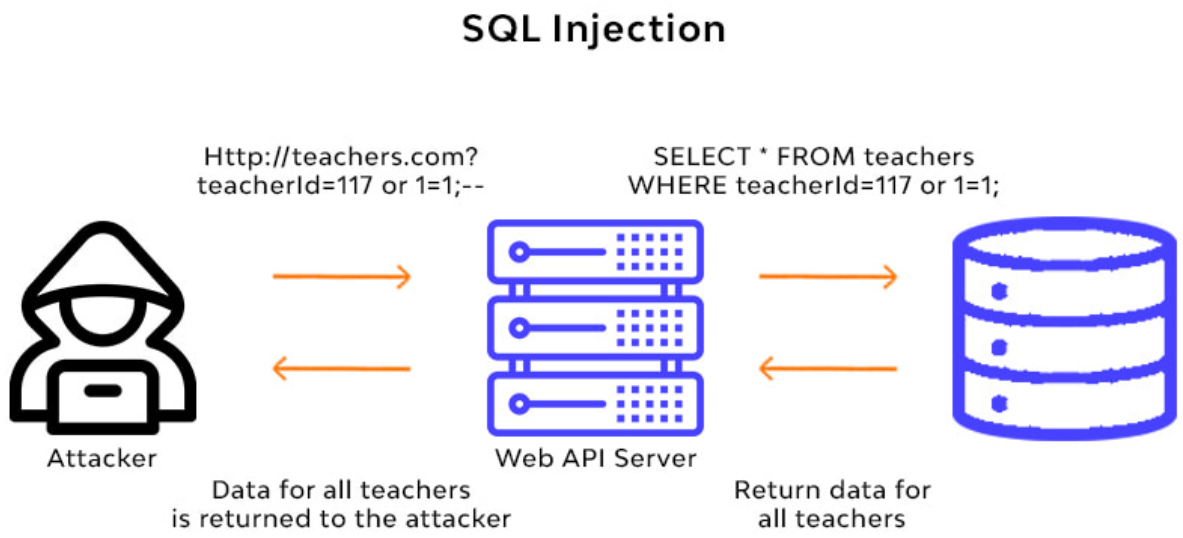
SQL Injection



What is SQL?

Structured Query Language (SQL) is a powerful database tool that is used to perform operations such as creating, maintaining and retrieving data stored in the relational database. It is a standard language for data manipulation in a Database Management System (DBMS).

This query language became the standard of ANSI in the year of 1986 and ISO in the year of 1987.

Types of SQL Statements:

- DML (Data Manipulation Language)
- DDL (Data Defination Language)
- DCL (Data Control Language)
- TCL (Transaction Control Language)

Some of The Most Important SQL Commands:

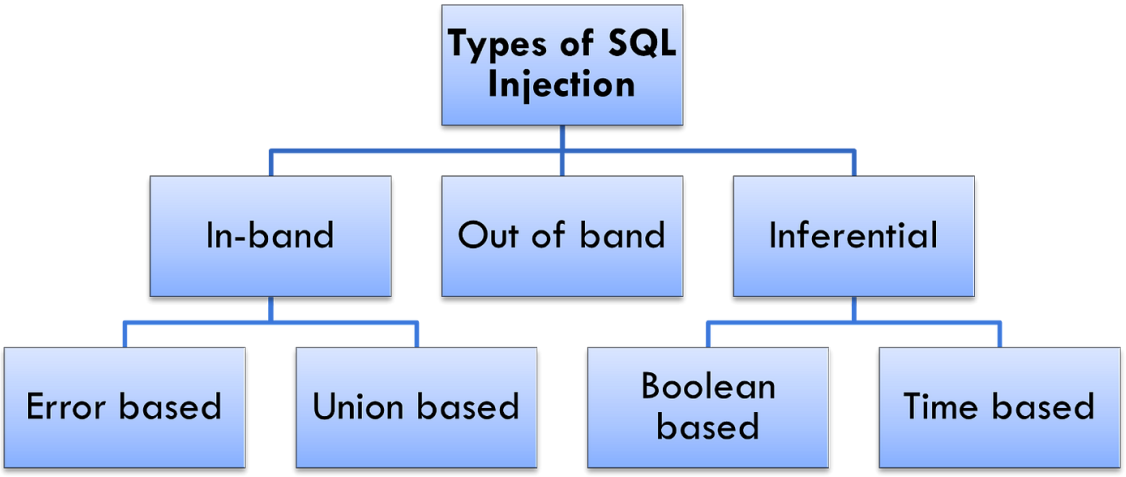
- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

What is SQL Injection?

SQL injection enables an attacker to tamper with database queries that an application conducts. In most cases, it enables an attacker to view data that they would not typically be able to access.

Other users' data or any other data that the application itself has access to may fall under this category. In many instances, an attacker can update or remove this data, permanently altering the application's behaviour or content. An attacker may in some circumstances escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure or launch a denial-of-service attack.

Types of SQL Injection:



1. In-Band SQLi

The attacker uses the same channel of communication to launch their attacks and to gather their results. In-band SQLi's simplicity and efficiency make it one of the most common types of SQLi attacks. There are two sub-variations of this method:

- **Error-Based SQLi**:
  The attacker performs actions that cause the database to produce error messages. The attacker can potentially use the data provided by these error messages to gather information about the structure of the database.

  Example:
  http://www.example.com/index.php?item=1

  The attacker can try adding a single quote at the end of the parameter value:

  http://www.example.com/index.php?item=1'

  If the database returns an error, the attack succeeded:If you have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "VALUE".

  This error message provides the attacker with:
  - ➔ Information about the database used—MySQL
  - ➔ The exact syntax that caused the error—single quote
  - ➔ Where the syntax error occurred in the query—after the parameter value

- **Union-Based SQLi**:
  This technique takes advantage of the UNION SQL operator, which fuses multiple select statements generated by the database to get a single HTTP response. This response may contain data that can be leveraged by the attacker.

  The UNION keyword lets you execute one or more additional SELECT queries and append the results to the original query.

Example:
SELECT a, b FROM table1 UNION SELECT c, d FROM table2

This SQL query will return a single result set with two columns, containing values from columns a and b in table1 and columns c and d in table2.

For a UNION query to work, two key requirements must be met:

- ➔ The individual queries must return the same number of columns.
- ➔ The data types in each column must be compatible with the individual queries.

To carry out an SQL injection UNION attack, you need to ensure that your attack meets these two requirements. This generally involves figuring out:

- ➔ How many columns are being returned from the original query?
- ➔ Which columns returned from the original query are of a suitable data type to hold the results from the injected query?

2. Inferential (Blind) SQLi

In order to understand the server's architecture, the attacker sends data payloads to the server and watches how it responds and behaves. Blind SQLi is the name of this technique since the attacker cannot see information about the inband attack because no data is passed from the website database to the attacker.

Blind SQL injections are often slower to run but may be just as dangerous because they depend on the server's behaviour and reaction. These are some categories for blind SQL injections:

- **Boolean-Based SQLi**:

  The attacker sends a SQL query to the database prompting the application to return a result. The result will vary depending on whether the query is true or false. Based on the result, the information within the HTTP response will modify or stay

unchanged. The attacker can then work out if the message generated a true or false result.

- **TIme-Based SQLi**:

  The attacker sends a SQL query to the database, which makes the database wait (for a period of seconds) before it can react. The attacker can see from the time the database takes to respond, whether a query is true or false. Based on the result, an HTTP response will be generated instantly or after a waiting period. The attacker can thus work out if the message they used returned true or false, without relying on data from the database.

3. Out-of-Band SQLi

The attacker can only carry out this form of attack when certain features are enabled on the database server used by the web application. This form of attack is primarily used as an alternative to the in-band and inferential SQLi techniques.

Out-of-band SQLi is performed when the attacker can't use the same channel to launch the attack and gather information, or when a server is too slow or unstable for these actions to be performed. These techniques count on the capacity of the server to create DNS or HTTP requests to transfer data to an attacker.

Prevention:

- **Use Planned Commands with Parameterized Queries**: For SQL calls, utilize arranged expressions as opposed to building dynamic queries utilizing string links.

- **Perform Information Validation**: Validate input information to identify noxious characters. For NoSQL databases, moreover, approve input types against anticipated sorts.

- **Provide the Least Privilege**: To limit the potential harm of a prosperous injection assault, do not allot DBA or administrator rights to your application accounts. Additionally, limit the benefits of the working framework account that the database procedure runs under.

- **Avoid Using Shared Database Accounts**: Do not employ shared database accounts between various websites or web applications.

- **Keep Database Credentials Isolated and Encrypted**: If you are thinking about where to store your database credentials, additionally consider what amount of harming it tends to be on the off chance that it falls into inappropriate hands. So consistently store your database credentials in a different document and encrypt it safely to ensure that the aggressors can't profit a lot.

References:

- PortSwigger.net
- Beaglesecurity.com
- Medium.com
- Acunetix.com
- Infosecacademy.io