

Crypto: Follow the Currents

We are given the source code + encrypted message (ciphertext). Here is the source code:

```
*~/Desktop/CTF/AngstromCTF/followthecurrent/current.py - Mousepad
File Edit Search View Document Help
import os
import zlib
def keystream():
    key = os.urandom(2)
    index = 0
    while 1:
        index+=1
        if index >= len(key):
            key += zlib.crc32(key).to_bytes(4,'big')
        yield key[index]
ciphertext = []
with open("plain","rb") as f:
    plain = f.read()
    assert b"actf{" in plain
    k = keystream()
    for i in plain:
        ciphertext.append(i ^ next(k))
with open("enc","wb") as g:
    g.write(bytes(ciphertext))
```

What vulnerable is that the **key** is urandom of length of 2. Brute Force it is ☹️ $256 * 256 = 65536$ cases.

```
with open("enc","rb") as g:
    cipher = g.read()
print(cipher)
#for c in cipher:
#    print(cipher)

for i in range(65536):
    msg21 = []
    k = keystream(i.to_bytes(2,byteorder="big"))
    for j in cipher:
        msg1.append(j ^ next(k))
    msg2 = bytes(msg1)
    if b"actf{" in msg2:
        print(msg2)
```

Flag is: actf{low_entropy_keystream}

Crypto: I'm so RANDOM

Here is the source code:

```
File Edit Search View Document Help
~/Desktop/CTF/AngstromCTF/iamsorandom/prng.py - Mousepad

import time
import random
import os

class Generator():
    DIGITS = 8
    def __init__(self, seed):
        self.seed = seed
        assert(len(str(self.seed)) == self.DIGITS)

    def getNum(self):
        self.seed = int(str(self.seed**2).rjust(self.DIGITS*2, "0")[self.DIGITS//2:self.DIGITS//2 + self.DIGITS//2])
        return self.seed

r1 = Generator(random.randint(10000000, 99999999))
r2 = Generator(random.randint(10000000, 99999999))
query_counter = 0
while True:
    query = input("Would you like to get a random output [r], or guess the next random number [g]? ")
    if query.lower() not in ["r", "g"]:
        print("Invalid input.")
        break
    else:
        if query.lower() == "r" and query_counter < 3:
            print(r1.getNum() * r2.getNum())
            query_counter += 1
        elif query_counter >= 3 and query.lower() == "r":
            print("You don't get more random numbers!")
        else:
            for i in range(2):
                guess = int(input("What is your guess to the next value generated? "))
                if guess != r1.getNum() * r2.getNum():
                    print("Incorrect!")
                    exit()
            with open("flag", "r") as f:
                flag = f.read()
            print("Congrats! Here's your flag: ")
            print(flag)
            exit()
```

```
kali@kali: ~/Desktop/CTF/AngstromCTF/iamsorandom
File Actions Edit View Help

(kali@kali)~/Desktop/CTF/AngstromCTF/iamsorandom:
$ nc crypto.2021.chall.actf.co 21600

Would you like to get a random output [r], or guess the next random number [g]? r
367563595578114
Would you like to get a random output [r], or guess the next random number [g]? r
3609996477624344
Would you like to get a random output [r], or guess the next random number [g]? r
2706639348317245
Would you like to get a random output [r], or guess the next random number [g]?
```

We are given 3 random numbers, and we will have to guess the next random numbers. Each random number is a product of two numbers with PNRG function.

Here the pseudorandom generator (PRNG) function generates new random by squaring its input + padding 0s on the right side, then only choose from digit 4:12 (8 digits).

As we know the seed is 8 digits and given the first random number, we can factorize it to a product of two numbers of length 8. Then for each pair as new seeds, generate new PNRG number to see if it matches. 😊 Here is the flag: **actf{middle_square_method_more_like_middle_fail_method}**