**Activity 3: Authentication**

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

<https://travis-ci.org/bms9294/CSEC380_Final_Project>

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

The pages check for a valid session before displaying any data or allowing access to any protected page such as video upload or the dashboard

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

It is not possible for a session to be set by a user, the sessions are generated and assigned by the server. We also do not allow the session cookie to be accessed via Javascript. These protections in place protect against various methods of session fixation.

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

The passwords are stored in the database in hashed form, this prevents possible password exposure if the database were to get stolen.

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

There is a timeout on the login functionality, after every login there is a waiting time of 5 seconds before another login can occur. This is not the best method for preventing password brute force but it would slow down an attacker unless they used multiple different machines to perform the brute force attack

**How do you prevent username enumeration?**

We do not indicate whether the user name exists attempting to login to the website which helps prevent username enumeration as a malicious user would not be able to know if certain user accounts exist.

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

If session IDs are predictable then a malicious user could gain access to the application simply by predicting a valid session ID and using that to get in. We prevent this by not using a predictable method of session generation, we did not use simple hash functions such as md5 and we avoided using predictable seed values for our hash function like time.

**Activity 5: The Content**

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

Content Security Policy prevents scripts from being loaded

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

We check the session to ensure that the user that is deleting the video is also the owner of the video.

**Activity 5: SQL Injection**

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

We would switch to using prepared statements with PHP PDO instead of using mysqli. While some SQL injection is still possible when using PDO to make mysql queries, we would also sanitize the inputs

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

There isn’t much in terms of limitations of the SQL injections we’ve included, the classic injection allows for subqueries by placing a mysql query in the title of the upload form. You can get pretty much any information you want through the classic SQL injection. As for the blind SQL injection there is also no limit, if you want to drop an entire table or the entire database it is entirely possible.

**Activity 6: SSRF**

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

The current SSRF attack is possible by requesting a thumbnail for a specific video and instead of passing in a video ID you pass in the path of a file. In our test case we showed that you could get our certificate through this method. We would mitigate this issue by instead of opening a file such as the thumbnail and returning it, we would just query the database for the path to the thumbnail and directly add that to the site instead of opening files and allowing users to pass in their own “video ids”.

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

Our test demonstrates that it is possible to use server functionality to request and read internal resources in this case, the server's certificate which was definitely not intended to be exposed or accessible by the user.

**Activity 7: Command Injection**

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

We added the command injection vulnerability in the video download part of code. To fix this problem we would switch from using the exec shell command to call curl and instead use the php specific curl command to download videos. We would also sanitize inputs of download urls to ensure that we we’re actually get urls.