IJSETT International Journal

"International Journal for Science and Emerging Technologies with Latest Trends" 3(1):46-53 (2012)

An analysis on Blocking of SQL Injection Attacks by Comparing Static and Dynamic Queries

Jaskanwal Minhas
Dept. of Computer Science and Engineering,
Sant Baba Bhag Singh Institute of Engineering and Technology,
Jalandhar, Punjab, India.

Raman Kumar
Dept. of Computer Science and Engineering,
D A V Institute Engineering and Technology,
Jalandhar, Punjab, India.

Abstract

Due to rapid expansion of internet, web applications now becomes a part of everyday life. This in turns increased the numbers of web applications incidents and exploit web application vulnerabilities are increasing. As the Web applications support static and dynamic queries to access the database A large number of these incidents are SQL Injection attacks which are a serious security threat to databases which contain sensitive information, the leakage of which cause a large amount of loss. SQL Injection Attacks occur when an intruder changes the query structure by inserting any malicious input. There are number of methods available to detect and prevent SQL Injection Attacks. But these are too complex to use. This paper discuses a very simple, effective and time saving technique to detect SQLIAs which uses combined static and dynamic analysis.

KEYWORDS

*Dynamic query, SQL query, SQLIAs, Static query, SQL injection attack, Static Analysis, Dynamic Analysis, Code-Injecton, Validation***.**

1. Introduction

Nowadays most of the activities are done by using web applications. For example paying bills, online shopping, online booking etc. But to store all types of data databases are required. But these databases are not safe from attacks by various intruders. Most of the attacks on these databases are SQL Injection attacks. For a successful SQL Injection attack the intruder append a SQL query to the original query. The intruder thus steal large amount of important information from databases. SQLIA is an attack which does not harm the system like any other attack but because of its ability to steal important information from databases makes this type of attacks a serious security threat.

The main aim of this paper is to discuss about various techniques available for detection of SQLIAs and to propose an effective method for detection and prevention from SQLIAs . Some of the available techniques are not able to detect all types of attacks. Also some requires modification in source code of web application to detect all types of attacks. The paper is organized in various sections. Section II discusses the categories of SQLIAs. Section III discusses the related work. Section IV defines the proposed method. Section V concludes this paper.

2. Categories of SQL injection attacks

Numbers of SQL Injection attack methods are available. Halfond, Viegas and Orso[1] classified SQLIAs in various categories which are given below.

A. Tautologies

This type of attack injects SQL tokens to the conditional query statement which always evaluates true. This type of attack is used to bypass authentication control and access to data by exploiting vulnerable input field which use WHERE clause.

"Select * from branch where branch_name ='State bank of patiala' and address ='a' OR '1'='1' "

As the tautology statement (l = 1) has been added to the query statement so it is always true.

B. Illegal/Logically Incorrect Queries

When any query is rejected, an error message is returned by SQL Oracle engine. This error messages help attacker to find vulnerable parameters in the Database.

C. Union Query

By using this technique, attackers join SQLIA to safe query by using word UNION and then can get data about other tables from the application. Consider the following example

Select department, designation from employee where emp_code=$emp_code

By injecting the following loan_no value:

$emp_code=24576 UNION select address from branch

We will have the following query:

Select department,designation from employee where emp_code=24576 UNION select address from branch

Which will join the result of the original query with all the customer names.

D. Piggy-backed Queries

In this type of attack, intruders exploit database by using query delimiter, such as ";" by appending extra query to the original query. In this attack database receives and execute a multiple distinct queries. Normally the first query is legitimate query, whereas following queries could be illegitimate. So attacker can inject any command related to SQL to the database. In the following example, attacker inject " 0; drop table user" into the pin input field instead of logical value. Then the application would produce the query:

Select info from employee where emp-code='123' AND designation='PO'; drop table branch

Because of ";" character, database accepts both queries and executes them. The second query is not legitimate and can drop users table from the database.

E. Stored Procedure

Stored procedure is a part of database that programmer could set an extra abstraction layer on the database. By using stored procedure a user can store its own function according to the need. In stored procedure, a collection of SQL queries are included. As stored procedure could be coded by programmer, so, this is also one of the causes of SQLIA. Depending on specific stored procedure in the database there are number of different ways to attack.

Create procedure abc @emp_code varchar2, @designation varchar2

AS

Exec("select * from employee where emp_code='"+@ecode+"' and designation='"+ desi"+"');

GO

If the intruder adds one more query after the legitimate query, then the normal query is converted into piggy backed query which is a type of SQLIA.

Select * from users where emp_code='123' and designation='PO';Shutdown;

After the execution of original query the second query which is illegitimate is executed and causes database shut down.

F. Alternate Encoding

In this type of attack the regular strings and characters are converted into hexadecimal, ASCII and Unicode. Because of this the input query is escaped from filter which scans the query for some bad characters which results in SQLIAs i.e. the converted SQLIA is considered as normal query.

G. Inference

By this type of attack, intruders change the behaviour of a database or application. There are two well known attack techniques that are based on inference: blind injection and timing attacks.

• Blind Injection: Sometimes developers hide the error details which help attackers to compromise the database. In this situation attacker face to a generic page provided by developer, instead of an error message. So the SQLIA would be more difficult but not impossible. An attacker can still steal data by asking a series of True/False questions through SQL statements. Consider two possible injections into the login field:

SELECT * FROM employee WHERE emp_code='12365' and 1 =0 -- AND department=''

If the application is secured, both queries would be unsuccessful, because of input validation. But if there is no input validation, the attacker can try the chance. First the attacker submit the first query and receives an error message because of "1=0". So the attacker does not understand the error is for input validation or for logical error in query. Then the attacker submits the second query which always true. If there is no login error message, then the attacker finds the login field vulnerable to injection.

• Timing Attacks: A timing attack lets attacker gather information from a database by observing timing delays in the database's responses. This technique by using if-then statement cause the SQL engine to execute a long running query or a time delay statement depending on the logic injected. This attack is similar to blind injection and attacker can then measure the time the page takes to load to determine if the injected statement is true. This technique uses an if-then statement for injecting queries. WAITFOR is a keyword along the branches, which causes the database to delay its response by a specified time. For example, in the following query:

declare@s varchar(8000) select@d = abc_nameO if (ascii(substring(@s, I, I)) & (power(2, 0))) > 0 waitfor delay '0:0:5'

Database will pause for five seconds if the first bit of the first byte of the name of the current database is 1. Then code is then injected to generate a delay in response time when the condition is true. Also, attacker can ask a series of other questions about this character. As these examples show, the information is extracted from the database using a vulnerable parameter.

3. Literature Survey of SQL injection attacks

S. Boyd, A. Keromytis (2004) discussed SQLrand: preventing SQL injection attacks. In this paper a practical protection mechanism is presented against SQL injection attacks[12]. In this the concept of instruction-set randomization is applied to SQL, creating instances of the language that are unpredictable to the attacker. Queries injected by the attacker will be caught and terminated by the database parser. A proxy server is used for the de-randomization process. According to the paper by using this proxy a high degree of portability and security is achieved. But the use of this proxy imposes a significant overhead in terms of infrastructure. The second problem with this technique is that this technique could be circumvented if the key used for the randomization were exposed.

Buehrer. G, Weide. B. W, Sivilotti. P A (2005) discussed Using Parse Tree Validation to Prevent SQL Injection Attacks[8]. In this paper parse tree is used to eliminate SQL injection vulnerabilities. The technique is based on comparing, at run time, the parse tree of the SQL statement before inclusion of user input with that resulting after inclusion of input. This method is useful if the correct parse tree is statically generated for any database. This method detects all types of attacks.

F. Valeur, D. Mutz, G. Vigna (2005) discussed A Learning-Based Approach to the Detection of SQL Attacks[10]. In this paper development of an anomaly-based system is discussed that learns the profiles of the normal database access performed by web-based applications using a number of different models. These models allow for the

detection of unknown attacks with reduced false positives and limited overhead. The SQL queries generated in a web application are learned to generate models. Then runtime SQL queries are compared to the generated model to check for discrepancies. If a poor training set is used, many false positive and negative results may occur.

Halfond W. G, Orso. A (2005) discussed AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks[6]. In this paper a new technique for detecting and preventing SQL injection attacks is presented and evaluated. This technique uses a model-based approach to detect illegal queries before they are executed on the database. In its static part, by using program analysis the technique automatically builds a model of the legitimate queries that could be generated by the application. In its dynamic part, by using runtime monitoring the technique inspects the dynamically-generated queries and checks them against the statically-built model. To achieve this, a tool AMNESIA is used that implements this technique. The results of the study show that this technique was able to stop all of the attempted attacks without generating any false positives. The primary limitation of this tool is that its success depends on the accuracy of its static analysis for building static query models.

V. Haldar, D. Chandra, and M. Franz (2005) discussed Dynamic Taint Propagation for Java. In this paper dynamic solution is proposed that tags and tracks user input at runtime and prevents its improper use to maliciously affect the execution of the program. In this paper, a framework for tagging is proposed for tracking and detecting the improper use of improperly validated user input (also called tainted input) in web applications.

Wei. K, Muthuprasanna. M, Kothari. S (2006) discussed Preventing SQL injection attacks in stored procedures[14]. In this paper a technique is presented to prevent databases from SQL Injection Attacks by preventing injections through stored procedures. As stored procedure is a collection of number of SQL queries, so stored procedures is also vulnerable to SQL Injection Attacks. This technique works by combining static analysis with runtime validation. The basis of such a technique is that the control flow graph of the stored procedures can be represented as an SQL-graph which indicates what user inputs the dynamically built SQL statements depend on. By using an SQL-graph, the set of SQL statements are reduced which need to verify, by looking at only a small subset of all the SQL statements in the stored procedure at runtime. During runtime, Finite State Automaton(FSA) is retrieved from the EXEC(@SQL) procedure call and check the SQL statement with inclusion of user inputs for compliance, flagging them safe or unsafe.

William G.J. Halfond et al (2006) discussed A Classification of SQL Injection Attacks and Countermeasures[11]. In this paper an extensive review of the different types of SQL injection attacks known to date is presented. For each type of attack, descriptions and examples of how attacks of that type could be performed is provided. This paper also present and analyze existing detection and prevention techniques against SQL injection attacks. The strength and weakness of each technique is discussed. This paper helps us to understand the various SQL Injection Attack types as various SQLIA are discussed with explained examples.

Stephen Thomas, Laurie Williams (2007) discussed Using Automated Fix Generation to Secure SQL Statements[26]. In this paper an automated fix generation solution is presented for replacing vulnerable statement based SQL statements with secured Prepared statement based SQL statements. The limitation is that this technique is not used for Batch SQL statements.

Yuji Kosuga et al (2007) discussed Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection[5]. In this paper the technique, Sania, is presented for detecting SQL injection vulnerabilities in web applications during the development and debugging phases. Sania takes the SQL queries between a web application and a database, and automatically generates elaborate attacks according to the syntax and semantics of the SQL queries. In addition, Sania compares the parse trees of the intended SQL query and those resulting after an attack to prevent SQLIAs. Sania is a effective dynamic analysis technique. But in dynamic analysis techniques the vulnerabilities found in web application must be manually fixed by the developer and not all of them can be found without predefined attack codes.

Shanmughaneethi et al (2009) discussed Securing Web Applications with Service Based SQL Injection Detection[27]. This paper proposes methodology in which when the user submits the SQL query at runtime, the query has to be parsed by the independent service for the correctness of the syntactic structure and user data. The limitation is that this technique is not used to detect SQLIAs in stored procedures.

Inyong Lee, Soonki Jeong, Sangsoo Yeo, Jongsub Moon (2011) discussed A novel method for SQL injection attack detection based on removing SQL query attribute values[23]. This paper proposes a very simple and effective detection method for SQL injection attacks. In the method by using static analysis static query structure is stored. Then in dynamic analysis attribute values are removed from the query. After removing attribute values exclusive OR operation is performed between the static query structure stored and the run time query (dynamic query). If the result of this exclusive OR operation is zero then the requested query is valid otherwise it is a SQLIAs.

This paper may be used to propose two main algorithms from which one is used for removing attribute values and the second algorithm is used to perform exclusive OR operation to check whether the requested query is valid or is a SQL Injection Attack.

4. Proposed method

In this research work combined static and dynamic analysis technique is proposed to reduce false positives and false negatives. The static query structure is compared with dynamic query. In this database is maintained to store the valid query structure. These valid queries are also known as static queries. The attribute values of dynamic queries are removed during run time and compared with previously stored static queries having same number of tokens as in dynamic query. After removing attribute values the method locates for static queries having same number of tokens as in dynamic query. Then the dynamic query is compared character by character only with that static queries having same number of tokens. The technique helps in improving response time. If match is found requested dynamic query is valid query otherwise it is SQL Injection Attack.

4. Conclusion

In this paper an effort is made to propose a simple method to detect SQLIAs in which dynamic queries are compared with static queries after removing attribute values. The algorithm to be proposed will compare dynamic query character by character only with that static queries having same number of tokens. This will result in improved response time. Removing of attribute values will make a SQL query independent from the database.

Acknowledgements

The authors also wish to thank many anonymous referees for their suggestions to improve this paper.

References

[1] Paros. Parosproxy.org. http://www.parosproxy.org/.

[2] C. Gould, Z. Su, P. Devanbu, "JDBC checker: a static analysis tool for SQL/JDBC applications", In Proceedings of the 26th International Conference on Software Engineering, ICSE, 2004, pp. 697–698.

[3] G. Wassermann, Z. Su, "An analysis framework for security in web applications", In Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems, SAVCBS, 2004, pp. 70–78.

[4] Paros. Parosproxy.org. http : // www.Parosproxy.org/.

[5] Yuji Kosuga et al, "Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection", In Computer Security Applications Conference, 2007, pp.107-117.

[6] Halfond W. G, Orso. A, "AMNESIA : Analysis and Monitoring for Neutralizing SQL-Injection Attacks", In Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering, 2005, pp. 174-183.

[7] Z. Su, G. Wassermann, "The essence of command injection attacks in web applications", In Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2006, pp. 372–382.

[8] Buehrer. G, Weide. B. W, Sivilotti. P A, "Using Parse Tree Validation to Prevent SQL Injection Attacks", In Proceedings of the 5th international Workshop on Software Engineering and Middleware, 2005, pp. 105-113.

[9] Wei. K, Muthuprasanna. M, Kothari. S, "Preventing SQL injection attacks in stored procedures", In Software Engineering Conference 2006. Australian, 2006, pp. 18-21.

[10] F. Valeur, D. Mutz, G. Vigna , "A Learning-Based Approach to the Detection of SQL Attacks", In Proceedings of the Conference of Detection of Intrusions and Malware and Vulnerability Assessment, 2005, pp. 123-140.

[11] William G.J. Halfond et al, "A Classification of SQL Injection Attacks and Counter measures", In Proceedings of the Intern. Symposium on Secure Software Engineering, 2006, pp. 101-111.

[12] S. Boyd, A. Keromytis, "SQLrand: preventing SQL injection attacks", In Applied Cryptography and Network Security, In LNCS, vol. 3089, 2004, pp. 74-82.

[13] M. Martin, B. Livshits, and M. S. Lam, "Finding Application Errors and Security Flaws Using PQL: A Program Query Language", In Proceedings of the 20th Annual ACM

SIGPLAN conference on Object oriented programming systems languages and applications, 2005.

[14] V. Haldar, D. Chandra, and M. Franz, "Dynamic Taint Propagation for Java", In Proceedings 21st Annual Computer Security Applications Conference, 2005.

[15] T.C. Pietraszek, V. Berghe, "Defending against injection attacks through context–sensitive string evaluation", In Proceeding of Recent Advances in Intrusion Detection, in: LNCS, vol. 3858, 2006, pp. 124–145.

[16] A. Nguyen-Tuong, S. Guarnieri, D. Greene, J. Shirley, D. Evans, "Automatically hardening web application using precise tainting information", In Twentieth IFIP International Information Security Conference, in: LNCS, vol. 181, 2005, pp. 295–307.

[17] V.B. Livshits, M.S. Lam, "Finding security errors in Java programs with static analysis", In Proceedings of the 14th Usenix Security Symposium, 2005, pp. 271–286.

[18] W. R. Cook and S. Rai, "Safe Query Objects: Statically Typed Objects as Remotely Executable Queries", In Proceedings of the 27th Intern. Conf. on Software Engineering, 2005, pp. 97–106.

[19] D. Scott, R. Sharp, "Abstracting application-level web security", In Proceedings of the 11th International Conference on the World Wide Web, 2002, pp. 396–407.

[20] R. McClure and I. Kr¨uger, "SQL DOM: Compile Time Checking of Dynamic SQL Statements", In Proceedings of the 27th Intern. Conf. on Software Engineering , 2005, pp. 88–96.

[21] Y. Huang, S. Huang, T. Lin, C. Tasi, "Web application security assessment by fault injection and behavior monitoring", In Proceedings of the 12th International Conference on World Wide Web, 2003, pp. 148–159.

[22] Y. Huang, F. Yu, C. Hang, C.H. Tsai, D.T. Lee, S.Y. Kuo, "Securing web application code by static analysis and runtime protection", In Proceedings of the 12th International World Wide Web Conference ACM, 2004, pp. 40–52.

[23] Inyong Lee, Soonki Jeong, Sangsoo Yeo, Jongsub Moon, "A novel method for SQL injection attack detection based on removing SQL query attribute values", In Center for Information Security Technologies, Korea University, 2011, pp. 136-713.

[24] Jeom-Goo Kim , "Injection Attack Detection using the Removal of SQL Query Attribute Values", In IEEE, 2011.

[25] W. G. Halfond and A. Orso, "Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks", In Proceedings of the Third Intern. ICSE Workshop on Dynamic Analysis (WODA 2005), 2005, pp. 22–28.

[26] Stephen Thomas, Laurie Williams, "Using Automated Fix Generation to Secure SQL Statements", In Third International Workshop on Software Engineering for Secure Systems, 2007, pp. 287-293.

[27] V. Shanmughaneethi et al, "Securing Web Applications with Service Based SQL Injection Detection", In International Conference on Advances in Computing, Control and Telecommunication Technologies, 2009, pp. 702-704.