**SQL injection** is an application security weakness that allows attackers to control an application’s database – letting them access or delete data, change an application’s data-driven behavior, and do other undesirable things – by tricking the application into sending unexpected SQL commands. SQL injections are among the most frequent threats to data security. SQL injection weaknesses occur when an application uses untrusted data, such as data entered into web form fields, as part of a database query. When an application fails to properly sanitize this untrusted data before adding it to a SQL query, an attacker can include their own SQL commands which the database will execute. Such SQLi vulnerabilities are easy to prevent, yet SQLi remains a leading web application risk, and many organizations remain vulnerable to potentially damaging data breaches resulting from SQL injection.

Attackers provide specially-crafted input to trick an application into modifying the SQL queries that the application asks the database to execute. This allows the attacker to:

* Control application behavior that’s based on data in the database, for example by tricking an application into allowing a login without a valid password.
* Alter data in the database without authorization, for example by creating fraudulent records, adding users or “promoting” users to higher access levels, or deleting data.
* Access data without authorization, for example by tricking the database into providing too many results for a query.

A developer defines a SQL query to perform some database action necessary for their application to function. This query has an *argument* so that only desired records are returned, and the value for that argument can be provided by a user (for example, through a form field, URL parameter, web cookie, etc.).

A SQLi attack plays out in two stages:

1. Research- Attacker tries submitting various unexpected values for the argument, observes how the application responds, and determines an attack to attempt.
2. Attack- Attacker provides a carefully-crafted input value that, when used as an argument to a SQL query, will be interpreted as part of a SQL command rather than merely data; the database then executes the SQL command as modified by the attacker.

There are easy ways to avoid introducing SQLi vulnerabilities in an application, and to limit the damage they can cause.

* Discover SQLi vulnerabilities by routinely testing your applications both using static testing and dynamic testing.
* Avoid and repair SQLi vulnerabilities by using parameterized queries. These types of queries specify placeholders for parameters so that the database will always treat them as data rather than part of a SQL command. Prepared statements and object relational mappers (ORMs) make this easy for developers.
* RemediateSQLi vulnerabilities in legacy systems by escaping inputs before adding them to the query. Use this technique only where prepared statements or similar facilities are unavailable.
* Mitigate the impact of SQLi vulnerabilities by enforcing least privilege on the database. Ensure that each application has its own database credentials, and that these credentials have the minimum rights the application needs.