Scientific Journal of Impact Factor(SJIF): 3.134
e-ISSN(O): 2348-4470
p-ISSN(P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 1,Issue 12, December -2014

PREVENTION OF SQL INJECTIONS FROM WEB APPLICATIONS

Ketan S. Chavda

Assistant Professor, Computer Engineering Department, C.U. Shah College of Engineering and Technology, Wadhwancity, meetketan33@gmail.com

Abstract— *In recent year's web applications usage is increased. We need to emphasize to achieve integrity, confidentiality and availability of web applications. In web applications, data privacy and end user privacy is one of the most controversial legal issues. Backend database being the main target for external attacks such as SQL injection attacks, there is an actions need to take to handle such attacks to secure stored information. A novel approach is proposed by us to handle SQL injection using framework. Two levels of classes are used in framework to detect the SQL injection which check data type of variables, detect the characters and keywords through which SQL injections is happen.*

Keywords—*SQL injection attacks, Web application security, Malicious code, Vulnerability, Database*

I. INTRODUCTION

The Internet is growing now a day, most of the people are not aware of security and privacy. Web applications are used by organizations to provide services like online shopping, online banking and social networking; over the recent years our dependence on web applications has increased drastically in our everyday routine activities. Web applications should be reliable and secure when we are shopping online, paying bills, making transactions etc. Web applications consists of underlying databases which contains confidential user's information like financial information records, medical information records, personal information records which are highly sensitive and valuable, which in turn makes web applications an ideal target for attacks. SQL injection attacks, is one of the major security threats to web applications [1, 4]. In this attack attackers access the database underlying the web applications retrieve, modify and delete confidential user information stored in the database resulting in security violations, identity theft, etc. SQL injections attacks occur when data provided by the user is included directly in a SQL query and is not properly validated. Attackers take advantage of this improper input validation and submit input strings that contain specially encoded database commands. Application builds a query using these vulnerable strings and submits the query to its database, the attacker's embedded commands are executed by the database and the attack succeeds [2]. SQL injections pose a serious threat to the data security of the Web applications. These attacks are the most popular and harmful to databases. In SQL injections, the attacker provides SQL code instead of the valid input in the input fields of the web form in order to change the meaning of the original SQL query issued by the database [3].

II. BACKGROUND ON SQL INJECTION ATTACKS

SQL Injection Attack (SQLIA) occurs when an attacker changes SQL query by inserting new SQL keywords or operators. There are two important characteristics of SQLIAs that we use for describing attacks: injection mechanism and attack intent.

A. Injection Mechanisms

Vulnerable SQL statement can be injected in application using different input mechanism.

Injection through cookies:

Cookies contain state information stored on the client machine and generated by Web applications. Cookies can be used to restore the client's state information. If the client has control over the storage of the cookie then malicious client can use cookie's contents. If a client uses the cookie's contents to build SQL queries, it can easily submit an attack on web application [5].

Injection through user input:

Attackers inject SQL commands by using user input. In most SQLIAs which target the Web applications, user input typically comes from submissions that are sent to the Web application via HTTP GET or POST requests [6]. These requests contain the user input which is accessed by web application.

First order Attacks

The first order attacks are basic attacks. It is When UNIONS or Sub query added to the existing statement.

Second order Attacks

In the second order attacks attacker insert the malicious code into the application but not activated immediately by the application. The objective of this kind of attack differs significantly from a first order injection attack. Attacks in second order injection are not occurring when the malicious input initially reaches the database. Instead, attackers have knowledge of where the input will be subsequently used and craft their attack so that it occurs during that usage.

Second-order injections can be especially difficult to detect and prevent because the point of injection is different from the point where the attack actually manifests itself. A developer may properly escape, type-check, and filter input that comes from the user and assume it is safe. Later on, when that data is used to build a different type of query, the previously sanitized input may result in an injection attack.

B. Attack Intent

Attacks can be characterized based on the goal, or intent, of the attacker.

Performing database finger-printing:

The attacker wants to discover the type and version of database of Web application. Certain types of databases respond differently to different queries and attacks, and this information can be used to find type and version of the database.

Identifying injectable parameters:

The attacker identifies which parameters and user-input fields are vulnerable to SQL injection attacks.

Determining database schema:

Attacker often needs to know database schema information, such as table names, column names, and column data types to correctly extract data from a database. Attacks with this intent are created to collect or infer this kind of information.

Extracting data:

These types of attacks in which attackers extract data values from the database. Attacks with this intent are the most common type of SQL injection attacks.

Adding or modifying data:

The goal of these attacks is to add or change information in a database.

Performing denial of service:

These attacks are performed to shut down the database of a Web application, thus denying service to other users. Attacks involving locking or dropping database tables also fall under this category.

Evading detection:

This category refers to certain attack techniques that are employed to avoid auditing and detection by system protection mechanisms.

Bypassing authentication:

The goal of these types of attacks is to allow the attacker to bypass database and application authentication mechanisms. Bypassing such mechanisms could allow the attacker to assume the rights and privileges associated with another application user.

Executing remote commands:

These types of attacks attempt to execute arbitrary commands on the database. These commands can be stored procedures or functions available to database users.

III. SQL INJECTION ATTACKS TYPES

There are various types of SQL injection attacks. Below is the description of each attack types [7].

A. Tautologies

A SQL tautology is a statement that is always true. The general goal of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. Tautology-based SQL injection attacks are usually used to bypass user authentication or to retrieve unauthorized data by inserting a tautology into a conditional statement. The

general goal of a tautology-based attack is to inject SQL tokens that cause the query's conditional statement to always evaluate the true. For example,

*"Select * from Employee where EmpName = ' ' or 1=1 -- and Password= 'xxxxx'"*

The 'or 1=1' is the most commonly known tautology.

B. Piggy-Backed Queries

In the piggy-backed Query attacker tries to append additional queries to the original query string. On the successful attack the database receives and executes a query string that contains multiple distinct queries. In this method the first query is original whereas the subsequent queries are injected. This attack is very dangerous; attacker can use it to inject virtually any type of SQL command. For example,

"SELECT info FROM employee WHERE login='abc' AND pin=0; drop table employee"

Here database treats above query string as two query separated by ';', and executes both. The second sub query is malicious query and it causes the database to drop the employee table in the database. There are so many other types of queries such as inserting new employees in the database.

C. Logically Incorrect Queries

This attack takes advantage of the error messages that are returned by the database for an incorrect query. These database error messages often contain useful information that allow attacker to find out the vulnerable parameter in an application and the database schema. Suppose after inserting an incorrect query if attacker gets following error message,

"Microsoft OLEDB provider for SQL Server (0×80040E07) Error converting nvarchar value 'CreditCards' to a column of data type int"

Now there are two useful pieces of information in this error message. First, the attacker come to know that the database in in SQL server database. Second, attacker comes to know that the name of the first user-defined table in the database is 'CreditCards'. So by using the same strategy attacker can find the name and type of each column in the given table.

D. Union Query

Union query injection is called as statement injection attack. In this attack attacker insert additional statement into the original SQL statement. This attack can be done by inserting either a UNION query or a statement of the form ";< SQL statement >" into vulnerable parameter. The output of this attack is that the database returns a dataset that is the union of the results of the original query with the results of the injected query. For example,

*"Select * from users where UserName=' ' union select * from employee -- and Password='anypwd'"*

The above query becomes the union of two SELECT queries. Here first query returns a null set because of no matching records in the table USERS. The second query returns all the data from the table EMPLOYEE.

E. Stored Procedure

In this technique, attacker focuses on the stored procedures which are present in the database system. Stored procedures run directly by the database engine. Stored procedure is nothing but a code and it can be vulnerable as program code. [8] For authorized/unauthorized user the stored procedure returns true/false. As an SQLIA, intruder input " , ; SHUTDOWN; - -" for username or password. Then the stored procedure generates the following query:

"SELECT accounts FROM users WHERE login= 'doe' AND pass=' '; SHUTDOWN; -- AND pin ="

This type of attack works as piggy-back attack. The first original query is executed and consequently the second query which is illegitimate is executed and causes database shut down. So, it is considerable that stored procedures are as vulnerable as web application code.

F. Inference

This type of attack create queries that cause an application or database to behave differently based on the result of the query. These attacks allow an attacker to extract data from the database and detect vulnerable parameter. There are to well-known attack techniques based on inference: blind-injection and timing attacks.

Blind Injection

An attacker performs queries that have a Boolean result. If the answer is true then the application behaves correctly and if the answer is false then it cause an error. So attacker can get the indirect response from database.

Timing attacks

In this attack attacker observe the database delays in the database response and gather the information. To perform the timing attack attacker writes the query in the form of an if-then statement and then uses the WAITFOR keyword in one of the branch, which causes the database to delay its response by specified time.

G. Alternate Encodings

To avoid the signature and filter based checks the attacker modify their injection strings called as alternate encoding technique, such as ASCII, Hexadecimal and Unicode can be used in conjunction with other techniques to allow an attack and to escape from various detection methods.

IV. SQL INJECTION PREVENTION TOOLS

There are many ways to prevent SQL injection attacks. [8] The most popular in the source code. There are some approaches for testing Web applications to identify the presence of SQL injection vulnerabilities, e.g. using black-box testing techniques, methods are tainting and tracking of the user input, analyse the correctness of SQL statement statically; appending random numbers to SQL statements

A. VIPER tool for penetration testing

[9]According to Angelo Ciampa, Corrado Aaron Visaggio and Massimiliano Di Penta, they have suggested a tool called Viper to perform penetration testing of Web applications. This tool relies on a knowledge base of heuristics that guides the generation of the SQL queries. This tool first identifies the hyperlink structure and its input form.

B. Attack Injection Methodology

[10]Joao Antunes,Nuno Neves,Miguel Correia, Paulo Verissimo and Rui Neves has suggested the attack injection methodology i.e.AJECT tool which adapts and extends classical fault injection techniques to look for security vulnerabilities. In this Attack Injection Tool first the attacks are generated on the target system to evaluate the system. Means they first build test cases that would not only exercise all reachable computer instructions but also try them with every possible instance of input.

C. Static analysis And Runtime Monitoring Tool

[11]William G.J. Halfond and Alessandro Orso has suggested the tool that detects and prevents SQL injection attacks by combining static analysis and runtime monitoring. The name of the tool is AMNESIA (Analysis and Monitoring for Neutralizing SQL-Injection Attacks). This tool uses both static and runtime analysis. At static analysis it analyse the web application code and at runtime this techniques monitors all the dynamically generated queries.

D. Identifying SQL and XSS vulnerability

[12]Adam Kie˙zun, Philip J. Guo, Karthick Jayaraman, Michael D. Ernst has suggested a technique for finding vulnerabilities in Web Application such as SQL attack and Cross site scripting(XSS).This technique works on existing code, creates concrete inputs that expose vulnerabilities and operates before software is deployed. It analyses application internals to discover vulnerable code. The tool that identifies the SQL and XSS vulnerability is known as ARDILLA. It is based on input generation, taint propagation, and input mutation to find variants of an execution that exploit vulnerability.

E. Obfuscation-based Analysis of SQL Injection Attacks

[13]Raju Halder and Agostino Cortesi proposes the obfuscation/deobfuscation based based technique to detect the presence of possible SQL Injection Attacks (SQLIA) in a query before submitting it to a DBMS. Now the Obfuscated code is a source code that has been made difficult for human. [13]In obfuscation approach the possible attack injection are verified at atomic formula level and only those atomic formulas which are tagged as vulnerable, also this approach avoids the root cause of SQL injection attacks in dynamic query generation .

F. SQLInjectionGen SQLIA Detector

[14] MeiJunjin has suggested a tool SQLInjectionGen tool which combines the static analysis, runtime analysis and automatic testing. This is an automated test case generation tool to identify SQL injection vulnerability. According to author the prototype tool SQLInjectionGen had no false positives and small number of false negatives.

G. SQLrand Practical Protection mechanism

[15] S. W. Boyd and A. D. Keromytis has suggested the practical protection mechanism for preventing SQL injection attacks against web server. This tool uses SQL randomized query CGI application and detect and correct the queries injected into the code.

V. PROPOSED METHOD

We have proposed one solution to prevent the sql injection. Our solution is based on function. We have made one function to check the argument which we have passed during the input field. When user passes the arguments to input boxes its values are passed in our function. Our function has two arguments. One argument contains the injected query and second argument contains the level of checking the injected parameters.

1.) Argument -1 : Injected query
2.) Argument-2: Level of checking of injected parameters.

We have to put this function in inline query. When we call to this function then its first check that query is passed or not. If query is not passed then it simple return from the function, if query is passed then after it checks the level of vulnerability.

1.) Level -1: This level contains some special characters (‘, --, [, %)
2.) Level-2 : This level contains the sql keywords and sql operators (“xp_ ”, “update ”, “union ” “insert ”, “select ”, “drop ”, “alter ”, “create ”, “rename ”, “delete ”, “replace ”)

When user calls this function its check the level and then compares the value of injected query with both level parameters. If the value match to the parameter then function replace this value with other value to prevent the sql injection.

Proposed Function with code

```
#region MySQLinjectinPreventionFunction
  public string SafeSqlQuery(System.Object theValue, System.Object theLevel)
  {
    string strValue = (string)theValue;
    int intLevel = (int)theLevel;
    if (strValue != null)
    {
      if (intLevel > 0)
      {
 strValue = strValue.Replace("'", "''");
        strValue = strValue.Replace("--", "");
        strValue = strValue.Replace("[", "[[]");
        strValue = strValue.Replace("%", "[%]");
      }
      if (intLevel > 1)
      {
        string[] myArray = new string[] { "xp_ ", "update ", "insert ", "select ", "drop ", "alter ", "create ", "rename ", "delete ", "replace " };
        int i = 0;
        int i2 = 0;
        int intLenghtLeft = 0;
        for (i = 0; i < myArray.Length; i++)
        {
          string strWord = myArray[i];
          Regex rx = new Regex(strWord, RegexOptions.Compiled | RegexOptions.IgnoreCase);
         MatchCollection matches = rx.Matches(strValue);
```

```
            i2 = 0;
            foreach (Match match in matches)
            {
               GroupCollection groups = match.Groups;
               intLenghtLeft = groups[0].Index + myArray[i].Length + i2;
               strValue = strValue.Substring(0, intLenghtLeft - 1) + " " + strValue.Substring(strValue.Length -
(strValue.Length - intLenghtLeft), strValue.Length - intLenghtLeft);
               i2 += 5;
            }
          }
        }
        return strValue;
      }
      else
      {
        return strValue;
      }
    }
  #endregion
```

Function Call

```
sqlCommand = "SELECT 'b' FROM dbo.tbl_users WHERE username='" + SafeSqlQuery (txtUserName.Text, 2) + "'
AND password='" + txtPassword.Text + "'";
```

VI. CONCLUSION

To make SQL injection attack, an attacker should necessary use a space, double quotes, double dashes or sql keyword in his input. The method to detect one of the above symbols has been discussed.

In our approach we have used two levels to detect injected parameters. In the first level, check some symbols like double dashes present in the query. If symbols are present in the original query then we replace it by appropriate non injected symbol. In the second level, arrays of sql keywords are used and compare array's element with original query. If the sql symbols are present in the input then it is replaced by blank space. The work presented in this paper has been implemented using .net codes.

REFERENCES

[1] OWASP – Open Web Application Secuirty Project. Top ten most web application vulnerabilities. http: //ww.owasp.org/index.php/OWASP_TOP_Ten_Project, April 2010.
[2] W. G. J. Halfond, A. Orso and P. Manolios, "Using Positive Tainting and Syntax-aware Evaluation to Counter SQL Injection Attacks", In SIGSOFT'06/FSE-14: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2006.
[3] Chris Anley. Advanced SQL Injection In SQL Server Applications. [EB/OL]. http://w ww.nextgenss.com/research.html,2004-3-25.
[4] T. O. Foundation. Top Ten Most Critical Web Application Vulnerabilities,2005. http://www.owasp.org/documentation/topten.html.
[5] M. Dornseif. Common Failures in Internet Applications, May 2005. http://md.hudora.de/presentations/2005-common-failures/dornseif-common-failures-2005-05-25.pdf.
[6] N. W. Group. RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1.Request for comments, The Internet Society, 1999.
[7] Nilesh Khochare, Satish Chalurkar ,Santosh Kakade and B.B. Meshramm, "Survey on SQL Injection attacks and their Countermeasures", IJCEM International Journal of Computational Engineering & Management, Vol. 14, October 2011
[8] William G.J. Halfond and Alessandro Orso: "Preventing SQL Injection Attacks Using AMNESIA", ICSE'06, May 20–28, 2006, Shanghai, China.ACM 1-59593-085-X/06/0005.
[9] Angelo Ciampa, Corrado Aaron Visaggio, Massimiliano Di Penta :"A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications".
[10] Adam Kie˙zun,Philip J. Guo,Karthick Jayaraman,Michael D. Ernst:"Automatic Creation of SQL Injection and Cross-Site Scripting AttackS", ICSE'09, May 16-24, 2009, Vancouver, Canada,978-1-4244-3452-7/09/$25.00 © 2009 IEEE.
[11] Diallo Abdoulaye Kindy and Al-Sakib Khan Pathan: "A survey on SQL injection: vulnerabilities, attacks, and prevention techniques". 2011 IEEE 15th International Symposium on Consumer Electronics.

[12] Li Shan, Dong Xiaorui, RaoHong:"An Adaptive Method Preventing Database from SQL Injection Attacks". 2010 3rd International Conference on Advanced Computer Theory and Engineering (1CACTE).
[13] Raju Halder and Agostino Cortesi, "Obfuscation-based Analysis of SQL Injection Attacks". 978-1-4244-7755-5/10/$26.00 ©2010 IEEE
[14] MeiJunjin: "An approach for SQL injection vulnerability detection". 2009 Sixth International Conference on Information Technology: New Generations.
[15] 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, June 2004.

