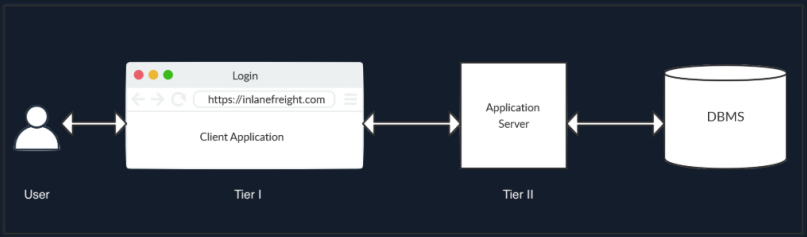
Introduction

Most modern web applications utilize a database structure on the back-end. Such databases are used to store and retrieve data related to the web application, from actual web content to user information and content, and so on. To make the web applications dynamic, the web application has to interact with the database in real-time. As HTTP(S) requests arrive from the user, the web application's back-end will issue queries to the database to build the response. These queries can include information from the HTTP(S) request or other relevant information.



When user-supplied information is used to construct the query to the database, malicious users can trick the query into being used for something other than what the original programmer intended, providing the user access to query the database using an attack known as SQL injection (SQLi).

SQL injection refers to attacks against relational databases such as `MySQL` (whereas injections against non-relational databases, such as MongoDB, are NoSQL injection). This module will focus on `MySQL` to introduce SQL Injection concepts.

SQL Injection (SQLi)

Many types of injection vulnerabilities are possible within web applications, such as HTTP injection, code injection, and command injection. The most common example, however, is SQL injection. A SQL injection occurs when a malicious user attempts to pass input that changes the final SQL query sent by the web application to the database, enabling the user to perform other unintended SQL queries directly against the database.

There are many ways to accomplish this. To get a SQL injection to work, the attacker must first inject SQL code and then subvert the web application logic by changing the original query or executing a completely new one. First, the attacker has to inject code outside the expected user input limits, so it does not get executed as simple user input. In the most basic case, this is done by injecting a single quote (`'`) or a double quote (`"`) to escape the limits of user input and inject data directly into the SQL query.

Once an attacker can inject, they have to look for a way to execute a different SQL query. This can be done using SQL code to make up a working query that executes both the intended and the new SQL queries. There are many ways to achieve this, like using stacked queries or using Union queries. Finally, to retrieve our new query's output, we have to interpret it or capture it on the web application's front end.

Use Cases and Impact

A SQL injection can have a tremendous impact, especially if privileges on the back-end server and database are very lax.

First, we may retrieve secret/sensitive information that should not be visible to us, like user logins and passwords or credit card information, which can then be used for other malicious purposes. SQL injections cause many password and data breaches against websites, which are then re-used to steal user accounts, access other services, or perform other nefarious actions.

Another use case of SQL injection is to subvert the intended web application logic. The most common example of this is bypassing login without passing a valid pair of username and password credentials. Another example is accessing features that are locked to specific users, like admin panels. Attackers may also be able to read and write files directly on the back-end server, which may, in turn, lead to placing back doors on the back-end server, and gaining direct control over it, and eventually taking control over the entire website.

Prevention

SQL injections are usually caused by poorly coded web applications or poorly secured back-end server and databases privileges. Later on, we will discuss ways to reduce the chances of being vulnerable to SQL injections through secure coding methods like user input sanitization and validation and proper back-end user privileges and control.

Next →

Mark Complete & Next

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left