CS-058 · Application Security

SQL Injection

An attack where untrusted input is interpreted as part of a database query.

Executive Summary

SQL injection is a class of attack in which untrusted input is treated as part of a database query rather than as plain data. When an application builds queries by concatenating user input, an attacker can change the meaning of the query to read, alter, or delete data. It is one of the oldest and most damaging web application flaws, and it is almost entirely preventable with parameterized queries.

What It Is

SQL injection occurs when an application sends user-supplied input to a database in a way that lets the input change the structure of the query, not just its values. The root cause is mixing code and data: when input is concatenated directly into a query string, the database cannot tell where the developer's instructions end and the attacker's input begins. Depending on the application, an attacker may be able to bypass a login, read data belonging to other users, modify or delete records, or in some cases run commands on the database server. Variations include blind injection, where results are inferred indirectly, and injection through any input path the application trusts, not only obvious form fields.

Why It Matters

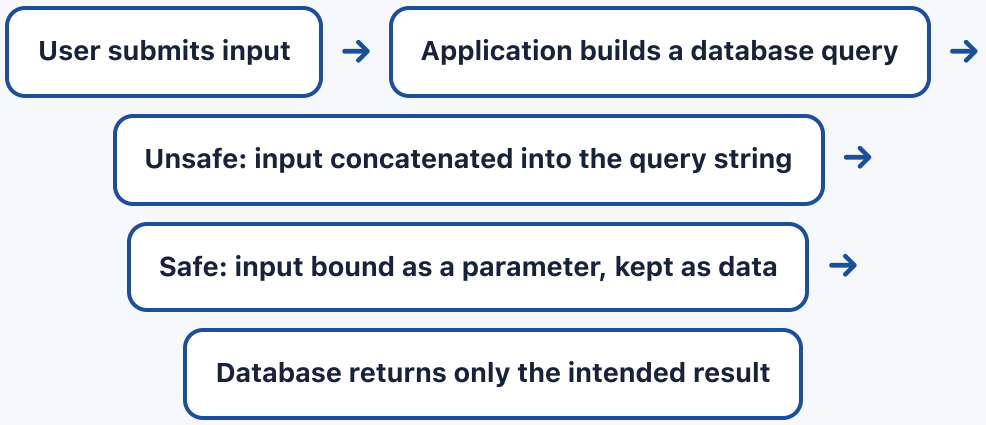
Databases usually hold the most sensitive assets an organization has, including customer records, credentials, and financial data. A single SQL injection flaw can expose an entire database, which makes it a favorite target for attackers and a frequent cause of large breaches. Beyond data theft, injection can corrupt records or take a service offline. Because the flaw is well understood and highly preventable, an SQL injection in production is often read by auditors

and customers as a sign of weak development practices. For professionals, recognizing and eliminating it is a baseline expectation in application security.

How It Works

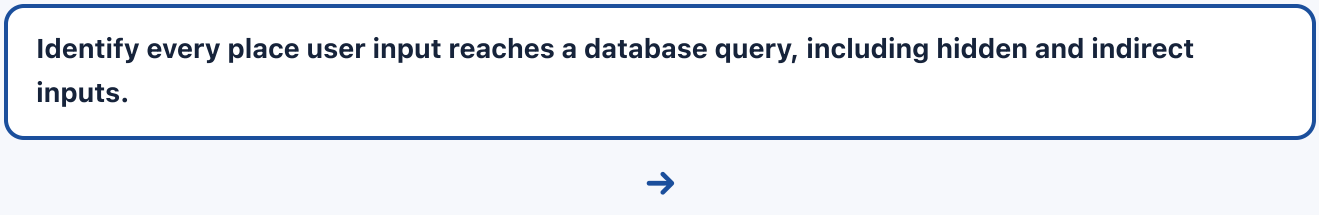
At a conceptual level, the problem is that the query and the data travel together. When an application builds a query by pasting input directly into a string, input that contains query syntax can be interpreted as instructions. The defense reverses this by keeping code and data strictly separate. With parameterized queries, also called prepared statements, the developer writes the query with placeholders and sends the user input separately as values, so the database always treats it as data and never as executable syntax. Additional layers help: validating input against an expected format, applying least privilege so the database account can do only what the application needs, and avoiding detailed database errors that reveal structure to an attacker.

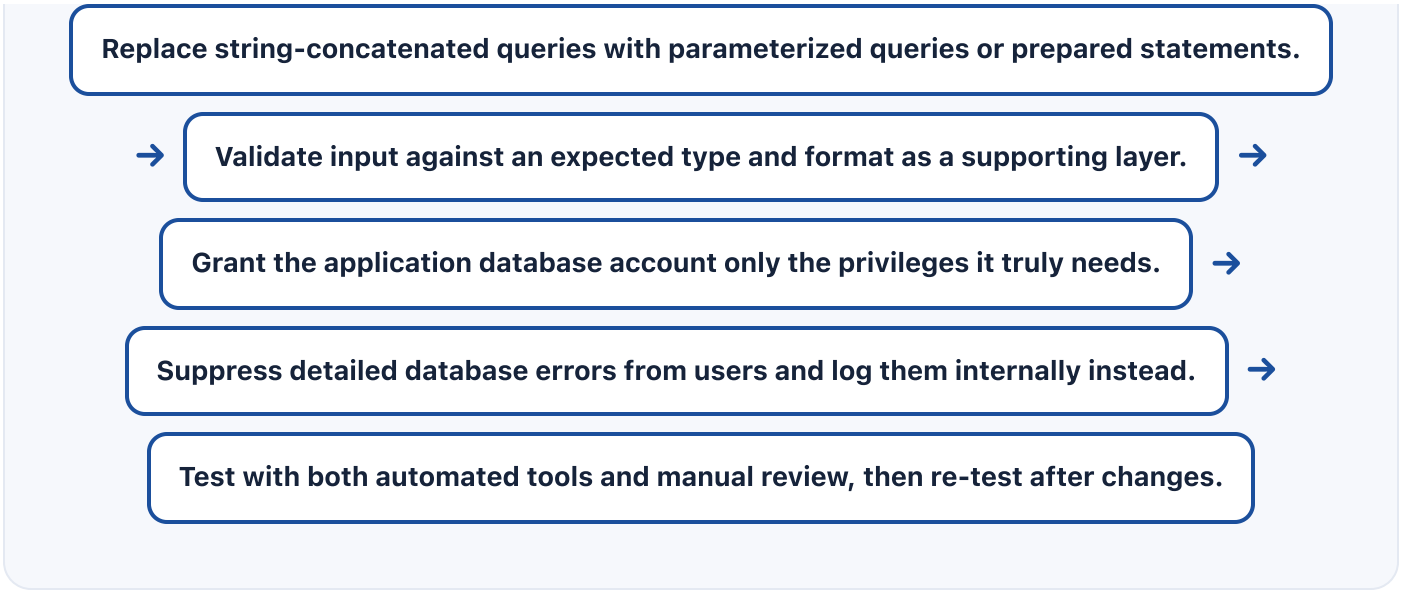
Architecture Diagram



SQL injection happens when input mixes into query code; parameterized queries keep the two separate so input stays data.

Visual Workflow





Common Attacks

- Concatenating user input directly into a query so input becomes query syntax
- Bypassing authentication by manipulating a login query's logic
- Reading data from other tables or other users through a manipulated query
- Blind injection where results are inferred from application behavior or timing
- Injection through non-obvious inputs such as headers, cookies, or API fields

Common Mistakes

- Building queries by concatenating strings instead of using parameters
- Relying on input filtering or blocklists as the primary defense
- Trusting inputs that are not visible form fields, such as headers or JSON
- Running the application with an over-privileged database account
- Returning raw database error messages that reveal schema details

Best Practices

- Use parameterized queries or prepared statements everywhere as the primary defense
- Prefer well-reviewed data access libraries that parameterize by default
- Validate input against an allowlist of expected formats as a supporting layer
- Apply least privilege to the database account the application uses

- Return generic error messages to users and keep detailed errors in logs
- Include injection checks in automated testing and code review

Quick Checklist

- [x] All database access uses parameterized queries or an equivalent safe layer
- [x] No user input is concatenated into query strings anywhere
- [x] Input validated against expected type and format
- [x] Application database account limited to least privilege
- [x] Detailed database errors hidden from users
- [x] Automated and manual injection testing part of the release process

Recommended Tools

Static application security testing (SAST)
Flags string-built queries and unsafe input handling in source code

Dynamic application security testing (DAST)
Probes a running application for injectable inputs

Parameterized query library or ORM
Provides safe query construction that separates code from data

Web application firewall (WAF)
Acts as a compensating control to filter obvious injection attempts

Industry Standards

OWASP Top 10
Injection is a longstanding category of critical web application risk

OWASP Application Security Verification Standard (ASVS)
Testable requirements for safe data access and query construction

CWE-89
The Common Weakness Enumeration

entry for SQL injection

Career Relevance

SQL injection is core knowledge for application security engineers, penetration testers, and secure code reviewers, who look for it in nearly every web assessment. Developers are expected to prevent it by default through safe query practices, and GRC and audit professionals treat its presence as a meaningful risk signal. For the AI-Governance-Jobs.com audience, it is one of the most common technical interview topics in appsec.

Interview Questions

- In plain terms, what causes SQL injection?
- Why are parameterized queries the primary defense rather than input filtering?
- What is blind SQL injection, and how does it differ from a direct one?
- How does least privilege on the database account reduce the impact of injection?
- Where besides visible form fields might injectable input enter an application?

Related Certifications

GIAC Web Application Penetration Tester (GWAPT)

Offensive Security Web Assessor (OSWA)

CompTIA Security+

Further Reading

- OWASP: SQL Injection
- CWE-89: SQL Injection
- OWASP Top 10

Key Takeaways

- SQL injection lets untrusted input change the meaning of a database query.
- The root cause is mixing code and data in query construction.
- Parameterized queries are the primary and highly effective defense.

- Least privilege and input validation add defense in depth.
- It is well understood and largely preventable, so its presence is a red flag.

FAQ

Related Careers

RELATED ROLES

Application Security Engineer

Penetration Tester

Secure Code Reviewer

Security Engineer

RELATED CERTIFICATIONS

GIAC Web Application Penetration Tester (GWAPT)

Offensive Security Web Assessor (OSWA)

CompTIA Security+

CURRENT OPENINGS

Live roles are on the job board.

Browse all jobs

GUIDES & SALARY

Career guides

Role roadmaps

Salary data

Career tools

SUGGESTED LEARNING PATH

1. Ground the basics with CS-001 Cybersecurity
2. Study this sheet: **SQL Injection**
3. Go deeper: OWASP Top 10

4. Go deeper: Cross-Site Scripting (XSS)
5. Validate it: work toward **GIAC Web Application Penetration Tester (GWAPT)**
6. Find the role: browse current openings

RELATED SHEETS

CS-057 OWASP Top 10 | CS-059 Cross-Site Scripting (XSS) | CS-061 Server-Side Request Forgery (SSRF) | CS-065 Secure Coding | CS-001 Cybersecurity