Reversing Task

Purpose

The purpose of this task is to figure out the password of the “crackme” application using various tool.

Procedure

In order to figure out the correct password of “crackme” application I am using “IDA” to disassemble the program.

My approach is starting to examine the code and focusing on the relevant code until I find the password – see figure #1 below.

Figure #1 – First step

In the first step I started the program and saw the message “Please enter the password:”.

In order to estimate my entry point, I looked for the message from the program inside the assembly code and found:

```
.text:00401045          push    offset Format   ; "Please enter the password: "
```

For the entry point of the research - see figure #2.

```
.text:00401034 sub_401034      proc near               ; CODE XREF: start-7B↓p
.text:00401034
.text:00401034 Buffer          = byte ptr -6Ch
.text:00401034 var_68          = byte ptr -68h
.text:00401034 var_4           = dword ptr -4
.text:00401034
.text:00401034                 push    ebp
.text:00401035                 mov     ebp, esp
.text:00401037                 sub     esp, 6Ch
.text:0040103A                 mov     eax, ___security_cookie
.text:0040103F                 xor     eax, ebp
.text:00401041                 mov     [ebp+var_4], eax
.text:00401044                 push    ebx             ; ArgList
.text:00401045                 push    offset Format   ; "Please enter the password: "
.text:0040104A                 call    sub_401006
.text:0040104F                 xor     ebx, ebx
.text:00401051                 push    ebx             ; Ix
.text:00401052                 call    ds:__acrt_iob_func
.text:00401058                 push    eax             ; Stream
.text:00401059                 lea     eax, [ebp+Buffer]
.text:0040105C                 push    64h ; 'd'       ; MaxCount
.text:0040105E                 push    eax             ; Buffer
.text:0040105F                 call    ds:fgets
.text:00401065                 lea     ecx, [ebp+Buffer]
.text:00401068                 add     esp, 14h
.text:0040106B                 lea     edx, [ecx+1]
.text:0040106E
```

Figure #2 – Entry point

The program asks for input, and I notice the use of **fgets.**

```
.text:0040105F          call    ds:fgets
```

fgets stored the user input into the buffer.

The register **ecx** containing the location of the first character.

The code continued and counted how many characters are in the buffer, including the "new line" character at the end – see figure #3.

```
.text:0040106E loc_40106E:                      ; CODE XREF: sub_401034+3F↓j
.text:0040106E          mov     al, [ecx]
.text:00401070          inc     ecx
.text:00401071          test    al, al
.text:00401073          jnz     short loc_40106E
```

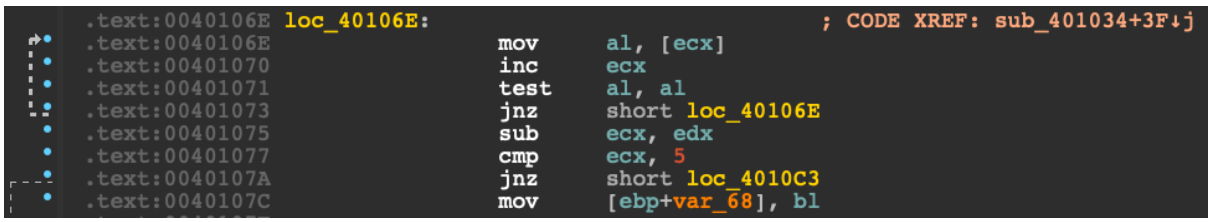

Figure #3 - Counting Characters

The code subtracted the address of the last input character with the address of the first input character in order to calculate the buffer length.

```
.text:00401075          sub     ecx, edx
```

If the buffer length is different than 5 (including the "new line" character) the code jumps to label **loc_4010C3**, that prints out "Ohhhh noo... try again" and terminates the program. – see figure #4.

```
.text:00401077          cmp     ecx, 5
.text:0040107A          jnz     short loc_4010C3
```

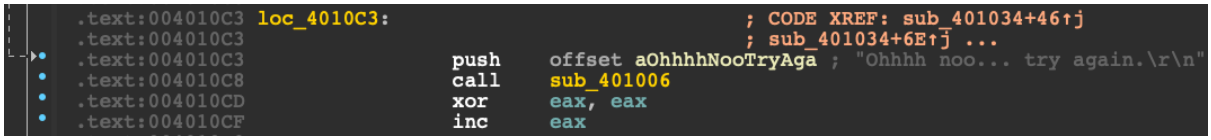

Figure #4 – Incorrect password

Meaning, the length of the password should be exactly 5 characters (including the "new line" character).

When the buffer is exactly 5 characters (including the new line char) I can advance to the final step.

The final step has a loop that runs 10 times. (**ebx** is 0 and it keeps running until **ebx** equals 0A). – see figure #5.

```
.text:0040107F loc_40107F:                      ; CODE XREF: sub_401034+5E↓j
.text:0040107F          mov     eax, ebx
.text:00401081          and     eax, 3
.text:00401084          mov     al, [ebp+eax+Buffer]
.text:00401088          xor     byte_403018[ebx], al
.text:0040108E          inc     ebx
.text:0040108F          cmp     ebx, 0Ah
.text:00401092          jb      short loc_40107F
```

Figure #5 –data manipulation loop

Due to the password length, the loop only used the first four bytes of the buffer **(eax, 3)**.

```
.text:0040107F          mov    eax, ebx
.text:00401081          and    eax, 3
.text:00401084          mov    al, [ebp+eax+Buffer]
```

And perform **xor** with each character of the buffer [0, 1, 2, 3, 0, 1, 2, 3] with every character of the string "G00D job!". "G00D job!" string – see figure #6.

```
.text:00401088          xor    byte_403018[ebx], al
```

Figure #6 – "GOOD job" message

After that loop I sew a comparison between the **ecx** and **edx** to check if the password is correct.

```
.text:00401094          mov    ecx, offset byte_403018
.text:00401099          mov    edx, offset unk_403024
.text:0040109E          mov    eax, [ecx]
.text:004010A0          cmp    eax, [edx]
```

I saw that **ecx** contains our xor values (between the password and "G00D job!" string) from the loop above. Furthermore, the **edx** contains the character: **5q}eb+"Cc.** – see figure #7.

```
:00403024 unk_403024      db    5                 ; DATA XREF: sub_401034+65↑o
:00403025                 db  71h ; q
:00403026                 db  7Dh ; }
:00403027                 db  65h ; e
:00403028                 db  62h ; b
:00403029                 db  2Bh ; +
:0040302A                 db  22h ; "
:0040302B                 db  43h ; C
:0040302C                 db  63h ; c
```

Figure #7 – edx data

Meaning, the inserted password **xor** “G00D job!” string should be equal to **5q}eb+”Cc** (to pass the **cmp** operator).

The next formula was build:

- **password ^ G00D job! = 5q}eb+”Cc**

To find the password I **xor** “G00D job!” with **5q}eb+”Cc** (xor is invertible operation) and get BAM!BAM!B.

It can be concluded from the previous steps that the password contains only 4 characters (excluding the "new line").

The password is: **BAM!**

The password **BAM!** was insert to the application and the message “"YOU DID IT !~!" appeared.