ISSN:0975-9646

Detection and Prevention of SQL Injection attack

Manish Kumar , L.Indu

Computer Science and Engineering,
P. B. College of Engineering, Sriperumbudur-602 105

***Abstract*— SQL injection is a technique where the attacker injects an input in the query in order to change the structure of the query intended by the programmer and gaining the access of the database which results modification or deletion of the user's data. In the injection it exploits a security vulnerability occurring in database layer of an application. SQL injection attack is the most common attack in websites in these days. Some malicious codes get injected to the database by unauthorized users and get the access of the database due to lack of input validation. Input validation is the most critical part of software security that is not properly covered in the design phase of software development life-cycle resulting in many security vulnerabilities. This paper presents the techniques for detection and prevention of SQL injection attack. There are no any known full proof defences available against such type of attacks. In this paper some predefined method of detection and the some modern techniques of preventions are discussed. This paper also describes countermeasures of SQL injection.**

***Keywords*— web application, SQLIA, detection, prevention, vulnerabilities, web architecture,**

I. INTRODUCTION

Now a days web application is widely used in various applications it is the reliable and efficient solution to the challenges of communicating and conducting the various organisation, business or commerce over the internet. Now each and every important assignment is done by using the web application which is connected through the internet. For example electricity bill, online shopping, gaming, banking, messaging, shopping, conferences, etc. So the increase of web application involving the various security issues in the web world.

The SQLIA (structured query language injection attack) is a code injection attack technique commonly used for attacking websites in which an attacker injects some SQL codes in place of the original codes to get access the database. The open web application security project (OWASP) ranks SQLI as the most widespread website security risk in 2011. The National Institute of Standards and Technology's National vulnerability Database reported 289 SQL vulnerabilities in websites including those of IBM, HP, and MICROSOFT. In December 2011, SANS Institute security experts reported a major SQL injection attack that affects approximately 160000 websites using Microsoft's Internet Information Services (IIS), ASP.NET, and SQL Server Frameworks.

There are variety of techniques are available to detect SQLIA. The most preferred are Web Framework, Static Analysis, Dynamic Analysis, combined Static and Dynamic Analysis and Machine Learning Technique. Web Framework provides filters to filter special characters but other attacks are not detected. Static Analysis checks the input parameter type, but it fails to detect attacks with correct input type. Dynamic Analysis technique is capable of scanning vulnerabilities of web application but is not able to detect all types of SQLIA. Combined Static and Dynamic Analysis includes the benefit of both, but this method is very complex in order to proceed. Machine Learning method can detect all types of attacks but results in number of false positives and negatives.

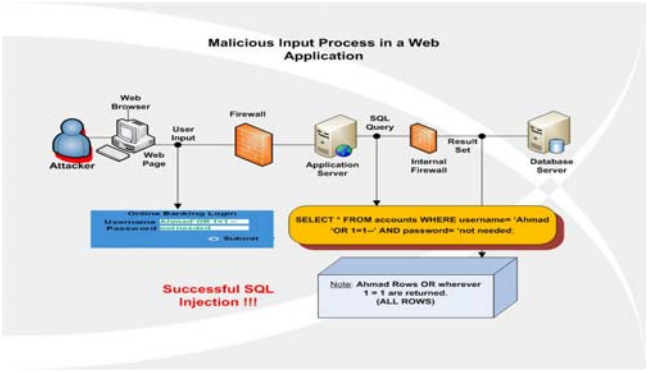


Fig – successful SQLIA

II. SQLIA MECHANISMS

Malicious SQL statements can be introduced into a vulnerable application using many of different input mechanisms. These are the most common mechanisms

2.1 Injection through user input: In the type of injection the attacker injects SQL commands by providing suitably crafted user input. A web application can read user's input in several ways based on the environment in which the application is deployed.

2.2 Injections through cookies: Cookies are the small files that containing state information generated by Web applications and stored on the client machine. When a client returns to the Web application the cookie is used to be restore the client information. Since the client has control over the storage of cookie, a malicious client could tamper with the cookie's content. And then if Web application uses the cookie content to build SQL queries, an attacker could easily submit an attack by embedding it in the cookie.

2.3 *Injections through the server variables:* Server variables are collections of variables that contain HTTP, network headers, and environmental variables .Web applications used these server variables in a variety of ways like logging usage. If these servers logged to a database without sanitization, this could create SQLI vulnerability

because attacker can forge the values that are placed in HTTP and network headers. They can exploit this vulnerability by placing an SQLIA directly into the headers. And when the query to log the server variable is issued to the database, the attack in the forged header is triggered automatically.

2.4 *Second order injection:* In second order injection, attacker seed malicious inputs in to a system or database to indirectly trigger an SQLIA when that input is used at a later time. The attack takes place when the malicious input reaches to the database.

III. CLASSIFICATION OF SQLIA

*3.1 Tautology***:** In the tautology attack the attacker tries to use a conditional query statement to be evaluated always true. Attacker uses WHERE clause to inject and turn the condition into a tautology which is always true. The simplest form of tautology

Example

SELECT *FROM Accounts WHERE user=''or1=1—'AND pass=''AND eid=

The result would be all the data in accounts table because the condition of the WHERE clause is always true.

3.2 Illegal/Logical Incorrect queries: When a query is rejected an error message is returned from the database including useful debugging information. This information helps attackers to make move further and find vulnerable parameters in the application and consequently database of the application.

Example

SELECT * FROM Accounts WHERE user=' ' AND pass=' 'AND eid =convert(int,(SELECT TOP 1name FROM sysobjects WHERE xtype='u'))

In the example the attacker attempts to convert the name of the first user defined table in the metadata table of the database to 'int'. This type conversion is not legal therefore the result is an error which reveals some information that should not be shown.

3.3 Union queries: In this type of queries unauthorised query is attached with the authorised query by using UNION clause.

Example

SELECT * FROM Accounts WHERE user='' UNION SELECT *FROM Students—'AND pass=''AND eid=

The result of the first query in the example given above is null and the second one returns all the data in students table so the union of these two queries is the student table.

3.4 Piggy-Backed query: In the query attack attacker tries to add an additional queries in to the original query string .In this injection the intruders exploit database by the query delimiter, such as ";", to append extra query to the original query

Example

SELECT*FROM Accounts WHERE user='';drop table Accounts—'AND pass=' ' AND eid=

The result of the example is losing the credential information of the accounts table because it would be dropped branch from database.

3.5 Inference: In this type of attack, intruders change the behaviour of a database of application. These are the well known types of inference

3.5.1 Blind Injection: This is little difficult type of attack for attacker. During the development process sometime the developer hides some error details which help the attacker to compromise with database. In this situation the attacker face the generic page provided by developer in place of an error message

Example

SELECT * FROM Accounts WHERE user='user1'AND1=1 - -'AND pass=' 'AND eid=

During injection it is always evaluated as true if there are no any error message, and the attacker realizes that the attack has passed user parameter is vulnerable to injection.

3.5.2 Timing attack: In the Timing attack the attacker gathers information about the response time of the database. This technique is used by executing the if-then statement which results the long running query or time delay statement depending upon the logic injected in database and if the injection is true then the "WAITFOR" keyword which is along with the branches delays the database response for a specific time.

Example

SELECT * FROM Accounts WHERE user='user1' AND ASCII (SUBSTRING((SELECT TOP 1 name FROM sysobjects),1,1))>X WAITFOR DELAY '000:00:09'- -'AND PASS=' ' AND eid=

In the example the attacker trying to find the first character of the first table by comparing its ASCII value with X . if there is a 9 second delay he realize that the answer to his question is yes. So by continuing the process the name of the first table would be discovered.

3.6 Alternate encoding: In this type of attack the regular strings and characters are converted into hexadecimal, ASCII and Unicode. Because of this the input query is escaped from filter which scans the query for some bad character which results SQLIA and the converted SQLIA is considered as normal query.

Example

SELECT * FROM Accounts WHERE user='user1'; exec(char(0x8774675u8769e)) - -' AND pass=' ' AND eid=

The example char () function and ASCII hexadecimal encoding are used. The functions will get integer number as a parameter and return as a sample of that character. In the example it will return "SHUTDOWN", so whenever the query is interpreted the SHUTDOWN command is executed.

3.7 Stored procedure: Stored procedure is the built in extra abstraction layer on the database defined by the programmer. By using the stored procedure the user can store its own function according to the need. It is extending the functionality of database and interacting with the system operating system. Then the attacker tries to identify the underlying database in order to exploit the database information.

Example

SELECT * FROM Accounts WHERE user=' '; exec xp_logininfo ' BUILTIN\Administrators'; - - ' AND pass=' ' AND eid=

In this example the built in stored procedure "xp_logininfo" is executed in order to get the information about the BUILTIN\Administrators windows group.

IV DETECTION SQLIA

Several ways to detect the SQLIA vulnerabilities are:

4.1 code based detection techniques: This approach generally occupies for developing test suit based on codes for detecting the SQLI vulnerabilities .But the suit does not find vulnerable program points explicitly.

SQLUnitGen is a prototype tool that uses static analysis tool to generate the user input to database access point and generate unit test report contacting SQLIA patterns for these points.

MUSIC (mutation based SQL injection vulnerability checking) it uses nine mutation operators to replace original queries with mutated queries. Jose fonseca, Marco Vieira, Henrique Madeira developed a tool. This tool automatically detects the mutated queries and runs the test tool after it generated the test results after the detection.

4.2 concrete attack generations: This type of approach uses state of art symbolic execution techniques to automatically generate test inputs that expose SQLI vulnerability in Web program.

The symbolic execution based approaches use constraint solvers that can only handle numeric operation. Because inputs of Web applications are string by default .If a constraint solver can solve myriad string operations applied to inputs, developers could use symbolic execution to both detect the vulnerability of SQL statements that use inputs and generate concrete inputs that attack them.

4.3 Taint-based vulnerability detection: SQLIA can be avoided by using static and dynamic technique to prevent tainted data from affecting untainted data, such as programmer –defined SQL query structures.

Several of researchers have applied prominent static analysis techniques such as flow sensitive analysis, context sensitive analysis, alias analysis and interprocedural dependency analysis, to identify input sources and database access points and check whether every flow from a source to a sink is subject to an input validation and /or input sanitization routine, but these approaches have some limitations. They do not consider input validation using prediction, fail to specify vulnerability patterns.

Gary Wassermann and Zedong Su used context free grammar to model the effects of input validation and sanitization routines .Their techniques checks whether SQL queries syntactically confine the string values returned from those routines and, if so, automatically concludes that the routines used are correctly implemented.

V PREVENTION SQLIA

5.1 Defensive coding: Developers have approached a range of code based development practices to counter SQLIA. These techniques are generally based on proper input filtering, potentially harmful character and rigorous type checking of inputs.

5.1.1 *Manual defensive coding practices*: Based on the security reports such as OWSAP's SQL cheat sheet and Chris Anley's white paper provide useful manual defensive coding guidelines.

Parameterized queries or stored procedures:The attacker take advantage of dynamic SQL by replacing the original queries and create some parameterized query in database. These attacks force to developer for first define the SQL code structure before including parameters in query. Because parameters are bound to the defined SQL structure, thereafter it is not possible to inject additional SQL code

Escaping: If dynamic queries cannot be avoided, escaping all user-supplied parameters is the best option. Then the developer should identify the all input sources to define the parameter that need escaping, follow database-specific escaping procedures, and use standard defining libraries instead of the custom escaping methods.

Data type validation: After following the steps for the parameterized query and escaping the developer must properly validate the input data type. The developer must define the input data type is string or numeric or any other type and input data given by user is incorrect then it could easily reject.

White list filtering: Some of the special character which is normally used during injection .so the developer should characterise such special character as the *black list filtering*. The filtering approach is suitable for the well structured data. Such as email address, dates, etc. and developer should keep a list of legitimate data patterns and accept only matching input data.

5.1.2 SQL DOM: The manual defensive coding is the best way to avoid the SQLIA. The approach SQL DOM is introduced by Russell McClure and Ingolf Kruger. In the SQL DOM uses the encapsulation of database queries to provide a safe way to avoid the SQLIA problem by changing the query building process from one that uses string concatenation to a systematic one that uses a type-checked API. In the process a set of classes that enables automated data validation and escaping. Developers provide their own database schema and construct SQL statement using its API's.

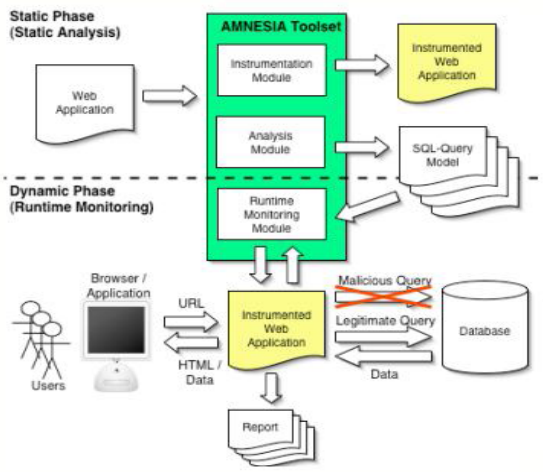
It is especially useful when the developer needs to be using the dynamic SQL in place of the parameterized queries for getting flexibility.

5.2 *Runtime prevention*: Runtime prevention may be more complex than the defensive coding .Because some of the approaches require code instrumentation to enable runtime checking. But it is able to prevent from all SQLIA.

5.2.1 *Randomization*: The approach is proposed by Boyd and Keromytis in which randomized SQL query language is used, pointing a particular CGI in an application, where a proxy server used in between the SQL server and Web server. It sends SQL query with a randomized value to the proxy server, which is received by the client and de-randomized and sends it to the server. This technique has two main advantages is security and portability. But if the random value is predicted then it is not useful.

5.2.2 *Learning based prevention*: This approach is based on a runtime monitoring system deployed between the application server and database server, it intercept all queries and check SQL keywords to determine whether the queries syntactic structure are legitimate before the application sends them to the database .

Static Analysis or AMNESIA (analysis for monitoring and neutralizing SQL injection attacks): This approach is suggested by Hal fond W.G, Orso.A, it is model based approach to detect illegal query before execution into database.



Schematic diagram of AMNESIA

This technique uses program analysis to automatically build a model of legitimate queries that could be generated by the application .And its dynamic part the technique uses runtime monitoring to inspect the dynamically-generated queries and check them against the statically built model. The main drawback of the model is that it requires modification of the Web application source code for successful collaboration with the security monitor officer.

Dynamic analysis : The statically inferred query structures might not be accurate. And the attackers could attack in the weakness .So the dynamic analysis can provide more accuracy. It can locate the vulnerabilities of SQLIA without any source code modification.

SQL Check: SQLCheck tracks tainted data at runtime by marking it with metacharacters. And when a Web application invokes a query, SQLCheck learns the query legitimate structure by excluding marked data from it.

SQL Prob: SQLProb executes a program of interests with various valid inputs to collect all possible queries that might legitimately appear during runtime. During runtime it usage a global pair wise alignment algorithm to compare issued user queries against those in the legitimate query repository and extracts the user inputs.

CANDID: CANDID dynamically mines a program's legitimate query structure at each path by executing the program with the valid and nonattacking inputs and thereafter comparing actual issued query with the legitimate query structure mined for the same path.

VI CONCLUSIONS

In this paper various types of SQL injection mechanism, detection type and prevention techniques are discussed .we found that there is no one complete foolproof solution to database security and have some issues hard to eliminate .Any organization that attempts to secure a database system, must consider the security of the overall environment including the communication channel, user access methods, the database, and any application which is used to access the database

As all we can say a well thought –out combination of hardware and software solutions with modern database security approach need to be implemented to make modern database system more secure.

ACKNOWLEDGMENT

I am sincerely thankful to Asst professor L.INDU for support and guidance .and I also like to thank the management of P.B. College of Engineering for their support to carry out this work efficiently.

REFERENCES

[1] Justin Clarke, *SQL Injection Attacks and Defense*, Second Edition,Syngress Publication, July 2, 2012,ISBN-13: 978-1597494243

[2] Inyong Lee , Soonki Jeong Sangsoo Yeoc, Jongsub Moond, "A novel method for SQL injection attack detection based on removing SQL query attribute", *Journal Of mathematical and computer modeling, Elsevier* 2011.

[3] W.G.J. Halfond, J. Viegas, and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures," *Proc. Int'l* Symp. Secure Software Eng.(ISSSE 06),IEEECS, 2006; www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf.

[4] S. W. Boyd and A. D. Keromytis,"SQLrand: Preventing SQL Injection Attacks", in *Proceedings of the 2nd Applied Cryptography and Network Security Conference*, pages 292–302, Jun. 2004.

[5] W. G. Halfond and A. Orso, "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.

[6] Varian Luong "Intrusion detection and prevention system: SQL injection attacks", 2010.

[7] Wi11iamGJ.Halfond,JeremyViegas,AlessandroOrs. AClassifieatio of SQL Injeetion Attacks and Countermeasures. In Proceedings of the IEEE International Symposium on Secure Software Engineering,2006.

[8] David Morgan. Web Injection Attacks[J]. Network Security,2006**.**

[9] Ferruh Mavituna. Deep Blind SQL Injection. Portcullis security. com2008.

[10] *Advanced SQL injection* [Online]. Available: http://www.nextgenss.com/papers/advanced_SQL_injection.pdf.

[11] Atefeh Tajpour et al. "Evaluation of SQL Injection Detection and Prevention Techniques" Second International Conference on Computational Intelligence, 2010.