```
64   v31 = 0LL;
65   v32 = 0LL;
66   v33 = 0LL;
67   v34 = 0LL;
68   v35 = 0LL;
69   printf("Enter the password: ");
70   fgets(s, 256, stdin);
71   v36 = check(s);
72   if ( v36 == 1 )
73   {
74     puts("Wrong :(");
75     return 1;
76   }
77   else
78   {
79     puts("Correct!! :D");
80     return 0;
81   }
82 }
```

IDA에서 보면 사용자 입력을 check 함수 인자로 넘겨준다.

```
14   if ( strlen(a1) != 27 )
15     return 1LL;
16   v3 = 0x617B2375F81EA7E1LL;
17   v4[0] = 0xD269DF5B5AFC9DB9LL;
18   *(v4 + 7) = 0xF467EDF4ED1BFED2LL;
```

Check 함수에서 초기에 사용자의 입력 길이가 27인지 확인한다.

```
16   v3 = 0x617B2375F81EA7E1LL;
17   v4[0] = 0xD269DF5B5AFC9DB9LL;
18   *(v4 + 7) = 0xF467EDF4ED1BFED2LL;
19   v11 = 0;
20   v10 = 0;
21   v7 = 0;
22   for ( i = 0; i <= 0x16; ++i )
23   {
24     for ( j = 0; j <= 7; ++j )
25     {
26       if ( !v10 )
27         v10 = 1;
28       v6 = 1 << (7 - j);
29       v5 = 1 << (7 - v10);
30       if ( (v6 & *(&v4[-1] + i)) > 0 != (v5 & a1[v11]) > 0 )
31         return 1LL;
32       if ( ++v10 == 8 )
```

그리고 반복문을 돌며 사용자 입력과 이미 저장해둔 v3, v4 값과 비교한다.

```
v6 = 1 << (7 - j);
v5 = 1 << (7 - v10);
if ( (v6 & *(&v4[-1] + i)) > 0 != (v5 & a1[v11]) > 0 )
  return 1LL;
```

핵심 코드는 이 부분이다.

v5 부분에서 연산한 비트 값을 사용자가 입력한 특정 인덱스 값과 and 연산한 후 v4 값과 비교한다. 특정 위치에 있는 비트가 v4에 있는 비트값과 일치하면 pass다.

v5는 1 << 6, 1 << 5, 1 << 4, 1 << 3, 1 << 2 1 << 1, 1 << 0 의 값을 갖는다. 이 의미는 마지막 비트는 검사하지않고, 7비트까지만 검사한다는걸 의미한다.

그래서 최종 복호화 코드는 다음과 같아진다.

```
import struct



c1_bytes = 0x617B2375F81EA7E1

c2_bytes = 0xD269DF5B5AFC9DB9

c3_bytes = 0xF467EDF4ED1BFED2


c1 = struct.pack("<Q", c1_bytes)

c2 = struct.pack("<Q", c2_bytes)

c3 = struct.pack("<Q", c3_bytes)
```

```
secret = c1 + c2[:7] + c3 + b'₩x00'*4


bits = "".join(f"{byte:08b}" for byte in secret)
bits = bits[:189]


password = ""
for i in range(27):
    chunk = bits[i*7:(i+1)*7]
    char_val = int(chunk, 2)
    password += chr(char_val)


print(password)
```