Software and Computer Security

SOFE 4840U

Final Project Phase 2: Development & Design

SQL Injection

Date: 03/18/22

Project Group 7

Github: https://github.com/Anujpatel2/SQL-Injection-Prevention

Student Name	Student Number
Ireni Ruthirakuhan	100657302
Raj Parikh	100655911
Mitul Patel	100700131
Anuj Patel	100651964

Project Scope

The aim of this project is to defend against an SQL Injection attack that may allow an adverse entity to target critical data that had been stored in the database for the given web application developed. We present many viable solutions to defend against possible harmful SQL injection attacks. Through the construction of the web application using HTML and PHP we were able to successfully develop a simple login web application connected to a database where SQL injection attacks could be verified by the adverse end-user.

To prevent SQL Injection attacks five different techniques are in the process of being developed and utilized to ensure the security of the individual's data that had been compromised and exposed. These techniques include parameterizing queries, escaping, hashing, utilization of sanitation packages such as filter_var, and regex. For this submission, the group spent significant time setting up the application structure, and as a result only the parameterizing queries functionality has been completed. The rest of the techniques are in process and will be completed/tested as a part of the next phase.

Project Setup

The development of the web application had been constructed through the use of HTML and PHP. Below present images showcasing the front-end UI, backend structure, as well as successful and unsuccessful login scenarios executed by the end-user.

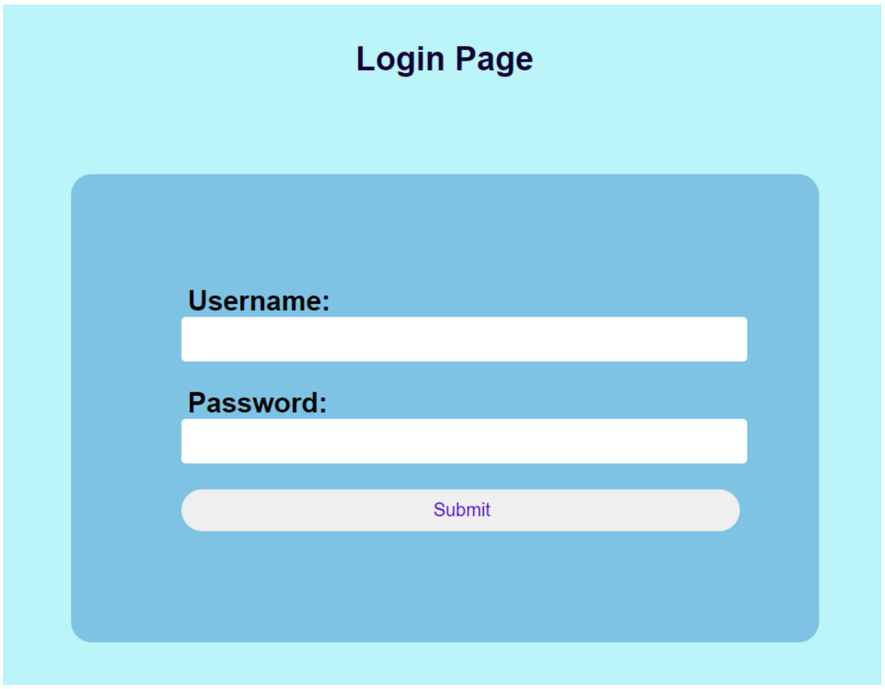
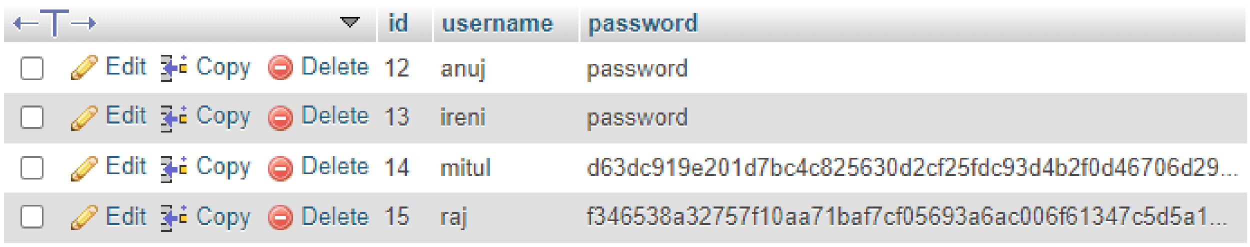
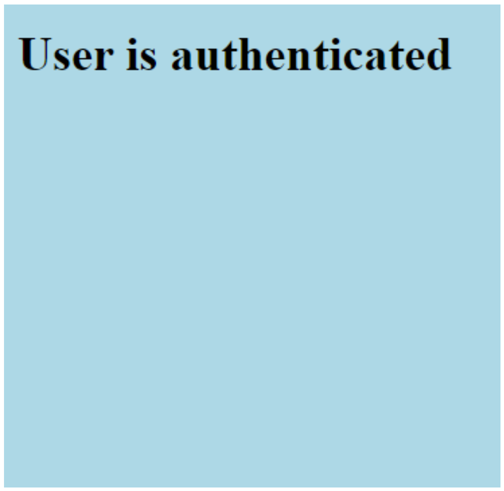


Figure 1: Front-end of web application used for SQL Injection



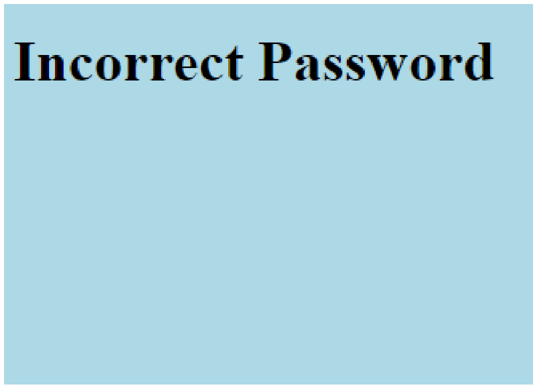
	id	username	password
Edit Copy Delete	12	anuj	password
Edit Copy Delete	13	ireni	password
Edit Copy Delete	14	mitul	d63dc919e201d7bc4c825630d2cf25fdc93d4b2f0d46706d29...
Edit Copy Delete	15	raj	f346538a32757f10aa71baf7cf05693a6ac006f61347c5d5a1...

Figure 2: Backend structure utilized had been created through the use of phpMyAdmin



```
User is authenticated
```

Figure 3: Successful login with valid credential would present the following screen

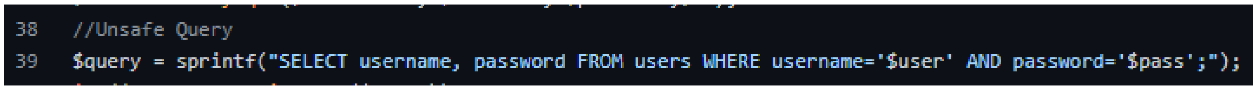


```
Incorrect Password
```

Figure 4: User enters invalid login credentials are presented with the following output.

SQL Injection techniques to breach data (Inputs and Outputs)

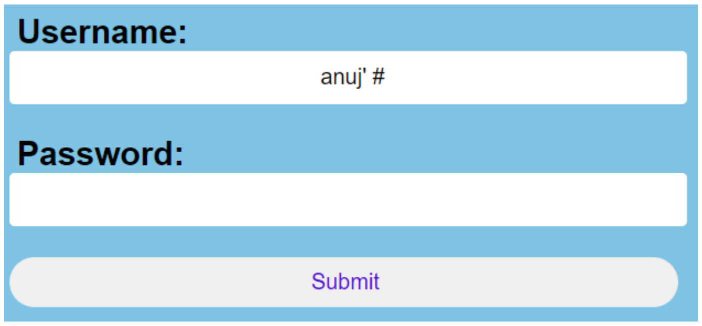
Presented below are two ways the group had determined that the web application developed was a victim of possible SQL injection attacks. Both of these examples make use of the following query:



```
//Unsafe Query
$query = sprintf("SELECT username, password FROM users WHERE username='$user' AND password='$pass';");
```

Figure 5: Unsafe query

Example 1: This is an example of an SQL injection where an unauthorized user is able to gain access to the system. The “#” symbol represents a comment in SQL, and as a result the password validation portion in the SQL query is ignored.



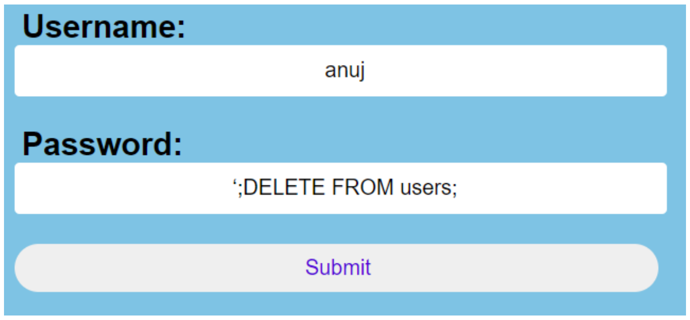
Username:

anuj' #

Password:

Figure 6: Successful login: anuj ‘ #

Example 2: This is an example of an SQL injection where an attacker is deliberately causing harm to the system. A harmful query has been entered in the password section, and as a result it gets executed after the query in figure 5. This harmful query deletes all of the rows in the users table in the database.



Username:

anuj

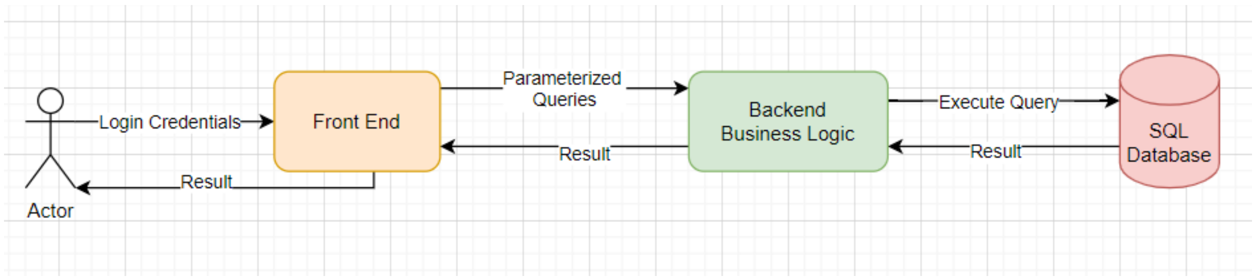
Password:

‘;DELETE FROM users;

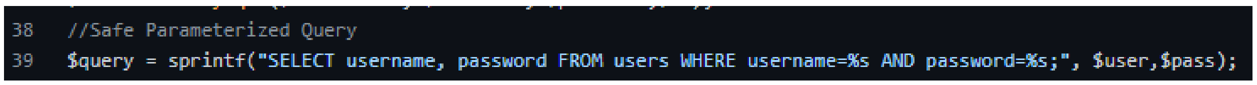
Figure 7: Successful deletion of table: ‘;DELETE FROM users;

SQL Injection Prevention Techniques

Technique 1: Parameterized Queries



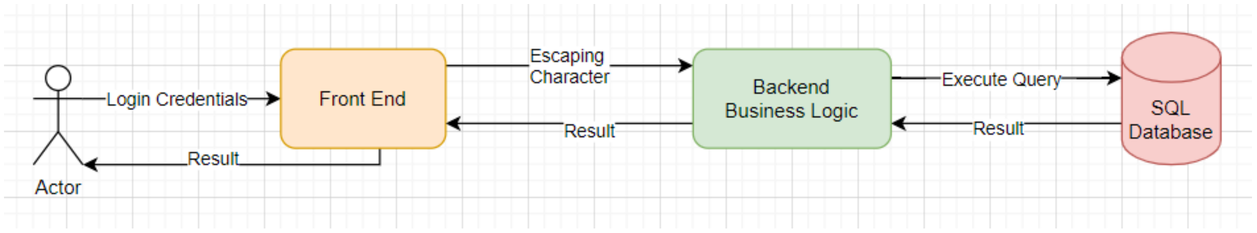
Parameterized queries are a means of pre-compiling an SQL statement so that you can then supply the parameters in order for the statement to be executed. This method makes it possible for the database to recognize the code and distinguish it from input data [1]. As the user input is directly quoted by the program, the input cannot cause harm and therefore the risk of an SQL injection attack is mitigated.



```
//Safe Parameterized Query
$query = sprintf("SELECT username, password FROM users WHERE username=%s AND password=%s;", $user,$pass);
```

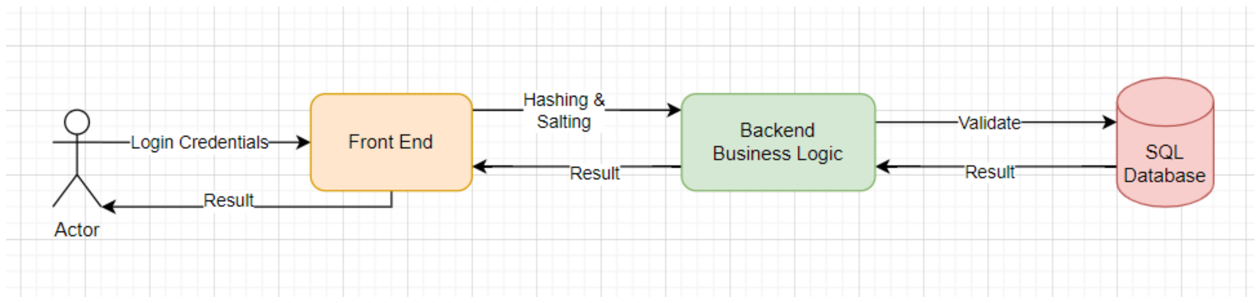
Figure 8: Parameterized Query

Technique 2: Escaping



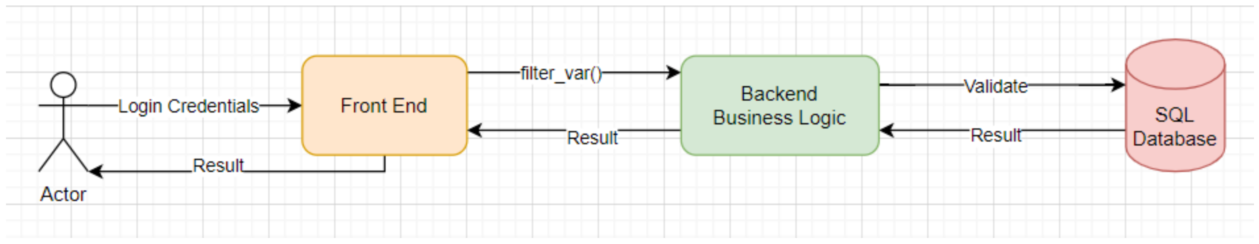
Another technique that can be applied would be the Escaping technique. This is used to make sure the database does not execute any noninted SQL queries. The procedure to escape data is stripping out unwanted data like malformed HTML or any javascript tags. This is so the system doesn't read the string as a code. All data should be SQL-Escaped before the SQL queries are executed to prevent SQL Injection attacks. This can be an issue when the front end would want a user to upload files or images to the inputs. That would be a problem as the code would eliminate all HTML tags which would end up the content not being passed through correctly. This can be solved by whitelisting certain HTML tags to allow the upload of specific formats only.

Technique 3 : Hashing user input



If the password is hashed and saved to the database during signup, then only a single string of user known password will hash to a specific string value which is saved in the database. We utilize SHA-2 to encode passwords saved in the database to authenticate returning users. Salting would also add another layer of security to the SQL injection by appending a unique random string of characters only known to the website. When the returning user inputs a user and password field into the database, the input will be hashed along with salting and compared to the value saved in the database. This would prevent executing any unwanted query from the back side.

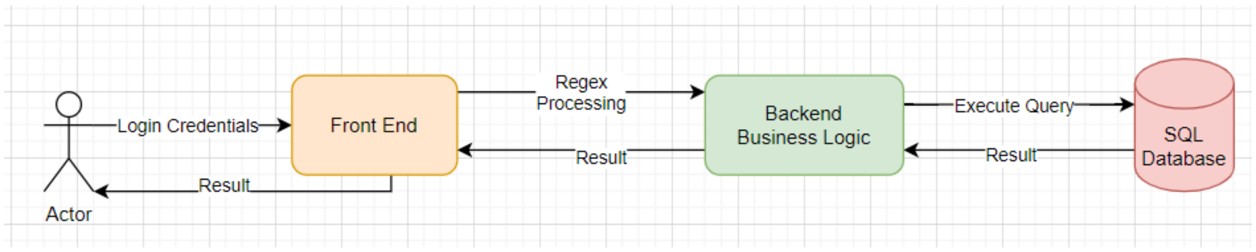
Technique 4 : filter_var() function



Another technique utilized towards the prevention of possible SQL Injection attacks include the introduction and use of the filter_var function available on PHP to sanitize and validate inputted data. This function is able to filter the variable with the expected filter, returning the success or failure of inputted data. The utilization of this function can only be introduced in the case where the adverse entity is accessing information through the username field. The reason for this is due to the filter_var package not having the ability to verify the inputted value of the password field appropriately. This technique is then to be paired with technique 3, hashing targeted towards the use of password entry. In the case where the username is of the format of an email, this function is able to take in value inputted by the user and sanitize it as an email structure to verify. This can be seen as `filter_var($email, FILTER_VALIDATE_EMAIL)`.

Technique 5 : Regex

One last technique that we will utilize is the use of regex to detect input validation from the client for possible SQL injections. Through the utilization of this technique we are able to identify whether some parts of the inputted field are not of the format expected and include characters that are prohibited, such as SQL meta-characters. Knowing the structure of an SQL injection statement we are able to construct an expression that would validate whether there is a block or inline comment placed within the string SQL command entered by the adverse entity. This should be implemented on the server side to further encourage secure behavior of the application.



References

[1]
https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks/

[2] https://www.php.net/manual/en/function.filter-var.php

[3] https://benhoyt.com/writings/dont-sanitize-do-escape/