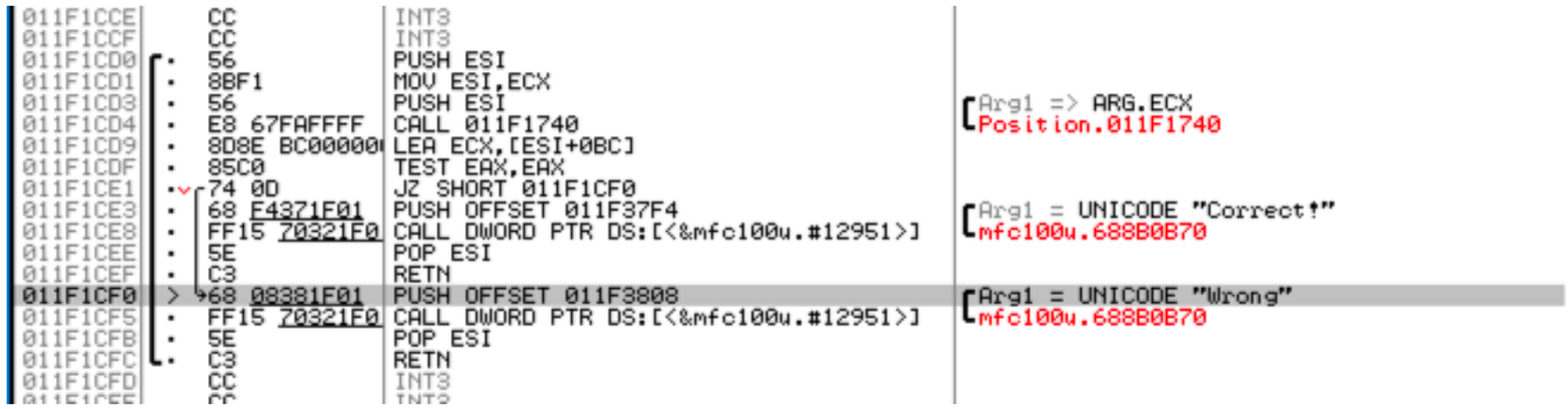
리버싱 2주차

응창

Position

- 노가다의 악몽이 떠오릅니다.



- correct와 wrong구간에서 보면 저기 위에서 TEST EAX, EAX에서 and 값이 0이면 zf가 1로 설정되서 011F1CF0 으로 가버리게 됩니다. 그러면 wrong이니깐, 반대로 and 값이 1 즉, EAX = 1 return 값으로 추정을 해보고 1이 되어야 correct가 됩니다. 그럼 저기 CALL 011F1740 부분을 확인해 보아야 겠죠.

```
011F173F     CC             INT3
011F1740  $  55             PUSH EBP                                 Position.011F1740(guessed Arg1)
011F1741  .  8BEC           MOV EBP,ESP
011F1743  .  6A FF          PUSH -1
011F1745  .  68 CB2A1F01    PUSH 011F2ACB                            Entry point
011F174A  .  64:A1 000000(  MOV EAX,DWORD PTR FS:[0]
011F1750  .  50             PUSH EAX
011F1751  .  83EC 1C        SUB ESP,1C
011F1754  .  53             PUSH EBX
011F1755  .  56             PUSH ESI
011F1756  .  57             PUSH EDI
011F1757  .  A1 28501F01    MOV EAX,DWORD PTR DS:[11F5028]
011F175C  .  33C5           XOR EAX,EBP
011F175E  .  50             PUSH EAX
011F175F  .  8D45 F4        LEA EAX,[LOCAL.3]
011F1762  .  64:A3 000000(  MOV DWORD PTR FS:[0],EAX
011F1768  .  8D4D E8        LEA ECX,[LOCAL.6]
011F176B  .  FF15 40331F0   CALL DWORD PTR DS:[<&mfc100u.#296>]      Cmfc100u.686B616E
011F1771  .  33FF           XOR EDI,EDI
011F1773  .  8D4D EC        LEA ECX,[LOCAL.5]
011F1776  .  897D FC        MOV DWORD PTR SS:[LOCAL.1],EDI
011F1779  .  FF15 40331F0   CALL DWORD PTR DS:[<&mfc100u.#296>]      Cmfc100u.686B616E
011F177F  .  8D4D F0        LEA ECX,[LOCAL.4]
011F1782  .  FF15 40331F0   CALL DWORD PTR DS:[<&mfc100u.#296>]      Cmfc100u.686B616E
011F1788  .  8B4D 08        MOV ECX,DWORD PTR SS:[ARG.1]
011F178B  .  8D45 E8        LEA EAX,[LOCAL.6]
011F178E  .  50             PUSH EAX                                 ┌Arg1 => OFFSET LOCAL.6
011F178F  .  81C1 3001000(  ADD ECX,130                              │
011F1795  .  C645 FC 02     MOV BYTE PTR SS:[LOCAL.1],2              │
011F1799  .  FF15 80321F0   CALL DWORD PTR DS:[<&mfc100u.#7006>]     └mfc100u.6889AC70
011F179F  .  8B4D E8        MOV ECX,DWORD PTR SS:[LOCAL.6]
011F17A2  .  8379 F4 04     CMP DWORD PTR DS:[ECX-0C],4
```

- 이 부분 entry point에 왔습니다. 시리얼 값에 주어진 76876-77776을 넣어줍니다. 그전에 아시다시피 문자를 브레이크포인트 시 넣을때마다 자동실행되어 한 문자씩 입력이 되는 시스템입니다.

- 4글자라는 힌트까지 주어졌고, 그러면 3글자를 넣고 ebp에 브포를 걸고 마지막 글자를 넣어봅니다.

- 넣은 값 abcd가 확인이 됩니다. 그리고 LOCAL.6 에 있구요.

- ECX-0xC 는 4임을 알 수 있습니다.

```
                    LEA ECX,[ECX]
                    PUSH ESI
8D E8               LEA ECX,[LOCAL.6]
E5 C4321F0          CALL DWORD PTR DS:[<&mfc100u.#4478>]
83F8 61             CMP AX,61
B8                  JB SHORT 011F17A8
                    PUSH ESI
8D E8               LEA ECX,[LOCAL.6]
E5 C4321F0          CALL DWORD PTR DS:[<&mfc100u.#4478>]
83F8 7A             CMP AX,7A
A8                  JA SHORT 011F17A8
                    INC ESI
7E 04               CMP ESI,4
DA                  JL SHORT 011F17E0
76                  XOR ESI,ESI
7E                  CMP EDI,ESI
1C                  JE SHORT 011F1828
                    PUSH ESI
8D E8               LEA ECX,[LOCAL.6]
E5 C4321F0          CALL DWORD PTR DS:[<&mfc100u.#4478>]
                    PUSH EDI
8D E8               LEA ECX,[LOCAL.6]
8BD8                MOV BX,AX
E5 C4321F0          CALL DWORD PTR DS:[<&mfc100u.#4478>]
3BC3                CMP AX,BX
80                  JE SHORT 011F17A8
                    INC ESI
7E 04               CMP ESI,4
DA                  JL SHORT 011F1808
                    INC EDI
7F 04               CMP EDI,4
D2                  JL SHORT 011F1806
```

- INC ESI, CMP ESI,4 INC EDI, CMP EDI, 4 를 보면 이중 포문임이 짐작이 됩니다. 그리고 포문안에서 AX 61, AX 7A를 보면 입력받은 4자리 문자가 a~z안에 속해있는지 판별하며, 밑에 이중포문에서는 같으면 어디론가 점프해버린다는데 이를 통해 4자리 문자에 같은 문자가 있으면 안됨을 유추할수 있습니다.

```
push    0
lea     ecx, [ebp+var_18]
call    ds:?GetAt@?$CSimpleStringT@_W$00@ATL@@QBE_WH@Z ; ATL::CSimpleStringT<wchar_t,1>::GetAt(in
mov     cl, al
and     cl, 1
mov     [ebp+var_20], cl
add     [ebp+var_20], 5
mov     cl, al
shr     cl, 1
mov     dl, al
mov     bl, al
and     cl, 1
shr     dl, 2
shr     bl, 3
shr     al, 4
add     cl, 5
and     dl, 1
and     al, 1
and     bl, 1
mov     [ebp+var_1C], al
add     [ebp+var_1C], 5
mov     [ebp+var_1F], cl
add     dl, 5
add     bl, 5
push    1
lea     ecx, [ebp+var_18]
mov     [ebp+var_1E], dl
mov     [ebp+var_1D], bl
call    ds:?GetAt@?$CSimpleStringT@_W$00@ATL@@QBE_WH@Z ; ATL::CSimpleStringT<wchar_t,1>::GetAt(in
mov     cl, al
and     cl, 1
mov     [ebp+var_28], cl
inc     [ebp+var_28]
mov     cl, al
shr     cl, 1
mov     dl, al
```

```
011F1876  .  80E1 01        AND CL,01
011F1879  .  884D E0        MOV BYTE PTR SS:[LOCAL.8],CL
011F187C  .  8045 E0 05     ADD BYTE PTR SS:[LOCAL.8],5
011F1880  .  8AC8           MOV CL,AL
011F1882  .  D0E9           SHR CL,1
011F1884  .  8AD0           MOV DL,AL
011F1886  .  8AD8           MOV BL,AL
011F1888  .  80E1 01        AND CL,01
011F188B  .  C0EA 02        SHR DL,2
011F188E  .  C0EB 03        SHR BL,3
011F1891  .  C0E8 04        SHR AL,4
011F1894  .  80C1 05        ADD CL,5
011F1897  .  80E2 01        AND DL,01
011F189A  .  24 01          AND AL,01
011F189C  .  80E3 01        AND BL,01
011F189F  .  8845 E4        MOV BYTE PTR SS:[LOCAL.7],AL
011F18A2  .  8045 E4 05     ADD BYTE PTR SS:[LOCAL.7],5
011F18A6  .  884D E1        MOV BYTE PTR SS:[LOCAL.8+1],CL
011F18A9  .  80C2 05        ADD DL,5
011F18AC  .  80C3 05        ADD BL,5
011F18AF  .  6A 01          PUSH 1
011F18B1  .  8D4D E8        LEA ECX,[LOCAL.6]
011F18B4  .  8855 E2        MOV BYTE PTR SS:[LOCAL.8+2],DL
011F18B7  .  885D E3        MOV BYTE PTR SS:[LOCAL.8+3],BL
011F18BA  .  FF15 C4321F0   CALL DWORD PTR DS:[<&mfc100u.#4478>]
```

- LOCAL에 value값이 들어간다고 유추했으니 ida랑 비교해서 노가다를 해봅니다. 저는 ida가 가독성이 편해 ida에서 var_X 를 olly에 LOCAL로 다 바꿀 예정입니다.

```
LOCAL.8 = (name[0] & 1) + 5
LOCAL.8+1 = ((name[0] >> 1) & 1) + 5
LOCAL.8+2 = ((name[0] >> 2) & 1) + 5
LOCAL.8+3 = ((name[0] >> 3) & 1) + 5
LOCAL.7 = ((name[0] >> 4) & 1) + 5

LOCAL.10 = (name[1] & 1) + 1
LOCAL.10+1 = ((name[1] >> 1) & 1) + 1
ECX = ((name[1] >> 1) & 2) + 1
LOCAL.10+3 = ((name[1] >> 3) & 1) + 1
LOCAL.9 = ((name[1] >> 4) &1) + 1
```

```
itow(LOCAL.8 + ECX) == serial[0] = 7
itow(LOCAL.8+3 + LOCAL.10+3) == serial[1] = 6
itow(LOCAL.8+1 + LOCAL.9) == seiral[2] = 8
itow(LOCAL.8+2 + LOCAL.10) == serial[3] = 7
itow(LOCAL.7 + LOCAL.10+1) == serial[4] = 6

NEW LOCAL.8 = (name[2] & 1) + 5
NEW LOCAL.8+1 = ((name[2] >> 1) & 1) + 5
NEW LOCAL.8+2 = ((name[2] >> 2) & 1) + 5
NEW LOCAL.8+3 = ((name[2] >> 3) & 1) + 5
NEW LOCAL.7 = ((name[2] >> 4) & 1) + 5

NEW LOCAL.10 = (p & 1) + 1
NEW LOCAL.10+1 = ((p >> 1) & 1) + 1
NEW EDX = ((p >> 2) & 1) + 1
NEW LOCAL.10+3 = ((p >> 3) & 1) + 1
NEW LOCAL.9 = ((p >> 4) & 1) + 1
```

```
itow(NEW LOCAL.8 + NEW EDX) == serial[5] = 7
itow(NEW LOCAL.8+3 + NEW LOCAL.10+3) == serial[6] = 7
itow(NEW LOCAL.8+1 + NEW LOCAL.9) == serial[7] = 7
```

```
itow(NEW LOCAL.8 + NEW EDX) == serial[5] = 7
itow(NEW LOCAL.8+3 + NEW LOCAL.10+3) == serial[6] = 7
itow(NEW LOCAL.8+1 + NEW LOCAL.9) == serial[7] = 7
itow(NEW LOCAL.8+2 + NEW LOCAL.10) == serial[8] = 7
itow(NEW LOCAL.7 + NEW LOCAL.10+1) == serial[9] = 6
```

- LOCAL.6 이 name의 첫번째 문자인 것 같습니다. itow 는 문자를 정수로 바꿔줍니다. 여기서 시리얼의 존재이유를 알 수 있습니다.

```c
#include <stdio.h>
int main(){
    int name1, name2, name3;
    int v1,v2,v3,v4,v5,v6,v7,v8,v9,v10,v11,v12,v13,v14,v15,v16,v17,v18,v19,v20,v21;
    for(name1 = 'a'; name1 <= 'z'; name1++){
        for(name2 = 'a'; name2 <= 'z'; name2++){
            for(name3 = 'a'; name3 <= 'z'; name3++){
                v1 = (name1 & 1) + 5;
                v2 = ((name1 >> 1) & 1) + 5;
                v3 = ((name1 >> 2) & 1) + 5;
                v4 = ((name1 >> 3) & 1) + 5;
                v5 = ((name1 >> 4) & 1) + 5;
                v6 = (name2 & 1) + 1;
                v7 = ((name2 >> 1) & 1) + 1;
                v8 = ((name2 >> 2) & 1) + 1;
                v9 = ((name2 >> 3) & 1) + 1;
                v10 = ((name2 >> 4) & 1) + 1;
                v11 = (name3 & 1) + 5;
                v12 = ((name3 >> 1) & 1) + 5;
                v13 = ((name3 >> 2) & 1) + 5;
                v14 = ((name3 >> 3) & 1) + 5;
                v15 = ((name3 >> 4) & 1) + 5;
                v16 = 'p';
                v17 = (v16 & 1) + 1;
                v18 = ((v16 >> 1) & 1) + 1;
                v19 = ((v16 >> 2) & 1) + 1;
                v20 = ((v16 >> 3) & 1) + 1;
                v21 = ((v16 >> 4) & 1) + 1;
                if(v1+v8==7&&v4+v9==6&&v2+v10==8&&v3+v6==7&&v5+v7==6)
                    if(v11+v19==7&&v14+v20==7&&v12+v21==7&&v13+v17==7&&v15+v18==6)
                        printf("%c%c%cp\n",name1,name2,name3);
            }
        }
    }
}
```

- 노가다 했으니 bruteforce는 원리를 알고 v1부터 정렬해서 안 헷갈리게 만들었습니다.

- 요놈중 맨밑은 안되고 세개중 bump가 되며.. 두개는 아직도 미궁
- 이유가 혹시 있나요!?