EXPERIMENT 7

Aim -

Implementation of data validation.(SQL Injection)

Theory -

What is SQL Injection?
A SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

SQL injection attack occurs when:

1. An unintended data enters a program from an untrusted source.
2. The data is used to dynamically construct a SQL query.

The main consequences are:

- Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL Injection vulnerabilities.
- Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.
- Authorization: If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL Injection vulnerability.
- Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL Injection attack.

Why is SQL Injection used?
SQL injection works by exploiting vulnerabilities in a website or computer application – usually through a data entry form. Hackers type SQL commands into fields such as login boxes, search boxes or ‘sign up’ fields. The aim is to use complex code sequences to gain access to a system and reveal the data held inside.
If your website uses weak or outdated security measures, then the hacker may be able to gain access by confusing the system. With this, cyber criminals can either steal the

data or shape it in ways which are disruptive to your business. In some cases, they can infiltrate your entire network.
Possible effects of a successful SQL injection attack include duplication, modification and deletion of data sets – all of which take time and money to resolve. Sometimes, data damage may be permanent.
The motivations behind an SQL injection attack are often financial. Hackers might sell sensitive data on the dark web, or malicious groups may wish to give themselves an advantage by setting your business back.

How to Prevent an SQL Injection?
The only sure way to prevent SQL Injection attacks is input validation and parameterized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.

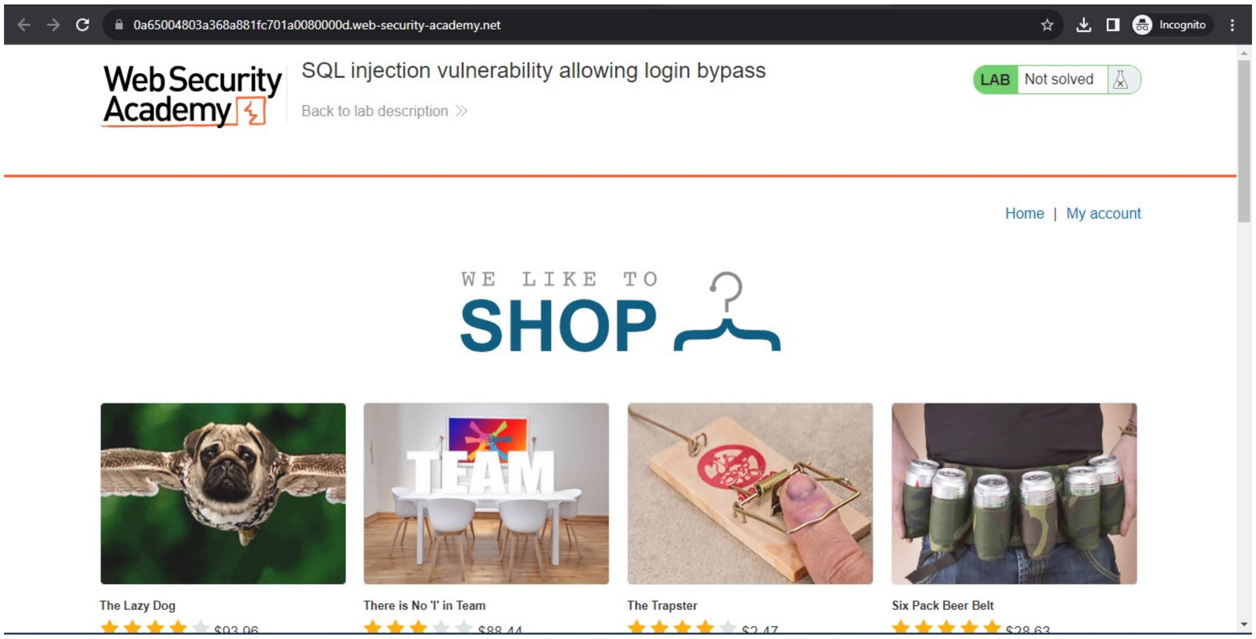
Process of SQL Injection-
To make an SQL Injection attack, an attacker must first find vulnerable user inputs within the web page or web application. A web page or web application that has an SQL Injection vulnerability uses such user input directly in an SQL query. The attacker can create input content. Such content is often called a malicious payload and is the key part of the attack. After the attacker sends this content, malicious SQL commands are executed in the database.

1. SQL injection vulnerability allowing login bypass

In cases where the results of an SQL query are returned within the application's responses, an attacker can leverage an SQL injection vulnerability to retrieve data from other tables within the database. Here, an attacker can log in as any user without a password simply by using the SQL comment sequence -- to remove the password check from the WHERE clause of the query. For example, submitting the username administrator'-- and a blank password results in the following query:

```
SELECT * FROM users WHERE username = 'administrator'--' AND password = “
```

This query returns the user whose username is administrator and successfully logs the attacker in as that user.



0a65004803a368a881fc701a0080000d.web-security-academy.net

SQL injection vulnerability allowing login bypass

Back to lab description

LAB Not solved

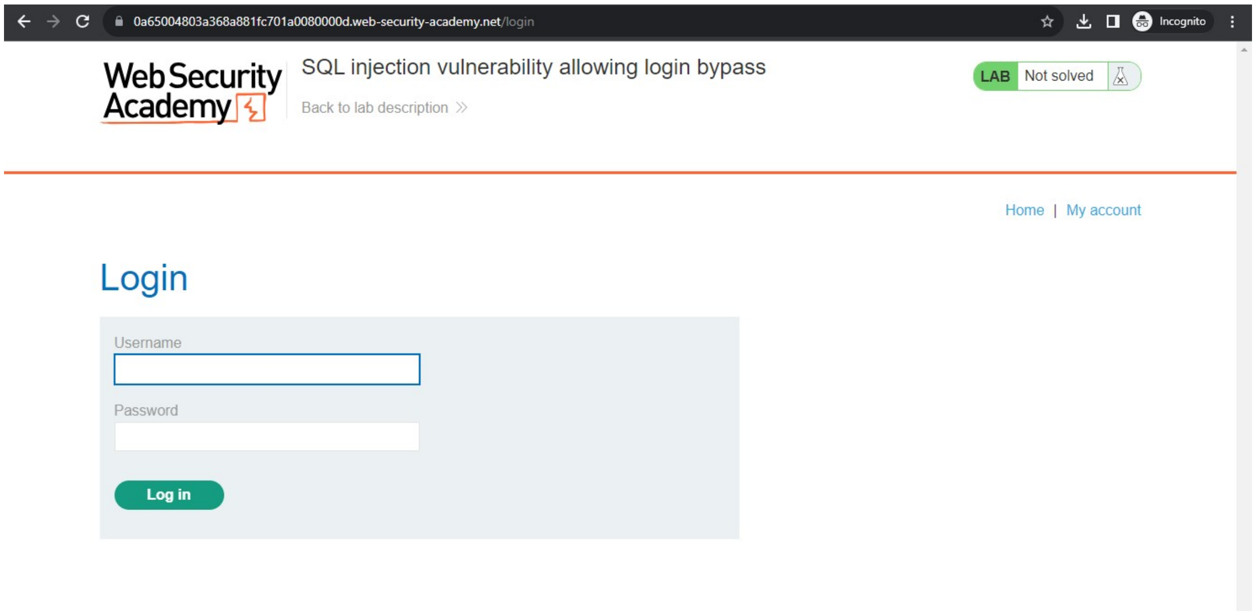
Home | My account

The Lazy Dog

There is No 'I' in Team

The Trapster

Six Pack Beer Belt



0a65004803a368a881fc701a0080000d.web-security-academy.net/login

SQL injection vulnerability allowing login bypass

Back to lab description

LAB Not solved

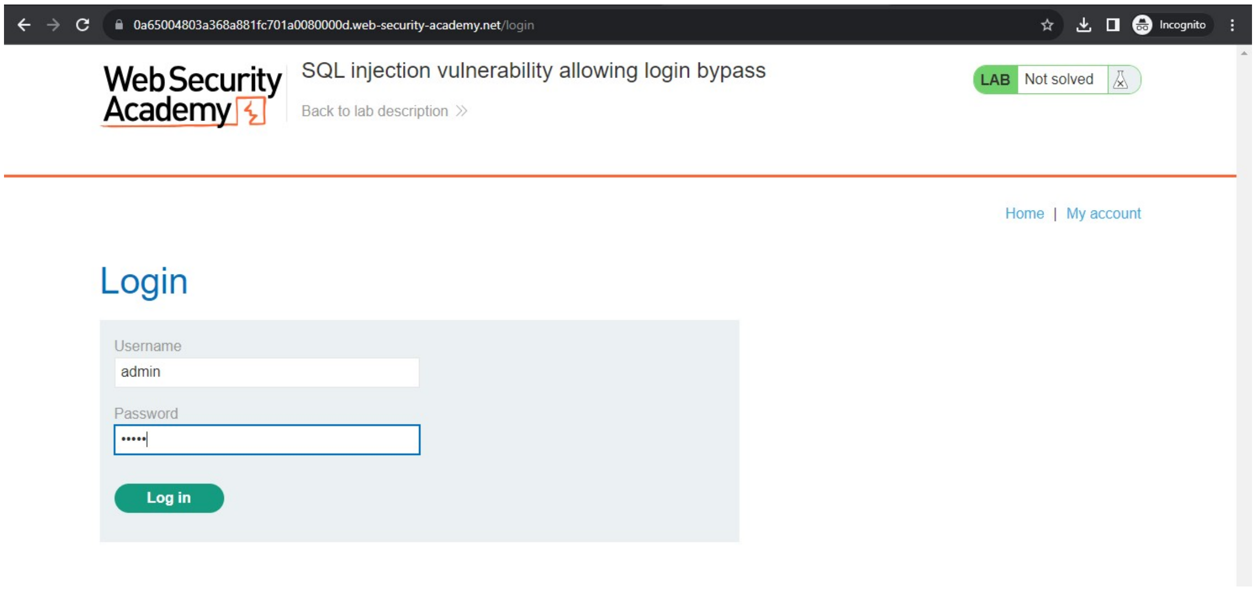
Home | My account

Login

Username

Password

Log in


0a65004803a368a881fc701a0080000d.web-security-academy.net/login

SQL injection vulnerability allowing login bypass

Back to lab description

LAB Not solved

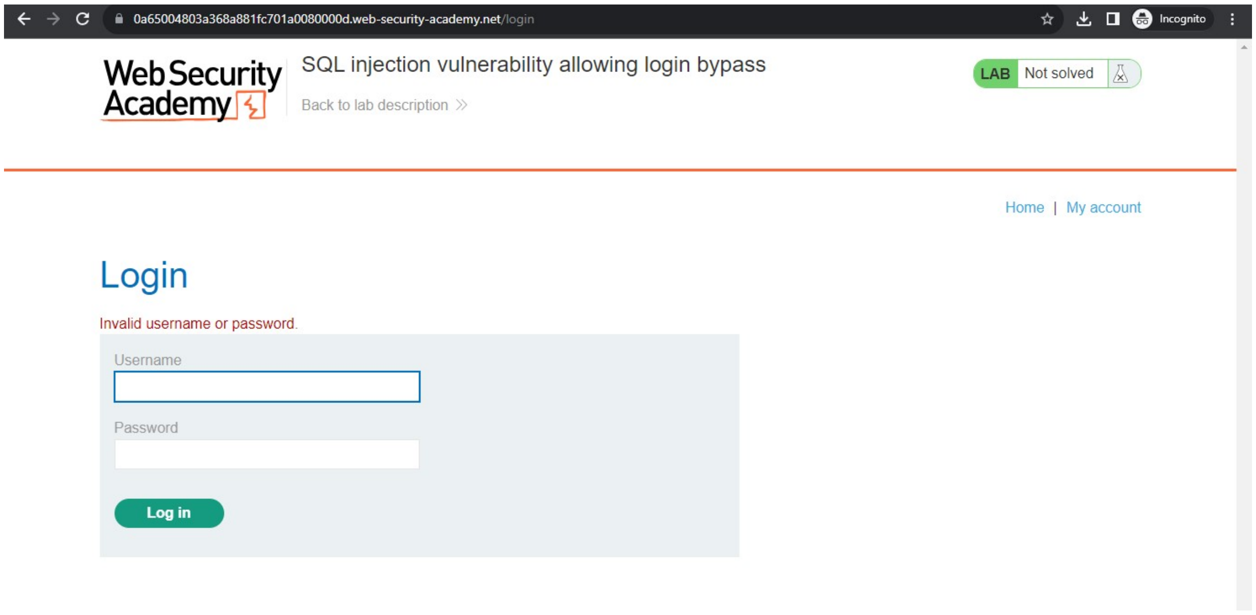
Home | My account

Login

Username: admin

Password: •••••

Log in


0a65004803a368a881fc701a0080000d.web-security-academy.net/login

SQL injection vulnerability allowing login bypass

Back to lab description

LAB Not solved

Home | My account

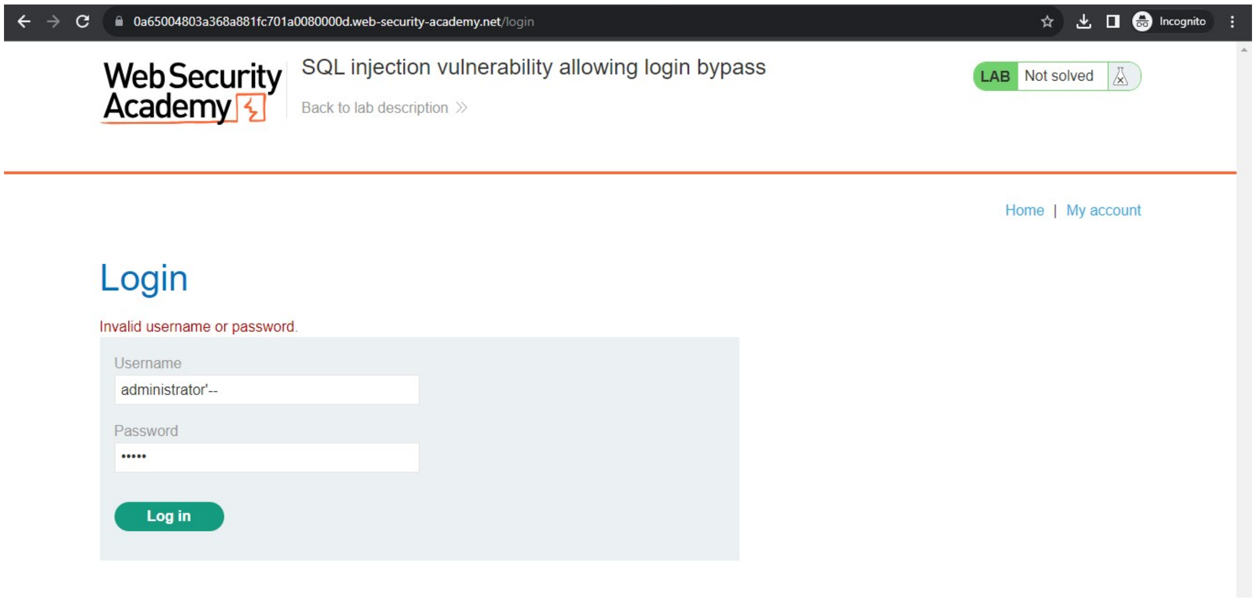
Login

Invalid username or password.

Username

Password

Log in


0a65004803a368a881fc701a0080000d.web-security-academy.net/login

SQL injection vulnerability allowing login bypass

Back to lab description

LAB Not solved

Home | My account

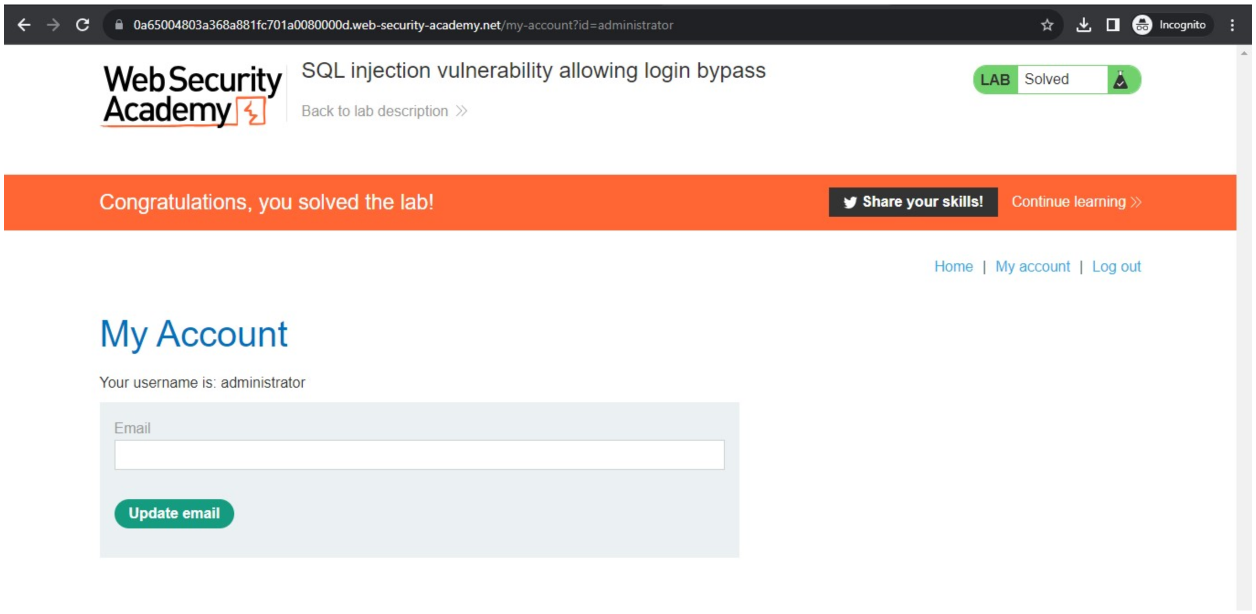
Login

Invalid username or password.

Username: administrator'--

Password: •••••

Log in



SQL injection vulnerability allowing login bypass

Congratulations, you solved the lab!

My Account

Your username is: administrator

2. SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

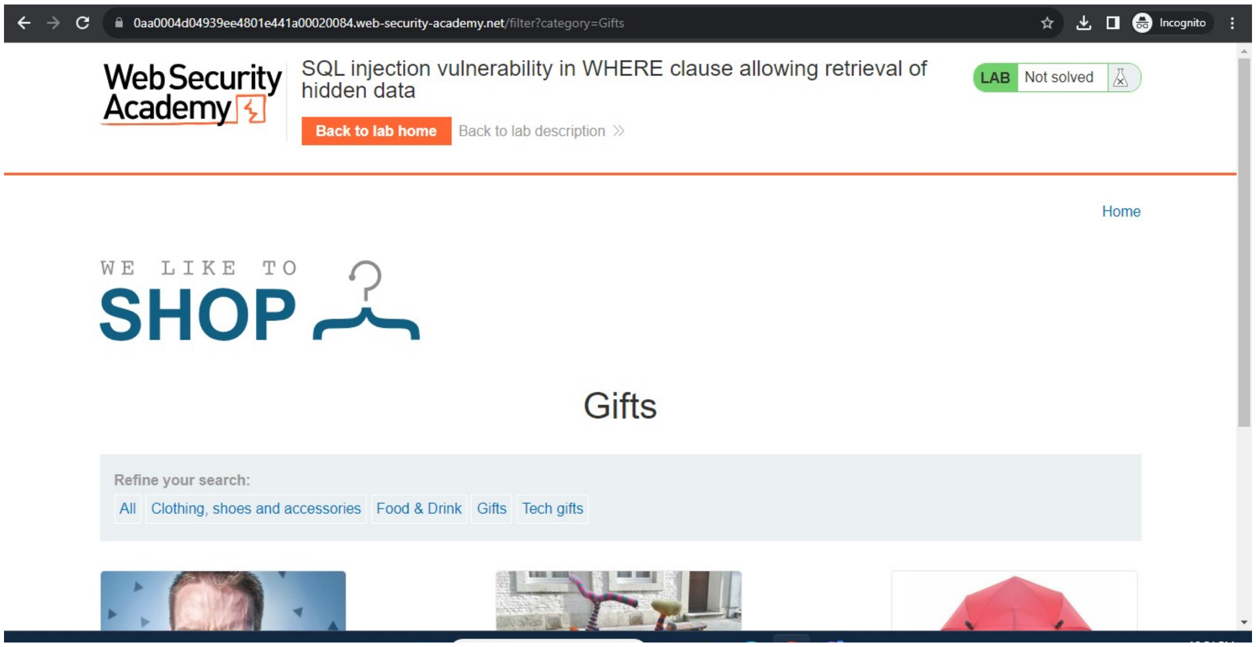
Consider a shopping application that displays products in different categories. This causes the application to make an SQL query to retrieve details of the relevant products from the database:

```
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 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. 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.



0aa0004d04939ee4801e441a00020084.web-security-academy.net/filter?category=Gifts

SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

LAB Not solved

Back to lab home Back to lab description

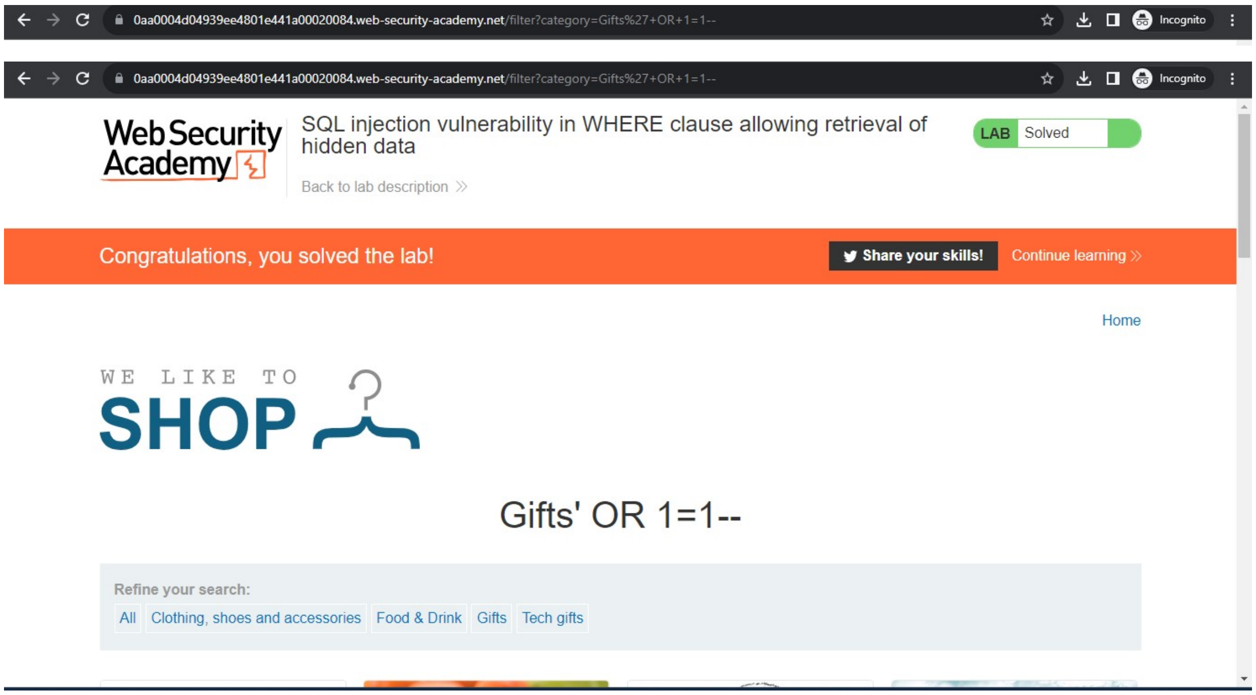
Home

WE LIKE TO SHOP

Gifts

Refine your search:

All Clothing, shoes and accessories Food & Drink Gifts Tech gifts



0aa0004d04939ee4801e441a00020084.web-security-academy.net/filter?category=Gifts%27+OR+1=1--

SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

LAB Solved

Back to lab description

Congratulations, you solved the lab! Share your skills! Continue learning

Home

WE LIKE TO SHOP

Gifts' OR 1=1--

Refine your search:

All Clothing, shoes and accessories Food & Drink Gifts Tech gifts

<u>Conclusion-</u>

Thus, we have successfully studied about SQL Injection and the different types of vulnerabilities which can be exploited using SQL injection along with mitigation methods for tackling these vulnerabilities.