Introduction

This is a report on the security implementation of the lottery app. The lottery web app is a simple way for users to submit draw numbers for rounds of lottery. The winning draw number is decided by the admin. Users can view their playable draws, and the results of past rounds. The app has 2 user roles, 'admin', and 'user'.

Web security is very important. Without web security, we would not be able to purchase items online, have personal accounts, or even have online banking, as all of these require a high level of security, and cannot risk being successfully attacked. If we did have online banking and purchases that were not secure, we would be subject to constant fraud and theft.

Initial Examination

- Can access admin page without logging in
- Can access the profile page without logging in
- Can create lottery draws without logging in
- Can submit winning draws without logging in
- Passwords are stored as plain text inside of the database
- No input validation on any fields, including register, and lottery fields, only type validation
- Cannot login or logout
- Method not allowed error when trying to access logs and then clicking login
- Can view all users when not logged in
- No server-side validation
- Logged in as admin by default
- Database has no role-based access control

After examining the provided project, I have found many security issues. I can access the admin page without logging in. I believe this is because it thinks I am logged in as the admin, as when I submit a draw, then make that draw the winner using the admin page, I can see the winning draw belongs to the admin.

I can also access the profile, and account pages, without being logged in, as well as submit draws and view all registered users. Although, this may be related to the fast it thinks I am logged in as the admin.

There is very little input validation on fields. The only input validation I was able to find was on the draw inputs, where it limits you to use only numbers, however, this can easily be overcome by editing the html of the webpage, which tells me there is no server-side input validation either.

A lot of features appear to be broken, and show no error message, so I could be using them incorrectly, and I would not know. An example of this would be the logs button in the admin menu. I also get a 'Method Not Allowed' error message when I click on the logs button, and then try to navigate to a broken page, such as the Login page.

The database has passwords stored as plain text inside, which is a huge problem, as if the database is compromised, the attackers will have easy access to them. Hashing the passwords would make the

database a lot more secure in the scenario of a data breach, as they would have to De-hash the passwords, which is very difficult, and mostly involves guessing.

id	email	password
1	admin@email.com	Admin1!

The database does not have role -based access control, however, this is a limitation of SQLite, and should not be too much of a problem, as the python flask application should be the only thing accessing the database on behalf of the users using it.

There is very little to no error handling. An example of this is the 404 page not found error, and the 'Method Not Allowed' error. These take me to the default error page, with no way to go back without using the back button in the browser.

Security Programming

Input Validation

Input validation is validating the data the user has given you follows a set of rules. For example, aa password must be between 6 and 12 characters, and must contain 1 digit, 1 lowercase, 1 uppercase, and 1 special character.

Input validation is used to prevent attacks such as SQL injection attacks, to force users to input valid and relevant data, and to ensure that data is a high standard, such as the password example, so that user accounts are safe from having their password guessed.

Error Handling

Error handling means to catch when errors happen, and perform actions to mitigate, or alert someone about the issue.

This is important to implement, as default error messages can have sensitive information inside of them. We also want to make sure that the user understands what has happened, and has an easy way to return to other pages of the app.

Cryptography

Cryptography involves taking a piece of data, and transforming it into an unreadable string of bytes. We have used 2 types, hashing, and encryption.

This is essential for security, as when you have a data breach, you do not want your database to have passwords, and other sensitive information clearly readable by the attacker.

User Logins

The login and register forms have some simple security measures built in on the client, and others on the web server itself, such as validation. On the client, we obscure the password as it is being typed.

Obscuring the password ensures that others cannot read it off of the screen. This could be a very big problem if you are in a public location.

Two Factor Authentication

Two factor authentication is where you and the server both have a 'pin key' which is used to generate a pin. The pin changes based on the current time. This is then used as an extra layer of security when logging in.

This ensures that if someone gets access to your password, they will still not be able to login, as they will need the 2fa pin, which is very hard to guess.

Limiting login attempts

Limiting login attempts is where you give users a limited number of failed logins before not letting them try anymore.

This means that executing brute force attacks is much more difficult, as spamming login attempts will lock you out.

Logging

Logging is when you make a note whenever a significant event happens, such as an invalid user login attempt.

Logging allows admins to monitor the way their app is being used, and make it easier to find potentially malicious actions.

Role Based Access Control

Role based access control restricts what users have access to based on their role. For example, the finance department needs access to employees pay, but does not need access to engineering schematics.

Restricting access this way means that in the event of someone getting access to someone's account, they will not be able to access everything, it also means rouge employees can not cause as much damage.

Security Headers

Security headers tell the browser what features the app will use, sources to allow content from, and more.

They allow us to have fine control over feature set the app can use, which limits the chance of attacks like cross site scripting from occurring and having a large affect.

Secure Randomness

Secure randomness is ensuring that randomly generated numbers are indeed random. Normal random number generation is not necessarily random, as they don't generally need to be truly random, and doing it this way will save computer resources. Secure randomness is where the number generated is very difficult to be guessed or predicted.

This is important for our lottery numbers, as we don't want users to be able to guess or predict the winning draw.

Evaluation

Improving existing security further

There are a few areas which could be improved, and made more secure.

- Login limiter
    - This used the user's session to keep track of login attempts, however, this can be easily worked around by clearing your cookies. I suggest using something like the user's IP Address to keep track of this instead. Maybe even limit the login attempts for each user account for a period of time.
- Unencrypted pin keys
    - The pin keys for 2FA are unencrypted, which makes it a lot easier for an attacker to use if they get access to it.
- Decryption keys stored in the database
    - The decryption keys for the draws are stored in the database, which makes it trivial for an attacker to decrypt them if they get access to the database.

Missing security mechanisms

- Captcha would improve security against bots trying to brute force attack the app.
- HTTPS would make transmitting login credentials and other sensitive information to and from the web server more secure.
- Storing the database on a separate computer to the web server, and using credentials to access it would make it harder to steal, rather than just a file.

Difficulties

I had an issue with the Mimetype of my rng.js file. The browser would not execute it because of the secure headers setting strict Mimetypes. I spent quite a lot of time trying to debug this, but after some research I figured it was an issue with a registry on my windows install. The Mimetype module for python uses the registry set values on windows, but not on other operating systems, and for some reason my '.js' Minetype was set to text/plain instead of 'text/javascript'. After changing the registry on my computer and restarting, the issue was fixed.