Level 6

You are presented with the following code:

```
<?php
    $flag = "CSC{fake_flag}";

    function hashing_function($password) {
        return hash('md5', hash('md5', $password));
    }

    $username = $_POST['username'];
    $password = $_POST['password'];

    if ($username != null && $password != null) {
        if ($username == "admin" && hashing_function($password) == "0e910057037170584158391016456192") {
            echo "<h3 style='color: lime;'>ACCESS GRANTED!</h3><pre>" . $flag . "</pre>";
        } else {
            echo "<h3 style='color: darkred;'>ACCESS DENIED.</h3>";
        }
    }
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Login</title>
</head>
<body style="background-color: grey; font-family: arial; padding: 25px;">
    <h2 style="width: 500px">Administrator Login</h2>
    <form method='POST' action=''>
        <input name='username' type='text' placeholder='Username'/><br>
        <input name='password' type='password' placeholder='Password'/><br>
        <input type='submit' value='Login'/>
    </form>
</body>
</html>
```

This code creates a website with an Administrator login.

Looking at the following PHP code:

```
if ($username == "admin" && hashing_function($password) == "0e910057037170584158391016456192")
```

We can tell that the valid username is `admin`. We can also see what seems to be a password hash.

The password appears to be encrypted using the function `hashing_function` which is defined above. So the username must be `admin` and the password must be some string, that when ran through our hashing function, the output hash is `0e910057037170584158391016456192`.

Looking at the definition of `hashing_function` we see the following:

```
function hashing_function($password) {
    return hash('md5', hash('md5', $password));
}
```

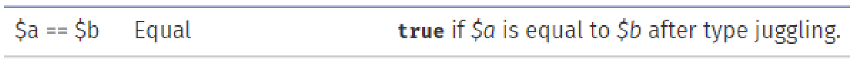
We can see that the hashing function is simply double MD5. This means that the password is first encrypted with the MD5 hashing algorithm, then the resulting hash is encrypted again in MD5.

Looking a bit further into the code, we can see that upon getting the correct username and password, the flag is given to us.

```
if ($username == "admin" && hashing_function($password) == "0e91005703717058415839101645619 2") {
    echo "<h3 style='color: lime;'>ACCESS GRANTED!</h3><pre>" . $flag . "</pre>";
}
```

Solution:

Notice how in the IF statement for checking the username and password, we use the `==` operator. If you do a bit of research on the `==` operator, we can see the following:

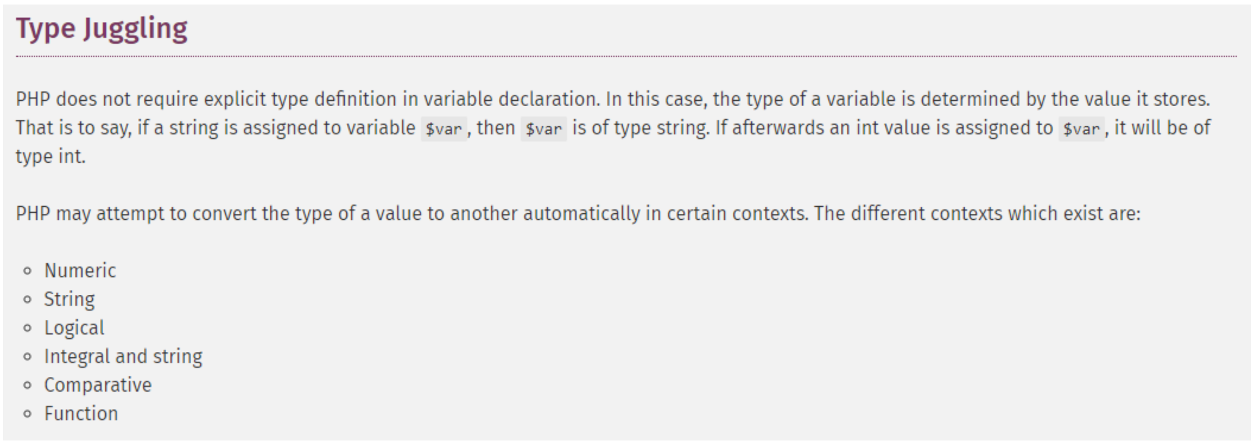


$a == $b   Equal   **true** if *$a* is equal to *$b* after type juggling.

Source: https://www.php.net/manual/en/language.operators.comparison.php

Notice the following text: "after type juggling".

If we do more research on type juggling in PHP, we can see the following:



Type Juggling

PHP does not require explicit type definition in variable declaration. In this case, the type of a variable is determined by the value it stores. That is to say, if a string is assigned to variable `$var`, then `$var` is of type string. If afterwards an int value is assigned to `$var`, it will be of type int.

PHP may attempt to convert the type of a value to another automatically in certain contexts. The different contexts which exist are:

- Numeric
- String
- Logical
- Integral and string
- Comparative
- Function

Source: https://www.php.net/manual/en/language.types.type-juggling.php

From what we can see, using the `==` operator will result in PHP attempting to convert the type of both values being compared. This is useful for us, because the password hash found in the PHP code is conveniently "`0e910057037170584158391016456192`", which code be interpreted by PHP as the NUMBER 0e910057037170584158391016456192 which is the scientific notation for

$0 \times 10^{910057037170584158391016456192}$ which equals 0.

Therefore, it is safe to say that in the following comparison:

```
hashing_function($password) == "0e910057037170584158391016456192"
```

the result of the comparison will be true if we know any password that when ran through the `hashing_function`, produces a hash starting with `0e` and then followed by ONLY NUMBERS.

For example:

> Let's say we find out that the password "123" produces a hash which starts with `0e` and is followed by only numbers, such as `0e838547058483980038792095699017` ("123" does not produce this hash, this is just for the sake of the example).
>
> We can clearly see that `0e838547058483980038792095699017` and `0e910057037170584158391016456192` are not the same hashes. But because we used the `==` operator, when PHP runs the comparison:
>
> `"0e838547058483980038792095699017" == "0e910057037170584158391016456192"`
>
> It will output true. This is because type juggling tries to convert both hashes to different datatypes, such as integers. When both hashes are converted to integers, you get the following comparison:
>
> `0e838547058483980038792095699017 == 0e910057037170584158391016456192`
>
> which is gets simplified to `0 == 0` since both hashes represent the value 0 in scientific notation ($0 \times 10^{838547058483980038792095699017} = 0 \times 10^{910057037170584158391016456192} = 0$.)

So our goal is now to find a password that when ran through `hashing_function` produces a hash starting with `0e` and followed by only numbers.

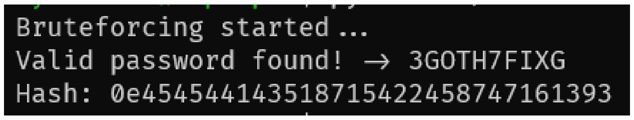
To do so, we will use BRUTEFORCE! Everyone's favorite method!

I wrote this script in Python which tries random generates passwords until one of them produces a hash starting with `0e` and followed by only number.

```
from hashlib import md5
import string
import random

def hashing_function(password): # replicate the hashing_function from PHP into Python
    first_md5 = md5(password.encode('utf-8')).hexdigest()
    second_md5 = md5(first_md5.encode('utf-8')).hexdigest()
    return second_md5

def generate_random_password(): # generates a random 10 character-long string
    length = 10
    # copied from https://stackoverflow.com/questions/2257441/random-string-generation-with-upper-case-letters-and-digits
    generated_password = ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in range(length))
    return generated_password

print('Bruteforcing started...')
while True:
    random_pass = generate_random_password() # generate a random password
    password_hash = hashing_function(random_pass) # get the hash of the random password by running it through our hashing_function
    if password_hash[:2] == '0e' and password_hash[2:].isdigit(): # if the hash starts with '0e' and is followed by ONLY NUMBERS
        # print the valid password and hash to the screen and break out the loop
        print(f'Valid password found! -> {random_pass}')
        print(f'Hash: {password_hash}')
        break
```

This will take a few minutes to complete. To increase your luck, you could run this script in multiple terminal windows.

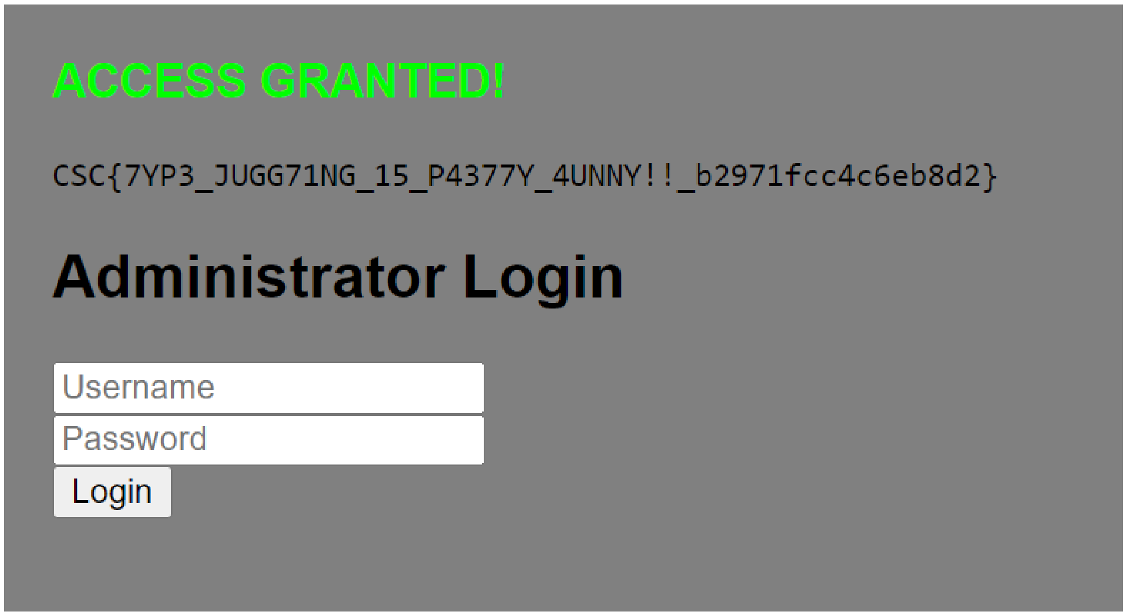
After a few minutes, I found a valid password:



```
Bruteforcing started...
Valid password found! → 3GOTH7FIXG
Hash: 0e4545441435187154224587471613 93
```

The Python program found a string “3GOTH7FIXG” which produces a hash starting with `0e` and followed by only numbers!

We then navigate to our login screen, input `admin` as the username, and `3GOTH7FIXG` as the password and we are now logged in!



ACCESS GRANTED!

CSC{7YP3_JUGG71NG_15_P4377Y_4UNNY!!_b2971fcc4c6eb8d2}

Administrator Login