Volume 3, Issue 11, November 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Database Security by Preventing SQL Injection Attacks in Stored Procedures

Shelly Rohilla*, Pradeep Kumar Mittal
DCSA, Kurukshetra University
India

Abstract – With widespread adoption of the Web as an instant means of information dissemination and various other transactions, including those having financial consequences like e-banking, e-shopping, online payment of bills etc, we are becoming more and more dependent on web applications. An unauthorized access to this much of confidential data by a crafted user can threat their confidentiality, integrity, and authority. As a result, the system could bear heavy loss in giving proper services to its users or it may face complete destruction. Sometimes such type of collapse of a system can threaten the existence of a company or a bank or an industry. SQL Injection attacks are one of the most dangerous security threats to web applications. Several researchers have proposed several ways to prevent SQL injection attacks in the application layer but very little emphasis is laid on preventing SQL Injection attacks in stored procedures in the database layer. In this paper a novel technique to prevent SQLIA in stored procedures is proposed. This technique provides a two phase security to the application, so that, if one phase is compromised, the second phase can still prevent the attack.

Keywords – Web Application Security, SQL Injection, Stored Procedures.

I. Introduction

With time, World Wide Web has experienced remarkable growth in businesses, individuals, governments & it found that web applications can give effective, efficient and reliable solutions to the challenges of communicating and conducting commerce. For example many people pay their bills, book the hotels or pass exams by dynamic websites instead of spending time for communicating. Web applications like e-commerce, online banking, enterprise collaboration and supply chain management sites, concludes that at least 92% of Web applications are vulnerable to some form of attack [6]. It is clear that private information of people must be kept secret and confidentiality and integrity of them must be provided by developer of web application but unfortunately there is no any guarantee for preserving the underlying databases from current attacks. As a result, the system could bear heavy loss in giving proper services to its users or it may face complete destruction. Sometimes such type of collapse of a system can threaten the existence of a company or a bank or an industry. SQL Injection attacks are one of the most dangerous security threats to web applications. SQL is a special-purpose programming language used to communicate with databases. SQL can insert data, retrieve data, and update and delete data. Of course, any system can be misused, and the most common form of misuse of SQL is an SQL injection [9]. SQL injection is nothing but, using the above operations against the database in a way that it no more fulfills the desired results but give the attacker an opportunity to run his own SQL command against the database that too using the front end of web sites[8]. The SQL injection technique tricks the target into passing malicious SQL code to a database by embedding portions of code with user input [9]. An SQL injection is a kind of injection vulnerability in which the attacker tries to inject arbitrary pieces of malicious data into the input fields of an application, which, when processed by the application, causes that data to be executed as a piece of code by the back end SQL server, thereby giving undesired results which the developer of the application did not anticipate, leveraging almost a complete compromise of system in most cases.

A. *Attack Intent*

There are different types of SQL Injection attacks and each attack is performed for some intended purposes. These purposes are as follows.

1) *Identifying Injectable Parameters:* The attacker wants to find which parameters and user-input fields are vulnerable to SQLIA in a web application.
2) *Performing Database Finger-Printing:* The attacker wants to discover the type and version of database used by the Web application. Different types of databases respond differently to different queries and attacks, and this information can be used to “fingerprint” the database. If the attacker knows the type and version of the database used by a Web application then it allows the attacker to craft database specific attacks.
3) *Determining Database Schema:* The attacker wants to know database schema information, such as table names, column names, and column data types in order to correctly extract data from a database.
4) *Extracting Data:* These types of attacks employ techniques that will extract valuable data values from the database.

5) *Adding or Modifying Data:* The aim of these attacks is to add or change information in a database.
6) *Performing Denial of Service:* These attacks are performed to shut down the database of a Web application, thus denying service to other users even to legitimate ones.
7) *Evading Detection:* This type of attacks refers to certain those which are employed to avoid auditing and detection by system protection mechanisms.
8) *Bypassing Authentication:* The goal of these types of attacks is to allow the attacker to bypass authentication mechanisms of application and database. Bypassing such mechanisms could allow the attacker to assume the rights and privileges associated with another application user.
9) *Executing Remote Commands:* These types of attacks attempt to execute arbitrary commands on the database. These commands can be stored procedures or functions available to database users.
10) *Performing Privilege Escalation:* These attacks take advantage of implementation errors or logical flaws in the database in order to escalate the privileges of the attacker [1].

II. RELATED WORKS

Over years, many tools for detection and prevention of SQL Injection attacks have been developed. AMNESIA developed by Halfond and Orso in [4] is a detection and prevention tool for SQL injection attack. It uses static analysis and runtime monitoring for the purpose. The tool builds a model of the legitimate queries at each hotspot i.e. where SQL queries are issued to database engine and monitors the application at runtime to ensure that all generated queries match the statically-generated model. In [5], a tool named CANDID is proposed for detecting SQL injection. The tool dynamically infers the programmer-intended query structure on any input, and detects attacks by comparing them against the intended query structure. In [6], SQLRand uses instruction set randomization to detect and abort queries with injected code and every SQL keyword is joined with a random integer to mislead the attacker. The proposed technique in [7] prevents SQLIA in stored procedures by combining static application code analysis with runtime validation. In the static part, a stored procedure parser is designed and it instruments the necessary statements in order to compare the original SQL statement structure to that including user inputs for every SQL statement which depends on user inputs. The technique abstracts the intended SQL query behavior in an application in the form of an SQL-graph and this graph is then validated against all the different user inputs at runtime to capture all malicious SQL queries, before they are sent for execution. An efficient technique is presented in [2] for detecting and preventing SQL Injection attack using pattern matching algorithm. Pattern matching identifies or detects any anomaly packet from a sequential action, as the malicious code includes many anomaly packets or strings. The technique proposed in [3] uses a new middle-ware-based prevention mechanism: SQLIMW. The SQLIMW avoids SQL-Injection attack from the programmer to the server. Hash function is used to replace encryption for better security. Furthermore, by combining the hash with XOR, it protects username, password and private key of SQLIMW. The proposal provides better security and efficiency. This kind of middleware is transparent to the programmer is not limited to sign-on authentication mechanism or single sign-on system, it can exist in any layer of Web application system exchanging information with the database.

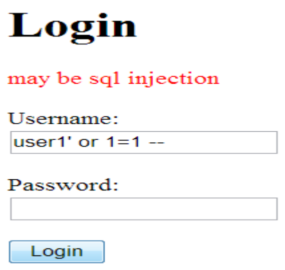
Almost every solution given to detect or prevent SQL Injection attacks is either for application layer or for database layer. But none of them provide security at both application and database layers. Also very little emphasis is laid on preventing SQL Injection attacks in stored procedures. [7] Although the mechanism of SQLIA is the same for both stored procedure and application layer program, the same detection technique could not be applied to stored procedures, because of limited programmability of stored procedures and the technique's usability and deployability. Many existing techniques, such as filtering, information-flow analysis, penetration testing, and defensive coding, can detect and prevent a subset not all of the vulnerabilities that lead to SQLIAs.

III. PROPOSED WORK

Here an efficient scheme for detecting and preventing SQL Injection attacks has been introduced. This scheme provides security from both the frontend as well as the backend of the application so that, if security is compromised at one end, the second end can still prevent SQL injection attacks. It consists of two modules:

- Frontend phase
- Backend phase

In the frontend phase, user input validation takes place. A list of several malicious known symbols or say anomaly tokens has been maintained. When a user provides an input in the input field, the validation process checks that input and matches it with the anomaly token list and if a match exists then it restricts further access by the attacker as shown in fig. 1.


Login

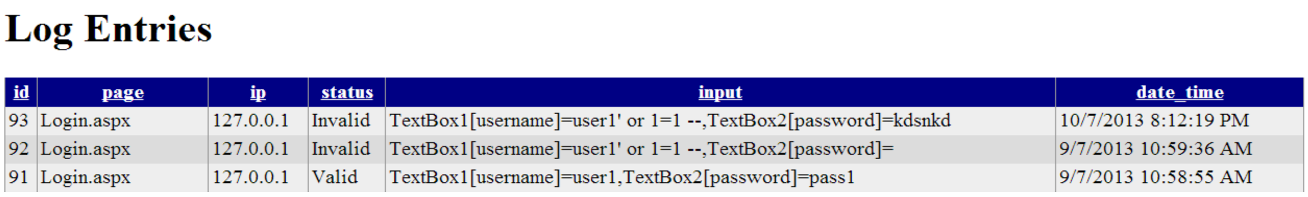
may be sql injection

Username: user1' or 1=1 --

Password:

Fig. 1 Login Form

At every illegal attempt a log entry will be saved in the database in which the date, time, page of the application on which illegal attempt was made and the IP address of the attacker got saved as shown in fig. 2. The IP address can help us track the attacker in future if we want to.



Log Entries

id	page	ip	status	input	date_time
93	Login.aspx	127.0.0.1	Invalid	TextBox1[username]=user1' or 1=1 --,TextBox2[password]=kdsnkd	10/7/2013 8:12:19 PM
92	Login.aspx	127.0.0.1	Invalid	TextBox1[username]=user1' or 1=1 --,TextBox2[password]=	9/7/2013 10:59:36 AM
91	Login.aspx	127.0.0.1	Valid	TextBox1[username]=user1,TextBox2[password]=pass1	9/7/2013 10:58:55 AM

Fig. 2 Log entry

In this way one can prevent SQLIA (SQL Injection Attacks) through the frontend of the application. The architecture for frontend phase is shown below in fig.3.

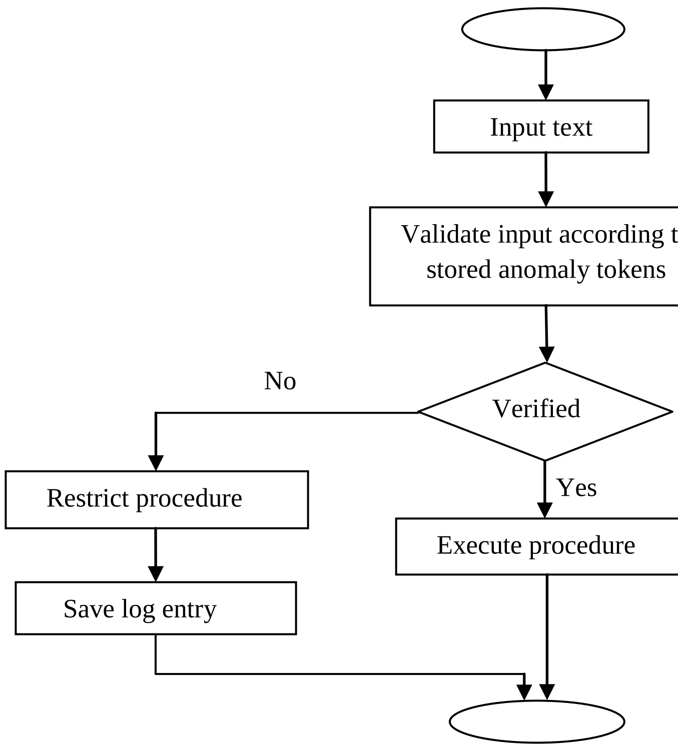


Fig. 1 Architecture of Frontend Phase

But if by any means the attacker gets the ability to compromise the frontend security even then he is not able to access the database because of the backend security phase.

In the backend phase SQLIA has been prevented through database stored procedures. First the tool gets all the stored procedures that are in the database and then analyzes them one by one. The stored procedure code is searched for any hotspots i.e. the point in code where there is vulnerability means possibility of SQL injection. If the tool finds any of such hotspot in the code, it optimizes the code using Transact-SQL technique and updates it in the database. At present the tool works on MSSQL server but in future can be made to work with any server. For MSSQL, the tool searches the code for " + " i.e. concatenation symbol and " ' " i.e. single quote, because they make the code vulnerable and treat the input as a string, so the attacker can merge any malicious string with the parameters in the query. For example:

```
Create procedure dbo.get_user
    @username varchar(45), @password varchar(45)
As
    Declare @sql varchar(max)
 set @sql = 'select * from users where username=' "+ @username+ " ' And password = ' "+
@password + " ' '
Exec (@sql)
```

This example shows how a parameterized stored procedure can be exploited via an SQLIA. In this example, we assume that the query string constructed at lines 5, 6 and 7 of our example has been replaced by a call to the stored procedure defined. As we can see this stored procedure is vulnerable because the attacker can merge any malicious query with the username and password. For example:

SELECT employees FROM users WHERE login='abc' ; SHUTDOWN; -- OR pass=' '

At this point, this attack works like a piggy-back attack. A malicious query gets executed, which results in a database shut down. This example shows that stored procedures can be vulnerable in the same way as traditional application code [1]. But in Transact-SQL any malicious string or query cannot get merged with the parameters because there are no concatenation symbols as mentioned above. For example:

```
 Create procedure dbo.get_user
       @username varchar(45),
       @password varchar(45)
As
 select username, password from users where username= @username and password =  @password
```

This type of code can prevent other types of SQLIA as well. The architecture for proposed backend phase is shown in fig. 4.

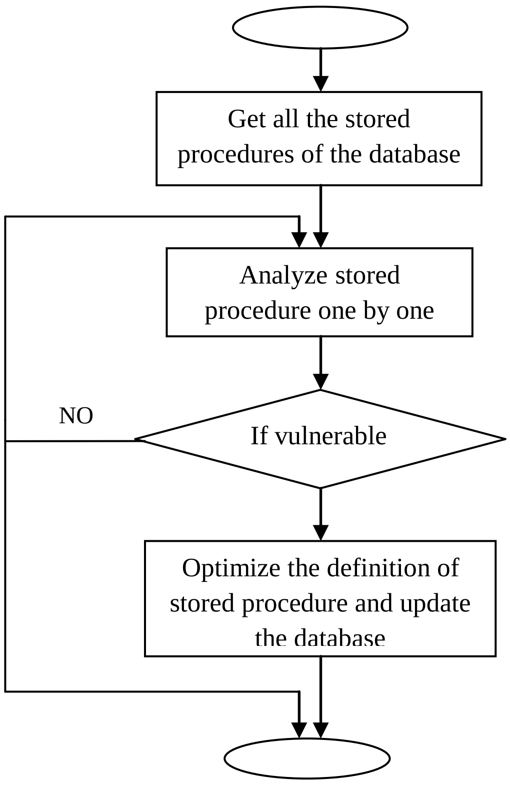

Get all the stored procedures of the database
Analyze stored procedure one by one
NO
If vulnerable
Optimize the definition of stored procedure and update the database

Fig. 4 Architecture of Backend Phase

In this way, the backend phase can prevent SQL Injection attacks.

IV. Result

The proposed solution was implemented in Microsoft SQL server. The performance of our solution based on its ability to counter different attacks is given below in table I.

TABLE I
PERFORMANCE OF PROPOSED SOLUTION

Tautology	yes
Illegal/Logically Incorrect Queries	yes
Union Queries	yes
Piggy-Backed Queries	yes
Stored Procedures	yes
Inference	yes
Altenate Encodings	yes

V. CONCLUSION AND FUTURE WORK

SQL Injection attacks are one of the most dangerous types of threats to web applications. Many solutions to these attacks have been proposed over years. But almost none of them provide security to the full extent of this attack. Also very little emphasis is laid on preventing SQLIA in stored procedures. The proposed solutions for preventing or detecting SQLIA provide security to either application layer or database layer but not to both. We have proposed a technique that provides security to both application layer as well as database layer via frontend phase and backend phase. Researchers have provided this two phase security because if security is compromised at one phase, the second phase can still provide security from attacks. The technique currently works with Microsoft SQL server but in future it can be modified to work with other servers as well.

REFERENCES

[1]. William G.J. Halfond, T. Viegas and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures," in *Proc. of ISSSE06*, 2006.

[2]. Dr. M. Amutha Prabakar, M.KarthiKeyan, Prof.K. Marimuthu, "An Efficient Technique for Preventing SQL Injection Attack Using Pattern Matching Algorithm*," in Proc. of ICECCN*, 2013.

[3]. Gao Jiao, Chang-Ming XU and Jing Maohua, "SQLIMW: a new mechanism against SQL-Injection," in *Proc. of CSSS*, 2012.

[4]. William G.J. Halfond and Alessandro Orso, "AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks," in *Proc. of ASE*, 2005, p. 174–183.

[5]. Sruthi Bandhakavi, Bisht, P. Madhusudan, V.N. Venkatakrishnan, "CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations," in *Proc. of CCS'07*, 2007.

[6]. S. W. Boyd and A. D. Keromytis, "SQLRand: Preventing SQL injection attacks," in *Proc. of ACNS*, 2004.

[7]. Ke Wei, M. Muthuprasanna and Suraj Kothari, "Preventing SQL Injection attacks in stored procedures," in *proceedings of ASWEC*, 2006.

[8]. Amit Kukreti. (2005) SQL Injection Attacks homepage on codeproject. [Online]. Available: http://www.codeproject.com/Articles/11020/SQL-injection-attacks/

[9]. (2013) Malicious SQL Injection: an introduction homepage on hackmac. [Online]. Available: http://www.hackmac.org/tutorials/malicious-sql-injection-an-introduction/