Journal of Engineering and Applied Sciences 14 (Special Issue 5): 9185-9192, 2019
ISSN: 1816-949X

Detection and Prevention SQL Injection using MCA Technique

Bashar M. Nema and Hanan Abed AL Wally
Department of Computer Science, College of Science, Mustansiriyah University, Baghdad, Iraq
bashar_sh77@uomustansiriyah.edu.iq

Abstract: Web Application (WA) is one of the common prevalent platforms for the delivery of information and services via. internet now a days. Therefor, the importance of web application has increased. At the same time, security risks have also increased. Depend on the OWASP's reports (Open Web Application Security Project) there are ten risks have been listed as the top ten security risks for any web application in the world. SQL injection is one of topmost security risks in OWASP which is considered in this research. So, securing and maintaining the web application against SQL injection is very hard task and can be classified as a challenge. In this study, a proposed system works on detecting and preventing SQL injection via. using a method named Multi-Connect Architecture (MCA) has been developed. The number of queries used for training about 252 queries while the number of queries used for testing about 1806 queries. The result of the proposed system are 99.88 and 100% for classification rate and detection rate, respectively.

Key words: SQL injection, tautology, union, piggyback, illegal, stored procedure, alternate encoding, inference, detection, prevention, MCA

INTRODUCTION

Today, the internet has become a common information infrastructure. However, WA's are becoming an important platforms for the delivery of information and services via. internet now a days (Rahul *et al*., 2012). WA's obtain an active role in different aspect such as: organizations, wide financial transactions, commercial and other internet-depend services (Prabakar *et al*., 2013). Mostly saved sensitive information that make them a good goal for attackers (Halfond *et al*., 2011). If a hacker can get the confidential information by sending harmful code, produces downtime and harmful for the WA and its database (db) (Kindy and Pathan, 2012). Therefore, security has become one of the major challenges in the latest years. However, it is significance to establish WA's protection from the aimed SQLI (Ali *et al*., 2015). SQL injection is famous common and harmful for WA's. Indeed, SQLI classified as one of the top 10 security risks for any web application in the OWASP (Athanasopoulos *et al*., 2010). This study focuses on the detecting and preventing of SQL injection attacks in real environment using MCA technique. SQL injection, related works, proposed system design and the results will be discussed in the following studies.

SQL-Injection (SQLI): SQL injection is a code injection attack that is used to hack web application and its database. Typically, it is occurs when the input provided by user is not correctly validated and is included in a direct manner as part of the query. via. advantaging from those vulnerabilities, the attacker (malicious user) can submit and run arbitrary SQL commands in a direct manner on the database. As a result, the malicious user can get confidential data from the database, alter the data and run administrative processes (Aich, 2009; Pinzon *et al*., 2013).

Tautology attack: Tautology is one of the most common kinds of SQLI attacks that are used for bypassing authentication process and extracting of data. Basically, it works through injecting malicious code into the conditional statement this is why they typically evaluate to true. To do that, the attacker takes advantage of any injectable field used in a query's "WHERE clause" and transform to tautology.

Illegal/logical incorrect queries: This attack is considered as kinds of collecting information attacks, since, the attacker collect significant information concerning the kind and structure of a web application's database. The weakness that is leveraged by this type is that the mistake messages which are retrieved by WA's are typically extensively descriptive. Originally, these error messages have the aim to aid programmers in

Corresponding Author: Bashar M. Nema, Department of Computer Science, College of Science, Mustansiriyah University, Baghdad, Iraq

debugging their applications and additionally aidatt ackers in gaining information about the database.

Piggy-backed query: It is one of the most harmful attacks that are used for data extraction, executing remote commands and performing denial of service. Attackers do this by adding more queries to original one. Therefore, the database gets multiple queries. The first is legal query whereas the subsequent ones are illegal queries as illustrated in the form (Singh *et al*., 2015). Legal query+';'+insert (update, select, delete, drop)+<rest of the inserted query>.

Union queries: Union query attacks are used for tricking the application to the retrieval of data from a table that differs from the table which has been intended by the developer. The attackers do this by the injection of a query of the form: union select<rest of injected query>. Due to the fact that the attacker fully controls the injected query they are capable of using this query for returning data from an identified table (Tajpour *et al*., 2010).

Inference: In inference attack, the attacker alters the behavior of a web application which is done by two techniques: blind injection and timing attacks. Blind injection is a common kind of inference attacks, due to the fact that the attacker simply injects a query and monitors the behavior of web application: in the case where the application keeps functioning in a normal manner this will refer to the fact that the injected query resulted to true in the opposite case, the attacker will see a generic page which could be different from the regular one (Gadgil, 2013). Timing attack is identical to blind injection, since, it also lets the attacker collect information from a database. In addition to using SQL query commands such as "WAITFOR" to cause delay in the execution of database queries (Kumar and Chatterjee, 2014).

Stored procedure: It is a part of database which the programmer can set as an additional level of abstraction layer on. One of the most common misconceptions is that the use of the stored procedures in web application can avoid SQLI attacks. Due to the fact that the stored procedures can be written by the programmer this portion is as vulnerable as normal queries (Jane and Chaudhari, 2012).

Alternate encoding: This kind of attack is used combined with other attacks. In other words, alternate encodings give no distinct way for attacking the database of an application they are merely an enabling approach allowing the attackers evading the methods of detection and prevention. To do that attackers have used alternative approaches to encode their attack strings such as hexadecimal, ASCII and unicode encoding. This is why attackers are rather successful to use alternative encodings for concealing their attack strings (Manoj *et al*., 2014).

Literature review: Many contributions have been achieved to solve the problem of SQL injection some of them are present below.

Rawat and Shrivastav (2012), proposed a method to detect SQL Injection (SQLI) using Support Vector Machine (SVM). It consists of two phases: training and testing. In training phase, each query in the dataset is divided into tokens and vector of strings which is created for training the SVM. In the testing phase, SVM classifier was used to classify the input query into normal or SQLI. The method has been tested off-line with (800) queries and the detection rate has reached 96.47% (Rawat and Shrivastav, 2012).

Matsuda (2013), suggested a method called Soft Confidence-Weighted (SCW) to detect SQLI. It consists of two phases: learning and testing. In the learning phase, two distinct ways are used to construct feature space and learning SCW: in the first way, feature space construct by giving importance to the alphabets and the numbers in the input query and the second way, gives importance to the symbols in the input query. In the testing phase, SCW was used to classify the input into normal or SQLI. The method has been tested with 500 queries that have been classified as 387 SQLI and 113 normal queries. The detection rate of SCW with the first method was 75% whereas with the second method it has reached 84% (Matsuda, 2013).

Shafie (2013), presented a system to detect and prevent SQLI using Anatempura tool. The system was composed of three checking components: input checker, output checker and database observer. In the first step of the input checker, information has been submitted it is sent to the server and will be reformatted in Anatempura format which is the IP address, the submitted data and the submission time. Anatempura tool analyzes the data entry against existing attacks. Error messages are sent to the user and analyzed by Anatempura tool which is used from the output checker. The message containing details about the database structure is considered as unsafe message and needs to be blocked. The database observer is used to monitor the outcome of each database transaction to check whether it is a safe or unsafe transaction. The system has been tested with no false negative and low rate of false positives (Shafie, 2013).

Dharam and Shiva (2014), suggested a system to detect and prevent tautology attack. It consists of three phases: Critical Variables (CVs) identification, Critical Paths (CPS) identification and runtime monitoring. In the first phase all CVs that are in the source code of the application are found by scanning the software repository. In the second, phase all the legitimate performance paths of the application which is called as CPs are identified by using basis path (bp) and data flow (df) testing methods. In the third phase (runtime monitor phase) is developed using AspectJ which is provided a compiler of a special kind known as the AspectJ Compiler (AJC), used to detect application's abnormal behavior, notifies the admin about it and stops the execution of the application as a preventive measure. The system has been tested with 50 queries categorized as 15 tautology attacks and 35 normal queries without the generation of any false positives and false negatives (Dharam and Shiva, 2014).

Joshi and Geetha (2014), proposed a system to detect SQLI using Naïve Bayes classifier. It consists of three phases: preprocessing, training and testing. In the preprocessing phase, each query in the dataset is divided into tokens, vector of features is created using blank separation method and used for training. In the testing phase, Naïve Bayes classifier is used to classify the queries into normal queries and SQLI. The system has been tested with 158 queries divided into 101 as normal queries and 57 as SQLI (comments, union and tautology) and the detection rate was 93.3% (Joshi and Geetha, 2014).

Verma and Kaur (2015), introduced a system to prevent SQLI using hybrid approach that consists of two phases: positive tainting and negative tainting. In the positive tainting, input query is checked at the compile time to detect SQLI (i.e., tautology, piggyback and union) on the base of syntax. If any SQLI is found in the input query, the system blocks the query at the compile time. The second phase (negative tainting) is divided into two stages: training and detection. In training stage, all the possible malicious queries (SQLI) are divided into tokens that are converted into an integer number and stored in the database as a primary list. In the second stage (detection), the input query is checked at the run time after being converted into an integer number. The integer number is saved in the database as a secondary list to be compared with each value of the primary list. The matching numbers mean SQLI detection, or else, the query is normal. The system has been tested with 170 queries and a prevention rate that reached 88.28% (Verma and Kaur, 2015).

Ladole and Phalke (2016), proposed a system to detect SQLI using SVM classifier. It is also capable of classifying users into normal users or attackers based on the query that they have submitted. The system consists of two phases: training and testing. In the training phase, Weka library has been utilized for SVM classification of features and for feature selection, Fisher score has been utilized. In the testing phase, SVM was used to classify the input queries into normal queries or SQLI. It has been tested with 1600 queries and the detection rate has reached 94.11% (Ladole and Phalke, 2016).

Raj and Sherly (2018), suggested a model to prevent SQLI utilizing a technique known as Direct Reverse Resemblance (DRR). The model consists of two phases: Routine Phase (RP) and Methodical Phase (MP). The first phase is implemented by comparing a user entered data with the data in the table and if the comparison is successful then the data will pass to the second phase. In the second phase, the data are reversed and divided into two parts by inserting special characters between them, the same procedure is applied on the data of the table and is divided into two parts for comparison. The successful matching means that the data is normal, otherwise, SQLI is detected. The system has been tested with 2285 queries and a prevention rate of 94.77% (Raj and Sherly, 2018).

Proposed system design: In this research, a proposed system has been developed for detecting and preventing SQL injection as illustrate in Fig. 1. Also, a web application called Student Registration System (SRS) has been developed to explain the implementation of the proposed system.

Preprocessing phase: The input query may contain undesirable words or mix of upper and lower case letters. Therefore, the preprocessing is essential phase in the proposed system because it makes this input query appropriate to the features extraction process. The preprocessing consists of three stages: tokenization, Lower Case (LC) conversion and Stop Words (SWs) removal as illustrate in Fig. 2. In the first stage, the input query has been divided into various tokens depend on special characters called delimiters. The delimiters in the proposed tokenizer include the punctuation, whitespace, operators. The string between two single quotes (‘ ’), between (/* */) and after double dash (--) considered as single token. In the second stage, the tokens (words) resulted from the previous stage may contain a mix of both uppercase and lowercase letters. These words affect negatively on feature extraction process, subsequently will effect on the possibility of detection SQLI. To solve this problem all words are converted into lowercase letters. In the third stage, stop words (such as table name, field name and field's value) are removed because these words did not have any valuable information useful in the detection process.

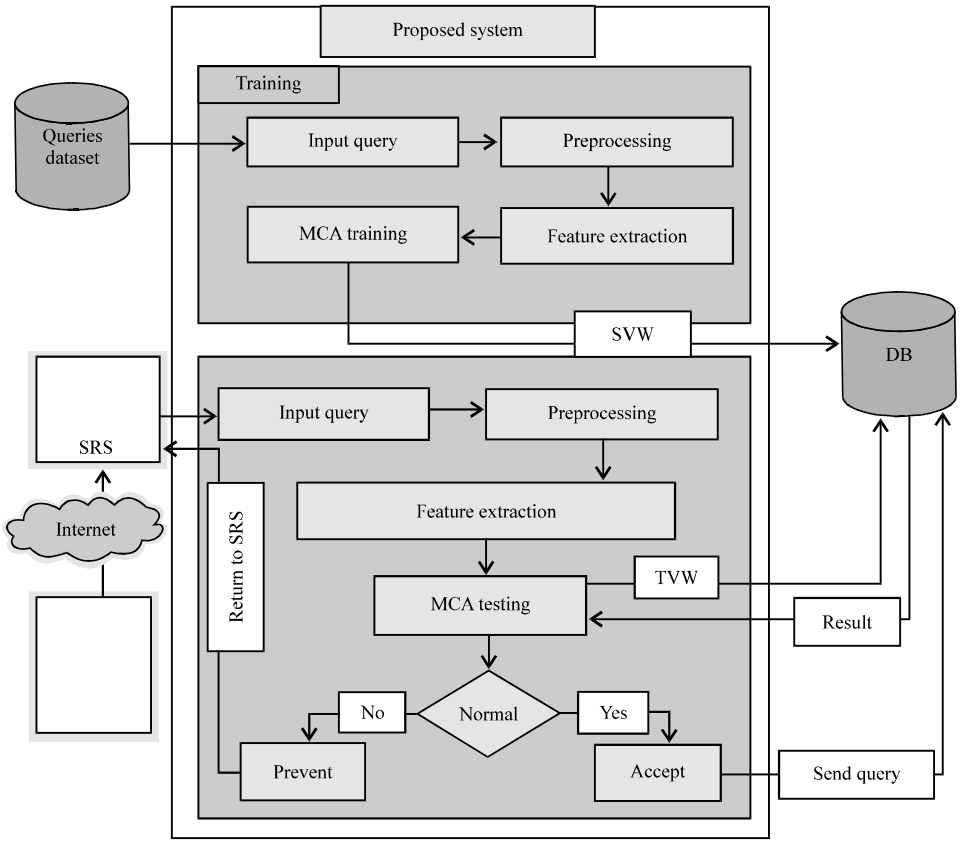

Fig. 1: The proposed system

Table 1: List of features

Bag of words										
F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	
sysobjects	db_name	xtype	char	and	shutdown	drop	delete	insert	update	
F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	
1	union	1	number	waitfor	convert	order	1	exec	set	
F21	F22	F23	F24	F25	F26	F27	F28	F39	F30	
@@servername	@@version	)	(	>	<	cast	1	<=	>=	
F31	F32	F33	F34	F35	F36	F37	F38	F39	F40	
like	1	table	+	-	*	/	1	1	ascii	
F41	F42	F43	F44	F45	F46	F47	F48	F49	F50	
substring	by	top	all	values	into	delay	;	int	<>	

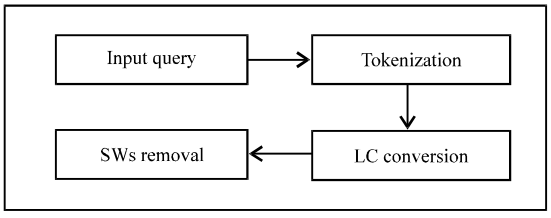

Fig. 2: Preprocessing of query

Feature extraction phase: In this phase, bag of words was used as a method of features extraction. Generally, bag of word's features are collected from multiple sources to build a special list called features list. In proposed system, the special list consists of 50 features as shown in Table 1. Where (F1, F2, …, F50) are the features collected from previous studies. In bag of words, each query represented by what so-called feature vector is an one-dimension list of fifty features. The features in

$$w_0=\begin{bmatrix}0&1&1\\1&0&1\\1&1&0\end{bmatrix},\quad w_1=\begin{bmatrix}0&-1&-1\\-1&0&1\\-1&1&0\end{bmatrix},\quad w_2=\begin{bmatrix}0&-1&1\\-1&0&-1\\1&-1&0\end{bmatrix},\quad w_3=\begin{bmatrix}0&1&-1\\1&0&-1\\-1&-1&0\end{bmatrix}$$

Fig. 3: MCA associative memory weight matrices

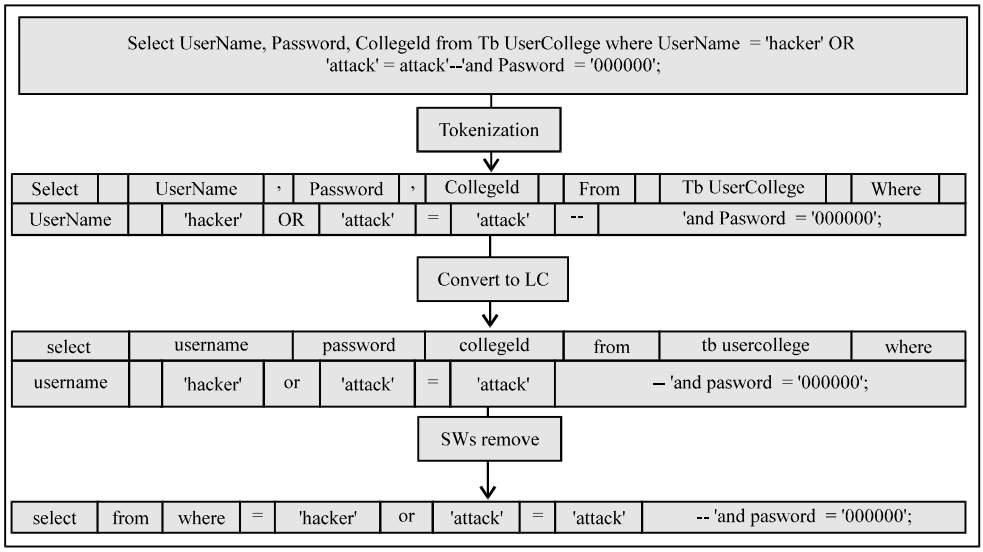

Fig. 4: Example of preprocessing phase

previous table will represent in binary form: 1 means exist feature and 0 means not exist feature.

Classification phase: In this phase, MCA technique has been used. MCA is a single layer neural network uses auto-association tasks and working with to phases which are: training and testing as explain below (Kareem, 2012): training phase, this phase is conducted using a dataset consists of 252 query representing kinds of SQLI in addition to normal queries. The following steps illustrate the process of MCA training:

Step 1: Initializing the four weights matrices (i.e. w_0, w_1, w_2, w_3) as illustrated in Fig. 3 (Kareem, 2012). Those weights are fixed and require no computations to be computed during this stage.

Step 2: Each query in dataset after converting to the feature vector will be divided into n vectors v with length 3, repeat steps 2.1 and 2.2.

Step 2.1: Compute Stored Vector Weight (SVW) for each vector. SVW is a set of weights indexes which compute using Eq. 1(Kareem, 2012):

$$SVW = f(Dcode(v))\begin{cases}0 \text{ or } 7\ 0 & \{\text{means } w_0\\ 1 \text{ or } 6\ 1 & \{\text{means } w_1\\ 2 \text{ or } 5\ 2 & \{\text{means } w_2\\ 3 \text{ or } 4\ 3 & \{\text{means } w_3\end{cases} \quad (1)$$

Step 2.2: SVW will be saved in the specific table called lookup.

In testing phase: Login page of SRS is used as example to explain the implementation of proposed system. For example when a user enters the following data in username field such as: username = 1' and password = convert (int, @@Version)_, SRS will generate the following query: Select UserName, Passwored, CollegeId from TbUserCollege where UserName = 'hacker' OR 'attack' = 'attack' --- 'and Password = '000000'; after that the proposed system (using MCA) will intercept the generated query for checking, if it is contains SQL injection or not as illustrated in the following steps:

Step 1: The generated query will pass to the preprocessing phase as illustrate in Fig. 4.

Table 2: Bag of words

Bag of words										
F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	
0	0	0	0	0	0	0	0	0	0	
F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	
1	0	1	0	0	0	0	1	0	0	
F21	F22	F23	F24	F25	F26	F27	F28	F39	F30	
0	0	0	0	0	0	0	1	0	0	
F31	F32	F33	F34	F35	F36	F37	F38	F39	F40	
0	1	0	0	0	0	0	1	1	0	
F41	F42	F43	F44	F45	F46	F47	F48	F49	F50	
0	0	0	0	0	0	0	0	0	0	

Fig. 5: Login page with propsed system

Step 2: After preprocessing, the query will convert to the feature vector using bag of words as shown in Table 2. Then, the feature vector are passed to the MCA for testing as illustrate in the following steps:

Step 3: Initialize the 4 weights matrices as illustrated in Fig. 3.

Step 4: Initialize the energy function matrix as illustrated in Eq. 2 (Kareem, 2012). Eenergy function is used to correlate the unknown query (i.e., normal or SQLI) with queries that have stored data (which is the SVW) in the lookup table throughout the training stage:

$$e = \begin{bmatrix} -3 & 1 & 1 & 1 \\ 1 & -3 & 1 & 1 \\ 1 & 1 & -3 & 1 \\ 1 & 1 & 1 & -3 \end{bmatrix} \quad (2)$$

Step 3: The feature vector regarded as testing query are divided into n vectors (v) of the length 3.

000	000	000	010	100	001
000	000	000	100	010	000
011	000	000	000	000	

Step 5: For each vector compute Test Vector Weights (TVW) by using Eq. 3 (Kareem, 2012):

$$TVW = f(Dcode(v)) \begin{cases} 0 \text{ or } 7 \ 0 & \{\text{means } w_0 \\ 1 \text{ or } 6 \ 1 & \{\text{means } w_1 \\ 2 \text{ or } 5 \ 2 & \{\text{means } w_2 \\ 3 \text{ or } 4 \ 3 & \{\text{means } w_3 \end{cases} \quad (3)$$

TVW(000) = 0, TVW(000) = 0,TVW(000) = 0, TVW(010) = 2, TVW(100) = 3, TVW(001) = 1, TVW (000) = 0, TVW(000) = 0, TVW(000) = 0, TVW(000) = 0, TVW(010) = 2, TVW(000) = 0, TVW(011) = 3, TVW(000) = 0, TVW(000) = 0, TVW(000) = 0, TVW(000) = 0.

Step 6: Sum up the energy function for all of the n vectors in the generated query each with its corresponding vector in the stored query by using Eq. 4 (Kareem, 2012). The minimum energy function represents the class of generated query (Table 3).

$$ep = \sum_{i=1}^{n} e[SVW_i, TVW_i] \quad (4)$$

After calculating the energy function, it is shown that the generated query is one of the type of SQL injection as illustrate in the following Fig. 5. Therefore, the proposed

Table 3: Classification rate of MCA

Name of classes	No. of queries (training)	No. of queries (testing)	Classification rate (%)
Normal	26	127	100
Tautology	114	980	99.89
Illegal	13	67	100
Inference: blind	9	61	100
Inference: timing	8	67	98.50
Piggyback	38	359	100
Stored procedure	8	66	100
Union	29	26	100
Alternate encoding	7	53	100
All classes			99.88

system (using MCA) will prevent the query from accessing and executing on the database by closing the conecction with it and return the user to the login page.

RESULTS AND DISCUSSION

In the proposed system, the dataset consists of 252 queries has been used for training the MCA technique while in testing phase about 1806 queries have been used to test the proposed system manually. The result of MCA is shown in Table 3. In the experiment of proposed system, MCA classified all input queries (about 1806 queries) correctly except two cases which are: one query in the "tautology attack" (SQLI) classified as "stored procedure" (SQLI) and one query in "inference: timing attack" (SQLI) classified as "inference: blind injection" (SQLI). These queries were incorrectly classified and did not effect on the detection and prevention processes because it is classified as SQLI regardless on its type. As a result, MCA detect and prevent all kinds of SQLI from accessing and executing on database. Therefore, the classification rate of MCA is 99.88% for all normal and SQL injection while the detection rate of MCA is 100% for SQL injection.

CONCLUSION

In this study, a proposed system works on detecting and preventing SQL injection has been developed. Based on the result of the proposed system, the conclusion is as follows: first, in query tokenization the selected set of delimiters has impoved dividing query in order to enhance feature extraction. Second, removing stop words have reduced the dimention of the feature space and led to minimizing the complexity. Third, the MCA technique needed few number of queries in the training phase. Finally, by using MCA in detecting and preventing SQL injection gave better performane.

ACKNOWLEDGEMENT

The researchers expresse their acknowledgement and apprication for Computer Sceince Department, College of Science, Mustansiriyah University for their support and encouragement in accomplish this research work , seeking more effective study in the future.

REFERENCES

Aich, D., 2009. Secure query processing by blocking SQL injection. MSc Thesis, National Institute of Technology Rourkela, Rourkela, India.

Ali, N.S., A.S. Shibghatullah and M.H. Al Attar, 2015. Review of the defensive approaches for structured query language injection attacks and their countermeasures. J. Theoret. Applied Inform. Technol., 2076: 258-269.

Athanasopoulos, E., A. Krithinakis and E.P. Markatos, 2010. An architecture for enforcing javascript randomization in web2.0 applications. Proceedings of the International Conference on Information Security Vol. 6531, October 25-28, 2010, Springer, Berlin, Germany, pp: 203-209.

Dharam, R. and S.G. Shiva, 2014. Testing for tautology based SQL injection attack using runtime monitors. Intl. J. Eng. Technol., 6: 1-4.

Gadgil, S., 2013. SQL injection prevention in banking. Intl. J. Comput. Sci. Inf. Technol., 4: 345-349.

Halfond, W.G.J., S.R. Choudhary and A. Orso, 2011. Improving penetration testing through static and dynamic analysis. Software Test. Verif. Reliabil., 21: 195-214.

Jane, P.Y. and M.S. Chaudhari, 2012. SQLIA: Detection and prevention techniques: A survey. IOSR. J. Comput. Eng., 2: 56-60.

Joshi, A. and V. Geetha, 2014. SQL injection detection using machine learning. Proceedings of the 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), July 10-11, 2014, IEEE, Kanyakumari, India, ISBN:978-1-4799-4191-9, pp: 1111-1115.

Kareem, E.I.A., 2012. A real time visual monitoring module for traffic conditions based on a modified auto-associative memory. Ph.D Thesis, University Sains Malaysia, Penang, Malaysia.

Kindy, D.A. and A.S.K. Pathan, 2012. A detailed survey on various aspects of SQL injection in web applications: Vulnerabilities, innovative attacks and remedies. Cryptography Secur., 1: 1-13.

Kumar, D.G. and M. Chatterjee, 2014. SQL injection prevention in banking. Intl. J. Comput. Network Inf. Secur., 1: 56-63.

Ladole, A. and M.D. Phalke, 2016. SQL injection attack and user behavior detection by using query tree fisher score and SVM classification. Intl. Res. J. Eng. Technol., 3: 1505-1509.

Manoj, R.J., A. Chandrasekhar and M.D.A. Praveena, 2014. An approach to detect and prevent tautology type SQL injection in web service based on XSchema validation. Intl. J. Eng. Comput. Sci., 3: 3695-3699.

Matsuda, T., 2013. On the property of the distribution of symbols in SQL injection attack. Intl. J. Intell. Comput. Res., 4: 376-381.

Pinzon, C.I., J.F. De Paz, A. Herrero, E. Corchado and J. Bajo *et al*., 2013. idMAS-SQL: Intrusion detection based on MAS to detect and block SQL injection through data mining. Inf. Sci., 231: 15-31.

Prabakar, M.A., M.K. Keyan and K. Marimuthu, 2013. An efficient technique for preventing SQL injection attack using pattern matching algorithm. Proceedings of the IEEE International Conference on Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN), June 13, 2013, Tirunelveli, India, pp: 503-506.

Rahul, S., J. Bhattacharyji and R. Soni, 2012. SQL injection attacks in database using web service: Detection and prevention. Asian J. Comput. Sci. Inform. Technol., 2-6: 162-165.

Raj, S.N. and E. Sherly, 2018. SQL injection attack prevention by direct reverse resemblance technique. Intl. J. Pure Appl. Math., 118: 599-614.

Rawat, R. and S.K. Shrivastav, 2012. SQL injection attack detection using SVM. Intl. J. Comput. Appl., 42: 1-4.

Shafie, E., 2013. Runtime detection and prevention for structure query language injection attacks. Ph.D Thesis, Gateway House, Leicester, UK.

Singh, P., K. Thevar, P. Shetty and B. Shaikh, 2015. Detection of SQL injection and XSS vulnerability in web application. Intl. J. Eng. Appl. Sci., 2: 16-21.

Tajpour, A., Z. JorJor and M. Shooshtari, 2010. Evaluation of SQL injection detection and prevention techniques. Proceeding of the 2nd International Conference Computational Intelligence, Communication Systems and Networks, July 28-30, 2010, Liverpool, pp: 216-221.

Verma, N. and A. Kaur, 2015. Prevention of SQL injection attack using hybrid approach. Intl. J. Adv. Res. Comput. Sci. Software Eng., 5: 1355-1359.