[2023 JBUCTF] crypto

same_nonce_1

Write-Up

문제 개요

제공 파일 : same_nonce_1.py

```python
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from base64 import b64decode, b64encode

flag = open('/flag', 'rb').read()
key = get_random_bytes(16)
nonce = get_random_bytes(12)
crypto = AES.new(key, AES.MODE_CTR, nonce=nonce)
enc_flag = crypto.encrypt(flag)
enc_flag = b64encode(enc_flag).decode()

for _ in range(5):
    print('[1] encrypt')
    print('[2] flag')
    try:
        n = int(input('>> '))
    except:
        continue
    if n == 1:
        try:
            data = input('data : ')
            data = b64decode(data)
            crypto = AES.new(key, AES.MODE_CTR, nonce=nonce)
            cipher = crypto.encrypt(data)
            cipher = b64encode(cipher).decode()
            print(cipher)
        except:
            continue
    elif n == 2:
        print(enc_flag)
        exit()

```

1번을 입력하면 base64 인코딩 된 data를 입력 받고, 입력 받은 data를 디코딩 후, AES-CTR 모드로 동일한 key, nonce를 사용해서 암호화 한 값을 base64 인코딩해서 출력을 한다.

2번을 입력하면 flag를 AES-CTR 모드로 동일한 key, nonce를 사용해서 암호화 한 값을 base64인코딩하여 출력한다.

문제 풀이

AES-CTR 참고 : https://url.kr/lzcyk9

CTR 운영모드는 nonce값과 counter값을 이어 붙여서 key로 블록 암호화를 한 결과를 평문과 xor 해서 암호문을 생성해낸다.

문제에서는 입력 받은 값과 flag를 동일한 key와 nonce로 암호화한다.

즉 입력한 값의 암호문과 flag의 암호문은 동일한 값으로 xor되어 있는 상태이다.

따라서 flag와 동일한 길이의 평문을 암호화 한 값을 flag의 암호화 값에 xor하고 그 값에 평문을 다시 xor하면 flag 값을 알 수 있다.

$X \,||\, Y$: The concatenation of two bit strings X and Y.
$X \oplus Y$: The bitwise exclusive − OR of two bit strings X and Y of the same length.
$E_K(X)$: The output of the forward cipher function of the block cipher under the key K applied to the block X

$$E_k(flag) = flag \oplus E_k(Nonce \,||\, Counter)$$
$$E_k(plain) = plain \oplus E_k(Nonce \,||\, Counter)$$
$$E_k(flag) \oplus E_k(plain) = flag \oplus E_k(Nonce \,||\, Counter) \oplus plain \oplus E_k(Nonce \,||\, Counter)$$
$$E_k(flag) \oplus E_k(plain) = E_k(Nonce \,||\, Counter) \oplus E_k(Nonce \,||\, Counter) \oplus flag \oplus plain$$
$$E_k(flag) \oplus E_k(plain) = flag \oplus plain$$
$$\therefore flag = E_k(flag) \oplus E_k(plain) \oplus plain$$

exploit.py

```python
from pwn import *
from base64 import b64encode, b64decode

def xor_bytes(a, b, block_size):
    result = int(a.hex(), 16) ^ int(b.hex(), 16)
    result = bytes.fromhex(hex(result)[2:].zfill(block_size*2))
    return result

BLOCK_SIZE = 16

p = remote('172.17.0.2', 10005)
p.recvuntil(b'>> ')
p.sendline(b'2')
enc_flag = p.recvline()[:-1]
enc_flag = b64decode(enc_flag)
cipher_szie = len(enc_flag)

p = remote('172.17.0.2', 10005)
p.recvuntil(b'>> ')
p.sendline(b'1')
data = b'A'*cipher_szie
p.sendafter(b'data : ', b64encode(data) + b'\n')
enc_data = b64decode(p.recvline())

p.recvuntil(b'>> ')
p.sendline(b'2')
enc_flag = p.recvline()[:-1]
enc_flag = b64decode(enc_flag)
p.close()

tmp = b''
flag = b''
for i in range(0, cipher_szie, BLOCK_SIZE):
    tmp += xor_bytes(enc_flag[i:i + BLOCK_SIZE], enc_data[i:i + BLOCK_SIZE], len(enc_data[i:i + BLOCK_SIZE]))

for i in range(0, cipher_szie, BLOCK_SIZE):
    flag += xor_bytes(tmp[i:i + BLOCK_SIZE], data[i:i + BLOCK_SIZE],len(data[i:i + BLOCK_SIZE]))

print(f'flag : {flag.decode()}')
```

FLAG

scpCTF{5e9f0fc6422fb589154a7827204c5720c61347e629df}