

## 1. Introduction

Web applications can have sensitive and confidential data which is stored in a database, Web applications accept the data from the user, this data is retrieved from the database through the queries, SQL Injection attack is one of the most popular attack used in system hacking or cracking, Injecting web applications means having illegal access to data stored in database, In this chapter we try to explain the meaning of SQLI and select the hotspots where SQLI happen, in order to build a secure web application and avoid the vulnerability.

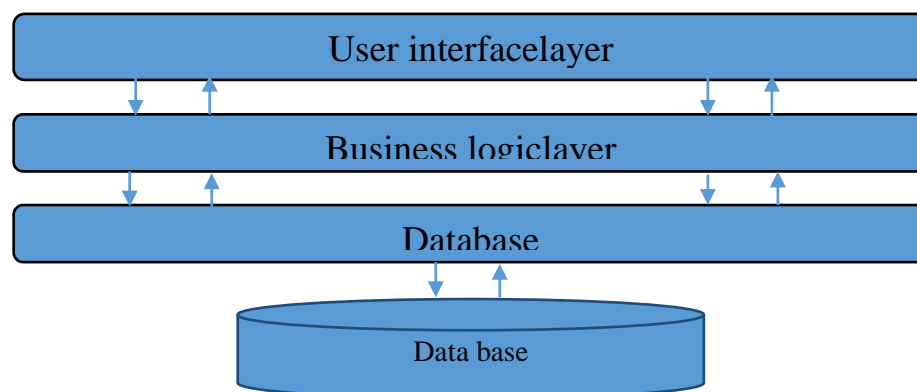
## 2. Webapplications

A Web application refers to any program that is accessed over a network connection using HTTP instead of existing within a device's memory. Web applications often run inside a Web browser. However, Webapplications also may be client, where a small part of the program is downloaded to a user's desktop, but processing is done over the Internet on an external server[7].

### 2.1. 3-tier Architecture of web application

In order to select the hotspots where SQLI happen, we first discuss the 3-tier architecture of web applications:

- ✓ User interface tier : This layer forms the front end of the web application it interacts with the other layers based on the inputs provided by the user.
- ✓ Business logic tier: The user request server side and its processing are done here. It involves the server side programming logic, it Forms the intermediate layer between the user interface tier and the database tier.
- ✓ Database tier : It involves the database server, it is useful in storage and retrieval of data.[9]



**Fig .1.1.** Web 3-tier architecture

## **2.2. Web application security**

Most of the attacks on web application are performed to steal confidential data or to deface website or stealing session cookies. And reason for this attacks are some vulnerabilities in web application coding and in application developing methods. This vulnerabilities from which on some of very severe damaging attack can be perform. So while developing application itself developer can avoid that vulnerability.

Here are several Web application attacks that can be used to take control over application or deface application or stealing confidential data. According to OWASP(Open Web Application Security Project) there are 10 attacks that are most severe and dangerous for any web application. That attacks are as follows[7, 8] :

### **2.2.1.SQL Injection**

Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.

### **2.2.2.Cross Site Scripting (XSS)**

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

### **2.2.3.Broken Authentication and Session Management**

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit implementation flaws to assume other users' identities.

### **2.2.4.Insecure Direct Object References**

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

### **2.2.5 Cross Site Request Forgery (CSRF)**

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

### **2.2.6 Security Misconfiguration**

Security depends on having a secure configuration defined for the application, framework, web server, application server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults.

### **2.2.7 Failure to Restrict URL Access**

Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks when these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway.

### **2.2.8 Invalidated Redirects and Forwards**

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

### **2.2.9 Insecure Cryptographic Storage**

Many web application do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may use this weakly protected data to conduct identity theft, credit card fraud, or other crimes.

### **2.2.10 Insufficient Transport Layer Protection**

Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly.

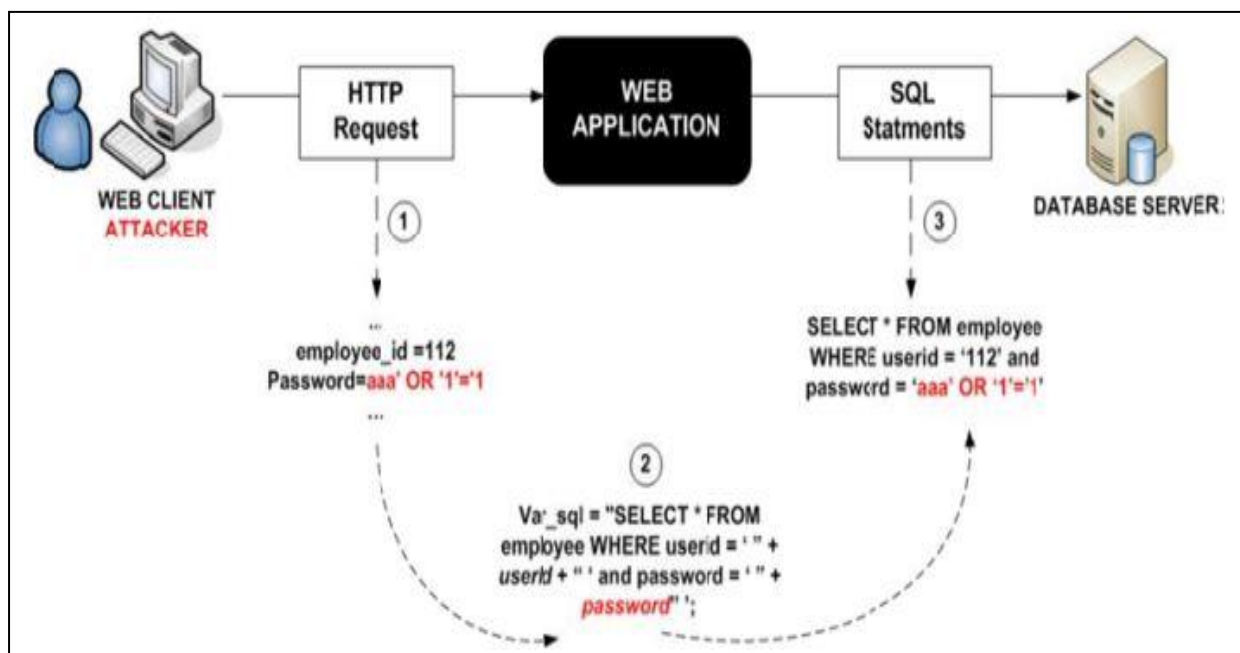
### 3.SQLInjection:

#### 3.1 SQL injection Definition

A SQL injection is a kind of web application security exposure, in which an attacker is able to expose a database SQL command.

SQL injection is nothing but injection malicious queries by the hackers into the web application to get the desired output from the database, The SQL injection also allows an attacker to create, read, or update or delete data stored in the database.

SQL injection can take place when the web application utilizes use –supplied data without validation,(attacker provides specially crafted input data)where is sent to the SQL interpreter as a piece of an SQL query, and trick the SQL interpreter to execute orders, in such a way the interpreter is unable to distinguish between the actual command and the attacker's specially crafted data[2].



**Fig.1.2.**Example of a SQL injection attack[2].

#### 3.2 Consequence of SQLInjection

The results of SQLIA can be disastrous because the successful SQL injection can read sensitive data from the database, modify database data (insert/update/delete), execute administrative operations on the DB, recover the contents on the DBMS file system, the main consequence of these vulnerabilities are attack on[2]:

- **Authorization:** The important data that is stored in a vulnerable SQL database may be changed by a successful SQLI, as an authorization privilege.
- **Authentication:** If there is no any proper control on username or password inside the authentication page, it may be possible to login to a system as a normal user without knowing the right username or password.
- **Confidentiality:** Databases are usually contain of sensitive data like personal information or credit card numbers, therefore, the loss of confidentiality is a big problem with SQL injection vulnerability
- **Integrity:** With a successful SQLI not only an attacker reads sensitive information, but it is also possible to change or delete this private information

## 4. The SQL injection mechanisms

Protecting a web application against unchecked input vulnerabilities is difficult because applications can obtain information from the user in a variety of different ways. One must check all sources of user-controlled data such as form parameters, http headers and cookie values systematically, while commonly used client-side filtering of malicious values is not an effective defence strategy[6].

### 4.1. Parametertampering

The most common way for a web application to accept a parameter is through HTML form, when a form is submitted, parameters are sent as part of an HTML request. An attacker can easily tamper with parameter passed to a web application by entering malicious crafted values into the text field HTML form.

### 4.2. URL tampering

For the HTML form that are submitted using the HTTP GET method, form parameter from as well as their values appear (show) as part of the URL that is accessed after the form submitted. An attacker may directly edit the URL string, with malicious data in it, and then enter this malicious data to the application.

**Example:** Consider a web page at a bank site that allows an authenticated user to select one of the accounts from a list and debit of 100\$ from the account. When the submit button is pressed in the web browser, the following URL is requested:

[http://www.mybank.com/myaccount?accountnumber=341948&debit\\_amount=100](http://www.mybank.com/myaccount?accountnumber=341948&debit_amount=100)

If no additional precautions are taken by the web application receiving this request, access may in fact increase the accounts balance.

```
http://www.mybank.com/myaccount?accountnumber=341948&debit_amount=-5000
```

### 4.3. Cookie poisoning

Cookie poisoning attacks consist of modifying a cookie, which is a small file accessible to Web applications stored on the user's computer. Many Web applications use cookies to store information such as user login/password pairs and user identifiers. This information is often created and stored on the user's computer after the initial interaction with the Web application, such as visiting the application login page.

For example, consider the HTTP GET request in figure 3 the URL on host <http://www.mybank.com> requested by the browser transfer and the string transferparameter=yes indicates that the user wants to perform funds transfer. The request includes a cookie that contains the following parameters:

SESSION which is a unique identification string that associates the user with the site and amount, which is the transfer amount for this transaction. The amount is validated by the web application before being stored in a cookie, however, an attacker can easily edit the cookie and change the amount value in order to circumvent account overdraft checks that are performed before the cookie is created to transfer more money that is contained in an account.

```
GET transfer?complete=yes
HTTP/1.0 Host: www.mybank.com Accept: */*
Referrer: http://www.mybank.com/login
Cookie: SESSION=89DSSSXX89JJSYUJG; Amount=5000
```

**Fig.1.3.** an HTTP GET request containing a cookie GET [6].

### 4.4. Hidden field manipulation

Because http is stateless, many web applications use hidden field to emulate persistence, hidden fields are just form made invisible to the end-user.

For example consider an order form that includes a hidden field to store the prices of items in the shopping cart:

```
<input type="hidden" name="total_price" value="25.00">
```

A typical web site using multiple form, such as an online store will likely rely on hidden fields to transfer state information between pages for instance [Amazon.com](https://www.amazon.com).

Unlike regular fields,hidden fields cannot be directly modified by typing values into an HTML form however since the hidden field is part of the page source saving the HTML page, editing the hidden field value, and reloading the page will cause the web application to receive the newly update value of the hidden field,This attack technique is commonly used to forge information being sent to the web application and to mount SQL injection.

#### 4.5. HTTP header manipulation

HTTP headers typically stay invisible to the user and are only by the HTTP browser and the web server however, some web applications do process these headers, and attackers can inject malicious data into application through them. While a normal web browser will not allow forging the outgoing headers, multiple freely available tools allow a hacker to craft an HTTP request leading to an exploit.

<pre>con.executeUpdate("UPDATE EMPLOYEES " + " SET SALARY = " + salary + " WHERE ID = " + id);</pre>	<pre>PreparedStatement pstmt = con.prepareStatement( "UPDATE EMPLOYEES " + " SET SALARY = ? " + " WHERE ID = ?");  pstmt.setBigDecimal(1, salary); pstmt.setInt(2, id);</pre>
(a)	(b)

**Fig.1.4.**two different ways to update an employee's salary :(a) may lead to a SQL injection (b) safely updates the salary using a preparedstatement [6].

The **accept language** header indicates the preferred language of the user,an web application may take the language label from the HTTP request and pass it to a database to look up a language specific text message, if this header is sent to the database, an attacker may inject SQL commands by modifying the header value,Also if the header value is used to build a file name with message for the correct language an attacker may be able to launch a path traversal attack.

## 5. SQL injection attack types

There are different methods of attacks that rely on the aims of the attacker performed, and the classification is given as below[1]:

### 5.1. Tautologies

This type of attack injects SQL tokens to the conditional query statement to be always evaluated true. This type of attack is used to avoid authentication control and access to data by exploiting a vulnerable input field which uses the WHERE clause.

#### Example

```
SELECT * FROM employee WHERE userid= '211' and password='bbb' OR '1'='1'
```

The tautology statement (1=1) has been added to the query, so the statement is always true.

### 5.2. Logically incorrect queries

When a query is not needed, an error message is returned from the database including useful debugging information. This error message helps the attacker to discover vulnerable parameters in the application and consequently the database of the application. In fact, the attacker injects junk input or SQL token in query to produce a syntax error.

#### Example:

✓ Original:

```
URL:http://www.arch.polimi.it/eventi/?id_nav=886
```

✓ SQL injection :

```
URL:http://www.arch.polimi.it/eventi/?id_nav=8864'
```

✓ Error message showed :

```
SELECT name FROM employee WHERE id=8864 from the message error we can find  
out name of table and fields: Name: employee; id.
```

By the gained information the attacker organizes more strict attacks.

### 5.3. Union Query

By this method attackers join injected query to the safe and sound query by the word UNION and then can get to other tables from the application. The output of this attacker is



that the database returns a dataset that is the union of the results of the original query with the results of the inject query.

**Example:**

✓ By

```
SELECT Name, Address FROM Users WHERE Id=$id
```

```
$id=1 UNION ALL SELECT creditCardNumber, 1 FROM CreditCarTable.
```

✓ We will have the following query:

```
SELECT Name, Address FROM Users WHERE Id=1  
UNION ALL SELECT creditCardNumber, 1 FROM CreditCarTable
```

Which will join the result of the original query with all credit card users.

**5.4. Blindinjection**

Sometimes developers hide the error details which help attackers to compromise the database. In this situation, the attacker faces a general page provided by the developer instead of an error message. So, the SQL injection attack would be very difficult but not impossible. The attacker can still steal data by asking a series of true/ false questions through a SQL statement.

**Example:**

```
SELECT accounts FROM users WHERE login='doe' and 1=0--AND  
pass=AND pin=o
```

```
SELECT accounts FROM users WHERE login='doe' and 1=1-- AND  
pass=AND pin=o
```

If there is input validation, both queries would be unsuccessful. But if the application is not secured, so the attacker can use this opportunity. First, the attacker submits the first query and receives an error because of `1=0`. So the attacker does not understand the error in input validation or for logical error in query. Then, the attacker submits the second query which is always true. If there is no login error message, the attacker finds the login field vulnerable to injection.

### 5.5. piggy-backed queries

The piggy-backed query attacker tries to add additional queries to the original query string. In this method the first query is original while the following queries are injected by the query delimiter such as ";" to attach extra query to the original query. If the attack was successful that mean execute a multiple queries (original and injected query).

#### Example:

✓ Original :

```
SELECT * FROM user WHERE login= '$login' AND pass='$pass'
```

✓ By injecting

```
\'; DROP TABLE users ; --
```

✓ We will have the following query:

```
SELECT * FROM user WHERE login= '\'; DROP TABLE' users; --' AND pass= ''
```

Because of ";" character, the database accepts both queries and executes them, the second query is illegal and can drop the users table from the database. It is clear that some databases do not need a special separation character in multiple distinct queries.

### 5.6. Stored procedure

In this technique, the attacker focuses on the stored procedures which are present in the database system. Stored procedures run directly by the database engine. Stored procedures are nothing but codes and it can be vulnerable as programme codes. For authorized – unauthorized users, the stored procedure becomes true or false, as a SQL injection attack intruder inputs SHUTDOWN for username or password then the stored procedure generates the following query:

#### Example:

```
SELECT accounts FROM users WHERE login= '1111' AND pass='1234  
'; SHUTDOWN;--;
```

This type of attack works as piggyback.

## 5.7. Timing attacks

The timing attack lets the attacker gather information from a database by observing timing delay in the database. This technique by using if-then statement causes the SQL engine to execute a long running query or time delay statement depending on the logic injected. This attack is similar to blind injection and the attacker can measure the time the page takes to load to determine if the injected statement is true. Such a technique uses an if-then statement for injecting queries. WAITFOR is a keyword along the branches which causes the database to delay its response by a specified time.

### Example:

In the following query

```
declare @ varchar (8000) select @ = db_name() if (ascii(substring(@, 1, 1))  
& ( power(2, 0))) > 0 waitfor delay '0:0:5'
```

The database will pause for five seconds. If the first bit of the first byte of the name of the current database is 1, then the code is injected to generate a delay in the response time when the condition is true. In addition, the attacker can ask a series of other questions about this character. As these examples show, the information is extracted from the database using a vulnerable parameter.

## 6. SQL injection prevention mechanism

There are a lot of techniques available to cancel SQLI attacks. Here we talk about some techniques used to prevent the SQL injection[4]:

### 6.1. Defensive coding practices

#### 6.1.1. Input type checking

SQLI attacks can be performed by injecting commands into a string or numeric parameter, so a simple check of such an input can prevent many attacks.

#### 6.1.2. Encoding of inputs

An injection into a string parameter is often accomplished through the use of meta-characters that trick the SQL parser into interpreting the user's input. So the solution is to use the function that encode a string in such a way that all meta characters are specially encoded and interpreted by the database as normal characters.

### **6.1.3. Positive pattern matching**

Input validation should be able to identify all good inputs as opposed to all bad inputs, because the negative validation is not always possible due to the new type of attack signature. The best solution remains to implement the positive validation.

### **6.1.4. Identification of all input points**

Developers must check all input points to their application. There are many possible sources of input to an application. If used to construct a query, these input sources can be a way for an attacker to introduce an SQLI. So, all input sources must be checked.

## **6.2. Black box testing**

The web vulnerability scanner is used for the black box testing. The vulnerability scanner is used for finding the loop holes in the existing application. It mainly visits the web application's input point and simulates the attack. If the attack is possible successful then, it is summarized in the form of a report.

## **6.3. White box testing**

The static code analyzers are used for the white box testing. The static code analyzers basically analyse the byte code of the web application with the intention of finding the vulnerability.

## **6.4. Run time monitoring (control)**

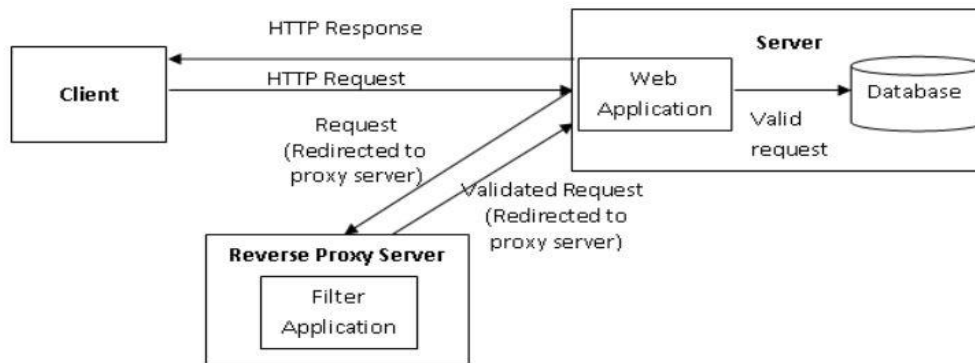
For the run time monitoring, the IDS (Intrusion Detection system) can be used. The latter is based on machine learning technique that is trained using a set of typical application queries. The technique builds models of the typical queries and then controls the application at run time to identify queries that do not match the model

## **7. Existing solutions**

### **7.1. Proxy filtering**

A proxy filtering system that focuses on input validation rules on the data of a web application is called a security Gateway. This technique transfers parameters from a web page to an application server. The developer should use a security policy descriptor language (SPDL).

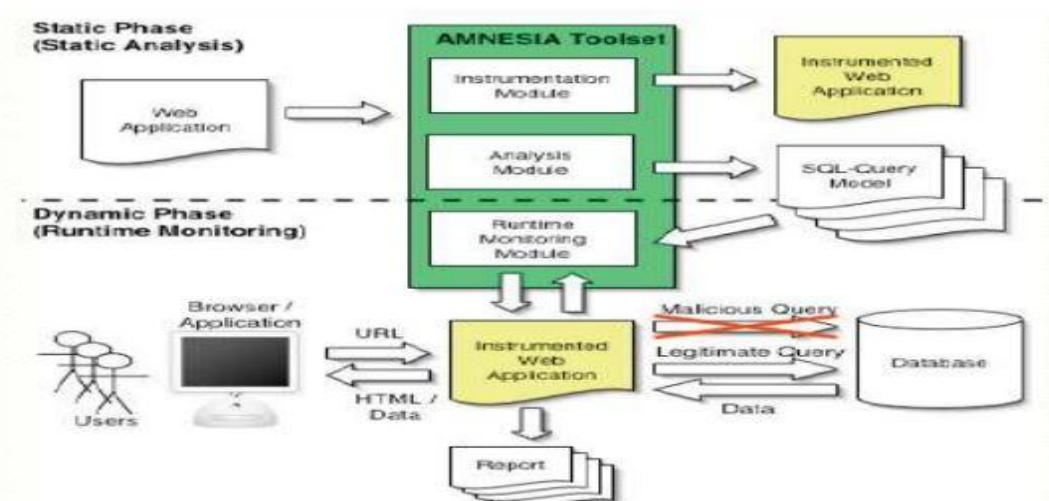
So, a developer has to know which data should be filtered and also what type should apply to the data[5].



**Fig.1.5.** Proxy filtering architecture[5].

## 7.2.AMNESIA

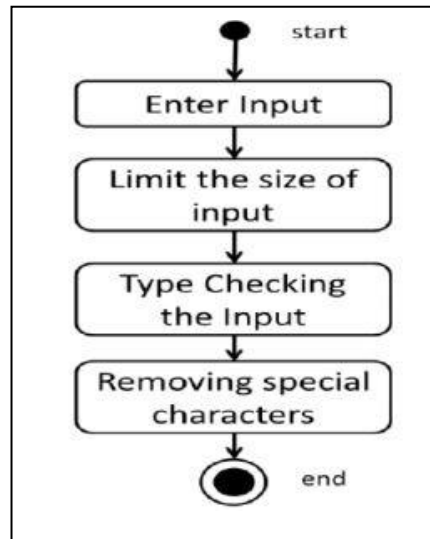
It combines static analysis and runtime monitoring. In the static phase, it builds models of the different types of queries that an application can legally build up at each point of access to the database. Queries are intercepted before they are sent to the database and are checked against the statically built models. In the dynamic phase, queries that violate the model (the query which not matching with the model) are prevented from accessing the database, the primary limitation of this tool is that its success is rely on the accuracy of its static analysis for building query models[3].



**Fig.1.6.** AMNESIA architecture[3].

### 7.3. Client side validation

Using a client side script validation such as JavaScript, a lot of SQL injection attacks can be prevented in Web applications. Though this approach does not solve all the attack types it is necessary to provide the basic security to prevent illegal attacks. The sequence of steps that increase the level of security in case of vulnerabilities is depicted in the activity diagram[10].



**Fig.1.7.** Activity Diagram for Client Side Validation[10].

The advantage of a client side validation is that it reduces CPU cycles since it avoids a number of round trips to the server. Some of the steps involved in client side validation include limiting the input size, restricting the use of special characters etc. But limiting the size of the input and restricting the use of special characters cannot be imposed on users in all applications.

In addition, the protection provided by client side scripts can be easily bypassed. The use of this approach can solve attacks using tautology or incorrect queries. It cannot solve the threat posed by blind injection techniques. Hence we prefer server side validation techniques.

## 8. Conclusion

It is important to know how to identify SQL injection vulnerabilities because the majority of data breaches is due to poorly coded web application. Any code that focuses on SQL statements should be reviewed for SQL injection vulnerabilities since, a database server will execute all queries that are valid and keep in mind the parameterized data can be manipulated by skillful attackers.

Therefore, web application should be built with security in mind and regularly tested for SQL injection vulnerabilities.