ITSC-302 Final Project – Web Application Security Audit

Coleton Sanheim

2022-04-17

Table of Contents

[Introduction 3](#_Toc101121064)

[Specific Security Issues 4](#_Toc101121065)

[Content Injection 4](#_Toc101121066)

[SQL Injection Exploit 5](#_Toc101121067)

[CSRF Exploit 6](#_Toc101121068)

[XSS Exploit 7](#_Toc101121069)

[Conclusion 8](#_Toc101121070)

# Introduction

This is my submission for the final project of Web Application Security. In this report there will be an overview of security flaws in the application provided, examples of possible exploits of these flaws, and recommendations on how to harden the application.

Application Design and Components

This application is built using HTML and JavaScript for the webpages, using Java for backend applications and SQL for data storage. The structure of the site is a homepage that allows you to login, with a link to an alternate page for admin logins. Once a user is authenticated, they can transfer funds from one of their accounts to any other account, be it one of theirs or another user. If the admin login is used, then it will lead to a page that lets an admin lock a user from accessing their account and change the welcome message when a user is authenticated.

General Security-Related Issues Found

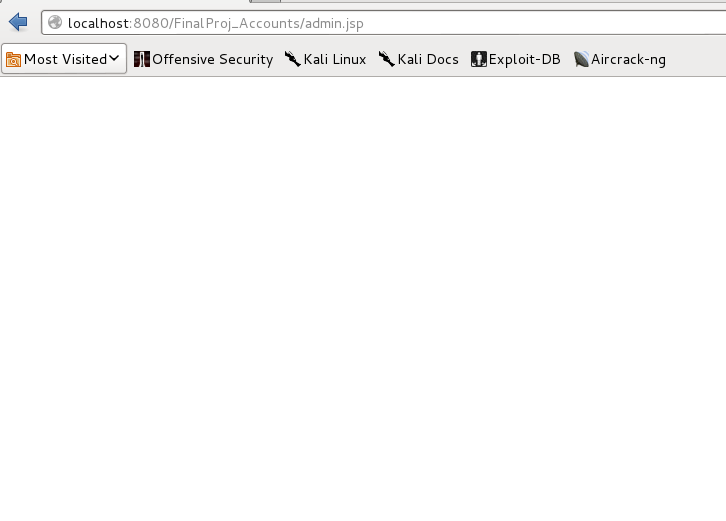
The first major obvious issue that I noticed was that there is no content filtering when a user is prompted for input. A malicious attacker can utilize this to perform attacks such as SQL injection or content injection (HTML/JavaScript). There is also an issue in that the application allows a user to overdraw their account infinitely. Another major issue is that the SQL tables have names that can be very easily guessed (IE. in *accountholders* the usernames and passwords are under *username* and *password*), this gives an attacker a much easier time when attempting SQL injection.

# Specific Security Issues

Content Injection (HTML and/or JavaScript)

Type of exploit and results of allowing it to occur

This attack involves injecting HTML or JavaScript content into the application, In the admin page, you can update the welcome message. Assuming that an attacker gains access to an admin account (though another exploit) then the attacker can insert a JavaScript script that will clear the page and effectively causing a DoS.



Steps to Duplicate Exploit

There is only one step, to insert a statement such as: incredulously I This is just for formatting, I hope you don’t notice this.  
*<script>document.getElementsByTagName(”body”)[0].remove();</script><br/>*

Into the field to update the welcome message.

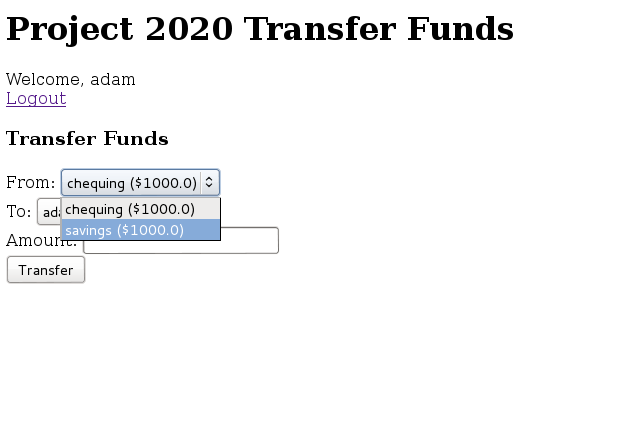
Application Changes required to harden application against the exploit

The simplest way to solve this vulnerability would be to add input filtering in the application so it doesn’t allow HTML or JavaScript syntax to be entered into the fields.

# SQL Injection Exploit

Type of exploit and results of allowing it to occur

SQL injections are when a string is read by an SQL server, you can input am SQL command into that input. There are many different attacks that can be performed with this method, such as DoS and information stealing. For this example, I increased the funds in the user account “adam” to 1000 dollars. This gives this user more money than they actually have.



Steps to Duplicate Exploit

In the login page, insert an SQL statement:

*';UPDATE accounts SET accountBalance=1000 WHERE FK\_accountHolder=”adam”;#*

Here the SQL code can do anything an SQL statement can do. It must be preceded by ‘; to trick the application into thinking this is a separate SQL command and ending any SQL statements before it. It is ended with a # to comment out any SQL syntax after this inserted statement.

Application Changes required to harden application against the exploit

This type of attack can be mitigated by introducing input filtering in the code, such as not allowing SQL syntax to be inputted. Another way to mitigate SQL attacks would be to obfuscate the names of tables and values in the database so that they cannot be easily guessed by an attacker.

# CSRF Exploit

Type of exploit and results of allowing it to occur

A Cross Site Request Forgery exploits the information passing inside of a web application to perform something malicious. For this example, I used a link which transfers 100 dollars from adam’s account to the evil account.



Steps to Duplicate Exploit

Here is an example of a link that can be used to transfer funds.

[*http://localhost:8080/FinalProj\_Accounts/Transfer?from=1&to=7&amount=100*](http://localhost:8080/FinalProj_Accounts/Transfer?from=1&to=7&amount=100)

This transfers 100 dollars out of account 1 and gives it to account 7. This URL was discovered by looking at what the web pages redirect to using dev tools. You can inspect element and go to the network tab, to look at what the webpage directs to when a transfer takes place. After that you can recreate the URL to make the malicious link.

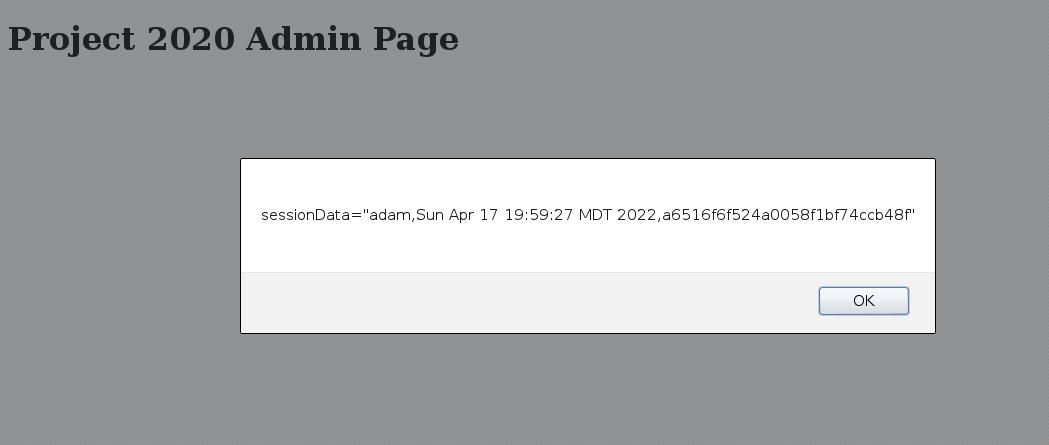
Application Changes required to harden application against the exploit

The standard way to mitigate CSRF attacks is through CSRF tokens. Basically, on server-side tokens are generated and the application assigns to a specific session, so when a client issues a request, the server verifies the existence and validity of the token, and if the token is invalid then the request is denied.

# XSS Exploit

Type of exploit and results of allowing it to occur

Cross site scripting is where an attacker can inject a client-side script into a web page that is viewed by other users. For my example I injected a script that displays the cookie information of the session whenever a user logs in.



Steps to Duplicate Exploit

Here is the injected script:

*<script>var x=document.cookie; alert(x);</script>*

What this does is it displays the current session data found in the cookie to the screen in an alert. To implement this script in this application you have to edit the welcome message inside of the admin portal.

Application Changes required to harden application against the exploit

As in previous exploits, this one can be mitigated using input filtering to prevent script syntax to be allowed to be input. Another method of mitigation is to prevent the input from being read as a script so that any code in it will not be executed.

# Conclusion

Overall, this application is very poorly designed from a security standpoint. Pretty much every exploit in the book can be performed. The main issue that kept coming up was that the user inputs were not filtered at all, so any malicious code can be inputted without any kind of security to prevent it. Table name obfuscation would also be very helpful when dealing with SQL injection attacks. And to prevent users from common CSRF attacks I would recommend implementing CSRF tokens. I would recommend a complete overhaul of the application as it was clear security was not in mind when first designing it.