**SQL Injection**

SQL (Structured Query Language) Injection, mostly referred to as SQLi, is an attack on a web application database server that causes malicious queries to be executed. When a web application communicates with a database using input from a user that hasn't been properly validated, there runs the potential of an attacker being able to steal, delete or alter private and customer data and also attack the web applications authentication methods to private or customer areas. This is why as well as SQLi being one of the oldest web application vulnerabilities, it also can be the most damaging.

**Contents**

[**In-Band SQL Injection** 2](#_Toc91890657)

[**Error-Based SQL Injection** 2](#_Toc91890658)

[**Union-Based SQL Injection** 2](#_Toc91890659)

[**Boolean Based** 2](#_Toc91890660)

[**Useful Commands** 2](#_Toc91890661)

[**Blind SQL Injection** 2](#_Toc91890662)

[**How to detect** 2](#_Toc91890663)

[**Time Based SQL Injection** 2](#_Toc91890664)

[**Demo** 3](#_Toc91890665)

### **In-Band SQL Injection**

In-Band SQL Injection is the easiest type to detect and exploit; In-Band just refers to the same method of communication being used to exploit the vulnerability and also receive the results, for example, discovering an SQL Injection vulnerability on a website page and then being able to extract data from the database to the same page.

### **Error-Based SQL Injection**

This type of SQL Injection is the most useful for easily obtaining information about the database structure as error messages from the database are printed directly to the browser screen. This can often be used to enumerate a whole database.

### **Union-Based SQL Injection**

This type of Injection utilises the SQL UNION operator alongside a SELECT statement to return additional results to the page. This method is the most common way of extracting large amounts of data via an SQL Injection vulnerability.

### **Boolean Based**

Boolean based SQL Injection refers to the response we receive back from our injection attempts which could be a true/false, yes/no, on/off, 1/0 or any response which can only ever have two outcomes. That outcome confirms to us that our SQL Injection payload was either successful or not. On the first inspection, you may feel like this limited response can't provide much information.

### **Useful Commands**

1. List tables in database: SELECT \* FROM information\_schema.tables

### **Blind SQL Injection**

This means that the application does not return the results of the SQL query or the details of any database errors within its responses.

### **How to detect**

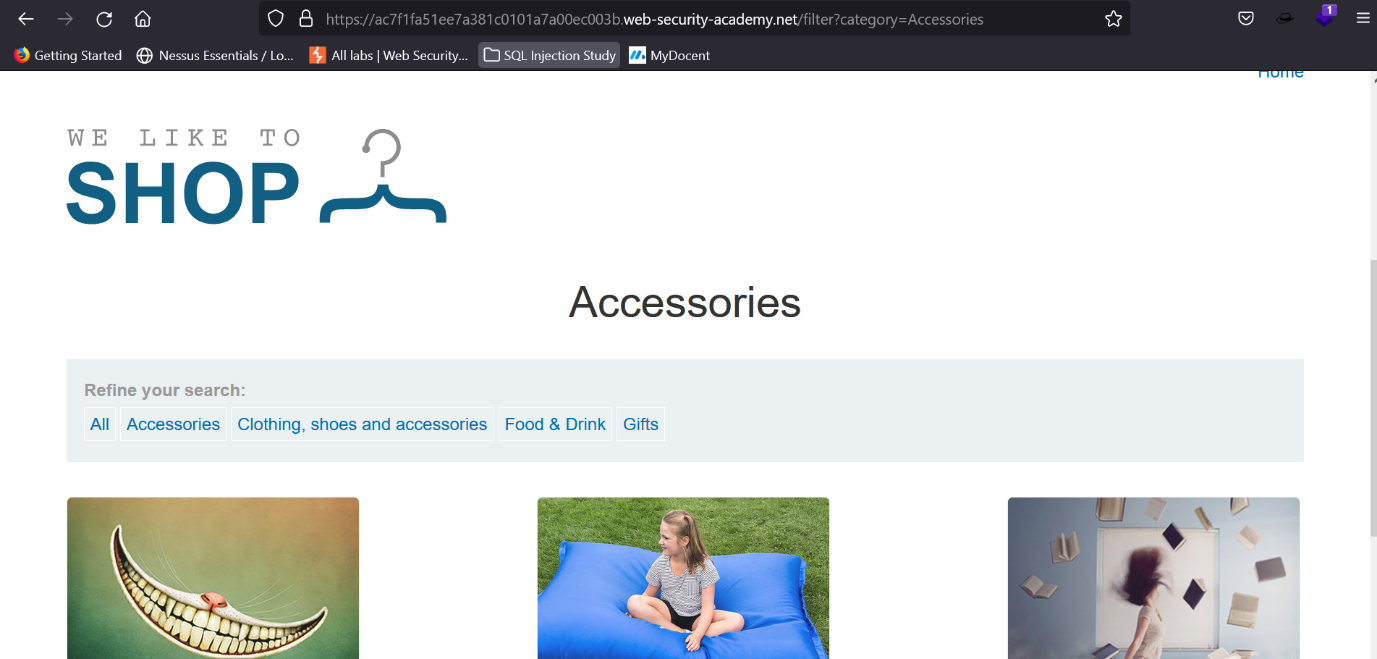
1. Using ‘
2. Using common SQL syntax like ASCII (97)
3. Using ‘ OR 1=1--
4. Using ‘; waitfor delay (‘0:0:20’)--

### **Time Based SQL Injection**

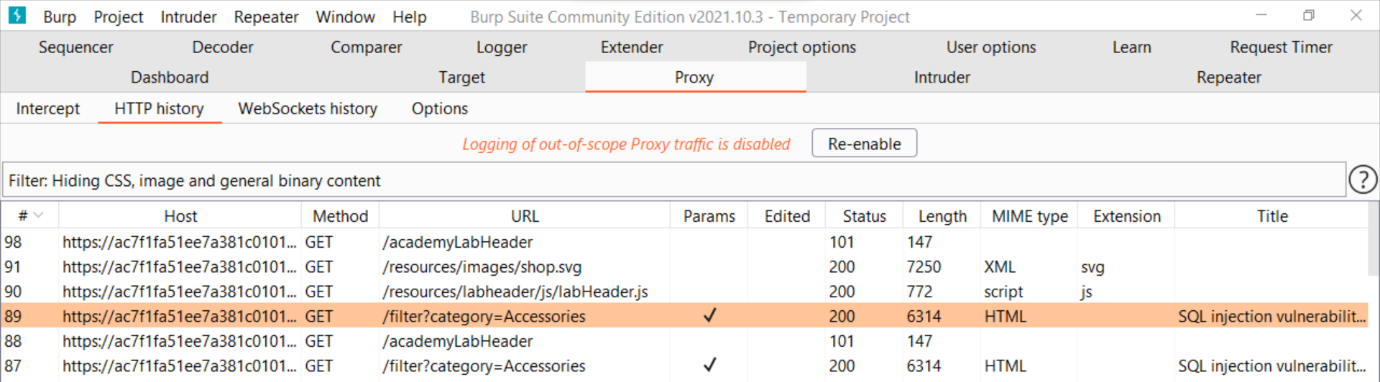
A time-based blind SQL Injection is very similar to the above Boolean based, in that the same requests are sent, but there is no visual indicator of your queries being wrong or right this time. Instead, your indicator of a correct query is based on the time the query takes to complete. This time delay is introduced by using built-in methods such as SLEEP(x) alongside the UNION statement. The SLEEP() method will only ever get executed upon a successful UNION SELECT statement.

### **Demo**

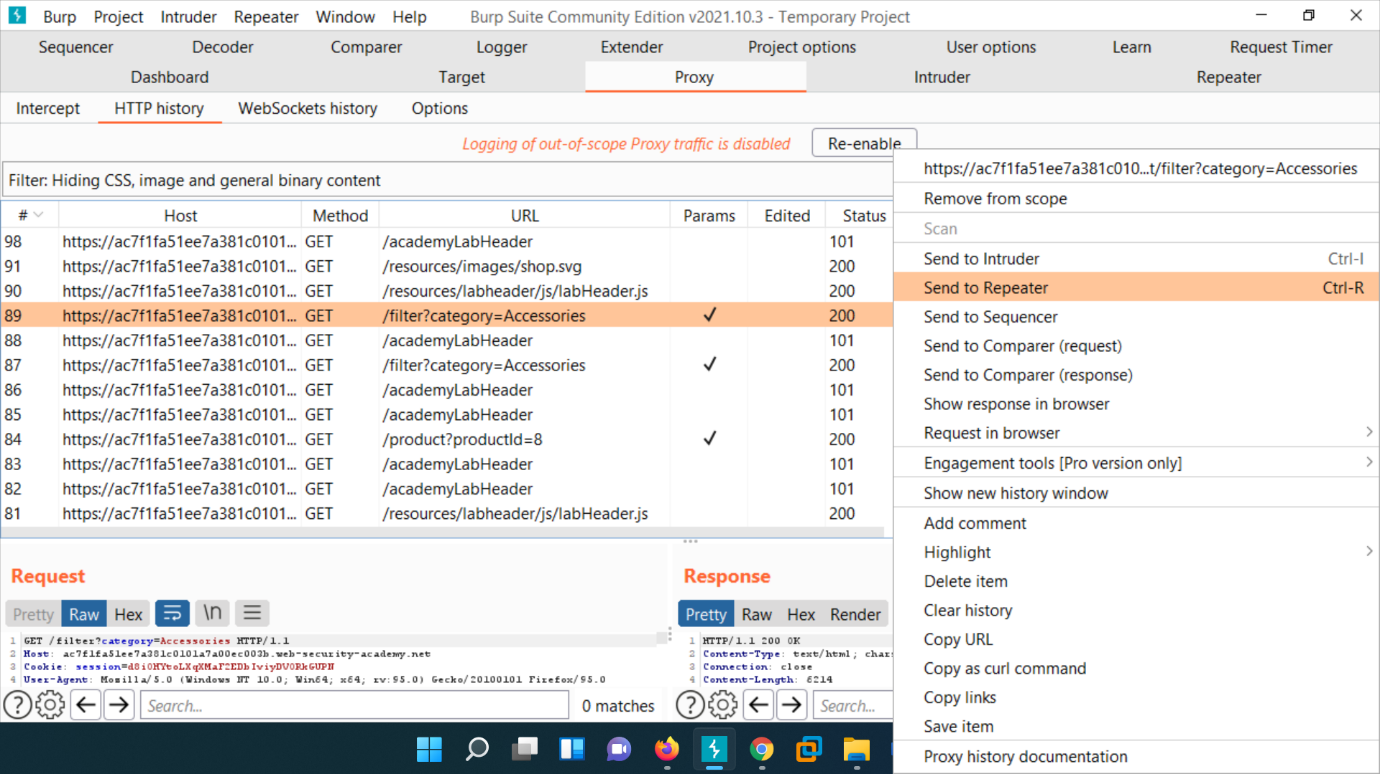
Target is category parameter mentioned in URL of following application:



Capture Request in burpsuite

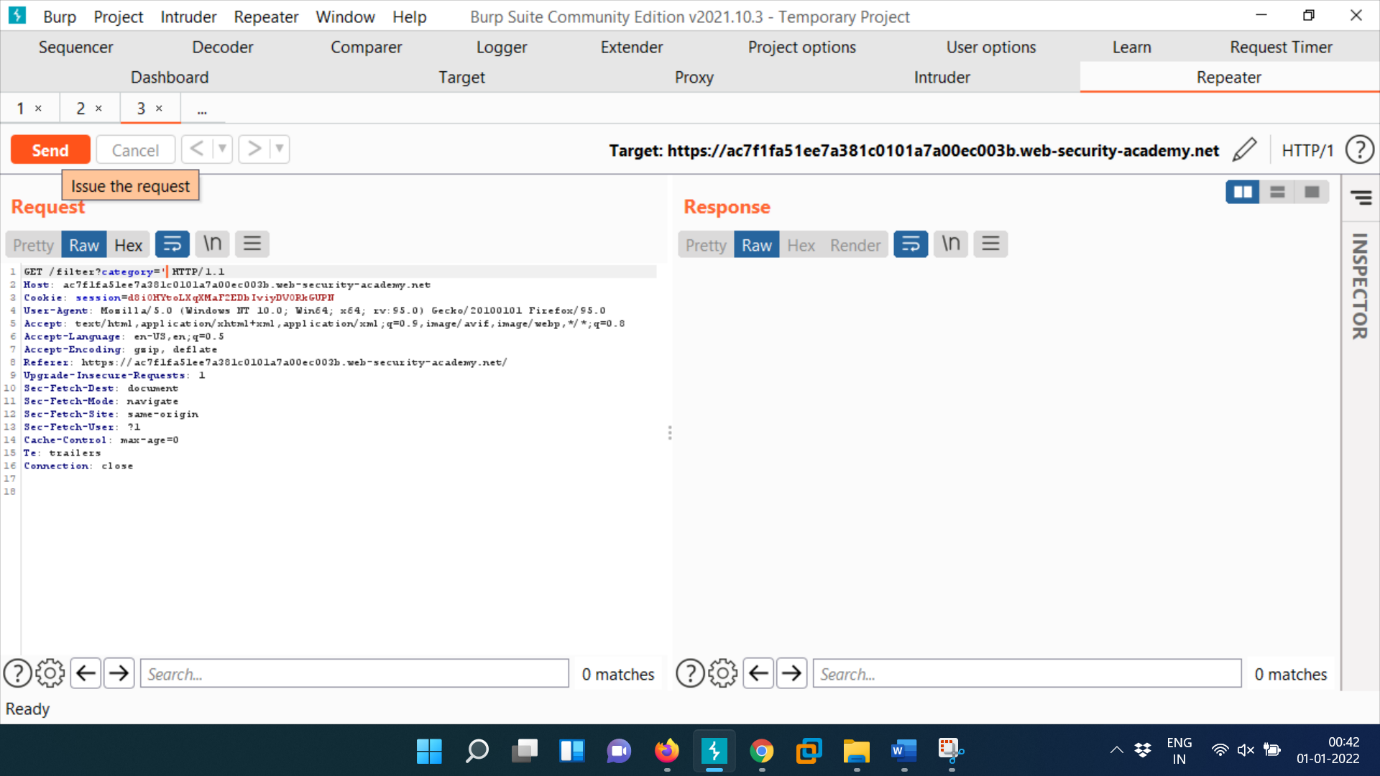


Send request to repeater



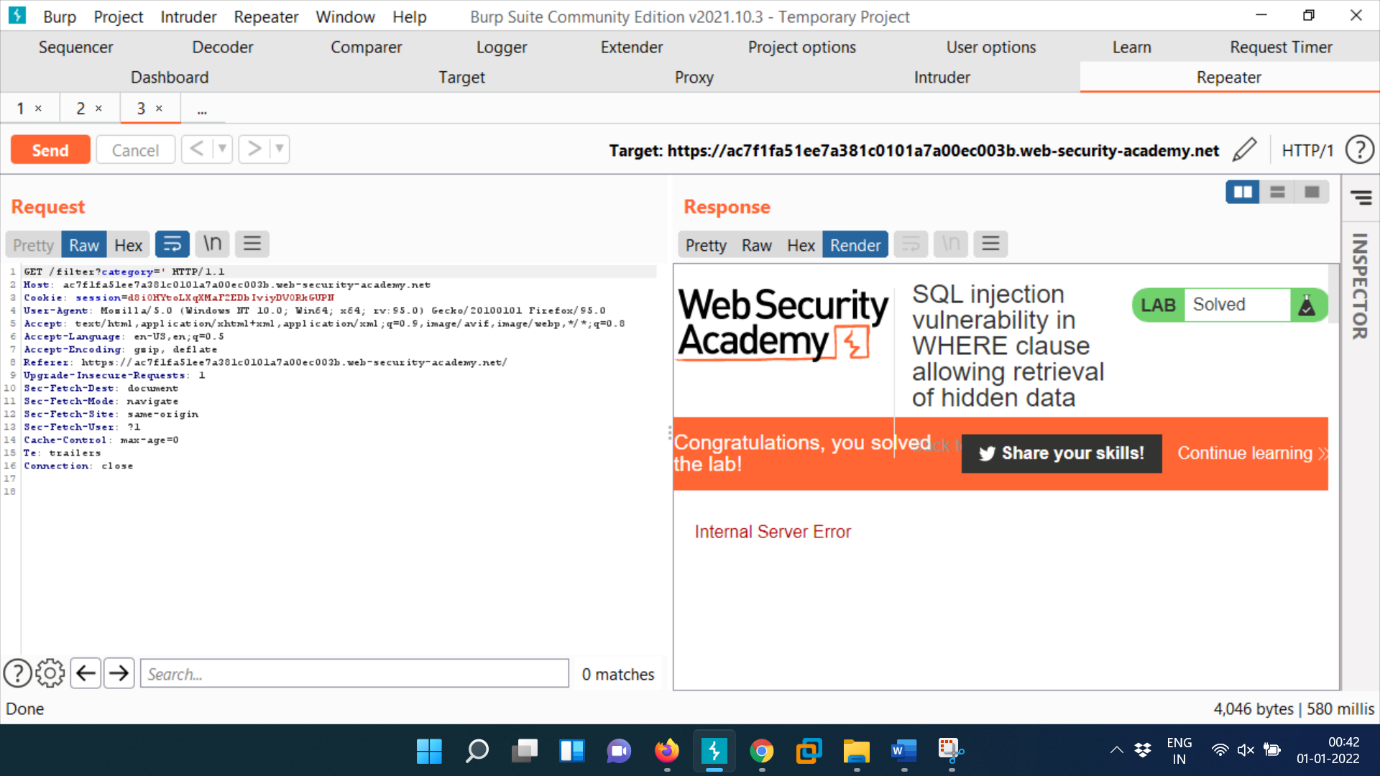


Change the value of category parameter to ‘:



And click send

You will receive a response on your right-hand side. Click on render to see browser display



Our payload (‘) got executed and lab is solved.