EXPERIMENT NO:8

AIM: To perform SQL-Injection

THEORY:

What is SQL Injection?

SQL Injection is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data: customer information, personal data, trade secrets, intellectual property, and more. SQL Injection attacks are one of the oldest, most prevalent, and most dangerous web application vulnerabilities.

To make an SQL Injection attack, an attacker must first find vulnerable user inputs within the web page or web application. A web page or web application that has an SQL Injection vulnerability uses such user input directly in an SQL query. The attacker can create input content. Such content is often called a malicious payload and is the key part of the attack. After the attacker sends this content, malicious SQL commands are executed in the database.

SQL is a query language that was designed to manage data stored in relational databases. You can use it to access, modify, and delete data. Many web applications and websites store all the data in SQL databases. In some cases, you can also use SQL commands to run operating system commands. Therefore, a successful SQL Injection attack can have very serious consequences.

Attackers can use SQL Injections to find the credentials of other users in the database. They can then impersonate these users. The impersonated user may be a database administrator with all database privileges.

SQL lets you select and output data from the database. An SQL Injection vulnerability could allow the attacker to gain complete access to all data in a database server.

SQL also lets you alter data in a database and add new data. For example, in a financial application, an attacker could use SQL Injection to alter balances, void transactions, or transfer money to their account.

You can use SQL to delete records from a database, even drop tables. Even if the administrator makes database backups, deletion of data could affect application availability until the database is restored. Also, backups may not cover the most recent data.

In some database servers, you can access the operating system using the database server. This may be intentional or accidental. In such case, an attacker could use an SQL Injection as the initial vector and then attack the internal network behind a firewall.

Preventing SQL Injection

Server-side scripting languages are not able to determine whether the SQL query string is malformed. All they can do is send a string to the database server and wait for the interpreted response.

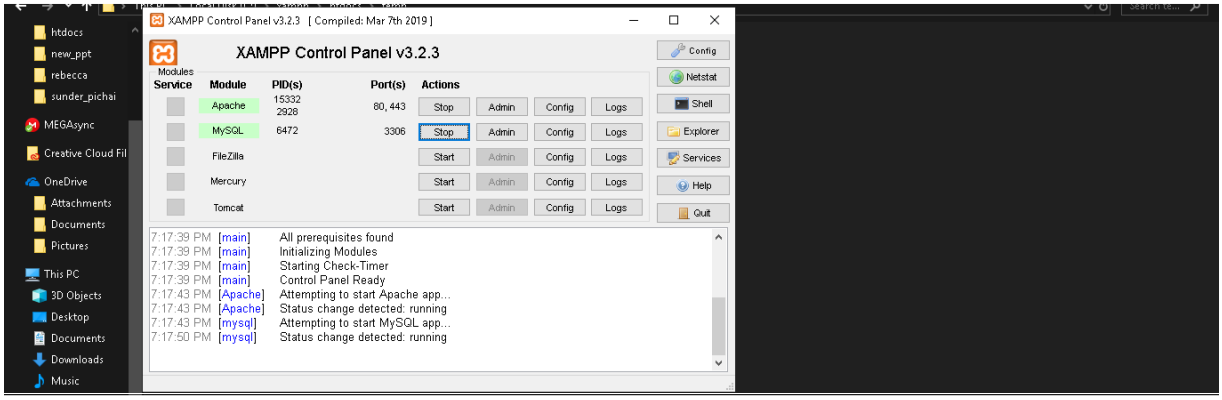
There are an infinite number of ways to sanitize user input, from globally applying PHP's addslashes() to everything (which may yield undesirable results), all the way down to applying the sanitization to "clean" variables at the time of assembling the SQL query itself. However, applying sanitization at the query itself is a very poor coding practice and difficult to maintain or keep track of. This is where database systems have employed the use of prepared statements.

Using Prepared Statements as SQL Injection Prevention

When you think of prepared statements, think of how printf works and how it formats strings. Literally, you assemble your string with placeholders for the data to be inserted, and apply the data in the same sequence as the placeholders. SQL prepared statements operate on a very similar concept, where instead of directly assembling your query string and executing it, you store a prepared statement, feed it with the data, and it assembles and sanitizes it for you upon execution.

Screenshots:

Step1: Install XAMPP server from https://www.apachefriends.org/download.html on Windows



Step2: Write a PHP script for the webpage shown below and save it in temp folder in C:\xampp\htdocs\temp

---

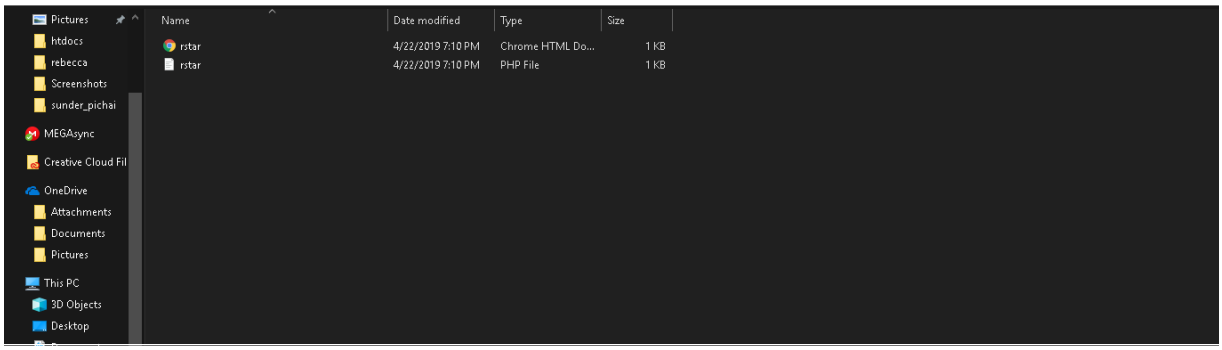


Figure 2:htdocs/temp

Step3: Make database[css] and add few tables in it as shown in phpMyAdmin

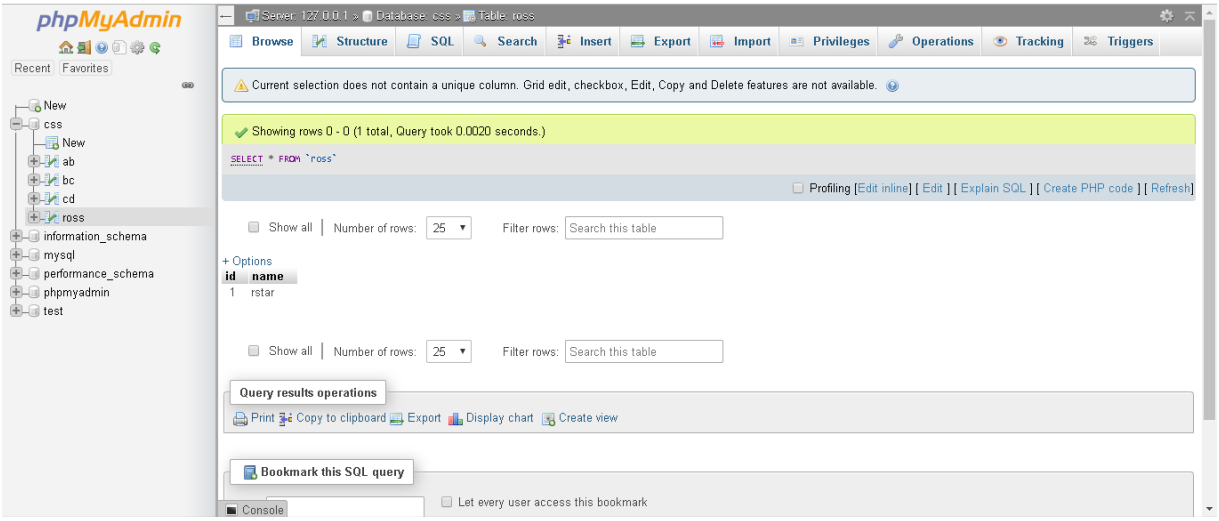


Figure 3: phpmyadmin

Step4: Run Script

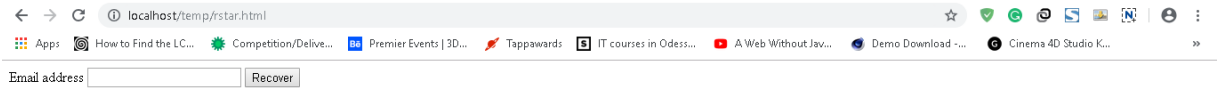


Figure 4: Run Script

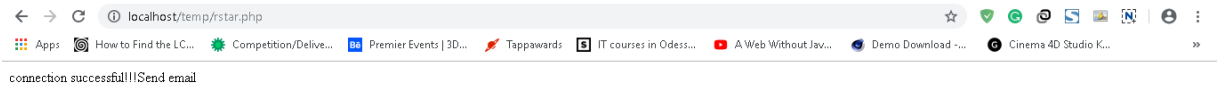


Figure 5: executing script

Step5: Perform SQL Injection by passing '; DROP TABLE ross; --

Figure 6: SQL Injection

Step6: Check the database for ross it should be dropped/deleted

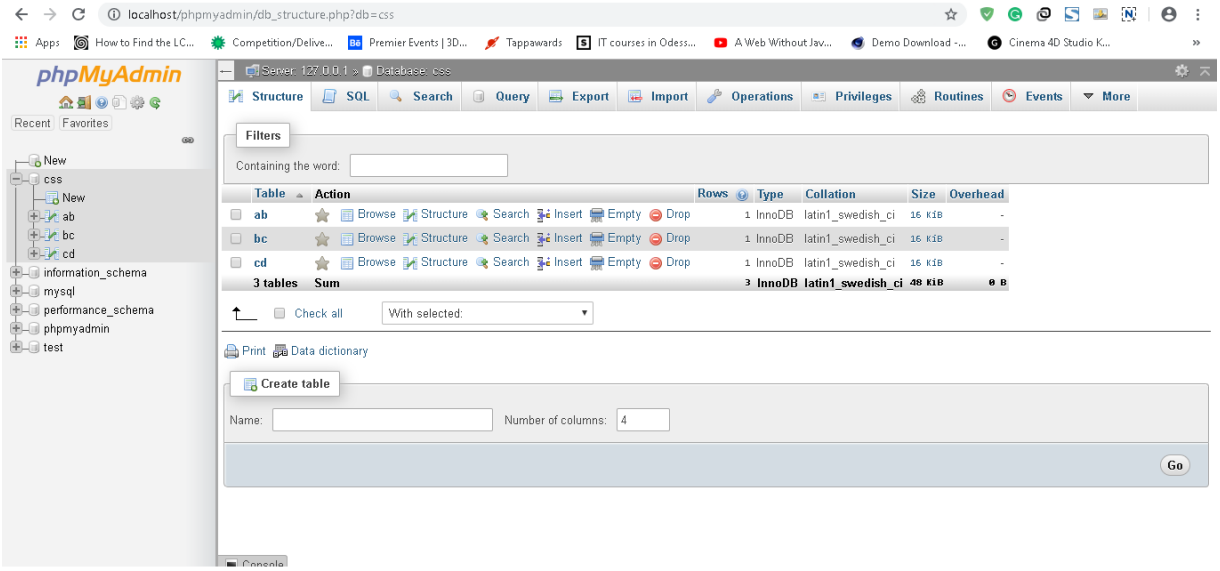


Figure 7: Table ross deleted

Step 7: Preventing SQL Injection

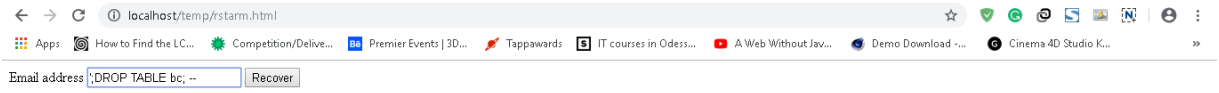


Figure 8: SQL injection in modified code

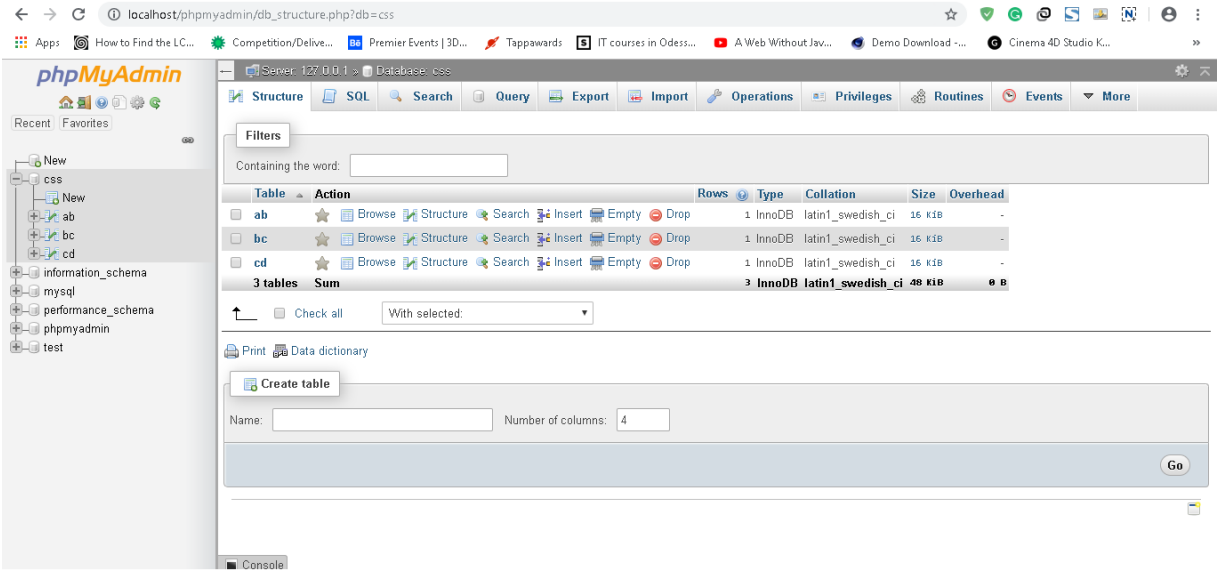


Figure 9: Table is not dropped

Scripts:

Figure 10 : PHP script vulnerable to SQL injection

Figure 11: html code

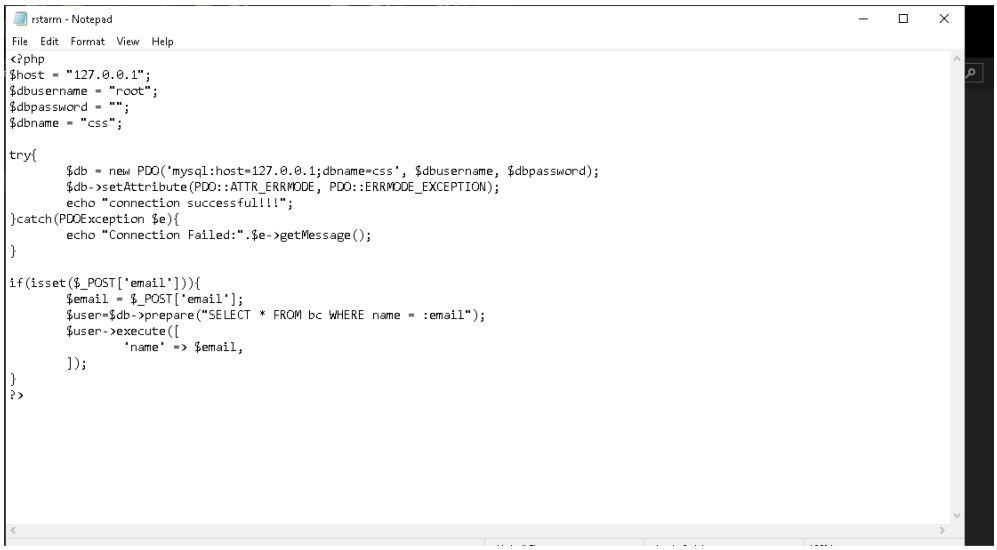


Figure 12: modified PHP code to prevent SQL injections

Conclusion:

SQL Injection was performed on the database created which was then accessed by PHP script i.e. on application of malformed string to the input field a serious damage to database was done which was then to prevent this damage prepare statement was used.