**Security Analysis – Shotmaniacs Project**

There are 3 major types of security concerns we have prevented while developing this project. These include:

1. SQL Injection
2. Authentication / Authorization of the client
3. Stored XSS attacks
4. SQL Injection was prevented on the server side by sanitizing user input. This means that we used prepared statements whenever we established a connection to the database. Therefore, any query we have constructed using parameters given by the users through the request, the input was automatically sanitized by the prepared statement objects in java. These objects include a wide range of blacklist words which are omitted when inserted into the string query, therefore preventing SQL injection attacks.
5. Authentication / Authorization of the client. This was implemented completely on the server side. Firstly, authentication was implemented. This was done by generating a unique JSON web token upon user login (HTTP request to our auth routes). Our server is stateless, so we do not store states, therefore this token generated for the user is sent back to the user and stored in the user’s browser cache (session storage). We used external libraries to generate the JSON web token for increased security. Upon server launch we generate a new SHA-256 private key for the server. This key is stored on the server to **verify** and **generate** new tokens for previous and new clients. The tokens expire after 5 minutes or if the client closes the browser (session storage is emptied). Authentication goes hand in hand with authorization as we used the generated tokens to authorize different clients to different routes. As stated in the project requirements we have 2 possible roles for employees of Shotmaniacs – either admin or a normal crewmember. Whenever we generate a new token upon login, we also add a new Claim inside the token that represents the current role of the account with which the client logged in. Therefore, we can use this feature to authorize specific users to specific routes. Whenever a client makes a request to a route, we can extract the authorization token from the header and furthermore extract the role with which the token was signed (this was done with the private key so therefore we have confidentiality and integrity protection). On the server-side we also have a local list containing all the routes and their protection level. Whenever we receive a request, the request filter matches the route to this list and therefore checks the required role that is inside the token. The request filter can either accept the request if the required role (authorization) is present in the token or deny it. Therefore, these security measures that we have considered will prevent unwanted access to server routes and resources.
6. Stored XSS attacks are prevented since we do not store any user-input on the server side. All user-input is sanitized by means of prepared statements.