CRITICAL ISSUE TO CONSIDER WHILE DEVELOPING SQL INJECTION PREVENTION MECHANISM

Muhammad Saidu Aliero *, Dr. Imran Ghani, Muhammad Murad Khan, Nkima L.H
Universiti Teknologi Malaysia.
msaidua2000@gmail.com

Keywords SQL Injection, Injection Parameter , Defensive Coding

Abstract: SQL injection vulnerability is one of the most common web-based application vulnerabilities that can be exploited by SQL injection attack to gain access to restricted data, bypass authentication mechanism and execute unauthorized data manipulation language. Defensive coding is the simple and affordable way to tackle this problem, by applying secure coding in each and every queries used in application. In this paper we provide a detailed background of Sqli attack, we classify defensive coding into different categories, review existing techniques that are related to each technique, and also evaluate such techniques based on number of attacks they were able to stop. We also evaluated each category of approach based on its deployment requirement related to inheritance. Currently, to the best of our knowledge no papers have classified defensive coding as we do.

I. INTRODUCTION
SQLI (SQL injection) vulnerability is one of the most dangerous vulnerabilities in web-based database driving applications. It occurs as a result of inappropriate user input validation, which enables the attacker to manipulate programmer intended queries by adding new SQL operator, command, keyword, or clause to perform unauthorized database extraction modification, thereby bypassing authentication mechanism.

The main cause of SQL injection vulnerability is improper validation of user input. Input validation is a technique by which a programmer applies defense code practice to secure each static query manually. One of the objectives of defensive programming is to write secure queries so that it behaves in a predictable manner despite unexpected inputs or user actions. It is based on the idea that every program module is solely responsible for itself. After web development, its code should be reviewed by security analysts for proper use of function.

II. BACKGROUND OF SQLIA
Injection Parameter
Injection through User input field: user input fields are provided in web applications to enable web application users to request information that are stored in application database to the user with help of HTTP POST and GET such as Login input, URL input and SEARCH input.

SQLIA Types
1. Tautology attack
2. Piggy-backend query attack
3. Illegal/Incorrect logical query attack
4. Inference attack
5. Store procedure attack
6. Union query attack
7. Alternate encoding attack

III. CATEGORIZATION OF DEFENSIVE CODING
Basically defensive coding can be characterized as shown in Figure 1 below. Programmer uses one or more approaches to patch up their application depending on the need of the enterprise. For example some enterprises do not allow search box in the website to reduce the risk of being attacked. In such cases they provide the user with predefined search inputs, thus there is no need of applying security code in search box. As shown not all approaches are applicable in all injection points.

IV. EVALUATION
Evaluation Based on Deployment
We analyze each approach as shown in Figure 2 based on different deployment requirements.
Figure 2 Evaluation based on deployment requirements.

"◦" indicate method can be deployed to that injection parameter.

"x" indicate method cannot be deployed to that injection parameter.

Evaluation Based on the Attack Type

We analyzed and evaluate each proposed method as shown in Figure 3 to assess whether it is capable of addressing particular attack. Evaluation was done analytically based on our experience; we have not assess any of the method in real time practices because implementation codes of most methods are not available or some methods are not implemented.

Figure 3 Evaluation Based on the Attack Type

"◦" indicate method can successfully stop attack of that type.

"x" indicate method cannot stop attack of that type.

Conclusion

In this paper we present background of SQLIA. We introduced most common approach used to prevent SQLIA, categorized such approaches into a survey of different methods, surveyed existing techniques related to each method, identify some common issues related to each approach as well as programmer mistakes. We also evaluated each approach based on its inherited deployment requirement and each technique based on the type of attack able to address.

REFERENCE


