## Abstract

Existing vulnerabilities of Web system threaten the regular work of information systems. The most common Web system vulnerability is SQL injection. There is known approaches to protect Web applications against SQL injection attacks in the article. To improve the Web software security it is developed defense mechanism that protects Web resources from SQL injection performing. **To implement this software it is used PHP, JavaScript and formal language theory known as regular expression. As a result it is received a software tool which allows to protect Web software from SQL injection vulnerability.** Developed software tool allows user to protect his own Web application from an attack with using SQL.

***Keywords:*** SQL Injection, Web App Security, Detection Strategies, Scanning Method, Prevention of SQL Injection Attacks, Input Validation, Parameterized Queries, Escaping, Stored Procedures, Mitigation using Log-In Gateway System.

# Introduction

SQL injection attack is widely used by attackers to gain unauthorized access to systems. This software system is developed to prevent unauthorized access to system using SQL injection attacks. This is done by adding unique value and a signature based authentication technique to verify authenticity. SQL injection is a major security issue these days that allows an attacker to gain access of a web system or application exploiting certain vulnerabilities. This method exploits various web application parameters such as transmitting the traveling form data parameters with an efficient integration of amino acid codes aligned in it. In other words, this software project puts forth a method to analyze and detect the malicious code to find out and prevent the attack. It uses an alternative algorithm for signature based scanning method; this method is based on a different divide and conquers strategy that detects attacks based on various time/space parameters. This innovative system has proved successful in preventing various SQL injection attacks based on its efficient attack detection strategies.

# Literature Survey

| Sr. No. | Title (Year) | Authors | Algorithm/Method used | Advantages |
|---|---|---|---|---|
| 1 | A SQL Injection Detection Method Based on Adaptive Deep Forest | Qi Li, Weishi Li, Junfeng Wang, Mingyu Cheng | 1 An adaptive deep forest-based method is incorporated to detect the complex SQL injection attacks.<br>2 An algorithm, AdaBoost, algorithm is used. It is based on deep forest model which utilizes error rate to update the weights of features on each layer. | 1 Using this methodology, the existing problem of deterioration in efficiency of features of deep forests as the number of layers increases is tackled completely.<br>2 The structure of the tree model can be modified automatically.<br>3 Multi-dimensional fine-grained features can be dealt with properly to avoid the problem of over-fitting.<br>4 The methodology used experimentally demonstrates not only better detection accuracy and low computational cost but also high flexibility and high robustness. |

| | | | | |
|---|---|---|---|---|
| 2 | SQL Injection Detection for Web Applications Based on Elastic-Pooling CNN | Xin Xie, Chunhui Ren, Yusheng Fu, Jie Xu, Jinhong Guo | 1 Convolutional Neural Network (CNN) is a powerful deep feedforward neural network which can replicate the formation mechanism employed in organisms for vision cognition.<br>2 CNN is applied to the detection of SQL injection in Web applications, and detection of SQL injection attacks from very vast web logs is done. | 1 Identification of new and harmful attacks can be successfully done with the help of irregular matching characteristics and is thus, much more difficult to bypass.<br>2 Since the vocabulary is usually small, the training difficulty and cost are reduced considerably.<br>3 This method can also bebeneficially used as an auxiliary method of existing traditional SQL detection methods. |
| 3. | Web based testing application security system using | Akbar Iskandar, Muhammad Resa Fahlepi Tuasamu , | 1. This research uses semantic comparison method to detect SQL injection. Semantic comparisons are | 1. The design and testing of the system on the registration form shows that if there is field that has not been filled or empty on the registration |

| | | | | |
|---|---|---|---|---|
| | semantic comparison method | Suryadi Syamsu , M Mansyur , Tri Listyorini , Sulfikar Sallu , S Supriyono, Kundharu Saddhono , Darmawan Napitupulu and Robbi Rahim | performed by parsing each statement and comparing syntactic data structures. If the syntax structure of the two queries is equivalent, then the query induces equivalent semantic action on the database server.<br>2 Research based on a use case of a registration and login form. | form then it shows a message notification that the field must be filled and after all the fields are filled, semantic comparison will compare the query in the system database. The password created by the user has been encrypted and the encrypted password will be decrypted using MD5 decrypter where the data is encrypted using an encryption key to be something that is difficult to read by someone<br>2 Therefore, every website or web application needs to apply the semantic comparison method and apply Message Digest 5 (MD5) algorithm which is widely used to prevent the occurrence of attacks from creckers who are trying to find a web vulnerability because most applications accessed via the internet has a login page which can be used to authenticate app users.<br>3 The attacker intends to log in without using the correct username and correct password. But as if entering the correct username, where if the attacker uses injection like "hacker OR 1 '=' 1'-" as username and suppose "something" is used as password, then the query will be like this: Select * from login where user = 'hacker' OR '1' = '1' - 'and pass =' something 'When this query is run in the |

| | | | | |
|---|---|---|---|---|
| | | | | database, it will always be considered correct and authentication will work.<br>4 Based on the results and design of research, it can be concluded that using semantic comparison method can prevent dangerous Structured Query Language Injection Attack and can secure the account of web testing users by using MD5. |
| 4. | Using Parse Tree Validation to Prevent SQL Injection Attacks | Gregory T. Buehrer, Bruce W. Weide, and Paolo A. G. Sivilotti | 1.A parse tree is a data structure for the parsed representation of a statement. Parsing a statement requires the grammar of the statement's language. By parsing two statements and comparing | 1. Most web applications employ a middleware technology designed to request information from a relational database in SQL. SQL injection is a common technique hackers employ to attack these web-based applications. These attacks |
| | | | their parse trees, we can determine if the two queries are equal. When a malicious user successfully injects SQL into a database query, the parse tree of the intended SQL query and the resulting SQL query do not match. The programmer-supplied portion is the hard-coded portion of the parse tree, and the user-supplied portion is represented as empty leaf nodes in the parse tree. These nodes represent empty literals. | reshape SQL queries, thus altering the behaviour of the program for the benefit of the hacker. That is, effective injection techniques modify the parse tree of the intended SQL. The authors have illustrated that by simply juxtaposing the intended query structure with the instantiated query, we can detect and eliminate these attacks. They have provided an implementation of their technique in a common web application platform, J2EE, and demonstrated its efficacy and effectiveness. This implementation minimizes the effort required by the programmer, as it captures both the intended query and actual query with minimal changes required by the programmer, throwing an exception when appropriate |

| | | | | |
|---|---|---|---|---|
| 5. | Study On SQL Injection Attacks: Mode, Detection And Prevention | Subhranil Som, Sapna Sinha ,Ritu Kataria | 1. The paper has dealt with the security on the ends by proposing the two systems for avoiding SQL Injection Attacks. Its two stages are:<br>a) Frontend Phase<br>b) Backend<br>2. Initially at front end the Database is secured from any SQLIA. An additional section in client table is uesd to store the Final Hash Code, which is obtained during enrollment time of a client for the first time and is put into client table along with client name and secret key.<br>3. In the backend phase, The framework notices on how SQLIA on Web applications by tokenization and encryption for detection and prevention. | 1. This paper has demonstrated a strategy to change over SQL query into number of helpful tokens by applying tokenization and after that encoding all literals, fields, table and information on the query by AES-algorithm to avoid SQLIA.<br>2. This methodology encourages quick and proficient getting to system with database and keeps away from memory necessities to store the actual query in storehouse.<br>3. This methodology because of its low preparing overhead has immaterial impact on execution even at higher burden conditions and does not require real changes to application code. |
| | | | | |

## Proposed Work

- **Technique:**

An SQL injection is a technique that attackers apply to insert SQL query into input fields to then be processedby the underlying SQL database. These weaknesses are then able to be abused when entry forms allow user- generated SQL statements to query the database directly.

- **Harmful Threats Related to SQL Injection :**

1. Extraction of private data, such as credit cards, passports, hospital records, etc.
2. Enumeration of the authentication user details, allowing these logins to be used on other websites.
3. A corrupted database, execution of OS commands, deleted/inserted data and destroyed operationsfor the entire website.
4. Full system compromise.

- Let us Learn by example:

To give you a typical scenario, take a typical login form consisting of a user/email field and a password field.

After the login info is submitted, it is combined with an SQL query on your web server. In PHP, the commandis written in the following way:

```
$sql="SELECT * FROM lab where Uname='$Uname' AND Pass='$Pass' "; //20BCE2210
```

Sir simple hacker uses above **vulnerability** use this statement and vulnerabilities in it to enter their malicious logic and surpass the databaseverification systems.

**Following is the malicious Logic:**
**' OR 1=1--'**
When this logic enters the process it acts like a logic instead of a string and hence confusing the verificationlogics and getting verified henceforth.

**Truth Table :**
The malaicious logic plays with the verification system just as the given truth table demonstrates.

| A | B | OUT |
|---|---|---|
| FALSE | FALSE | FALSE |
| FALSE | TRUE | TRUE |
| TRUE | FALSE | TRUE |
| TRUE | TRUE | TRUE |

- **Projects to be Demonstrated:**

**Most Secure Login gateway Project** is the most secure way to login and is protected against sql injection.
**Sql Injection Testing Cum Prevention Project** is filled with vulnerabilities and serves as a perfect target for sqlinjection.

- **Prevention:**

With user input channels being the main vector for such attacks, the best approach is controlling and vetting user input to watch for attack patterns. Developers can also avoid vulnerabilities by applying the following mainprevention methods.

Following are the ways to Protect against SQL Injection:

1. **Input validation:**

The validation process is aimed at verifying whether or not the type of input submitted by a user is allowed. Input validation makes sure it is the accepted type, length, format, etc. Only the value whichpasses the validation can be processed. It helps counteract any commands inserted in the input string.In a way, it is similar to looking to see who is knocking before opening the door.

2. **Parametrized queries:**

Parameterized queries are a means of pre-compiling a 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 torecognize the code and distinguish it from input data.

3. **Stored procedures**

Stored procedures (SP) require the developer to group one or more SQL statements into a logical unit to create an execution plan. Subsequent executions allow statements to be automatically parameterized.Simply put, it is a type of code that can be stored for later and used many times.

4. **Escaping:**

Always use character-escaping functions for user-supplied input provided by each database management system (DBMS). This is done to make sure the DBMS never confuses it with the SQLstatement provided by the developer.

**Implementation**

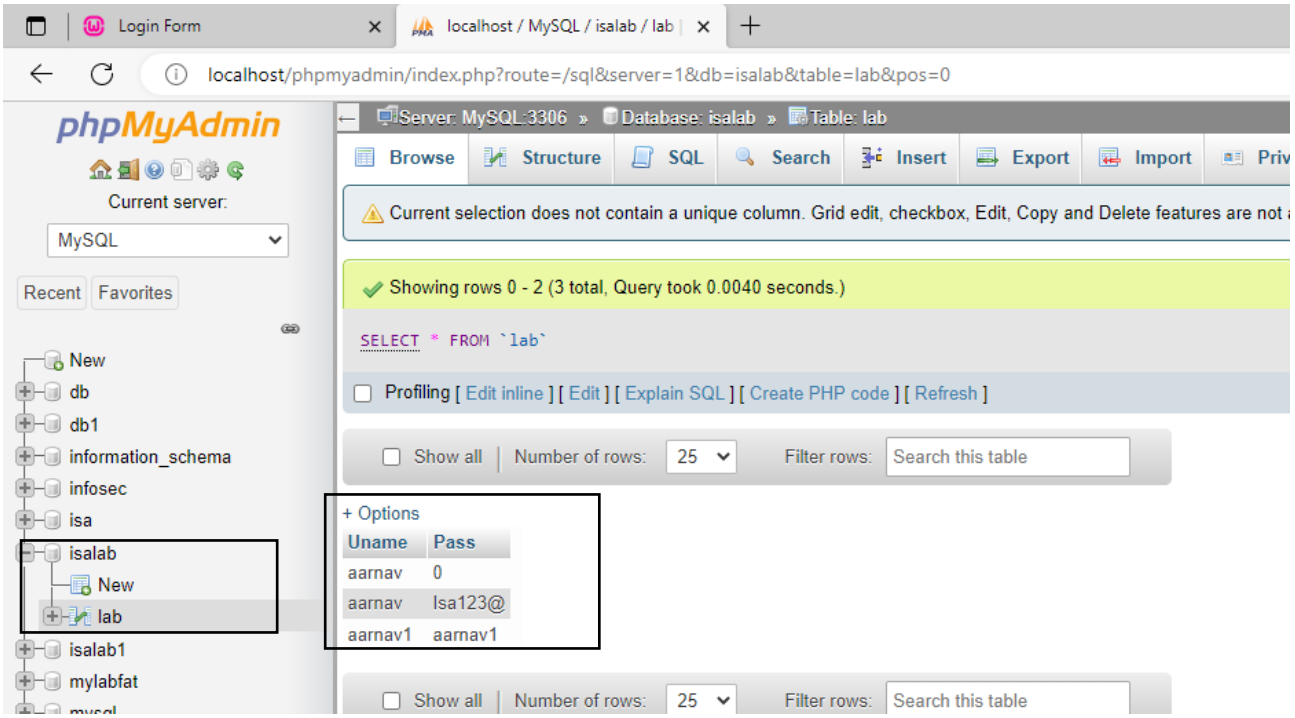
For the project I have used post statement and escaping technique which makes it the **most secure Log-In Gateway System an individual can design.**

It defies **any sort of malicious logic input by hackers** and will let only the genuine personwith correct credentials proceed further.

Most Secure Login gateway Project is the most secure way to login and is protectedagainst SQL injection.

SQL Injection Testing Cum Prevention Project is filled with vulnerabilities and serves as a perfect target for SQL injection.

Now sir I will show the database that I have used in the project ie isalab with table name lab:




**As we can see we have preset username and password in the database to verify the process**
**Now I will show simple implementation of our frontend login page**

**So as discussed above I will simply demonstrate how the gateway works without escaping technique**
**And how sql injection is successful in this case**

**Code for the Project:**
**Without Escaping technique**

```
<!DOCTYPE html>
<html>
<head>
    <title>Login Form</title>
    <link rel="stylesheet" type="text/css"
href="style.css">
</head>
<body>
<img src="vitlogo.png" alt="centred-image">
    <h1>20BCE2210 ISA</h1>
    <h2>Login gateway</h2><br>
    <div class="login">
    <form id="login" method="get"
action="login1.php">
        <label><b>User Name
        </b>
        </label>
        <input type="text" name="Uname" id="Uname"
placeholder="Username"
        ="<?php if (array_key_exists('Uname', $_POST))
{
         echo $_POST['Uname'];

}
?>">
        <br><br>
        <label><b>Password
        </b>
        </label>
        <input type="Password" name="Pass" id="Pass"
placeholder="Password"
        value= "<?php if (array_key_exists('Pass',
$_GET)) {
```

```
        echo $_GET['Pass'];
        }
?>">  <br><br>
      <button>Submit</button>

    </form>
</div>
</body>
</html>
```

```
<?php
$Uname =$_GET['Uname'];
$Pass = $_GET['Pass'];
$servername = "localhost";
$username = "root";
$password = '';
// Create connection
$conn = mysqli_connect($servername, $username,
$password,'isalab');
if (!$conn) {
 die("Connection failed: " . mysqli_connect_error());
}
$sql="SELECT * FROM lab where Uname='$Uname' AND
Pass='$Pass' "; //20BCE2210
$result = mysqli_query($conn,$sql);
$check = mysqli_fetch_array($result);
if(isset($check)){
 echo 'Success you have logged in!';


}
else{
 echo 'Failure, try again';
```

```
}
?>
body
{
    margin: 0;
    padding: 0;
    background-color:#6abadeba;
    font-family: 'Arial';
}
.login{
        width: 350px;
        overflow: hidden;
        margin: auto;
        margin: 20 0 0 450px;
        padding: 80px;
        background: #23463f;
        border-radius: 15px ;

}
h1{
    text-align: center;
    color: #277582;
    padding: 20px;
}
h2{
    text-align: center;
    color: #277582;
    padding: 20px;
}
label{
    color: #08ffd1;
    font-size: 17px;
}
#Uname{
```

```
    width: 300px;
    height: 30px;
    border: none;
    border-radius: 3px;
    padding-left: 8px;
}
#Pass{
    width: 300px;
    height: 30px;
    border: none;
    border-radius: 3px;
    padding-left: 8px;

}
#log{
    width: 300px;
    height: 30px;
    border: none;
    border-radius: 17px;
    padding-left: 7px;
    color: blue;


}
span{
    color: white;
    font-size: 17px;
}
a{
    float: right;
    background-color: grey;
}
img {
    display: block;
```

```
    margin-left: auto;
    margin-right: auto;
    height: 30%;
    width: 30%;
}
```

**With escaping technique ie Most Secures gate way and sql prevention technique**

```
<?php
$uname =$_POST['uname'];
$pass = $_POST['pass'];
// Create connection
/*$conn = mysqli_connect('localhost', 'root',
'','infosec');
if (!$conn) {
 die("Connection failed: " . mysqli_connect_error());
}*/
$db_connection = mysqli_connect("localhost", "root",
"", "infosec");
$username = mysqli_real_escape_string($db_connection,
$_POST['uname']);
$password = mysqli_real_escape_string($db_connection,
$_POST['pass']);
$query = "SELECT * FROM details WHERE uname = '" .
$username. "' AND pass = '" . $password . "'";
//$sql="SELECT * FROM details where username='$uname'
AND password='$pass' ";
$result = mysqli_query($db_connection,$query);
if (!$result) {
    printf("Error: %s\n",
mysqli_error($db_connection));
    exit();
}
$check = mysqli_fetch_array($result);

if(isset($check)){
```

```
 echo 'success';

}
else{
 echo 'Failure';

}
?>
<!DOCTYPE html>
<html>
<head>
    <title>Login Form</title>
    <link rel="stylesheet" type="text/css"
href="style.css">
</head>
<body>
<img src="vitlogo.png" alt="centred-image">
    <h1>20BCE2210 ISA</h1>
    <h2>Login gateway</h2><br>
    <div class="login">
    <form id="login" method="POST">
        <label><b>User Name
        </b>
        </label>
        <input type="text" name="uname"
placeholder="Username"
        value="<?php if (array_key_exists('Uname',
$_POST)) {
        echo $_POST['Uname'];

}
?>" >
        <br><br>
        <label><b>Password
```

```
        </b>
        </label>
        <input type="Password" name="pass"
placeholder="Password"
        value= "<?php if (array_key_exists('Pass',
$_GET)) {
        echo $_GET['Pass'];
}

?>">
        <br><br>
        <button type="submit" class="btn btn-
primary">Submit</button>

    </form>
</div>
</body>
</html>
```

- **Note:**

**The 1st part was filled with vulnerabilities which were later removed in the 2nd part showcased. This is an exceptional demonstration of Sql Injection with a live problem.**