Reversing.kr

Easy Keygen Write Up

Author : hideroot(M.O.K)

Data : 2017/11/29

오랜만에 리버싱 문제를 풀었다. 문제가 리버싱 문제인지 암호학 문제인지 모르겠다. 암호학이 더 맞는것 같다. 일단 Reversing.kr 에서 Easy Keygen을 받아서 뜯어보자.

```
v9 = 0;
v13 = 0;
memset(&v10, 0, 0x60u);
v11 = 0;
v12 = 0;
memset(&v14, 0, 0xC4u);
v15 = 0;
v16 = 0;
v6 = 16;
v7 = 32;
v8 = 48;
print(aInputName);
scanf(aS, &v9);
v3 = 0;
for ( i = 0; v3 < (signed int)strlen(&v9); ++i )
{
  if ( i >= 3 )
    i = 0;
  sprintf(&v13, aS02x, &v13, *(&v9 + v3++) ^ *(&v6 + i));
}
memset(&v9, 0, 0x64u);
print(aInputSerial);
scanf(aS, &v9);
if ( !strcmp(&v9, &v13) )
{
  print(aCorrect);
  result = 0;
}
else
{
  print(aWrong);
  result = 0;
}
return result;
```

메인 함수로 들어가보면 input name과 input Serial이 있는데 input name 으로 준 문자열을 xor 방식의 암호화를 통해서 Serial을 만들고 이 Serial과 입력한 Serial을 비교해서 맞으면 correct 틀리면 Wrong을 출력한다.

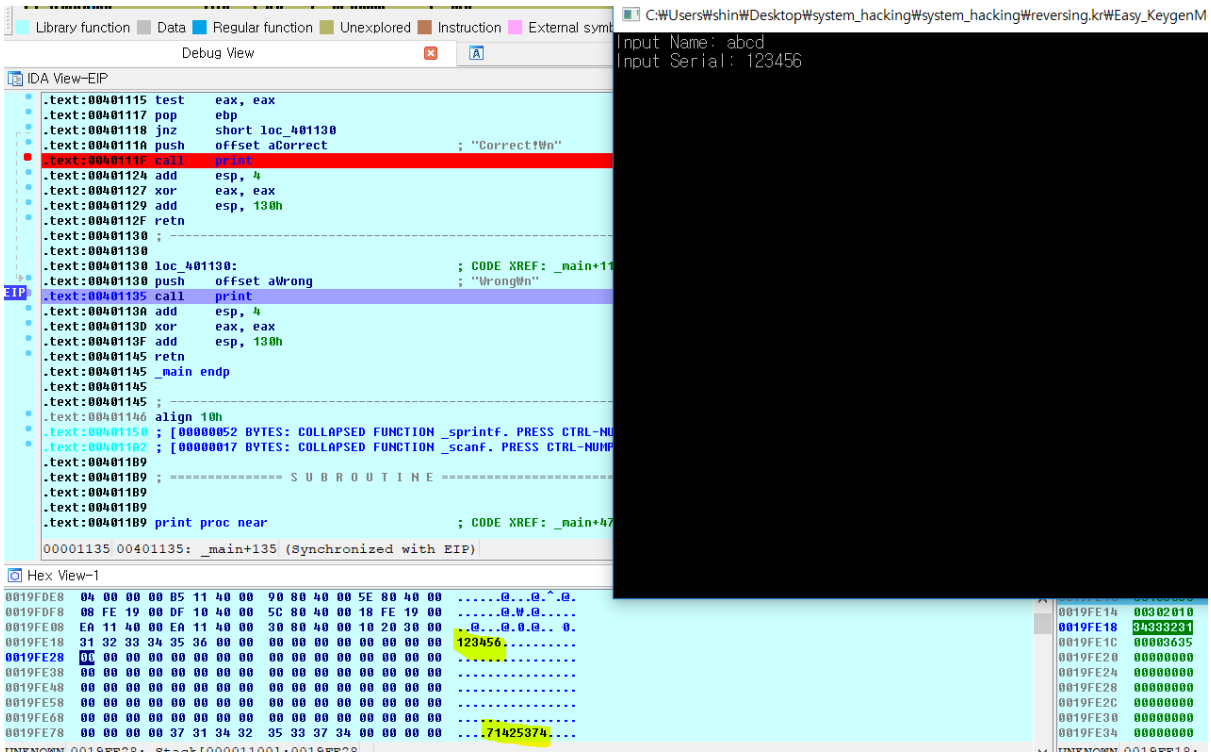
ReadMe - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

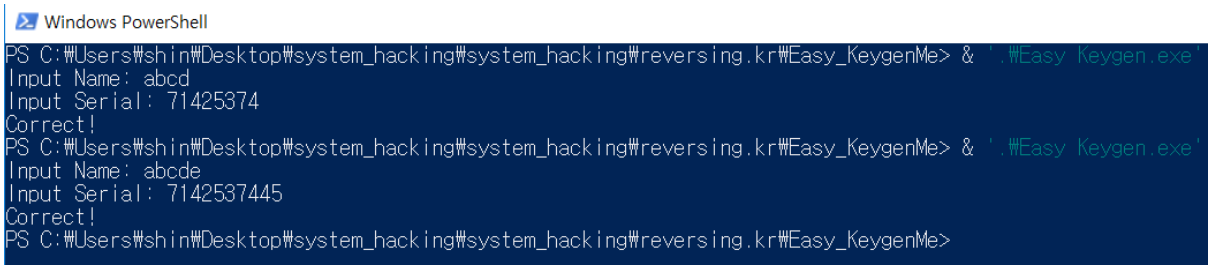
ReversingKr KeygenMe

Find the Name when the Serial is 5B134977135E7D13

자 그럼 문제를 보자 다음과 같은 Serial을 갖는 Name을 찾으라고 한다. 아니 이게 암호학 문제지 리버싱 문제인가 싶다. 암호학은 기본적으로 Brute Forcing이 기본이다. 일단 위에 Main에서는 XOR 암호화라는 건 금방 알 수 있다.' ^ ' 를 통해서 xor을 하니까... 자 그럼 풀어보자



실제로 값을 넣고 디버깅을 해보자. Name에는 abcd를 넣어주었고 serial에는 123456을 넣었다. 그리고 틀렸다는 문자열 출력전에 bp를 걸고 Stack을 확인해봤더니 abcd는 71425374로 암호화된 시리얼이랑 비교를 한다. 확인해 보자.



실제로 저 시리얼을 넣어주면 Correct를 보여준다. 자 그럼 이걸 어떻게 역으로 찾아내느냐 이건데 아까 말한 brute Forcing으로 풀어보자. 조금 풀어서 설명을 해보겠다. Sprintf의 definition은 구글링을 해보기 바란다. 수식에 대해서만 자세히 설명하겠다

*(&v9 + v3++) ^ (&v6 + I)

이 수식이 중요하다. *(&V9 + V3++)은 문자열에서 문자 마다 암호화 하는 것이니까 건너 뛰고 뒤쪽이 중요하다. 뒤쪽 수식과 앞의 문자마다 XOR 을 해준다 (^ 표시는 XOR 표시) 자 그럼 어떤 값하고 XOR을 하느냐 이건데 i는 for문에서의 index number가 된다. 앞쪽 v6는 앞의 코드를 보면 16으로 되어있다. 16진수로 나타내면 0x10이다 여기다가 i가 0일 때부터 3까지 더해주다가 3되면 다시 0으로 바뀐다. 자 여기서 0x10 + 1 을 하면 어떤 값을 출력을 할까? 이게 중요하다 0x11이 아니라 0x20이 된다. (궁금하면 직접 C에다가 Sprint로 출력해보길 바란다) 다시 정리 해보자.

(CHAR) ^ (0x10 + I) 단, I는 0부터 3까지다.

If문 을 잘보면 3이 되면 0이 되므로 실제적으로는 2까지 들어 간다. 값을 예상 해보자. 첫 번째 문자는 0x10과 xor을 하고 두 번째 문자는 0x20과 xor을 하고 세 번째 문자는 0x30과 xor을 한다. 그리고 다시 네 번째 문자는 0x10과 xor을 한다. 자 이제 답은 다나왔다. 이걸 python으로 코드를 짜보자. Brute Force를 하기로 했으니까 ASCII 전체를 돌리... 진 않고 Capital 하고 low만 돌려보자.

```
s = "5B134977135E7D13"
count = 0
for j in range(0,3) :
        print "-"*10, "count : 0x%02x" % count
        for i in range(97,122) :
                print chr(i), hex(ord(chr(i))^(int(0x10)+count))
        for k in range(65,90) :
                print chr(k), hex(ord(chr(k))^(int(0x10)+count))
        count = count + 0x10
        if count >= 0x30 : count = 0
print hex(ord('3')^int(0x20))
```

97 부터 122는 low , 65 부터 90은 Captial 이다. 돌리고 나서 매칭을 하나씩 해보면 값이 나올텐대 여기서 0x13은 안나 올 것이다. 근데 대충 예상을 해보면 K?yg?nm? 인데 ? 는 왠지 느낌이 오지 않는가? Reversing.kr은 실제 CTF에서 자주쓰는 문자처럼 o를 0(숫자)으로 E를 3으로 나타낸다. 근데 K?y 딱보면 너무 뻔해 보인다. 그래서 숫자 3을 0x20에서 돌려보니 0x13을 정확히 뱉었

다. 그래서 flag는

K3yg3nm3