СЕКЦИЯ «ЗАЩИТА ИНФОРМАЦИИ»

UDC 004.056

SQL INJECTION ATTACKS AND PREVENTION

Deng Y.Y., gr.367311/ master student

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Vrublevsky I.A. – PhD in Technical Sciences

Annotation. The paper discusses SQL injection, a common editing language used to inject malicious SQL commands into input forms or queries in order to gain access to a database or manipulate its data. Corresponding precautions for each SQLI are considered.

Keywords. SQL language injection, attacks, query, prevention methods.

Introduction. A web application is a software system that provides an interface to its users through a web browser on any operating system (OS). Despite their growing popularity, web application security threats have become more diverse, resulting in more severe damage. Malware attacks, particularly SQLI attacks, are common in poorly designed web applications. This vulnerability has been known for more than two decades and is still a source of concern. This highlights the need to develop effective methods to protect data from unauthorized access. In this context, we need an effective self-detection method for prevention.

The main part. To grasp how SQL language can be abused, SQLI attackers and defenders need to understand how it functions [1]. The queries must be prepared in the SQL language and adhere to specified syntax to retrieve data from databases or modify the data, such as:

```
"SELECT * FROM authortable WHERE book_name = 'Advanced Database Systems'"
```

The aforementioned search will provide all books with book name "Advanced Database Systems." The queries are typically typed into a web browser and sent to the database management system [3].

What if the attacker in this case extends the original SQL query?

For example:

```
"SELECT * FROM authortable WHERE book_name = 'Advanced Database Systems' OR
'1'='1'"
```

The above query will return all book names in the database, not just the book names labeled as "Advanced Database Systems," because the sentence '1=1' is always true. If the stored list of book names is not a secret and the previous example might not pose a problem [4]. If successful, it might return sensitive data, such as passwords, bank accounts, trade secrets, and personal information, which might be regarded as a privacy breach among other negative effects. However, it could be applied to value using different syntax.

By using SQL injection, the attackers can alter the SQL statement by replacing user's supplied data with their own data as shown in Figure 1. Thus, the attackers can have direct access to a database server to retrieve confidential information. However, the impacts of the SQL injection attacks are far reaching and they vary depending on the database applications. Some of these impacts of a successful SQL injection attacks include authentication bypass, information disclosure, compromised data integrity, compromised availability of data, and remote command execution. Authentication bypass enables an attacker to access database application by using fake username and password. Information disclosure allows an attacker to obtain sensitive data from the database. Compromised data integrity assists an attacker to change some contents of the data in the database. Compromised availability data enables an attacker to delete some information from the database.

Remote command execution allows an attacker to affect the host operating system [3].

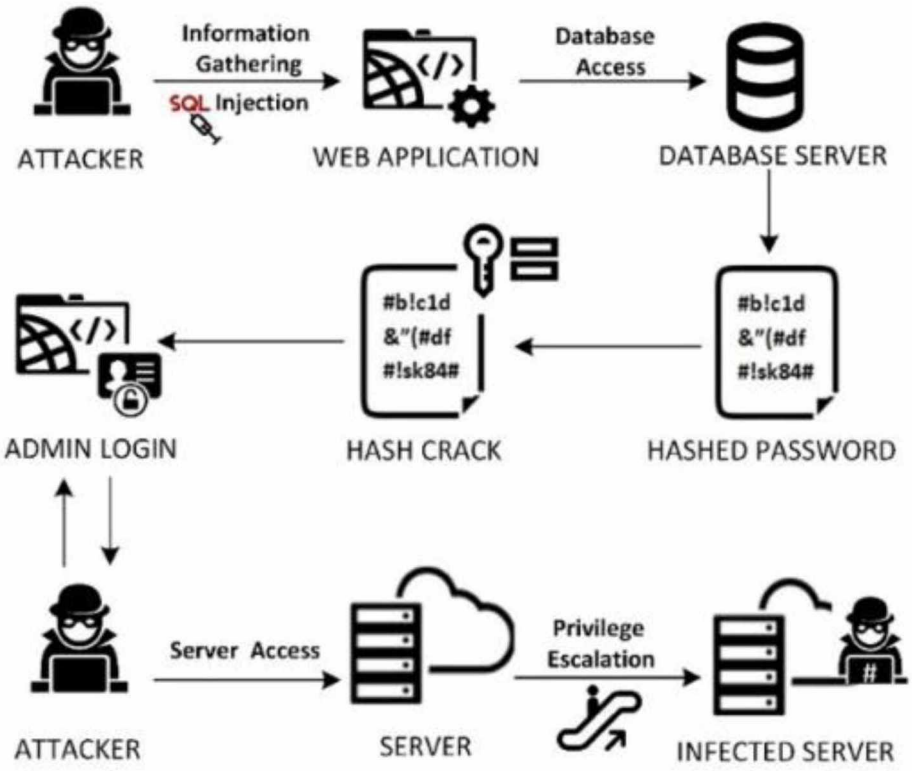

Figure 1. How SQL injection attacks work

Most common attacks on SQLI. As we know, SQL is a programming language that is used to create, update, and access data in a database. A hacker can intentionally cause the application to fail, delete data, steal data, or gain unauthorized access by carefully crafting SQL commands [5]. To address the aforementioned issue, we provide a detailed overview of the various types of SQLI attacks discovered to date. For each type of attack, we provide explanations and examples of how such attacks can be carried out.

Tautology attack

The attacker attempts to use a conditional query argument to test always true in the tautology attack, such as (1=1) or (− −). The attacker injects the condition and transforms it into a tautology that is always valid using the WHERE clause. This type of attack is commonly used to access databases without requiring authentication on websites [1].

Union query

In this category of attack, the UNION operator is only used if both queries have the same form, the attacker constructs a SELECT statement that is similar to the original query [5]. To do so, it must be known that the correct table name, as well as the number of columns and their data types from the first query. As a result, two conditions may be satisfied, or an attack on the union query will be launched, and each query returns the same number of columns [6]. If the data type of a column is incompatible with string data, the injected query will fail.

Piggybacked query

The piggybacked query attack concatenates more query statements onto the initial query ";" [1]. This technique is especially risky since it enables an attacker to insert virtually any SQL command. Data extraction, addition or modification of data, denial of service (DoS), and remote command execution are all examples of determined attacks [16]. In this type of attack, an attacker attempts to inject additional queries into the original query. Unlike other forms, attackers attempt to add new and distinct queries that "piggyback" on the original query rather than changing it [3].

Illegal or incorrect query

This kind of attacker takes advantage of a database query that was improperly executed [1]. It will show database error messages, which frequently provide crucial facts that enable an attacker to learn the application's database specifics. The attack goal includes identifying injection parameters,

performing database fingerprinting, and extracting data. This attack assists an attacker in gathering critical information concerning the nature and function of the back-end database of a web application [8]. The attack is thought to be a practice run for future attacks aimed at gathering information. This attack takes advantage of the fact that the default error pages on application servers are frequently excessively descriptive [7].

Stored procedure query

Here, an attacker can use this technique to modify the database's stored procedures [1]. Both authorized and unauthorized users will receive true or false results from the process. The users can save their features and use them whenever they want. A collection of SQL queries is provided with the feature to use it. The intruder uses malicious SQL codes to execute the database's built-in stored procedures [8]. This leads to cause the cached stored procedure query plans being recompiled. A stored procedure's constraint is that it can only be used in the database.

Inference query

In this attack, the query is recast as an operation and executed based on the answer to a true or false question about database data values [10]. For this method of injection, attackers attempt to break into a site that has been sufficiently protected that when an injection is successful and there is no accessible feedback in the form of database error messages. Since database error messages are not available to provide feedback and attackers must rely on another approach to get a response from the database.

Existed prevention methods. This section provides an overview of the existed approach and explains the rationale for detecting SQL injection.

A. Any comment character is present

Having a single or multiple lines comment character in a statement generated from an application is not a common practice by the programmers. The reason is that when a user executes her/his query in the applications, she/he never includes a comment with it. On the other hand, the attackers use this approach to include a comment character after their malicious command to let the system neglect the rest of the query maintained by the application.

B. Number of semicolons

The SQL statements are terminated with one semicolon at the end of it. When an attacker injects her/his malicious commands in the middle of the statement, she/he places a semicolon and then comments the rest of the statement, which result in having multiple semicolons. As we are dealing with it as a string and not command, we can identify if multiple semicolons are present or not.

C. Presence of always true conditions

Always true conditions rarely can be found in the benign SQL statements while they are the most common terms used by attackers in their injection to make the condition always satisfied after the OR operator. Researchers have considered this approach; however, they only have considered the checking for number1=number1, variable1=variable1, and 'string'='string'. In our approach, we add the checking for number1!=number2, variable1!=variable2, and 'string1'!='string2' so that it will also always return a true value.

D. The number of commands per statement

Regular queries generated by applications usually consist of one request only such as SELECT. When an attacker injects her/his code in the original SQL statement, it results in having one statement with more than one requests.

E. Presence of abnormal commands

When an application reads the input from a user and inserts it into a statement, it usually uses SELECT, INSERT, or DELETE requests. Having statement coming from the application with DROP, CREATE, COMMIT, ROLLBACK, GRANT, REVOKE, and DECLARE, requests may indicate that a malicious action is happening.

F. Presence of special keywords

Having a statement coming from an application, which is operated by a user requesting for the version of the SQL server, could indicate a malicious action. A set of similar keywords are used to

be checked in the statement through the developed features identifier code. If any of the keywords is present, the program will detect it.

Conclusion. This paper involves SQLI statements and analyzes them based on their principles to detect whether malicious commands are injected into the system and protect the system from this type of attack.

List of literature:

1. A comprehensive study on SQL injection attacks, their mode, detection and prevention / S. Dasmohapatra [et al.] // Proceedings of Second Doctoral Symposium on Computational Intelligence: DoSCI 2021, Springer Singapore, 2022. –P. 617-632.

2. Static analysis approaches to detect SQL injection and cross-site scripting vulnerabilities in web applications: a survey / M.K Gupta [et al.] // Int Conf Recent Adv Innov Eng ICRAIE, 2014. –P. 9-13.

3. Open source web application security: a static analysis approach / M. Alenezi, Y. Javed // 2016 International Conference on Engineering & MIS (ICEMIS). IEEE. 2016. –P. 1-5.

4. A review of static code analysis methods for detecting security faws / B.S Basutakara, P.N. Jeyanthi // J Univ ShanghaiSci Technol. 2021. –P. 47-53.

5. Dynamic multi-process information fow tracking for web application security / S. Nanda [et al.] // Proceedings of the 2007 ACM/IFIP/USENIX international conference on Middleware companion, 2007. –P.1-20.

6. Hybrid approach to detect SQLi attacks and evasion techniques / A. Makiou [et al.] // 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing. IEEE, 2014. –P. 452-456.

7. Analysis of vulnerability detection tool for web services / A. Murugan [et al.] // Int J Eng Technol, 2018. –P. 3-8.

8. Design and implementation of SQL injection vulnerability scanning tool / P. Techniques [et al.] // Journal of Physics: Conference Series. IOP Publishing, 2020. –P. 1-6.

9. Vulnerability detection and prevention of SQL injection / P.P Anaswara [et al.] // International Journal of Engineering & Technology, 2018. –P. 16-18.

10. Detection of SQL injection attacks: a machine learning approach / M. Hasan [et al.] // 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA). IEEE, 2019. –P. 1-6.