Building a Secure Web Application - Detection and Mitigation of Security Vulnerabilities

CSC429 - COMPUTER SECURITY

Student Name	Student ID
Aliyah Aljarallah	443201214
Shoug Alsaleem	443200641
Lamya Alhaqbani	442202278
Waad Alrashidi	442202356

How to start the websit(on mac devices):
Open your terminal and run : python app.py

And then Open your browser and go to https://127.0.0.1:5000

Identified Vulnerabilities and Fixes:

1. SQL Injection

- **What was wrong:**

At first, the login system used raw SQL queries with user input directly inserted into the query. This means someone could type special input to trick the database. For example:

```
sql = text(f"SELECT * FROM user WHERE username = '{username}' AND password = '{password}'")
```

- **How we fixed it:**

We switched to using SQLAlchemy's ORM instead of writing raw SQL. This safely handles user input and avoids injection:

```
user = User.query.filter_by(username=username).first()
```

- **How to test it :**

On the login page, enter the following in the username field: `' OR 1=1 -` And type anything in the **password** field(ex:1).

2. Weak Password Storage

- **What was wrong:**

At first, passwords were saved in plain text.

```
sql = text(f"INSERT INTO user (username, password) VALUES ('{username}', '{password}')")
 db.session.execute(sql)
 db.session.commit()
```

- **How we fixed it:**

We used bcrypt to hash the passwords before storing them. Now, even if someone gets the database, the passwords are unreadable:

```
hashed_password = bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode('utf-8')
```

How to test it :

Open the terminal and type :

```
sqlite3 instance/database.db
.tables
SELECT * FROM user;
```

here is an example From the database:

```
11|VulnerableTest|VulnerableTest|
12|SecureTest|$2b$12$Cu1Hq9e79iuz15064HiPTubXdGFX6lcadABL6Pw5CbzRSkSCI/h.u|user
sqlite> .quit
```

3. Cross-Site Scripting (XSS)

- **What was wrong:**

The app displayed user-submitted comments directly in HTML using the safe filter.

In the dashbored.html :
{{ comment|safe }}
In the app.py:

```
comments.append(content)
```

- **How we fixed it:**

We removed the safe filter and sanitized the comment using Python's html.escape():

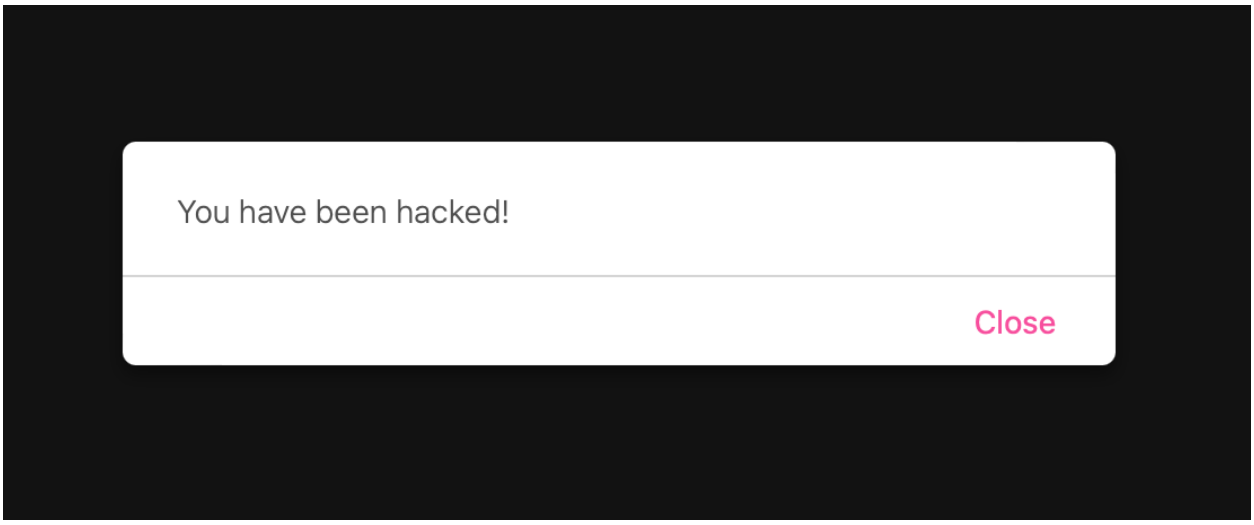
In the dashbored.html :

```
{# <li>{{ comment }}</li> #}
```

In the app.py:

```
sanitized = html.escape(content)
comments.append(sanitized)
```

How to test it :
In the user DashBored comment section type : <script>alert('You have been hacked!')</script>
Here is an example of the JavaScript injection.



4. Access Control

- **What was wrong:**

The `/admin` route was accessible by **any** logged-in user. There was no check to ensure the user had admin privileges.

```
return render_template('admin.html', username=session.get('username'),
role=session.get('role'), users=[])
```

- **How we fixed it:**

We added a role-based check using the session:

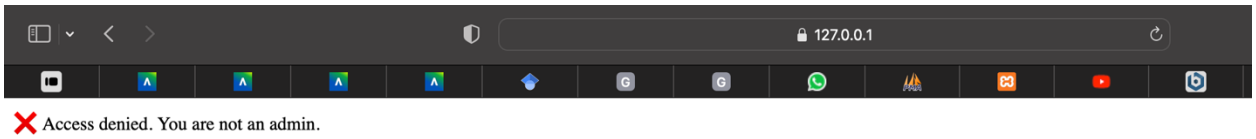
```
if 'username' not in session:
         return redirect(url_for('login'))

     if session.get('role') != 'admin':
         return '❌ Access denied. You are not an admin.', 403

     users = User.query.all()
     return render_template('admin.html', users=users)
```

- How to test it:

  Log in with as a user and manually type /admin.



5. Insecure Communication (Lack of HTTPS)

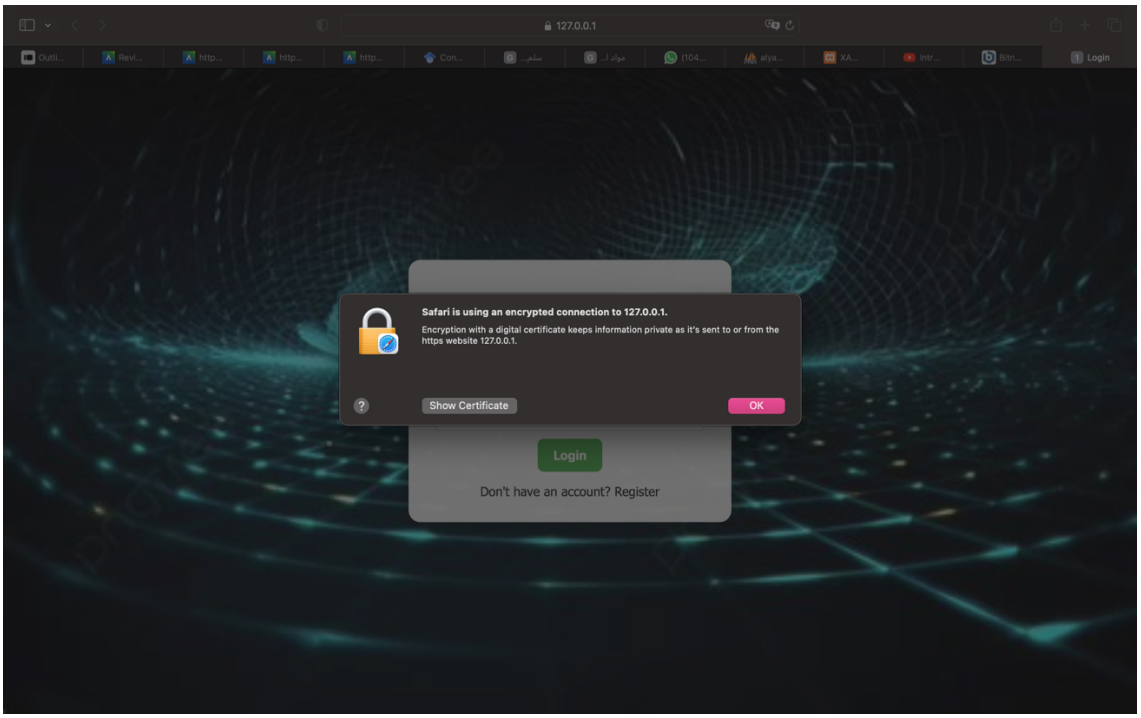
- **What was wrong:**

Initially, the application was only accessible over plain HTTP, which transmits data in clear text

- How we fixed it:

Generated self-signed SSL certificates (cert.pem, key.pem) and ran Flask with HTTPS enabled:

app.run(debug=True, ssl_context=('cert.pem', 'key.pem'))



Challenges Faced in our Project:

Database Not Found / "No Such Table: user" Error → Ensured the database was located inside the instance/ folder.

HTTPS Not Being Activated → Switched to running the app using python app.py instead of flask run, which properly enabled HTTPS and also switched to safari instead of Crome.

Python's Strict Indentation (4-space rule) → We resolved this by ensuring consistent 4-space indentation across all files using proper code editors like VS Code with Python linting enabled.

Resource we Used in this project:

[1] https://youtu.be/qgpsIBLvrGY

[2] https://youtu.be/AzA_LTDoFqY

[3] https://youtu.be/wcaiKgQU6VE

[4] https://youtu.be/EoaDgUgS6QA

[5] OpenAI (chatGPT).