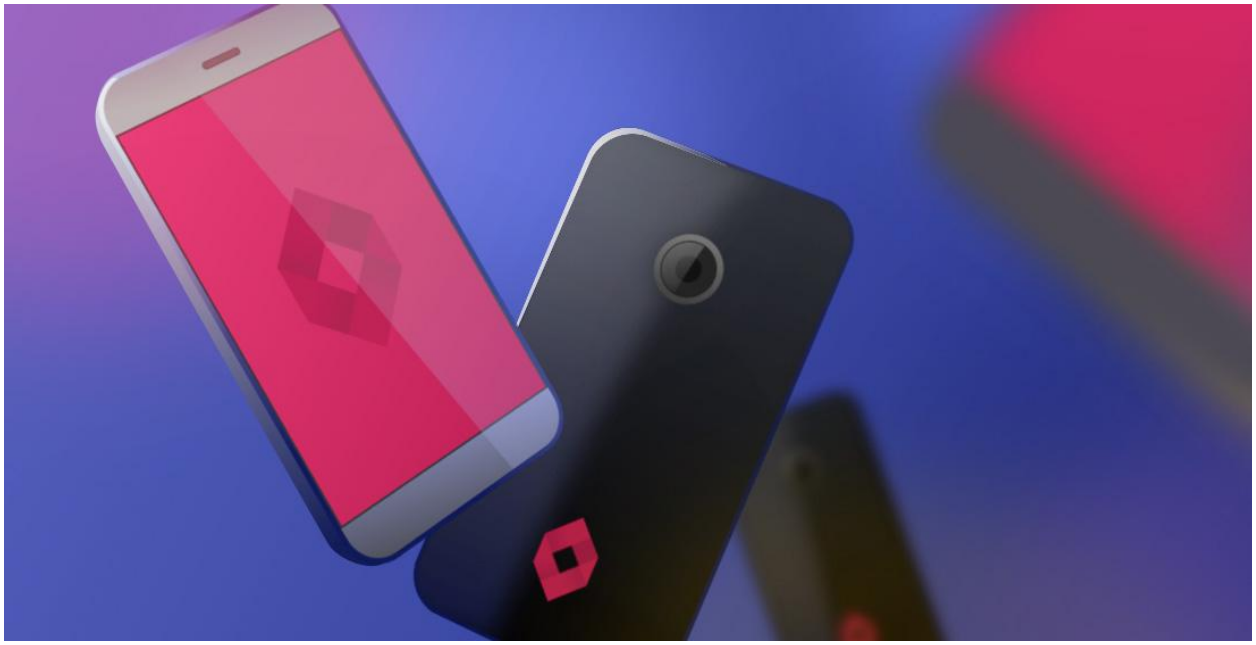
CrackMe Basic Writeup

Basic

December 01, 2021



Writeup

This reverse engineering challenge involved decoding a password from memory. However, the number of math operations made it a little more difficult than normal and was a good refresher on various assembly instructions and how they were used. The program requests a password of 8 characters and then does various checks to see if it is correct. There are a number of ways to solve this through patching, but I decided to attempt to solve it by generating the true password.

First Decoding

```
mov     eax, ecx
mov     esi, 0F0h ; 'ð'
mov     ebx, 255
cdq
idiv    esi
lea     eax, [ecx+edx]
mov     esi, edx
cdq
idiv    ebx
cmp     edx, 92h ; ''
jnz     short loc_40270E
```

First Decoding

This was the first function that appeared to do checks on the characters of the password. No other function calls were made here so it was pretty simple to follow. However, as stated previously, some of these instructions I had not messed with in a while so I had to refer to instructions ***cdq*** and ***idiv.*** This function just checks the first character of the password and fails if it's immediately wrong, if the character is correct it proceeds to go to the rest of the body of the program. Division as shown above is done twice by F0 and FF. What makes this harder to figure out is the password is not in readable acii characters where the first character should correspond to a 92h when decoded. The resulting answer from tinkering with this function block was **I.**

CDQ Instruction

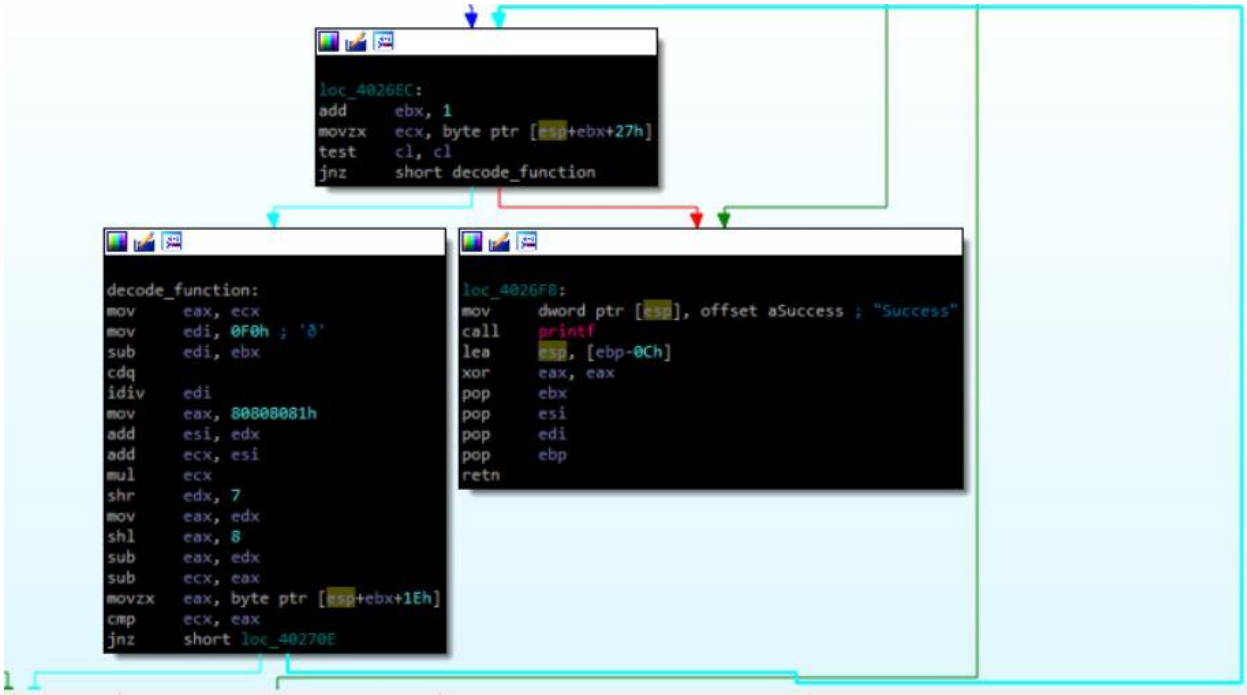
https://www.felixcloutier.com/x86/cwd:cdq:cqo

idiv instruction

http://www.c-jump.com/CIS77/MLabs/M11arithmetic/M11_0120_idiv_instruction.htm

---

Remaining Function Blocks



Function Blocks

The above showed the rest of the program and it was fairly simple to see there was another decode function that took the remaining characters in the scanned password and attempted to

decode and compare them with the values at **[esp+ebx+1Eh]** on the 2nd to last line in the renamed function **decode_function.** With this in mind I focused my remaining efforts on that function block.

Decode Function

```
decode_function:
mov     eax, ecx
mov     edi, 0F0h ; 'ð'
sub     edi, ebx
cdq
idiv    edi
mov     eax, 80808081h
add     esi, edx
add     ecx, esi
mul     ecx
shr     edx, 7
mov     eax, edx
shl     eax, 8
sub     eax, edx
sub     ecx, eax
movzx   eax, byte ptr [esp+ebx+1Eh]
cmp     ecx, eax
jnz     short loc_40270E
```

Decode Function

The last function block ended up doing the same thing the first decode function did, but with some additional math operations. There is probably more efficient ways to determine the correct answer but I pretty much sprayed values in each time and adjusted for the difference if they were wrong on the comparison with **[esp+ebx+1Eh]**. Doing this multiple times the resulting password ends up being ***I am back*.**

Resource Links

- https://www.felixcloutier.com/x86/cwd:cdq:cqo
- http://www.c-jump.com/CIS77/MLabs/M11arithmetic/M11_0120_idiv_instruction.htm