future

> rust로 만들어진 바이너리이며, 암호를 입력받고 맞다면 correct를 출력, 틀리다면 wrong을 출력하고 종료한다.

Analysis

화면에서 출력되는 string을 디스어셈블러에서 검색하면 해당 함수를 찾을 수 있다. 출력 후 입력을 받기 때문에 디스어셈블러에서 찾은 주소를 디버거에 브레이크포인트로 걸고, 디버거로 쭉 진행하다보면 I/O가 블락되는 부분이 존재한다.

```
00007FF6CB5916BA | 48:8D55 18              | lea rdx,qword ptr ss:[rbp+18]
|
00007FF6CB5916BE | 4C:8D45 78              | lea r8,qword ptr ss:[rbp+78]
|
00007FF6CB5916C2 | E8 094C0000             | call <future.input>
| input
00007FF6CB5916C7 | 48:837D 30 00           | cmp qword ptr ss:[rbp+30],0
|
00007FF6CB5916CC | 0F85 6B070000           | jne future.7FF6CB591E3D
|
```

입력을 대강 넣고 살펴보면 세번 째 인자에 입력 값이 들어가는 것을 알 수 있다. 코드가 방대하기 때문에 입력 값이 변경되는 시점을 위주로 동적 디버깅을 진행하다 보면 다중 while문을 만나는데, 이 곳이 특정 루틴을 수행한 후 내부 암호값과 비교하는 부분이다.

```
while ( while1_cnt >= 8 )                       // crypt start 길이-1만큼 진행
  {
    ...
    do                                            // 10번 진행
    {
        ...
      do                                          // 10번 진행
      {
        v67 = while3_cnt;
        ++while3_cnt;
        buf_byte = _buf[_buf_index >> 3];
        calc = ((v41 & 0x94AF) != 0) & _bittest(&buf_byte, _buf_index & 7);
        v47 = crypt_value_buf[crypt_value_buf_index >> 5];
        v48 = _bittest(&v47, crypt_value_buf_index);
        v49 = v47 | (1 << crypt_value_buf_index);
        if ( !((v48 ^ calc) ^ (v39 != 0)) )
          v49 = crypt_value_buf[crypt_value_buf_index >> 5] & ~(1 <<
```

```
crypt_value_buf_index);
        crypt_value_buf[crypt_value_buf_index >> 5] = v49;
        v39 = (v39 != 0) & (v48 ^ calc) | v48 & calc;
        ++crypt_value_buf_index;
        ++_buf_index;
      }
      while ( while3_cnt != 0x10 );
      ...
    }
    while ( while2_cnt != 0x10 );
    ...
  }                                             // crypt done
```

위 루틴을 통해 생성된 값을 내부 암호값과 비교한다. 데이터 부분을 살펴보면 R/O 영역에서 암호값을 확인할 수 있다.

```
.rdata:00007FF6CB5B0450 qword_7FF6CB5B0450 dq 437D7F8C3CAC3559h ; DATA XREF:
sub_7FF6CB5915E0:loc_7FF6CB591AFF↑r
.rdata:00007FF6CB5B0458                    dq 376FBB9D1CACABF1h
.rdata:00007FF6CB5B0460 qword_7FF6CB5B0460 dq 374A8FDD3CAA774Ch ; DATA XREF:
sub_7FF6CB5915E0+52A↑r
.rdata:00007FF6CB5B0468                    dq 43A2169D443630EEh
.rdata:00007FF6CB5B0470 ; __m128i qword_7FF6CB5B0470
.rdata:00007FF6CB5B0470 qword_7FF6CB5B0470 dq 3CAC355916369111h ; DATA XREF:
sub_7FF6CB5915E0+535↑r
.rdata:00007FF6CB5B0478                    dq 42FBFB1B377337B7h
```

위 암호키 생성 루틴을 python으로 재작성하고, 입력 값을 랜덤으로 주어 올바른 암호키 값을 찾도록 brute force를 진행하면 플래그를 얻을 수 있다.

Code

```
import math
from array import array
import struct
import string

key = b"3CAC3559"
key += b"437D7F8C1CACABF1"
key += b"376FBB9D3CAA774C"
key += b"374A8FDD443630EE"
key += b"43A2169D16369111"
key += b"3CAC3559"
key += b"377337B742FBFB1B"
key += b"374A8FDD"
key += b"437BC17F"
# key = b"00249711" # ?

input_list = [int(key[i:i+8],16) for i in range(0, int(len(key)), 8)]
```

```
def _bittest(x, n):
    return (x & (1 << n)) >> n

printable = string.printable

flag = ''
toggle = False

for ip in input_list:
    for i in printable:
        for j in printable:
            dummy = "QQ"
            buf = str(i) + str(j) + dummy # hardcoding
            buf = array('B', [ord(i) for i in buf])
            v31 = 0

            _buf = buf
            while1_cnt = v31 << 6
            lpMem = 4
            v81 = 0
            v34 = 0

            lpMem_index = v34
            v35 = while1_cnt
            v36 = while1_cnt >> 3
            v38 = 0x10
            v69 = v35 & 7
            # next_buf = &_buf[(v38 + v69) >> 3]
            next_buf = _buf[(v38 + v69) >> 3]
            v71 = v38
            crypt_value_buf = array('I', [0, 0])
            crypt_value_buf[0] = 0
            v39 = 0
            crypt_value_buf_index = 0

            while True:
                while2_cnt = crypt_value_buf_index + 1
                v41 = 1 << crypt_value_buf_index
                _buf_index = v69
                v43 = v71
                while3_cnt = 0

                while True:
                    v67 = while3_cnt
                    while3_cnt+=1
                    buf_byte = _buf[_buf_index >> 3]
                    calc = ((v41 & 0x94AF) != 0) & _bittest(buf_byte, _buf_index &
7)
                    v47 = crypt_value_buf[crypt_value_buf_index >> 5]
                    v48 = _bittest(v47, crypt_value_buf_index)
                    v49 = v47 | (1 << crypt_value_buf_index)
                    if not ((v48 ^ calc) ^ (v39 != 0)):
                        v49 = crypt_value_buf[crypt_value_buf_index >> 5] & ~(1 <<
```

```
crypt_value_buf_index)
                    crypt_value_buf[crypt_value_buf_index >> 5] = v49
                    v39 = (v39 != 0) & (v48 ^ calc) | v48 & calc
                    crypt_value_buf_index+=1
                    _buf_index+=1
                    if while3_cnt == 0x10:
                        break
                crypt_value_buf_index = while2_cnt
                if while2_cnt == 0x10:
                    break

            if crypt_value_buf[0] == ip:
                print(f"FOUND!: {str(i) + str(j)}")
                flag += str(i) + str(j)
                toggle=True
                break
        if toggle:
            toggle=False
            break

print('flag: apollob{'+str(flag)+'?}')
```

```
cpstd@krrr:/mnt/c/Users/cpstd/Downloads/Future$ python find.py
FOUND!: wh
FOUND!: 4t
FOUND!: _1
FOUND!: s_
FOUND!: th
FOUND!: 3_
FOUND!: ru
FOUND!: st
FOUND!: ?&
FOUND!: wh
FOUND!: y_
FOUND!: Us
FOUND!: 3_
FOUND!: 1t
flag: apollob{wh4t_1s_th3_rust?&why_Us3_1t?}
```