rev-basic-4 문제 write-up

강승민

```
__int64 __fastcall sub_140001000(__int64 a1)
{
  int i; // [rsp+0h] [rbp-18h]

  for ( i = 0; (unsigned __int64)i < 0x1C; ++i )
  {
    if ( ((unsigned __int8)(16 * *(_BYTE *)(a1 + i)) | ((int)*(unsigned __int8 *)(a1 + i) >> 4)) != byte_140003000[i] )
      return 0i64;
  }
  return 1i64;
}
```

문제를 IDA로 열어서 분석을 해보면 위 함수의 결과에 따라 Correct, Wrong을 표기해준다.

이 함수는 입력 값에 16을 곱한 값과 입력 값에 오른쪽 쉬프트 4번 한 값을 or 연산하여 아래 그림의 배열과 비교한다.

```
.data:0000000140003000 ; unsigned __int8 byte_140003000[32]
.data:0000000140003000 byte_140003000  db 24h, 27h, 13h, 2 dup(0C6h), 13h, 16h, 0E6h, 47h, 0F5h
.data:0000000140003000                                                 ; DATA XREF: sub_140001000+50↑o
.data:0000000140003000                 db 26h, 96h, 47h, 0F5h, 46h, 27h, 13h, 2 dup(26h), 0C6h
.data:0000000140003000                 db 56h, 0F5h, 2 dup(0C3h), 0F5h, 2 dup(0E3h), 5 dup(0)
```

이를 계산해보면 1바이트 기준으로 이진수의 앞의 4자리와 뒤의 4자리를 바꾸는 계산이다.
(16을 곱한 값은 왼쪽 쉬프트 4번과 같다.)

이를 파이썬으로 적어보면

```
temp = [0x24,0x27,0x13,0xC6,0xC6,0x13,0x16,0xE6,0x47,0xF5,0x26,0x96,0x47,0xF5,0x46,0x27,0x13,0x26,0x26,0xC6,0x56,0xF5,0xC3,0xC3,0xF5,0xE3,0xE3,0x0,0x0,0x0,0x0,0x0]

for i in range(0x1C):
    print(chr(((temp[i]<<4)|(temp[i]>>4))&0b11111111),end="")
```

이와 같고 1바이트만 끊어서 보기위해 11111111 과 and 연산하여 출력해보면

```
┌──(ubuntu㉿LAPTOP-tmdalsBoB)-[/mnt/…/CTF_Study/CTFStudy/tmdals010126/rev-basic-4]
└─$ python3 test.py
Br[redacted]
```

이와 같이 flag를 얻을 수 있다.