Pertanika J. Sci. & Technol. 27 (1): 159 - 174 (2019)

PERTANIKA JOURNALS

SCIENCE & TECHNOLOGY

Journal homepage: http://www.pertanika.upm.edu.my/

SQL-Injection Vulnerabilities Resolving using *Valid* Security Tool in Cloud

Niharika Singh[1]* and Ashutosh Kumar Singh[2]

[1]*Department of Computer Engineering, National Institute of Technology Kurukshetra, Haryana, 136118 India*
[2]*Department of Computer Applications, National Institute of Technology Kurukshetra, Haryana, 136118 India*

ABSTRACT

The cloud is storing a huge amount of the data, including personal and confidential details. It involves the third party over the internet and proposes many unreliable strings which can be proven as loopholes. Thus, securing the data in the cloud tends to be a major point of concern. SQL Injection Attacks (SQLIAs) are being acknowledged as one of the foremost web applications security threats. It initiates a vulnerable query to destroy the connected server systems and help attackers with unauthorized access to the databases resulting in identity theft and security violations. The paper proposes a hybrid solution whose information utility is higher than the solutions that are being proposed earlier. As the methodology is concerned over static, dynamic and runtime detection and prevention mechanism. It also classifies the malicious queries and inspires the system to be well prepared for the secure working environment by implementing a demonstration design. Through the experimental implementation, the query associativity makes success probability of 0.775 using the associativity formula that in fraction, results in a durable comparative solution proposed till date.

Keywords: Cloud security, malicious nodes, SQL injection attack

INTRODUCTION

During the last decade, cloud computing and big data have appeared as two emerging technologies. These technologies have become one of the fieriest topics of recent times in the IT industry that made large steps in a relatively very short period of time. The methodology groundwork of cloud computing includes virtualizations and Service Oriented Architecture (SOA) of hardware and software. One mainstream of

ARTICLE INFO

Article history:
Received: 30 September 2017
Accepted: 28 August 2018
Published: 24 January 2019

E-mail addresses:
niharika.academics@gmail.com (Niharika Singh)
ashutosh@nitkkr.ac.in (Ashutosh Kumar Singh)
* Corresponding author

ISSN: 0128-7680
e-ISSN: 2231-8526

resource sharing promotes various cloud assistances (such as software cloud, application cloud, infrastructure cloud, business cloud, storage cloud). These offer various services for different domains (Casassa-Mont, et al., 2015). Involving the third party over the internet proposes many unreliable strings as loopholes (Swanson & Stinson, 2015). The cloud is storing a huge amount of data including personal and confidential details. Thus, securing the data in the cloud tends to be a major concern. On applying and detecting suitable methods for providing the privacy check to the insecure uncertainties itself is a major challenge of the cloud computing (Plischkea et al., 2013).

Web servers which provide customer services are usually connected to highly-sensitive information contained backend databases. Sometimes, these backend databases are vulnerable to the harmful attacks, such as SQL Injection Attacks (SQLIAs). SQLIA are one of the foremost security threats to web services and applications (Eyal, Birman, & van Renesse, 2015). It initiates a vulnerable query to destroy the connected server systems and gives attackers unauthorized access to databases. It provides right to delete, modify and retrieve valuable and confidential information stored in databases. This results in identity theft and security violations (Dharam & Shiva, 2013). SQLIAs come in the picture when the data provided by the external users are directly included in SQL query but are not properly validated. According to a study, it was stated that 75% of the cyber-attacks are outperformed at the application layer. Also, over the audited websites where 97% of them are clearly targeted (Eyal et al., 2015; Narayanan et al., 2011; Plischkea et al., 2013). Thus, for the SQLIAs an inadequate input validation has been identified as one of the major causes within a web application. For the first level of defense against SQLIAs, these input validations can serve satisfactorily, but it may not be much defence against injecting SQL queries attack techniques.

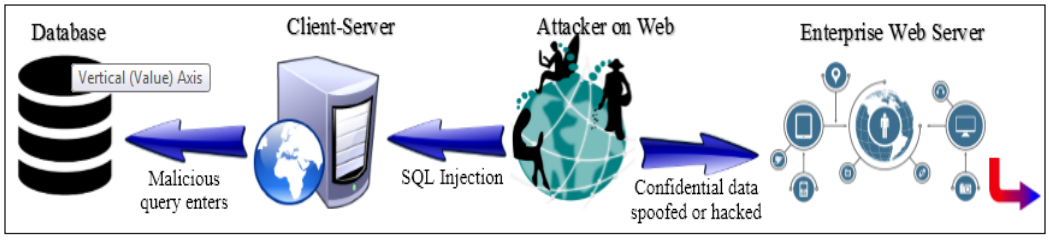

Figure 1. Representation of how attacker initiates SQL Injection Attack

A variety of tools are introduced as a solution for SQLIAs keyword based filtering, machine learning and decision trees. These solutions for the firewalls and Intrusion Detection Systems (IDSs) are ineffective against SQLIAs because some ports are open in firewalls for the regular web traffic at the application level that are used to perform SQLIAs. Thus, if attackers get breakthrough (Kiani et al., 2008) as shown in Figure 1 and find loopholes to attack, it will result in damaging the firewall and IDS.

The paper considers the SQL injection attack, which is its own kind of code injection attack. Though, the SQLIA is very common, there must be some privacy measures to protect our databases from SQL Injection Attacks. Day by day, our databases are getting modular, so is the Injection attacks, but proportionally. Thus, SQLIA is a considerable concern in the real world. Hence, research on this problem bears significant practical importance. In concern, we have developed the detection and prevention method that tries to achieve the high Information Utility Rate (IUR) using the query associativity and tautology logics. Where, 'A tautology is a statement that is always *True* regardless of the truth values of the individual statements'. This is considered as the base idea for the proposed approach and is processed with "query associativity". The methodology is dogged over static, dynamic and runtime detection and prevention mechanism. This also filters out the malicious queries and inspires the system to be well prepared for the secure working environment (Ping et al., 2016; Halfond & Orso, 2005; Narayanan et al., 2011).

Related Work

Today, for web applications SQL injection attacks are counted among the top-most threats. These attacks are launched through the specially crafted user inputs over web applications which uses low-level string operations in order to construct SQL queries. A research study can be outperformed on the basis of the work previously done in the subsequent manner.

A model-based tool AMNESIA was proposed by Halfond and colleague (2005) to detect illegal queries (before the execution on the database). This includes detection on both static as well as dynamic injected query attacks. In the static part, to automatically build a model of legitimate queries the technique uses program analysis. On the other hand, dynamic part runtime monitoring is used to inspect the dynamically generated queries (Narayanan et al., 2011; Halfond & Orso, 2005). Bandhakavi, Bisht, Madhusudan, & Venkatakrishnan (2007) introduced another model CANDID that detects SQL injection to dynamically mine programmer-intended query structure on any input and compare it against the structure of actual query issued. The approach is based upon inferring intended queries considering symbolic query computed on a program run. A new detecting method was proposed based on single character to detect parsing and black list based attacks which are experimentally defined by using both attacks and normal samples (Sonoda, Matsuda, Koizumi, & Hirasawa, 2011) in different types of SQLIAs that are extensively reviewed till date. It analyzes various recently developed defensive mechanisms also shows how each technique might help in preventing and detecting all SQLIA types. It introduces PSIAQOP (Preventing SQL Injection Attacks based on Query Optimization Process) depending on heuristic rules to prevent all SQLIAs types (Al-Khashab, Al-Anzi, & Salman, 2011). In order to overcome the weakness a detection method is proposed that uses machine learning and probabilistic study. It gives the formula to calculate parameter of the zeta distribution.

The model proposes an SQL injection attack detection methodology using the proposed formula (Oosawa & Matsuda, 2014).

Some other well-known interrelated web application vulnerabilities of SQL attacks are Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). By getting inspired to detect such vulnerable queries a method was proposed which detects and recognize SQL injection that are based on defined and identified criteria. The model is able to generate report regarding the vulnerability level and decrement in possibility of SQL injection attack onto the web application (Buja et al., 2014). To detect dynamic injected queries a new approach was suggested that detect and prevents SQL injection attacks on checking whether user inputs cause changes in query intended results. Attacker if uses space, single quotes or double dashes in input the proposed method detects and tokenizes the query and a query with injection separately. When tokens are formed, it stores onto an array representing every token as an element of an array. Two arrays are resulted from both queries (original and injected) then the length is compared to detect if it is injected or not. Further the access is granted or denied on the basis of comparison (Ntagwabira & Kang, 2010). Ping et al., (2016) proposed a method of preventing SQL injection attacks by ISR (Instruction Set Randomization), and built a prototype system based on this strategy. The prototype system randomizes the SQL keywords in the application, because the SQL statement injected by the attacker is not randomized, so the SQL injection can be easily detected. Also, the experimental results show that this system has a good effect on preventing SQL injection and low running cost. Similarly, Yassin et al. (2017) proposed the detection solution for Software as a Service (SaaS) providers. To achieve SQL query/HTTP request mapping, they planned an event-correlation based on the similarity between literals in SQL queries and parameters in HTTP requests. Many researchers worked in the field, but still an appropriate satisfactory solution was not found which could be announced as best to defeat the SQL injection attacks including (Yassin et al., 2017; Uwagbole et al., 2017). Hence, the comparison summary of work done in SQLIA solution techniques is shown in Table 1.

Problem Statement

The main objective of the research is connected to Table-1 where, we focus to get better information utility rate. It is said that 'the higher the Information Utility Rate, the higher will be the Privacy' but, this decreases the data accessibility. This limitation of higher IUR leads to examine models to get a "balance between Data accessibility, Data Privacy and Data Utility". Hence, our objective is based upon getting a Moderate/High information utility than the other research solutions. Thus, the results lead to achieve an IUR percentage range that would lie between 80-90% that is considered to be a balanced range in all technical aspects. In association, we proposed a hybrid solution working to get high information utility than the other research solutions. The solution works over Prevention and Detection

Table 1
Summarized View of State-of-art work done in SQLIA solution techniques

Information utility	***Privacy access***	***Approach***	***Advantage***	***Disadvantage***	***Complexity***
High	Pattern matching based privacy using authenticated data structures	Policy-oriented over low privacy strength	Reduction in communication overhead providing ease during the implementation and computations.	Lacks in diversity.	Low
Low	Prevention through parsing and triggering queries for privacy policy with defined data accessibility level	Query optimization over strong privacy strengths	Providing a record-level SQLIAs protection to prevent eves-dropping across multiple data environment.	Comparison results are less satisfactory.	High
Moderate	Filtered and linkable data privacy with specific data accessibility	Linkable policy with limited information access using Heuristic rules and output filtering	Provides required malicious and benign URLs for three phases of testing, validating, and training. Addresses issues of policy anomalies mechanisms, also reduces overhead computation.	Vulnerable to unauthorized access as only detection method is introduced	High

to improve firewall security. The other researchers encountered various solutions to get approximate results for a secure SQLIA free environment. Early experiments have a success score range of 50-70% (Plischkea et al., 2013; Eyal et al., 2015; Narayanan et al., 2011). The proposed logics were categorized into as either detection method or prevention methods but the combinations were hardly proposed. Hence, to achieve 100% success rate solution for SQLIA security is quite difficult. A few researchers have tried to achieve a satisfying success rate, but is not much convincing due to the quality attributes (Dharam et al, 2013). Hence, a model is required that would result with 80-90% success rate range of security bar. Also, a solution that would come up with all proximities and could be analyzed in proposed methodology. Thus, we strictly compare the work with Casassa-Mont et al., (2015) and Ntagwabira & Kang (2010) improving the loopholes using tautology logics. Therefore, the experimental results are reasonable enough to validate all those claims that are suggested through the proposed work.

Proposed Solution

Client-Side Attacking Shot. The introduced approach is a three-tier (Client-Logic Access- Data Server) runtime detection and prevention methodology. It deals with process organization, accessibility and exchange of queries. It ensures that the Data-Server tier would not execute any vulnerable code that affects the hosted operating systems and devices partially or completely. The technique works over the database server side being associated with a distributed cloud environment. It provides a security controlling system ensuring secure execution of all the requested queries without any database hacking or fabrication.

Procedure
Receive_Query Unveil_Message
(T:Tier level number)

Begin
Update *access_table* row T to increase *input_count;*

End

Procedure
Finish_Query (T: Tier level number)

Begin
Update *access_ table* row T to increase *consumed_count;*

End

Procedure *Upon_Idle*

Begin
Report to *controller_server* non-zero difference for previously unreported *access_table rows;*

End

This algorithm for tier-architecture detects the completion of the query exchange process at n-tier level. As the queries Q = $\{q_1, q_2, q_3 \ldots q_s\}$ $Q = \{q_1, q_2, q_3 \ldots q_s\}$ go through a tier architecture representation for T = $\{t_1, t_2, t_3 \ldots t_n\}$. For the proposed scenario, it works over up to n=3 levels. The architecture is dependent upon the three-tier architecture system which is divided as follows:

First tier (client tier) **-** This tier consists of applications that are accessed through a centralized system. Here, it is concerned over web browsers, servers or standalone application running on different machines that processes queries to request and response through the central server. If there are S servers that share a communication through Q queries, the ratio of detecting a breakthrough would be directly proportional to R number of activities where R = $\{r_1, r_2, r_3 \ldots r_t\}$. Now, on the whole the query associativity would be:

$$\boldsymbol{Q_i} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{t}} (r_1 + r_2 + r_3 \ldots r_t) \quad (1)$$

Here, the R outperforms s number of queries. Thus,

$$\boldsymbol{Q_i} = (q_1, q_2, q_3 \ldots q_s)_1 + (q_1, q_2, q_3 \ldots q_s)_2 + \cdots + (q_1, q_2, q_3 \ldots q_s)_t \quad (2)$$

$$\boldsymbol{Q_i} = \boldsymbol{t}(q_1, q_2, q_3 \ldots q_s) \quad (3)$$

$$\boldsymbol{Q_i} = tQ \quad (4)$$

For which if we have i = 1, Q ≅ t (5)

The queries when are processed through distributed servers produce results into HTML web pages. The web pages are uniquely identified with their corresponding *url*. To find the associative probability it is further divided by 100 for the overall evaluation.

Second tier (logic access tier) **–** The layer concerns over the server codes that may include platform or software applications. These codes process and set up communicational behavior among local and remote system servers outperforming over network languages. Thus, the layer is responsible for the following measures: authentication, authorization, caching, exception management, validation. These measures effectively logs and audit the progressive *Q* queries.

Third tier (data server tier) –This layer embraces all the database objects that might be used by applications (such as schemas, views, tables and stored procedures). It considers database services over distinct servers. Data server tier stores SQL server object definitions of available instant-level objects stored in database. The layer tools can be listed as: Application Developer, Database Administrator, Independent Software Vendor, IT Administrator, etc. supporting the operations: *Extract, Deploy, Register, Unregister, Upgrade* that helps in *Export_Import* of the request–response queries.

The 3-Tier architecture provides a support to the "*Valid* security" scheme which helps reducing SQL vulnerability rate in websites over the client side.

Security Policy Framework. Malicious SQLIA can be introduced into vulnerable applications using different input mechanisms. To get rid of such exasperating threats to the database-driven applications "*Valid* Security" can be introduced. During the process, a conditional query statement is fixed over the firewall on the client side it results to be always *True*. It bounds the parameters to some defined SQL structure, thereafter, for the attacker it is not possible to inject additional SQL code.

It is a runtime monitoring and prevention strategy which is more complex than defensive coding schemes. For the tautology logic encapsulation Proxy Server is introduced between SQL Server and Web Server. Attacker attacks using the SQL query with randomized value to proxy server, received by the client. Then, it is accessed through "*Valid* security" that stores an activity-access table over the proxy server to de-randomize the query. Further, it sends the satisfactorily secure and filtered query to the server for the processing then downloads HTML page or the website content, see Figure 2.

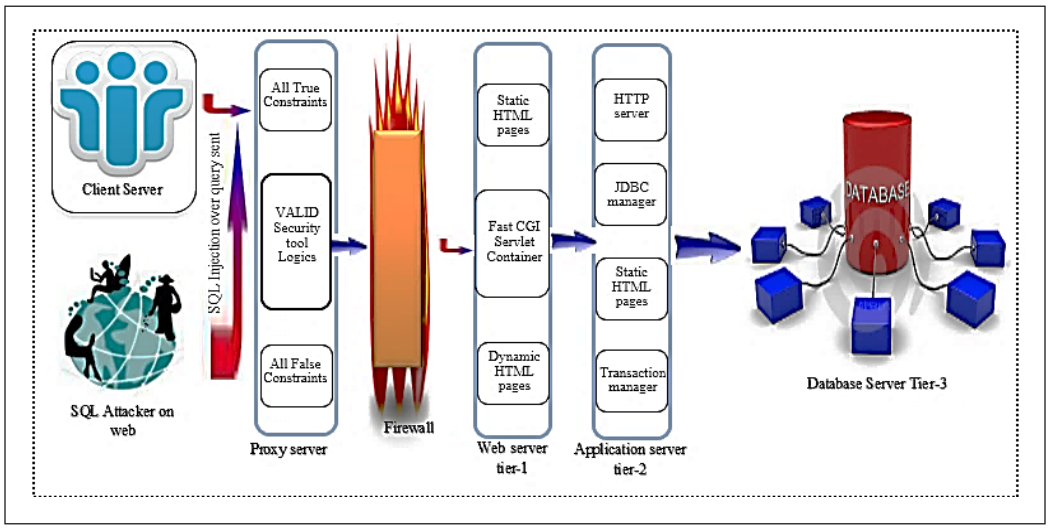

Figure 2. Working of proposed *Valid* Security Tool SQLIA solution

Take an example of spoofed SQL query that contains WHERE clause in it to consider the query processing as a tautology on the attacker's end or universally. The two clause elimination procedures, say V_1 and V_2 which are recommended to be *Valid* and *Unsatisfiable* respectively, encounters with a vulnerable SQL query Q. Influentially both the procedures are equally effective. In conjunction, when Q processes through the proxy server, it passes through either V_1 or through V_2 that in results of give any false for V_1 and any single true for V_2 it would filter out the query and remove the vulnerability attached using some parameterized queries. There it is followed as *Valid* constraints are updated on the basis of the proposed machine learning whereas *Unsatisfiability* constraints are manually updated. On passing through it pairs up as if:

$V_1(Q)$ and $V_2(Q)$ $V_2(Q)$ (6)

For which we have queries to pass through the filter if V_3 is the all formula clause $\forall$, i.e.

$V_1(Q) \subset V_2(Q) \subseteq \forall V_3(Q)$ $V_1(Q) \subset V_2(Q) \subseteq \forall V_3(Q)$ (7)

But, here V_2 would be equally effective only if it satisfies twice over the V_1 reordering in concern to the propositional logic:

$$\neg\neg Q \equiv Q \quad (8)$$

Where, Q is the SQL query represented in propositional logic. Thus, on the basis it is easy to remove the vulnerable or malicious queries on the runtime that neutralizes SQL injection attacks. Here the dynamic part of it inspects dynamically generated queries and checks them using some predefined rules. For the mapping concern of the inputs and vulnerabilities machine learning methodology is introduced to generalize the dynamic generation of the queries. The system learns the query activities with the help of cookies C that get attached to the browser and stores activity in the Activity-Access table *A* (with *i* number of entries) put away over proxy server. For which it would be satisfiable:

$$C \in A_i \quad (9)$$

When it learns from the frequent access it starts filtering the dynamically injected queries as well and extracts the user inputs.

Progressive Algorithm

To understand the detailed concept of the solution, a step-to-step process is explained. To implement the methodology, one needs to go through the main process that initially calls to setup a PROXY_SERVER then after validation CLOUD_SERVER_HANDSHAKE is configured. It proceeds to install VALID_SECURITY_TOOL (proposed method installation over proxy). It includes an ACTIVITY_ACCESS_TABLE putting in place to record all true and all false activities for corresponding queries in Algorithm 1.

Algorithm-1: Query Access and Denial System for SQL injection attack.

Begin
Initialize Q={q_1,q_2,q_3…q_s}
Install and Process VALID_SECURITY_TOOL()
Create ACTIVITY_ACCESS_TABLE()
Initialize matrix arr[i][j] and Set arr[0:i-1][0:j-1];
Apply STORE_CONSTAINT (where, i=MANUALLY_STORE_CONSTRAINT() and j=MACHINE_LEARNING_CONSTRAINT)
If TRUE → Q_s (i,j) Const=1

Else FALSE → Q_s (i,j) Const=0.
Call query Q_s () and Break STORE_CONSTRAINT to generate INPUT_QUEUE(Q)
Match INPUT_QUEUE()↔STORE_CONSTRAINT()
If (Match= all TRUE), then (ACCESS_QUERY →ALLOW)
Else if (Match= all FALSE), then
Repeat for TRUE and Match= all TRUE;
Set Access query →ALLOW;
Else (Access query →DENY)
Store access results →ACTIVITY_ACCESS_TABLE()
Report PROXY_SERVER() ABOUT *UPDATE*
Call THREE_TIER()
Process through CLIENT_TIER() and Calculate QUERY_ASSOCIATIVITY
Access P_i through LOGIC_ACCESS_TIER;
Store content over DATA_SERVER_TIER
Finish Query → Q_s () and Set S=s+1
Repeat for *Q*;
End

RESULTS AND ANALYSIS

Implementation Environment

The proposed architecture demands a powerful and effective justification environment for which a technical platform has been performed. Performance section is divided into two phases depicting as setup phase and experimental phase giving a description over hardware and software components.

System Setup

Initiating over a supercomputer sometimes is a difficult task, but here an archetype is to be designed for execution of queries and transactions for carrying up over inter and intra-cloud. Thus, Table 2 shows system configuration scenario instigating technical attributes like RAM, OS, Hard-disk, etc. required for the implementation of the proposed algorithm.

Experimental Setup

To set up the cloud, there is a need to arrange a client-server picture. It can be configured as master-slave server maintenance using open source platform**.** In concern to our proposed scheme justification we are using a latest configured machine which may satisfy the data owner, needs for preserving the sensitiveness following the *Valid* Security Tool whose configuration is defined in Table 3. For the further proceeding and understanding Table 4 is designed with the detailed description for experimental technical configurations, including the cluster server worked over different ports.

Table 2
Technical details of implementation environment

Setup phase	***Technical attributes***	***Configuration***
System setup (minimum requirements)	RAM Capacity	8 GB
	Processor	Intel(R) Core(TM) i7 CPU Q 740 @ 1.73GHz 1.73GHz Turbo up to 1.93 GHz
	Operating system	Windows 7 ultimate
	Hard-disk	1 TB
	Graphic card (if required)	NVIDIA GeForce GT 425M-2GB

Table 3
Experimental technical configuration details of implementation setup phase

Setup phase	***Technical attributes***	***Configuration***
Experimental Setup	Query size	30
	Valid Constraints (dynamic)	15 initially
	Unsatisfiable Constraints (static)	15 initially
	Target number of Nodes	3
	Database	MySQL
	Platform	Open Jave Development Kit JVM
	Infrastructure	Open Source Server-Oriented

Table 4
Server oriented details

Setup phase	***Source***	***Server type***	***Configuration***
Validation Setup	Server-1 (Client)	Master Server	WAMP SERVER, Open Source Platform
		Apache Server	80 PORT
		MySQL Server	3306 PORT
		Proxy Server	8080 PORT
	Server-2 (Attacker)	Master Server	WAMP SERVER, Open Source Platform
		Apache Server	8081 and 8180 PORT
		MySQL Server	3309 PORT
	Server-3 (Cloud Node-1)	Slave Server	XAMPP SERVER-1, Open Source
		Apache Server	9090 and 443 PORT
		MySQL Server	3307 PORT
		Language	Python
	Server-4 (Cloud Node-2)	Slave Server	XAMPP SERVER-2, Open Source
		Apache Server	8080 and 8181 PORT
		MySQL Server	3308 PORT
		Language	Python

Evaluation Scenario

Here, in this section our scheme is analyzed on the basis of the experimental computation and performance. We are evaluating some thorough perceptible results to justify the formulation with examples. We first assess the computation and communication overhead, and further give the details about the data spoofing through SQLIAs.

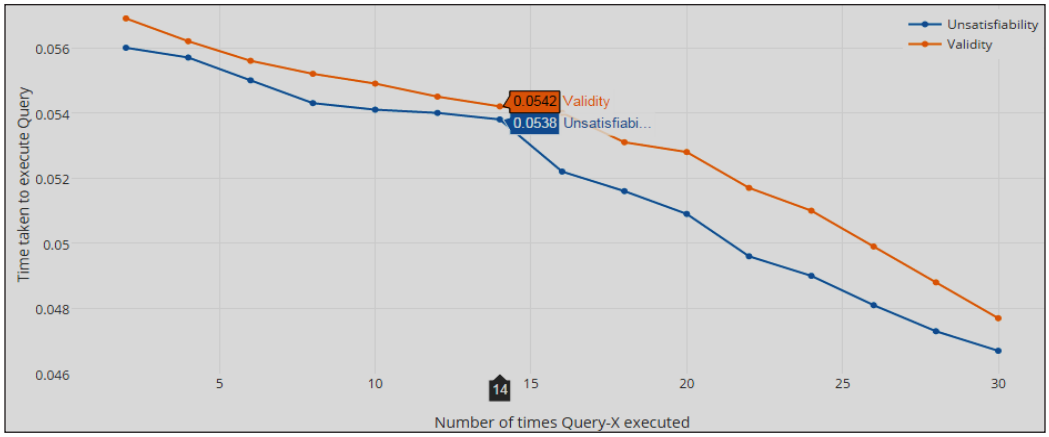

Figure 3. Minimum value calculated when *i* no *Valid* and *Unsatisfiable* query runs over

Figure 4. Maximum value calculated when *i* no. of *Valid* and *Unsatisfiable* query runs over

For the performance validation of the *Valid* Security tool set of thirty queries has been analyzed that includes original and injected queries. Taking two queries initially, i.e. n=2, including WHERE clause results in Figure 3 and Figure 4 that shows *Validity* and *Unsatisfiability* values. The Figures 3 and 4 show the minimum and maximum time

gaps, respectively, taken when run for about thirty (30) times, respectively. The lines are contracted somewhere with small gaps, but in some cases, it is wide which is all due to the learning process installed over ACCESS_ACTIVITY_TABLE.

On processing *n*=30 the results are shown in Figure 5. It depicts that it makes viable negotiated inferences during the time stamping, when these queries are analyzed in a queue. Minimum difference evaluated is measured to have 0.00003 seconds. In fraction, it is a very small difference when is outperformed on a single machine to get the utility. Calculating the query associativity makes success probability of 0.775 (from Equation 1-5) using the associativity formula. Here, we find the Information Utility Rate (IUR) in terms of probability and resulted in 0.775. This indicates the percentage of proposed approach in comparison to other proposed models to get 80-90% success rate range of security bar. This IUR rate is comparatively higher than references considered in related work. Also, to our competitive models i.e. Casassa-Mont et al., (2015) and Ntagwabira and Kang (2010). Where, Casassa-Mont and colleagues in 2015 attained 0.74 IUR and Ntagwabira and Kang (2010) achieved 0.65. We have considered one classic and one latest model for comparison. Figures 5 and 6 are comparative analysis in comparison to model proposed by Casassa et al., (2015), and Ntagwabira and Kang (2010). The work presented in these models are implemented in same platform. Thus, experimental results provide average negotiation comparison and query cycle satisfiability and un-satisfiability rate, see figure 5 and 6. In Figure 6, contracting lines are representing the smallest average by considering four different queries in each model. It has a very small difference of negotiation. A complete cycle includes the static and dynamic variability and the process that leads to filtration after the detection of injected SQL queries. Thus, this experimental setup justifies its work by producing a balance in-between.

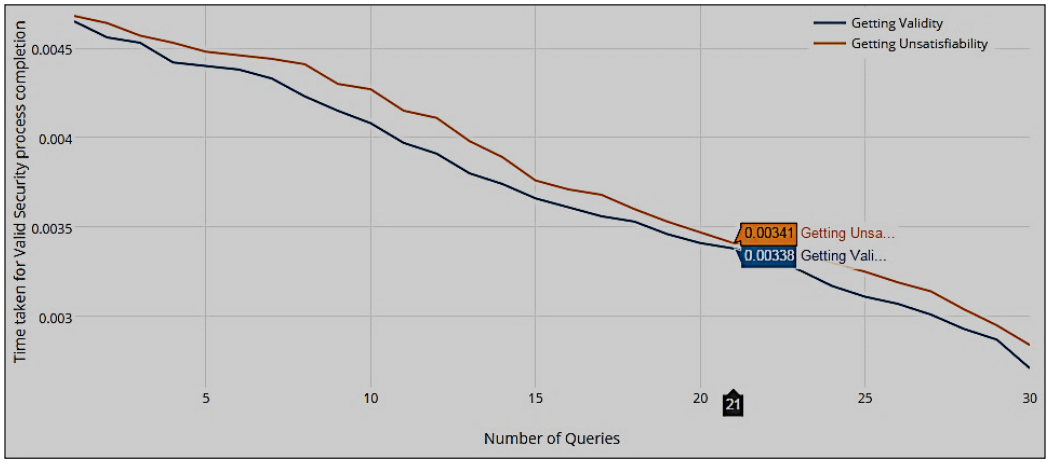

Figure 5. Representation of time taken to complete the cycle by *n*=30 queries

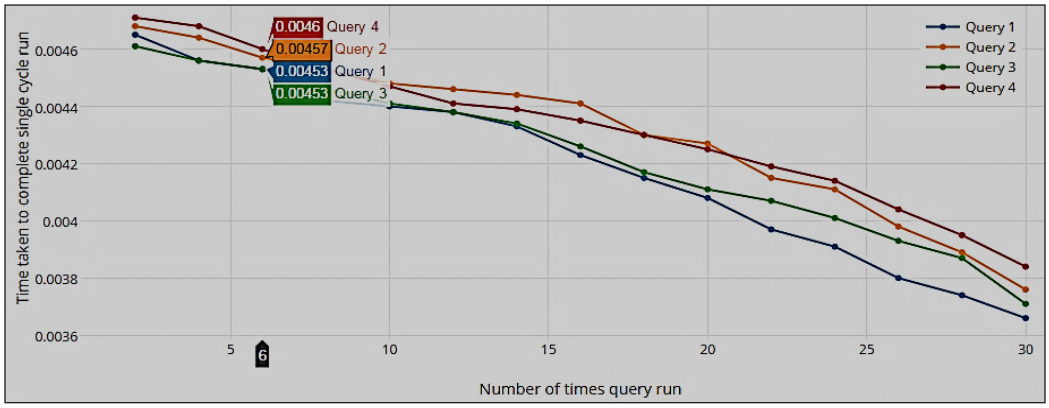

Figure 6. Average negotiation comparison for 4 random queries

CONCLUSION

SQLIAs inadequate input validation has been identified as one of the major causes within a web application. According to a study, it was stated that 75% of cyber-attacks are outperformed at the application layer and over the audited websites where 97% of them are clearly targeted. In the paper, the computation experimental performance of the proposed *Valid* Security tool solution results to justify the formulation with 70-80% success in securing the dataset from 'SQL injections' injected by attacker belongs to outside the cloud. We focus to get better information utility rate to get a Moderate/High information utility than the other research solutions. The solution works over Prevention and Detection to improve firewall security. Also, the results lead to achieve an IUR percentage range that would lie between 80-90% that is considered to be a balanced range in all technical aspects.

REFERENCES

Al-Khashab, E., Al-Anzi, F. S., & Salman, A. A. (2011, April). PSIAQOP: preventing SQL injection attacks based on query optimization process. In *Proceedings of the Second Kuwait Conference on e-Services and e-Systems* (p. 10). Kuwait City, Kuwait. doi:10.1145/2107556.2107566

Bandhakavi, S., Bisht, P., Madhusudan, P., & Venkatakrishnan, V. N. (2007, October). CANDID: preventing sql injection attacks using dynamic candidate evaluations. In *Proceedings of the 14th ACM conference on Computer and communications security* (pp. 12-24). Alexandria, Virginia, USA. doi:10.1145/1315245.1315249

Buja, G., Jalil, K. B. A., Ali, F. B. H. M., & Rahman, T. F. A. (2014, April). Detection model for SQL injection attack: An approach for preventing a web application from the SQL injection attack. In *2014 IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE)* (pp. 60-64). Penang, Malaysia.

Casassa-Mont, M., Matteucci, I., Petrocchi, M., & Sbodio, M. L. (2015). Towards safer information sharing in the cloud. *International Journal of Information Security, 14*(4), 319-334. doi:10.1007/s10207-014-0258-5

Dharam, R., & Shiva, S. G. (2013, April). Runtime monitors to detect and prevent union query based SQL injection attacks. In *2013 Tenth International Conference on Information Technology: New Generations (ITNG)* (pp. 357-362). Las Vegas, NV, USA. doi:10.1109/ITNG.2013.57

Eyal, I., Birman, K., & van Renesse, R. (2015, June). Cache serializability: Reducing inconsistency in edge transactions. In *2015 IEEE 35th International Conference on Distributed Computing Systems (ICDCS)* (pp. 686-695). Columbus, OH, USA. doi:10.1109/ICDCS.2015.75

Halfond, W. G., & Orso, A. (2005, May). Combining static analysis and runtime monitoring to counter SQL-injection attacks. In *ACM SIGSOFT software engineering notes* (Vol. 30, No. 4, pp. 1-7). St. Louis, Missouri. doi:10.1145/1083246.1083250

Kiani, M., Clark, A., & Mohay, G. (2008, March). Evaluation of anomaly based character distribution models in the detection of SQL injection attacks. In *Third International Conference on Availability, Reliability and Security (ARES 08)* (pp. 47-55). Barcelona, Spain. doi:10.1109/ARES.2008.123

Ntagwabira, L., & Kang, S. L. (2010, July). Use of Query Tokenization to detect and prevent SQL Injection Attacks. In *2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)* (Vol. 2, pp. 438-440). Chengdu, China. doi:10.1109/ICCSIT.2010.5565202

Narayanan, S. N., Pais, A. R., & Mohandas, R. (2011, August). Detection and prevention of sql injection attacks using semantic equivalence. In *5th International Conference on Information Processing (ICIP 2011)* (pp. 103-112). Bangalore, India.

Oosawa, T., & Matsuda, T. (2014, October). SQL injection attack detection method using the approximation function of zeta distribution. In *2014 IEEE International Conference on Systems, Man and Cybernetics (SMC)* (pp. 819-824). San Diego, CA, USA. doi:10.1109/SMC.2014.6974012

Ping, C., Jinshuang, W., Lin, P., & Han, Y. (2016, October). Research and implementation of SQL injection prevention method based on ISR. In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)* (pp. 1153-1156). Chengdu, China. doi:10.1109/CompComm.2016.7924885

Privacy Technical Assistance Center. (2013). *Data De-identification: An Overview of Basic Terms.* Retrieved November 15, 2016, from https://studentprivacy.ed.gov/sites/default/files/resource_document/file/data_deidentification_terms_0.pdf

Plischke, E., Borgonovo, E., & Smith, C. L. (2013). Global sensitivity measures from given data. *European Journal of Operational Research, 226*(3), 536-550. doi:10.1016/j.ejor.2012.11.047

Swanson, C. M., & Stinson, D. R. (2015). Extended results on privacy against coalitions of users in user-private information retrieval protocols. *Cryptography and Communications, 7*(4), 415-437. doi:10.1007/s12095-015-0125-x

Soria-Comas, J., Domingo-Ferrer, J., Sánchez, D., & Martínez, S. (2014). Enhancing data utility in differential privacy via microaggregation-based kk-anonymity. *The VLDB Journal—The International Journal on Very Large Data Bases, 23*(5), 771-794. doi:10.1007/s00778-014-0351-4

Sonoda, M., Matsuda, T., Koizumi, D., & Hirasawa, S. (2011, November). On automatic detection of SQL injection attacks by the feature extraction of the single character. In *Proceedings of the 4th international conference on Security of information and networks* (pp. 81-86). Sydney, Australia. doi:10.1145/2070425.2070440

Uwagbole, S. O., Buchanan, W. J., & Fan, L. (2017, May). Applied machine learning predictive analytics to SQL injection attack detection and prevention. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)* (pp. 1087-1090). Lisbon, Portugal. doi:10.23919/INM.2017.7987433

Yassin, M., Ould-Slimane, H., Talhi, C., & Boucheneb, H. (2017, June). SQLIIDaaS: A SQL injection intrusion detection framework as a service for SaaS providers. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)* (pp. 163-170). New York, USA. doi:10.1109/CSCloud.2017.27