

Exercise guide

1. Describe system resources

Resources	Sender(attacker)	Receiver(victim)
OS	Ubuntu 18.04	Windows 10
IP address		
URL	127.0.0.1	127.0.0.1
Web browser	Google chrome	Google chrome
CSS language	JavaScript	HTML, CSS, JavaScript
Web server	Apache	Apache
Web application	PHP	PHP
DB server script	MySQL	MySQL
Homepage		

2. Choose one SQL injection query model for SQL test.

https://www.w3schools.com/sql/sql_injection.asp

<https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications/>

SQL Injection Based on `1=1` is Always True.

SQL Injection Based on `""=""` is Always True.

3. Explain what is the vulnerabilities of SQL Injection query model.

SELECT * FROM Users WHERE UserId = 105 OR 1=1;

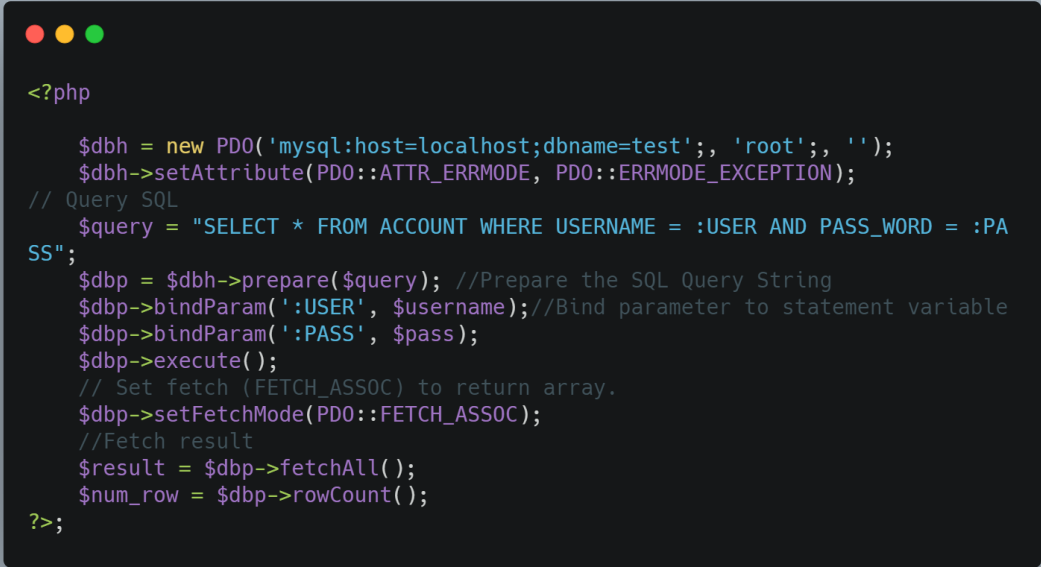
The SQL above is valid and will return ALL rows from the "Users" table,

since OR 1=1 is always TRUE.

SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;

A hacker might get access to all the username and password in database, by simply inserting 105 OR 1=1 into the input field.

4. Modify SQL Injection query model to normal query without vulnerabilities



```
<?php

$dbh = new PDO('mysql:host=localhost;dbname=test', 'root', '');
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// Query SQL
$query = "SELECT * FROM ACCOUNT WHERE USERNAME = :USER AND PASS_WORD = :PASS";
$dbp = $dbh->prepare($query); //Prepare the SQL Query String
$dbp->bindParam(':USER', $username); //Bind parameter to statement variable
$dbp->bindParam(':PASS', $pass);
$dbp->execute();
// Set fetch (FETCH_ASSOC) to return array.
$dbp->setFetchMode(PDO::FETCH_ASSOC);
//Fetch result
$result = $dbp->fetchAll();
$num_row = $dbp->rowCount();

?>
```

5. Explain how to prevent sql-injection.

5.1. Using Prepared Statements (with Parameterized Queries):

Using Prepared Statements is one of the best ways to prevent SQL injection. It's also simple to write and easier to understand than dynamic SQL queries.

How to prevent and protect SQLI

There are several methods to prevent SQL injection vulnerabilities:

Input validation

Input validation makes sure it is the accepted type, length, format, and so on. Only the value which passes the validation can be processed. It helps counteract any commands inserted in

the input string. In a way, it is similar to looking to see who is knocking before opening the door.

Parametrized queries

Parameterized queries are a means of pre-compiling an SQL statement so that you can then supply the parameters in order for the statement to be executed. This method makes it possible for the database to recognize the code and distinguish it from input data.

It is possible to use parameterized queries with the MySQLi extension, but PHP 5.1 presented a better approach when working with databases: PHP Data Objects (PDO). PDO adopts methods that simplify the use of parameterized queries. Additionally, it makes the code easier to read and more portable since it operates on several databases, not just MySQL.

Stored procedures

Stored procedures (SP) require the developer to group one or more SQL statements into a logical unit to create an execution plan. Subsequent executions allow statements to be automatically parameterized. Simply put, it is a type of code that can be stored for later and used many times.

Escaping

Always use character-escaping functions for user-supplied input provided by each database management system (DBMS). This is done to make sure the DBMS never confuses it with the SQL statement provided by the developer.

Avoiding administrative privileges

Don't connect your application to the database using an account with root access. This should be done only if absolutely needed since the attackers could gain access to the whole system. Even the non-administrative accounts server could place risk on an application, even more so if the database server is used by multiple applications and databases.

Web application firewall

One of the best practices to identify SQL injection attacks is having a web application firewall (WAF). A WAF operating in front of the web servers monitors the traffic which goes in and out of the web servers and identifies patterns that constitute a threat. Essentially, it is a barrier put between the web application and the Internet.

① Explain your scenario

First, I choose a web application that I already build before to test SQL injection. Then, I read refereces to understand how sql injection work and choose several examples to test in my website. I have tried to access an account with given email and ' ' OR 1=1 password. The result is I can access that account and manage all data belongs to it. After that, I demo a test with username and password both entered ' ' OR 1=1. This time it print out all user information in database.

Next, I researched some methods to prevent SQLi and decided to use Parametrized queries method. I rewrited Login.php and db.inc.php from using mysqli to PDO and apply parametrized queries. Finally, I test it again with username and password both entered ' ' OR 1=1.

② References

<https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications/>

<https://portswigger.net/web-security/sql-injection>

<https://stackoverflow.com/questions/48582230/prevention-of-sql-injection-in-password-input-field-php>

https://www.w3schools.com/sql/sql_injection.asp