CSC2031 – Security and Programming Paradigms

# Introduction

Newcastle University approached me to develop a prototype for their new lottery system. My application allows the user to create an account either as an administrator or public user. With a public role, the user will be able to view his account details, create a lottery draw with 6 number in the boundaries of 0 to 60 inclusive and create random lottery draws where six cryptographically secure random numbers are generated. In addition, they are also able check if they have won the lottery. As an admin role, the user will be able to view the account details (except their password) of all users.

Security requirements have also been fulfilled to confirm security threats are kept to a minimum as well as ensuring the application follows The Data Protection Act 2018. Implementations made include the of encryption /decryption of user’s lotters numbers, web servlet filters, input validations, hashing of passwords being stored in the database and Role Based Access Control.

During this report I will outline the approaches I took, steps used to solve various problems, the testing strategy used to ensure all requirements were met, further improvements that could be made to the prototype and my reflection on how well the project went.

# Approach

**Data input**

Various data input methods were designed and implemented to ensure the application could collect sufficient user data. Examples of data inputs can be found present in “account.jsp” and “userLogin.jsp”. Forms allowed the user to enter the details needed to create an account and gave the user a way to log into their account by providing necessary authentication credentials to the application. However, forms also make web pages vulnerable to a range of attacks. These attacks include cross site scripting (XSS) and SQL injections. The security approach I choose was creating a server filter called “ServletFilter”. A servlet filter is a server-side component which is used to pre-process and post-process client’s HTTP requests. Filters were mainly used in this application to perform input validation. This approach was justified as servlet filters successfully prevented SQL injections from occurring. The server filter queried the request and blocked any requests-response pair from passing any further if request parameters contained any SQL statements. Another security approach was input validation. Input validation meant the value entered by the user was checked against a set of validation criteria. This can be seen clear as phone numbers could only contain numerical digits. This approach Is supported as it prevents malicious statements being inputted by the user and thus stopping security threats such as Cross-Site Scripting from happening.

**Data creation**

Data creation can be seen visible in “account.jsp”. When a random number button is clicked, the application populates the user numbers form with 6 randomly secure numbers between the boundaries of 0 to 60 inclusive. A major security threat during this process of data creation can be found in insecure randomness. Insecure Randomness errors occur when a function that can produce predictable values is used as a source of randomness in security-sensitive context. Computers use a pseudo-random seed value to generate a pseudo-random number. This make the random numbers highly predictable. The security approach used to deal with this threat was Secure Randomness. The justification of secure randomness can be seen clear in that the values produces will be unpredictable. This will mean the users and attackers will not be able to predict the lottery draw the random function will produce.

**Data storage**

An example of data storage occurring can be found in “AddUserNumbers” servlet when the application stores into a text file the single string received from the 6 lottery numbers entered by the user. During data storage, RSA encryption and an hashing algorithm both were used. Encryption was implemented during the data storage of these numbers to minimise security threats. Encryption is the process of encoding information in a way that only authorized parties can understand it. Asymmetric cryptographic was used during the encryption algorithm meaning there is two different keys, a public key and a private key. Encryption can be justified as a secure approach as during the RSA encryption of the user’s numbers, a public key was used which encoded the data to help prevent unauthorised 3rd parties from understand the user’s lottery numbers. Another approach used was the hashing of user passwords when storing passwords in the database. A hash function was applied to the user’s password which produced a hash value. Hashing was chosen as it added another level of security to the application because user passwords in the database will not be understood by attackers without knowing the implementation of the hash function used.

**Data access**

The accessing of data allows the application to retrieve user data and process it. This can be seen clear in during the processing of user session attributes. The Application uses session attributes to ensure it is showing the user the correct data. A security threat that can occur with incorrect management of sessions is the possibility users viewing other user’s data. Correct session management is a security approach used to ensure this does not happen. Session management can be seen relevant as it means when the user logs out all relevant data about the user will be removed from the session preventing other users accessing the user’s data.

Another security approach used during data access was Role Based Access Control. Role Based Access Control can be justified since restricts system access to authorized users with the right authorisation level. Admins could access all user data except user’s hashed password and lottery draws whilst public roles could only access their own data and lottery draws. The final security approach used during data access was the limiting of user’s login attempts. If the user enters the incorrect login combination 3 or more times the application prevents the user from making any more login attempts. The use of limiting the user’s login attempts is justified because it prevents any brute force methods being used during the login process. My application also uses correct Java access modifiers to protect classes properties and methods this adds an extra layer of security preventing attackers from being able to change critical variables.

# Problem solving

During the initial setting up of the Java web application, I had difficulties getting the application running the way I needed. I recognized my project settings were incorrect which resulted to errors being shown and inability to run the lottery application on my device. To make the lottery web application runnable, the correct settings had to be in place and the Tomcat Sever had to be on my device locally. To solve this problem, I first had to downloaded Tomcat Sever locally from <http://tomcat.apache.org/>. I next had to ensure that Tomcat Server was deployed on my project and all artifacts were correctly built. Once Tomcat Server was successfully running, a problem arose in that I was unable to connect to the database and thus make new accounts in the application. To solve this issue, I watched a step by step tutorial on how to download docker locally and make MySQL runnable on a Docker Container. The solution used for this problem was <https://www.youtube.com/watch?v=NzKDlUVmIyE>. The final step was then making sure the correct Database URL was used during the application. This approach resulted in both Tomcat Server being successfully deployed on my localhost and the application being able to connect to a database using Docker.

A problem that arose during the development was the decryption of the user lottery draws in the text file. I recognized the whole text file could not be decrypted in one go since it would cause a IllegalBlockSizeException as a result of multiple sets of the user lottery draws being append to the file. To clear up this problem, I researched about how the bytes were being stored in the text file using <https://nullbeans.com/how-to-encrypt-decrypt-files-byte-arrays-in-java-using-aes-gcm/>. Once I understood this concept, I was able to come up with an algorithm with could split the text file into separate blocks of 256 bytes. I knew I had to have a counter variable which counted to 256 bytes. This would then indicate a user number has been read. Next step taken was then ensuring the correct decryption key was used in order to prevent InvalidKeyException from occurring. The problem was fixed by keeping a list of all encryption keys that have been used to encrypt the user’s lottery number and giving each instance of the encryption class an ID so the application knows which key is to be used to decrypt different parts of the text file. This solution resulted in me achieving my main goal of making the user numbers secure.

# Testing

The testing strategy used during the development of this project was functionality testing. Black box testing was carried out to test functionality of the application. Before I began testing the application, I created a Test Case document which outlined the list of tests to be conducted. Each requirement was treated as a functionality and thus treated as a single item on the list. A set of input values were designed to test that the functionality produced the desired results. One of the specific Black Box techniques used was boundary values. This can be seen evident in “account.jsp” where an input validation was implemented and tested to ensure the lottery draws numbers entered by the user were between 0 and 60 inclusively. Another black box technique used to ensure requirements were met was State-based testing. The application state changes were tested on provision of an input. An example of this was the testing of Role Based Access Control. I ensured program restricted users from being able to access certain pages if they held the wrong authorisation role. In addition, security requirement was also tested using White Box techniques. This ensured the control flow of the application branch conditions were tested for both being true and false. This meant that all statements can be covered. An example of this was ensuring the user could not attempt to log in again if they have 3 failed login attempts. After testing was completed, I created a Test Summary document with analysis of the testing done to conclude if the software was ready to lunch.

# Recommendations

One of the main requirements of this application was to minimise its vulnerabilities to security threats. Although this was achieved at a basic level, the application security could have improved. One-way security could be improved Is by changing the storage of the encryption keys to ensure it is securely stored. Attackers tend to target the weakest point of a security system and will not bother trying to crack an encryption algorithm if it is trivially easy to steal the keys. At the moment, the keys are stored in a session attribute. If the session attributes are comprised, the cyber attacker will be able decrypt all user’s numbers. To cover this security weakness, it would be better to store the encryption keys in the cloud using a third party cloud provider. Devices such as cloud encryption gateways that handle the encryption to and from the cloud automatically can help applications achieve this sort of security.

Another security enhancement would be the use of High-Level Authentication. Currently to access the application users only to provide their username and password. To make the access of data more secure the use of a static password and dynamic One Time Passcode could be implemented. OTP​ is a string of characters or numbers automatically generated to be used for one single login attempt. OTP’s add an extra layer of authentication and will minimize the risk of fraudulent login attempts as well as the risk of stolen data.

Furthermore, to make the application more secure I would also implement Transport Layer security. This will mean all data sent from the server and application will be encrypted. Including this layer will minimise the threat of Man In The Middle attacks and SSL hijacking to allow all data being intercepted by the attacker. In addition, the improvement to login attempts block could be made. Currently, attackers can use brute force methods to lock users out of their accounts on a shared PC to improve this. To prevent this the application, If the user reaches the login attempts limit the application could ask users to respond to a simple question. These questions are designed to differentiate humans from spambots. An example of this question could be to slide a button form left to right. This will stop any brute force method as the task is easy for humans to complete while the tool remains invisible to spambots.

# Reflections

During this project, all the requirements were achieved. I thought the project was challenging but I manged complete all 4 tasks. The easiest part of the project for me was the implementation of numerous data inputs to get data from the user and the addition of validation to ensure any input from the user was secure. I also found the creation of data and storage of data easy to implement. This was made easy because of the successful implementation of the data inputs forms. I understood the structure of the code and this allowed to quickly come up with the right algorithm to securely implement encryption and input validations to ensure the data was securely stored in text files and databases. A part of the project I found difficult was the initial setting up of Tomcat and Docker. I had to watch many tutorials on both software and make of use many external resource to make both runnable. I have learnt a lot from the setting up of this application and my focus of improvisation would be ensuring I understood the documentations of software required before I attempted to use them and configure them. Through the creation of this application, I have learnt new security measure such as the hashing of password, random secureness and Role Based Access Control. If I was to create the application again in the future, I would improve on what didn’t work as well such making a way for the application to hold on to the user’s encryption keys once the tomcat server stops and reruns again. This could be solved using a 3rd party cloud server where all user decryption keys would be stored.