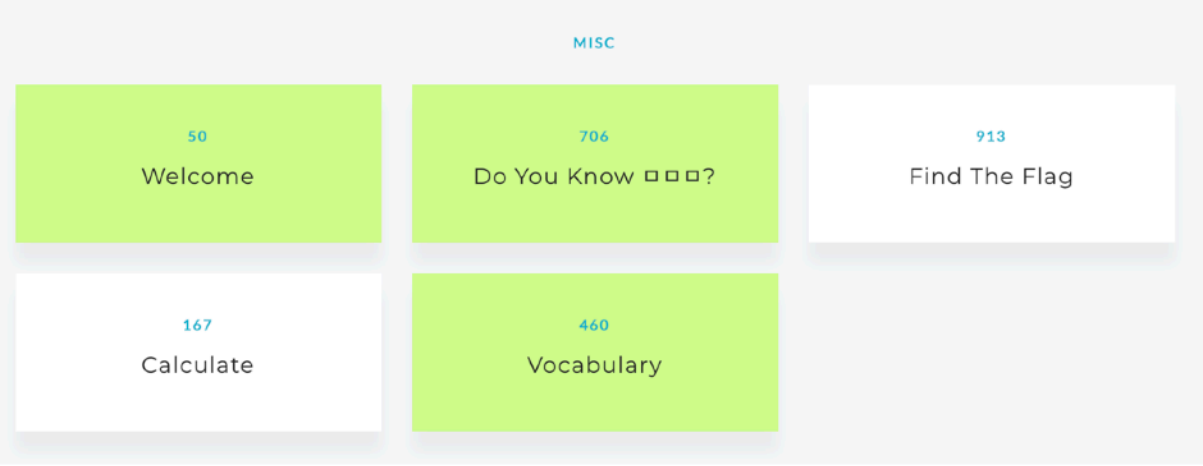
BAOBOBSN0WB4LL
CTF 14th
Sunrin Internet Highschool
2017년 12월 23일

Root CTF Write-up

13	『』	2647
14	BAOBOBSN0WB4LL	2507
15	hOwDayS	2469

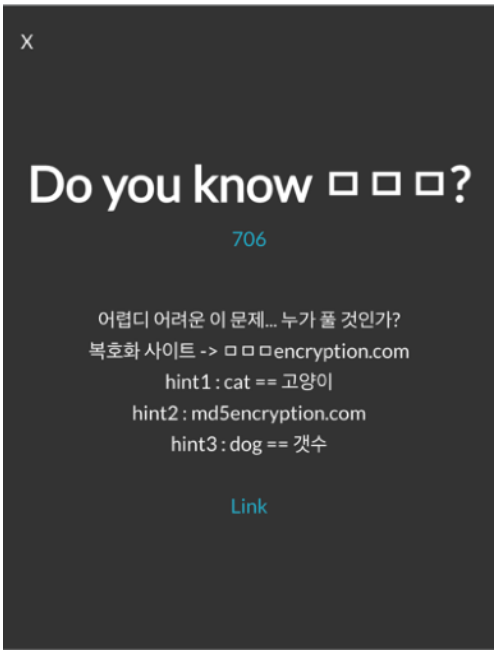


MISC Write-up:

1. Welcome 문제는 MIC Check 문제니 스킵.

2. Do You Know □□□?

이 문제는 사실 간단하다. Link에 들어가보면 md5 해시 값으로 추정되는 32바이트의 문자열이 있으며, 최상단에는 플래그에 대한 정보가 나와있다. 0 = 4dog 라는 것은 플래그가 숨어있는 해시 값에 0이 4개 있다는 뜻이다. 0이 4개 있는 문자열을 찾아보니 단 1개밖에 없는 것을 알 수 있다.




```
Find the Flag!
[0 = 4dog] [2 = 1dog] [4 = 4dog] [5 = 4dog]
[6 = 1dog] [7 = 1dog] [8 = 2dog] [9 = 3dog]
[a = 4dog] [b = 3dog] [c = 3dog] [d = 2dog]


g d a 6 v z 1 3
d o 9 8 j 0 1 x
1 b i 9 8 1 2 6
b z 9 6 y u 3 1

k 6 9 7 j i h z
k y i j t b i n
y 9 5 g f j 7 b
3 n i u t g h m
```

b 0 2 5 5 4 4 c
c 0 7 9 8 5 4 4
0 d a c 0 b a a
9 d b 8 a 5 9 6

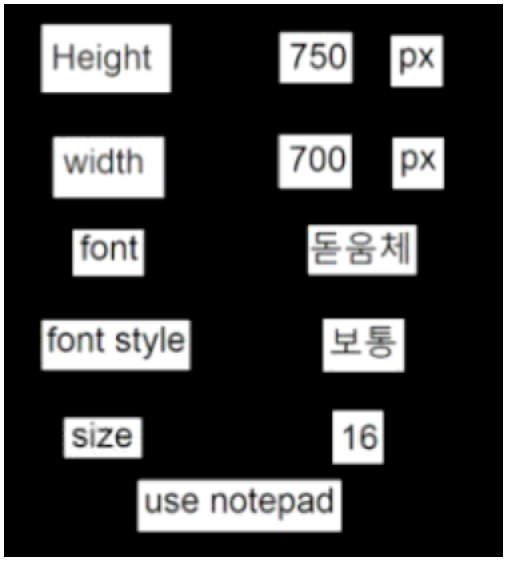
저 해쉬값을 md5encryption.com에서 복호화시켜주면 FLAG가 나온다.

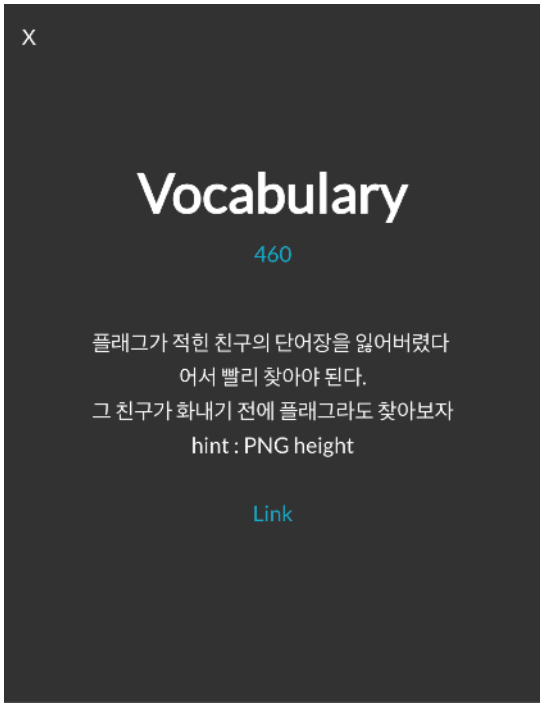
Md5 Hash: b025544cc07985440dac0baa9db8a596

Decrypted Text: FLAG is FLAG{MD5_3nCryPt_Ye@h!}

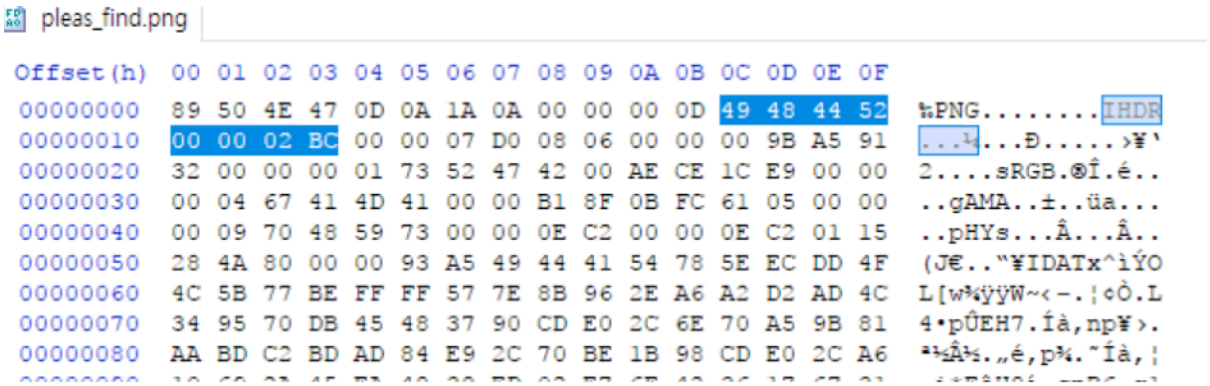
3. Vocabulary

이 문제는 PNG파일의 헤더에 있는 IHDR 청크에 대한 정보만 검색하거나 알고 있으면 쉽게 풀 수 있는 문제다. PNG파일을 다운받아보자.

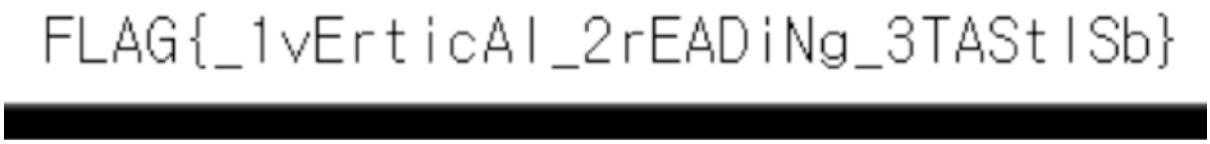




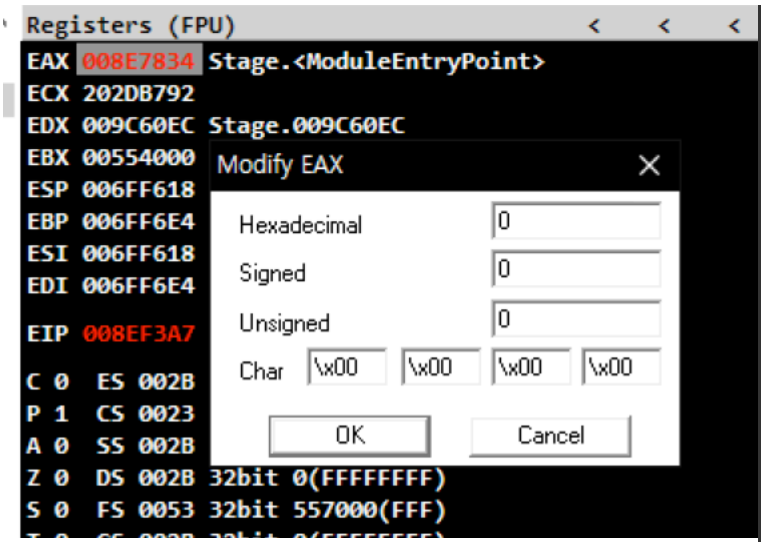
위와 같은 사진이 뜨는데, notepad를 사용하라고 한다. 하지만 필자는 더 쉬운 풀이를 위해 Hex Editor로 파일을 열어보았다.

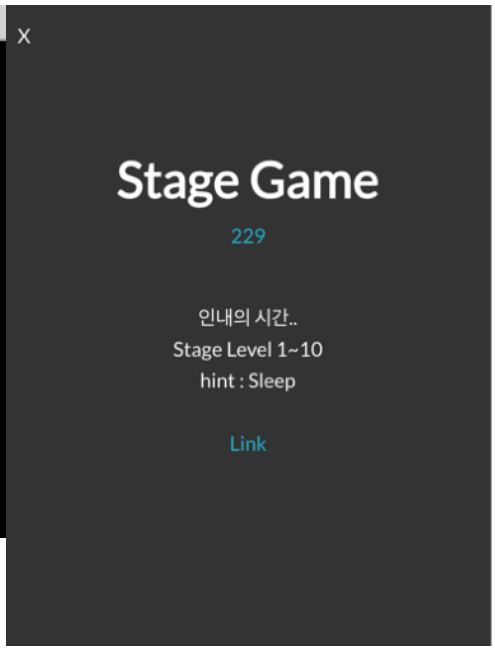


위의 사진에서 보이는 IHDR에서부터 4바이트가 높이에 대한 설정값이다. 이를 2000의 16진수 값인 07D0으로 바꾸어 주었다.



사진의 하단에 FLAG가 보인다.

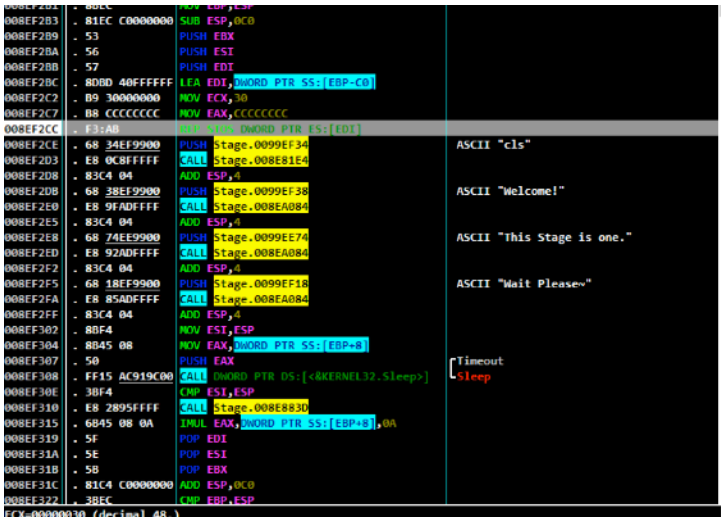




Reversing Write-up:

1. Stage Game

문제 프로그램을 다운받아 실행시켜 보면, 1level부터 10level까지 Sleep 함수를 호출하며 최종 레벨까지 도달하면 플래그가 나오게 설계되어 있는 것 같다. 모든 시간을 다 기다리면 CTF가 종료될 것 같으니 OllyDbg로 열어줘 흐름을 바꿔주도록 하자.



디버깅을 진행하다 보면, Stage가 넘어가면서 Sleep함수를 호출하는 부분이 반복적으로 나타나게 되는데, 이 때, 모든 함수에서 Sleep함수 인자를 0으로 바꿔주어 기다리는 시간이 없도록 해주자.

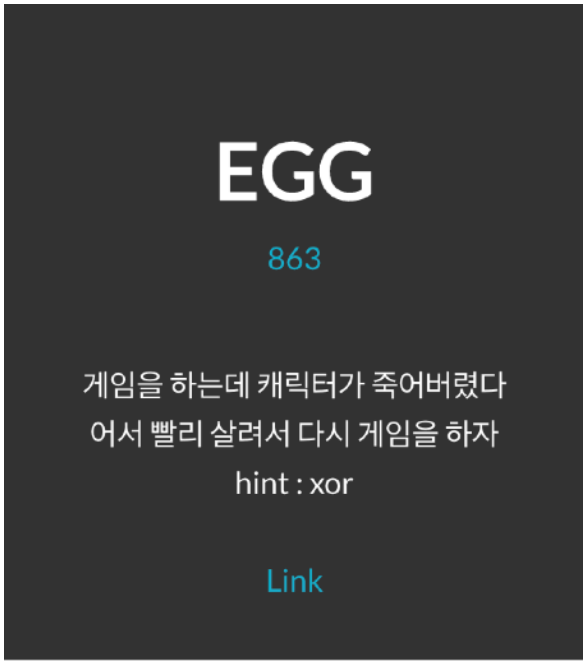
여기서 중요한 것은 Sleep함수를 호출하기 바로 전 들어가는 EAX의 값을 0으로 바꿔줘야 한다는 것이다. 10레벨까지 끝내면 플래그가 나온다.

FLAG{Y0ur_p4t1enc3_1s_gr3at!}

2. EGG

놀랍게도 익숙한 PE구조가 아닌 ELF 기반 파일이다. IDA Pro로 Disassemble 해 보니 사용자에게 입력받은 값을 XOR해 특정 값과 비교하고 있었다. 결론은 FLAG를 입력하면 알이 부화하게 된다.

```
1  __int64 __fastcall r(unsigned int a1)
2  {
3    unsigned int v2; // [sp+0h] [bp-14h]@1
4    int k; // [sp+8h] [bp-Ch]@5
5    int j; // [sp+Ch] [bp-8h]@3
6    int i; // [sp+10h] [bp-4h]@1
7
8    v2 = a1;
9    for ( i = 0; i < y; ++i )
10   {
11     v2 ^= y;
12     for ( j = 0; j < m; ++j )
13     {
14       v2 ^= m;
15       for ( k = 0; k < d; ++k )
16         v2 ^= d;
17     }
18   }
19   return v2;
20 }
```



나는 저 소스코드를 분석한 후, 최종적으로 비교하는 문자열을 같은 XOR 논리를 적용하여 플래그를 찾아내기로 설계했다.

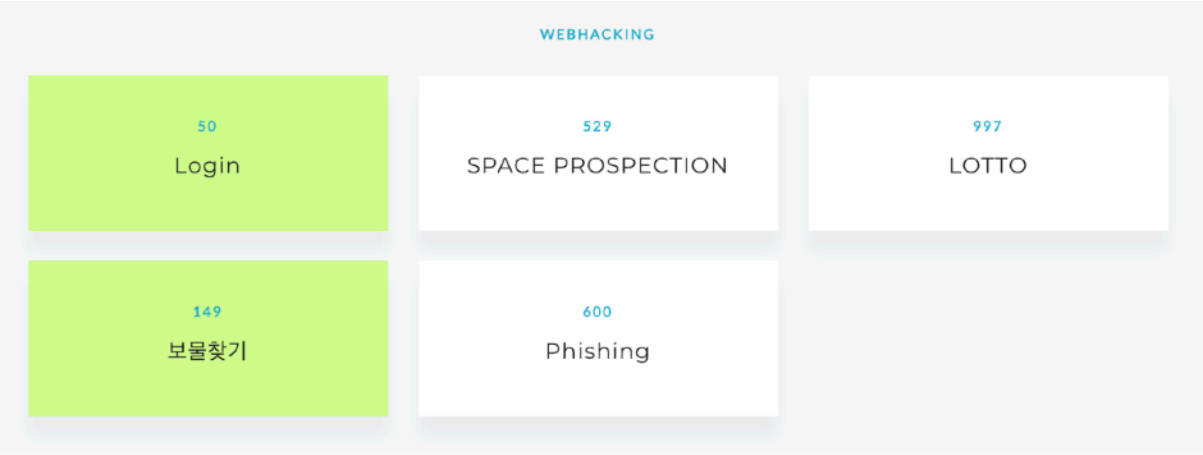
아래는 작성한 Script이다.

```
arr = 'Mh;y;mR1@OijQhHW6Ah=hB'
result = ""


def sha(a):
    v2 = [8,9,0xF,5,0xB,1,3,5,0xD,1,0xE,3,4,0xE,7,4,0xC1,6,6,0xC,0xF,0xA,0xC,0xA,2,0xD,8,7,3,7,1,7,5,8,0xC,3,4,0xB,0xE,5,1,0xB,0,3,9,
    return v2[r(a)]

def r(a):
    y = 17
    m = 12
    d = 21

    v = a

    for i in range(17): #0~16
        v ^= y
        for j in range(12):
            v ^= m
            for k in range(21):
                v ^= d

    return v

print("FLAG={",end='')
for i in range(22): #0~21
    tmp = ord(arr[i])^int(str(sha(i+2)))
    print(chr(tmp),end='')

print("}")
```

```
FLAG={An1v1a_3GGniViA_3Ni6mA}
>>>
```

FLAG={An1v1a_3GGniViA_3Ni6mA}

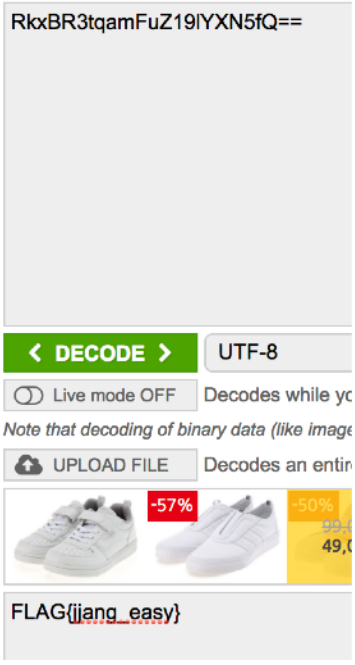
좀 노가다가 있었지만 꽤 재미있게 풀었던 것 같다.



WebHacking Write-up:

1. Login

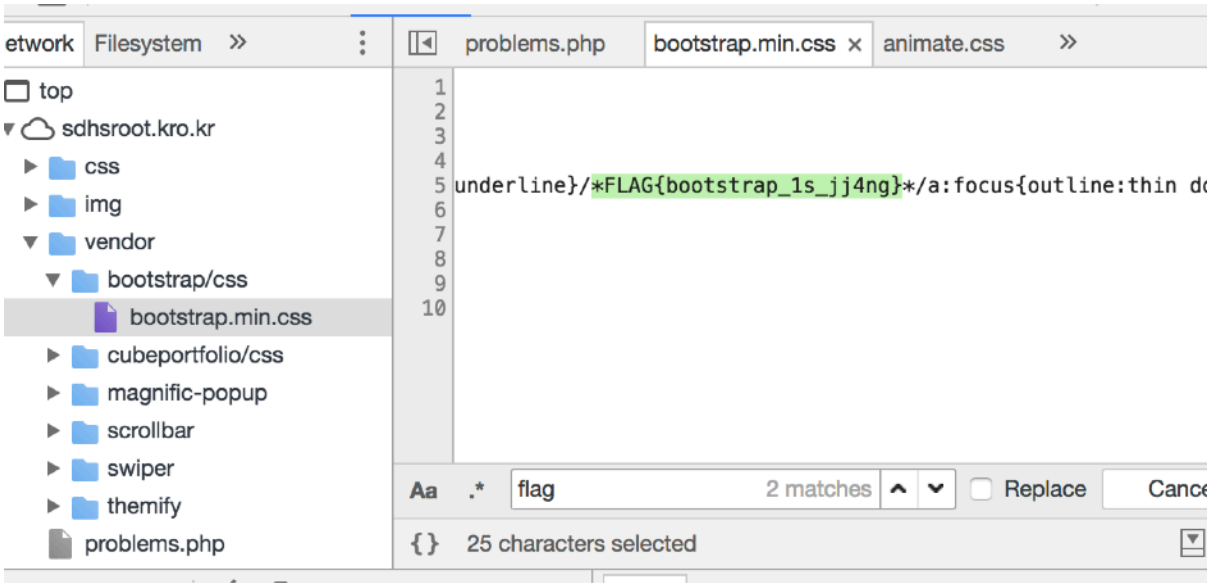
사실 이 문제는 힌트가 공개되기 전까진 너무 어려워서 보류하고 있었는데, 힌트가 공개되다가 어쩌다보니 화이트박스 문제가 되어있었다. setcookie의 값이 딱봐도 Base64 인코딩인 것 같으니 저 문자열을 여러 번 Decode하자.



FLAG{jjang_easy}

2. 보물찾기

어디에 있을지 잘 찾아보니 Bootstrap.min.css 파일에 주석으로 플래그가 숨어있었다.



FLAG{bootstrap_1s_jj4ng}