Hearts

Design, development and implementation of Hearts
2023-05-13

1. Introduction

This project is regarding the design, development, and implementation of the popular game of Hearts. It has been developed by Sanjeevkumar Sharma, Rajas Patil and Radhika Bilolikar. The focus of this project is on the security aspects of the game. We have developed it as part of the final project implementation of SWE 681 under the guidance of Professor David Wheeler at George Mason University. It has been submitted as part of the coursework on the 13th of May, 2023.

2. Design and Architecture of the code

This application follows the Client-Server architecture and uses the Polling protocol which facilitates communication between the server and the client. Polling is a reliable real-time communication method that our application leverages to ensure that users stay up-to-date with the latest information from the server. This approach involves the client periodically sending requests to the server to check for updates. User data and other game data are stored in a database. The user interface is required for displaying the game to the user and capturing any user input. A good UI design is both attractive to the user and easy to understand. This game includes database connectivity to store game data such as passwords and player data.

I. Language used

Our team has chosen Java as our programming language because it comes with a vast ecosystem of tools and libraries. Java is a good choice for building a multiplayer game like Hearts since it is well-suited for building scalable and reliable applications. In this project, our focus is on improving the security of the game so the use of Java is important because it comes with strong security features.

It provides developers with a complete security framework to ensure that the game code focuses on security from the beginning.

Because it is an object-oriented programming language, it enables code to be well-managed and more secure. It also ensures that the inputs and outputs are memory-safe.

II. Major components of the source code

The game has been developed using a combination of Spring Boot 3.1.0 for the backend, Angular 15 for the frontend, and PostgreSQL for data storage. This particular stack has become a popular choice among web developers for its ability to streamline the development process while offering advanced functionality. The combination of Spring Boot and Angular enables effective management of both the presentation layer and business logic, while PostgreSQL provides a robust data storage solution that can accommodate complex data requirements. In particular, PostgreSQL's support for features such as JSON data types and full-text search make it a reliable choice for managing modern web application data.

III. Major classes

- Controller-related classes:

  AuthenticationController - This class contains the functions register and authenticate.

  GameController - This class contains the functions createNewGame, joinGame, joinAnyGame, gamePoll, startGame

  PlayerStatisticsControllers - This class contains the function getPlayerStatistics.

- Domain-related classes:

  Card

  -Card - This class contains the functions getSuit, getRank

  -Deck - This class contains the function getCards

  Game

  Player - This class contains the functions getUsername, isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled

  PlayerDirection

  PlayerStatistics

  Role

Status

- Model-related classes:

  AuthenticationRequest

  AuthenticationResponse

  GameInstance

- Repository-related classes:

  GameMoveRepository

  GamePlayerRepository

- Service-related classes:

  AuthenticationService - This class contains the functions register and authenticate

  GameService - This class contains the functions joinGame, getGameInstance, joinAnyGame, startGame, getCards, playCard, getHandWinnerFromGameMoves, getPointsFromGameMoves, getRank

  JWTService - This class contains the functions extractPlayerEmail, isTokenValid, isTokenExpired

  PlayerStatisticsService - This class contains the function getPlayerStats

- Config-related classes:

  ApplicationConfig - This class contains the functions userDetailsService, authenticationProvider, passwordEncoder, authenticationManager

  JWTAuthFilter - This class contains the function doFilterInternal

  SecurityConfiguration - This class contains the function securityFilterChain

IV. Major functions

We use functions with least privilege to improve security so that the attack surface is minimized. The following are the main functions in our source code:

Register : This function is used to create a game account for new users. The user chooses their username and password which is then saved in the database. These user details are used for validating the identity of the user in consequent sessions.

Login : This function is used to validate the username and password for user authentication. If the user input details match with the data present in our database, then the user is logged into the account.

StartGame : This function has been implemented so that a user can create a new game session for other players to join.

JoinGame : This function has been implemented so that users can join an existing game. The game begins when four players have joined the game session.

Statistics : This function is implemented so that the users can view their win or loss statistics. This helps them keep a track of their performance in the game.

3. Installation Instructions

I. Installation procedure

To play the game, the user needs to have the following on their system:

- Updated Web Browser that includes Google Chrome, Safari etc
- OS Windows 7+, Mac OS X 10.7.3+
- The user needs to enable cookies

Step-by-step instructions for running the game:

1. Unzip the file
2. You will find three folders in the unzipped directory namely hearts-card-game-server and hearts-client, certificates.
3. The user also needs to have a PostgreSQL database installed. You can download that by going onto the link here https://www.postgresql.org/download/ .
4. Once the database is installed, start the pgadmin.exe file and create a schema with the name hearts. The SQL credentials need to be specified in application.yaml in the server side code. The database will get created once you run the server side code.

5. In the unzipped directory, the hearts-card-game-server contains the server side backend code and hearts-client contains the client side frontend code.
6. To run the server side code, open cmd.exe and change the directory using the cd command to enter the hearts-card-game-server directory.
7. Make sure that maven is installed using the ‘mvn -v’ command. If maven is not installed, please download maven from here https://maven.apache.org/download.cgi and follow the commands given here https://maven.apache.org/install.html to install it.
8. Once, ‘mvn -v’ gives you the maven version. Execute the command, ‘mvn clean install’ and wait for it to run completely.
9. Next follow that with the command ‘mvn spring-boot:run’. This should start the backend server.
10. Now, to run the client side code, go into the hearts-client directory. Make sure that npm is installed by using the command ‘npm’. If node.js is not installed, download node.js and npm using the instructions here https://nodejs.org/en/download and install it with all the defaults.
11. Make sure once again that npm is installed using ‘npm’ command and you are in the hearts-client directory. Now run the command ‘npm]’install’ and wait for the command to execute.
12. Now, enter the command ‘ng serve’ and wait for the server to start. Once it starts, you can access the application at https://localhost:4200
13. You will get an error since the certificates for the application are not installed.
14. To install the certificates you can go to the certificates folder in the unzipped directory and install both the certificates named, hearts-cert.crt and localhost.crt.
15. Double-click on the certificate file to begin the installation process.
16. Click on "Install Certificates" and select "Local Machine" when prompted.
17. Choose "Trusted Root Certification Authorities" from the list of available stores and click "Next" and "Finish".
18. Wait for the success message indicating that the import was successful, then click "OK".
19. Verify that the certificate has been installed by checking the list of installed certificates in your computer's certificate store.
20. Once the certificates are installed, you should restart the browser and your system if required.
21. Now on your browser try to access the link https://localhost:4200 and you should be able to access the application.

4. Operating Instructions

I. Registration

The user needs to register with their username and password details. In case of an existing account, the user can login with their details.

II. Login and Sign Up

For a new user, there is an option to sign up for a new account. The user can do so by providing their username and choosing a password. These details are then stored in the database and used for validating the identity of the user.

For existing users, they can enter their username and password which is then validated. If their details exist in the database, they are logged in to their accounts. They can then see the home page. They are given the options to start a new game, join an existing game and view their statistics.

III. Beginning the game

Once logged in, the user can start a new game, join an existing game or view player statistics. This game is played amongst four players. Once a user joins a game, they can view their names along with the names of all the players that have joined the game as well. Once four players have joined the game, they get the button to start the game session.

IV. Game statistics

When the player choose to view their statistics, we display the following to the player:

Games Played - The number of games that the user has played

Games Won - The number of games that the user has won

Games Lost - The number of games that the user has lost

V. In case of win or loss

After the end of a game session, it is decided whether a player has lost or won. Then their statistics are accordingly updated. This helps users keep track of their win and loss record.

VI. Active and inactive game sessions

Game sessions are considered to be active when they are currently running. This includes sessions that have just started or are just about to finish. Game sessions are considered to be inactive when the game is terminated which could occur due to connectivity issues, server issues or other failures.

VII. Logging out

There is an option for the user to log out at the click of a button. To log in again, the user has to re-enter their login details.

5. Game rules

I. Number of players allowed in a game

In one game session, we allow four players to play against each other. After a player starts or joins a game, they can view the names of all the players currently available for a game. Once four players have joined, they get the option to start the game.

II. Time of game session

The objective of a game of Hearts is that the player who scores the maximum points loses. The game session is played for 13 rounds after which the winners and loser are declared. Players can view their win and loss statistics to see how well they are doing.

III. Rules for entry and exit of players

Users can join a game which has less than four players. Upon exiting a game session, the player loses the game. This is done so that players do not just exit the game when they are about to lose.

IV. Valid moves made by the player

We follow the usual moves that can be made by players during a game of Hearts. They are as follows:

- In each round, players must follow suit rules i.e., they need to play a card of the same suit as that of the leading card if available. If this is not possible, the players can choose a card from any suit.

- The trick is won by the highest card of the leading suit. If the trick contains a Heart or the Queen of Spades, the player who wins a trick, they earn penalty points.
- The game session continues for 13 rounds at which point, the winners and the loser are declared.

6. Why do we believe it’s secure? (Assurance Case)

1. Common Vulnerabilities and Weaknesses taken care of

The main goal of working on this project was to implement a game that follows all security protocols; a game that protects users and developers alike. We have tried to ensure there are no malignant attacks on the system, and in case of an attack, its effects are mitigated. We have followed a comprehensive set of security measures to ensure that the game is free from vulnerabilities and weaknesses. The following are the measures we have taken to make the application more secure:

- Proper design and implementation

  The game was designed to follow the necessary security protocols. This required a proactive approach to eliminate vulnerabilities that began right from the ideation of our game. It has been developed using a combination of Spring Boot 3.1.0 for the backend, Angular 15 for the frontend, and PostgreSQL for data storage. This particular stack has become a popular choice among web developers for its ability to streamline the development process while offering advanced functionality. This has helped us with maintaining the code and getting better performance.

- Access Control

  We authenticate users to ensure that only authorized users can access the game and user data. This is essential to ensure that the user data and other resources are protected from unauthorized access. By doing this, we have protected against any possible data breaches and loss of data. We have specifically tackled the issue of Horizontal Privilege Escalation by using JWT Tokens for authentication. JWT Tokens are a means of mitigating Horizontal Privilege Escalation, a security vulnerability whereby authenticated users gain unauthorized access to resources or functionality. Upon logging into an application, the server generates a JWT Token that is transmitted to the client for subsequent requests. The server verifies the authenticity of the JWT Token and the user's identity and associated permissions. As a result, the risk of Horizontal Privilege Escalation is minimized since users are restricted to only the resources and functionality permitted by their token.

- Input Validation

  This step is critical to ensure that the user input is both valid and not malicious in any form. We only trust input from valid sources. Moreover, we validate the user input data to ensure that it meets the required parameters. In this step, we followed the process of whitelisting, that is, we considered all user inputs to be invalid unless they matched with our regular expression. By doing this, we ensure that untrusted data does not slip through the system.

- Server-side Validation

  After the user submits their data, we validate the user input on the server side. This step is essential since it protects the system from malicious or unintentional attacks. All user input is considered untrusted unless it is validated on the server-side. This process helped us prevent malicious attacks such as SQL injections, Buffer Overflows, and accepting invalid input by making sure that the user input is valid before processing any database queries.

- Buffer Overflow

  Buffer overflow vulnerabilities in software can be mitigated through the use of input validation techniques. By implementing input validation measures on both the client and server sides, we have attempted to prevent attackers from exploiting these vulnerabilities.

- TLS Data Encryption

  For secure network protocols, we have used TLS v1.3 on the front end as well as the back end for HTTP since it provides a secure communication channel to protect the data that is transmitted between the client and the server. It also prevents any possible man-in-the-middle (MITM) attacks and makes sure that the user data is secure.

- Password storing

  We have used the Bcrypt password-hashing function to store the user's password securely in the database. When the user creates an account, we first hash the password using the Bcrypt function, then the hashed password is stored in the database. We have used Bcrypt because it uses salted hashing that adds random characters to the password before hashing. This makes it more difficult for attackers to gain unauthorized access to user accounts.

- Logging

  For logging, we have used SLF4J (Simple Logging Facade for Java) which provides an efficient logging framework for Java applications. We have chosen SLF4J over other prevalent logging utilities because it is more secure. It has different logging levels such as trace, debug, info, warn, and error. This helped us monitor the security of our game more efficiently.

- SQL injection

  In our development process, we opted to utilize JPA as a security measure against SQL injection attacks. This approach allows for the automatic generation of parameterized SQL queries, based on the entity classes, which is a widely recognized technique for preventing SQL injection. The use of JPA in combination with input validation, as a means of sanitizing incoming data, allows us to further reduce the risk of SQL injection vulnerabilities. By leveraging the mapping of Java objects to database tables via entity classes, we aim to strengthen the security of our game and ensure the integrity of our data.

- Security Headers

  The following security headers have been implemented in the code for the requests:
  i. X-Frame-Options: Deny

  X-Frame-Options helps handle security issues related to clickjacking attacks, which can be used to trick users into performing actions they didn't intend to

  ii. X-XSS-Protection: 1; mode=block

  X-XSS-Protection helps handle security issues related to cross-site scripting (XSS) attacks, which can be used to inject malicious code into web pages and steal sensitive user information.

  iii. X-Content-Type-Options: nosniff

  X-Content-Type-Options helps handle security issues related to MIME type confusion attacks, in which a malicious actor tricks a server into interpreting a file as a different MIME type than it actually is.

iv. Strict-Transport-Security (HSTS): max-age=31536000; includeSubDomains; preload

HSTS helps handle security issues related to man-in-the-middle (MITM) attacks, in which an attacker intercepts and modifies network traffic between a user and a server, often to steal sensitive information like passwords or financial data.

v. Content-Security-Policy: `default-src 'self';`

CSP helps handle security issues related to content injection attacks, such as cross-site scripting (XSS), code injection, and clickjacking attacks.

2. Review Process

   We have performed Peer Review to ensure that the security of the game is up to the mark. We have done a line-by-line analysis of each other's code to look for possible vulnerabilities or security threats. The goal of this process was to minimize the attack surface, gain a deeper understanding of the code, and ensure that we have followed the necessary security standards.

3. Penetration Testing

   We conducted a thorough Penetration Testing using the state-of-the-art OWASP ZAP Manual Testing feature at an elementary level, and we were able to discover that if an ongoing request is intercepted and the data being sent in the request is altered, client-side validation can be compromised. However, we have taken proactive measures to ensure the security of our system by implementing not only client-side validation but also server-side validation. As a result, even when we attempted to tamper with the data after intercepting the request, the request was blocked by the server and an error message was issued, indicating that our system is secure against malicious attacks. The goal of this process was to simulate a real-world attack scenario and attempt to exploit any vulnerabilities present in the system. By doing so, we could improve the overall security of our game.

4. Static Analysis

   I. Tools that were used

We have performed static analysis on our game code to check the quality of the code. We have scanned for vulnerabilities present in the software. We focus on finding the weaknesses that may turn into vulnerabilities. By doing this process during compilation, we have reduced the number of issues that could have occurred at runtime.

We have used SonarQube which is a commercial static analysis tool to analyze software quality and find weaknesses. It supports Java and gives detailed analysis reports on code quality, weaknesses, and possible security vulnerabilities. We have also used SonarLint which provides real-time feedback on possible security issues present in the code. It helped us write safe and clean code. By doing this process, we have improved the security of the game.

II. Results

The tools we used have provided a complete report on the security vulnerabilities, code quality and recommendations.

Some issues found during our analysis using SonarQube and SonarLint are:

- Null Pointer Dereferencing
- Immediately return this expression instead of assigning it to the temporary variable "response".
- Rename this field "SECRET_KEY" to match the regular expression '^[a-z][a-zA-Z0-9]*$'.
- Define a constant instead of duplicating this literal "authorization" 5 times.
- Remove this useless assignment to local variable
- Remove this unused import 'jakarta.persistence.Id'

III. How did we utilize the results?

Upon using these tools, we have found some issues present in the code. As a result of performing the analysis, we have been able to remove some vulnerabilities present in our code thereby improving the security of our game. We have fixed the issues that could be potential security vulnerabilities, whereas we could not get rid of a few due to design constraints.

5. Dynamic Analysis

To perform Dynamic Analysis, we have chosen OWASP's Zed Attack Proxy (ZAP) since it is a free and open-source security tool that provides various scanning tools to detect vulnerabilities in software. This dynamic analysis tool is very popular and is used by many developers.

ZAP scans applications for vulnerabilities and security issues present in the software. It includes a lot of security tests that check for SQL injections, cross-site scripting (XSS) and so on.

I. What was checked

This tool generated a report that assigned risk severity levels to each risk that it found in the game code. The risk levels are classified as: High, Medium, Low, Informational. We began by fixing the risks that were classified as having high severity and then moved down the list.

II. Results

Alert Type	**Risk**	**Count**
CSP: Wildcard Directive	Medium	2
Content Security Policy (CSP) Header Not Set	Medium	2
Cross-Domain Misconfiguration	Medium	2
Strict-Transport-Security Header Not Set	Low	12
Modern Web Application	Informational	2
Re-examine Cache-control Directives	Informational	2

III. How did we utilize the results?

Upon receiving the results of this report, we have managed to remove a lot of vulnerabilities present in our game code. We were able to identify the absence of security response headers in our application because of the analysis done by the automated scan of OWASP ZAP. After adding the necessary security headers in the server requests, we were able to minimize a majority of the issues. Our goal was to make sure that the attack surface is minimized and that the probability of an attack is greatly reduced and the risk involved if the application gets attacked is mitigated.

6. Measures we have taken to make our code more secure

I. While designing our game, we considered the perspective of a user to make the software more user-friendly and understandable to the average user. We also performed logging to help manage the vulnerabilities present in the game code. We performed peer review to gain a deeper understanding and potentially improve the quality of the code.

II. We have followed the principle of least privilege so that no changes may be made by an unauthorized user and also so that the data may only be accessed by those users with proper authorization. We have minimized the attack surface by limiting privilege for users and processes so that attackers do not get to exploit them.

III. To make the game more secure, we have begun by establishing a HTTP connection using TLS v1.3 which provides a secure communication channel to transmit the data between the server and the client. We have used the Bcrypt password-hashing function to store passwords securely. We validate the user input data to prevent unauthorized access or malicious attacks.

IV. We have used SpringBoot 3.1.0 that is a popular Java framework which provides preconfigured components and dependencies to be integrated into the project. This includes an embedded web server and database connectivity. We have used Lombok, which is an annotation-based Java library, to remove a lot of boilerplate code. This helped us focus on the security aspects of the game.

V. The game has been developed using a combination of Spring Boot 3.1.0 for the backend, Angular 15 for the frontend, and PostgreSQL for data storage. This particular stack has become a popular choice among web developers for its ability to streamline the development process while offering advanced functionality.

VI. We have also implemented JWT (JSON Web Token) for authentication. JWT is a standard method for securely transmitting information between parties as a JSON object. By using JWT, we are able to securely authenticate users and transmit encrypted data, helping to prevent unauthorized access to our systems and user data. JWT Token can internally send in an XSRF token protecting the system against XSRF attacks.

VII. We have used JPA(Java Persistence API) in the Data Access Layer. JPAis an excellent choice from a security standpoint because it provides a standardized and secure way to manage database access. By using JPA, we can easily apply security measures such as access control, encryption, and data validation. Additionally, JPA reduces the risk of SQL injection attacks by providing a built-in mechanism for parameterized queries, further enhancing the security of our database operations.

7. Why do we believe that our security mechanisms are enough?

As a team, we have taken a multi-faceted approach to security, including the implementation of a range of measures designed to protect our systems and our users' data. For example, we use BCrypt to securely store passwords using salted hashes, and TLS 1.3 to provide strong encryption and authentication for internet communication. Additionally, we have implemented the necessary security headers, to help prevent different types of attacks on our web applications. We have also implemented client-side and server-side input validation to prevent data injection attacks. Using JPA to manage database access securely is a good practice as it helps reduce the risk of unauthorized data access and modification by enforcing strict access control and preventing SQL injection attacks. All the measures taken by us for securing the application are in line with the industry standards.

8. Future improvements

I. To improve this project in the future, we would better the GUI. This would make the game more user-friendly and visually pleasing to the players. It would also promote better understanding and enable the users to perform multitasking.

II. We can perform Penetration Testing more comprehensively to make our game that much more secure.

III. We can also add a provision so that players who create a game are given an access code. They can then share this access code with other players so that they may also join the game session.

IV. We can add rewards and achievements so that users have a goal to work towards while playing this game.

9. Our evolving thoughts on securing software

We believe that any amount of security on an application is not enough. Given enough computational power, resources, and time, we believe that no matter how impregnable a system is, its security can be breached. However, practically speaking, this is not a feasible task. For example, in 2014, a group of researchers estimated that it would take a single Nvidia GTX 580 graphics card 12 days to crack an 8-character Bcrypt hash with a work factor of 10. For longer passwords or higher work factors, the time required would increase significantly. We can run these kinds of numbers all day, however, these are just estimates and can only be theorized. But these kinds of implications raise open ended questions like "When do we stop?".
To answer this, we have standards in place. We can think of these standards as a benchmark set by researchers after running these numbers. These benchmarks help us quantify application security in terms of probability. Simply speaking, following these standards decreases the probability of an application's security being breached. This leads us to another question: "Why do we need to stop? Can't we just keep securing the system until we no longer can?"

Well, security comes at a cost. And the benefit that you can achieve from that

should be comparable to the costs incurred. In real-time systems, a cost-benefit analysis is of the utmost importance, when it comes to security mechanisms. Evaluating the cost of additional security measures against the potential benefits of implementing them is a significant factor in deciding whether to secure the system further or not. If the cost of additional security measures exceeds the potential benefits, it may not be necessary to continue securing the application.

What we have come to believe as a result of working on this project is that securing an application while in the development phase is fine, but it is not enough. Attackers come up with new malicious attacks everyday. If we do not continue to evolve the security measures that we take to counter their ever-evolving attacks, the security of our systems will eventually fall short. Our idea is not only to look into re-securing our game code down the line, but also to apply this knowledge every time we develop applications in the future.

References

[Wheeler2015] Wheeler, David A. Secure Programming HOWTO. 2015-09-19. https://dwheeler.com/secure-programs/Secure-Programs-HOWTO/

[Auriemma2013]Auriemma, Luigi. Ferrante, Donato. Multiplayer Online Games Insecurity.2013-01-25. https://media.blackhat.com/eu-13/briefings/Ferrante/bh-eu-13-multiplayer-online-games-ferrante-wp.pdf