An Effective Technique for Detection and Prevention of SQLIA by utilizing CHECKSUM Based String Matching

Mohammad Abu Kausar, Mohammad Nasar

Abstract— with the growth of the Internet, web applications, for example, online shopping, online banking and email, have turned out to be indispensable to numerous people's every day lives. Web applications have carried with them new classes of PC security vulnerabilities, for example, SQL Injection. It is a class of information approval based vulnerabilities However, the vast majority of the web application exists have some weakness as there are some reckless people known as hacker that able to corrupt the data. Some of well-known web application vulnerabilities are SQL Injection, Buffer Overflow, Cross Site Scripting etc. Typical uses of SQL injection release private data from a database, by-pass confirmation rationale, or add unapproved records to a database. This security keeps the unapproved access to your database and furthermore it keeps your information from being changed or erased by clients without the appropriate permissions. Malicious Text Detector, Restriction Validation, Query length authentication and Text based Key Generator are the four kinds of filtration strategy used to identify and keep the SQL Infusion Attacks from getting to the database.

Index Terms— SQL Injection Attack, Web applications, Web security, Data validation, Web Application Vulnerabilities, CHECKSUM Code, Malicious Text Detector.

1 Introduction

SQL injection is an assault procedure that adventures a security vulnerability happening in the database layer of an application. Hikers utilize infusions to get unapproved access to the sensitive information, structure, and DBMS. It is a standout amongst the most well-known web application vulnerabilities.

A Database is the core of many web-applications and is utilized to store data required by the application, for example, bank account number, password, credit card information, client orders and many more. Thus, databases have turned out to be appealing and extremely lucrative focuses for hackers to hack into. SQL Injections happen when a software developer or user input data that is specifically put into a SQL Statement and doesn't legitimately approve and sift through dangerous characters. This can enable an aggressor to change SQL statement that go to the database as parameters and empower to take information from your database, as well as adjust and erase it. SQL injection attack happen when a web application does not approve values got from a web page, cookie, input, and so on., before passing them to SQL query that will be executed on a database server. This will enable hackers to control the data. The below table 1 summarizes SQL injection examples which shows the different types of threats.

TABLE 1
DIFFERENT TYPES OF SQL INJECTION EXAMPLE

Types of Threat	SQL Injection Examples
Elevation of privilege	➢ Fetch and use credentials for administrator ➢ Run shell commands
Information disclosure	➢ get saved debit/credit card information ➢ Gain insight into internal design of app
Spoofing	➢ get and use another user's information ➢ change Author value for messages
Repudiation	➢ remove transaction details ➢ remove database logs
Tampering	➢ modify different data in the database ➢ change product information
Denial of service	➢ erase sqlservr.exe process ➢ execute resource-intensive SQL queries

2 Related work

SQL injection attack is a typical risk to web applications that uses poor input validation to implement attack on database. It is turning into an intense issue in web applications as hackers steal sensitive information from database this makes very important issue for web application as well as other types of application available on the web.

- *Author Mohammad Abu Kausar is currently working as Assistant Professor in Department of Information Systems in University of Nizwa, Oman E-mail: kausar@unizwa.edu.om*
- *Co-Author Mohammad Nasar is currently working as Assistant Professor in Department of Computer Science and Information Technology in Mazoon College, Oman E-mail: nasar31786@gmail.com*

Research on SQL injection attacks (SQLIA) can be generally categorize into two classifications: vulnerability detection approaches and attack prevention approaches. The past classification comprises of methods that identify helpless areas in a Web application that may prompt SQL infusion assaults. With a specific end goal to avoid SQL injection attacks, a software developer repeatedly subjects all contributions to enter approval and sifting schedules that distinguishes endeavors to infuse SQL orders. The procedures displayed in [1, 2, 3] characterize the major static analysis techniques for vulnerability identification, where computer code is analyzed to guarantee that each bit of info is liable to an information approval register before being consolidated into a query. While these static investigation approaches scale well and distinguish vulnerabilities, their utilization in tending to the SQL infusion issue is restricted to just recognizing possibly unvalidated inputs. The tools do not give any approach to check the accuracy of the input validation routines, and programs using incomplete input validation routines may certainly pass these checks and cause SQL injection attacks. Shin et al.[4] apply SQLUnitGen, a Statical analysis tool that is used to automate the testing for recognizing input control vulnerabilities. Shin et al. used SQLUnitGen tool which author is matched with FindBugs tool, FindBugs tool is a static investigation tool. The planned method is revealed to be proficient as regard to the way that false positive was totally absent in the experiments. XI-Rong Wu et al. [5] projected a new process named k-centers (KC) to identify SQL injection attacks (SQLIAs). The total number and the centers of the clusters in KC are fixed based on unseen SQL query in the adversarial situation; in this technique the types of SQL injection attacks are changed time to time, to adjust various types of attacks. The experimental output demonstrates that the proposed technique has a delightful outcome on the SQL infusion assaults (SQLIAs) recognition in the antagonistic condition. Ramya Dharam et al. [6] proposed a structure which able to be utilized to deal with SQL Injection Attacks in light of repetition utilizing post-arrangement checking system. The structure utilizes two pre-arrangement testing systems i.e. data flow and testing basis path techniques to recognize legitimate execution ways of the product. Runtime screens are then created and incorporated to screen the execution of the product for perceived execution ways with the end goal that their infringement will recognize and avoid redundancy based SQL Injection Attacks. Shin [7] anticipated a technique to construct test input data to trace SQL injection vulnerabilities by creating a white-box from both input flow analysis and input validation analysis. Ali et al. [8] used the hash value concept to get better user verification tool. Author used the user id and password hash values SQLIPA (SQL Injection Protector for Authentication) model was produced keeping in mind the end goal to test the system. The user id and password hash values are formed and calculated at runtime for the first run through the particular client account is made. Valeur [9] planned an interruption discovery system using a machine learning method. The SQL statement produced in a web application were learned to produce the parameters of the detection model. Then, executing SQL statement was evaluated to the developed model in order to confirm for discrepancy. If the proposed model will not efficiently train, many negative and positive results can occur. Park [10] used the SQL statement of a web application and evaluated it with the SQL statement generated at runtime dynamically using the pair wise sequence alignment of amino acid formulate method to detect SQL injection attacks. This method having lot of benefits because it can identify SQL injection attacks without rewriting the web application. However, the web application will be profiled when it will be changed. Su and Wassermann [11] planned a resolution of static analysis of an SQL query using parse tree validation, for filtering user input and for generation the input validation code author used that static structure.

3 Proposed Self-protective tool in Static Level Authentication

Proposed method is the mixture of static analysis with dynamic validation. In the static analysis phase, the prevention technique signifies in three level stages Malicious Text Detector, Field Restriction Authentication and Static Query Length Authentication. In dynamic validation phase, the user input information is validated with all these stages and results the client contribution as safe or risky.

```
Function wildcardCharacters(ByVal strWords)
        Dim array1 As New ArrayList
        array1.Add("select")
        array1.Add("--")array1.Add("drop")ar-
        ray1.Add(";")array1.Add("insert")ar-
        ray1.Add("delete")
        array1.Add("xp_")array1.Add("'")
        Dim x As Integer
                For x = 0 To array1.Count-1
                strWords = Replace(strWords, ar-
        ray1.Item(x), "'", CompareMethod.Text)
                Next
Return strWords
End Function
```

Fig. 1. Sample code for Malicious Text Detector

3.1 Malicious Text Detector

1. Statically form a model for staying away from Meta characters (figure 1)
2. Detect the vulnerability character which is added with the user's data and stop the malicious attacker from getting to the web application.

Test 1: select * from registration where user_id ='kausar' - - 'and user_password=123456

```
userid_valid1 = object1.stripQuotes(UCase('kausar'- -'))
userid_valid2 = object1.killChars('kausar'- -')
password_valid1 = object1.stripQuotes(0)
password_valid2 = object1.killChars(0)
correct_userid = 'kausar'
correct_password = 0
```

Test 1: express a malignant query, the user input information is pattern matched with malicious text detector and identifies the wild card characters (in this test wild card character is --); system throw an exception as a SQL injection attack.

Field Limitation Validation
The Login stage is set with restriction as username is permitted only the sixteen characters and Password is permitted only six characters as shown in (Fig. 2).

```
length_username = Len('kausar') length_password = Len(0)

If length_username = 16 and length_password = 6 Then

[Do action]

End If
```

Fig. 2. Constraint validation Example code

4 Proposed SQL Injection Attack - Protector Mechanism

In view of high level security the current system comprises of three purification stages with static and runtime level. In the first place level stage is Malicious Text Detector, Second level stage is Restriction Identifier, Third level stage is Static Query Analysis and the Fourth level Stage is the CHECKSUM Key Generator based on Text.

4.1 Basic Principle in Run time Observing

At the time of application running, the user's input data are analysed against the relating filtration methods used to check for their legitimacy. The dynamically produced user's input are not fulfilled with all the three level of purification method then they are identified as malicious else authentic user and permitted to get to the web application.

4.2 Text based Key Generator

1. Converting User input into CHECKSUM code
2. Searching the availability of converted CHECKSUM in table and returns valid UserName and Password,
3. Four parameters are kept in database that is UserName, password, CHECKSUM UserName and CHECKSUM Password (Table 2)

UserName	Password	CHECKSUM UserName	CHECKSUM Password
Kausar	123456	151292031	148828186
Nasar	324563	9652351	150864167
Unizwa	123987	179490254	148829515
Oman	213904	610939	149812680

TABLE 2: Sensitive Data conversion into CHECKSUM code in database table

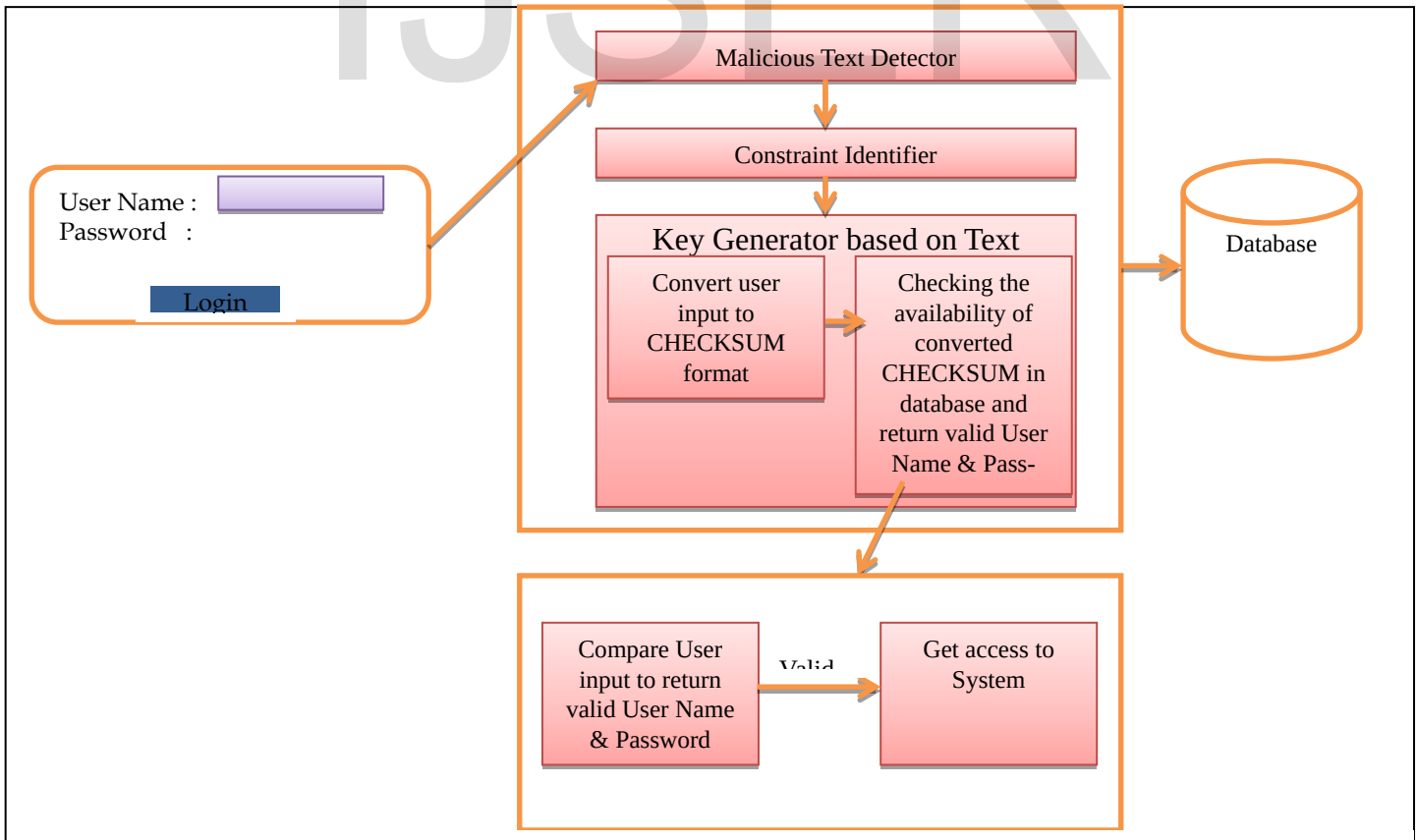

Fig. 3. Proposed Architecture for CHECKSUM based String Matching system.

This current method is used to authenticate the user input with static and dynamic analysis to identify and avoid SQL Injection. Figure 4 demonstrates that the client input information enters in the Login stage from the User Interface, Application Server behave as a middleware to filter the SQL Injection.

The first level stage: When the client input enters inside the login phase, the malicious text detector is utilized to identify the vulnerability character which is attached with the user's information and throw an exception that the client as a malignant assault and keep from getting to the application.

In the Second level stage: The login stage is set with restriction for example, User Name is permitted just the sixteen characters and the Password is permitted just six characters. So the customer input is coordinated with this confinement if this level satisfies the information will be changed over as SQL Query with database and begin to match with third level.

In the third level stage: The length of the number of possible queries is kept in the array layout in statically created technique. Each and every character is analysed and number of static Query is included and put away in a static model and after that the input data is also calculated with the existed static value and matched if it matches it moves to the CHECKSUM key generator or else it will be disallowed as SQL Injection attack.

5 Comparison with Existing Systems

Existing framework, for example, In AMNESIA [12] static model form SQL-query models: For every hotspot, construct a model that speaks to all the conceivable SQL inquiries that might be created at that hotspot. A SQL-query model is a nondeterministic Finite - state automaton in which the change labels consist of SQL keywords and operators, delimiters, and place holders for string values. Author [13] demonstrates that static investigation is handled by utilizing SQL Graph demonstration using FSM. The existing system [13], [14] is fully Query based authentication but the proposed system is data based authentication in Static and dynamic authentication to secure the web application. The execution time of proposed system shows that the result is better performance than existing system and in addition the computational cost is additionally least contrasted with this proposed system.

5.1 Result and Discussion

Table 3 gives comparison of prevention and detection overhead for the proposed system with existing system [13] [14]. The proposed CHECKSUM based string matching is developed by utilizing two types of databases, SQL Server and Oracle. The prevention overhead and detection overhead is calculated by using the formula (1 and 2). The Figure 4 and Figure 5 provides comparison chart for prevention and detection overhead for the current system with query based system [13] [14]. The following formula is used for calculating prevention and detection overhead.

Overhead Detection = Detection Time / Round-trip Time (1)

Where Detection Time is time required for identifying malicious characters in the client input and Round-trip Time is the reaction time for finishing a single query of the proposed system.

Overhead Prevention = Prevention Time / Round-trip Time (2)

Where Prevention Time is the time postpone expected to keep the malicious. The Round-trip Time is the round-trip reaction time for finishing the single query implementation.

Database	Technique	Overhead Detection in ms per query	Overhead Prevention in ms per query
Sql Server	CHECKSUM Based String Matching	10	17
	Transparent Defense Mechanism [13]	18	24
	SQLPrevent Techique [14]	16	22
Oracle	CHECKSUM Based String Matching	14	17
	Transparent Defense Mechanism [13]	20	31
	SQLPrevent Techique [14]	17	26

TABLE 3: COMPARISON OF DETECTION AND PREVENTION OVERHEAD FOR CHECKSUM BASED STRING MATCHING SYSTEM WITH EXISTING SYSTEMS

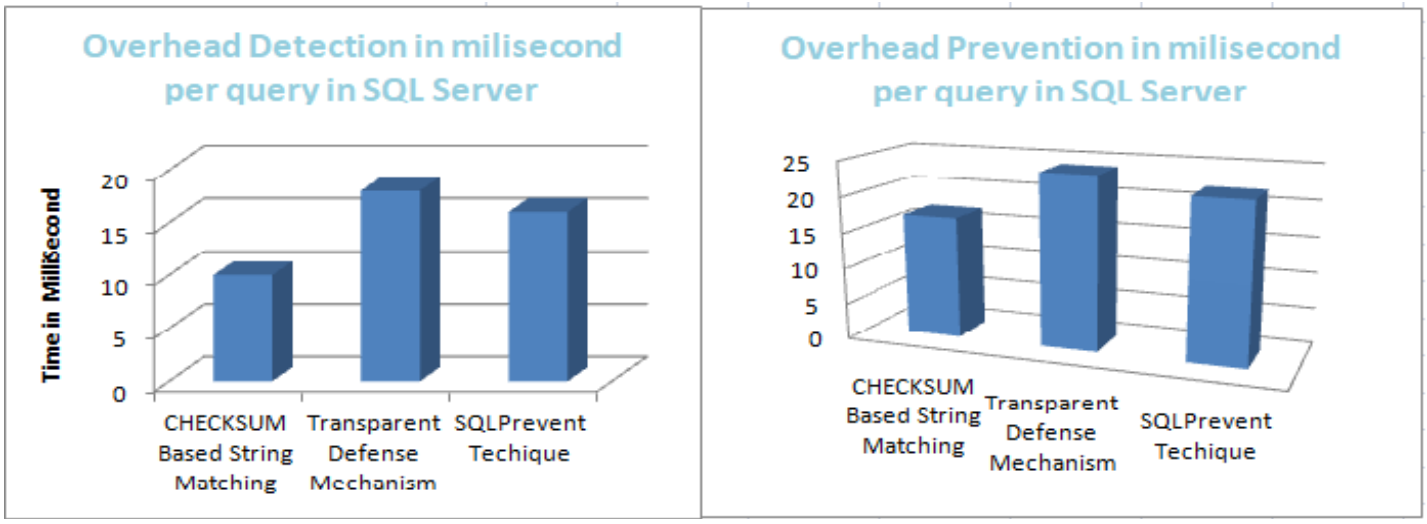

Fig. 4. Comparison Chart for Detection and Prevention Overhead in SQL Server for CHECKSUM based String Matching System with Current System

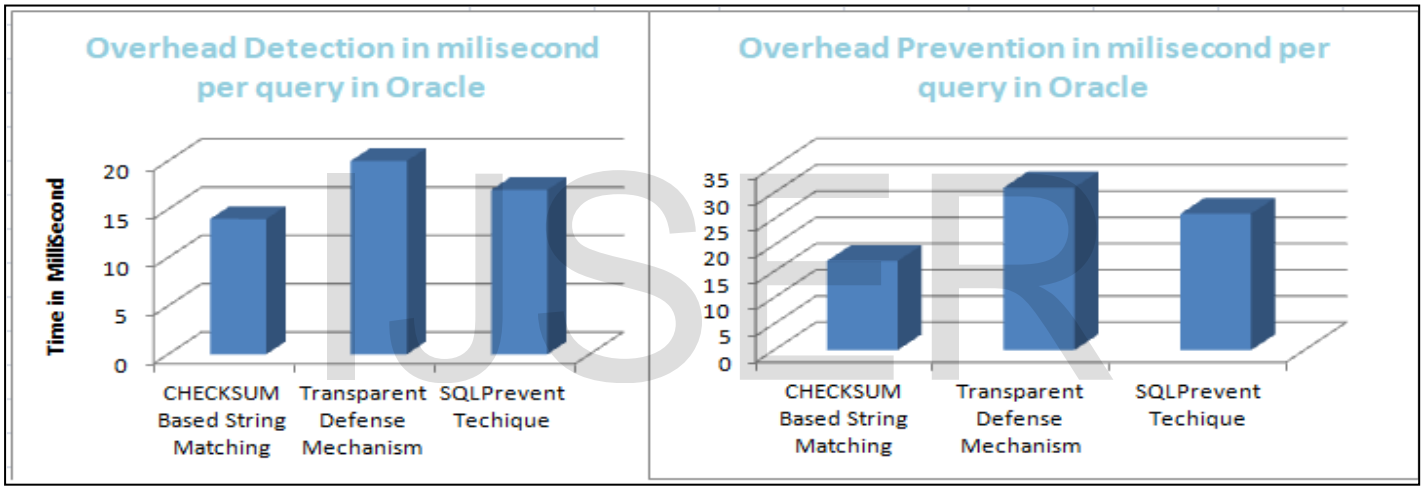

Fig. 5. Comparison Chart for Detection and Prevention Overhead in Oracle for CHECKSUM based String Matching System with Current System

6 Conclusion

The proposed system joins static and dynamic analysis. In the static investigation phase, the prevention method characterizes in three level stages Malicious Text Detector, Field Constraint Authentication and Static Query Length Authentication. The proposed system uses CHECKSUM which generates small amount of code to store in database. In runtime validation phase, the client input information is approved with every one of these stages and results the client input as safe or risky. This proposed system implements .NET based web applications; proposed system is able to prevent almost all type of SQL Injection Attacks. This system could effectively distinguish all assaults as SQL Injection Attacks, while enabling every single query to be performed.

References

[1] Y. Shin, L. Williams and T. Xie, "SQLUnitGen: Test Case Generation for SQL Injection Detection," North Carolina

[2] Xi-Rong Wu; Chan, P.P.K., "SQL injection attacks detection in adversarial environments by k-centers," Machine Learning and Cybernetics (ICMLC), 2012 International Conference on , vol.1, no., pp.406,410, 5-17 July 2012.

[3] Dharam, R.; Shiva, S.G., "Runtime monitors for tautology based SQL injection attacks," Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on , vol., no., pp.253,258, 26-28 June 2012.

[4] Preventing SQL Injections in Online Applications: Study, Recommendations and Java Solution Prototype Based on the SQL DOM .Etienne Janot, Pavol Zavarsky Concordia University College of Alberta, Department of Information Systems Security

[5] Xie, Y., and Aiken, A. Static detection of security vulnerabilities in scripting languages. In USENIX Security Symposium (2006).

[6] Mcclure, R. A. and Kr¨Uger, I.H. 2005. SQL DOM: Compile time checking of dynamic SQL statements.In Proceedings of the 27th International Conference on Software Engineering (ICSE'05).ACM, New York, 88–96.

[7] Y. Shin, Improving the identification of actual input manipulation vulnerabilities, in: 14th ACM SIGSOFT Symposium on Foundations of Software Engineering ACM, 2006.

[8] Shaukat Ali, Azhar Rauf, Huma Javed "SQLIPA:An authentication mechanism Against SQL Injection"

[9] F. Valeur, D. Mutz, G. Vigna, A learning-based approach to the detection of SQL attacks, in: Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment, 2005, pp 123–140.

[10] J. Park, B. Noh, SQL injection attack detection: profiling of web application parameter using the sequence pairwise alignment, in: Information Security Applications, in: LNCS, vol. 4298, 2007, pp. 74–82.

[11] 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.

[12] 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.

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

[14] 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.