ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 1, Issue 8, October 2012

Preventing SQL Injection Attacks Using Combinatorial Approach

Dimple D. Raikar[1], Sharada Kulkarni[2],Padma Dandannavar.[3]
[1,2,3]Department of Computer Science and Engineering,
KLS Gogte Instituite of technology,Belgaum,Karnataka, India.

Abstract- SQL injection is a hazard to web applications, which gives attackers unrestricted access to the database. The attacker thereby can supply malicious or corrupted input that compromises an application. This leads to destruction of the security and the privacy of the users. To avoid this scenario, combinatorial approach is used for protecting Web application against SQL injection. Novel methods of protecting the web applications against the SQL injection are available such as positive tainting, signature based etc. Here Web Application SQL-injection Preventer (WASP) tool is used to prevent the SQL injection. Further the WASP tool is modified by implementing the proposed approach which is an enhancement to this tool.

Keywords- Positive Tainting, Character Level Tainting, Syntax Aware Evaluation.

1. INTRODUCTION

Application of web has become one of the easiest means of communication between the service provider and the clients. Instances of more script kiddies and sophisticated hackers target victims for fun, commercial reasons or personal reasons have been taking place. Attacks can bypass the security mechanism such as firewall, cryptography and traditional intrusion detection systems. The personal or professional loss caused by these, has created awareness in this system.

Web application security has become one of the most important measures in day to day life and most of the organizations concentrate on this aspect. The security of data involves several threats and hence organizations involved in security of web application have faced major challenges amongst which protecting the data from corrupting, and malicious access are few among many.

Program developers show keen interest in developing the application with usability rather than cost and incorporating security policy rules. An attacker makes use of this vulnerability in the application to get the unauthorized access to the data or information.

Input validation is one of the issues which we are addressing in this project, where attacker finds that an application makes unfounded assumptions about the type, length, format, or range of input data. The attacker can then supply a malicious input that compromises an application. When a network and host level entry points are fully secured; the public interfaces exposed by an application become the only source of attack. It could be very dangerous in many cases depending on the platform where the attack is launched.

1.1 Background Information

Structured Query Language (SQL) injection is a technique to attack the database. Here the exploitation of security is done in a way wherein the attacker adds the SQL code to a web form input box and by doing so the attacker can gain access to database or can also make changes to the data.

There are types of SQL injection, by which attack can be performed [2]. Few are enlisted below:

- Tautologies:

The general goal of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The consequences of this attack depend on how the results of the query are used within the application. The most common usages are to bypass authentication pages and extract data. In this type of injection,an attacker exploits an injectable field that is used in a query's WHERE conditional. Transforming the conditional into a tautology causes all of the rows in the database table targeted by the query to be returned. In general, for a tautology-based attack to work, an attacker must consider not only the injectable parameters, but also the coding constructs that evaluate the query results. Typically, the attack is successful when the code either displays all of the returned records or performs some action if at least one record is returned. In this example attack, an attacker submits " ' or 1=1 - - " for the *login* input field (the input submitted for the other fields is irrelevant). The resulting query is:*SELECT accounts FROM users WHERE login='' or 1=1 -- AND pass='' AND pin=*

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology. The database uses the conditional as the basis for evaluating each row and deciding which ones to return to the application. Because the conditional is a tautology, the query evaluates to true for each row in the table and returns all of them.

- Union Query:

In union-query attacks, an attacker exploits a vulnerable parameter to change the data set returned for a given query. With this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer. Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query>. Because the attackers completely control the second/injected query, they can use that query to retrieve information from a specified table. The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the results of the injected second query. an attacker could inject the text "' UNION SELECT card No. from Credit Cards where acct No.=10032 - -" into the login field, which produces the following query: *SELECT accounts FROM users WHERE login='' UNION SELECT cardNo from CreditCards where acctNo=10032 -- AND pass='' AND pin=*

Assuming that there is no login equal to "", the original first query returns the null set, whereas the second query returns data from the "CreditCards" table. In this case, the database would return column "cardNo" for account "10032." The database takes the results of these two queries,unions them, and returns them to the application. In many applications, the effect of this operation is that the value for "cardNo" is displayed along with the account information.

- Using Comments:

SQL supports comments in queries. Most SQL implementations, such as T-SQL and PL/SQL use { { to indicate the start of a comment (although occasionally # is used).By injecting comment symbols, attackers can truncate SQL queries with little effort. For example, SELECT * FROM users WHERE username='greg' AND password='secret' can be altered to SELECT * FROM users WHERE username='admin' {{AND password=''. By merely supplying admin' {{as the username, the query is truncated, eliminating the password clause of the WHERE condition. Also, because the attacker can truncate the query, the tautology attacks presented earlier can be used without the

supplied value being the last part of the query. Thus attackers can create queries such as SELECT * FROM users WHERE username= 'anything' OR 1=1 {{AND password='irrelevant'.This is guaranteed to log the attacker in as the first record in the users table, often an administrator.

- Piggy-Backed Queries

In this attack type, an attacker tries to inject additional queries into the original query. We distinguish this type from others because, in this case, attackers are not trying to modify the original intended query; instead, they are trying to include new and distinct queries that "piggy-back" on the original query. As a result, the database receives multiple SQL queries. The first is the intended query which is executed as normal; the subsequent ones are the injected queries, which are executed in addition to the first. Vulnerability to this type of attack is often dependent on having a database configuration that allows multiple statements to be contained in a single string.If the attacker inputs "'; drop table users - -" into the *pass* field, the application generates the query:*SELECT accounts FROM users WHERE login='doe' AND pass=''; drop table users - - ' AND pin=123*

After completing the first query, the database would recognize the 1Stored procedures are routines stored in the database and run by the database engine. These procedures can be either user-defined procedures or procedures provided. These procedures can be either user-defined procedures or procedures provided by the database by default. query delimiter (";") and execute the injected second query. The result of executing the second query would be to drop table users, which would likely destroy valuable information. Other types of queries could insert new users into the database or execute stored procedures. Note that many databases do not require a special character to separate distinct queries, so simply scanning for a query separator is not an effective way to prevent this type of attack.

1.2 Working system of SQL

Login page is the most complicated web application which allows users to enter into the database after authenticating him. In this page, the user provides his identity like username and password. There might be some invalid input validations which can bypass the authentication process using some mechanism like SQL injection. Normally, web applications is a three tier architecture, (i) the application tier at the user side, (ii) middle tier which converts the user queries into the SQL format, and the (iii) backend database server which stores the user data as well as the user's authentication table. Whenever a user wants to enter into the web database through application tier, the user inputs his/her authentication information from a login form as shown in Figure 1.

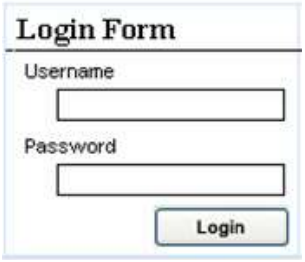

Fig 1: Login Form

The middle tier server will convert the input values of user name and password from user entry form into the format as shown in example.

Example of simple SQL query

If result of the query is true then the user is authenticated otherwise, denied. But, there are some malicious attacks which can deceive the database server by entering malicious code through SQL injection which always return true results

of the authentication query. For example the hacker enters the expression in the Username field like " ' OR 1=1 – – ' ". So, the middle tier will convert it into SQL query format as shown in example. This deceives the authentication server.

Example of SQL query having input violation

```
Query_result = "SELECT * FROM User_account WHERE name = '' OR 1=1 – – '' AND password= 'Password'
```

Analyzing the above query, the result is always true for variable Query_result. It is because malicious code has been used in the query. Here, in this query the mark (') tells the SQL parser that the user name string is finished and " OR 1=1 " statement is appended to the statement which always results in true. The (– –) is comment mark in the SQL which tells the parser that the statement is finished and the password will not be checked. So, the result of the whole query will return true for Query_result variable which authenticates the user without checking password.

1.3 Threats caused due to SQL injection

SQL injection attack is a major threat in the web application. SQL injection attacks breach the database mechanism such as integration, authentication, availability and authorization. SQL injection attack involves placing SQL statements in the user input for corrupting or accessing the database. SQL injection attacks can bypass the security mechanism such as firewall, cryptography and traditional intrusion detection systems. SQL injection can be performed very easily; even the application developer can perform the task very easily. The basic idea behind this attack is that the malicious user counterfeits the data that a web application sends to the database aiming at the modification of the SQL query that will be executed by the DBMS software. Technologies vulnerable to the SQL injection are ASP, ASP.net, PHP, CGI etc. All types of database have been severely vulnerable in such type of SQL injection attacks. Keeping this in vision we came with the technique which deals with identification and prevention of the SQL injection attack.

Attackers may even use such type of attack to get confidential information that is related to the national security of a country. Hence, SQL Injection could be very dangerous in many cases depending on the platform where the attack is launched and it gets success in injecting rogue users to the target system. Sometimes such type of collapse of a system can threaten the existence of a company or a bank or an industry. If it happens against the information system of a hospital, the private information of the patients may be leaked out which could threaten their reputation or may be a case of defamation. The main reason contributing to the successful SQLIAs is due to bad web application design and implementation.

2. SYSTEM DESIGN

The Proposed work aims for detecting and preventing SQL Injection attack. This approach is based on following steps:

Step 1: The user invokes the services provided by the application server using a browser. The input provided by the user is usually sent to the application server in the form of a parameter string.

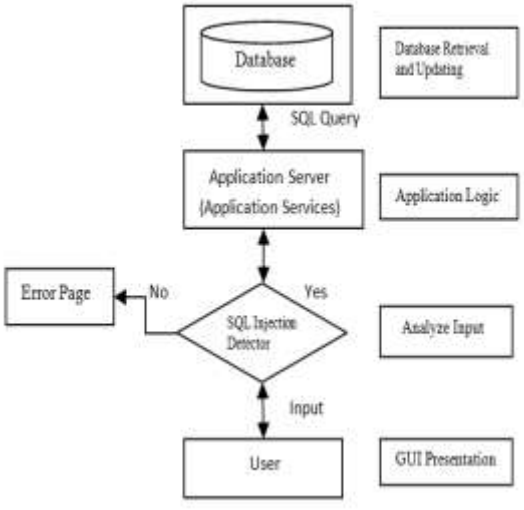

Fig 2: Block diagram of the proposed Approach

Step 2: SQL injection detector is positioned between the user and application server. The SQL injection detector intercepts the input from the user, parses it into SQL Meta character tokens.

Step 3: If the input from the user contains any attack signature then the injected input is treated as an attack and an error page is displayed otherwise the input is processed by the application server normally. Therefore, SQL injection detector checks for the presence of SQL Meta characters before the input is processed by the application server.

Step 4: The application server uses this input to generate a SQL query to retrieve information from the database or update it.

Step 5: Therefore, the proposed solution will not block legal inputs.

2.1 SQL Injection Detection Techniques

By using these techniques we can find the injection data and send the correct query to the SQL server. This technique is a sub-part of WASP, which can be effectively used [4].

2.1.1 Syntax-Aware Evaluation

This technique performs syntax-aware evaluation of a query string immediately before the string is sent to the database to be executed. To evaluate the query string, the technique first uses a SQL parser to break the string into a sequence of tokens that correspond to SQL keywords, operators and literals. The technique then iterates through the tokens and checks whether tokens contain only trusted data. If all such tokens pass this check, the query is considered safe and is allowed to execute. If an attack is detected, a specified action can be invoked. The key feature of syntax aware evaluation is that it considers the context in which trusted data is used to make sure that all parts of a query other than string or numeric literals (for example, SQL keywords and operators) consist only of trusted characters. Conversely, if this property is not satisfied for example, if a SQL operator contains characters that are not marked as trusted we can assume that the operator has been injected by an attacker and identify the query as an attack [4].

2.1.2 Positive Tainting

This technique is based on the identification, marking, and tracking of trusted data. The keywords are said to be trusted data, example, SELECT, FROM. To implement this positive tainting WASP must be able to identify and mark trusted strings. There are strings such as hard-coded strings. Here the hard-coded strings are identified in an application's bytecode. In Java, hard-coded strings are represented using String objects that are automatically created by the Java Virtual Machine (JVM) when string literals are loaded onto the stack. The JVM is a stack-based interpreter. Therefore, to identify hard-coded strings, WASP simply scans the bytecode and identifies all load instructions whose operand is a string constant. WASP then instruments the code by adding, after each of these load instructions, code that creates an instance of a MetaString class by using the hard-coded string as an initialization parameter. Finally, because hard-coded strings are completely trusted, WASP adds to the code a call to the method of the newly created MetaString object that marks all characters as trusted [4].

2.1.3 Character Level Tainting

In character level tainting, taint information is tracked at the character level rather than at the string level. This is done because, for building SQL queries, strings are constantly broken into substrings, manipulated, and combined. By associating taint information to single characters, this approach can precisely model the effect of these string operations.

This project proposes a new approach for dynamic detection and prevention of SQLIAs. Our approach works by identifying "trusted" strings in an application and allowing only these trusted strings to be used to create the semantically relevant parts of a SQL query such as keywords or operators. The general mechanism used to implement this approach is based on dynamic tainting, which marks and tracks certain data in a program at run time. The kind of dynamic tainting proposed here has several important advantages over techniques based on other mechanisms.

Our approach provides specific mechanisms for detecting false positives, identify their sources, and easily eliminate them in future runs by tagging the identified sources as trusted. The second conceptual advantage of our approach is the use of flexible syntax-aware evaluation. Syntax-aware evaluation lets us address security problems that are derived from mixing data and code while still allowing for this mixing to occur. It provides a mechanism for regulating the usage of string data based not only on its source but also on its syntactical role in a query string. In this way a wide range of external input sources are being provided to build queries while protecting the application from possible attacks introduced via these sources[4].

2.1.4 Algorithm

Input: User input.

Output: Detecting and Preventing SQL Injection Attack.

Step 1: Identify the hotspot

Step 2: Hirschberg Algorithm uses Divide and Conquer methodology for comparison of tokens.

Step 2a: It maintains a table that contains keywords which are present in horizontal and vertical line.

Step 2b: It will compare the incoming tokens with the predefined values for identity.

Step 2c: The hotspot which has been identified by the analyzer module is sent to this table for detecting and preventing SQL Injection Attack.

Step 3: The algorithm divides the token and it checks each token with the predefined tokens using divide and Conquer methodology.

Step 4: Therefore Analyzer Module detects SQL Injection taken place in order to prevent SQL Injection Attack.

3. IMPLEMENTATION

Implementation is carried out using a tool called WASP (Web Application SQL-injection Preventer) which is written in Java. The proposed system is implemented for Java-based Web applications. It is a tool used to avoid SQL Injection. WASP works by identifying "trusted" strings in an application and allowing only these trusted strings to be used to create the semantically relevant parts of SQL query, such as keywords or operators.

Data Flow Diagram

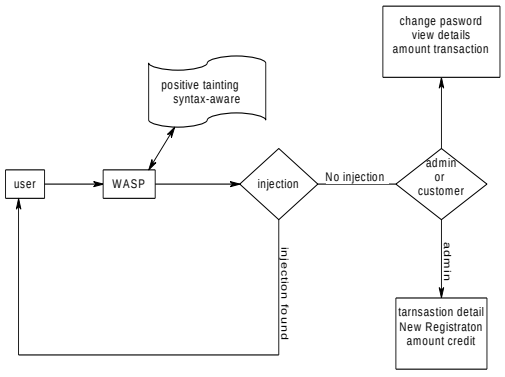

Fig 3: Data flow diagram

Here the user interacts with a Web form that takes account number and password as input and submits them to a Web server. It uses input parameters such as account number and password to dynamically build an SQL query. The account number and password are checked against the credentials stored in the database. If they match, the corresponding user's account information is returned. Otherwise, the authentication fails. The authenticated user can view the respective details, change the password, amount transaction and the administrator can view the transaction detail, new registration and the amount credit. If the attacker tries to perform SQL Injection, then the WASP tries to block the attack and the attacker cannot proceed further. Hence by using this tool most of the attacks can be blocked successfully.

4. RESULTS and CONCLUSION

This work presents a new approach for protecting Web applications (banking application) from SQLIAs. The approach consists of Identifying hotspot from the application and trusted data sources and marking data coming from these sources as trusted, allowing only trusted data to form the semantically relevant parts of queries such as SQL keywords and operators.

This approach also provides practical advantages over the many existing techniques whose application requires customized and complex runtime environments: It is defined at the application level, requires no modification of the runtime system, and imposes a low execution overhead.

REFERENCES

[1] W. G. Halfond and A. Orso, "Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks", In Proceedings of the Third International ICSE Workshop on Dynamic Analysis (WODA 2005), pages 22–28, St. Louis, MO, USA, May 2005.

[2] Kim H.K, "Frameworks for SQL Retrieval on Web Application Security", In Proceedings of the International Multi Conference of Engineering and Computer Society (IMECS), Vol. 1, March 17-19, Hong Kong , 2010, 6 pp.

[3] W. Halfond, A. Orso, and P. Manolios, "Using Positive Tainting and Syntax-Aware Evaluation to Counter SQL Injection Attacks", Proc. ACM SIGSOFT Symp. Foundations of Software Eng., pp. 175-185, Nov. 2006.

[4] William G.J. Halfond, Alessandro Orso,Panagiotis Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation", IEEE Transaction of Software Engineering Vol 34, Nol, January/February 2008.

[5] R. Ezumalai, G. A, "Combinatorial Approach for Preventing SQL Injection Attacks", IEEE International Advance Computing Conference (IACC 2009). Patiala, India: pp.1212-1217, 2009.

AUTHOR PROFILES:

Dimple D. Raikar Received the B.E in (Computer Science & Engineering) from K.L.E college of Engineering and pursuing M.Tech in (Computer Science & Engineering) from Gogte Institute of Technology, Belgaum, Affiliated to Visvesvaraya Technological University.

Sharada Kulkarni Received the M.Tech in (Computer Science & Engineering) from Visvesvaraya Technological University. She is working as an Assistant Professor in Gogte Institute of Technology. She has got 16 years of academic experience. She has published papers at international, national conferences.

Padma Dandannavar Received the M.Tech in (Computer Science & Engineering) from SDMCET Dharwad, Affliated to Visvesvaraya Technological University,Belgaum. She is working as an Assistant Professor in Gogte Institute of Technology. She has got 2 years of Industrial Experience and 11 years of academic experience. She has published papers at international, national conferences.