International Journal For Technological Research In Engineering *ISSN (Online): 2347 - 4718*
3rd International Conference on Emerging Technologies in Engineering, Biomedical, Medical and Science (ETEBMS – July 17)

SQL INJECTION : VULNERABILITY AND APPROACHES FOR ITS PREVENTION

S.Nagalakshmi
Assoc Prof, Dept of Information Science and Engineering, DrAmbedkar Institute of Technology, Bangalore

Abstract: The security of Web applications is important; a security assessment shows that Web applications are vulnerable to some form of attack. Some of the vulnerability in web applications is caused by permitting unchecked input to take control and exploit the application. The databases of organizations store the data needed to deliver specific content to the visitors. Web applications and databases allow an organization to regularly run businesses on Internet. SQL injection attack is a major threat to web applications. It is one of the mechanisms used by hackers to destroy data of organizations. By manipulating user input attackers can expose the back end data base of a web application. The scheme of attack is to convince the susceptible application to run SQL code that was not intended. If the application is creating SQL strings naively on the fly it is in for some serious consequences. Our paper discusses the various ways in which SQL can be injected into an application, the issues that are related to this class of attack and the various techniques in which these SQL injection attacks can be prevented.
Keywords: SQL Injection, Blind SQL Injection.

I. OVERVIEW

STRUCTURED Query Language (SQL) is a general standard query language for operating , querying and administering databases . Database systems like Microsoft SQL Server, Oracle, or MySQL provide backend functionality to web applications. Businesses, individuals, and governments use web applications that offer effective, efficient and reliable solutions for efficient communication and also for conducting online commerce like online banking, enterprise collaboration, and supply chain management sites. Legitimate website visitors can submit and retrieve data over the Internet using a web browser. Important user information, finance and payment details, statistical data of companies are all made resident within a database and can be accessed by users of web applications who are authorized. Businesses depend heavily on their online application, database management system which holds their data, hardware and the underlying operating system for providing ht maximum benefit through the online interface [1]. This makes the web application vulnerable to any kind of attack [2]. Hence the security of Web applications is increasingly important.

A. What is SQL Injection attack?

The user information is dynamically built into SQL statements used to interact with the databases which provide database support for web applications. A SQL injection attack is a destructive technique which is associated with altering SQL statements and taking advantage of susceptible web applications with maliciously coded attacker data. Some of the weaknesses in web applications that attackers take advantage are inadequate input validation, vulnerable SQL statements exposing them to SQL injection attacks and thereby breaking into the database.[3] In an SQL injection attack the user of an application form enters SQL code with malicious intent and this code is wrapped up in special characters and that code entered doesn't get used for the purpose the application had intended as a web application, it has the capacity to corrupt the database associated with the application and maybe to even destroy it. When attacker enters the data into the form, that data is directly used to build a dynamic SQL query to retrieve the data from database. Such code intended to be malicious in nature forms an SQL Injection attack. In these attacks the attacker gains access to restricted data by unauthorized means and uses snooping techniques through advanced querying to potentially overcome authentication and has a control over the web application and an unauthorized access to the database server. It is important that web applications validate the clients input before executing SQL queries. Hackers test and identify SQL injection vulnerabilities by sending the application data input that would cause the server to generate invalid SQL query. In case the server returns an error message to the client, the attacker tries to back track portions of the original SQL query using information gained from these error messages.

Therefore when this exploit is successful, a user gets a query to run on the server. The result can be:

- Table dropping (deletion of databases or tables in the system)
- Data loss or modification
- Modification of any stored procedure, query, or rule
- Log deletion or modification
- User management like adding removing users, modifying permissions for access.

One of the ways for the application to maintain safety from such attacks is to refrain from displaying database server messages.

B. Types of SQL Injection attacks

SQL injection attacks can be categorized broadly into two types:

First Order attacks: In this type of attacks the results are

available to the attacker immediately after the attack as a response directly from the affected application through an Email or a messaging system.

Second Order attacks: In this order of attack the code with malicious intent which was injected into the web application is executed much later by the way of storing in a cache log or in the database itself. It is later retrieved and used as part of a vulnerable SQL statement. The Second order attacks are more difficult to locate.

SQL injection attacks can generally cause:

i. Authentication Bypass: Without any user id and password authentication the attacker has logged onto the application.

ii. Information Disclosure: The attacker could view important data in the database.

iii. Data Integrity compromised: The attacker could alter the contents of a database.

iv. Compromised Availability of Data: The attacker could delete information.

II. WORKING OF SQL INJECTION

The attacker finds a parameter that the application passes through to the database and carefully embeds malicious SQL commands into the content of the parameter, i.e. by carefully embedding malicious SQL commands into the content of the parameter, the attacker tricks the web application into forwarding a malicious query to the database and SQL injection flaw is exploited.

For example, consider the login form which accepts the username and password from the user.

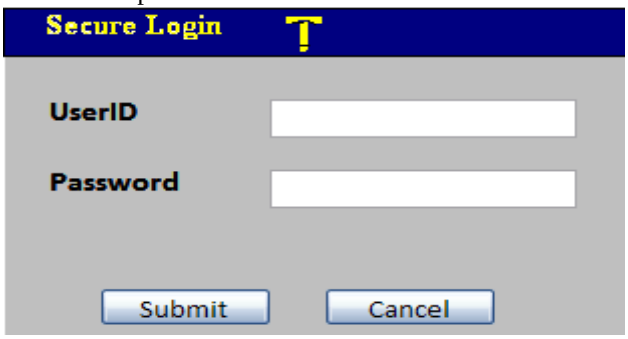

Fig 3.1: Snapshot of Secure Log On

The fields in the form containing data of the User Identification and Password provide values to build SQL query which runs at the database level:

*SELECT * FROM members WHERE username = ' & name & ' AND password = ' & password'*

Now, Suppose the user entered the Username as "Admin" and Password as "manager". Then the query will become:

*SELECT * FROM members WHERE name = 'Admin' AND password = 'manager'*

This code will work absolutely ok , but suppose the user supplies a poorly created string of code then that will allow the attacker to step around the authentication and access the information from the database like for example the attacker tries something like *username=' OR 1=1—* then the query will be sent as:

*SELECT * FROM members WHERE name = '' OR 1=1--' AND password = ' ';* It works as shown in Table 3.1

Sl No	Character	Description
1	'	**Closes the user input field.**
2	OR	**Continues the SQL query so that the process should equal to what come before OR what come after.**
3	1=1	**A statement which is always true.**
4	**--**	**Comments outs the rest of the lines so that it won't be processed.**

Table 3.1: Description of Query

The data filled is used in the WHERE clause as the application is not really thinking about the query – but really only constructs a string – and the use of OR has turned a single-element WHERE clause into a two-element clause, the 1=1 clause makes sure is always true irrespective of what the first clause is. This query becomes "Select all the fields data from the table Customers if the name equals "nothing" Or if 1=1 and anything else that follows this line is to be ignored.

The server is fooled into allowing the attacker to have more access than necessary as the query which reaches the server has received a true statement. The code associated with reference to the password input field would never be run by the server and hence the authentication is never done or is bypassed in these attacks.

A. Methods of SQL Injection

The two methods with which SQL Injections are made through are :

- Normal SQL Injection
- Blind SQL Injection

Normal SQL Injection

In this method of SQL injection, when an attacker tries to an execute SQL Query, at times the server returns an error page to the user describing the type and cause of the error in detail. The attacker uses this information contained in the error messages returned as a response by the database server [4] and matches his query with the query used by the developers in the application being attacked.

Blind SQL Injection

Blind SQL injection is similar to normal SQL Injection except that when an attacker attempts to take advantage of an application, instead of getting a useful error message, gets a very general message page with some developer specifications instead. This makes the task of SQL injection attack on the web application more difficult for the attacker though not exactly impossible.. The attacker would still try to steal data through series SQL statements. Using Blind SQL Injection, an attacker could investigate the database to obtain sensitive information, or to alter database contents say alter authentication data. One of the examples of the Blind SQL Injection technique is to say introduce delay as part of a malicious SQL statement or build a SQL statement causing the database server to perform a time-consuming action. If the web application is taking unusually longer to respond

then the database allows the attackers to understand that their SQL statements are being executed with some level of certainty.

III. CATEGORIES OF SQL INJECTION ATTACKS

The main categories of SQL Injection attacks against databases [5]. They are:

- SQL Manipulation
- Code Injection
- Function Call Injection
- Buffer Overflows

SQL Manipulation

This is the most common type of SQL Injection attack. Here the attacker tries to modify the existing SQL statement by inserting additional conditions to the WHERE clause or extending the SQL statement using operators associated with union of data , intersection of data or MINUS. [6]

For example if the original query in the application was intended to be

*SELECT * FROM members*
WHERE membername = 'abc'
And PASSWORD = 'abc '

The attacker tries to manipulate the SQL statement to execute as:

*SELECT * FROM members*
WHERE membername = 'abc '
AND Password = 'abc ' or 'a' = 'a'--

The WHERE clause returns true for every row and the attacker therefore has gained access to the application.

Code Injection

This type of attack appends additional SQL statements to the existing SQL statement. The EXECUTE statement in some Database Management systems become a frequent target of SQL injection attacks. Oracle Databases are more robust to Code Injection attacks.

In PL/SQL and Java, Oracle does not support multiple SQL statements per database request. Thus, the following code injection attack will not work against an Oracle database either in PL/SQL code or in the Java application. This statement will result in an error:

*SELECT * FROM members WHERE membername = 'abc'*
AND Password = 'abc'; DELETE FROM members
WHERE username = 'admin';

Injecting Function calls through SQL

The injected function calls could be used to make operating system calls or manipulate data in the database, allow native functions to be accessed, run functions residing in the database through SQL commands which could severely compromise sensitive data; passwords allow the attacker to run transactions etc. Simple SQL statements can be used to effectively compromise the data and exploit the system.

Buffer Overflows

Buffer overflow is effectively used by attackers to gain access to a remote operating system through an SQL injection attack. Most web servers that hang up do not gracefully handle when the database connection to a database is lost due to a buffer overflow. Usually, the web process enters into a hang state until the connection to the client is stopped or terminated, thus making this a very effective denial of service attack.

IV. PREVENTING SQL INJECTION ATTACKS

User input with weak validation when is directly used by the developers of the web applications makes it prone to SQL Injection attacks so also highly privileged database logins. Some of the common mistakes which make an application susceptible to SQL Injection attacks are [7] :

- Weak input validation.
- Use of parameters which cause the dynamic construction of SQL statements.
- Use of database logins for which additional precautions haven't been provided.

Small programming precautions help in challenging SQL Injection attacks. Developers need to be disciplined enough to apply the precautionary methods to every web accessible procedure and function that make up the web application. Every dynamic SQL statement must be protected. Even a single unprotected SQL statement can result in compromising of the application and the entire system.

Some Programming techniques used for preventing SQL injection attacks:

Bind Variables must be used

Bind variables are often cited as the key to solid application performance Bind variables provide a powerful protection against SQL injection attacks..They are used to improve application performance. Application coding standards require their usage in all SQL statements. SQL statements shouldn't be created through concatenating together strings, passed parameters etc. or A second order complex SQL injection attack could possibly exploit an application by storing a string in the database, which would be later executed by a dynamic SQL statement Bind variables which are placeholders in a SQL statement are replaced with a valid value or value address for the statement to execute successfully.

Validate the Input

User input has to be validated more so the string parameter should be validated. Before sending the user data to the server, validations have to be applied at the client level itself. Again the data passed from the client form must be checked for its validity at the server level too before it is submitted as a query to the database server for execution. At any stage if the data doesn't clear the validation processes, it must be rejected with appropriate error messages. For Web applications using hidden fields and other techniques, extra care has to be taken to provide for input validation. When a bind variable is not being used, special database characters must be removed or escaped. Oracle databases have an issue with a single quote. Here the simplest method is to escape all single quotes It is important to note that escape of single quotes and the use of bind variables should not be done for the same string. A bind variable content is stored as it is in

the input string in the database and escaping any single quotes will result in double quotes being stored in the database.

Function Security

Inbuilt functions as well as functions and procedures created for use by web applications for eg. functions to perform operations like changing passwords or creating users could be exploited in SQL injection attacks. These functions whether user defined or inbuilt could effectively aid the attackers. Any function that isn't necessary to the functioning of the application should be restricted.

Limit the Open-Ended Input

As far as possible select text or options are provided instead of text boxes and the number of text boxes is limited. All client side validations must be performed before sending the client data to the server. Only the options that have been provided in the select box should be considered and user should not be allowed to enter text as option.

Verify the Type of Data

Data has to be verified using functions for its data type too before passing it into a SQL statement. In case of string data all the single quotes are replaced with two single quotes

Use Stored Procedures

Web applications security can be enhanced against SQL injection attacks, by not allowing the client-supplied data to modify the syntax of SQL statements. The web application is isolated from SQL by storing all the PL/SQL contents as stored procedures in database server. For example the application can execute these stored procedures using the interface provided by JDBC or ADO's Command Object. PreparedStatements and stored procedures compile the SQL statement before the user input is added, thereby it is not possible for user input to modify the actual SQL statement.

WebInspect may be used for .NET and WebSphere Studio applications in order to automate the discovery of security vulnerabilities as applications are built, helping in remediation of vulnerabilities, and delivering of secure code for final quality assurance testing..

It is important to discover the security vulnerabilities at an early stage and address the issue which in turn reduces the overall cost of secure application deployment, and enhances overall organizational security.

Additional programming practices

- User input shouldn't be directly used to create dynamic SQL statements. The user data which is not should not be concatenated with SQL statements, and must be validated.
- Unnecessary characters like NULLs, new line character, backslashes etc must be removed from user input and parameters from URL.
- User Accounts should have well defined privileges.
- Length of the user input should be limited.
- Input containing suspicious characters should be rejected.

Researchers have proposed tools using which developers may compensate for any shortcomings implemented through defensive coding [8, 9, 10].

WAVES is a black-box technique used for testing web applications for any SQL injection attacks it was proposed by Huang and colleagues [11]. This tool identifies all the points of a web application that can be used to inject SQL injection attacks. It builds attacks which target these points and monitors how the application responds to the attacks and thus this machine learning is used enhance the security of web applications.

JDBC- Checker [10, 12] can be used to prevent SQL injection attacks that use type mismatches in a dynamically-generated query strings.

CANDID's [13, 8] natural and simple approach turns out to be very powerful for detection of SQL injection attacks. CANDID is a tool used for Web applications developed in Java. This tool dynamically digs into the programmer-queries for any input and detects attacks by comparing it against the structure of the actual query issued.

SQL Guard [9] and SQL Check [14]check queries at runtime with a base model expressed as a grammar accepting only legal queries. In SQL Guard the queries are checked before the addition of the user input and examine the structure of the query after the addition of user-input based on the model. In SQL Check, developer specifies the model. The use of a secret key to delimit user input during parsing by the runtime checker is used in both the approaches. The security of this approach is dependent on attackers not being able to determine the key.

AMNESIA [15,16] makes use of an analysis in static state and a check at runtime. The static phase includes a variety type of queries which an application can legally generate at each point of access to the database are used to build models. In the dynamic phase, queries are intercepted and then sent to the database where they are checked against the statically built models. Queries that do not adhere to the model are denied access to the database. A possible limitation of this tool is that its success is highly dependent on the accuracy of the queries provided during the static analysis phase.

WebSSARI [17] checks taint flows against preconditions for sensitive functions using static analysis. It's functioning is based on sanitized input which has passed through a set of predefined filters. The limitation of such an approach is that many preconditions for sensitive functions cannot be expressed with enough accuracy leading to a set of insufficient filters.

SecuriFly [18] is another tool that was built for web applications developed using java. Unlike other tools, SecuriFly chases strings instead of characters for taint information. It attempts to clean up those query strings which have been created using tainted input. However, injection in numeric fields cannot be stopped by this approach. The primary limitation of this tool lies in the difficulty of identifying all sources of user input.

Intrusion Detection System (IDS) [19] identifies SQL Injection attacks, based on a machine learning technique. The machine learning creates a model of typical queries. The knowledge gained in creating this model is used by the IDS at runtime and any queries that do not match the model

would be classified as an attack. This tool could be deemed successful in detecting attacks but is solely dependent on training seriously if otherwise false positives and false negatives would produce erroneous results. SQL-IDS [20] focuses on writing specifications for the web application which describes the developer perspective of the structure of SQL statements as needed in the application, and an automatic monitoring of the execution of the SQL statements for violations with respect to these specifications is done.

SQLPrevent [3] is a tool which prevents SQL Injection attacks with the use of an HTTP request interceptor. When SQLPrevent is deployed into a web server the flow of the data is changed. A current thread stores all the HTTP requests, after which the SQL interceptor passes the SQL statements made by the web application to the SQL Injection Attack detector module. On examination if the HTTP request from thread-local storage fetched is suspected to contain any malicious SQL statement, it would be prevented to be sent to database.

V. CONCLUSION

SQL Injection attack methodology targets the resident data in an applications database. This attack tries to modify the parameters of a Web -based application whilst altering the SQL statements, to retrieve, modify and delete data from the database. Exploits occur due to coding errors as well as inadequate validation checks. Prevention involves ensuring better coding practices and efficient database administration procedures. Hence we conclude that the application developer has to always patch and keep updating any loopholes in the application as the attacker is waiting to exploit.

REFERENCES

[1] DorinCarstoiu, Elena Lepadatu, Mihai Gaspar, "Hbase - non SQL Database, Performances Evaluation", Journal of IJACT, Vol. 2, No. 5, pp. 42-52, 2010.

[2] T. Pietraszek and C. V. Berghe. Defending against Injection Attacks through Context-Sensitive String evaluation. Recent Advances in Intrusion Detection , Volume: 3858, Pp: 124-145, 2006.

[3] BogdanCarstoiu, DorinCarstoiu. Zatara, the Plug-in-able Eventually Consistent Distributed Database, Journal of AISS, Vol. 2, No. 3, pp. 56-67, 2010.

[4] www.arcanesecurity.net/component/docm

[5] http://www.net-necurity.org/dl/articles/ IntegrigyIntrotoSQLInjectionAttacks.pdf

[6] http://msdn.microsoft.com/en-us/library/ff648339.aspx

[7] W. G. Halfond, J. Viegasand A. Orso, "A Classification of SQL Injection Attacks and Countermeasures," College of Computing Georgia Institute of Technology IEEE, 2006.

[8] P. Bisht, P. Madhusudan.CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. ACM Transactions on Information and System Security Volume: 13, Issue: 2, 2010.

[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) , 20.

[10] C. Gould, Z. Su, and P.Devanbu. JDBC Checker:A Static Analysis Tool for SQL/JDBC Applications. In Proceedings of the 26th International Conference on Software Engineering (ICSE 04) Formal Demos ,pp 697–698, 2004.

[11] Y. Huang, S. Huang, T. Lin, and C. Tsai. A Testing Framework for Web Application Security Assessment. Journal of Computer Networks, Volume: 48 Issue: 5, Pp: 739-761, 2005.

[12] Wassermann, G; Gould, C; Su, Z, et al. Static Checking of Dynamically Generated Queries in Database Applications. ACM Transactions on Software Engineering and Methodology .-- Volume: 16, Issue: 4, 2007.

[13] SruthiBandhakavi, PrithviBisht, P. Madhusudan, CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluation Proceedings of the 14th ACM conference on Computer and communications security.ACM, Alexandria, Virginia, USA.page:12-24.

[14] Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications. ACM SIGPLAN Notices. Volume: 41, pp: 372-382, 2006.

[15] W. G. Halfond and A. Or so. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In Proceedings of the IEEE and ACM International Conference on Automated Software Engineering (ASE 2005) , Long Beach, CA, USA, Nov 2005.

[16] W. G. Halfond and A. Orso. Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks. In Proceedings of the Third International ICSE Workshop on Dynamic Analysis (WODA 2005), pp 22–28, St. Louis, MO, USA, May 2005.

[17] Y. Huang, F. Yu, C. Hang, C.H. Tsai, D. T. Lee, and S. Y. Kuo. Securing Web Application Code by Static Analysis and Runtime Protection. In Proceedings of the 12th International World Wide Web Conference (WWW 04) , May 2004.

[18] M. Martin, B. Livshits, and M. S. Lam. Finding Application Errors and Security Flaws Using PQL: A Program Query Language. ACM SIGPLAN Notices , Volume: 40, Issue: 10, pp: 365-383, 2005.

[19] F. Valeur, D. Mutz, and G. Vigna. A Learning-Based Approach to the Detection of SQL Attacks. Detection of IntruMalware, And Vulnerability Assessment, Proceedings, Volume: 3548, pp: 123-140, 2005.

[20] AtefehTajpour, Suhaimi Ibrahim, Maslin Masrom,

"Evaluation of SQL Injection Detection and Prevention Techniques". International Journal of Advancements in Computing Technology, 2011, Korea.

[21] Marco Cova, DavideBalzarotti. Swaddler:An Approach for the Anomaly-based Detection of State Violations in Web Applications.Recent Advances in Intrusion Detection, Proceedings , Volume: 4637 Pages: 63-86 Published: 2007.

[22] William G.J. Halfond, Jeremy Viegas and Alessandro Orso, "A Classification of SQL Injection Attacks and Countermeasures," College of Computing Georgia Institute of Technology IEEE , 2006.

[23] K. Kemalis and T.Tzouraman is. SQL-IDS: A Specification-based Approach for SQL Injection Detection Symposium on Applied Computing.2008, pp: 2153-2158 , Fortaleza, Ceara, Brazil. New York, NY, USA: ACM.

[24] A. S. Christensen, A. Moller, and M. I. Schwartzbach. Precise Analysis of String Expressions . In Proc. 10th International Static Analysis Symposium, SAS '03, volume 2694, pp 1-18. Springer-Verlag, June 2003.

[25] F.Monticelli, PhD SQLPrevent thesis. University of British Columbia (UBC) Vancouver, Canada.2008.

[26] unixwiz.net/techtips/sql-injection.html

[27] http://community.livejournal.com/database_root/ 881 .html

[28] en.wikipedia.org/wiki/SQL_injection

