2013 International Conference on Informatics and Creative Multimedia

A Taxonomy of SQL Injection Detection and Prevention Techniques

Amirmohammad Sadeghian, Mazdak Zamani, Azizah Abd. Manaf
Advanced Informatics School
Universiti Teknologi Malaysia
Kuala Lumpur, Malaysia
i@root25.com , mazdak@utm.my , azizah07@ic.utm.my

***Abstract*— While using internet for proposing online services is increasing every day, security threats in the web also increased dramatically. One of the most serious and dangerous web application vulnerabilities is SQL injection. SQL injection attack took place by inserting a portion of malicious SQL query through a non-validated input from the user into the legitimate query statement. Consequently database management system will execute these commands and it leads to SQL injection. A successful SQL injection attack interfere Confidentiality, Integrity and availability of information in the database. Based on the statistical researches this type of attack had a high impact on business. Finding the proper solution to stop or mitigate the SQL injection is necessary. To address this problem security researchers introduce different techniques to develop secure codes, prevent SQL injection attacks and detect them. In this paper we present a comprehensive review of different types of SQL injection detection and prevention techniques. We criticize strengths and weaknesses of each technique. Such a structural classification would further help other researchers to choose the right technique for the further studies.**

Keywords- SQL injection; Web application vulnerability; Information security;

I. INTRODUCTION

Structured Query Language injection is a code injection technique that used to attack database driven web application. In this attack the attacker inserts a portion of SQL statement via not sanitized user input parameters into the original SQL query and passes them to database server [1].

Based on Open Web Application Security Project (OWASP) studies, SQL injection has the first position in the top 10 list of web application vulnerabilities [2]. The targets of these attacks are not only limited to the web application but they also can hits desktop applications which their databases are powered by SQL. The amount of financial losses in result of SQL Injection was enormous, therefore finding a solution to stop SQL Injection attacks is necessary.

Attackers may insert the malicious query via a web form or directly by appending the malicious query to the end of the URL in the address bar of browser.

In a more unusual way of attack, attacker might try to inject the malicious variable through HTTP headers. For instance when the web application have a module that record the statistic related to the users activities such as users IP address, browser type and language. Basically these data will fetch from the HTTP header which comes from the user browser and it will be stored inside the database for further analysis or drawing charts. Changing the HTTP headers is very simple by using specific programs which are designed for this goal or headers add-ons in browsers.

Below is an example of PHP script which will receive the IP address of the user from the HTTP header and put it in a variable named $ip_address. Then without any input sanitization the variable concatenated with MySQL query.

```
$ip_address = $_SERVER["REMOTE_ADDR"];
$result=mysql_query('select username from area where ip='.$ip_address.' and level=admin');
```

And below is the structure of the header that manipulated by the attacker which will lead to insert the SQL injection.

```
GET /index.php HTTP/1.1
Host: [host]
X_FORWARDED_FOR :127.0.0.1' or 1=1#
```

The final query after concatenation which will send to the database management system for running will look like below:

```
'select username from area where ip='127.0.0.1' or 1=1# and level=admin');
```

The above query will select the "username" from the "area" table and will not pay any attention to the `where` clause. Because based on the logic after the SQL injection, the query condition is asking to seek the row that the IP address is equal to 127.0.0.1 or 1=1 . As 1 is always equal to 1 and it used an "OR" to join these two conditions, only one of these conditions need to be true to statement runs. And finally "#" sign which used for commenting will ignore the rest of the statement.

SQL injection attacks are classified under seven main categories:

- Tautologies,
- Illegal/Logically Incorrect Queries
- Union Query
- Piggy-Backed Queries
- Stored Procedures
- Inference
- Alternate Encodings

Currently wide ranges of detection and prevention techniques are proposed and used by developers and application owners. We can divide these techniques based on the nature of their defense to three main category of:

- Best code practices: They are set of guidelines and policies for developers to improve the quality of

978-0-7695-5133-3/13 $26.00 © 2013 IEEE
DOI 10.1109/ICICM.2013.18

their code by following them. Using of the best practices can highly decrease the potential chance of SQL injection vulnerabilities.

- SQL injection detection: This technique detects the SQL injection attacks.
- SQL injection runtime prevention: This technique prevents SQL injection attack in the execution time and compares them against the legitimate query [3].

The rest of this paper is organized as follows. Section 2 explained the best code practice techniques following by some example for them. Section 3 explained SQL injection detection techniques. Section 4 explained SQL injection runtime prevention techniques. Finally section 5 is the conclusion.

II. BEST CODE PRACTICES

A. Manual Defensive Coding Practices

In this technique developer will learn the SQL injection attack techniques and how to prevent them by securing the source code. There are many cheat sheets available which can help the developer with secure coding guidelines. We can divide these practices to four main categories:

- Parameterized queries or stored procedures:
  A parameterized query is a type of query which has some placeholders. In these queries instead of making dynamic queries by concatenating the parameters with SQL statement, it will replace the placeholders with the value of parameters at the runtime. Using stored procedures also can be effective in combating SQL injection. Because they check the type of parameters, if the attacker passes a wrong type of value to the stored procedure, they will throw an exception but these exceptions should handle properly. In fact stored procedures independently cannot eliminate the SQL injection but they do hide the structure of the database from the attacker.

```
   MySqlConnection conn = new
MySqlConnection(_connectionString);
   MySqlCommand command =
conn.CreateCommand();
   MySqlParameter Parameter = new
MySqlParameter("?id", id);
   command.Parameters.Add(Parameter);
command.CommandText = "SELECT * FROM
news where id = ?id";
```

- Escaping:
  Escaping in aspect of SQL injection refers to a technique which escapes SQL language keywords. First by referring to DBMS manual a blacklist of dangerous keywords is created and later they apply the black list. Each database has its own escaping functions and libraries. For example in PHP mysql_real_escape_string() will handle the escaping process. The following example is a PHP script that demonstrates escaping function. In result of calling the escape function it will escape all dangerous characters like ' with \ char.

```
$att = "This is Amir's laptop";
$escaped_att=mysql_real_escape_string($
att);
printf("Escapedstring:",$escaped_att);
OUTPUT: Escaped string: This is Amir\'s
laptop
```

- Data Type Validation:
  In this method developer should check the data that comes from the form fields. For example if the field is a phone number, it should check that it doesn't contain a string. This method cannot guaranty that it will fully stop the SQL injection but it makes the process harder for the attacker.
- White List Filtering:
  White list filtering is the opposite of black list filtering. It only allows those inputs that look legitimate to get accepted and executed in the database. The negative point about this technique is that they are hard to implement because input should be normalized before submitting into the detection algorithm [4].

B. SQL DOM

SQL DOM is a technique which proposed by McClure and Kruger to give the ability of using dynamic statements without security problems. Their application is made of a main file named "sqldomgen". At the first stage the developer need to run this file, to generate a DLL file. The output DLL file contains powerful classes which can help the developer to make dynamic queries with them. All field data types and structure of database will be available in the DLL file. In case later the database developed or any changes happen in the database structure, "sqldomgen" should run again on the database to provide the new classes for the development [5].

C. Parameterized Query Insertion

This solution can find the vulnerable SQL queries inside the source code and replaces them with safe parameterized SQL queries. The drawback of this technique is that it only works with SQL structures created with explicit strings [3]. In the following example the first line is showing the vulnerable dynamic statement and the second line onward is the replaced bounded variable by this solution.

```
$result = mysql_query("SELECT title FROM news
WHERE nid = '$id');

$conn=new PDO("mysql:host=localhost;
dbname=MyDB;","DBusername","DBpassword");
$UserInput[] = Array();
$query = "SELECT title FROM news WHERE nid
=?";
$UserInput[] = $id;
$stmt = $conn->prepare($query);
$j = 1;
foreach($UserInput as $item){
      $statement->bindParam($j++, $item);
}
$result = $statement->execute();
```

III. SQL Injection Detection

A. SQLUnitGen

SQLUnitGen is the short form of "SQL Injection Testing Using Static and Dynamic Analysis", which proposed by Shin and colleagues. Their solution use static analysis to track the flow of user input for attack testing. The core of their tools is based on "JCracher". They did some changes to create the test cases for the attack [6].

B. MUSIC

MUSIC is the short form of "MUtation-based SQL Injection vulnerabilities Checking" method which proposed by Zulkernine. He used mutation based testing approach for testing of SQL injection vulnerability. "Mutation is an error-based testing method which will inject syntax fault to see if any mutant exists. Then by comparing the output it can determine if the statement contains a mutant". Their method uses nine mutation operators to do the injection in the code [7].

C. Vulnerability and Attack Injection

This solution is proposed by Fonseca and colleagues. They present a method to attack the application by realistic SQL injection vulnerabilities. This will prepare an environment for testing the countermeasure of security tools such as intrusion detection systems, firewalls and vulnerability scanners [8]. For reaching to more realistic results they used collected data from real attacks and patches. Analyzing this information helped them to understand the vulnerable codes and their distinctions which are not vulnerable. The proposed tool is made of two main parts: "Attack Injection tool" and "Vulnerability Injection tool" which work automatically together.

The Vulnerability Injection program is used to inject the SQL injection vulnerabilities in the code of the application. It reviews the source code and looks for possible places that are suitable for the injection. When it founds the place for injection, it will uses the realistic patterns from the previous section (real world samples) and inject them.

Attack injection tool is an application which works almost the same way of previous part. The only additional feature is that after injection of the vulnerability, it will attempt to attack the application. By the help of a HTTP proxy this tool will sniff the traffic between the web application and the database management system. These data will used to launch the attack against vulnerable files. Next by using "Attack success detector" tool, it will verify that the attack is successfully launched.

Attack injection tool has a weakness which will make mistake when a variable value processed before using in the SQL statement. For example the phone number of the user might be split into section of country code and phone number and they will be used against two columns in the database. This will lead this algorithm to make a mistake in the detection.

D. SUSHI

SUSHI is a "string constraint solver" proposed by Fu and Li. They proposed a recursive algorithm which can solve Simple Linear String Equation (SISE) in an efficient way [9]. They break the SISE constraint to atomic string operations. They proved that their solution is very effective in finding complicated SQL injection attacks.

E. Ardilla

Ardilla is an approach for creating SQL injection attacks proposed by Kiezun and colleagues. This tool is able to generate attacks to use as the input of the web application, to detect SQL injection vulnerabilities [10]. They believe that their solution has no overhead in execution time also the modification of the source code is not needed. Ardilla will generate some inputs, and then run the application with each input for testing. Afterward it receive the output and analysis it, to see is there any data sending to database between the times of receiving of the input until the query execution.

F. String Analyzer

String Analyzer is proposed by Wassermann and Su, they proposed a grammar based algorithm which model the string values as context free grammars and string operations as language transducers following Minamide [11]. This solution labels those strings that come from the user side as nonterminal. It will assign the "direct label" to those strings that are come directly from the user side such as GET requests. And assign "indirect label" to strings that are come from database side. Next they summarize the labeled strings to find the contexts and afterward by using regular languages and context free languages check the security of each string in aspect of syntax.

G. PHPMiner

PHP Miner is a tool that proposed by Khin Shar and Kuan Tan, which mines static code attributes and then make a model of vulnerability prediction based on the data collected in the previous phase [12].

IV. SQL Injection Runtime Prevention

A. SQLrand

Boyd and Keromytis proposed a technique for SQL injection prevention which use randomized SQL query to detect malicious statements and abort them. For this purpose they made randomized instances of the SQL query by randomizing the template query inside the CGI script and the database parser [13]. For example the SELECT keyword will replace by SELECT921. SELECT921 is a random name which generated for the current execution. Later the developer by using a proxy will intercept the traffic between the application and the database, and if any keywords without randomization found that is a SQL injection. In result attacker cannot do the SQL injection without knowing the random key. The positive point about this solution is that it will not affect the performance.

B. AMNESIA

Halfond and Orso proposed the AMNESIA model which is the combination of static analysis and runtime analyze of behavior of application [14]. AMNESIA creates a model of original queries in the static part. Afterward in the dynamic part monitors "dynamically generated queries" at runtime and compare them against the legitimate model. SQL statements which cannot meet the requirements will know as SQL injection attack and tool will stop them before sending them to the database. The main weakness of this technique is that cannot support segmented queries [4].

C. WASP

Halfond and colleagues proposed WASP (Web Applications Using Positive Tainting and Syntax-Aware Evaluation) solution which works based on the dynamic tainting [15]. Their approach is the improved version of classic tainting. The first improvement is the using of positive tainting which is based on making trust. Classic tainting is based on untrusted data. Next improvement is the accuracy and efficiency of the tainting by tracing of trust making at character level. Third improvement is the blocking of queries which contains SQL keywords and operators without trust making. The last one is the minimal implementation requirements which make this method practical.

D. SQLprob

SQLProb is the short form of SQL Proxy based Blocker, a SQL injection detection system which proposed by Liu and colleagues. This approach extracts the user input from the query generated by the web application. Then validate these data in context of the generated query's syntactic structure. For this purpose they used genetic algorithm. This system has few advantages. The first one is that, it does not require source code of the application. Next one is that the validation process does not need learning. Third improvement is that this technique used a proxy which needs the minimum requirement for the implementation. The last one is the independency from programming language [16].

E. CANDID

CANDID is stand for (CANdidate evaluation for Discovering Intent Dynamically) which proposed by Bisht, Madhusudan and Venkatakrishnan. This technique dynamically mines the programmer intended query structure at the runtime with valid inputs. Then compare it with legitimate query statement. If the result is not the same, it is a SQL injection attack [17].

V. CONCLUSION

One of the most dangerous vulnerabilities in the web application is SQL injection. Until now many different techniques are proposed by researchers to defeat it. However attackers always found a new method to bypass these solutions. In this paper we have presented taxonomy of different detection and prevention techniques for this vulnerability. In this taxonomy we divide all the techniques to three main categories of 1.Best code practices, 2.SQL injection detection, and 3.SQL injection runtime prevention. In future this comprehensive classification can help other researchers in their studies on SQL injection.

REFERENCES

[1] 1.Antunes, N. and M. Vieira, "Defending against Web Application Vulnerabilities." Computer, 2012. 45(2): p. 66-72.

[2] (OWASP), "O.W.A.S.P. Top 10 Vulnerabilities."; Available from: https://www.owasp.org/index.php/Top_10 2013.

[3] Shar, L.K. and T. Hee Beng Kuan, "Defeating SQL Injection." Computer, 2013. 46(3): p. 69-77.

[4] Janot, E. and P. Zavarsky. "Preventing SQL Injections in Online Applications: Study, Recommendations and Java Solution Prototype Based on the SQL DOM." in OWASP App. Sec. Conference. 2008.

[5] McClure, R.A. and I.H. Kruger. "SQL DOM: compile time checking of dynamic SQL statements. in Software Engineering, 2005." ICSE 2005. Proceedings. 27th International Conference on. 2005.

[6] Shin, Y., L. Williams, and T. Xie. "Sqlunitgen: Sql injection testing using static and dynamic analysis." in supplemental Proceedings of the 17th IEEE International Conference on Software Reliability Engineering (ISSRE 2006). 2006.

[7] Shahriar, H. and M. Zulkernine. "MUSIC: Mutation-based SQL Injection Vulnerability Checking." in Quality Software, 2008. QSIC '08. The Eighth International Conference on. 2008.

[8] Fonseca, J., M. Vieira, and H. Madeira. "Vulnerability & attack injection for web applications". in Dependable Systems & Networks, 2009. DSN '09. IEEE/IFIP International Conference on. 2009.

[9] Fu, X. and C.-C. Li. "A string constraint solver for detecting web application vulnerability." in Proceedings of the 22nd international conference on software engineering and knowledge engineering (SEKE). 2010.

[10] Kieyzun, A., et al. "Automatic creation of SQL Injection and cross-site scripting attacks." in Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on. 2009.

[11] Wassermann, G. and Z. Su, "Sound and precise analysis of web applications for injection vulnerabilities." SIGPLAN Not., 2007. 42(6): p. 32-41.

[12] Shar, L.K. and H.B.K. Tan, "Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities" in Proceedings of the 2012 International Conference on Software Engineering2012, IEEE Press: Zurich, Switzerland. p. 1293-1296.

[13] Boyd, S.W. and A.D. Keromytis. "SQLrand: Preventing SQL injection attacks." in Applied Cryptography and Network Security. 2004. Springer.

[14] Halfond, W.G.J. and A. Orso, "AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks." in Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering2005, ACM: Long Beach, CA, USA. p. 174-183.

[15] Halfond, W.G.J., A. Orso, and P. Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation." Software Engineering, IEEE Transactions on, 2008. 34(1): p. 65-81.

[16] Liu, Anyi, Yi Yuan, Duminda Wijesekera, and Angelos Stavrou. "SQLProb: a proxy-based architecture towards preventing SQL injection attacks." In Proceedings of the 2009 ACM symposium on Applied Computing, pp. 2054-2061. ACM, 2009.

[17] Bisht, P., P. Madhusudan, and V.N. Venkatakrishnan, "CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks." ACM Trans. Inf. Syst. Secur., 2010. 13(2): p. 1-39.