

rev-basic-3

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

    for ( i = 0; (unsigned __int64)i < 0x18; ++i )
    {
        if ( byte_140003000[i] != (i ^ *(unsigned __int8 *)(a1 + i)) + 2 * i )
            return 0;
    }
    return 1;
}
```

루프 검증문으로 보인다.

0x18이면 24번 검사하는 듯? 문자열 길이 24 추정

그리고 if 조건문은 좀 복잡한데, 차근차근 분해해 보자.

```
; unsigned __int8 byte_140003000[32]
byte_140003000 db 49h, 60h, 67h, 74h, 63h, 67h, 42h, 66h, 80h, 78h, 2 dup(69h)
; DATA XREF: sub_140001000+28↑o
db 7Bh, 99h, 6Dh, 88h, 68h, 94h, 9Fh, 8Dh, 4Dh, 0A5h, 9Dh
db 45h, 8 dup(0)
qword_140003020 dq 0FFFFFFD466D2205DCDh ; DATA XREF: __report_gsfailure+B4↑r
; __security_init_cookie+9F↑w
```

우선 byte_1400.... 문자열

그러니까 문자열 원본 origin이라고 하겠음 대강

if origin[1] \neq $(1 \wedge (a1 + 1) + 2 * 1) \rightarrow$ 루프문 종료 (wrong)

a1이야 당연히 입력값

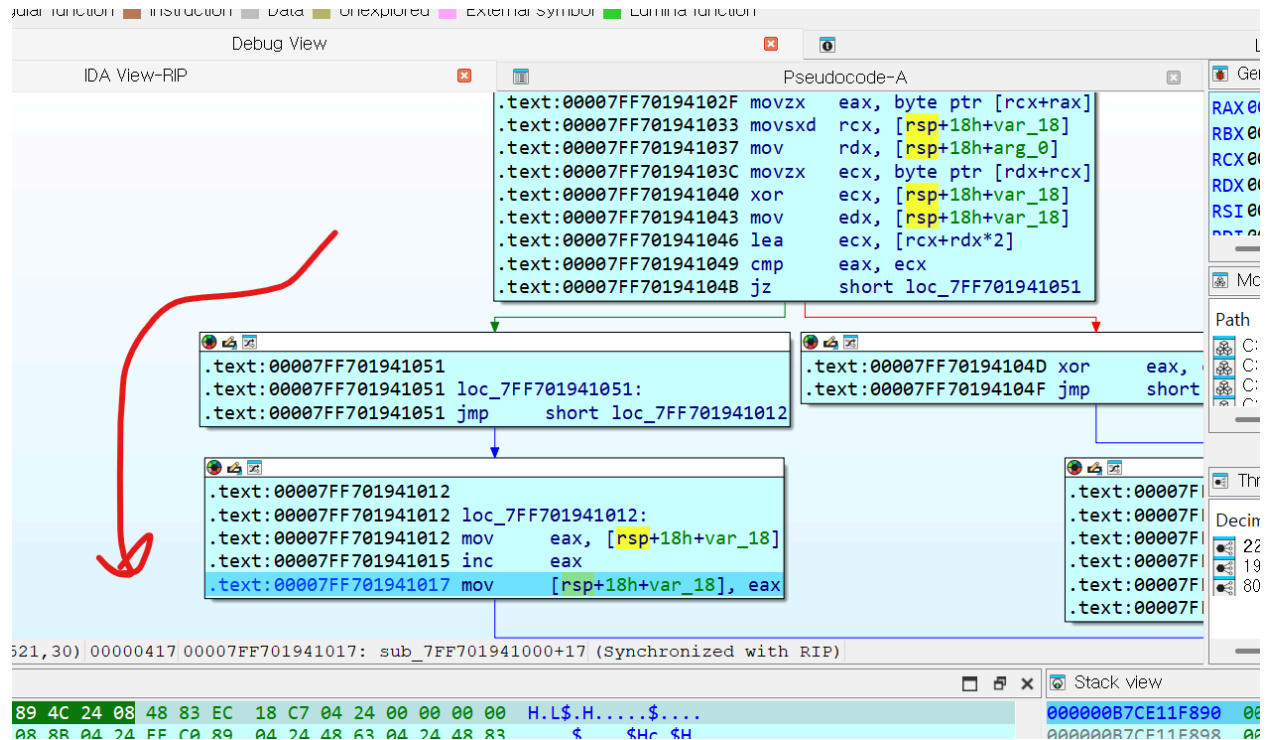
그러니까 입력값에 1을 더해서 1과 xor 한 값에 2*1을 더한 값이 origin[1]과 같아야 한다고?

이걸 역으로 파이썬 코드르 짜서 원본 hex스값이 나오게 하던가, 아니면

일단 해보자

차례대로 라고 하면

jnb → a가 b보다 작거나 같을 때



이렇게 오면 eax 값 1 증가하면서 루프문이 진행되는 듯하다....

그리고 진행하니까 49h , 60h 순으로 검사하는 거 보면 그냥 그 순서대로 해도 될 듯
한번 더 해서 rax (eax)에

```

; unsigned __int8 byte_140003000[32]
byte_140003000 db 49h, 60h, 67h, 74h, 63h, 67h, 42h, 66h, 80h, 78h, 2 dup(69h)
; DATA XREF: sub_140001000+28↑
db 7Bh, 99h, 6Dh, 88h, 68h, 94h, 9Fh, 8Dh, 4Dh, 0A5h, 9Dh
db 45h, 8 dup(0)
word_140003020 dq 0FFFFD466D2205DCDh ; DATA XREF: __report_gsfailure+B4↑
; __security_init_cookie+9F↑w

```

이거 49, 60, 67... 순서대로 담기는 거 확인함.

그럼 이제 파이썬 코드 짜면 될 듯.

파이썬에 뭔가 수 모듈 있었던 것 같은데?

```
# byte_14003000 데이터 (16진수 값들을 10진수로 변환)
byte_data = [
    0x49, 0x60, 0x67, 0x74, 0x63, 0x67, 0x42, 0x66, 0x80, 0x78, 0x69, 0x69,
    # 첫 번째 줄
    0x7B, 0x99, 0x6D, 0x88, 0x68, 0x94, 0x9F, 0x8D, 0x4D, 0x0A, 0x9D,    #
    두 번째 줄 일부
    0x45, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00                    # 나머지 (0
    으로 패딩)
]
```

```
def reverse_calculate():
```

```
    result = []
```

```
    # i=0일 때는 특별 처리 (0으로 나누기 방지)
```

```
    # 원래 조건: byte_data[0] == (0 * input[0]) + 2 * 0 = 0
```

```
    # 하지만 byte_data[0] = 0x49이므로 이 경우는 다른 처리가 필요할 수 있음
```

```
    for i in range(len(byte_data)):
```

```
        if i == 0:
```

```
            # i=0인 경우, 원래 공식이 0이 되어야 하는데 실제로는 0x49
```

```
            # 이는 특별한 경우이거나 데이터 해석에 문제가 있을 수 있음
```

```
            # 일단 직접 값으로 설정
```

```
            if byte_data[i] != 0:
```

```
                result.append(byte_data[i]) # 또는 다른 로직 적용
```

```
            continue
```

```
        if byte_data[i] == 0:
```

```
            break # 0이면 문자열 끝
```

```
        # 공식: byte_data[i] = (i * input[i]) + 2 * i
```

```
        # 역산: input[i] = (byte_data[i] - 2 * i) / i
```

```
        calculated_value = (byte_data[i] - 2 * i) // i
```

```
        # ASCII 범위 체크
```

```
        if 32 <= calculated_value <= 126:
```

```

        result.append(calculated_value)
        print(f"i={i}: byte_data[{i}]=0x{byte_data[i]:02x}({byte_data[i]}) → input[{i}]={calculated_value} ('{chr(calculated_value)}')")
    else:
        print(f"i={i}: byte_data[{i}]=0x{byte_data[i]:02x}({byte_data[i]}) → input[{i}]={calculated_value} (비정상적인 ASCII 값)")

# 결과를 문자열로 변환
try:
    decoded_string = ''.join(chr(x) for x in result if 32 <= x <= 126)
    print(f"\n복원된 문자열: '{decoded_string}'")
    return decoded_string
except:
    print("문자열 변환 실패")
    return result

def verify_result(input_string):
    """결과 검증"""
    print(f"\n검증: '{input_string}'")
    for i in range(min(len(input_string), len(byte_data))):
        if i == 0:
            continue
        expected = (i * ord(input_string[i])) + 2 * i
        actual = byte_data[i]
        match = "✓" if expected == actual else "×"
        print(f"i={i}: {match} expected={expected}, actual={actual}")

if __name__ == "__main__":
    print("=== 역산 계산 ===")
    decoded = reverse_calculate()

    if isinstance(decoded, str) and decoded:
        verify_result(decoded)

```

일단 claude 시켰는데 결과가 영 마뜩치 않았다.

=====

=== 역산 계산 ===

```
i=1: byte_data[1]=0x60(96) -> input[1]=94 ('^')
i=2: byte_data[2]=0x67(103) -> input[2]=49 ('1')
i=3: byte_data[3]=0x74(116) -> input[3]=36 ('$')
i=4: byte_data[4]=0x63(99) -> input[4]=22 (비정상적인 ASCII 값)
i=5: byte_data[5]=0x67(103) -> input[5]=18 (비정상적인 ASCII 값)
i=6: byte_data[6]=0x42(66) -> input[6]=9 (비정상적인 ASCII 값)
i=7: byte_data[7]=0x66(102) -> input[7]=12 (비정상적인 ASCII 값)
i=8: byte_data[8]=0x80(128) -> input[8]=14 (비정상적인 ASCII 값)
i=9: byte_data[9]=0x78(120) -> input[9]=11 (비정상적인 ASCII 값)
i=10: byte_data[10]=0x69(105) -> input[10]=8 (비정상적인 ASCII 값)
i=11: byte_data[11]=0x69(105) -> input[11]=7 (비정상적인 ASCII 값)
i=12: byte_data[12]=0x7b(123) -> input[12]=8 (비정상적인 ASCII 값)
i=13: byte_data[13]=0x99(153) -> input[13]=9 (비정상적인 ASCII 값)
i=14: byte_data[14]=0x6d(109) -> input[14]=5 (비정상적인 ASCII 값)
i=15: byte_data[15]=0x88(136) -> input[15]=7 (비정상적인 ASCII 값)
i=16: byte_data[16]=0x68(104) -> input[16]=4 (비정상적인 ASCII 값)
i=17: byte_data[17]=0x94(148) -> input[17]=6 (비정상적인 ASCII 값)
i=18: byte_data[18]=0x9f(159) -> input[18]=6 (비정상적인 ASCII 값)
i=19: byte_data[19]=0x8d(141) -> input[19]=5 (비정상적인 ASCII 값)
i=20: byte_data[20]=0x4d(77) -> input[20]=1 (비정상적인 ASCII 값)
i=21: byte_data[21]=0x0a(10) -> input[21]=-2 (비정상적인 ASCII 값)
i=22: byte_data[22]=0x9d(157) -> input[22]=5 (비정상적인 ASCII 값)
i=23: byte_data[23]=0x45(69) -> input[23]=1 (비정상적인 ASCII 값)
```

복원된 문자열: '^1\$'

검증: '^1\$'

```
i=1: ✓ expected=96, actual=96
i=2: ✗ expected=102, actual=103
i=3: ✗ expected=114, actual=116
```

흠....

아니 클라우드가 md 문법 때문에 수식 제대로 못 알아들어서 난 오류였음

sub_140001000 에서 수행하는 함수는 `if (byte_14003000[i] != (i ^ (unsigned __int8) (a1 + i)) + 2 * 1)` 이야! md 문법때문에 오류가 있었던 것 같아. ``의 경우엔 md 문법으로 인식 되길래 일단 ``로 감쌌어.

라고 고쳐주니까 결과 잘 나왔다!!

```
# byte_14003000 데이터 (16진수 값들을 10진수로 변환)
```

```
byte_data = [
```

```
    0x49, 0x60, 0x67, 0x74, 0x63, 0x67, 0x42, 0x66, 0x80, 0x78, 0x69, 0x69,
```

```

# 첫 번째 줄
    0x7B, 0x99, 0x6D, 0x88, 0x68, 0x94, 0x9F, 0x8D, 0x4D, 0x0A, 0x9D,    #
두 번째 줄 일부
    0x45, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00    # 나머지 (0
으로 패딩)
]

```

```

def reverse_calculate():

```

```

    result = []

```

```

    # i=0일 때는 특별 처리 (0으로 나누기 방지)

```

```

    # 원래 조건: byte_data[0] == (0 * input[0]) + 2 * 0 = 0

```

```

    # 하지만 byte_data[0] = 0x49이므로 이 경우는 다른 처리가 필요할 수 있음

```

```

    for i in range(len(byte_data)):

```

```

        if i == 0:

```

```

            # i=0인 경우, 원래 공식이 0이 되어야 하는데 실제로는 0x49

```

```

            # 이는 특별한 경우이거나 데이터 해석에 문제가 있을 수 있음

```

```

            # 일단 직접 값으로 설정

```

```

            if byte_data[i] != 0:

```

```

                result.append(byte_data[i]) # 또는 다른 로직 적용

```

```

            continue

```

```

        if byte_data[i] == 0:

```

```

            break # 0이면 문자열 끝

```

```

    # 공식: byte_data[i] = (i ^ input[i]) + 2 * i

```

```

    # 역산: input[i] = i ^ (byte_data[i] - 2 * i)

```

```

    calculated_value = i ^ (byte_data[i] - 2 * i)

```

```

    # ASCII 범위 체크

```

```

    if 32 <= calculated_value <= 126:

```

```

        result.append(calculated_value)

```

```

        print(f"i={i}: byte_data[{i}]=0x{byte_data[i]:02x}({byte_data[i]}) → input[{i}]=
{calculated_value} ('{chr(calculated_value)}')")

```

```

    else:

```

```
print(f"i={i}: byte_data[{i}]=0x{byte_data[i]:02x}({byte_data[i]}) → input[{i}]={calculated_value} (비정상적인 ASCII 값)")
```

```
# 결과를 문자열로 변환
```

```
try:
```

```
    decoded_string = ''.join(chr(x) for x in result if 32 <= x <= 126)
```

```
    print(f"\n복원된 문자열: '{decoded_string}'")
```

```
    return decoded_string
```

```
except:
```

```
    print("문자열 변환 실패")
```

```
    return result
```

```
def verify_result(input_string):
```

```
    """결과 검증"""
```

```
    print(f"\n검증: '{input_string}'")
```

```
    for i in range(min(len(input_string), len(byte_data))):
```

```
        if i == 0:
```

```
            continue
```

```
            expected = (i ^ ord(input_string[i])) + 2 * i
```

```
            actual = byte_data[i]
```

```
            match = "✓" if expected == actual else "×"
```

```
            print(f"i={i}: {match} expected={expected}, actual={actual}")
```

```
if __name__ == "__main__":
```

```
    print("=== 역산 계산 ===")
```

```
    decoded = reverse_calculate()
```

```
    if isinstance(decoded, str) and decoded:
```

```
        verify_result(decoded)
```

```

===== RESTART: C:/Users/노희민/OneDrive/바탕 화면/solve.py =====
=====
=== 역산 계산 ===
i=1: byte_data[1]=0x60(96) -> input[1]=95 ('_')
i=2: byte_data[2]=0x67(103) -> input[2]=97 ('a')
i=3: byte_data[3]=0x74(116) -> input[3]=109 ('m')
i=4: byte_data[4]=0x63(99) -> input[4]=95 ('_')
i=5: byte_data[5]=0x67(103) -> input[5]=88 ('X')
i=6: byte_data[6]=0x42(66) -> input[6]=48 ('0')
i=7: byte_data[7]=0x66(102) -> input[7]=95 ('_')
i=8: byte_data[8]=0x80(128) -> input[8]=120 ('x')
i=9: byte_data[9]=0x78(120) -> input[9]=111 ('o')
i=10: byte_data[10]=0x69(105) -> input[10]=95 ('_')
i=11: byte_data[11]=0x69(105) -> input[11]=88 ('X')
i=12: byte_data[12]=0x7b(123) -> input[12]=111 ('o')
i=13: byte_data[13]=0x99(153) -> input[13]=114 ('r')
i=14: byte_data[14]=0x6d(109) -> input[14]=95 ('_')
i=15: byte_data[15]=0x88(136) -> input[15]=101 ('e')
i=16: byte_data[16]=0x68(104) -> input[16]=88 ('X')
i=17: byte_data[17]=0x94(148) -> input[17]=99 ('c')
i=18: byte_data[18]=0x9f(159) -> input[18]=105 ('i')
i=19: byte_data[19]=0x8d(141) -> input[19]=116 ('t')
i=20: byte_data[20]=0x4d(77) -> input[20]=49 ('1')
i=21: byte_data[21]=0x0a(10) -> input[21]=-11 (비정상적인 ASCII 값)
i=22: byte_data[22]=0x9d(157) -> input[22]=103 ('g')
i=23: byte_data[23]=0x45(69) -> input[23]=0 (비정상적인 ASCII 값)


복원된 문자열: 'I_am_X0_xo_Xor_eXcit1g'
거즈: 'I_am_X0_xo_Xor_eXcit1g'

```

1 LEVEL 1 rev-basic-3 문제를 해결했습니다.

대단해요. 문제를 어떻게 해결하셨나요?
풀이를 작성하면 포인트까지 받을 수 있어요.

괜찮아요

 풀이 작성하기

역산 코드 어떻게 짤 건지는 좀 이따 살펴보자....