Secure Programming Assignment - 1 Vulnerable Web Application

**Kenert Salm (B00153878)**

**Daniel Mocek (B00152633)**

**Shahenwaz Muzahid (B00155676)**

**Technological University, Blanchardstown**

**Course: Digital Forensics and Cyber Security (TU863)**

**Table Of Contents**

[Task and vulnerabilities 3](#_Toc182969068)

[Section 1: Identifying and Exploiting Vulnerabilities 4](#_Toc182969069)

[(Kenert Salm) 4](#_Toc182969070)

[1.SQL Injection Attacks 4](#_Toc182969071)

[2.Reflected XSS Vulnerability 6](#_Toc182969072)

[3.Stored XSS Vulnerability 7](#_Toc182969073)

[4.Brute-force Authentication Attack 8](#_Toc182969074)

[5.Command Injection 9](#_Toc182969075)

[6.Insecure Direct Object Reference 9](#_Toc182969076)

[7.Broken Access Control 10](#_Toc182969077)

[8.Vulnerable and Outdated Components 11](#_Toc182969078)

[9.Cross-Site Request Forgery 12](#_Toc182969079)

[Section 2: Fixing The Vulnerabilities 13](#_Toc182969080)

[(Daniel Mocek) 13](#_Toc182969081)

[1. SQL Injection 13](#_Toc182969082)

[2. Reflected XSS Vulnerability 14](#_Toc182969083)

[3. The Stored XSS vulnerability 14](#_Toc182969084)

[4.Brute forcing 15](#_Toc182969085)

[5.Command Injection 15](#_Toc182969086)

[6. Insecure objects 16](#_Toc182969087)

[7. Broken access control 16](#_Toc182969088)

[8.Vulnerable and Outdated Components 16](#_Toc182969089)

[9. Cross-site request Forgery 17](#_Toc182969090)

[Section 3: Demonstrating the Fixed Vulnerabilities 17](#_Toc182969091)

[(Shahenwaz Muzahid) 17](#_Toc182969092)

# Task and vulnerabilities

The purpose of this assignment is to secure vulnerable source code in the web application provided and provide a report on how you achieved this. You will be supplied with a simple Web application. The app is coded using Python Flask and connects to an SQLite backend database.

There are 3 parts to the assignment:

* Demonstrate 10 exploits in the code using screenshots and a description on how the application is vulnerable in the source code. We should aim for 5 exploits we have learned in the labs and 5 we have not.
* Fix the 10 exploits in the code. 5 fixes we have learned in the labs and 5 we have not.
* Demonstrate that the 10 exploits are no longer possible (again use screenshots and a description).

The final code must still run properly with the changes and remain as close as possible to the original code.

Marking Rubric goes as follows:

**Identifying and exploiting vulnerabilities (50%):**

• Lab vulnerabilities - 20%

• Non-lab vulnerabilities - 30%

**Fixing the vulnerabilities (50%)**

• Lab vulnerabilities - 20%

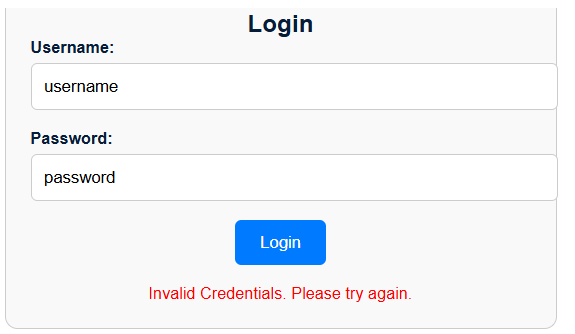
• Non-lab vulnerabilities - 30%

# Section 1: Identifying and Exploiting Vulnerabilities

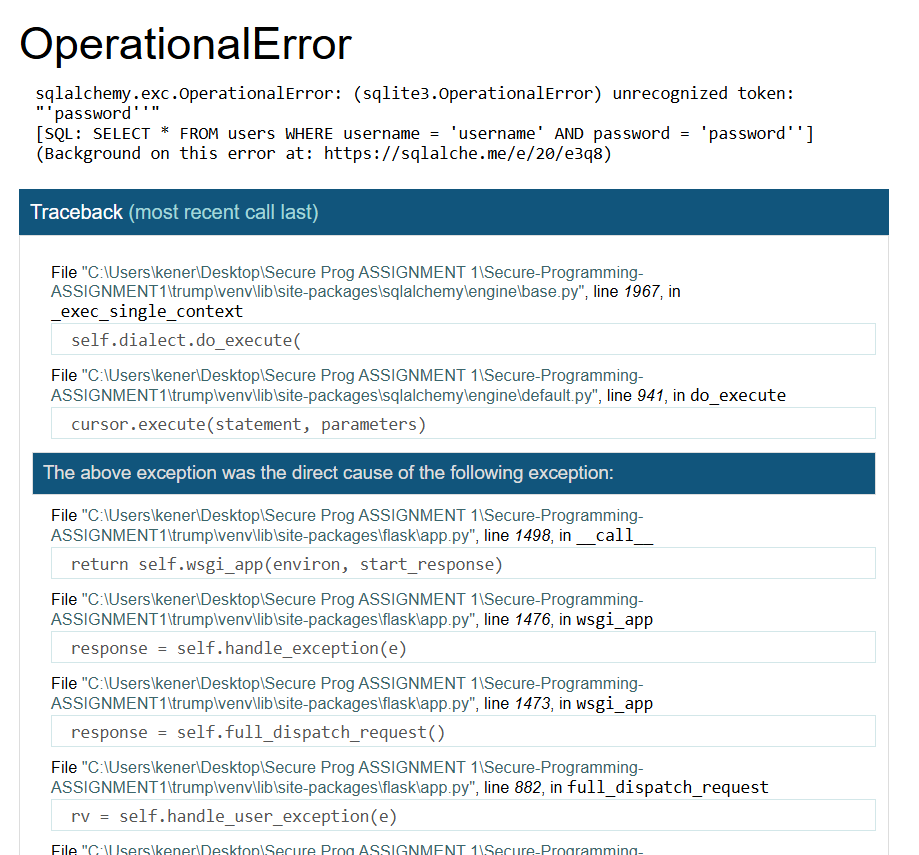
# (Kenert Salm)

## 1.SQL Injection Attacks

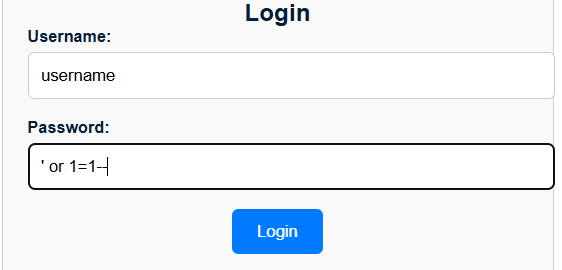
Looking at the Login section first, we can see there are prompts for a username and password. When trying to enter random details it will say its invalid



However, if we add a quote character (‘) to the end of password we get this error.



This means that the quote character is significant to the code that runs the login form and adding one caused the whole process to crash. Exploiting this fact, we can try to write this in the password section:



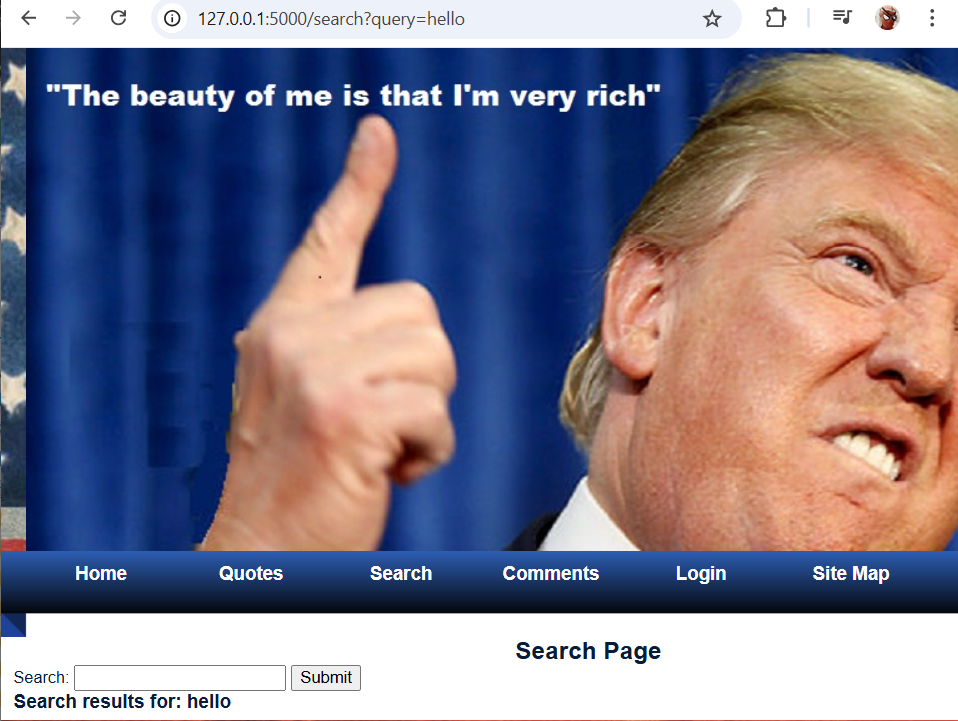
Once we press login, details of a user known as Holmes show up including very sensitive details such as his email and card details.



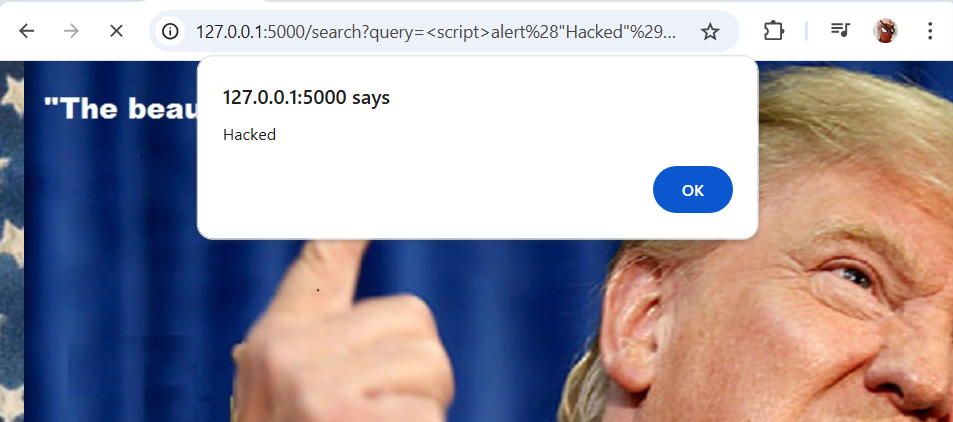
Since 1=1 is always true, the entered password will exploit the database into ignoring any security protocols allowing hackers to bypass any need for a password to get access to details.

## 2.Reflected XSS Vulnerability

When we write anything into the search function of this website, we can see that the entered term is displayed in the URL of the website.



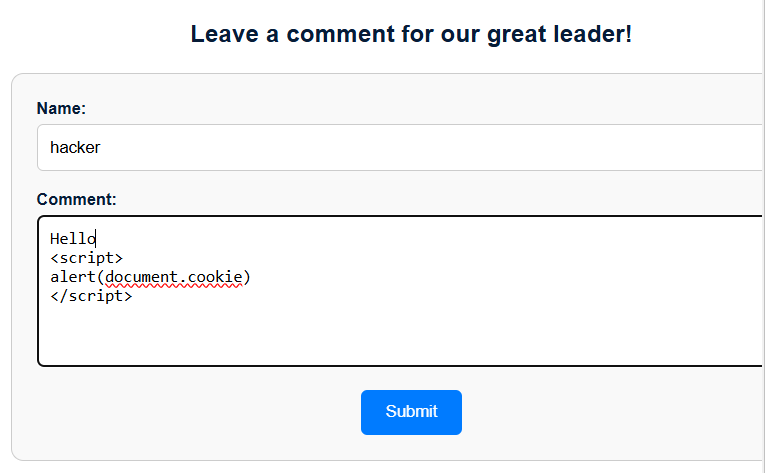
Knowing this, we can make the page display a banner like the following:



Hackers can exploit this vulnerability by redirecting victims and their authentication cookies to their own server and then steal the cookies for impersonation.

## 3.Stored XSS Vulnerability

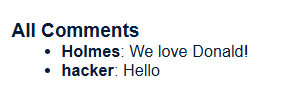
Since users can leave comments on this website, attackers might be able to inject their own JavaScript into the website database. We will write a comment with a basic script like this for example.



Once we submit the comment, whenever anyone visits the comment section on the website, a banner prompt shows up.



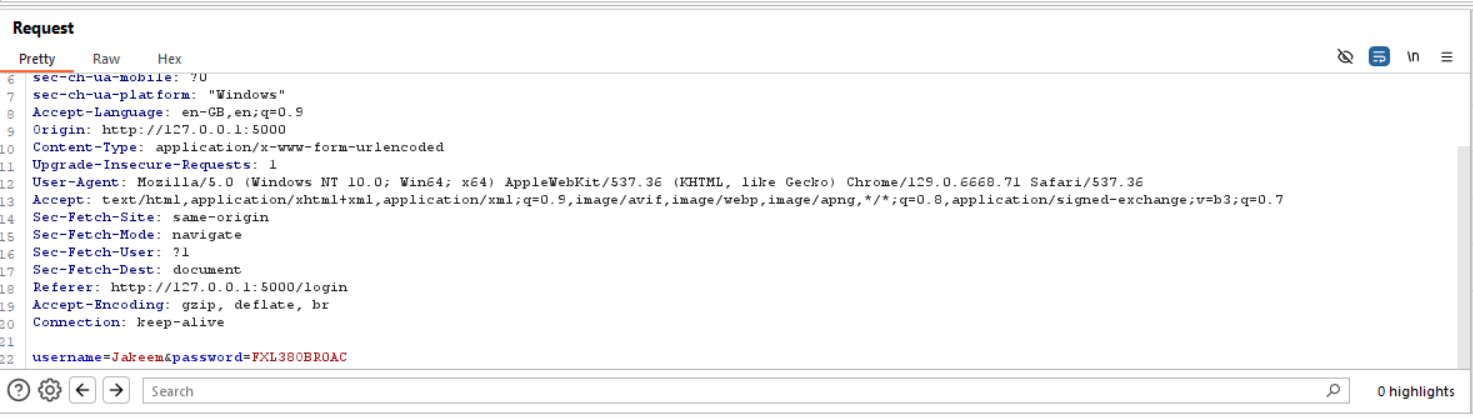
While in our instance, the banner says nothing, this vulnerability could be exploited if the hacker decides to write a more in-depth script that will show the users cookies on the hacker's server if they enter this part of the website. The code itself will also be hidden on the page so only the message “Hello” will be displayed.



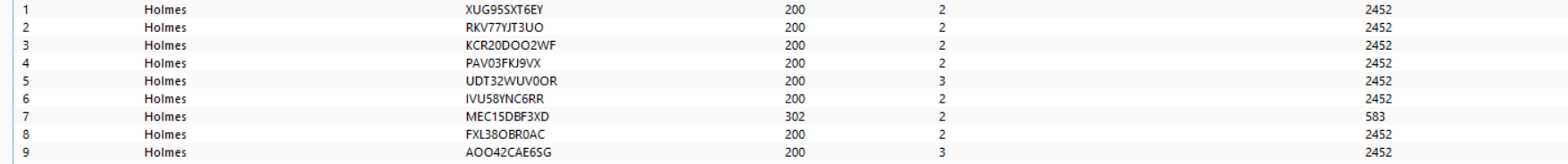
## 4.Brute-force Authentication Attack

Let’s say that a hacker wanted access to a certain account, for this instance we want to find out the password for Holmes. The hacker has a list of passwords, but the website has a limit to how many attempts to login. The hacker can use Burpsuite to brute-force attempts and find out which password is Holmes’.

Using Burpsuite, the hacker can get something like this to display by using another account.



After the hacker follows a process of setting up a “Cluster Bomb” attack with two payloads, one being the username Holmes and the other being passwords, they can start attack and Burpsuite will give back the results.



We can see that the status code and length for the 7th password is different from the rest, this indicates that the password could be correct. This can be then tested if we see the response in browser. As we can see the Authentication Brute-Force attack has worked.

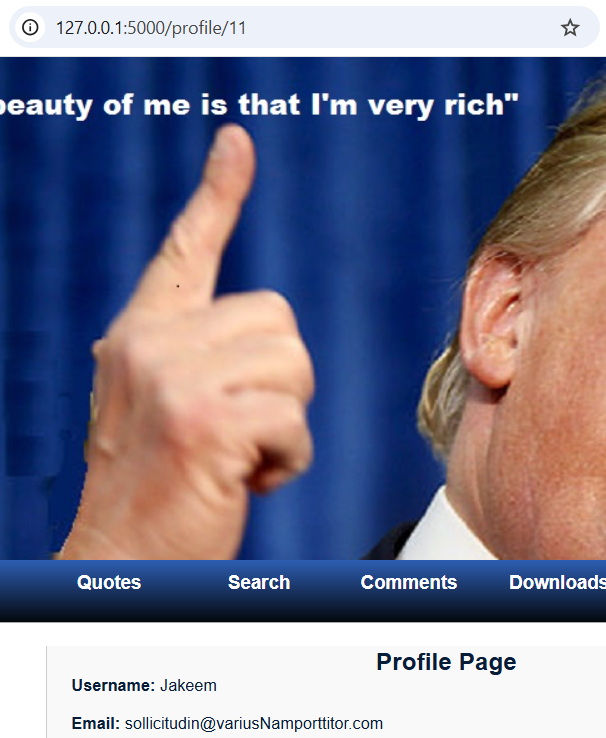


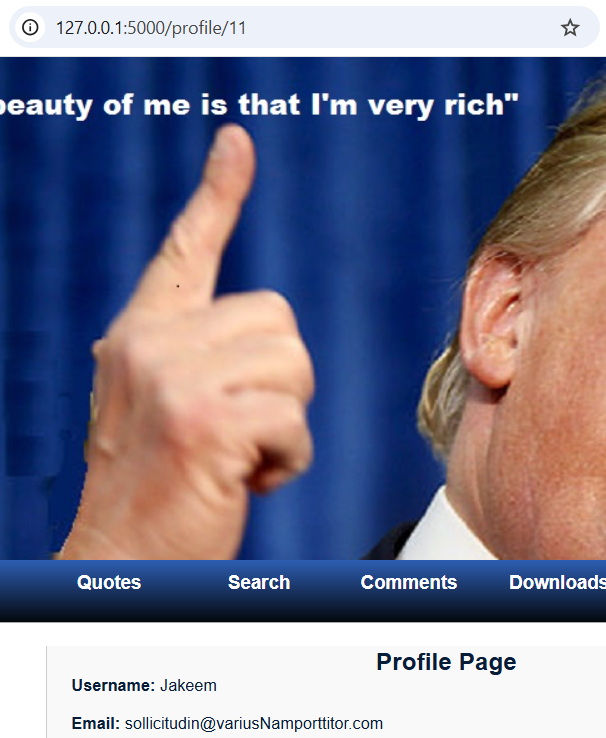
## 5.Command Injection

As the port of the website is 5000, the hacker an create a reverse shell on a netcat session which will allow them to execute commands remotely on the server. If done correctly they could get access to every user's info on the website.

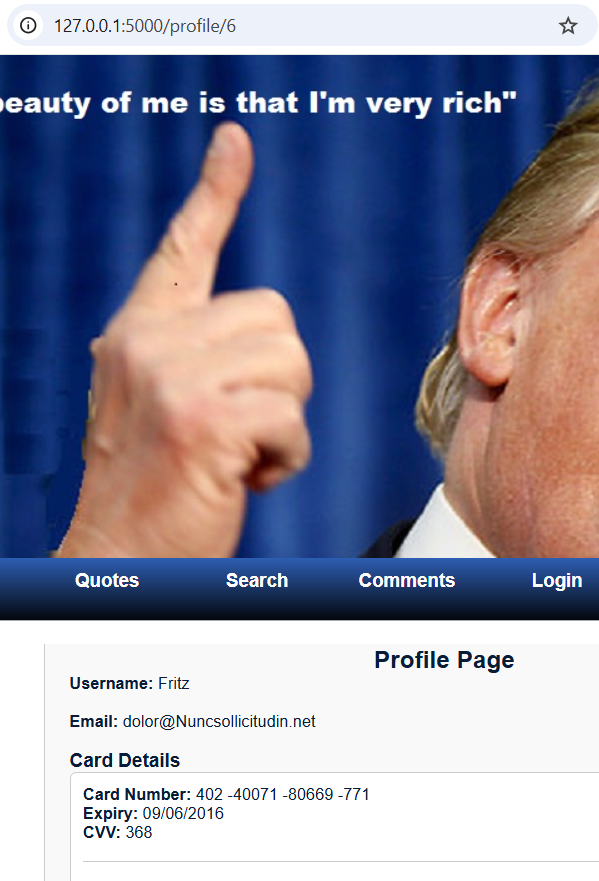
## 6.Insecure Direct Object Reference

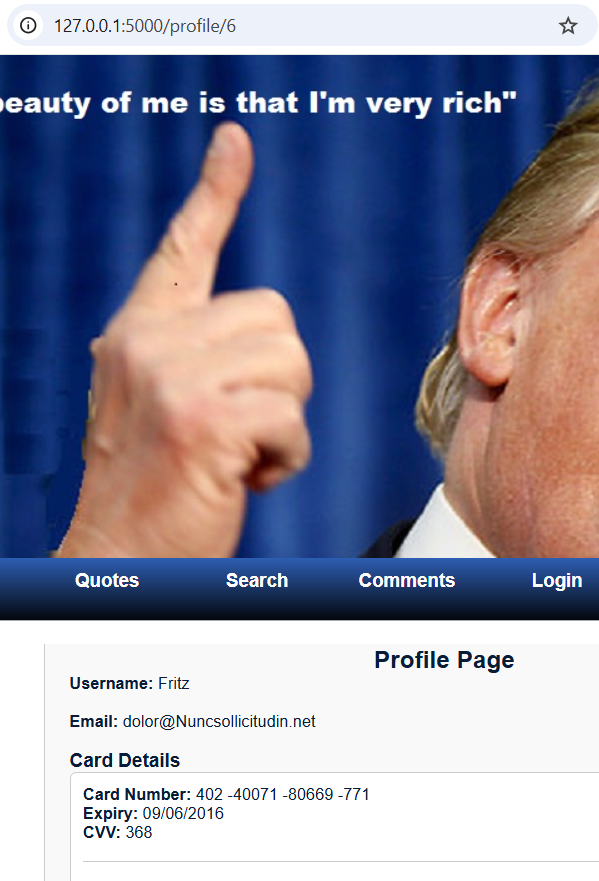
When logging in as Jakeem for example, the URL changes to this:





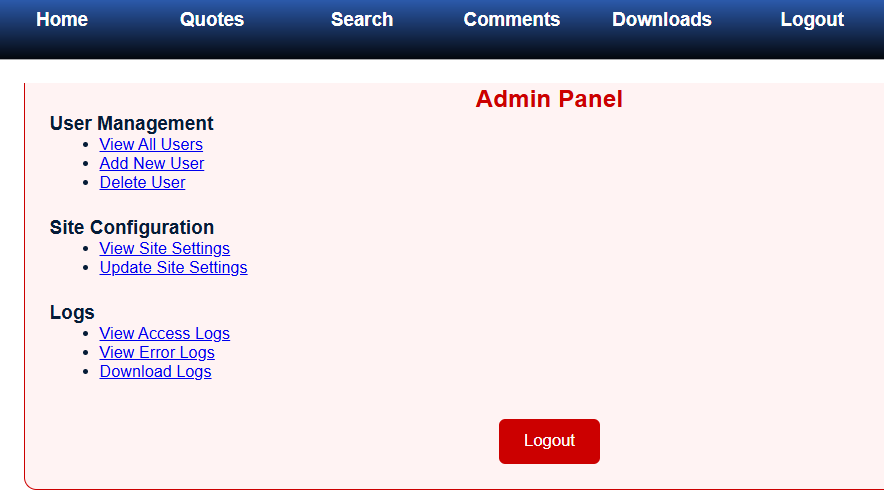
The hacker can now see any profile including their bank details by just changing the profile number. If he wants to see details of profile 6 he can just do this.





## 7.Broken Access Control

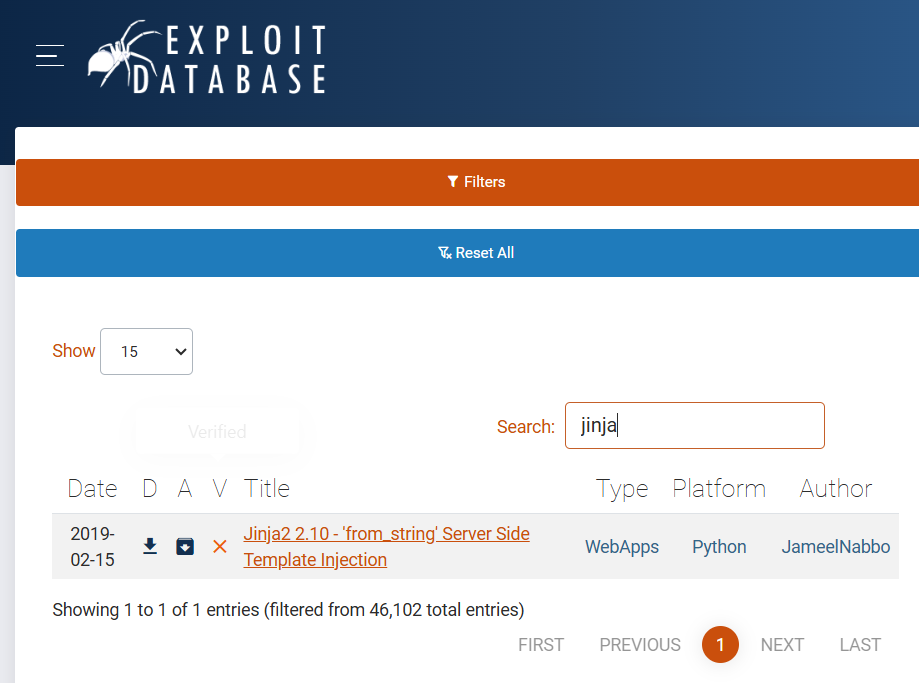
Looking at the code we can see that there is an admin panel on this website. By changing the URL of the site to “127.0.0.1:5000/admin\_panel” we get access to this panel.



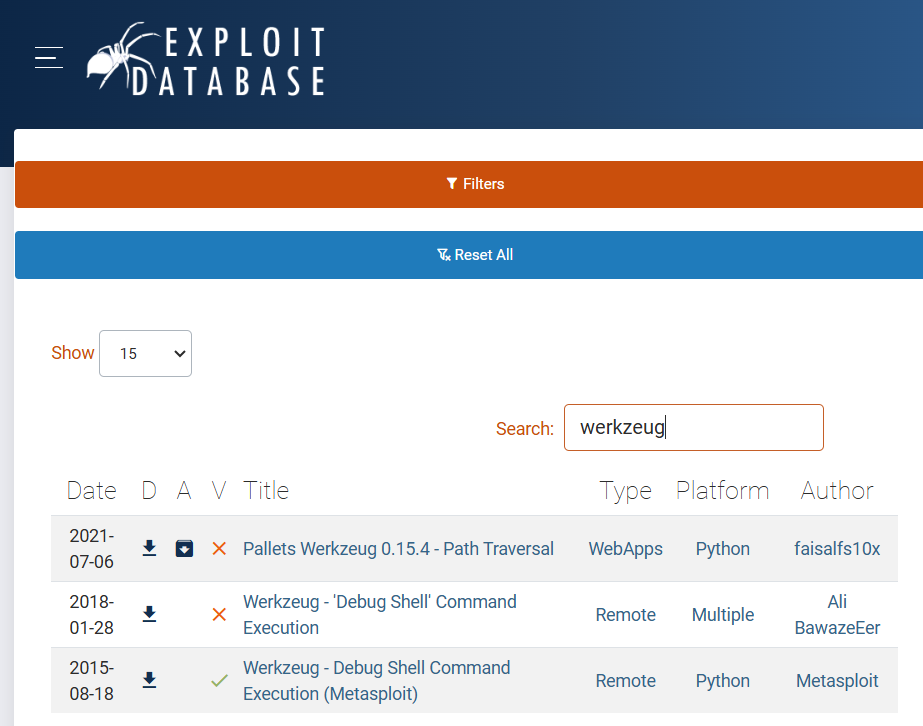
With this panel, the hacker could now do anything with the site such as deleting users, changing setting or checking access/ download logs.

## 8.Vulnerable and Outdated Components

The code for the website does include some vulnerable components in the lib folder. Two components “Jinja2” and “Werkzeug” display vulnerabilities on exploit-db.com



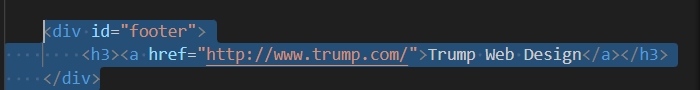
We can see that Jinja is vulnerable to a Server-Side Template injection. This essentially puts users at risk as the hacker could take complete control over the server.



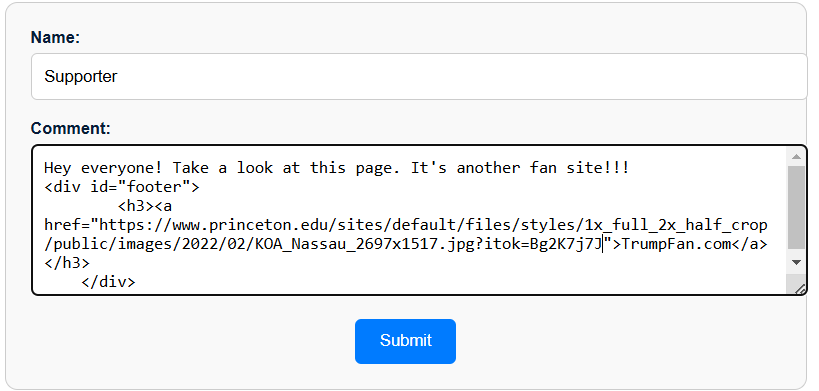
Werkzeug has 2 vulnerabilities here. Path Traversal and Debug Shell Command Execution.

## 9.Cross-Site Request Forgery

Looking at the code used in the Quotes section, we can see that there is a link made to the trump website.



A hacker can exploit this link in the Quotes section by going into the Comments section and then writing something like this



The hacker can put any href link that they want and when submitted, the page will only display the comment and fake URL added as a clickable link.



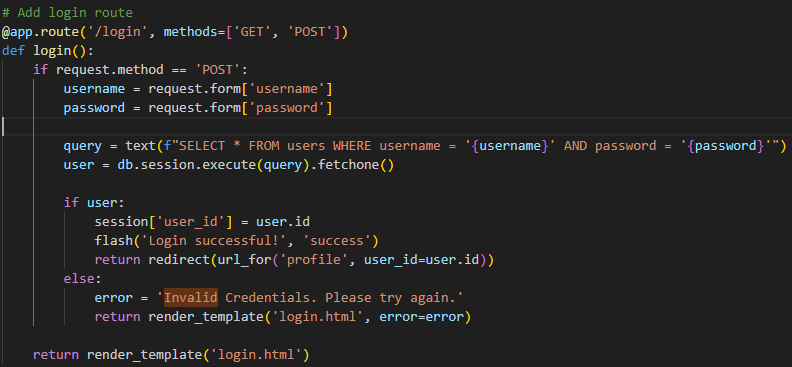
Let’s say that this could be an inline frame link, the hacker could use an oblivious users' cookies if they clicked on the link. They could then issue commands such as account deletion etc.

# Section 2: Fixing The Vulnerabilities

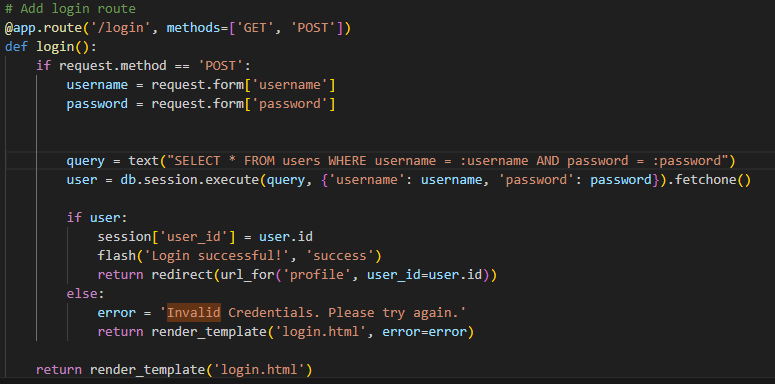
# (Daniel Mocek)

1. SQL Injection

The code has SQL Injections attack vulnerability because the user input for username and password is used directly inside the query, found in apps.py, instead, we should use prepared statements.



As we can see here the username and password are directly being selected here, the issue lies within the query and db.session code instead we should use

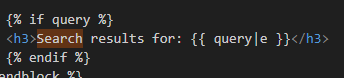


The updated code prevents SQL Injections by using :username and :password placeholders, this enables safer input handling.

2. Reflected XSS Vulnerability

The attack is enabled by one line of code in the search.html



The “|safe” function of the code indicates that the text is safe and does not perform any safety functions. This can be easily fixed by replacing |safe with |e

this enabled html escaping, this replaces special characters with their HTML equivalents, this ensures the browser displays plain text.

3. The Stored XSS vulnerability

The attack is enabled by one line of code in comment.html

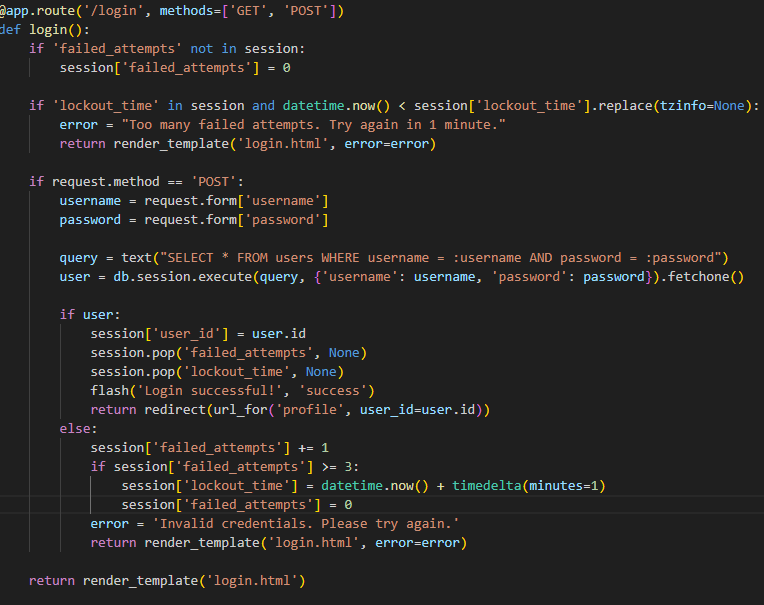


The “|safe” function of the code indicates that the text is safe and does not perform any safety functions. This can be easily fixed by removing |safe, the code will treat the comment as potentially malicious and will prevent any sort of XSS attacks



4.Brute forcing

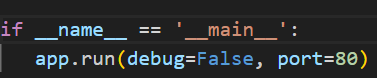
This attack is possible due to the fact there is no code preventing it from happening such as a captcha verification or time out due to too many failed attempts, i have went ahead and added a function which times out users for 1 minute after 3 failed attempts to reduce the possibility of brute forcing attacks.



5.Command Injection

To Fix command injection attacks we have to hide the port within the url a normal HTTP site should run on port number 80 however flasks default port is set to 5000, to fix this we need to force the application to run on port 80.

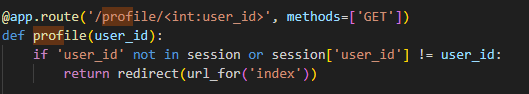
To do this we write a simple bit of code torun on port 80 and add it to app.py



This sets the port as 80, this way we avoid the port number being shown on the url bar.

## 6. Insecure objects

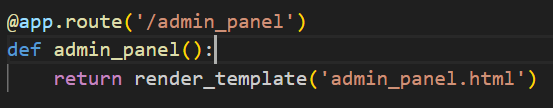
To prevent hackers from being able to access anyone's profile we need to add a function to check if the user trying to access the profile is the same user that's currently logged in, this can be done by checking if there is currently a session active with said user,



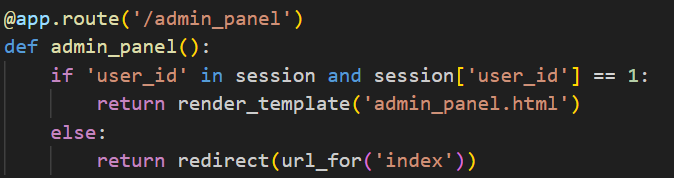
I have slightly modified the code to check for user sessions and if there is no session the hacker will be brought back to the login page.

## 7. Broken access control

To fix this vulnerability we must look at app.py at admin\_panel.



This can easily be fixed by only allowing one user to have access to admin panel, I implemented a feature where only user #1 has access to admin panel.

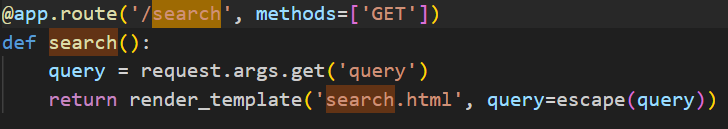


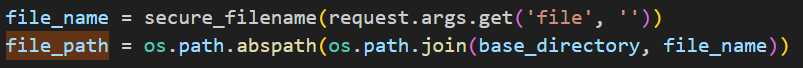
This code ensures that only when user number 1 is logged in you can go to admin panel otherwise you are redirected back to home page.

## 8.Vulnerable and Outdated Components

To fix these vulnerabilities we need to address some of the issues within jinj2 and Werkzeug,

Firstly, we need to clean our user inputs such as login, to do this we need to import the markup escape, and implement it within our code

Secondaly, we need to fix the path traversal attack to ensure the file is within its allowed file path, to do this we need to import secure\_filename from wekzeug.utils then we need to add it to our code by adding a line of code to check for the correct file path



## 9. Cross-site request Forgery

This exploit has already been fixed when dealing with the comments removing | safe disabled the possibility of submitting href links as it automatically clears the code of any special characters and prints only plain text.



# Section 3: Demonstrating the Fixed Vulnerabilities

# (Shahenwaz Muzahid)