Reverse Engineering Exam

To find the serial number of 04_crackme.exe

Student:

PATIL Kunal

MSc Artificial Intelligence System

Teacher:

KHALDI Adel

Table of Contents

Introduction

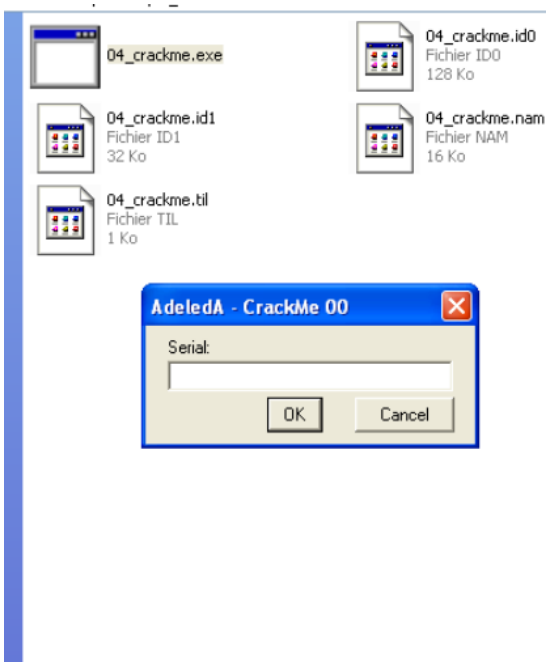
Reverse engineering is the process by which an artificial object is deconstructed to reveal its designs, architecture, code, or to extract knowledge from the object. It is similar to scientific research, the only difference being that scientific research is conducted into a natural phenomenon.

(Source: Wikipedia)

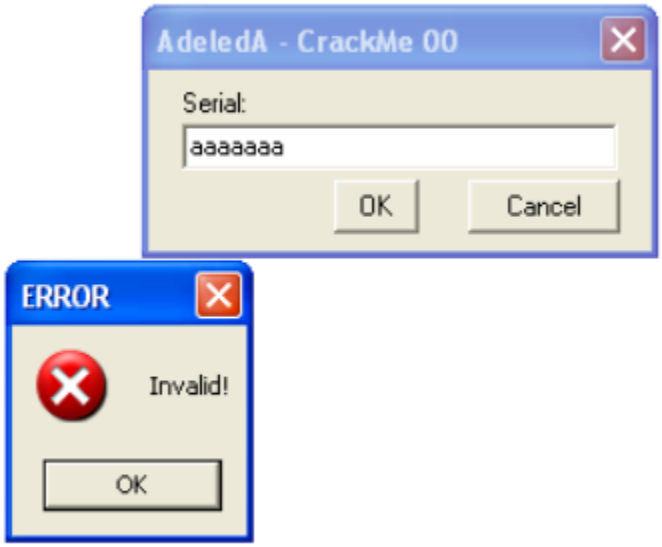
The approach allows us to break-in into application by calculating the serial number from its functional opcode. There are two methods in reverse engineering, viz. disassembler and debugger. Disassembler is used to translate machine language into assembly language. Whereas debugger to test and debug programs. The main use of a debugger is to run the target program under controlled conditions.

Application

1. Open application 04_crackme.exe



We don't know the serial number of the application.



Disassembling the code

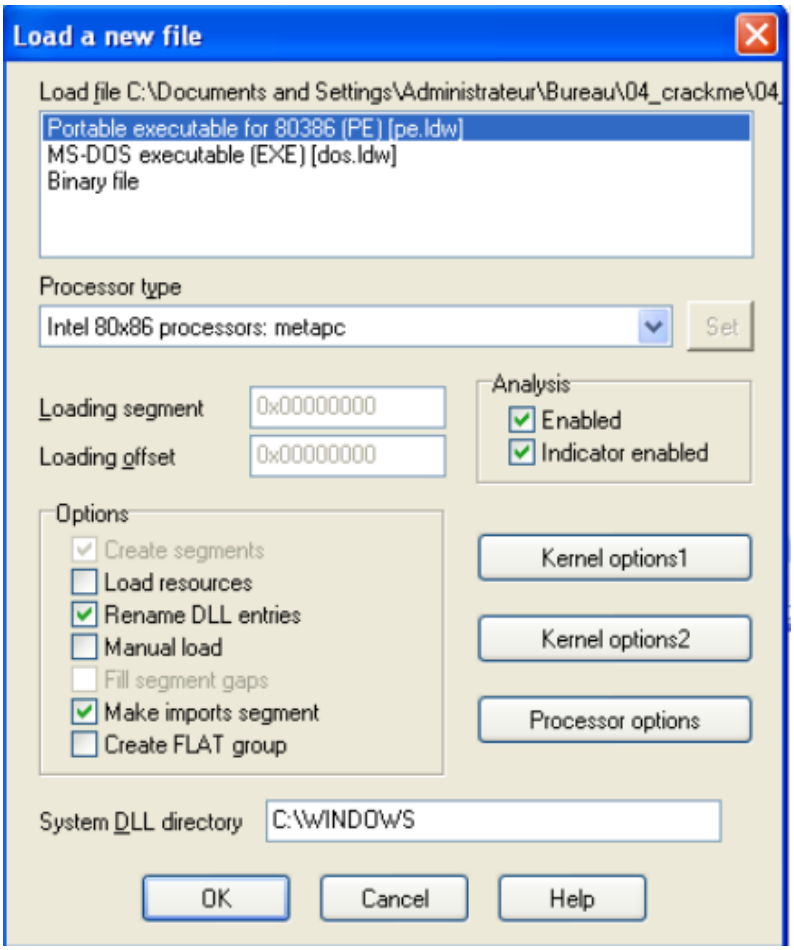
Tool used: IDA Pro

Disassembler and Debugger is an interactive, programmable, extensible, multi-processor disassembler hosted on Windows, Linux, or Mac OS X.

(Source: https://www.hex-rays.com/)

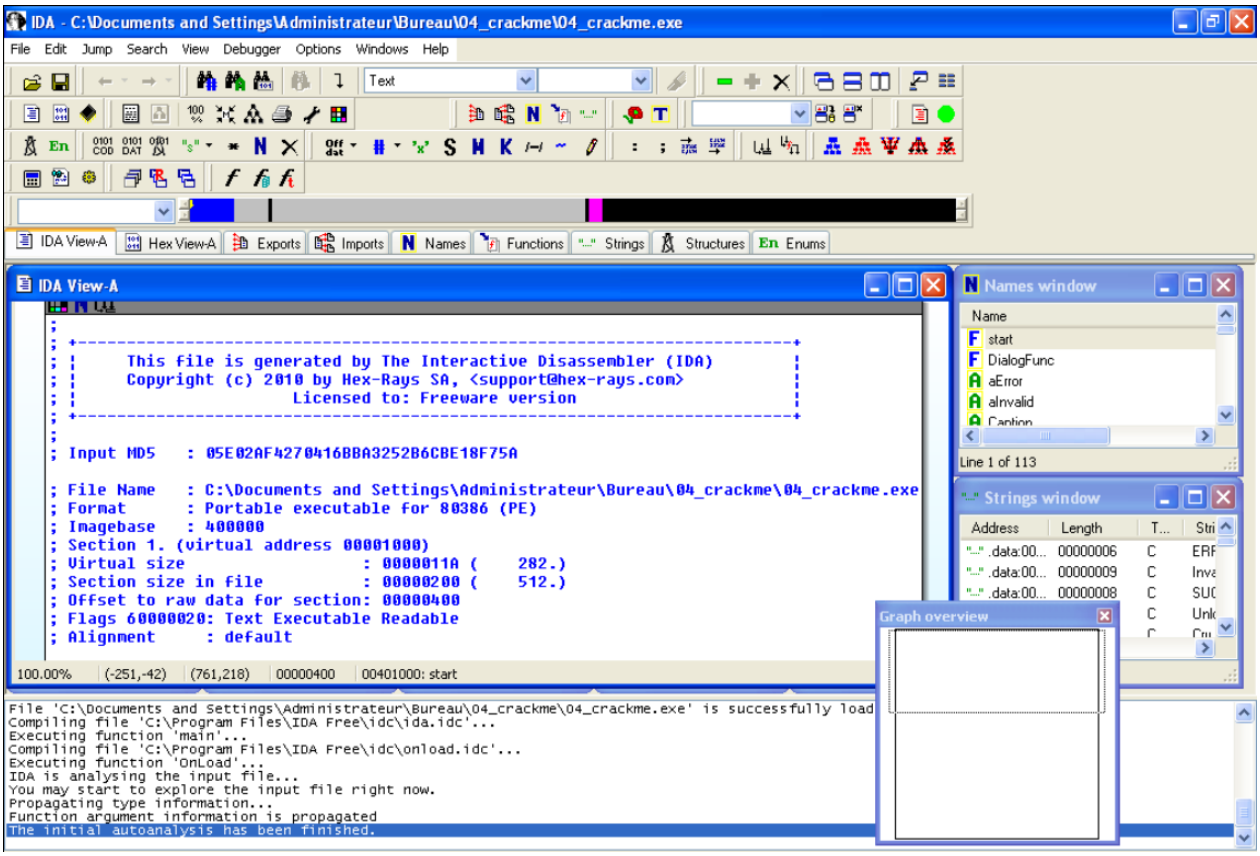
Loading file into IDA

1. Open/Drag and Drop exe file into IDA.



Check the description once before press OK

2. Showing the details



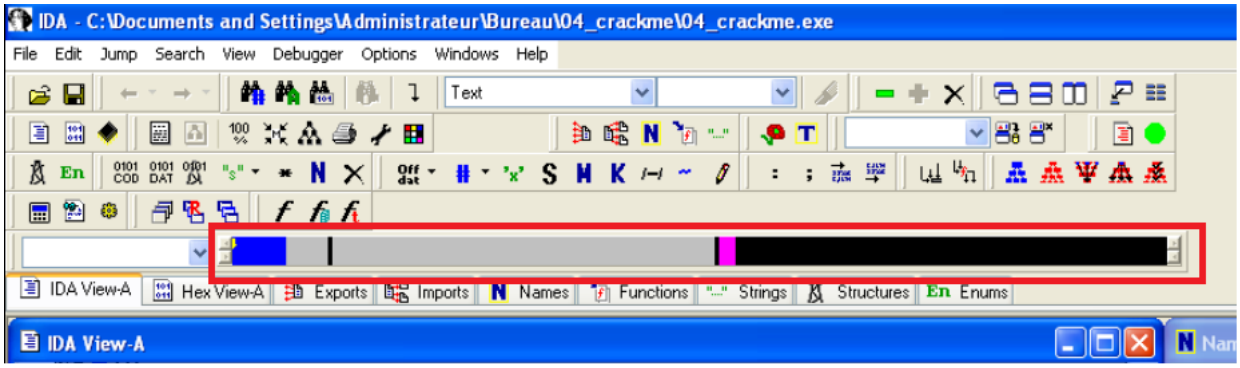
We can see multiple windows opened displaying various data extracted from the application.

Names window: Shows all the available functions in code

Strings window: Shows strings and addresses and length from the code

Graph window: shows the connected graphical view of the code functions

IDA view A: Displays the details related to the code



Displays the sections in the executable. Which section is data, text and rsrc.

Analyzing the process of execution

1. Expand IDA view A and double click on it.
2. It will redirect it to the starting point of the function

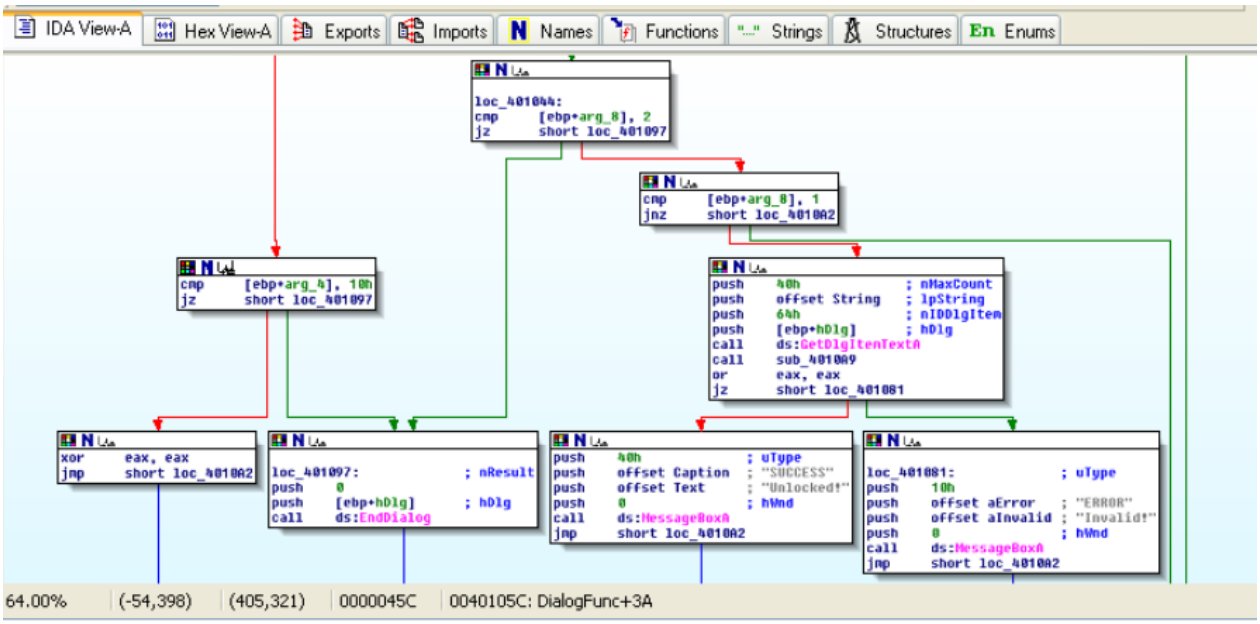
```
public start
start proc near
push    0                   ; lpModuleName
call    ds:GetModuleHandleA
push    0                   ; dwInitParam
push    offset DialogFunc ; lpDialogFunc
push    0                   ; hWndParent
push    25h                 ; lpTemplateName
push    eax                 ; hInstance
call    ds:DialogBoxParamA ; Create a modal dialog box from a
                            ; dialog box template resource
push    0                   ; uExitCode
call    ds:ExitProcess
start endp
```

As we can see, we need DialogBoxParamA function for our analysis which is responsible for the identification of serial number.

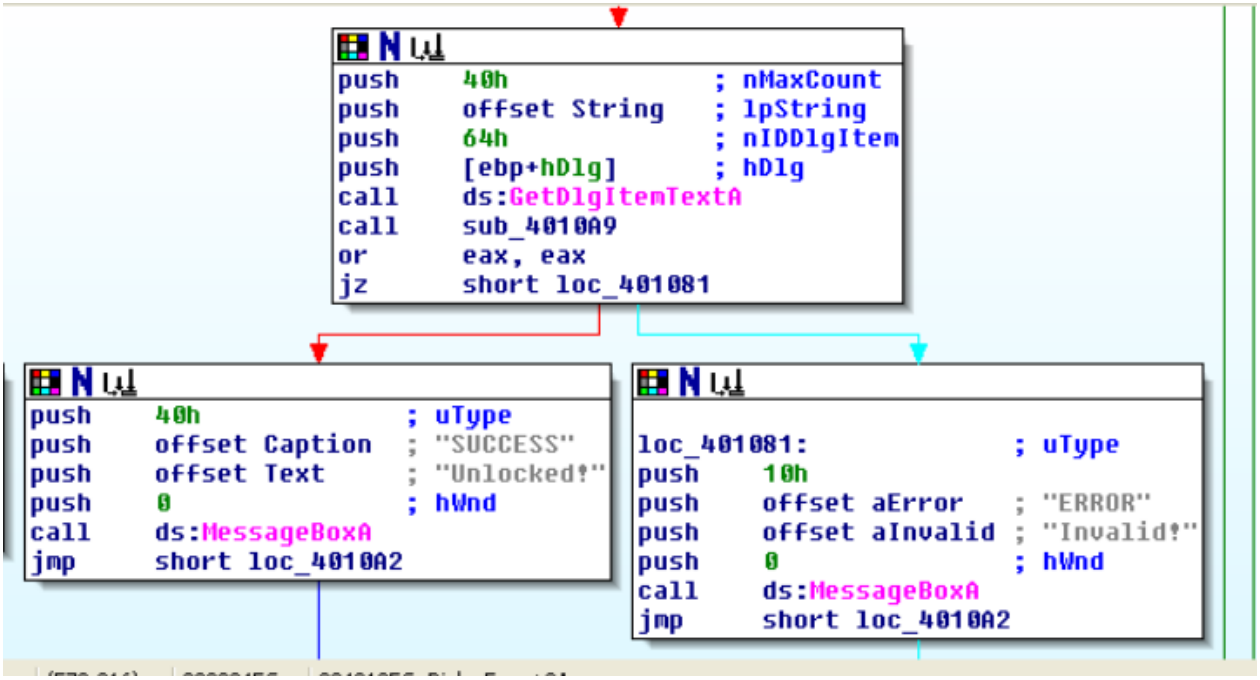
3. Double click on the function will redirect to the .idata location of the function

```
IDA View-A | Hex View-A | Exports | Imports | Names | Functions | Strings | Structures | Enums
.idata:0040306C
.idata:00403070
.idata:004030A8 ;
.idata:004030A8 ; Imports from USER32.DLL
.idata:004030A8 ;
.idata:004030A8 ; int __stdcall DialogBoxParamA(HINSTANCE hInstance,LPCSTR lpTemplateName,HWND hWndParent,DLGPROC lp
.idata:004030A8                 extrn DialogBoxParamA:dword ; DATA XREF: start+14↑r
.idata:004030A8                                             ; Create a modal dialog box from a
.idata:004030A8                                             ; dialog box template resource
.idata:004030AC ; UINT __stdcall GetDlgItemTextA(HWND hDlg,int nIDDlgItem,LPSTR lpString,int nMaxCount)
.idata:004030AC                 extrn GetDlgItemTextA:dword ; DATA XREF: DialogFunc+3A↑r
.idata:004030B0 ; int __stdcall MessageBoxA(HWND hWnd,LPCSTR lpText,LPCSTR lpCaption,UINT uType)
.idata:004030B0                 extrn MessageBoxA:dword ; DATA XREF: DialogFunc+57↑r
.idata:004030B0                                         ; DialogFunc+6D↑r
.idata:004030B4 ; BOOL __stdcall EndDialog(HWND hDlg,int nResult)
.idata:004030B4                 extrn EndDialog:dword   ; DATA XREF: DialogFunc+7A↑r
.idata:004030B8
.idata:004030B8
.idata:004030B8
.idata:004030B8                 end start
```

4. Press space bar to change the view to graphical



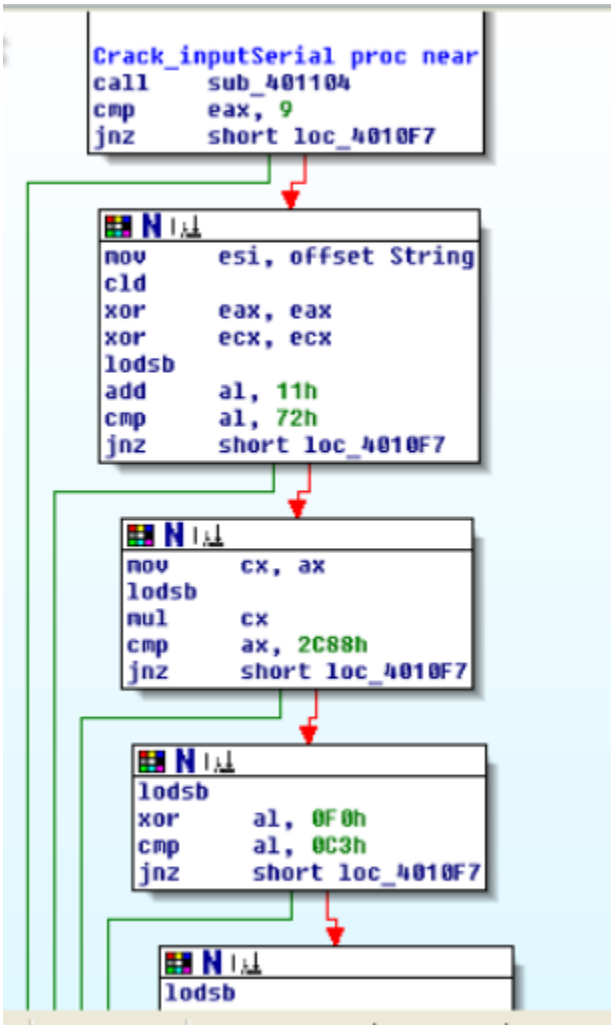
5. Focus on the function GetDigItemtextA which is responsible for getting input serial number from user.



After execution of this function, further it calling the function sub_4010A9

For convenience, rename the function crack_inputSerial

6. Once clicked on the function, it will redirect to the extension where it will call whole process to check whether the input password is correct or not.



Analyzing each block one by one

(PS. The calculation is performed by each block using manual on paper as it is done on windows xp and x32dbg was not available due to some technical issues with the VM)

```
Crack_inputSerial proc near
call    sub_401104
cmp     eax, 9
jnz     short loc_4010F7
```

This block is calling the function sub_401104, which is responsible to calculate the length of the input Serial number

```
sub_401104 proc near
cld
lea     esi, String
xor     eax, eax

loc_40110D:
lodsb
cmp     al, 0
jnz     short loc_40110D

lea     eax, [esi-4027A5h]
dec     eax
retn
sub_401104 endp
```

Further it is comparing whether the length of eax is 9 or not

If the length is equal to 9, the flag ZF returned from cmp will be 0 if not ZF=1

If the non-zero value returned, the function sub_4010F7 will be executed which is responsible for fail (password incorrect)

Rename the function to “failure” for convenience and will be used for future reference.

```
mov     esi, offset String
cld
xor     eax, eax
xor     ecx, ecx
lodsb
add     al, 11h
cmp     al, 72h
jnz     short failure
```

The String refers to the input provided by the user.

Cld is for Auto incrementing (which starts from lowest to highest)

Xor eax, eax ➔ Clear the EAX register

Xor ecx, ecx ➔ Clear the ECX register

Lodsb ➔ It loads into AL

Add AL, 11h

cmp AL, 72h

Here, Addition of AL + 11h = 72h which further will be compared with AL

So to get value of AL, AL = 72h-11h = 61h

Checking the ASCII value for 61h it gives “a”

So, first character from the serial number is “a”

```
mov     cx, ax
lodsb
mul     cx
cmp     ax, 2C88h
jnz     short failure
```

In next block, ax will have the value 72 (add 61h, 11h = 72h)

The value of ax is moved to cx

Lodsb ➔ loads into ax

To get a value for ax, which further will be compared with 2C88h

Cx will be multiplied with that value.

So, cx = ax * 2c88h

So, ax = 2c88h/cx = 2c88h/72h = 64h

Checking the ASCII value for the 64h, gives “d”

So, second character from the serial number is “d”

```
lodsb
xor     al, 0F0h
cmp     al, 0C3h
jnz     short failure
```

In this block, again lodsb will loads value to AL

Calculating XOR between values 0F0h and 0C3h = 33h

Checking ASCII value for 33h, gives “3”

So, third character from the serial number is “3”

```
lodsb
mov     ecx, 1B0h
shr     ecx, 2
cmp     al, cl
jnz     short failure
```

In this block, lodsb will loads value to AL

1B0h is moved to ECX and then ECX is shift right by 2 bits

Right shifting 1B0h gives 6ch

Calculating ASCII value for 6ch, gives “l”

So, fourth character from the serial number is “l”

```
lodsb
cmp     al, 2Dh
jnz     short failure
```

In this block, lodsb loads value to AL.

Here there is comparison between AL and 2Dh

Converting 2Dh to ASCII, gives “-”

So, fifth character from the serial number is “-”

```
lodsd
xor     eax, 66666666h
cmp     eax, 702550Ah
jz      short loc_4010FE
```

Here, lodsd loads the value EAX but in reverse.

Calculating XOR between values 66666666h and 702550Ah = 6164336ch

Checking ASCII value for 6164336ch, gives “ad3l”

And reversing this string gives, “l3da”

The further 4 characters of the serial number are “l3da”

As the input serial number is length 9, the correct serial number is.

“ad3l-l3da”

Trying the serial number