SQL Injection

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

What is Database?

A database is an organized collection of data, so that it can be easily accessed and managed.You can organize data into tables, rows, columns, and index it to make it easier to find relevant information.

What is a Query?

A query is a request for data or information from a database table or combination of tables. This data may be generated as results returned by Structured Query Language (SQL) or as pictorials, graphs or complex

results, e.g., trend analyses from data-mining tools.

What is SQL Injection?

SQL Injection allows an attacker **to view** `data` that attackers are normally not able to retrieve.

`Data` can be information about users their credentials, personal details etc.

It is the process of inserting or injecting SQL queries through input fields to an application to make the application give the hacker, the data he wants!

Attacker can modify or delete this data causing persistent changes to the application's content or behavior.

SQL Injection can also be escalated to compromise the underlying server (or) other back-end infrastructure, or perform a denial-of-service attack

How does SQL Injection works?

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.

Types of SQL Injection:

In-band SQLi (Classic SQLi) : In-band SQL Injection is the most common and `easy-to-exploit` of SQL Injection attacks. In-band SQL Injection occurs when an attacker is able to use the `same communication channel` to both launch the attack and gather results. The two most common types of in-band SQL Injection are Error-based SQLi and Union-based SQLi.

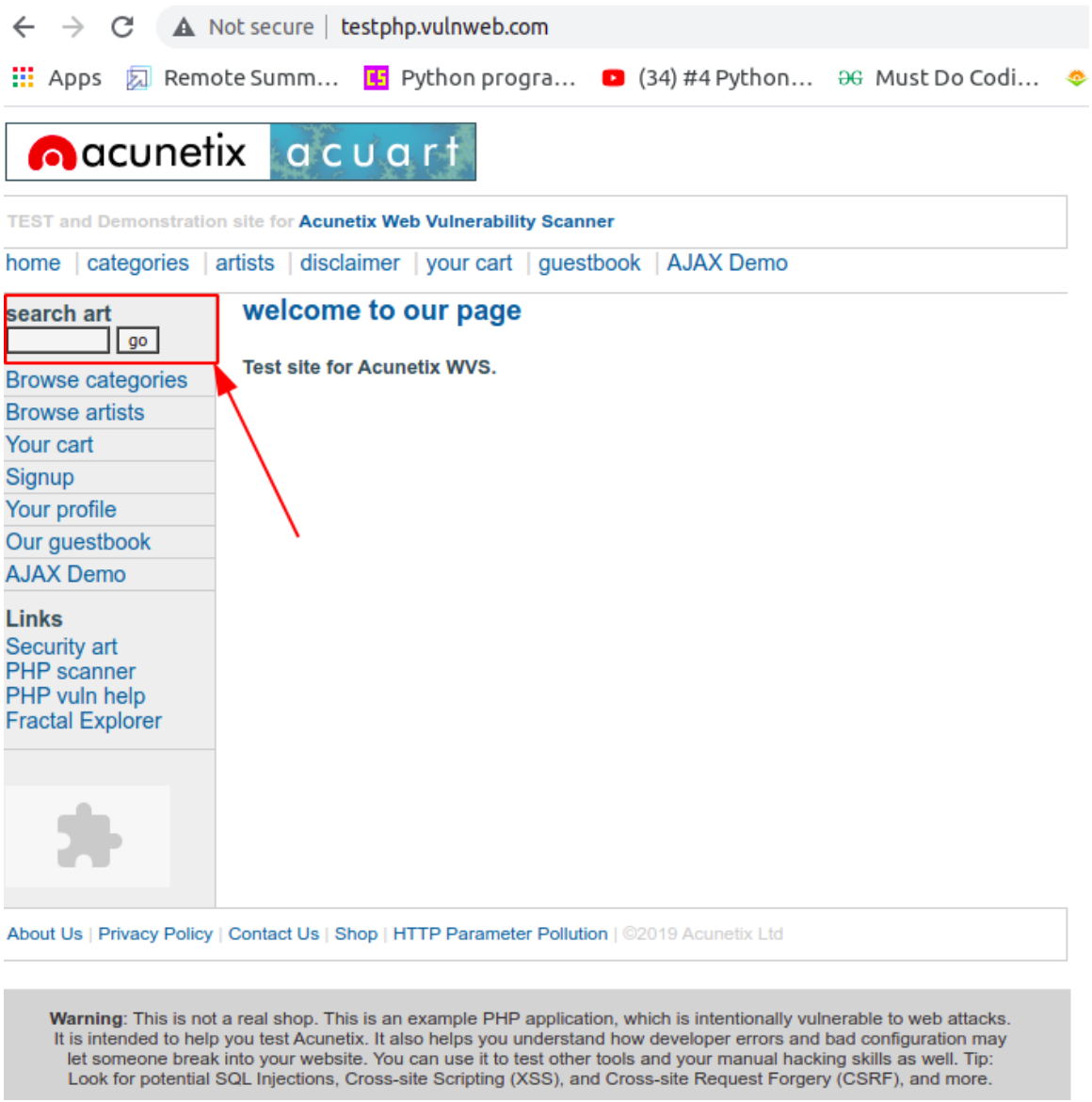
- **Error-based SQLi :** Error-based SQLi is an in-band SQL Injection technique that relies on `error messages` thrown by the database server to obtain information about the structure of the database. In some cases, error-based SQL injection alone is enough for an attacker to enumerate an entire database.

  Let's take an example for better understanding:

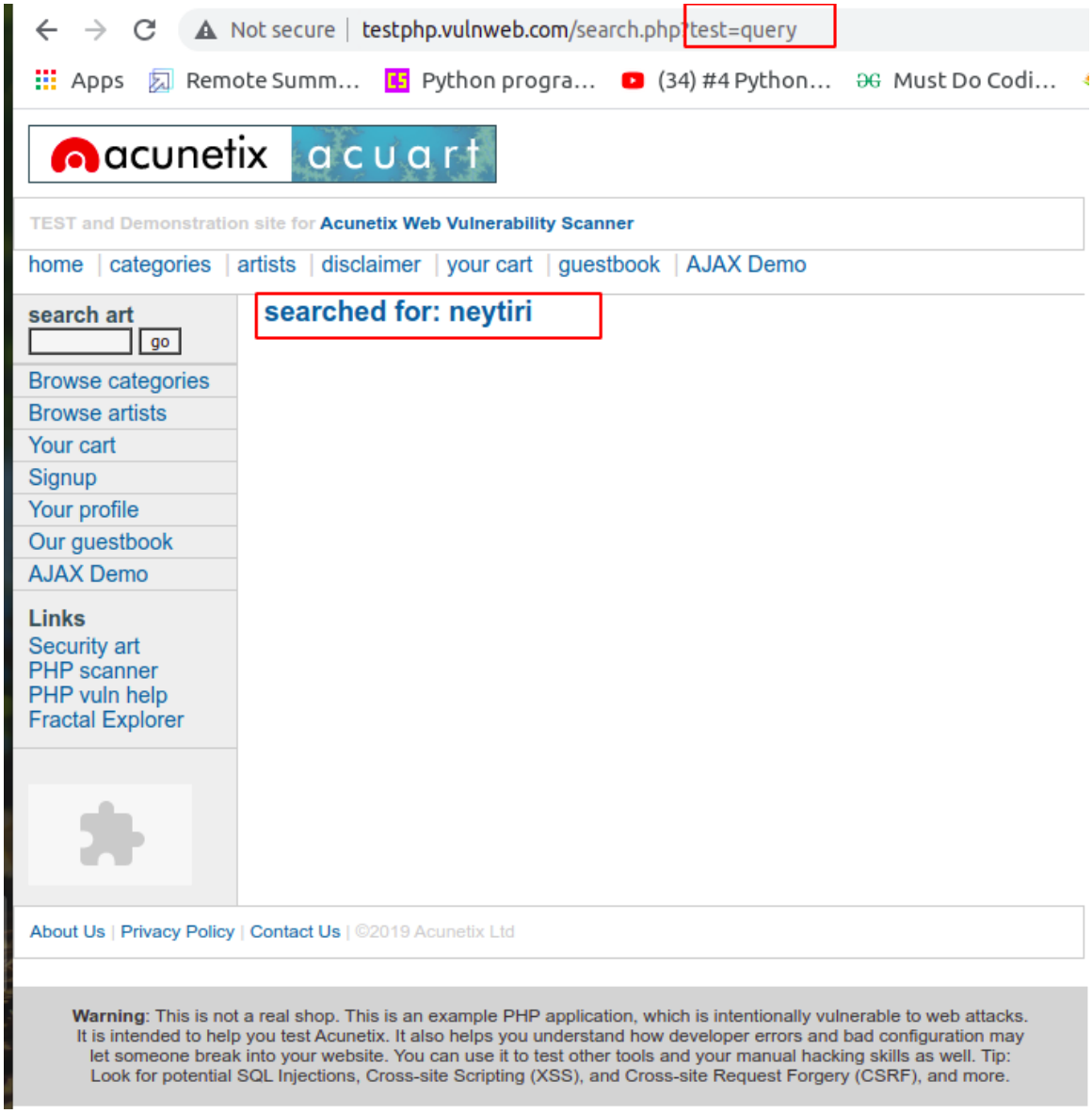
  This is the vulnerable website: testphp.vulnweb.com


acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
search art
go
Browse categories
Browse artists
Your cart
Signup
Your profile
Our guestbook
AJAX Demo
Links
Security art
PHP scanner
PHP vuln help
Fractal Explorer
welcome to our page
Test site for Acunetix WVS.
About Us | Privacy Policy | Contact Us | Shop | HTTP Parameter Pollution | ©2019 Acunetix Ltd
Warning: This is not a real shop. This is an example PHP application, which is intentionally vulnerable to web attacks. It is intended to help you test Acunetix. It also helps you understand how developer errors and bad configuration may let someone break into your website. You can use it to test other tools and your manual hacking skills as well. Tip: Look for potential SQL Injections, Cross-site Scripting (XSS), and Cross-site Request Forgery (CSRF), and more.

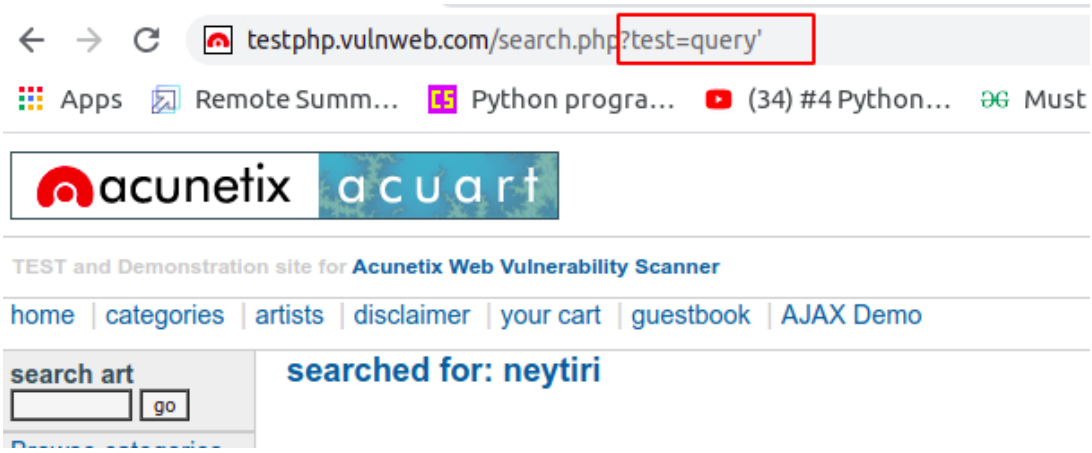
Let's begin!



We have a search box over here which can be used as our injection point. Lets first try to inject a simple search query.



Perfect! We are getting a reflection on the page and on the URL which is having a parameter where test=query.



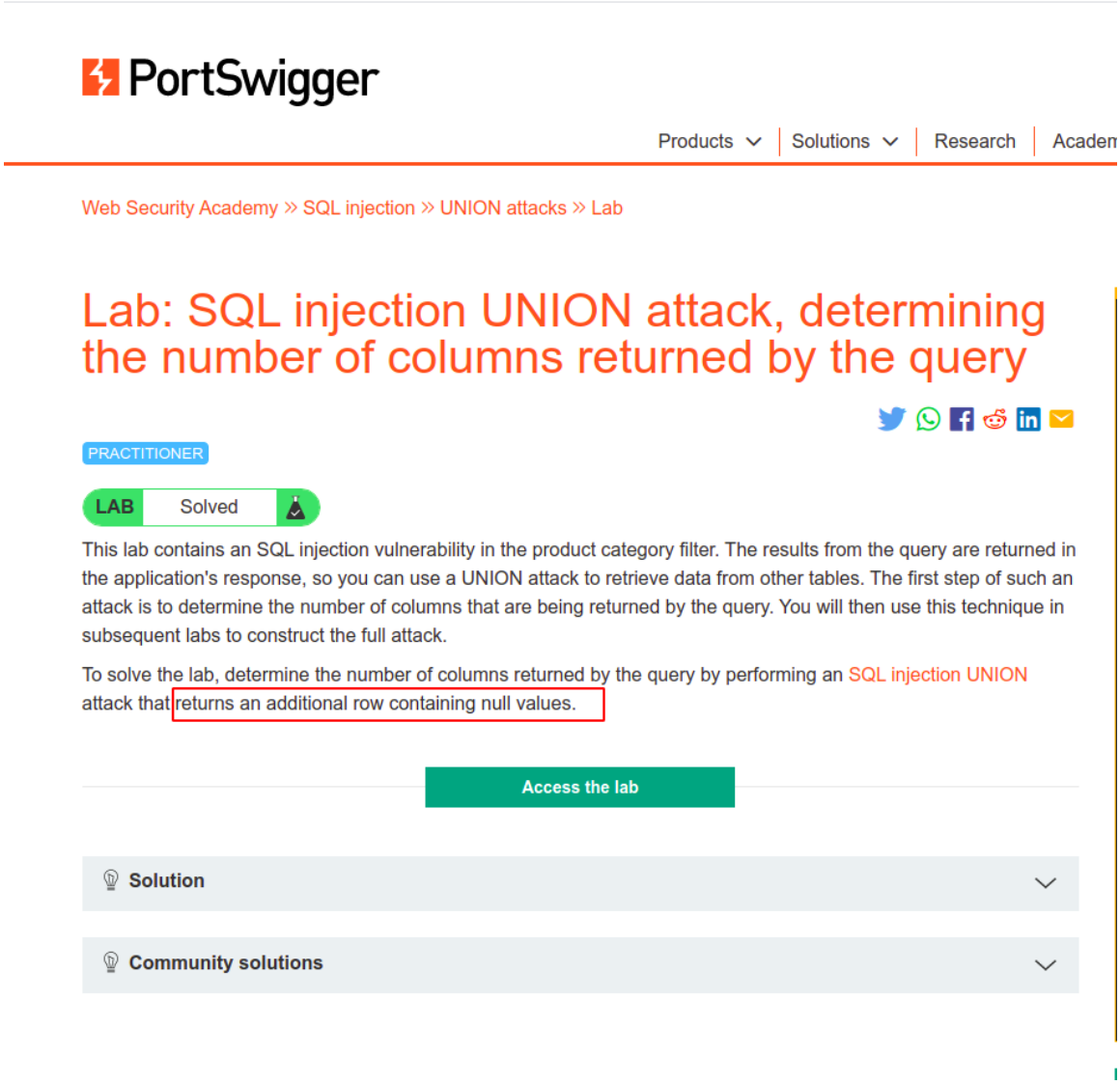
Now, as we know we can inject a malicious payload into the URL which may reflect a `SQL error` if there is any SQL vulnerability. Let's try to inject a simple payload `'` in the URL where test=query.

Notice that there is a SQL error on the page , which means the payload got excuted successfully and the webpage is vulnerable to `SQL injection` attack.

- **Union-based SQLi :** Union-based SQLi is an in-band SQL injection technique that leverages the `UNION SQL` operator to combine the results of two or more `SELECT` statements into a single result which is then returned as part of the HTTP response.

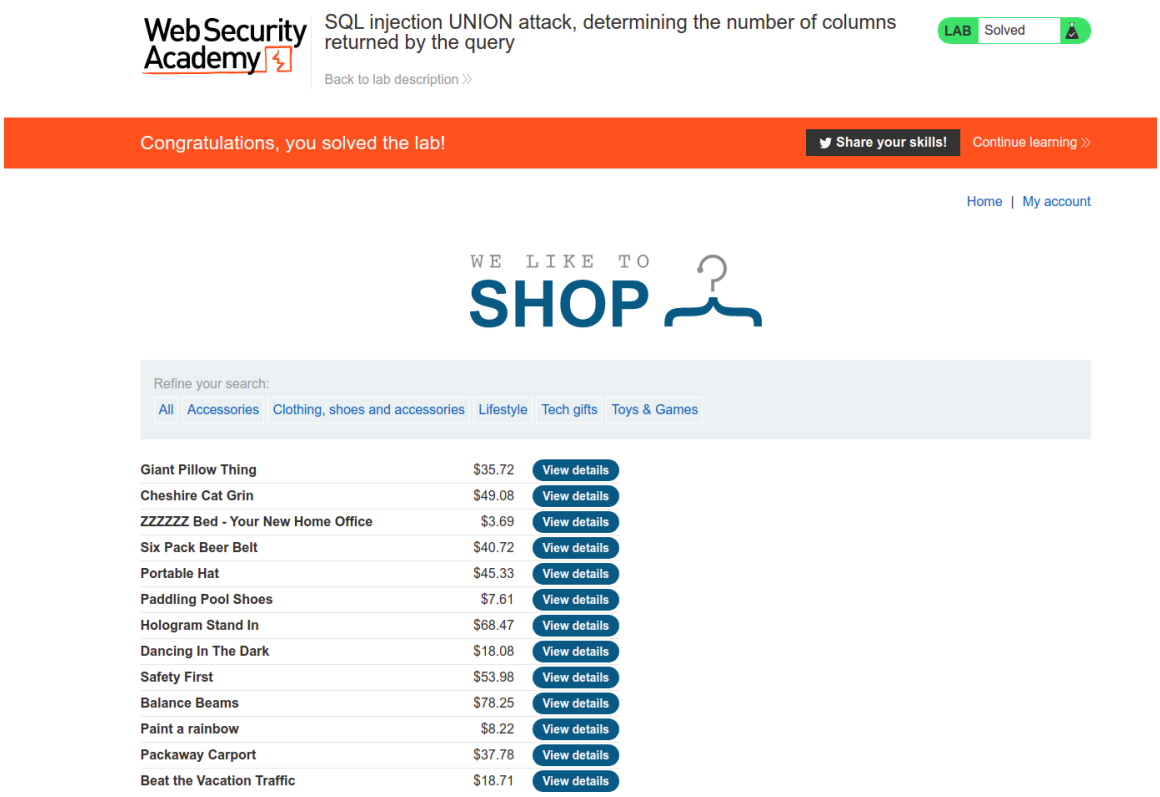
  Let's take an example for better understanding:

  This is the vulnerable website :https://portswigger.net/web-security/sql-injection/union-attacks/lab-determine-number-of-columns

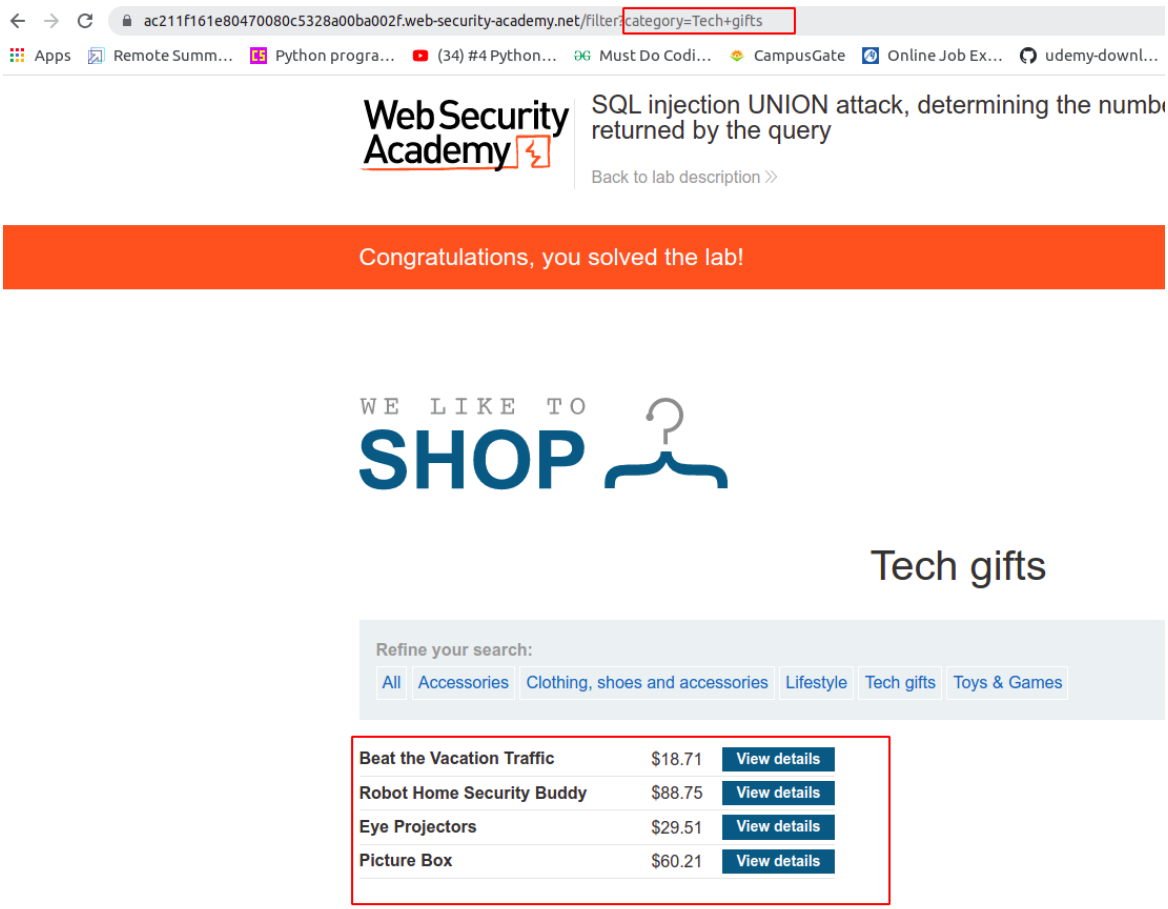


According to the question if there is SQL Injection attack it will return an additional row containing null values.

Let's Begin!

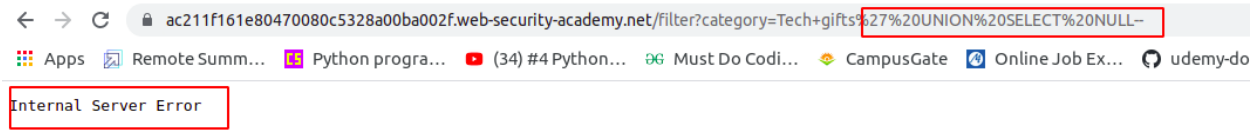


This is the web page that contains all product category.Lets go for Tech gifts category page and lets see what it returns.



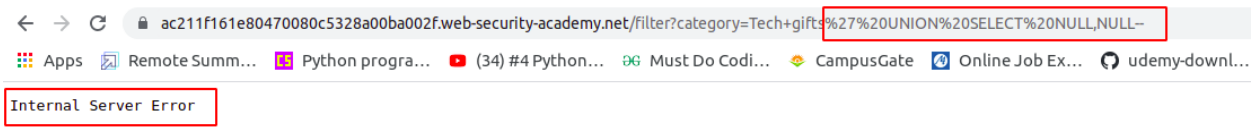
Tech gifts cateogry returned us with an url which has a paramter and there are 4 products listed in the cateogry. Now lets start testing the URL.

Lets first try with the simple payload `' UNION SELECT NULL–`



This payload gave us an error which means that the number of nulls does not match the number of columns therefore the database returned an error.

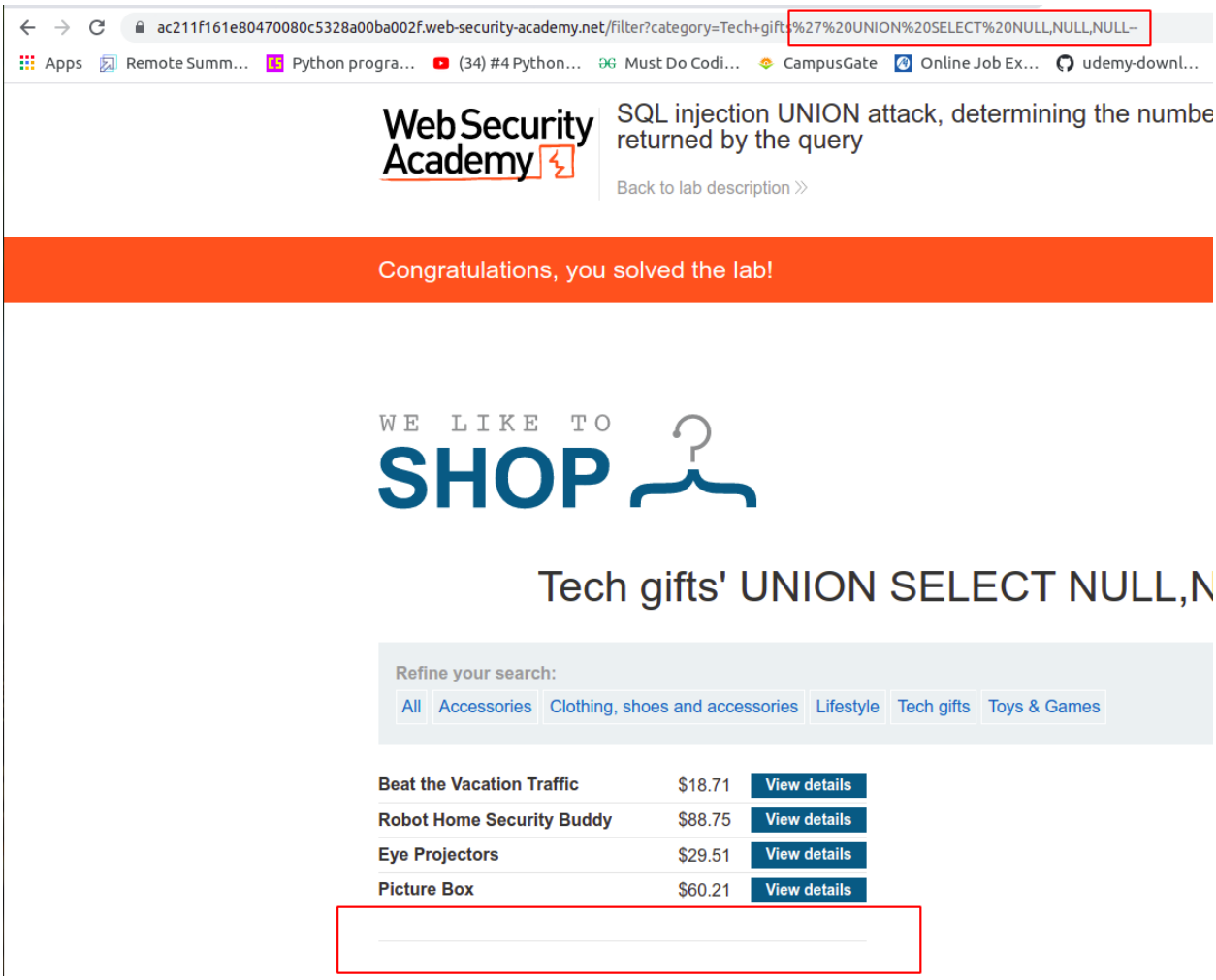
Lets increase the number of Null `' UNION SELECT NULL,NULL–`



It again gave us a error.

If the number of nulls does not match the number of columns, the database returns an error, such as: `All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in their target lists.`

Lets try to increase the Null `' UNION SELECT NULL,NULL,NULL--`



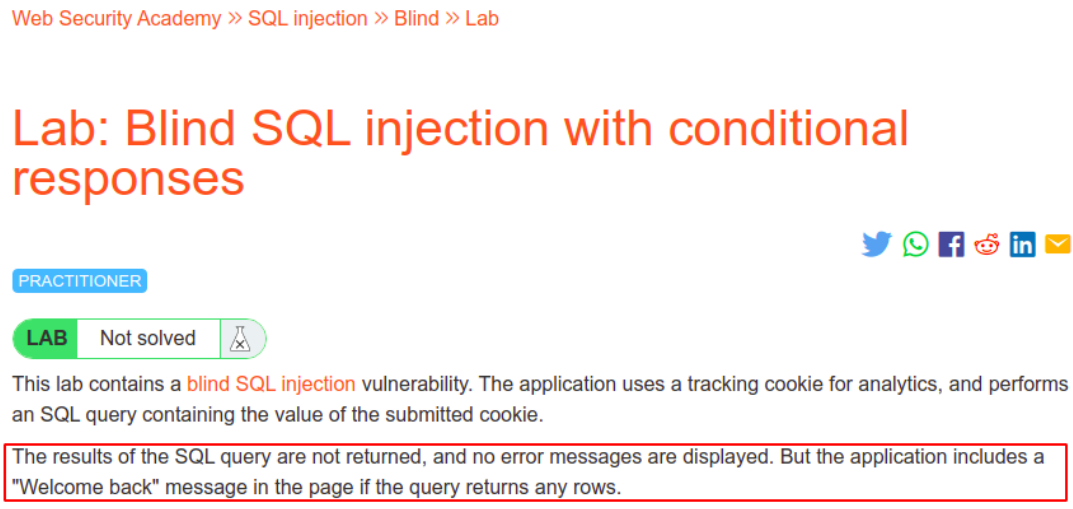
BOOM!!!!! It returned an additional row containing null values.Which means the `SQL Injection` attack was successful.

Inferential SQLi (Blind SQLi) : Inferential SQL Injection, unlike in-band SQLi, may take longer for an attacker to exploit, however, it is just as `dangerous` as any other form of SQL Injection. In an inferential SQLi attack, no data is actually transferred via the `web application` and the attacker would not be able to see the result of an attack in-band (which is why such attacks are commonly referred to as “blind SQL Injection attacks”). Instead, an attacker is able to `reconstruct` the database structure by sending payloads, observing the web application’s response and the resulting behavior of the database server. The two types of inferential SQL Injection are Blind-boolean-based SQLi and Blind-time-based SQLi.

- **Boolean-based (content-based) Blind SQLi :** Boolean-based SQL Injection is an inferential SQL Injection technique that relies on sending an `SQL query` to the database which forces the application to return a different result depending on whether the query returns a TRUE or FALSE result. Depending on the result, the content within the HTTP response will change, or remain the same. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned.

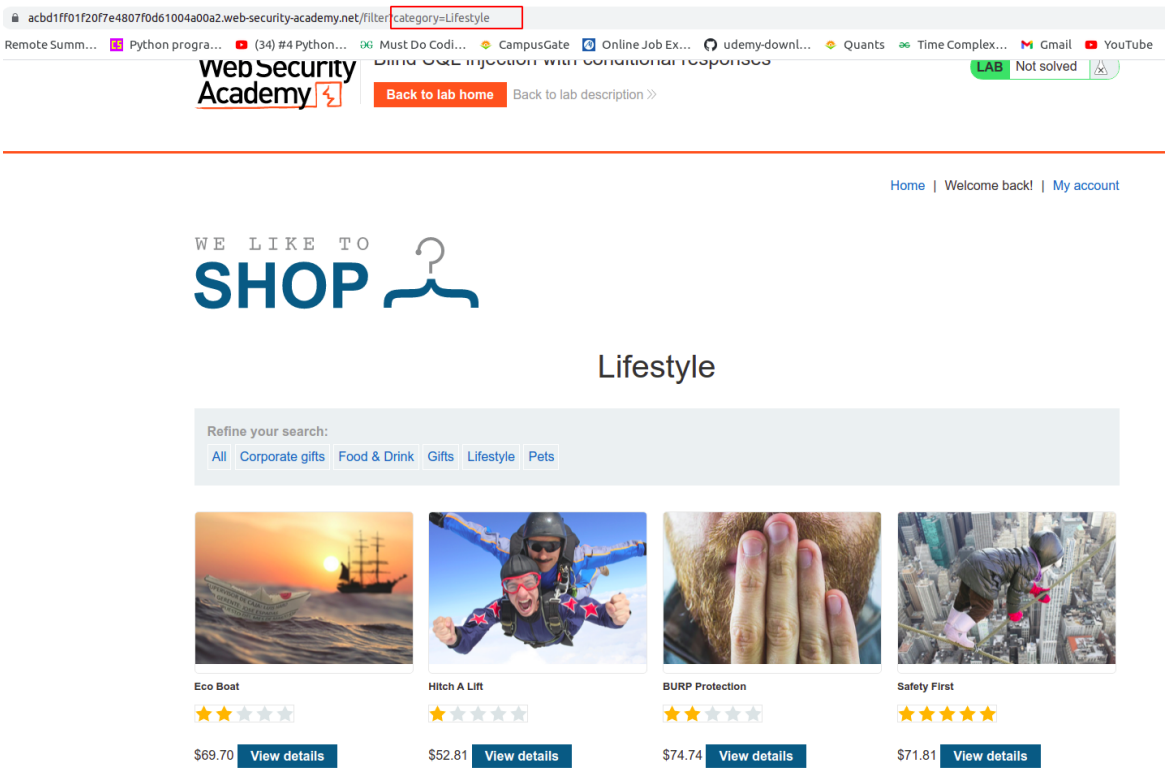
  Let's take an example for better understanding:

  This is the vulnerable website : https://portswigger.net/web-security/sql-injection/blind/lab-conditional-responses



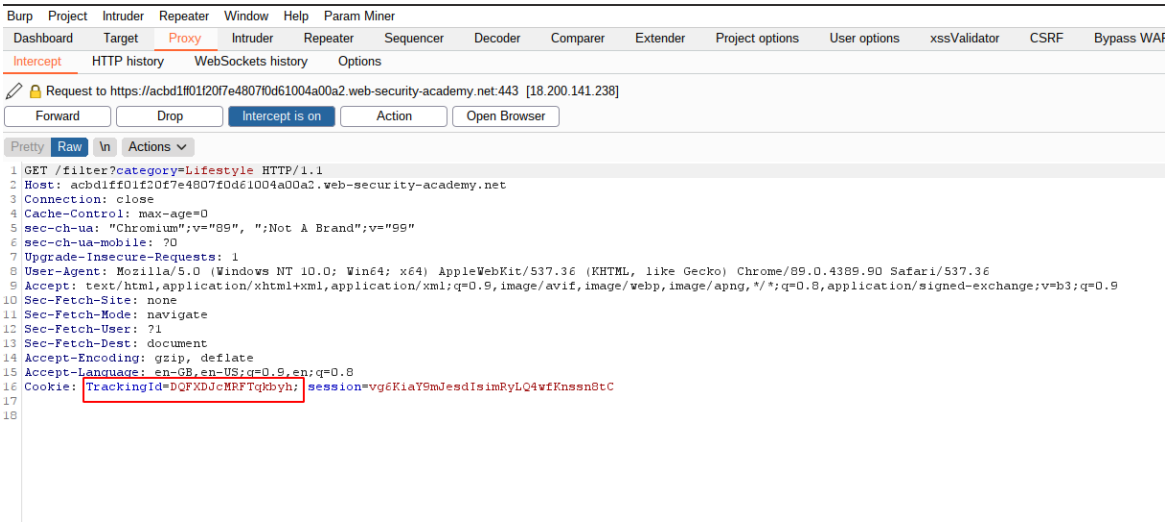
In this lab we have to check for Welcome back message in the response .

Let's Begin!



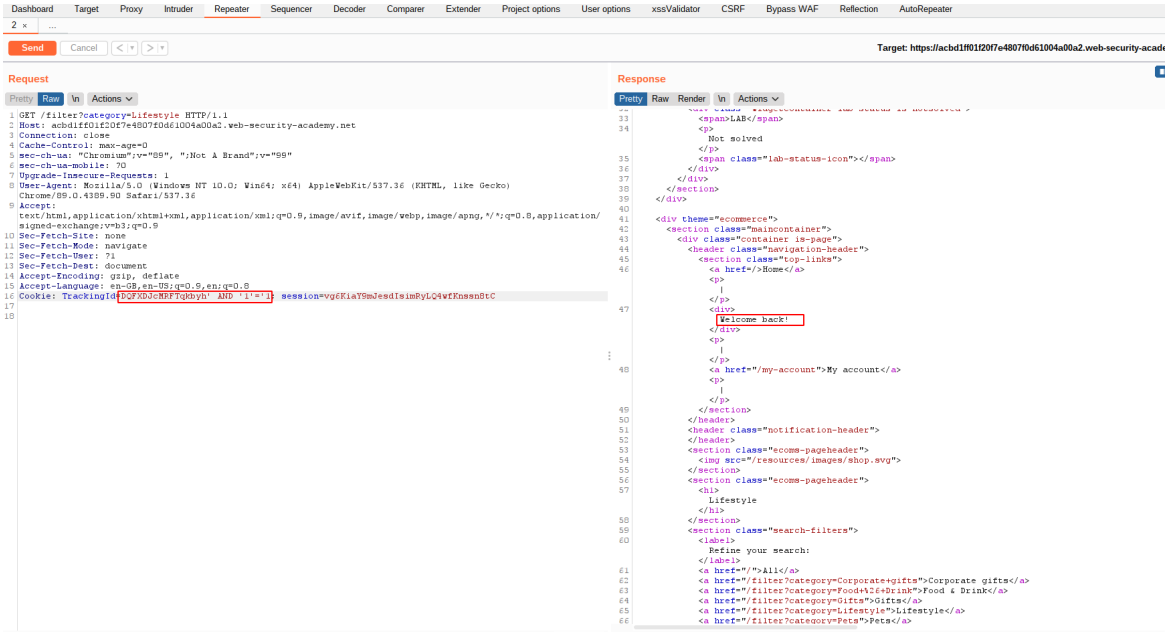
Here in the webpage when we select the category Lifestyle we can clearly see the parameter reflection in the URL page.

Lets start our BrupSuite and reload the page.



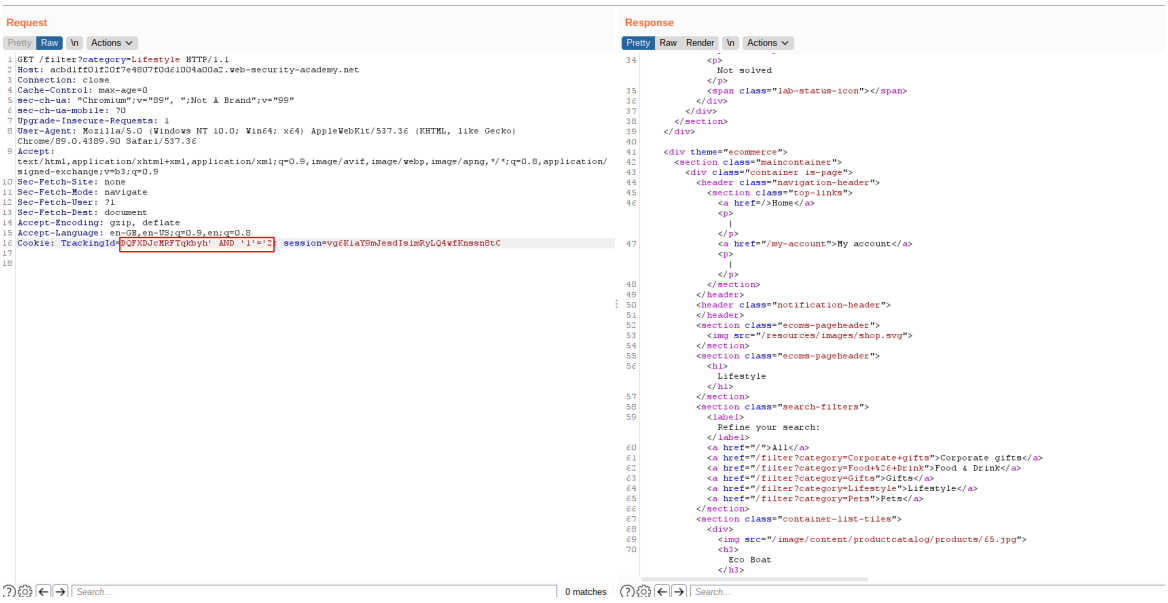
Here is the request we can see the cookie which contains TeackingId.Lets try to modify it and check the response in the repeater tab.

Where `TrackingId=DQFXDJcMRFTqkbyh' AND '1'='1`



Since here "1 = 1" condition is true we get `Welcome Back!` message in the response tab.

Now lets change the payload where `TrackingId=DQFXDJcMRFTqkbyh'AND '1'='2`



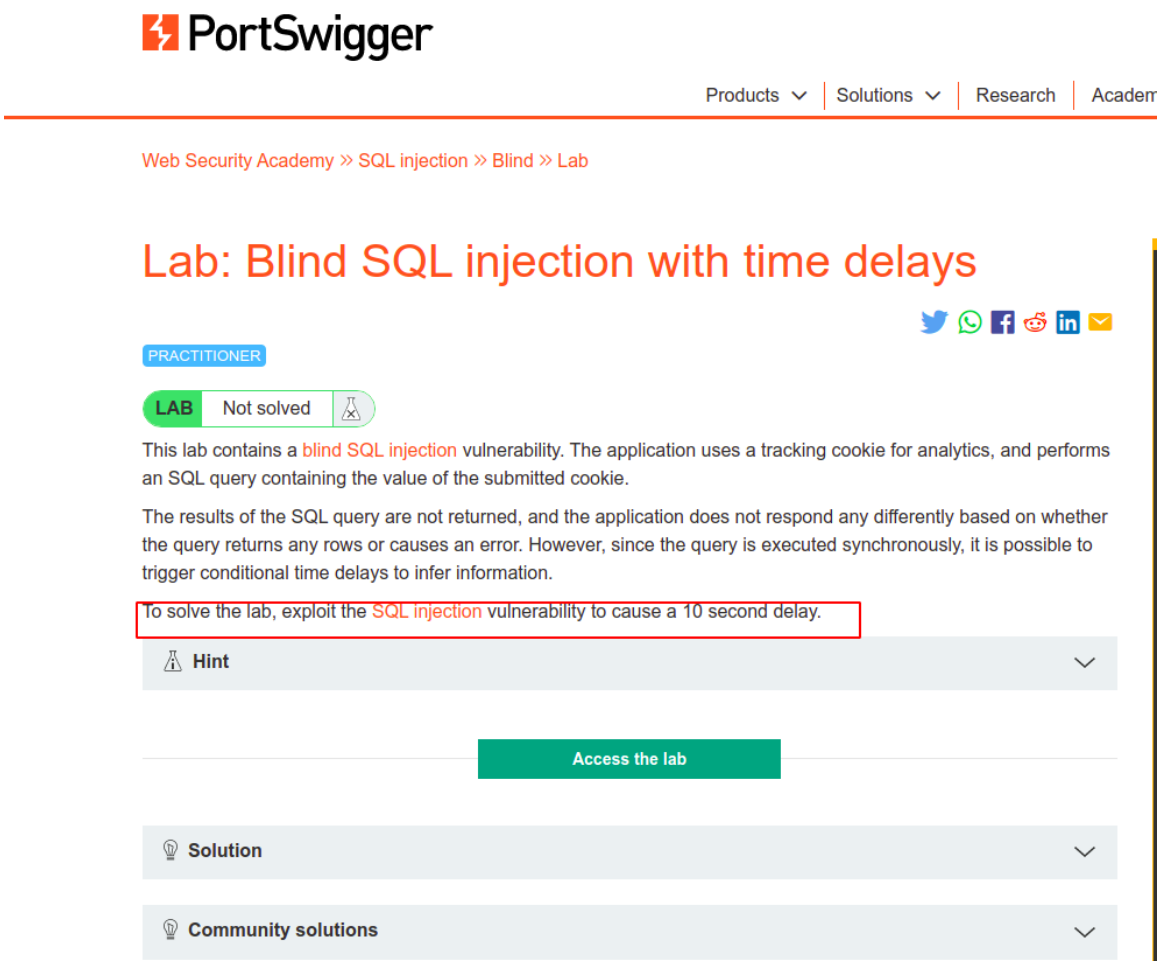
Here we didn't get the Welcome Back message since "1 = 2 " is a `false` condition.

- **Time-based Blind SQLi :** Time-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which `forces the database` to wait for a specified amount of time (in seconds) before responding. The response time will indicate to the attacker

whether the result of the query is `TRUE` or `FALSE`. epending on the result, an HTTP response will be returned with a delay, or returned immediately. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned.

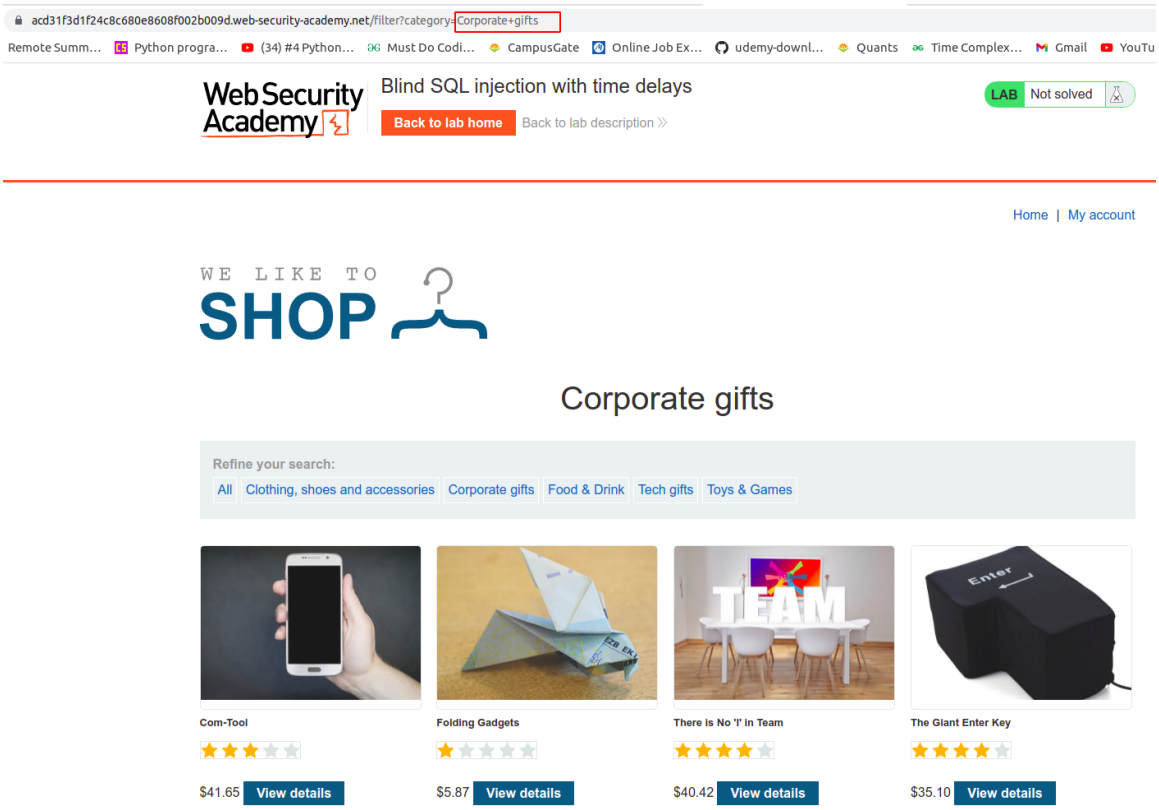
Let's take an example for better understanding:

This is the vulnerable website: https://portswigger.net/web-security/sql-injection/blind/lab-time-delays

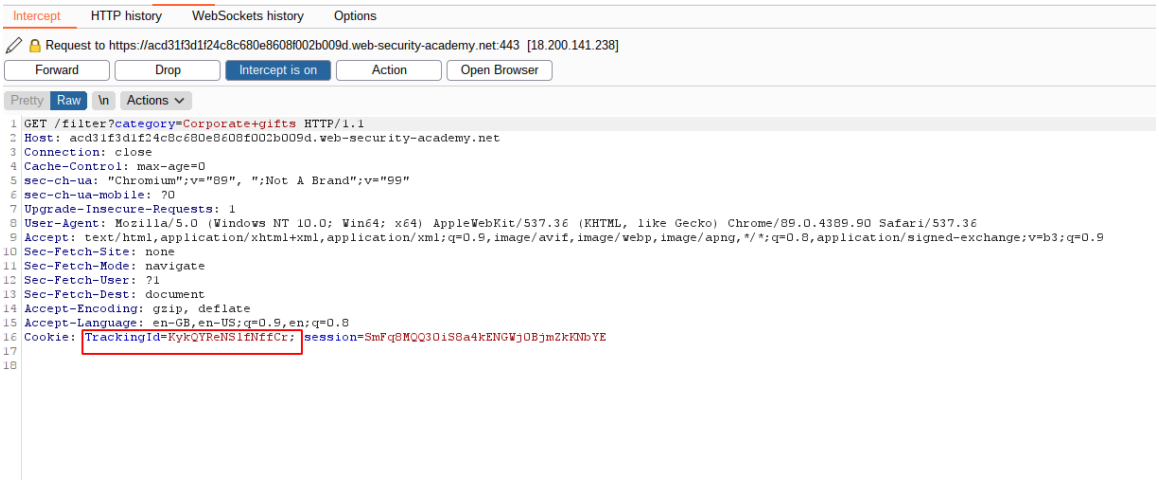


Acoording to the question given , to solve the lab we have to exploit the SQL Injection vulnerability to cause a 10 second delay.

Let's Begin!

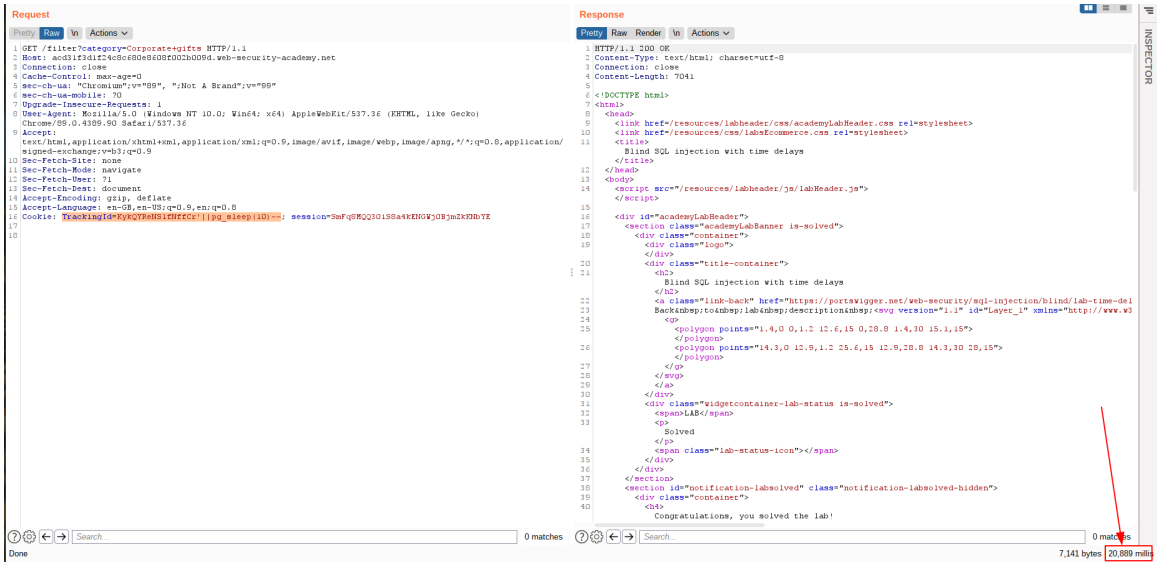


When we load the webpage and go to the Coporate gifts category page we can see there is a parameter in the URL .Let's try to reload the page and see the request in the BurpSuite.



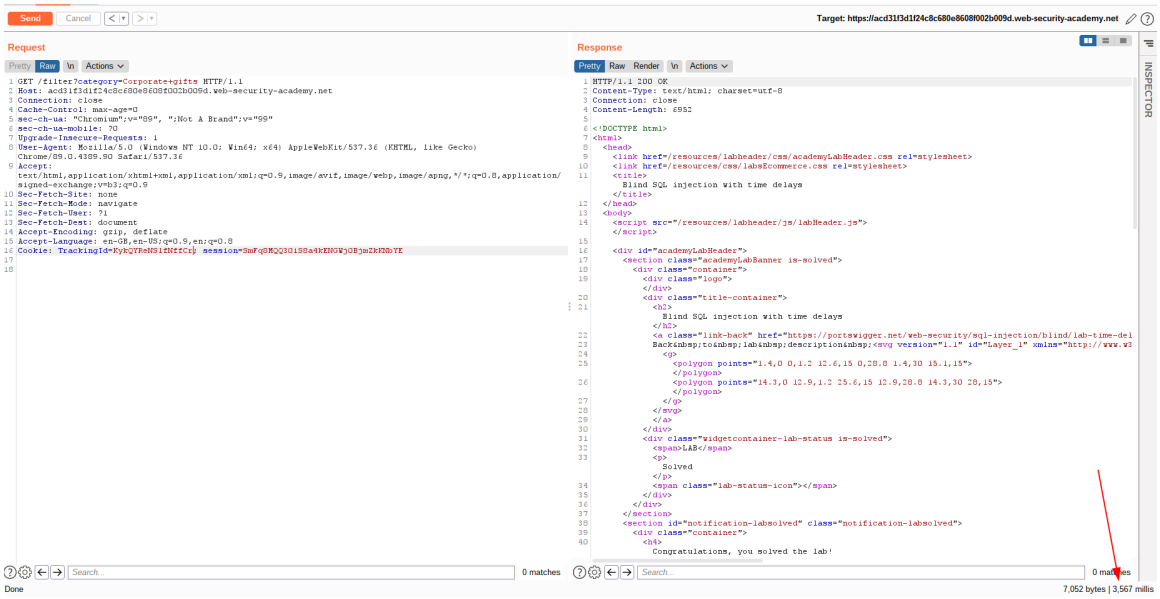
Here is the request we can see the cookie which contains TeackingId.Lets try to modify it and check the response in the repeater tab.

Where `TrackingId=KykQYReNSlfNffCr'||pg_sleep(10)–`



There was time delay of 10 seconds.

Now lets check without the payload.



The page got loaded without any delay.

Out-of-band SQLi: Out-of-band SQL Injection is not very common, mostly because it depends on features being enabled on the database server being used by the web application. Out-of-band SQL Injection occurs when an attacker is unable to use the same channel to launch the attack and gather results. Out-of-band

techniques, offer an attacker an alternative to `inferential time-based techniques`, especially if the server responses are not very stable (making an inferential time-based attack unreliable).

Voice Based Sql Injection: It is a sql injection attack method that can be applied in applications that provide access to databases with `voice command`. An attacker could pull information from the database by sending sql queries with sound.

SQL Injection Attack through sqlmap

sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers.

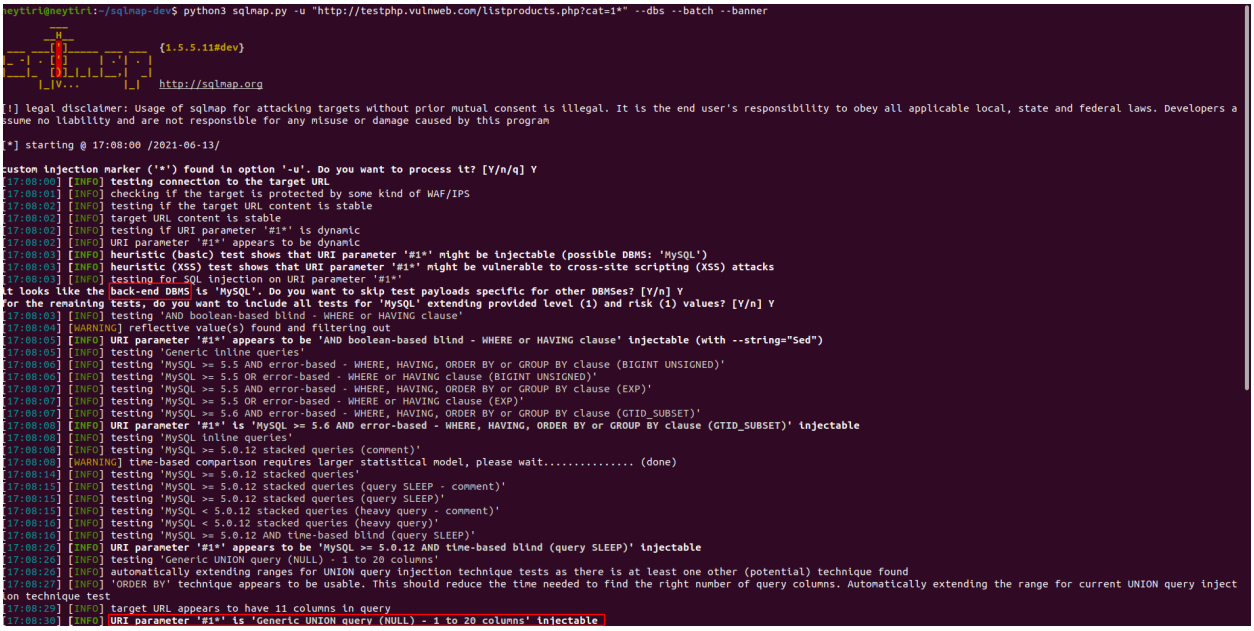
It can be downloaded from https://github.com/sqlmapproject/sqlmap

Let's see with an example how to use sqlmap:

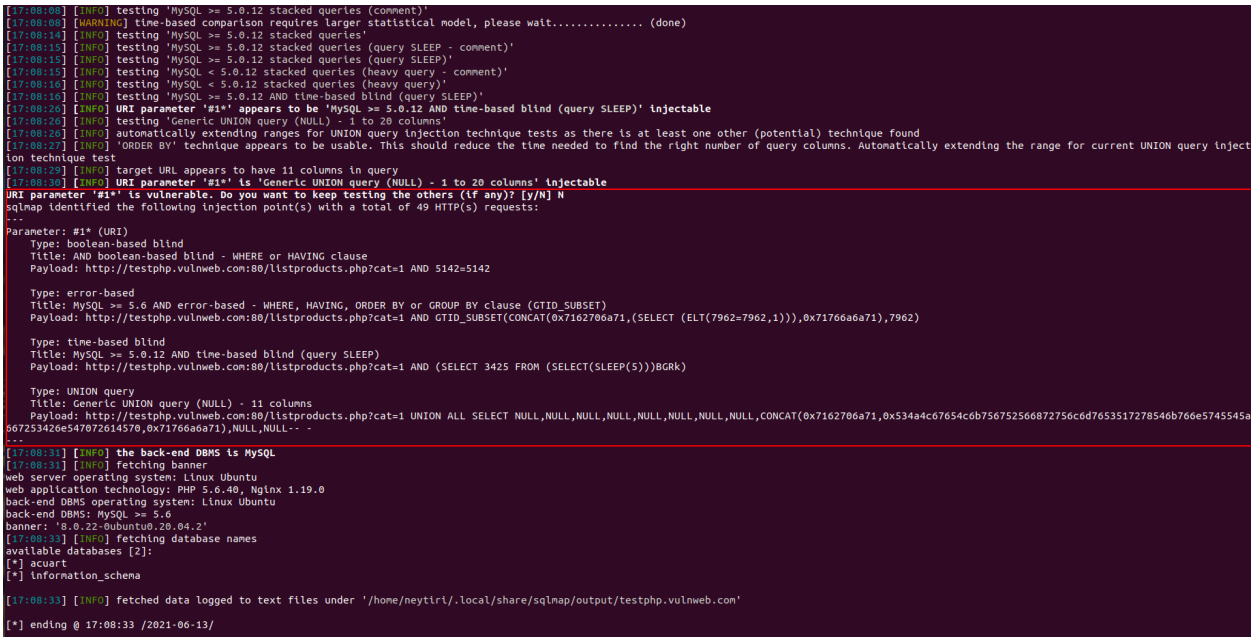
So let's test the vulnerable endpoint http://testphp.vulnweb.com/listproducts.php?cat=1

The Syntax for the command is

```
python3 sqlmap.py -u "http://testphp.vulnweb.com/listproducts.php?cat=1*" --dbs --batch --banner
```



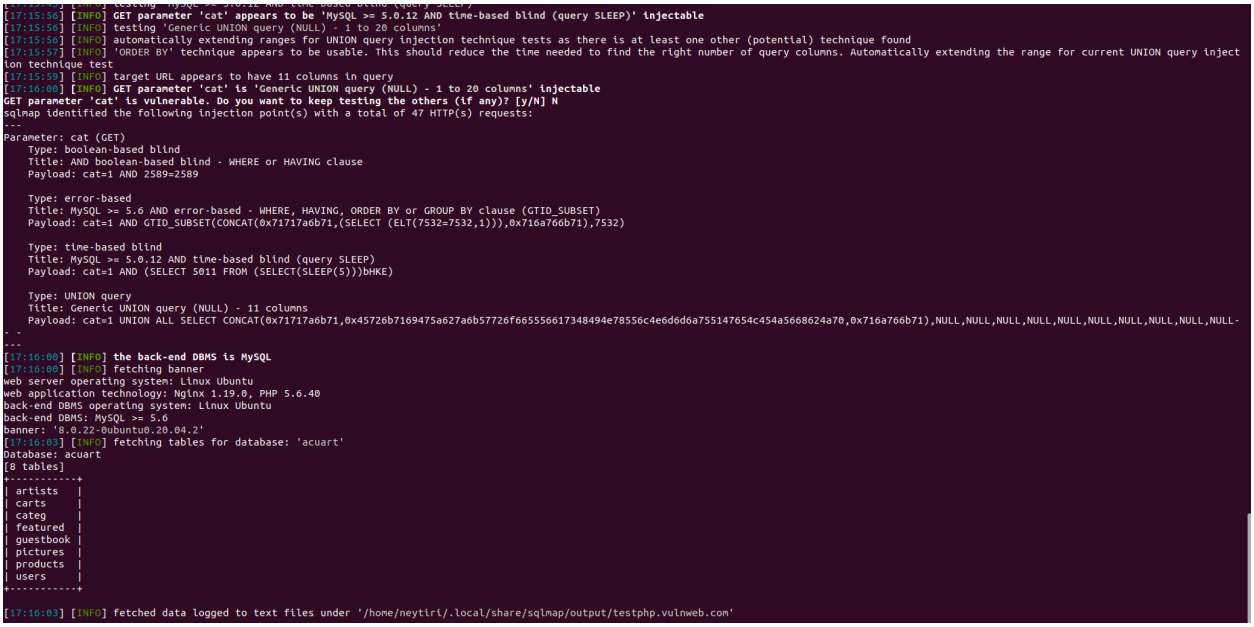
Here we can see cat parameter is vulnerable.Lets wait for sql map to complete its scanning.



Perfect!!! Here is the result of SQLmap which shows that the URL is vulnerable to SQL Injection Attack.

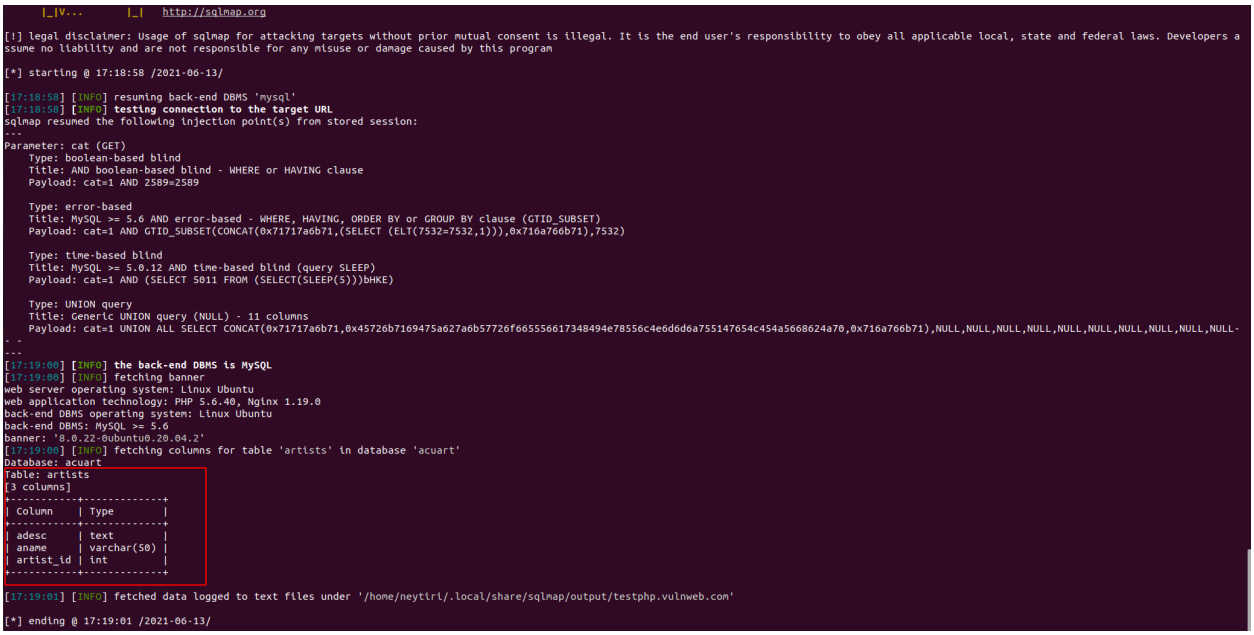
List information about Tables present in a particular Database:

Run: `python3 sqlmap.py -u http://testphp.vulnweb.com/listproducts.php?cat=1 --batch --banner -D acuart --tables`



List information about the columns of a particular table

Run: `sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 –batch –banner -D acuart -T artists --columns`



Severity of SQL Injection:

The severity of SQL Injection varies from P2 to P1 depending on what data is being exposed and if are able to get the shell or not.

Impacts of SQL Injection:

- **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.

SQL Injection Prevention:

- Most prevention example ⇒ `using parameterized queries`
  - also known as prepared statements

- Mostly **parametrerized queries used ⇒ instead of ⇒ string concatenation within the query. ⇒ which is good**

```
String query = "SELECT * FROM products WHERE category = '"+ input + "'";

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(query);
```

- Above code is vulnerable to SQL injection ⇒ because ⇒ the **user input ⇒ is concatenated directly into the query**
- To prevent this code ⇒

```
PreparedStatement statement = connection.prepareStatement("SELECT * FROM products WHERE category = ?");

statement.setString(1, input);

ResultSet resultSet = statement.executeQuery();
```

- **Parameterized Queries** ⇒
  - can be used for any situation where untrusted input appears as data within the query
    - Including the WHERE clause and values in an INSERT or UPDATE statement.
  - Can't be used to handle untrusted input other parts of the query, such as table or column names, or the ORDER BY clause.
- Application functionality that places untrusted data into those parts of the query will need to take a different approach,
  - such as white-listing permitted input values, or using different logic to deliver the required behavior.

> 💡 For a parameterized query to be effective in preventing SQL injection,
>
> the string that is used in the query must always be a **hard-coded constant, and must never contain any variable data from any origin.**

SQL injection cheat sheet

SQL injection cheat sheet:https://portswigger.net/web-security/sql-injection/cheat-sheet

References:

PortSwigger :https://portswigger.net/web-security/sql-injection

OWASP : https://owasp.org/www-community/attacks/SQL_Injection

🧩 Lab Documentation