Classification of SQL Injection Attacks Using Fuzzy Tainting

Surya Khanna and A.K. Verma

Abstract The embellishment of the Internet has escalated the need to resolve cyber security issues. SQL injection attacks (SQLIAs) being one of the oldest yet tenacious attacks captivating the Web applications pose a serious threat. Various techniques are introduced over the years to tackle this problem, but there are times when it becomes difficult to meticulously define the fine line separating a valid input from a malicious one. This work proposes a SQL injection (SQLI) threat level indicator based on fuzzy logic for handling SQL injection attacks. The fuzzy tainting approach helped in ruling out the possibilities of false positives.

Keywords SQL injection · Fuzzy logic · Tainting · Web security · Threat level indicator

1 Introduction

The past decade has seen a rapid shift of products and services to the world of Internet. Whether it is the commercial sector or financial, even the government is going online. All this hype of being a part of the online world has led these services to become the focal point of attacks. Hence, the need of the moment is protect the user's data from maleficent attacker(s).

SQL injection is among the top rankers in the threat rating lists as analyzed by various organizations like OWASP [1], MITRE [2]. An average number of 6,800 SQL injection attacks (SQLIAs) per hour were observed in Imperva's Web application attack report edition #6 [3]. Even alarming observation was an increase of

S. Khanna (✉) · A.K. Verma
Computer Science & Engineering, Thapar University, Patiala, Punjab, India
e-mail: surya.khanna88@gmail.com

A.K. Verma
e-mail: akverma@thapar.edu

P.K. Sa et al. (eds.), *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications*, Advances in Intelligent Systems and Computing 518,
DOI 10.1007/978-981-10-3373-5_46

29.63% in SQLIA than the previous year. Further, Verizon's 2015 Data Breach Investigations Report (DBIR) enlists 2,122 confirmed data breaches out of 70 organizations [4]. Information is the most valuable asset to the organizations. So, the databases should be hardened against injection attacks.

2 Motivation

The first point of initiating this work was to legalize valid inputs containing keywords usually considered insecure. Hence, the keywords alone cannot be relied upon. They give us results that are partially true. Now, fuzzy logic is a technology that can handle partial truth values. If that be combined with the known attacks, we can resolve our problem. Work has been done using fuzzy logic to assess the security risks [5] using various density functions but validation of such inputs has not been pondered upon.

3 SQL Injection Attacks

This section provides a brief background on SQL injection. SQLIAs can be broadly categorized as:

- First Order/Direct Attacks
  - Through user input
  - Through cookies
  - Through server variables
- Second Order/Indirect Attacks, i.e., the malicious input is inserted at a place different from where the attack is intended to be performed.

4 Related Work

This section acknowledges the work of various researchers who contributed to provide solution for SQLIA.

In 2003, Huang et al. [6] proposed a black box approach to assess the security of Web application using fault injection and behavior monitoring. It uses Web crawler to determine SQLIA target points of a Web application. A knowledge expansion model is used to learn the behavior of malicious pattern and determine high confidence terms. Lastly, the negative response extraction (NRE) algorithm is used to determine the impact of input on the results. Later on, Stephen W. Boyd and Angelos D. Keromytis 2004 [7] used a secret key to randomize the SQL queries

with the help of a proxy. The security of this technique depends on the strength of the secret key.

In 2005, William G.J. Halfond and Alessandro Orso [8] built SQL query models combining static as well as dynamic monitoring techniques to detect injection. The SQL query model is a non-deterministic finite-state automaton which checks the dynamically generated queries to figure out any violation. Further, X. Fu et al. 2007 [9] proposed a white box model for static analysis of byte code to determine vulnerabilities at compile time. A hybrid constraint resolver following string analysis approach is used to decide security breaches. Further, in 2010, Bisht et al. brought forward CANDID [10], a query structure mining approach to determine the deviation of queries formed by candidate input from programmer-intended queries.

In 2013, A.S. Gadgikar [11] came forward with a negative tainting approach to detect SQLIA. In this approach, all entry points are checked for known attack keywords. Recently, in 2015, B. Hanmanthu and colleagues [12] proposed data mining technique using decision trees to classify attack signatures, thereby preventing SQL injection. Decision making is based on associative classification rules.

5 Methodology

We propose a fuzzy logic-based tainting solution to determine SQLI vulnerabilities. In this technique, we initially use negative tainting [11] to determine any malicious string in input parameters and attempt to determine their attack types according to which we can associate a severity level with them as shown in Table 1. The risk level is based on the hindrance; it can cause in the smooth working of a system.

Table 1 Severity level of different types of SQLIAs

Attack type	Example	Risk level
Tautology	' or a = a –	Medium
Logically incorrect queries	Pass character in integer data type, e.g., pin = '#123\' or pin = convert (int, (select top 1 name from sysobjects where xtype = 'u'))	Low
Union query	' UNION SELECT accNo from users where uid = 1349 –	Medium
Piggybacked queries	(I)'; insert into users values (666, 'attacker', 'admin', 0xffff) — (II)'; Drop table accounts;–	High
Stored procedures	Pass following as parameters (I)'; SHUTDOWN; – (II)'; Drop table users;–	Medium, high
Alternate encoding	'; exec (char(0x73687574646f776e)) –	Medium, high
Inference (I) Blind injection (II) Timing attacks	(I) john' and 1 = 0 – john' and 1 = 1– (II) id = 1') or sleep(25) = 0 limit 1–	Low

Scores are calculated from the taints found against the known attacks for all input parameters of a query. These scores are used to indicate the risk level of the malicious input strings. Attackers use logically incorrect queries or inference techniques to determine information about the database though error messages. Not much information can be retrieved through these methods and is usually considered as a part of database fingerprinting. So, no sensitive data breach happens; therefore, they are enlisted to pose a low-level threat, whereas attacks like tautology, union, stored procedure may lead to breach of confidentiality and hence given a relatively higher-level threat indication.

Then, we look over the programmer-intended query corresponding to the input parameter and determine the probability with which our system will be affected if any malicious input is given to the query. Malicious content in a DDL poses highest level of threat. Any modification at the schema level can lead to huge loss/leakage of sensitive data as they can affect the entire table, e.g., the attacker can create another table in which he/she can dump sensitive information about the system and make it available for access by unauthorized users or a table containing monetary records can be deleted affecting large no. of users.

Fuzzy logic basically reduces to paradoxes or multivalued logic to half-truths (or half-falsities). A quote (‘) is considered as a dangerous parameter for SQL inputs, although it is partially true. We encountered places where the quote is a part of a perfectly valid input. Due to the various guidelines laid down in the defense strategies of SQLIAs, these legal inputs are also considered dangerous. To resolve such problems, we have designed a set of fuzzy rules in their deductive form. A fuzzy rule set used to determine the overall threat level is shown in Table 2. An example rule (as shown in 4th row of Table 3) is “**IF** input_risk_level **IS** high **AND** statement_type **IS** DDL **THEN** SQLI_risk **IS** severe.”

For each query, the set of rules is applied to obtain the degree of membership value which is then defuzzified to produce the output. To predict the exposure of query to SQL injection attacks, we use center of gravity (COG) method of defuzzification [13] given by the algebraic expression:

$$x^* = \frac{\int \mu(x) x \, dx}{\int \mu(x) \, dx} \tag{1}$$

Table 2 Examples of fuzzy rules

Input risk level	Statement type	SQLI risk
Valid	Any	None
Medium	DML	Average
High	DML	Severe
Medium	DDL	Severe

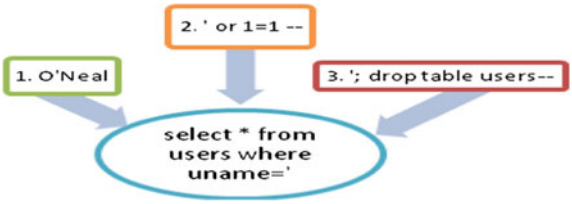

Fig. 1 Various sorts of input to a query

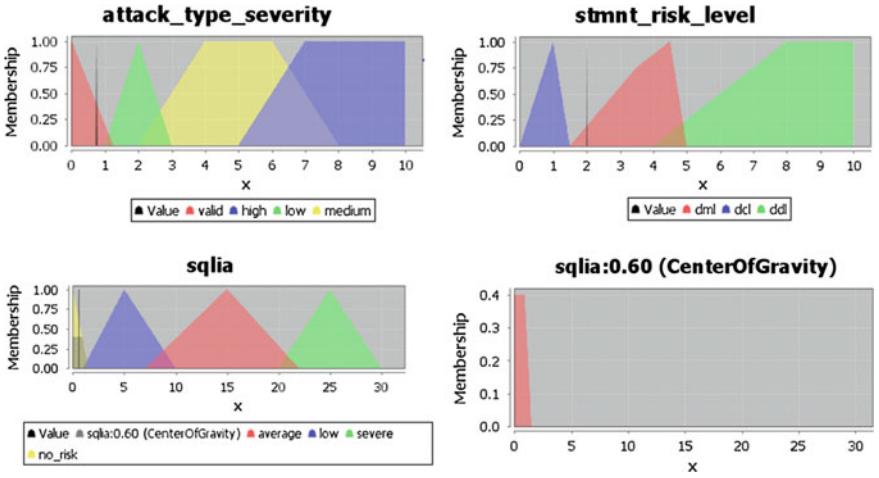

Fig. 2 Input: O'Neal

Consider the different kinds of input values in a SQL query as shown in Fig. 1. First one is a valid query despite containing quotation mark. The second form of input leads to breach in confidentiality of the system, while the last one results in the loss of sensitive data.

Initially, the negative taint values of all inputs are calculated followed by determining the type of query, in this case, a DML. Lastly, these values are fuzzified to indicate SQLIA. Our system shows appropriate indications for threat levels produced by the different inputs as shown in Figs. 2 and 3.

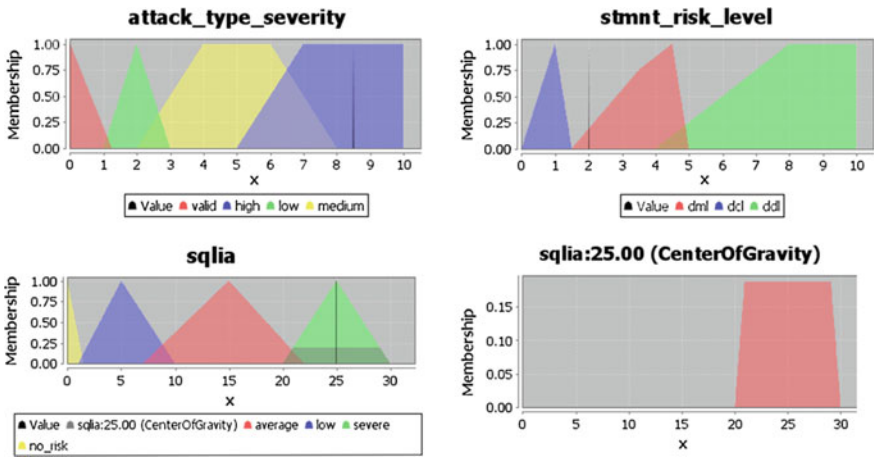

Fig. 3 Input: '; drop table users—

6 Conclusion and Future Work

In this paper, we have classified SQL injection threat level with the help of fuzzy logic and tainting techniques. The proposed model detects the impact of a full fledged query on the data source and works on successfully removing false positives. This approach uses the commonly known COG method, while implementation of other method can be explored in the near future. Security of the application can be enhanced by including more encoding patterns. This method can further be extended to solve the issue of no SQL injection.

References

1. Top 10 2013-Top 10. In: - OWASP. https://www.owasp.org/index.php/top_10_2013-top_10. Accessed 4 Mar 2016.
2. Common Weakness Enumeration. In: CWE - 2011 CWE/SANS Top 25 Most Dangerous Software Errors. http://cwe.mitre.org/top25/. Accessed 4 Mar 2016.
3. 2015 Web Application Attack Report (WAAR) - Imperva. http://www.imperva.com/docs/hii_web_application_attack_report_ed6.pdf. Accessed 5 Mar 2016.
4. 2015 Data Breach Investigations Report. In: Verizon Enterprise Solutions. http://www.verizonenterprise.com/dbir/2015/. Accessed 5 Mar 2016.
5. Shahriar H, Haddad H (2014) Risk assessment of code injection vulnerabilities using fuzzy logic-based system. Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14.
6. Huang Y-W, Huang S-K, Lin T-P, Tsai C-H (2003) Web application security assessment by fault injection and behavior monitoring. Proceedings of the twelfth international conference on World Wide Web - WWW '03 148–159.

7. Boyd SW, Keromytis AD (2004) SQLrand: Preventing SQL Injection Attacks. Applied Cryptography and Network Security Lecture Notes in Computer Science 292–302.
8. Halfond WGJ, Orso A (2005) Combining static analysis and runtime monitoring to counter SQL-injection attacks. SIGSOFT Softw Eng Notes ACM SIGSOFT Software Engineering Notes 30:1–7.
9. Fu X, Lu X, Peltsverger B, Chen S, Qian K, Tao L (2007) A Static Analysis Framework For Detecting SQL Injection Vulnerabilities. 31st Annual International Computer Software and Applications Conference - Vol 1- (COMPSAC 2007) 1:87–94.
10. Bisht P, Madhusudan P, Venkatakrishnan VN (2010) Candid. ACM Transactions on Information and System Security TISSEC ACM Trans Inf Syst Secur 13:1–39.
11. Gadgikar AS (2013) Preventing SQL injection attacks using negative tainting approach. 2013 IEEE International Conference on Computational Intelligence and Computing Research 1–5.
12. Hanmanthu B, Ram BR, Niranjan P (2015) SQL Injection Attack prevention based on decision tree classification. 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO) 1–5.
13. Ross, T. J.: Fuzzy Logic with Engineering Applications. 2nd edn. Wiley (2004).