**CSGY – Assignment 3 – DB Security**

Luna Koval – [rak529@nyu.edu](mailto:rak529@nyu.edu)

2019/11/10 v0.1

**Summary** - Report on the development, testing, and bug fixes in adding db interactions to the website in assignment 2.

**Task** - Add user login tracking to the user database backend, as well as reporting of history. The goal is to test for and protect against sql injection.

1. **Overview**
   1. **Repository**

<https://github.com/lak529/cybersec-csgy9163-assign_03>

* 1. **Setup**

The setup was the same as in assignment 2, with the addition of SQLMap for sql injection testing, and MySQL for a backend database. VirtualBox 6, Ubuntu 18.02, python3.6, pip3, flask, venv, flask-wtf/sqlalchemy/login, requests, unittest, BeautifulSoup, SQLMap.

* Install
  + flask-wtf
  + flask-sqlalchemy
  + flask-login
  + flask-mysql
  1. **Base system**

Since I had already implemented a database backend, I just copied/extended the assignment 2 site. Primary modifications were:

* /history – GET provides a list of all tests performed by the user, with the count as id=numqueries, and each test listed with id=query#. Clicking a test shows the /history/query# page for that query. The admin is provided a form with the username in id=userquery to provide the history for a specific user.
* /history/query[#] – GET provides the tests performed, including the test id in id=queryid, the user that performed it in id=username, the test text in id=querytext, and the results in id=queryresults. No info is shown if the user doesn’t have access to the test info. Note, the id # is unique across all users.
* /login\_history – GET provides the admin a form to enter a user in id=userid, POST returns the login history for a user, with id=login[#] in each item, and the login time (id=login#\_time) and the logout time (id=logout#-time). If no logout was recorded, then logout time should be “N/A”.
  1. Schema
     1. User
* id
* username
* mfaid
* password\_hash
* password\_salt
  + 1. TestLog
* id
* user\_id
* timestamp – DateTime, when the test was performed.
* test\_input – Text, the input to the test. Note, text was chosen as inputs could be extremely long.
* test\_output – Text, the output results of the test, list of misspelled words.
  + 1. SecLog
* id
* user\_id
* login\_time
* logout\_time

1. **Security First Pass**
   1. **Flask, SQLAlchemy**

.Flask-Wtf is used to validate and filter all inputs, as well as SQLAlchemy for all database interactions. By using these libraries, parameters and outputs are bound through code, rather then string concatenated: In particular, this helps mitigate SQLi, via SQLAlchemy, by automatically performing parameter binding of inputs.

* 1. **Passwords**
* Salting and Hashing
  1. **Injection Inputs**

There are a couple inputs that need to be protected against, as they can act as injection vectors:

* /url[#] – The use of a parameter in the url means that it could be used to inject values other then the intended number. Positional processing (characters 4-10 etc), and casting to an integer should be sufficient for processing.
* username/mfaid/test input/output – These values are stored in the database, and performing queries against them could perform SQLi. Parameter binding through SQLAlchemy prevents direct execution of the input as SQL.

1. **Testing**

Testing

).

* 1. **csrf**

[Test: TestNormalUsage.test\_CSRF()]

Basic csrf testing was performed by making a valid login request, without including the csrf token. While this did not truly test the csrf functionality, it did test that the Flask-WTF CSRF token was operating correctly. This test correctly redirected the user to the /login.