James Van Gilder, 9081186117

LoginServlet.java

Summary:

Use of unsanitized user input strings in SQL Queries allow for SQL Injection attacks allowing users to either log in without valid credentials or modify the USERS database. Using prepared statements would mitigate this type of attack.

Access Required:

Access to public login portal.

Effort Required: low

Low-level SQL knowledge could be used to access or modify the program.

Impact/Consequences: high

Users could log into the admin account, allowing modification to many files including the config and make files, which could be catastrophic.

Full Details:

Because an unprepared string is entered as a SQL query and parsed without sanitization, any SQL query input submitted by the user could result in an unwanted command being executed by the server on the USERS database.

Cause:

The cause of this is a failure to utilize prepared statements when using user input as a SQL Query modifier. This allows the user to input whatever they want and the server will parse the query without any safeties in place.

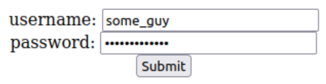
Proposed Fix:

Utilize prepared statements whenever user input is able to modify a SQL query.

Actual Fix:

Proposed fix.

Fix:



username: some_guy
password: •••••••••••••

password in this case is ‘ OR ‘x’ = ‘x and results in failed login but ‘his_password’ still results in correct login

LoginServlet.java

Summary:

In checking the presence of a valid username and password combination, the program returns a “True” login value as long as there are more than 0 matches in the database, but there should only ever be 1 matching value.

Access Required:

Access to public login portal.

Effort Required: low

Low-level SQL knowledge could be used to access or modify the program.

Impact/Consequences: high

Users could log into the admin account, allowing modification to many files including the config and make files, which could be catastrophic.

Full Details:

Because any number of returned matches in the database allows for login access, user input that results in multiple matches in the login portal grants login access. This is bad practice because in a valid login attempt, there should only ever be one (1) correct, valid match so there is no reason to allow more.

Cause:

The cause of this vulnerability is checking if there are more than 0 matches rather than checking if there is one match and one only.

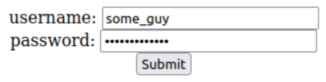
Proposed Fix:

Replace the check requirement ‘count > 0’ with ‘count == 1’.

Actual Fix:

Proposed fix.

Fix:


username: some_guy
password: •••••••••••••

password in this case is ‘ OR ‘x’ = ‘x and results in failed login but ‘his_password’ still results in correct login

LoginServlet.java

Summary:

When an exception is thrown during a login attempt, too much information is returned to users.

Access Required:

Access to public login portal.

Effort Required: low

Knowledge of how to force an error to be thrown in login.

Impact/Consequences: medium

Users could get more information on the code of this file than they are supposed to have.

Full Details:

Because the entire exception is casted to a string and printed whenever an exception is thrown in the login portal, a user who can force an exception can get more information on the workings of the login portal than likely desired by the designer.

Cause:

The catch statement on an exception being thrown by the login portal prints the entire exception rather than a controlled message.

Proposed Fix:

Replace the exception print with a standardized error message.

Actual Fix:

Proposed fix.

jetty

Summary:

Use of HTTP instead of HTTPS opens the entire site up to a massive number of attacks including but not limited to cross-site referencing forgery and XSS attacks as well as POST requests and HTTP requests.

Access Required:

Access to any part of the program.

Effort Required: low

Knowledge of CSRF attacks or any other HTTP specific vulnerabilities.

Impact/Consequences: high

Any part of the program could be modified in pretty much any way.

Full Details:

Not using a secure version of HTTP opens the software up to a great number of attack types that could be detrimental to the functionality and safety of the program and its users.

Cause:

Use of HTTP instead of the far superior HTTPS.

Proposed Fix:

Convert software to HTTPS instead of HTTP.

Actual Fix:

Proposed fix.

CookieHelper.java

Summary:

When an exception is thrown during a login attempt, too much information is returned to users.

Access Required:

Lowest level login access.

Effort Required: low

Knowledge of how to force an error to be thrown in the CookieHelper file.

Impact/Consequences: medium

Users could get more information on the code of this file than they are supposed to have.

Full Details:

Because the entire exception is casted to a string and printed whenever an exception is thrown in the cookie helper, a user who can force an exception can get more information on the workings of the cookie helper than likely desired by the designer.

Cause:

The catch statement on an exception being thrown by the cookie helper prints the entire exception rather than a controlled message.

Proposed Fix:

Replace the exception print with a standardized error message.

Actual Fix:

Proposed fix.

SubmitReview.java

Summary:

Review input by the user is not sanitized, allowing users to input javascript and execute commands on the server

Access Required:

Lowest level login access.

Effort Required: low

Knowledge javascript scripting

Impact/Consequences: high

Users could force the server to do their bidding

Full Details:

Because user input for the review is not sanitized, the user is allowed to input anything including javascript code that forces the server to execute commands that the user has input.

Cause:

The user input for “review” is not sanitized

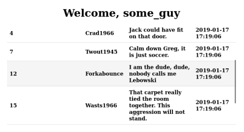
Proposed Fix:

Sanitize the user input

Actual Fix:

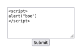
Proposed fix.

Fix:



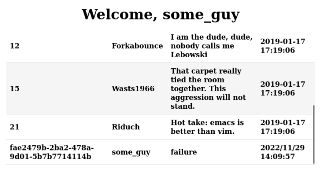
Welcome, some_guy

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
12	Forkabounce	I am the dude, dude, nobody calls me Lebowski	2019-01-17 17:19:06
15	Wasts1966	That carpet really tied the room together. This aggression will not stand.	2019-01-17 17:19:06



```
<script>
alert("boo")
</script>
```

submitting this javascript script will result in failure now



Welcome, some_guy

12	Forkabounce	I am the dude, dude, nobody calls me Lebowski	2019-01-17 17:19:06
15	Wasts1966	That carpet really tied the room together. This aggression will not stand.	2019-01-17 17:19:06
21	Riduch	Hot take: emacs is better than vim.	2019-01-17 17:19:06
fae2479b-2ba2-478a-9d01-5b7b7714114b	some_guy	failure	2022/11/29 14:09:57

LoginServlet.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 java.sql.PreparedStatement;
import java.sql.Connection;


import security.helper.SqlQuery;
import security.helper.Config;
import security.helper.CookieHelper;
/**
 * 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
    */
   private Connection c;
    private Statement statement;
    private static final String DB_URL =
"jdbc:sqlite:/home/user/Desktop/EXERCISES/5.0.1_FPVA/jetty/webapps/root/WEB-INF/db/application.db";

            @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;
            }
        PreparedStatement pstmt = null;
        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);

            boolean login = false;
            try{
                ResultSet results = pstmt.executeQuery();
                if(results.getInt("count") == 1)
                  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");
                }
                pstmt.close();
            }
}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 java.sql.PreparedStatement;
import java.sql.Connection;
import java.sql.DriverManager;

import security.helper.SqlQuery;
import security.helper.Config;
import security.helper.CookieHelper;
/**
 * 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
    */
   private Connection c;
   private static final String DB_URL =
"jdbc:sqlite:/home/user/Desktop/EXERCISES/5.0.1_FPVA/jetty/webapps/root/WEB-INF/db/application.db";

            @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;
                }
          PreparedStatement pstmt = null;
          try {
              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(SQLException e) {
              System.out.println("failed parse");
          }
                boolean login = false;
                try{
                        ResultSet results = pstmt.executeQuery();
                        if(results.getInt("count") == 1)
                          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");
            }
      try {
                pstmt.close();
      } catch(SQLException e) {
        System.out.println("failed to close prepared statement!");
      }
      try {
        c.close();
      } catch(Exception e) {
        System.out.println("failed to close connection!");
      }
        }
}
```

SubmitReviewServlet.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.ServletException;
import java.sql.SQLException;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.UUID;

import security.helper.SqlQuery;
import security.helper.CookieHelper;
/**
 * Simple servlet to handle the insertion of a new review into the database.
 * This handles the /submitreview route and should either redirect to /index.html
 * or /home
 *
 * @author Evan Kivolowitz
 */
public class SubmitReviewServlet extends HttpServlet {

        @Override
        public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
          // Error checking
          if(!CookieHelper.checkCookies(request)) {
                System.out.println("Cookies are invalid from SubmitReview Post Request");
                response.sendRedirect("/index.html");
                return;
          }

          String username = CookieHelper.getUsernameCookieName(request);
          if(username == null) {
                System.out.println("Session is not null but username is");
                response.sendRedirect("/index.html");
                return;
          }

          String review = (String) request.getParameter("review");
      review.replaceAll("(?i)<script.*?>.*?</script.*?>", "failure");
      review.replaceAll("(?i)<.*?javascript:.*?>.*?</.*?>", "failure");
          if(review.equals("")) {
                response.sendRedirect("/home");
                return;
          }
          // end error checking

          // Gathers the four elements of a review and sends it to the insert method of
          // the sql class. Upon success, or failure, the user is silently redirected to home.
          String id = UUID.randomUUID().toString();
```

```
                DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
                String sqlQuery = "INSERT INTO REVIEWS VALUES (?,?,?,?)";
                String[] values = {id, username, dateFormat.format(new Date()),
review.replaceAll("(?i)<script.*?>.*?</script.*?>", "failure")};

                SqlQuery sql = new SqlQuery();
                try{
                        sql.insert(sqlQuery, values);
                        response.sendRedirect("/home");
                } catch (SQLException e) {
                        request.setAttribute("error", e.toString());
                        request.getRequestDispatcher("/WEB-INF/jsp/error.jsp").forward(request, response);
                }
                sql.close();

             }
}
```

CookieHelper.java

```
package security.helper;
import java.io.File;
import java.io.PrintWriter;
import java.util.Scanner;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;

/**
 * Class full of static methods to make dealing with cookies easier.
 * Contains methods for writing cookies to files, checking if they're valid,
 * getting the username cookie, and checking the cookies to see if the session is valid.
 *
 * @author kivolowitz
 */

public class CookieHelper {

  // Constant declaring the conversion between milliseconds and seconds.
  // System time is in milliseconds while cookie timeouts are in seconds.
  public static final long MS_TO_SEC = 1000l;

  /*
   * (non-Javadoc)
   *
   * Write cookie is a helper function to write the contents of the cookie to
   * a file for the server to read. Cookies are stored in the /WEB-INF/cookies/
   * directory with the filename being the current user's username. For example,
   * if Cher, Bono, and Enya were the only three users using the site, the cookies
   * directory would look like
   * cookies/
   * ├── Bono.txt
   * ├── Cher.txt
   * └── Enya.txt
   *
   * The three components of a cookie that we use are: Name|Value|Expiration
   *
   * @param Cookie cookie - cookie to write to the server.
   * @param String username - username of the current user to create the cookie file.
   */
  public static void writeCookie(Cookie cookie, String username) {
        try {
          File f = new File("webapps/root/WEB-INF/cookies/" + username + ".txt");
          if(!f.exists())
                f.createNewFile();

          PrintWriter writer = new PrintWriter(f);
          long currTime = System.currentTimeMillis() / MS_TO_SEC;
          long expirationTime = (long) cookie.getMaxAge() + currTime;

          // Pipes are used to delimit the three important components of a cookie in the text
          // file.
          writer.println(cookie.getName() + "|" + cookie.getValue() + "|" + expirationTime);
          writer.close();
```

```
            } catch (Exception e) {
               System.out.println("Failed to write");
            }
   }

   /*
    * (non-Javadoc)
    * This is a simple method that checks if a cookie is valid. By valid,
    * that means that the cookie sent by the request matches the corresponding
    * 1st value in the file named cookies/<username>.txt, as well the expiration date
    * being greater than the current time. It is unlikely that you will get an
    * invalid due to expiration, as the browser deletes the cookie when it expires.
    *
    * @param Cookie cookie - the cookie to be checked
    * @param String username - username of the client to be checked
    * @return boolean True if valid, otherwise False
    */
   public static boolean isCookieValid(Cookie cookie, String username) {
            // cookie values are client side
            boolean valid = false;
            try {
               File f = new File("webapps/root/WEB-INF/cookies/" + username + ".txt");
               Scanner sc = new Scanner(f);
               String line = sc.nextLine();
               sc.close();
               System.out.println(line);
               String[] values = line.split("\\|");
               String cookieValue = values[1];
               long expirationDate = Long.parseLong(values[2]);

               if(expirationDate > System.currentTimeMillis() / MS_TO_SEC &&
cookie.getValue().equals(cookieValue))
                      valid = true;
               if(expirationDate < System.currentTimeMillis() / MS_TO_SEC)
                      System.out.println("Invalid due to expiration");
               if(!cookie.getValue().equals(cookieValue))
                      System.out.println("Invalid due to invalid cookie value");
            } catch(Exception e) {
               System.out.println("Cookie validation failed!");
            }
            return valid;

   }

   /*
    * (non-Javadoc)
    * This method simply finds a cookie named "username" and returns its value.
    * @param HttpServletRequest request - request with the cookies attached
    * @return String username if the cookie exists, otherwise null.
    *
    */
   public static String getUsernameCookieName(HttpServletRequest request) {
            Cookie[] cookie = request.getCookies();
            if(cookie != null) {
               for(Cookie c : cookie) {
                      if(c.getName().equals("username"))
                          return c.getValue();
               }
            }
            return null;
   }

   /*
    * (non-Javadoc)
    * This method checks all the cookies for a request to see if there is a valid session
    * cookie. To enforce this, it finds the username of the logged in user (via the cookie "username").
    * Then the method will look for a cookie with the name in cookieUsername and check if that cookie
    * is valid. It is important to default to returning false. If there is no username, this will return
    * false.
    *
    * @param HttpServletRequest request - request with the cookies attached to it.
    * @return boolean True iff the session cookie "username" has a valid corresponding valid session.
    *         Otherwise false.
    */
   public static boolean checkCookies(HttpServletRequest request) {
            Cookie[] cookies = request.getCookies();
            String sessionUsername = getUsernameCookieName(request);
            if(sessionUsername == null)
               return false;
            if(cookies != null) {
               System.out.println("Cookies length: " + cookies.length);
               for(Cookie c : cookies) {
                      String cookieUsername = (String) c.getName();
                      if(sessionUsername.equals(cookieUsername)) {
                          if(CookieHelper.isCookieValid(c, sessionUsername)) {
                              System.out.println("checkCookies cookie is valid " + c.getName());
                              return true;
                          }
                      }
                      else System.out.println("Cookie is invalid: " +
                                      c.getName() + " " + c.getValue());
               }
            }
            if(cookies == null) System.out.println("Cookies were null");
            return false;
   }

}
```

List of Bugs:

- Any user can delete any review
- Submitting a poorly formatted javascript script to a review will force a log out
-

Code Exploration:

I think that the code would be much more secure if fewer helpers were used and instead helper methods were written within the servlets, allowing fewer options for attackers to intercept or modify requests, especially considering the use of HTTP, which I will now address. By using HTTP instead of HTTPS, the program and server as well as all users are significantly less safe and secure than they could be by converting. This is because with HTTP, CSRF, XSS, and other types of attacks like ZAP and POST/HTTP request type attacks are possible. Also, I thought it was an interesting choice to use the XML files in the way that they were used.