Level 4

We have the following code:

```
from flask import Flask, request
from codecs import encode

app = Flask(__name__)

# on url / the following HTML is rendered
@app.route('/')
def login():
    return '''
        <form action="/attempt-login" method="POST">
            <label>Username: <input type="text" name="username"></label><br>
            <label>Password: <input type="password" name="password"></label><br>
            <input type="submit" value="Login">
        </form>
    '''

# when a request to the url /attempt-login is sent, check if the inputted username and password are valid.
@app.route('/attempt-login', methods=['POST'])
def login_submit():
    username = request.form['username']
    password = request.form['password']

    password = encode(password, 'rot_13') # encodes the password in ROT13: https://en.wikipedia.org/wiki/ROT13

    if username == "geegee" and password == "supersecretpassword": # if the login is successful, give the user the flag.
        with open('flag.txt', mode='r') as flag:
            flag_value = flag.read()
            return f'''
                <div style="font-family: arial;">
                    <h1 style="color: green;">ACCESS GRANTED!</h1>
                    <p>Here is the flag:</p>
                    <pre>{flag_value}</pre>
                </div>
            '''
    else: # if the login is not successful, return "ACCESS DENIED"
        return "<h1 style='font-family: arial; color: red;'>ACCESS DENIED</h1>"

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=False)
```

This is the code for a web application containing a login page.

Looking at this line of code:

```
 if username == "geegee" and password == "supersecretpassword":
```

We can tell that at some point in the code, the `username` should equal "`geegee`" and `password` should equal "`supersecretpassword`".

Looking at this line of code prior to the IF statement above:

```
password = encode(password, 'rot_13')
```

This tells us that the inputted password is ran through a ROT13 cipher before being compared to the string "`supersecretpassword`" in the IF statement.

To cipher text using ROT13, you would offset each character by 13 places in the alphabet.
https://en.wikipedia.org/wiki/ROT13

So to sum it up, the program works like the following:

1. You input the username and password
2. The password is ran through a ROT13 cipher
3. If the username is "geegee" and the password (has been ROT13-ed) is "supersecretpassword", then authenticate the user. If not, do not authenticate the user.

We need to find a string that when ran through the ROT13 cipher, becomes "`supersecretpassword`".

Since the English alphabet has 26 letters, and 13 is half of 26, any string ran through the ROT13 cipher twice will not change.

In other words:

$\mathrm{ROT13}(\mathrm{ROT13}(x)) = x$

For all strings x.

Therefore, if we declare some string a as:
$\mathrm{a} = \mathrm{ROT13}(\text{"supersecretpassword"})$

We know that:

$\mathrm{ROT13}(a) = \mathrm{ROT13}(\mathrm{ROT13}(\text{"supersecretpassword"})) = \text{"supersecretpassword"}$

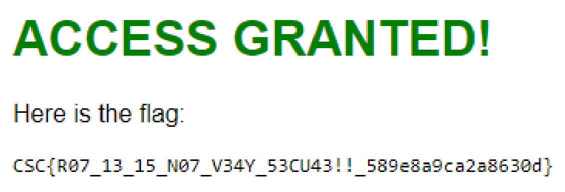
Therefore: $\mathrm{ROT13}(a) = \text{"supersecretpassword"}$

This means that the password the user inputs must be a for it to output "supersecretpassword" when ROT13-ed.

Using what we established above:
$\mathrm{a} = \mathrm{ROT13}(\text{"supersecretpassword"}) = \text{"fhcrefrpergcnffjbeq"}$

You can easily run strings through the ROT13 cipher using https://rot13.com.

Now that we know our username must be “geegee” (username is untouched) and output password must be “fhcrefrpergcnffjbeq”, we go ahead and try it:



ACCESS GRANTED!

Here is the flag:

```
CSC{R07_13_15_N07_V34Y_53CU43!!_589e8a9ca2a8630d}
```

IT WORKS!

I have to admit, the difficulty of this challenge does not fit a level 4 challenge but through writing and scrapping challenges, this ended up being level 4. :P

What to take away from this challenge (optional):

!! ROT13 IS NOT BY ANY MEANS A SAFE WAY TO STORE SENSITIVE DATA !!

Maybe some have you have heard the word “encryption” being thrown around here and there in the cybersecurity/cryptography world. (we are looking into organizing a cryptography workshop so stay tuned 👀)

Basically, an encryption is a computer algorithm which takes data and encodes it in some way. Some types of encryptions allow you to **decrypt** the encrypted data in order to recover the initial data. But some types of encryptions are impossible to decrypt. You can encrypt data, but once it’s encrypted, you mathematically cannot decrypt it.

One of the types of encryption which allows you to both encrypt and decrypt data, is called “Ciphers”.

ROT13 is a good example of a cipher. Caesar Cipher is another example of a cipher very similar to ROT13.

Some ciphers require a secret key used to encrypt and decrypt data. If a piece of data is encrypted with secret key a and you try to decrypt it with secret key b, you won’t be able to recover the data.

Some examples of ciphers who use a key are XOR Cipher and One-Time Pad Cipher.

The ciphers used today are much more complex. Here are a few in case you wanna go down a rabbit-hole (rabbit-holes are encouraged at this club):

- Advanced Encryption Standard (AES)

- RSA
- Data Encryption Standard (DES)
- Blowfish

Encryptions that can only be encrypted and can never be decrypted are called “Cryptographic Hash Functions”.

Cryptographic Hash Functions, or just hash functions are algorithm which takes data as an input, and outputs what we call a “hash”. A hash is a fixed-size unique string which acts as a digital representation of data.

For example, if we used the hash function “Message-Digest Algorithm (MD5)” :

If I encrypt a string a with MD5:

$$\text{MD5}(a) = b$$

The outputted hash from encrypting a, is b.

Since we used a hash function, if we only know b, we mathematically cannot calculate a.

Clearer example:

$$\text{MD5}(\text{"hello world"}) = \text{"5eb63bbbe01eeed093cb22bb8f5acdc3"}$$

The hash “5eb63bbbe01eeed093cb22bb8f5acdc3” is the digital representation of “hello world”. But if we were only given “5eb63bbbe01eeed093cb22bb8f5acdc3”, it is mathematically impossible to find “hello world”.

Some examples of hash functions are:

- Message-Digest Algorithm (MD5) (used in level 6!)
- Secure Hash Algorithm 256 (SHA 256)
  - Fun fact: SHA 256 is used a lot in cryptocurrencies to check for data integrity, generate data signatures, and lots more.
- BCrypt
- Argon2 (very strong I heard, surprised not too many people are talking about this)

Now let’s say a password is encrypted with a hash function, and you only have it’s hash. There are ways to “crack” hashes. https://en.wikipedia.org/wiki/Password_cracking

Others links which might take you down a rabbit-hole:

- https://en.wikipedia.org/wiki/RockYou#Data_breach
- https://en.wikipedia.org/wiki/John_the_Ripper
- https://hashcat.net/hashcat/
- https://www.google.com/search?q=quantum+computing+cracking+encryption