# CTF Writeup -- Houseplant CTF

## Preston Kemp -- NYU Offensive Security Spring 2020

## Introduction

For my CTFtime ranked CTF challenge I chose the Houseplant CTF. I focused on the reverse engineering challenges, since they were most like the kind of work that we had done in class.

### Other challenges first:

In order to get to a challenge that was *actually* difficult, I had to complete several easy challenges first, most were basic python reverse engineering problems, I'll touch on those briefly:

1. EZ:This was the first challege that had to be solved in order to move on. The flag for this challenge was `rtcp{tH1s_i5_4_r3aL_fL4g_s0_Do_sUbm1T_1t!}` and it was simply in a comment inside the provided python file.

2. PZ: This was the second challenge that had to be solved. This challenge was also very straight forward, the program is a series of functions with if statements, most of the functions aren't really used and their intent appears to be to confuse the reader. The function worth analyzing in this example is `checkpass()` which, does as it sounds it does and checks the password (flag) that the user enters. If the user enters `rtcp{iT5_s1mPlY_1n_tH3_C0d3}` the user is granted access. The password in this case is also the flag.

3. Lemon: This was the third challenge that was necessary to continue. This challenge was also straight forward and that it involved using what essentially amounted as multiple substring to reconstruct the flag that was present in the code itself. The flag ended up being: `rtcp{y34H_tHiS_a1nT_sEcuR3}`

4. Squeezy: This was the fourth and final challenge before getting access to the more difficult challenges. This challenge involved an encoded version of the flag, meaning it needed to be transformed to recover the original flag. When analyzing the code, I noticed a couple of calls to Base64 encode and decode, so obviously the flag was encoded.

    1. I started by decoding the encoded flag `HxEMBxUAURg6I0QILT4UVRolMQFRHzokRBcmAygNXhkqWBw=` from Base64 and recovered the unencoded flag
    2. We are also given a variable called "key" which is `meownyameownyameownyameownyameownya`
    3. The flag was more than just encoded though, there was a function called "woah" that did the following:

```
def woah(s1,s2): #part of pass 3
    return ''.join(chr(ord(a) ^ ord(b)) for a,b in zip(s1,s2))
```

    the ^ symbol in Python is used to bitwise XOR two operands. This is pretty clearly just XORing the input of s1 and s2 (the key and the user's input) charcter by character. Since XOR is commutative,

(a⊕b=b⊕a), we can simply run our decoded Base64 flag back through this function as the input (s2) and pass the same key (s1) through the function and retrieve the actual flag.

Example solution:

```
import base64

def woah(s1,s2): #part of pass 3
    return ''.join(chr(ord(a) ^ ord(b)) for a,b in zip(s1,s2))

def solvePass3():
    flag = b'HxEMBxUAURg6I0QILT4UVRolMQFRHzokRBcmAygNXhkqWBw='
    key = "meownyameownyameownyameownyameownya"
    a = base64.b64decode(flag, altchars=None)
    b = a.decode()
    print(woah(key, b))

def main():
    solvePass3()

main()
```

4. Bingo! We got the flag `rtcp{y0u_L3fT_y0uR_x0r_K3y_bEh1nD!}`

## The Problem