Advanced Detecting and Defensive Coding Techniques to prevent SQLIAs in Web Applications: A Survey

Vinod Kumar .K, Jatin Das .D

Abstract— SQL injection attacks are more dangerous than other web attacks because these attacks can get sensitive data stored in the database by manipulating the original SQL queries. In spite of different tools and frameworks to detect and prevent SQL Injection, it is still a top most threat to web applications. In this paper, we provide detailed survey of different coding techniques along with recent trends in detecting and preventing SQLIAs' that can be used to develop secured web applications.

Index Terms— Web applications, SQL Injections.

I. INTRODUCTION

Nowadays, web applications are more prevalent around the world. More and more companies and organizations use web applications to provide various services to users. Web applications receive users' requests from the browser, interact with the database, and return relevant data for users.

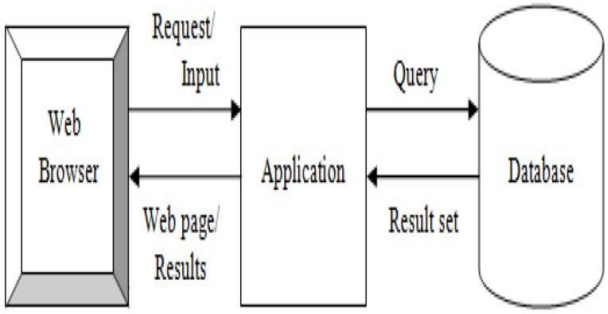

Fig. 1. Structure of Web Application (REQUEST/RESULT)

The back-end database often contains sensitive user data that interest attackers. To compromise a database, SQL injection is one of the techniques used by attackers. These attacks exploit vulnerabilities existing in web applications. It allows attackers to inject harmful SQL query segments in the application oriented queries, so that attackers can obtain unauthorized access to a database. This unauthorized user can read or modify existing data, make the data unavailable to other users, or even corrupt the database server. According to

OWASP report released in 2013, Injection attacks are top most threat to Web applications [1]. Web applications and their underlying databases require not only careful configuration and programming to assure security, but also effective protection mechanisms to prevent attacks. Researchers have proposed various solutions and techniques

to address the SQL injection problems. However, there are many solutions that can prevent SQLIAs, but researchers are more interested in analysing and detecting the SQLIAs. This research will present a survey of different advanced SQLIAs detecting, preventive and efficient coding techniques to avoid SQL Injection attacks in Web applications.

II. BACKGROUND

I. Web Application Environment

Web application data is presented to the Server by the client, in the form of forms, cookies and URLs' using different methods. These inputs contains both logical data for the application and the queries those applications send to a database to extract relevant data.

Present, Web applications do not adequately validate clients input with respect to SQL injection. Using those flaws in the application attackers attempt to get sensitive information about the users from the databases other than what the application intended.

II. Overview of SQL Injection Attack

SQL injection attacks are nothing but injecting malicious queries by the attacker into the application intended queries to get the desired outputs from the database.

The following code explains SQL injection attack using tautology.

"SELECT * FROM users WHERE name = '" + userName + "';"

Instead of providing genuine user_name, attacker uses the following code to manipulate the original query.

'or '1'='1'—'

Now the meaning of the manipulated query will be

"SELECT * FROM users WHERE name = '' or '1'='1'—';"

The term, ' or 1=1 --, does two things. First, it causes the first term in the SQL statement to be true for all rows of the query; second, the -- causes the rest of the statement to be treated as a comment and, therefore, ignored. The result is that all the details in the database, up to the limit the Web page will list, are returned. This is a very basic injection attack. The hardcore attackers would use very logical and efficient 'terms' to get the desired output.

Manuscript received May 01, 2013.

Mr.Vinod Kumar Kottem, Computer Science and Engineering, Sree Vidyanikethan Engineering College, Tirupathi, A.P, India.

Prof. Jatin Das .D, Computer Science and Engineering, Sree Vidyanikethan Engineering College, Tirupathi, A.P, India

III. TYPES OF SQL INJECTION ATTACKS

Generally SQLIAs' are classified into three types, In-Band, Out-Of-Band, Inference. In-Band attacks are those attacker interacts with the website or web application directly, where as Out-Of-Band attacks are those which uses third party data to attack a web application. Most of the SQL injection attacks fall into these categories. The SQL injection attacks can best be understood through a variety of examples demonstrating the various SQL injection attacks [2].

TABLE I- Types of SQLIA's

Type of Attack	***Procedure***
Tautologies	SQL injection codes are injected into one or more conditional statements so that they are always evaluated to be true
Union Query	Injected query is joined with a safe query using the keyword UNION in order to get information related to other tables from the application
Logically Incorrect Queries	Using error messages rejected by the database to find useful data facilitating injection of the backend database.
Stored Procedure	Many databases have built-in stored procedures. The attacker executes these built-in functions using malicious SQL Injection codes.
Piggy-Backed Queries	Inserting two or more queries into one query
Inference - Blind Injection - Timing Attacks	An attacker derives logical conclusions from the answer to a true/false question concerning the database. - Information is collected by inferring from the replies of the page after questioning the server true/false questions. - An attacker collects information by observing the response time of the database.

A. Tautologies

This attack works by inserting an "always true" statement into a WHERE clause to extract data. These are often used in combination with the insertion of a – to cause the remainder of a statement to be ignored ensuring extraction of largest amount of data. Tautological injections can be string type or numerical type or comment type expression-snippets, as demonstrated by the following examples:

Fig. 2. Normal Output for given input

Numerical : '101' OR '1'='1'
String : 'vinod' OR 'x'='x'
Comments:'101' OR '1'='1- -'

Injection Example:

Query: **"select x,y,z from emp where e_code="+lvalue+" "**
Input: **'255' OR '1'='1'**
Query: **"select x,y,z from emp where e_code= '255' OR '1'='1' "**

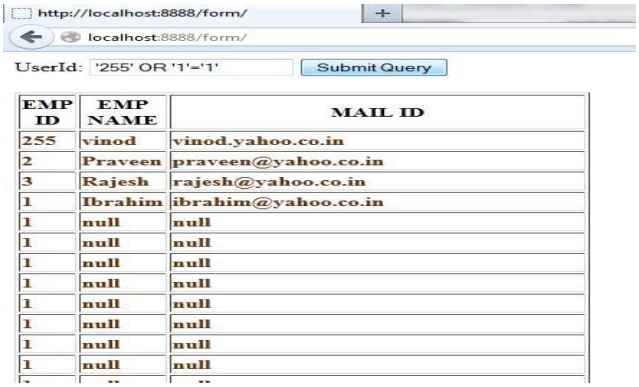

Fig. 3.Exploited Output for Malicious input

B. Union Query

This attack exploits a vulnerable parameter by injecting a statement of the form:

SELECT * FROM users WHERE login='' *UNION SELECT Phno from emp where e_code= '255' OR '1'='1- -''* AND pass=''

The attacker can insert any appropriate query to retrieve information from a table different from the one that was the target of the original statement. The database returns a dataset that is the union of the results of the original first query and the results of the injected second query.

In the above query the italicized code is an example for union type of injection. The original application query is intended to get the 'e_code', 'e_name', 'mail' from 'user' table ,but the attacker injected UNION query to get the phone numbers of the employee from 'emp' table. The example show here is simple union query but they should meet minimum criteria.

C. Illegal/Logically Incorrect Queries

Attackers use different queries to get information from about the type of database used by the web application and structure of the application, sometimes schemas. In the first phase attacker gather information about the backend of the application by errors generated i.e., syntactic error reports. In the second phase attacker write logical queries by using holes in the applications that were discovered in phase-I.

Even though attacker uses Tautological class of queries but these types of attacks differ from the original Tautological attacks. Approach to find holes in the web application is different. In Fig.2 the input value must be a integer value, the attacker uses Logically Incorrect Method to exploit the application error information by providing a String values to the input.

Then the application returns an error message stating that input must be a integer value along with the database name, table name, and schema information.

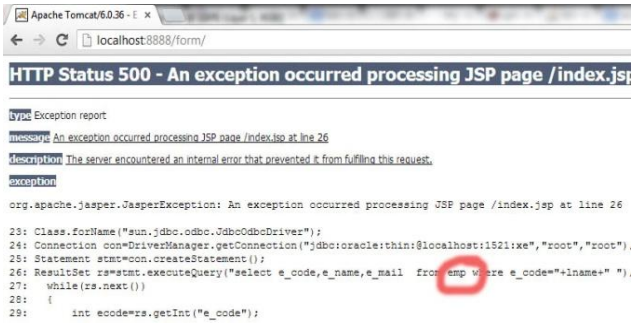

Fig. 4. Example for logical incorrect attack (Exploited Page)

. Using the error information the attacker can use another type of SQLIAs to exploit the web application.

D. Stored Procedure Attacks

Stored Procedures are used to run dynamic SQL queries. These attacks attempt to execute stored procedures. The attacker initially determines the database type and then uses that knowledge to determine what stored procedures might exist. Stored procedures can be susceptible to privilege escalation, buffer overflows, and even provide access to the operating system.

E. Piggy-Backed Queries

In this attack, an attacker tries to inject additional queries into the original query. As a result, the database receives multiple SQL queries. The first is the intended query which is executed as normal; the subsequent ones are the injected queries, which are executed in addition to the first. This type of attack can be extremely harmful. If successful, attackers can insert any type of SQL command. Vulnerability to this type of attack often allows multiple statements to be contained in a single string.
Example: If the attacker inputs ***"'; drop table emp - -"*** into the pass field, the application generates the query:

SELECT x, y, z FROM emp WHERE e_code='255''; drop table emp--
'

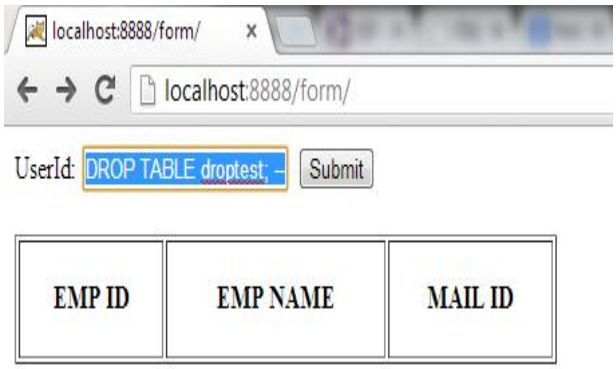

Fig. 5. Example for Piggy-backed queries

F.Inference

Inference is considered to be a advanced SQL injection attack. These types of attack create queries that because an application or database to behave differently based on the result of the query. These attacks allow an attacker to extract data from the database and detect vulnerable parameter. There are to well-known attack techniques based on inference: blind-injection and timing attacks.

a. Blind-injection

An attacker performs queries that have a Boolean result. If the answer is true then the application behaves correctly and if the answer is false then it cause an error. So attacker can get the indirect response from database.

Scenario: Employee checks his/her count of leaves by entering his/her id.

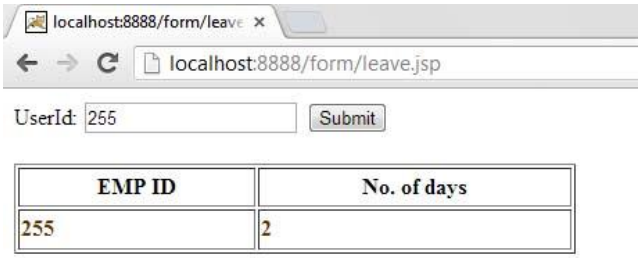

Fig. 6. Exploiting a page using inference attacks

Even though generic error pages were defined, the attacker exploits application blindly.

SELECT COUNT FROM leave WHERE emp_id='255' and SUBSTRING (SYSTEM_USER,1,1)='a' (False)

SELECT COUNT FROM leave WHERE emp_id='255' and SUBSTRING (SYSTEM_USER,1,1)='b' (False)

SELECT COUNT FROM leave WHERE emp_id='255' and SUBSTRING (SYSTEM_USER,1,1)='v' (True)

After 22nd attempt the attacker gets response from the application, by that he can know that employee name starts with 'v'. Similarly he can find other information by querying the application blindly.

b. Timing attacks

In this attack attacker observe the database delays in the database response and gather the information. To perform the timing attack attacker writes the query in the form of an if-then statement and then uses the WAITFOR keyword in one of the branch, which causes the database to delay its response by specified time.

IV. REVIEW OF RECENT TRENDS IN DETECTION AND PREVENTIVE TECHNIQUES

In this section, let us see the on-going and past few years research work to analyse, detect and prevent SQLIAs.

A. "The Design of SQL Injection Analysis System based on Honeynet" (Zelong Yin, Zhen Niu and Feifan Tong)-(2013)

Zelong Yin, Zhen Niu and Feifan Tong designed an SQL injection attack analysis system by merging Honeynet technology with SQL Injection principle [3]. Honeynet is a technology for data control and data capture mechanism includes one or more honey pots.

Initially system attracts attacks to perform SQL injection with Honeynet, then the system filter and analyses attacks to judge whether it is a SQL injection or not.

B. "Preventing SQL Injection with Input Rectification" (Tiffany Bao, Steve Matsumoto, JD Nir)-(2013)

Tiffany Bao, Steve Matsumoto, JD Nir developed an Intrusion Detection System that will rectify the input [4]. They have clearly mentioned difference between rectification and sanitization in the proposed system. It has three stages,

- Training
- Detecting
- Rectifying.

Automatic Input Rectification (Long et al.) is the method used for rectifying input [5]. The developed system is very efficient in rectifying the anonymous inputs at considerable rate.

C. "Web Anomaly Misuse Intrusion Detection Framework for SQL Injection Detection" (Shaimaa Ezzat Salama, Mohamed I. Marie, Laila M. El-Fangary & Yehia K. Helmy)-(2012)

Shaimaa Ezzat Salama, Mohamed I. Marie, Laila M. El-Fangary & Yehia K. Helmy proposed an Intrusion Detection system to detect SQL injections using Misuse and Anomaly detection techniques [6]. The log queries are converted into xml format and then association rules are applied to retrieve relation between queries and the table schema. These rules represent the normal behavior .Any query that deviate these rules, considered as attack. The system takes certain period to mature the association rules.

D. "Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities "(Shar, L. K. and Tan, H. B. K.) -(2012)

Shar, L. K. and Tan, H. B. K, proposed an technique to mine both sanitized code patterns and validation code patterns to find vulnerable inputs [7]. From that information of existing web site or web application, vulnerability prediction models are trained rigorously to improve efficiency.

E. "Random4: An application Specific Randomized Encryption Algorithm to prevent SQL injection" (S.Avireddy at el.)- (2012)

S.Avireddy at el, proposed a secured approach using randomized encryption algorithm [8]. Each character in input values is substituted by one of four values stored in the lookup table. The main intension behind assigning one of four random values is to decrease the probability of decrypting those values by hackers. This proposed approach can be a better alternative for simple hash based approach.

F. "A novel method for SQL injection attack detection based on removing SQL query attribute values "(Inyong Lee, Soonki Jeong , Sangsoo Yeo, Jongsub Moon)-(2011)

Inyong Lee, Soonki Jeong, Sangsoo Yeo, Jongsub Moon proposed a simple and effective method to detect SQL injection attacks [9]. This detection method uses combined static and dynamic analysis method along with SQL query parameter removal algorithm. The parameters are separated from the query and a generalized algorithm based on static and dynamic analysis is used to detect whether the parameters are genuine or infected. Because of its simple nature, the proposed algorithm can be implemented both with web applications and any application connected to databases.

G. "An Authentication Scheme using Hybrid Encryption" (Indrani Balasundaram, E.Ramaraj)-(2011)

Indrani Balasundram and E.Ramaraj proposed an authentication scheme in which they propose an algorithm which uses both Advance Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA) to prevent SQL injection attacks. In this method a unique secret key is fixed or assigned for every client or user [10]. On the server side server uses private key and public key combination for RSA encryption. In this method, two level of encryption is applied on login query:

To encrypt user name and password symmetric key encryption is used with the help of user's secret key.

Asymmetric encryption mechanism used for encrypting the user values. The proposed method needs 961.88ms for encryption or decryption and this can be negligible. It is Very difficult to maintain every user secret key at server side and client side. There is no security mechanism at registration phase.

H. "Effective SQL Injection Attack Reconstruction Using Network Recording"(Allen Pomeroy and QingTan)-(2011)

Allen Pomeroy and Qing Tan has suggested a technique for finding vulnerabilities in Web Application such as SQL injection attack by network recording [11]. In this approach network forensic techniques and tools are used to analyze the network packets containing get and post requests of a web application. This approach uses network based Intrusion Detection System (IDS) to trigger network recording of suspected application attacks.

Some disadvantages also exist with this approach:

- Difficult to record high volume traffic.
- Packet fragmentation attack could bypass this approach.

I. "Dynamic Candidate Evaluations Approach to prevent SQL injection"(P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan) -(2010)

Prithvi Bisht and his team members propose a tool called Candidate evaluation for Discovering Intent Dynamically (CANDID) [12]. This method record the programmer-intended SQL query structure on any input(candidate inputs) from the legitimate user and compare this with the query structure generated with the attackers input. Some disadvantages also exist with this approach are:

• Developer learning is required.

• It is not possible to make a complete set of legitimate inputs for a large web application.

J. "Obfuscation-based Analysis of SQL Injection Attacks"(Raju Halder and Agostino Cortesi)(2010)

In this method an obfuscation/de-obfuscation based technique is proposed to detect SQL Injection Attacks (SQLIA) in a SQL query before sending it to database [13]. This technique has three phases:

- Static phase: In the static phase, the SQL Queries in the web application code are replaced by queries in obfuscated form.
- Dynamic Phase: In this phase user inputs are merged with the obfuscated query at run-time. After merging, dynamic verifier checks the obfuscated query at atomic formula level to detect the SQL injection attack.
- If no SQL injection found during the verification phase reconstruction of the original query from the obfuscated query is carried out before submitting it to the database.

K. "SQL injection Detection via Automatic Test Case Generation of Programs" (Michelle Ruse, TanmoySarkar, Samik Basu)-(2010).

This approach uses automatic test case generation to detect SQL Injection Vulnerabilities [14]. The main idea behind this framework is based on creating a specific model that deals with SQL queries automatically. It also captures the dependencies between various components of the query. The used CREST(Automatic Test Generation Tool for C) test-generator and identify the conditions in which the queries are vulnerable. Based on the results, the methodology is shown to be able to specifically identify the causal set and obtain 85% and 69% reduction respectively while experimenting on few sample examples.

L. "Combinatorial Method for Preventing SQL Injection Attacks" (R. Ezumalai, G. Aghila)-(2009)

This approach uses both static and dynamic approach to detect SQL injection. It is a signature based SQL injection detection technique [15]. In this approach they generate hotspots for SQL queries in web application code and divide these hotspots into tokens and send it for validation where it uses Hirschberg's algorithm, which is a divide and conquer version of the Needleman-Wunsch algorithm, used to detect SQL injection attacks. Since, it is defined at the application level, requires no change in the runtime system, and imposes a low execution overhead.

M. "An Approach for SQL Injection Vulnerability Detection- AMNeSIA"(M. Junjin)-(2009)

Analysis and Monitoring for NEutralizing SQLInjection Attacks (AMNeSIA) is a fully automated technique for detecting and preventing SQL injection attacks [16].

It works in two phases.

- Static analysis: In this phase it analyze web application code and automatically generate the SQL query mode on the basis of possible legitimate queries.
- Runtime analysis: In this phase it scan all dynamically generated SQL queries and checks them to be with compliance to the statically generated models in the previous step. When this step detects that a query is not matched with the query model, it classifies the input as an SQL injection attack, logs the necessary information and throws an predefined exception that the application can then deal with suitably.

V. DEFENSIVE CODING TECHNIQUES TO PREVENT SQL INJECTION ATTACKS IN WEB APPLICATIONS

According to OWASP and SANS there are certain standard coding practices that can prevent SQLIAs with performance as a trade-off [17][18].

A. Validate Input or Data sanitization.

Input Validation in web applications is a basic technique to mitigate SQLIAs'. Best way to validate data is to use default deny, regular expression. The regular expression shown below would return only letters and numbers.

/[^0-9a-zA-z/

Similarly for checking email id which contains symbols like @ , _ , ,the regular expression should be

/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/

As far as possible use numbers, numbers and letters. If there is a need to include punctuation marks of any kind, convert them by HTML Encoding. So that " vinod "'" or > vinod ">" For instance if the user is submitting the E-mail address allow only @, -, . and _ in addition to numbers and letters to be used and only after they have been converted to their HTML substitutes.

Client side validation can be bypassed by the attackers, it is better to use sever side validation mechanism by writing filters. These filters protect the web servers from malicious code by analyzing input parameters.

B. Binding Dynamic SQL Query parameters

Use Prepared Statements instead of using Stored Procedures for dynamic SQL queries. Prepared Statements bind the user inputs and compare inputs as a whole with the database values[3]. This is one of the best practices to reduce SQLIAs' in web applications. Example for Prepared Statements:

```
Statement = "SELECT * FROM User WHERE userName= ? ";
PreparedStatement ps =
con.prepareStatement(selectStatement);
ps.setString(1, userId);
ResultSet rs = ps.executeQuery();
```

In the above statement, if the input is **' or '1'='1'** , this term can not affect the original SQL query ,because the Prepared Statement considers entire " **' or '1'='1'** " as a single word and compares that word with the usernames in the database. Prepared Statements with " ? " binding variables can prevent SQL Injection attacks .

However with the usage of Prepared Statements improperly can lead to SQLIAs'

```
PreparedStatement ps = con.prepareStatement("SELECT * FROM
user WHERE userId = '+UserName+'");
```

The above code has same impact as explained in Tautologies even though the query is Prepared Statement.

C. *Crafting Error reports*

This is the final practice during coding the web applications. During development the web applications the custom/generic error messages are need to be mentioned for different error reports generated by the application. So that attacker cannot know any information about the database.

D. *Limiting user privileges*

By limiting the privileges given to database accounts in an application will reduce the amount of damage incurred by SQL injection attacks. This includes removing admin privileges from running web server application accounts. Initially this may introduce some delay and increase workload into the deployment of applications, the added security of giving the application accounts only the required privileges will secure the application environment to be another deterrent for would be attackers.

VI. CONCLUSION

Though there are number of approaches for detecting SQLIAs' in web applications and preventive measures, still remains as a major issue because of poor developing strategies [2]. According to OWASP top 10 threats in 2013, injection attacks stands first. The survey of different attacks is summarized and different detective and preventive coding mechanisms are explained with examples to mitigate SQL Injection Attacks in Web Applications.

REFERENCES

1. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.
2. Justin Clarke, "SQL Injection Attacks " 2nd Edition,2012.
3. Zelong Yin, Zhen Niu and Feifan Tong (2013,March,13-15). The Design of SQL Injection Analysis System based on Honeynet .VOL-I. Available: http://www.iaeng.org/publication/IMECS2013/IMECS2013_pp403-406.pdf
4. Tiffany Bao, Steve Matsumoto, JD Nir (2013,March,31). Preventing SQL Injection with Input Rectification Available: http://18739c.ece.cmu.edu/bravo-2/wp-content/uploads/sites/9/2013/03/report.pdf
5. Fan Long Ganesh, Carbin, Rinard,"Automatic Input rectification",IEEE Conf. on Software Engineering,June-2012,pp.80-90.
6. Shaimaa Ezzat Salama, Mohamed I. Marie, Laila M. El-Fangary & Yehia K. Helmy ," Web Anomaly Misuse Intrusion Detection Framework for SQL Injection Detection ", IJACSA ,vol.3,2012,pp.123-129
7. Shar, L. K. and Tan, H. B. K .(2012). Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities. Available : http://dl.acm.org/citation.cfm?id=2337399
8. S.Avireddy, " Random4: An application Specific Randomized Encryption Algorithm to prevent SQL injection" IEEE conf. Trust, Security and Privacy,2012,June,pp.1327-1333.
9. Inyong Lee, Soonki Jeong, Sangsoo Yeo, Jongsub Moon ," A novel method for SQL injection attack detection based on removing SQL query attribute values" .ELSEVIER Trans. On Mathematical and Modelling, 2011,pp58-68.
10. Indrani Balasundaram, E.Ramaraj,"An Authentication Scheme for Preventing SQL injection Attack using Hybrid Encryption" (ISSN 1450-216,2011, Vol.53,pp.359-368.
11. Allen Pomeroy and QingTan ,"Effective SQL Injection Attack Reconstruction Using Network Recording" IEEE Conf. on Computer and Information Technology, Sept.2,2011,pp.552-556.
12. P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan ,"Dynamic Candidate Evaluations Approach to prevent SQL injection" ACM Trans. Inf. Syst. Secure., 13(2):1–39, 2010.
13. Raju Halder and Agostino Cortesi," Obfuscation-based Analysis of SQL Injection Attacks",IEEE Conf. On Computers and Communication- Italy,June,2010,pp.931-938. Digital Object Identifier : 10.1109/ISCC.2010.5546750
14. Michelle Ruse, TanmoySarkar, Samik Basu," SQL injection Detection via Automatic Test Case Generation of Programs", IEEE conf. on Application and the Internet,July,2010,pp.31-37.
15. Ezumalai, G. Aghila, "Combinatorial Method for Preventing SQL Injection Attacks", IEEE Conf. on Advance Computing, March 2009.
16. M. Junjin," An Approach for SQL Injection Vulnerability Detection-AMNeSIA",IEEE Conf. on Information technology,April,2009,pp.1411-1414.
17. https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet,Dec-2012
18. http://www.sans.org/top25-software-errors/