Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# Enhanced Security Model for SQL Injection attacks for web Database

**Deepti U Telang**
Research Scholar, Dept.of CSE,
NMAMIT, Nitte India,

**Prasanna Kumar H.R**
HOD, Department of CSE
NMAMIT, Nitte India

***Abstract— Information security is a critical element of business operations. The SQL Injection attack is the attack to the web application to have abused the vulnerability which is called SQL Injection. SQL injection is a technique often used to attack a website. The SQL Injection attack poses a serious threat to the security of web applications because SQL is used to operate the database in a lot of applications. The SQL Injection attack is the method that executes any SQL sentence when using arbitrarily character for the search condition or the update contents. By generating SQL sentence and changing existing command, attackers can make data public which should be hidden, and interpolate.***

***Keywords— SQL Injection, Web Database, Protection, Threats, WASP***

## I. INTRODUCTION

SQL injection is a technique often used to attack a website. This is done by including some portions of SQL statements in a web form entry field in an attempt to get the website to pass a newly formed rogue SQL command to the database. SQL injection is a code injection technique. SQL injection attack risk is usually very high. A successful attack can bypass authentication and authorization to gain full control of the database. It steals sensitive data, change users passwords, retrieve users credential information, add non-existent accounts, drop tables, make illegal financial transactions, and destroy the existing database software etc [1]. The structure of the SQL Injection attack is relatively simple if the attack is not encoded. It may be able to defend the new web application introduced in recent year by sanitizing or restricting the input. The damage by the SQL Injection attack keeps rising because it is difficult to maintain the web application which has been already introduced. The SQL Injection attack is the method that executes any SQL sentence when using arbitrarily character for the search condition or to update contents. By generating SQL sentence and changing existing command, attackers can make data public which should be hidden, and interpolate.

The vulnerability happens, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection attack. Methodology targets the data residing in a database through the firewall that shields it. The attack takes advantage of poor input validation in code and website administration. It also occurs when an attacker is able to insert a series of SQL statements into a query by manipulating user input data into a web-based application. The attacker can take advantages of web application programming security flaws and pass unexpected malicious SQL statements through a web application for execution. [2]

## II. SECURITY THREATS

There is a way of introducing the web application firewalls to defend web application against this attack. The web application firewall employs the blacklist method which has accumulated an attack in the past and SQL Injection attack is becoming clever and complicated. It is becoming difficult to detect the SQL Injection attack. The blacklist of the web application firewall must also be updated when a new attack is developed. However, it is not difficult to camouflage the SQL Injection attack by changing encoding setting. Therefore, it is not easy to detect unknown attack using conventional method and to classify the pattern of SQL Injection attack completely. [3]

### *A. Denial of Service and other dangers:*

Denial-of-service attacks involve crashing a Web server or database or locking it into an endless process that consumes all system resources. The result is the inability to provide a response to Website browser requests or even to core database requests. Firewall is one way to protect server.

### *B. DB2 Security:*

When analysing DB2 database, there are several areas in the database layer where one should validate security. In DB2, the authentication type is used to identify users, it defines where and how authentication occurs. Several authentication types are available, which one to use should be carefully determined by the environment, operating system, and purpose of the DB2 server. The authentication type is configured on both the client and server. However, authentication type is defined only on the server in the database manager configuration file. The configuration file is associated with an instance and applies to all databases with that instance as well as all users within the database.

 

## III. PROTECTION FROM SQL INJECTION

There are two primary methods to protect database from SQL injection.

1).Applications validates user input by blocking invalid characters. In many cases, only alphanumeric Characters should be accepted. At minimum, single quotes should be blocked.

2).Use protected queries that bind variables rather than combining SQL statements together as strings such as stored procedures.

The simplest way to find out for vulnerable to an SQL injection attack is to enter a single quote into each field on each form in applications and verify the results. Some applications will return a message claiming a syntax error. Some applications will catch the error and not report anything [4]

*A. Protection Process:*

A hacker will use SQL injection to modify a query in order to send (via the application) commands to the database that the developer never intended. There is following protection process.

***1) Control dynamic SQL***: Ensure that dynamic SQL is not generated into the source code especially based on user controlled inputs. This is very common way of Cross Site Scripting attack.

***2) Use parameterized queries***: Most of platforms like .Net and Java supports parameterized queries to generate dynamic SQL.

3***) Use database functions/stored procedures***: Using Stored Procedures, parameters for using with SQL queries is very straight forward and easy to use. Dynamic SQL queries should only be used in application code when something is not feasible with Stored Procedures.

***4) Use web application firewall*:** It is used to create protection shield without any change in existing applications.

***5) Use least privilege access*:** Never use system admin user accounts to provide connectivity between application code and database. Create different account groups for application connectivity with least privilege access policy. In case any part of implementation need higher privilege, separate such deployment and use a higher privilege account that should be used by limited application code.

***6) Use server side triggers*:** Database triggers are mostly avoided due to performance reasons. But Triggers may provide good control to restrict mass changes/deletion for highly critical data tables.

***7) Exception handling*:** The application does not crash and expose critical information like database structure or configuration details due to an unhandled exception. Take measures to protect from blind SQL Injection when an attacker does not get desired data by SQL Injection attack but use the exception incidents to validate his guess/assumptions about system.

***8) Avoid obvious names*:** For successful SQL Injection, many times, attacker needs to guess the database structure. Hence avoid very obvious names in database structure for sensitive data.

***9) Validate user inputs***: User input validation is required not only to prevent SQL Injection but also many other critical attacks like Cross Site Scripting. Validate data for type, length, format, and range. Also check parameter/inputs for such characters that have special meaning for database platform such as the single quote character in SQL Server. Test the content of string variables and accept only expected values. Reject entries that contain binary data, escape sequences, and comment characters. Implement multiple layers of validation. A better practice is to validate input in the user interface and at all subsequent points where it crosses a trust boundary. When working with XML documents, validate all data against its schema as it is entered.[5]

There are following steps for SQL protection for web database.

- Use original names for tables and columns to make the names harder to guess.
- Set length limits on form fields and validate data for content length and format.
- Keep up-to-date on patches.
- Make schema unique.
- Lock down server.
- Make sure the application is running with the minimal rights necessary to complete its task.
- Remove any unnecessary accounts, features and stored procedures.

## IV. PROPOSED SOLUTION

To evaluate SQL injection, a prototype tool called WASP Web Application SQL Injection Preventer that is written in Java and implements technique for Java-based Web applications. This proposed model WASP is based on dynamic tainting, which has been widely used to address security problems related to input validation. Traditional dynamic tainting approaches mark certain untrusted data typically, user input as tainted, track the flow of tainted data at runtime, and prevent this data from being used in potentially harmful ways. Second, WASP performs accurate taint propagation by precisely tracking trust markings at the character level. Third, it performs syntax-aware evaluation of query strings before they are sent to the database and blocks all queries whose non-literal parts SQL keywords and operators, contain one or more characters without trust markings.[6]

*A. Positive tainting:*

Based on the identification, marking, and tracking of trusted, rather than untrusted, In case of negative tainting, incompleteness leads to trusting data that should not be trusted and, ultimately leads false negatives. Incompleteness may

 

thus leave the application vulnerable to attacks and can be very difficult to detect even after attacks occur. With positive tainting, incompleteness may lead to false positives, but never results in an SQLIA escaping detection. This security model, accurately and automatically identifies all SQLIAs and generates no false positives; our basic approach, automatically marks as trusted all hard-coded strings in the code and then ensures that all SQL keywords and operators are built using trusted data. Taint propagation consists of tracking taint markings associated with the data while the data is used and manipulated at runtime. When tainting is used for security-related applications, it is especially important for the propagation to be accurate. It accurately mark and propagate taint information by (1) tracking taint markings at a low level of granularity and (2) precisely accounting for the effect of functions that operate on the tainted data. [7]

B. *Character-level tainting*:
We track taint information at the character level rather than at the string level. Building SQL queries, strings are constantly broken into substrings, manipulated, and combined. By associating taint information to single characters, it can precisely model the effect of these string operations. To accurately maintain character-level taint information, we must identify all relevant string operations and account for their effect on the taint markings. [9]

*C. Syntax- aware evaluation:*
Syntax-aware evaluation uses the function of sanitizing. A sanitizing function is typically filters that performs operations such as regular expression matching or declassification is based on the assumption that sanitizing functions are able to eliminate or neutralize harmful parts of the input-string replacement The key feature of syntax-aware evaluation is that it considers the context in which trusted and untrusted data is used to make sure that all parts of a query other than string or numeric literals (e.g., SQL keywords and operators) consist only of trusted characters WASP 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 an 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 (i.e., substrings) other than literals contain only trusted data. If all of the tokens pass this check, the query is considered safe and allowed to execute. [10]

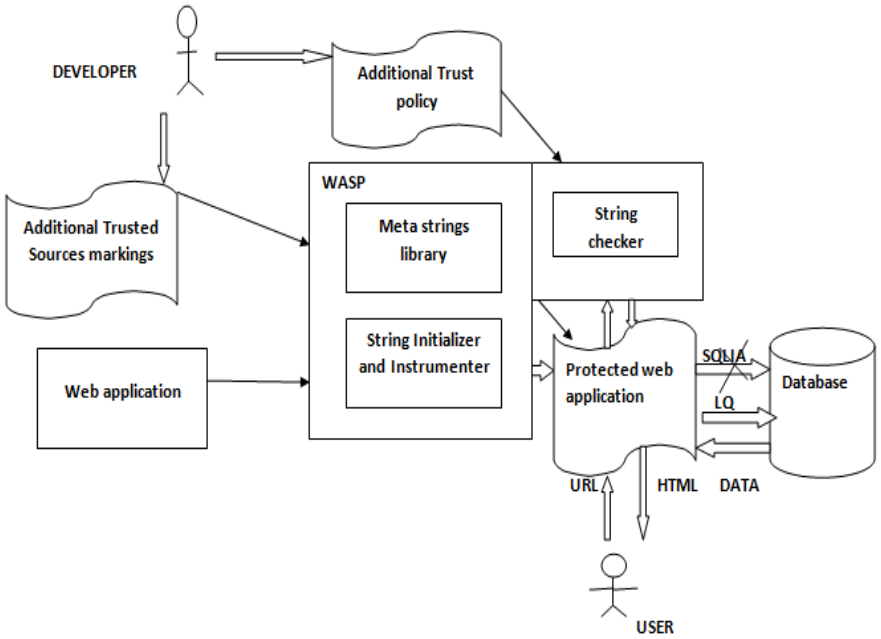


**Fig 1.High-Level overview of WASP**

## V. IMPLEMENTATION

WASP consists of a library Meta-Strings and two core modules, String Initializer and Instrumenter and String checker. The Meta-Strings library provides functionality for assigning trust markings to strings and precisely propagating the markings at runtime. Module string initializer and instrumented instruments Web applications to enable the use of the MetaStrings library and add calls to the string checker module. Module String checker performs syntax-aware evaluation of query strings right before the strings are sent to the database. [12]

A. *The Meta String library*:
It is library classes of Java's standard string classes Strings Library. Meta-Strings classes also provide methods for setting and querying the metadata associated with the use of Meta-Strings. The Meta-Strings class String, has the same internal representation, and provides the same methods. Meta-String also contains additional data structures for storing metadata and associating the metadata with characters in the string. Each method of class Meta-String overrides the corresponding method in String, class that corresponds to Java's String class. Meta-String extends class String, has the

 

same internal representation, and provides the same methods. Meta-String also contains additional data structures for storing metadata and associating the metadata with characters in the string. Each method of class Meta-String overrides the corresponding method in String, providing the same functionality as the original method, but also updating the metadata based on the method's semantics. It has the following benefits of a string's characters. • It allows for associating trust markings at the granularity level of single characters.

• It accurately maintains and propagates trust markings.

• It is defined completely at the application level and therefore does not require a customized runtime system.

• Its usage requires only minimal and automatically performed changes to the application's byte code it Imposes a low execution overhead on the Web application. The main limitations of a string's characters the current implementation of the MetaStrings library is related to the handling of primitive types, native methods, and reflection.

• Meta-Strings cannot currently assign trust markings to primitive types.

*B. Initialization of Trusted Strings:*

To implement positive tainting, WASP must be able to identify and mark trusted strings. There are three categories of strings that are:

*1).Hard-Coded Strings:*

The identification of hard-coded strings in an application's byte code is a straightforward process. In Java, hard-coded strings are represented using String objects that are created automatically by the Java Virtualization Machine (JVM) when string literals are loaded onto the stack, To identify hard-coded strings, WASP simply scans the byte code and identifies all load instructions whose operand is a string constant. WASP then instruments the code by adding, after each of these load instructions. Finally, hard-coded strings are completely trusted, WASP adds to the code a call to the method of the newly created Meta-String object that marks all characters as trusted .the method of the newly created Meta-String object that marks all characters as trusted at runtime, polymorphism and dynamic binding allow this instance of the MetaString object to be used in any place where the original String object would have been used.

*2).Implicitly-Created Strings.*

In Java programs, the creation of some string objects is implicitly added to the byte code by the compiler. For example, Java compilers typically translate the string concatenation operator ("+") into a sequence of calls to the append method of a newly-created String Builder object. WASP must replace these string objects with their corresponding Meta-Strings objects so that they can maintain and propagate the trust markings of the strings on which they operate. WASP scans the byte code for instructions that create new instances of the string classes used to perform string manipulation and modifies each such instruction so that it creates an instance of the corresponding Meta-Strings class instead. WASP does not associate any trust markings with the newly-created Meta-Strings objects. These objects are not trusted and they become marked .the string concatenation operator ("+"). WASP replaces the instantiation at offset 28 with the instantiation of a MetaStringBuilder class and then changes the subsequent invocation of the constructor at offset 37 so that it matches the newly instantiated class. Because MetaStringBuilder extends String Builder, the subsequent calls to the append method invoke the correct method in the MetaStringBuilder class.

*3).Strings from External Sources:*

To use query fragments coming from external (trusted) sources, WASP uses the information in the configuration file to instrument the external trusted sources according to their type; WASP scans the byte code for all instantiations of new file objects and adds instrumentation that checks the name of the file being accessed. WASP also instruments all calls to methods of file-stream objects that return strings, at runtime, the added code checks to see whether the object on which the method is called is in the list of currently trusted file objects.

*C. Syntax-Aware Evolution*

The STRING CHECKER module performs syntax-aware evaluation of query strings and is invoked right before the strings are sent to the database. To add calls to the STRING CHECKER module, WASP first identifies all of the database interaction points: points in the application where query strings are issued to an underlying database are Evaluation. WASP inserts a call to the syntax-aware evaluation function, immediately before each interaction point. Metachecker takes the MetaStrings object that contains the query about to be executed as a parameter. MetaChecker enforces the default trust policy by iterating through the tokens that correspond to keywords and operators and examining their trust markings. If any of these tokens contains characters that are not marked as trusted, the query is blocked and reported.

*1). Meta Checker*

Invokes the corresponding checking function(s) to ensure that the query complies with them. Trust policies are that they take the list of SQL tokens as input, perform some type of check on them based on their trust markings, and return a true or false value depending on the outcome of the check. [13]

## VI. FUTURE SCOPE

A new security trend is to provide multiple layers of security within a computing environment. These layers can include multiple firewalls between the Internet and the organization and even firewalls within an organization to protect high-value assets. Most databases are set up in a way that makes breaking in relatively easy. But securing the database has become simpler. One of the most common examples of exploiting this risk is known as SQL injection. SQL injection isn't a direct attack on the database. It takes advantage of the way many Web applications that access databases are developed. It attempts to modify the parameters passed to a Web application via a Web form to change the resulting SQL

 

statements that are passed to the database and compromise its security. If successful, an attacker can hijack the database server and be granted the same permissions to add, drop, and change users that the application has. From that point, the database is fully exposed.

## VII. CONCLUSION

Applications can be the poison of databases given their super-user privileges, especially when the applications aren't created with as much concern for security as is typical for databases. SQL injections are dangerous because they are a door wide open to hackers to enter system through Web interface and to do delete tables, modify databases, even get hold of corporate network. SQL injections are a programming error and they have nothing to do with web site hosting provider.

**References**

[1] Rahul johari, pankajsharma "A survey on web application vulnerabilities (SQLIA, XSS) security engine for SQL injection", 2012 international conference

[2] A. Tajpour; M. JorJor Zade Shooshtari, "Evaluation of SQL Injection Detection and Prevention Techniques," Proc. of CICSyN, 2010, pp.216-221, 28-30 July 2010

[3] A. Tajpour; M. Masrom; M. Z. Heydari.; S. Ibrahim; "SQL injection detection and prevention tools assessment," Proc. Of ICCSIT 2010, vol.9, no., pp.518-522, 9-11 July 2010

[4] M. Ruse, T. Sarkar and S. Basu, "Analysis & Detection of SQL Injection Vulnerabilities via Automatic Test Case Generation of Programs", 10th Annual International Symposium on Applications and the Internet pp. 31 – 37 (2010)

[5] S. Thomas, L. Williams, and T. Xie, "On automated prepared statement generation to remove SQL injection vulnerabilities", Information and Software Technology 51, 589–598 (2009).

[6] M. Junjin, "An Approach for SQL Injection Vulnerability Detection," Proc. of the 6th Int. Conf. on Information Technology: New Generations, Las Vegas, Nevada, pp. 1411-1414, April 2009.

[7] K. Amirtahmasebi, S. R. Jalalinia, S. Khadem, "A survey of SQL injection defense mechanisms," Proc. Of ICITST 2009, vol., no., pp.1-8, 9-12 Nov. 2009

[8] Y. Haixia, N. Zhihong, "A database security testing scheme of web application," Proc. of ICCSE '09, pp. 953-955, 25-28 July 2009.

[9] K. Kemalis, and T. Tzouramanis (2008). "SQL-IDS: A Specification-based Approach for SQLinjection Detection", SAC'08. Fortaleza, Ceará, Brazil, ACM: pp. 2153 2158.

[10] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao. "A Static Analysis Framework for Detecting SQL Injection Vulnerabilities", COMPSAC 2007, pp.87-96, 24-27 July 2007.

[11] W. G. Halfond, J. Viegas, and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures", In Proc. of the Intl. Symposium on Secure Software Engineering, Mar. 2006.

[12] R.A. McClure, and I.H. Kruger, "SQL DOM: compile time checking of dynamic SQL statements," Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on, pp. 88- 96, 15-21 May 2005.

[13] S. W. Boyd and A. D. Keromytis, "SQLrand: Preventing SQL Injection Attacks. In Proceedings of the 2nd Applied Cryptography and Network Security" Conference, pages 292–302, June 2004.

## ACRONYM

***i)*** ***SQL****: Structured query language.*
***i)*** ***WASP****: Web Application SQL Injection Preventer*
***ii)*** ***XML***:Extended mark of language
***iii)*** ***DB2***:Database2.

 