International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 4, April - 2013

SQLIA: Attack's By SQL Injection Attack And Their Detection Mechanism

Pushkar Y Jane
M.Tech IV Sem. CSE
Smt.Bhagwati Chaturwedi College of Engg .Nagpur

M. S. Chaudhari
Assst.Prof.& HOD CSE.
Smt.Bhagwati Chaturwedi College of Engg. Nagpur

Abstract

The uses of web application has become increasingly popular in our daily life as reading news paper, reading magazines, making online payments for shopping etc. At the same time there is an increase in number of attacks that target them. In particular, SQL injection, a class of code injection attacks in which specially crafted input strings result in illegal queries to a database, has become one of the most serious threats to web applications. This paper proposes a novel specification-based methodology for the prevention of SQL injection Attacks. The two most important advantages of the new approach against existing analogous mechanisms are that, first, it prevents all forms of SQL injection attacks; second, Current technique does not allow the user to access database directly in database server. The innovative technique "Web Service Oriented XPATH Authentication Technique" is to detect and prevent SQL Injection Attacks in database the deployment of this technique is by generating functions of two filtration models that are Active Guard and Service Detector by using web service.

Keywords:*Database security , world-wide web, web application security, SQL injection attacks, Runtime Monitoring*.

1.Introduction:

Nowadays the world connected to every person by the internet. In these situations the web security is very important and it is a challenging part of the web applications. A number of techniques are in use for securing the web applications. The most common way is the to validatation process of a user and authentication process through the username and password.

One of the major problems in the authentication process is the input validation checking . There are some major threads in web application security for example SQL injection .A SQL injection is the one of the type of code injection, by which attacker used the malicious keyword to get unauthorized access over the web application.SQL injection is too much vulnerable that it can bypass many traditional security layers like Firewall, encryption, and traditional intrusion detection systems. It can also bypass the database mechanisms of authentication and authorization. SQL injection can not only be used for violating the security by seeing the private data of the people but also can be used for bypassing the authentication of user which is a big flaw in the web security applications..Major problem in the web applications is the SQL injection, it is to be considered that SQL injection is an easy attack and every developer or internet user can easily perform the attack by using malicious strings or keywords that is the 'SQL Injection' which is the most worrying aspect of the today's world.

Login page is the most complicated web application which give the permission to the users to enter into the database after authenticating him. In this stage, the user provides his identity proof such as username and password. There might be some invalid input validations which can bypass the authentication process using some method like SQL injection.

2. Related Work:

In order to protect a Web application from SQL Injection attacks, there are two major concerns. Firstly, there is a great need of a mechanism to detect and exactly identify SQL Injection attacks. Secondly, knowledge of SQL Injection Vulnerabilities (SQLIVs) is a must for securing a Web application. So far, many frameworks have been used and/or suggested to detect SQLIVs in Web applications. Here, we mention the some existing prominent solutions and their working methods.

William G.J.Halfond et al.'s Scheme[6]- This approach works by combining static analysis and runtime monitoring. In its static part, technique uses program analysis to automatically build a model of the legitimate queries that could be generated by the application. In its dynamic part, technique monitors the dynamically generated queries at runtime and checks them for compliance with the statically-generated model. Queries that violate the model represent potential SQLIAs and are thus pre- vented from executing on the database and reported.

SAFELI – Proposes a Static Analysis Framework in order to detect SQL Injection Vulnerabilities. SAFELI framework aims at identifying the SQL Injection attacks during the compile-time. This static analysis tool has two main advantages. Firstly, it does a White-box Static Analysis and secondly, it uses a Hybrid-Constraint Solver. For the White-box Static Analysis, the proposed approach considers the byte-code and deals mainly with strings. For the Hybrid-Constraint Solver, the method implements an efficient string analysis tool which is able to deal with Boolean, integer and string variables.

Thomas et al.'s Scheme[8] - Thomas et al., in suggest an automated prepared statement generation algorithm to remove SQL Injection Vulnerabilities. They implement their research work using four open source projects namely: (i) Net-trust, (ii) ITrust, (iii) WebGoat, and (iv) Roller. Based on the experimental results, their prepared statement code was able to successfully replace 94% of the SQLIVs in four open source projects.

Ruse et al.'s Approach[12] - Ruse et al. propose a technique that uses automatic test case generation to detect SQL Injection Vulnerabilities. The main idea behind this framework is based on creating a specific model that deals with SQL queries automatically. Adding to that, the approach identifies the relationship (dependency) between sub-queries. Based on the results, the methodology is shown to be able to specifically identify the causal set and obtain 85% and 69% reduction respectively while experimenting on few sample examples.

Ali et al.'s Scheme[11] - 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.

Parse Tree Validation Approach - Buehrer et al. adopt the parse tree framework. They compared the parse tree of a particular statement at runtime and its original statement. They stopped the execution of statement unless there is a match. This method was tested on a student Web application using SQLGuard. Although this approach is efficient, it has two major drawbacks: additional overheard computation and listing of input (black or white).

Dynamic Candidate Evaluations Approach - In, Bisht et al. propose CANDID. It is a Dynamic Candidate Evaluations method for automatic prevention of SQL Injection attacks. This framework dynamically extracts the query structures from every SQL query location which are intended by the developer (programmer).

Hence, it solves the issue of manually modifying the application to create the prepared statements.

2. Types Of SQLIA:

There are multiple methods by which a Web application can be attacked. We discussed each of these methods in detail in next section to illustrate how each of them is used to attack the database of the application.

a. Tautologies:

Tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The most common usages of this technique are to bypass authentication pages and extract data. If the attack is successful when the code either displays all of the returned records or performs some action if at least one record is returned.

Example: In this example attack, an attacker submits " ' or 1=1 - -".The Query for Login mode is:

SELECT * FROM user_info WHERE loginID='' or 1=1 -- AND pass1='' .

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology the query evaluates to true for each row in the table and returns all of them. In our example, the returned set evaluates to a not null value, which causes the application to conclude that the user authentication was successful. Therefore, the application would invoke method user_main.aspx and to access the application.

b. Union Query:

In union-query attacks, Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query> because the attackers completely control the second/injected query they can use that query to retrieve information from a specified table. The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the results of the injected second query.

Example: An attacker could inject the text "' UNION SELECT pass1 from user_info where LoginID='secret - -" into the login field, which produces the following query:

SELECT pass1 FROM user_info WHERE loginID='' UNION SELECT pass1 from user_info where LoginID='secret' -- AND pass1=''

Assuming that there is no login equal to "", the original first query returns the null set, whereas the second query returns data from the "user_info" table. In this case, the database would return column "pass1" for account "secret". 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 "pass1" is displayed along with the account information .

c. Stored Procedures:

SQL Injection Attacks of this type try to execute stored procedures present in the database. Today, most database vendors ship databases with a standard set of stored procedures that extend the functionality of the database and allow for interaction with the operating system. Therefore, once an attacker determines which backend database is in use, SQLIAs can be crafted to execute stored procedures provided by that specific database, including procedures that interact with the operating system. It is a common misconception that using stored procedures to write Web applications renders them invulnerable to SQLIAs. Developers are often surprised to find that their stored procedures can be just as vulnerable to attacks as their normal applications.

Additionally, because stored procedures are often written in special scripting languages, they can contain other types of vulnerabilities, such as buffer overflows, that allow attackers to run arbitrary code on the server or escalate their privileges.

CREATE PROCEDURE BO.UserValid(@LoginID varchar2, @pass1 varchar2 AS EXEC("SELECT * FROM user_info HERE loginID='" +@LoginID+ "' and pass1='" +@pass1+ "'");GO

Example: This example demonstrates how a parameterized stored procedure can be exploited via an SQLIA.

The stored procedure returns a true/false value to indicate whether the user's credentials authenticated correctly. To launch an SQLIA, the attacker simply injects " ' ; SHUTDOWN; --" into either the LoginID or pass1 fields. This injection causes the stored procedure to generate the following query:

SELECT * FROM user_info WHERE loginID='secret' AND pass1='; SHUTDOWN; --

At this point, this attack works like a piggy-back attack. The first query is executed normally, and then the second, malicious query is executed, which results in a database shut down. This example shows that stored procedures can be vulnerable to the same range of attacks as traditional

application codeThe stored procedure returns a true/false value to indicate whether the user's credentials authenticated correctly. To launch an SQLIA, the attacker simply injects " ' ; SHUTDOWN; --"

into either the LoginID or pass1 fields. This injection causes the stored procedure to generate the following query:

SELECT * FROM user_info WHERE loginID='secret'

AND pass1='; SHUTDOWN; --

At this point, this attack works like a piggy-back attack.

The first query is executed normally, and then the second, malicious query is executed, which results in a database shut down. This example shows that stored procedures can be vulnerable to the same range of attacks as traditional application code.

d. Alternate Encodings :

Alternate encodings do not provide any unique way to attack an application they are simply an enabling technique that allows attackers to evade detection and prevention techniques and exploit vulnerabilities that might not otherwise be exploitable. These evasion techniques are often necessary because a common defensive coding practice is to scan for certain known "bad characters," such as single quotes and comment operators. To evade this defense, attackers have employed alternate methods of encoding their attack strings (e.g., using hexadecimal, ASCII, and Unicode character encoding). Common scanning and detection techniques do not try to evaluate all specially encoded strings, thus allowing these attacks to go undetected. Contributing to the problem is that different layers in an application have different ways of handling alternate encodings. The

application may scan for certain types of escape characters that represent alternate encodings in its language domain. Another layer (e.g., the database) may use different escape characters or even completely different ways of encoding. For example, a database could use the expression char(120) to represent an alternately-encoded character "x", but char(120) has no special meaning in the application language's context. An effective code-based defense against alternate encodings is difficult to implement in practice because it requires developers to consider of all of the possible encodings that could affect a given query string as it passes through the different application layers. Therefore, attackers have been very successful in using alternate encodings to conceal their attack strings.

Example: Because every type of attack could be represented using an alternate encoding, here we simply provide an example of how esoteric an alternatively-encoded attack could appear. In this attack, the following text is injected into the login field: "secret'; exec(0x73687574646f776e)

- - ". The resulting query generated by the application is:

SELECT * FROM user_info WHERE loginID='secret';
exec(char(0x73687574646f776e)) -- AND pass1='' .

This example makes use of the char() function and of ASCII hexadecimal encoding. The char() function takes as a parameter an integer or hexadecimal encoding of a
character and returns an instance of that character. The stream of numbers in the second part of the injection is the ASCII hexadecimal encoding of the string "SHUTDOWN." Therefore, when the query is interpreted by the database, it would result in the execution, by the database, of the SHUTDOWN command.

e. Deny Database service:

This attack used in the websites to issue a denial of service by shutting down the SQL Server. A powerful command recognized by SQL Server is SHUTDOWN WITH NOWAIT. This causes the server to shutdown, immediately stopping the Windows service. After this command has been issued, the service must be manually restarted by the administrator.

select password from user_info where LoginId=';shutdown with nowait; --' and Password='0'

The '--' character sequence is the 'single line comment' sequence in Transact - SQL, and the ';' character denotes the end of one query and the beginning of another. If he has used the default sa account, or has acquired the required privileges, SQL server will shut down, and will require a restart in order to function again. This attack is used to stop the database service of a particular web application.

3. Proposed Technique:

This Technique is used to detect and prevent SQLIA's with runtime monitoring. The solution technique are that for each application, when the login page is send to checking module , it was to detect and prevent SQL Injection attacks without stopping legitimate accesses. Moreover, this technique may be efficient, a low overhead on the Web applications. The contribution of this system is as follows:

A new automated technique for preventing SQLIA's where no code modification required, Web service which has the functions of db_2_XMLGenrerator and XPATH_Validator such that it is an XML query language to select specific parts of an XML document. XPATH is simply the ability to traverse nodes from XML and obtain information. It is used for the temporary storage of sensitive data's from the database, Active Guard model is used to detect and prevent SQL Injection attacks. Service Detector model allow the Authenticated or legitimate user to access the web applications. The SQLIA's are captured by altered logical flow of the application. Innovative technique monitors dynamically generated queries with Active Guard model and Service Detector model at runtime and check them for compliance. If the Data Comparison violates the model then it represents potential SQLIA's and prevented from executing on the database. This proposed technique consists of two filtration models to prevent SQLIA'S. 1) Active Guard filtration model 2) Service Detector filtration model. The steps are summarized and then describe them in more detail in following sections.

1. Active Guard Filtration Model: Active Guard Filtration Model in application layer build a Susceptibility detector to detect and prevent the Susceptibility characters or Meta characters to prevent the malicious attacks from accessing the data's from unauthorized user of the database.

2. Service Detector Filtration Model: Service Detector Filtration Model in application layer validates user input from XPATH_Validator where the Sensitive data's are stored from the Database at second level filtration model. The

user input fields compare with the data existed in XPATH_Validator if it is identical then the Authenticated /legitimate user is allowed to proceed.

3. Web Service Layer: Web service builds two types of execution process that are DB_2_Xml generator and XPATH_Validator. DB_2_Xml generator is used to create a separate temporary storage of Xml document from database where the Sensitive data's are stored in XPATH_ Validator, The user input field from the Service Detector compare with the data existed in XPATH_ Validator, if the data's are similar XPATH_ Validator send a flag with the count iterator value = 1 to the Service Detector by signifying the user data is valid otherwise the user is no.

Experimental Results:

The following modules shows the how the injection attacks are performed and how the susceptibility characters or malicious keyword are detected in the login page module.In this stage attacker will perform the SQL injection attack on the LOGIN PAGE. And will try to get unauthorized access over the database. Here the attacker performing the attack in the LOGIN PAGE: ***User Name :-- Pankaj Password :-- pankaj***

If the user name and password will match the the user will be considered as a valid user and permission will granted to enter into the database. And the information will be shown to him as per his requirement.Now if the user or attacker performing the SQL injection attack then all information of the database will be shown to him.(Shown in diagram 02).

LOGIN PAGE: Attacker performing SQL injection attack.

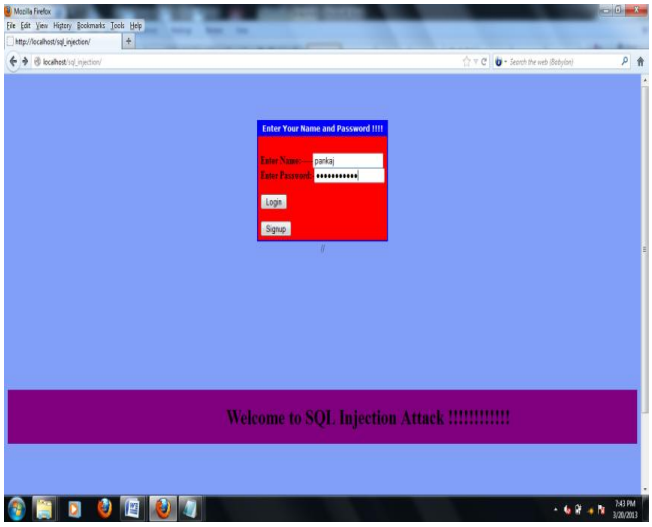

Fig. 1 Login Page

Result of SQL injection attack:.

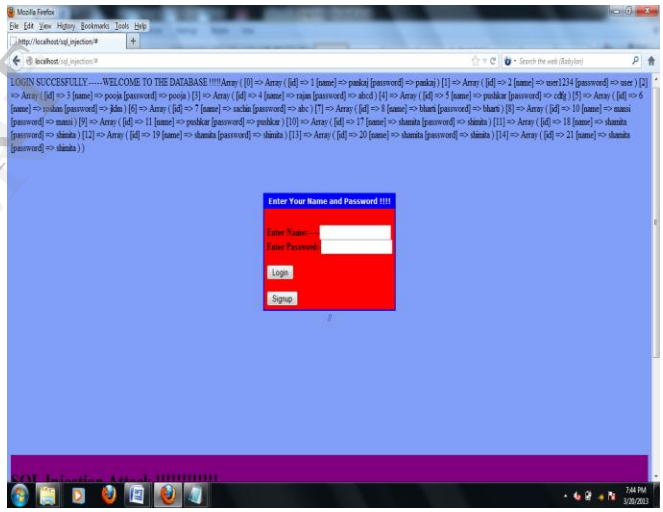

Fig. 2 After SQL injection attack

Detection Phase: In detection phase we are detecting the SQL injection attack. If attacker using the various methods of attacking to the database then he will be unsuccessful in that and does not get the permission to enter into the database. The figures below shows the detection phase. If the attacker trying to get access of database by using SQL injective statement i.e with the help of injective symbols , expression

etc. ,then that a attack will be detected in this module**.**

Result after implementing Active Guard Filteration Model:

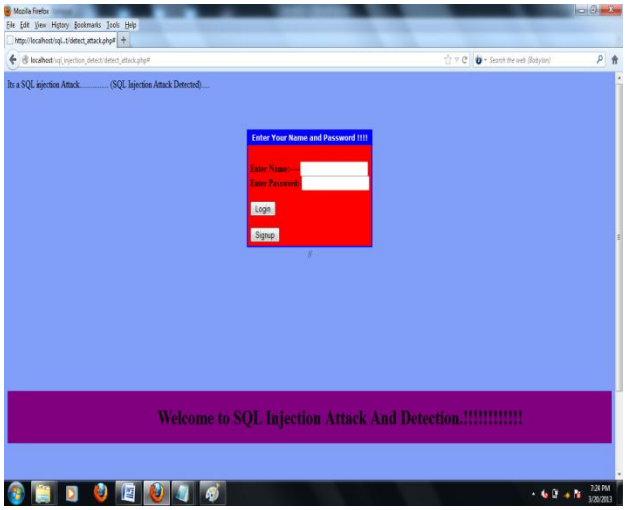

Fig 3. After applying Active guard Model

4.Conclusion:

It is obvious from above description that SQL injection attacks are one of the largest classes of security problems. Most existing technique either require developers to manually specify the interfaces to an application or, if automated, are often inadequate when applied to modern,complex web applications. We proposed a new technique based on X-PATH authentication method. We shown the results of , user get the access of database after the SQL injection Attack ,that takes the access of database directly after the injection. In the next step we have shown the detection of SQL injection attacker by applying method the method of Active Guard filtration model. If attacker trying to the SQL attack then that attack will be detected.

References

[1] Indrani Balasundaram, Dr.E.Ramaraj "An Approach to Detection of SQL Injection Attacks in Database Using Web Services"(IJCSNS ,VOL. 11 No.1,January 2011).

[2] Rahul Shrivastava, Joy Bhattacharyji, Roopali Soni "SQL INJECTION ATTACKS IN DATABASE USING WEB SERVICE: DETECTION AND PREVENTION – REVIEW" Asian Journal Of Computer Science And Information Technology 2: 6 (2012) 162 – 165. Also Available at http://www.innovativejournal.in/index.php/ajcsit.

[3] Shubham Srivastava, Rajeev Ranjan Kumar Tripathi "Attacks Due to SQL Injection & Their Prevention Method for Web-Application" (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (2) , 2012,3615-3618.

[4]Prasant Singh Yadav, Dr pankaj Yadav, Dr. K.P.Yadav "A Modern Mechanism to Avoid SQL Injection Attacks in Web Applications" (IJRREST Volume-1 Issue-1, June 2012)

[5] V.Shanmughaneethi ,S.Swamynathan "Detection of SQL Injection Attack in Web Applications using Web Services" (ISSN : 2278-0661 Volume 1, Issue 5 (May-June 2012), PP 13-20).

[6] William G.J.Halfond and Alessandro Orso "AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks"

[7] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao. A StaticAnalysis
Framework for Detecting SQL Injection Vulnerabilities, COMPSAC 2007, pp.87-96, 24-27 July 2007.

[8] 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).

[9] A.SRAVANTHI, K.JAYASREE DEVI,K.SUDHA REDDY, A.INDIRA, V.SATISH KUMAR "DETECTING SQL INJECTIONS FROM WEB APPLICATIONS" [IJESAT Volume-2, Issue-3, 664 – 671].

[10] Diallo Abdoulaye Kindy and Al-Sakib Khan Pathan "A SURVEY ON SQL INJECTION: VULNERABILITIES, ATTACKS, AND PREVENTION TECHNIQUES"

[11] Shaukat Ali, Azhar Rauf, Huma Javed "SQLIPA:An authentication mechanism Against SQL Injection"

[12] M. Ruse, T. Sarkar and S. Basu. Analysis & Detection of SQL Injection Vulnerabilities via Automatic Test Case Generation of Programs. 10th Annual International Symposium on Applications and the Internet pp. 31 – 37 (2010)

[13] Sruthi Bandhakavi,Prithvi Bisht,P. Madhusudan,V.N. Venkatakrishnan "CANDID: Preventing SQL Injection Attacks usingDynamic Candidate Evaluations."