# SQL injection

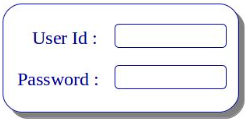
What is SQL injection (SQLi)?

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure, or perform a denial-of-service attack.

# Introduction to SQL Injection

For Example, in user login screen, username and password are the dynamic fields where users enter the data. Depending upon the user’s inputs dynamic queries will be constructed; the usual query will be



Select \* from users table where username=’Username.txt’ and password=’Password.txt’.

If the input fields are not sanitized properly, then the malicious user can enter some data like this

Username = blah’ or 1=1—

Password = password

Here both username and password are incorrect. But the query which is constructed will be

Select \* from users where username=’blah’ or 1=1—and password=’password’

The query will run and the user will be granted access. This is because the first part of the query is

Select \* from users where username=’blah’ or 1=1—

Because – is a comment line in SQL, everything following that will be ignored. The query will only validate between username=’blah’ or 1=1.

Because 1=1 is always true, the user will be granted access.

## What is the impact of a successful SQL injection attack?

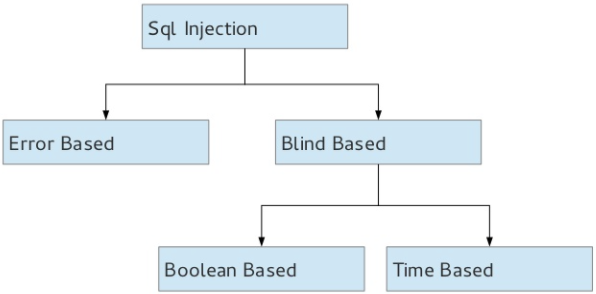
A successful SQL injection attack can result in unauthorized access to sensitive data, such as passwords, credit card details, or personal user information. Many high-profile data breaches in recent years have been the result of SQL injection attacks, leading to reputational damage and regulatory fines. In some cases, an attacker can obtain a persistent backdoor into an organization's systems, leading to a long-term compromise that can go unnoticed for an extended period.

SQL injection examples

There are a wide variety of SQL injection vulnerabilities, attacks, and techniques, which arise in different situations. Some common SQL injection examples include:

* [Retrieving hidden data](https://portswigger.net/web-security/sql-injection#retrieving-hidden-data), where you can modify an SQL query to return additional results.
* [Subverting application logic](https://portswigger.net/web-security/sql-injection#subverting-application-logic), where you can change a query to interfere with the application's logic.
* [UNION attacks](https://portswigger.net/web-security/sql-injection/union-attacks), where you can retrieve data from different database tables.
* [Examining the database](https://portswigger.net/web-security/sql-injection/examining-the-database), where you can extract information about the version and structure of the database.
* [Blind SQL injection](https://portswigger.net/web-security/sql-injection/blind), where the results of a query you control are not returned in the application's responses.

# Types of SQL Injection



## Error based Injection:

The attacker sends some malicious query to the database which results in errors.  The errors should be very generic, otherwise, they may give useful hints to the attacker.

**Comment-Line:** Using comment line to cause the database to ignore a part of a valid query.

E.g. Select \* from stores where product\_id = blah’ or 1=1-- (everything after this will be neglected)

**Tautology:** There are a lot of strings which always evaluates to be true, like ‘1’ = ‘1’ ‘a’ = ‘a’, etc., using them in the query to create constantly true conditions.

E.g. Select \* from users where username=’blah’ or ‘a’=’a’ -- and password=’pass’

## Union Based SQL injection:

Using union command in SQL query to execute additional queries; thereby, modifying/inserting/deleting or dropping the contents of the table.

E.g. Select \* from stores where product\_id=1 union select 1,database(),user(),4#

**Stored procedures:** Creating malicious inputs to execute malicious queries.

**Incorrect queries:** Coming up with logically incorrect queries to see the error messages to get more information about the target database.

Select \* from stores where id=1’

The above query will result in a syntax error and might reveal the backend database type.

## Blind SQL injection:

This is a type of SQL injection where we don’t have a clue as to whether the web application is vulnerable to injection attack or not.

### Types:

**Boolean:** Only correct queries show the result, wrong queries do not return anything. Attackers should try to generate logically correct queries.

If suppose the original query to the database is

Select \* from users where id=’id.txt’

If we give blah’ and 1=1# as input which evaluates to be a right query

Select \* from users where id=’blah’ or 1=1#, we will see the user results.

If we give blah’ and 1=2# as input which is a wrong query then we don’t see any results.

Select \* from users where id=’blah’ or 1=2#

**Time delay:**  Depending on some conditions, setting a time delay. If that condition is satisfied, we can observe the time delay; thereby, concluding that the input we gave produced a positive result. This is a time consuming process.

## Tools:

SQLMAP, Marathon tool.