Michael Tse
11/8/19

Application Security – Assignment 3 – Fall 2019

Github Repository - https://github.com/mt1836/Assignment3

INTRODUCTION

In this assignment we were tasked to integrate a database to our spell checker web application from Assignment 2 that would enable persistent storage of user accounts, login/logout history and spell checking results from all submissions by each user. An administrator account is created during startup of the web application and its credentials are stored in the database. The administrator is able to query all user submissions/results as well as login/logout history. Users are only allowed access to their submission/result history and are not allowed to view any login/logout history. At the conclusion of the database integration, we attempt to perform a SQLi on our web app and mitigate any findings.

DATABASE CREATION

The database chosen for this Assignment was a SQLite DB using the SQLAlchemy extension for Flask. SQLAlchemy is an Object Relational Mapper (ORM) that translates Python classes to database tables and allows users to use function calls that are converted to SQL statements providing ease of use, and a secure and consistent implementation. Three tables were created using the classes defined below:

- **User-** This table/class stores all user account information which includes an auto generated id based on registration sequence, a username, phone number, password taken as form input during the registration process, a salt for salting passwords and two additional columns as back references for the remaining two tables in the database (post and login history)
- **Post**- This table/class stores all spell checking related data including an auto generated id, the submitted text to run against the spell checker, the results of the misspelled words, the date the text was submitted for spell checking, the number of spell check submissions by a user, and finally a user id foreign key that references the primary key id generated in the User table.
- **Login_history –** This table/class stores login/logout activity to record session detail. It includes an auto generated id, a login and logout timestamp for a users session and finally a user id foreign key that references the primary key id generated in the User table.

TEMPLATES

To support the additional functionality, three html pages were created:

- **History.html** – Web page to display a users spell check history
- **Query_details.html** – Web page to display details of a specific spell check submission/results
- **Login_history.html** – Web page for the administrator to view the login/logout history for any user. Logging is critical in providing accountability/traceability if there is an incident that needs to be investigated.

TESTING

We needed to ensure that the additional functionality was implemented correctly and tested the following conditions:

- Non-admin users are only allowed to view their own submission history - To address this we modified our login_history.html to only display the input (user search) form to an administrator and added an if statement in our app.py file which only allows an admin to submit form data. For all other users the application automatically queries for the current username and queries the database for the necessary history information and returns it to history.html.
- Non-admin user is not able to view login history – This was performed on two fronts. First we eliminated the long history link from appearing when a non-admin user is logged in, however this does not restrict the user to manually manipulating the URL to get to the login history page. To address this we added an if statement in our app.py file which only allows an admin to submit form data and modified our login_history.html to only display the input (user search) form to an administrator.
- Non-admin users should not be allowed to access query history by directly modifying the URL – It is possible for users to directly access query history by modifying the URL to go to specific query details (localhost:5000/history/query#). To address this we added the following code that checked who the owner was of the query#. If the owner is the current user then access was allowed. If the owner was not the current user a second check was performed to see if the current user was an admin. If they were not an admin they were not allowed access. If they were an admin they were allowed direct access via URL modification.

Since most of the testing for the web app was performed in the previous two assignments, the security test performed in this assignment was focused specifically on SQL injections. We used SQL Map as a tool to perform these tests. SQL Map is a piece of software that detects and exploits database vulnerabilities and automates the process of detecting and exploiting SQL injection flaws. It supports many databases and techniques and provides an easy and comprehensive way for developers to test their implementation.

To test using SQL Map we first disabled CSRF from our application to reduce complexity as the SQL Map tool would need to have the CSRF token in order to run through its sequence of tests. We then entered the command below to initiate the test against the POST form input fields username, password and 2fa for our registration page:

The resulting output is shown below:

```
appsec@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application Security/Assignment3$ sqlmap -u "localhost:5000/register" --data "username=admin&password=Administrator@1&phone_number=12345678901&
submit=Sign Up"
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.2.4#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. D
evelopers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 21:47:55

[21:47:55] [INFO] testing connection to the target URL
[21:47:55] [INFO] testing if the target URL content is stable
[21:47:56] [INFO] target URL content is stable
[21:47:56] [INFO] testing if POST parameter 'username' is dynamic
[21:47:56] [WARNING] POST parameter 'username' does not appear to be dynamic
[21:47:57] [WARNING] heuristic (basic) test shows that POST parameter 'username' might not be injectable
[21:47:57] [INFO] testing for SQL injection on POST parameter 'username'
[21:47:57] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:47:57] [WARNING] reflective value(s) found and filtering out
[21:47:58] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[21:47:58] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[21:47:58] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[21:47:59] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[21:47:59] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[21:47:59] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[21:47:59] [INFO] testing 'MySQL inline queries'
[21:47:59] [INFO] testing 'PostgreSQL inline queries'
[21:47:59] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[21:47:59] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[21:47:59] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[21:47:59] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[21:48:00] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[21:48:00] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[21:48:00] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[21:48:00] [INFO] testing 'Oracle AND time-based blind'
[21:48:00] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[21:48:02] [WARNING] POST parameter 'username' does not seem to be injectable
[21:48:02] [INFO] testing if POST parameter 'password' is dynamic
[21:48:02] [WARNING] POST parameter 'password' does not appear to be dynamic
[21:48:02] [WARNING] heuristic (basic) test shows that POST parameter 'password' might not be injectable
[21:48:02] [INFO] testing for SQL injection on POST parameter 'password'
[21:48:03] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:48:03] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[21:48:03] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[21:48:03] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[21:48:03] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[21:48:03] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[21:48:04] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[21:48:04] [INFO] testing 'MySQL inline queries'
[21:48:04] [INFO] testing 'PostgreSQL inline queries'
[21:48:04] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[21:48:04] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[21:48:04] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[21:48:04] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[21:48:04] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[21:48:04] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[21:48:04] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[21:48:04] [INFO] testing 'Oracle AND time-based blind'
[21:48:04] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[21:48:06] [WARNING] POST parameter 'password' does not seem to be injectable
[21:48:06] [INFO] testing if POST parameter 'phone_number' is dynamic
[21:48:06] [WARNING] POST parameter 'phone_number' does not appear to be dynamic
[21:48:06] [WARNING] heuristic (basic) test shows that POST parameter 'phone_number' might not be injectable
[21:48:06] [INFO] testing for SQL injection on POST parameter 'phone_number'
[21:48:06] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:48:07] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[21:48:07] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[21:48:07] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[21:48:07] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[21:48:08] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[21:48:08] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[21:48:08] [INFO] testing 'MySQL inline queries'
[21:48:08] [INFO] testing 'PostgreSQL inline queries'
[21:48:08] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[21:48:08] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[21:48:08] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[21:48:08] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[21:48:08] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[21:48:08] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[21:48:09] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[21:48:09] [INFO] testing 'Oracle AND time-based blind'
[21:48:09] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[21:48:11] [WARNING] POST parameter 'phone_number' does not seem to be injectable
[21:48:11] [INFO] testing if POST parameter 'submit' is dynamic
[21:48:11] [WARNING] POST parameter 'submit' does not appear to be dynamic
[21:48:11] [WARNING] heuristic (basic) test shows that POST parameter 'submit' might not be injectable
[21:48:11] [INFO] testing for SQL injection on POST parameter 'submit'
[21:48:11] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:48:12] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[21:48:12] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[21:48:12] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[21:48:12] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[21:48:12] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[21:48:12] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[21:48:12] [INFO] testing 'MySQL inline queries'
[21:48:12] [INFO] testing 'PostgreSQL inline queries'
[21:48:13] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[21:48:13] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[21:48:13] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[21:48:13] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[21:48:13] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[21:48:13] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[21:48:13] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[21:48:14] [INFO] testing 'Oracle AND time-based blind'
[21:48:14] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[21:48:15] [WARNING] POST parameter 'submit' does not seem to be injectable
[21:48:15] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there i
s some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment')

[*] shutting down at 21:48:15
```

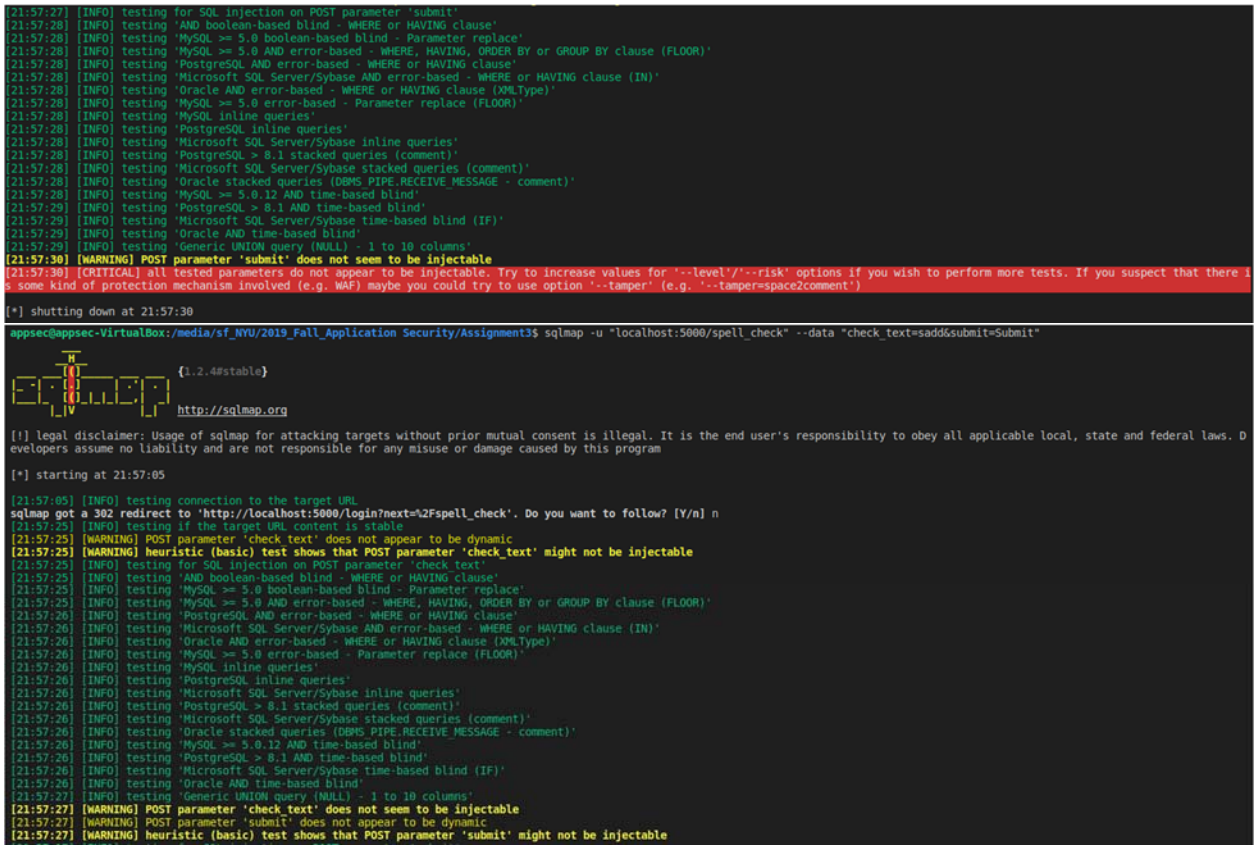
The same was done for the login page:

```
appsec@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application Security/Assignment3$ sqlmap -u "localhost:5000/login" --data "username=admin&password=Administrator@1&phone_number=123456789016&sub
mit=Login"
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.2.4#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. D
evelopers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 21:51:46

[21:51:46] [INFO] testing connection to the target URL
[21:51:47] [INFO] testing if the target URL content is stable
[21:51:47] [INFO] heuristics detected web page charset 'ascii'
[21:51:47] [WARNING] target URL content is not stable. sqlmap will base the page comparison on a sequence matcher. If no dynamic nor injectable parameters are detected, or in case of junk resu
lts, refer to user's manual paragraph 'Page comparison'
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit] c
[21:52:03] [INFO] searching for dynamic content
[21:52:03] [CRITICAL] target URL content appears to be heavily dynamic. sqlmap is going to retry the request(s)
[21:52:04] [INFO] dynamic content marked for removal (1 region)
[21:52:04] [INFO] testing if POST parameter 'username' is dynamic
[21:52:05] [WARNING] POST parameter 'username' does not appear to be dynamic
[21:52:05] [INFO] testing for SQL injection on POST parameter 'username'
[21:52:05] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:52:05] [WARNING] reflective value(s) found and filtering out
[21:52:06] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[21:52:06] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[21:52:06] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[21:52:06] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[21:52:07] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[21:52:07] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[21:52:07] [INFO] testing 'MySQL inline queries'
[21:52:07] [INFO] testing 'PostgreSQL inline queries'
[21:52:07] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[21:52:07] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[21:52:07] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[21:52:07] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[21:52:08] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[21:52:08] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[21:52:08] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[21:52:08] [INFO] testing 'Oracle AND time-based blind'
[21:52:09] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[21:52:11] [WARNING] POST parameter 'username' does not seem to be injectable
[21:52:11] [INFO] testing if POST parameter 'password' is dynamic
[21:52:11] [WARNING] POST parameter 'password' does not appear to be dynamic
```

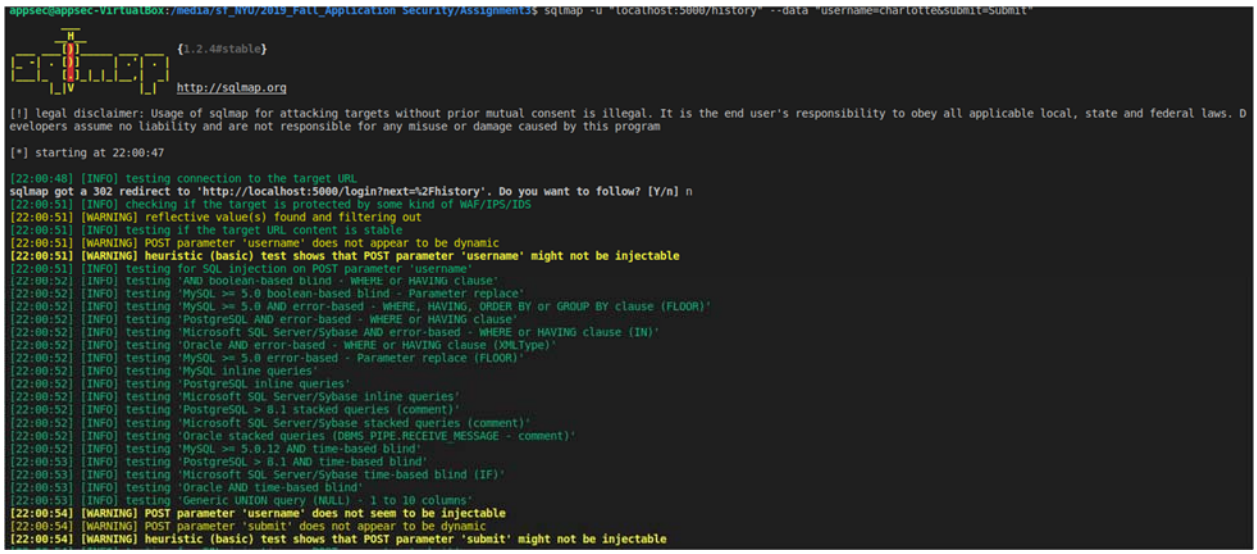
```
[21:52:11] [INFO] testing for SQL injection on POST parameter 'password'
[21:52:12] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:52:12] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[21:52:12] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[21:52:12] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[21:52:12] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[21:52:13] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[21:52:13] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[21:52:13] [INFO] testing 'MySQL inline queries'
[21:52:13] [INFO] testing 'PostgreSQL inline queries'
[21:52:13] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[21:52:13] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[21:52:13] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[21:52:13] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[21:52:13] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[21:52:13] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[21:52:14] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[21:52:14] [INFO] testing 'Oracle AND time-based blind'
[21:52:14] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[21:52:16] [WARNING] POST parameter 'password' does not seem to be injectable
[21:52:16] [INFO] testing if POST parameter 'phone_number' is dynamic
[21:52:16] [WARNING] POST parameter 'phone_number' does not appear to be dynamic
[21:52:16] [INFO] testing for SQL injection on POST parameter 'phone_number'
[21:52:16] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:52:17] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[21:52:17] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[21:52:17] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[21:52:18] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[21:52:18] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[21:52:18] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[21:52:18] [INFO] testing 'MySQL inline queries'
[21:52:18] [INFO] testing 'PostgreSQL inline queries'
[21:52:18] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[21:52:18] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[21:52:18] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[21:52:19] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[21:52:19] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[21:52:19] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[21:52:19] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[21:52:19] [INFO] testing 'Oracle AND time-based blind'
[21:52:20] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[21:52:22] [WARNING] POST parameter 'phone_number' does not seem to be injectable
[21:52:22] [INFO] testing if POST parameter 'submit' is dynamic
[21:52:22] [WARNING] POST parameter 'submit' does not appear to be dynamic
```

```
[21:52:22] [INFO] testing for SQL injection on POST parameter 'submit'
[21:52:23] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:52:28] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[21:52:28] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[21:52:31] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[21:52:34] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[21:52:37] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[21:52:40] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[21:52:40] [INFO] testing 'MySQL inline queries'
[21:52:41] [INFO] testing 'PostgreSQL inline queries'
[21:52:42] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[21:52:42] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[21:52:45] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[21:52:47] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[21:52:49] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[21:52:52] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[21:52:55] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[21:52:58] [INFO] testing 'Oracle AND time-based blind'
[21:53:01] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[21:53:34] [WARNING] POST parameter 'submit' does not seem to be injectable
[21:53:34] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. You can give it a go with t
he switch '--text-only' if the target page has a low percentage of textual content (~3.56% of page content is text). If you suspect that there is some kind of protection mechanism involved (e.
g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment')
[21:53:34] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 1 times

[*] shutting down at 21:53:34
```

For the spell check page, we were required to log in first before running the SQL Map command. It resulted in the following:



Same was performed for the history page. Results shown below:



```
[22:00:54] [INFO] testing for SQL injection on POST parameter 'submit'
[22:00:54] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[22:00:54] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[22:00:54] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[22:00:54] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[22:00:54] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[22:00:54] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[22:00:54] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[22:00:54] [INFO] testing 'MySQL inline queries'
[22:00:54] [INFO] testing 'PostgreSQL inline queries'
[22:00:54] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[22:00:54] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[22:00:55] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[22:00:55] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[22:00:55] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[22:00:55] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[22:00:55] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[22:00:55] [INFO] testing 'Oracle AND time-based blind'
[22:00:55] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[22:00:56] [WARNING] POST parameter 'submit' does not seem to be injectable
[22:00:56] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there i
s some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment')

[*] shutting down at 22:00:56
```

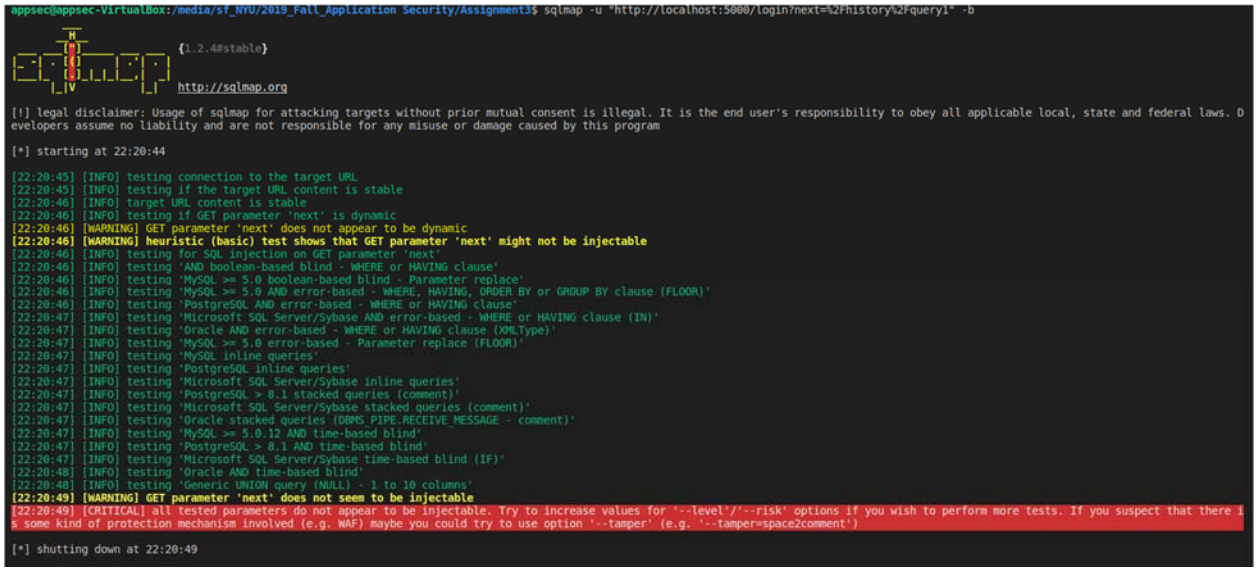
The login_history page is the final page accepting form input to be POST tested. Results shown below:

```
appsec@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application Security/Assignment3$ sqlmap -u "localhost:5000/history/login_history" --data "username=charlotte&submit=Submit"
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.2.4#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. D
evelopers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 22:03:21

[22:03:21] [INFO] testing connection to the target URL
sqlmap got a 302 redirect to 'http://localhost:5000/login?next=%2Fhistory%2Flogin_history'. Do you want to follow? [Y/n] n
[22:03:25] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[22:03:25] [WARNING] reflective value(s) found and filtering out
[22:03:25] [INFO] testing if the target URL content is stable
[22:03:25] [WARNING] POST parameter 'username' does not appear to be dynamic
[22:03:25] [WARNING] heuristic (basic) test shows that POST parameter 'username' might not be injectable
[22:03:25] [INFO] testing for SQL injection on POST parameter 'username'
[22:03:25] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[22:03:26] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[22:03:26] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[22:03:26] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[22:03:26] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[22:03:26] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[22:03:26] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[22:03:26] [INFO] testing 'MySQL inline queries'
[22:03:26] [INFO] testing 'PostgreSQL inline queries'
[22:03:26] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[22:03:26] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[22:03:26] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[22:03:26] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[22:03:26] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[22:03:26] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[22:03:26] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[22:03:27] [INFO] testing 'Oracle AND time-based blind'
[22:03:27] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[22:03:27] [WARNING] POST parameter 'username' does not seem to be injectable
[22:03:27] [WARNING] POST parameter 'submit' does not appear to be dynamic
[22:03:27] [WARNING] heuristic (basic) test shows that POST parameter 'submit' might not be injectable
```

```
[22:03:27] [INFO] testing for SQL injection on POST parameter 'submit'
[22:03:28] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[22:03:28] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[22:03:28] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[22:03:28] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[22:03:28] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[22:03:28] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[22:03:28] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[22:03:28] [INFO] testing 'MySQL inline queries'
[22:03:28] [INFO] testing 'PostgreSQL inline queries'
[22:03:28] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[22:03:28] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[22:03:28] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[22:03:28] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[22:03:29] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[22:03:29] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[22:03:29] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[22:03:29] [INFO] testing 'Oracle AND time-based blind'
[22:03:29] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[22:03:30] [WARNING] POST parameter 'submit' does not seem to be injectable
[22:03:30] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there i
s some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment')

[*] shutting down at 22:03:30
```

The output of all POST tests showed that our web application was not likely vulnerable to SQLi. The final test was to check if our GET requests for the query# directly via the URL was vulnerable. We did this using the command and results below:

```
appsec@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application Security/Assignment3$ sqlmap -u "http://localhost:5000/login?next=%2Fhistory%2Fquery1" -b
```



All of our GET and POST tests against user data entry points from our web application turned up to be unlikely for SQLi vulnerability. This is due to the SQLAlchemy ORM we used which not only provides ease of use but provides consistent implementation in a secure manner. The developers of the ORM are focused in secure design such that translated code into SQL statements are not subject to common vulnerabilities. For developers who choose not to use an ORM, they would need to understand in detail how SQLi work and the different types in order to develop applications in a secure manner. Even with this knowledge they cannot guarantee that they will not overlook something and code in a consistent manner. For these reasons ORMs provide developers an easy way to secure their applications from a majority of vulnerabilities and should be used where possible.