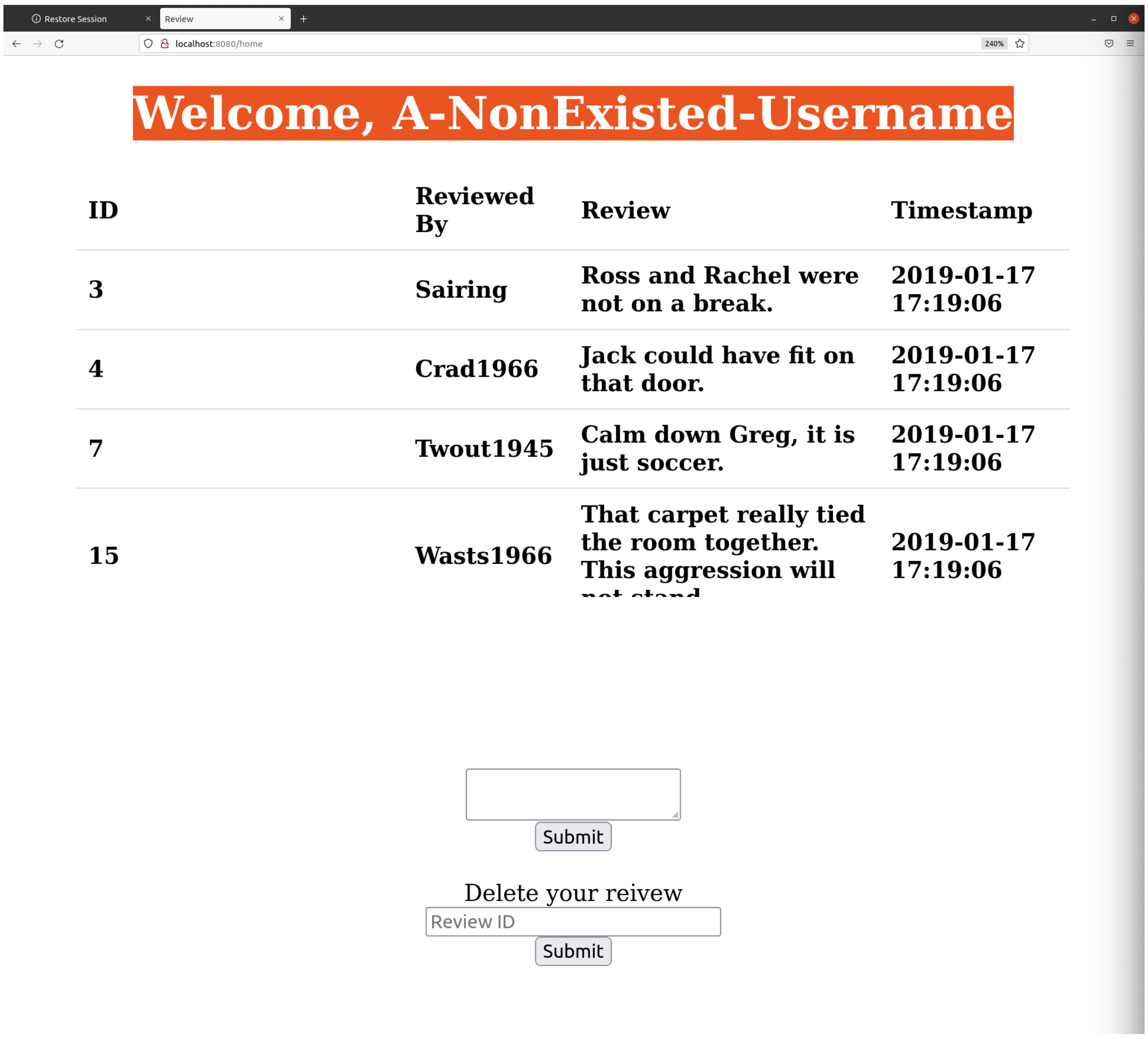
CS 542 – Introduction to Software Security
Exercise on Vulnerability Assessment using FPVA (Part2)

Binhao Chen (`bchen276@wisc.edu`), Steven Yang (`yang558@wisc.edu`)

Due: November 29 at 2:30pm.

1 Vulnerability 1

This screenshot shows how we attack the login system by exploiting the login SQL vulnerability. We can easily login with a username that does not exist in the user database at all.



Welcome, A-NonExisted-Username

ID	Reviewed By	Review	Timestamp
3	Sairing	Ross and Rachel were not on a break.	2019-01-17 17:19:06
4	Crad1966	Jack could have fit on that door.	2019-01-17 17:19:06
7	Twout1945	Calm down Greg, it is just soccer.	2019-01-17 17:19:06
15	Wasts1966	That carpet really tied the room together. This aggression will	2019-01-17 17:19:06

Delete your reivew

Review ID

1.1 Full details

The login function does not use a prepared statement to execute the query. To be able to attack the system, we can enter a password that, when inserted into the SQL query, will ensure that the WHERE clause is always satisfied. We can achieve it by passing in **' OR 'a' = 'a**. The first single quote will match the one in the query and allow us to introduce a new OR logic term. The statement ’a’ = ’a’ will always be true. Therefore, the WHERE clause is always true, the returned count will always be greater than 0 and the SQL query will always be executed without errors. Therefore, a user can login with any username even though it does not exist.

1.2 Cause

The root cause is that the programs parse the attacker’s input, which allows unexpected result to be generated. The attacker can close the previous single quote and add malicious input.

1.3 Proposed fix

We mitigate by using the Prepared Statement. We put two question mark placeholders for username and password in the query. Then, we set the input values with setString() function. In this way, the input is not parsed so that the user is not allowed to interact with the SQL query. The malicious input will only be treated as a strange string.

1.4 Actual Fix

As proposed. Below is the code scripts.

```java
package security.servlets;

import java.io.IOException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Cookie;
import javax.servlet.ServletException;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.util.UUID;

import security.helper.SqlQuery;
import security.helper.Config;
import security.helper.CookieHelper;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.PreparedStatement;

/**
 * This class handles the login logic in the /login route.
 *
 * @author kivolowitz
 */
public class LoginServlet extends HttpServlet {

    /*
     * (non-Javadoc)
     * This method is invoked whenever a get request is sent to jetty with a
        url of
     * /login.
     * It will check for existing cookies. If there are some and they are valid
        ,
     * then the user
     * is redirected to their home page. If the cookies are invalid or missing,
         they
     * are redirected
     * to sign in.
     *
     * @param HttpServletRequest request - request to be handled
     *
     * @param HttpServletResponse response - response to be returned
     */
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
        if (CookieHelper.checkCookies(request))
            response.sendRedirect("/home");
        else
            response.sendRedirect("/index.html");
    }

    /*
     * (non-Javadoc)
     * Most likely this servlet will be activated via a post request. The
         submit
     * button on index.html triggers a post
     * request which should have username and password data. If the username or
     * password are null or empty strings, the
     * user will be prompted to reattempt logging in.
     *
     * A simple sql query checks whether or not the user should be
         authenticated.
     * The database is stored in /WEB-INF/db/application.db.
     * The usernames and passwords are stored as plaintext in the USERS table
         of the
     * database.
     *
     * A successful login will create two cookies, a cookie named "username"
         with
     * the value being the username, and a
     * cookie named the value of username, with a random UUID as the session
         token.
     * Those together form the authentication
     * method for this application. The details of the session ID (the cookie
     * containing the UUID) will be written into
     * the server's filesystem under /WEB-INF/cookies/<username>.txt. Those
         cookies
     * are then added to the response and
     * returned to the user, redirecting them to /home which will then check
         the
     * validity of the cookies.
     *
     * @param HttpServletRequest request - request to be handled
     *
     * @param HttpServletResponse response - response to be returned
     */
    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response
        ) throws ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        if (username == null || username.equals("") || password == null ||
            password.equals("")) {
            response.sendRedirect("/index.html");
            return;
        }

        // String sqlQuery = "SELECT COUNT(*) AS count FROM USERS WHERE
            USERNAME == '"
        // + username + "' AND password == '" + password + "'";




        // This code block is how we did to mitigate the SQL Injection
            Vulunerability.
        // We use the PreparedStatement module by importing java.sql.
        // PreparedStatement; This will reserve spaces in the query statement
            for data input and make the SQL parse the original query without
            the input data from the user. And then later compare the input from
             the user with the database we have.
        // We put two question mark placeholders for username and password in
            the query. Then, we set the input values with setString().
        //In this way, the input is not parsed so that the user is not allowed
            to interact with the SQL query. The malicious input is simply a
            strange string.

        String DB_URL = "jdbc:sqlite:/home/user/Desktop/EXERCISES/5.0.1_FPVA/
            jetty/webapps/root/WEB-INF/db/application.db";
        PreparedStatement pstmt = null;
        try {
            Connection c = DriverManager.getConnection(DB_URL);
            pstmt = c.prepareStatement("SELECT COUNT(*) AS count FROM USERS
                WHERE USERNAME = ? AND password = ?");
            pstmt.setString(1, username);
            pstmt.setString(2, password);
        } catch (Exception e) {
        }

        boolean login = false;
        SqlQuery sql = new SqlQuery();

        try {
            // ResultSet results = sql.query(sqlQuery);
            // System.out.println(pstmt);

            ResultSet results = pstmt.executeQuery();
            String queryString = results.getStatement().toString();
            System.out.println(queryString);

            if (results.getInt("count") > 0)
                login = true;
        } catch (SQLException e) {
            request.setAttribute("error", e.toString());
            request.getRequestDispatcher("/WEB-INF/jsp/error.jsp").forward(
                request, response);
            return;
        }
        if (login) {
            try {
                Cookie cookieSession = new Cookie(username, UUID.randomUUID().
                    toString());
                Cookie cookieUsername = new Cookie("username", username);
                cookieSession.setMaxAge(Config.TIMEOUT);
                cookieUsername.setMaxAge(Config.TIMEOUT);
                response.addCookie(cookieSession);
                response.addCookie(cookieUsername);
                CookieHelper.writeCookie(cookieSession, username);
                response.sendRedirect("/home");
                System.out.println("Logged in successfully");
            } catch (Exception e) {
                request.setAttribute("error", e.toString());
                request.getRequestDispatcher("/WEB-INF/jsp/error.jsp").forward(
                    request, response);
                return;
            }
        } else {
            response.sendRedirect("/index.html");
        }
        sql.close();
    }
}
```

2 List of bugs

1. Any user can delete reviews owned by others without permission/authorization. In principle, users can only delete those reviews added by themselves.

3 Code exploration

Delete review and insert review do not lead SQL injection vulnerability since they have used PreparedStatment in the SQLQuery.java.