Cybersecurity Risks Detection and Prevention

Bashar M. Nema, Ph.D. (Asst. Prof.)[*] Hanan Abed AL Wally[*]

Abstract: Cybersecurity is a collection of techniques and processes that are designed for protecting computers, networks, database, and applications from attacks, unauthorized accessing, alteration, or destruction also can be a significant part in the development of information technology as well as Internet services. In Cybersecurity, there are several trends; the biggest one is Web application. Web application is one of the most prevalent platforms for the delivery of information and services via Internet nowadays. Therefore, the importance of web application has increased. At the same time, security risks have also increased. Based on the reports of OWASP (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 paper. So securing and maintaining the web application against SQL injection is very hard task and can be classified as a challenge. In this paper, a proposed system works on detecting and preventing SQL injection via using a method named Multi-Connect Architecture (MCA) is presented. The number of queries used in this paper is 130 query. The result of the proposed system is (95%).

Keywords: Cybersecurity, SQL Injection, Detection, Prevention, MCA.

[*] Mustansiriyah university college of science-Dept. of CS.

1. Introduction

Cyber security is a collection of techniques and processes that are modeled for protecting computers, networks, database, and applications from attacks, unauthorized access, change, or destruction [1]. It takes a significant part in the development of information technology as well as Internet services. In Cybersecurity, there are several trends; the biggest one is Web application. Web application is one of the most important platforms for the delivery of information and services via Internet nowadays [2].

Today, the internet and web application are considered as a fourth necessity for humans after air, water and food. The Web is utilized for communication, services and sharing the information on the internet through some applications. Most of online services are implemented in the form of web applications. Online banks, search engines, e-mail applications, social networks are just a few examples of those web application services. Therefore the relying on web application has increased. At the same time, the number of web application security risks has also increased. So securing and maintaining the web application against security risks is very hard and challenging task [3].

SQL Injection (SQLI) is one of the topmost security risks used by attackers for bypass authentication process, extracting data, manipulating or deleting the contents via inserting malicious code [4]. SQL Injection, theoretical background of proposed system and proposed system will be discussed in the following sections.

2. Literature Survey

In recent years many contributions have been achieved in the field of detection and prevention of SQL Injection; some of them are present below briefly:

Manpreet Kaur and Anand Kumar Mittal, (2016) [5]: proposed a system for the detection and prevention of SQLIA by using hybrid approach. The proposed system works in two phases: First phase involves the syntax evaluation of input query, the second phase divided into two parts: One part includes the training phase and the second part includes

SQLI detection. The system tests with (140) input queries. The result of the proposed system was 94.44%.

Yogesh Bansal and Jin H. Park, (2015) [6]: proposed protection system model to prevent SQLIA by using two-level hash approach. It is consisted of three stages, namely: registration, login and validation stages. The proposed system has been executed with HTML, PHP and MySQL, and cryptographic hash function SHA-512 has been used in the coding. The outcomes of the suggested system show the high degree of effectiveness for the prevention of SQLIA.

WITT YI WIN and HNIN HNIN HTUN, (2014) [7]: suggested a system to detect SQLIA by restricting the number of queries to be scanned during runtime. The proposed system has two modules; Input Checker and Query Checker. In the Input Checker the request checks and if the malicious input is found in the input query, it is declined and not send to the Query Checker. Only input query that contains proper input is forwarded to the next module. Query Checker detects the input query by comparing with the legitimate queries before forwarding to the server. The system is tested on four real world applications, namely: Portal, Event Manager, Employee Directory and Classifieds. The results of the proposed system show efficient in detect SQLIA.

Gaurav Shrivastava and Kshitij Pathak, (2013) [8]: proposed a system to prevent SQLIA by using double authentication process. The proposed system uses two databases: relational and hierarchical. As the first step, the input query is forwarded to both databases. After that, it is divided into various tokens by using tokenization process and compare the results. If the results are the same, there is no injection, otherwise it is present. The proposed system showed its ability to prevent a union and alias query.

V.Shanmughaneethi and S.Swamynathan, (2012) [9]: proposed a system to detect SQLI with two layers: syntactic verification and customize error generation. The First layer consists: query format engine, XML file generation, query parser, query structure analyzer, SQL keyword verifier and XML Schema. The second layer consists: SQL verb verifier, query processor, error logger and error customizer. Both layers are used to analyze user input and protect web applications from SQL injections. The proposed system showed its ability to detect a tautology, illegal/ logically incorrect queries and piggy pack.

3. SQL-Injection (SQLI)

It is a type of attack which exploits the fact that data supplied by users is directly inserted into an SQL query in a way which portion of the input from the user is considered as an SQL code **[10].** This type of attack is one of the most dangerous attacks in web application in these days; it is occur due to lack of input validation or encoding as part of a command or query **[11].** Injection attacks as SQL injection allow the attackers gain accesses to the database which result to discloser sensitive information and loss of confidentiality, unauthorized modifications, bypass the authentication process and producing useful information to further attacks [12]. Generally, SQL injection attacks may be categorized as follow:

a) Tautologies

Tautology is one of the kinds of SQLI attack that are used to insert malicious code in conditional query statement to assess true constantly. By using tautology attack, the attacker can bypass authentication process and gain unauthorized access to the database **[13].**

For Example: SELECT * FROM student WHERE username = 'aaa' AND password='222' **OR '1 '='1';**

In this query, **OR '1 '='1'** was inserted in the **where clause** statement and if the web application not validates the input correctly, then all the records from the student table will be returned to the attacker **[14].**

b) Illegal/Logical Incorrect Queries

One of the most important kinds of SQLI attack, used to collect significant information (i.e., kind and structure) about the back-end database. By using illegal queries, the attacker will be injected malicious code in query that causes a syntax, type conversion, or logical error into the database **[13].**

For Example: SELECT * FROM student WHERE username = 'aaa' AND password=convert (int, (SELECT top 1 name FROM sysobjects WHERE xtype='u'));

In this injected query, the attacker attempts to extract the first user table (xtype='u') from the database's metadata table and then convert to the integer (assume the application is using Microsoft SQL Server, for which the metadata table is called sysobjects). When performing this query, the database returned error message would be:

"Microsoft OLE DB Provider for SQL Server (0x80040E07) Error converting nvarchar value "Credit Cards" to a column of data type int". There are two valuable information pieces in this message which help an attacker: the first one is that the attacker is capable of seeing that the database is an SQL Server database. The second one, the error message points out the value of the string which resulted in the type conversion. In this case, this value is the name of the first user-defined table in the database: "Credit Cards" as well [14].

c) Piggy-Backed Query

This kind of SQLI can be done by adding more queries to the original query; as a result several SQL statements in one string will be fed into the database to be executed. Generally, the first query is legal and executes without any damage followed by illegal queries used for the purpose of fetching data or update database [13].

For Example: SELECT * FROM student WHERE username = 'aaa' AND password='222'; DROP TABLE student;

In the example above, the first query is legal and the second one (drop table student) is illegal. When executing the second query all the student useful information will be lost [14].

d) Union Queries

This kind of SQLI can be used to join malicious query to the safe query by using union operator. As a result, the attacker can bypass authentication process and extract data [13].

For Example: SELECT * FROM student WHERE username = 'a1a1' UNION SELECT addresses FROM admin WHERE id='3142'-- password='222';

In this example, assume that there is no username equal to 'a1a1'. The result of executing the first query is null and the second query is data of 'admin' table. As a result, the data of admin will be return to the attacker [14].

e) Blind Injection

In blind injection, the attacker can infer information from the behavior of application by asking the server true/ false questions. In case of inserted query assesses to true, the application keeps functioning normally. Otherwise, the behavior of application turns significantly different from the normally-functioning application **[13].**

For Example: SELECT * FROM student WHERE username = 'aaa' AND @@version like '%2012'-- password='222';

In this example, assume that there is username equal to 'aaa'. If the application would return error massage, the attacker would know the query assesses to false **[14].**

f) Timing attack

In timing attack, the attacker can infer information via the measurement of the increasing or decreasing in response time of the database **[13].**

For Example: SELECT * FROM student WHERE username = 'aaa' AND ASCII (SUBSTRING ((SELECT top 1 name FROM sysobjects), 1, 1)) >X wait for delay '00:00:01'-- password='222';

In example above, the attacker tries extracting the first character of the first table's name with the use of ASCII value. In the case where ASCII value of the character is greater-than the value of X then the attacker will observe 5 second delaying in the database response. In this case, the attacker may utilize a binary search by varying the value of X for identifying the value of the first character **[14].**

g) Stored procedure

Stored procedures are the codes that are already present in the database and they are vulnerable as program code. They return true for authorized users and false for unauthorized ones. If the attacker input ('; SHUTDOWN ;') for username then stored procedure creates following query **[13].**

For Example: SELECT * FROM student WHERE username = 'aaa'; SHUTDOWN; -- and password='222';

In the query above, the first query is executed normally, and then the second one. Malicious query is executed, and that yields a database shutdown [14].

h) Alternate Encoding

In alternate encoding, the attacker utilized alternate encoding approaches (such as the use of hexadecimal, ASCII, and Unicode character encoding) to evade detection and prevention methods [13].

For Example: SELECT * FROM student WHERE username='aaa'; exec (char (0x73687574646j776e))—and password='111'

In the query above, the first query is executed normally, and then the second one. Malicious query is executed, and that yields a database shutdown [14].

4. Theoretical Background of Proposed System

4.1 Parser

In general, the parser is combined with lexical analysis. The main task of lexical analysis is to read the sequence of characters as input and produce sequence of tokens as output. Then, the sequence of tokens is sent to the parser for syntax analysis and inform on errors. The second benefit to extract the general structure of sentence [15].

4.2 Hashing (SHA-256)

Hashing is the conversion of characters of a string into a usually shorter fixed-length key or value which denotes the actual string [16]. SHA-256 hash function is used to convert the input sentence with length between 0-2^{64} bits into a 256-bit sentence digest. The conversion operation can be split into 4 stages as depicted in figure (1) [17].

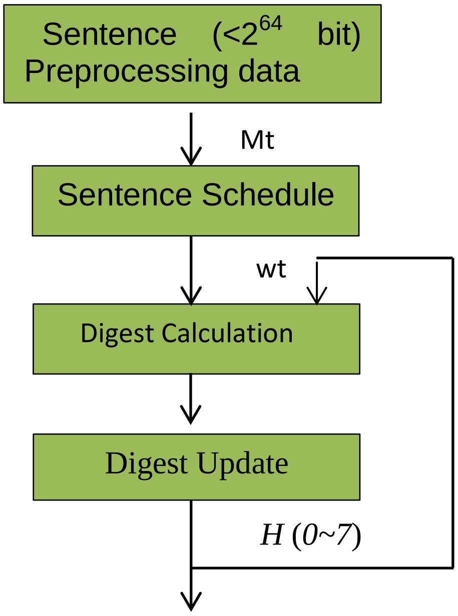


Figure (1): SHA-256 algorithm flow diagram

In the first stage (Pre-Processing), the sentence is segmented and padded into data blocks (Mt) of length 512 bits. These Mt are then sequentially fed into the 2nd stage, which are the sentence Schedule where they are expanded to 64 of 32-bit words (Wt) according to the generation equation:

$$W_t = M_t,\ 0 \le t \le 15$$

$$\sigma_1 (W_{t-2}) + W_{t-7} + \sigma_0 (W_{t-15}) + W_{t-16},\ 16 \le t \le 64 \qquad (1)$$

Then, every 32-bit word (Wt) is put into the 3rd stage known as the Digest Calculation being processed in an iterative loop. There are 8 of 32-bit working variables ($a \sim h$) with their initialization value H(0~7), and two 32-bit intermediate values (T1, T2) in every loop to be calculated and updated in this stage, according to a series of dedicated functions as follows:

$$\left.\begin{array}{c} T_1 = h + \sum_1 (e) + ch(e, f, g) + k_t + W_t \\ T2 = \sum_0 (a) + maj(a, b, c) \\ a = T_1 + T_2 \\ b = a \\ c = b \\ d = c \\ e = d + T_1 \\ f = e \\ g = f \\ h = g \end{array}\right\} \quad (2)$$

This stage is the most computationally complex of the hash process. After 64 iterations in this stage, values of the eight operating variables are sent to the last stage, which is the Digest Update. In this stage, the digest variables ((0~7)) are afterwards updated via adding them to the operating variables as follows:

$$H(0 \sim 7) = H(0 \sim 7) + (a \sim h) \quad (3)$$

4.3 Multi-Connect Architecture (Mca) Associative Memory:

The MCA is a single layered neural network (NN) which utilizes auto-association tasks and performs its work in 2 phases: training and testing. It is a modified Hopfield NN. It was developed depended on two principles. First, the pattern will be split into several vectors with length three; which means that the size of the network is set to three. Thus, the network deals with portions of the pattern rather than the whole pattern as a single vector. Which results in the benefit of working with the smallest size of network independent of the length of the pattern, in addition to several connections between the three neurons. These connections refer to the corresponding weight of every one of the vectors. Even though the expected number of vectors is, the number of connections will be just four according to the fact that every pair of orthogonal vectors has the same weights. Second, the learning process will be done to the predetermined portions of the pattern just in order to avoid learning identical portions more than once**.**

With a bipolar pattern representation, the elements will be either (1 or -1). The reason behind selecting this vector length is the shortest odd length of any of the vectors is three. And the need of this odd length, is because of the fact that the vector majority description (*vmd*; the summation of all of the elements of the vector are either positive or negative) [18].

5. Proposed System

In this work, a proposed system works on detecting and preventing SQL injection via using a method named Multi-Connect Architecture (MCA) is presented. These operations exactly focusing on Login operation that require a user name and password. The proposed system consists of preprocessing stage and MCA with two stages: training and testing stages, as illustrate in figure (2).

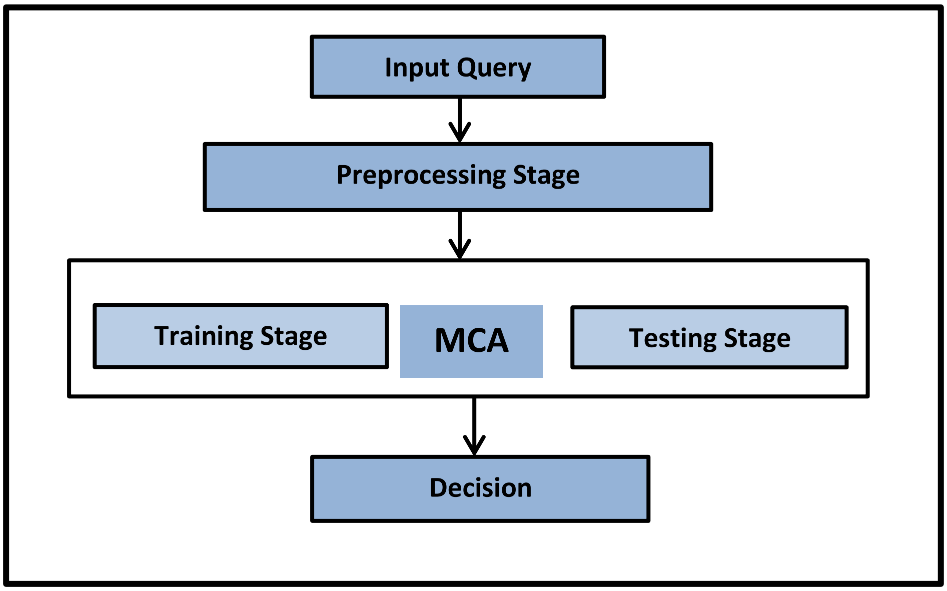


Figure (2): The proposed system stages

5.1 Preprocessing Stage

When a user enters a username and password, a SQL query is created. The queries entered by user usually have different lengths. MCA method need fixed lengths and bipolar representation as input patterns. For this reason, there is a need to address the queries to make it suitable for training and testing stages. The first step in the proposed system, the query is taken from the login page and passed to the preprocessing stage.

In this stage, the general structure of query is extracted by using parsing and used as input in SHA-256 to produce a set of vectors with fixed lengths (i.e. 256) as output. Finally, the output of SHA-256 convert to binary with corresponding bipolar representation as illustrated in figure (3).

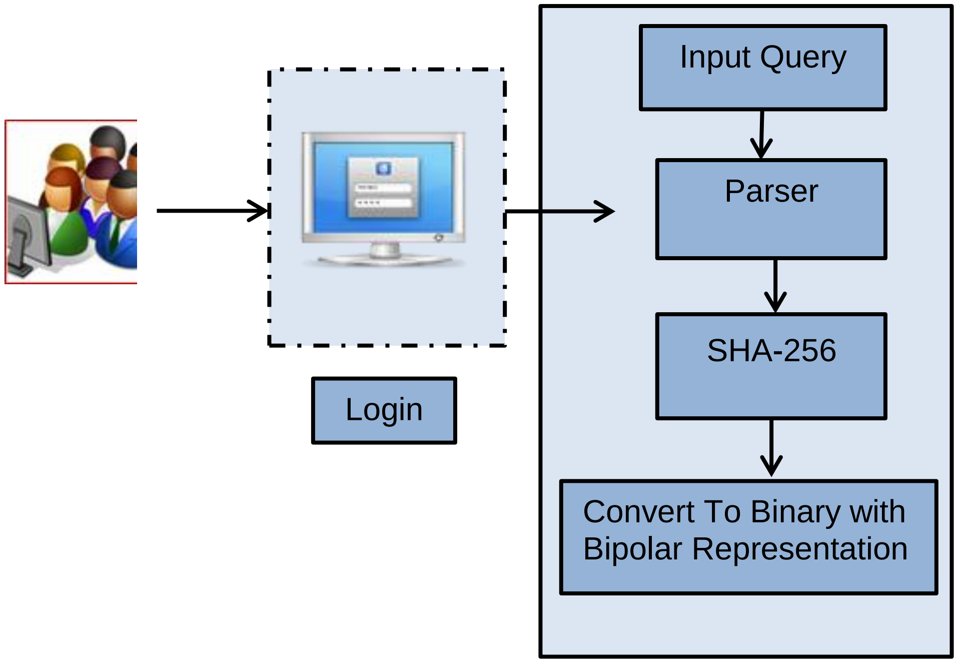


Figure (3): The Query preprocessing stage

5.2 Training Stage

The input of this stage is a set of normal queries. As the first step, the queries are passed to the preprocessing stage to produce a set of bipolar vectors with fixed lengths (i.e. 256). Then, the bipolar vectors are passed to the training stage.

The training stage includes the following steps:

Step 1: initializing the four weights matrices (i.e. $w_{0,} w_1, w_2, w_3$) as illustrated in figure (4) [19].Those weights are fixed and require no computations to be computed during this stage.

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

Figure (4): MCA associative memory weight matrices

Step 2: the bipolar vectors regarded as training pattern are split into n vectors v with length 3. Then, for each vector assign stored majority description (smd) and stored vector weights (svw).

Step 2.1: Smd is a set of vectors that which are the majority description for every vector of the training patterns, which assign by using Equation (4) **[19]**.

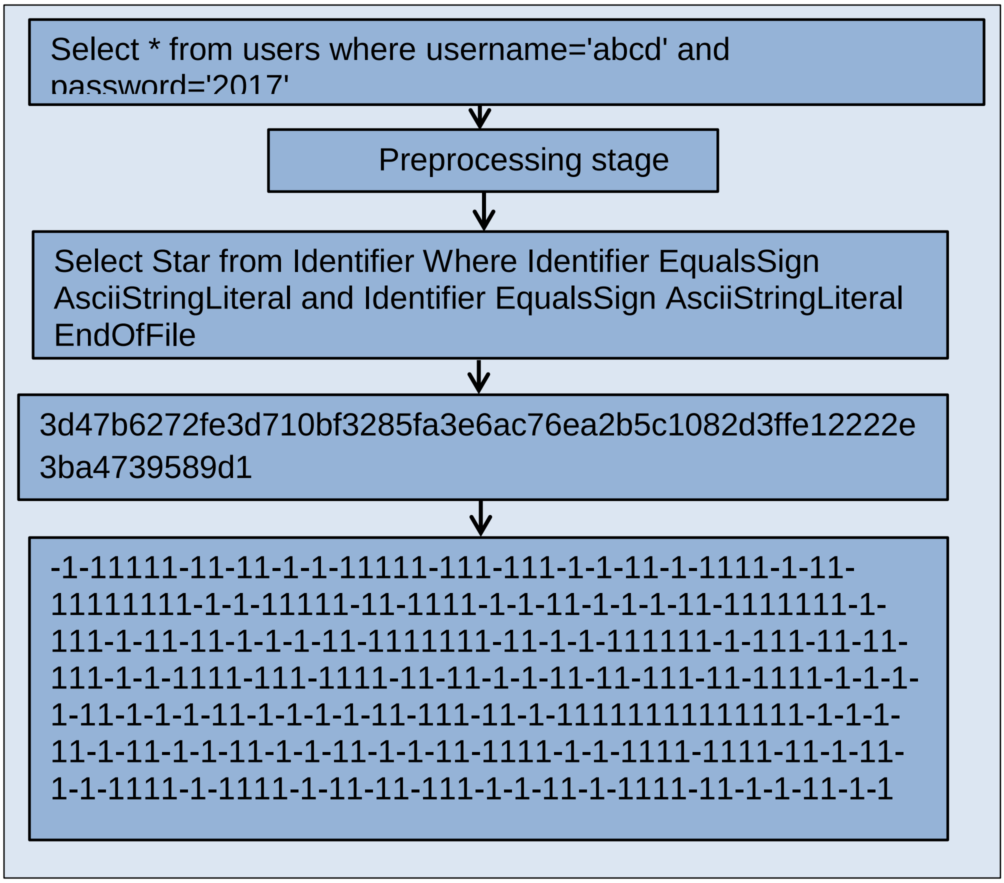
$$\text{md (v)} = \sum_{i=1}^{3} v\,i$$

$$\text{smd = hard\_limiter (md (v))} \begin{cases} 1 & \text{md(v)} \geq 1 \\ -1 & \text{md (v)} \leq \end{cases} \qquad (4)$$

Step 2.2: svw is a set of weights indexes, which assign by using Equation (5) **[19]**.

Step3: Smd and Svw are saved in the specific lookup table. Figure (5) shows the results of preprocessing stage and training stage on a simple normal query.

$$svw = f(Dcode(v)) \begin{cases} 0 \;\; or \;\; 7 & 0 \quad \{means\, w0 \\ 1 \;\; or \;\; 6 & 1 \quad \{means\, w1 \\ 2 \;\; or \;\; 5 & 2 \quad \{means\, w2 \\ 3 \;\; or \;\; 4 & 3 \quad \{means\, w3 \end{cases} \qquad (5)$$



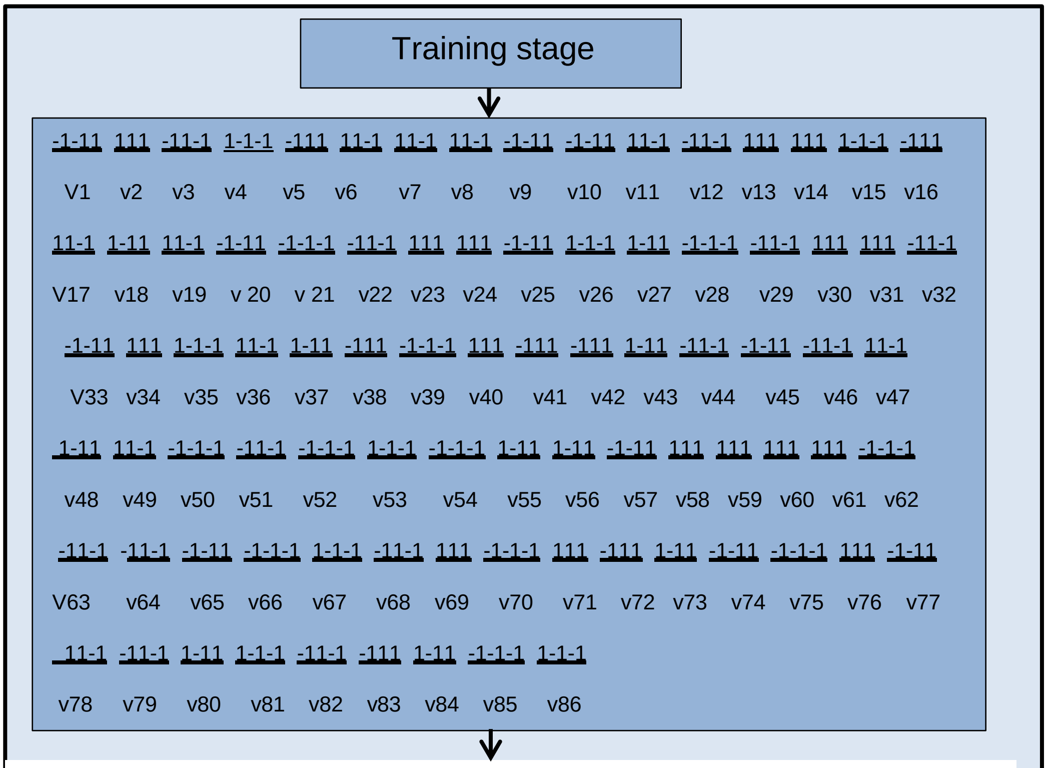
```
Select * from users where username='abcd' and
password='2017'
```

```
Select Star from Identifier Where Identifier EqualsSign
AsciiStringLiteral and Identifier EqualsSign AsciiStringLiteral
EndOfFile
```

```
3d47b6272fe3d710bf3285fa3e6ac76ea2b5c1082d3ffe12222e
3ba4739589d1
```

```
-1-11111-11-11-1-1-11111-111-111-1-1-11-1-1111-1-11-
11111111-1-1-11111-11-1111-1-1-11-1-1-1-11-1111111-1-
111-1-11-11-1-1-1-11-1111111-11-1-1-111111-1-111-11-11-
111-1-1-1111-111-1111-11-11-1-1-11-11-111-11-1111-1-1-1-
1-11-1-1-1-11-1-1-1-1-11-111-11-1-11111111111111-1-1-1-
11-1-11-1-1-11-1-1-11-1-1-11-1111-1-1-1111-1111-11-1-11-
1-1-1111-1-1111-1-11-11-111-1-1-11-1-1111-11-1-1-11-1-1
```

(a)



Vi(smd,svw)	Vi(smd,svw)	Vi(smd,svw)	Vi(smd,svw)	Vi(smd,svw)	Vi(smd,svw)	Vi(smd,svw)
V1(-1,1)	V2(1,0)	V3(-1,2)	V4(-1,3)	V5(1,3)	V6(1,1)	V7(1,1)
V8(1,1)	V9(-1,1)	V10(-1,1)	V11(1,1)	V12(-1,2)	V13(1,0)	V14(1,0)
V15(-1,3)	V16(1,3)	V17(1,1)	V18(1,2)	V19(1,1)	V20(-1,1)	V21(-1,0)
V22(-1,2)	V23(1,0)	V24(1,0)	V25(-1,1)	V26(1,3)	V27(1,2)	V28(-1,0)
V29(-1,2)	V30(1,0)	V31(1,0)	V32(-1,2)	V33(-1,1)	V34(1,0)	V35(-1,3)
V36(1,1)	V37(1,2)	V38(1,3)	V39(-1,0)	V40(1,0)	V41(1,3)	V42(1,3)
V43(1,2)	V44(-1,2)	V45(-1,1)	V46(-1,2)	V47(1,1)	V48(1,2)	V49(1,1)
V50(-1,0)	V51(-1,2)	V52(-1,0)	V53(-1,3)	V54(-1,0)	V55(1,2)	V56(1,2)
V57(1,3)	V58(1,0)	V59(1,0)	V60(1,0)	V61(1,0)	V62(-1,0)	V63(-1,2)
V64(-1,2)	V65(-1,1)	V66(-1,0)	V67(-1,3)	V68(-1,2)	V69(1,0)	V70(-1,0)
V71(1,0)	V72(1,3)	V73(1,2)	V74(-1,1)	V75(-1,0)	V76(1,0)	V77(-1,1)
V78(1,1)	V79(-1,2)	V80(1,2)	V81(-1,3)	V82(-1,2)	V83(1,3)	V84(1,2)
V85(-1,0)	V86(-1,3)					

(b)

Figure (5): The Normal Query (a) Preprocessing Stage (b) MCA Training

5.3 Testing Stage

When a user enters a username and password, a SQL query is created. As the first step, the query is taken from the login page and passed to the preprocessing stage to produce a set of bipolar vectors with fixed length (i.e.256). Then, the bipolar vectors are passed to the testing stage.

The testing stage consists of two parts: detecting part and preventing part. The mechanism work of the stage is started at the detecting part as illustrate in the following steps:

Step 1: initialize the four weights matrices as illustrated in figure (4).

Step 2: Initialize the energy function matrix as illustrated in Equation (6) **[19]**. Energy function is used to correlate the unknown query (i.e. normal or SQLI) and the normal query that has stored its data (which is the *svw* and *smd*) in the lookup table throughout the training stage.

$$e = \begin{bmatrix} -3 & 1 & 1 & 1 \\ 1 & -3 & 1 & 1 \\ 1 & 1 & -3 & 1 \\ 1 & 1 & 1 & -3 \end{bmatrix} \tag{6}$$

Step 3: the bipolar vectors regarded as testing pattern are divided into n vectors (v) of the length 3.

Step 3.1: for each vector assign test majority description (tmd) by using **Equation (7) [19].**

$$md(v)=\sum_{i=1}^{3} vi$$

$$tmd = hard\_limiter(\,(v \begin{cases} 1 \; md\,(v) \geq 1 \\ -1 \; md\,(v) \leq 0 \end{cases} \tag{7}$$

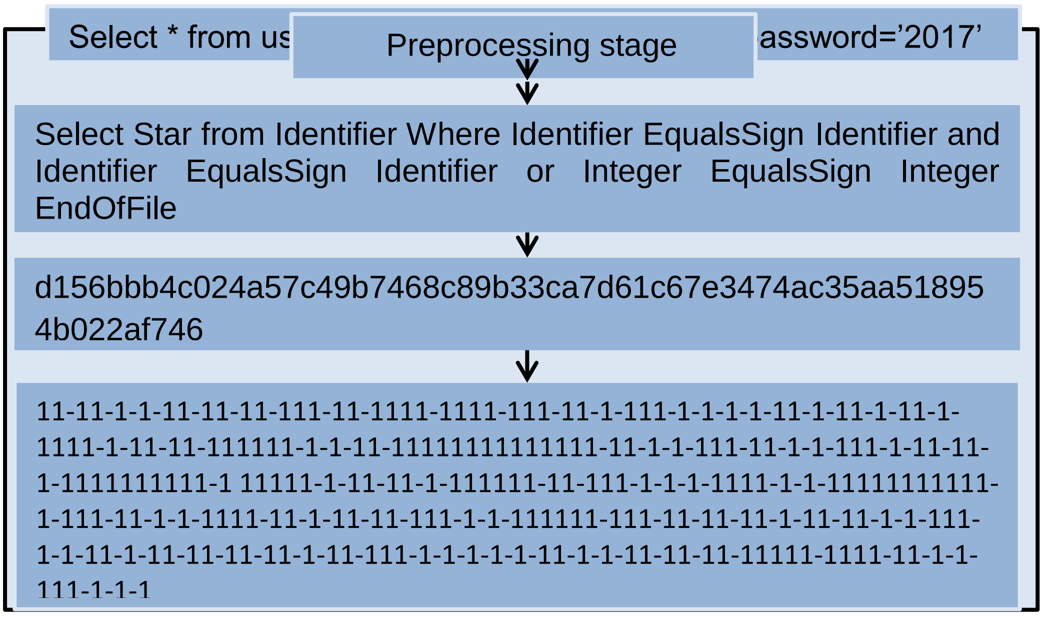
Step 3.2: for each vector assign test vector weights (tvw) by using Equation (8) **[19]**.

$$tvw = f(Dcode(v))\begin{cases} 0 \quad or \quad 7 \qquad 0 \qquad \{means\, w0 \\ 1 \quad or \quad 6 \qquad 1 \qquad \{means\, w1 \\ 2 \quad or \quad 5 \qquad 2 \qquad \{means\, w2 \\ 3 \quad or \quad 4 \qquad 3 \qquad \{means\, w3 \end{cases} \tag{8}$$

Step 4: Sum up the energy function for all of the *n* vectors in the unknown query (i.e. normal or SQLI) each with its corresponding vector in the normal query by using Equation (9) [19].

$$ep = \sum_{i=1}^{n} e[svw_i, tvw_i] \tag{9}$$

If the result of summation equal to -255, this means all the vectors in the unknown query are fully match with vectors in the normal query and the unknown query is normal. Otherwise the unknown query is SQL injection. Figure (6) shows the results of preprocessing stage and detecting part on a simple unknown query (i.e. normal or SQLI).

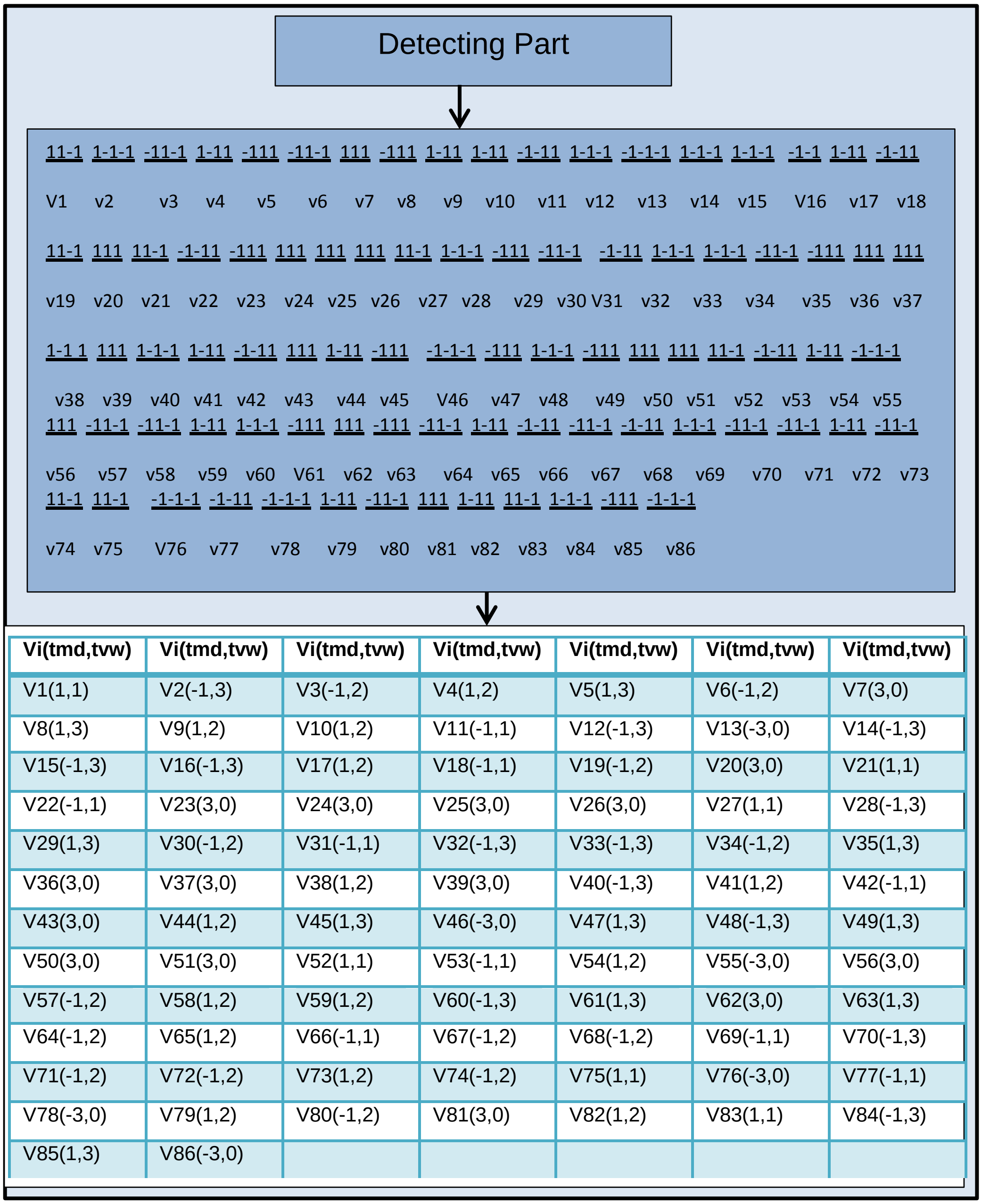


```
Select Star from Identifier Where Identifier EqualsSign Identifier and Identifier EqualsSign Identifier or Integer EqualsSign Integer EndOfFile

d156bbb4c024a57c49b7468c89b33ca7d61c67e3474ac35aa51895 4b022af746

11-11-1-1-11-11-11-111-11-1111-1111-111-11-1-111-1-1-1-1-11-1-11-1-11-1-
1111-1-11-11-111111-1-1-11-11111111111111-11-1-1-111-11-1-1-111-1-11-11-
1-1111111111-1 11111-1-11-11-1-111111-11-111-1-1-1-1111-1-1-11111111111-
1-111-11-1-1-1111-11-1-11-11-111-1-1-111111-111-11-11-11-1-11-11-1-1-111-
1-1-11-1-11-11-11-11-1-11-111-1-1-1-1-1-11-1-1-11-11-11-11111-1111-11-1-1-
111-1-1-1
```

(a)



Detecting Part

11-1	1-1-1	-11-1	1-11	-111	-11-1	111	-111	1-11	1-11	-1-11	1-1-1	-1-1-1	1-1-1	1-1-1	-1-1	1-11	-1-11	
V1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13	v14	v15	V16	v17	v18	
11-1	111	11-1	-1-11	-111	111	111	111	11-1	1-1-1	-111	-11-1	-1-11	1-1-1	1-1-1	-11-1	-111	111	111
v19	v20	v21	v22	v23	v24	v25	v26	v27	v28	v29	v30	V31	v32	v33	v34	v35	v36	v37
1-1 1	111	1-1-1	1-11	-1-11	111	1-11	-111	-1-1-1	-111	1-1-1	-111	111	111	11-1	-1-11	1-11	-1-1-1	
v38	v39	v40	v41	v42	v43	v44	v45	V46	v47	v48	v49	v50	v51	v52	v53	v54	v55	
111	-11-1	-11-1	1-11	1-1-1	-111	111	-111	-11-1	1-11	-1-11	-11-1	-1-11	1-1-1	-11-1	-11-1	1-11	-11-1	
v56	v57	v58	v59	v60	V61	v62	v63	v64	v65	v66	v67	v68	v69	v70	v71	v72	v73	
11-1	11-1	-1-1-1	-1-11	-1-1-1	1-11	-11-1	111	1-11	11-1	1-1-1	-111	-1-1-1						
v74	v75	V76	v77	v78	v79	v80	v81	v82	v83	v84	v85	v86						

Vi(tmd,tvw)	**Vi(tmd,tvw)**	**Vi(tmd,tvw)**	**Vi(tmd,tvw)**	**Vi(tmd,tvw)**	**Vi(tmd,tvw)**	**Vi(tmd,tvw)**
V1(1,1)	V2(-1,3)	V3(-1,2)	V4(1,2)	V5(1,3)	V6(-1,2)	V7(3,0)
V8(1,3)	V9(1,2)	V10(1,2)	V11(-1,1)	V12(-1,3)	V13(-3,0)	V14(-1,3)
V15(-1,3)	V16(-1,3)	V17(1,2)	V18(-1,1)	V19(-1,2)	V20(3,0)	V21(1,1)
V22(-1,1)	V23(3,0)	V24(3,0)	V25(3,0)	V26(3,0)	V27(1,1)	V28(-1,3)
V29(1,3)	V30(-1,2)	V31(-1,1)	V32(-1,3)	V33(-1,3)	V34(-1,2)	V35(1,3)
V36(3,0)	V37(3,0)	V38(1,2)	V39(3,0)	V40(-1,3)	V41(1,2)	V42(-1,1)
V43(3,0)	V44(1,2)	V45(1,3)	V46(-3,0)	V47(1,3)	V48(-1,3)	V49(1,3)
V50(3,0)	V51(3,0)	V52(1,1)	V53(-1,1)	V54(1,2)	V55(-3,0)	V56(3,0)
V57(-1,2)	V58(1,2)	V59(1,2)	V60(-1,3)	V61(1,3)	V62(3,0)	V63(1,3)
V64(-1,2)	V65(1,2)	V66(-1,1)	V67(-1,2)	V68(-1,2)	V69(-1,1)	V70(-1,3)
V71(-1,2)	V72(-1,2)	V73(1,2)	V74(-1,2)	V75(1,1)	V76(-3,0)	V77(-1,1)
V78(-3,0)	V79(1,2)	V80(-1,2)	V81(3,0)	V82(1,2)	V83(1,1)	V84(-1,3)
V85(1,3)	V86(-3,0)					

$e(svw_i,tvw_i)$	$e(svw_i,tvw_i)$	$e(svw_i,tvw_i)$	$e(svw_i,tvw_I)$	$e(svw_i,tvw_i)$	$e(svw_i,tvw_i)$	$e(svw_i,tvw_i)$
e(1,1)=-3	e (0,3)=1	e (2,2)=-3	e (3,2)=1	e (3,3)=-3	e (1,2)=1	e (1,0)=1
e (1,3)=1	e (1,2)=1	e (1,2)=1	e (1,1)=-3	e (2,3)=1	e (0,0)=-3	e (0,3)=1
e (3,3)=-3	e (3,3)=-3	e (1,2)=1	e (2,1)=1	e (1,2)=1	e (1,0)=1	e(0,1)=1
e (2,1)=1	e (0,0)=-3	e (0,0)=-3	e (1,0)=1	e (3,0)=1	e (2,1)=1	e(0,3)=1
e (2,3)=1	e (0,2)=1	e (0,1)=1	e(2,3)=1	e (1,3)=1	e (0,2)=1	e (3,3)=-3
e (1,0)=1	e (2,0)=1	e (3,2)=1	e (0,0)= -3	e (0,3)=1	e (3,2)=1	e(3,1)=1
e (2,0)=1	e (2,2)=-3	e (1,3)=1	e (2,0)=1	e (1,3)=1	e (2,3)=1	e(1,3)=1
e (0,0)=-3	e (2,0)=1	e (0,1)=1	e (3,1)=1	e (0,2)=1	e (2,0)=1	e (2,0)=1
e (3,2)=1	e 0,2)=1	e (0,2)=1	e (0,3)=1	e (0,3)=1	e (0,0)=-3	e (2,3)=1
e (2,2)=-3	e (1,2)=1	e 0,1)=1	e (3,2)=1	e (2,2)=-3	e (0,1)=1	e (0,3)=1
e (0,2)=1	e (3,2)=1	e (2,2)=-3	e (1,2)=1	e (0,1)=1	e (0,0)=-3	e(1,1)=-3
e (1,0)=1	e (2,2)=-3	e (2,2)=-3	e (3,0)=1	e(2,2)=-3	e (3,1)=1	e (2,3)=1
e (0,3)=1	e (3,0)=1	ep = -2 detect SQL injection				

(b)

Figure (6): The unknown query (a) Preprocessing Stage (b) Detecting Part

In preventing part, the query is prevented from executing on the database. Then, the user is blocked for five minutes and the IP and MAC addresses are sent to a specific table know as black list table. After five minutes, the user is allowed to retry the login but remains under surveillance, as illustrate in figure (7).

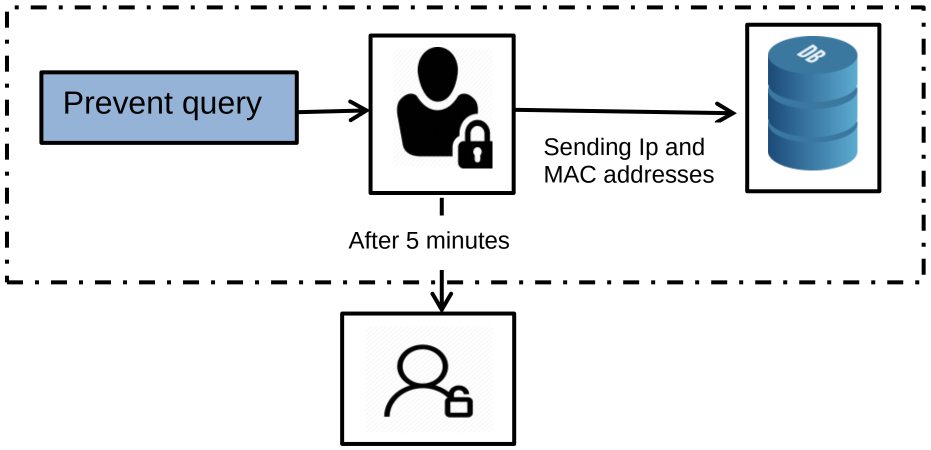


Figure (7): Preventing Part

6. Experimental Results and Discussion

The number of queries used in this paper is 130, details and result concerning those queries are listed below:

1. The number of queries that used with training stage are (30) as normal query.
2. The number of queries that used with testing stage are (100), 30 of 100 as normal and 70 of 100 as SQL injection.
3. The number of queries that have successfully been detecting its normal query are 30 of 30 query.
4. The number of queries that have successfully been detecting its SQLI and preventing it at the same time, they are 65 of 70 query.
5. The types of SQLI that have successfully been detecting its SQLI and preventing it at the same time, they are tautology, illegal incorrect queries, piggy-backed query, union query, blind injection, timing attack and stored procedure.
6. The result of the proposed system is (95%).

7. CONCLUSION

The proposed system works on detecting and preventing SQL injection via using a method named Multi-Connect Architecture (MCA). From result of the proposed system, the conclusion is as follows:

1- The MCA needed few number of queries in the training stage; this benefit was useful in this paper because it was difficult to get large number of queries.
2- By using MCA in detecting and preventing SQL injection, a high-level of protection has been obtained.

References

[1] Atul M. Tonge et al., “Cyber security: challenges for society- literature review”, IOSR Journal of Computer Engineering, ISSN: 2278-872, Vol.12, issue: 2, pp. 67-75, 2013.

[2] G.NIKHITA REDDY and G.J.UGANDER REDDY, “A Study of Cyber Security Challenges and Its Emerging Trends on Latest Technologies”, International Journal of Engineering and Technology, ISSN: 2049-3444, Vol.4, No. 1, 2014.

[3] Abhishek Kumar Baranwal, “Approaches to detect SQL injection and XSS in web applications”, EECE 571B, TERM SURVEY PAPER, APRIL 2012.

[4] Naveesha Saharan and Aditi Kajala, “SQL Injection and Proposed Methods:Comparison of tools and efficiency for preventing SQL attacks”, International Journal of Multidisciplinary Research and Development, ISSN: 2349-4182, Vol.2, Issue: 5, pp. 375-379, 2015.

[5] ManpreetKaur and Anand Kumar Mittal, "A REVIEW ON SQL INJECTION DETECTION AND PREVENTION TECHNIQUES", International Journal of Advanced Research in Computer and Communication Engineering, Volume: 4, Issue: 9, ISSN 2278-1021, 2016.

[6] Yogesh Bansal and Jin H. Park, “Multi-hashing for Protecting Web Applications from SQL Injection Attacks”, International Journal of Computer and Communication Engineering, Vol.4, NO.3, Pages: 187-195, 2015.

[7] WITT YI WIN and HNIN HNIN HTUN, “A Detection Method for SQL Injection Attacks in Web Applications”, International Journal of Scientific Engineering and Technology Research, ISSN 2319-8885, Vol.03, Issue: 07, pp. 1159-1163, 2014.

[8] Gaurav Shrivastava and Kshitij Pathak, “SQL Injection Attacks: Technique and Prevention Mechanism”, International Journal of Computer Applications, Vol.69, No: 07, pp.0975 – 8887, 2013.

[9] V.Shanmughaneethi and S.Swamynathan, “Detection of SQL Injection Attack in Web Applications using Web Services”, IOSR Journal of Computer Engineering, Volume: 1, Issue: 5, ISSN: 2278-0661, PP. 13-20, 2012.

[10] Ramya Dharam and Sajjan G. Shiva, “Runtime Monitoring Framework for SQL Injection Attacks”, International Journal of Engineering and Technology, Vol. 6, No.5, 2014.

[11] Manish Kumar and L.Indu, “Detection and Prevention of SQL Injection attack”, International Journal of Computer Science and Information Technologies, Vol.5, Issue: 1, pp. 374-377, 2014.

[12] Gowthami S and K.R.Prasanna Kumar, “Detecting SQL Injection Attacks in Web Application Using REGEX and Query Result Size”, International Journal of Innovative Research in Computer Science and Engineering, ISSN: 2394-6364, Vol.2, Issue: 5, 2016.

[13] Amit Banchhor and Tushar Vaidya , “SQL INJECTION: A SURVEY PAPER”, International Journal of Advanced Technology in Engineering and Science, ISSN: 2348 – 7550, Vol.3,ISSUE :1, 2015.

[14] Prem Shanker Dwivedi and Atma Prakash Singh, "SQL Injection Attack Prevention for Web Applications ", International Journal of Advanced Research in Computer and Communication Engineering, Volume: 4, Issue: 9, ISSN 2278-1021, 2015.

[15] ALFRED V. Aho, "COMPILERS: PRINCIPLES, TECHNIQUES, AND TOOLS ", second edition, ISBN: 0-321-49169-6, book, 2007.

[16] Vibhanshu Nehra and Nidhi Gulati, “DATABASE SECURITY AGAINST SQL INJECTION ATTACKS USING THREE LEVEL SECURITY APPROACH”, International Journal of Emerging Technology and Advanced Engineering, ISSN: 2321 - 3361, Vol. 6, Issue: 5, 2016.

[17] Xiaolin Cao et al., “A Compact SHA-256 Architecture for RFID Tags”, International Journal of Emerging Technology and Advanced Engineering, 2011.

[18] Emad I. Abdul Kareem, W.A.H. Ali Alsalihy, A. Jantan, ” Multi-Connect Architecture (MCA) Associative Memory: A Modified Hopfield Neural Network”, Intelligent Automation And Soft Computing, Volume:18, No. 3, Pages: 291-308, TSI® press, USA, 2012.

[19] Emad Mohammed Abod, ” Real Time System To Recognition Of Iraqi License Plate For Vehicle Tracking”, M.SC.Thesis, University of al-mustansiriyah-iraq, 2015.

كشف ومنع المخاطر في الامن السيبراني

أ.م.د بشار مكي نعمة*

الباحث: حنان عبد الولي عبد الله*

المستخلص: الأمن السيبراني هو مجموعة من التقنيات والعمليات التي تم تصميمها لحماية أجهزة الكمبيوتر والشبكات وقاعدة البيانات والتطبيقات من الهجمات، والوصول غير المصرح به، والتغيير، أو الدمار يمكن ايضا ان يكون جزءا كبيرا في تطوير تكنولوجيا المعلومات ، فضلا عن خدمات الإنترنت. وفي مجال الامن السيبراني ، هناك عدة اتجاهات ; اكبر واحد هو تطبيق ويب. تطبيق ويب هو واحد من المنصات الأكثر انتشارا لتقديم المعلومات والخدمات عبر الإنترنت في الوقت الحاضر. لذلك ، زادت اهمية تطبيق الويب . وفي الوقت نفسه، زادت المخاطر الامنيه ايضا. استنادا الى تقارير (OWASP)، هناك عشرة مخاطر تم ادراجها كاعلى عشرة مخاطر امنية لاي تطبيق ويب في العالم. حقن SQL هو واحد من اعلى المخاطر الامنية في OWASP، الذي اخذ بنظر الاعتبار في هذا البحث. لذا فان تامين والحفاظ على تطبيق الويب ضد حقن SQL مهمة صعبة للغاية ويمكن تصنيفها على انها تحدي . في هذا البحث ، تم تقديم نظام مقترح يعمل على الكشف عن ومنع حقن SQL عن طريق استخدام اسلوب يسمى (MCA) . عدد الاستعلامات المستخدمة في هذا البحث هي 130 استعلام . نتيجة النظام المقترح هي (95%) .

الكلمات المفتاحية: الأمن السيبراني، حقن SQL، الكشف، الوقاية، MCA.

*الجامعة المستنصرية/كلية العلوم/قسم علوم الحاسوب.