Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 30 (2012) 183 – 190

Procedia Engineering

www.elsevier.com/locate/procedia

International Conference on Communication Technology and System Design 2011

An Efficient Technique for Detection and Prevention of SQL Injection Attack using ASCII Based String Matching

Indrani Balasundaram[a], E. Ramaraj[b], a*

a Department of Computer Science, Madurai Kamaraj University, TN, INDIA
b Department of Communication, Madurai Kamaraj University, TN, INDIA

Abstract

With the rise of the Internet, web applications, such as online banking and web-based email, have become integral to many people's daily lives. Web applications have brought with them new classes of computer security vulnerabilities, such as SQL injection. It is a class of input validation based vulnerabilities. Typical uses of SQL injection leak confidential information from a database, by-pass authentication logic, or add unauthorized accounts to a database. This security prevents the unauthorized access to your database and also it prevents your data from being altered or deleted by users without the appropriate permissions. Malicious Text Detector, Constraint Validation, Query length validation and Text based Key Generator are the four types of filtration technique used to detect and prevent the SQL Injection Attacks from accessing the database

Keywords: SQL Injection Attack; Web applications; Web security; Data validation.

1. Introduction

Web applications enable much of today's online business includes banking, shopping, university admissions, email, social networking, and various governmental activities. They have become ubiquitous because users only need a web browser and an Internet connection to receive a system-independent interface to some dynamically generated content. No database is safe. An attack technique that has been widely used is Structured Query Language (SQL) Injection. SQL injection is a method for exploiting web applications that use client-supplied data in SQL queries. SQL Injection refers to the technique of inserting SQL meta-characters and commands into Web-based input fields in order to manipulate the execution of the back-end SQL queries [3]. The data web that applications handle, such as credit card numbers and product inventory data, Successful SQL injection attacks enable malicious users to execute

* Indrani Balasundaram. Tel.: +91-9865199366.
E-mail address: indrani.phd@gmail.com.

1877-7058 © 2011 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
doi:10.1016/j.proeng.2012.01.850

commands in an application's. In this paper, we introduce a novel technique for the detection and prevention of SQLIA' s. Current technique consist four types of validation model approach specifically designed to target SQLIA's. The key intuition behind our technique is: (1) malicious text detector is used to detect and prevent the wildcard, which is used by the attacker to attack the database. 2) Userid and Password have constraint in Login phase 3) Generating ASCII value in database for the sensitive data's as UseridASCII and PasswordASCII to secure the database from the attackers.

1.1. Techniques of SQL Injection Attacks

An SQL Injection Attack is a code injection technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability occurs if the user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or the user input is not strongly typed.

Tautology
A website uses this source (figure 2), which would be vulnerable to SQLIA. For example, if a user enters"' OR 1=1--" and"", instead of Userid ="Raja" and Password = "rani", the resulting query is:

SELECT * from FROM User_info WHERE userid='' OR 1=1 --' AND password =''

The database interprets everything after the WHERE token as a conditional statement, and the inclusion of the "OR 1=1" clause turns this conditional into a tautology. As a result, the database returns the records for all users in the database. An attacker could insert a wide range of SQL commands via this exploit, including commands to modify or destroy database tables.

Remote execution of stored procedures
This category of attack is conducted by executing the procedures, stored previously by the web application developer. Following is an example

"'; SHUTDOWN; -- "to any of the input field, the query generates as given below:

Query = "Select accounts from User_info WHERE Userid = ''; SHUTDOWN; -- password='';

Illegal/ Malicious Queries
Attackers gathers sufficient information from the error display on web application thus the attacker gain more information about the database schema , this helps the attacker by using SQL Injection Attack technique to access the web applications Labs.

2. Related work

Su and Wassermann [1] proposed a solution of static analysis of an SQL statements using parse tree validation, they used that static structure how to filter the user input and generate the input validation code. They conducted a study using five open-source Web application projects on GotoCode.com, the same five projects used by [3], and applied their wrapper, SQLCHECK, to each application. They found that their wrapper stopped all 18,424 SQLIA's in their attack suite without generating any false positives and it may not contain all possible attacks. The parse tree approach is effective in identifying the structure of the SQL statement and using structure comparisons to detect potential SQLIAs. However, the parse tree approach focuses on the structure of the attacks instead of the removal of the SQLIVs.
In [3] [6] [7] secure vulnerable SQL statements by combining static analysis with statement generation and runtime monitoring. Their solution, AMNESIA, analyzes a vulnerable SQL statement, generates a

generalized statement structure model for the statement, and allows or denies each statement based on how it compares to the model at runtime. Their solution throws an exception for each SQLIA, which the developer handles and builds in attack recovery logic. They found that their solution stopped all of the SQLIA' s in their attack set, a set ranging from 140 to 280 elements for each application, without generating any false positives. Their model generation and runtime comparison approach is effective at detecting SQLIA's and does a comparison similar to SQLGuard. AMNESIA, like SQLGuard, also adds a computational overhead by including an additional process that has to be integrated into the runtime environment. Additionally, AMNESIA adds the capability for developers to add logic to how SQLIA's are handled, by throwing an exception in the vulnerable code. Sun [8] proposed an effective approach for detecting and preventing SQL injection attacks. Author claims that the proposed technique satisfy the following points, (1) is resistant to evasion techniques, such as hexadecimal encoding or inline comment, (2) does not require analysis or modification of the application source code, (3) does not need training traces, (4) does not require modification of the runtime environment, such as PHP interpreter or JVM, and (5) is independent of the back-end database used.

2.1. Positive Tainting and Syntax Aware Evaluation

In this approach valid input strings are initially provided to the system for detection of SQLIA. At runtime, it categorizes input strings and propagates the untrusted or other-than-trusted markings based on the initialization. After that, a 'syntax aware evaluation' is performed for evaluating the propagated strings. Thus, based on the evaluation, if untrusted strings are found, such queries are restricted from passing into the database server for processing. During initialization of the trusted strings, it performs identification and marking based on inputs. The strings are categorized as: (i) hard coded strings, (ii) strings implicitly created by Java and (iii) strings originated from external sources. In case of syntax-aware evaluation, it performs syntax evaluation at the database interaction point. Syntax defines the trust policies which are the functions defined by the web programmer. Functions perform pattern matching and if the result of matching gives positive outcome, the tool allows the query to be executed on the database server. Following issues are there in this method:

(i) Initialization of trusted strings is developers dependent
(ii) Persistent storage of trusted strings may cause second order attack [2]

2.2. Prepare Statement

SQL provides the prepare statement, which separates the values in a query from the structure of SQL [4]. The programmer defines a skeleton of an SQL query and then fills in the holes of the skeleton at run time. The prepare statement makes it harder to inject SQL queries because the SQL structure cannot be changed. To use the prepare statement, we must modify the web application entirely; all the legacy web application must be re-written to reduce the possibility of SQL injections. Other approaches against SQLIA's rely purely on static analysis. Wassermann and Su [1] propose a technique that combines static analysis and automated reasoning to detect whether an application can generate queries that contain tautologies. This technique is limited, by definition, in the types of SQLIA's that it can detect.

3. Proposed Defensive mechanism in Static Level Validation

Current approach is the combination of static analysis with runtime validation. In the static analysis stage, the prevention technique represents in three level phases Malicious Text Detector**,** Field Constraint

Validation and Static Query Length Validation. In runtime validation stage, the user input data is validated with all these stages and results the user input as safe or unsafe.

3.1. Malicious Text Detector

1. Statically build a model for preventing Meta characters (figure 1)
2. Detect the susceptibility character which is appended with the user's data and prevent the malicious attacker from accessing the web application.

```
Function wildChars(ByVal strWords)
    Dim arr1 As New ArrayList
    arr1.Add("select")
    arr1.Add("--") arr1.Add("drop") arr1.Add(";") arr1.Add("insert") arr1.Add("delete")
    arr1.Add("xp_") arr1.Add("""")
    Dim i As Integer
    For i = 0 To arr1.Count - 1
        strWords = Replace(strWords, arr1.Item(i), "", , , CompareMethod.Text)
    Next
    Return strWords
End Function
```

Figure 1: Sample code for Malicious Text Detector

Sample 1 state that as malicious query, the user input data is pattern matched with malicious text detector and detects the wild card character (--); throw an exception as an SQLIA's.

Sample1: Select * from user_info where Userid ='INDIRA'- -'and password=1007

```
user_valid1 = obj1.stripQuotes(UCase('INDIRA'- -'))
user_valid2 = obj1.killChars('INDIRA'- -')
pass_valid1 = obj1.stripQuotes(0)
pass_valid2 = obj1.killChars(0)
corr_userName = 'INDIRA'
corr_password = 0
```

Field Constraint Validation

Login phase is set with constraint as User id is allowed only the ten characters; Password is allowed only four characters shown in (figure 2).

```
len_username = Len('INDIRA')  len_password = Len(0)
If len_username = 10 And len_password = 4 Then End If
```

Figure 2: Sample code for 'CONSTRAINT VALIDATION

4. Proposed SQLIA - Protector Mechanism

Because of high level security the proposed scheme consists of three filtration phases with static and dynamic level. First level phase is Malicious Text Detector, Second level phase is Constraint Identifier, Third level phase is Static Query Analysis and the Fourth level Phase is Text based Key Generator.

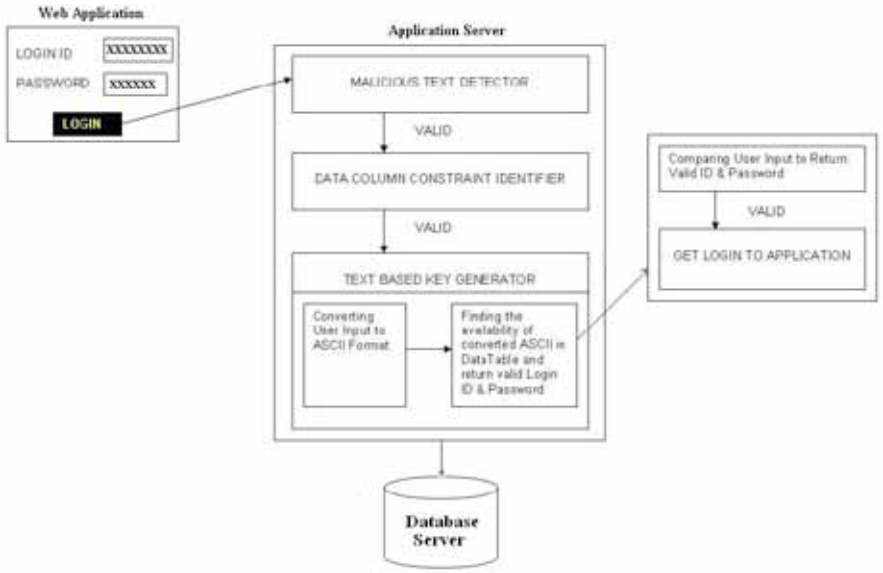

Figure 4: Proposed Architecture for ASCII based String Matching technique

4.1. Underlying Principle in Run time Monitoring

During runtime, the user input data's are compared against the corresponding filtration techniques used to check for their validity. The dynamically generated user inputs are not satisfied with all the three level of filtration technique then they are flagged as malevolent else legitimate user and allowed to access the web application.

4.2. Text based Key Generator

1. Converting User input in to ASCII code
2. Searching the availability of converted ASCII in Data table and returns valid Userid and Password,
3. Here four parameters are maintained in database server that is Userid, password, ASCII Userid and ASCII password (Table 1)

USERID	PASSWORD	USERID ASCII	PASSWORD ASCII
INDIRA	1007	7378688265	49484855
RANI	1008	82657873	49484856
ARUN	1009	65828578	49484857
JEGADISH	1011	7469716568738372	49484949

Table 1: ASCII conversion of Sensitive Data in Data Tables

This proposed technique is used to validate the user input with static and run time analysis to detect and prevent SQLIA' s. Figure 4 shows that the user input data enters in the Login phase from the web browser, Application Server acts as a middleware to filter the SQLI. The first level phase: When the user

input enters inside the login mode, the malicious text detector is used to detect the susceptibility character which is appended with the user's data and throw an exception that the user as a malicious attack and prevent from accessing the web application. In the Second level phase the login phase is set with constraint such as User id is allowed only the ten characters and the Password is allowed only four characters**.** So the user input is compared with this constraint if this level satisfies the data will be converted as SQL Query with database engine and start to compare with third level**.** In the third level phase Length of the number of possibility queries is stored in the array format in statically generated method. Each character is parsed and number of static Query is counted and stored in a static model then the input data is also calculated with the existed static value and compared if it matches it moves to the last phase otherwise it will be rejected as SQLIA's.

5. Comparison with Existing Techniques

Existing technique such as Eliminating SQL Injection Attacks [5] shows that static analysis is processed by using SQL Graph representation using FSM. In AMNESIA [3] static model build SQL-query models**:** For each hotspot, build a model that represents all the possible SQL queries that may be generated at that hotspot. A *SQL-query model* is a non-deterministic Finite - state automaton in which the transition labels consist of SQL tokens (SQL keywords and operators), delimiters, and place holders for string values. The existing technique [8] [5] is fully Query based validation but the current technique is data based validation in Static and Runtime validation to secure the web application. The execution time shows that the current technique results a better performance than existing mechanism as well as the computational cost is also minimum compared to this existing mechanism.

5.1 Result and Discussion

The Table 2 provides comparison of detection and prevention overhead for the proposed technique with existing technique [5] [8]. The proposed ASCII based string matching is developed by using two types of databases, SQL Server and MS Access. The detection overhead and prevention overhead is calculated by using the formula (5.1 and 5.2). The Figure 6 (a) and (b) provides comparison chart for detection and prevention overhead for the proposed technique with query based technique [5] [8]. The following equation is used for calculating detection and prevention overhead.

$$\text{Detection Overhead} = T_{detection}/T_{round\text{-}trip} \quad (5.1)$$

Where $T_{detection}$ is time needed for detecting malicious characters in the user input and $T_{round\text{-}trip}$ is the response time for completing a single query of the proposed scheme.

$$\text{Prevention Overhead} = T_{prevention}/T_{round\text{-}trip} \quad (5.2)$$

Where $T_{prevention}$ is the time delay needed to prevent the malevolent. The $T_{round\text{-}trip}$ is the round-trip response time for completing a single query execution.

6. Conclusion

The technique combines static and dynamic analysis in the static analysis stage, the prevention technique represents in three level phases Malicious Text Detector**,** Field Constraint Validation and Static Query Length Validation. In runtime validation stage, the user input data is validated with all these stages and results the user input as safe or unsafe. This prototype tool that implements our technique for .NET based web applications; Current technique was able to stop all of the 500 attacks that we performed on the considered applications without producing any false positive for the 1500 legitimate accesses to the

applications. This technique was able to correctly identify all attacks as SQLIA's, while allowing all legitimate queries to be performed. This proposed technique is able to stop all the attacks that performed on the considered applications without producing any false positive.

Database Type	Technique	Detection overhead per query in ms	Prevention overhead per query in ms	Average Detection overhead in ms	Average Prevention overhead in ms
SQL Server	ASCII Based String Matching	≈11	≈18	≈9.6	≈24.2
	Transparent Defense Mechanism [41]	≈19	≈25	≈18.5	≈29.4
	SQLPrevent [57]	≈18	≈24	≈17.7	≈28.6
MS Access	ASCII Based String Matching	≈15	≈18	≈17.2	≈24.2
	Transparent Defense Mechanism [41]	≈21	≈32	≈24.25	≈29.7
	SQLPrevent [57]	≈18	≈27	≈22.7	≈25.8

Table 5.2: Comparison of Detection and Prevention Overhead for ASCII based String Matching Technique with Existing Techniques

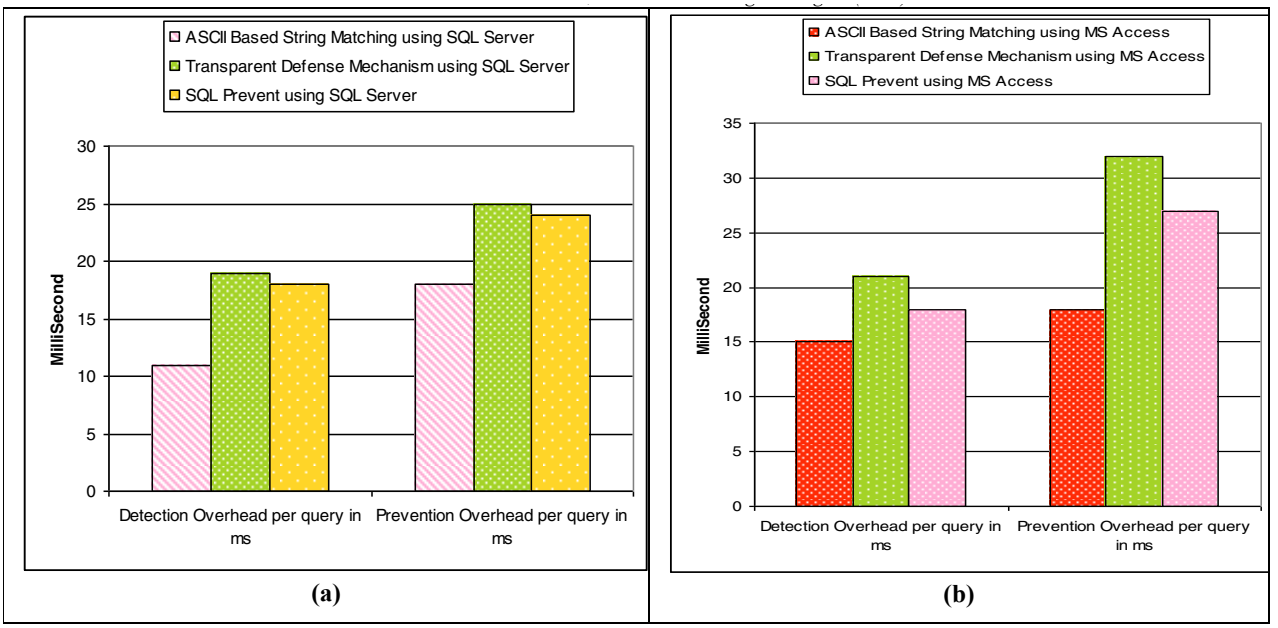

Figure 6: Comparison Chart for Detection and Prevention Overhead for ASCII based String Matching Technique with Existing Technique References

[1] Z.Su and G. Wassermann, "*The Essence of Command Injection Attacks in Web Application,*" in the 33rd Annual Symposium on Principles of Programming languages, 2006, pages 372-382.

[2] C Anley, "Advanced SQL Injection in SQL Server Applications," White Paper *Next Generation Security Software Ltd*., http://www.nextgenss.com/papers/advanced sql injection.pdf., 2002.

[3] W.G.J. Halfond, A. Orso, "AMNESIA: analysis and monitoring for Neutralizing SQL-injection attacks," 20th IEEE/ACM *International Conference on Automated Software Engineering,* Long Beach, CA, USA, 2005, pp. 174–183.

[4] V. B. Livshits, ,"Finding Security Errors in Java Programs with Static Analysis," in *Proceedings of the 14th Use nix Security Symposium*, 2005, pages 271–286.

[5] Muthuprasanna,KeWei, Suraj Kothari N, "Eliminating SQL Injection Attacks - A TransparentDefenceMechanism", *SQL Injection Attacks Prof. Jim Whitehead CMPS 183. Spring* , 2006.

[6] W.G.J. Halfond, A. Orso, P. Manolios, "Using positive tainting and syntax-aware evaluation to counter SQL-injection attacks," 14th ACM SIGSOFT *International Symposium on Foundations of Software Engineering,* 2006, pp. 175–185.

[7] W.G.J. Halfond, A. Orso, P. Manolios, "WASP: protecting web applications using positive tainting and syntax-aware evaluation," *IEEE Transactions on Software Engineering*, 2008, pp. 65–81.

[8] Sun, S.T., and Beznosov, K. "SQLPrevent: Effective dynamic detection and prevention of SQL injection attacks", (Technical Report No. LERSSE-TR-2009-032). *Laboratory for Education and Research in Secure Systems Engineering*, 2009, http://lersse-dl.ece.ubc.ca.