Journal of Basrah Researches ((Sciences)) Vol. (46). No. 1 (2020)

Available online at: http://brsj.cepsbasra.edu.iq

ISSN –1817 –2695

Encoding Query Based Lightweight Algorithm for Preventing SQL injection attack

Alaa Khudhair Shwaish[1], Mohammed Abdulridha Hussain[1] and Hayder A. A. Al-Kashoash[2]

[1] *Computer Science Department, Education College for pure Science, University of Basrah, Basrah, Iraq*
[2] *Computer Systems Technical Department, Technical Institute/ Qurna, Southern Technical University, Basrah, Iraq*

E-mail: alakhms@gmail.com , mohsubber@gmail.com , hayderaam@stu.edu.iq

Abstract

SQL injection attacks are still common issue in web applications. Although different techniques have been proposed to prevent SQL injection attack, it has a high impact on web applications, especially associated with large and sensitive databases. In this attack, an attacker can inject malicious code into the data entry field of the input form and bypass authentication, access, then modification and deletion of data within the database. In this paper, a lightweight algorithm is introduced based on encoding of the query named (EQA) considering HTTP traffic parameters e.g. request and response time and message length. EQA hides the SQL relationship with the database, and prevents some common types of SQL injection (Tautology, Piggybacked and Comment). EQA is implemented and tested using MySQL and PHP environment and Wireshark platform. The results demonstrate that the proposal has a good performance in terms of security level, reducing HTTP request and response time and message length.

Keywords: SQL injection, Encoding Query, Vulnerabilities, Tautology, Piggybacked, Comment, HTTP traffic.

1. INTRODUCTION

Web application hacking is one of the most common attacks on both organizations and individuals [1]. The most common attack on websites is SQL injection attack (SQLIA) which remained the same for many years [2]. SQL injection and cross-site scripting attacks increased by 38% in 2018 where the most popular WordPress hosts use SQL by default [3]. The Trust Wave reported in 2018 that the most common forms of web application attacks are those that exploit SQL injections where they constituted about 24% of the website attacks [4].

SQL injections are simply to exploit SQL vulnerabilities found in software where a web browser is only needed to execute an SQL attack. SQL injections usually happen during user input such as user name where the attacker inputs a SQL phrase that will carry out on the database. SQLIA can access sensitive information in the database, manipulate database data (update, delete and insert), run administration operations on the database (e.g. shutdown the database management system (DBMS)), write files into the file system, and return the content of a given file from the DBMS [5].

Several methods have been proposed to solve this problem [6-14], however according to our best knowledge; none considers reducing HTTP request time and response time, and HTTP message length. Thus, this work is motivated by this consideration to carry out and solve this issue. In this paper, we present a lightweight method in the prevention and authentication of access the database using the encoding of the query named (EQA). The proposed method provides a layer of prevention through the programming and design environments of databases and websites. The proposal is tested using the MySQL and PHP environment, and then analyze the data in the Wireshark platform. The results show that EQA reduces HTTP request and response time (i.e. Round Trip Time (RTT)) and message length during HTTP traffic, with a high level of security.

The remaining of the paper is organized as follows: In Section 2, some types of SQL injection are explained. Section 3 reviews most recent related work, In Section 4, problem statement is explained, and Section 5 describes the proposed method. In Section 6, Evaluation, results and discussion are given. Finally, Section 7 concludes the paper and states future work.

2. TYPES OF SQL INJECTION

The SQL attacker exploits some vulnerabilities that may exist in the database layer. Most of the vulnerabilities are when the developer ignores the input data filter that exists in the registration page or in the login page. The attacker uses one of the available vulnerabilities to carry out his agenda. This section gives brief definitions for the most popular types. Later, in Section 6, an example will be given for each one:

1. ***Tautology:***

In this type of attack, the attacker exploits the negligence of filtering conditional data during login by injecting a code that complies with the comparison condition and grants unauthorized real access to the site. The risk of this type is the attacker can manipulate the database as an authorized member [15-16].

2. Piggy-Backed Queries:

The concept of this type is to merge an additional query along with the initial query. Because of this fraud, the target database receives several queries for executing. Thus, the attacker may order to delete, modify or add. The most malicious code of this type is to drop the table from contents [17-18].

3. Commenting the Code:

Here the attacker can use the comment line in the entry form by setting double hyphen '--' where this prevents code to execute. In other words, attackers place a comment to disable the rest of the code that follows it [19].

3. RELATED WORK

Many methods have been proposed to solve the problem of SQL injection attack. However, none of them takes into account the request and response time and message length over HTTP traffic. This section reviews some related papers as follows:

M. Namdev et al. in [20] proposed, “A Novel Approach for SQL Injection

Prevention Using Hashing & Encryption" **(**SQL-ENCP), which includes computing the hash value of the username and password for any user and storing it in extra column in DB. This method is unable to handle all the SQLIA, but it can prevent tautology, union, and illegal query attacks.

Avireddy et al. in [21] proposed, " Random4: An application specific randomized encryption algorithm to prevent SQL injection" (Random4), The main idea of this algorithm is to convert user inputs into an encrypted query based on a lookup table which consists of four columns (lowercase, uppercase, numbers and special characters) of random values in 72 states. The encryption process for each symbol depends on the next symbol where it is chosen according to its status from the lookup table. The resulting query is complicated for the attacker. This method provides solution for only four attacks, and consumes the space of memory.

D. G. Kumar et al. in [22] proposed, "MAC based solution for SQL injection" (MACBSSI). The proposal works on client and server sides. At the client side, the method implements a filter function that checks the length and data type of the user input and detects the injection-sensitive characters/keywords. If it is found or the size is greater than the specified length, return an error message. At the server side, entropy is computed and MAC (message authentication code) is applied on static query (SMAC) and dynamic query (DMAC). Then, comparing SAMC and DMAC, if equal, the query is genuine. Else, the attack is detected and blocking IP address. However, this approach does not address SQLIA in case of stored procedures. Also, it consumes the network bandwidth because it checks the length .

A. K. Dalai et al. [19] proposed, "Neutralizing SQL injection attack using server side code modification in web applications" (NSIASSCM). This method only considers the SELECT statement containing a WHERE clause. All characters in the query are extracted and stored in a string S1. Then checking, if the type matches the required type, the input parameters are added to the query. Otherwise, the parameters are rejected. Then extracted new string S2, after converting into a simple statement by replacing the encoding. Then S1 and S2 are compared, if they match, the query is executed. Otherwise, cancel it. However, this approach does not address, illogical queries and Piggyback attacks.

Shaji N. et al. in [23] proposed, "An SQL Injection Defensive Mechanism Using Insertion Technique" (RIA), to filter the SQL command and store the data in a coded format. In the register phase, it finds the length, inserts a special character and reverses the user input. In the login phase, it repeats the same steps as in the register phase. Then, it compares the length and user input. However, this approach does not address alternative encoding and the stored procedure solved partially. In addition, calculation of length leads to increase of the size of the query, and then consuming bandwidth.

4. PROBLEM STATEMENT

Many web sites use open source web applications to provide certain services, such as (WordPress). Web applications are not only used by a private web site, but also by companies and governmental institutions. Often a database is used as the primary resource to retrieve information that is requested by the users [20]. The attacker can inject malicious code in the user input as shown in Fig. 1. The malicious input by user/attacker may modify the SQL query, and extract sensitive information from the database and get full control over it. In this paper, we present a practical solution to the problem that depends on the input encoding in order to prevent malicious hacking.

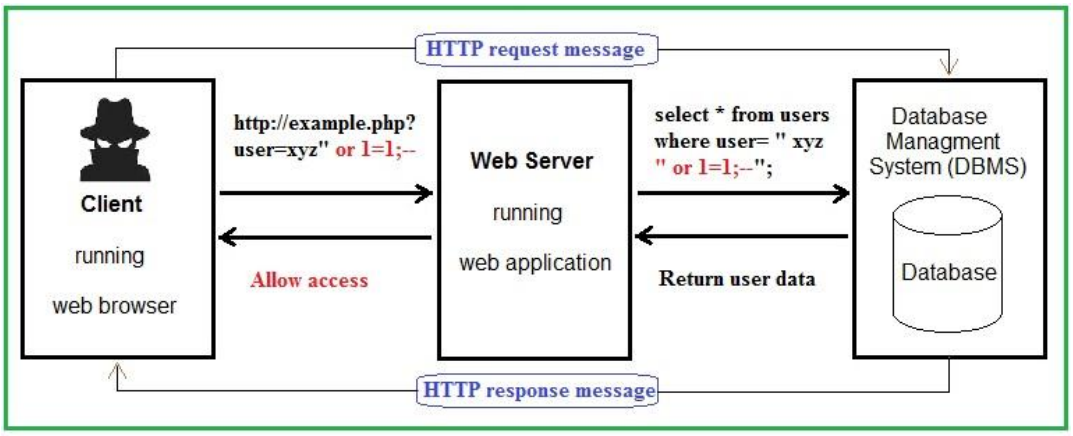

Fig. 1: SQL injection attack diagram.

5. PROPOSED METHOD

EQA is proposed to prevent SQL injection attack based on two mathematical operations on a matrix for the production of encoding to hinder the attacker. The first operation is matrix transpose, while the second operation is swap process between the main and secondary diagonals of the square matrix. The proposal is server-side solution and contains two phases: the registration and login phases.

5.1 Registration phase

The registration phase is to store the encoded username and password into the database table that is demonstrated in Algorithm 1. The result encoded value named (keyvalue) is saved in a single column in the table. The encoding is based on the square matrix operation where Table 1 and Table 2 show an encoding example.

Algorithm 1. EQA - Registration Phase:

1. User inputs: username & password.
2. Merge (username & password) into a single string.
3. Calculate string length.
4. Calculate the square root of the length.
5. Approximate the output of step 4 to the largest integer, to generate the key.
6. Convert the string into a square matrix by size of key value, and fill blank cells with an '*'.
7. Convert the square matrix into a transpose matrix.
8. Swap between the main and secondary diagonals.
9. Convert the final matrix into a vector string.
10. Store the resulting data (key value) into the user's table in DB.

Table 1. Data during registration:

username	***password***	***Keyvalue***
ala	*web*	*Alaweb*
moh	*sub*	*Mohsub*

Table 2. Data after the encoding process:

username	***password***	***text_enc***
		waleba*
		smoubh*

5.2 Login phase

The basic function of the login page is to grant authorization to access the web application where the decision is made upon username and password columns. In this proposal, the decision is made upon a single column named (key value) which helps to reduce complexity and prevent against SQL common attack. Algorithm 2 demonstrates the login phase where the web server handles the encoding and compares rests on the database.

Algorithm 2. EQA - Login Phase:

1. User inputs: username & password.
2. Merge (username & password) into a single string.
3. Calculate string length.
4. Calculate the square root of the length.
5. Approximate the output of step 4 to the largest integer, to generate the key.
6. Convert the string into a square matrix by size of key value, and fill blank cells with an '*'.
7. Convert the square matrix into a transpose matrix.
8. Swap between the main and secondary diagonals.
9. Convert the final matrix into a vector string.
10. Compare the vector string with key value in the users table.
11. If the comparison result is correct, login to the page. Otherwise, Error.

5.3 Example

In this sub section, the proposal is expressed in more details as an example with details program encoding algorithm.

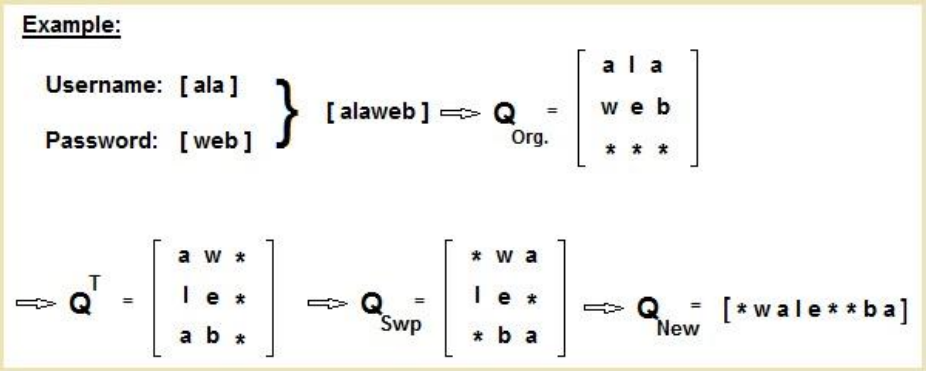

Fig. 2: An example of how the proposed algorithm works.

Pseudo code of login algorithm:

ENCODING_QUERY (username, password)

```
keyvalue ← ( )
Str ← Merge (username, password)
length ← strlen(Str)
len ← sqrt(length)
If ( intval(len) = len)
    Then sqr ← len
Else
    Sqr ← intval(len)+1
c ← 0
for i ← 0 i<sqr i ← i+1
  for j←0 j<sqr j ← j+1
   if(c < length)
     Str _array(j, i) ← Str (i)
   c ← c+1
   else
    Str _array(j, i) ← '*'
  for i ← 0 i<sqr i ← i+1
   for j←0 j<sqr j ← j+1
    if(i+j = sqr-1)
      temp ← Str_array(i, j)
      Str _array(i, j) ← Str _array(i, i)
          Str _array(i, i) ← temp
  for i ← 0 i<sqr i ← i+1
   for j←0 j<sqr j ← j+1
    keyvalue ← convert(keyvalue, Str _array(i, j))
    if (keyvalue = row(keyvalue)
      print 'You are logged in'
     else
      print 'Could not log you in'
```

Table 3. The variables description	
variable	***Meaning***
keyvalue	*Query after encoding.*
Str	*Query after merging input.*
length	*The total length of the query.*
len	*Length after taking the square root of it.*
sqr	*Length after rounding to the largest integer.*
c, i, j	*Counters.*
Str _array	*The query matrix.*
temp	*Temporary variable.*

6. RESULTS AND DISCUSSION

In this section, the proposal is evaluated in two scenarios using the MySQL and PHP environment. In the first scenario, the proposal is tested against some common attacks. In the second scenario, the proposed method is compared with normal mode and most recent related work that uses encoding to solve the SQLIA, RIA.

A. First Scenario:

In this scenario, different malicious queries (Tautology, Piggybacked and Comment) are entered in the authentication page. Table 4 shows that the proposal prevents the attacks.

Table 4. Test results of our proposed method against the some common attacks:

Method	State	Query	Output
Tautology	SQL Injection	*SELECT * FROM users WHERE username = a' or '1'='1 AND password = a' or '1'='1*	*Blocked*
	System Response	*SELECT * FROM users WHERE keyvalue = " 'oa'r1'= 1a 'o'''1*= 1r'*	
Piggy-Backed	SQL Injection	*SELECT * FROM users WHERE username = xyz AND password = a' or 1='1; drop table table1;*	*Blocked*
	System Response	*SELECT * FROM users WHERE keyvalue = 'eo1pexy ; r1z t t;a1ada*'brb=**'oll '*	
Comments	SQL Injection	*SELECT * FROM users WHERE username = --' AND password = --'*	*Blocked*
	System Response	*SELECT * FROM users WHERE keyvalue = '*----**'''*	
	System Response	*SELECT * FROM users WHERE keyvalue = '*eh746piea36x*ne46r)*=c(86)*O j7O**;* 57(**cx77* '*	
	System Response	*SELECT * FROM users WHERE keyvalue = 'W;UxyT NzDS;*HO''*	

Table 5. Comparison of different models with our model.

Methods	Tautology	Comment	Piggybacked
SQL-ENCP	Y	Φ	N
Random4	Y	Φ	Y
MACBSSI	Y	Φ	Y
NSIASSCM	Y	Y	N
RIA	Y	Φ	Y
Proposed method	Y	Y	Y

Y: Prevented.
N: Not Prevented.
Φ: Not indicated.

Table 5 refers to a summary of the results of the method, in comparison with other works related. This table shows that our work is superior to previous works.

B. Second Scenario:

In this scenario, Wireshark is used to analyze the length of HTTP message and round trip time (RTT) to send HTTP request and receive HTTP response of a website data transfer. Here, the proposed method is compared with normal mode and RIA.

The experiments are analyzed in two phases: register phase and login phase. Fig. 3 shows the HTTP message length in the registration phase for normal, the proposal and RIA. In this figure, the EQA does not add overhead bytes where it is almost similar to the normal and RIA methods. Fig. 4, shows that the response time when entering data in EQA is lower than both methods, normal and RIA. The reason for this is that our method stores one parameter (key value), while the other two methods stores (2 - 4) parameters. Thus, this takes time and reduces the effectiveness of performance.

Fig. 5 and Fig. 6 show the HTTP message length in the login phase under without attack and with attack respectively for normal, the proposal and RIA. In both cases, the length of HTTP message is the same for all methods as shown in Fig. 5 and Fig. 6. Fig. 7 and Fig. 8 show RTT for HTTP traffic in the login phase on without attack and with attack respectively. From these figures, it is clear that RTT for the proposal is less than normal and RIA method for reasons stated above. Also, form Fig. 8, the normal mode still takes more time because the attacker has bypassed the authentication of traffic.

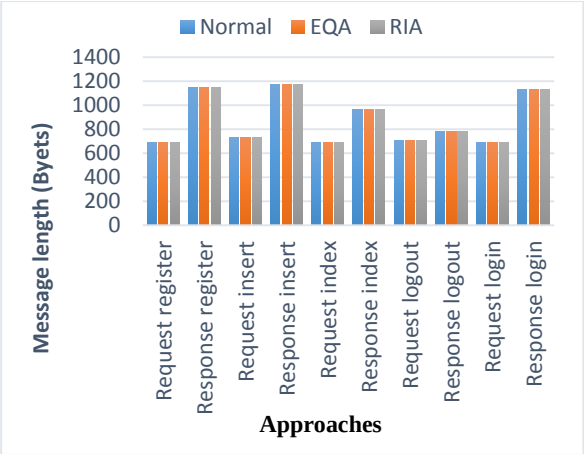

Fig. 3: The length of message during the registration phase.

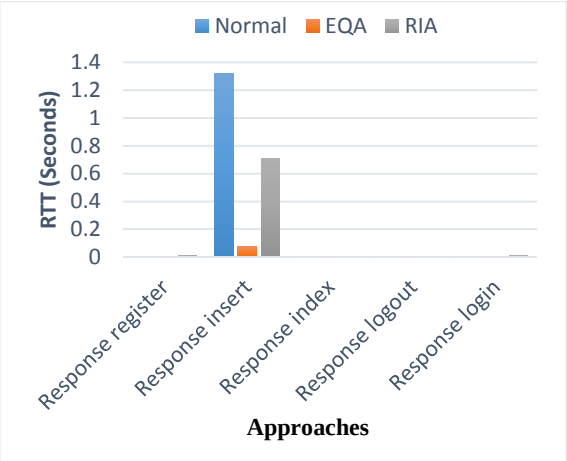

Fig. 4: RTT during the registration phase.

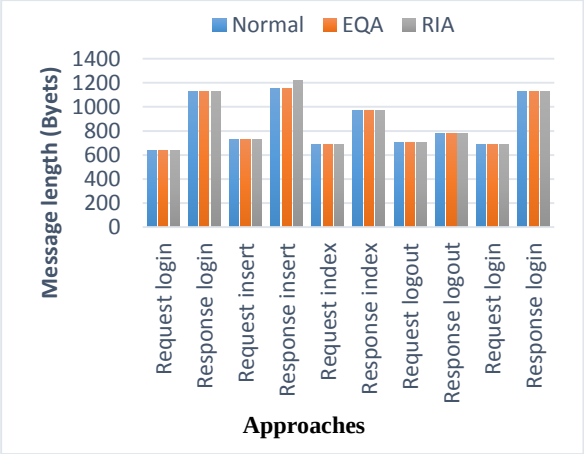

Fig. 5: The length of message during the login phase, without attack.

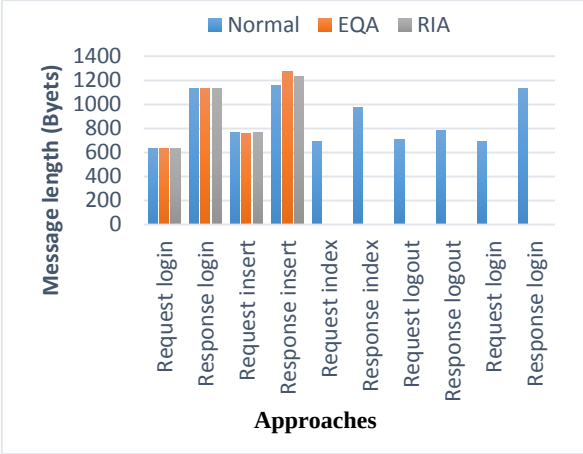

Fig. 7: The length of message during the login phase, with attack.

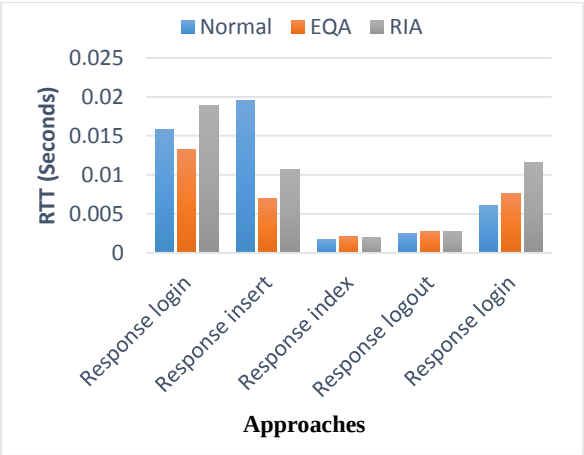

Fig. 6: RTT during the login phase, without attack.

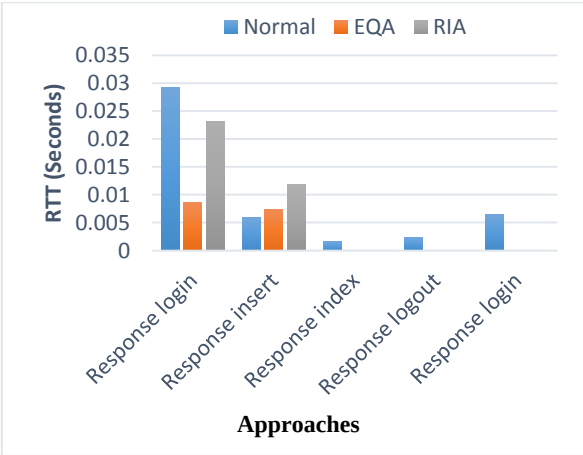

Fig. 8: RTT during the login phase, with attack.

7. CONCLUSION

Attacks on web applications are increasing day by day, where the most type of attack is SQLIA. Several models have been proposed to address this issue, but none of them considers HTTP traffic parameters such as RTT and message length. In this paper, we proposed a lightweight algorithm based on encoding of the query taking into account RTT and message length over HTTP traffic. The principle of the proposal depends on transpose matrix, and exchange of diagonals. The proposal tested under two scenarios and compared with comparative algorithms. The results show that the algorithm has better performance in terms of RTT and HTTP message length while keeping a high level of security against different types of attacks. In the future, we going to work on the expansion of this method to include the levels of other vulnerabilities.

REFERENCES

[1] P. Bhattacharyya, H. G. Sastry, V. Marriboyina, and R. Sharma, Smart and Innovative Trends in Next Generation Computing Technologies: Third International Conference, NGCT 2017, Dehradun, India, October 30-31, vol. 828. Springer Singapore, 2018.

[2] Lucero, Davalos, Vizcarra, 29th May ,19' Top 10 Web Security Vulnerabilities,https://cai.tools.sap/blog/ top-10- web- security-vulnerabilities-to-watch-out-for- in 2019.

[3] Gary Stevens, Sep. 24, 2019, https://hostingcanada.org/website-vulnerabilities, 2018.

[4] Acunetix, "Web Application Vulnerability Report 2019," p. 27, 2019, https://www.acunetix.com/ web-Vulnerability- scanner, 2019.

[5] R. Alsahafi, "SQL injection attacks: Detection and prevention techniques," Int. J. Sci. Technol. Res., vol. 8, no. 1, pp. 182–185, 2019.

[6] K. I. Takane, S. Tajima, and H. Kouchi, "Structural and expression analysis of uricase mRNA from Lotus japonicus," Mol. Plant-Microbe Interact., vol. 13, no. 10, pp. 1156–1160, 2000.

[7] W. G. J. Halfond and A. Orso, "Preventing SQL injection attacks using AMNESIA," Proc. - Int. Conf. Softw. Eng., vol. 2006, pp. 795–798, 2006.

[8] D. Kar and S. Panigrahi, "Prevention of SQL Injection attack using query transformation and hashing," Proc. 2013 3rd IEEE Int. Adv. Comput. Conf. IACC 2013, no. November, pp. 1317–1323, 2013.

[9] M. Kaushik and G. Ojha, "Attack Penetration System for SQL Injection," Int. J. Adv. Comput. Res., vol. 4, no. 2, pp. 724–732, 2014.

[10] D. Kar, K. Agarwal, A. K. Sahoo, and S. Panigrahi, "Detection of SQL injection attacks using hidden markov model," Proc. 2nd IEEE Int. Conf. Eng. Technol. ICETECH 2016, no. March 2016, pp. 1–6, 2016.

[11] L. Batista, G. Silva, V. Araújo, "Fuzzy neural networks to create an expert system for detecting attacks by SQL Injection," Int. J. Forensic Comput. Sci., vol. 13, no. 1, pp. 8–21, 2018.

[12] S. Scholarworks and K. Ross, "SQL Injection Detection Using Machine Learning Techniques and Multiple Data Sources," 2018.

[13] A. Sinha, C. Pandya, D. Patil, M. Mulmuley, R. Makh, and A. Meshram, "Defending Against

Web Application Attacks Using Offensive Decoy Techniques," vol. 6, no. 1, pp. 284–290, 2019.

[14] H. Gu, J. Zhang, T. Liu, M. Hu, "DIAVA: A Traffic- Based Framework for Detection of SQL Injection Attacks and Vulnerability Analysis of Leaked Data," IEEE Trans. Reliab., no. 1, pp. 1–15, 2019.

[15] J. Viegas and A. Orso, "A classification of SQL-injection attacks and countermeasures," Int'l Symp. Secur. Softw., 2006.

[16] B. Shehu and A. Xhuvani, "A Literature Review and Comparative Analyses on SQL Injection: Vulnerabilities, Attacks and their Prevention and Detection Techniques," Int. J. Comput. Sci. Issues, vol. 11, no. 4, p. 28, 2014.

[17] S. M. H. Chaki and M. Mat Din, "A Survey on SQL Injection Prevention Methods," Int. J. Innov. Comput., vol. 9, no. 1, pp. 47–54, 2019.

[18] A. Tajpour and M. J. Z. Shooshtari, "Evaluation of SQL injection detection and prevention techniques," Proc. - 2nd Int. Conf. Comput. Intell. Commun. Syst. Networks, CICSyN 2010, no. July 2010, pp. 216–221, 2010.

[19] A. K. Dalai and S. K. Jena, "Neutralizing SQL injection attack using server side code modification in web applications," Secur. Commun. Networks, vol. 2017, 2017.

[20] M. Namdev, F. Hasan, and G. Shrivastav, "A Novel Approach for SQL Injection Prevention Using Hashing & Encryption (SQL-ENCP)," vol. 3, no. 5, pp. 4981–4987, 2012.

[21] S. Avireddy, V. Perumal, N. Gowraj, "Random4: An application specific randomized encryption algorithm to prevent SQL injection," Proc. 11th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. Trust. - 11th IEEE Int. Conf. Ubiquitous Comput. Commun. IUCC-2012, pp. 1327–1333, 2012.

[22] D. G. Kumar and M. Chatterjee, "MAC based solution for SQL injection," J. Comput. Virol. Hacking Tech., vol. 11, no. 1, pp. 1–7, 2015.

[23] Shaji N. Raj, Elizabeth Sherly, "An SQL Injection Defensive Mechanism Using Insertion Technique" Kottayam, Kerala, India, vol. 828. Springer Singapore, 2018.

خوارزمية خفيفة تستند الى ترميز الاستعلام لمنع هجوم حقن SQL

علاء خضير شويش[1]، محمد عبد الرضا حسين[1]، حيدر احمد عبد المحسن الكشوش[2]

[1]قسم علوم الحاسوب، كلية التربية للعلوم الصرفة، جامعة البصرة, البصرة, العراق

[2]قسم تقنيات أنظمة الحاسوب، المعهد التقني / القرنة، الجامعة التقنية الجنوبية, البصرة, العراق

E-mail: alakhms@gmail.com , mohsubber@gmail.com , hayderaam@stu.edu.iq

خلاصة:

هجمات حقن SQL لا تزال مشكلة شائعة في تطبيقات الويب. على الرغم من أنه تم اقتراح تقنيات مختلفة لمنع هجوم حقن SQL، إلا أنه له تأثير كبير على تطبيقات الويب، خاصةً المرتبطة بقواعد البيانات الكبيرة والحساسة. في هذا الهجوم، يمكن للمهاجمين إدخال تعليمات برمجية ضارة في حقل إدخال البيانات في نموذج الإدخال وتجاوز المصادقة والوصول ثم تعديل البيانات وحذفها داخل قاعدة البيانات. في هذه الورقة، نقدم خوارزمية خفيفة الوزن تستند على ترميز الاستعلام، اسميناها (EQA) تأخذ في الاعتبار معلمات حركة مرور HTTP مثل طلب ووقت الاستجابة وطول الرسالة. تخفي EQA علاقة SQL مع قاعدة البيانات، وتمنع بعض الأنواع الشائعة من حقن SQL مثل (Tautology, Piggybacked and Comment). يتم تطبيق واختبار EQA باستخدام بيئة MySQL و PHP ومنصة Wireshark. توضح النتائج أن اقتراحنا له أداء جيد من حيث مستوى الأمان، مما يقلل من طلب HTTP ووقت الاستجابة وطول الرسالة.

الكلمات المفتاحية:حقن SQL, ترميز الاستعلام, نقاط الضعف, حركة مرور HTTP.