PROTECTION OF WEB APPLICATION AGAINST SQL INJECTION ATTACK

Manisha A. Bhagat[*], Prof. Vanita Mane[**]

[*] Department of Computer Engineering, R.A.I.T.
[**] Department of Computer Engineering, R.A.I.T.

Abstract- Web applications are used by many users.web applications are consist of web forms, web server and backend. These applications are vulnerable due to attacks and scripts as the number of web application users are increasing. Web application can have sensitive and confidential data which is stored in database.web applications accepts the data from the users. This data is retrieved from the database through the queries.SQL Injection attack is one of the most popular attack used in system hacking or cracking. Using SQL INJECTION ATTACK attacker can gain information or have unauthorized access to the system. When attacker gains control over web application maximum damage is caused. This paper illustrates SQLIA methods and prevention and detection tools.

Index Terms- SQLIA, Attacker, SQL injection attach

I. INTRODUCTION

Web application can have sensitive and confidential data which is stored in database.web applications accepts the data from the users. This data is retrieved from the database through the queries.SQL Injection attack is one of the most popular attack used in system hacking or cracking. Web applications can be harmed by SQL INJECTION ATTACK Using SQL INJECTION ATTACK attacker can gain information or have unauthorized access to the system. When attacker gains control over web application maximum damage is caused.

To insert, retrieve, update, & delete the data from database SQL language is used. When we enter data in the input fields it becomes part of the SQL query written at the backend. For example, to login in our inbox, we provide loginid and password. The loginid and password form the part of the internal SQL query. Then the SQL query is executed on the database to check whether the login credentials provided match with those present in the tables on the database. The attacker, who wants to gain access to the inbox, provides injected code instead of correct input in the input fields of the web application. This injected code changes the structure of the original SQL query and consequently, allows the attacker to gain access to the information it was not authorized for. This type of attack which allows the attacker to alter the original SQL query by adding the injected SQL code in the input field is known as SQL Injection Attack (SQLIA).[1].

In SQLIA, Attacker attempts to change SQL query by inserting new SQL keywords. The attacker modifies the original SQL query by inserting new SQL query through user input field. Injected query formed syntactically correct when concatenated with sql command. The data within the database will be altered, extracted or even dropped.

II. CATEGORIES OF SQL INJECTION ATTACK

The injection attacks are divided into three categories. [6].

First order attack- By entering malicious string in input field of web application original SQL query is modified and modified code is executed immediately.

*Unions added to an existing statement to execute a second statement.
*Subquery added to an existing statement.
*Existing SQL short-circuited to bring back all data(for example, adding a query condition such as OR 1=1)

The second order attack- The trusted source such as persistent storage is injected by an attacker as another activity is executed by an attack. The malicious database object is created by attacker such as function called as part of an API or spitefully named table to introduce dangerous constructs by using double quotation marks.

Lateral Injection-The implicit functions can be manipulated by attacker by changing environmental variables. The PL/SQL procedure that does not take user input can be exploited by an attacker. The risk of injection arises when variable whose data type is date or number is concatenated into text of SQL statement. Using NLS_Date_format or NLS_Numeric_characters, the implicit function TO_CHAR can be manipulated.

III. SQL INJECTION PROCESS

The attacker can gain access to web applications using several methods. Through the web application's input fields or hidden parameters the attackers adds SQL statement to access to resources is known as SQL Injection Attack (SQLIA). Due to the lack of input validation in web applications hackers can be successful. Injecting web applications means having illegal access to data stored in database

A.NORMAL PROCESS IN WEB APPLICATION

In normal user input process in web application, user sends request by providing user inputs to the application server. The application server creates the SQL query statement. This SQL

statement is submitted to the backend database. The result is fetched from the database and given back to the user.Fig.1 shows the normal User input process in web application. [9]

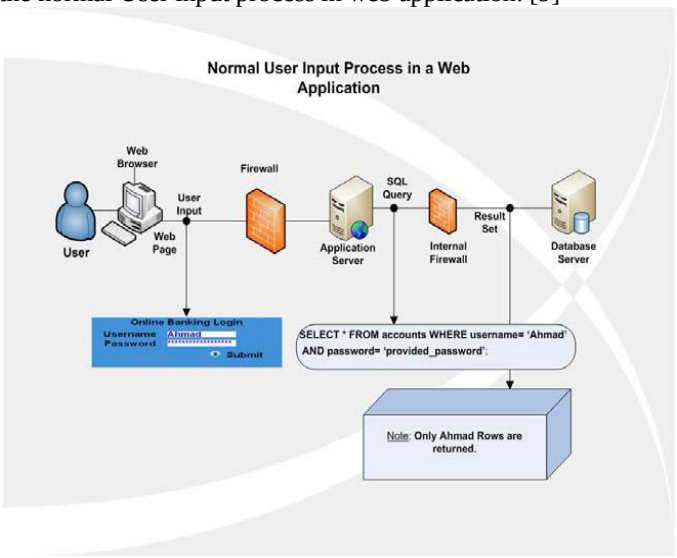

Fig.1 Normal User Input Process in Web Application

B. Malicious Input Process in web Application

In SQLIA, attacker enters malicious input in the input field for example in fig. a attacker enters username as Ahmad OR 1=1- and password as not needed. Because of this malicious input SQL query is altered which is always evaluated to be true. The result of such query will return all the rows of the table.Fig.2 shows malicious input process in web application. [9].

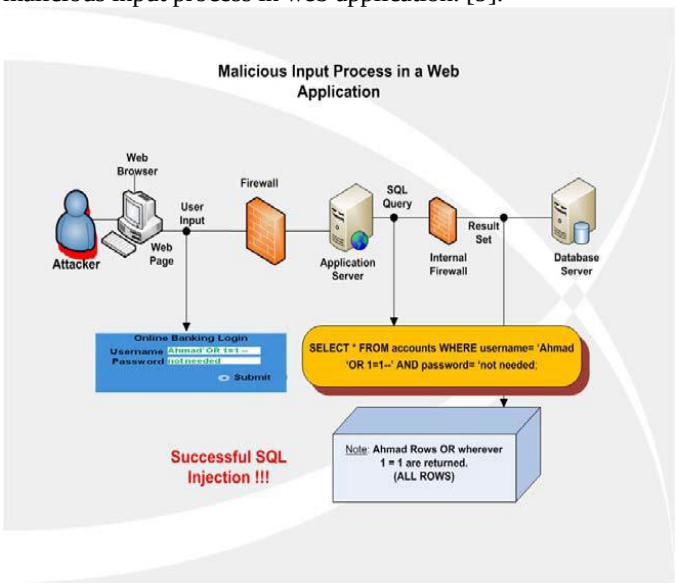

Fig.2 Malicious Input Process in Web Application

IV. SQL INJECTION ATTACK METHODS

Web applications can be attacked by multiple methods. Following are some methods to attack the web applications. ([1],[4],[9]).

1. Tautology based SQL Injection-

Tautology statement is attached to the conditional statement (i.e.1=1') so that it evaluates to true always. Here where clause is vulnerable in SQL query.

Example:

Original query: Select salary from employee where empid='abc' and pwd='xxxxx'

Injected query: Select salary from employee where empid='' or 1=1--' and pwd='not required'
Result: It returns salary of all employees from employee table.

2. Statement Injection-

Original query is altered by injecting new SQL query to the original SQL query.
Example:

Original query: Select salary from employee where empid='abc123' and pwd='xxxxx'

Injected query: Select salary from employee where empid='';
Delete from employee where empid='abc123'--'and pwd='not required'

Result: Record of employee "abc" is deleted.

3. Stored Procedures-

It is group of SQL statements compiled into single execution plan.

Example - Consider the stored procedure below:

```
CREATE PROCEDURE new_dept(new IN varchar2, old IN varchar2)
IS line varchar2(8000);
BEGIN
        line:='begin
                update department set dept='" || new || "'
where dept= '"|| old || "';' || 'END;';
         DBMS_OUTPUT.PUT_LINE ('line: ' || line);
          EXECUTE IMMEDIATE line; END;
```

This procedure has two input fields, old department name and new department name and replaces old name with the new one. The attacker injects the code [„ "; SHUTDOWN;--] in either of the two fields. This injection generates the following query: Update department Set dept ="abc"; SHUTDOWN; -- where dept="aaa" At this stage, the attack behaves like the statement injection attack where the injected query is made to execute with the original query using query delimiter,,;.[1].

4. Illogical/Incorrect queries-

In order to gather information about the internal database structure of application, the attacker deliberately inputs incorrect information in the input fields. The attacker gains the information through the displayed error.

Example:

Original URL: www.samsung.com/proucts?id=23

Injected Query: www.samsung.com/product?id=23'

Result: Error message showed:
SELECT product_name FROM Products WHERE id =23\'.
From the message error we can find out name of table and fields: Products;producr_name. By the gained information attacker can organize more strict attacks.

5. Union query-
The injected query is joined with the injected query by using SQL keyword, UNION, to gather the information from the tables.

Example:

Original Query: select salary from employees where empid='abc123'

Injected Query: select salary from employees where empid='abc123' UNION select * from employee'

Result: It gives records of all employees from employee.

6. Alternate Encoding:
To inject the code, the attacker inserts alternate encoding like ASCII, Unicode, EBCDIC and Hexadecimal to bypass the validation on input.

Example:

Original Query: select * from employee where empid ='abc123' and pwd='xxx'

Injected Query: select * from employee where empid ='; exec (char (0x73687574646f776e))--' and pwd='not required'.

Result: The hexadecimal value for the SHUTDOWN is passed To the char () function. This code will execute the SHUTDOWN command and bypass the input validation.

7. Inference: It is the attack in which data is analyzed in order to illegitimately gain knowledge about database. When a user is able to infer from unimportant information more vigorous information about a database without directly accessing it an inference attack occurs. There are two major types of Inference attack: Blind Injection and Timing Attack.

Blind Injection: This attack asks question which will give answer as true or false based on the applications response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.

Example-When we search for some product in a website, we see something like the following in URL:

Original Query:www.samsung.com/products?id=23

Injected Query: www.samsung.com?id=23 or '1'='0'

This is translated into following SQL query:

Select * from TABLE where id='23' or '1'='0'

Result: This query will return always false which will return error message that infers the information about the tables like table name.
Timing Attacks: The attacker sets the time delay in SQL query through the conditions. If the condition is true, the delay takes place.During this delay attacker gains access to the information.

Example-

Original Query: select*from employee where empid='abc123' and pwd='xxx'

Injected Query: select*from employee where empid='abc123' and ascii (substring (pwd, 1, 1))>z waitfor delay '0:0:5'--'and pwd='not required''

Result: The query will generate delay for 5 seconds if the ascii value of the first character of pwd is greater than the value z.

V. SQL INJECTION DETECTION & PREVENTION TOOLS

To reduce the effect of SQLIA, more research has been done. Many detection and prevention methods have been proposed. Following are some tools invented to detect and prevent the SQL injection attack.

1. JDBC-Checker: It is developed to prevent attacks that take advantage of type mismatch in dynamically generated query string. [1].

2. ADMIRE: It is threat risk model which provides a thorough and step by step technique to identify and moderate the effect of SQL Injection. [1].

3. SQL-PROB: In this tool, SQL proxy based blocker which fetches the user input from SQL query of the application and checks it against syntactic structure of query. It uses proxy that seamlessly integrates with existing operational environments offering protection to front end web server and backend databases. [14].

4. WAVES: It is black box technique for testing web application for SQL injection vulnerabilities The tool identify all points a web application that can be used to inject SQLIA .It builds attacks that target these points and monitors the application how response to attacks by utilizes machine learning. [15].

5. SQLRand: It is a system for preventing SQLIA against web server. The main intension is of using randomized SQL query language to detect and abort the queries that contains injected query.SQL standard keywords are manipulated by appending

the random number that attacker cannot easily guess, to them. In this system one proxy server sits between client web server and SQL server. The de-randomized request is received from client and conveys query to the server. If an SQLIA has occurred, the proxy's parser will fail to recognize the randomized query and reject it. [16].

6. POSITIVE TAINTING: It is identification and marking of trusted data. It tracks the trust marked string and performs syntax aware evaluation i.e. nothing but SQL parsing of query string to differentiate literal and non-literal parts. The string which contains characters without trust marking will not allow to pass database. [17].

7. AMNESIA: It uses combination of static analysis & dynamic analysis to detect and prevent SQLIA.It consists of 4 main steps:

1. Identify hotspot: In this step it scans the application to identify the hotspot point that issue SQL queries underlying database.
2. Build SQL query model: For each hotspot it builds the model that represents the all possible queries that may be generated at that hotspot.
3. Instrument Application:-At each hotspot in application adds call to runtime monitor.
4. Runtime monitoring: It checks the dynamically generated queries against the SQL query model at the run time and reject and report queries that violate the model. [18].

8. SQL DOM: It creates one class per table and for each class table one method per possible operation per column, making the API both insufficient and cumbersome. All database structure mapping information will be access statically to avoid unnecessary object duplication. [19].

9. VIPER: It uses heuristic approach for detecting SQL Injection. It relies on knowledge base of heuristics that guides the generation of SQL queries .Firstly it analyzes the web application with the aim of determining its hyperlinks structure of identifying its input forms. Then it stacks seeding a series of standards SQL attacker. Then it matches the output produced by web application against library of regular expression related to error message that database can produce. It continues the attack using text mined from error message with object of identifying likely table of field name until it is able to retrieve database structure. [20].

10. CANDID: It computes the intended query by running the application on candidate inputs that are self evidently non-attacking. It creates benign sample input (candidate input for every user input. It executes program simultaneously over actual input and candidate input .Generates candidate query along with actual query. Issue actual query only if parse structure matches. [21].

VI. COMPARATIVE ANALYSIS OF SQLIA METHODS AND TOOLS

Depending on the settings of the system configured, every approach has benefits, so it would not be easy to get an idea about which one is best. In table1.[8][10] we show a chart of different approaches against various SQL injection attacks .Table1 shows comparative analysis of SQL injection detection and prevention techniques with attack type. The symbol " " is used for tool that can successfully stop all attacks of that type. The symbol " ○ "is used for tool that attack type only partially because of natural limitations of underlying approach. The symbol – is used for tool that is not able to stop attacks of that type.

Table 1.COMPARATIVE ANALYSIS OF SQLIA METHODS AND TOOLS

Attacks → / **Approaches ↓**	**Tautology**	**Logically Incorrect Queries**	**UnionQuery**	**StatementInjection**	**StoredProcedure**	**BlindInjection**	**TimingAttacks**	**AlternateEncoding**
AMNESIA	●	●	●	●	–	●	●	●
CANDID	○	○	○	○	○	–	–	○
SQLrand	●	–	●	●	–	●	●	–
SQLDOM	●	●	●	●	–	●	●	●
POSITIVE TENTITIVE	●	●	●	●	●	●	●	●
WAVES	○	○	○	○	○	○	○	○
JDBC CHECKER	○	○	○	○	○	○	○	○

VII. CONCLUSION

This paper alerts the people who are related to database maintenance, DBA and other people who are introducing their sites on Internet. This paper gives idea about the hole which can be secured either by code or protection security like firewalls. It is necessary to check the code before introducing the site.

SQL Injection Attacks are dangerous to the applications on Internet. The intention of the attacker is to gain access to the database. We have analyzed all common attack methods and provided illustration for each of them. We have proposed one solution for input validation. That is create one table which contains special characters like; ',--,'.If the input contains such special characters, the SQL query is terminated and is not allowed to be executed on database.

REFERENCES

[1] Neha Singh,Ravindra Kumar Purwar,SQL Injection –A HazardTo web applications, International Journal of Advanced Research in computer Science and Software Engineering,vol.2,Issue 6,June 2012,pp. 42-46.

[2] Permulasway Ramasamy,Dr.Sunitha Abburu,SQL Injection attack detection and prevention, International Journal Of Engineering Science and Technology(IJEST),vol.4,April 2012,pp.1396-1401.

[3] Nikita Patel,Fahim Mohammed,Santosh Soni,SQL Injection attacks Techniques and Protection Mechanism", International Journal on Computer

Science and Engineering (IJCSE), ISSN: 0975-3397,vol 3 No.1,Jan 2011,pp.199-203.
[4] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso,in Classification of SQL Injection Attacks and countermeasures ,ISSSE 2006,March 14th 2006.
[5] San Tsai Sun, Ting Han Wei,Stephen Liu Sheung Lau, Classification of SQL Injection Attack,Nov 17th 2007.
[6] Nilesh Khochare,Santosh Kakade and B.B.Meshramm,Survey on SQL Injection attacks and their Countermeasures,IJCEM international Journal of Computational Engineering & Management,ISSN(Online):2230-7893,vol.14,October 2011,111-114.
[7] William G.J.Halfond And Alessandro Orso,AMNESIA Analysis and Monitoring for Neutralizing SQL Injection Attacks, November 7-11,2005.
[8] Atefeh Tajpour ,Suhaimi Ibrahim,Mohammad Sharifi,Web Application Security by SQL Injection Detection Tools,IJCSI International Journal of Computer Science Issues,vol.9,Issue 2,NO.3,March 2012
[9] Diallo Abdoulaye Kindy and Al-Sakib Khan Pathan,A Detailed Survey on various Aspects of SQL Injection:Vulnerabilities,Innovative Attacks, and Remedies accepted version for information journal.
[10] Abhishek Kumar Baranwal, Approaches to detect SQL Injection and XSS in web applications,EECE 571b,Term Survey paper, April 2012.
[11] V.Shanmughaneethi,S.Swaminathan,Detection of SQL Injection Attack in web applications using web services,IOSR Journal of computer Engineering(IOSRJCE) ISSN:2278-0661 volume 1,Issue 5,May-June 2012,pp.13-20.
[12] Atefeh Tajpour,mohammad JorJor zade Shooshtari,Evaluation of SQL Injection Detection and Prevention Techniques.
[13] Katkar Anjali S.,Kulkarni Raj B.,Web Vulnerability Detection and Security Mechanism ,International Journal of Soft Computing and Engineering(IJSCE)ISSn:2231-2307,volume-2,Issue-4,September 2012,pp.237-241.
[14] Anyi Liu , Yi Yuan , Duminda Wijesekera , Angelos Stavrou,SQLProb: A Proxy-based Architecture towards Preventing SQL Injection Attacks,
[15] Atefeh Tajpour , Suhaimi Ibrahim,Mohammad Sharifi,Web Application Security by SQL Injection Detection Tools,IJCSI,International Journal Computer Science Issues,Vol.9,Issue 2,No.3,March 2012,332-339
[16] Stephen W. Boyd,Angelos D. Keromyti,SQLrand:Preventing SQL Injection Attacks.
[17] Devata R. Anekar ,Prof. A. N. Bhute,SQL Injection Detection and Prevention Mechanism using Positive Tainting and Syntax Aware Evaluation, *International Journal of Advances in Computing and Information Researches, ISSN:2277-4068, Volume 1– No.3,August 2012*
[18] William G.J.Halffond,Alessandro Orso,Preventing SQL Injection Attacks Using AMNESIA,ICSE,2006,Shanghai,China
[19] Etinene Janot ,Pavol Zavarsky,Preventing SQL Injection in online applications:Study,Recommendations and Java Solution Prototype based on SQL DOM,Application Security Conference,Ghent,Belgium,19-22 May 2008.
[20] Angelo Ciampa,Corrado Aaron Visaggio,Massimiliano Di Penta,A heuristic-based approach for detecting SQL Injection vulnerabilities in Web applications,ICSE Capetown,2-8 May 2010,pp 43-49.
[21] Sruthi Bandhakavi,Prithvi Bisht,P. Madhusudan,V.N. Venkatakrishnan, CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations

AUTHORS

First Author –Manisha A.Bhagat, M.E. (Computer pursuing), Department of Computer Engineering, R.A.I.T., Nerul
Email:bhagat.manisha@yahoomail.com
Second Author – Vanita Mane, M.E. (Computer), Department of Computer Engineering, R.A.I.T., Nerul,
Email:vanitamane1@gmail.com