**CSI4139**
Lab 4
NamChi Nguyen - 7236760

Group members:
Julian Templeton - 8229400
Sean McAvoy - 8263210

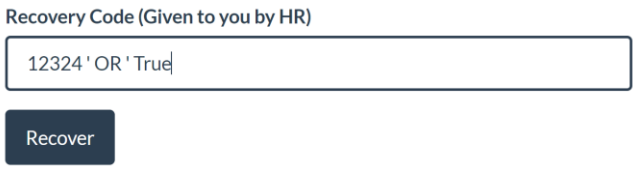
1. **Register an account, you may need to use a SQL injection to help you out here. You will find a flag once your SQL injection is successful.**

   The account was created with an SQL injection after entering *12324' OR 'True* in the recovery code field in order to obtain the employee access code. The simple portion of an SQL query would always evaluate to True which would return all records from the database table due to the OR regardless if the recovery code of 12324 was incorrect.

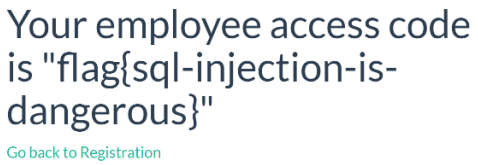
   To prevent this vulnerability, some methods that can be taken by the admin are the following [1]:

   - User input validation: sanitize the user input by filtering only permitted characters/strings or digits as required by the input.
   - Use prepared SQL statements that don't depend on user input.
   - Use a web application firewall that would help filter malicious data.

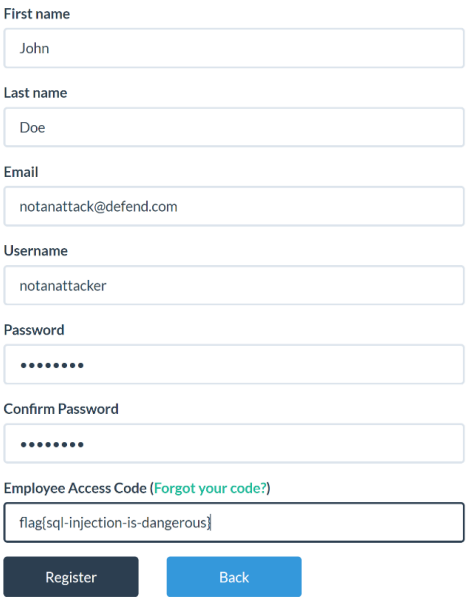
**SQL injection:**




**Flag:**




**Registration using the recovery code:**



2. **Create a link that redirects the user to when they log in. There is no flag for this challenge, just record the URL that you used.**
   The URL used was: http://192.168.99.100:1337/employee?page=dashboard.html.

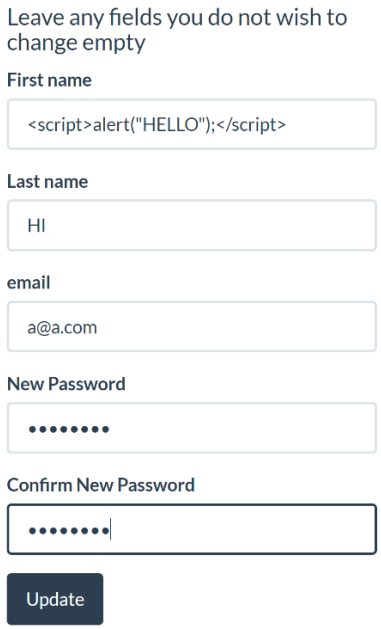
   Although, to help mitigate the redirecting to a malicious website, SSL certificates should be enabled and the website should be using HTTPS [2].

3. **Find a stored cross site scripting vulnerability, have an alert pop up on the dashboard page. Once successful a second alert will pop up with the flag.**

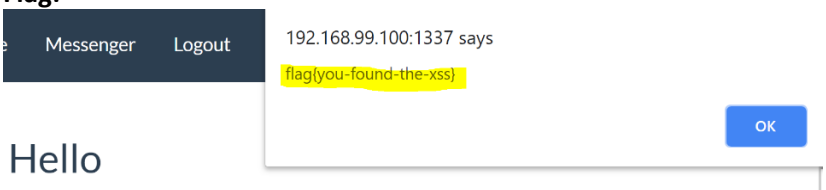
   A cross site scripting attack is the injection of malicious code or script that is executed in the target's browser [3]. This can be used to steal data, particularly a user's cookies to steal their identity. A simple alert script was inputted in the first name field and although it is not a valid input, the script was executed and the flag appeared in the pop-up.

   One way to help prevent this vulnerability is user input validation (i.e. data sanitization), in which if a script was entered, it would not execute because it doesn't conform to the proper format that was expected. Another method is filtering which removes inappropriate keywords such as <script></script> tags and HTML markup and replaces them with the empty string [3].

   **Cross site script:**




   **Flag:**



4. **A piece of information is exposed on the employee dashboard. Find this and a flag with it.**

   The information exposed on the dashboard is the backup of the user database and its location in the root directory. The flag was found by reading the HTML source code in the web browser of the site.

   To prevent this, a good practice to follow would to ensure that debugging comments left by the developers are removed from the source code before the website or web application is released and goes live as they may reveal sensitive data. In this case, the developer made a note about the backup location in their comments.

```
<body>
    <!-- ---------------------->
    <!-- DEVELOPER NOTES BELOW-->
    <!-- ---------------------->
    <!--flag{sensitive-data-exposure-isnt-good}-->
    <!--backup of user database can be found at backup.sql in the application's root directory-->
    <!-- ---------------------->
    <!-- END DEV NOTES-->
    <!-- ---------------------->
```

5. **Based on the piece of information exposed in Question 4, are you able to find a file on the system which shouldn't be accessible?**

   The file was found by accessing the backup.sql on the root directory using the following URL: http://192.168.99.100:1337/employee?page=../backup.sql.

   *File found:*

```
flag{local-file-inclusion-found}

 user_id | username |             password             |          first_name           |   last_name   |          email
---------+----------+----------------------------------+-------------------------------+---------------+--------------------------
       1 | admin    | E258490A37CCCA434D2FE712DDD82A01 | The                           | Administrator | someadmin@somewebsite.com
       2 | fred     | 1A1DC91C907325C69271DDF0C944BC72 | Fred                          | Johnston      | fred@somewebsite.com
```

   To help prevent this, if access to the database is compromised, then the above table should be at least separated into multiple tables in order to avoid inference from queries, however in this case, all the data is in a single table so it is easy for the attacker to steal sensitive data.

   For example, the login credentials should be a separate table with a primary key of the user_id whereas basic information of the user such as the first and last name would be in a different table and it'd reference the credentials table with the foreign key of user_id. Furthermore, the data itself should be encrypted and the passwords hashed along with salting if not already done.

6. **So, in question 5 you've found some data that shouldn't be there. See if you can figure out Fred's password. This is another form of sensitive data exposure.**

   From the file found, Fred's password can easily be determined from the table.

   Similarly to Question 5, to help mitigate sensitive data exposure, all the data shouldn't be stored in a single database table but multiple tables and then these tables can reference other tables using the user_id as a foreign key. This way, if an attacker attempts to query for Fred's password for example, and they only knew his first name, they wouldn't be able to since the password is not stored in the same table as his name.

   **Fred's password:**

```
             password             |          first_name
----------------------------------+----------------------
 E258490A37CCCA434D2FE712DDD82A01 | The
 1A1DC91C907325C69271DDF0C944BC72 | Fred
```

7. **This one is optional and quite challenging! Find a way to change the admin account's password, you won't be able to crack the hash... there might be another way through the messaging system. After all his assistant does automatically read the messages.**

   Since the site doesn't protect against cross site scripting, a Javascript function could be used in the message input that would execute after the message was sent and attempt to alter the admin's password.

   As mentioned in previous questions, a way to prevent this vulnerability would be to protect against cross site scripting by data sanitization and forbidding the capability to inject JavaScript scripts in an input field.

To conclude, web applications are prone to numerous attacks that malicious entities can exploit. This lab only explored a few important vulnerabilities that included SQL injection and cross-site scripting. However, precautions must be taken to minimize unauthorized exposure and access of sensitive data on the web. A key factor is to not trust the user and their input, in which any sort of input entered must be validated and sanitized as well as always encrypt sensitive data in case of exposure.

**References**:


[1] P. Rubens, "How to Prevent SQL Injection Attacks", May 2, 2018, [Online]. Available: https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks.html. [Accessed: Nov. 19, 2018].

[2] S. Helpton, "How to get HTTPS: Setting up SSL on your website", Jun. 6, 2013, [Online]. Available: https://www.howto-expert.com/how-to-get-https-setting-up-ssl-on-your-website/. [Accessed: Nov. 23, 2018].

[3] Software Testing Help, "Cross Site Scripting (XSS) Attack Tutorial with Examples, Types & Prevention", Aug. 12, 2018, [Online]. Available: https://www.softwaretestinghelp.com/cross-site-scripting-xss-attack-test/. [Accessed: Nov. 16, 2018].