# How to Protect Against SQL Injection Attacks

## What is SQL Injection?

SQL injection is one of the most common web attack mechanisms utilized by attackers to steal sensitive data from organizations. While SQL Injection can affect any data-driven application that uses a SQL database, **it is most often used to attack web sites**.

SQL Injection is a code injection technique that hackers can use to insert malicious SQL statements into input fields for execution by the underlying SQL database. This technique is made possible because of improper coding of vulnerable web applications.

These flaws arise because entry fields made available for user input unexpectedly allow SQL statements to go through and query the database directly.

A few examples of SQL Injection are available for review at the following resources:

- OWASP Testing for SQL Injection (https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005))
- ProgrammerInterview.com SQL Injection Example (http://www.programmerinterview.com/index.php/database-sql/sql-injection-example/)

## Why is SQL Injection such a security risk for the campus?

Attackers are constantly probing the Internet at-large and campus web sites for SQL injection vulnerabilities. They use tools that automate the discovery of SQL injection flaws, and attempt to exploit SQL injection primarily for financial gain (e.g. stealing personally identifiable information which is then used for identity theft).

Because so many modern applications are data-driven and accessible via the web, SQL Injection vulnerabilities are widespread and easily exploited. Additionally, because of the prevalence of shared database infrastructure, a SQL Injection flaw in one application can lead to the compromise of other applications sharing the same database instance.

Once exploited, SQL Injection attacks can lead to:

- Theft, modification, or even destruction of sensitive data such as personally identifiable information and usernames and passwords
- Elevation of privileges at the application, database, or even operating system level
- Attackers "pivoting" by using a compromised database server to attack to other systems on the same network

# How to protect a web site or application from SQL Injection attacks

Developers can prevent SQL Injection vulnerabilities in web applications by utilizing **parameterized database queries with bound, typed parameters and careful use of parameterized stored procedures in the database**.

This can be accomplished in a variety of programming languages including Java, .NET, PHP, and more.

Please consult the following resources for implementing parameterized database queries and preventing SQL Injection in your code base:

- OWASP: SQL Injection (https://www.owasp.org/index.php/SQL_Injection)
- OWASP SQL Injection Prevention Cheat Sheet (https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html) (https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)
- OWASP Query Parameterization Cheat Sheet (https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html)

Additionally, developers, system administrators, and database administrators can take further steps to minimize attacks or the impact of successful attacks:

1. Keep all web application software components including libraries, plug-ins, frameworks, web server software, and database server software up to date with the latest security patches available from vendors.
2. Utilize the principle of least privilege (https://en.wikipedia.org/wiki/Principle_of_least_privilege) when provisioning accounts used to connect to the SQL database. For example, if a web site only needs to retrieve web content from a database using SELECT statements, do not give the web site's database connection credentials other privileges such as INSERT, UPDATE, or DELETE privileges. In many cases, these privileges can be managed using appropriate database roles for accounts. Never allow your web application to connect to the database with Administrator privileges (the "sa" account on Microsoft SQL Server, for instance).
3. Do not use shared database accounts between different web sites or applications.
4. Validate user-supplied input for expected data types, including input fields like drop-down menus or radio buttons, not just fields that allow users to type in input.
5. Configure proper error reporting and handling on the web server and in the code so that database error messages are never sent to the client web browser. Attackers can leverage technical details in verbose error messages to adjust their queries for successful exploitation.

# Is input filtering enough to stop SQL Injection?

A common misconception is that input filtering and escaping can prevent SQL Injection. While input filtering can help stop the most trivial of attacks, **it does not fix the underlying vulnerability**.

In many cases, input filtering can be evaded by attackers leaving your web application vulnerable despite attempts to, for example, deny-list certain characters on a web form.

# SQL Injection vulnerability email notification from security@berkeley.edu (mailto:security@berkeley.edu)

Security Contacts that receive a SQL Injection vulnerability notice are responsible for identifying and notifying any stakeholders about the SQL Injection attack including functional owners, developers, system administrators, and database administrators in order to determine the vulnerable and potentially compromised resources.

Immediate action must be taken to address any confirmed SQL Injection flaws discovered:

- Once a person responsible for coordinating remediation is identified, please respond to the notice so that Information Security and Policy can work directly with the coordinator to ensure full remediation
- Coordinate an investigation of potentially vulnerable web pages and resources amongst developers or other stakeholders
- A review of web, application, and database logs may reveal the point of vulnerability and source of attacks
- Develop a plan to remediate any confirmed SQL Injection flaws and prevent future attacks

**Additionally, if your system stores, processes, or transmits sensitive data such as UC P2/3 (formerly UCB PL1) or UC P4 (formerly UCB PL2) data as described in the Berkeley Data Classification Standard (https://security.berkeley.edu/data-classification-standard#plclassification), you should immediately reply to the security notice (to security@berkeley.edu (mailto:security@berkeley.edu)) and notify the Information Security Office.**



Back to Top