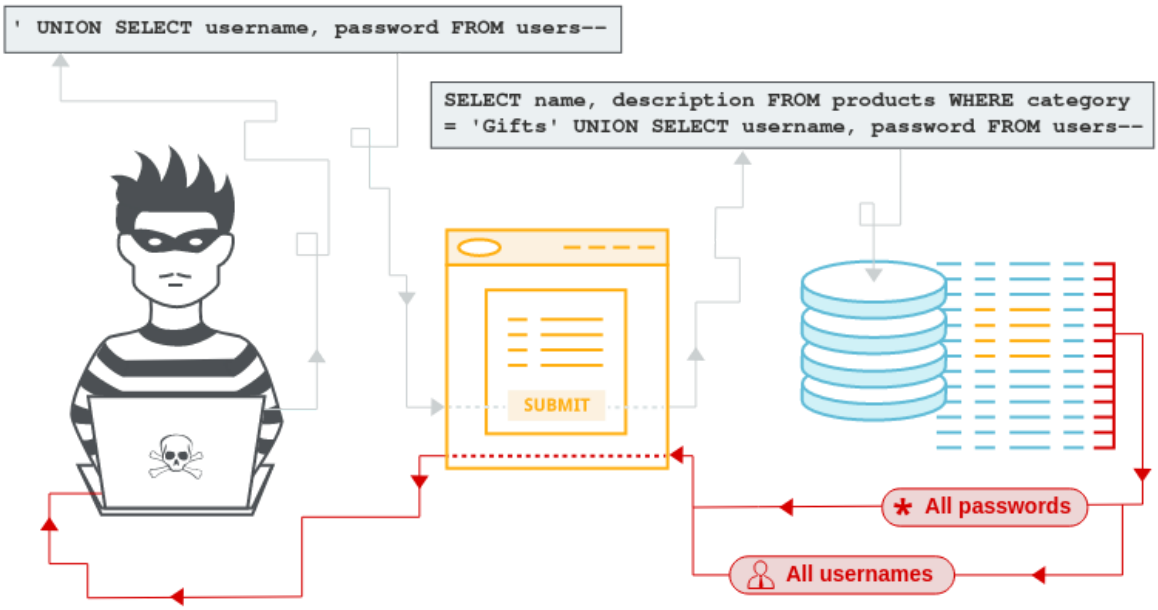
SQL injection

In this section, we'll explain what SQL injection (SQLi) is, describe some common examples, explain how to find and exploit various kinds of SQL injection vulnerabilities, and summarize how to prevent SQL injection.



Labs

If you're already familiar with the basic concepts behind SQLi vulnerabilities and just want to practice exploiting them on some realistic, deliberately vulnerable targets, you can access all of the labs in this topic from the link below.

View all SQL injection labs

What is SQL injection (SQLi)?

SQL injection (SQLi) 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.

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, where you can modify an SQL query to return additional results.
- Subverting application logic, where you can change a query to interfere with the application's logic.
- UNION attacks, where you can retrieve data from different database tables.
- Examining the database, where you can extract information about the version and structure of the database.
- Blind SQL injection, where the results of a query you control are not returned in the application's responses.

Retrieving hidden data

SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

Consider a shopping application that displays products in different categories. When the user clicks on the Gifts category, their browser requests the URL:

https://insecure-website.com/products?category=Gifts

This causes the application to make an SQL query to retrieve details of the relevant products from the database:

SELECT * FROM products WHERE category = 'Gifts' AND released = 1

This SQL query asks the database to return:

- all details (*)
- from the products table
- where the category is Gifts
- and released is 1.

The restriction released = 1 is being used to hide products that are not released. For unreleased products, presumably released = 0.

The application doesn't implement any defenses against SQL injection attacks, so an attacker can construct an attack like:

https://insecure-website.com/products?category=Gifts'--

This results in the SQL query:

SELECT * FROM products WHERE category = 'Gifts'--' AND released = 1

The key thing here is that the double-dash sequence -- is a comment indicator in SQL, and means that the rest of the query is interpreted as a comment. This effectively removes the remainder of the query, so it no longer includes AND released = 1. This means that all products are displayed, including unreleased products.

Going further, an attacker can cause the application to display all the products in any category, including categories that they don't know about:

https://insecure-website.com/products?category=Gifts'+OR+1=1--

This results in the SQL query:

SELECT * FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1

The modified query will return all items where either the category is Gifts, or 1 is equal to 1. Since 1=1 is always true, the query will return all items.

SOLUTION:

1. Use Burp Suite to intercept and modify the request that sets the product category filter.
2. Modify the category parameter, giving it the value '+OR+1=1--
3. Submit the request, and verify that the response now contains additional items.

https://0a5c00a0040cb1abc06e0f6400f10092.web-security-academy.net/filter?category=Gifts%27+OR+1=1--