

Setting security level: We can set security level as shown in below image.

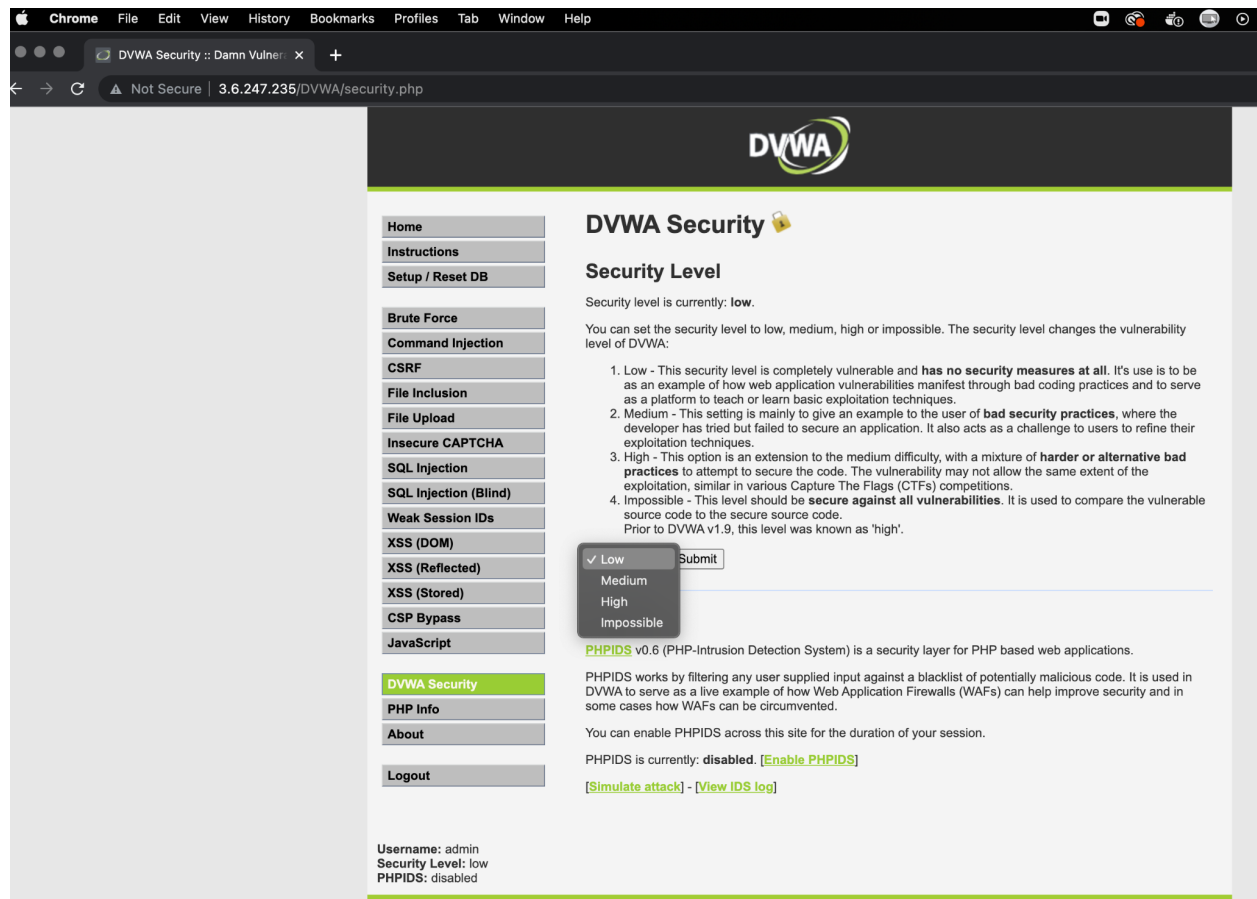


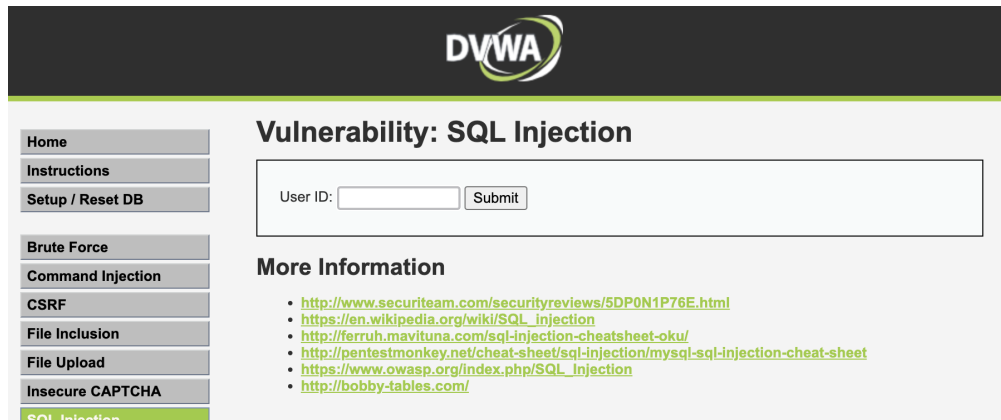
Fig 1. Security level

Que 1. Describe the SQLi attack you used, how did you cause the user table to be dumped? What was the input string you used?

Ans.

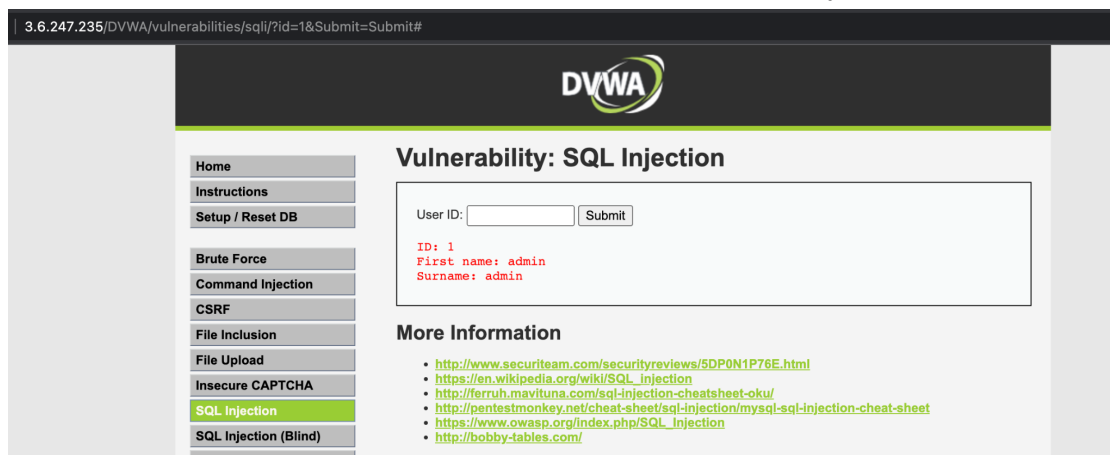
Step 1: Go to SQL Injection

As can be seen in Fig. 1, the security level is set to “low”. On the left panel menu options, we need to select SQL Injection. We can see following image:

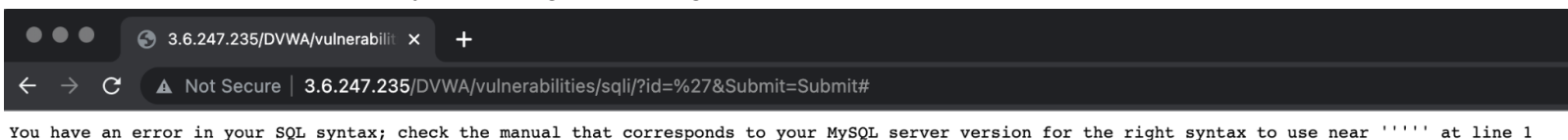


Step 2: Identify database

When I entered a user id, I could see details of the user correctly as follows for ID 1:



I was not getting any data when I entered an alphabet or string. This led me to the assumption that there is a “where” type of clause that accepts string as an id. I tried to enter ‘ as user id to mimic erroneous syntax and got following error:



Now, I know that the database is MySQL.

Step 3: Query format identification

From step 2, we are sure of the syntax of the SQL query to be somewhat as follows:

SELECT first_name, last_name FROM User WHERE ID='\$'

\$ is any ID

I now entered the string as **1' OR '1'>0**. Therefore, the select query will be like following:

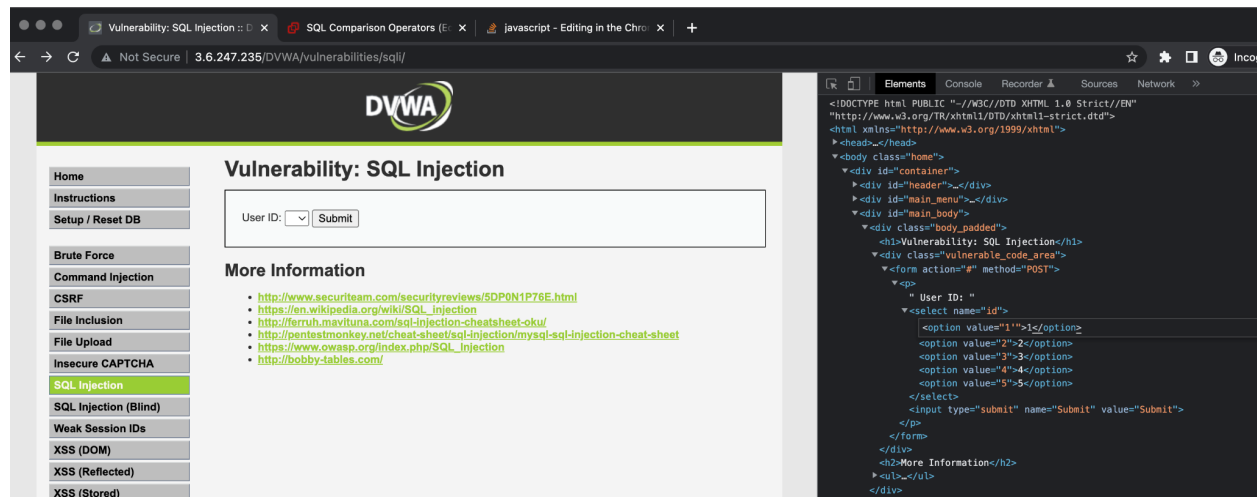
SELECT first_name, last_name FROM User WHERE ID='1' OR '1'>0'

We reached a conclusion that low level of security permitted users to send any data from UI and simply appended to the query in the backend. This resulted in an extremely vulnerable app prone to sql injection attacks.

Que 2. If you switch the security level in DVWA to “Medium”, does the SQLi attack still work?

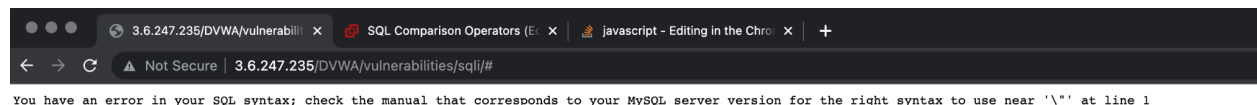
Ans. After changing the security level to “Medium”, I noticed that I was unable to perform SQLi attack. Since the first level of security is drop down instead of text box, it limits a user:

Then, we tried to update the HTML page and change `<select>` value to a simple `1`” as follows:



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface for the SQL Injection vulnerability. The page title is "Vulnerability: SQL Injection". On the left, there is a sidebar with navigation links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (selected), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), and XSS (Stored). The main content area has a form with a "User ID:" label and a dropdown menu, and a "Submit" button. Below the form, there is a "More Information" section with several links. On the right side, the browser's developer tools are open, showing the "Elements" panel. The HTML structure is visible, including the header, main menu, and the vulnerable code area. The vulnerable code area contains a form with a "User ID:" label and a `<select name='id'>` dropdown menu with five options: "1", "2", "3", "4", and "5". The "Submit" button is an `<input type='submit' name='Submit' value='Submit'>`.

However, doing so resulted in the following error from the backend. This leads us to a conclusion that the query no longer used simple appending.



The screenshot shows an error message from the backend. The message is: "You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1". The error message is displayed in a dark background with white text.