Asian Journal of Information Technology 12 (4): 131-139, 2013
ISSN: 1682-3915

The SQL Injection Attack Detection and Prevention by Classification and Analysis

[1]V. Nithya, [1]S. Lakshmana Pandian and [1, 2]R. Regan
[1]Department of Computer Science and Engineering,
Pondicherry Engineering College, Puducherry, India
[2]University College of Engineering, Panruti, Anna University, Tamil Nadu, India

Abstract: Securing the web application against hacking is a big challenge. One of the most common types of hacking technique to attack the web application is SQL Injection Attack (SQLIA). In resulting of this attack an attacker can access, modify and even destroy the database of a web application. SQL injection is occurring when data provided by user is not properly validates and is included directly in a SQL query. The analysis of detection and prevention of SQLIA help to avoid this type of attack. Researchers describe a technique to detect and prevent this kind of manipulation and hence eliminate SQL injection attack. The existing solution to SQL injection requires source code modification and increases the possibilities of new injection points. In this study, researchers propose static and dynamic analysis to detect the SQL injection attack and we propose decision tree classification to prevent them.

Key words: SQL injection attacks, classification, detection, static and dynamic analysis, prevention

INTRODUCTION

With the rise internet services now days all web applications are depended on the internet. Example: online banking, shopping, university admissions and various government activities. So, researchers can say that these activities are the key component of today's internet infrastructure. Web applications are vulnerable to a variety of new security threats. SQL injection attacks are one of the most significant of such threats. SQL injection attacks are increasing continuously and posy high level security risks because they allow attacker's access to the database that lie under web applications.

Information is the one of important business asset today and achieving an appropriate level of information security can be viewed as essential in order to maintain a competitive edge (Firdos and Sheikh, 2011). SQL Injection Attacks (SQLIAs) is considered as one of the top 10 web application vulnerabilities of 2010 by the Open Web Application Security Project (OWASP) (Wassermann and Su, 2004) and Semiannual report (July to December 2010) from the Web Hacking Incidents Database (WHID) (WHID, 2010) shows that SQL injection are consistently or near the top 21% of the reported vulnerabilities in 2010, consider as top second attack and recently in August, 2011, hacker steals user records from Nokia Developer Site using SQL injection attack (Su and Wassermann, 2006). They are easy to detect and exploit that is why SQLIAs are frequently employed by malicious user for different reasons. Financial fraud theft, confidential data, deface website, sabotage, espionage, cyber terrorism or simply for fun. Throughout, 2010, government, finance and retail verticals faced different but equally important, outcomes. Attacks against government agencies resulted in defacement in 26% of SQL injection attacks while retail was most affected by credit card leakage at 27% of SQL injection and finance experienced monetary loss in 64% of attacks (WHID, 2010). Furthermore, SQL injection attack techniques have become more common more ambitious and increasingly sophisticated so there is to need to find an effective and feasible solution for this problem in the computer security community. One of the important reasons of this shortcoming is that there is lack of complete methodology for the evaluation either in terms of performance or needed source code modification which in an over head for an existing system. A mechanism which will easily deployable and provide a good performance to detect and prevent the SQL injection attack is essential one. So, researchers proposed new modified SQL injection detection and prevention technique by classification, static and dynamic analysis.

Corresponding Author: V. Nithya, Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry, India

OVERVIEW OF SQL INJECTION ATTACK

SQL (Structured Query Language) is a textual language used to interact with relational database. The execution of unit of Structured Query Language is query which is a collection of statements that return a single resultset. SQL statements can modify the structure of databases and manipulate the contents of databases by using various DML, DDL commands respectively. SQL injection attack occurs when an attacker insert a series of SQL code into a query by manipulating data input into an application (Kiani *et al*., 2008).

Definition of SQLIA: Most web applications today use a multi-tier design usually with three tiers: a presentation, a processing and a data tier. The presentation tier is the HTTP web interface, the application tier implements the software functionality and the data tier keeps data structured and answers to requests from the application tier. Meanwhile, many large companies developing SQL based Database Management Systems (DBMS) which rely on hardware to ensure the desired performance. SQL injection is a type of attack which the attacker adds Structured Query Language code to input box of a web form to gain access or make changes to data. SQL injection vulnerability allows an attacker to flow commands directly to web applications underlying database and destroy functionality or confidentiality.

SQL Injection Attacks (SQLIA) process: SQLIA is hacking technique which the attacker adds SQL statements through a web application's input field or hidden parameter to access to resources. Lack of input validation in web applications causes hacker to be successful. Basically SQL process structured in three phases (Fig. 1):

- An attack sends the malicious HTTP request to the web application

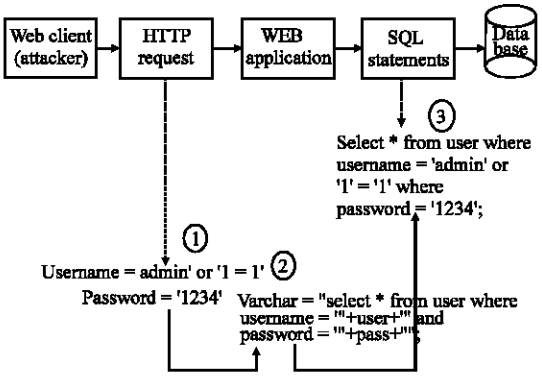

Fig. 1: SQL injection attack data flow

- Create the SQL statement
- Submits the SQL statements to the back end database

Consequence of SQLIA: The result of SQLIA can be disastrous because a successful SQL injection can read and modify (such as insert, update and delete) sensitive data from the database (Tajpour *et al*., 2011), recover the data on the database management systems and file system, execute commands (xp cmdshell) to the operating system and it can able to execute administrative operations on the database (such as shutdown the database). The main consequences of these vulnerabilities are attacks on.

Authorization: Critical data that are stored in a vulnerable SQL database may be altered by a successful SQLIA, a authorization privilege.

Authentication: If there is no any proper control on username and password inside the authentication page, it may be possible to login to a system as a normal user without knowing the right username and/or password.

Confidentially: Usually databases are consisting of sensitive data such as personal information, credit card numbers and/or social numbers. Therefore, loss of confidentially is a big problem with SQL injection vulnerability. Actually, theft of sensitive data is one of the most common intentions of attackers.

Integrity: By a successful SQLIA not only an attacker reads sensitive information but also, it is possible to change or delete this private information.

Type of SQL injection attack

Tautologies: A SQL tautology is a statement that is always true. Tautology-based SQL injection attacks are usually used to bypass user authentication or to retrieve unauthorized data by inserting a tautology into a conditional statement. A typical SQL tautology has the form or <comparison expression> where the comparison expression uses one or more relational operators to compare operands and generate an always true condition. The aim of tautology-based attack is to inject SQL tokens that cause the query's conditional statement to always evaluate the true. For example, Select * From user Where username = malani' or 1 = 1; the "or 1 = 1" is the most commonly known tautology (Fig. 2).

Piggy-backed query: In the piggy-backed query attacker tries to append additional queries to the original query

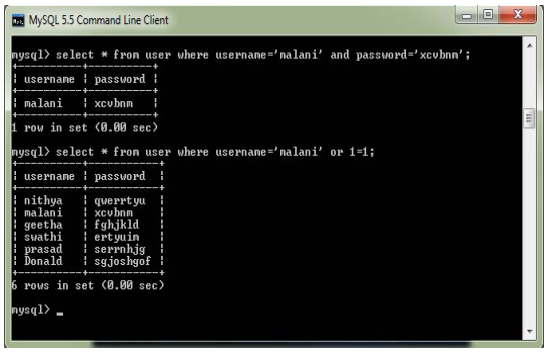

Fig. 2: Tautology SQL injection attack

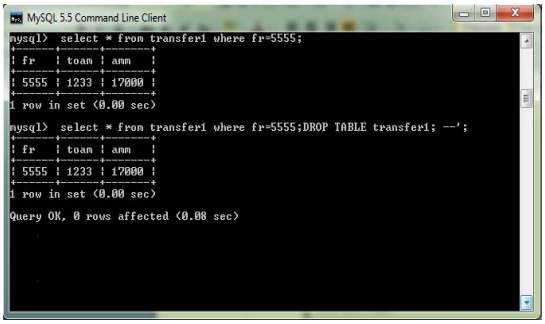

Fig. 3: Piggy-backed query

string. On the successful attack the database receives and executes a query string that contains multiple distinct queries. In this method, the first query is original whereas the subsequent queries are injected. This attack is very dangerous; attacker can use it to inject virtually any type of SQL command. For example, Select *From transfer1 Where fr = '5555'; DROP Table transfer1; --'; Here database treats above query string as two query separated by ";" and executes both. The second sub query is malicious query and it causes the database to drop the user table in the database (Fig. 3).

Logically incorrect queries: This attack takes advantage of the error messages that are returned by the database for an incorrect query. These database error messages often contain useful information that allow attacker to find out the vulnerable parameter in an application and the database schema. For example, Select * From user Where id = '1111' AND password = '1234' and convert (char, no); the purpose of this attack is to collect the structure and information of CGI.

Union query: Union query injection is called as statement injection attack. In this attack, attacker insert additional statement into the original SQL statement. This attack can be done by inserting either a union query or a statement of the form "< SQL statement>" into vulnerable parameter. The output of this attack is database will return a dataset that is union of the result of original query with the result of the injected query. For example, Select * From user Where id = '1111' Union Select * From member Where id = 'admin' --' and password = '1234'.

Stored procedure: In this technique, attacker focuses on the stored procedures which are present in the database system. Stored procedures run directly by the database engine. Stored procedure is nothing but a code and it can be vulnerable as program code. For authorized/ unauthorized user the stored procedure returns true/false. As an SQLIA, intruder input "; Shutdown; --" for username or password. Then, the stored procedure generates the following query: Select accounts From users Where login id = '1111' And pass =' '; Shutdown; This type of attack works as piggy-back attack. The first original query is executed and consequently the second query which is illegitimate is executed and causes database shut down. So, it is considerable that stored procedures are as vulnerable as web application code (Howard and LeBlanc, 2003).

LITERATURE REVIEW

In order to detect and prevent SQL injection attacks, filtering and other detection methods are being researched. Huang *et al*. (2005) proposed WAVES which is a black box testing used for testing whether SQL injection vulnerabilities occur in web applications. The tool identifies all vulnerable points that can be used by SQL injection attack to attack a web application. It builds attacks that target these points and monitors the application how response to the attacks by utilize machine learning. However, like all black-box testing techniques, it cannot able provide guarantees of completeness. JDBC-Checker (Gould *et al*., 2004) was not purely developed for preventing and detecting SQL injection attacks but it is used to prevent attacks that takes advantage of type mismatches in a dynamically generated SQL query string. As most of the SQLIAs consist of syntactically and type correct queries so this technique not possible to detect many general forms of these attacks.

Wassermann and Su (2004) proposed an approach that uses static analysis combined with automated reasoning to verify whether SQL injection queries generated in the application layer does not contain a tautology attack. The major drawback of this technique

is that it able detect only tautology based attack but unable to detect other possible types of attacks.

WebSSARI (Huang *et al*., 2004) use static analysis phases to check taint flows against preconditions for sensitive functions. It works based on sanitized input that has passed through a predefined set of filters. The limitation of approach is adequate preconditions for sensitive functions cannot be accurately expressed so some filters may be omitted.

Su and Wassermann (2006) implement their algorithm with SQLCHECK on a real time environment. It checks whether the input queries conform to the expected ones defined by the programmer. A secret key is applied for the user input delimitation. The analysis of SQLCHECK shows no false positives or false negatives. Also, the overhead runtime rate is very low and can be implemented directly in many other web applications using different languages.

Boyd and Keromytis (2004) proposed SQLrand which uses instruction set randomization of SQL statement to check SQL injection attack. It uses a proxy to a append key to SQL keyword. A de-randomizing proxy then converts the randomized query to proper SQL queries for the database. The key is not known to the attacker, so the code injected by attacker is treated as undefined keywords and expressions which cause runtime exceptions and the query is not sent to database. The disadvantage of this system is its complex configuration and the security of the key. If the key is exposed, attacker can formulate queries for successful attack.

AMNESIA is a model based technique that combines static analysis and runtime monitoring (Halfond and Orso, 2005a, b). In static analysis phase, it builds models of the different types of queries of an application that can legally generate at each access point of database. Where in dynamic analysis phase, it intercepts all queries before they are sent to the database and it check each query against the statically built models. Queries that violate against model are reported as SQL injection attack and prevented from accessing the database. But the limitation of this technique is it dependent on the accuracy of its static analysis.

Ali *et al*. (2009)'s scheme adopts the hash value approach to further improve the user authentication mechanism. They use the user name and password hash values. SQLIPA (SQL Injection Protector for Authentication) prototype was developed in order to test the framework. The user name and password hash values are created and calculated at runtime for the first time the particular user account is created.

Bisht *et al*. (2010) proposed CANDID. It is a Dynamic Candidate Evaluations Method for automatic prevention of SQL injection attacks. This framework dynamically extracts the query structures from every SQL query location which are intended by the developer (programmer). Hence, it solves the issue of manually modifying the application to create the prepared statements.

Swaddler (Cova *et al*., 2007) analyzed web application internal states: It works based on both single and multiple variables and shows an impressive way against complex attacks to web applications. First the approach describes the normal values for the application's state variables in critical points of the application's components. Then, during the detection phase it monitors the application's execution to identify abnormal states.

Wassermann and Su (2004) proposed an approach that uses a static analysis combined with automata reasoning. This technique verifies that the SQL queries generated in the application usually do not contain tautology. This technique is effective only for SQL injections that insert a tautology. This technique is effective only for SQL injections that insert a tautology in the SQL queries but cannot detect other type of SQL injection attacks.

Security Gateway (Scott and Sharp, 2002) is a proxy filtering system that enforces input validation constraints on the data flowing to a web application. For application parameters such as data flow from web page to application server, developers provided with specific constraints and transformations applied on such parameters using Security Policy Descriptor Language (SPDL). Because SPDL allow developers with desired freedom in expressing their polices. However, this approach is human based technique where developers not only know about data to be filtered but also about filters that apply to the data.

Fu *et al*. (2007) proposed a framework for web application provide a function that can be to prevent SQL injection. An input validator prohibits user input from including meta-characters to avoid SQL injection. But if we want to include meta-character in the input we cannot prevent SQL injection.

Valeur *et al*. (2005) proposed Intrusion Detection System (IDS) to detect SQL injection attacks. Their IDS system is based on a machine technique which trained using a set of application queries. The technique monitors the application at runtime using typical set queries to identify queries that does not match the model system. It can able to detect attacks with a high rate of success. But limitation of techniques is that they cannot provide guarantees about their detection capabilities since their success is dependent on the quality of the training set used. A poor training set to technique does not to generate a large number of false positive and negatives.

Buehrer *et al.* (2005) adopt the parse tree framework. They compared the parse tree of a particular statement at runtime and its original statement. They stopped the execution of statement unless there is a match. This method was tested on a student web application using SQLGuard. Although, this approach is efficient, it has two major drawbacks: additional overheard computation and listing of input (black or white).

SAFELI: Fu *et al.* (2007) proposes a static analysis framework in order to detect SQL injection vulnerabilities. SAFELI framework aims at identifying the SQL injection attacks during the compile-time. This static analysis tool has two main advantages. Firstly, it does a white-box static analysis and secondly, it uses a hybrid-constraint solver. For the white-box static analysis, the proposed approach considers the byte-code and deals mainly with strings. For the hybrid-constraint solver, the method implements an efficient string analysis tool which is able to deal with Boolean, integer and string variables.

DIWeDa approach: Roichman and Gudes (2008) proposed IDS (Intrusion Detection Systems) for the backend databases. They use DIWeDa, a prototype which acts at the session level rather than the SQL statement or transaction stage to detect the intrusions in web applications. The proposed framework is efficient and could identify SQL injections and business logic violations too.

SecuriFly: SecuriFly (Martin *et al.*, 2005) is tool that is implemented for java. Despite of other tool, chase string instead of character for taint information and try to sanitize query strings that have been generated using tainted input but unfortunately injection in numeric fields cannot stop by this approach. Difficulty of identifying all sources of user input is the main limitation of this approach.

Lee *et al.* (2011) proposed an approach to detect SQL injection attacks is based on static and dynamic analysis. This method removes the attribute values of SQL queries at runtime (dynamic method) and compares them with the SQL queries analyzed in advance (static method) to detect the SQL injection. Since, detection is based on removing attribute value of query which leads to integrity problem.

PROPOSED MODEL

The application programmer can send query requests by running the application program. The requested query sent to the classification module where it enquires with past queries of all registered users and using decision making rules classify user as ethical and unethical. The classification directs the ethical user queries to undergo static query and dynamic analysis. During static analysis extract query structure from the programs and generate DFA (Deterministic Finite Automata) for static query called as static model. Similarly at runtime capture the dynamic query and generate DFA (Deterministic Finite Automata) known as dynamic model.

Now Validate Dynamic Query Model with the Static Query Model. If SQL injection attack occurs it will add SQL token to the user input and hence there will be change in the query structure. If the dynamic query executed contains any malicious code then it does not match with the Static Query Model then it will be rejected not allowed to access the database as show in Fig. 4.

Classification module: The classification is used to find the similar data items which belong to the same class. Decision tree learning, used in data mining, statistics and machine learning. Decision tree is a predictive model which maps observations about a data item to conclusions about the data item's target value.

Descriptive names for such tree models are called as classification trees or regression trees. In classification tree structures, leaves denotes class labels and branches represent conjunctions of features that lead to those class labels. In decision analysis, a classification tree can be used to represent decisions and decision making.

For the purpose of prevention, the users are first classified into ethical and unethical users using their past

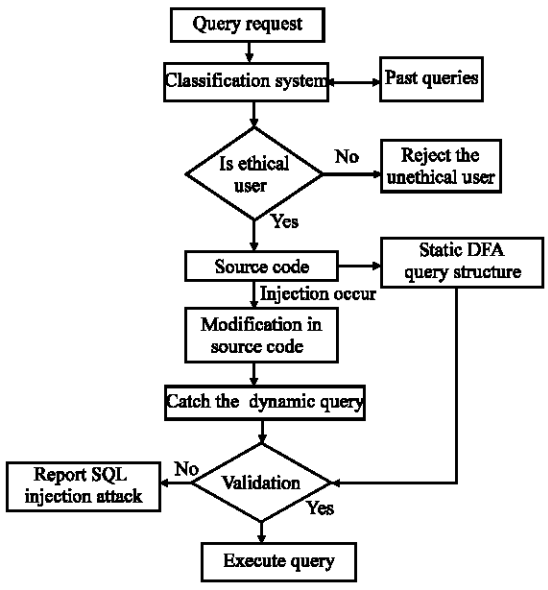

Fig. 4: Overview of proposed system

queries by using the decision tree classification algorithm where the patterns are stored in the database. After getting the results of classification, only the ethical users are allowed to access the next query in web application and further queries are validated using static and dynamic analysis. Example: when the user login to online banking application by analysising their past queries of that user they is considered as ethical and unethical. Some time ethical user also hack the web application so to, enhance detection of SQL injection attack further the application is validated using static and dynamic analysis.

Figure 5 and 6 show dynamic query generated in login page is compared with past query, since the login form is not injected the SQL injection attack its result as ethical user and allow access the next page in the web application, e.g., Net banking application.

In Fig. 7 and 8 since user name is injected with tautology expression the dynamic query generated is compared with past query it results as unethical user and block access of the web page.

Static analysis: In the context of computer science, static analysis defines to the automated process of analyzing code without running the code that is being analyzed. In static analysis process, it determines the query structure. This step identifies the all possible queries present in the program and generates the structure for these queries that is named as Static Query Model.

Fig. 5: Login form

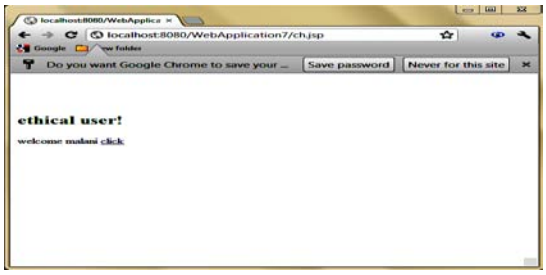

Fig. 6: Successful login

Static Query Model can be represented by the Deterministic Finite Automata (DFA) of the query which represents all possible values query. DFA is generated by exacting regular grammar from query. Example: Select * from transfer1 where fr = 'num'. Figure 9 shows the DFA structure of the earlier query similarly exact DFA structure of all queries in web application, e.g., Net banking application.

Dynamic analysis: Dynamic analysis is the testing and evaluation of a program by executing data in real-time. Dynamic analysis is a process of testing and evaluation of a program by executing data in real-time. In dynamic analysis step, it takes input as the query formed at the runtime and form the Dynamic Query Model for each query. In Dynamic Query Model string of queries are tokenized into SQL token such as special strings, characters, keywords, identifiers and numbers. Example: Query string present in the program.

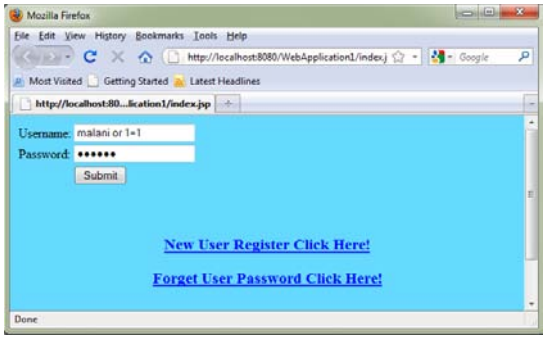

Fig. 7: Login form with tautology injection

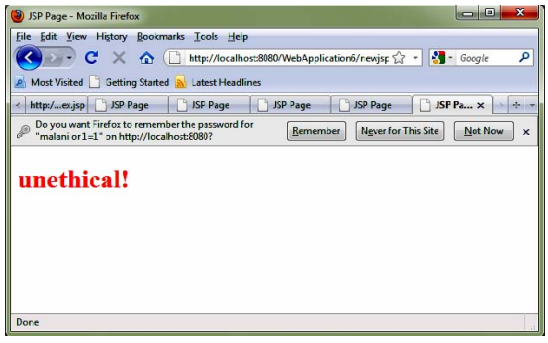

Fig. 8: Tautology injected query blocked

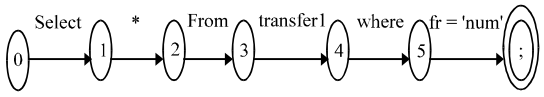

Fig. 9: Static DFA form of the query

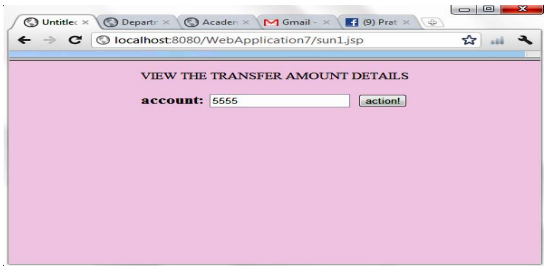

Fig. 10: View the transfer amount details

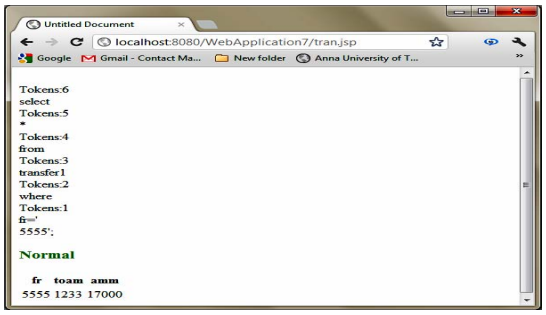

Fig. 11: Display of transfer amount details

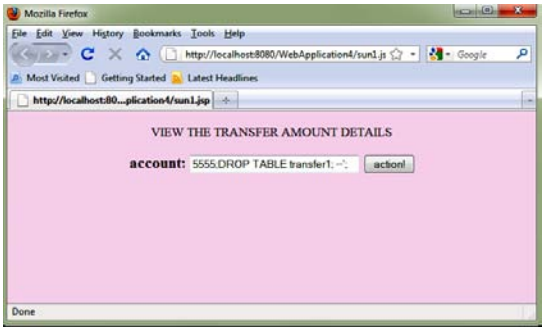

Fig. 12: Fund transfer details with Piggy backed injection

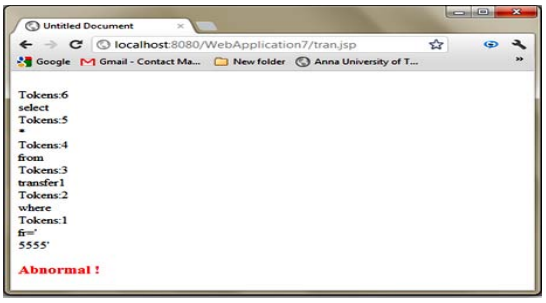

Fig. 13: Piggy backed injected query blocked

Select * from tranfer1 where fr = ' +num+'; When user type account number information as '5555' then the query at runtime become "Select name from tranfer1 where fr = '5555 '; ", then its tokenized and compare with static DFA Model.

In Fig. 10 view fund tranfer page is not injected with SQL injection and query is compared with DFA structure and result as normal and display transfer details in Fig. 11.

Validation: In validation step dynamic query is parsed against static DFA Query Model. If any malicious SQL input code given to the system then the dynamic query will not be validated against static DFA Query Model and hence this abnormal query is reported as SQL Injection Attack (SQLIA) and blocked accessing from database.

Since, the Fig. 12 view transfer details page is injected using Piggy backed injection the queries are compared with DFA structure of the query and query stated as abnormal and blocked from accessing to database as in Fig. 13.

EXPERIMENT AND EVALUATION

The proposed system simulated by implementing online Banking application which is known to be SQLIA vulnerable (Table 1). The application have been deployed on glassfish server with MySQL as database. The users of application are classified into ethical and un-ethical users using their past queries by applying the classification algorithm where the patterns are stored in the database. After getting the results of classification, only the ethical users are allowed to access the next query and remaining queries are validated further using static and dynamic analysis. Validation function parse Dynamic Query Model against Static Query Model. If validation functions returns true then the query is allowed to access the database else query is reported as SQL injection and block accessing from the database.

From the above results by enhancing static and dynamic analysis technique it is possible to detect all types SQL injection attacks in web application.

Comparative analysis: In this study, researchers note down how various schemes researchers against the identified SQL injection attacks and compare with the proposed research. Table 2 shows the comparative analysis of the SQL injections prevention techniques and the attack types. The symbol "•" is used to denote that techniques possible to stop successfully all attacks of that type.

Table 1: Results of SQLIA prevention and detection

Application	No. of SQL queries	Average No. of tokens per query	No. SQL injection queries	Prevention	Detection
Online banking application	758	13	68	90%	99%

Table 2: Comparison of SQLIA detection and prevention techniques with respect to attack types

Detection and prevention techniques	Tautologies	Piggy-backed query	Logically incorrect	Union query	Storedprocedure
AMNESIA (Halfond and Orso, 2005a, b)	●	●	●	●	×
CANDID (Bisht *et al.*, 2010)	o	o	o	o	o
DIWeDa (Roichman and Gudes, 2008)	×	×	×	×	×
JDBC Checker (Gould *et al.*, 2004)	o	o	o	o	o
IDS (Valeur *et al.*, 2005)	o	o	o	o	o
Tautology checker (Wassermann and Su, 2004)	●	×	×	×	×
SAFELI (Fu *et al.*, 2007)	×	●	●	●	●
SecuriFly (Martin *et al.*, 2005)	o	o	o	o	o
Security gateway (Scott and Sharp, 2002)	o	o	o	o	o
SQLIPA (Ali *et al.*, 2009)	●	×	×	×	×
SQLCheck (Su and Wassermann, 2006)	●	●	●	●	×
SQLGuard (Buehrer *et al.*, 2005)	●	●	●	●	×
SQLrand (Boyd and Keromytis, 2004)	●	×	●	●	×
Swaddler (Cova *et al.*, 2007)	o	o	o	o	o
WAVES (Huang *et al.*, 2005)	o	o	o	o	o
Proposed method	●	●	●	●	●

●: Possible, o: Partially possible ×: Impossible

The symbol "×" is used to denote that technique is not able to stop attacks of that type. The symbol "o" is used for technique that stop the attack type only partially because of limitations of the underlying approach. Even though many techniques are used for detection or prevention techniques, only some of them were implemented in practicality. Hence, this comparison is not based on empirical experience but rather it is an analytical evaluation.

CONCLUSION

Most of the web applications employs intermediate layer to accept a request from the user and retrieve data from the database. Most of the time scripting language used to build intermediate layer. SQL injection attack is common hacking techniques used by hacker to attack back end of web application. Generally attacker tries to confuse the intermediate layer technology by reshaping the SQL queries. A number of methods are used to avoid SQL injection attack at application level but only some of them were implemented in practicality. In this proposed research, the SQL injection attack will be prevented by classifying users as ethical and unethical and by comparing static and dynamic analysis the SQL injection attack will be detected. In static analysis, the DFA structure of the query at compile time will be analyzed where as in dynamic DFA structure of the query at run time is analyzed.

REFERENCES

Ali, S., S.K. Shahzad and H. Javed, 2009. SQLIPA: An authentication mechanism against SQL injection. Eur. J. Sci. Res., 38: 604-611.

Bisht, P., P. Madhusudan and V.N. Venkatakrishnan, 2010. Candid: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. ACM Trans. Inf. Syst. Security, 5: 1-39.

Boyd, S.W. and A.D. Keromytis, 2004. SQLrand: Preventing SQL injection attacks. Proceedings of the 2nd Applied Cryptography and Network Security Conference, June 8-11, 2004, Yellow Mountain, China, pp: 292-302.

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.

Cova, M., D. Balzarotti, V. Felmetsger and G. Vigna, 2007. Swaddler: An approach for the anomaly-based detection of state violations in web applications. Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection, September 5-7, 2007, Gold Goast, Australia, pp: 63-86.

Firdos, M. and A. Sheikh, 2011. Secure query processing by blocking SQL injection attack (SQLIA). Int. J. Res. Manage., Vol. 3.

Fu, X., X. Lu, P. Verger, B.S. Chen, K. Qian and L. Tao, 2007. A static analysis framework for detecting SQL injection vulnerabilities. Proceedings of the IEEE Annual International Computer Software and Application Conference, Volume 1, July 24-27, 2007, Beijing, pp: 87-96.

Gould, C., Z. Su and P. Devanbu, 2004. JDBC checker: A static analysis tool for QL/JDBC applications. Proceedings of the 26th International Conference on Software Engineering, May 23-28, 2004, Davis, CA., USA., pp: 697-698.

Halfond, W.G. and A. Orso, 2005a. Combining static analysis and runtime monitoring to counter SQL-injection attacks. Proceedings of the 3rd International ICSE Workshop on Dynamic Analysis, May 2005, St. Louis, MO., USA., pp: 22-28.

Halfond, W.G. and A. Orso, 2005b. Amnesia: Analysis and monitoring for neutralizing SQL-injection attacks. Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, November 7-11, 2005, Long Beach, CA., USA., pp: 174-183.

Howard, M. and D. LeBlanc, 2003. Writing Secure Code. 2nd Edn., Microsoft Press, New York USA., ISBN-13: 9780735617223, pp: 768.

Huang, Y.W., C.H. Tsai, T.P. Lin, S.K. Huang, D.T. Lee and S.Y. Kuo, 2005. A testing framework for Web application security assessment. Comput. Networks, 48: 739-761.

Huang, Y.W., F. Yu, C. Hang, C.H. Tsai, D.T. Lee and S.Y. Kuo, 2004. Securing web application code by static analysis and runtime protection. Proceedings of the 13th International Conference on World Wide Web, May 17-20, 2004, ACM, New York, USA., pp: 40-52.

Kiani, M., A. Clark and G. Mohay, 2008. Evaluation of anomaly based character distribution models in the detection of SQL injection attacks. Proceedings of the 3rd International Conference on Availability, Reliability and Security, March 4-7, 2008, Barcelona, pp: 47-55.

Lee, I., S.J.S. Yeoc and J. Moond, 2011. A novel method for SQL injection attack detection based on removing SQL query attribute. J. Math. Comput. Mod., 55: 58-68.

Martin, M., B. Livshits and M.S. Lam, 2005. Finding application errors and security flaws using PQL: A program query language. Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications, Volume 40, October 16-20, 2005, San Diego, CA, USA., pp: 365-383.

Roichman, A. and E. Gudes, 2008. DIWeDa-detecting intrusions in web databases. Proceedings of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security London, July 13-16, 2008, UK., pp: 313-329.

Scott, D. and R. Sharp, 2002. Abstracting application-level web security. Proceedings of the 11th International Conference on the World Wide Web, May 7-11, 2002, Honolulu, Hawaii, USA., pp: 396-407.

Su, Z. and G. Wassermann, 2006. The essence of command injection attacks in web applications. Proceedings of the 33rd ACM Symposium on Principles of Programming Languages, January 11-13, 2006, Charleston, South Carolina, USA., pp: 372-382.

Tajpour, A., S. Ibrahim and M. Masrom, 2011. SQL injection detection and prevention techniques. Int. J. Adv. Comput. Technol., 3: 82-91.

Valeur, F., D. Mutz and G. Vigna, 2005. A learning-based approach to the detection of SQL attacks. Proceedings of the 2nd International Conference on Detection of Intrusions and Malware and Vulnerability Assessment, July 2005, Vienna, Austria, pp: 123-140.

WHID, 2010. Report from July December 2010. Trust Wave Holdings.

Wassermann, G. and Z. Su, 2004. An analysis framework for security in web applications. Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems, October 2004, Atlanta, GA., pp: 70-78.