Ertan Can Güner

60610

**Comp430 Homework 4**

**Report**

**Question 2**

**Part 1**

For challenge 1, when we look at the source code (Fig. 1) we can see that there is there is no sanitation or any security mechanism to check the username and password

field taken from the user. It is vulnerable to basic escape payloads from the user

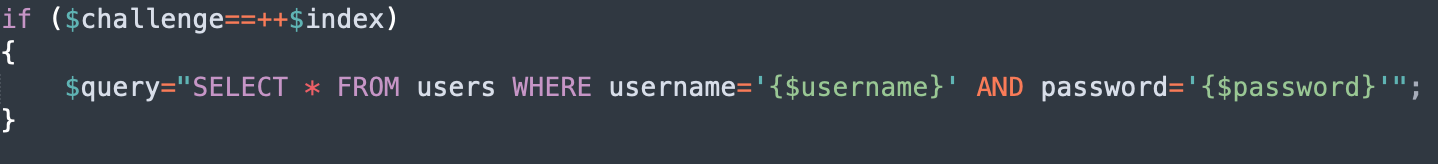
meaning that we can escape the sql query that was formed by the server by entering an

apostrophe in the password field and following with a logical statement such as

“or 1 == 1 “ or “or ‘a’ == ‘a’ “(any logical statement equivalent to true is sufficient). This

makes the query to evaluate to True and disregard whether the username and

password is present in the database, or they are correctly entered.

**Payload:** **username:** anything

**password:** ‘ or 1==1;--

Figure 1: Source Code of Challenge 1

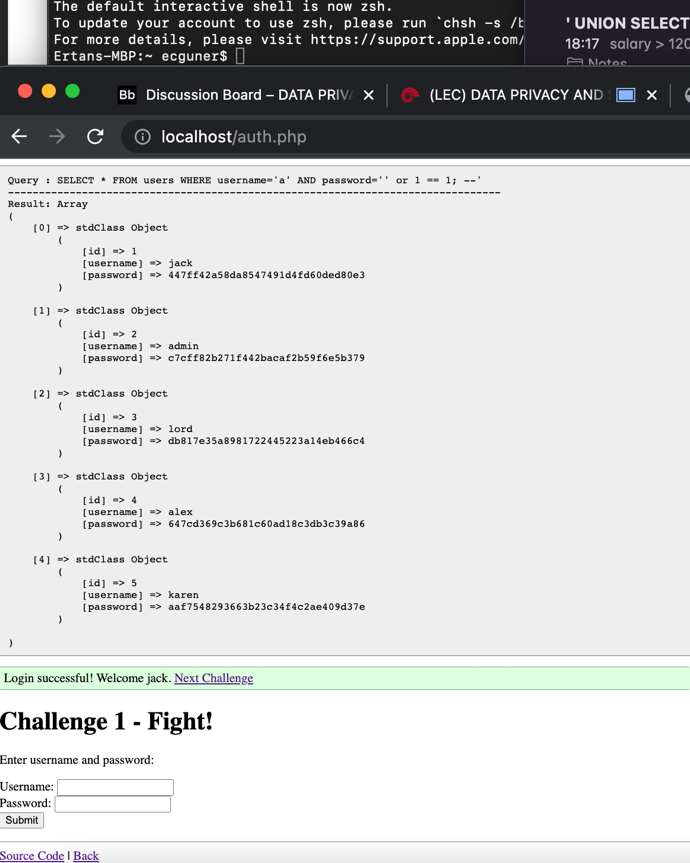


Figure 2: Payload Executed Bypassing Login

**Part 2**

For challenge 2, when we examine the source code (Fig. 3) we can see that some low level of sanitization is applied to the input taken from the user. The payload can’t

have escape characters such as apostrophe or quotes in it because the server escapes

those character by putting ‘\’ in front of them to escape. But there is a missed escape

that can be used in the payload, which is the escape character itself. We can escape the

apostrophe that is used in the sql query. The payload uses both the username and

password field to make it work. First, we should escape the apostrophe in the username by providing \’ (backslash and apostrophe which we escape the escape character that

was used by the server making the value in the username field to ’ \\'' AND password=’

and then we use the same principle in part 1 to evaluate the expression to always true

by providing ‘or 1==1;--’ to the password field.

Graphical user interface, text

Description automatically generated **Payload: username:** ‘\ **password:** or 1==1;--

Graphical user interface, text, application

Description automatically generated

Figure 4: Payload Executed Bypassing Login

Figure 3: Source Code of Challenge 2

**Part 3**

For challenge 3, when we examine the source code (Fig. 6), we can see that the

server uses parametrized input from the server, which means it takes parameters from

the URL and appends them to the at the end of the query. When we enter only

username and password we can see that “ORDER BY 1” is added at the end of the query

if “ord” parameter in the URL is empty (Fig. 7). This means that we change the value of

“ORDER BY 1” if we enter some value for “ord” parameter in the URL. First, we follow

same principle in Part 1 and enter this URL <http://localhost/auth.php?challenge=3&ord=or%201==1;--> and press enter (%20 is the Unicode character corresponds to space we can also change special character such as “=” or “;” to their corresponding Unicode values and the URL will still work). Then we enter arbitrary values to username and password field and submit it to bypass login.

**Payload: URL:** http://localhost/auth.php?challenge=3&ord=or 1==1;-- **username:** anything **password:** anything

Text

Description automatically generated

Figure 6: Source Code of Challenge 3

Graphical user interface, text, application, email

Description automatically generated

Figure 7: No Value to "ord" Parameter in the URL

Graphical user interface, text, application

Description automatically generated

Figure 8: Payload Executed to Bypass Login

**Part 4**

For the last challenge, when we examine the source code (Fig. 9) we can see that there is no sanitation is applied to the input from the user and it uses the parameters

from the URL. Just like challenge 3 we can enter our payload by manipulating these

parameters. It retrieves two parameters from the URL first one is “id” and the second

one is “username. We cannot do anything with the “id” parameter because it is integer

and we need a string value to inject our malicious code so the “username” parameter is

better suited for the attack surface. We can use the same intuition in challenge 1 and

escape the query by starting the payload with an apostrophe. Then to retrieve

information that we are not supposed to see we use a “UNION” keyword followed by

“SELECT null,id,null,role,salary,null,null from salaries where salary > 12000 and

age > 40;--” which displays the employee information (just their id, role and

salary) that we want (salary greater that 12000 and age greater than 40 in this case). This payload means that merge the results of the first query that was in the server with null,id,null,role,salary,null,null column names (null values indicate that we will not use those values and we have to enter null because both sides of UNION have to equal to each other in terms of column counts) from the table “salaries” which have greater values than 12000 and 40 in salary and age columns respectively.

**Payload: URL:**

<http://localhost/union.php?username=admin%27%20UNION%20SELECT%20null,id,null,role,salary,null,null%20from%20salaries%20where%20salary%20%3E%2012000%20and%20age%20%3E%2040;-->

**Note:** admin is not needed we can start the payload with and apostrophe. Also

the character starting with “%” sign is the Unicode character to special characters

such as space and ‘,’.

Text

Description automatically generated

Figure 9: Source Code of the Last Challenge

Graphical user interface, text, application

Description automatically generated

Figure 10: Payload Executed to Show the Data that Satisfies the Constraints in the Payload.