Detection and Prevention of SQL Injection Attack Using Hashing Technique

Surya Pratap Singh, Upendra Nath Tripathi, Manish Mishra

***Abstract*— In today's environment the database is most important part of any online application and in recent years the database technology evolved to a greater extent so as the attacks to the database also increases, the most important type of attack these days are SQL injection attacks, these attacks are very serious security threats to web applications because they enable the attackers to gain unrestricted asses to databases and potentially confidential information these database contain. Although researchers have proposed numerous methods to help overcome form the SQL injection problem, the current approaches either have limitations or fail to cover full scope of the problem. In SQL injection attack (SQLIA) the attacker can trick the server to obtain illegal authorization and asses the database using SQL queries. This is because the developers of the applications are not fully aware of attacks by SQL injection and its causes. This paper gives an overview to the SQL Injection attacks (SQLIA) and methods to prevent them. We will discuss all the proposed models to block SQL Injections. We also present and analyses existing detection prevention techniques against SQL injection attacks.**

***Index Terms*— SQL, SQLIA, and Hash function, Validation**

I. Introduction

In today's environment, the Database is a fast growing and emerging field, as it is needed in every field which needs computerization. So the Database security is also very vital factor that the researchers and practitioners are facing nowadays. SQL injection vulnerabilities have been described as one of the most serious threats for Web applications [13], [17]. Web applications that are vulnerable to SQL injection may allow an attacker to gain complete access to their underlying databases. Because these databases often contain sensitive consumer or user information, the Resulting security violations can include identity theft, loss of confidential information, and fraud In some cases, attackers can even use an SQL injection vulnerability to take control of and corrupt the system that hosts the Web application.

SQL Injection Attacks (SQLIA's) are one of the most severe threats to web application security. They are frequently employed by malicious users for a variety of reasons like financial fraud, theft of confidential data, website defacement, sabotage, etc. The number of SQLIA's reported in the past few years has been showing a steadily increasing trend and so is the scale of the attacks. It is, therefore, of paramount importance to prevent such types of attacks, and SQLIA prevention has become one of the most active topics of research in the industry and academia. There has been significant progress in the field and a number of models have been proposed and developed to counter SQLIA's, but none have been able to guarantee an absolute level of security in web applications, mainly due to the diversity and scope of SQLIA's. One common programming practice in today's times to avoid SQLIA's is to use database stored procedures instead of direct SQL statements to interact with underlying databases in a web application, since these are known to use parameterized queries and hence are not prone to the basic types of SLQIA's.

Although recently there has been a great deal of attention to the problem of SQL injection vulnerabilities, many proposed solutions fail to address the full scope of the problem. There are many types of SQLIAs and countless variations on these basic types. Researchers and practitioners are often unaware of the myriad of different techniques that can be used to perform SQLIAs. Therefore, most of the solutions proposed detect or prevent only a subset of the possible SQLIAs.

Manuscript received September 16, 2014.

Surya Pratap Singh , Department of Computer Science, DDU Gorakhpur University, Gorakhpur, India, Mobile No. +919450138221
Upendra Nath Tripathi, Department of Computer Science, DDU Gorakhpur University, Gorakhpur, India, Mobile No. +919450181905
Manish Mishra, Department of Electronics, DDU Gorakhpur University, Gorakhpur, India, Mobile No. +919415875144

II. BASICS OF SQLIA

SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker) [12]. SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. Any procedure that constructs SQL statements should be reviewed for injection vulnerabilities because SQL Server will execute all syntactically valid queries that it receives. Even parameterized data can be manipulated by a skilled and determined attacker.

III. INJECTION PROCESS

The primary form of SQL injection consists of direct insertion of code into user-input variables that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed.

The injection process works by prematurely terminating a text string and appending a new command. Because the inserted command may have additional strings appended to it before it is executed, the malefactor terminates the injected string with a comment mark "--". Subsequent text is ignored at execution time.[16]

The following script shows a simple SQL injection. The script builds an SQL query by concatenating hard-coded strings together with a string entered by the user:

varShipcity;
ShipCity = Request.form ("ShipCity");
varsql = "select * from OrdersTable where ShipCity = '" + ShipCity + "'";
The user is prompted to enter the name of a city. If she enters Redmond, the query assembled by the script looks similar to the following:
SELECT * FROM OrdersTable WHERE ShipCity = 'Redmond'
However, assume that the user enters the following:
Redmond'; drop table OrdersTable—

In this case, the following query is assembled by the script:
SELECT * FROM OrdersTable WHERE ShipCity = 'Redmond';drop table OrdersTable--'

TYPES OF SQL INJECTION ATTACK

The semicolon (;) denotes the end of one query and the start of another. The double hyphen (--) indicates that the rest of the current line is a comment and should be ignored. If the modified code is syntactically correct, it will be executed by the server. When SQL Server processes this statement, SQL Server will first select all records in OrdersTable where ShipCity is Redmond. Then, SQL Server will drop OrdersTable.

IV. TYPES OF SQL INJECTION ATTACK

There are various types of SQL Injection attacks, these depends on the intent of attacker. The attacker can perform the attack sequentially or altogether. We can classify the SQLIA into following types –

A. *Tautologies*

The general goal of a tautology-based attack is to injectcode in one or more conditional statements so that they always evaluate to true. The consequences of this attack depend on how theresults of the query are used within the application. The most commonusages are to bypass authentication pages and extract data. Inthis type of injection, an attacker exploits an injectable field that isused in a query's WHERE conditional. Transforming the conditionalinto a tautology causes all of the rows in the database table targetedby the query to be returned.
Example: 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.

B. *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>.
Example: Referring to the running example, an attacker could inject the text "UNION SELECT cardNo from CreditCards where acctNo=10032 - -" into the login field, which produces the following query:
SELECT accounts FROM users WHERE login='' UNION SELECT cardNo from CreditCards whereacctNo=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 "card No" is displayed along with the account information. References: [10, 14,8]

C. *Illegal/Logically Incorrect Queries:*

When a query is rejected, an error message is returned from the database including useful debugging information. This error messages help attacker to find vulnerable parameters in the application and consequently database of the application. In fact attacker injects junk input or SQL tokens in query to produce syntax error, type mismatches, or logical errors by purpose. In this example attacker makes a type mismatch error by injecting the following text into the pin input field:
1) Original URL: http://www.arch.polimi.itleventil?id nav=8864
2) SQL Injection: http://www.arch.polimLitieventil?id_nav=8864'
3) Error message showed: SELECT name FROM Employee WHERE id =8864\'
From the message error we can find out name of table and fields: name; Employee; id. By the gained information attacker can organize more strict attacks.

D. *Piggybacked Queries*

In this attack type, an attacker tries to inject additionalqueries into the original query. We distinguish this type from othersbecause, in this case, attackers are not trying to modify the originalintended query; instead, they are trying to include new and distinctqueries that "piggy-back" on the original query. As a result, thedatabase receives multiple SQL queries. The first is the intendedquery which is executed as normal; the subsequent ones are theinjected queries, which are executed in addition to the first.
Example: If the attacker inputs "'; drop table users - -" into the pass field, the application generates the query:
SELECT accounts FROM users WHERE login='ram' ANDpass=''; drop table users – ' AND pin=123
After completing the first query, the database would recognize the 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.

E. *Inference*

In this attack, the query is modified to recast it in the form of an action that is executed based on the answer to a true/false question about data values in the database. In this type of injection, attackers are generally trying to attack a site that has been secured enough so that, when an injection has succeeded, there is no usable feedback via database error messages.[7] Since databaseerror messages are unavailable to provide the attacker with feedback, attackers must use a different method of obtaining a response from the database. In this situation, the attacker injects commandsinto the site and then observes how the function/response of the website changes.

Example: Using the code from our running example, we illustrate two ways in which Inference based attacks can be used. The first of these is identifying injectable parameters using blind injection. Consider two possible injections into the login field. The first being “legalUser’ and 1=0 - -” and the second, “legalUser’ and 1=1 - -”.

These injections result in the following two queries:

```
SELECT accounts FROM users WHERE
login=’legalUser’and 1=0 – ’ AND pass=’’ AND pin=0
SELECT accounts FROM users WHERE
login=’legalUser’and 1=1 – ’ AND pass=’’ AND pin=0
```

F. *Stored Procedure*

Stored procedure is a part of database that programmer could set an extra abstraction layer on the database. As stored procedure could be coded by programmer, so, this part is as inject able as web application forms.[9] Depend on specific stored procedure on the database there are different ways to attack. In the following example, attacker exploits parameterized stored procedure.

```
CREATE PROCEDURE
        DBO.isAuthenticated
    @userName varchar2, @pass varchar2, @pin int
AS
EXEC (“SELECT accounts FROM users WHERE login=’”
+@userName+ “’ and pass=’” +@password+”’ and pin=”
+@pin);
GO
```

V. RELATED WORK

The techniques which are currently available can cover a subset of the vulnerabilities of the SQL Injections, some work of the researchers are listed in the following section-

Roichman and Gudes’s Scheme – [1] suggests using a fine-grained access control to web databases. The authors develop a new method based on fine-grained access control mechanism. The access to the database is supervised and monitored by the built-in database access control. This is a solution to the vulnerability of the SQL session traceability.

Shaukat Ali et al.’s Scheme – [2] 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

Thomas et al.’s Scheme – Thomas et al., in [3] proposed an automated prepared statement generation algorithm to remove SQL Injection Vulnerabilities. The authors implement their research work using four open source projects namely: (i) Net-trust, (ii) Itrust, (iii) WebGoat, and (iv) Roller. On the basis of the experimental results, their prepared statement code was able to successfully replace 94% of the SQLIVs in four open source projects.

SQLIA Prevention Using Stored Procedures – Stored procedures are subroutines in the database which the applications can make call to [4]. The prevention in these stored procedures is implemented by a combination of static analysis and runtime analysis. The static analysis used for commands identification is achieved through stored procedure parser and the runtime analysis by using a SQL Checker for input identification

SAFELI – [5] This research deals with the Static Analysis Framework in order to detect SQL Injection Vulnerabilities. This framework aims to identifying the SQL Injection attacks during the compile-time. The two main advantages of this static analysis tool are: first, it does a White-box Static Analysis and secondly, it uses a Hybrid-Constraint Solver. If we consider the White-box we found the Static Analysis, the proposed approach considers the byte-code and deals mainly with strings. While on the other hand, the Hybrid-Constraint Solver implements the methods to an efficient string analysis tool which is able to deal with Boolean, integer and string variables.

William G.J.Halfond et al.’s Scheme- [6]- proposed an approach that works by combining static analysis and runtime monitoring of database queries. In its static part, technique uses program analysis to automatically build a model of the legitimate queries that will be generated by the application. While in the dynamic part, the technique monitors the dynamically runtime generated queries and checks them for acceptability with the statically-generated model. A query that doesn’t match with the model represent potential SQLIAs and are hence prevented from executing on the database and reported.

VI. PROPOSED TECHNIQUE

After studying the work of various researchers, we propose the methods by which the SQLIA can be prevented easily, for this purpose we propose some defence mechanism and username and password validation using cryptographically Hash function.

A. *SQLIA Prevention using Data Validation –*

For validation of data we propose the following three approaches –

a) escape single quotes –

```
functionescape ( input )
        input = replace(input, “’”, “’’”)
        escape = input
end function
```

b) Reject input that is known to be bad –

```
functionvalidate string( input )
```

```
        known_bad = array( "select", "insert", "update",
        "delete", "drop", "--", "'" )
        validate_string = true
    for i = lbound( known_bad ) to ubound( known_bad )
            if ( instr( 1, input, known_bad(i), vbtextcompare )
            !=0 ) then
                    validate_string = false
                    exit function
                    end if
            next
        end function
```

c) Accept only input that is known to be good –

```
functionvalidatepassword( input )
good_password_ch="abcdefghijklmnopqrstuvwxyzABCDE
FGHIJKLMNOPQRSTUVWXYZ0123456789"
validatepassword = true
for i = 1 to len( input )
        c = mid( input, i, 1 )
        if ( InStr( good_password_ch, c ) = 0 ) then
        validatepassword = false
        exit function
        end if
next
end function
```

B. SQLIA Prevention using Crypto graphical Hash function –

For the purpose of protecting against unauthorized login by using SQLIA, we provide Hash function mechanism. In this mechanism we need to add two additional attributes in the login database one for Hash value for user name and another is for has value for password. When the admin first create the user account and assigns a password the Hash value is automatically generated by using Hash function algorithm and stored in the database along with the login information of the user. These information is stored in the database in encrypted form. Now when the user needs to login to the server he/she passes his/her username and password and the Hash value is generated automatically and the Hash value is also sent from client computer to server along with the username and password in encrypted form as HTTP request.

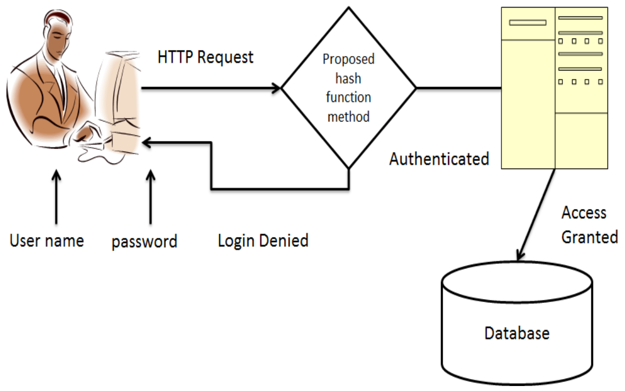

Fig1. Architecture of Prevention of SQLIA

Now receiving the HTTP request server first decrypt the incoming data. Now the username and password is matched with the value stored in the database and the Hash value of username and passwords are matched with the stored values of username and password. If it succeeds the login is provided else if the username password or Hash value does not match to the value stored in the database the login request is rejected as attempt of SQLIA. In this manner there is no chance that someone can bypass the login process without correct values and it is impossible to produce has value for a hacker or intruder because it is dynamic in nature. So no one can do any malicious activities to the database.

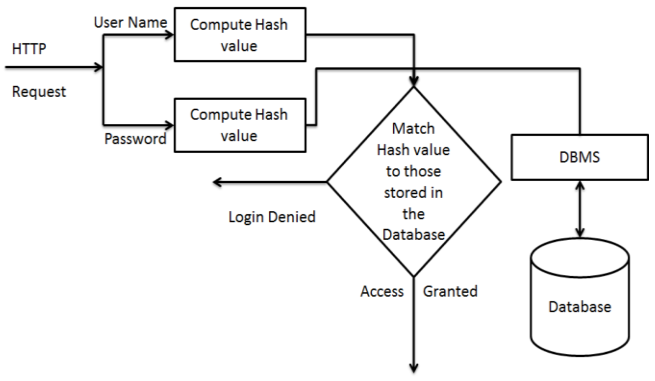

Fig2. Validation of Hash value on Server

VII. CONCLUSION

The SQLIA is most vulnerable security threat to database in recent years because every hacker or intruder tries to break the database security using this type of attack, so the prevention from these threats are more acute. Various solutions are given to prevent from SQLIA by different researchers but none of the solutions is fully able to prevent the database from these attacks.

So this paper explains the nature and injection process of SQLIA. It also explains possible cases in which the SQL Injection attack can be done. To prevent the system from these attacks this paper proposed the validation mechanism by which we can only allow good inputs to be executed. This paper also proposes the Hash function mechanism by which login process into the server can be prevented from SQLIA. There are various other ways to attack on the database using SQLIA that needs to be covered, so there is much scope in this area of research.

REFERENCES

[1] Roichman, A., Gudes, E.: Fine-grained Access Control toWeb Databases. In: Proc. of 12th SACMAT Symposium, France (2007)
[2] Shaukat Ali, Azhar Rauf, and Huma Javed “SQLIPA:An authentication mechanism Against SQL Injection
[3] 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)
[4] K. Amirtahmasebi, S. R. Jalalinia, S. Khadem, "A survey of SQLinjection defence mechanisms," Proc. Of ICITST 2009, vol., no., pp.1-8, 9-12 Nov. 2009
[5] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao. A StaticAnalysis framework for Detecting SQL Injection Vulnerabilities, OMPSAC 2007, pp.87-96, 24-27 July 2007.
[6] William G.J.Halfond and Alessandro Orso “AMNESIA:Analysis and Monitoring for Neutralizing SQL-Injection Attacks”.
[7] M. Martin, B. Livshits, and M. S. Lam. Finding Application Errorsand Security Flaws Using PQL: A Program Query Language.
[8] S. McDonald. SQL Injection: Modes of Attack, Defence, and Why It Matters.

[9] G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti. Using Parse Tree Validation to Prevent SQL Injection Attacks. In International Workshop on Software Engineering and Middleware (SEM), 2005.

[10] Advanced SQL Injection in SQL Server Applications An NGSSoftware Insight Security Research (NISR) Publication ©2002 Next Generation Security Software Ltd

[11] W. G. Halfond and A. Orso. Combining Static Analysis and RuntimeMonitoring to Counter SQL-Injection Attacks.2005

[12] www.wikipedia.com

[13] Vulnerability Management in Web Applications R. Thenmozhi, M. Priyadharshini, V. VidhyaLakshmi, K. Abirami http://www.ciitresearch.org/dl/index.php/dmke/article/view/DMKE042013007

[14] David Litchfield: Web Application Disassembly with ODBC Error Messages

[15] C. Gould, Z. Su, and P. Devanbu. Static Checking of Dynamically Generated Queries in Database Applications. In Proceedings of the 26th International Conference on Software Engineering (ICSE 04), pages 645–654, 2004.

[16] http://technet.microsoft.com/en-s/library/ms161953%28v=SQL.105%29.aspx

[17] T. O. Foundation. Top Ten Most Critical Web Application Vulnerabilities, 2005. http://www.owasp.org/documentation/topten.html

Mr. Surya Pratap Singh is pursuing PhD. In the department of computer science DDU Gorakhpur University, Gorakhpur (U.P. India) under the supervision of Dr. U.N. Tripathi. Area of research interest is Database Security, Networking. Mr. Surya Pratap Sing has published 05 papers in different national and international conferences/ Journals.

Dr. Upendra Nath Tripathi is Assistant professor in Department of computer science DDU Gorakhpur University, Gorakhpur (U.P. India) . He has 13 years of teaching and research experience. He has published 40 papers in various National and International Journals/conferences. His area of research interest is database systems, networking.

Dr. Manish Mishra is Assistant professor in Department of Electronics DDU Gorakhpur University, Gorakhpur (U.P. India). He has 13 years of teaching and research experience. He has published 45 papers in various National and International Journals/conferences. His area of research interest is Computer Technology, fast processor design.