Lab Report (CSC2031 Coursework)

Introduction

The university has asked me to design and implement some basic functionality and security elements to their lottery web application using Java, Tomcat server and MySQL database. My application allows you to create an account with either public user or admin role. Public users log in and participate in a six-number lottery, where they enter the numbers by themselves or ask the application to generate them at random. Admins have no draws in the lottery, instead, they have the ability to check all participants' account details, except password. I have used different types of approaches to ensure that the application is stable and the system is protected from data theft.

Approach

DATA INPUT

I used JavaScript library jQuery to implement a client-side input validation, which accepts account creation details only if they are filled in a correct format. Invalid email or phone number might lead to communication problems (e.g., admin cannot contact the lottery winner), while password validation prevents users from creating a weak password, which would be very easily cracked by the hackers. The program also ensures that details provided in the login form actually matches with an existing account. SQL injection is a popular web hacking technique, that uses special characters and words like "<, >, {, }, insert, into" and so on, to place malicious code into SQL statements. Therefore, I created a server-side filter, which denies form input involving any of those.

DATA CREATION

I used MD5 algorithm to create a hash of user's password for safe data storage, which I will discuss about later. As I have already mentioned, public user can either enter the lottery by providing his own six numbers or pressing a button to generate six random numbers. I developed a JavaScript function, that uses Crypto library, to generate cryptographically secure random numbers. Besides that, I created a data encryption Java class. When user submits his lottery draws, the program creates a file containing those draws and the file name is derived from the user's hashed password. If anyone without permission gets access to the computer that is storing those files, they wouldn't know which file is related to which user.

DATA STORAGE

I developed a program so that it stores all the account information, including hashed password, into MySQL database. Unexpected security flaws (e.g., SQL injection is performed using symbols we didn't include into prohibited characters and words list) would allow hackers to see inside the database, therefore using a hash instead of password written in plain readable text is much more secure. I used RSA cryptographic algorithm to make this system more reliable, the program encrypts lottery draw numbers which are stored as ciphertext in the text files mentioned before and the reason for that is the same.

DATA ACCESS

Brute force attack is a popular method used to guess already existing client's password by trying every possible combination, thus using JavaScript I designed a script that locks the login form after three unsuccessful attempts to enter the username and password. I have implemented Role Based Access Control strategy, which allows users to access different information depending on their role.

In this program, it allows admins to access all accounts' info, while lottery participants can only see their own data. I added a functionality to check and display if any of the user's current draws match with a winning draw, which is stored and accessed from the database.

Problem Solving

Most of the problems I faced when setting up the environment and tools for the development of the application were pretty clear after I watched John Mace software setup videos, so I fixed them quite easily. I had one setup problem linked to server port, which I could not solve, but Bradley suggested to change the port in Tomcat server configuration and it worked. When it comes to problems I got while developing the application, I approached them in different ways. We had a task to make a button populating form with six random numbers, so I read some module material and created a separate JavaScript file with a function inside it. I tested the functionality and it seemed like it worked, because the button was actually generating numbers from 0 to 60, but when I later accidentally changed the code, I realized that however I change the values assignment inside function does not change numbers being generated (e.g., I could change the function so that it assign a value 0 to all six numbers, but the button would still generate six numbers from 0 to 60, not six zeros). I sat there thinking – my application generates six numbers from 0 to 60 without even having a function to generate it, how cool is that. After testing it in different ways, I formulated an idea of what could actually be happening: the account page cannot reach the JS file, so it uses the earlier version of that JS file/function. I made sure the importing script tags were correctly placed and the number generating function had all brackets and symbols needed, but after that the problem was still the same. I thought that if my assumption is right, then deleting a file and putting the function inside "account.jsp" page would solve the problem, because the page could reach the function locally. I did that and it finally worked, but there is one interesting thing – I said if my assumption is right, then it should work, but it doesn't mean that if it works, then my assumption is right. Therefore, I still don't know if it is actually possible for a program to do that or was I just really blind and the error was something like a missing character. Another problem I was stuck on for a long time was the encryption problem. I was getting the common 256 bytes block size encryption error, so I used some variables and a for loop to divide the bytes into smaller blocks, but I later approached John's video on encryption and decryption of large files and I realized that creating a different function like he did was a better example, because the code looked cleaner, so I did just that. Decryption was working on a single draw, but it broke if I tried to decrypt several draws from the same file. I checked all the encryption class methods: is the program passing correct values as arguments, are the functions returning the correct data type and so on. Everything seemed alright, so I felt a bit lost until a thought came into my mind when I was reading module material again and again. I realized that my encryption class object was generating different key-pair object for every draw, while trying to decrypt the ciphertext of all draws using latest key-pair. I implemented an if statement to check if key-pair object has already been created, but it still didn't work… The if statement is inside the encryption class and a new encryption class object was created for every submitted draw, therefore I moved encryption class object instantiation outside the "doPost" method, which finally fixed the problem.

Testing

I have used several common approaches in order to test the web application. First one is exceptions. I have used try and catch clauses in pretty much every java class or servlet. Some try statements even have several catch clauses with different exceptions. I also set different session message attribute's texts depending on the functionality that caused the error to occur and display them on the web pages. Another technique that I used for testing is trying to break the system as a user. If I implemented an input validation that didn't allow password length to be less than 7 characters, I would simply enter a 5-character length password and see if the functionality works. If the system limited user logins to some number of attempts, I would just try to log in with a fake account enough times to pass the limit and watch the results. And it was the same for almost all functionality and security elements of this application – just put yourself in client's shoes. Besides that, I have used another common (probably not the best, but works for less experienced programmers in this field) testing technique, which is simply printing out messages or variables into IDE's console. Sometimes you add the functionality that uses several classes or methods and when it crashes, the error message doesn't give you a clear idea of which part of the program caused that, therefore you can print out different statements or variables to follow the steps that program executed before crashing. This technique contributed to my solution to the problem I faced with key-pair object causing decryption error. I printed out key-pair object variable in different places to check if my assumption about different key-pair objects for each encryption was correct. It was.

Recommendations

Even though I have implemented a lot of functionality and security elements into the application, it still has flaws. I will point out some of them, but there might be some unknown weaknesses I don't know about just like in any application.

First of all, choosing either a public user or admin role in the account creation page. I completely understand that the task was given to help us develop an idea of how system with role-based access works. In real life applications relying on peoples' conscience and giving an option for everybody to create an admin account would be a terrible idea. Companies should assign a role to each user depending on its task.

I used JavaScript library jQuery, to validate registration and login form input. For most of the account details it works pretty well, but when validating an email, the program only checks if the email has @ symbol in it (e.g., name@name would pass the test, even though it is not real). Therefore, additional email validation functionality should be implemented to strengthen the system.

It should be computationally infeasible to find two distinct messages hashing to the same value for a good cryptographic hashing algorithm. MD5 algorithm that I used is considered weak, because such collisions can be found pretty quickly (The algorithm is fast, which is a bad property for hashing algorithms). Although every algorithm has its pros and cons, I believe widely used SHA-256 would be a safer choice than MD5. Salting technique should also be used to add another layer of security.

The application does not check if the username inputted in the account creation form match with an existing username, therefore it is available to create two accounts with the same usernames, which depending on the application purpose might be a problem. Solution is simple – compare given input to the data that is already in the database.

Moreover, the program stores lottery draws in a text file using user's hashed password and that sounds secure, but imagine if two different lottery participants had the same password. It would

raise either of these two problems (depends on how the final application would look like). First – trying to get draws would cause decryption error to occur, because the program would try to decrypt data that was encrypted with two different key-pair objects. Second – user would get draws of both accounts with same password. Solution – text file's name could be combined from both username and hashed password.

Winning draw is stored to the database as a string, so if anyone got access to the database, they would win the lottery. Encryption is required to prevent this from happening.

Reflection

This project once again proved to me that one thing works all the time. It is the time itself. I got stuck for several days on some problems like making sure the password has at least one uppercase, one lowercase letter and a digit. I read the module material on this topic, then searched the internet for information about regular expressions, tried different ways, but it didn't work. I had no idea what to do next, but I still sat just looking at the screen for hours, because I knew that given enough time I would come up with a solution and I did. It is difficult to describe what didn't work, because probably every functionality I implemented didn't work at first, but after testing and changing the code, I was glad that all elements eventually did their job.

Before this module, I only had experience with Java programming language, but I have never developed an application on my own that combines several tools (e.g., Java, JavaScript, jQuery, Tomcat server and so on). Now the development of more complicated applications is much clearer to me. I have also gained experience using libraries and tools mentioned before, as well as strengthened my Java programming skills. Besides that, I now understand how database interacts with the application and how security elements like hashing and encryption works.

References

Java: https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html

HTML, JavaScript, jQuery: https://www.w3schools.com/

Security and functionality techniques:

CSC2031 Module material on https://ncl.instructure.com/courses/24641/modules