TestCase1:

Error-Based SQL Injection

1. Special Character Injection: When we provide special characters such as ', ", /, \, } etc., in the URL parameter, it can cause an SQL query to break and return an error message. These error messages often reveal valuable information, such as:

The type of database (e.g., MySQL, SQL Server, PostgreSQL).

The exact line number or part of the query where the error occurred. This helps an attacker understand the structure of the backend database.

TestCase2:

Order By Injection:

2. By using the ORDER BY clause in the URL parameter (e.g., id=1 ORDER BY 2), an attacker can determine how many columns are present in the database query. For example:

If ORDER BY 2 shows the normal page content, but ORDER BY 3 throws an error or breaks the page, it means the query only uses 2

columns. This helps attackers structure their injection payloads accurately.

3. Usage in GET and POST Methods: Error-based SQL injection can be tested through:

GET Method: By modifying the query parameters in the URL.

POST Method: By injecting payloads in the body of the request.

TestCase3:

Union-Based SQL Injection

Union-based SQL Injection is a technique used to extract data from the database by combining the result of the original query with another SELECT query using the UNION keyword.

Steps:

1. First, use the ORDER BY clause to check how many columns are present in the SQL query.
For example, try using id=1 ORDER BY 1, then id=1 ORDER BY 2, and so on.
Keep increasing the number until you get an error. The last working number tells you how many columns exist.

2. After finding the number of columns (for example, 11), you can construct a payload like this:
id=-1 UNION SELECT 1,2,3,4,5,6,7,8,9,10,11
(We use -1 or any invalid ID to make sure no real data is shown from the original query.)

3. If the page loads successfully, it means the UNION SELECT is working. You can now try placing useful SQL functions in the columns to extract information.

Useful SQL functions you can use to extract information:

- database() – shows the name of the current database
- user() – shows the database user
- current_user – shows the current session user
- @@version – shows the database version
- @@datadir – shows the data directory
- @@tmpdir – shows the temporary directory
- system_user – shows the operating system user running the database

Example:
You can modify the payload to:
id=-1 UNION SELECT 1,2,database(),4,5,6,7,8,9,10,11
This will display the database name in the position where database() is placed (if that column's output is visible on the page).

TestCase4:

Boolean-Based SQL Injection:

Boolean-based SQL Injection is a method used to check whether a web application is vulnerable to SQL injection by injecting Boolean expressions using the OR operator.

To test for this, you can use payloads like:

OR 1=1 — This condition is always true.

OR 1=2 — This condition is always false.

If the application shows different behavior or responses between these two inputs (e.g., displaying data with OR 1=1 but not with OR 1=2), it indicates that the input is being directly used in the backend SQL query.

This difference in response confirms that the application is likely vulnerable to Boolean-based SQL Injection. The key idea is to observe changes in the application's behavior when switching between a true and false condition using the OR operator.

TestCase5:

Time-Based SQL Injection

Time-based SQL injection is a type of blind SQL injection where the attacker checks for vulnerabilities by observing the delay in the server's response.

To test for time-based SQL injection, we use a payload that causes a deliberate delay in the response. For example, in MySQL, we can use the following payload:

SLEEP(10)

If the page takes around 10 seconds to load after injecting this payload, it indicates a possible SQL injection vulnerability.

Example:

https://example.com/page?id=1' AND SLEEP(10)--

If the page pauses for about 10 seconds, it means the database executed the injected command.

For Oracle databases, the equivalent payload is:

dbms_lock.sleep(10)

For Microsoft SQL Server, you can use:

WAITFOR DELAY '00:00:10'

Each of these commands delays the server's response, and the delay confirms whether the SQL injection is working.