**Gathering The Information And Structure Of The Data Base Using**

**Logically Incorrect Queries**

Nikhil V, Aashish Raj, Srinivas

18BIT0196 ,18BIT0113, 18BIT0007

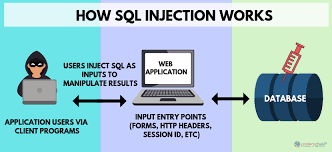
School Of Information Technology

**Abstract-** **SQL injection is a technique where the attacker injects an input in the query in order to change the structure of the query intended by the programmer and gaining the access of the database which results modification or deletion of the user’s data. In the injection it exploits a security vulnerability occurring in database layer of an application. SQL injection attack is the most common attack in websites in these days. Some malicious codes get injected to the database by unauthorized users and get the access of the database due to lack of input validation. Input validation is the most critical part of software security that is not properly covered in the design phase of software development life-cycle resulting in many security vulnerabilities. This paper presents the techniques for detection and prevention of SQL injection attack. There are no any known full proof defences available against such type of attacks. In this paper we have made a website based on charity works to donate funds as it contains the information of all the account details and there history of donations.**

**Keywords— web application, SQLIA, detection, prevention, vulnerabilities, web architecture**

1. **INTRODUCTION**

Now a days web application is widely used in various applications it is the reliable and efficient solution to the challenges of communicating and conducting the various organisation, business or commerce over the internet. Now each and every important assignment is done by using the web application which is connected through the internet. For example electricity bill, online shopping, gaming, banking, messaging, shopping, conferences, etc. So the increase of web application involving the various security issues in the web world. The SQLIA (structured query language injection attack) is a code injection attack technique commonly used for attacking websites in which an attacker injects some SQL codes in place of the original codes to get access the database. There are variety of techniques are available to detect SQLIA. The most preferred are Web Framework, Static Analysis, Dynamic Analysis, combined Static and Dynamic Analysis and Machine Learning Technique. Web Framework provides filters to filter special characters but other attacks are not detected. Static Analysis checks the input parameter type, but it fails to detect attacks with correct input type. Dynamic Analysis technique is capable of scanning vulnerabilities of web application but is not able to detect all types of SQLIA. Combined Static and Dynamic Analysis includes the benefit of both, but this method is very complex in order to proceed. Machine Learning method can detect all types of attacks but results in number of false positives and negatives.



1. **SQLIA MECHANISMS**

Types of SQL Injection Attacks There are multitude techniques of attack for exploiting SQL injection vulnerabilities in web applications. In this paper we describe the major types of attacks.

**Illegal/Logical Incorrect Query**:

This kind of attack manipulate the where clues to get the names of the tables and columns that the inquiry works on through using the "having" condition of the "select" statement to generate error message by the server. **Select \* from usertable where username = ‘ ” + g\_name + ” ’ and password = ‘ “ + g\_password + ” ‘ “ ; "** having 1=1- - Result: Column 'usertbl.ID' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

1. **PROPOSED APPROACH AND EXPERIMENTAL RESULTS**

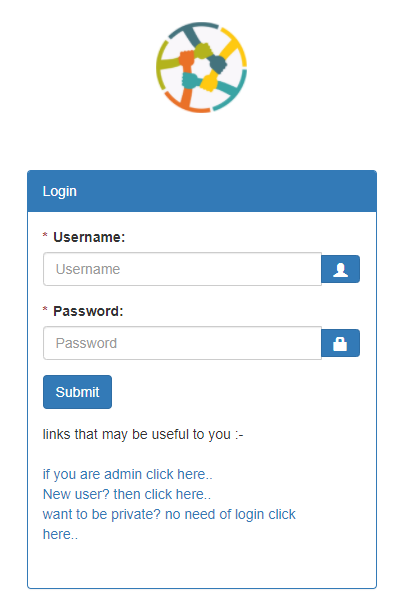
Here we’ll use different logical operators to go sneak inside the website and then entering in it without the correct username and the passwords and by just using vulnerabilities of SQL queries.

1. **Using OR operator**

We will first close the string of the SQL string statement and then we will add an **TRUE statement** with **OR** which will eventually join the whole SQL statement and we’ll get the output of that query as **TRUE** or we can say that the query produces results.

**Example:**

Explaining it with the query used in the charity website mentioned above.



**FIGURE: 3.1**

As we can see from figure 3.1 that in the username form we can fill any thing this will not affect the query and in password column we will write this query:

**" or ""="**

The SQL statement will look like:

SELECT \* FROM users WHERE u\_name ="fasd” AND u\_pass ="**" or ""=""**

The query will produce some results as the whole statement after where returns true

There can be another way to use the same **OR query** by using **comment.**

This statement in the username field:

**" or ""="" –**

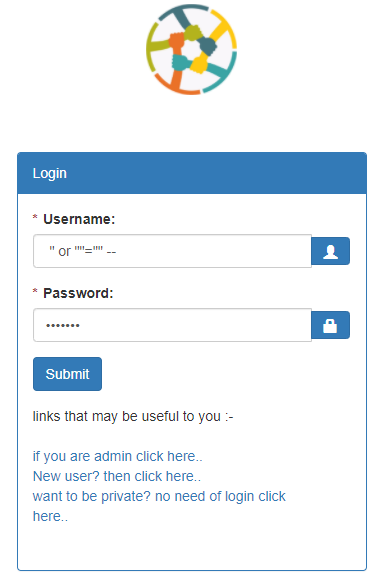
And in password any thing can be written as its going to be comment it out.

The resulting **SQL**statement we will get after this will be

SELECT \* FROM users WHERE u\_name ="**" or ""="" --**

AND u\_pass ="FASDF**"**

The whole underlined part will be commented and the part executed will whole become true and return some value for the result.



**FIGURE: 3.2**

1. **Using XOR operator**

We will first close the string of the SQL string statement and then we will add an **TRUE statement** with **XOR** which will eventually join the whole SQL statement and we’ll get the output of that query as **TRUE** or we can say that the query produces results.

NOTE:

The whole statement will be true as we know that the previous value of the username we entered is wrong as we include many random texts and it can’t belong to a person, and xor of two of these things will give the result as true.

**Example:**

Explaining it with the query used in the charity website mentioned above.

As we can see from figure 3.1 that in the username form we can fill any thing this will not affect the query and in password column we will write this query:

**" xor ""="**

The SQL statement will look like:

SELECT \* FROM users WHERE u\_name ="fasd” AND u\_pass ="**" xor ""=""**

The query will produce some results as the whole statement after where returns true

There can be another way to use the same **XOR query** by using **comment.**

This statement in the username field:

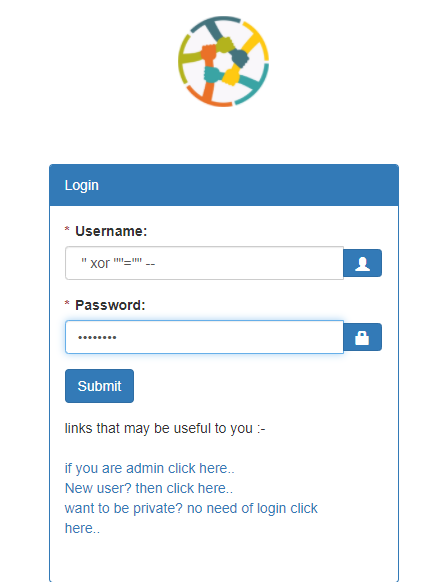
**" xor ""="" –**

And in password any thing can be written as its going to be comment it out.

The resulting **SQL**statement we will get after this will be

SELECT \* FROM users WHERE u\_name ="**" xor ""="" --**  AND u\_pass ="FASDF**"**

The whole underlined part will be commented and the part executed will whole become true and return some value for the result.



**FIGURE: 3.3**

1. **Using NOT operator with OR operator**

Taking NOT operator into consideration, we will make a wrong logical value i.e. which will generate **false** as a result, so the result after NOT of false will be **true** and we will combine this result with **OR** gate, getting the whole SQL statement result as **true.**

**Example:**

Explaining it with the query used in the charity website mentioned above we can do that in two ways, as described in previous approaches like:

Giving any random value in username and giving the proposed query in password field, the resulting SQL statement generated after this is:

SELECT \* FROM users WHERE u\_name ="fasd” AND

u\_pass ="**" or not "a"=""**

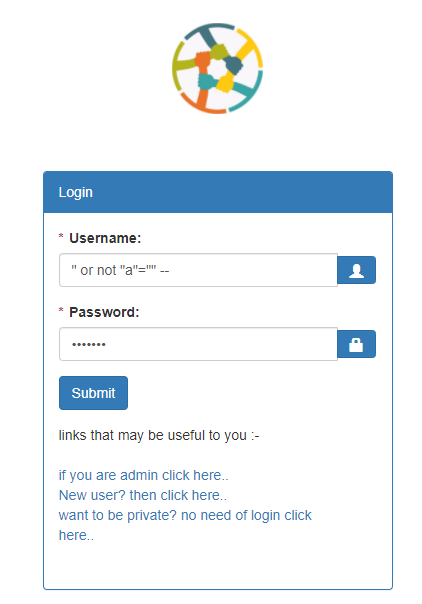
This query will generate the whole result as **true** and we can successfully enter in the website’s user account.

For the approach by using comments we can just enter the proposed SQL statement in the username column and in the password section any random value.

SELECT \* FROM users WHERE u\_name ="**" or not "a"="" --** ” AND

u\_pass ="fasdf**"**

So all the things after – will be commented and we will get the result as **true.** The figure 3.4 shows it.



**Figure 3.4**

1. **Using NOT operator with XOR operator**

Taking NOT operator into consideration, we will make a wrong logical value i.e. which will generate **false** as a result, so the result after NOT of false will be **true** and we will combine this result with **XOR** gate, getting the whole SQL statement result as **true.**

**Example:**

Explaining it with the query used in the charity website mentioned above we can do that in two ways, as described in previous approaches like:

Giving any random value in username and giving the proposed query in password field, the resulting SQL statement generated after this is:

SELECT \* FROM users WHERE u\_name ="fasd” AND

u\_pass ="**" xor not "a"=""**

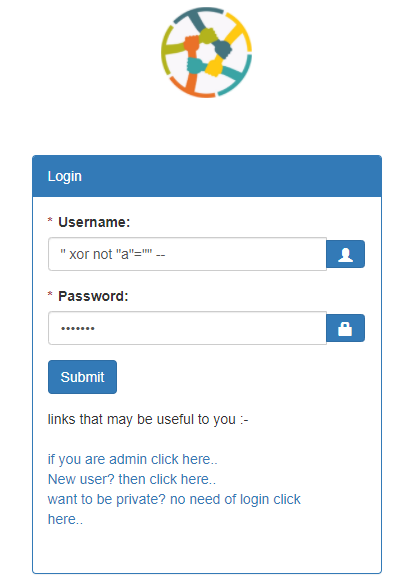
This query will generate the whole result as **true** and we can successfully enter in the website’s user account.

For the approach by using comments we can just enter the proposed SQL statement in the username column and in the password section any random value.

SELECT \* FROM users WHERE u\_name ="**" xor not "a"="" --** ” AND

u\_pass ="fasdf**"**

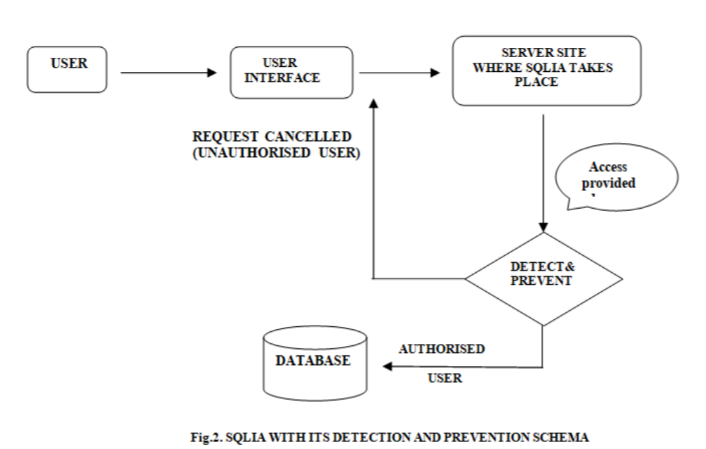
So all the things after – will be commented and we will get the result as **true.** The figure 3.5 shows it.



**Figure 3.5**

1. **PREVENTION AND DETECTION**

During detection and prevention of SQLIA,when the user requested the needed data after passing through the user interface the request is send to the server site .If there is no attack takes place the only authorised user can fetch the needed data .Now a days due to SQLIA the unauthorised user can also fetch the data and other important and confidential details by just adding some query string or by editing the existing query. We can overcome this unauthorised attempt by testing the string through the SQLIA detection and prevention methods as shown



With user input channels being the main vector for SQL injection attacks, most of the defensive methods involve controlling and vetting user input for attack patterns.

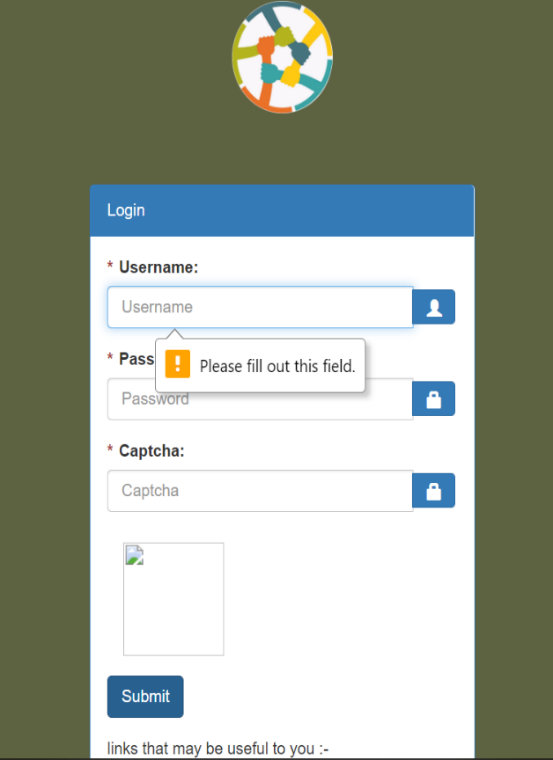
Here are several measures that can ensure user input safety.

1. **Never trust user input**

The first rule of thumb about user input is “[don’t trust and verify](https://en.wikipedia.org/wiki/Trust,_but_verify),” which effectively means all forms of the user input should be considered malicious unless proved otherwise. This accounts not only for simple input boxes such as text areas and text boxes but for everything else as well — such as hidden inputs, query string parameters, cookies, and file uploads.

1. **Validate input string on the server side** his is one of the most important steps to preventing SQL injection. Any data that a user can provide, whether via a web form, file, API, or other application needs to be cleansed and validated. This process will check user input for invalid characters, unacceptable length, or any other abnormalities prior to processing or storing it on any production systems.

The simplest step is for the application UI to detect invalid characters and provide instant feedback. Our example form from earlier provided a solid example of this behavior:



it provided a clear error message as to why my input was unacceptable. Username and password fields are often the dubious targets for this treatment, but realistically, ALL freeform input should be scrutinized for validity. include built-in features that will automatically evaluate user input for malicious content on page postbacks. But they can be circumvented by hackers with enough nerves and subtlety, so you should nonetheless run user input through your security check procedures. You can never be too cautious.

1. **Use command parameters**

A better alternative to escaping would be to use command parameters. Command parameters are defined by adding placeholder names in SQL commands, which will later be replaced by user input.

1. **Explicitly cast your input**

This tip is for languages such as PHP, which are weakly typed, which means you do not usually define data types for variables, and the language automatically takes care of converting different data types between each other.

Explicit casts can act as a shortcut to escaping input where non-string types are involved. So, if you’re expecting the user to input an int for the age parameter, you can ensure the safety of the input with the following code in PHP

1. **RESULT**

We performed logical incorrect queries of **SQL injection** we are successfully able to use the vulnerabilities of the SQL queries and we get inside the website breaking its security.

We can remove this vulnerability by having the SQL page conforming the username and password in the separate page so that the error result sent must not be appeared in the login window.

We can change our approach of password confirming by having the password checked only if the username of a particular person exists. Like we will run query for the person username then after that if the username exists then we will see its password and we will compare it with the given value. This will not allow the system to accept the query even if it is producing some results for the query.

Checking the username and password in another page, so that the SQL error is not showed in the login page.

Using another approach for the SQL query, instead of checking the both username and password in the same query, we can split it in two queries:

First checking any row exists with this username

If yes then check that the password is same as the retrieved one or not

This will help to prevent the logic to be true in the password checking time.

Converting the string of input values into mysqli\_real\_escape\_string() function as it, escapes special characters in a string for use in an SQL query, taking into account the current character set of the connection.

This function is used to create a legal SQL string that can be used in an SQL statement

1. **CONCLUSION**

We performed logical incorrect queries of **SQL injection** we gave a method to protect the website from this type of attack.