Five Card Stud Poker with Betting

Linus W. Freeman III & Brian Woltemate

2015-12-16

ABSTRACT: This is our SWE-681 Semester Project Final Report for our secure multiplayer Internet (web) gaming system for Five Card Stud Poker with Betting.

# 1. Introduction

We designed/implemented our Five Card Stud Poker with Betting project to allow users to:

1. create a new registration of username and password,

2. logon securely to our secure multiplayer Internet (web) gaming system.

3. start a new game.

4. join a game.

5. logoff.

6. resume a game by joining again.

7. take turns betting (raising), calling, or folding for each 2 through fifth card dealt.

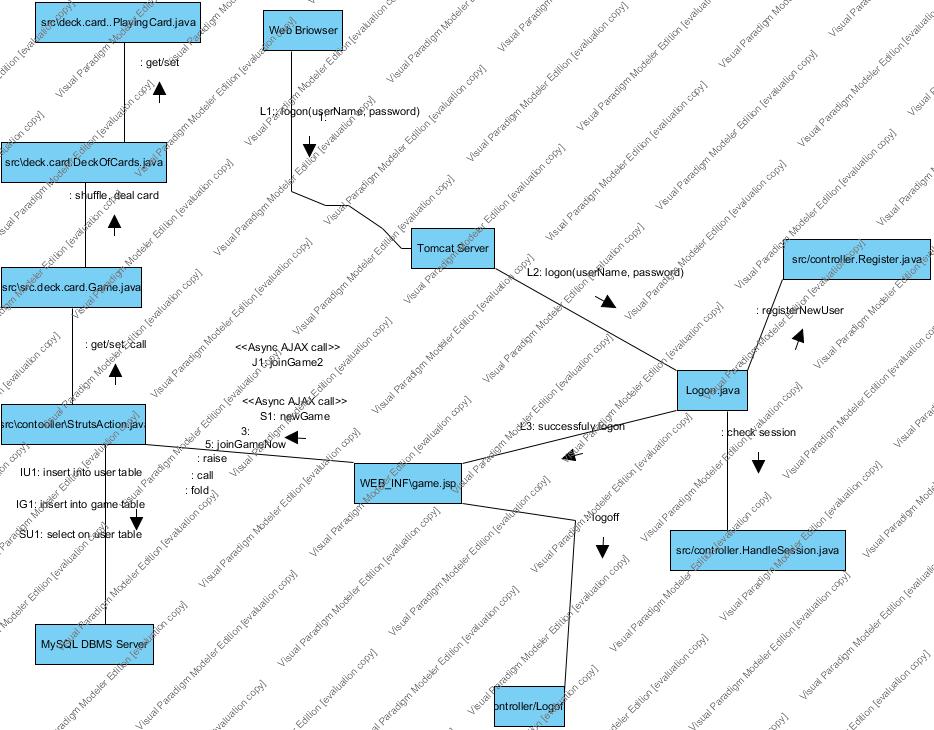
8. Listing completed games.

This web application uses SSL/TLS for encrypted communications between Web Browser clients (Chrome and Internet Explorer) and the web server. This application was built using Apache Web Server, Tomcat Server 8.x with JSP/Struts 2.x, MySQL DBMS Server 5.6.x, FindBugs for defect resolution, JQuery, AJAX, Java J2EE, and GitHub server & GitHub Desktop.

After logging in you are directed to a WEB\_INF\games.jsp page where strictly secure asynchronous AJAX mini-http calls are made to allow the users to take turns with game state management.

# 2. Design/Architecture

Here is a UML communications model of the major components and methods.



The user may start the Five Card Stud Poker with betting game with this URL:

localhost:8443/StrutsAjax/index.jsp

The login page is displayed. If the user clicks the New User Registration link the UserRegistration.jsp web page is displayed to capture a UserName and Password. When the user clicks the register action button the StrutsAction.java component inserts a valid authorized user record into the Users table and the login page is displayed.

The username and password are filtered using robust regular expressions.

After the player logs on the Game.jsp page is displayed that accepts actions to Start a New Game, Join a Game, and List Completed games.

If the user selects to Start a New Game a newGame an asynchronous AJAX call ‘newGame’ is made to the StrutsAction.java component that uses modular methods on the Game.java component.

If a second player logs on and selects the Join Game action button a ‘joinGame2’ async AJAX method is sent to the StrutsAjax.java component that lists the in play games to join.

The user selects one of the games to join and a ‘jonGameNow’ AJAX method is invoked in the StrutsAjax.java component.

A card is de

Now player 1 may either choose to

# 4. Installation Instructions

**Here are the current software download products and versions (version number and 64-bit):**

**MySQL Server Development Tool:**

http://sqldeveloper.solyp.com/download/

**Static Source Analysis Case Tool:**

findbugs-noUpdateChecks-3.0.1

**Integrated Development Environment:**

eclipse-jee-mars-1-win32-x86\_64.zip

**Model View Controller (MVC2) System Libraries:**

struts-2.3.24.1

**Java Web Application Server:**

apache-tomcat-8.0.27

**Database Management System (DBMS):**

mysql-5.6.27-winx64.zip

# 5. Operating Instructions

Here are the steps to running the Five Card Stud with Betting application:

1.) Start the MySQL DMBS Server 5.6.x by running [ bin\mysqld ]

2.) Using the MySQL bin\mysql client run the following from the source code zip file:

Source c:\StrutsAjax\MSQL\_Server\_DDL\fcs\_2.sql

3.) Start the Tomcat J2EE JSP/Struts Server with a valid self-signed server certificate.

Build and deploy the WAR file either using Eclipse with a embedded Tomcat Server, or an a separately installed Tomcat instance.

4.) Using either Google Chrome or/and Internet Explorer go to this web address:

localhost:8443/StrutsAjax/index.jsp

5.) An existing user may simply just logon. Or click on the New User Registration link and after creating a new account, logon.

6.) After player 1 is logged on he may a.) List Completed games, b.) Start a New Game, or c.) Join an existing game.

7.) If the user Starts a New Game then he waits for a player to join his game and then the first card is dealt down to both players and the next card up. Player 1 my raise or fold for each turn and player 2 may call or fold.

8.) Cards dealt second to fifth are bet on each turn with only cards 1 and 5 dealt facing down.

9.) At the conclusion of the game winner information with what each players hand ranks with total betting amount displayed.

# 6. Game rules

**Here are the five card stud poker and betting rules we will design and implement:**

https://www.pagat.com/poker/variants/5stud.html

**Here is a summary of the rules for five-card stud poker and betting:**

1. The dealer rotates counter clockwise and the application deals with secure shuffling of a deck of cards.

2. The first card is dealt "down" to each player.

3. Then each player receives the next three cards "up", one at a time with betting (up to $1.00 max bet for our project) for each new card dealt.

4. Each player can bet by doing one of the following: (1) call, (2) raise, or (3) fold.

5. After the fifth card is dealt “down" and bet on, a winner is determined based on the following card-hand rankings and receives the pot of money:

**The rank of hands from lowest to highest is:**

1.high card

2. pair

3. 4-straight

4. 4-flush

5.two pair

6.three of a kind

7.straight

8.flush

9.full house

10.four of a kind

11.straight flush

# 7. Assurance Case Why we believe it’s secure

In the implementation of this project we implemented the following in order to ensure the security of our software.

The first assurance we implemented was only granting the user fcs\_user SELECT, INSERT, DELETE and UPDATE on the FCS schema. This will prevent an attacker who compromises the application from gaining additional rights on the database outside the functionality of the game. We also made use of java prepared statements to reduce the applications exposure to SQL Injection.

A second assurance we implemented was the use of bcrypt password hashing. Bcrypt is based on the blowfish cipher and incorporates salt, which eliminates rainbow table attacks while also being adaptive so its complexity can be increased over time to combat better and faster hardware. [Wikipedia 2015]

A Third assurance we incorporated a regular expression for the username and password on the registration page. The user name must be alpha numeric and less then or equal to 11 characters and the password must match 3 of 4 Character categories: 1.) at least 1 upper case character 2.) at least 1 lower case character 3.) at least 1 numerical character 4.) at least 1 special character. We also do a character count on the password to verify it is a least 8 characters. The username regular expression is also used on the Login page. The use of this input validation strengthens user passwords making them harder to guess as well limits the potential number of inputs into the system thus reducing the risk of injection or denial of service attacks.

**Regular Expression From Stuts 2 Validation File Register-Validation.xml**

<validators>

<field name="username">

<field-validator type="regex">

<param name="regexExpression">[a-zA-Z0-9]{1,11}</param>

<message>Username must be AlphaNumeric and no more then 11 charactors.</message>

</field-validator>

</field>

<field name="password">

<field-validator type="regex">

<param name="regexExpression">(?=^.{6,255}$)((?=.\*\d)(?=.\*[A-Z])(?=.\*[a-z])|(?=.\*\d)(?=.\*[^A-Za-z0-9])(?=.\*[a-z])|(?=.\*[^A-Za-z0-9])(?=.\*[A-Z])(?=.\*[a-z])|(?=.\*\d)(?=.\*[A-Z])(?=.\*[^A-Za-z0-9]))^.\*</param>

<message>Password must match 3 of 4 Character catagories: 1.) at least 1 upper case character 2.) at least 1 lower case character 3.) at least 1 numerical character 4.) at least 1 special character</message>

</field-validator>

</field>

</validators>

A fourth assurance we incorporated is using a cookie based authentication token that is set to HTTP only as well as to secure. The authentication cookie is a java-generated uuid that is sent to the client via cookie accompanied with a sha2-generated HMAC. The HMAC uses a secure random long stored in the database as its key and is recomputed and compared with the client cookie UUID and HMAC combination every time an action is completed. The Authentication token is then binded to the JSession based Session token by comparing the user id returned from the session retrieved from the database by the authentication token and the user id stored in the JSession. This allows us to use JSession functionally through out the rest of the application. This is implemented using Struts 2 interceptor that intercepts all action that implement the login-required interface. This allows the Login and registration pages to be view with out needing to log in. By using a HTTP on secure cookie HMAC verified security token the application is significantly less vulnerable to session fixation.

**Password verification, authentication token creation, HMAC creation and JSession binding in Login.java**

if(BCrypt.checkpw(password, hashPassword))

{

//if successful create authentication token and bind with JSession token

**try** {

connection.revokeUserSession(inComingUser.getUserID());

UUID Token = UUID.randomUUID();

Cookie PokerToken = **new** Cookie("PokerToken", Token.toString());

**Long** TokenRand = RandomContainerEnum.INSTANCE.randomContainer.getRandom().nextLong();

connection.createSession(inComingUser.getUserID(), Token.toString(), TokenRand);

Cookie TokenHMAC = **new** Cookie("TokenHMAC", CipherUtils.createHMACSHA256(Token.toString(),TokenRand.toString()));

PokerToken.setMaxAge(RandomContainerEnum.sessionTimeout);

PokerToken.setHttpOnly(**true**);

PokerToken.setSecure(**true**);

TokenHMAC.setMaxAge(RandomContainerEnum.sessionTimeout);

TokenHMAC.setHttpOnly(**true**);

TokenHMAC.setSecure(**true**);

servletResponse.addCookie(PokerToken);

servletResponse.addCookie(TokenHMAC);

**this**.session.put("UserID", inComingUser.getUserID());

**this**.session.put("user", inComingUser);

} **catch** (**NoSuchAlgorithmException** | **InvalidKeyException** | **UnsupportedEncodingException** ex) {

Logger.getLogger(Login.class.getName()).log(Level.SEVERE, **null**, ex);

addActionError("There was an error processing your request");

**return** ERROR;

}

}

else

{

addActionError("Invalid user name or password! Please try again!");

**return** ERROR;

}

**Session Verification in the Strut2 interceptor HandleSession.java**

if (servletRequest.getCookies() != null)

for(Cookie c : servletRequest.getCookies())

{

**switch** (c.getName())

{

**case** "PokerToken":

token = UUID.fromString(c.getValue());

tokenCookie = c;

**break**;

**case** "TokenHMAC":

HMAC = c.getValue();

HMACCookie = c;

**break**;

**default**:

//not an authentication token

**break**;

}

}

//If the get cookies is null or if there is no Token cookie/HMAC cookie or if the

//User ID is not in the session

if(token != null && HMAC != null && Servletsession.get("UserID") != null)

userSession = connection.getSession(token.toString());

else

{

tokenCookie.setMaxAge(0);

HMACCookie.setMaxAge(0);

tokenCookie.setValue("");

HMACCookie.setValue("");

servletResponse.addCookie(tokenCookie);

servletResponse.addCookie(HMACCookie);

Servletsession.clear();

**return** "SessionExpiredRedirect";

}

String recalculatedHMAC = CipherUtils.createHMACSHA256(userSession.getToken(),userSession.getTokenRand().toString());

if(HMAC.equals(recalculatedHMAC) && userSession.getUserID() == (int)Servletsession.get("UserID") && (userSession.getCurrentTime() - userSession.getTimestamp()) < RandomContainerEnum.sessionTimeout)

{

tokenCookie.setMaxAge(0);

HMACCookie.setMaxAge(0);

servletResponse.addCookie(tokenCookie);

servletResponse.addCookie(HMACCookie);

tokenCookie.setMaxAge(RandomContainerEnum.sessionTimeout);

tokenCookie.setHttpOnly(**true**);

tokenCookie.setSecure(**true**);

HMACCookie.setMaxAge(RandomContainerEnum.sessionTimeout);

HMACCookie.setHttpOnly(**true**);

HMACCookie.setSecure(**true**);

servletResponse.addCookie(tokenCookie);

servletResponse.addCookie(HMACCookie);

**return** ai.invoke();

}

else

{

tokenCookie.setMaxAge(0);

HMACCookie.setMaxAge(0);

tokenCookie.setValue("");

HMACCookie.setValue("");

servletResponse.addCookie(tokenCookie);

servletResponse.addCookie(HMACCookie);

Servletsession.clear();

connection.revokeSession(userSession.getSessionID());

**return** "SessionExpiredRedirect";

}

To find additional vulnerabilities we used the static analysis tool Find Bugs from the University of Maryland. Find Bugs helped us find vulnerabilities such as not marking instance variables that can be final to final such as instance variables in an enum, not properly closing database connections and hard coding the database password. In order to move away from hard coded database passwords we created a config file called DatabaseSetting.Properties where we store the password encrypted with AES128.

**Findbugs finding before code was update to correct found issues**

**Bad practice Warnings**

|  |  |
| --- | --- |
| **Code** | **Warning** |
| **DE** | controller.AjaxAction.sum() might ignore java.lang.Exception |
|  |  |
| **DE** | controller.AjaxAction.hello() might ignore java.lang.Exception |
|  |  |
| **DE** | controller.FiveCardStudPokerAjaxAction.getPlayerTurn() might ignore java.lang.Exception |
|  |  |
| **DE** | controller.FiveCardStudPokerAjaxAction.sum() might ignore java.lang.Exception |
|  |  |
| **DE** | controller.FiveCardStudPokerAjaxAction.hello() might ignore java.lang.Exception |
|  |  |
| **ES** | Comparison of String objects using == or != in deck.cards.Game.isFlush(PlayingCard, PlayingCard, PlayingCard, PlayingCard, PlayingCard) |
|  |  |
| **ES** | Comparison of String objects using == or != in deck.cards.Game.isFourFlush(PlayingCard, PlayingCard, PlayingCard, PlayingCard, PlayingCard) |
|  |  |
| **ES** | Comparison of String objects using == or != in deck.cards.Game.isStraightFlush(PlayingCard, PlayingCard, PlayingCard, PlayingCard, PlayingCard) |
|  |  |
| **ME** | entities.RandomContainerEnum.RandomContainer field is public and mutable |
|  |  |
| **ME** | entities.RandomContainerEnum.continuousDeck field is public and mutable |
|  |  |
| **Nm** | Confusing to have methods controller.FiveCardStudPokerAjaxAction.getUserName() and controller.Login.getUsername() |
|  |  |
| **Nm** | Confusing to have methods controller.FiveCardStudPokerAjaxAction.setUserName(String) and controller.Login.setUsername(String) |
|  |  |
| **Nm** | The class name controller.handleSession doesn't start with an upper case letter |
|  |  |
| **Nm** | The field name controller.Login.Password doesn't start with a lower case letter |
|  |  |
| **Nm** | The field name controller.Login.UserName doesn't start with a lower case letter |
|  |  |
| **Nm** | The class name controller.register doesn't start with an upper case letter |
|  |  |
| **Nm** | The field name controller.register.Password doesn't start with a lower case letter |
|  |  |
| **Nm** | The field name controller.register.Password2 doesn't start with a lower case letter |
|  |  |
| **Nm** | The field name controller.register.Username doesn't start with a lower case letter |
|  |  |
| **Nm** | The method name deck.cards.DeckOfCards.Copy() doesn't start with a lower case letter |
|  |  |
| **Nm** | The method name entities.MySQLConnection.CreateSession(int, String, long) doesn't start with a lower case letter |
|  |  |
| **Nm** | The field name entities.RandomContainerEnum.RandomContainer doesn't start with a lower case letter |
|  |  |
| **ODR** | controller.FiveCardStudPokerAjaxAction.getUserByUserName(String) may fail to close database resource on exception |
|  |  |
| **ODR** | controller.FiveCardStudPokerAjaxAction.register() may fail to close database resource on exception |
|  |  |
| **ODR** | controller.register.execute() may fail to close database resource on exception |
|  |  |
| **ODR** | deck.cards.Game.insertIntoGamesForGameComplete() may fail to close database resource on exception |
|  |  |
| **ODR** | entities.MySQLConnection.CreateSession(int, String, long) may fail to close PreparedStatement |
|  |  |
| **ODR** | entities.MySQLConnection.getSession(String) may fail to close PreparedStatement |
|  |  |
| **ODR** | entities.MySQLConnection.getUserData(String) may fail to close PreparedStatement |
|  |  |
| **ODR** | entities.MySQLConnection.revokeSession(int) may fail to close PreparedStatement |
|  |  |
| **ODR** | entities.MySQLConnection.revokeUserSession(int) may fail to close PreparedStatement |
|  |  |
| **RR** | encryption.DeleteMe\_AES.decrypt(char[], InputStream, OutputStream) ignores result of java.io.InputStream.read(byte[]) |
|  |  |
| **Se** | Class controller.Login defines non-transient non-serializable instance field servletResponse |
|  |  |
| **SnVI** | controller.AjaxAction is Serializable; consider declaring a serialVersionUID |
|  |  |
| **SnVI** | controller.handleSession is Serializable; consider declaring a serialVersionUID |
|  |  |
| **SnVI** | controller.Login is Serializable; consider declaring a serialVersionUID |
|  |  |
| **SnVI** | controller.register is Serializable; consider declaring a serialVersionUID |
|  |  |
| **SnVI** | controller.ShowGame is Serializable; consider declaring a serialVersionUID |
|  |  |

**Correctness Warnings**

|  |  |
| --- | --- |
| **Code** | **Warning** |
| **NP** | Possible null pointer dereference of ? in encryption.DeleteMe\_AES.decrypt(char[], InputStream, OutputStream) on exception path |
|  |  |
| **NP** | Possible null pointer dereference of ? in encryption.DeleteMe\_AES.encrypt(int, char[], InputStream, OutputStream) on exception path |
|  |  |
| **NP** | Possible null pointer dereference of ? in encryption.DeleteMe\_AES.keygen(int, char[], byte[]) on exception path |
|  |  |
| **NP** | Null passed for non-null parameter of java.io.OutputStream.write(byte[]) in encryption.DeleteMe\_AES.encrypt(int, char[], InputStream, OutputStream) |
|  |  |

**Experimental Warnings**

|  |  |
| --- | --- |
| **Code** | **Warning** |
| **OBL** | controller.FiveCardStudPokerAjaxAction.register() may fail to clean up java.sql.Statement on checked exception |
|  |  |
| **OBL** | controller.register.execute() may fail to clean up java.sql.Statement on checked exception |
|  |  |
| **OBL** | deck.cards.Game.insertIntoGamesForGameComplete() may fail to clean up java.sql.Statement on checked exception |
|  |  |
| **OBL** | entities.MySQLConnection.CreateSession(int, String, long) may fail to clean up java.sql.Statement |
|  |  |
| **OBL** | entities.MySQLConnection.getSession(String) may fail to clean up java.sql.ResultSet |
|  |  |
| **OBL** | entities.MySQLConnection.getSession(String) may fail to clean up java.sql.Statement |
|  |  |
| **OBL** | entities.MySQLConnection.getUserData(String) may fail to clean up java.sql.ResultSet |
|  |  |
| **OBL** | entities.MySQLConnection.getUserData(String) may fail to clean up java.sql.Statement |
|  |  |
| **OBL** | entities.MySQLConnection.revokeSession(int) may fail to clean up java.sql.Statement |
|  |  |
| **OBL** | entities.MySQLConnection.revokeUserSession(int) may fail to clean up java.sql.Statement |
|  |  |

**Internationalization Warnings**

|  |  |
| --- | --- |
| **Code** | **Warning** |
| **Dm** | Found reliance on default encoding in cipher.CipherUtils.decrypt(String): new String(byte[]) |
|  |  |
| **Dm** | Found reliance on default encoding in cipher.CipherUtils.encrypt(String): String.getBytes() |
|  |  |
| **Dm** | Use of non-localized String.toUpperCase() or String.toLowerCase() in controller.Login.execute() |
|  |  |
| **Dm** | Use of non-localized String.toUpperCase() or String.toLowerCase() in entities.MySQLConnection.getUserData(String) |
|  |  |

**Malicious code vulnerability Warnings**

|  |  |
| --- | --- |
| **Code** | **Warning** |
| **EI** | entities.User.getTimestamp() may expose internal representation by returning User.timestamp |
|  |  |
| **EI2** | new entities.User(int, String, String, Timestamp) may expose internal representation by storing an externally mutable object into User.timestamp |
|  |  |
| **EI2** | entities.User.setTimestamp(Timestamp) may expose internal representation by storing an externally mutable object into User.timestamp |
|  |  |
| **MS** | deck.cards.DeckOfCards.NUMBER\_OF\_CARDS isn't final but should be |
|  |  |
| **MS** | deck.cards.DeckOfCards.NUMBER\_OF\_CARDS\_PER\_SUIT isn't final but should be |
|  |  |
| **MS** | entities.RandomContainerEnum.reseedAfter isn't final but should be |
|  |  |
| **MS** | entities.RandomContainerEnum.sessionTimeout isn't final but should be |
|  |  |

**Security Warnings**

|  |  |
| --- | --- |
| **Code** | **Warning** |
| **Dm** | Hardcoded constant database password in controller.FiveCardStudPokerAjaxAction.getDatabaseConnection() |
|  |  |
| **Dm** | Hardcoded constant database password in deck.cards.Game.getDatabaseConnection() |
|  |  |
| **Dm** | Hardcoded constant database password in entities.MySQLConnection.getDatabaseConnection() |
|  |  |

**Dodgy code Warnings**

|  |  |
| --- | --- |
| **Code** | **Warning** |
| **DLS** | Dead store to $L1 in controller.AjaxAction.joinGame2() |
|  |  |
| **DLS** | Dead store to $L7 in controller.AjaxAction.joinGameNow() |
|  |  |
| **DLS** | Dead store to $L2 in controller.AjaxAction.raise() |
|  |  |
| **DLS** | Dead store to $L4 in controller.AjaxAction.waitForTurnOrGameToStart() |
|  |  |
| **DLS** | Dead store to $L4 in deck.cards.Game.insertIntoGamesForGameComplete() |
|  |  |
| **REC** | Exception is caught when Exception is not thrown in controller.AjaxAction.newGame() |
|  |  |
| **REC** | Exception is caught when Exception is not thrown in controller.AjaxAction.sum() |
|  |  |
| **REC** | Exception is caught when Exception is not thrown in controller.FiveCardStudPokerAjaxAction.getPlayerTurn() |
|  |  |
| **REC** | Exception is caught when Exception is not thrown in controller.FiveCardStudPokerAjaxAction.sum() |
|  |  |
| **ST** | Write to static field deck.cards.Game.uniqueSerialNumber from instance method new deck.cards.Game() |
|  |  |
| **UC** | Useless object stored in variable var$6 of method deck.cards.Game.insertIntoGamesForGameComplete() |
|  |  |
| **UC** | Useless object stored in variable var$7 of method deck.cards.Game.insertIntoGamesForGameComplete() |
|  |  |

After analyzing the findings from Find bugs we were able to reduce the number of findings on the report to just one. This warning was caused because all calluses that extend the strut2 action support class are Serializable. When implementing recommended struts 2 construct of implementing the ServletResponseAware interface, then creating a local private variable of HTTPServletResponse for storing the servlet response and creating setter and getters for struts to pass in the Servlet response we found HTTPservletRepsonse is not Serializable and thus produces the below warning.

**Bad practice Warnings**

|  |  |
| --- | --- |
| **Code** | **Warning** |
| **Se** | Class controller.Login defines non-transient non-serializable instance field servletResponse |
|  |  |

Since we found this issue not to be detrimental we then used the @SuppressFBWarnings annotation provided by the annotations library that comes with findbugs to tell findbugs to ignore the warning.

@SuppressFBWarnings(justification="No bug", value="SE\_BAD\_FIELD")

For the purpose of tracking the game state we stored the logged in users information along with the current game in in the java session state. This allowed us to avoid using query string for tracking game inputs and tracking results.

[Wikipedia 2015] bcrypt From Wikipedia, the free encyclopedia 2015-11-28

<https://en.wikipedia.org/wiki/Bcrypt>