

SQL Injection – Lab Report

Submitted By:

Name: Arsalan Khan

Position/Role: Internee

Date: August 8, 2025

Platform:

PortSwigger Labs

Objective:

Understand and exploit SQL injection vulnerabilities to extract sensitive data from databases, bypass authentication, and manipulate backend SQL queries. Learn different types of SQL injection attacks and defenses.

Tools Used:

- Burp Suite
- PortSwigger Lab Environment

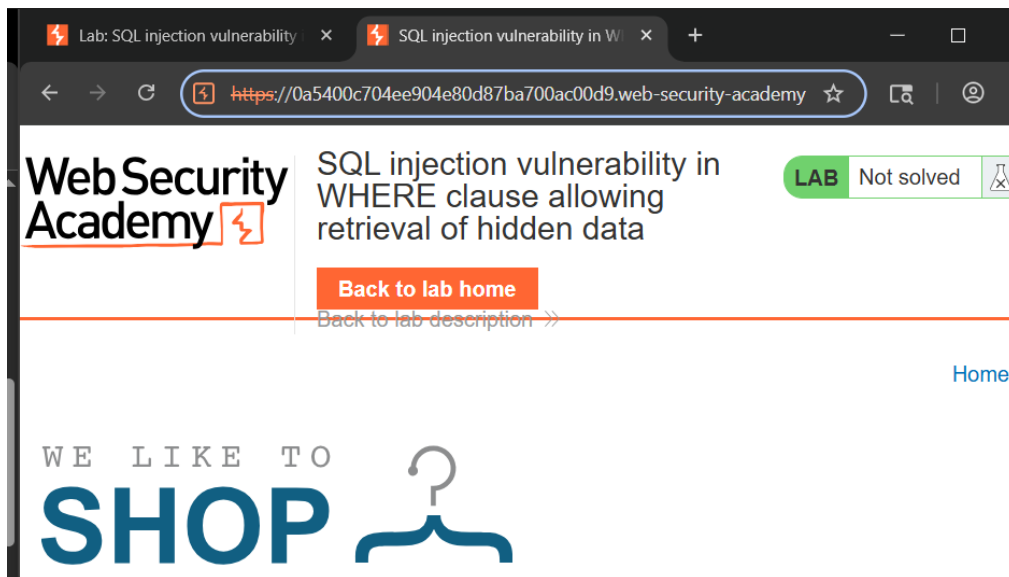
Lab 1: SQL Injection Vulnerability in WHERE Clause Allowing Retrieval of Hidden Data

Description:

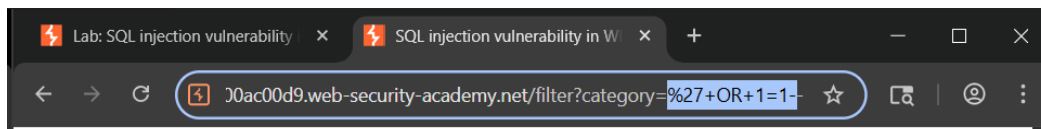
This lab demonstrates an error-based SQL injection vulnerability in the product category filter. By injecting SQL syntax into the category parameter, it's possible to manipulate the WHERE clause and retrieve hidden data (unreleased products) from the database.

Steps Performed:

1. Accessed the lab and located the product category filter input.

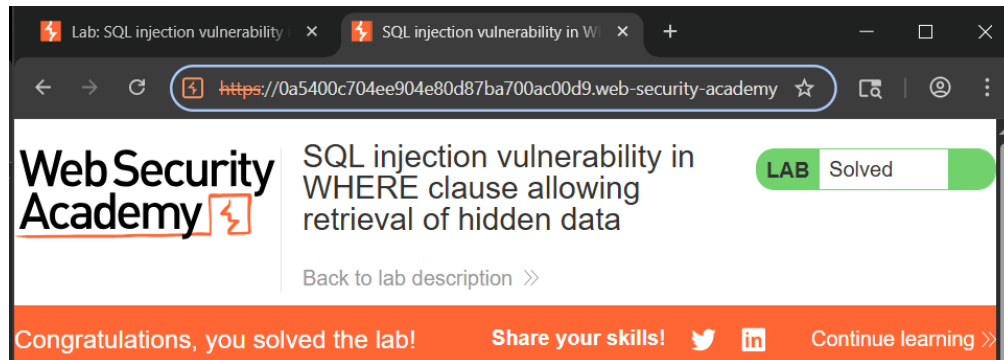


2. Modified the category parameter in the intercepted request to the payload:



3. Forwarded the modified request to the server.
4. Observed that the response contained additional products, including unreleased ones that were previously hidden.

5. Verified that the injection was successful and that hidden data was retrieved, solving the lab.



Vulnerability:

Error-Based SQL Injection in WHERE Clause

The application concatenates user input directly into SQL queries without sanitization or parameterization. Injecting SQL logic (`OR 1=1--`) bypasses the filter conditions and exposes hidden data in the query result.

Mitigation:

- Use parameterized queries/prepared statements to separate code from data.
- Validate and sanitize all user inputs rigorously.
- Avoid exposing detailed database errors to users.
- Employ proper database permissions and limit data exposure.

Lab 2: SQL Injection Vulnerability Allowing Login Bypass

Description:

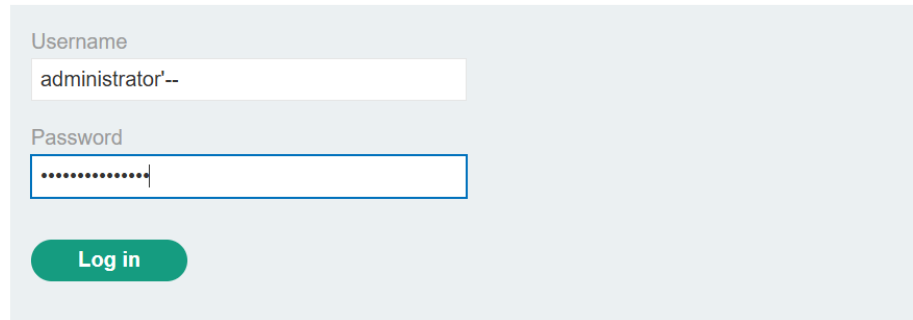
This lab contains a classic SQL injection vulnerability in the login function. By injecting SQL syntax into the username parameter, the authentication logic can be bypassed, allowing login as the administrator without knowing the password.

Steps Performed:

1. Accessed the lab login page.

2. Modified the username parameter in the intercepted request to:

Login



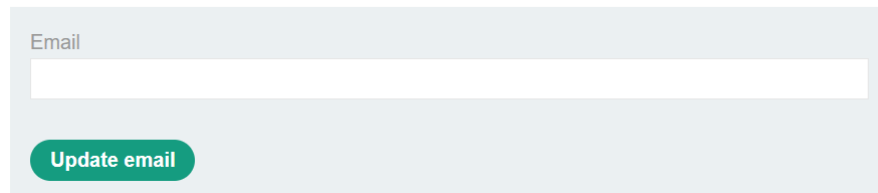
A screenshot of a web application's login page. The page has a light blue background. At the top, the word "Login" is written in blue. Below it, there are two input fields: "Username" and "Password". The "Username" field contains the text "administrator'--". The "Password" field is empty and has a blue border. Below the input fields is a green button with the text "Log in".

3. Forwarded the modified request to the server. Observed that the login was successful, granting access as the administrator without providing a password.

[Home](#) | [My account](#) | [Log out](#)

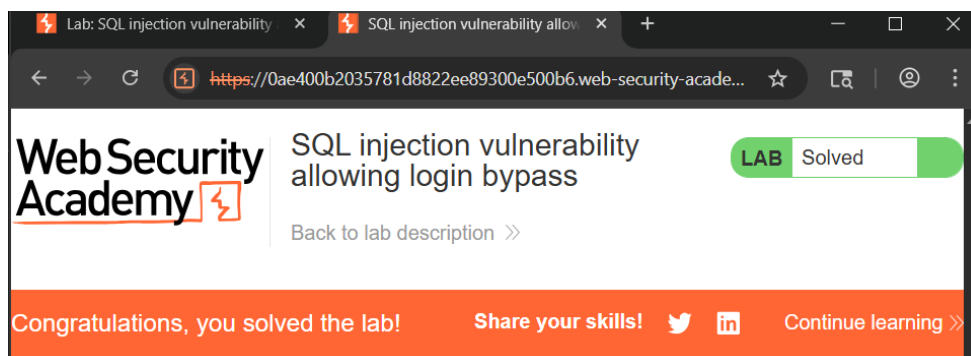
My Account

Your username is: administrator



A screenshot of a web application's "My Account" page. The page has a light blue background. At the top, the text "Your username is: administrator" is displayed in blue. Below it, there is an "Email" input field. At the bottom, there is a green button with the text "Update email".

4. Confirmed that the lab was solved by successfully logging into the admin account.



Vulnerability:

SQL Injection in Login Authentication

The login function fails to properly sanitize user input, allowing injected SQL comments (--) to bypass the password check by truncating the query logic.

Mitigation:

- Use parameterized queries/prepared statements for authentication queries.
- Avoid concatenating user input directly into SQL statements.
- Implement robust input validation and sanitization.
- Employ account lockout and monitoring to detect brute force and injection attempts.

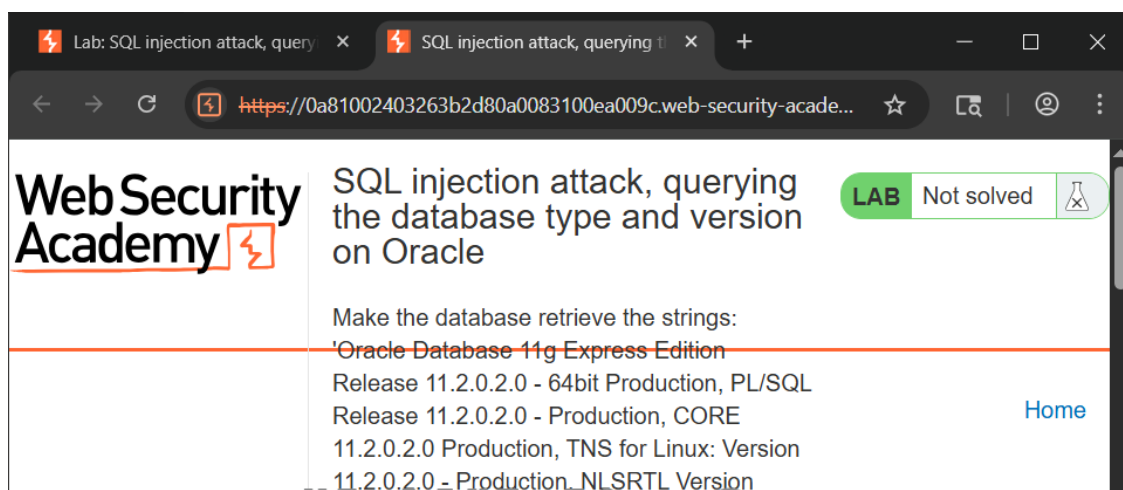
Lab 3: SQL Injection Attack, Querying the Database Type and Version on Oracle

Description:

This lab involves exploiting a SQL injection vulnerability in the product category filter to perform a UNION-based attack. The goal is to determine the database type and version by injecting crafted SQL queries and displaying the version string from the Oracle database.

Steps Performed:

1. Accessed the lab and identified the product category filter as vulnerable to SQL injection.



- Used Burp Suite to intercept the request containing the category parameter.

https://0a81002403263b2d...	GET	/resources/iaoneader/js/iaonead...	200	1673	script	js
https://0a81002403263b2d...	GET	/resources/images/shop.svg	200	7258	XML	svg
https://0a81002403263b2d...	GET	/resources/labheader/images/log...	200	8852	XML	svg
https://0a81002403263b2d...	GET	/resources/labheader/images/ps-l...	200	942	XML	svg
https://0a81002403263b2d...	GET	/academyLabHeader	✓	400	130	text
https://0a81002403263b2d...	GET	/filter?category=Gifts	✓	200	8861	HTML
https://0a81002403263b2d...	GET	/academyLabHeader	✓	400	130	text
https://portswigger.net	GET	/academy/labs/launch/974bd9df0...	✓	302	2153	
https://ps.piwik.pro	POST	/ppms.php	✓	202	419	HTML php

- Tested the number of columns and their data types by injecting the payload:

Pretty	Raw	Hex	Render
1 GET /filter?category= Gifts'+UNION+SELECT+Null,Null+FROM+dual-- HTTP/2 2 Host: 0a81002403263b2d80a0083100ea009c.web-security- academy.net 3 Cookie: session=			1 HTTP/2 200 OK 2 Content-Type: text/html; charset=utf-8 3 X-Frame-Options: SAMEORIGIN 4 Content-Length: 8844 5 6 <!DOCTYPE html> 7 <html>

- Confirmed the query returns two columns with text data.
- Used the following payload to retrieve the Oracle database version:

Pretty	Raw	Hex	Render
1 GET /filter?category= Gifts'+UNION+SELECT+BANNER,+Null+FROM+v\$version-- HTTP/2 2 Host: 0a81002403263b2d80a0083100ea009c.web-security-acad emy.net 3 Cookie: session=9QpbRuS5GRoURUIX9QwMPkxgaFfbj2e9 4 Sec-Ch-Ua: "Not)A;Brand";v="8", "Chromium";v="138"			1 HTTP/2 200 OK 2 Content-Type: text/html; charset=utf-8 3 X-Frame-Options: SAMEORIGIN 4 Content-Length: 9499 5 6 <!DOCTYPE html> 7 <html> 8 <head>

- Sent the modified request and observed the response displaying the database version string.
- Verified the successful extraction of the Oracle database version, solving the lab.

```
</a>  
</section>  
<table class="is-table-longdescription">  
  <tbody>  
    <tr>  
      <th>  
        CORE 11.2.0.2.0 Production  
      </th>  
    </tr>  
    <tr>  
      <th>  
        Conversation Controlling Lemon  
      </th>  
    </tr>  
  </tbody>  
</table>
```



Vulnerability:

Union-Based SQL Injection

The application fails to sanitize input used in SQL queries, allowing the injection of UNION SELECT statements to append additional query results. This enables attackers to extract database metadata like version information.

Mitigation:

- Use prepared statements to separate SQL code from user input.
- Enforce strict input validation and whitelisting.
- Limit error messages and database information exposed to users.
- Use least privilege for database accounts to reduce impact of injections.

Lab 4: SQL Injection UNION Attack, Determining the Number of Columns Returned by the Query

Description:

This lab focuses on discovering the number of columns returned by the original SQL query by performing a UNION-based SQL injection. The application reflects the query results in the response, allowing an attacker to infer the correct number of columns by observing errors when injecting different numbers of NULL values.

Steps Performed:

1. Accessed the lab and navigated to the product category "gift."

[Back to lab home](#)

[Back to lab description](#)

[Home](#) | [My account](#)

WE LIKE TO
SHOP

Corporate gifts

Refine your search:

[All](#) [Accessories](#) [Corporate gifts](#) [Food & Drink](#) [Lifestyle](#) [Toys & Games](#)

Folding Gadgets	\$29.63	View details
The Giant Enter Key	\$28.22	View details

2. Intercepted the HTTP request using Burp Suite.

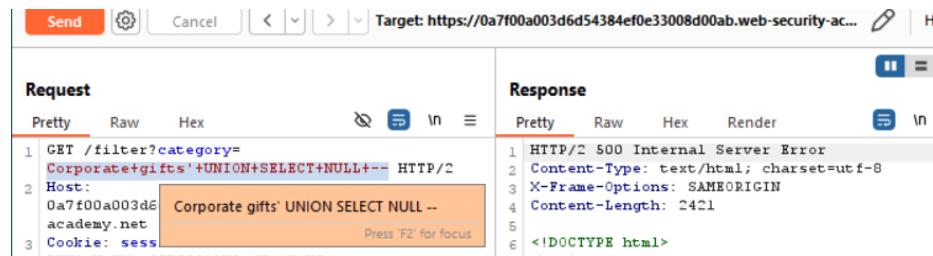
https://0a7f00a003d6d543...	GET	/resources/labheader/images/ps-l...		200	942	XML
https://0a7f00a003d6d543...	GET	/academyLabHeader	✓	400	130	text
https://0a7f00a003d6d543...	GET	/filter?category=Corporate+gifts	✓	200	5037	HTM
https://0a7f00a003d6d543...	GET	/academyLabHeader	✓	400	130	text
https://0a81002403263b2d...	GET	/		200	27936	HTM

3. Modified the category parameter by injecting a UNION SELECT statement with NULL values, starting with:

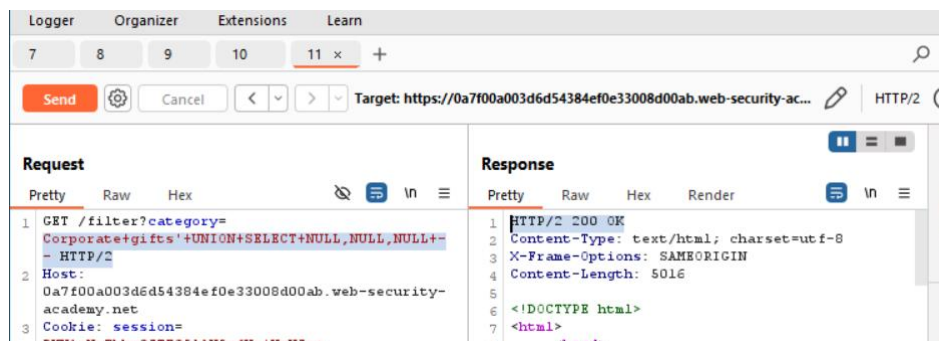
```

Pretty Raw Hex
1 GET /filter?category=Corporate+gifts HTTP/2
2 Host: 0a7f00a003d6d54384ef0e33008d00ab.web-security-academy.net
3 Cookie: session=RKULxNqEbbw3JTF351AU5ofUrjWnUJoz
  
```

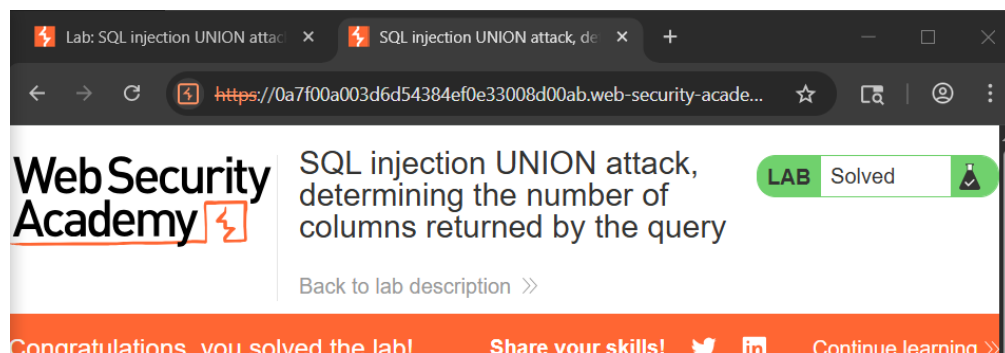
4. Received an error indicating a column count mismatch.



5. Incrementally added more NULL values in the payload (e.g., NULL, NULL, NULL, NULL, NULL) to match the number of columns expected by the query.



6. After three attempts, the request was successful, and the response included the injected NULL row, confirming the number of columns returned by the query.
7. Lab solved.



Vulnerability:

Union-Based SQL Injection - Column Enumeration

The application fails to properly sanitize input used in SQL queries. By injecting UNION SELECT statements with different numbers of columns, an attacker can deduce the structure of the query and craft more effective injections to retrieve data.

Mitigation:

- Use parameterized queries to separate code and data.
- Implement strict input validation and sanitization.

- Limit error message detail to avoid leaking information.
- Employ proper database permissions to restrict data access.

Lab 5: SQL Injection UNION Attack, Finding a Column Containing Text

Description:

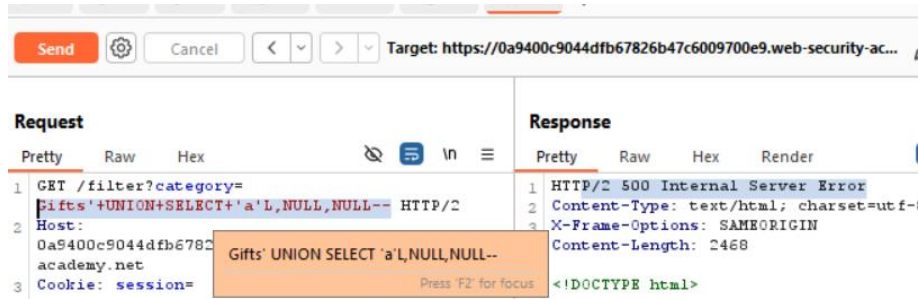
This lab requires identifying which column(s) in the UNION SELECT statement can accept string data. This is essential for extracting meaningful text data from the database using UNION-based SQL injection. The lab provides a random string that must appear in the query result to confirm successful injection.

Steps Performed:

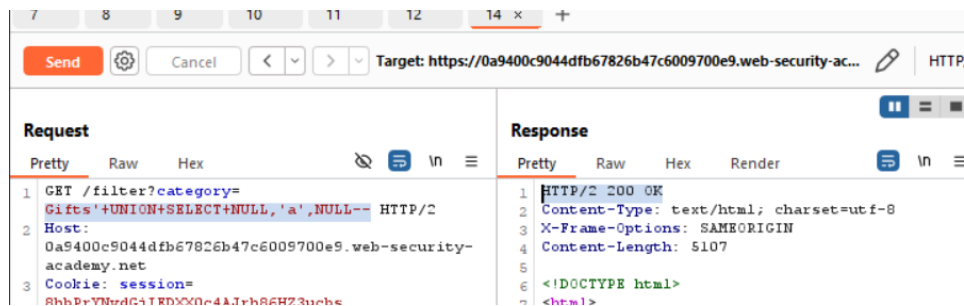
1. Accessed the lab and intercepted the request filtering by the "gift" category using Burp Suite.



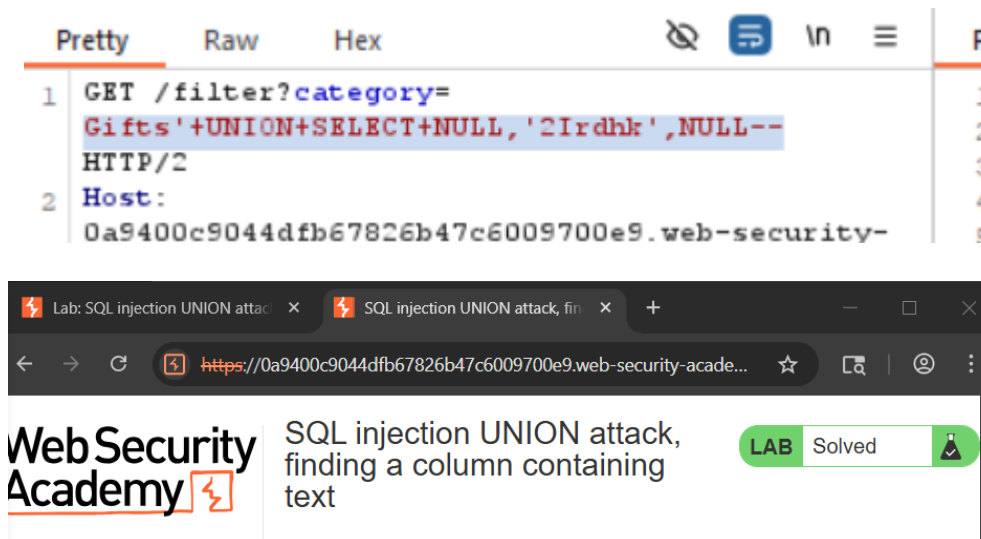
2. Sent the request to Burp Repeater and modified the category parameter with the payload.
3. Replaced each NULL one at a time with the letter 'a' to test which column accepts string data:



4. Replaced the 'a' in the second column with the lab-provided random string 2lrdhk.



5. Sent the request and confirmed the lab's random string appeared in the response, solving the lab.



Vulnerability:

Union-Based SQL Injection - Data Type Identification

The application lacks input sanitization and allows UNION SELECT injections. Identifying the correct column data types is crucial to retrieve meaningful data from the database, especially string-based data.

Mitigation:

- Use prepared statements to avoid injection vulnerabilities.
 - Validate and sanitize user input to enforce expected data types.
 - Suppress detailed database error messages.
 - Restrict database user permissions to minimize data exposure.
-