**SURE TRUST**

**MINI PROJECT -3**

**To understand and exploit SQL injection vulnerabilities**

**Guided by : PRESENTED BY**

**Mr.Himaneesh sir K.SAI JAYANTH**

**Sure Trust Organisation**

**Cyber Security**

**Objective:**

To understand and exploit SQL injection vulnerabilities.

**Background:**

SQL injection is a type of vulnerability that occurs when user-supplied data is not properly sanitized before being incorporated into an SQL query. This can allow attackers to inject arbitrary SQL code into the query, potentially enabling them to read sensitive data, modify data, or even execute commands on the database server.

**Procedure:**

Identify potential SQL injection vulnerabilities:

a. Review the application's code to identify locations where user-supplied data is incorporated into SQL queries.

b. Look for instances where user-supplied data is directly concatenated into SQL queries without proper sanitization.

c. Use tools such as automated vulnerability scanners to identify potential SQL injection vulnerabilities.

Verify SQL injection vulnerabilities:

a. Manually test potential SQL injection vulnerabilities using various techniques, such as:

i. Submitting single quotation marks (') to see if they are properly escaped.

ii. Injecting simple mathematical expressions to see if they are evaluated as intended.

iii. Using SQL-specific keywords and syntax to see if they are interpreted correctly.

b. Use automated tools to further verify SQL injection vulnerabilities.

Exploit SQL injection vulnerabilities:

a. Once a SQL injection vulnerability has been verified, determine the extent of the vulnerability.

b. Use the vulnerability to extract sensitive data, modify data, or execute commands on the database server.

c. Be careful not to cause data corruption or other damage to the database.

**Documentation:**

Document the steps taken to identify, verify, and exploit SQL injection vulnerabilities.

Include screenshots or other evidence to support your findings.

Provide a detailed explanation of the vulnerability and its potential impact.

Recommend mitigation strategies to prevent SQL injection vulnerabilities.

**Example:**

Scenario:

A web application allows users to search for products by name. The application uses the following SQL query to retrieve the search results:

SQL

SELECT \* FROM products WHERE product\_name LIKE '%{}%';

Use code with caution. Learn more

content\_copy

The {} placeholder is replaced with the user-supplied search term.

Step 1: Identify potential SQL injection vulnerabilities:

The user-supplied search term is directly concatenated into the SQL query without proper sanitization. This makes the application vulnerable to SQL injection.

Step 2: Verify SQL injection vulnerabilities:

To verify the SQL injection vulnerability, submit the following search term:

' OR 1=1--

This search term will cause the following SQL query to be executed:

SQL

SELECT \* FROM products WHERE product\_name LIKE '%'' OR 1=1--%'';

Use code with caution. Learn more

content\_copy

The -- comment syntax tells the database to ignore the rest of the query, so only the 1=1 part of the query will be executed. This will cause the query to return all products, regardless of the search term.

Step 3: Exploit SQL injection vulnerabilities:

Since the SQL injection vulnerability has been verified, the attacker can exploit it to extract sensitive data, modify data, or execute commands on the database server.

**Example:**

To extract the names of all users from the database, the attacker can submit the following search term:

' OR 1=1-- SELECT username FROM users;

This search term will cause the following SQL query to be executed:

SQL

SELECT \* FROM products WHERE product\_name LIKE '%'' OR 1=1-- SELECT username FROM users;%'';

Use code with caution. Learn more

content\_copy

This query will return all products, as well as the names of all users from the users table.

Mitigation:

To prevent SQL injection vulnerabilities, it is important to properly sanitize user-supplied data before incorporating it into SQL queries. This can be done using prepared statements or parameterization.

Conclusion:

SQL injection is a serious vulnerability that can have a significant impact on the security of applications. By understanding and exploiting SQL injection vulnerabilities, security professionals can better identify and mitigate these risks.