

# Security Assessment Report

Date of Assessment: 2024-09-09 13:41:58

## Vulnerabilities Found

Possible SQL Injection vulnerability at <https://tryhackme.com/>

Payload: ' OR " = '

Reason: Attempts to bypass authentication by injecting a condition that is always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' OR 1=1 --

Reason: Classic SQL injection payload that often returns all records from the database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: '; DROP TABLE users --

Reason: Payload that attempts to drop a database table.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' UNION SELECT password FROM users --

Reason: Payload that attempts to retrieve passwords from the database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: '='

Reason: This payload tries to trick the query into accepting the input as a valid condition.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 'LIKE'

Reason: Tests if the SQL query is vulnerable to a LIKE clause, often used in wildcards.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: =0--+

Reason: Attempts to bypass by terminating the query and adding a comment.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=1

Reason: Classic SQL injection that always returns true, bypassing any logical checks.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.

- Use parameterized queries to prevent injection attacks.

Payload: ' OR 'x'='x

Reason: Bypasses authentication by injecting a condition that is always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' AND id IS NULL; --

Reason: Attempts to exploit null conditions in the query logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: """"""""UNION SELECT '2

Reason: Uses excessive quotes to attempt to bypass input sanitization.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: %00

Reason: Null byte injection, used to terminate strings prematurely in some databases.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: /\*...\*/

Reason: This payload uses SQL comments to bypass restrictions or manipulate logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: +

Reason: Tests for SQL concatenation vulnerabilities, often used in UNION or SELECT queries.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ||

Reason: Checks if the database supports string concatenation via the double-pipe operator.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: %

Reason: Tests for wildcard characters that might bypass query logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: @variable

Reason: Attempts to exploit SQL variables to manipulate the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: @@variable

Reason: Tests for vulnerabilities related to server-level variables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1

Reason: Attempts to inject a true condition, testing for basic logical vulnerabilities.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 0

Reason: Tests if false logical conditions are handled properly.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND true

Reason: Tries to inject a true boolean condition to manipulate the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND false

Reason: Attempts to break the logic by injecting a false condition.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1-false

Reason: Tests for vulnerabilities by manipulating boolean values in the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1-true

Reason: Tests if the query allows manipulation of boolean values.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1\*56

Reason: Attempts to manipulate mathematical operations in the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: -2

Reason: Injects a negative number to test for vulnerabilities in numeric fields.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 1--+

Reason: Orders the results by the first column, which can reveal data structure.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 2--+

Reason: Orders the results by the second column, probing for more information.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 3--+

Reason: Continues to probe for available columns.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 1,2--+

Reason: Orders by multiple columns to test for vulnerabilities.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 1,2,3--+

Reason: Further tests column enumeration and query structure.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' GROUP BY 1,2,--+

Reason: Groups results by multiple columns to manipulate query logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' GROUP BY 1,2,3--+

Reason: Tests grouping vulnerabilities in the database query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' GROUP BY columnnames having 1=1 --

Reason: Attempts to exploit HAVING clauses for injection.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: -1' UNION SELECT 1,2,3--+

Reason: Union-based injection, attempting to select additional columns.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' UNION SELECT sum(columnname) from tablename --

Reason: Tests for arithmetic operations in the SQL query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: -1 UNION SELECT 1 INTO @,@

Reason: Attempts to insert results into user-defined variables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: -1 UNION SELECT 1 INTO @,@,@

Reason: Similar to the previous, but with three variables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1 AND (SELECT \* FROM Users) = 1

Reason: Injects a subquery to access sensitive data like user tables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' AND MID(VERSION(),1,1) = '5';

Reason: Probes for the version of the SQL database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' and 1 in (select min(name) from sysobjects where xtype = 'U' and name > '!') --

Reason: Attempts to query database metadata.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ,(select \* from (select(sleep(10))))a)

Reason: Tests for time-based SQL injection (delaying response).

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: %2c(select%20\*%20from%20(select(sleep(10))))a)

Reason: URL-encoded version of the sleep-based time delay injection.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ';WAITFOR DELAY '0:0:30'--

Reason: Time-delay attack to test if the query pauses for the specified time.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=1

Reason: Classic boolean-based injection, making the query always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=0

Reason: Tests the opposite scenario, making the query always false.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=x

Reason: Checks for identical comparisons to always return true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=y

Reason: Checks if non-identical comparisons will throw errors or vulnerabilities.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=1#

Reason: Comment-based bypass, ensuring the injected part is always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=0#

Reason: Similar to the previous but tests for false logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=x#

Reason: Tests comment-based injections with boolean true logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=y#

Reason: Tests false boolean logic in comment-based injection.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=1--

Reason: Tests injection by terminating the query and adding a comment.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=0--

Reason: Tests for false condition injection with query termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=x--

Reason: Checks if identical conditions in the injection work as expected.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.

- Use parameterized queries to prevent injection attacks.

Payload: OR x=y--

Reason: Tests non-identical conditions in comment-based injections.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 3409=3409 AND ('pytW' LIKE 'pytW')

Reason: Checks for a true condition using the LIKE operator.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 3409=3409 AND ('pytW' LIKE 'pytY')

Reason: Checks for a false condition using the LIKE operator.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=1

Reason: Injects into HAVING clauses to bypass group filtering.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=0

Reason: Injects into HAVING clauses with a false condition.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=1#

Reason: Tests for comment-based injection within the HAVING clause.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=0#

Reason: Tests false logic in comment-based HAVING injections.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=1--

Reason: Injects true conditions in HAVING clauses and terminates the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=0--

Reason: Injects false conditions in HAVING clauses and terminates the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1

Reason: Simple true condition to manipulate logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0

Reason: False condition to manipulate the logic flow.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1--

Reason: True condition injection with query termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0--

Reason: False condition injection with query termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1#

Reason: True condition injection with comment-based termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0#

Reason: False condition injection with comment-based termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1 AND '%'=

Reason: True condition injection using wildcards.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0 AND '%'=

Reason: False condition injection using wildcards.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1083=1083 AND (1427=1427

Reason: Tests for multiple true numeric conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7506=9091 AND (5913=5913

Reason: Tests false and true conditions together.

Remedies and Precautionary Measures:



- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1083=1083 AND ('1427=1427

Reason: Checks for vulnerabilities with string comparisons.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7506=9091 AND ('5913=5913

Reason: Tests for injection with a mix of false and true conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7300=7300 AND 'pKIZ'='pKIZ

Reason: Tests string comparisons for always true conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7300=7300 AND 'pKIZ'='pKIY

Reason: Tests string comparisons for always false conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AS INJECTX WHERE 1=1 AND 1=1

Reason: Tests true conditions in WHERE clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AS INJECTX WHERE 1=1 AND 1=0

Reason: Tests false conditions in WHERE clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: WHERE 1=1 AND 1=1--

Reason: Tests WHERE clause injection with termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: WHERE 1=1 AND 1=0--

Reason: Tests false logic in WHERE clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ORDER BY 1--

Reason: Orders by the first column, probing for SQL injection points.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ORDER BY 2--

Reason: Orders by the second column.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ORDER BY 31337#

Reason: Tests for large numbers in ORDER BY clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: RLIKE (SELECT (CASE WHEN (4346=4346) THEN 0x61646d696e ELSE 0x28 END))

Reason: Tests RLIKE condition for true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: IF(7423=7423) SELECT 7423 ELSE DROP FUNCTION xcjl--

Reason: Tests for conditional logic within SQL.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: '%' AND 8310=8310 AND '%'=

Reason: Tests for wildcard handling in SQL queries.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: and (select substring(@@version,1,1))='X'

Reason: Probes for the SQL version to understand the database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Possible SQL Injection vulnerability at <https://tryhackme.com/login>

Payload: ' OR " = '

Reason: Attempts to bypass authentication by injecting a condition that is always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' OR 1=1 --

Reason: Classic SQL injection payload that often returns all records from the database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: '; DROP TABLE users --

Reason: Payload that attempts to drop a database table.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' UNION SELECT password FROM users --

Reason: Payload that attempts to retrieve passwords from the database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: '='

Reason: This payload tries to trick the query into accepting the input as a valid condition.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 'LIKE'

Reason: Tests if the SQL query is vulnerable to a LIKE clause, often used in wildcards.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: =0--+

Reason: Attempts to bypass by terminating the query and adding a comment.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=1

Reason: Classic SQL injection that always returns true, bypassing any logical checks.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' OR 'x'='x

Reason: Bypasses authentication by injecting a condition that is always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' AND id IS NULL; --

Reason: Attempts to exploit null conditions in the query logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: """"""""UNION SELECT '2

Reason: Uses excessive quotes to attempt to bypass input sanitization.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: %00

Reason: Null byte injection, used to terminate strings prematurely in some databases.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: /\*...\*/

Reason: This payload uses SQL comments to bypass restrictions or manipulate logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: +

Reason: Tests for SQL concatenation vulnerabilities, often used in UNION or SELECT queries.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ||

Reason: Checks if the database supports string concatenation via the double-pipe operator.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: %

Reason: Tests for wildcard characters that might bypass query logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: @variable

Reason: Attempts to exploit SQL variables to manipulate the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: @@variable

Reason: Tests for vulnerabilities related to server-level variables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1

Reason: Attempts to inject a true condition, testing for basic logical vulnerabilities.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 0

Reason: Tests if false logical conditions are handled properly.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND true

Reason: Tries to inject a true boolean condition to manipulate the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND false

Reason: Attempts to break the logic by injecting a false condition.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.

- Use parameterized queries to prevent injection attacks.

Payload: 1=false

Reason: Tests for vulnerabilities by manipulating boolean values in the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1=true

Reason: Tests if the query allows manipulation of boolean values.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1\*56

Reason: Attempts to manipulate mathematical operations in the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: -2

Reason: Injects a negative number to test for vulnerabilities in numeric fields.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 1--+

Reason: Orders the results by the first column, which can reveal data structure.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 2--+

Reason: Orders the results by the second column, probing for more information.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 3--+

Reason: Continues to probe for available columns.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 1,2--+

Reason: Orders by multiple columns to test for vulnerabilities.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 1,2,3--+

Reason: Further tests column enumeration and query structure.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' GROUP BY 1,2,--+

Reason: Groups results by multiple columns to manipulate query logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' GROUP BY 1,2,3--+

Reason: Tests grouping vulnerabilities in the database query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' GROUP BY columnnames having 1=1 --

Reason: Attempts to exploit HAVING clauses for injection.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: -1' UNION SELECT 1,2,3--+

Reason: Union-based injection, attempting to select additional columns.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' UNION SELECT sum(columnname) from tablename --

Reason: Tests for arithmetic operations in the SQL query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: -1 UNION SELECT 1 INTO @,@

Reason: Attempts to insert results into user-defined variables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: -1 UNION SELECT 1 INTO @,@,@

Reason: Similar to the previous, but with three variables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1 AND (SELECT \* FROM Users) = 1

Reason: Injects a subquery to access sensitive data like user tables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' AND MID(VERSION(),1,1) = '5';

Reason: Probes for the version of the SQL database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' and 1 in (select min(name) from sysobjects where xtype = 'U' and name > '.') --

Reason: Attempts to query database metadata.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: `;(select * from (select(sleep(10))))a)`

Reason: Tests for time-based SQL injection (delaying response).

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: `%2c(select%20*%20from%20(select(sleep(10))))a)`

Reason: URL-encoded version of the sleep-based time delay injection.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: `';WAITFOR DELAY '0:0:30'--`

Reason: Time-delay attack to test if the query pauses for the specified time.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: `OR 1=1`

Reason: Classic boolean-based injection, making the query always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: `OR 1=0`

Reason: Tests the opposite scenario, making the query always false.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: `OR x=x`

Reason: Checks for identical comparisons to always return true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: `OR x=y`

Reason: Checks if non-identical comparisons will throw errors or vulnerabilities.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: `OR 1=1#`

Reason: Comment-based bypass, ensuring the injected part is always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: `OR 1=0#`

Reason: Similar to the previous but tests for false logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=x#

Reason: Tests comment-based injections with boolean true logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=y#

Reason: Tests false boolean logic in comment-based injection.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=1--

Reason: Tests injection by terminating the query and adding a comment.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=0--

Reason: Tests for false condition injection with query termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=x--

Reason: Checks if identical conditions in the injection work as expected.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=y--

Reason: Tests non-identical conditions in comment-based injections.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 3409=3409 AND ('pytW' LIKE 'pytW')

Reason: Checks for a true condition using the LIKE operator.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 3409=3409 AND ('pytW' LIKE 'pytY')

Reason: Checks for a false condition using the LIKE operator.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=1

Reason: Injects into HAVING clauses to bypass group filtering.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=0

Reason: Injects into HAVING clauses with a false condition.



Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=1#

Reason: Tests for comment-based injection within the HAVING clause.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=0#

Reason: Tests false logic in comment-based HAVING injections.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=1--

Reason: Injects true conditions in HAVING clauses and terminates the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=0--

Reason: Injects false conditions in HAVING clauses and terminates the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1

Reason: Simple true condition to manipulate logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0

Reason: False condition to manipulate the logic flow.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1--

Reason: True condition injection with query termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0--

Reason: False condition injection with query termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1#

Reason: True condition injection with comment-based termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.

- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0#

Reason: False condition injection with comment-based termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1 AND '%'=

Reason: True condition injection using wildcards.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0 AND '%'=

Reason: False condition injection using wildcards.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1083=1083 AND (1427=1427

Reason: Tests for multiple true numeric conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7506=9091 AND (5913=5913

Reason: Tests false and true conditions together.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1083=1083 AND ('1427=1427

Reason: Checks for vulnerabilities with string comparisons.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7506=9091 AND ('5913=5913

Reason: Tests for injection with a mix of false and true conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7300=7300 AND 'pKIZ'='pKIZ

Reason: Tests string comparisons for always true conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7300=7300 AND 'pKIZ'='pKIY

Reason: Tests string comparisons for always false conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AS INJECTX WHERE 1=1 AND 1=1

Reason: Tests true conditions in WHERE clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AS INJECTX WHERE 1=1 AND 1=0

Reason: Tests false conditions in WHERE clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: WHERE 1=1 AND 1=1--

Reason: Tests WHERE clause injection with termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: WHERE 1=1 AND 1=0--

Reason: Tests false logic in WHERE clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ORDER BY 1--

Reason: Orders by the first column, probing for SQL injection points.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ORDER BY 2--

Reason: Orders by the second column.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ORDER BY 31337#

Reason: Tests for large numbers in ORDER BY clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: RLIKE (SELECT (CASE WHEN (4346=4346) THEN 0x61646d696e ELSE 0x28 END))

Reason: Tests RLIKE condition for true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: IF(7423=7423) SELECT 7423 ELSE DROP FUNCTION xcjl--

Reason: Tests for conditional logic within SQL.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: '%' AND 8310=8310 AND '%'=

Reason: Tests for wildcard handling in SQL queries.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: and (select substring(@@version,1,1))='X'

Reason: Probes for the SQL version to understand the database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Possible SQL Injection vulnerability at <https://tryhackme.com/signup>

Payload: ' OR " = '

Reason: Attempts to bypass authentication by injecting a condition that is always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' OR 1=1 --

Reason: Classic SQL injection payload that often returns all records from the database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: '; DROP TABLE users --

Reason: Payload that attempts to drop a database table.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' UNION SELECT password FROM users --

Reason: Payload that attempts to retrieve passwords from the database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: '='

Reason: This payload tries to trick the query into accepting the input as a valid condition.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 'LIKE'

Reason: Tests if the SQL query is vulnerable to a LIKE clause, often used in wildcards.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: =0--+

Reason: Attempts to bypass by terminating the query and adding a comment.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=1

Reason: Classic SQL injection that always returns true, bypassing any logical checks.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' OR 'x'='x

Reason: Bypasses authentication by injecting a condition that is always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' AND id IS NULL; --

Reason: Attempts to exploit null conditions in the query logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: """"""""UNION SELECT '2

Reason: Uses excessive quotes to attempt to bypass input sanitization.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: %00

Reason: Null byte injection, used to terminate strings prematurely in some databases.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: /\*...\*/

Reason: This payload uses SQL comments to bypass restrictions or manipulate logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: +

Reason: Tests for SQL concatenation vulnerabilities, often used in UNION or SELECT queries.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ||

Reason: Checks if the database supports string concatenation via the double-pipe operator.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: %

Reason: Tests for wildcard characters that might bypass query logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: @variable

Reason: Attempts to exploit SQL variables to manipulate the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: @@variable

Reason: Tests for vulnerabilities related to server-level variables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1

Reason: Attempts to inject a true condition, testing for basic logical vulnerabilities.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 0

Reason: Tests if false logical conditions are handled properly.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND true

Reason: Tries to inject a true boolean condition to manipulate the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND false

Reason: Attempts to break the logic by injecting a false condition.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1-false

Reason: Tests for vulnerabilities by manipulating boolean values in the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1-true

Reason: Tests if the query allows manipulation of boolean values.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1\*56

Reason: Attempts to manipulate mathematical operations in the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: -2

Reason: Injects a negative number to test for vulnerabilities in numeric fields.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 1--+

Reason: Orders the results by the first column, which can reveal data structure.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 2---+

Reason: Orders the results by the second column, probing for more information.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 3---+

Reason: Continues to probe for available columns.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 1,2---+

Reason: Orders by multiple columns to test for vulnerabilities.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' ORDER BY 1,2,3---+

Reason: Further tests column enumeration and query structure.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' GROUP BY 1,2,---+

Reason: Groups results by multiple columns to manipulate query logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1' GROUP BY 1,2,3---+

Reason: Tests grouping vulnerabilities in the database query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' GROUP BY columnnames having 1=1 --

Reason: Attempts to exploit HAVING clauses for injection.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: -1' UNION SELECT 1,2,3---+

Reason: Union-based injection, attempting to select additional columns.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' UNION SELECT sum(columnname) from tablename --

Reason: Tests for arithmetic operations in the SQL query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.

- Use parameterized queries to prevent injection attacks.

Payload: -1 UNION SELECT 1 INTO @, @

Reason: Attempts to insert results into user-defined variables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: -1 UNION SELECT 1 INTO @, @, @

Reason: Similar to the previous, but with three variables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: 1 AND (SELECT \* FROM Users) = 1

Reason: Injects a subquery to access sensitive data like user tables.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' AND MID(VERSION(),1,1) = '5';

Reason: Probes for the version of the SQL database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ' and 1 in (select min(name) from sysobjects where xtype = 'U' and name > '.') --

Reason: Attempts to query database metadata.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ,(select \* from (select(sleep(10)))a)

Reason: Tests for time-based SQL injection (delaying response).

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: %2c(select%20\*%20from%20(select(sleep(10)))a)

Reason: URL-encoded version of the sleep-based time delay injection.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ';WAITFOR DELAY '0:0:30'--

Reason: Time-delay attack to test if the query pauses for the specified time.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=1

Reason: Classic boolean-based injection, making the query always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=0



Reason: Tests the opposite scenario, making the query always false.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=x

Reason: Checks for identical comparisons to always return true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=y

Reason: Checks if non-identical comparisons will throw errors or vulnerabilities.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=1#

Reason: Comment-based bypass, ensuring the injected part is always true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=0#

Reason: Similar to the previous but tests for false logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=x#

Reason: Tests comment-based injections with boolean true logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=y#

Reason: Tests false boolean logic in comment-based injection.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=1--

Reason: Tests injection by terminating the query and adding a comment.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 1=0--

Reason: Tests for false condition injection with query termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=x--

Reason: Checks if identical conditions in the injection work as expected.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR x=y--

Reason: Tests non-identical conditions in comment-based injections.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 3409=3409 AND ('pytW' LIKE 'pytW'

Reason: Checks for a true condition using the LIKE operator.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: OR 3409=3409 AND ('pytW' LIKE 'pytY'

Reason: Checks for a false condition using the LIKE operator.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=1

Reason: Injects into HAVING clauses to bypass group filtering.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=0

Reason: Injects into HAVING clauses with a false condition.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=1#

Reason: Tests for comment-based injection within the HAVING clause.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=0#

Reason: Tests false logic in comment-based HAVING injections.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=1--

Reason: Injects true conditions in HAVING clauses and terminates the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: HAVING 1=0--

Reason: Injects false conditions in HAVING clauses and terminates the query.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1

Reason: Simple true condition to manipulate logic.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0

Reason: False condition to manipulate the logic flow.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1--

Reason: True condition injection with query termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0--

Reason: False condition injection with query termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1#

Reason: True condition injection with comment-based termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0#

Reason: False condition injection with comment-based termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=1 AND '%'='

Reason: True condition injection using wildcards.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1=0 AND '%'='

Reason: False condition injection using wildcards.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1083=1083 AND (1427=1427

Reason: Tests for multiple true numeric conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7506=9091 AND (5913=5913

Reason: Tests false and true conditions together.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 1083=1083 AND ('1427=1427

Reason: Checks for vulnerabilities with string comparisons.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7506=9091 AND ('5913=5913

Reason: Tests for injection with a mix of false and true conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7300=7300 AND 'pKIZ'='pKIZ

Reason: Tests string comparisons for always true conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AND 7300=7300 AND 'pKIZ'='pKIY

Reason: Tests string comparisons for always false conditions.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AS INJECTX WHERE 1=1 AND 1=1

Reason: Tests true conditions in WHERE clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: AS INJECTX WHERE 1=1 AND 1=0

Reason: Tests false conditions in WHERE clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: WHERE 1=1 AND 1=1--

Reason: Tests WHERE clause injection with termination.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: WHERE 1=1 AND 1=0--

Reason: Tests false logic in WHERE clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ORDER BY 1--

Reason: Orders by the first column, probing for SQL injection points.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.

- Use parameterized queries to prevent injection attacks.

Payload: ORDER BY 2--

Reason: Orders by the second column.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: ORDER BY 31337#

Reason: Tests for large numbers in ORDER BY clauses.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: RLIKE (SELECT (CASE WHEN (4346=4346) THEN 0x61646d696e ELSE 0x28 END))

Reason: Tests RLIKE condition for true.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: IF(7423=7423) SELECT 7423 ELSE DROP FUNCTION xcj|--

Reason: Tests for conditional logic within SQL.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: '%' AND 8310=8310 AND '%'=

Reason: Tests for wildcard handling in SQL queries.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

Payload: and (select substring(@@version,1,1))='X'

Reason: Probes for the SQL version to understand the database.

Remedies and Precautionary Measures:

- Validate and sanitize input parameters in SQL queries.
- Use parameterized queries to prevent injection attacks.

## Main Domain Links

<https://tryhackme.com/leaderboards>

<https://tryhackme.com/glossary>

<https://tryhackme.com/work-at-thm>

<https://tryhackme.com/hackactivities>

<https://tryhackme.com/>

<https://tryhackme.com/cdn-cgi/l/email-protection#ef878a838380af9b9d96878e8c84828ac18c8082>

<https://tryhackme.com/network/throwback>

<https://tryhackme.com/legal/terms-of-use>

<https://tryhackme.com/subscriptions>

<https://tryhackme.com/develop-rooms>

<https://tryhackme.com/business>

<https://tryhackme.com/r/hackactivities/search>

<https://tryhackme.com/login>

<https://tryhackme.com/careers>

<https://tryhackme.com/resources/blog>  
<https://tryhackme.com/about>  
<https://tryhackme.com/resources/newsroom>  
<https://tryhackme.com/classrooms>  
<https://tryhackme.com/r/hackactivities/search?&page;=1&kind;=rooms■=newest>  
<https://tryhackme.com/r/hackactivities/practice>  
<https://tryhackme.com/games/koth>  
<https://tryhackme.com/r/login/sso>  
<https://tryhackme.com/forum>  
<https://tryhackme.com/r/pricing>  
<https://tryhackme.com/room/wreath>  
<https://tryhackme.com/signup>

## **Subdomain Links**

### **Subdomain: <http://www.tryhackme.com>**

<http://www.tryhackme.com/forum>  
<http://www.tryhackme.com/r/hackactivities/search>  
<http://www.tryhackme.com/business>  
<http://www.tryhackme.com/network/throwback>  
<http://www.tryhackme.com/r/login/sso>  
<http://www.tryhackme.com/hackactivities>  
<http://www.tryhackme.com/resources/blog>  
<http://www.tryhackme.com/login>  
<http://www.tryhackme.com/room/wreath>  
<http://www.tryhackme.com/careers>  
<http://www.tryhackme.com/work-at-thm>  
<http://www.tryhackme.com/r/hackactivities/search?&page;=1&kind;=rooms■=newest>  
<http://www.tryhackme.com/legal/terms-of-use>  
<http://www.tryhackme.com/develop-rooms>  
<http://www.tryhackme.com/about>  
<http://www.tryhackme.com/r/pricing>  
<http://www.tryhackme.com/classrooms>  
<http://www.tryhackme.com/r/hackactivities/practice>  
<http://www.tryhackme.com/leaderboards>  
<http://www.tryhackme.com/signup>  
<http://www.tryhackme.com/games/koth>  
<http://www.tryhackme.com/glossary>  
<http://www.tryhackme.com/cdn-cgi//email-protection#d1b9b4bdbdbe91a5a3a8b9b0b2babcb4ffb2bebc>  
<http://www.tryhackme.com/>  
<http://www.tryhackme.com/subscriptions>  
<http://www.tryhackme.com/resources/newsroom>  
<http://www.tryhackme.com>

### **Subdomain: <http://admin.tryhackme.com>**

<http://admin.tryhackme.com/new>  
[http://admin.tryhackme.com/t/contact\\_us/](http://admin.tryhackme.com/t/contact_us/)  
<http://admin.tryhackme.com/>  
<http://admin.tryhackme.com/t/terms>  
<http://admin.tryhackme.com/t/privacy>

### **Subdomain: <http://docs.tryhackme.com>**

<http://docs.tryhackme.com/en/>

<http://docs.tryhackme.com#main-content>

### **Subdomain: <http://store.tryhackme.com>**

<http://store.tryhackme.com/collections/all/products/3-million-users-short-sleeve-unisex-t-shirt>

<http://store.tryhackme.com/collections/all/products/short-sleeve-unisex-t-shirt-5>

<http://store.tryhackme.com>

<http://store.tryhackme.com/>

<http://store.tryhackme.com/collections/all>

<http://store.tryhackme.com/collections/all/products/short-sleeve-unisex-t-shirt-3>

<http://store.tryhackme.com/collections/all/products/unisex-hoodie>

<http://store.tryhackme.com/collections/all/products/baseball-cap>

<http://store.tryhackme.com/collections/all/products/unisex-sweatshirt>

<http://store.tryhackme.com/collections/all/products/short-sleeve-unisex-t-shirt-1>

<http://store.tryhackme.com/collections/all/products/white-tryhackme-sticker>

<http://store.tryhackme.com/collections/all/products/stainless-steel-water-bottle>

<http://store.tryhackme.com#MainContent>

<http://store.tryhackme.com/collections/all/products/short-sleeve-unisex-t-shirt>

<http://store.tryhackme.com/cart>

<http://store.tryhackme.com/collections/all/products/gray-tryhackme-sticker-1>

### **Subdomain: <http://help.tryhackme.com>**

<https://help.tryhackme.com/en/collections/4062562-king-of-the-hill>

<https://help.tryhackme.com/en/collections/3601905-getting-started-with-tryhackme>

<https://help.tryhackme.com/en/collections/3601907-tryhackme-for-organisations>

<http://help.tryhackme.com#main-content>

<https://help.tryhackme.com/en/collections/3665114-tryhackme-for-users>

<http://help.tryhackme.com/en/>

### **Subdomain: <http://auth.tryhackme.com>**

### **Subdomain: <http://business.tryhackme.com>**

<https://business.tryhackme.com/solutions/career-builder>

<https://business.tryhackme.com/solutions/career-switcher?ref=navbar>

<https://business.tryhackme.com/use-cases/career-launcher?ref=navbar>

<https://business.tryhackme.com?ref=navbar>

<http://business.tryhackme.com>

<https://business.tryhackme.com/book-a-demo>

<https://business.tryhackme.com/industries/financial-services?ref=navbar>

<https://business.tryhackme.com/features>

<https://business.tryhackme.com/industries/mssp?ref=navbar>

<https://business.tryhackme.com/teams/defensive?ref=navbar>

<http://business.tryhackme.com/>

<https://business.tryhackme.com/solutions/career-builder?ref=navbar>

<https://business.tryhackme.com/teams/cloud?ref=navbar>

<https://business.tryhackme.com/teams/offensive?ref=navbar>

<http://business.tryhackme.com#chat-with-sales-2>

<https://business.tryhackme.com/teams/devsecops?ref=navbar>

<https://business.tryhackme.com/industries/government?ref=navbar>

<https://business.tryhackme.com/free-trial>