ESET Crackme Reversing Challenge

Objective 1

- Determine how it works execution flow?

Know-How

- The program is waiting for a password of unknown length and it terminates immediately itself if it's wrong.

Knowledge Required

- How much characters length of the password?
- Is there any anti-reversing technique to avoid analysis? If any what is it?
- To accomplish this stage, all meaningful data which obtained from static-dynamic analysis.

The main function is sub_4013F0 and start with a anti-debugging API function(IsDebuggerPresent).[1] We can avoid this detection by toggling returned value in eax register. Then program jumps to loc_401408.

```
push    ebp
mov     ebp, esp
sub     esp, 20h ; Integer Subtraction
call    ds:IsDebuggerPresent ; Indirect Call Ne
test    eax, eax ; Logical Compare
jz      short loc_401408 ; Jump if Zero (ZF=1)
```

In this location there are some operations on entered password. These are get string length, decode xor encoded stack strings and in an additional for anti-debugging PEB structure BeingDebugged flag and GetTickCount functionality.

```
loc_401408:
        mov     [ebp+NumberOfCharsWritten], 0
        push    3
        push    25h
        push    1Fh
        push    offset Buffer
        call    xor_decode_console_strings ; Call
        push    0       ; lpReserved
        lea     eax, [ebp+NumberOfCharsWritten] ;
        push    eax     ; lpNumberOfCharsWritten
        push    offset Buffer
        call    strlen  ; Call Procedure
        add     esp, 4  ; Add
        push    eax     ; nNumberOfCharsToWrite
        push    offset Buffer ; lpBuffer
        mov     ecx, hConsoleOutput
        push    ecx     ; hConsoleOutput
        call    ds:WriteConsoleA ; Indirect Call
```

```
call    ds:GetTickCount ; Indirect Call Nea
mov     [ebp+var_14], eax
mov     edx, 1
imul    edx, 7  ; Signed Multiply
movsx   eax, [ebp+edx+Buffer] ; Move with S
mov     ecx, 1
imul    ecx, 6  ; Signed Multiply
movsx   edx, [ebp+ecx+Buffer] ; Move with S
add     eax, edx ; Add
cmp     eax, 0CDh ; Compare Two Operands
```

When I was performing static analysis, I couldn't find interesting strings. But while program was executed shown a console string to enter a password. This basic indicator, the PE file has dynamically resolving strings. To confirm this theory we need to analyze sub_4013A0 function(labeled xor_decode_console_strings).

4 parameters pass to this function. These are string buffer, XORing initial key, key increment value and buffer length. Performs respectively following operations:

1. Read the char from buffer
2. Initialize xor key value

1

3. Perform xoring between char and xor key
4. Increment counter and add 3 to old xor key
5. Write decrypted string to same location for using later.

Now, we must determine right password length. To do this, we should focused 40147E address. At 40147E there are some arithmetic operations performing on entered password.
Like nested if statements, the program controls different character parts of password. For example firstly, 7nd and 6th indexes, then 8th and 5th so on. Each control statement perform similar operations.

1. Get characters hex value and sum it.
2. Then check it with relevant hard-coded value.

```
mov     ecx, 1
imul    ecx, 7  ; Signed Multiply
movsx   edx, [ebp+ecx+Buffer] ; Move with Sign
mov     eax, 1
imul    eax, 6  ; Signed Multiply
movsx   ecx, [ebp+eax+Buffer] ; Move with Sign
add     edx, ecx ; Add
mov     eax, 1
imul    eax, 3  ; Signed Multiply
movsx   ecx, [ebp+eax+Buffer] ; Move with Sign
add     edx, ecx ; Add
cmp     edx, 13Ah ; Compare Two Operands
jnz     loc_401612 ; Jump if Not Zero (ZF=0)
```

A part of password checking routine.(Character indexes 7, 6, 3)

This part like a keygen. You have to fake up characters which verify control flow. For first control, program checks 7. and 6. indexes(Notice I entered 8 character length password. It's AAAAAAAA) A's hex value is 0x41. 7. and 6. characters are A. But it's sum are not match with 0xCD. In this case 7. and 6. characters hex value sum must 0xCD. You can choose any char from ascii table which supply this condition. And repeat same operations for another characters.

Now I know right password length and temporary password. For me, I decided "aa0mSedidS" password.

After password checking completed there is a PEB.BeingDebugged flag check. You can avoid it with the similar method I mentioned above.

```
loc_401616:
        movzx   edx, [ebp+var_1]
        test    edx, edx ; Logica
        jz      loc_401711 ; Jump
```

Is password validate?

```
        mov     eax, large fs:30h
        movzx   eax, byte ptr [eax+2]
        test    eax, eax ; Logical Co
        jnz     short loc_401632 ; Ju
```

PEB.BeingDebugged

If you evasion from it, you encounter GetTickCount API function. First GetTickCount call at 401475 after entered password and second call is after PEB.BeingDebugged flag check. Returned value from both calls are important for Timing Check and stored at 18FF2C and 18FF30, respectively.

```
loc_40163A:
        call    ds:GetTickCount ; I
        mov     [ebp+var_10], eax
        mov     eax, [ebp+var_10]
        sub     eax, [ebp+var_14] ;
        cmp     eax, 64h ; Compare
        jbe     short loc_401656 ;
```

Now, move to 2nd console write function at 4016F9. Finally, the program a little help to obtain right password.

```
Please enter valid password : aa0mSedidS
!Good work. Little help:
char[8] = 85
char[0] + char[2] = 128
char[4] - char[7] = -50
char[6] + char[9] = 219
```

At first sight, it looks like char buffer and its members ASCII decimal values. To obtain right password, you must referenced which entered temp password and password checking routine. At this point, We perform basic arithmetic.

Index	0	1	2	3	4	5	6	7	8	9
Char	a	a	0	m	S	e	d	I	d	S
Decimal	97	97	48	109	83	101	100	73	100	83

char9 + char4 = A6h = 166

char6 – char7 = 3
char6 + char7 = 205(dec)
char0 + char1 = C2h
char0 + char1 + char2 + 6D + 33 + 74h + 68 + 65 + 55 + 73 = 39b

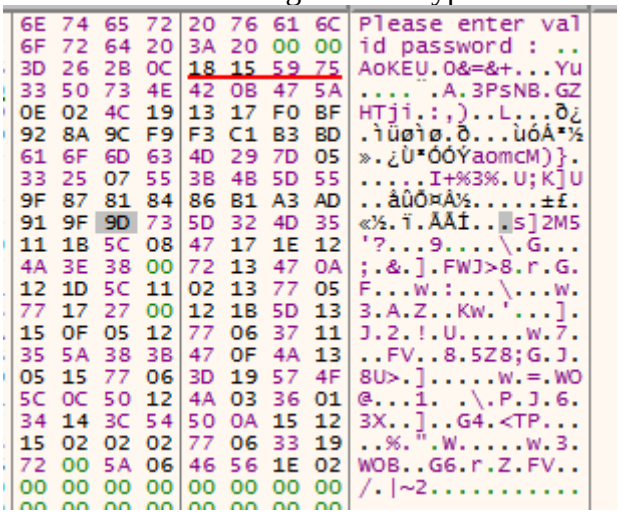
char0 = 50h = P
char1 = 72h = r
char2 = 30h = 0
char3 = 6Dh = m
char4 = 33h = 3
char5 = 74h = t
char6 = 68h = h
char7 = 65h = e
char8 = 55h = U
char9 = 73h = s

Now, We have obtained password. It’s Pr0m3theUs.

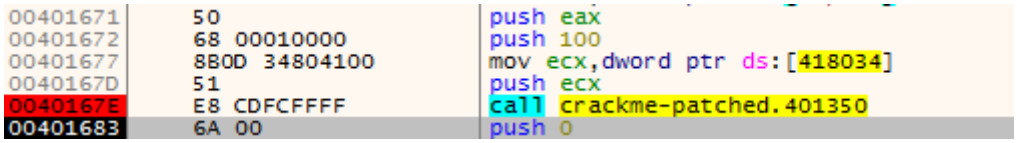
```
Please enter valid password : Pr0m3theUs
Congratulations! You guessed the right password, but the message you see is wrong.
Try to look for some unreferenced data, that can be decrypted the same way as this text.
```

What…What are you meaning?

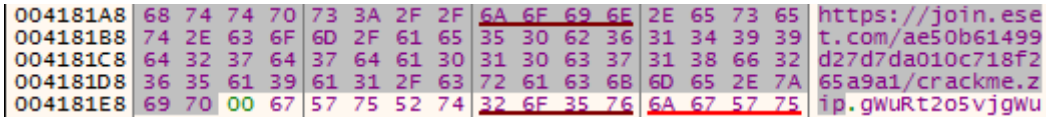
OK. We already determined earlier console strings and decryption routine. Others may be near its.



If I pass different buffer address to decryption routine, it can easily decrypt and I can obtain other data.



Encrypted buffer's address passed to ecx register. I modified ecx register to 4181A8(other unreferenced string buffer).



I have obtained new URL https://join.eset.com/ae50b61499d27d7da010c718f265a9a1/crackme.zip