**Activity 3 Questions:**

**Provide a link to the test cases you generated for this activity.**

<https://github.com/f1rehaz4rd/csec380_final_project/tree/master/tests>

**How do you ensure that users that navigate to the protected pages cannot bypass authentication requirements?**

Sessions. We check their session token and see if it is expired or not matching. If they do not match, then the page will not be loaded.

**How do you protect against session fixation?**

One way we mitigate session fixation is by regenerating the session IDs when a user logs in to access the page and is a randomly generated 32 byte string. Additionally, the session IDs timeout to create a safer environment. When logging out, the token gets marked and is popped off at the correct index.

**How do you ensure that if your database gets stolen passwords aren’t exposed?**

We have a function that validates the login by checking the database. In the database though, the password uses hashlib in python to hash the password using SHA-256. Since hashing provides the one-to-many mapping, if someone were to steal our database, the attacker could not decrypt the hash function.

**How do you prevent password brute force?**

Using long passwords that have the potential to be up to 200 characters. Additionally, the passwords are hashed using SHA-256.

**How do you prevent username enumeration?**

It returns a generic error message when a login attempt fails to be authenticated for both username and password failures.

**What happens if your sessionID is predictable, how do you prevent that?**

If sessionID is predictable, then an attacker can guess a valid ID and in turn, get access to the application. To prevent it, randomizing session IDs helps with the mitigation of an attacker taking access to the application.

**Activity 4 Questions:**

**How do you prevent XSS is this step when displaying the username of the user who uploaded the video?**

Sanitizing username input.

Sanitize input looking for specific HTML elements and HTML encoding. The way that we prevent XSS is by not using any javascript as well. Everything gets sent to out Flask backend.

**How do you ensure that users can’t delete videos that aren’t their own?**

Check the username between the current user and the owner of the video. If the username matches then the users can delete the video, otherwise the user can not. The user must go to their page in order to delete a video.

**Activity 5 Questions:**

**How would you fix your code so that these issues were no longer present?**

Parameterized statements to make sure that the parameters (i.e. inputs) passed into SQL statements are treated in a safe manner. Input sanitization is very key to all of this. We would use a safer manner of input like SQLAlchemy, which handles the input sanitization for you. For the vulnerability we use Pymysql.

**What are the limitations, if any that, of the SQL Injection issues you’ve included?**

When using Pymysql it doesn’t accept some commands, but it does just drop unsanitized user input into the database. This will allow for the user to execute anything that they want. The issue is when they input some of the commands they may not work with how Pymysql will but running it.

**Activity 6 Questions:**

**How would you fix your code so that this issue is no longer present?**

To fix SSRF you would simply have to sanitize the inputs of the URL before you made the request to the website. A couple of different ways that this can be done are checking the end filetype to make sure that it is an allowed type. You can also restrict certain URLs from being used like github for example.

**How does your test demonstrate SSRF as opposed to just accessing any old endpoint?**

Yeah the test demonstrates it downloading a script from github. Also, I did the tests that had been just send a request to google.com and that worked just fine. The code sends get requests to anything you pass to it which means that in theory, you could give it anything to access even local machines with the host name ‘mariadb’.

**Activity 7 Questions:**

**How would you fix your code so that this issue is no longer present?**

Encode or stop escape inputs correctly - scrub input values for potentially malicious characters:

; & | `

To fix the code I would do this process in a much more secure way. It does ‘ls’ on a folder to get all the files and takes some user input to filter the names out. If a user escapes the command though they can execute whatever they want on the system.