What is SQL injection?

SQL injection is a vulnerability where the database sees attacker-supplied text as part of the query the application is trying to run. The fix has been understood for decades: parameterize the query so the database knows which parts are code and which parts are data. The reason it keeps appearing in production is that any single forgotten string-concatenation is enough to reopen the door.

HOW IT WORKS

01 What do attackers actually do with SQL injection?

Five outcomes show up most often in real engagements:

- Read everything. Dump the user table, the payments table, the admin sessions. The classic data breach.
- Bypass authentication. Log in as any user, including the admin, without knowing the password.
- Modify data. Change a balance, mark an order shipped, grant themselves a role.
- Pivot to the operating system. Some databases allow file read, file write, or command execution from inside a query.
- Stay hidden. Add a backdoor user, modify audit logs, install a persistent web shell.

02 Modern frameworks fixed this, right?

Mostly. Mistakes still happen.

- Raw SQL escape hatches. Most ORMs offer a way to run raw SQL when the query gets complex. Some teams reach for the escape hatch and concatenate strings inside it.
- Dynamic ORM filter construction. Frameworks like SQLAlchemy or Hibernate let you build dynamic filter conditions. Done carelessly, this turns into string concatenation against the parser.
- Stored procedure injection. A stored procedure that uses `EXEC` against a parameter is still vulnerable.

HOW TO DEFEND

- Use the parameterization features of your data access layer everywhere. Every modern language has them.
- Use an ORM (object-relational mapper) consistently. ORMs default to parameterization, but you still need to audit raw-query escape hatches.
- Apply least-privilege to the database user the application connects as. If the application only needs read access to most tables, the user should not have write or DROP rights.
- Validate input shape (an email looks like an email, an ID is a number) as a second layer, not the only layer.
- Run automated and manual testing for SQL injection on a recurring schedule. Code review alone misses one-off mistakes that reopen the door.

SOURCES

[1] OWASP A03:2021 Injection

[2] OWASP SQL Injection Prevention Cheat Sheet

[3] CWE-89: SQL Injection

- Search and reporting endpoints. When the team writes a custom query builder for an admin-only search page, the admin-only assumption often turns out to be wrong.
- NoSQL injection. Document stores like MongoDB have their own injection class where attacker-supplied JSON operators (`{$ne: null}`) bypass authentication logic.

03 How does SecureLayer7 test for SQL injection?

Every application engagement runs both automated and manual coverage.

- Automated layer. Tools fuzz each input parameter for common payload shapes (boolean-based, error-based, time-based, union-based). This catches the easy cases fast and gives a baseline.
- Manual layer. The pentester focuses on endpoints that the automated tools cannot reach: authenticated flows, multi-step forms, parameters that only appear after a specific user action, search builders, custom export jobs. This is where real findings hide.
- Impact proof. Every finding ships with a reproducible request, the database response, and an estimate of the realistic blast radius (what data could be read, what writes are possible, what privilege escalation it unlocks).

Deliverable maps findings to OWASP A03:2021 (Injection) and includes the specific code change required.

Test your application for SQL injection and 30+ other classes.
securelayer7.net/learn/application-security/sql-injection

Open online