

Enumeration Fix



```
# Add registration route
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # Check if username already exists
        check_query = text("SELECT * FROM users WHERE username = :username")
        existing_user = db.session.execute(check_query, {'username': username}).fetchone()

        if existing_user:
            # Don't reveal that username exists - show generic success message
            flash('Registration request received. If the username is available, an account will be created.', 'info')
            return redirect(url_for('login'))

        # Hash the password before storing
        hashed_password = generate_password_hash(password)

        # Insert new user with hashed password
```

As you can see i changed the message that is outputted to be more generic to try and prevent the amount of recon that an attacker may be able to do



Invalid Credentials. Please try again.

Copyright © Donald. All Rights Reserved.

Contribution statement

Evan Paritchamon And Kate all agreed to divide the work evenly between them Evan found 4 exploits Kate and Paritchamon both found 3 initially and then it was left to whoever could find any additional ones they were also asked to fix them if possible if not was passed too another member of the group.

The same process was used for the fixes however this time Kate did 4 fixes Evan and Paritchamon did three fixes. Paritchamon then formatted and prepared the document to ensure it was corrected and helped piece all the work together ensuring an even workload across the group.

Evan found SQL Injection, xss , weak secret key, password vulnerability and username enumerartion

Paritchamon found path traversal, idor ,no authentication on downloadsand weak admin panel

Kate found open redirect, data extraction ,vulnerable code locations