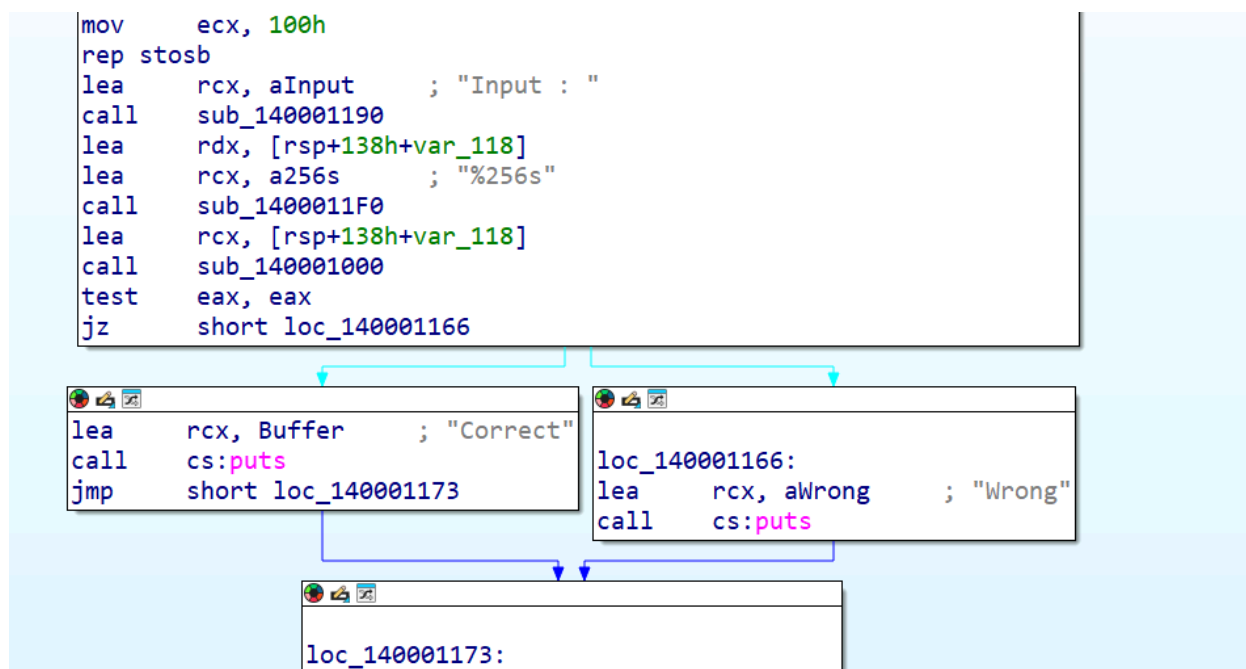


# rev-basic-0

원래는 올리디버거로 열려고 했는데 32비트 프로그램 아니라 안된다길래 강 ida로 열었음



딱 봐도 뭘 검증하고 있다...

input된 문자열을 rcx에 복사한 뒤 이런저런 검증 과정을 거치는 듯

그렇게 한 최종 값에서 `eax eax` 테스트(and)한 결과가 zero(같으면) 오답. 즉 test (and 연산) 결과가 0이 아니어야 함.

자기자신을 and해서 0이 아니려면 `eax` 에 값이 없어야 함.

아마 저 검증과정들을 역추적 하면 플래그가 나오는 구조겠지만... 일단 심심하니까 패치 먼저 해보자

일단 `jnz`로 패치하고 실행해 봤다.

그러자 그냥 종료함 ㅇ

그럼 일단 input이 거치는 검증 함수들이나 살펴보자

일단 첫번째인 sub\_7FF7D8CF1190

```
__int64 sub_7FF7D8CF1190(__int64 a1, ...)\n{\n    FILE *v1; // rax\n    va_list va; // [rsp+48h] [rbp+10h] BYREF\n\n    va_start(va, a1);\n    v1 = _acrt_iob_func(1u);\n    return (unsigned int)sub_7FF7D8CF1060(v1, a1, 0, (__int64 *)va);\n}
```

c언어 변환 결과는 이런데, 이게 더 알아보기 힘들

일단 rax에 복사된 v1은 파일 포인터(디스크립터) 같다는 것을 알 수 있따.

```

mov     [rsp+arg_0], rcx
mov     [rsp+arg_8], rdx
mov     [rsp+arg_10], r8
mov     [rsp+arg_18], r9
sub     rsp, 38h
lea     rax, [rsp+38h+arg_8]
mov     [rsp+38h+var_10], rax
mov     ecx, 1 ; Ix
call    cs:__acrt_iob_func
mov     r9, [rsp+38h+var_10]
xor     r8d, r8d
mov     rdx, [rsp+38h+arg_0]
mov     rcx, rax
call    sub_7FF7D8CF1060
mov     [rsp+38h+var_18], eax
mov     [rsp+38h+var_10], 0
mov     eax, [rsp+38h+var_18]
add     rsp, 38h
retn
sub_7FF7D8CF1190 endp

```

일단 rsp 레지스터에 변수를 쌓는 과정 같음

$[rsp+arg\_0] \leftarrow rcx$  (aInput)

위의 3줄 실행하자 아래와 같이 스택에 변수들 값이 쌓임

00000014D94FFB00	00007FF7D8CF1130	main+30
00000014D94FFBC0	00007FF7D8CF2238	.rdata:aInput
00000014D94FFBC8	0000001CBA32A8DE0	debug029:0000001CBA32A8DE0
00000014D94FFBD0	0000001CBA32AD450	debug029:0000001CBA32AD450

arg\_8 = 1c...어쩌고

arg\_10 : 1c...어쩌고

이후 sub rsp, 38h

00000014D94FFB80	0000000000000000	
00000014D94FFB88	0000000000000000	
00000014D94FFB90	0000000000000000	
00000014D94FFB98	0000000000000000	
00000014D94FFBA0	0000000000000000	
00000014D94FFBA8	0000000000000000	
00000014D94FFBB0	0000000000000000	
00000014D94FFBB8	00007FF7D8CF1138	main+38
00000014D94FFBC0	00007FF7D8CF2238	.rdata:aInput
00000014D94FFBC8	000001CBA32A8DE0	debug029:000001CBA32A8DE0

스택이 저만큼 위로 이동

아마 main에 있는 변수를 꺼내오려는 게 아닐까 싶음

lea rax [rsp+38h+arg\_8]

rax에 해당 주소 복사

RAX 00000014D94FFBC8	Stack[00004D44]:00000014D94FFBC8
----------------------	----------------------------------

mov

00000014D94FFBA0	0000000000000000	
00000014D94FFBA8	00000014D94FFBC8	Stack[00004D44]:00000014D94FFBC8
00000014D94FFBB0	0000000000000000	
00000014D94FFBB8	00007FF7D8CF1138	main+38
00000014D94FFBC0	00007FF7D8CF2238	.rdata:aInput
00000014D94FFBC8	000001CBA32A8DE0	debug029:000001CBA32A8DE0
00000014D94FFBD0	000001CBA32AD450	debug029:000001CBA32AD450
00000014D94FFBD8	0000000000000000	

mov ecx, 1

이후 call 있는 걸 보니 이 함수를 위한 인자일지도

```
#define stdout (__acrt_iob_func(1))
```

acrt\_iob\_fun(1) : 표준출력

즉 파일에 무언가를 출력하는 것으로 보임

이후 r9 레지스터에 다시 메모리값 복사

```
RIP 00007FF7D8CF11C2 SUB_7FF7D8CF1190+32
R8 000001CBA32AD450 debug029:000001CBA32AD450
R9 00000014D94FFBC8 Stack[00004D44]:00000014D94FFBC8
```

이후 r8d xor → 0

rdx로 "인풋 : " 문자열 복사

rax에 있는 dll rcx 로 옮기기

이후 call로 sub\_다른 함수를 부른다.

지금까지의 함수 내용은 cmd 창에 "input : " 문자열을 출력하는 내용으로 보임. 아마 여기서 호출하는 sub\_이 인풋값을 버퍼에 넣어 대조하는 함수이지 않을까?

```
IDA View-RIP Pseudocode-A
.text:00007FF7D8CF1060 ; __int64 __fastcall sub_7FF7D8CF1060(_QWORD, _QWORD, _QWORD, _QWORD)
.text:00007FF7D8CF1060 sub_7FF7D8CF1060 proc near
.text:00007FF7D8CF1060
.text:00007FF7D8CF1060 ArgList= qword ptr -18h
.text:00007FF7D8CF1060 Stream= qword ptr 8
.text:00007FF7D8CF1060 Format= qword ptr 10h
.text:00007FF7D8CF1060 Locale= qword ptr 18h
.text:00007FF7D8CF1060 arg_18= qword ptr 20h
.text:00007FF7D8CF1060
.text:00007FF7D8CF1060 mov     [rsp+arg_18], r9
.text:00007FF7D8CF1065 mov     [rsp+Locale], r8
.text:00007FF7D8CF106A mov     [rsp+Format], rdx
.text:00007FF7D8CF106F mov     [rsp+Stream], rcx
.text:00007FF7D8CF1074 sub     rsp, 38h
.text:00007FF7D8CF1078 call    sub_7FF7D8CF1040
.text:00007FF7D8CF107D mov     rcx, [rsp+38h+arg_18]
.text:00007FF7D8CF1082 mov     [rsp+38h+ArgList], rcx ; ArgList
.text:00007FF7D8CF1087 mov     r9, [rsp+38h+Locale] ; Locale
.text:00007FF7D8CF108C mov     r8, [rsp+38h+Format] ; Format
.text:00007FF7D8CF1091 mov     rdx, [rsp+38h+Stream] ; Stream
.text:00007FF7D8CF1096 mov     rcx, [rax] ; Options
.text:00007FF7D8CF1099 call    cs:__stdio_common_vfprintf
.text:00007FF7D8CF109F add     rsp, 38h
.text:00007FF7D8CF10A3 retn

5) 00000460.00007FF7D8CF1060: sub_7FF7D8CF1060 (Synchronized with RTP)
```

f7 로 서브함수 내로 들어왔다.

레지스터에 있는 값들을 메모리로 옮김.

00000014D94FFB70	0000000000000000	
00000014D94FFB78	00007FF7D8CF11D2	sub_7FF7D8CF1190+42
00000014D94FFB80	00007FFF1C627C98	ucrtbase.dll:00007FFF1C627C98
00000014D94FFB88	00007FF7D8CF2238	.rdata:aInput
00000014D94FFB90	0000000000000000	

이후 서브 함수 호출했다가.... 다시 arglist, locale, format, stream 등의 인자를 레지스터로 옮기고 studio\_common\_vfprintf 호출

```
int vfprintf(FILE *stream, const char *format, va_list arg_ptr);
```

포맷된 출력을 스트림에 기록한다.

성공하면 stream에 기록된 바이트 수 리턴

printf 함수였군아

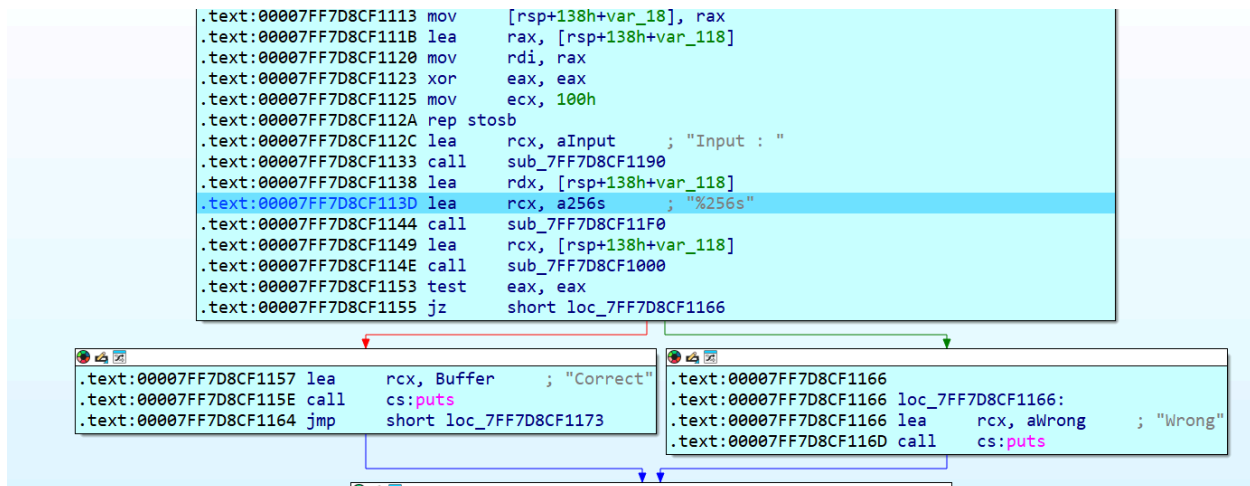
근데 위에도 iob\_func 있었는데

암튼 리턴해서, 다시 메인으로 왔다.

첫 번째 sub\_함수에서는 플래그에 대한 힌트는 얻을 수 없었던 걸로....

다음으로 RDX에 var\_118 변수가 있는 주소 기록

RDX 00000014D94FFBE0	Stack[00004D44]:00000014D94FFBE0
----------------------	----------------------------------



이후 rcx에 a256를 넣는다. 이걸 뭘 위한 거지? 아마 아래 서브함수 위한 것 같긴 한데.

```

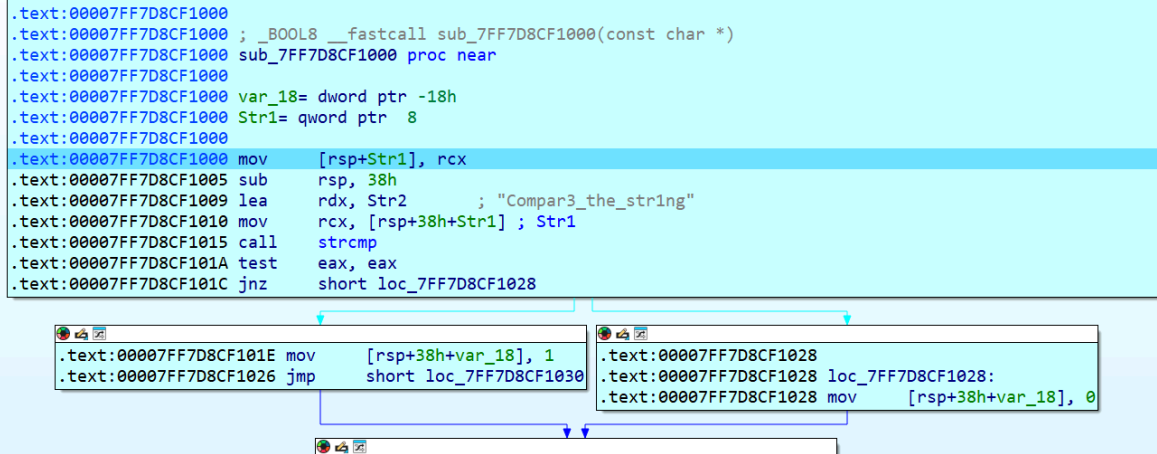
1  __int64 sub_7FF7D8CF11F0(const char *a1, ...)
2  {
3      FILE *v1; // rax
4      va_list va; // [rsp+48h] [rbp+10h] BYREF
5
6      va_start(va, a1);
7      v1 = _acrt_iob_func(0);
8      return (unsigned int)sub_7FF7D8CF10B0(v1, a1, 0, (__int64 *)va);
9  }

```

궁금해서 변환해봤는데 이것도 중요한 함수는 아닌 것 같다. 패스.

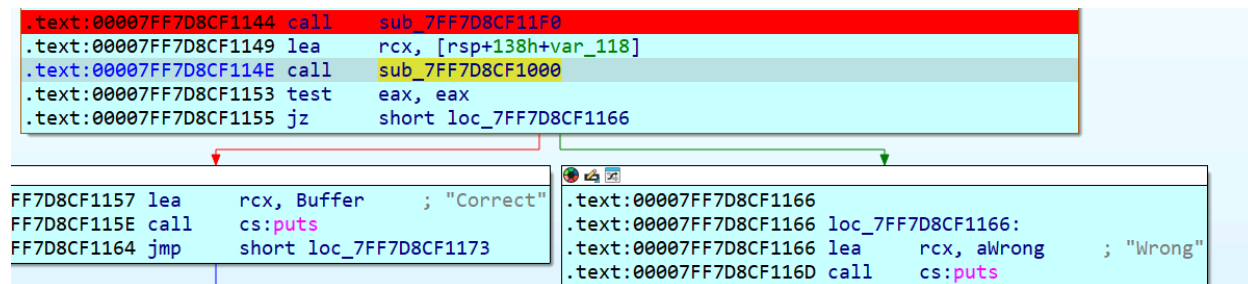
	Output
1	BOOL8 __fastcall sub_7FF7D8CF1000(const char *a1)
2	{
3	return strcmp(a1, "Compar3_the_str1ng") == 0;
4	}

디버깅 모드가 버벅여서 최종 분기문 바로 위의 sub 함수를 가봤음



안으로 들어오면 모양은 이렇다

우선 메모리에 rcx 값 넣기



RCX 0000009D5334F980 ↩ Stack[000059C8]:0000009D5334F980  
 RDX 00007FF7D8CF2220 ↩ "Compar3\_the\_str1ng"

이후 rdx(컴패어...)와 rcx 의 값을 비교한 뒤 이 둘이 같지 않으면 loc\_7ff7d8cf어쩌고로 점프한다

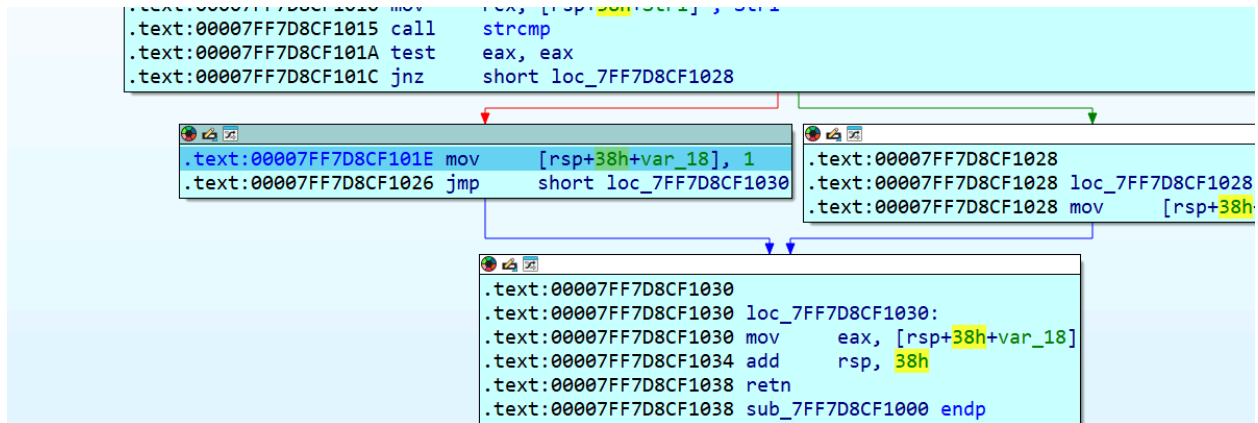
점프한다는 함수 한번 봄

아니 안 들어가지네

일단 rdx 레지스터 값을 rcx와 같게 만들어서 점프를 안 하게 해 봤다.



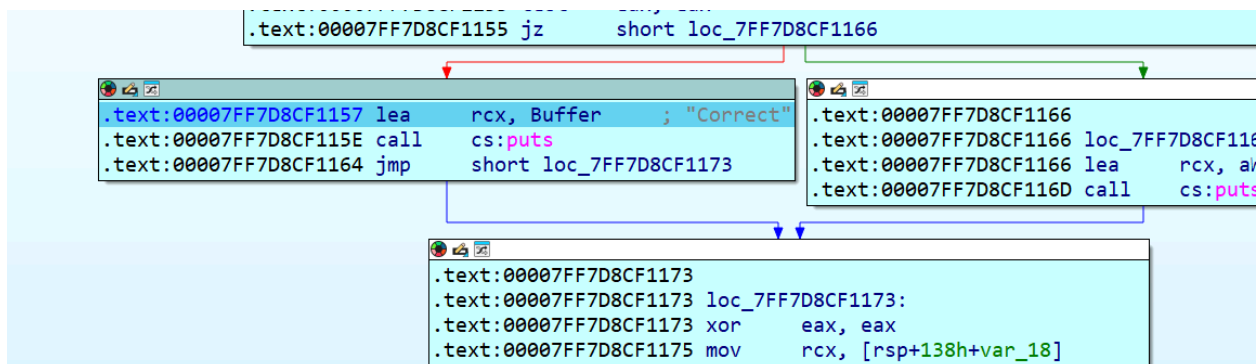
RCX 00007FF7D8CF2220 ↙ "Compar3\_the\_str1ng"  
 RDX 00007FF7D8CF2220 ↙ "Compar3\_the\_str1ng"



그러자 점프 안 하고 이쪽으로 옴.

이후 메인으로 리턴.

eax, eax 테스트해서 안에 값이 없으면 점프하는데, 아까 값을 조작해서 eax에 1이 들어있다. 따라서 점프하지 않을 것.



Correct 쪽으로 점프했다!

```

.text:00007FF7D8CF1173
.text:00007FF7D8CF1173 loc_7FF7D8CF1173:
.text:00007FF7D8CF1173 xor     eax, eax
.text:00007FF7D8CF1175 mov     rcx, [rsp+138h+var_18]
.text:00007FF7D8CF117D xor     rcx, rsp          ; StackCookie
.text:00007FF7D8CF1180 call    __security_check_cookie
.text:00007FF7D8CF1185 add     rsp, 130h
.text:00007FF7D8CF118C pop     rdi
.text:00007FF7D8CF118D retn
.text:00007FF7D8CF118D ; } // starts at 7FF7D8CF1100
.text:00007FF7D8CF118D main endp
.text:00007FF7D8CF118D

```

이후로는 함수 종료.

즉 입력값이 compare the string 과 동일하면 correct , 아니면 wrong 출력하는 단순한 함수다.


7 1 0 1 1



## Beginner rev-basic-0 문제를 해결했습니다.

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

관찰아요

 풀이 작성하기

성공!