CP400S Project Report

Group #3

Member 1: Zehao Liu 193074000

Member 2: Jialong Zhang 190227130

Wilfrid Laurier University

Computer Security – CP400S

Dr. Gao Shuan

Wednesday, Apr 5, 2023

Table of Contents

CP460A Project Report

1 Introduction

1.1 What is SQL injection attack?

SQL injection attacks are a type of cyberattack that targets databases that use SQL (Structured Query Language) to manage and access data. Attacker can exploit vulnerabilities in a website or application to inject malicious SQL code into a database.

1.2 Purpose

The purpose of this project report is to explore SQL injection attacks and to identify effective ways to detect and prevent them. By the end of the project, a comprehensive of SQL injection attacks and practical recommendations for mitigating this threat.

2 Background

2.1 when it discovered?

The SQL injection exploit was first documented in 1998 by cybersecurity researcher and hacker Jeff Forristal. His findings were published in the long running hacker zine *Phrack*.

2.2 Consequences

It occupied a wide range of consequences, depending on the attacker's goals. For example, an attacker may be able to steal sensitive data such as credit card numbers or personal information by modify or delete data in the database.

3 How SQL injection attack achieved

In a SQL injection attack, an attacker insert malicious code interface given by the web application. If web application develop did not prepared prevention mechanism. Those code can

be executed by the database. This can allow the attacker to access or modify sensitive data, steal login credentials, or even take control of the entire database.

4 Injection Types

SQL injection attacks can be classified into several types, including in-band attacks, inferential attacks, and out-of-band attacks.

4.1 In-Band Attack

In-band attacks involve the attacker using the same communication channel as the application to both send the attack payload and receive the result of the attack. There are three types of in-band attacks:

4.1.1 Tautology

A tautology attack involves adding a statement that is always true to the end of a query. For example, consider the following SQL statement:

SELECT * FROM users WHERE username = 'admin' AND password = 'password123'

To perform a tautology attack, an attacker could append ' OR 1=1;' ' after the admin. The resulting query would be:

SELECT * FROM users WHERE username = 'admin OR 1=1;" AND password = 'password123'

This query would always return true, as 1=1 is always true. As a result, the attacker would be able to bypass the authentication and gain unauthorized access to the system.

4.1.2 End-of-Line Comment

After injecting code into a particular field, legitimate code that follows are nullified through usage of end of line comments. For example, consider the following SQL statement:

SELECT * FROM users WHERE username = 'admin' AND password = 'password123'

To perform an end-of-line comment attack, an attacker could add the comment symbol to the end of the statement, followed by the attack payload. For example:

SELECT * FROM users WHERE username = 'admin';#' AND password = 'password123'

This query would omit all the code after #. As a result, the attacker would be able to bypass the authentication and gain unauthorized access to the system.

4.1.3 Piggybacked Queries

A piggybacked query attack involves adding additional queries to the original query to achieve the attacker's goals.

For example, consider the following SQL statement:

SELECT * FROM users WHERE username = 'admin' AND password = 'password123'

To perform a piggybacked query attack, an attacker could add additional queries to the original query. For example: '; DELETE FROM users; #

The resulting query would be:

SELECT * FROM users WHERE username = 'admin' AND password = 'password123'; DELETE FROM users; #'

The semicolon (;) would terminate the original query, and the attacker could then add their own query to delete all users from the database. The double hyphens (--) would comment out the remaining part of the statement, ensuring that the attacker's query would be executed without error. As a result, the attacker would be able to delete all users from the system.

4.2 Inferential Attack

Inferential attacks, also known as blind SQL injection attacks, do not directly display the results of a query. Instead, the attacker uses the application's response to infer whether the injected query is true or false. For example, the attacker might input a username that does not exist in the database and observe whether the application returns an error message indicating that the user does not exist. This allows the attacker to deduce the structure of the underlying database and launch further attacks.

Example of Inferential Attack: Suppose an attacker wants to determine whether a particular username exists in a database. They could inject the following SQL statement:

SELECT COUNT(*) FROM users WHERE username = 'admin'

If the query returns a non-zero value, then the attacker knows that the username 'admin' exists in the database. They can repeat this process for other usernames until they identify a valid username to use for further attacks.

4.3 Out-of-Band Attack

In an out-of-band attack, the attacker uses an alternative channel to extract data from the database, rather than relying on the application's response. This can be done by injecting queries that cause the database server to perform an action, such as sending an email or making a DNS lookup, that can be monitored by the attacker. Out-of-band attacks are less common than in-band attacks, but they can be more difficult to detect and prevent since they do not rely on the application's response.

5 Prevention mechanism

The three core reasons that SQL injection can be success is user input is untrusted, user input executed by SQL database and the user knows too many about the function or variable names inside the system.

Here are serval common prevention methods to SQL injection and reason that how they prevent:

1. Validating and sanitizing user input: Removing any special characters that user inserted.
2. Using parameterized SQL queries and prepared statement: These methods involve separating the SQL code from user input to prevent malicious code from being executed. It is also a kind of standardization toward user input.
3. Limiting error messages: Avoid providing detailed error messages to users, as these can provide attackers with valuable information about the database structure and potential vulnerabilities.
4. Implementing least privilege access control: Restrict database user permissions to only the necessary operations and functions.
5. Use a web application firewall: A web application firewall can provide an additional layer of protection against SQL injection attacks by monitoring and filtering incoming traffic.

Specifically, the sanitization and standardization of user input prevented attacks that relied on malicious input to exploit vulnerabilities in the system. This ensured that user input was restricted to expected values and prevented attempts to inject malicious SQL code.

6 Implementation

6.1 Environment

To evaluate the effectiveness of technical means to prevent SQL injection attacks, attack environment has to be developed. Two web pages that use SQL to retrieve and display data had been created. One web page without any protection and another with four prevention mechanisms that will be discussed in 6.2. All types of in band attacks are implemented to compromise two systems.

6.2 Prevention mechanism applied

First four prevention mechanism in chapter 5 were applied:

1. Build-in php provide stripslashes() and htmlspecialchars() to sanitize user input.
2. prepared($sql) separating the SQL code from user input.
3. if (!$_SESSION['logged']) header("Location: " . “func/logIn.php"); make each unexpected attempt back to the login page.
4. if($_SESSION[“isAdmin”]) proved difference access for diverse user.

6.3 Result

The experiments shows that version without prevention mechanisms is effortless to change the database. While the safe version showed that the technical means be employed were effective in preventing SQL injection attacks. None of the attacks attempted were successful compromising the system.

6.4 Conclusion

Overall, our experiments showed that the technical means we employed were highly effective in preventing SQL injection attacks. By validating and sanitizing user input, using parameterized SQL queries, implementing least privilege access control, and limiting error messages, we were able to significantly reduce the risk of SQL injection attacks.

7 Conclusion

In conclusion, SQL injection attacks are a serious threat to data security that can lead to severe consequences for affected organizations. However, with proper prevention mechanisms can efficiently prevented the attack. The project report provided an overview of SQL injection attacks, how they are achieved, the different types of injection, and prevention mechanisms and its implementation.

Reference

1. W. Halfond, A. Orso, and P. Manolios, "Classification and Evaluation of Attacks in Web Applications," IEEE Transactions on Software Engineering, vol. 36, no. 5, pp. 617-635, Sept.-Oct. 2010.
2. S. S. Bhattacharyya and S. K. Das, "A Review on SQL Injection Attack and Prevention Techniques," International Journal of Computer Applications, vol. 66, no. 11, pp. 12-16, Apr. 2013.
3. N. K. Jaiswal and D. B. Ojha, "An Overview of SQL Injection Attack Techniques," International Journal of Computer Science and Mobile Computing, vol. 3, no. 5, pp. 902-906, May 2014.
4. M. M. Alam, S. A. Aljohani, and A. H. Alabdulkarim, "A Comprehensive Study on SQL Injection Attack: Detection and Prevention Techniques," Journal of Information Security, vol. 8, no. 4, pp. 269-285, Oct. 2017.
5. N. K. Jaiswal and D. B. Ojha, "A Survey on Prevention Techniques for SQL Injection Attack," International Journal of Engineering Research and Technology, vol. 3, no. 2, pp. 169-172, Feb. 2014.