**CYBER SECURITY-MINOR PROJECT**

**TASK 1**

Reflected XSS into HTML context with nothing encoded

This lab contains a simple reflected cross-site scripting vulnerability in the search

functionality.

To solve the lab, perform a cross-site scripting attack that calls the alert function.

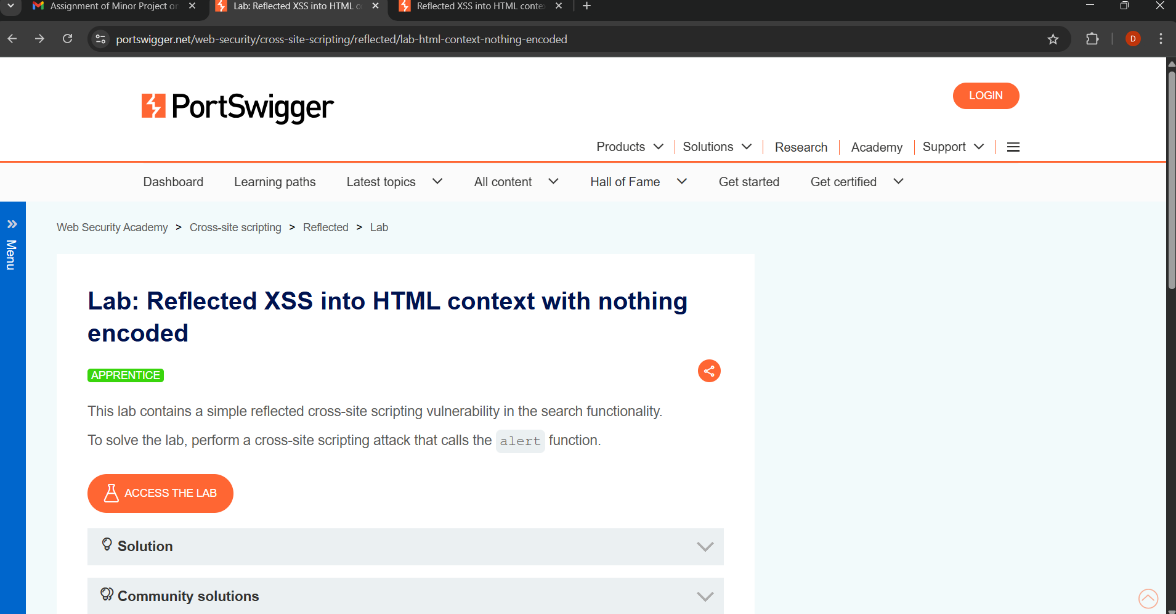
**PROJECT DETAILS:**

**Objective:**

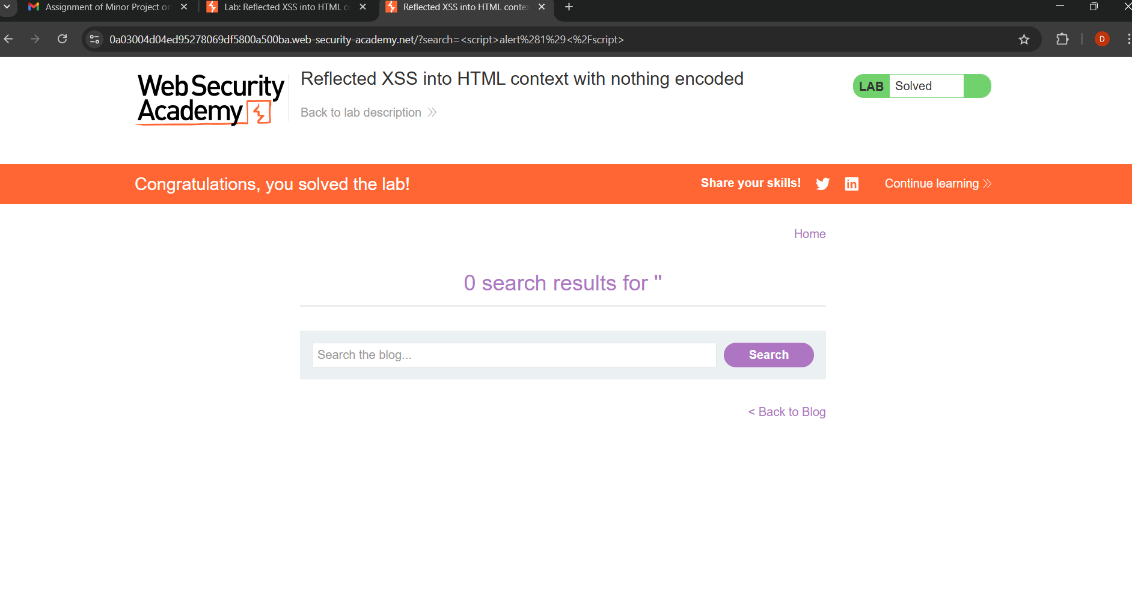
The objective of this project is to identify and exploit a reflected Cross-Site Scripting (XSS) vulnerability within a web application’s search functionality. The goal is to demonstrate how unvalidated user input can be used to inject and execute arbitrary JavaScript code—specifically, by triggering a browser alert()—thus highlighting the importance of input sanitization and output encoding in secure web development.

**LAB USE FOR TASK 1:**

[**https://portswigger.net/web-security/cross-site-scripting/reflected/lab-html-context-nothing-encoded**](https://portswigger.net/web-security/cross-site-scripting/reflected/lab-html-context-nothing-encoded)

****

**OUTPUT :**

****

**SUMMARY:**

This project focuses on performing a reflected Cross-Site Scripting (XSS) attack where user input is inserted into the HTML page without any encoding. In this lab, the vulnerable web application reflects the search parameter directly into the page’s HTML content, allowing attackers to inject JavaScript code such as <script>alert(1)</script>. When the payload is executed in the victim’s browser, it demonstrates how XSS can be used to execute arbitrary scripts, which could potentially lead to session hijacking, credential theft, or other malicious actions. The project successfully demonstrates the risks associated with insecure input handling and the importance of implementing proper security mechanisms in web applications.

**TASK 2:**

SQL injection vulnerability allowing login bypass

This lab contains a SQL injection vulnerability in the login function.

To solve the lab, perform a SQL injection attack that logs in to the application as the

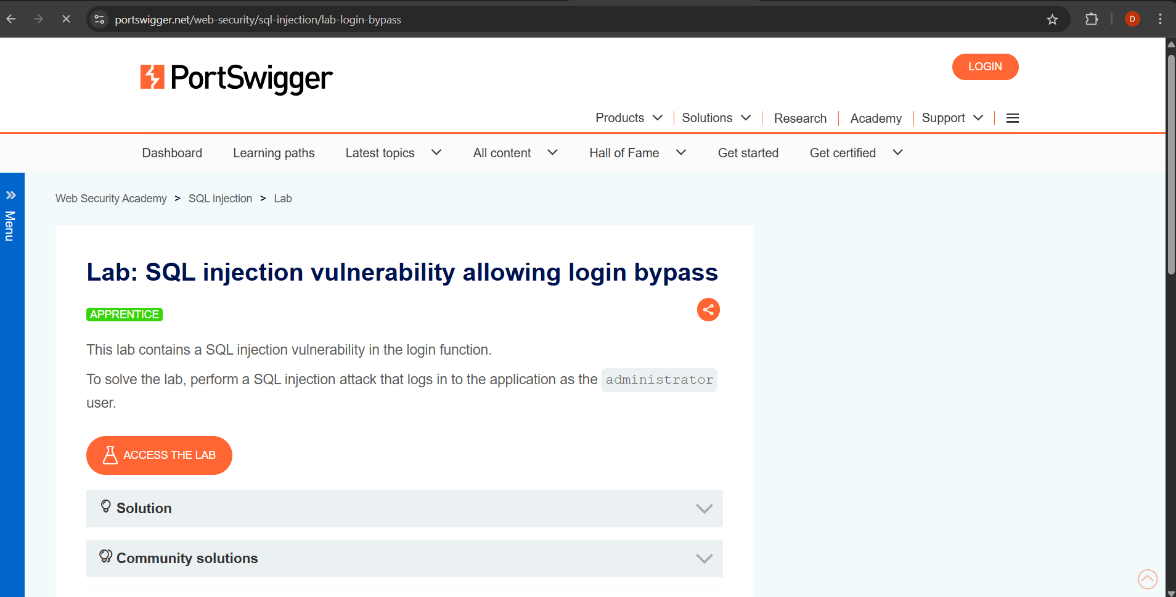
administrator user.

**OBJECTIVE:**

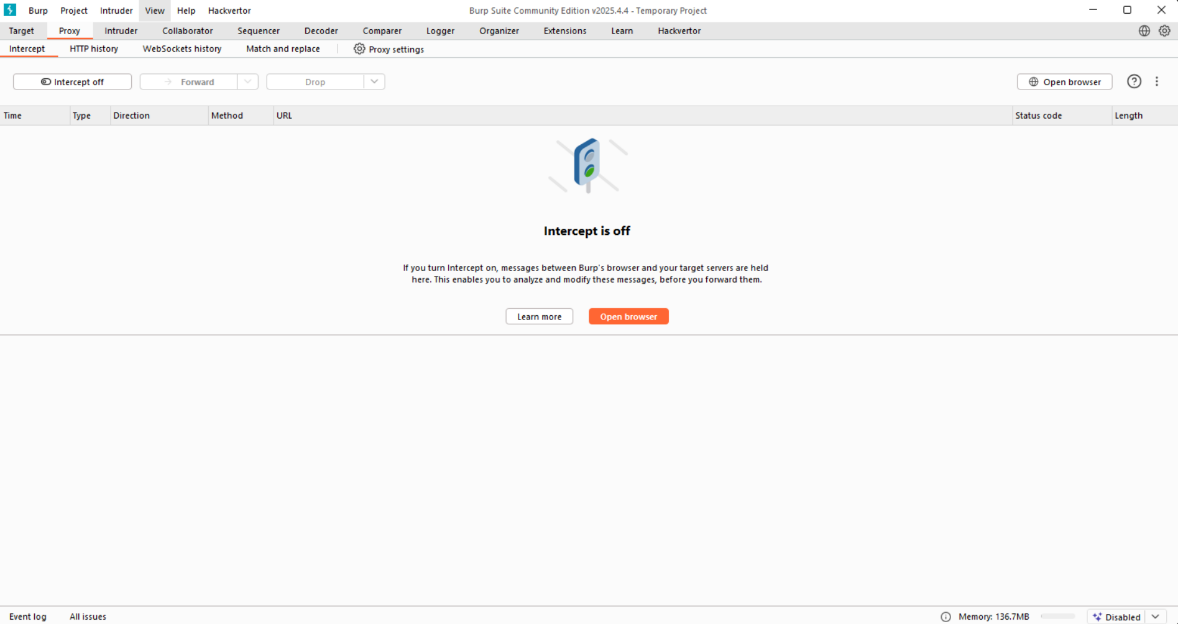
The objective of this project is to identify and exploit a SQL injection vulnerability in a web application's login functionality in order to bypass authentication and gain unauthorized access to an administrator account. The goal is to demonstrate how improperly sanitized input can be used to manipulate backend SQL queries, highlighting the critical need for secure coding practices such as input validation and the use of parameterized queries.

**LAB USED FOR TASK 2:**

[**https://portswigger.net/web-security/sql-injection/lab-login-bypass**](https://portswigger.net/web-security/sql-injection/lab-login-bypass)

****

Downloaded burp suite community edition

****

**STEPS INVOLVED:**

**Step 1: Setup Burp Suite and Browser**

1. Open **Burp Suite**.
2. Start a new **temporary project** or existing one.
3. Go to **Proxy > Intercept > Intercept is on**.
4. Configure your browser to use **Burp’s proxy** (default is 127.0.0.1:8080).
5. Visit the target **login page** (e.g., from PortSwigger lab).

**🔹 Step 2: Intercept the Login Request**

1. In your browser, enter:
   * **Username: user name**
   * **Password**: password
2. Click **Login**.
3. The request will be intercepted by Burp.
4. Go to **HTTP history (Proxy > HTTP history)** to review the request.

**🔹 Step 3: Modify the Request**

1. Right-click on the intercepted login request → **Send to Repeater**.
2. In **Repeater**, locate the POST parameters:

Example:

POST /login HTTP/1.1

Host: vulnerable-website.com

Content-Type: application/x-www-form-urlencoded

username=test&password=test

1. Modify the parameters like this:

username=administrator'--&password=anything

So the body becomes,

username=administrator'--&password=abc

**🔹 Step 4: Send the Modified Request**

1. Click **Send** in Repeater.
2. Observe the **response**.
   * If successful, you’ll see something like a redirect or a “Welcome administrator” message.

**🔹 Step 5: Verify Login in Browser**

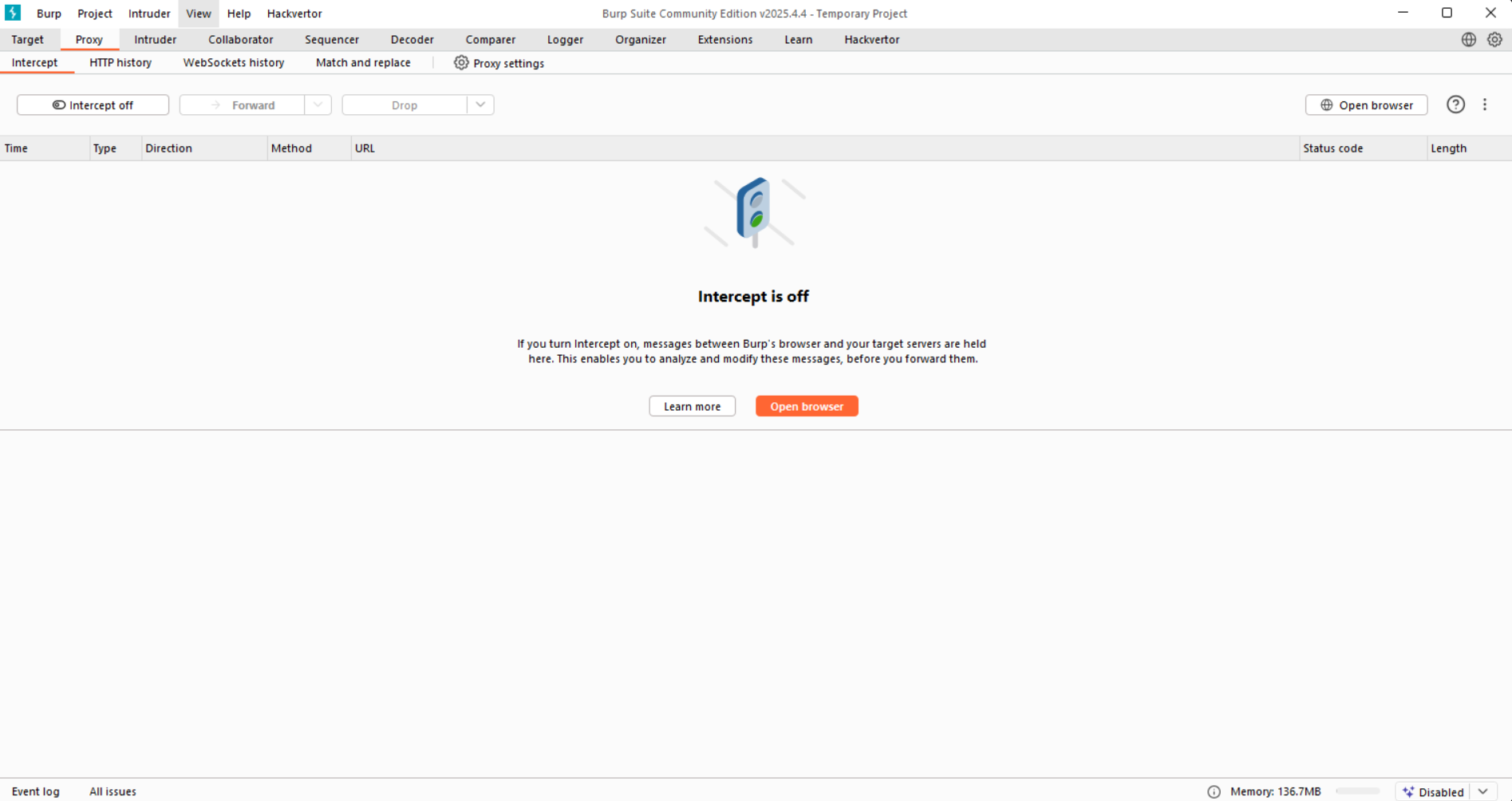
1. Copy any **cookies** from the Repeater response if needed.
2. Paste them into your browser (using dev tools or an extension like "EditThisCookie").
3. Reload the site — you should now be logged in as the administrator.

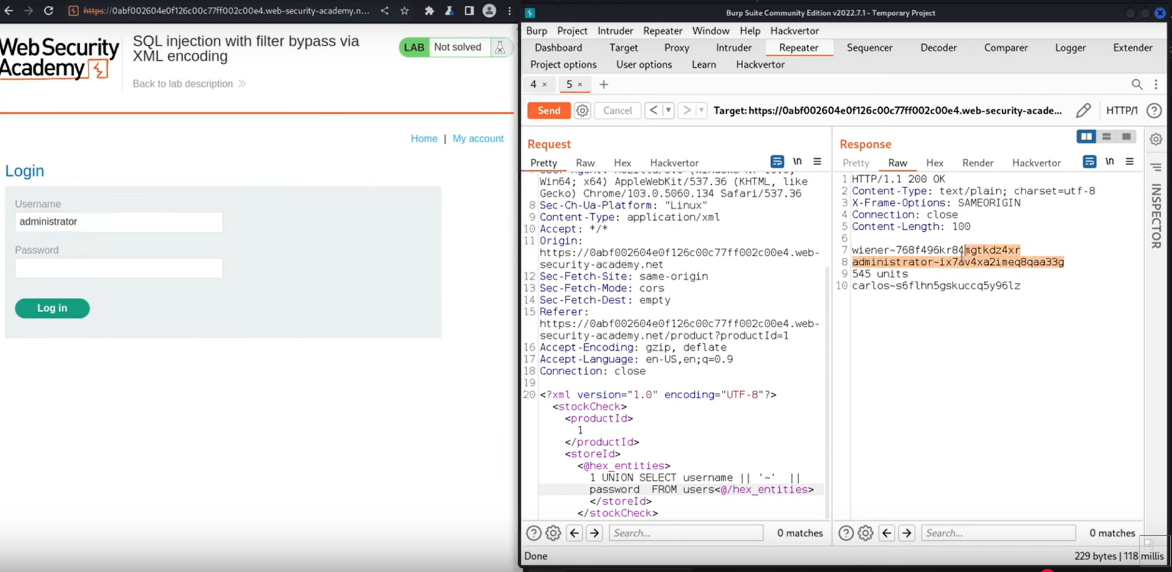
**Example SQL Injection Payload**

administrator'--

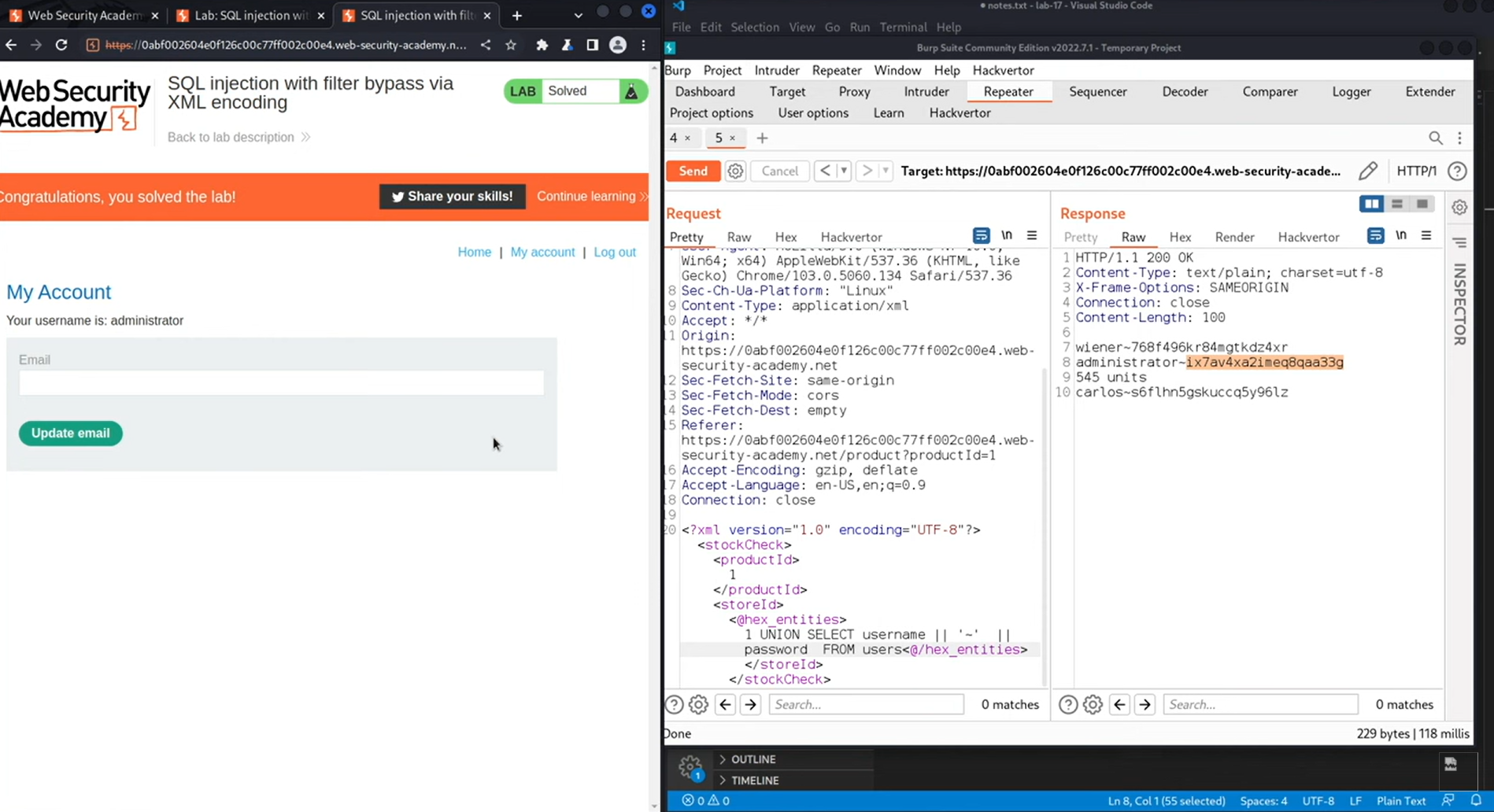
**Explanation:**

* Closes the username string
* Uses -- to comment out the password condition
* Bypasses authentication

****

****

**TASK 2 OUTPUT :**

****

**SUMMARY:**

This project focuses on exploiting a SQL injection vulnerability in a web application’s login form. The vulnerability arises from the application's failure to properly sanitize user input before incorporating it into a SQL query. By injecting crafted SQL syntax—specifically using the payload administrator' --—the attacker is able to alter the query logic and bypass password verification. As a result, the attacker successfully logs in as the administrator without valid credentials. This exercise demonstrates how SQL injection attacks can lead to unauthorized access, data breaches, and full system compromise, reinforcing the importance of secure database query handling and input validation.