Security Project

Capture the Flags

Submitted by:

Abdelrahman Ahmed Mohamed	**1210248**
Ahmed Essam Elshahat	**1210346**
Karim Magdy Monir	**1210070**
Mariam Essam	**1180027**

Table of Contents:

Chest1:

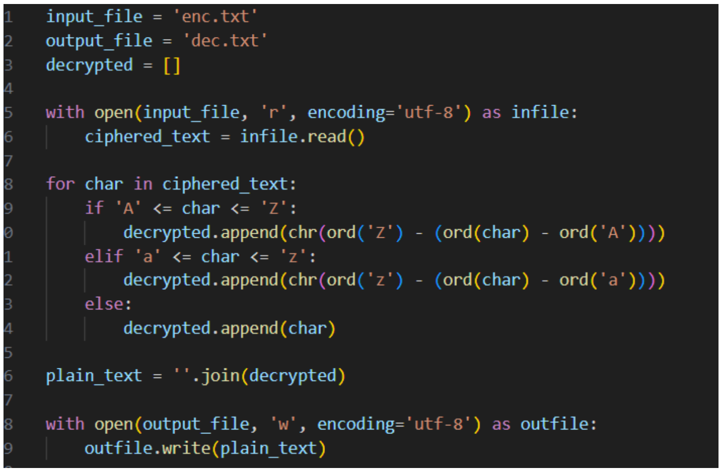
Hints:

- Count them
- Find lengths
- Patterns hidden in plain sight
- Cast aside the meaningless—ignore the special characters and the empty spaces
- CLASSICAL

Steps:

- We will use a classical encryption/decryption technique
- Since the same series of characters are repeated over and over again this exclude transposition and leaves us with substitution algorithms.
- We took a sample and tried all possible shifts in alphabet (1- 26) but did not find any understandable words.
- We noticed decrypted text looks like a dialog between to persons WRL and QLGZIL (Dio and Jotaro), therefore we matched the letters to find that:
  W->D R->I L->O J->Q G->T Z->A I->R
- With the above mapping we tried to complete the full mapping of the alphabet.
- We Searched for common words like [The, Is, And, ….]. We matched the letters:
  F->E E->S B->N
- Using the above conclusions, we searched for transposition classical encryption techniques and found Atbash cipher algorithm that matches the above mappings.
- We finished the mappings (the alphabet reversed) and tried the algorithm to get the plaintext.

Code:



```
input_file = 'enc.txt'
output_file = 'dec.txt'
decrypted = []

with open(input_file, 'r', encoding='utf-8') as infile:
    ciphered_text = infile.read()

for char in ciphered_text:
    if 'A' <= char <= 'Z':
        decrypted.append(chr(ord('Z') - (ord(char) - ord('A'))))
    elif 'a' <= char <= 'z':
        decrypted.append(chr(ord('z') - (ord(char) - ord('a'))))
    else:
        decrypted.append(char)

plain_text = ''.join(decrypted)

with open(output_file, 'w', encoding='utf-8') as outfile:
    outfile.write(plain_text)
```

Key:

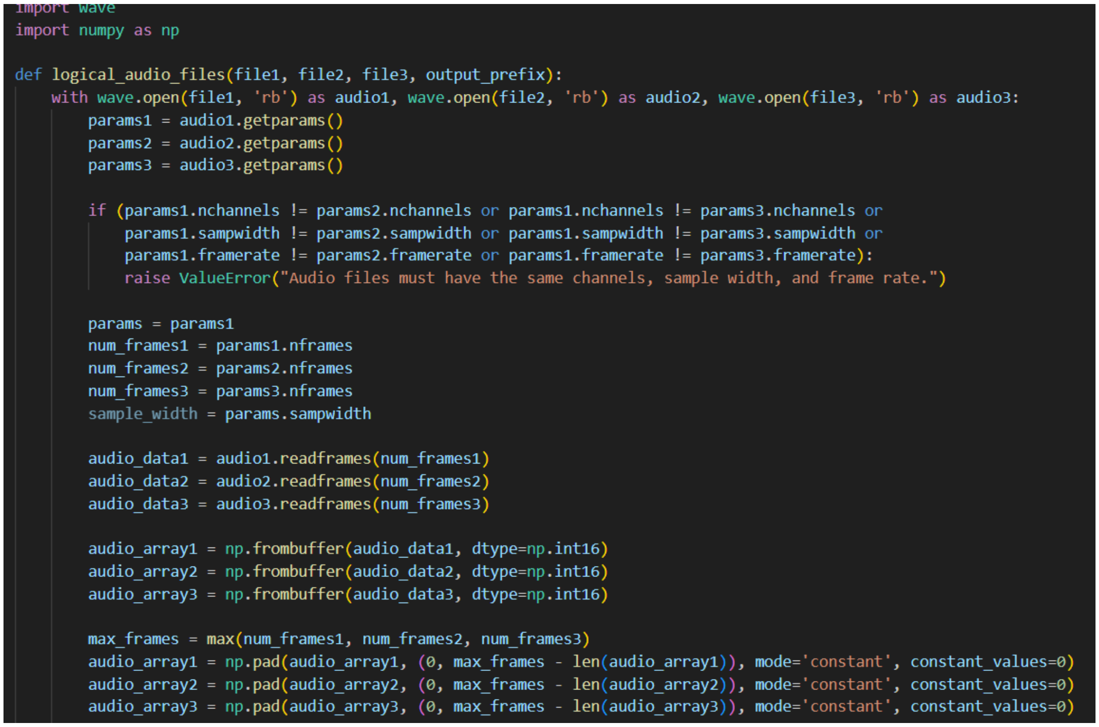
CMPN{i_luv_jojo}

Chest2:

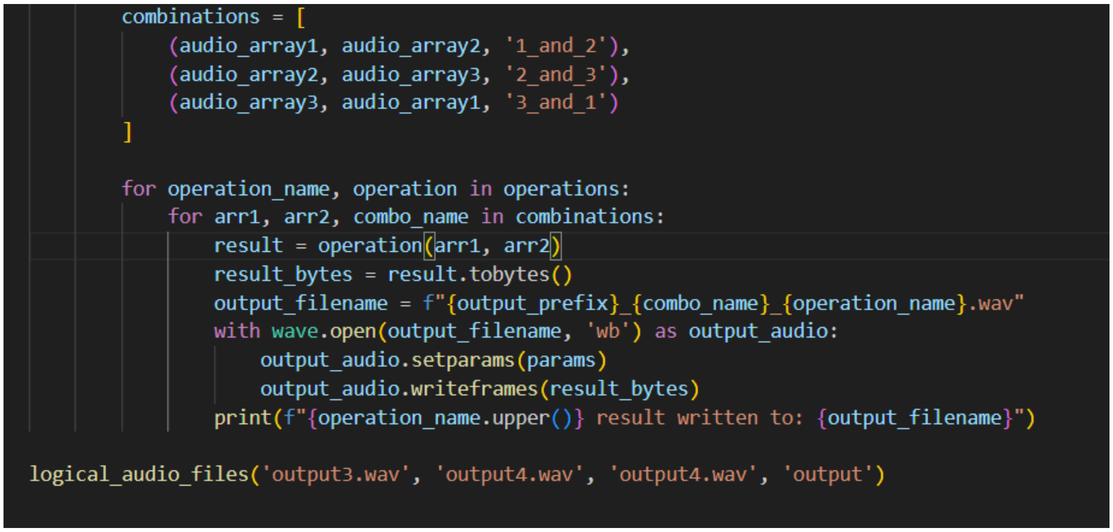
Steps:

- Try all possible logical operations on all file combinations
- Manually hear each output file to search for meaningful words.
- Shuffle the audio manually to find the correct key.

Code:



```python
import wave
import numpy as np

def logical_audio_files(file1, file2, file3, output_prefix):
    with wave.open(file1, 'rb') as audio1, wave.open(file2, 'rb') as audio2, wave.open(file3, 'rb') as audio3:
        params1 = audio1.getparams()
        params2 = audio2.getparams()
        params3 = audio3.getparams()

        if (params1.nchannels != params2.nchannels or params1.nchannels != params3.nchannels or
            params1.sampwidth != params2.sampwidth or params1.sampwidth != params3.sampwidth or
            params1.framerate != params2.framerate or params1.framerate != params3.framerate):
            raise ValueError("Audio files must have the same channels, sample width, and frame rate.")

        params = params1
        num_frames1 = params1.nframes
        num_frames2 = params2.nframes
        num_frames3 = params3.nframes
        sample_width = params.sampwidth

        audio_data1 = audio1.readframes(num_frames1)
        audio_data2 = audio2.readframes(num_frames2)
        audio_data3 = audio3.readframes(num_frames3)

        audio_array1 = np.frombuffer(audio_data1, dtype=np.int16)
        audio_array2 = np.frombuffer(audio_data2, dtype=np.int16)
        audio_array3 = np.frombuffer(audio_data3, dtype=np.int16)

        max_frames = max(num_frames1, num_frames2, num_frames3)
        audio_array1 = np.pad(audio_array1, (0, max_frames - len(audio_array1)), mode='constant', constant_values=0)
        audio_array2 = np.pad(audio_array2, (0, max_frames - len(audio_array2)), mode='constant', constant_values=0)
        audio_array3 = np.pad(audio_array3, (0, max_frames - len(audio_array3)), mode='constant', constant_values=0)
```



```python
        combinations = [
            (audio_array1, audio_array2, '1_and_2'),
            (audio_array2, audio_array3, '2_and_3'),
            (audio_array3, audio_array1, '3_and_1')
        ]

        for operation_name, operation in operations:
            for arr1, arr2, combo_name in combinations:
                result = operation(arr1, arr2)
                result_bytes = result.tobytes()
                output_filename = f"{output_prefix}_{combo_name}_{operation_name}.wav"
                with wave.open(output_filename, 'wb') as output_audio:
                    output_audio.setparams(params)
                    output_audio.writeframes(result_bytes)
                print(f"{operation_name.upper()} result written to: {output_filename}")

logical_audio_files('output3.wav', 'output4.wav', 'output4.wav', 'output')
```

Key:

CMPN{cybersecurity_OTP}

Chest3:

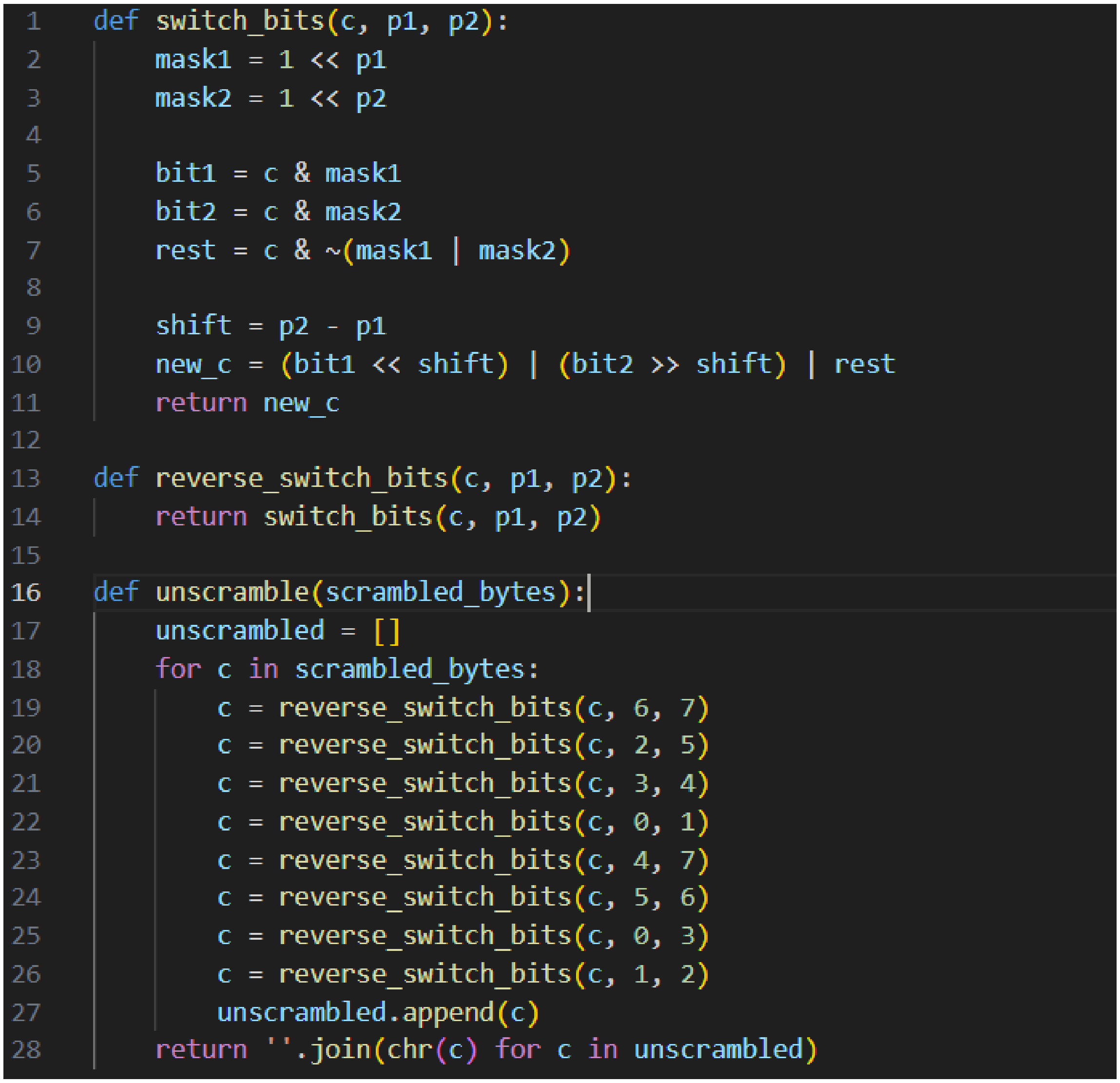
Hints:

- Reverse engineering

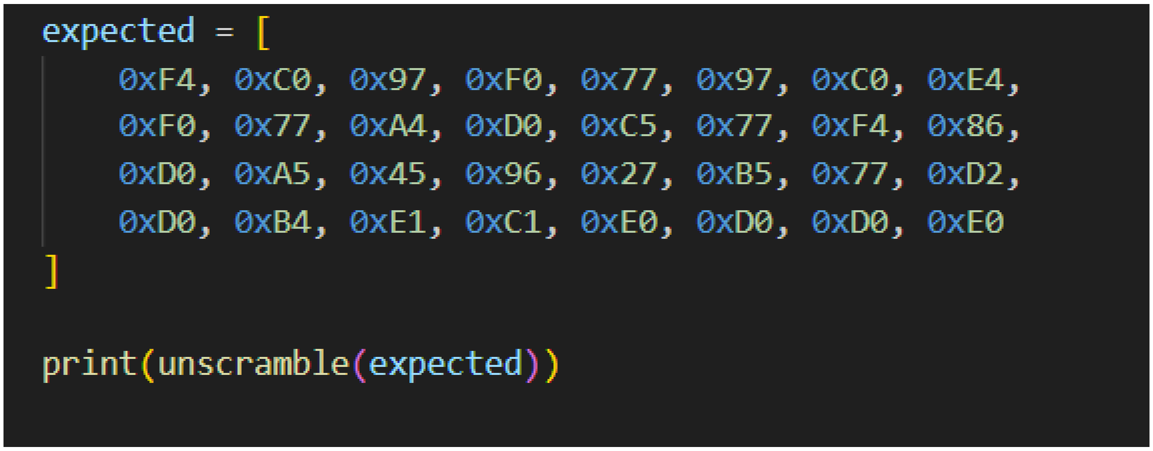
Steps:

- We tried a brute-force attack but after a few hours of running the script we came to nothing, we needed more time and stronger computation machines.
- We tried to understand the code and found out if we reversed the steps in the original code we will find the password.

Code:



```
def switch_bits(c, p1, p2):
    mask1 = 1 << p1
    mask2 = 1 << p2

    bit1 = c & mask1
    bit2 = c & mask2
    rest = c & ~(mask1 | mask2)

    shift = p2 - p1
    new_c = (bit1 << shift) | (bit2 >> shift) | rest
    return new_c

def reverse_switch_bits(c, p1, p2):
    return switch_bits(c, p1, p2)

def unscramble(scrambled_bytes):
    unscrambled = []
    for c in scrambled_bytes:
        c = reverse_switch_bits(c, 6, 7)
        c = reverse_switch_bits(c, 2, 5)
        c = reverse_switch_bits(c, 3, 4)
        c = reverse_switch_bits(c, 0, 1)
        c = reverse_switch_bits(c, 4, 7)
        c = reverse_switch_bits(c, 5, 6)
        c = reverse_switch_bits(c, 0, 3)
        c = reverse_switch_bits(c, 1, 2)
        unscrambled.append(c)
    return ''.join(chr(c) for c in unscrambled)
```



```
expected = [
    0xF4, 0xC0, 0x97, 0xF0, 0x77, 0x97, 0xC0, 0xE4,
    0xF0, 0x77, 0xA4, 0xD0, 0xC5, 0x77, 0xF4, 0x86,
    0xD0, 0xA5, 0x45, 0x96, 0x27, 0xB5, 0x77, 0xD2,
    0xD0, 0xB4, 0xE1, 0xC1, 0xE0, 0xD0, 0xD0, 0xE0
]

print(unscramble(expected))
```

Key:

s0m3_m0r3_b1t_sh1fTiNg_91c642112

Chest4:

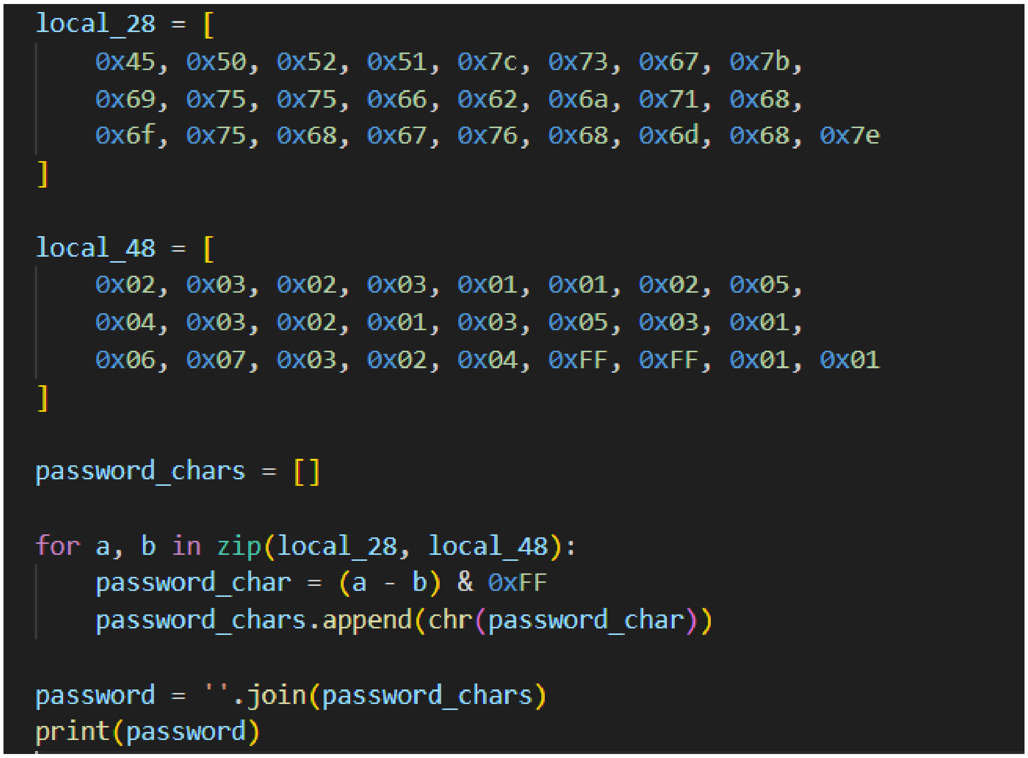
Hints:

- Use Ghidra to analyze the binary as hinted in the message.
- Start with the main function.
- Locate the check_pw function.
- Use the ASCII table to.
- Watch for obfuscation or tricks.
- Check the Strings window in Ghidra.
- Recreate logic manually or with a script.
- Stay organized.

Steps:

- Install and setup Ghidra.
- Imported the .exe file.
- Exported the main function.
- Exported the check_pw function.
- Get the byte representation of local_28 and local_48.
- Compute the password using the below code.

Code:



```python
local_28 = [
    0x45, 0x50, 0x52, 0x51, 0x7c, 0x73, 0x67, 0x7b,
    0x69, 0x75, 0x75, 0x66, 0x62, 0x6a, 0x71, 0x68,
    0x6f, 0x75, 0x68, 0x67, 0x76, 0x68, 0x6d, 0x68, 0x7e
]

local_48 = [
    0x02, 0x03, 0x02, 0x03, 0x01, 0x01, 0x02, 0x05,
    0x04, 0x03, 0x02, 0x01, 0x03, 0x05, 0x03, 0x01,
    0x06, 0x07, 0x03, 0x02, 0x04, 0xFF, 0xFF, 0x01, 0x01
]

password_chars = []

for a, b in zip(local_28, local_48):
    password_char = (a - b) & 0xFF
    password_chars.append(chr(password_char))

password = ''.join(password_chars)
print(password)
```

Key:

CMPN{reverse_engineering}