**Cyber Pentest Project (Penetration Testing Research)**

Group Members: Robert Yelas, Nelson Kangethe

Teaching Assistant: Cheng Zhen

Instructor: Kirsten Winters

CS 462

March 10, 2023

Penetration Testing Research using a Web Application Programming Interface (API):

# Report of Known Vulnerabilities and Effective Countermeasures

## Project Description

This project involves building a simple web app (faux bank account, or similar), trying to attack the app in various ways, and hardening the app to prevent these attacks. This report includes a list of the app’s known vulnerabilities and countermeasures that have been implemented to handle these attacks.

The app must meet several requirements. The app must have user accounts and authentication, of course. It must have user input, a database, and whatever else is needed to offer a target for attacks. The output from this research project is a GitHub repo with all the code and a How-to write up for each of the attacks and mitigations. This report primarily (along with other supporting documentation) constitutes this write up. Enough detail should be present so that anyone can reproduce the results.

The final demonstration of the project should have a site or sites that are easy to test. You could have 2 different versions of the site running on a server: One that is vulnerable to attacks, and one that is not. Or you could have one site with options that turn on or off vulnerabilities. The choice is up to you. Note: You may expand upon an existing vulnerable web app such as this one: http://www.dvwa.co.uk/ or google for another similar site.

Requirements:

1. Perform penetration testing on the infrastructure of the web app for 3-4 of the top ten attacks (linked below).

2. Learn from each attempt at pen testing, and harden the web app accordingly.

3. Create a writeup for the attack and the solution and add to the GitHub repo how-to.

4. Repeat 1-3 until done.

***Stretch Goal:***

The app will have a database of passwords hashed using different encryption algorithms with various levels of difficulty. The goal is to steal and crack the passwords. attempt to steal the password hashes and attempt to break the cryptography, the idea being to see how to correctly implement cryptography and to see how easily different hashing algorithms are broken/ very difficult to impossible to break create a writeup for the password cracks and what worked or did not work for prevention, and add to the GitHub repo how-to.

## Current Security Vulnerabilities

**Password storage**

✅ The password is stored in the database in plain text, which is a security vulnerability. In case the database gets compromised, the attacker will have access to all the passwords. You should store the passwords securely, for example, using a hash function and a salt.

**Brute-force protection / IP rate-limiting**

✅ The current implementation provides a 5-second countdown timer in case of a failed login attempt, but this is not enough to protect against brute-force attacks. A better solution would be to implement rate limiting or IP blocking after several consecutive failed attempts.

**Input validation**

✅ The password checker only checks the length of the password and if it contains both uppercase and lowercase letters. It does not check for other requirements such as the presence of numbers or special characters, which are commonly required for strong passwords.

**Cross-Site Scripting (XSS)**

✅ The application does not sanitize user input, which leaves it vulnerable to XSS attacks. An attacker could inject malicious JavaScript into the application, which would execute in the user's browser. To prevent XSS, you should sanitize user input by escaping special characters or using a library that automatically escapes user input.

**Cross-Site Request Forgery (CSRF)**

✅ The application is vulnerable to CSRF attacks. An attacker could trick a user into submitting a malicious request, such as a request to change the password. To prevent CSRF, you should include a unique token in each form and validate it on the server.

**SQL Injection**

✅ The application is vulnerable to SQL injection attacks. An attacker could inject malicious SQL statements into the application, which would execute in the database. To prevent SQL injection, you should use parameterized queries or a library that automatically escapes user input.

Works Cited

AuthorLastName, FirstName. *Title of the Book Being Referenced*. City Name: Name of Publisher, Year. Type of Medium (e.g. Print).

LastName, First, Middle. "Article Title." *Journal Title* (Year): Pages From - To. Print.