An efficient model to detect and prevent SQL Injection Attack: detect and prevent SQLIA

Abdalla Hadabi
Eltyeb Elsamani
Ali Abdallah
Rashad Elhabob

Online ISSN: 8000-1858
Print ISSN: 800-1858X

Article history:
Received 22, October 2021
Received in revised form 04, March 2022
Accepted 13, March 2022
Available online 30, March 2022
An Efficient Model to Detect and Prevent SQL Injection Attack

Abdalla Hadabi*, Eltyeb Elsamani*, Ali Abdallah†, Rashad Elhabob‡

*Computer Science Department, Alneelain university, Khartoum, 12304, Sudan
†Information Systems Department, Alneelain university, Khartoum, 12304, Sudan
‡College of Computer Science and Information Technology, Karary University, Khartoum, Sudan
Email: abdoadel30@gmail.com

Abstract

SQL injection attack (SQLIA) is considered one of most threats used to attack web applications. Therefore, attackers use SQL injection vulnerability to gain ultimate access to databases that belong to applications and expose their sensitive information. Thus, attackers use SQL injections vulnerability to manipulate data also it could be used to take full control of the target machine. Accordingly, several methods were proposed in the literature to address this vulnerability widely because of its importance and high impact on the security of web applications. Thus, we propose a model to detect and prevent SQL injection attack, which uses runtime validation to detect the occurrence of such attacks, our proposed model is adaptable to any existing system, with no need to modify the client or server and either no need to know web application source code. Furthermore, the modification independence is done by adding additional middleware between client and server. Thus, any check process is done on this middleware, and it is represented as a proxy that can do sanitize the inputs for detecting and preventing SQLIA. Furthermore, our proposed model accuracy reaches 86.6% for detecting and preventing SQLIA.

Key words - SQL injection; Web applications security; Proxy server.

I. INTRODUCTION

applications due to the accessibility and convenience they provide, which makes web applications a suitable target for attackers. Accordingly, web applications security becomes necessary. web applications have different sorts of attacks One of the most important attacks that are used to attack Web applications is SQL Injection Attack (SQLIA). SQL Injection is a vulnerability that results when you give an attacker the ability to influence the Structured Query Language (SQL) queries that an application passes to a back-end database [1]. By being able to influence what is passed to the database, the attacker can leverage the syntax and capabilities of SQL itself. As well as the power and flexibility of supporting database functionality and operating system functionality available to the database [2][3]. SQL injection is not a vulnerability that exclusively affects Web applications. Any application that accepts input from an untrusted source then uses that input to build SQL statements. Furthermore, the attacker injects a malicious SQL query into the back-end database through a web application interface. Thus, when developers write their web applications they focus on functionality rather than security, for this reason, no insufficient input validation[4]. Accordingly, more companies and organizations provide various services to users via web applications by receiving their requests interacting with the back-end database, and returning relevant data for users. According to the Application Architecture used the injected code is taken via a Web browser within the presentation tier to deal with the back end of Database [5] The injected SQL statements are executed by the back-end database and it also returns the corresponding execution results to the attacker. Furthermore, attackers can use this vulnerability to extract confidential information or even obtain the root privilege of database [6][7]. This is done when the attacker manipulates the request data in URL (Unified Resource Locator) to retrieve data in an illegal way as shown in Fig. 1.

According to OWASP Top 10 in 2017[8] (Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted). Furthermore, according to the OWASP report, SQLIA has the highest frequency among other web application attacks which shows the importance of securing web applications from this type of attack by protecting the web applications and their data. In addition, according to a survey report released in 2012 by the IBM X-Force [9] research and development team, the number of SQL injection attacks has increased rapidly in recent years, and SQL injection has become the Predominant type of attacks that target web applications. The SQL Injection signature ranked second highest in 2010 and climbed to the highest in 2011 with an indication of a continued upward trend. 2011 was a banner year for exploiting SQL weaknesses. In 2012, we are
seeing even higher levels of SQL injection attempts and the expansion rate of this type of attack appears to be higher than at the end of 2011 the average number of daily SQL injection attacks around the world is about 2000,000 [9]. Furthermore, several studies discovered that the number of SQL injection attack attempts reaches 71 attacks hourly [10]. Additionally, the attacker finds new ways to exploit stored data, in 2020 more than 8.4 records of stored data have been exploited, the data exploit hits several companies such as eBay, Uber, Sony, Adobe, and others [11]. However, SQL injection occurs when the attacker has the ability to reach the database back-end. Thus, the back-end database often contains confidential and sensitive information such as financial or medical data that interest attackers. In addition, to assure good programming and configuration to secure web applications, it also needed effective protection mechanisms to prevent attacks[12][13]. There are many approaches proposed to address the problem of SQL injection attacks. They can be roughly divided into two categories: static analysis which is techniques about how to build secure applications and database systems such as defensive programming practices, dynamic analysis which is techniques focus on how to protect an existing system such as intrusion detection systems) [4],[1],[14].

A. Types of SQLIA:

SQL injection attacks are divided into several types [2],[15],[16] as follow:

1) tautologies: Attacks that are based on Tautology inject code in one or more conditional statements, as a result, the injected SQL statement is always evaluated to be true.

2) illegal/logically incorrect queries: In this type of attack, the attacker tries to do some false queries intentionally seeking an error message generated by a web server that contains important information for debugging. Therefore, the attacker can identify the vulnerable applications on the webserver.

3) UNION query: Attackers do this by injecting a statement of the form: UNION SELECT is used to retrieve information from a specified table.

4) Piggy-backed queries: This attack occurs when multiple statements such as INSERT and UPDATE are allowed to be executed at the back-end of the database. By concatenating these statements with ‘;’.

5) Stored Procedures: In this type, the attacker tries to execute stored procedures present in the database. Accordingly, stored procedures are often written in special scripting languages, which contain other types of vulnerabilities, such as buffer overflows, which allow the attacker to run arbitrary code on the server or even escalate their privileges.

6) Blind SQL: Occurs when making queries that output true or false which aim to retrieve database errors that are coming from an application that built to show generic errors.

7) Alternate Encoding: In this kind of attack, the attacker tries to evade filters and input sanitizer by encoding the attack string with hexadecimal, ASCII, and Unicode. Accordingly, the attack will be undetected. This means it’s not a standalone attack but it is used to encode other attacks. However, the IBM and OWASP surveys show that web applications are suffering widely from SQL Injection attacks. In this paper we will build a model to detect and prevent SQL Injection attacks, by Studying literature and then provide a model to prevent (Piggy-backed queries, Logically Incorrect Query, UNION Query, Tautologies ) of SQLIA types by using a regular expression which is defined as a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. They can be used to search, edit, or manipulate text and data[16][17].

B. Finding SQL Injection vulnerability:

A standard test to see if a web form is vulnerable to SQL injection is by putting a single quote ‘’ in the field. If you get back a database error, the web Application is most likely vulnerable to SQL injection[18]. However, the absence of a returning MySQL error doesn’t imply that the web application is protected against SQL injection at all [19]. Furthermore, Sometimes the returning error could be present inside the source of the webpage. However, different developers have developed their web applications differently. Accordingly, the structure and syntax of SQL queries may also vary from one web Application to another[20]. The same generic ‘true condition’ (tautology type) approach may not work with all web applications. Thus, the attacker may have to try different combinations from SQL Cheatsheets. Some developers implement JavaScript validation at the client side which prevents the user from making a request that has a single quote (‘) on it. JavaScript validation offers no security to the actual execution of SQL statements. Also, whenever an SQL error occurs, the developer may return a generic page with no information at all. In such a case, blind SQL injection is used which is very difficult but not impossible to execute [12][21].

C. Research motivations

The main motivation is to protect web applications that have been attacked by this vulnerability. Thus, it’s clear how SQLIA is the most threatening attack as mentioned by the OWASP and IBM [8] [9]. The main question is why and how does this vulnerability make a gate to retrieve our data? And what is the way to close this gate?

II. RELATED WORK

The research area of SQLIA has seen various methodologies proposed over the years by researchers which used several ways to detect and prevent SQL injection attack as follow: S. Bandhakavi, et al. [22] In this approach authors dynamically build the SQL query structure that is intended by the programmer whenever the execution reaches a program location that issues an SQL query. The author’s approach computes the awaited query by running the application on candidate inputs, that are self-evidently nonattacking, as expected by the programmer while coding. Additionally, the awaited query must exercise the same control path in the program. Therefore, by applying these conditions on candidate inputs, SQL injection attacks could be detected by comparing the query structures of the programmer’s intended query. S.
Bandhakavi proposed a novel technique to dynamically deduce the programmer’s intended structure of SQL queries and used it to effectively transform applications so applications guard themselves against SQL injection attacks. However, new needs arise to detect and prevent SQLIA in runtime. I. Balasundaram [23] proposed a Technique that is used to detect and prevent SQLIA with runtime monitoring using a web service that gives the ability to be free from code modification. Furthermore, the Service Detector model allows the Authenticated or legitimate user to access the web applications, with two input filters. Balasundaram’s technique was able to suitably classify the attacks that were performed on the applications without blocking legitimate accesses to the database. However, the previously mentioned schemes [22] [23] suffer from the problem of authentication. Thus, the authentication problem was solved by Shubham and Rajeev [24] as they proposed a new technique for securing the database against SQL injection attacks. Furthermore, the solution was adding one extra column in the user account table to store the final hash value of the users name and password. The value is created when adding a new user and stored in the user account table. Accordingly, at the time of login, the final hash code is calculated using the stored procedure at run time and the authentic user is identified by exact matching of username, password, and final hash code. Shubham’s schema presents an authentication method for preventing SQL injection attacks and limiting access to those authorized to access the application. In addition, Kaleem [25] proposed a system in which the user credentials are hashed and encrypted. Thus, the input is filtered by the system to avoid the issue of inserting the malicious code into the database to prevent SQL Injection Attacks. Thus, new needs arise that is to provide a general framework to help protect from SQLIA even while software development, in addition to running web applications. However, H. Alattar et al. [26] proposed Detection Model of SQL Injection Vulnerabilities and SQL Injection Mitigation Framework. Which is based on SQL Injection grammar to identify the SQL Injection vulnerabilities during software development and SQL Injection Attack on running web applications. The author’s technique is based on Tainting identification either positive (identification of the trusted data rather than untrusted data) or negative (the vice versa of positive). Additionally, if Taint Propagation is done inaccurately it may lead to data misuse. To achieve accuracy tainting at the character level, Strings are always divided into substrings for constricting SQL quires. Therefore, Sampada [27] proposed an application that is developed for an online banking application. Furthermore, This application prevents various types of SQL injection attacks. Sampada builds and uses a tool called WASP( Web Application SQL Injection Preventer tool). Thus, the WASP tool consists of two approaches. Which is positive tainting and syntax-aware evaluation, and it seems like what was discussed earlier [26]. Accordingly, two modules are developed, the first one shows how the attacker can access the data, and the other shows the prevention of these attacks. It is found that the attacks can be prevented. However, the previous schema suffer the need for more response time, then Gu et al.[28] proposed a novel traffic-based SQLIA detection and vulnerability analysis framework named DIAVA, which can proactively send warnings to tenants promptly. By analyzing the bidirectional network traffic of SQL operations and applying the author’s proposed multilevel regular expression model, DIAVA can accurately identify successful SQLIAs among all the suspects. Meanwhile, the severity of such SQLIAs and the vulnerabilities of the corresponding leaked data can be quickly evaluated by DIAVA based on its GPU-based dictionary attack analysis engine. Experimental results show that DIAVA not only outperforms state-of-the-art WAFs in detecting SQLIAs from the perspectives of precision and recall, but also enables real-time vulnerability evaluation of leaked data caused by SQL injection.

A. Discussion:

In [23] The runtime validation incurs some overhead in terms of execution time, because of the extra computation time at the query validation, and it is good by making a separation of data and sensitive data by butting that last in Service Detector. And [24] there must be focusing on the efficiency of the system because if an attacker finds an equivalent system it will break the approach. And [26] the problem with traditional tainting is incompleteness. Furthermore, Incompleteness will lead to false negatives and may leave the application vulnerable to attacks. And [22] is the best approach of all of them but it partially prevents stored procedure attack. And [27] the authors make an application that has some issues in deploying it on the web. We can conclude that most approaches that depend on static analysis and certain types of code obfuscation or query development techniques could result in both false positives and false negatives by using reverse engineering techniques. Our solution comes with an additional tier, which acts as a middle tier represented by a proxy server that makes the process of checking SQL injection attacks done without any modification on the server or client-side.

III. PROPOSED MODEL:

The process of preventing SQLIA needs more investigation when it comes to inputs, there will be a need to check the syntax of an SQL query, this check can be done on the client-side or the server-side but change must do on one side. However, the needed modifications lead to additional cost and complexity to the solution. Thus, our proposed model adds a proxy server as a middle layer between the client and database server which is do the check process without any need to change the client or server-side. Accordingly, the proposed solution works as the following scenario: the client connects to the proxy server which is first authenticating the dedicated user and check if its authenticated or not by using the hash value of username and password after the authentication process in case of success the proxy server start to filter the content of the query. Hence, the proxy server filters the input query and check if there is an SQLIA which is done at runtime for all types of SQLIA except for stored procedure. Furthermore, the input checking process used a regular expression which is a sequence of characters that specifies a search pattern. As in Fig 2, the Proposed solution work as flow:
The proposed model has been built over many components as follows: The Client and It represents the entity that asks for a page and makes a query (such as Web Browser). The Web server contains a website with its database and serves clients. In the middle of other components, we will find the proxy server, which is responsible for the authentication process in addition to altering inputs, it could be implemented within a web server or in a standalone server. Lastly, a Database that contains website data like users information. Fig. 2. shows the system components and how these components communicate with each other.

### A. System Component:

The proposed model has been built over many components as follows: The Client and It represents the entity that asks for a page and makes a query (such as Web Browser). The Web server contains a website with its database and serves clients. In the middle of other components, we will find the proxy server, which is responsible for the authentication process in addition to altering inputs, it could be implemented within a web server or in a standalone server. Lastly, a Database that contains website data like users information. Fig. 2. shows the system components and how these components communicate with each other.

### B. Model Evaluation:

Our evaluation measure will be accuracy as shown below. The number of correct and incorrect classifications in each potential value of the classified variables to evaluate the outcomes gained. The following formulas are used to calculate the accuracy[?].

\[
\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)} \quad (1)
\]

Where TP, TN, FP and FN are true positive, true negative, false positive and false negative respectively.

### C. Prevent SQLIA:

Before starting to implement the proposed solution first we need to know how to prevent the attack and the keywords that are used for the black-box technique as it was used before in Literature [27]. As we mentioned before the java language will be used to implement our model which is need Regular Expression provided by this language to perform the process of keyword matching patterns.

### D. Keywords:

The application requires validating the URL strings with a different set of keywords which used to perform an injection attack. The SQL keywords are part of the SQL language which is normally used to perform operations on the tables. The application scans the network traffic and uses a combination of keywords to effectively detect the SQL Injection attack. When the keyword combination is found within the inputs the application will abort the connection. The proposed solution uses a set of keywords which are the SQL commands that are detected within inputs such as Update, Delete and Select.

### IV. Experiment and Results:

Our experiment was accomplished by using a Laptop with an Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz 2.60 GHz CPU, 8 GB of RAM, Win 10 operating system, VMware workstation to manage VM’s. However, we use a regular expression to investigate the query structure based on historical web log data, which is represented in the dataset that is collected from an online repository website, containing 4201 entries [29] queries with the label 1 means SQLIA exists, and queries with the label 0 mean its normal query, the adopted data set contains 1129 malicious query and 3072 normal query. In this experiment, we use the XAMPP server which was released by Apache Friends as a web server, in addition to WebScrap proxy from OWASP with the virtualized environment as a proxy server. WebScrap is written in Java and portable to many platforms. WebScrap has several modes of operation, implemented by several plugins. Furthermore, most common usage, WebScrap operates as an intercepting proxy, allowing the operator to review and modify requests created by the browser before they are sent to the server, and to review and modify responses returned from the server before they are received by the browser. Thus, one of the main reasons to use webscrap proxy is because it contains a plugin called BeanShell that we use to write our java code on it. Furthermore, we write a java code that checks the structure of input if contains an SQL injection attack or not, by testing the dataset over our proposed model to calculate the resulting false positive, false negative, true positive, true negative. Additionally, our proposed model intercepts the browser traffic and check if the input contains any malicious inputs, at first the proxy server checks the input using our prepared regular expression which checks the structure of the request if it contains any malicious SQL. However, if the request is clean the request will be bypassed else it will be aborted and the attacker will be directed to another page that shows its attack attempt. WebScarab can intercept both HTTP and HTTPS communication.
A. Results:

In this section, our proposed model results will be presented. Furthermore, we will take accuracy as an evaluation measure which is measured by using the terms of resulting false positive and false negative. True positive occurs when the technique blocks the execution of non-malicious input, and False negative occurs when the technique allows actual malicious input. However, to measure our proposed model accuracy we need to check the classifications of malicious input, and normal input as shown in table I. Accordingly, calculate the false positive, false negative, true positive, and true negative as shown in Fig. 3 that shows the distribution True decision and False decision. then calculate the total accuracy as calculated in equation (1) as shown in table II. Our resulting model accuracy reaches 86.6% to detect and prevent SQLIA.

![Fig. 3. Shows the distribution of false positive, false negative, true positive, and true negative (Green: True decision; Red: False decision.)](image)

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>THE DISTRIBUTION OF TP, FN, FP AND TN VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>True</td>
</tr>
<tr>
<td>Positive</td>
<td>TP</td>
</tr>
<tr>
<td>Negative</td>
<td>FP</td>
</tr>
</tbody>
</table>

Where TP, TN, FP and FN are true positive, true negative, false positive and false negative respectively.

<table>
<thead>
<tr>
<th>TABLE II</th>
<th>CALCULATE TRUE POSITIVE, TRUE NEGATIVE, FALSE POSITIVE AND FALSE NEGATIVE VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>True</td>
</tr>
<tr>
<td>Positive</td>
<td>2722</td>
</tr>
<tr>
<td>Negative</td>
<td>219</td>
</tr>
<tr>
<td>Total</td>
<td>3024</td>
</tr>
</tbody>
</table>

V. CONCLUSION

Our paper first defines the importance of web application data security. Furthermore, the most used vulnerability to exploit web application databases is SQL Injection Attack (SQLIA) as reported by OWASP and IBM reports, then we explore the attack types. However, many solutions proposed to solve this problem. Our proposed model is using a proxy in the middle between the client and server to check any input that is forwarded to the database server and prevent the attack using patterns that are implemented using regular expression. Furthermore, if we deal with a system that is having a large database then distributed environment could be used. As an experiment, we implement the patterns using java programming language inside the Webscrap proxy plugin called BeanShell. Our proposed model detects and prevents all types of S QLIA except stored procedure and alternate encoding. Our proposed model accuracy reaches 86.6%.

REFERENCES


Abdalla Hadabi is a lecturer and researcher of computer science Department, Faculty of Computer Science, Al-Neelain University, Khartoum, Sudan. Received the B.S. degree, in 2013 from the Faculty of Computer Science and Information Technology, Al-Neelain University, Khartoum, Sudan. In 2014 received my Master degree from the School of Software Engineering, University of Electronic Science and Technology of China, Chengdu, China. His current research interests include cryptography, network security and Computer Vision.

Ali Abdallah is a lecturer and researcher of Information Systems Department, Faculty of Computer Science, Al-Neelain University, Khartoum, Sudan. He received his M.Sc. and Ph.D. degrees in Information Systems. His current research area of interests data science, data analytics. In addition to publishing scientific papers.

Rashad Elhabob Received the B.S. degree, in 2010 from the Faculty of Computer Science and Information Technology, University of Karary, Khartoum, Sudan. In 2014 received my Master degree from Faculty of Mathematical Science, University of Khartoum, Khartoum, Sudan. In 2019, I received the Ph.D. degree from the School of Software Engineering, University of Electronic Science and Technology of China, Chengdu, China. He is currently assistant professor in the College of Computer Science and Information System, Karary University, Khartoum, Sudan. His current research interests include cryptography, Internet of Things, and Cloud computing.

Elyteb Elsamani is an Associate professor of computer science in computer science Department, Faculty of Computer Science and Information Technology, Al-Neelain University, Khartoum, Sudan. He received his B.Sc. and M.Sc. degrees in computer science from Al-Neelain University and his Ph.D. degree in Information Security, Encryption, Information Hiding Al-Neelain University, Khartoum, Sudan. His current research interests include Information Security, Steganography, Authentication, Security attacks, and Cryptography.