Homework 4

Java Web Application with Corrected Security Issues

Student: Patrick Walsh

School: University of Maryland Global Campus

Course: SDEV 425 6980

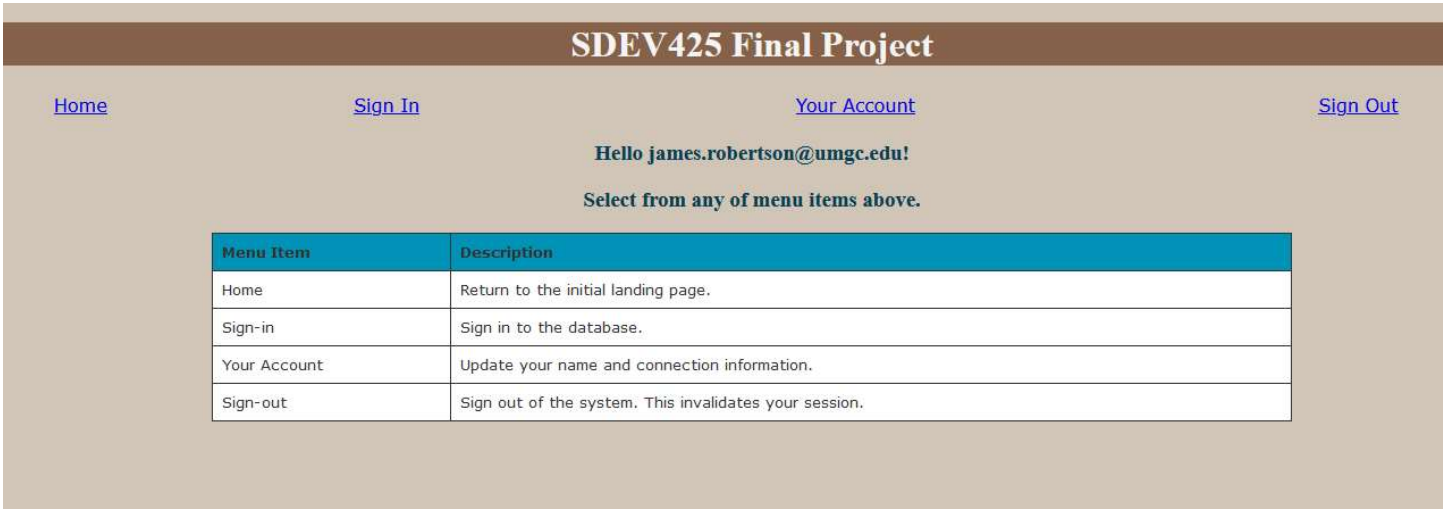
Date: 8/5/2021

Professor: Dr. Nicholas Duchon

Running the Web Application

I setup the Java web application using NetBeans and Glassfish version 4. I had to add the Derby drivers to my project Library, then I was able to successfully launch the application in a web browser from http://localhost:8080/SDEV425_HW4/index.jsp. To make sure my web application was communicating with the relational database and that the basic functionality of the website was working, I logged in with one of the users with username *james.robertson@umgc.edu* and password *mypassword (I changed this password later):*

Here is the successful login screen:



Checking the Web Application for PCI Compliance

PCI Requirement 3.2

The web application violates a number of PCI compliance protocols. For example, it violates PCI requirement 3.2 for storing sensitive credit card holder data, including the Full Track Data, CAV2, and PIN (PCI Security Standards Council, 2015a, p.15). As stated in PCI requirement 3.2: "Do not store sensitive authentication data after authorization (even if it is encrypted). Render all sensitive authentication data unrecoverable upon completion of the authorization process" (p. 14).

To meet this requirement, I modified the web page, the Java file, and the database files. Specifically, I took out references to Full Track Data, CAV2, and PIN so that these values were not being stored. I made these changes in the following files: ShowAccount.java, createHW4Tables_secureVersion.sql, and account.jsp. Below are examples of these changes:

From ShowAccount.java

```
63
64              // Set the Attribute for viewing in the JSP
65              request.setAttribute("Cardholdername", Cardholdername);
66              request.setAttribute("CardType", CardType);
67              request.setAttribute("ServiceCode", ServiceCode);
68              request.setAttribute("CardNumber", CardNumber);
69  //            request.setAttribute("CAV_CCV2", CAV_CCV2);  // Cannot store per PCI requirement 3.2
70              request.setAttribute("expiredate", expiredate);
71  //            request.setAttribute("FullTrackData", FullTrackData);  // Cannot store per PCI requirement 3.2
72  //            request.setAttribute("PIN", PIN);  // Cannot store per PCI requirement 3.2
73
74              RequestDispatcher dispatcher = request.getRequestDispatcher("account.jsp");
75              dispatcher.forward(request, response);
```

From createHW4Tables_secureVersion.sql:

```sql
-- Account data
CREATE TABLE CustomerAccount (
  account_id INTEGER Primary Key,
  user_id INTEGER NOT NULL references sdev_users (user_id),
  Cardholdername VARCHAR(75) NOT NULL,
  CardType VARCHAR(20) NOT NULL,
  ServiceCode VARCHAR(20) NOT NULL,
  CardNumber VARCHAR(30) NOT NULL,
--   CAV_CCV2 INTEGER NOT NULL, --Cannot store per PCI requirement 3.2
  expiredate date NOT NULL
--   FullTrackData varchar (75) Not NULL,  --Cannot store per PCI requirement 3.2
--   PIN varchar(10) NOT Null  --Cannot store per PCI requirement 3.2
);
```

From account.jsp:

```jsp
<%-- Gather the data from the servlet--%>
<% String UserEmail = (String) session.getAttribute("UMUCUserEmail");%>
<% String user_id = (String) session.getAttribute("UMUCUserID").toString();%>
<% String Cardholdername = (String) request.getAttribute("Cardholdername");%>
<% String CardType = (String) request.getAttribute("CardType");%>
<% String ServiceCode = (String) request.getAttribute("ServiceCode");%>
<% String CardNumber = (String) request.getAttribute("CardNumber");%>
<%-- <% String CAV_CCV2 = (String) request.getAttribute("CAV_CCV2").toString(); %> Cannot store per PCI requirement 3.2 --%>
<% String expiredate = (String) request.getAttribute("expiredate").toString();%>
<%-- <% String FullTrackData = (String) request.getAttribute("FullTrackData");  %> Cannot store per PCI requirement 3.2 --%>
<%-- <% String PIN = (String) request.getAttribute("PIN");  %> Cannot store per PCI requirement 3.2 --%>

<form action="ContinueIt" method="post">
```

Now, when the user logs in and clicks on the Your Account page, this is the information that is displayed:

SDEV425 Final Project

Home Sign In Your Account Sign Out

Account Data

Email:	james.robertson@umgc.edu
User ID:	1
Card Holder Name:	James Robertson
Card Type:	MasterCard
Service Code:	27aD
Card Number:	1111111111111
Expire Date:	2016-02-23

PCI Requirement 3.1

This requirement states that credit card data storage should have limits and should be purged from databases when no longer needed for business purposes (PCI Security Standards Council, 2015a). As stated in 3.1, “Limit cardholder data storage and retention time to that which is required for business, legal, and/or regulatory purposes, as documented in your data retention policy. Purge unnecessary stored data at least quarterly” (p. 14).

To meet this requirement, I added a current date stamp that automatically includes today’s date in the database when a new account record is added. When the web page loads a user’s account information, this storage date information is also shown. This allows database administrators to keep track of user accounts and purge old accounts that are no longer in use or are past a certain create date. See screen shots below.

From createHW4Tables_secureVersion.sql:

```
-- Insert CustomerAccount
insert into CustomerAccount (account_id, user_id,
CardType, ServiceCode, CardNumber, Cardholdername, expiredate, storedate)
values (1,1,'MasterCard','27aD','1111111111111','James Robertson','02/23/2016', CURRENT_DATE);

insert into CustomerAccount (account_id, user_id,
CardType, ServiceCode, CardNumber, Cardholdername, expiredate, storedate)
values (2,2,'Visa','34q4','222222222222','Test Administrator','09/16/2018', CURRENT_DATE);

insert into CustomerAccount (account_id, user_id,
CardType, ServiceCode, CardNumber, Cardholdername, expiredate, storedate)
```

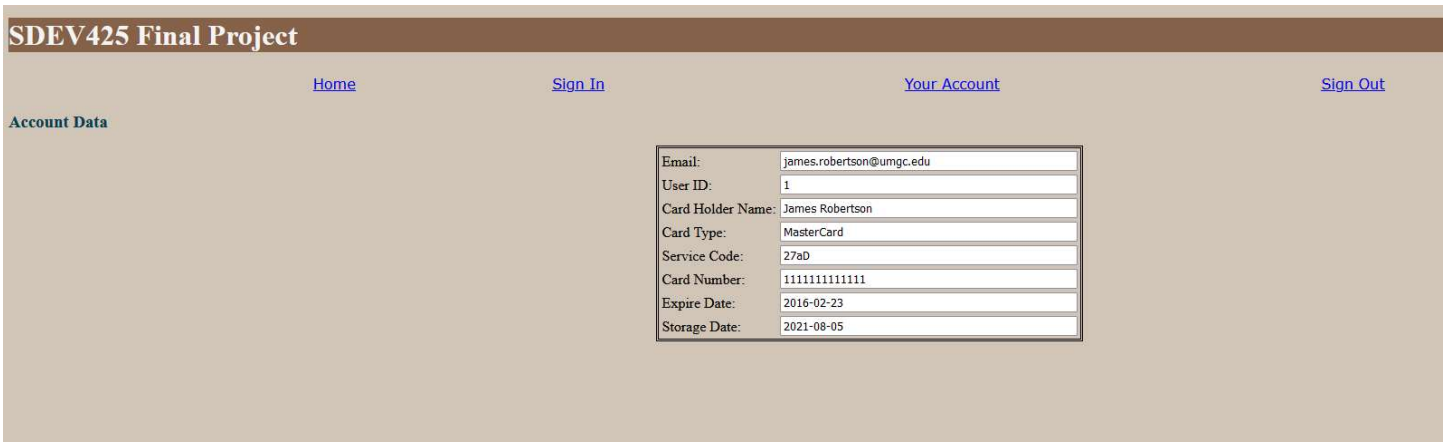
From ShowAccount.java:

```
            // Set the Attribute for viewing in the JSP
            request.setAttribute("Cardholdername", Cardholdername);
            request.setAttribute("CardType", CardType);
            request.setAttribute("ServiceCode", ServiceCode);
            request.setAttribute("CardNumber", CardNumber);
//            request.setAttribute("CAV_CCV2", CAV_CCV2);  // Cannot store per PCI requirement 3.2
            request.setAttribute("expiredate", expiredate);
            request.setAttribute("storedate", storedate);  // date when record was stored for PCI compliance of 3.1
//            request.setAttribute("FullTrackData", FullTrackData);  // Cannot store per PCI requirement 3.2
```

From account.jsp:

```
31          <% String CardType = (String) request.getAttribute("CardType");%>
32          <% String ServiceCode = (String) request.getAttribute("ServiceCode");%>
33          <% String CardNumber = (String) request.getAttribute("CardNumber");%>
34          <%-- <% String CAV_CCV2 = (String) request.getAttribute("CAV_CCV2").toString(); %
35          <% String expiredate = (String) request.getAttribute("expiredate").toString();%>
36          <% String storedate = (String) request.getAttribute("storedate").toString();%>
37          <%-- <% String FullTrackData = (String) request.getAttribute("FullTrackData");  %
```

Now, when a user logs in and view the account information, the storage date is displayed:



PCI Requirement 2.1

This requirement specifies that default, vendor-supplied passwords should be changed prior to systems being implemented to production (PCI Security Standards Council, 2015a). As stated in the requirement: “always change ALL vendor-supplied defaults and remove or disable unnecessary default accounts before installing a system on the network” (p. 13).

To meet this PCI requirement, I changed the default passwords and entered a more secure password that includes a combination lowercase letters, uppercase letters, numbers, and special characters. I made this change in the database.

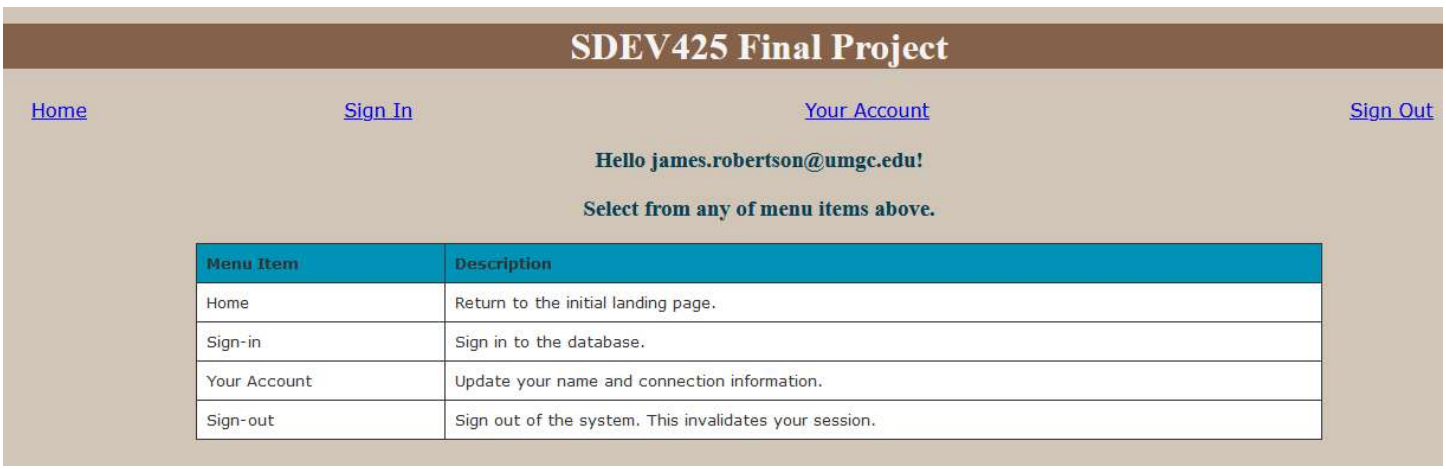
From createHW4Tables_secureVersion.sql:

```
--Insert user_info
insert into user_info (user_id, password)
-- values (1,'mypassword');  changed default password per PCI requirement 2.1
values (1,'mypasswordSecure99&*');

insert into user_info (user_id, password)
-- values (2,'adminpasstest');  changed default password per PCI requirement 2.1
values (2,'adminpasstestCompl3x31!@');

insert into user_info (user_id, password)
-- values (3,'customerpasstest');  hanged default password per PCI requirement 2.1
values (3,'customerpasstestSecrett63%@');
```

I successfully logged into the web application with this new password:



PCI Requirement 6.5

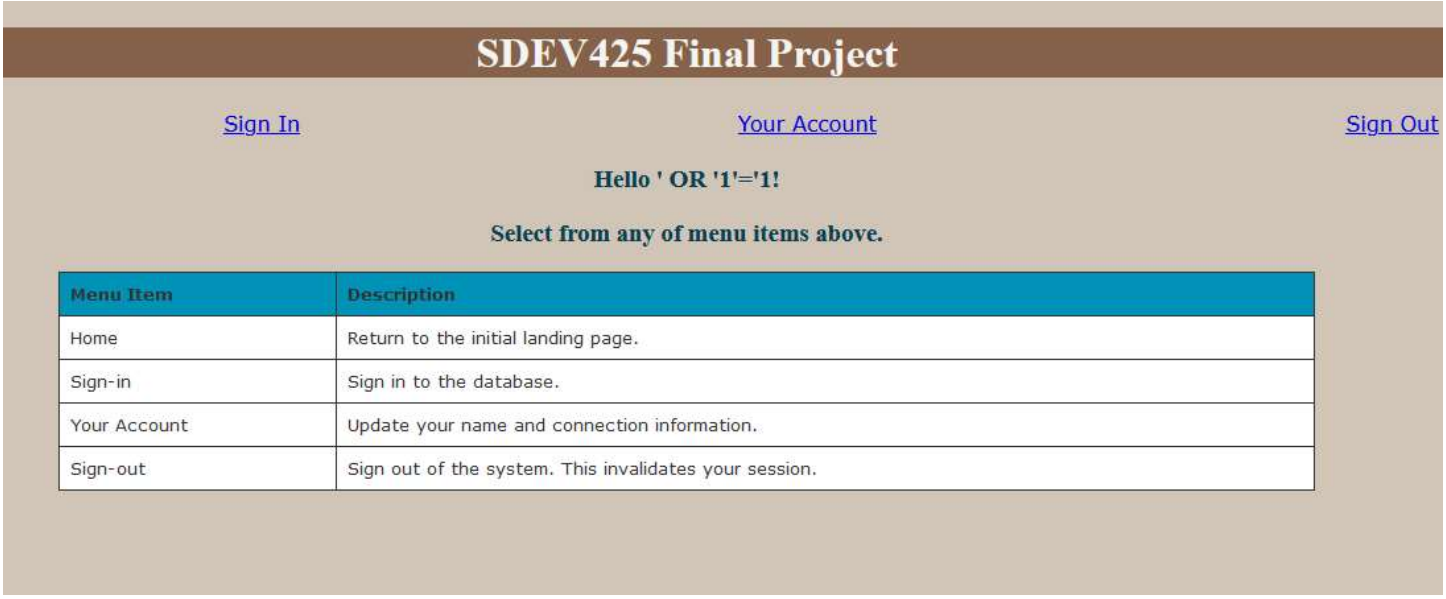
This requirement states that software should be written to prevent common software security vulnerabilities from being exploited (PCI Security Standards Council, 2015a). The PCI Security Standards Council further specifies (2015b) that SQL injections are a common software vulnerability that can be exploited when a software application accepts user-input (p. 59).

To meet this requirement, I modified the Java file to validate user input through a prepared statement that prevents a malicious query from bypassing the user login page. For example, the following string used for the email and password of the login will execute an SQL injection:

' OR '1'='1

On the login page, this query can be used to login:

Having the OR 1=1 portion inserts an SQL statement that always returns true, thus allowing a malicious user to gain access:



To prevent this, I modified the SQL query to include a prepared statement. The original statement from Authenticate.java reads:

```
// OLD CODE
// Vulnerable to SQL injection attack, violating PCI requirement 6.5
Statement stmt = conn.createStatement();
String sql = "select user_id from sdev_users where email = '" + this.username + "'";
ResultSet rs = stmt.executeQuery(sql);
// END OF OLD CODE
```

The user_id and password are passed as plain text in the original code:

```
// OLD CODE
// Vulnerable to SQL injection attack, violating PCI requirement 6.5
String sql2 = "select user_id from user_info where user_id = " + user_id + "and password = '" + this.pword + "'";
ResultSet rs2 = stmt.executeQuery(sql2);
// END OF OLD CODE
```

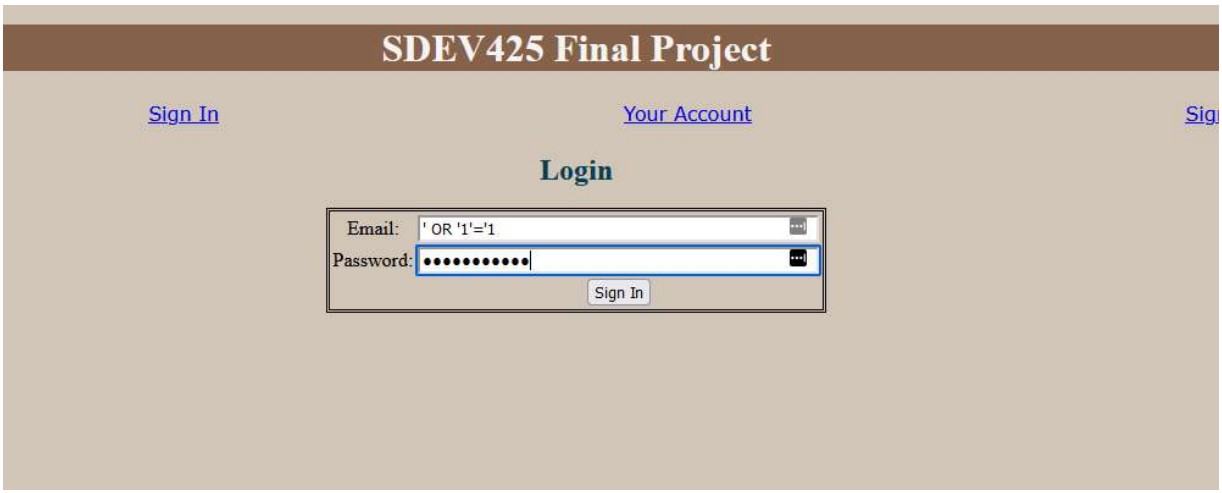
These queries do not validate user input. The modified version of these statements is seen below:

```
// NEW CODE
// Prevents SQL injection attack
String sql = "select user_id from sdev_users where email = ?";
PreparedStatement stmt = conn.prepareStatement(sql);
stmt.setString(1, this.username);
ResultSet rs = stmt.executeQuery();
// END OF NEW CODE
```

The user_id and password are now passed through a prepared statement which prevents SQL injections:

```
// NEW CODE
// Prevents SQL injection attack
String sql2 = "select user_id from user_info where user_id = ? and password = ?";
PreparedStatement stmt2 = conn.prepareStatement(sql2);
stmt2.setInt(1, user_id);
stmt2.setString(2, this.pword);
ResultSet rs2 = stmt2.executeQuery();
// END OF NEW CODE
```

When I use the secure statements and try to use the same SQL injection attack, the attack fails. See the screenshots below:



The usernames and passwords are seen below:

User email	Password
james.robertson@umgc.edu	mypasswordSecure99&*
test.admin@umgc.edu	adminpasstestCompl3x31!@
test.customer@umgc.edu	customerpasstestSecrett63%@

References

PCI Security Standards Council. (2015a, May). *PCI DSS Quick Reference Guide* (3.1).

PCI Security Standards Council. (2015b, April). R*equirements and Security Assessment Procedures* (3.1).