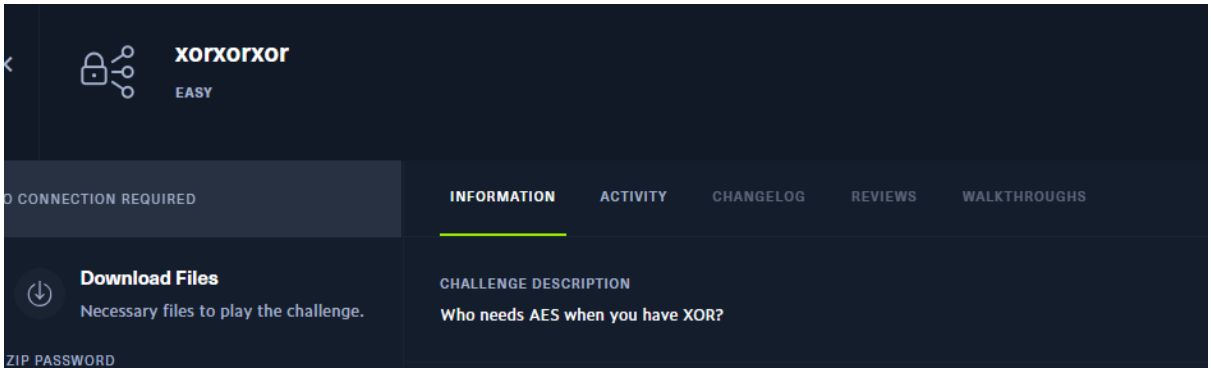
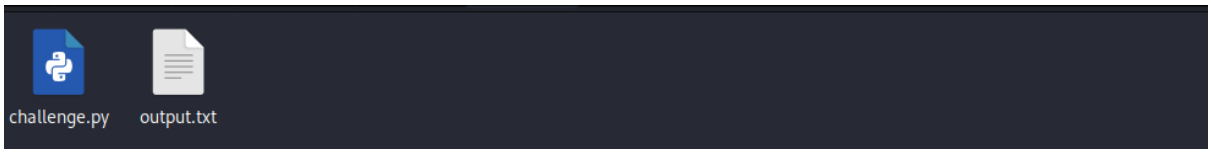
Erel Regev

Table of Contents

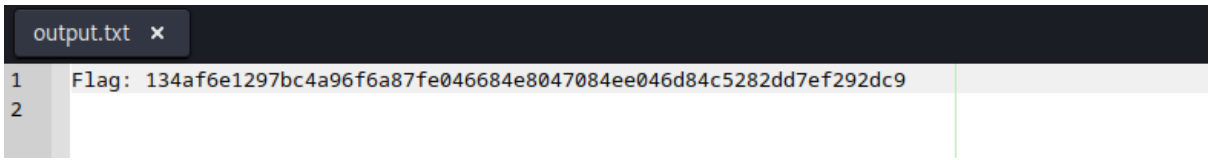
Intro



Received files:



Viewing output.txt:



Viewing challenge.py:

HTB Challenge: Xorxorxor - Subject: Crypto - Difficulty: Easy

Erel Regev

```python
#!/usr/bin/python3
import os
flag = open('flag.txt', 'r').read().strip().encode()

class XOR:
    def __init__(self):
        self.key = os.urandom(4)
    def encrypt(self, data: bytes) -> bytes:
        xored = b''
        for i in range(len(data)):
            xored += bytes([data[i] ^ self.key[i % len(self.key)]])
        return xored
    def decrypt(self, data: bytes) -> bytes:
        return self.encrypt(data)

def main():
    global flag
    crypto = XOR()
    print ('Flag:', crypto.encrypt(flag).hex())

if __name__ == '__main__':
    main()
```

Challenge.py

This Python script demonstrates a simple XOR encryption/decryption using a random key of length 4 bytes.

flag = open('flag.txt', 'r').read().strip().encode():

This line reads the contents of the file named flag.txt, strips any leading/trailing whitespace, and encodes the resulting string into bytes. The content of the flag is stored in the flag variable as **bytes**.

class XOR:

Defines a class named XOR for performing XOR encryption and decryption operations.

def __init__(self):

Constructor method for the XOR class. It initializes an instance of the class by generating a random 4-byte key using os.urandom(4).

def encrypt(self, data: bytes) -> bytes:

This method takes a bytes object as input and performs XOR encryption using the stored key. For each byte in the input data, it XORs it with the corresponding byte from the key. The resulting encrypted data is returned as bytes.

def decrypt(self, data: bytes) -> bytes:

This method is a wrapper for the encrypt method since XOR encryption and decryption are the same operation. It takes encrypted data and returns the decrypted data.

def main():

The main function that executes the encryption and printing of the flag.

global flag:

Declares the flag variable as global so that it can be accessed within the main function.

crypto = XOR():

Creates an instance of the XOR class named crypto.

print ('Flag:', crypto.encrypt(flag).hex()):

Encrypts the contents of the flag variable using the encrypt method of the XOR instance. The encrypted data is then converted to a hexadecimal string using .hex() and printed with the prefix "Flag:".

if __name__ == '__main__':

This condition checks if the script is being run directly (as opposed to being imported as a module). If true, the main function is executed.

Rev.py

The code will generate a random key with the length of 4. Then XOR each byte of the flag with each byte of the key.

We know the flag starts with 'HTB{', and the output is:

134af6e1297bc4a96f6a87fe046684e8047084ee046d84c5282dd7ef292dc9

From the code, we can see that each character will be XOR with each character of the key and the length of the key is 4 characters.

Let's create a script that is doing the reverse action using the information we hacve:

```python
import itertools

RAW_ENC_FLAG = "134af6e1297bc4a96f6a87fe046684e8047084ee046d84c5282dd7ef292dc9"
KNOWN_KEY_START = b'HTB{'
flag_enc = bytes.fromhex(RAW_ENC_FLAG)

def bxor(b1, b2):
    return bytes([x ^ y for x, y in zip(b1, b2)])

def decrypt(data: bytes, key) -> bytes:
    return bytes([d ^ k for d, k in zip(data, itertools.cycle(key))])

key = bxor(KNOWN_KEY_START, flag_enc[:4])
print(decrypt(flag_enc, key))
```

Importing itertools Module:

The code begins by importing the itertools module, which is necessary for the itertools.cycle function used later in the code.

Variables:

RAW_ENC_FLAG:

This variable holds a hexadecimal string representing the encrypted flag.

KNOWN_KEY_START:

HTB Challenge: Xorxorxor - Subject: Crypto - Difficulty: Easy

Erel Regev

This is a bytes object containing the known start of the flag in its plaintext form.

flag_enc:

This variable converts the hexadecimal string RAW_ENC_FLAG into bytes using the bytes.fromhex method.

bxor Function:

This function takes two byte sequences, b1 and b2, and performs a bitwise XOR operation on corresponding bytes of the sequences.

It returns a new bytes object containing the XORed result of the two inputs.

decrypt Function:

This function takes two arguments: a bytes sequence data to decrypt and a key sequence for decryption.

Inside the function, a list comprehension is used to XOR each byte of the data with the corresponding byte from the key.

The itertools.cycle function is used to cycle through the bytes of the key repeatedly to match the length of the data.

The result of the XOR operation is collected into a new bytes object, which is then returned.

Key Calculation:

The key for decryption is calculated using the bxor function. It XORs the KNOWN_KEY_START with the first four bytes of flag_enc. This is done to obtain the initial key that was used for encryption.

Decrypt and Print:

The decrypt function is called with flag_enc as the data and the calculated key. The decrypted bytes are printed to the console