

[2023 JBUCTF] crypto

ecb_mode

Write-Up

문제 개요

제공 파일 : ecb_mode.py

```
1  from Crypto.Util.Padding import pad
2  from Crypto.Cipher import AES
3  from Crypto.Random import get_random_bytes
4  from base64 import b64decode, b64encode
5
6  flag = open('/flag', 'rb').read()
7
8  BLOCK_SIZE = 16
9  key = get_random_bytes(16)
10 crypto = AES.new(key, AES.MODE_ECB)
11
12 while True:
13     try:
14         data = b64decode(input('plain text (base64) >> '))
15     except:
16         print('Retry')
17         continue
18     enc_data = crypto.encrypt(pad(data + flag, BLOCK_SIZE))
19     print(f'enc_data : {b64encode(enc_data).decode()}')
20
```

Base64 인코딩 된 문자열을 입력으로 받고, 그 문자열을 base64 디코딩하여 **flag와 더한 후**, 패딩을 하고, 동일한 key를 사용해서 AES-ECB로 암호화 한 결과를 다시 base64 인코딩해서 출력해준다.

그리고 이를 반복한다.

패딩

Wikipedia Padding (cryptography) : <https://url.kr/v76dwc>

IBM PKCS padding method : <https://url.kr/rblvmy>

pad 함수의 동작은 입력 받은 문자열이 n 개의 바이트로 이루어져 있을 때, 바이트의 개수가 입력 받은 블록의 크기의 배수가 되도록 특정한 규칙을 가진 값을 채운다.

(문제에서는 블록의 크기가 16이므로 16을 기준으로 설명 하겠다.)

n 을 16으로 나눈 나머지의 값을 x 라고 하면 $(16 - x)$ 를 x 번 만큼 뒤에서 채운다.

만약 n 이 16의 배수이면 16을 16번 채운다.

예를 들어 010203040506 을 패딩 처리한다고 했을 때,

010203040506 의 바이트 개수가 6개 이므로 $16 - 6 = 10$ 을 10번 채운다.

즉 010203040506 을 패딩 처리하면

0102030405060a0a0a0a0a0a0a0a0a 가 된다.

AES-ECB 모드

Wikipedia 전자 코드북 (ECB) : <https://url.kr/tpevx7>

AES-ECB 운영모드는 각각의 16byte 블록마다 암호화를 적용하는 모드이다.

암호문 블록들 서로의 연관성이 없으므로 동일한 16byte 평문을 암호화 한 암호문 블록들의 결과 값은 서로 같다.

이 문제에서는 이러한 원리를 이용하여 암호화된 flag의 값을 알아낸다.

문제 풀이

flag의 길이는 평문이 16byte 단위로 패딩 되는 것을 이용해서 구한다.

입력 값의 크기를 1 ~16 만큼 줘서 입력하다 보면 암호문의 크기가 변하는 순간이 있다. 그 때의 입력 값의 크기를 변하지 않았던 암호문의 길이에서 빼면 flag의 길이가 된다.

1 ~ 16 길이만큼의 입력 값을 넣었는데도 암호문의 크기가 변하지 않았다면 암호문의 길이에

1 + 16을 뺀 값이 flag의 길이이다.

실제로 적용해보면 flag의 크기가 48byte임을 알 수 있다.

아래의 코드는 이를 자동화하여 구현한 것이다.

exploit.py

```

1 from pwn import *
2 from Crypto.Util.Padding import pad
3 from base64 import b64decode, b64encode
4
5 p = remote('172.17.0.2', 10009)
6 BLOCK_SIZE = 16
7 flag = []           # flag length : 48 byte
8
9 flag_len = 11111111111111111111111111111111
10 count = 0
11 for i in range(1, 17):
12     p.recvuntil(b'plain text (base64) >> ')
13     p.sendline(b64encode(b'A'*i))
14     p.recvuntil(b'enc_data')
15     result = b64decode(p.recvline()[::-1])
16     if len(result) > flag_len:
17         flag_len = flag_len - i
18         break
19     else:
20         count += 1
21         flag_len = len(result)
22 if count == 16:
23     flag_len = flag_len - 17
24 print('flag length : ', flag_len)
25
26 data = b''
27 for i in range(3):
28     result = b''
29     for j in range(1, 17)):
30         data = b'A' * (j + BLOCK_SIZE)
31         p.recvuntil(b'plain text (base64) >> ')
32         p.sendline(b64encode(data))
33         p.recvuntil(b'enc_data : ')
34         cipher = b64decode(p.recvline()[::-1])
35         if i == 0:
36             if j != 16:
37                 # j = 16 일 때, 암호문의 마지막은 패딩으로 인해 0f*16 이므로 마지막 두번째 블록을 봐야함
38                 cipher_block = cipher[-16:]
39             else:
40                 cipher_block = cipher[-32:-16]
41         else:
42             if j != 16:
43                 cipher_block = cipher[-32-BLOCK_SIZE*(i-1):-16-BLOCK_SIZE*(i-1)]
44             else:
45                 cipher_block = cipher[-32-BLOCK_SIZE*i:-16-BLOCK_SIZE*i]
46
47         for prefix in range(0x0100):
48             prefix = bytes((prefix))
49             if i == 0:
50                 payload = pad(prefix + result, BLOCK_SIZE)
51             else:
52                 payload = prefix + result + flag[i-1][::-len(prefix + result)]
53
54             p.recvuntil(b'plain text (base64) >> ')
55             p.sendline(b64encode(payload))
56             p.recvuntil(b'enc_data : ')
57             cipher = b64decode(p.recvline()[::-1])
58             if cipher[:16] == cipher_block:
59                 result = prefix + result
60                 break
61         flag.append(result)
62 flag.reverse()
63 print(f'flag : {(flag[0] + flag[1] + flag[2]).decode()}')
64 
```

FLAG

scpCTF{42533c6a4e762e17bfe825c43331d7fb25842cb0}