OPEN ACCESS

Journal of Artificial Intelligence

ISSN 1994-5450
DOI: 10.3923/jai.2019.

Research Article

A Two-Phase Pattern Matching-parse Tree Validation Approach for Efficient SQL Injection Attacks Detection

Randa Osman Morsi and Mona Farouk Ahmed

Department of Computer Engineering, Faculty of Engineering, Cairo University, Giza, Egypt

Abstract

Background and Objective: Data is one of the most valuable assets as it is the core for any organization website. SQL Injection Attack (SQLIA) is the way by which hackers gain access to data. An approach was proposed in this paper to efficiently detect SQLIA. **Methodology:** One of the most powerful algorithms, Parsing Tree validation (PT), depends only on accurate detection but takes much time so combining it with a fast dynamic algorithm with the purpose of learning and storing the malicious input patterns to compare with the next coming inputs will be a great achievement. An algorithm was proposed that is based on the combination of two of the existing detection algorithms: pattern matching algorithm using Aho-Corasick (AC) and PT. **Results:** Experiments showed that the proposed approach guarantees high accuracy of 99.9%, reasonable time which was 53.6% of PT's time and less memory usage. **Conclusion:** SQLIA is one of the most severe threats to the database. In general, the approaches that provide the best guard for the database against SQLIA are those that make use of a mix of primitive approaches as this leads to strengthening their merits and improving their weaknesses.

Key words: Aho-corasick, data security, parsing tree, pattern matching algorithm

Citation: Randa Osman Morsi and Mona Farouk Ahmed, 2019. A two-phase pattern matching-parse tree validation approach for efficient SQL injection attacks detection. J. Artif. Intel., CC: CC-CC.

Corresponding Author: Randa Morsi, Department of Computer Engineering, Faculty of Engineering, Cairo University, Giza, Egypt Tel: 00201003593792

Competing Interest: The authors have declared that no competing interest exists.

Data Availability: All relevant data are within the paper and its supporting information files.

INTRODUCTION

SQLIA allows hackers to penetrate the databases of banks or companies for illegal purposes like obtaining confidential information or controlling and damaging these databases. There are many ways malicious inputs can be injected in SQL[1]. Injection can be done through user input, cookie fields containing attack strings, server variables and second order injection[2-3].

Most of the proposed SQLIA detection and prevention approaches targeted the five most critical layers in the end-to-end web application architecture. Web client layer approaches were focused on client side through client's web browser[4]. Most of the Web application firewall layer approaches focused on examining the HTTP request and finding anomalies. A solution was proposed that depends on learning using neural network[5]. Another approach was based on a hybrid system using Bayesian classifier and pattern matching[6]. The majority of contributions were focused at the Web application layer and a general classification was presented for them[7-8]. Defensive coding approaches require programmers to write the application code in a specific manner. Vulnerability Testing discovers and fixes possible injection hotspots. Some approaches were based on automatic generation of test inputs and cases[9-11], while others detected server-side vulnerabilities by designing a black-box testing method[12]. Yet, another approach was based on webpage scanning to discover the vulner abilities and it used a defined database error table[13]. Prevention focuses on organizing a model of sample queries and the application's response during normal use. SQLLEARN was developed for learning safe SQL statements by program tracing techniques[14]. Negative taint model used evasion methods by maintaining a vulnerability lookup table for all possible attacks[15]. Parsing Tree (PT) validation was based on building a tree structure out of the input query before execution for validating its components[16]. The AC pattern matching technique was focused on finding the occurrences of certain words in the queries[17]. Another technique modeled a SQL query as a graph of tokens and trained an SVM classifier to be able to recognize possible malicious inputs[18]. Many approaches fall in the Database Firewall layer. Some techniques validated the queries at runtime and detected the attacks compared with normal query whose specifications were predefined using SQL-IDS (Intrusion Detection Systems)[19]. Another approach executed each query twice, once on the SQL database and another one on a copy of it maintained by LDAP (Lightweight Directory Access Protocol)[20]. A novel technique dynamically analyzed the developer-intended query result size for any input and compared it against the result of the actual query[21]. Concerning the Database Sever layer approaches: An approach was proposed that was mainly based on data mining but was dedicated to PostgreSQL database as it used the database server internal query trees[22]. Another approach kept an anomaly pattern list and used Apriori for checking the queries[23]. An experimental evaluation of the effectiveness of SQLI detection tools showed that they had very low effectiveness[24]. Other approaches can be classified as Multi-Layer approaches. One of them was based on enhancing the security using one IDS at the web server and another at the back end database[25]. The code conversion algorithm[26] converted the user input to code like ASCII in a database table. A combined technique was proposed to carry the best features of both the PT validation technique and the code conversion one[1].

Most proposed approaches trade accuracy over time or have high memory requirements; the PT validation algorithm is one of the most powerful algorithms, however, it offers accurate detection at the expense of time. For the AC pattern matching algorithm the dynamic phase involves human intervention for checking which is a major drawback and was the driver for proposing an approach that replaced the human intervention part with a stable algorithm for detection like PT. Therefore, this work proposed a hybrid SQLIA detection algorithm which guarantees good accuracy while providing very low time and little dependency on memory usage. The proposed approach also provides a learning methodology.

MATERIALS AND METHODS

Overview of the approach: The proposed approach was derived from the static and dynamic algorithm which used AC in the static phase and manual human intervention in the dynamic phase[17]. The approach tried to combine the best features of both AC pattern matching algorithm and PT validation technique[16] so it was called AC-PT. It replaced the manual part in the dynamic phase of the static and dynamic algorithm by the PT technique.

Initialization: At the beginning of operation of the AC-PT the PT module was the major part and AC was passive; PT parsed the user input and checked whether it was vulnerable, if this input was safe it would pass normally but if there was any chance of vulnerability it would be detected and prevented then it would be added to the static pattern list. Once this list encompassed more than min_attacks (where min_attacks is a threshold which indicates the minimum number of attacks that has to be found in the attacks list for the AC to start matching the input against the patterns in the list) AC could start making a side decision and try to match the input pattern against the attacks list so AC's response could be monitored. The AC-PT is a two-phase algorithm with AC as the static phase and PT as the dynamic one.

Static phase: In static analysis a modified version of AC was used which added the normalization part of the input string. The user input was to kenized and normalized before processing for more accurate and fast matching for example:

Administrator OR '1' = '1'

was transformed to an array of tokens (strings) with the values:

{"_TEXT_", "OR", "_NUMBER_", "=", "_NUMBER_"}

Then this input was compared with the normalized set of attacks that were stored as list of known attacks where the attacks in the list were normalized in the same manner. The matching started when a letter was found in the input that matched the first letter of a keyword in the input list. However, the proposed approach compared the first letter and moved forward a number of letters equaled the count of letters in the keyword then compared the last letters if both of them matched then comparison for the rest of the letters in between was made else it escaped this substring and looked for the first letter in the next keyword and so on. The proposed algorithm was shown in Fig. 1.

As shown in Fig. 1 when the user input was first presented to the AC module the input was normalized then matched against all the normalized patterns in the stored attacks list. The matching procedure resulted in a matching percentage which was the proportion of the correctly matched parts against the whole string. If there was a strong matching, where the matching percentage was greater than 70%, the input was rejected immediately and this was the most efficient case as no need to spend time in going to PT. If the matching percentage was below 30% then some more checking was needed to decide whether it was possible at the current state to completely depend on the AC's matching decision. In this case the checking required is called the maturity check. The AC's maturity is defined in Eq. 1:

$$\text{Maturity} = \frac{\text{Detection flag}}{\text{No.of attacks}} \tag{1}$$

The maturity is defined as follows: Each time an attack was made to the system, where initially PT was responsible for deciding that it was an attack, a counter called "detection flag" was incremented if the attack was successfully detected using the AC. AC's maturity is the ratio of the number of attacks correctly detected by the AC to the total number of submitted attacks.

If the maturity exceeded the maturity threshold which was predefined as 80%, the AC was said to be "Mature". Fig. 1 showed that when the matching percentage was below 30% the AC-PT asked "Is Mature?"which meant: did AC reach the maturity level as it learnt from the previous iterations and the attacks list was sufficient to help AC in taking a decision or it was still not mature and should go to PT for a more accurate decision. If PT was asked to decide, AC would also make a side decision whose result would affect the maturity calculation. The vague area was where the matching percentage was between 30 and 70% where it was not clear whether to decide that the input was not accurately matched so it was a legal one or that it was somewhat matched to a convenient percentage so it could be taken as an attack. In this case AC needed to make a double check for higher accuracy so the PT algorithm was applied.

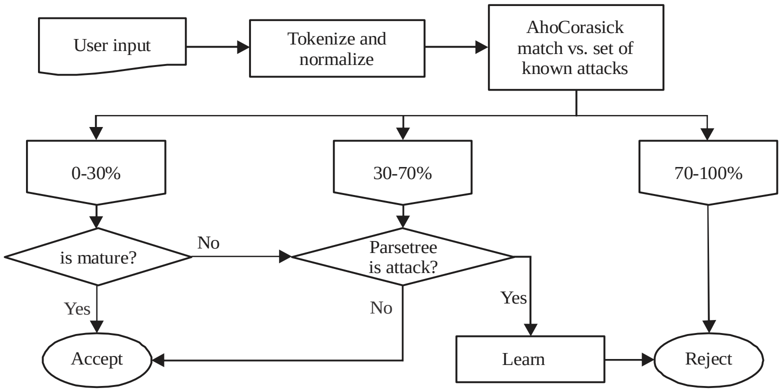


Fig. 1: AC-PT algorithm flowchart

Dynamic phase: In the dynamic analysis (PT) the original structure of the SQL query was compared with the SQL query statement containing the hard coded original parts with the user input inserted in the empty leaf nodes in the parse tree. While comparing the structures, if mismatch was detected the pattern would be added to the attacks list as a malicious input to improve the overall performance by incremental learning of the AC module for the static phase. This saved time if a similar attack was to be presented. The proposed implementation of AC-PT solved an inherent problem in the original PT validation technique where an alarm was raised even if legitimate user was having extra blank spaces in his/her input. Inputs with extra spaces might be a typing error that would not harm the database. AC-PT implementation dealt with this by ignoring extra spaces between different keywords and user inputs while parsing the input string leaving only one space as usual but in case of spaces in between the characters of username or password it raised an alarm. This gave an extra advantage for AC-PT over PT which was fewer resulting false positives.

Summary: In general, PT kept working alone till AC became mature. The AC's maturity was measured by recording the accuracy of its side decisions and when it reached a defined threshold it was said to be mature. When AC became mature it could take a decision alone in most cases. The algorithm as a whole became faster and the AC part got more mature by time as more patterns were introduced.

RESULTS

AC-PT was tested on all types of SQLIAs by generating SQL queries containing legitimate SQL commands and SQLIA using sqlmap[1] attacking tool to generate a number of SQLIA patterns. AC-PT was run on Intel i5 core 8GB RAM machine.

The AC-PT accuracy calculated after running an experiment on 400,000 attacks was about 99.9%. Table 1 showed the accuracy of AC-PT compared to two related algorithms AC and PT. The experiments also made showed that the number of false positive alerts made by AC-PT was about 16 only for a sample of 5000 SQL generated queries, this was shown in Table 2.

Figure 2 showed that after a period of time the attacks list contained all possible attack patterns which made the AC part of AC-PT stable and its maturity approached 1. In general, after learning the possibility of having totally different and completely new attacks approached zero so after maturity the main algorithm was AC.

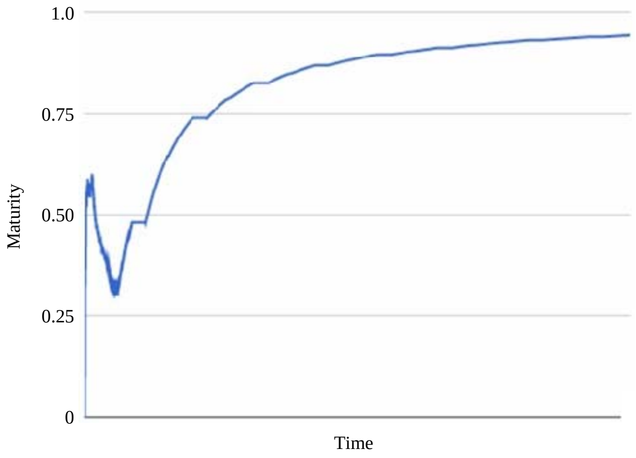


Fig. 2: Maturity vs. time in the AC part of AC-PT. After sometime the calculated maturity increases and it approaches 1, at this time the AC is said to be learnt and can make most decisions on its own

Table 1: AC-PT accuracy compared to that of AC and PT on 400,000 attacks

Algorithm	Accuracy (%)
AC	94.0
PT	~99.9
AC- PT	~99.9

PT: Parsing tree and AC: Aho-corasick

Table 2: Comparison between AC-PT and PT in terms of the number of false positives

Algorithm	No. of false positive alerts/5000 sample
PT	40
AC- PT	16

PT: Parsing tree

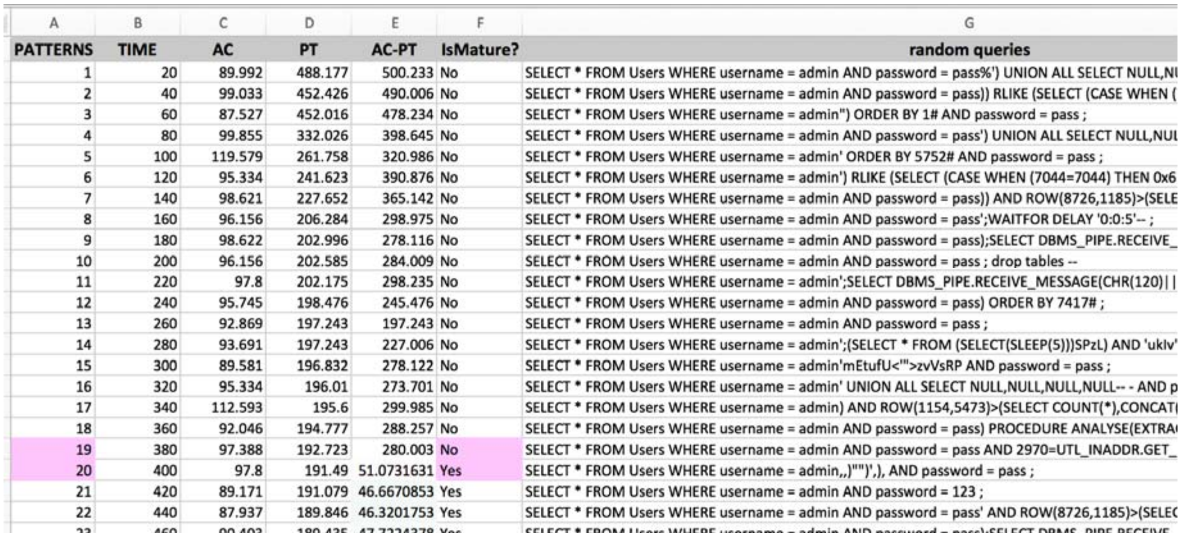
Turning to the other major performance parameter which is time, the average time taken for AC-PT was calculated over all the experimented patterns to be 57.32 msec.

Figure 3 showed the time performance comparison of AC-PT and both AC and PT.

DISCUSSION

From the previous results it was obvious that as AC-PT mainly depends on PT initially for learning and filling the attacks list so it achieved almost the same accuracy as PT. Also from Fig. 3 it was evident that at the beginning of the AC-PT the time was close to PT as it was the main component of the algorithm before maturity then after learning its time performance was near that of AC as all submitted attacks had more than 70% matching as the attacks list already contained all patterns after learning.

The accuracy and time performance of AC-PT were compared with the PT algorithm[16] alone and also with the AC pattern matching[17]. The three algorithms were run on the



A	B	C	D	E	F	G
PATTERNS	TIME	AC	PT	AC-PT	IsMature?	random queries
1	20	89.992	488.177	500.233	No	SELECT * FROM Users WHERE username = admin AND password = pass%') UNION ALL SELECT NULL,NU
2	40	99.033	452.426	490.006	No	SELECT * FROM Users WHERE username = admin AND password = pass)) RLIKE (SELECT (CASE WHEN (
3	60	87.527	452.016	478.234	No	SELECT * FROM Users WHERE username = admin") ORDER BY 1# AND password = pass ;
4	80	99.855	332.026	398.645	No	SELECT * FROM Users WHERE username = admin AND password = pass') UNION ALL SELECT NULL,NUL
5	100	119.579	261.758	320.986	No	SELECT * FROM Users WHERE username = admin' ORDER BY 5752# AND password = pass ;
6	120	95.334	241.623	390.876	No	SELECT * FROM Users WHERE username = admin') RLIKE (SELECT (CASE WHEN (7044=7044) THEN 0x6
7	140	98.621	227.652	365.142	No	SELECT * FROM Users WHERE username = admin AND password = pass)) AND ROW(8726,1185)>(SELE
8	160	96.156	206.284	298.975	No	SELECT * FROM Users WHERE username = admin AND password = pass';WAITFOR DELAY '0:0:5'-- ;
9	180	98.622	202.996	278.116	No	SELECT * FROM Users WHERE username = admin AND password = pass);SELECT DBMS_PIPE.RECEIVE_
10	200	96.156	202.585	284.009	No	SELECT * FROM Users WHERE username = admin AND password = pass ; drop tables --
11	220	97.8	202.175	298.235	No	SELECT * FROM Users WHERE username = admin';SELECT DBMS_PIPE.RECEIVE_MESSAGE(CHR(120)||
12	240	95.745	198.476	245.476	No	SELECT * FROM Users WHERE username = admin AND password = pass) ORDER BY 7417# ;
13	260	92.869	197.243	197.243	No	SELECT * FROM Users WHERE username = admin AND password = pass ;
14	280	93.691	197.243	227.006	No	SELECT * FROM Users WHERE username = admin';(SELECT * FROM (SELECT(SLEEP(5)))SPzL) AND 'uklv'
15	300	89.581	196.832	278.122	No	SELECT * FROM Users WHERE username = admin'mEtufU<'">zvVsRP AND password = pass ;
16	320	95.334	196.01	273.701	No	SELECT * FROM Users WHERE username = admin' UNION ALL SELECT NULL,NULL,NULL,NULL-- - AND p
17	340	112.593	195.6	299.985	No	SELECT * FROM Users WHERE username = admin) AND ROW(1154,5473)>(SELECT COUNT(*),CONCAT(
18	360	92.046	194.777	288.257	No	SELECT * FROM Users WHERE username = admin AND password = pass) PROCEDURE ANALYSE(EXTRA(
19	380	97.388	192.723	280.003	No	SELECT * FROM Users WHERE username = admin AND password = pass AND 2970=UTL_INADDR.GET_
20	400	97.8	191.49	51.0731631	Yes	SELECT * FROM Users WHERE username = admin,,)"")',), AND password = pass ;
21	420	89.171	191.079	46.6670853	Yes	SELECT * FROM Users WHERE username = admin AND password = 123 ;
22	440	87.937	189.846	46.3201753	Yes	SELECT * FROM Users WHERE username = admin AND password = pass' AND ROW(8726,1185)>(SELEC
23	460	90.403	189.435	47.7224378	Yes	SELECT * FROM Users WHERE username = admin AND password = pass);SELECT DBMS_PIPE.RECEIVE

Fig. 3: Time comparison of the three algorithms: AC-PT, AC and PT. The time measurements showed that before maturity AC-PT was slow as PT was the major component but after maturity the time greatly enhanced that it surpassed AC (73.1% of AC's time) due to the optimization done

same machine and the same tool generated attacks set were presented to all. On analyzing the results, many interesting features of AC-PT were made obvious. A recognized performance gain for AC-PT compared to each of the two mentioned algorithms, AC or PT alone was observed. The algorithm was run on around 400,000 different attacks with much improvement in accuracy over AC and achieving an accuracy equivalent to that of PT, this was shown in Table 1. From Table 2, it was obvious that the number of false positives made by AC-PT is 40% that of PT due to the modified space handling technique implemented in AC-PT.

Concerning time measurement, the experimental results showed that AC-PT surpassed both AC and PT algorithms in its time performance. This was the main addition of AC-PT over PT as they had the same accuracy as shown in Table 1. The AC pattern matching did not make any normalization or tokenization where comparison was done char by char. This was different from the AC module used in the proposed AC-PT approach where the attack patterns were saved in a normalized form and also the user inputs were converted to the same form.

The average time taken for PT was calculated over all experimented patterns to be 106.88 msce. The average time for AC over the same sample was 78.41 ms so AC-PT takes 53.63% of PT's time and 73.1% of AC's time which again proved a superior performance of the proposed approach.

As all the different approaches in the literature were run on some generated attacks set so it was not feasible to compare the accuracy and time measurements. Actually, some approaches did not mention an accurate value of their results so to assess the performance of AC-PT and point out the contributions provided, a methodology comparison was made with some recent counterparts. On analyzing the details of the proposed approach[13], the webpage scan was time consuming even when optimization was made, also the database error table was dependent on the database and its response to the queries which might sometimes not indicate an error. When considering[19], the predefined query specifications they used should be modified with the introduction of new patterns for intrusions. This contrasted with AC-PT that did not need special processing or application update and could change dynamically to incorporate any new type of attack. This also provided an advantage of AC-PT over[23] that maintained a list of known anomaly patterns and used a data mining tool for finding relevant patterns. GreenSQL was one of the tools assessed in Elia *et al.*[24], comparing its operation with AC-PT it was found that GreenSQL was limited to a certain database management system, MYSQL, this contrasted to AC-PT which could work for any database. Also GreenSQL kept a fixed white pattern list for comparison. It also made probability calculations to decide whether the pattern was malicious and used a risk scoring matrix which made it not flexible and did not incorporate the learning ability of AC-PT that could tolerate new malicious patterns and correspondingly offered better accuracy. Also concerning[1], AC-PT handled the false positive alarm problem in the PT part of this approach, also

AC-PT did not depend on a database to avoid time and resource consumption instead it utilized a text file to store the learnt patterns.

There is much scope in this study and the planned future work can include two directions; accomplishing the learning phase using neural networks. Also, the maturity threshold used in the experiments was settled upon by several trials to get the highest accuracy so genetic algorithms approach is suggested to be used to determine this threshold.

CONCLUSION

The SQLIA is the most severe security threat to the database in recent years. The proposed approach, AC-PT, made it possible to recognize when data was vulnerable. AC-PT was based on the static and dynamic algorithm but replaced the manual decision making part with PT validation algorithm. Taking the benefit of AC's high speed and PT's high accuracy while avoiding the false positive alerts resulting from PT, AC-PT surpassed them both in its overall performance.

SIGNIFICANCE STATEMENT

This study discovered a novel methodology that can be beneficial for securing the most valuable asset in any organization, which is data, against SQLIA. The study will help the researchers to uncover new paradigms in designing SQLIA detection algorithms as they have to concentrate on implementing a dynamic algorithm that is able to strictly react to and consequently combat completely new types of attacks in a feasible time according to the application.

REFERENCES

1. John, A., A. Agarwal and M. Bhardwaj, 2015. An adaptive algorithm to prevent SQL injection. Am. J. Networks Commun., 4: 12-15.
2. Venkatesan, K.G.S., R. Resmi and R. Remya, 2014. Anonymizing geographic routing for preserving location privacy using unlinkability and unobservability. Int. J. Adv. Res. Comput. Sci. Software Eng., 4: 523-528.
3. Venkatesan, K.G.S. and V. Khanaa, 2012. Inclusion of flow management for automatic and dynamic route discovery system by ARS. Int. J. Adv. Res. Comput. Sci. Software Eng., 2: 1-9.
4. Shahriar, H., S. North and W.C. Chen, 2013. Client-Side Detection of SQL Injection Attack. In: Advanced Information Systems Engineering Workshops, (Lecture Notes in Business Information Processing, Vol. 148), Franch, X. and P. Soffer (Eds.)., Springer, Berlin, Heidelberg, pp: 512-517.
5. Moosa, A., 2010. Artificial neural network based web application firewall for SQL injection. Int. Schol. Scient. Res. Innovat., 4: 610-619.
6. Makiou, A., Y. Begriche and A. Serhrouchni, 2015. Improving web application firewalls to detect advanced SQL injection attacks. Proceedings of the 10th IEEE International Conference on Information Assurance and Security (IAS), November, 2014, Okinawa, Japan, pp: 35-40.
7. Kumar, S., S. Dey, R. Karthikeyan and S. Venkatesan, 2015. Prevention of SQL injection attack on web applications. Int. J. Innov. Res. Comput. Commun. Eng., 3: 2313-2320.
8. Alwan, Z.S. and M.F. Younis, 2017. Detection and prevention of SQL Injection attack: A survey. Int. J. Comput. Sci. Mobile Comput., 6: 5-17.
9. Shin, Y., L. Williams and T. Xie, 2009. SQLunitgen: Test case generation for SQL injection detection. North Carolina State University, Raleigh Technical Report.
10. Wassermann, G., D. Yu, A. Chander, D. Dhurjati, H. Inamura and Z. Su, 2008. Dynamic test input generation for web applications. Proceedings of the 2008 International Symposium on Software Testing and Analysis, July 20-24, 2008, Seattle, WA., USA., pp: 249-259.
11. Ruse, M., T. Sarkar and S. Basu, 2010. Analysis & detection of SQL injection vulnerabilities via automatic test case generation of programs. Proceedings of the 10th IEEE/IPSJ International Symposium on Applications and the Internet, July 19-23, 2010, Seoul, Korea, pp: 31-37.
12. Bisht, P., T. Hinrichs, N. Skrupsky, R. Bobrowicz and V.N. Venkatakrishnan, 2010. NoTamper: Automatic blackbox detection of parameter tampering opportunities in web applications. Proceedings of the 17th ACM Conference on Computer and Communications Security, October 04-08, 2010, Chicago, Illinois, USA., pp: 607-618.
13. Roy, S., A.K. Singh and A.S. Sairam, 2011. Detecting and defeating SQL injection attacks. Int. J. Inform. Electron. Eng., 1: 38-46.
14. Wang, Y. and Z. Li, 2012. SQL injection detection via program tracing and machine learning. Proceedings of the International Conference on Internet and Distributed Computing Systems, November 2012, Springer, Berlin, Heidelberg, pp: 264-274.
15. Alazab, A. and A. Khresiat, 2016. New strategy for mitigating of SQL injection attack. Int. J. Comput. Applic., 154: 1-10.
16. Buehrer, G., B.W. Weide and P.A.G. Sivilotti, 2005. Using parse tree validation to prevent SQL injection attacks. Proceedings of the 5th International Workshop on Software Engineering and Middleware, September 5-6, 2005, Lisbon, Portugal, pp: 106-113.
17. 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.

18. Kar, D., S. Panigrahi and S. Sundararajan, 2016. SQLiGoT: Detecting SQL injection attacks using graph of tokens and SVM. Comput. Secur., 60: 206-225.
19. Kemalis, K. and T. Tzouramanis, 2008. SQL-IDS: A specification-based approach for SQL-injection detection. Proceedings of the 2008 ACM Symposium on Applied Computing, March 16-20, 2008, Fortaleza, Ceara, Brazil, pp: 2153-2158.
20. Zhang, K., C. Lin, S. Chen, Y. Hwang, H. Huang and F. Hsu, 2011. TransSQL: A translation and validation-based solution for SQL-injection attacks. Proceedings of the 1st IEEE International Conference on Robot, Vision and Signal Processing (RVSP), November 23, 2011, Kaohsiung, Taiwan, pp: 248-251.
21. Jang, Y.S. and J.Y. Choi, 2014. Detecting SQL injection attacks using query result size. Comput. Secur., 44: 104-118.
22. Kim, M.Y. and D.H. Lee, 2014. Data-mining based SQL injection attack detection using internal query trees. Expert Syst. Applic., 41: 5416-5430.
23. Jawanja, S., S. Shegokar, V. Nandurkar, R. Ardak, S. Chaudhari, S. Rithe and S. Sontake, 2018. An efficient technique for detection and prevention of SQL injection attack in cloud. Int. J. Res. Applied Sci. Eng. Technol., 6: 2670-2674.
24. Elia, I.A., J. Fonseca and M. Vieira, 2010. Comparing SQL injection detection tools using attack injection: An experimental study. Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering, November 1-4, 2010, San Jose, CA., USA.
25. Hiteshkumar, C., A.V. Nadargi, B. Narendra and S. Sushil, 2015. Doubleguard: Detecting intrusions in multi-tier web applications. Int. J. Adv. Res. Comput. Commun. Eng., 4: 473-476.
26. Balasundaram, I. and E. Ramaraj, 2012. An efficient technique for detection and prevention of SQL injection attack using ASCII based string matching. Procedia Eng., 30: 183-190.