Automated Test Input Generation for Detecting SQL Injection Vulnerability using Set Theory Concept

Nor Fatimah Awang, Azizah Abd Manaf, Ahmad Dahari Jarno

Abstract*: The use of web application has grown rapidly due to the change in lifestyle in doing business, daily activities and social life. E-commerce, E-banking, E-book, social applications and much more are among the examples of web applications. However, at the same time, the number of vulnerabilities existing in the web application has increased as well. SQL injection is among the most dangerous vulnerabilities in web applications that allow attackers to bypass the authentication and access the application database. Security testing is one of the techniques required to detect the existence of SQL injection vulnerability in a web application. However, inadequate test input during testing can affect the effectiveness of security testing. Therefore, the generation of test input is formulated by applying the Cartesian product in set theory concept to detect SQL injection vulnerability. The ideas obtained from our method will generate a set of test inputs automatically and able to exploit SQL injection vulnerability.

Index Terms*: Test Input Generation, SQL Injection Vulnerability, Security Testing.

I. INTRODUCTION

The ability to provide a variety of services which are easy to maintain, and update has made web applications very popular. This is because the development of web-based applications is becoming easier than the standalone application. Supporting technologies have grown significantly in developing web applications such as a Google web toolkit, web application development frameworks, and Content Management System. Since it is simple to use and highly accessible, web applications are used in numerous environments such as e-commerce, tax payments, human resource systems and student registration portal. Unfortunately, due to its popularity, web application attacks have also drastically increased in recent years with a variety of vulnerabilities found in computer systems and web applications. According to Trustwave's 2017 Global Security Report, almost 85.9% of the business entities have experienced with web application attacks (available at https://www2.trustwave.com/2017-Trustwave-Global-Security-Report.html). With the explosion of web application attacks, the number of web application vulnerabilities have also increased. Two prime targets of attackers are web applications vulnerabilities and web servers. According to Huang et al. [1] security vulnerabilities in the Web application layer can allow attackers to steal data, inject malicious code or break into other internal systems. Some of the most common vulnerabilities include SQL injection, cross site scripting, authorization and authentication errors. Recent studies indicate that the popularity of web applications attacks is due to the variety of vulnerabilities existing in web applications [2,3]. According to Lei et al. [4], SQL injection has become one of the topmost threats among all web application vulnerabilities. SQL injection is the most common vulnerability nowadays. A SQL code is attached to the application or user input parameters, which is then sent to a back-end SQL database for parsing and execution. By using SQL injection vulnerabilities existing in web applications, attackers can use various techniques to attack the applications. Therefore, to minimize the probability of vulnerabilities in web applications, several solutions were developed in order to detect, prevent and monitor vulnerabilities. One of the most important security practices used to mitigate the increasing number of vulnerabilities is security testing [5,6]. The goal of security testing is to search for potential vulnerabilities, weaknesses, flaws or loopholes in a system, which may compromise the applications [7]. In security testing, the preparation of proper test input is important [8-10]. In fact, the test input is an important factor that affects the test accuracy. An inadequate test input cannot fully test the application and trigger certain vulnerabilities which unable to detect the security vulnerabilities. According to Halfond, et al. [11], there are many types of inputs that can be used to detect SQL injection vulnerability. To effectively detect the security vulnerability in an application, a tester is required to have a strong knowledge of a set of input that triggers SQL injection vulnerability. Therefore, the automation process is useful to place suitable test inputs into the system. In this paper, we

Revised Manuscript Received on August 19, 2019.

Nor Fatimah Awang Faculty of Defence Science and Technology, National Defence University of Malaysia, Kuala Lumpur, Malaysia.

Azizah Abd Manaf, Advanced Informatics School (UTM AIS), UTM International Campus, Kuala Lumpur, Malaysia.

Ahmad Dahari Jarno, Cyber Security Malaysia, Level 7, SAPURA @MINES, Seri Kembangan , Selangor, Malaysia.

propose a method to generate test inputs automatically in order to detect SQL injection vulnerability. The proposed method is divided into three stages. i) Identify test input manually. ii) Categorise similar string into the template file. iii). Develop algorithm to formulate and model test input based on set theory concept.

II. SQL Injection Attack and Test Input Generation

SQL injection attack is a kind of attack where an attacker puts some invalid SQL statements mimicking a query to the web application with the aim of exploiting SQL injection vulnerability. In this attack, in order to detect SQL injection vulnerability, the attacker injects several invalid SQL statements to the web application. As a result, if a web application is susceptible to SQL injection vulnerability, the attacker can manipulate and gain access to the database server (also commonly known as a Relational Database Management System – RDBMS). By using Structural Query Language (SQL), any web application can interact with the database server dynamically to gain any information. However, most of the information stored in the databases are vulnerable to SQL injection vulnerability. The root cause of SQL injection vulnerability is that the program code of web application which involved user inputs are not filtered or sanitized [12,13]. As a result, if a web application is susceptible to SQL injection vulnerability, the attacker can manipulate and gain access to the database server.

Consider the following example, a login page prompts the user to enter the memberID into a form. The query then retrieves the memberID from table member. This query is typically used to check for login authentication. However, if the login form does not apply secure input validation, the attacker can inject several malicious inputs into the form and make a query. The vulnerability comes from the query parameter webform_memberID in which the value is inserted by the user and can be altered by an attacker. From the example above, the value stored in $_POST['webform_memberID'] is used to instruct the SQL query to perform SELECT keyword. Therefore, if the attacker wants to modify the query, he only needs to input something like 'or 1=1 -. At this stage, when the database engine finds the value 'or 1=1 --, the database interpreter considers the value of 'or 1=1 is always equal to true. Moreover, any code or value after the double – will become a comment. Everything condition or value after "- -" will be ignored by the SQL database server. Consequently, the authentication step will be bypassed by the attacker when the condition of query is always equal to true and the rest of the sentence becomes a comment.

However, the attacker also can use a lot of different test inputs in order to exploit SQL injection vulnerabilities. In general, the generation of test input is an activity that produces test input for the software under test (SUT) automatically and usually reduces testing effort. Random technique, symbolic execution, genetic algorithm, mutation-based and grammar-based are some of the test input generation approaches that can generally be described as test input generation techniques [2,14]. Symbolic execution and genetic algorithms are typically required source codes to generate and execute test input. Meanwhile, mutation-based approach is used to create new test inputs by altering the existing input. Grammar-based approaches require structure input and grammar rules to formulate and generate test input. In this paper, we use the concept of set theory to generate test input. The difference between set theory and grammar-based is generation of test input using set theory. It is easier to formulate and model the new test input without knowing the structure of grammar.

III. Method for Generating Test Inputs

In this section, we briefly describe our proposed method in order to formulate the test input and generate test input automatically. The first stage is to identify the test input that causes to SQL injection vulnerability. Table 1 shows the multiple test inputs that are using to exploit SQL injection vulnerabilities. There are several keywords in SQL such as UNION SELECT, HAVING and DROP TABLE have been used to alter database structure [15].

Table 1 Example of test input to exploit SQL Injection vulnerability

Test ID	Test Input
1	'
2	"
3	1=1 --
4	' OR 1=1 --
5	' OR 1=1 //
6	' OR 1=1 #
7	" OR 1=1
8	" OR 1=1 //
9	" OR 1=1 #

The second stage identifies and categorises special strings with similar features that contribute to the development of test inputs. The strings with similar features can be grouped and stored in one template file. In this paper, a set theory concept is employed where the set is referring to group object and elements are referring to members of a group. To construct elements to form as invalid test input, the elements in every set are associated and mapped together using the Cartesian product concept. A set is referring to a template file that has been defined while elements in a set are referring to a list of special string that already stored in template file. Besides that, a new element produced by Cartesian product is referring to a list of test input. This paper has defined and classified six different sets of template files as the following:

- Numeric stands for the set of all integer numbers, that is, Numeric = {0, 1, 2, 3, . . . }
- Alpha stands for the set of all alphabet characters, that is, Alpha = {a, b, c, …, A, B, C,..}
- NonAlpha stands for the set of all special characters which are not numeric or alphabet characters, that is, NonAlpha = {#, *, %, $, ‘, “, …. }
- SQL stands for the set of all SQL syntax, that is, SQL = {OR, AND, SELECT, HAVING, UNION, …. }
- Operator stands for the set of all arithmetic operators, that is, Operator = {>, =, >=, .. }
- Inline Comment stands for the set of all comments in programming language, that is, Inline Comment = {//, /*, --, … }

The last stage is to develop algorithm to automatically generate test input based on template that has been defined in second stage. The Cartesian Products is applied to take and combine all elements from selected groups and produce new elements. The result of combination using Cartesian Products method will produce new elements. Let A, B, C and D are sets representing the elements of Non alphanumeric, SQL, operator and Inline comment, respectively. Referring to the indicated sets of elements, the process of forming Cartesian Products is denoted as:

$$A \times B \times C \times D = \{(a,b,c,d) \mid a \in A \ \& \ b \in B \ \& \ c \in C \ \& \ d \in D\} \quad (1)$$

The combination of A×B×C×D collection within these elements will form test input known as attack pattern. Forming the Cartesian product of four sets A, B, C and D involved creating ordered quadruple pairs of elements as shown in Fig. 1.

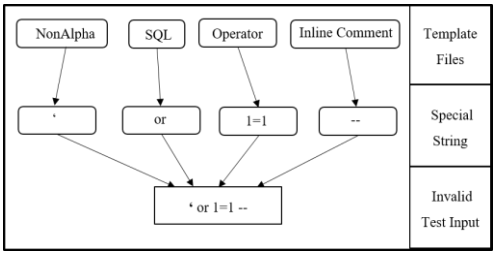

Fig. 1 Formation of test input

The following is the algorithm of test input generation by applying cartesian product algorithm to generate test input automatically**.**

Input : Template files (NonAlpha, SQL, operator)
Output : testinput.txt (a list of test input)
1: begin
2: NA ← extract element in NonAlpha
3: SQL←extract element in SQL
4: O ← extract element in operator
5: let n be the number of elements of NA
6: NA ← [‘’’, ‘!’, ‘%’]
7: SQL ← [‘OR’, ‘AND’]
8: O ← [‘1=1’, ‘1=2’]
9: for all i = 0 to n
10: CurrentPattern ← NA[i]
11: Let m be the number of elements of SQL
12: for all j = 0 to m
13: CurrentPattern ← NA[i], SQL[j]
14: Let p be the number of elements of O
15: for all k = 0 to p
16: CurrentPattern ← NA[i], SQL[j], O[k];
17: return testinput
18: k←k+1
19: end for
20: j←j+1
21: end for
22: i←i+1
23: end for
24: end

IV. EXPERIMENTAL RESULTS

This section presents the result generated from algorithm stated in the previous section. As shown in the previous algorithm, there are three template files will be read. Each element in a template file, will be mapped to the element of the next template file by applying the Cartesian Product algorithm. Once all variable and elements have been mapped and generated, the algorithm will generate the test input. By using this algorithm, the tester can easily modify any of the variable name and element stored in template files. Table 2 shows the partial of invalid test inputs. The invalid test inputs will be saved in a text file.

Table 2. List of test input generated from this method

Test ID	Test Input
1	' OR 1=1;
2	' OR 1=1 -- --
3	' OR 1=1 -- #
4	' OR 1=1 -- %
5	' OR 1=1 -- '
6	' OR 1=1 #
7	' OR 1=2;
8	' OR 1=2 -- --
9	' OR 1=2 -- #

In order to detect SQL injection vulnerability, a list of test input generated by our method will be used to perform injection attack. This experiment installed an Apache HttpClient library to retrieve response page from a web server. Wacko Picko website was chosen for this experiment. Wacko Picko is an online photo sharing website that allows users to upload,

comment and purchase pictures. The function of HttpClient is used to extract the HTTP header which containing the http response and this method manually observe the behavior of the http response. The idea of observation is usually to look for abnormal outputs or test results such as error message results that may expose to detect SQL injection vulnerability. To evaluate the effectiveness of test input generated, a total number of 624 invalid test inputs generated were tested with Wacko Picko vulnerable web application running apache and MySQL database server. Table 3 shows the test result after injecting all the test inputs into Wacko Picko website. The results categorized into types of error messages are bypass authentication, SQL error, unknown column, unknown table and shutdown the database server.

Table 3. Experimental Result

Application	Target parameter	Type of Error	% of test input cause to SQL injection vulnerability
WackoPicko	username/ password	Bypass authentication	4.8%
		SQL error	46.7%
		Unknown column, unknown table	34.3%
		Shutdown	0%

The experimental result in Table 3 shows that the test inputs generated by our method was successfully detecting SQL injection vulnerability. For the entire of 624 test inputs generated, 4.8% of test inputs injected, successfully bypassed the authentication and entered the application. Bypass the authentication page can cause to high severity level and require the developer to resolve the vulnerability issue immediately. It also shows that the SQL error message scores the highest number which is 46.7% in term of detecting SQL injection vulnerability compared to other type of errors. Based on these results, we can conclude that the test input generated by this method is able to detect the existence of SQL injection vulnerability in web application.

V. CONCLUSION

In this paper, we introduced and proposed a method to generate a set of test input automatically by using a set theory concept with applying Cartesian product algorithm. This method has been successfully tested. Based on the result shows in Table 3, this method is able to detect SQL injection vulnerability in web application. As future works, we plan to extend our approach to cover other types of vulnerabilities such as cross site scripting vulnerability.

VI. ACKNOWLEDGMENT

This work has been supported by the National Defence University of Malaysia, University Technology of Malaysia, and Cyber Security Malaysia.

REFERENCES

1. Y.W. Huang, S.K. Huang, T.P. Lin and C.H. Tsai, Web application security assessment by fault injection and behavior monitoring. Proceedings of the Twelfth International Conference on World Wide Web - WWW '03, (2003) 148.
2. D. Appelt, C.D. Nguyen and L.C. Briand, Behind an application firewall, are we safe from SQL injection attacks? 2015 IEEE 8th International Conference on Software Testing, Verification and Validation, ICST 2015 -Proceedings.https://doi.org/10.1109/ICST.2015.7102581, (2015). \
3. N. Antunes, N. Laranjeiro, M. and H. Madeira, Effective Detection of SQL / XPath Injection Vulnerabilities in Web Services. (2009)
4. L. Lei, X. Jing, L. Minglei and Y. Jufeng, A Dynamic SQL Injection Vulnerability Test Case Generation Model Based on the Multiple Phases Detection Approach. 2013 IEEE 37th Annual Computer Software and Applications Conference, (2013) 256–261.
5. F.T. Alssir and M. Ahmed, Web security testing approaches: Comparison framework. Advances in Intelligent and Soft Computing, 144 AISC, (2012) 163–169.
6. J. Bozic and F. Wotawa, Security testing based on attack patterns. Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2014, (2014) 4–11.
7. M. Bishop, About Penetration Testing. IEEE Security & Privacy, 5(6), (2007) 84–87.
8. A. Takanen, J. Demott and C. Miller, Fuzzing for software security testing and quality assurance, Artech House, Norwood, MA, 2008.
9. R. Akrout, E. Alata, M. Kaaniche and V. Nicomette, An automated black box approach for web vulnerability identification and attack scenario generation. Journal of the Brazilian Computer Society, (2014).
10. A. Kiezun, P. J. Guo, K. Jayaraman, M. D. Ernst, Automatic creation of SQL injection and cross-site scripting attacks. Proceedings - International Conference on Software Engineering, (2009) 199–209.
11. W.G.J. Halfond, J. Viegas and A. Orso, A Classification of SQL Injection Attacks and Countermeasures. Preventing Sql Code Injection By Combining Static and Runtime Analysis, (2008).
12. T. Scholte, D. Balzarotti and E. Kirda, Have things changed now? An empirical study on input validation vulnerabilities in web applications. Computers and Security, 31(3); 344–356, (2012)
13. Y.S. Jang and J.Y. Choi, Detecting SQL injection attacks using query result size. Computers & Security, 44, (2014) 104–118.
14. Fuzzing with Code Fragments on http://dl.acm.org/citation.cfm?id=2362793.2362831
15. MeiJunjin. An approach for SQL injection vulnerability detection. ITNG 2009 - 6th International Conference on Information Technology: New Generations, (2009) 1411–1414.