Lottery app lab report – Toby Dixon

Introduction

This report is being written to document the transformation of a web app with no security features implemented, to a web app that has sophisticated measures in place to deter hackers from attacking and to make it as difficult to attack as possible for those who do decide to attack.

The web app in question is a lottery app. The idea of the website is for users to submit a draw of 6 numbers, either randomly chosen by the website or manually inputted by the user. Once all users have submitted their draws, the admin can choose a winning draw. The draws submitted by users will be compared against the winning draw, and if one matches that user wins.

For this application to work, there should be 3 types of users: user – someone that has an account and is logged in to the website, admin – someone who is there to administrate the website and choose winning draws, anonymous – someone who has not yet logged in and can choose to either register or login.

When it comes to a web app like this, it is very important to implement proper measures for security. Without proper measures implemented, a website is significantly more vulnerable to attackers with malicious intent. The web page is also a lot more likely to break due to unforeseen events such as invalid characters in usernames or users accessing pages that don't exist or they don't have permission to access.

Examination of lottery app

The initial app has all the functionality for the lottery built in, but almost no security features implemented. Any user, whether logged in or not, can access every single page of the website. This is likely related to the fact that there is no way to log in to the web app. This also indicates that there has been no attempt at implementing role-based access as this would stop users of the wrong role from accessing certain pages, regardless of whether the link is accessible to them.

When looking at the registration page of the website, I noticed that there was no input validation implemented. There was also no way for the page to display any errors with input, as errors are not handled at all. There is also no attempt at validating email addresses or phone numbers which makes the

webpage an easy target for bots. The login page doesn't limit login attempts from the user which also makes the webpage an easy target for bots as they can just brute force any users account without being stopped.

The website lacks two factor authentication. Adding two-factor isn't essential, but it can greatly reduce the risk of user's accounts being stolen especially if the database with all the login information for users is leaked.

There is no security in place to stop an unauthorised user from accessing a page that they shouldn't have access to, or one that doesn't exist. This shows a lack of error handling for HTTP response status codes. This is an issue as not handling these errors could result in the web page crashing.

When a user registers for an account on the webpage, their details are stored completely unencrypted in the database making them very vulnerable to any kind of attacks. Any leak would result in all logins and passwords being freely available. The same applies to any lottery draws stored in the database. There is no attempt at hashing them before storing them. This makes it very easy for anyone who has access to the database to see winning draw numbers, or any users chose draw.

On the topic of draws, when a user tries to check their own submitted draws for a given round, they are shown a list of all draws stored in the database. This includes the ones the admin has submitted as the winning draw numbers.

Another issue is that on the admin page of the website, there is no way to check logs. In fact, the website doesn't log any activity whatsoever. This doesn't add any risk to the website, but it makes tracking suspicious behaviour significantly more difficult.

The website also makes no use of security headers. Though this adds no risk, it is a very easy to implement security feature, and it provides security against a lot of common web security vulnerabilities.

Security Programming

One feature I implemented was input validation. I implemented this feature for several reasons. One reason is that forcing users to have certain characters in a password, and forcing that password to be a given length, is a great way to guarantee that their passwords won't be completely useless or easy to crack.

Also, making sure that the email is in the form of an email, and a phone number is in the form of a phone number etc… means you won't have invalid accounts stored in your database. It also makes it harder to have a bot generate lots of fake accounts.

Error handling was implemented so that when a user tries to do something the website doesn't allow, they know exactly why. It is also implemented so that trying to perform an action the webpage isn't built to handle won't crash the entire site. In essence, it is there to make sure the website doesn't spontaneously break when it receives an unexpected action.

I implemented cryptography for passwords and lottery draws before they are stored in the database. This is done so that no one can know what they are despite their privileges. Any admin that can view the database shouldn't be allowed to know any user's login information or any lottery draws. It also adds an extra layer of security to login information in the case of the contents of the database being leaked.

When it comes to handling user logins, I made all fields of the registration form and login form compulsory inputs. I also made sure that users are redirected based on their role, for example an admin is taken to the admin page whereas a regular user is redirected to their profile page. These features are added so that invalid users can't be added to the database and to make sure that users are directed to the main functionality of the web page related to their role.

Two factor authentication was implemented to add an extra layer of security to user logins. With 2FA implemented, a hacker could have access to all information about the user (their email, password etc..) but without the correct pin key they still won't be able to login to that users' accounts.

Also, on the lines of making logging in more secure, I limited the amount of login attempts a user can make. This prevents brute force attacks on websites, as these are done by repeatedly trying to login to a user's account guessing their details each time.

Another feature I implemented is access management. By this I mean restricting the access of different pages on the app to their corresponding roles. This is an important feature to implement as without this, any user can access admin controls, or any anonymous user could submit a draw. This

would completely undermine all the other security features implemented as logging into an account with a given role wouldn't make a difference.

I also implemented a logging feature. This is so that the admin can keep track of all user activity. This makes it a lot easy to identify any unusual activity. It also tells the admin if a particular account is being bombarded with login attempts, therefore identifying an attack against that user.

Security headers are another security feature I implemented into the web app. I implemented this as it is something that is very simple to do and provides protection against a lot of common web app security vulnerabilities. Along with this I also made sure that all random generation is cryptographically secure.

Evaluation

During this coursework, I ran into a few hurdles that took a while to progress past. These mostly came from adapting my knowledge to this particular use case. Not only were some of the implementations difficult to understand in themselves but working with a database alongside that amplified the complexity. All the difficulties I faced however I managed to resolve using the python debugger and researching my error messages.

In terms of added security mechanisms, there are a few other features I could've added, one of these being googles reCAPTCHA. This adds another layer of security when users are logging in, it can however be quite disruptive and frustrating for the user and can be difficult for some audiences. I didn't implement this as you need a google API key, which requires a google developer account.

Another thing I could've changed is being more careful with which data I hard coded into the web app. For example, the secret key for the app is written in plain text which isn't very secure, I could've randomly generated this key or stored it in an external file to make it more difficult to access. The same goes for initialising the database with the admin account. The admins login details and password are also written in plaintext.