6.1 Introduction
In this world of Information technology, where E-commerce is most prevailing, the need for secure and safe data on Internet is must. Web applications, which are the foremost way of accessing data from web, are highly vulnerable to SQL Injection Attacks. Such applications and their underlying databases often contain confidential or even very sensitive information such as customer and financial records. With the increase in the availability and popularity of database driven web applications, there is a corresponding increase in number and sophistication of attacks that target them. Therefore it is very difficult to prevent these applications from attackers in order to save the critical information being hacked [111] [112].

In this thesis, I propose a Potential Framework to Secure Web Application and Database against SQL Injection Attacks and also to eliminate SQL Injection vulnerabilities. The proposed framework is identifying SQL Injection vulnerabilities on the basis of SQL Injection grammar [117] [118].

6.2 A Potential Framework to Secure Web Application against SQL Injection Attacks
The purpose of proposed framework “SQL Injection Detection Framework” (SQLID Framework), is to analysis the SQL injection attack on web applications, see figure 6.1 [117] [118]. The SQLID Framework step by step as described below:
6.2.1 STEP 1: To Detect Vulnerabilities:
The first step of SQLID framework detects the loopholes of SQL injection attack present the following two cases [117] [118]:
1. The loopholes are first detected by the automation tool of framework. The script parses the provided URL, and modifies parameters to simulate simple SQL. If resulting page contains error message generated by database management system (like MySQL,
MSSQL, etc.) then the SQLID framework halts its processing, confirms the completion and produce a warning. If the search results in failure, step (2) comes into existence.

II. The Authentication mode of the framework is kept “ON” by default. It apprehends vulnerabilities automatically in order to provide ancillary security to the targeted applications. If collaring of vulnerability leads to the failure in authentication mode also, the application is free from the undesirable activities that could harm it.

6.2.2 STEP 2: Analysis the Existing Vulnerability:

The vulnerabilities so collared in Step 1, are now analyzed deeply in order to check whether it is vulnerable to SQL Injection Attacks or not. The following parameters are kept into consideration to segregate the vulnerabilities that corresponds to SQLIA, see figure 6.2.

I. Language Used:

The computer programming language (like SQL, PL/SQL etc) is used to create queries that can be inserted into web application to attain SQL injection Attack.

II. Structure:

The query that can leads into SQLIA may includes:

a. Special Characters: ‘’, !, @, #, &, etc.

b. Operators: *, +, -, / etc.

c. Logical Characters: OR, NOT, LIKE etc.
6.2.3 STEP 3: Categorization of SQL Injection Attack:
The enmesh SQL Injection so founded are now categorized in order to comprehend the attacks more gently. The categorization of SQL Injection Attack is an exigent. It provides the developer and analyst a deep knowledge of attack and about its type that it may fall into. This helps the developer to overcome the affect of or to completely avoid the occurrence of SQL Injection Attack [117] [118].

6.2.4 STEP 4: Filter the Vulnerabilities Corresponding to SQL Injection Loopholes:
I am using FILTER in between web application server and database server to filter out the abnormal or malformed or bad SQL injection queries. If username and password are making SQL injection query then FILTER will not pass SQL injection query to database server and client side web page will show username and password are invalid.
Filter will check that the username and password are not a combination of operators, special characters and specific characters [113][118]. For example:
1. ‘or 0=0 #
2. ‘) or (’x’=’x
3. ‘or1=1--
4. hi’ or ‘a’=’a
5. hi” or 1=1 –
6. hi”) or (”a”=”a
7. ‘) or (’a’=’a
8.” or 0=0 #

Apart from the few SQL injection queries mentioned above, these operators can be used as username and password:
1. **ORIFICE**   2. **MLIKE**   3. **MOORE**

If operators with equal, special characters or specific characters become SQL injection queries, then FILTER will discard username and password. A single operator can not make SQL injection query. In this case, FILTER will allow only the single operator to become username and password and not the special characters and specific characters.

Filters can also check single line comment, multiple line comment, concatenate (double pipe) in the fields of username and password. If any comment, among those that are discussed above is present in username and password then filter discard the entered values, see figure 6.3 [113].

I.       -- or #   single-line comment
II.      /*...*/   multiple-line comment
III.     ||        concatenate (double pipe)
Figure 6.3 Filter

Internal Working of Filter:
The three components in the architecture of our proposed filter is as follows:

a) User Login Interface   
b) SQL Injection Protector for Authentication   
c) User Account table

User Account table is used to store the users’ account data. SQL Injection Protector for Authentication (SQLIPA) is the arc component of architecture (figure 6.4). SQLIPA component generates the hash values of user name and password first time during the creation of a user account (see in figure 6.5 & 6.6) and check against stored hash value. If both are matched, access to the account is granted otherwise access is denied.
Chapter – 6 A Potential Framework To Mitigate SQLIAs

Figure 6.4 Internal Architecture of Proposed Filter

Figure 6.5 Login Form
6.3 A Coalesce Model to Secure Database Against SQL Injection Attacks

The various features of all the models and techniques proposed, when combined together under one model, is bound to give a Security System for the database that is complete in itself, see figure 6.7. The multilevel access control of security system can be handled using data groups, for providing security at different levels, in a fine grained access control structure using security criterion expression. The two-phase locking technique and time-stamp ordering can be used in a distributed database environment to manage concurrency. Apart from the usual functioning it also needs to be protected from attacks. In order to protect the database from internal as well as external attacks, the user needs to login twice while access grants will be granted only on the basis of the user’s type. But instead of creating a wall around the web server, it is better if a check point is created between web server and database server and an application profile created for every executionable path for that profile. Column-based encryption is useful
for every numeric transaction along with the implementation of the 3Kdec algorithm. The summary of the model has been given figure 6.8 [114].

Figure 6.7 The Coalesce Model
6.4. How to Mitigate SQL Injection Attacks at Design Level?

It is incumbent to mitigate SQL Injection Attacks so that a malicious user cannot breach the security of any website or database that contains confidential data. The mitigation of SQL Injection Attack is carried out at Software and Hardware Design level, see figure 6.9 [117] [118].

6.4.1. At Software Design Level:

At software design level, a model named Software Security Design Model (SSDM) has been proposed. The SSDM follows an elemental process which is explained in steps written below.
I. Analyze SQL Query:
The structure of SQL Query commands is analyzed. This can be done either manually or using SQL Query analyzer tools. This allows programmers to determine and attain the deep knowledge of what specific part of query is bestowing ail performance.

II. Build a Parser:
The SSDM now builds a parser in which correct syntax and semantic of SQL Query is checked. The Parsers are written manually or generated by parser generators. The main objective of building a parser is to obtain error free SQL Query and produce a correct source code which is recognized and processed by computer systems only.

III. Create Proxy Server:
The proxy server routes requests of client to application server nodes. It does not reveal data about the original requester and thus obfuscate activities of SQL Injection Attack [115].
6.4.2. At Hardware Design Level:
The Hardware Security Design Model (HSDM) has been proposed to gently work out at hardware security design level. The arc focus of HSDM shifts entirely on the following two factors:

I. System Configuration:
All the hardware and software basic requirements are checked which are necessary to run the web applications and to create, access and manage the database.

II. Compatibility Issues:
The software and hardware compatibility is a major issue which is generally being ignored by many developers and programmers. For a success completion of any project it is important to attain system compatibility. The HSDM checks if the platform on which a programmer is working, is entirely compatible with the operating system and hardware components of the computer.

6.5 Customized Error Message
Error message also known as Blind SQL Injection attack. The binge of informative error message may accommodate the knowledge to access the database to the user. But it is difficult task for debugging if I try to remove error messages completely. Customized error message hinder the reconnaissance progress of threat agents, particularly in deducing specific details such as injectable parameters etc. [113] [116].

6.6. POST Method
For sending data to the server, the POST method is used. In this method, along the request object, the query string is appended but not in URL. That is why; transferable parameters are in the hidden form [113] [116].

6.7 Additional Features
Besides adopting proposed techniques, I also recommend to adopt the given below additional defenses, in order to provide security in depth [113] [116].

- Least Privilege
- White List Input Validation
6.7.1. Least Privilege:
To minimize the potential damage of a successful SQL injection attack, one should minimize the privileges assigned to every database account in their environment. Do not assign DBA or admin type access rights to one’s application accounts.

6.7.2. White List Input Validation:
It is always recommended to prevent attacks before the processing of the user’s (attacker's) request. Input validation can be used to detect unauthorized input before it is passed to the SQL query. Developers frequently perform black list validation in order to try to detect attack characters and patterns like the ‘ character or the string 1=1. But this is a massively flawed approach as it is typically trivial for an attacker to avoid getting caught by such filters. In addition, such filters frequently prevent authorized input, like O'Brian, when the character is being filtered out.