

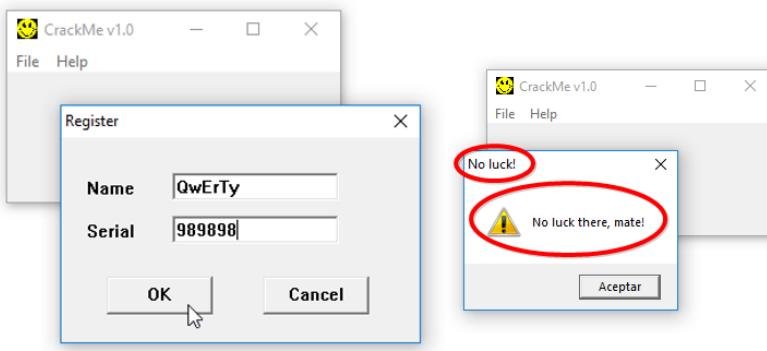
CRACKME	<i>CrackMe v1.0 by Cruehead</i>
<i>Misión</i>	<i>Registrarnos con un Name y Serial</i>
<i>Compilado</i>	<i>MASM/TASM</i>
<i>Protección</i>	<i>Ninguna</i>
<i>Herramientas</i>	<i>RDG Packer Detector V0.7.6 IDA Pro 6.8.150.423 (32bit)</i>
<i>Sistema Operativo</i>	<i>Windows 10 Home (64bits)</i>
<i>Cracker</i>	<i>QwErTy</i>
<i>Dedicado a</i>	<i>Ricardo Narvaja - <u>CracksLatinoS</u></i>
<i>Descargar Crackme</i>	https://mega.nz/#!fIUQWKgJlyfnALEkQkP61gZX5RfI0IJ86FIWDbVBmXYUVr-u7MS8

Probando diferentes herramientas para ver su comportamiento con Windows 10 Home. en mi anterior tutorial solucioné este entrañable Crackme con la Tool "x32dbg", ahora voy a hacerlo con "IDA". Reconozco que ésta vez llevo ventaja porque de antemano ya sé el resultado, pero es bueno practicar. Quiero dedicar este pequeño trabajo a **RICARDO NARVAJA** y a los demás integrantes de CracksLatinoS, del que también tengo el orgullo de formar parte con mis pequeñas aportaciones, y como siempre, espero y deseo que el lector pase un rato agradable y entretenido con su lectura de igual modo que yo lo he pasado escribiéndolo.

ESTUDIANDO LA VÍCTIMA



La ejecutamos, nos pide un "Name" y un "Serial", rellenamos datos, le damos a "OK" para validar ycomo ya no es de extrañar....., nos salta el mensaje de "**Chico Malo**", a decir verdad, ya esperaba que saliera el maldito "MessageBoxA"



Aceptamos que hemos fallado.

Pasamos el detector de ejecutables "RDG", y nos revela que estamos ante un compilado en "MASM/TASM" de 32 bits, y no está empacado.



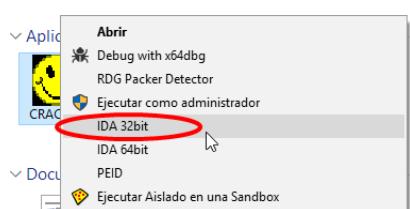
AL ATAQUE

Antes de continuar les voy a revelar un pequeño secreto...

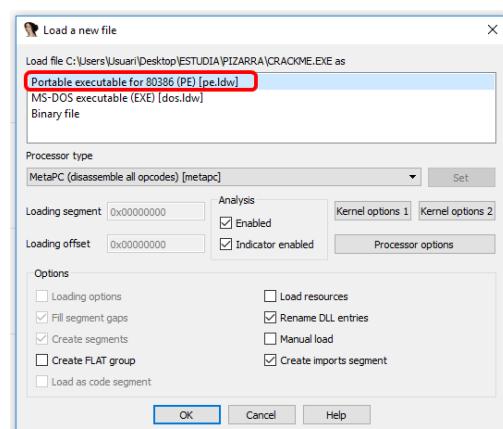
(En su momento cuando instalé IDA en mí nuevo y flamante hp portátil con el cual trabajo ahora, directamente en el "Regedit" me las ingené para crear unos útiles accesos directos en el menú contextual de Windows, y como pueden apreciar en la imagen inferior, puedo abrir IDA cuando me apetezca siempre que despliegue el menú "Abrir" sobre las víctimas con un simple click derecho de ratón.

P.D. Si alguien está interesado en saber cómo lo hice, solo tiene que pedírmelo....pero creo que la mayoría de ustedes ya sabrán como hacerlo).

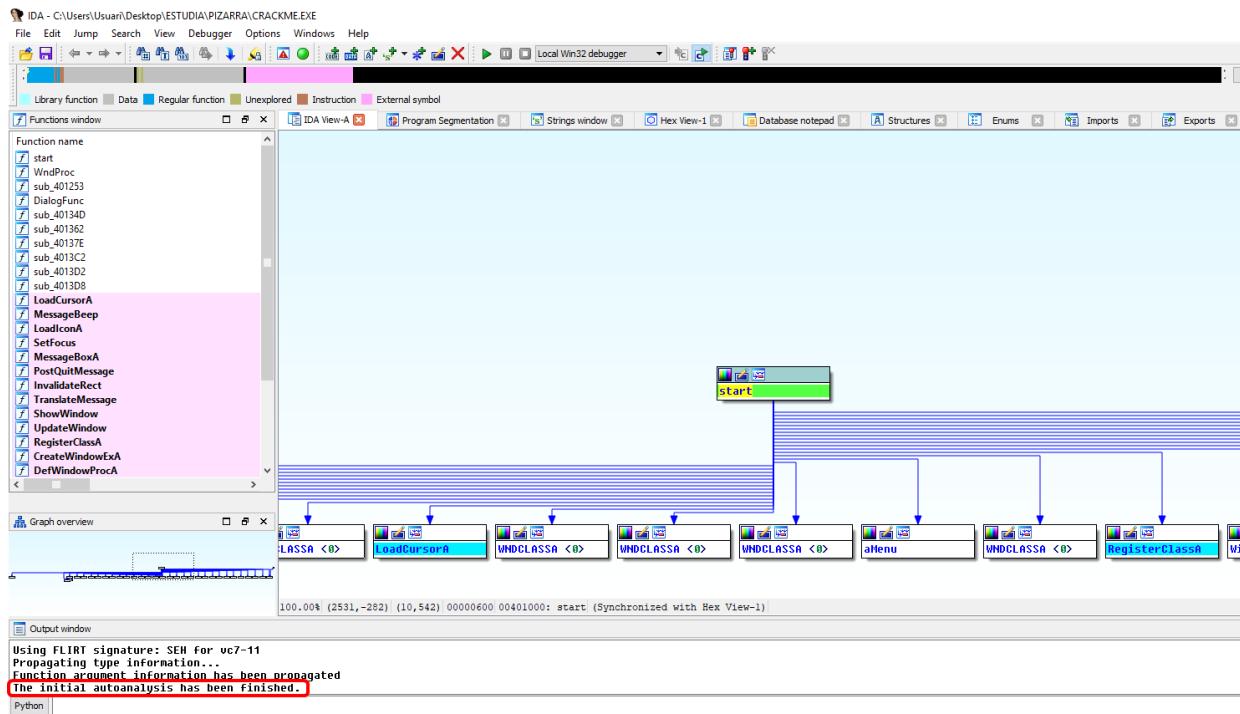
A lo que vamos.....cargamos el "Crackme", con "IDA 32bit"



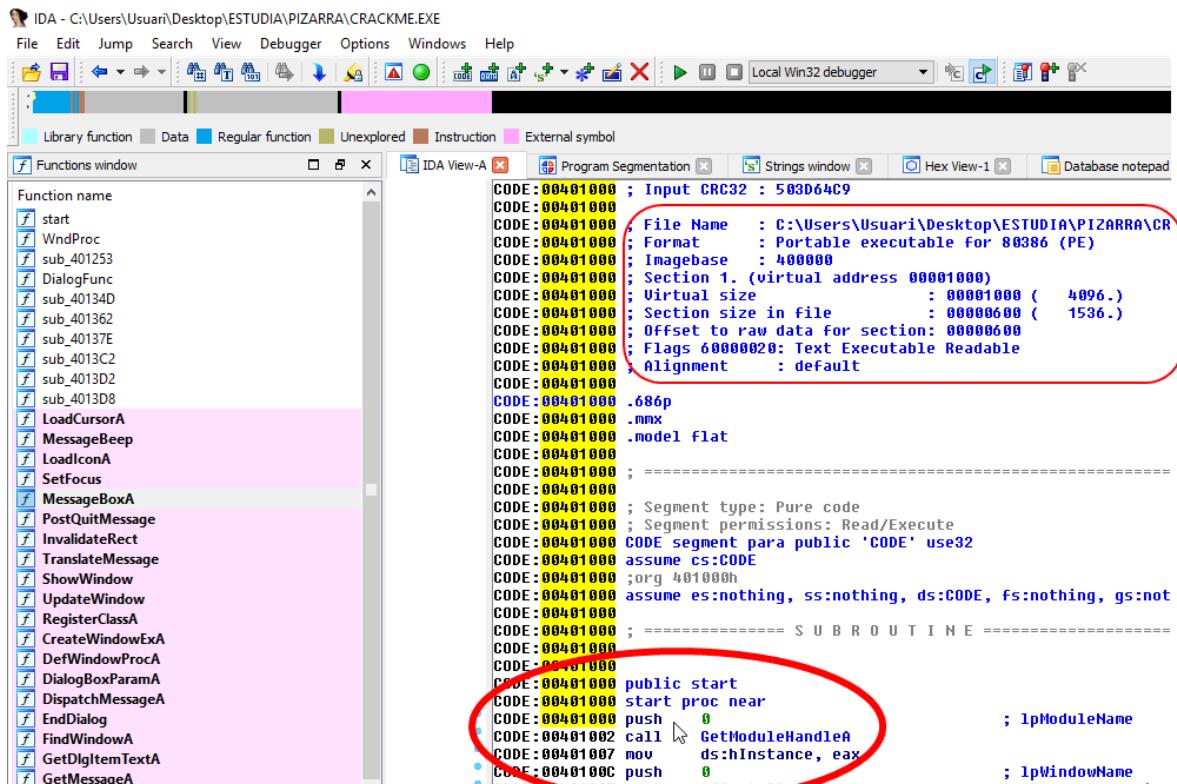
Nos detecta el portable/ejecutable, dejamos todas las tildes marcadas por defecto



Le damos a "OK", y apareceremos en la primera pantalla donde nos muestra el árbol de funciones, de paso también observamos que el autoanálisis se ha realizado correctamente



Damos dos veces a la barra espaciadora de nuestro teclado para pasar al LOADER (modo desensamblador), y nos encontramos parados en el OEP CODE:00401000" Section 1



Miramos las "String window", y ahí están los mensajes "Chico Bueno" y "Chico Malo"

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.EXE

Function name	Address	Length	Type	String
f_start	DATA:004020D6	00000011	C	Try to crack me!
f_WndProc	DATA:004020E7	0000000D	C	CrackMe v1.0
f_sub_401253	DATA:004020F4	0000001C	C	No need to disasm the code!
f_DialogFunc	DATA:00402110	00000005	C	MENU
f_sub_40134D	DATA:00402115	0000000A	C	DLG_REGIS
f_sub_401362	DATA:0040211F	0000000A	C	DLG_ABOUT
f_sub_40137E	DATA:00402129	0000000B	C	Good work!
f_sub_4013C2	DATA:00402134	00000002C	C	Great work, mate!\rNow try the next CrackMe!
f_sub_4013D2	DATA:00402160	00000009	C	No luck!
f_sub_4013D8	DATA:00402169	00000015	C	No luck there, mate!
f_LoadCursorA				
f_MessageBeep				
f_LoadIconA				

Nos posicionamos sobre "Chico Malo"

Address	Length	Type	String
DATA:004020D6	00000011	C	Try to crack me!
DATA:004020E7	0000000D	C	CrackMe v1.0
DATA:004020F4	0000001C	C	No need to disasm the code!
DATA:00402110	00000005	C	MENU
DATA:00402115	0000000A	C	DLG_REGIS
DATA:0040211F	0000000A	C	DLG_ABOUT
DATA:00402129	0000000B	C	Good work!
DATA:00402134	00000002C	C	Great work, mate!\rNow try the next CrackMe!
DATA:00402160	00000009	C	No luck!
DATA:00402169	00000015	C	No luck there, mate!

Dos clicks Izq de ratón y aparecemos en "DATA:00402169"

Function name	Assembly
f_start	DATA:004020F4 ; CHAR ClassName[]
f_WndProc	ClassName db 'No need to disasm the code!',0 ; DATA XREF: start+ETo
f_sub_401253	; start+81To ...
f_DialogFunc	DATA:004020F4 ; CHAR aMenu db 'MENU',0 ; DATA XREF: start+77To
f_sub_40134D	DATA:00402115 aMenu db 'MENU',0 ; DATA XREF: WndProc+EBTo
f_sub_401362	DATA:00402115 aDlg_Regis[],0 ; DATA XREF: WndProc+CFTo
f_sub_40137E	DATA:0040211F ; CHAR TemplateName[]
f_sub_4013C2	TemplateName db 'DLG_ABOUT',0 ; DATA XREF: WndProc+CFTo
f_sub_4013D2	DATA:00402129 ; CHAR Caption[]
f_sub_4013D8	DATA:00402129 Caption db 'Good work!',0 ; DATA XREF: sub_40134D+2To
f_LoadCursorA	DATA:00402134 ; CHAR Text[]
f_MessageBeep	DATA:00402134 Text db 'Great work, mate!',0Dh,'Now try the next CrackMe!',0 ; DATA XREF: sub_40134D+7To
f_LoadIconA	DATA:00402160 ; CHAR aNoLuck[]
f_SetFocus	DATA:00402160 aNoLuck db 'No luck!',0 ; DATA XREF: sub_401362+9To
f_MessageBoxA	DATA:00402160 ; sub_40137E+31To
f_PostQuitMessage	DATA:00402169 ; CHAR aNoLuckThereMat[]
f_InvalidateRect	DATA:00402169 aNoLuckThereMat db 'No luck there, mate!',0 ; DATA XREF: sub_401362+ETo
f_TranslateMessage	; sub_40137E+36To
f_ShowWindow	DATA:0040217E ; CHAR byte_40217E[16]
f_UpdateWindow	DATA:0040217E byte_40217E db 10h dup(0) ; DATA XREF: WndProc+10BTo
f_RegisterClassA	DATA:0040218E ; CHAR String[3698]
f_End	DATA:0040218E String db 72h dup(0), 0E00h dup(?) ; DATA XREF: WndProc+100To
	DATA:0040218E DATA ends

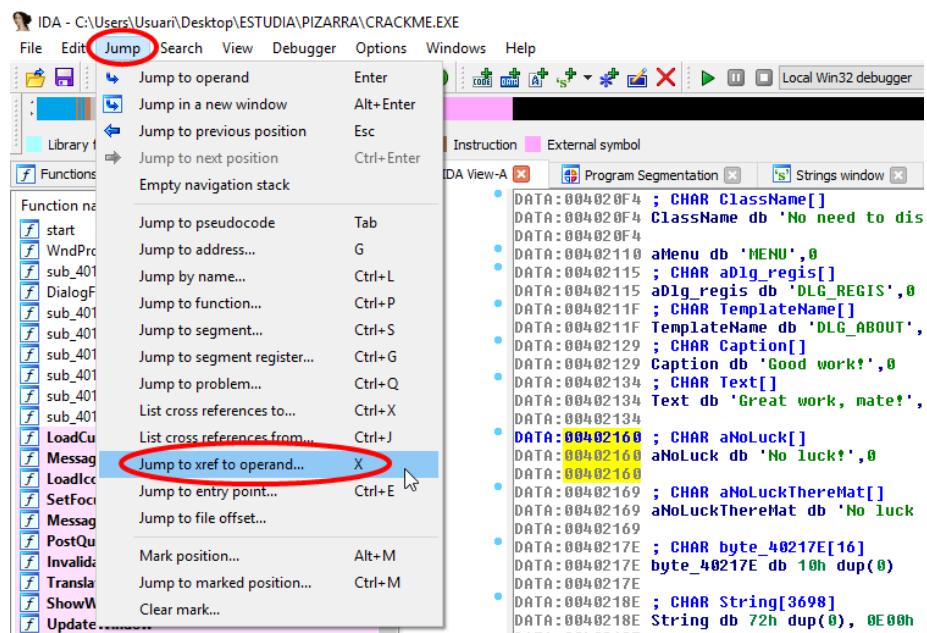
Nos posicionamos en la address "DATA:00402160" (donde se inicia el mensaje de Chico Malo)

```

DATA:00402115 aDlg_regs db 'DLG_REGS',0 ; DATA XREF: WndProc+EB↑o
DATA:0040211F ; CHAR TemplateName[]
DATA:0040211F TemplateName db 'DLG_ABOUT',0 ; DATA XREF: WndProc+CF↑o
DATA:00402129 ; CHAR Caption[]
DATA:00402129 Caption db 'Good work!',0 ; DATA XREF: sub_40134D+2↑o
DATA:00402134 ; CHAR Text[]
DATA:00402134 Text db 'Great work, mate!',0Dh,'Now try the next CrackMe!',0 ; DATA XREF: sub_40134D+7↑o
DATA:00402134
DATA:00402160 ; CHAR aNoLuck[]
DATA:00402160 aNoLuck db 'No luck!',0 ; DATA XREF: sub_401362+9↑o
DATA:00402160
DATA:00402169 ; CHAR aNoLuckThereMat[]
DATA:00402169 aNoLuckThereMat db 'No luck there, mate!',0 ; DATA XREF: sub_401362+E↑o
DATA:00402169
DATA:0040217E ; CHAR byte_40217E[16]
DATA:0040217E byte_40217E db 10h dup(0) ; DATA XREF: WndProc+10B↑o
DATA:0040217E
DATA:0040218E ; CHAR String[3698]
DATA:0040218E String db 72h dup(0), 0E00h dup(?) ; DATA XREF: WndProc+100↑o
DATA:0040218E
DATA:0040218E DATA ends

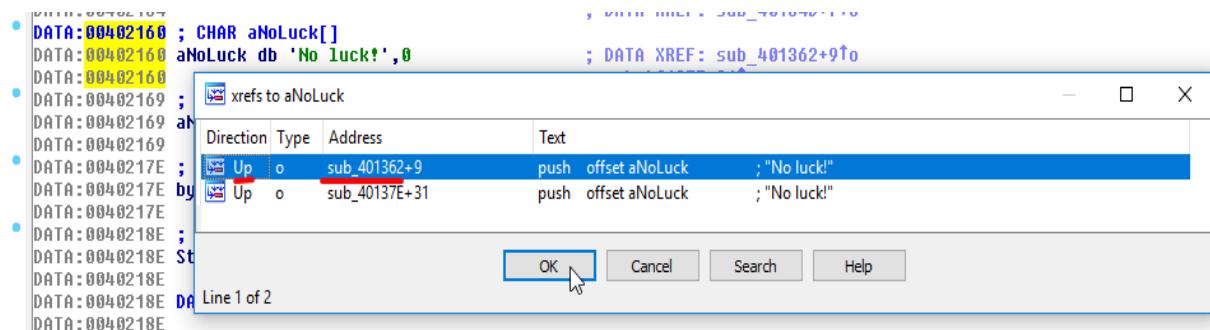
```

Miramos de donde es llamada



Y obtenemos dos direcciones que provienen de dos subrutinas. Ambas se encuentran por encima de donde estamos parados.

Address "sub_401362" y Address "sub_40137E"



Nos colocamos sobre la primera, le damos a "OK" y salimos aquí

IDA - C:\Users\Usuari\Desktop\ESTUDIA\PIZARRA\CRACKME.EXE

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window IDA View-A Program Segmentation Strings window Hex View-1 Database notepad Structures Enums

```

Function name
start
WndProc
sub_401253
DialogFunc
sub_40134D
sub_401362
sub_40137E
sub_4013C2
sub_4013D2
sub_4013D8
LoadCursorA
MessageBeep
LoadIconA
SetFocus
MessageBoxA
PostQuitMessage
InvalidateRect
TranslateMessage
ShowWindow
UpdateWindow
RegisterClassA
CreateWindowExA
DefWindowProcA
DialogBoxParamA
DispatchMessageA

CODE:0040134B
CODE:0040134D
CODE:0040134D ; ===== S U B R O U T I N E =====
CODE:0040134D
CODE:0040134D
CODE:0040134D
CODE:0040134D sub_40134D proc near
CODE:0040134D push 30h
CODE:0040134F push offset Caption
CODE:00401354 push offset Text
CODE:00401359 push dword ptr [ebp+8]
CODE:0040135C call MessageBoxA
; CODE XREF: WndProc:loc_40124Ctp
; uType
; "Good work!"
; "Great work, mate!\rNow try the next Cra"...
; hWnd

CODE:00401361 retn
CODE:00401361 sub_40134D endp
CODE:00401361
CODE:00401362
CODE:00401362 ; ===== S U B R O U T I N E =====
CODE:00401362
CODE:00401362
CODE:00401362 sub_401362 proc near
CODE:00401362 push 0
CODE:00401364 call MessageBeep
CODE:00401369 push 30h
CODE:0040136B push offset aNoLuck
CODE:00401370 push offset aNoLuckThereMat
CODE:00401375 push dword ptr [ebp+8]
CODE:00401378 call MessageBoxA
; CODE XREF: WndProc+110Tp
; uType
; "No luck!"
; "No luck there, mate!"
; hWnd

CODE:0040137D retn
CODE:0040137D sub_401362 endp
CODE:0040137D
CODE:0040137E

```

Ahora si nos posicionamos con el cursor sobre “`WndProc+110Tp`” se nos abre una ventana y obtenemos esta valiosa información:

CODE:00401362

===== S U B R O U T I N E =====

```

CODE:00401362 sub_401362 proc near
CODE:00401362 push 0
CODE:00401364 call MessageBeep
CODE:00401369 push 30h
CODE:0040136B push offset aNoLuck
CODE:00401370 push offset aNoLuckThereMat
CODE:00401375 push dword ptr [ebp+8]
CODE:00401378 call MessageBoxA
CODE:0040137D retn
CODE:0040137D sub_401362 endp
CODE:0040137D
CODE:0040137E

```

Zona Caliente

call sub_40137E
push eax
push offset byte_40217E
call sub_4013D8
add esp, 4
pop eax
cmp eax, ebx
jz short loc_40124C
call sub_401362
jmp short loc_4011E6

Para que nos entendamos..... vemos la comparación “`cmp`” de los registros “`eax, ebx`” seguida de un salto condicional “`jz`” que en función del resultado, nos mandará a “`40124C`” o a “`401362`” que es donde actualmente nos encontramos “**Chico Malo**”

Con que somos curiosos, nos posicionamos sobre “`WndProc:loc_40124Ctp`” donde se encuentra el mensaje de “**Chico Bueno**” y vemos la misma **ZONA CALIENTE**

IDA - C:\Users\Usuari\Desktop\ESTUDIA\PIZARRA\CRACKME.EXE

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window IDA View-A Program Segmentation Strings window Hex View-1 Database notepad Structures Enums

```

Function name
start
WndProc
sub_401253
DialogFunc
sub_40134D
sub_401362
sub_40137E
sub_4013C2
sub_4013D2
sub_4013D8
LoadCursorA
MessageBeep
LoadIconA
SetFocus
MessageBoxA
PostQuitMessage
InvalidateRect
TranslateMessage
ShowWindow
UpdateWindow
RegisterClassA
CreateWindowExA
DefWindowProcA
DialogBoxParamA
DispatchMessageA

CODE:0040134B
CODE:0040134D
CODE:0040134D ; ===== S U B R O U T I N E =====
CODE:0040134D
CODE:0040134D
CODE:0040134D
CODE:0040134D sub_40134D proc near
CODE:0040134D push 30h
CODE:0040134F push offset Caption
CODE:00401354 push offset Text
CODE:00401359 push dword ptr [ebp+8]
CODE:0040135C call MessageBoxA
; CODE XREF: WndProc:loc_40124Ctp
; uType
; "Good work!"
; "Great work, mate!\rNow try the next Cra"...
; hWnd

CODE:00401361 retn
CODE:00401361 sub_40134D endp
CODE:00401361
CODE:00401362
CODE:00401362 ; ===== S U B R O U T I N E =====
CODE:00401362
CODE:00401362
CODE:00401362 sub_401362 proc near
CODE:00401362 push 0
CODE:00401364 call MessageBeep
CODE:00401369 push 30h
CODE:0040136B push offset aNoLuck
CODE:00401370 push offset aNoLuckThereMat
CODE:00401375 push dword ptr [ebp+8]
CODE:00401378 call MessageBoxA
; CODE XREF: WndProc+110Tp
; uType
; "No luck!"
; "No luck there, mate!"
; hWnd

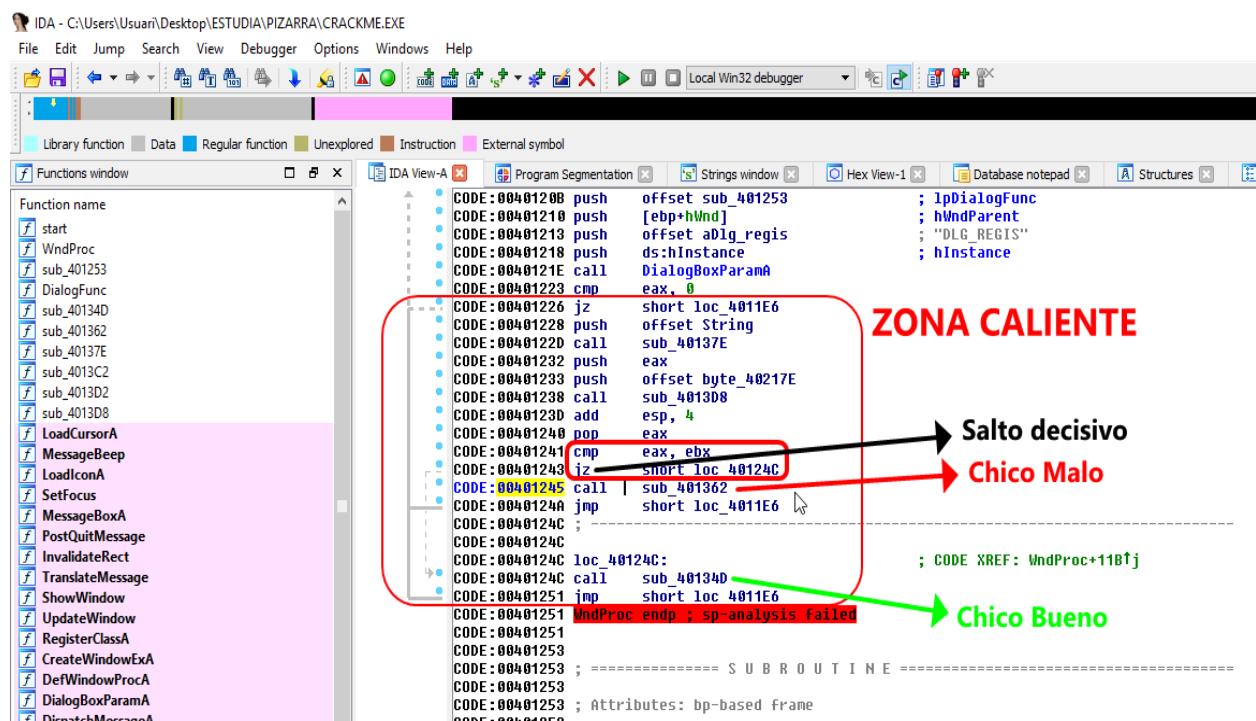
CODE:0040137D retn
CODE:0040137D sub_401362 endp
CODE:0040137D
CODE:0040137E
CODE:jmp short loc_40124C
; CODE XREF: WndProc:loc_40124Ctp
; uType
; "Good work!"
; "Great work, mate!\rNow try the next Cra"...
; hWnd

```

Pues vayamos a esa **ZONA CALIENTE**, para ello podemos hacerlo clicando con el ratón directamente sobre “`WndProc+11Dtp`” o sobre “`WndProc:loc_40124Ctp`”, o bien posicionándonos con el cursor sobre las address *CODE*: donde se inician cualquiera de esas dos subrutinas (address *CODE*:0040134D o address *CODE*:00401362”, le damos a la letra “x” de nuestro teclado y nos aparecerá de donde son llamadas.

Yo voy a elegir esta última opción, me posiciono con el cursor sobre "CODE:00401362" que es donde se inicia el mensaje de "Chico Malo", le damos a "x" y obtenemos el siguiente resultado

Le damos a "OK" para que nos lleve a esa ruta, y apareceremos aquí



Ahora si nos posicionamos con el cursor sobre "sub_4013D8" del "call" que se encuentra en la address "CODE:00401238", vemos cosas interesantes..je,je,je... que más adelante analizaremos con todo detalle.

d Instruction ■ External symbol

Program Segmentation Strings window Hex View-1 Database notepad Structures Enums

```

CODE:00401202 call DialogBoxParamA
CODE:00401207 jmp short loc_4011E6
CODE:00401209 ;
CODE:00401209 loc_401209: ; CODE XREF: WndProc+BA↑j
CODE:00401209 push 0 ; dwInitParam
CODE:0040120B push offset sub_401253 ; lpDialogFunc
CODE:00401210 push [ebp+hWnd] ; hWndParent
CODE:00401213 push offset aDlg ; arg_0= dword ptr 4
CODE:00401218 push ds:Instance; ===== SUBROUTINE =====
CODE:0040121E call DialogBoxPar
CODE:00401223 cmp eax, 0
CODE:00401226 jz short loc_4013D8 proc near ; CODE XREF: WndProc+110↑p
CODE:00401228 push offset Strin
CODE:0040122D call sub_40137E arg_0= dword ptr 4
CODE:00401232 push eax
CODE:00401233 push offset byte xor eax, eax
CODE:00401238 call sub_4013D8 xor edi, edi ; CODE XREF: sub_4013D8+1B↓j
CODE:0040123D add esp, 4 xor ebx, ebx
CODE:00401240 pop eax mov esi, [esp+arg_0]
CODE:00401241 cmp eax, ebx
CODE:00401243 jz short loc_4013E2: ; CODE XREF: sub_4013D8+1B↓j
CODE:00401245 call sub_401362 mov al, 0Ah
CODE:0040124A jmp short loc_4013F5 mov bl, [esi]
CODE:0040124C ; test bl, bl
CODE:0040124C jz short loc_4013F5
CODE:0040124C loc_40124C: sub bl, 30h
CODE:0040124C call sub_40134D imul edi, eax
CODE:00401251 jmp short loc_4013D8 add edi, ebx
CODE:00401251 WndProc endp ; sp-align esi
CODE:00401251 jmp short loc_4013E2
CODE:00401253 ;
CODE:00401253 loc_4013F5: ; CODE XREF: sub_4013D8+10↑j
CODE:00401253 Attributes: bp-base xor edi, 1234h
CODE:00401253 mov ebx, edi
CODE:00401253 ; INT_PTR __stdcall ret
CODE:00401253 sub_4013D8 endp
00000838 00401238: WndProc+110 (Syncr)

```

Hacemos lo mismo con la "call" de más arriba, address "CODE:0040122D" y también vemos otras instrucciones importantes para nuestro menester.

Unexplored ■ Instruction ■ External symbol

Program Segmentation Strings

```

; ===== SUBROUTINE =====
CODE:00401202 call DialogBoxParamA sub_40137E proc near ; CODE XREF: WndProc+105↑p
CODE:00401207 jmp short loc_40137E arg_0= dword ptr 4
CODE:00401209 ;
CODE:00401209 loc_401209: mov esi, [esp+arg_0]
CODE:00401209 push 0 push esi
CODE:0040120B push offset sub_40137E loc_401383: ; CODE XREF: sub_40137E+14↓j
CODE:00401210 push [ebp+hWnd] ; sub_40137E+1C↓j
CODE:00401213 push offset aDlg ; loc_401383:
CODE:00401218 push ds:Instance; test al, al
CODE:0040121E call DialogBoxPar jz short loc_40139C
CODE:00401223 cmp eax, 0 al, 41h
CODE:00401226 jz short loc_401394 inc ebx
CODE:00401228 push offset byte xor ebx, ebx
CODE:0040122D call sub_4013D8 inc esi
CODE:00401232 add esp, 4
CODE:00401240 pop eax loc_401394: ; CODE XREF: sub_40137E+11↑j
CODE:00401243 jz short loc_4013D2 call sub_401362
CODE:00401245 call sub_401362 inc esi
CODE:00401246 jmp short loc_401383 jmp short loc_401383
CODE:0040124C ; ;
CODE:0040124C loc_40124C: loc_40139C: ; CODE XREF: sub_40137E+9↑j
CODE:0040124C call sub_40134D pop esi sub_4013C2
CODE:00401251 jmp short loc_401394 call sub_4013C2
CODE:00401251 xor edi, 5678h
CODE:00401251 mov eax, edi
CODE:00401251 jmp short locret_4013C1 ; CODE XREF: sub_40137E+D↑j
CODE:00401253 ; Attributes: bp-base loc_4013AC: ; CODE XREF: sub_40137E+D↑j
CODE:00401253 locret_4013C1: ; uType
CODE:00401253 push 30h ; "No luck!"
CODE:00401253 push offset aNoLuck ; "No luck there, mate!"
CODE:00401253 push offset aNoLuckThereMat ; hWnd
CODE:00401253 call MessageBoxA ; hWnd
0000082D 0040122D: WndProc+105 (Syncr) ; CODE XREF: sub_40137E+2C↑j

```

Pues de momento vamos a poner puntos de parada "BP" en cada una de esas dos "call" sospechosas.

RACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

explored Instruction External symbol

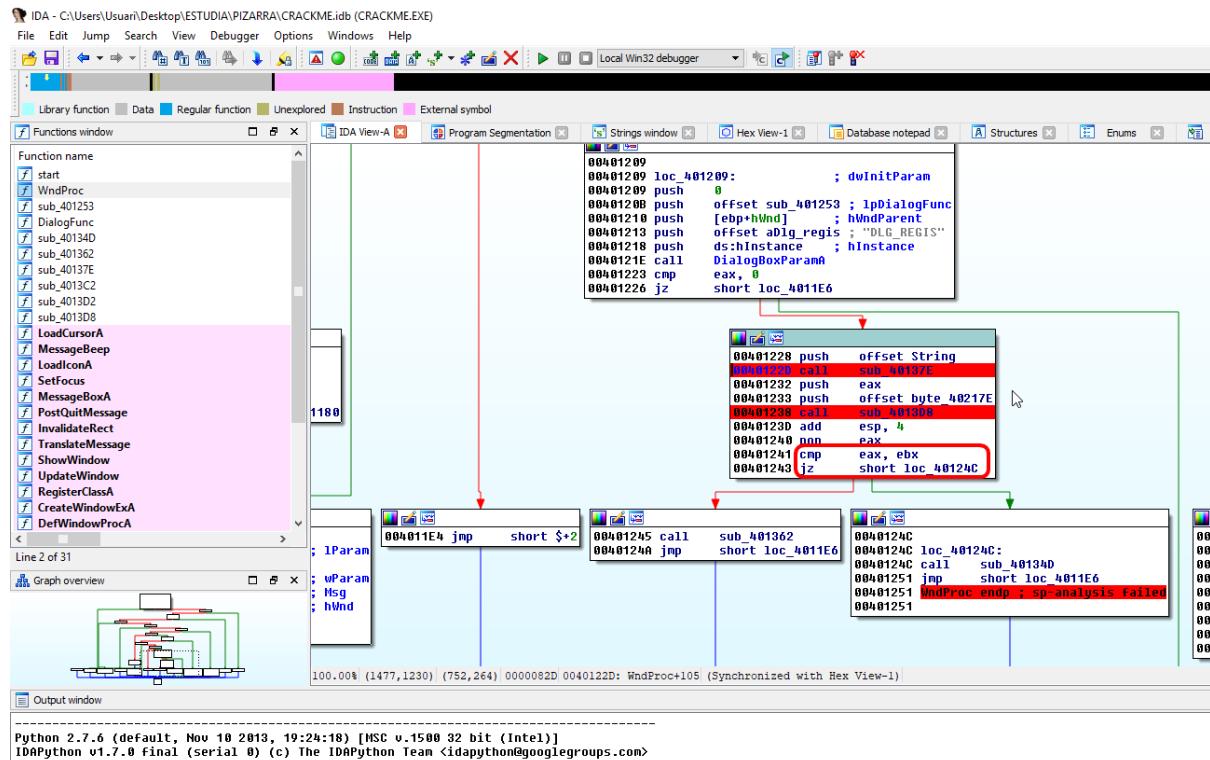
IDA View-A Program Segmentation Strings window Hex View-1 Database notepad Structures Enums

```

CODE: 00401202 call DialogBoxParamA
CODE: 00401207 jmp short loc_4011E6
CODE: 00401209 ;
CODE: 00401209 loc_401209: ; CODE XREF: WndProc+BA↑j
CODE: 00401209 push 0 ; dwInitParam
CODE: 00401208 push offset sub_401253 ; lpDialogFunc
CODE: 00401210 push [ebp+hWnd] ; hWndParent
CODE: 00401213 push offset aDlg_Regis ; "DLG_REGIS"
CODE: 00401218 push ds:getInstance ; hInstance
CODE: 0040121E call DialogBoxParamA
CODE: 00401223 cmp eax, 0
CODE: 00401226 jz short loc_4011E6
CODE: 00401228 push offset String
● CODE: 0040122D call sub_40137E
CODE: 00401232 push eax
CODE: 00401233 push offset byte_40217E
● CODE: 00401238 call sub_401308
CODE: 0040123D add esp, 4
CODE: 00401240 pop eax
CODE: 00401241 cmp eax, ebx
CODE: 00401243 jz short loc_40124C
CODE: 00401245 call sub_401362
CODE: 00401248 jmp short loc_4011E6
CODE: 0040124C ;
CODE: 0040124C loc_40124C: ; CODE XREF: WndProc+11B↑j
CODE: 0040124C call sub_401340
CODE: 00401251 jmp short loc_4011E6
CODE: 00401251 WndProc endp ; sp-analysis Failed
CODE: 00401253
CODE: 00401253 ; ===== S U B R O U T I N E =====
CODE: 00401253

```

Ahora le damos a la barra espaciada y apareceremos en el modo gráfico, donde vemos claramente lo que va a hacer el salto condicional "jz" que se encuentra en la address 00401243, en el que en función del resultado de la comparación "cmp eax,ebx"



nos mandará a la address "00401245" (Chico Malo)

```

00401228 push offset String
0040122D call sub_40137E
00401232 push eax
00401233 push offset byte_40217E
00401238 call sub_401308
0040123D add esp, 4
00401240 pop eax
00401241 cmp eax, ebx
00401243 jz short loc_40124C

```

short \$+2

```

00401245 call sub_401362
0040124A jmp short loc_4011E6

```

0040124C loc_40124C:

```

sub_401362 proc near
push 0 ; uType
call MessageBeep
push 30h ; uType
push offset aNoLuck ; "No luck!"
push offset aNoLuckThereMat ; "No luck there, mate!"
push dword ptr [ebp+8] ; hWnd
call MessageBoxA
ret

```

0000082D| 0040122D: WndProc+105 (Synchronized)

o a la address "0040124C" (Chico Bueno)

```

00401228 push offset String
0040122D call sub_40137E
00401232 push eax
00401233 push offset byte_40217E
00401238 call sub_401308
0040123D add esp, 4
00401240 pop eax
00401241 cmp eax, ebx
00401243 jz short loc_40124C

```

1Param

wParam

uParam

Msg

hWnd

```

004011E4 jmp short $+2
00401245 call sub_401362
0040124A jmp short loc_4011E6

```

sub_401340 proc near

```

push 30h ; uType
push offset Caption ; "Good work!!"
push offset Text ; "Great work, mate!\rNow try the next Cra..."
push dword ptr [ebp+8] ; hWnd
call MessageBoxA
ret

```

0040124C loc_40124C:

```

0040124C loc_40124C:
0040124C call sub_401340
00401251 jmp short loc_4011E6

```

00401251 ; sp-analysis Failed

i:18) [MSC v.1500 32 bit (Intel)]

Pintamos las dos ventanas para su mejor localización

```

00401228 push offset String
0040122D call sub_40137E
00401232 push eax
00401233 push offset byte_40217E
00401238 call sub_401308
0040123D add esp, 4
00401240 pop eax
00401241 cmp eax, ebx
00401243 jz short loc_40124C

```

00401245 call sub_401362

0040124A jmp short loc_4011E6

0040124C loc_40124C:

```

0040124C call sub_401340
00401251 jmp short loc_4011E6

```

00401251 ; sp-analysis Failed

00401251 WndProc endp : sp-analysis Failed

Volvemos al modo "Loader" con la barra espaciadora y ahora se ve mucho mejor.

ed Instruction External symbol

IDA View-A Program Segmentation Strings window Hex View-1 Database notepad Structures

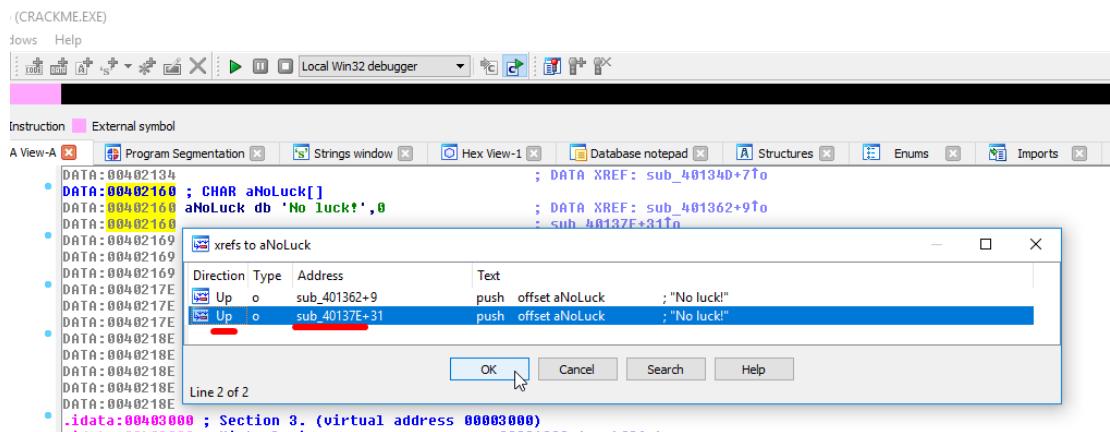
```

CODE:00401218 push ds:Instance ; hInstance
CODE:0040121E call DialogBoxParamA
CODE:00401223 cmp eax, 0
CODE:00401226 jz short loc_4011E6
CODE:00401228 push offset String
CODE:0040122D call sub_40137E
CODE:00401232 push eax
CODE:00401233 push offset byte_40217E
CODE:00401238 call sub_401308
CODE:0040123D add esp, 4
CODE:00401240 pop eax
CODE:00401241 cmp eax, ebx
CODE:00401243 jz short loc_40124C
CODE:00401245 call sub_401362
CODE:0040124C ; CODE XREF: WndProc+11B!j
CODE:0040124C loc_40124C: ; CODE XREF: WndProc+11B!j
CODE:0040124C call sub_401340
CODE:00401251 jmp short loc_4011E6
CODE:00401251 ; WndProc endp : sp-analysis Failed
CODE:00401251
CODE:00401253 ; ===== S U B R O U T I N E =====
CODE:00401253
CODE:00401253 ; Attributes: bp-based frame
CODE:00401253

```

Bien, con los dos "BP" colocados, y coloreadas las zonas de "Chico Malo y Chico Bueno" para su mejor identificación visual, ha llegado el momento de arrancar el crackme para descifrar que hace con el código, pero antes se nos ha pasado una cosa por alto.... si recordamos.... al principio del tute encontramos que había dos llamadas a "Chico Malo", y hasta el momento sólo hemos trabajado con la primera.

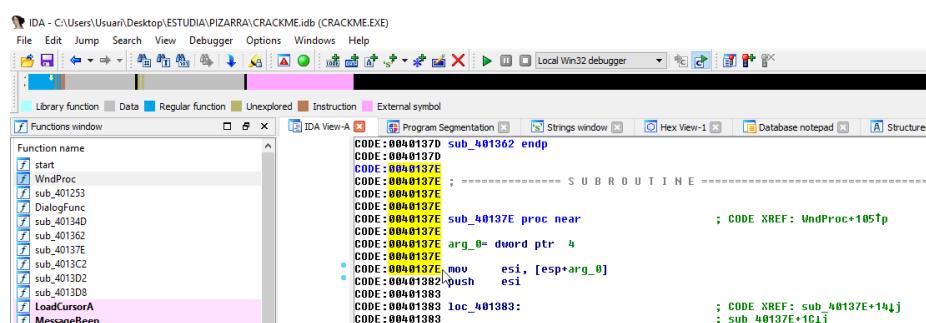
Vamos a averiguar de donde es llamada la segunda. Nos posicionamos sobre ella



le damos a "OK" y salimos en la address "CODE:004013AF" (40137E+31 = 4013AF)

```
CODE:004013AC ; CODE XREF: sub_40137E+0†j
CODE:004013AC loc_4013AC:
CODE:004013AC pop    esi
CODE:004013AD push    30h
CODE:004013AF push    offset aNoLuck
CODE:004013B4 push    offset aNoLuckThereMat
CODE:004013B9 push    dword ptr [ebp+8]
CODE:004013BC call    MessageBoxA
CODE:004013C1
CODE:004013C1 locret_4013C1: ; CODE XREF: sub_40137E+2C†j
CODE:004013C1 retn
CODE:004013C1 sub_40137E endp
CODE:004013C1
CODE:004013C2 ; ===== SUBROUTINE =====
CODE:004013C2
```

Si subimos con el cursor hasta el inicio de esta subrutina en la que nos encontramos, veremos directamente la address "CODE:0040137E"



Prosigamos..... bajamos de nuevo con el cursor hasta donde estábamos y vemos una comparación “**cmp**” y un salto condicional “**jb**” también muy sospechoso.

IDA View-A Program Segmentation Strings window Hex View-1 Database notepad Structures En

Code:

```
CODE:00401383 mov    al, [esi]
CODE:00401385 test   al, al
CODE:00401387 jz    short loc_40139C
CODE:00401389 cmp    al, 41h
CODE:0040138B jb    short loc_4013AC
CODE:0040138D cmp    al, 5Ah
CODE:0040138F jnb   short loc_401394
CODE:00401391 inc    esi
CODE:00401392 jmp    short loc_401383
CODE:00401394 :
CODE:00401394 loc_401394:
CODE:00401394 call   sub_4013D2
CODE:00401395 inc    esi
CODE:00401396 jmp    short loc_401383
CODE:0040139C :
CODE:0040139C loc_40139C:
CODE:0040139C pop    esi
CODE:0040139D call   sub_4013C2
CODE:0040139E xor    edi, 5678h
CODE:0040139F mov    eax, edi
CODE:004013A0 jmp    short locret_4013C1
CODE:004013A0 ; CODE XREF: sub_40137E+91†j
CODE:004013A0 loc_4013AC:
CODE:004013A0 pop    esi
CODE:004013A0 push   30h
CODE:004013A0 push   offset aNoLock
CODE:004013A0 push   offset ahLockThereHmat
CODE:004013A0 push   dword ptr [ebp+8]
CODE:004013A0 call   MessageBoxA
CODE:004013C1 :
CODE:004013C1 locret_4013C1:
CODE:004013C1 retn

; CODE XREF: sub_40137E+D†j
; uType
; "No Luck!"
; "No luck there, matet"
; hWnd
```

cmp al, 41h
jb short loc 4013AC

Pues, si los números van del 0h "0" al 9h "9"

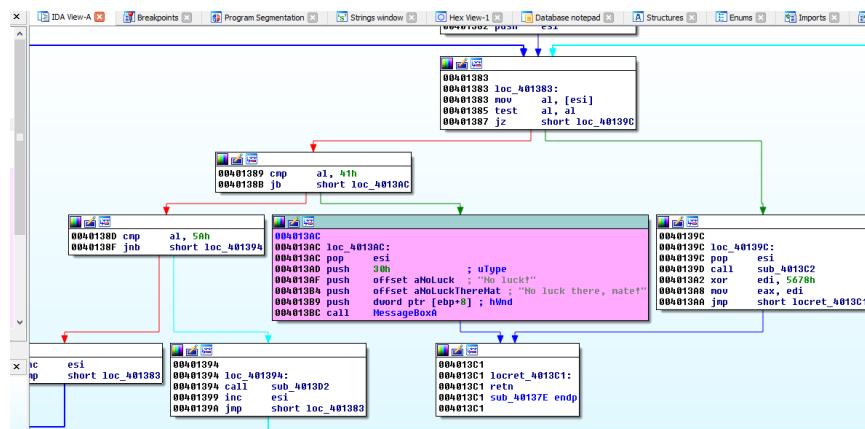
Si las letras MAYÚSCULAS van del 41h "A" al 5Ah "Z"

Si las letras minúsculas van del 61h "a" al 7Ah "z"

Entonces el resultado menor comparado con 41h "A" siempre será la de los números (o cualquier otro carácter que esté por debajo de 41h), llegando a la conclusión que lo que realmente está haciendo esta parte de código, es que si encuentra cualquier carácter numérico en el "Name" (o cualquier otro carácter que esté por debajo de 41h), en el contenido de "al", el salto condicional "jb" nos va a mandar directos a "**Chico Malo**"

Bien, pues con esta deducción, ya sabemos de dónde viene la segunda llamada a "**Chico Malo**" y lo que va a hacer.

Nos vamos al modo gráfico con la barra espaciadora, pintamos también esta segunda zona de “**Chico malo**” y nos queda así



Volvamos al modo "Loader"

Red Instruction External symbol

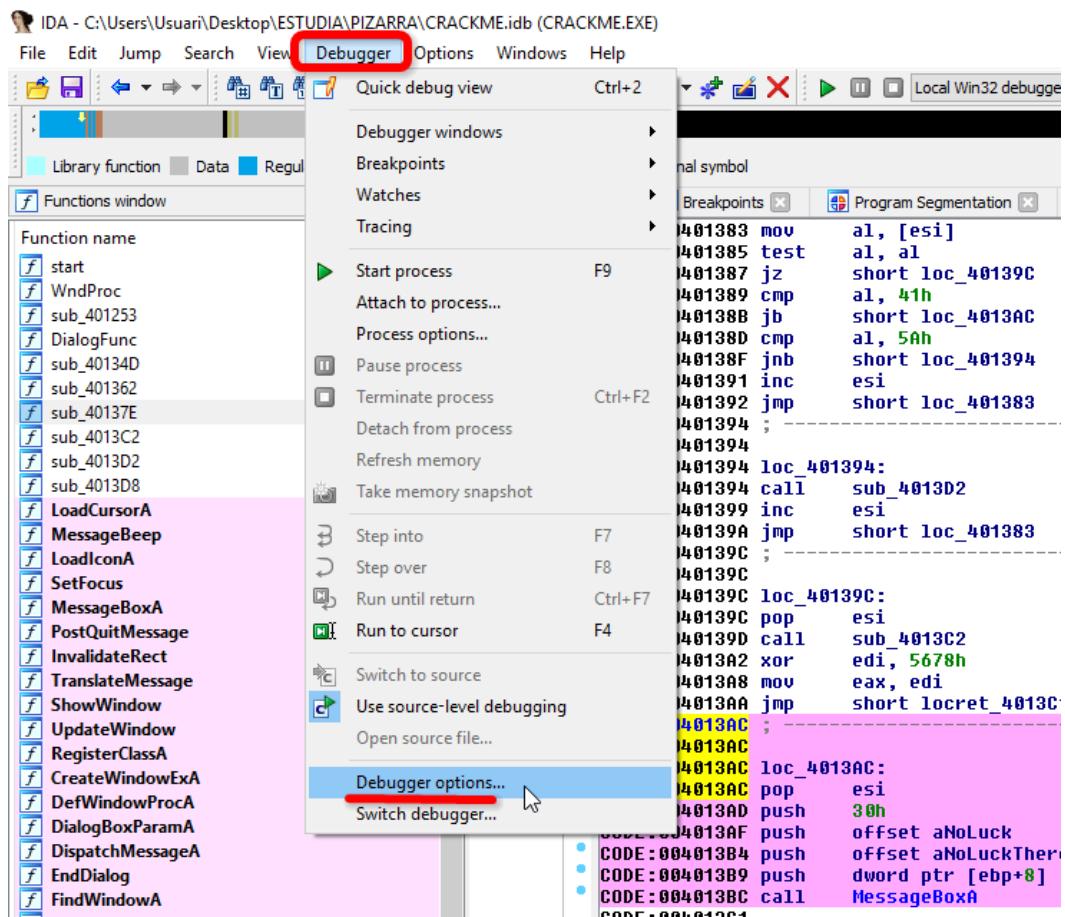
IDA View-A Breakpoints Program Segmentation Strings window Hex View-1 Database notepad Structures

```

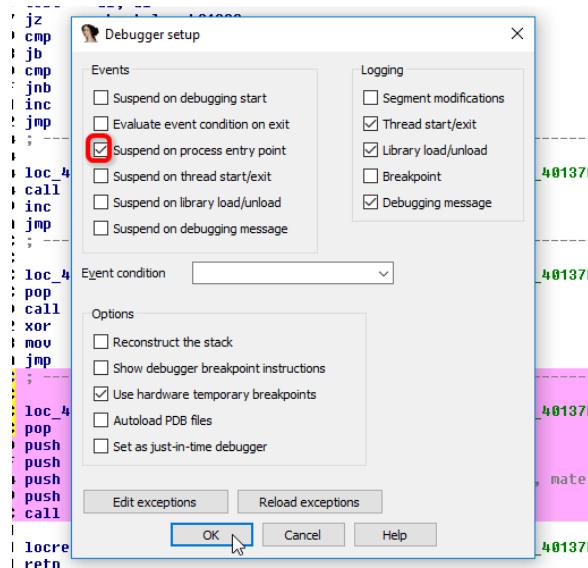
CODE:00401383 mov al, [esi]
CODE:00401385 test al, al
CODE:00401387 jz short loc_40139C
CODE:00401389 cmp al, 41h
CODE:0040138B jb short loc_4013AC
CODE:0040138D cmp al, 5Ah
CODE:0040138F jnb short loc_401394
CODE:00401391 inc esi
CODE:00401392 jmp short loc_401383
CODE:00401394 ; -----
CODE:00401394 loc_401394: ; CODE XREF: sub_40137E+11↑j
CODE:00401394 call sub_4013D2
CODE:00401399 inc esi
CODE:0040139A jmp short loc_401383
CODE:0040139C ; -----
CODE:0040139C loc_40139C: ; CODE XREF: sub_40137E+9↑j
CODE:0040139C pop esi
CODE:0040139D call sub_4013C2
CODE:004013A2 xor edi, 5678h
CODE:004013A8 mov eax, edi
CODE:004013AA jmp short locret_4013C1
CODE:004013AC ; -----
CODE:004013AC loc_4013AC: ; CODE XREF: sub_40137E+0↑j
CODE:004013AC pop esi
CODE:004013AD push 30h ; uType
CODE:004013AF push offset aNoLuck ; "No luck!""
CODE:004013B4 push offset aNoLuckThereMat ; "No luck there, mate!)"
CODE:004013B9 push dword ptr [ebp+8] ; hWnd
CODE:004013BC call MessageBoxA
CODE:004013C1 locret_4013C1: ; CODE XREF: sub_40137E+2C↑j
CODE:004013C1 ret
CODE:004013C1 sub_40137E endp
CODE:004013C1
000009AC 004013AC: sub_40137E:loc_4013AC (Synchronized with Hex View-1)

```

Y ahora sí que ya podemos arrancar el Crackme con el debugger.

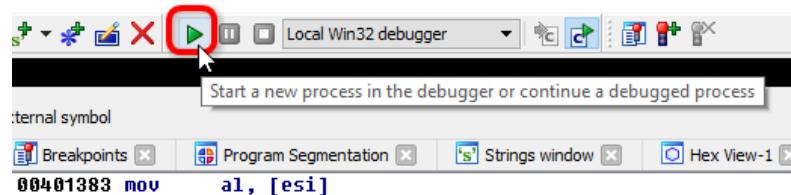


A mí siempre me gusta que pare en el "Entry Point", por lo que ponemos la palomilla en la opción "Suspend on process entry point"

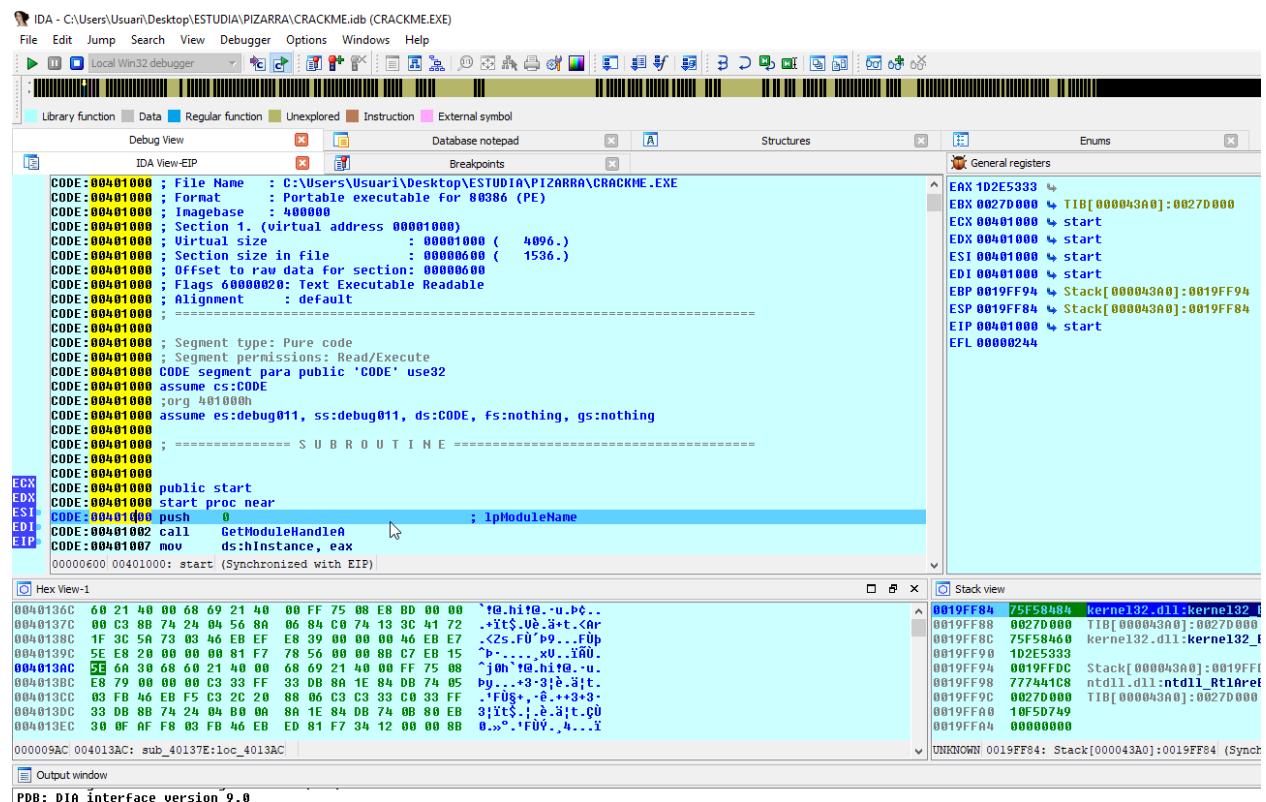


Le damos a "OK"

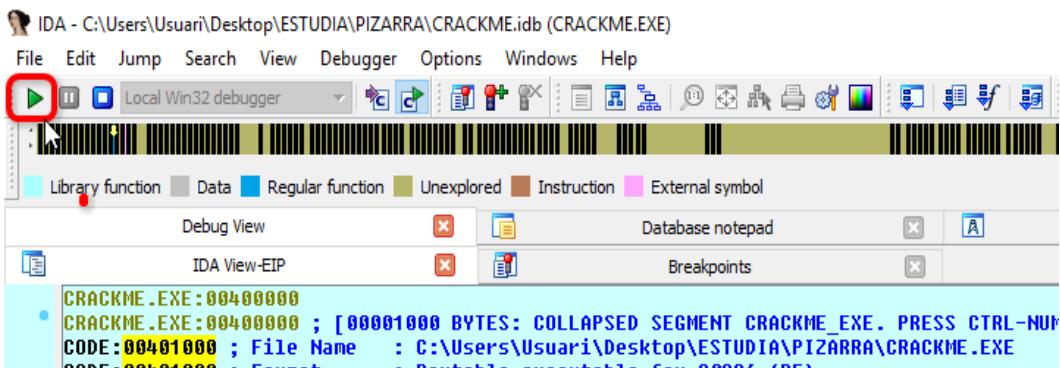
Ahora a "Start"



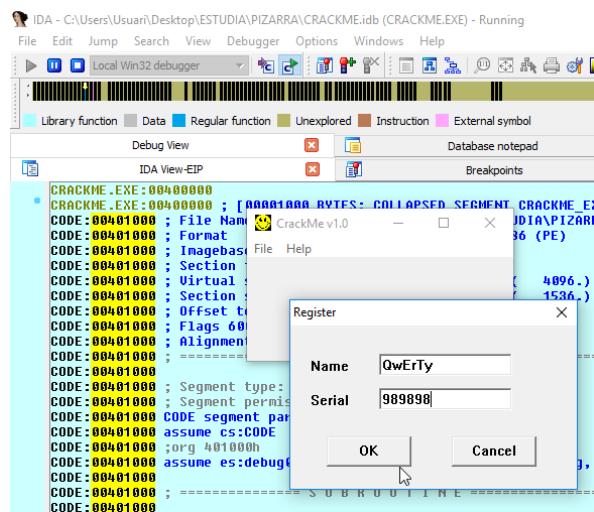
Y nos encontramos parados en el "OEP" CODE:00401000, con nuestro crackme en ejecución tal y como le pedimos a IDA que hiciera.



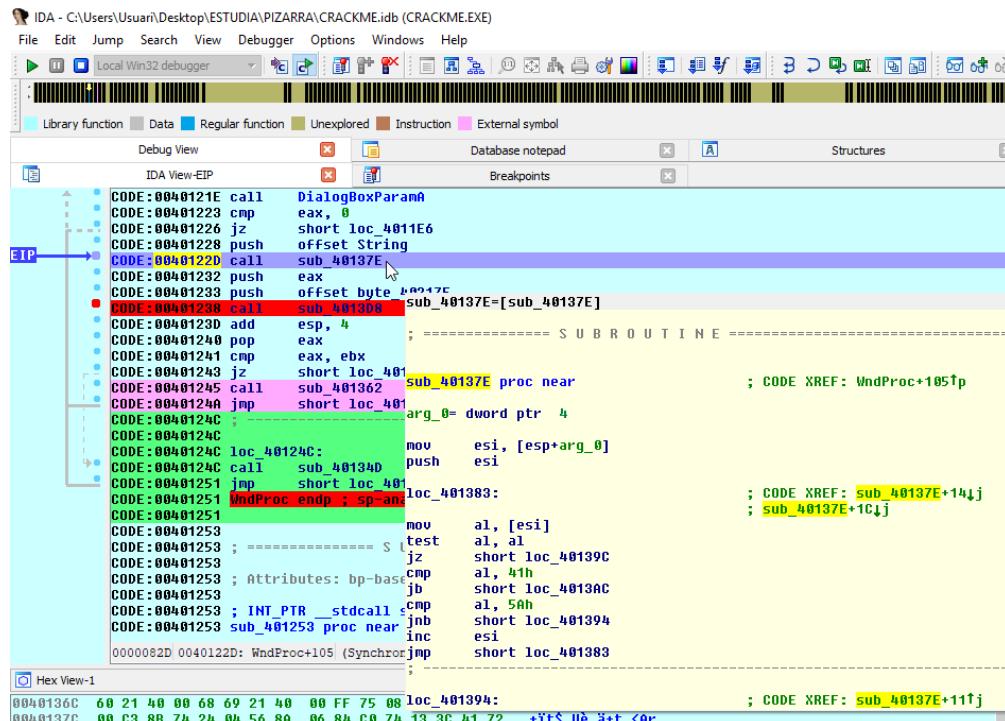
Volvemos a darle a "Start" para que siga corriendo



Rellenamos datos



Le damos a "OK", y IDA para en nuestro primer "BP" address "CODE:0040122D"



Entramos dentro del "call" y he agregado los comentarios a las instrucciones para su mejor comprensión

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

```

CODE:0040137E sub_40137E proc near ; CODE XREF: WndProc+105Tp
CODE:0040137E arg_0= dword ptr 4
CODE:0040137E
CODE:0040137E mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Name tipeado
CODE:00401382 push   esi             ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383:          ; CODE XREF: sub_40137E+14j
CODE:00401383 cmp    al, 41h         ; Mueve a AL el primer byte del contenido de ESI
CODE:00401385 test   al, al         ; Testea AL con AL
CODE:00401387 jz    short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp    al, 41h         ; Compara 4th con el valor hex de AL
CODE:0040138B jb    short loc_4013AC ; Salta a 4013AC si el resultado de la comparación es menor
CODE:0040138E cmp    al, 5Ah         ; Compara 4th con el valor de AL
CODE:0040138F jnb   short loc_401394 ; Salta a 40139C si el resultado de la CHP es mayor o igual
CODE:00401391 inc    esi             ; Pasa al siguiente byte de ESI
CODE:00401392 jmp    short loc_401383 ; Salta a 401383 (Inicio del Loop)
CODE:00401394
CODE:00401394 loc_401394:          ; CODE XREF: sub_40137E+11j
CODE:00401394 call   sub_4013D2
CODE:00401399 inc    esi             ; Salta a 40139C
CODE:0040139A jmp    short loc_401383 ; Salta a 401383
CODE:0040139C
CODE:0040139C loc_40139C:          ; CODE XREF: sub_40137E+9fj
CODE:0040139C pop    esi             ; Salta a 401383

```

La primera instrucción es "mov esi, [esp+arg_0]"

Pues traceamos para que se ejecute y en el registro "ESI" nos posicionamos con el cursor sobre la "String" y vemos esto

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

```

CODE:0040137E
CODE:0040137E arg_0= dword ptr 4
CODE:0040137E
CODE:0040137E mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Name tipeado
CODE:00401382 push   esi             ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383:          ; CODE XREF: sub_40137E+14j
CODE:00401383 cmp    al, [esi]        ; Mueve a AL el String db 51h, 77h, 45h, 72h, 54h, 79h
CODE:00401385 test   al, al         ; Testea AL con AL
CODE:00401387 jz    short loc_40139C ; Si el resultado
CODE:00401389 cmp    al, 41h         ; Compara 4th con el valor hex de AL
CODE:0040138B jb    short loc_4013AC ; Salta a 4013AC si el resultado de la comparación es menor
CODE:0040138E cmp    al, 5Ah         ; Compara 4th con el valor de AL
CODE:0040138F jnb   short loc_401394 ; Salta a 40139C si el resultado de la CHP es mayor o igual
CODE:00401391 inc    esi             ; Pasa al siguiente byte de ESI
CODE:00401392 jmp    short loc_401383 ; Salta a 401383 (Inicio del Loop)
CODE:00401394
CODE:00401394 loc_401394:          ; CODE XREF: sub_40137E+11j

```

Yo, ya sé que 51h 77h 45h 72h 54h 79h es mi "Name" tipeado en hexadecimal.....no es por alardear pero es que ya llevo algunos años tipeando QwErTyje.je.je)

Pero para que IDA nos lo muestre, clicamos sobre la flechita azul

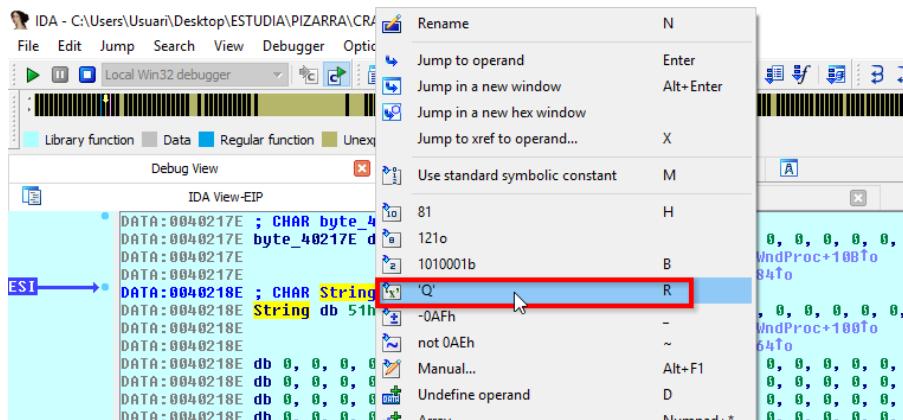
EAX 00000001
EBX 00401128
ECX 9C5DF2CE
EDX 00000000
ESI 0040218E DATA:String
EDI 00000111
EBP 0019FDE0
ESP 0019FDCC
EIP 00401382
EFL 00000202

Y ahora en la ventana del desensamblado

The screenshot shows the IDA Pro interface with the following details:

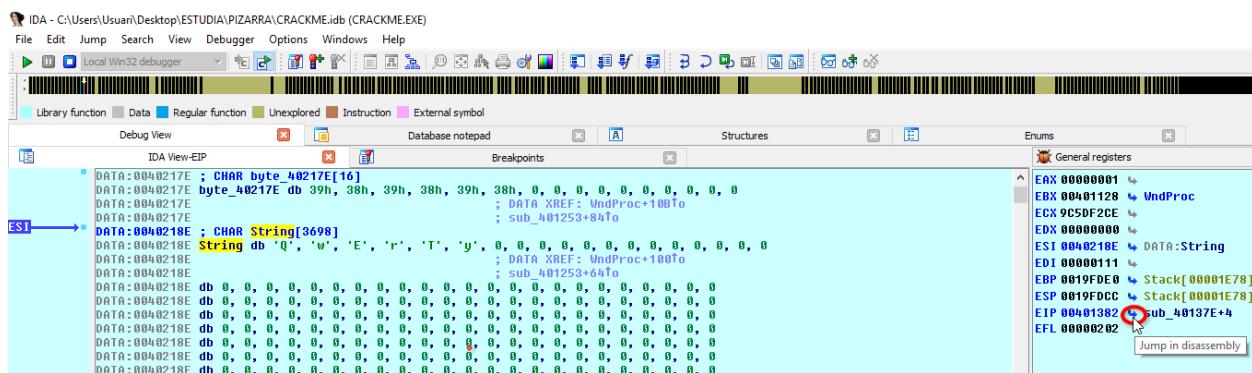
- Title Bar:** IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZZARA\CRACKME.idb (CRACKME.EXE)
- Menu Bar:** File Edit Jump Search View Debugger Options Windows Help
- Toolbar:** Includes icons for opening files, saving, zooming, and various analysis tools.
- Status Bar:** Shows "Library function" and "Data Regular function Unexplored Instruction External symbol".
- Registers Panel:** Shows the general registers (EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP, EFL) with their current values.
- Code View:** The assembly code pane displays several DATA and DB instructions. A red box highlights the string "51h, 77h, 45h, 72h, 54h, 79h," which corresponds to the string "Hello, world!" in memory.
- Breakpoints Panel:** Shows a list of breakpoints, all of which are currently disabled (indicated by a blue dot).
- Database Notepad:** A small window showing the database note "DATA XREF: VndProc+100To".
- Structures Panel:** Shows the structure "sub_4B1253+84To" which contains the highlighted string.
- Enums Panel:** Shows the enum "General registers" with entries for EAX through EFL.

Nos posicionamos sobre "String", click derecho de ratón y le damos a "Q"



Y ahora nos muestra claramente la conversión

Acto seguido le damos a la flechita azul del registro “EIP” para volver a la pantalla donde nos encontrábamos anteriormente en el desensamblado



y aquí estamos de nuevo, donde ahora podemos ver clarísimamente que el valor de "ESI" es nuestro "Name" tipeado

IDA - C:\Users\Usuario\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

```

CODE:0040137E
CODE:0040137E arg_0= dword ptr 4
CODE:0040137E
CODE:0040137E mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Name tipeado
CODE:00401382 push   esi               ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383:
CODE:00401383 mov    al, [esi]
CODE:00401383 test   al, al
CODE:00401387 jz    short loc_40139C ; Testea AL con A
CODE:00401389 cmp   al, 41h
CODE:0040138B jb    short loc_4013AC ; Si el resultado de la comparación es menor
CODE:0040138D cmp   al, 50h
CODE:0040138F jnb   short loc_401394 ; Salta a 40139C si el resultado de la comparación es mayor o igual
CODE:00401391 inc   esi
CODE:00401394 jmp   short loc_401383 ; Pasa al siguiente byte de ESI
CODE:00401394
CODE:00401394 loc_401394:
CODE:00401394 endI  sub_40137E+11fj ; CODE XREF: sub_40137E+11fj

```

La próxima instrucción es un “push esi”, por tanto si en “ESI” guarda nuestro Name, pues lo va a meter al “Stack” (primera carta del mazo)
Traceamos para que se ejecute esa instrucción, y efectivamente eso es lo que ha hecho

Stack view

0019FDC8	0040218E	DATA:String
0019FDCC	00402132	WndProc+10A
0019FDD0	0040218E	DATA:String
0019FDD4	00401128	WndProc
0019FDD8	00000011	
0019FDDC	00000000	
0019FDE0	0019FE0C	Stack[00001E78]:0019FE0C
0019FDE4	7637BF1B	user32.dll:user32_AddClipboardFormatListener+49B

La próxima instrucción es un “mov al, [esi]”, va mover a “AL” el primer byte del contenido de “ESI”

si “ESI” = QwErTy moverá 51h = “Q”

traceamos y aquí lo tenemos

```

CODE:0040137E
CODE:0040137E arg_0= dword ptr 4
CODE:0040137E
CODE:0040137E mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Name tipeado
CODE:00401382 push   esi               ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383:
CODE:00401383 mov    al, [esi]          ; Mueve a AL el primer byte del contenido de ESI
CODE:00401385 test   al, al           ; Testea AL con AL
CODE:00401387 jz    short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp   al, 41h           ; Compara 41h con el valor hex de AL
CODE:0040138B jb    short loc_4013AC ; Salta a 40139C si el resultado de la comparación es menor
CODE:0040138D cmp   al, 50h           ; Compara 41h con el valor de AL
CODE:0040138F jnb   short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
CODE:00401391 inc   esi
CODE:00401394 jmp   short loc_401383 ; Pasa al siguiente byte de ESI
CODE:00401394
CODE:00401394 loc_401394:
CODE:00401394 endI  sub_40137E+11fj ; CODE XREF: sub_40137E+11fj

```

Ahora con “test al, al” va a testear “al,al”, al = 51h = “Q”, y después con el “jz short loc_40139C” lo que va a hacer es que si el resultado del testeo de “al,al” es cero “0” el salto condicional “jz” nos llevaría a “loc_40139C” que es la salida del “Loop” en el que nos encontramos. Por el contrario si no es “0” nos deja continuar.

(Eso sucederá cuando ya no queden más caracteres para leer de nuestro Name tipeado)

```

CODE:0040137E mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Name tipeado
CODE:00401382 push   esi               ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383:
CODE:00401383 mov    al, [esi]
CODE:00401385 test   al, al           ; Testea AL con AL
CODE:00401387 jz    short loc_40139C ; Si el resultado del testeо AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp    al, 41h          ; Compara 41h con el valor hex de AL
CODE:0040138B jb    short loc_4013AC ; Salta a 4013AC si el resultado de la comparacióп es menor
CODE:0040138F cmp    al, 5Ah          ; Compara 5Ah con el valor de AL
CODE:00401391 jnb   short loc_401394 ; Salta a 401394 si el resultado de la CMP es mayor o igual
CODE:00401392 jmp   short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:00401394
CODE:00401394 loc_401394:
CODE:00401394 call   sub_4013D2      ; CODE XREF: sub_40137E+11†j
CODE:00401399 inc    esi               ; Pasa al siguiente byte de ESI
CODE:00401399 jmp   short loc_401383 ; Salta a 401383
CODE:0040139C
CODE:0040139C loc_40139C:
CODE:0040139C pop    esi               ; CODE XREF: sub_40137E+9†j
CODE:0040139D call   sub_4013C2
CODE:0040139D pop    esi
CODE:00401399 jmp   short loc_401383
CODE:0040139C
CODE:0040139C loc_40139C:
CODE:0040139C pop    esi
CODE:0040139D call   sub_4013C2
CODE:0040139D pop    esi
CODE:00401399 xor    edi, 5678h
CODE:00401398 mov    eax, edi
CODE:004013AA jmp   short locret_4013C1

```

La siguiente es un "cmp al, 41h", va a comparar 51h "Q" con 41h "A", y después con "jb short loc_4013AC" va a decidir que si el resultado de la comparación anterior es menor, saltaremos a la "loc_4013AC", que también es una de las zonas de "Chico malo".

Por si tenemos dudas, solo basta posicionar el cursor sobre "loc_4013AC" y veremos el "MessageBoxA"

```

CODE:0040137E sub_40137E proc near ; CODE XREF: WndProc+105†p
CODE:0040137E arg_0= dword ptr 4
CODE:0040137E
CODE:0040137E mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Name tipeado
CODE:00401382 push   esi               ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383:
CODE:00401383 mov    al, [esi]
CODE:00401385 test   al, al           ; Testea AL con AL
CODE:00401387 jz    short loc_40139C ; Si el resultado del testeо AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp    al, 41h          ; Compara 41h con el valor hex de AL
CODE:0040138B jb    short loc_4013AC ; Salta a 4013AC si el resultado de la comparacióп es menor
CODE:0040138F cmp    al, 5Ah          ; Compara 5Ah con el valor de AL
CODE:00401391 jnb   short loc_401394 ; Salta a 401394 si el resultado de la CMP es mayor o igual
CODE:00401392 jmp   short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:00401394
CODE:00401394 loc_401394:
CODE:00401394 call   sub_4013D2      ; CODE XREF: sub_40137E+11†j
CODE:00401399 inc    esi               ; Pasa al siguiente byte de ESI
CODE:00401399 jmp   short loc_401383 ; Salta a 401383
CODE:0040139C
CODE:0040139C loc_40139C:
CODE:0040139C pop    esi               ; CODE XREF: sub_40137E+9†j
CODE:0040139D call   sub_4013C2
CODE:0040139D pop    esi
CODE:00401399 xor    edi, 5678h
CODE:00401398 mov    eax, edi
CODE:004013AA jmp   short locret_4013C1

```

; uType
; "No luck!"
; "No luck there, mate!"
; hWnd

Y en el modo gráfico

The screenshot shows the assembly view of the debugger. The code is as follows:

```

00401389 cmp    al, 41h ; Compara 41h con el valor hex de AL
0040138B jb     short loc_4013AC; ; Salta a 4013AC si el resultado de la comparación es menor

Ah      ; Compara 41h con el valor de AL
loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual

004013AC loc_4013AC:
004013AC pop    esi
004013AD push   30h ; uType
004013AB push   offset aNoLuck ; "No luck!"
004013B4 push   offset aNoLuckThereHat ; "No luck there, mate!"
004013B9 push   dword ptr [ebp+8] ; hWnd
004013BC call   MessageBoxA ; MessageBoxA

```

A red box highlights the comparison and jump instructions. A red arrow points from the jump instruction to the label 'loc_4013AC'. Another red box highlights the assembly code starting at address 4013AC, which contains the MessageBoxA call.

Si por el contrario el resultado de la comparación es mayor, entonces nos dejará continuar.

Resumiendo:

Si los n鷁eros van del 0h "0" al 9h "9"

Si las letras MAYÚSCULAS van del 41h "A" al 5Ah "Z"

Si las letras minúsculas van del 61h "a" al 7Ah "z"

Entonces el resultado menor comparado con 41h "A" siempre será la de los números (o cualquier otro carácter que esté por debajo de 41h), pues lo que realmente está haciendo esta parte de código, es que si encuentra cualquier carácter numérico (o cualquier otro carácter que esté por debajo de 41h), como por ejemplo un espacio, cuyo valor es 20h) en el "Name", nos va a mandar directos a "**Chico Malo**"

Pero con que no es nuestro caso, ya que 51h "Q" es mayor que 41h "A", traceamos y seguimos.

Ahora viene un "cmp al, 5Ah" por lo que va a comprar 51h "Q" con 5Ah "Z" y después con la instrucción "jnb short loc_401394", va a decidir que si el resultado de la comparación es mayor o igual, nos llevará a "loc_401394" que es un "call". Si por el contrario el resultado de la comparación es menor, entonces nos dejará continuar. En nuestro caso con que 51h "Q" es menor que 5Ah "Z" nos deja continuar.

Por lo que:

Los caracteres numéricos (o cualquier otro carácter que esté por debajo de 41h) en el "Name" ya lo ha filtrado anteriormente.

Si las letras MAYÚSCULAS van del 41h "A" al 5Ah "Z"

Si las letras minúsculas van del 61h "a" al 7Ah "z"

Está claro que solo nos mandará a "loc_401394" donde se encuentra el "call" cuando encuentre una letra minúscula. (en nuestro Name tipeado, esto sucederá cuando haga la comparación con los caracteres "w,r,y")

The screenshot shows the IDA Pro interface with the following details:

- Title Bar:** IDA - C:\Users\Usuario\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)
- Menu Bar:** File Edit Jump Search View Debugger Options Windows Help
- Toolbar:** Local Win32 debugger, with icons for file operations, search, and debugger controls.
- Status Bar:** Library function Data Regular function Unexplored Instruction External symbol
- Views:** Debug View, Database notepad, Structures.
- Code View:** Shows assembly code with comments:
 - CODE : 0040137E ; Mueve a ESI nuestro Name tipeado
 - CODE : 0040137E mov esi, [esp+arg_0] ; Mete nuestro Name al Stack
 - CODE : 00401382 push esi
 - CODE : 00401383
 - CODE : 00401383 loc_401383:
 - CODE : 00401383 mov al, [esi]
 - CODE : 00401385 test al, al
 - CODE : 00401387 jz short loc_40139C ; Mueve a AL el primer byte del contenido de ESI
 - CODE : 00401389 cmp al, 41h ; Testea AL con AL
 - CODE : 0040138B jb short loc_4013AC ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
 - CODE : 0040138D cmp al, 5Ah ; Compara 41h con el valor hex de AL
 - CODE : 0040138F jnb short loc_401394 ; Compara 41h con el valor de AL
 - CODE : 00401391 inc esi
 - CODE : 00401392 jmp short loc_401383 ; Salta a 4013AC si el resultado de la comparación es menor
 - CODE : 00401394 ; Pasa al siguiente byte de ESI
 - CODE : 00401394 loc_401394:
 - CODE : 00401394 call sub_4013D2 ; Salta a 40139C si el resultado de la CMP es mayor o igual
 - CODE : 00401399 inc esi
 - CODE : 0040139A jmp short loc_401383
 - CODE : 0040139C :
 - CODE : 0040139C loc_40139C:
 - CODE : 0040139D pop esi
 - CODE : 0040139D call sub_4013C2 ; CODE XREF: sub_40137E+91j
 - CODE : 004013A2 xor edi, 5678h
 - CODE : 004013A8 mov eax, edi

Tracemos y ahora tenemos un "inc esi", que lo que va a hacer es **incrementar en "1h" el valor del registro "ESI"**

Como podemos observar en la imagen inferior, el valor de "ESI" es "0040218E", y hasta ahora hemos trabajado con el primer carácter de nuestro Name 51h "Q"

Traceamos para que se ejecute la instrucción y ahora "ESI" vale "0040218F" (0040218E+1) con lo cual, pasaremos al siguiente byte, que será 77h "w"

La siguiente instrucción es un “`jmp short loc_401383`”, este salto incondicional tal y como también nos muestra la imagen superior, la **flecha verde** de IDA nos indica que nos manda directos a “`CODE:00401383`”. tracemos para que se

ejecute y ahora nos encontramos de nuevo al inicio del “Loop”, donde observamos que “AL” todavía sigue valiendo 51h “Q”

```

CODE:00401382 push    esi
CODE:00401383
CODE:00401383 loc_401383:
CODE:00401383
CODE:00401383 mov     al, [esi]      ; Mueve a AL el primer byte del contenido de ESI
CODE:00401385 test    al, al       ; Testea AL con AL
CODE:00401387 jz     short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp    al, 41h      ; Compara 41h con el valor hex de AL
CODE:0040138B jb     short loc_4013AC ; Salta a 4013AC si el resultado de la comparación es menor
CODE:0040138D cmp    al, 5Ah      ; Compara 5Ah con el valor de AL
CODE:0040138F jnb    short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
CODE:00401391 inc    esi         ; Pasa al siguiente byte de ESI
CODE:00401392 jmp    short loc_401383 ; Salta a 401383 (Inicio del Loop)
CODE:00401394
CODE:00401394 loc_401394:
CODE:00401394 call    sub_4013D2
CODE:00401399 inc    esi         ; Pasa al siguiente byte de ESI
CODE:00401399 jmp    short loc_401383 ; Salta a 401383 (Inicio del Loop)
CODE:0040139C

```

La siguiente instrucción es “`mov al, [esi]`”, por lo que como ya dijimos anteriormente va a mover a “AL” el siguiente byte del contenido de “ESI” (String+1) es decir, ahora “AL” valdrá 77h “w”,

“ESI” 51h 77h 45h 72h 54h 79h “QwErTy”

Traceamos para que se ejecute y aquí lo tenemos

```

CODE:00401382 push    esi
CODE:00401383
CODE:00401383 loc_401383:
CODE:00401383
CODE:00401383 mov     al, [esi]      ; Mueve a AL el primer byte del contenido de ESI
CODE:00401385 test    al, al       ; Testea AL con AL
CODE:00401387 jz     short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp    al, 41h      ; Compara 41h con el valor hex de AL
CODE:0040138B jb     short loc_4013AC ; Salta a 4013AC si el resultado de la comparación es menor
CODE:0040138D cmp    al, 5Ah      ; Compara 5Ah con el valor de AL
CODE:0040138F jnb    short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
CODE:00401391 inc    esi         ; Pasa al siguiente byte de ESI
CODE:00401392 jmp    short loc_401383 ; Salta a 401383 (Inicio del Loop)
CODE:00401394
CODE:00401394 loc_401394:
CODE:00401394 call    sub_4013D2
CODE:00401399 inc    esi         ; Pasa al siguiente byte de ESI
CODE:00401399 jmp    short loc_401383 ; Salta a 401383 (Inicio del Loop)
CODE:0040139C

```

La siguiente es un “`test al,al`” Testea “AL,AL” y ahora “AL” = 77h “w”

```

CODE:00401382 push    esi
CODE:00401383
CODE:00401383 loc_401383:
CODE:00401383
CODE:00401383 mov     al, [esi]      ; Mueve a AL el primer byte del contenido de ESI
CODE:00401385 test    al, al       ; Testea AL con AL
CODE:00401387 jz     short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp    al, 41h      ; Compara 41h con el valor hex de AL
CODE:0040138B jb     short loc_4013AC ; Salta a 4013AC si el resultado de la comparación es menor
CODE:0040138D cmp    al, 5Ah      ; Compara 5Ah con el valor de AL
CODE:0040138F jnb    short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
CODE:00401391 inc    esi         ; Pasa al siguiente byte de ESI
CODE:00401392 jmp    short loc_401383 ; Salta a 401383 (Inicio del Loop)
CODE:00401394
CODE:00401394 loc_401394:
CODE:00401394 call    sub_4013D2
CODE:00401399 inc    esi         ; Pasa al siguiente byte de ESI
CODE:00401399 jmp    short loc_401383 ; Salta a 401383 (Inicio del Loop)
CODE:0040139C

```

Una vez traceada la anterior instrucción ahora tenemos un “`jz short loc_40139C`”, con lo cual aquí va a decidir de nuevo que si el valor de “AL” fuese cero nos llevaría a “loc_40139C” que es la salida del “Loop”, pero con que ahora “AL” vale 77h “w” nos dejará continuar

IDA - C:\Users\Usuari\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures Enums

IDA View-EIP Breakpoints

```

CODE:00401382 push    esi ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383: ; CODE XREF: sub_40137E+14j
CODE:00401383
CODE:00401385 mov     al, [esi] ; Mueve a AL el primer byte del contenido de ESI
CODE:00401385 test   al, al ; Testea AL con AL
CODE:00401387 jz    short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp    al, 41h ; Compara 41h con el valor hex de AL
CODE:0040138B jb    short loc_4013AC ; Salta a 4013AC si el resultado de la comparación es menor
CODE:0040138B cmp    al, 5Ah ; Compara 41h con el valor hex de AL
CODE:0040138F jnb   short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
CODE:00401391 inc    esi
CODE:00401392 jmp    short loc_401383 ; Pasa al siguiente byte de ESI
CODE:00401392 jmp    short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:00401394
CODE:00401394
CODE:00401399 loc_401394: ; CODE XREF: sub_40137E+11fj
CODE:00401399 call   sub_4013D2
CODE:0040139A inc    esi
CODE:0040139A jmp    short loc_401383
CODE:0040139C
CODE:0040139C loc_40139C: ; CODE XREF: sub_40137E+9tj
CODE:0040139C pop    esi
CODE:0040139D call   sub_4013C2
CODE:004013A2 xor    edi, 5678h

```

Traceamos y ahora viene la comparación "cmp al, 41h" donde compara 77h "w" con 41h "A"

Y en la siguiente instrucción "jb short loc_4013AC" , con que el resultado de la comparación (77h "w" es mayor que 41h "A") entonces nos deja continuar

IDA - C:\Users\Usuari\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures Enums

IDA View-EIP Breakpoints

```

CODE:00401382 push    esi ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383: ; CODE XREF: sub_40137E+14j
CODE:00401383
CODE:00401385 mov     al, [esi] ; Mueve a AL el primer byte del contenido de ESI
CODE:00401385 test   al, al ; Testea AL con AL
CODE:00401387 jz    short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp    al, 41h ; Compara 41h con el valor hex de AL
CODE:0040138B jb    short loc_4013AC ; Salta a 4013AC si el resultado de la comparación es menor
CODE:0040138B cmp    al, 5Ah ; Compara 41h con el valor hex de AL
CODE:0040138F jnb   short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
CODE:00401391 inc    esi
CODE:00401392 jmp    short loc_401383 ; Pasa al siguiente byte de ESI
CODE:00401392 jmp    short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:00401394
CODE:00401394
CODE:00401399 loc_401394: ; CODE XREF: sub_40137E+11fj
CODE:00401399 call   sub_4013D2
CODE:0040139A inc    esi
CODE:0040139A jmp    short loc_401383
CODE:0040139C
CODE:0040139C loc_40139C: ; CODE XREF: sub_40137E+9tj
CODE:0040139C pop    esi
CODE:0040139D call   sub_4013C2
CODE:004013A2 xor    edi, 5678h

```

A continuación el "cmp al, 5Ah" compara el contenido de "AL" 77h "w" con 5Ah "Z" , y con el "jnb short loc_401394" va a decidir que si el resultado de la comparación anterior es mayor o igual, (77h es mayor que 5Ah) nos llevará a la "call" que se encuentra en "loc_401394"

Recordemos que:

Si las letras MAYÚSCULAS van del 41h "A" al 5Ah "Z"

Y las letras minúsculas van del 61h "a" al 7Ah "z"

Por lo tanto, siempre que encuentre una letra minúscula (o cualquier otro carácter que esté por encima de 5Ah) nos mandará directos a "loc_401394" donde se encuentra el "call".

Traceamos hasta "CODE:0040138F" y en nuestro caso ya vemos que iremos directos a esa "call" (También nos lo indica IDA con la flecha verde)

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures

IDA View-EIP Breakpoints

```

CODE:00401382 push    esi ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383: ; CODE XREF: sub_40137E+14j
CODE:00401383
CODE:00401383 mov     al, [esi] ; Mueve a AL el primer byte del contenido de ESI
CODE:00401385 test   al, al ; Testea AL con AL
CODE:00401387 jz     short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp    al, 41h ; Compara 41h con el valor hex de AL
CODE:0040138B jb     short loc_4013AC ; Salta a 4013AC si el resultado de la comparación es menor
CODE:0040138D cmp    al, 5Ah ; Compara 5Ah con el valor de AL
CODE:0040138F jnb    short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
EIP
CODE:00401391 inc    esi ; Pasa al siguiente byte de ESI
CODE:00401392 jmp    short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:00401394
CODE:00401394 loc_401394: ; CODE XREF: sub_40137E+11f
CODE:00401394 call   sub_4013D2
CODE:00401399 inc    esi
CODE:0040139A jmp    short loc_401383
CODE:0040139C
CODE:0040139C loc_40139C: ; CODE XREF: sub_40137E+9tj
CODE:0040139C pop    esi

```

Si le damos a la barra espaciadora de nuestro teclado, también lo vemos en modo gráfico

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures Enums

IDA View-EIP Breakpoints

General registers

```

00401389 cmp    al, 41h ; Compara 41h con el valor hex de AL
0040138B jb     short loc_4013AC; Salta a 40139C si el resultado de la comparación
0040138D cmp    al, 5Ah ; Compara 41h con el valor de AL
0040138F jnb    short loc_401394; Salta a 40139C si el resultado de la CMP es mayor o igual
00401391 inc    esi ; Pasa al siguiente byte de ESI
00401392 jmp    short loc_401383 ; Salta a 401383 (inicio del Loop)
00401394 loc_401394: ; CODE XREF: sub_40137E+11f
00401394 call   sub_4013D2
00401399 inc    esi
0040139A jmp    short loc_401383

```

Registers:

- EX 00000077
- EBX 00401128 VndProc
- ECX 8BF9DFE5
- EDX 00000000
- ESI 0040218F DATA:String+1
- EDI 00000111
- EBP 0019FD00 Stack[000045C8]
- ESP 0019FD08 Stack[000045C8]
- EIP 0040138F sub_40137E+11
- EFL 00000216

Hex View-1

```

00401362 6A 00 E8 AD 00 00 00 6A 30 68 60 21 40 00 00 69 j.Pi...j0h`hi
00401372 29 40 00 00 C3 00 74 24 04 f@.u.Pc...@t$.
00401382 50 80 00 84 C0 74 13 3C 41 72 1F 3C 58 73 03 46 U@.t.Kn.<2s.F
00401392 ED EF E8 39 00 00 00 46 EB E7 5E E8 28 00 00 00 U^9...Rfb^p...
004013A2 01 F7 78 56 00 00 8C C7 EB 15 SE 6A 30 68 60 21 ..xU..iR0..i0h't

```

Stack view

0019FD08 0040218E DATA:String
0019FD00 0019FD22 VndProc+10h
0019FD00 0040218E DATA:String
0019FD04 00401128 VndProc
0019FD08 00000111

Volvemos al modo debugger y con un trazo más, efectivamente nos encontramos en esa "call"

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures

IDA View-EIP Breakpoints

```

CODE:00401382 push    esi ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383: ; CODE XREF: sub_40137E+14j
CODE:00401383
CODE:00401383 mov     al, [esi] ; Mueve a AL el primer byte del contenido de ESI
CODE:00401385 test   al, al ; Testea AL con AL
CODE:00401387 jz     short loc_40139C ; Si el resultado del testeo AL,AL es 0 :
CODE:00401389 cmp    al, 41h ; Compara 41h con el valor hex de AL
CODE:0040138B jb     short loc_4013AC ; Salta a 4013AC si el resultado de la comparación es menor
CODE:0040138D cmp    al, 5Ah ; Compara 41h con el valor de AL
CODE:0040138F jnb    short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
EIP
CODE:00401391 inc    esi ; Pasa al siguiente byte de ESI
CODE:00401394 loc_401394: ; CODE XREF: sub_40137E+11f
CODE:00401394 call   sub_4013D2
CODE:00401399 inc    esi
CODE:0040139A jmp    short loc_401383
CODE:0040139C
CODE:0040139C loc_40139C: ; CODE XREF: sub_40137E+9tj
CODE:0040139C pop    esi

```

Nos posicionamos con el cursor sobre "sub_4013D2" y IDA nos muestra las entrañas de lo que esconde esa "call"

```

CODE:00401380 loc_401383:
CODE:00401383 mov    al,[esi]
CODE:00401383 test   al,al
CODE:00401385 jz     short loc_40139C
CODE:00401387 cmp    al,41h
CODE:00401389 jb    short loc_4013AC
CODE:0040138B cmp    al,5AH
CODE:0040138F jne   short loc_401394
CODE:00401391 inc    esi
CODE:00401392 jmp    short loc_401383
CODE:00401394 ; 
CODE:00401394 loc_401394:
CODE:00401394 call   sub_4013D2 ; CODE XREF: sub_40137E+11↑j
CODE:00401399 inc    esi
CODE:0040139A jmp    short loc_4013D2=[sub_4013D2]
CODE:0040139C ;
CODE:0040139C ; ====== S U B R O U T I N E ======
00000994 00401394: sub_40137E:loc_401394

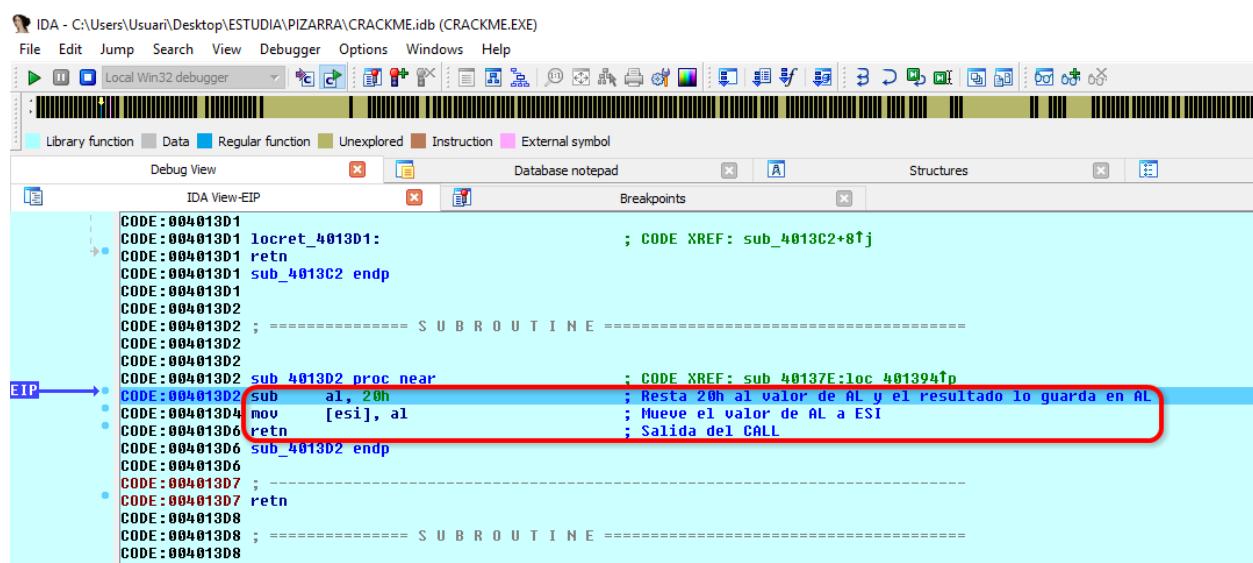
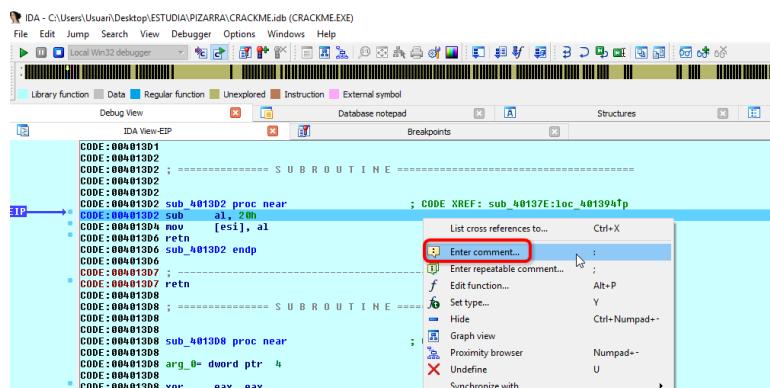
Hex View-1
Sub_4013D2 proc near
00401362 6A 00 E8 AD 00 00 00 6A 30 68 60 ;Sub al, 20h
00401372 21 40 FF 75 08 E8 BD 00 00 00 [esi], al
00401382 56 80 00 84 C0 74 13 3C 41 72 1F retn
00401392 EB EF E8 39 00 00 00 A6 EB E8 D0 endp
004013A2 81 F7 78 56 00 00 88 C7 EB 15 5E 6A 30 68 60 21 .,XU..^jh0h ?
004013B2 6A 00 6A 60 21 40 00 FF 75 88 F8 70 00 00 00 C3 A hi a ..bu +

```

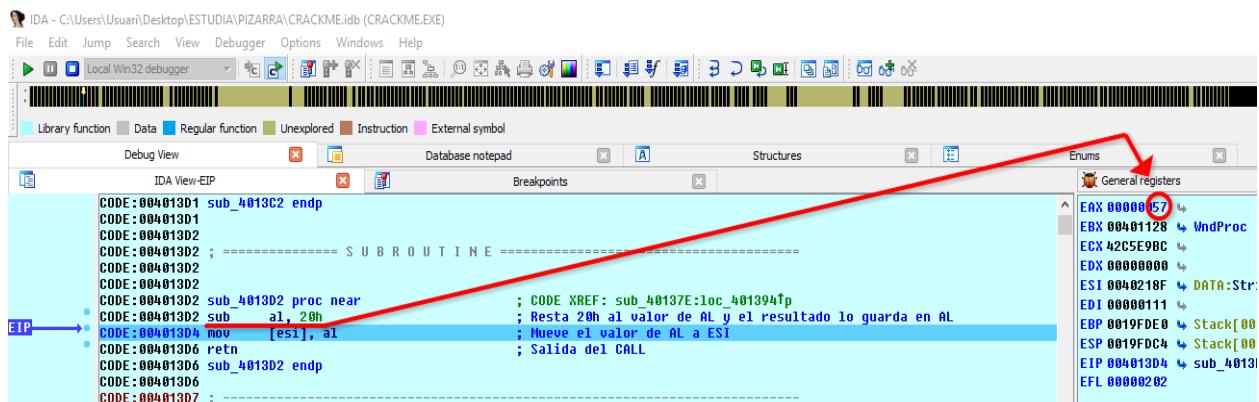
Vemos que con la instrucción "sub al, 20h" restará el segundo operando al primero, y guardará el resultado en el primer operando.

Si el primer operando es "al" que vale "77h" y el segundo operando es "20h" obtenemos un resultado de "57h", que cuando se ejecute la instrucción se guardará en "AL". Después con "mov [esi], al" va a mover el contenido de "AL" al registro "ESI", y por último el "retn" nos va a sacar del "call".

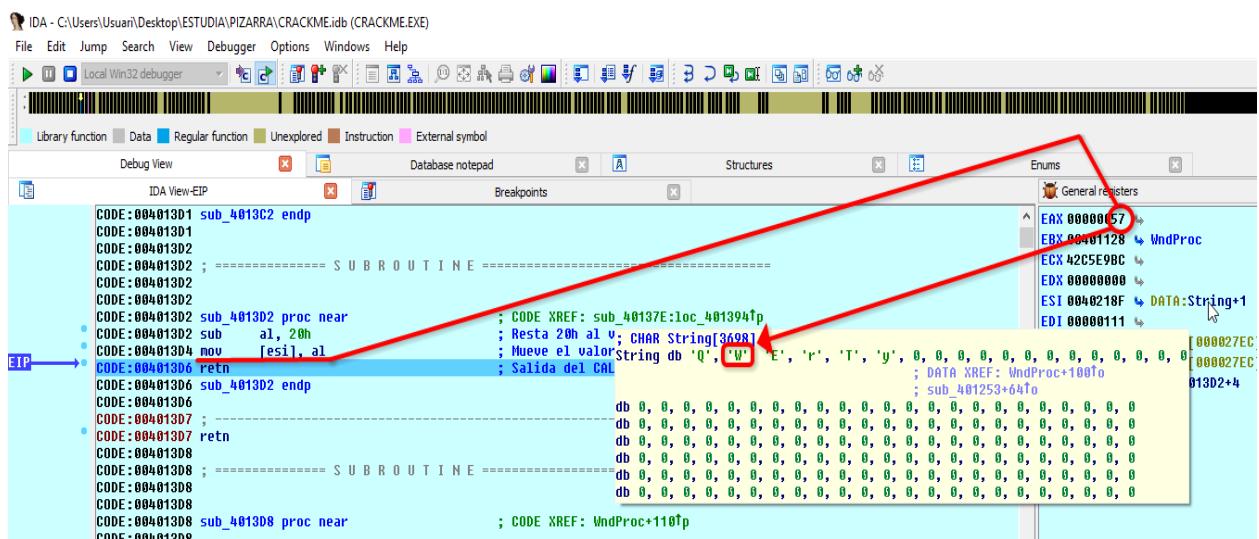
Pues entramos dentro del "call", anotamos los siguientes comentarios



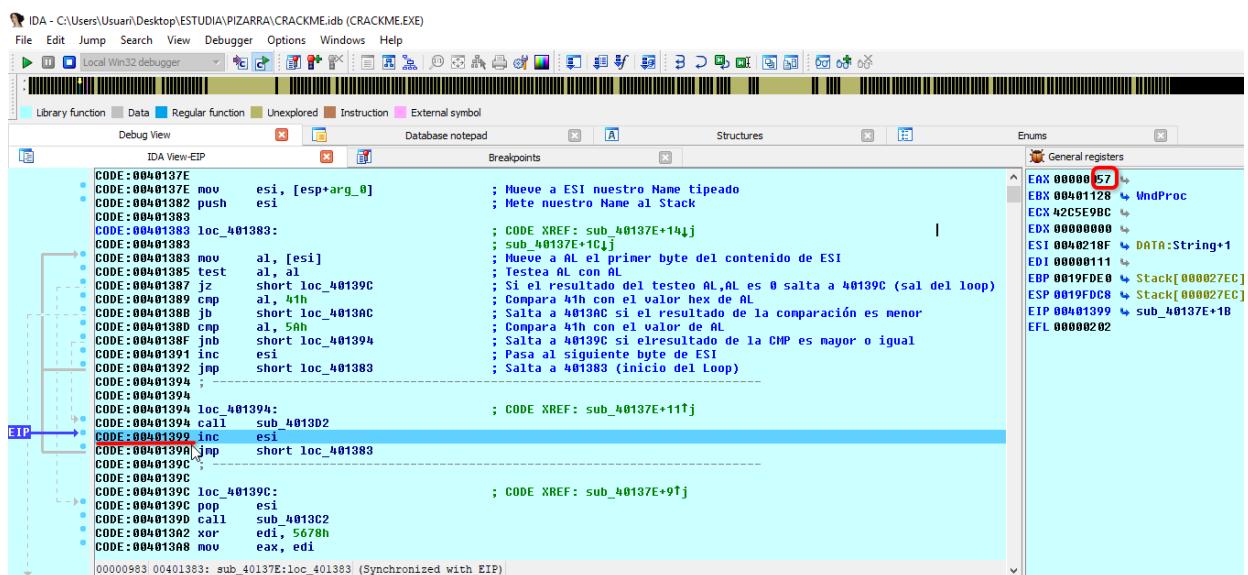
Damos el primer trazo y observamos que ahora "AL" vale "57h" que equivale a "W" (MAYÚSCULA)



En el segundo trazoe vemos como el valor de "al" pasa a "ESI"



Y por último, ejecutamos la tercera instrucción que es el "retn" con otro traceo, y efectivamente nos saca del "call" y aparecemos en CODE:00401399".



Bien, si hemos estado atentos, podemos concluir que esta pequeña "call" de dónde venimos se encarga de convertir letras minúsculas a MAYÚSCULAS

Es decir:

Si las letras MAYÚSCULAS van del 41h "A" al 5Ah "Z"

Si las letras minúsculas van del 61h "a" al 7Ah "z"

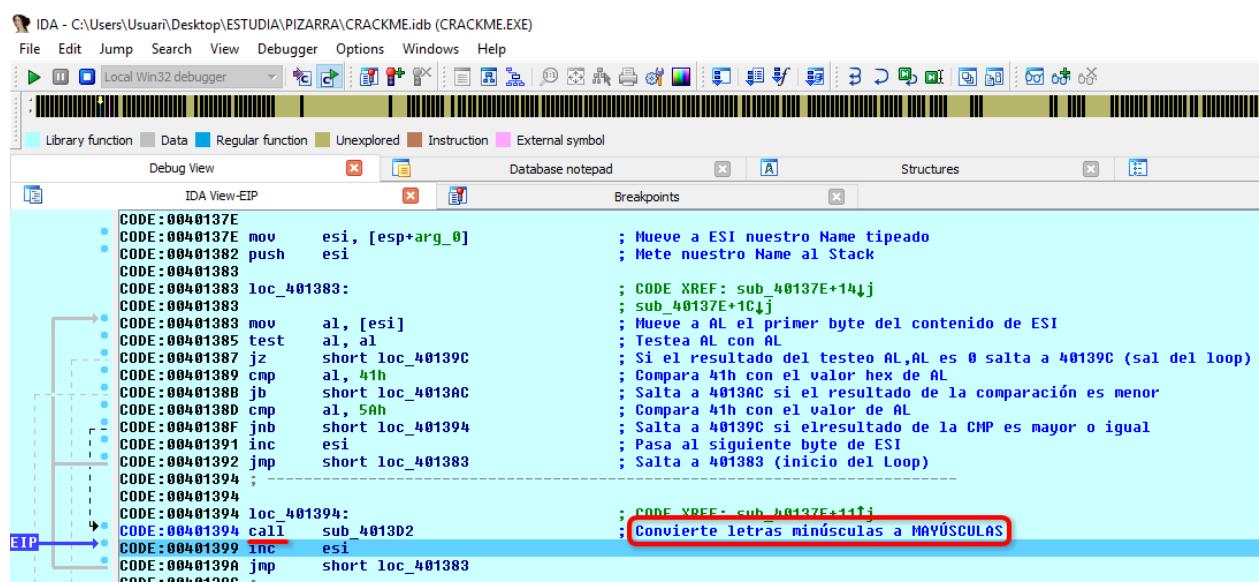
Está claro que si restamos 20h a cualquier valor hexadecimal que represente una letra minúscula, cuando pasemos por CODE:0040138D "cmp al, 5Ah" y por su siguiente instrucción de nuestro Crackme, CODE:0040138F "jnb short loc_401394", el resultado de la comparación nunca será mayor o igual a 5Ah

$$w = 77h - 20h = 57h = W$$

$$r = 72h - 20h = 52h = R$$

$$y = 79h - 20h = 59h = Y$$

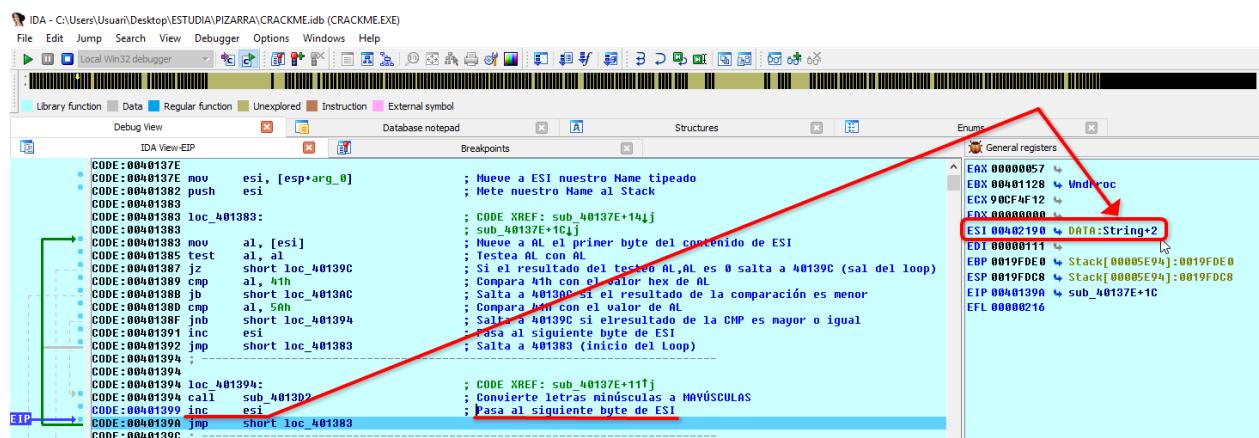
Pues, en la "call" le agregamos el siguiente comentario:



```
CODE:0040137E
CODE:0040137E mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Name tipeado
CODE:00401382 push   esi ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383:          ; CODE XREF: sub_40137E+14↓j
CODE:00401383 mov    al, [esi]      ; Mueve a AL el primer byte del contenido de ESI
CODE:00401385 test   al, al       ; Testea AL con AL
CODE:00401387 jz    short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp    al, 41h      ; Compara 41h con el valor hex de AL
CODE:0040138B jb    short loc_4013AC ; Salta a 4013AC si el resultado de la comparación es menor
CODE:0040138D cmp    al, 5Ah      ; Compara 5Ah con el valor de AL
CODE:0040138F jnb   short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
CODE:00401391 inc    esi         ; Pasa al siguiente byte de ESI
CODE:00401392 jmp   short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:00401394
CODE:00401394 loc_401394:          ; CODE XREF: sub_40137E+11↓j
CODE:00401394 call   sub_4013D2 ; :Convierte letras minúsculas a MAYÚSCULAS
EIP  CODE:00401399 inc    esi         ; :Convierte letras minúsculas a MAYÚSCULAS
CODE:0040139A jmp   short loc_401383
CODE:0040139C :
```

Recordemos que estamos parados en CODE:00401399 donde la instrucción "inc esi" va a pasar al siguiente byte de "ESI", en nuestro caso ahora cogerá 45h "E" (QWErt)

Ejecutamos la instrucción y también agregamos el comentario



```
CODE:0040137E
CODE:0040137E mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Name tipeado
CODE:00401382 push   esi ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383:          ; CODE XREF: sub_40137E+14↓j
CODE:00401383 mov    al, [esi]      ; Mueve a AL el primer byte del contenido de ESI
CODE:00401385 test   al, al       ; Testea AL con AL
CODE:00401387 jz    short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401389 cmp    al, 41h      ; Compara 41h con el valor hex de AL
CODE:0040138B jb    short loc_4013AC ; Salta a 4013AC si el resultado de la comparación es menor
CODE:0040138D cmp    al, 5Ah      ; Compara 5Ah con el valor de AL
CODE:0040138F jnb   short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
CODE:00401391 inc    esi         ; Pasa al siguiente byte de ESI
CODE:00401392 jmp   short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:00401394
CODE:00401394 loc_401394:          ; CODE XREF: sub_40137E+11↓j
CODE:00401394 call   sub_4013D2 ; :Convierte letras minúsculas a MAYÚSCULAS
CODE:00401399 inc    esi         ; :Pasa al siguiente byte de ESI
CODE:0040139A jmp   short loc_401383
CODE:0040139C :
```

Registers window showing the state of registers after execution:

General registers
EDI 00000057
EBX 00401128
ECX 90CF4F12
EDX 00000000
ESI 00002198 DATA:String+2
EDI 00000111
EBP 0019FDE0 Stack[00005E94]:0019FDE0
ESP 0019FD8C Stack[00005E94]:0019FD8C
EIP 0040139A sub_40137E+1C
EFL 00000216

La siguiente es un salto incondicional "jmp short loc_401383" que nos va a llevar directos al inicio del "Loop" donde nos encontramos, "CODE:00401383" tal y como también nos lo avanza la flecha verde de IDA.

Agregamos comentario, lo ejecutamos y aquí estamos

```

CODE:0040137E
CODE:0040137F mov    esi, [esp+arg_0]
CODE:00401380 push   esi
CODE:00401381
CODE:00401382 push   esi
CODE:00401383 loc_401383:
CODE:00401383 mov    al, [esi]
CODE:00401385 test   al, al
CODE:00401386 jz    short loc_40139C
CODE:00401389 cmp    al, 41h
CODE:0040138B jb    short loc_4013AC
CODE:0040138D cmp    al, 5Ah
CODE:0040138F jnb   short loc_401394
CODE:00401391 inc    esi
CODE:00401392 jmp    short loc_401383
CODE:00401394 ;
CODE:00401394 loc_401394:
CODE:00401394 call   sub_4013D2
CODE:00401399 inc    esi
CODE:0040139A jmp    short loc_401383
CODE:0040139C ;

```

Otro trazo más para que se ejecute el "mov al,[esi]" para asegurarnos que ahora va a trabajar con el byte 45h "E" (QWE_{Rty}) y efectivamente eso es lo que ha hecho, ahora "al" vale 45h "E"

```

CODE:0040137E
CODE:0040137F mov    esi, [esp+arg_0]
CODE:00401380 push   esi
CODE:00401381
CODE:00401382 push   esi
CODE:00401383 loc_401383:
CODE:00401383 mov    al, [esi]
CODE:00401385 test   al, al
CODE:00401386 jz    short loc_40139C
CODE:00401389 cmp    al, 41h
CODE:0040138B jb    short loc_4013AC
CODE:0040138D cmp    al, 5Ah
CODE:0040138F jnb   short loc_401394
CODE:00401391 inc    esi
CODE:00401392 imm   short loc_401382

```

Biennnn, ahora traceamos y traceamoshasta que queden convertidas las tres letras minúsculas de nuestro "Name" tipeado y llegaremos a siguiente pantalla:

```

CODE:0040137E
CODE:0040137F mov    esi, [esp+arg_0]
CODE:00401380 push   esi
CODE:00401381
CODE:00401382 push   esi
CODE:00401383 loc_401383:
CODE:00401383 mov    al, [esi]
CODE:00401385 test   al, al
CODE:00401386 jz    short loc_40139C
CODE:00401389 cmp    al, 41h
CODE:0040138B jb    short loc_4013AC
CODE:0040138D cmp    al, 5Ah
CODE:0040138F jnb   short loc_401394
CODE:00401391 inc    esi
CODE:00401392 jmp    short loc_401383
CODE:00401394 ;
CODE:00401394 loc_401394:
CODE:00401394 call   sub_4013D2
CODE:00401399 inc    esi
CODE:0040139A jmp    short loc_401383
CODE:0040139C ;

```

Al no quedar más caracteres de nuestro Name tipeado, ejecutamos el "mov al,[esi]" y observamos que ahora "al" vale "00"

```

CODE:00401382 push    esi          ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383:
CODE:00401384
CODE:00401385 mov     al, [esi]    ; Crea XREF: sub_40137E+14↓j
CODE:00401385 test    al, al      ; Hueve a AL el primer byte del contenido de ESI
CODE:00401386 cmp     al, 41h     ; Testea AL con AL
CODE:00401387 jz    short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401388 cmp     al, 41h     ; Compara 41h con el valor hex de AL
CODE:00401389 jb    short loc_40139C ; Salta a 40139C si el resultado de la comparación es menor
CODE:0040138A cmp     al, 5Ah     ; Compara 41h con el valor de AL
CODE:0040138B jnb   short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
CODE:0040138C inc     esi         ; Pasa al siguiente byte de ESI
CODE:0040138D jmp     short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:0040138E
CODE:0040138F
CODE:00401390 loc_40139A:
CODE:00401390 call    sub_4013D2
CODE:00401391 inc     esi         ; CODE XREF: sub_40137E+11↑j
CODE:00401391 jmp     short loc_401383 ; Convierte letras minúsculas a MAYÚSCULAS
CODE:00401392 inc     esi         ; Pasa al siguiente byte de ESI
CODE:00401392 jmp     short loc_401383 ; Salta a CODE:00401383 (Inicio del Loop)
CODE:00401393
CODE:00401394 loc_401394:
CODE:00401394 call    sub_4013D2
CODE:00401395 inc     esi         ; CODE XREF: sub_40137E+11↑j
CODE:00401395 jmp     short loc_401383 ; Convierte letras minúsculas a MAYÚSCULAS
CODE:00401396 inc     esi         ; Pasa al siguiente byte de ESI
CODE:00401396 jmp     short loc_401383 ; Salta a CODE:00401383 (Inicio del Loop)
CODE:00401397
CODE:00401398 loc_40139C:
CODE:00401398 sub    esi         ; CODE XREF: sub_40137E+9↑j
CODE:00401399 pop    esi         ; CODE XREF: sub_40137E+9↑j
CODE:00401399 call    sub_4013C2
CODE:004013A0 xor    edi, 508h
CODE:004013A1 mov    eax, edi
CODE:004013A2 jmp    short locret_4013C1 ; No offset aNoLuck
CODE:004013A3 loc_4013AC:
CODE:004013A3 sub    esi         ; CODE XREF: sub_40137E+D↑j
CODE:004013A4 pop    esi         ; utype
CODE:004013A4 push    30h       ; "No luck"
CODE:004013A5 push    offset aNoLuckThereMat ; "No luck there, mate!"
CODE:004013A6 push    dword ptr [ebp+8] ; hWnd
CODE:004013A7 call    MessageBoxA ; MessageBoxA
CODE:004013A8 locret_4013C1:
CODE:004013A8 reta
CODE:004013A8 sub    40137E endp

```

A continuación el resultado del testeо "test al,al" será "0", el "FLAG ZF" levanta bandera pasando de "0" a "1" con lo cual el salto condicional "jz" esta vez nos llevará a "CODE:40139C"

```

CODE:00401382 push    esi          ; Mete nuestro Name al Stack
CODE:00401383
CODE:00401383 loc_401383:
CODE:00401384
CODE:00401385 mov     al, [esi]    ; Crea XREF: sub_40137E+14↓j
CODE:00401385 test    al, al      ; Hueve a AL el primer byte del contenido de ESI
CODE:00401386 cmp     al, 41h     ; Testea AL con AL
CODE:00401387 jz    short loc_40139C ; Si el resultado del testeo AL,AL es 0 salta a 40139C (sal del loop)
CODE:00401388 cmp     al, 41h     ; Compara 41h con el valor hex de AL
CODE:00401389 jb    short loc_40139C ; Salta a 40139C si el resultado de la comparación es menor
CODE:0040138A cmp     al, 5Ah     ; Compara 41h con el valor de AL
CODE:0040138B jnb   short loc_401394 ; Salta a 40139C si el resultado de la CMP es mayor o igual
CODE:0040138C inc     esi         ; Pasa al siguiente byte de ESI
CODE:0040138D jmp     short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:0040138E
CODE:0040138F
CODE:00401390 loc_40139A:
CODE:00401390 call    sub_4013D2
CODE:00401391 inc     esi         ; CODE XREF: sub_40137E+11↑j
CODE:00401391 jmp     short loc_401383 ; Convierte letras minúsculas a MAYÚSCULAS
CODE:00401392 inc     esi         ; Pasa al siguiente byte de ESI
CODE:00401392 jmp     short loc_401383 ; Salta a CODE:00401383 (Inicio del Loop)
CODE:00401393
CODE:00401394 loc_401394:
CODE:00401394 call    sub_4013D2
CODE:00401395 inc     esi         ; CODE XREF: sub_40137E+11↑j
CODE:00401395 jmp     short loc_401383 ; Convierte letras minúsculas a MAYÚSCULAS
CODE:00401396 inc     esi         ; Pasa al siguiente byte de ESI
CODE:00401396 jmp     short loc_401383 ; Salta a CODE:00401383 (Inicio del Loop)
CODE:00401397
CODE:00401398 loc_40139C:
CODE:00401398 sub    esi         ; CODE XREF: sub_40137E+9↑j
CODE:00401399 pop    esi         ; CODE XREF: sub_40137E+9↑j
CODE:00401399 call    sub_4013C2
CODE:004013A0 xor    edi, 508h
CODE:004013A1 mov    eax, edi
CODE:004013A2 jmp    short locret_4013C1 ; No offset aNoLuck
CODE:004013A3 loc_4013AC:
CODE:004013A3 sub    esi         ; CODE XREF: sub_40137E+D↑j
CODE:004013A4 pop    esi         ; utype
CODE:004013A4 push    30h       ; "No luck"
CODE:004013A5 push    offset aNoLuckThereMat ; "No luck there, mate!"
CODE:004013A6 push    dword ptr [ebp+8] ; hWnd
CODE:004013A7 call    MessageBoxA ; MessageBoxA
CODE:004013A8 locret_4013C1:
CODE:004013A8 reta
CODE:004013A8 sub    40137E endp

```

Traceamos para ejecutar el salto condicional y aquí estamos

```

CODE:00401392 jmp     short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:00401393
CODE:00401394 loc_401394:
CODE:00401394 sub    esi         ; CODE XREF: sub_40137E+11↑j
CODE:00401394 call    sub_4013D2
CODE:00401395 inc     esi         ; Convierte letras minúsculas a MAYÚSCULAS
CODE:00401395 jmp     short loc_401383 ; Pasa al siguiente byte de ESI
CODE:00401396 inc     esi         ; Salta a CODE:00401383 (Inicio del Loop)
CODE:00401397
CODE:00401398 loc_40139C:
CODE:00401398 sub    esi         ; CODE XREF: sub_40137E+9↑j
CODE:00401399 pop    esi         ; CODE XREF: sub_40137E+9↑j
CODE:00401399 call    sub_4013C2
CODE:004013A0 xor    edi, 508h
CODE:004013A1 mov    eax, edi
CODE:004013A2 jmp    short locret_4013C1 ; No offset aNoLuck
CODE:004013A3 loc_4013AC:
CODE:004013A3 sub    esi         ; CODE XREF: sub_40137E+D↑j
CODE:004013A4 pop    esi         ; utype
CODE:004013A4 push    30h       ; "No luck"
CODE:004013A5 push    offset aNoLuckThereMat ; "No luck there, mate!"
CODE:004013A6 push    dword ptr [ebp+8] ; hWnd
CODE:004013A7 call    MessageBoxA ; MessageBoxA
CODE:004013A8 locret_4013C1:
CODE:004013A8 reta
CODE:004013A8 sub    40137E endp

```

La instrucción "pop esi", va a pasar el contenido de la primera carta del mazo del "Stack" (En nuestro caso "QWERTY"), al registro "ESI"

Para comprobar el contenido de la primera carta del "Stack", nos posicionamos sobre ella, click derecho de ratón y "Follow in Disassembly"

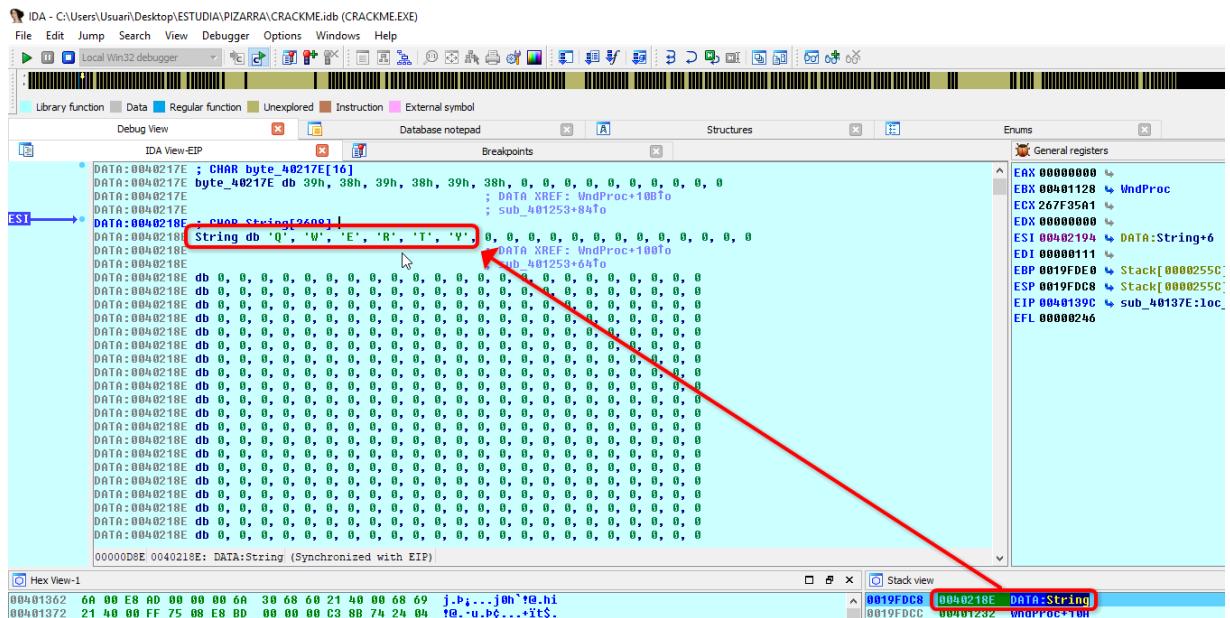
The screenshot shows the Immunity Debugger interface with several open windows:

- IDA View-EIP**: Shows assembly code for the current program. The assembly pane highlights the following code snippet:

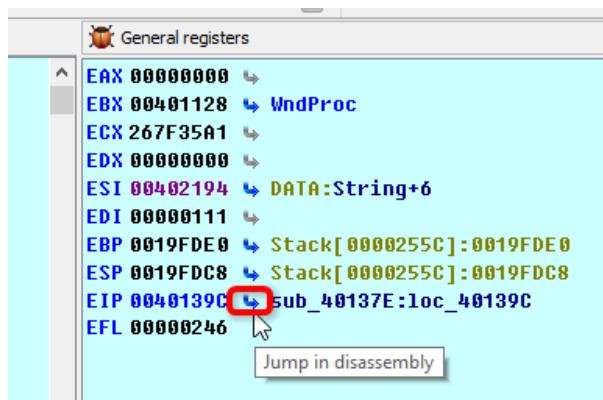

```

CODE:000401394 loc_401394: ; CODE XREF: sub_40137E+11†
CODE:000401394 call    sub_401302          ; Convierte letras minúsculas a MAYÚSCULAS
CODE:000401395 inc    esi               ; Pasa al siguiente byte de ESI
CODE:000401396 jmp    short loc_401383 ; Salta a CODE:000401383 (Inicio del Loop)
CODE:00040139C ;
CODE:00040139C loc_40139C: ; CODE XREF: sub_40137E+9†j
CODE:00040139C pop    esi               ; 
CODE:00040139D call    sub_4013C2          ; 
CODE:00040139E xor    edi, 5678h        ; 
CODE:0004013A0 mov    eax, edi          ; 
CODE:0004013A0 jmp    short locret_4013C1 ; 
CODE:0004013AC ;
CODE:0004013AC loc_4013AC: ; CODE XREF: sub_40137E+D†j
CODE:0004013AC pop    esi               ; 
CODE:0004013AD push    30h             ; uType
CODE:0004013AF push    offset aNoLuck     ; "No luck"
CODE:0004013B0 push    offset aNoLuckThereMat ; "No luck there, mate!"
CODE:0004013B0 push    dword ptr [ebp+8]   ; hWnd
CODE:0004013B0 call    MessageBoxA       ; 
CODE:0004013C1 locret_4013C1: ; CODE XREF: sub_40137E+20†j
CODE:0004013C1 retn               ; 
CODE:0004013C1 sub_40137E endp
CODE:0004013C1
CODE:0004013C2
CODE:0004013C2 ; ----- S U B R O U T I N E -----
0000095C 0040139C: sub_40137E:loc_40139C (Synchronized with EIP)
```
- Breakpoints**: Shows a list of breakpoints set in the code.
- Structures**: Shows the memory structure of the program.
- Enums**: Shows the enumeration definitions.
- Registers**: Shows the general registers (ERX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP, EFL) and their current values.
- Stack view**: Shows the stack contents.
- Memory dump**: Shows the memory dump of the current program.

Y en la ventana del desensamblado aquí lo tenemos



Para volver a la visualización anterior, ya sabemos que dándole un click Izquierdo a la flechita azul que se encuentra en el registro “EIP” lo conseguiremos



Y aquí estamos de nuevo

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures Enums

IDA View-EIP Breakpoints

EIP

```

CODE:00401394 jmp short loc_401383 ; Salta a CODE:00401383 (Inicio del Loop)
CODE:0040139C ;
CODE:0040139C loc_40139C: ; CODE XREF: sub_40137E+9tj
CODE:0040139C pop esi
CODE:0040139D call sub_4013C2
CODE:0040139D xor edi, 5678h
CODE:0040139B mov eax, edi
CODE:0040139A jmp short locret_4013C1
CODE:0040139C ;
CODE:0040139C loc_4013C1: ; CODE XREF: sub_40137E+D1j
CODE:0040139C pop esi
CODE:0040139D push 30h
CODE:0040139F push offset aNoLuck
CODE:00401384 push offset aNoLuckThereMat
CODE:00401389 push dword ptr [ebp+8]
CODE:0040138C call MessageBoxA
CODE:004013C1 locret_4013C1: ; CODE XREF: sub_40137E+2Ctj
CODE:004013C1 retn
CODE:004013C1 sub_40137E endp
CODE:004013C1
CODE:004013C2
CODE:004013C2 ; ===== SUB ROUTINE =====
CODE:004013C2
CODE:004013C2
CODE:004013C2
CODE:004013C2 sub_4013C2 proc near ; CODE XREF: sub_40137E+1Ftp
000009C 0040139C: sub_40137E:loc_40139C (Synchronized with EIP)

```

Hex View Stack view

```

00401362 6A 00 E8 AD 00 00 00 60 30 68 60 21 40 00 68 69 J.P...j0h!10.b!
00401372 21 40 00 FF 75 08 BD 00 00 C3 88 74 24 04 10..u.b...+t$.
00401382 56 88 06 84 C0 74 13 3C A1 72 1F 3C 58 73 03 46 U.e..<r.<s.F
00401392 EB EF E8 39 00 00 46 EB E7 5E E8 20 00 00 00 0'9...F0p>...
004013B2 81 F7 78 56 00 00 88 C7 EB 15 5E 6A 30 68 69 21 .xv...!A0..j0h!
004013B2 40 00 68 69 21 40 00 FF 75 08 E8 79 00 00 C3 0.hf@.u.p...+

```

0019FD8 0040218E DATA:String
0019FDCC 00402132 WndProc+18A
0019FD00 0040218E DATA:String
0019FD04 00401128 WndProc
0019FD08 00000111
0019FD0C 00000000

Ahora, agregamos el siguiente comentario a esa instrucción de código, y nos queda así:

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures Enums

IDA View-EIP Breakpoints

EIP

```

CODE:00401392 jmp short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:00401394 ;
CODE:00401394 loc_401394: ; CODE XREF: sub_40137E+11tj
CODE:00401394 call sub_4013D2 ; Convierte letras minúsculas a MAYÚSCULAS
CODE:00401394 inc esi ; Pasa al siguiente byte de ESI
CODE:00401394 jmp short loc_401383 ; Salta a CODE:00401383 (Inicio del Loop)
CODE:0040139C ;
CODE:0040139C loc_40139C: ; CODE XREF: sub_40137E+9tj
CODE:0040139C pop esi ; Pasa la primera carta del Stack a "ESI" (nuestro Name convertido a MAYÚSCULAS)
CODE:0040139D call sub_4013C2
CODE:0040139D xor edi, 5678h
CODE:0040139B mov eax, edi
CODE:0040139A jmp short locret_4013C1
CODE:0040139C ;
CODE:0040139C loc_4013AC: ; CODE XREF: sub_40137E+D1j
CODE:0040139C pop esi

```

Ejecutamos la instrucción con un trazado más y efectivamente vemos que eso es lo que ha hecho

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures Enums

IDA View-EIP Breakpoints

EIP

```

CODE:00401392 jmp short loc_401383 ; Salta a 401383 (inicio del Loop)
CODE:00401394 ;
CODE:00401394 loc_401394: ; CODE XREF: sub_40137E+11tj
CODE:00401394 call sub_4013D2 ; Convierte letras minúsculas a MAYÚSCULAS
CODE:00401394 inc esi ; Pasa al siguiente byte de ESI
CODE:00401394 jmp short loc_401383 ; Salta a CODE:00401383 (Inicio del Loop)
CODE:0040139C ;
CODE:0040139C loc_40139C: ; CODE XREF: sub_40137E+9tj
CODE:0040139C pop esi ; Pasa la primera carta del Stack a "ESI" (nuestro Name convertido a MAYÚSCULAS)
CODE:0040139D call sub_4013C2
CODE:0040139D xor edi, 5678h
CODE:0040139B mov eax, edi
CODE:0040139A jmp short locret_4013C1
CODE:0040139C ;
CODE:0040139C loc_4013AC: ; CODE XREF: sub_40137E+D1j
CODE:0040139C pop esi
CODE:0040139D push 30h ; utype
CODE:0040139F push offset aNoLuck
CODE:00401384 push offset aNoLuckThereMat
CODE:00401389 push dword ptr [ebp+8]
CODE:0040138C call MessageBoxA

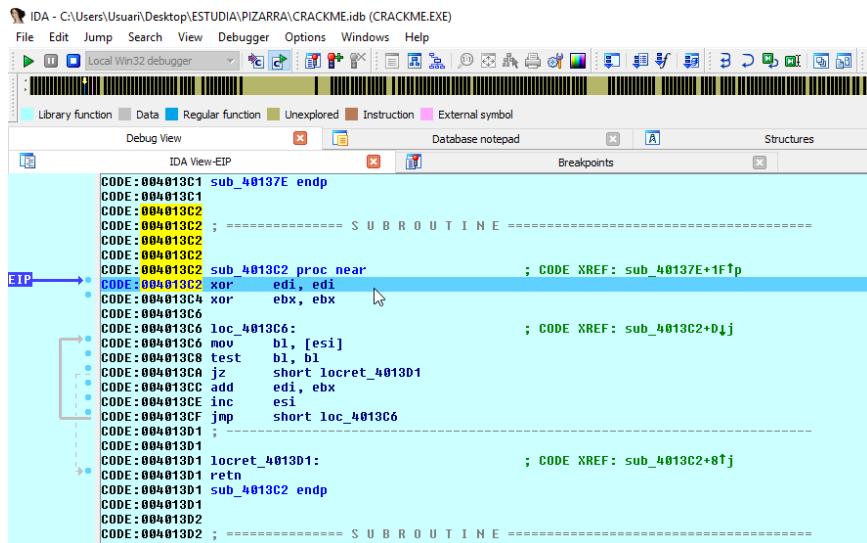
```

General registers

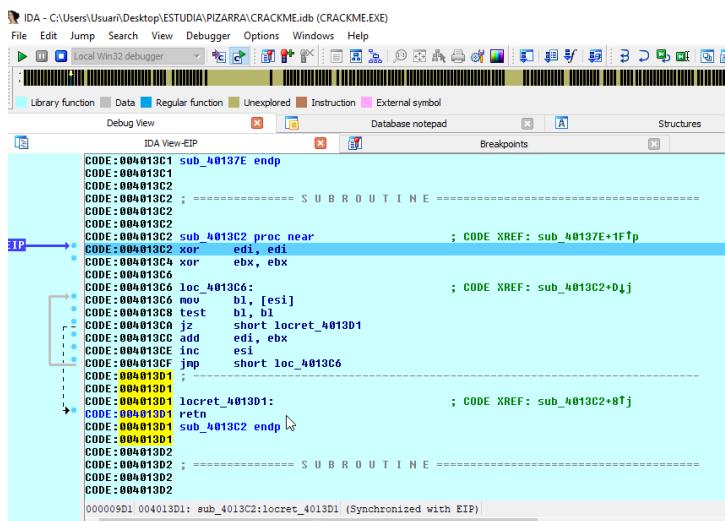
ERX 00000000	
EBX 00401128	WndProc
ECX 267F35A1	
EDX 00000000	
ESI 00402194	DATA:String+6
EDI 00000111	
EBP 0019FD00	Stack[0000255C]:0019FD00
ESP 0019FD08	Stack[0000255C]:0019FD08
EIP 0040139C	sub_40137E:loc_40139C
EFL 00000246	

000009C 0040139C: sub_40137E:loc_40139C (Synchronized with EIP)

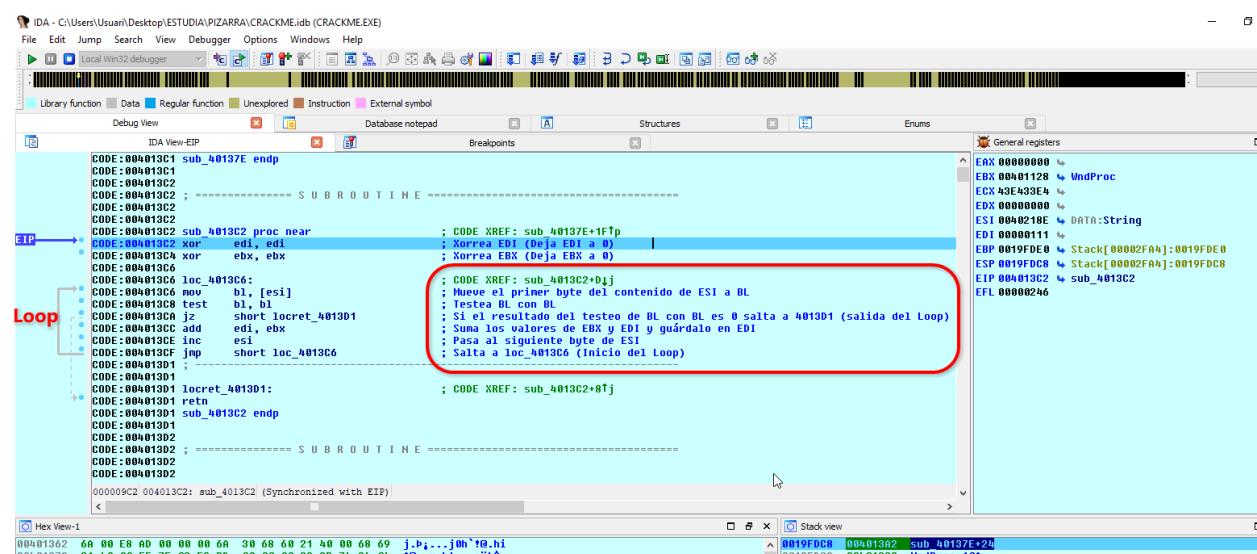
La próxima instrucción es un "call", entramos dentro y vemos que esta subrutina empieza en "CODE:004013C2"



Y termina en CODE:004013D1" donde se encuentra el "retn"



También vemos que dentro de esta subrutina hay otro pequeño “**Loop**”. Bien, pues vamos a agregar los comentarios de lo que van a hacer aquí las instrucciones del código



Traceamos para que se ejecute "xor edi,edi" y observamos como ahora el registro "EDI" vale "00000000"

```

CODE:004013C1 sub_4013E endp
CODE:004013C1
CODE:004013C2
CODE:004013C2 ; ===== SUBROUTINE =====
CODE:004013C2
CODE:004013C2
CODE:004013C2 sub_4013C2 proc near
CODE:004013C2 xor edi, edi
CODE:004013C2 xor ebx, ebx
CODE:004013C6
CODE:004013C6 loc_4013C6:
CODE:004013C6 mov bl, [esi]
CODE:004013C8 test bl, bl
CODE:004013CA jz short locret_4013D1
CODE:004013CC add edi, ebx
CODE:004013CE inc esi
CODE:004013CF jmp short loc_4013C6
CODE:004013D1 ; =====

; CODE XREF: sub_4013E+1F↑p
; Xorra EDI (Deja EDI a 0)
; Xorra EBX (Deja EBX a 0)

; CODE XREF: sub_4013C2+D↓j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

; CODE XREF: sub_4013C2+8↑j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

; CODE XREF: sub_4013C2+8↑j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

```

Otro trazo y el "xor ebx,ebx" también deja el registro "EBX" a cero

```

CODE:004013C1 sub_4013E endp
CODE:004013C1
CODE:004013C2
CODE:004013C2 ; ===== SUBROUTINE =====
CODE:004013C2
CODE:004013C2
CODE:004013C2 sub_4013C2 proc near
CODE:004013C2 xor edi, edi
CODE:004013C2 xor ebx, ebx
CODE:004013C6
CODE:004013C6 loc_4013C6:
CODE:004013C6 mov bl, [esi]
CODE:004013C8 test bl, bl
CODE:004013CA jz short locret_4013D1
CODE:004013CC add edi, ebx
CODE:004013CE inc esi
CODE:004013CF jmp short loc_4013C6
CODE:004013D1 ; =====

; CODE XREF: sub_4013E+1F↑p
; Xorra EDI (Deja EDI a 0)
; Xorra EBX (Deja EBX a 0)

; CODE XREF: sub_4013C2+D↓j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

; CODE XREF: sub_4013C2+8↑j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

; CODE XREF: sub_4013C2+8↑j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

; CODE XREF: sub_4013C2+8↑j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

```

La siguiente instrucción es "mov bl, [esi]" pues va a mover el primer byte del contenido de "ESI" y lo va a guardar en "BL"

(ESI = QwErTy 51h = Q)

La ejecutamos y eso es lo que ha hecho

```

CODE:004013C2 xor edi, edi
CODE:004013C4 xor ebx, ebx
CODE:004013C6
CODE:004013C6 loc_4013C6:
CODE:004013C6 mov bl, [esi]
CODE:004013C8 test bl, bl
CODE:004013CA jz short locret_4013D1
CODE:004013CC add edi, ebx
CODE:004013CE inc esi
CODE:004013CF jmp short loc_4013C6
CODE:004013D1 ; =====

; Xorra EDI (Deja EDI a 0)
; Xorra EBX (Deja EBX a 0)

; CODE XREF: sub_4013C2+D↓j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

; CODE XREF: sub_4013C2+8↑j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

; CODE XREF: sub_4013C2+8↑j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

; CODE XREF: sub_4013C2+8↑j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

; CODE XREF: sub_4013C2+8↑j
; Mueve el primer byte del contenido de ESI a BL
; Testea BL con BL
; Si el resultado del testeо de BL con BL es 0 salta a 4013D1 (salida del Loop)
; Suma los valores de EBX y EDI y guárdalo en EDI
; Pasa al siguiente byte de ESI
; Salta a loc_4013C6 (Inicio del Loop)

```

Ahora con "test bl, bl" va a testear "bl" = 51h = "Q", y con el "jz short locret_4013D1", con que el resultado del testeо de "bl" es 51h , el salto condicional "jz" nos dejа continuar.

Si "bl" valiera "00" el Flag ZF levantaría bandera y nos llevaría a la "locret_4013D1" donde se encuentra el "retn" que es la salida del "Loop" en el que ahora nos encontramos.

(Eso sucederá cuando ya no queden más caracteres para leer de nuestro Name tipeado)

Por si tenemos dudas, miramos en modo gráfico y también se ve muy claro que nos va a dejar continuar

```

Parpadeando
004013C2
004013C2
004013C2
004013C2 sub_4013C2 proc near
004013C2 xor edi, edi ; Xorrea EDI (Deja EDI a 0)
004013C4 xor ebx, ebx ; Xorrea EBX (Deja EBX a 0)

004013C6 loc_4013C6: ; Mueve el primer byte del contenido de ESI a BL
004013C6 mov bl, [esi]
004013C8 test bl, bl ; Testea BL con BL
004013C8 jz short locret_4013D1; Si el resultado del testeo de BL con BL es 0 salta a 4013D1 (salida del Loop)

004013CC add edi, ebx ; Suma los valores de EBX y EDI y guárdate en EDI
004013CE inc esi ; Pasa al siguiente byte de ESI
004013CF jmp short loc_4013C6 ; Salta a loc_4013C6 (Inicio del Loop)

004013D1 locret_4013D1:
004013D1 retn
004013D1 sub_4013C2 endp
004013D1

```

Registers pane:

- EAX 00000000
- EBX 00000051
- ECX 43E433E4
- EDX 00000000
- ESI 0040218E DATA:String
- EDI 00000000
- EBP 0019FDE0 Stack[00002FA4]:0019FDE0
- ESP 0019FDC8 Stack[00002FA4]:0019FDC8
- EIP 004013CA sub_4013C2+8
- EFL 00000002

Ejecutamos las instrucciones en modo gráfico y ahora nos encontramos parados en "004013CC" y con el "add edi,ebx" lo que va a hacer es adicionar (sumar) los valores de "EBX" + "EDI", y el resultado lo va a guardar en "EDI"

"EBX" = "00000051h" + "EDI" = "00000000h" pues "EDI" = "51h"

Traceamos y comprobamos que ahora "EDI" vale "00000051"

```

004013C2
004013C2
004013C2
004013C2 sub_4013C2 proc near
004013C2 xor edi, edi ; Xorrea EDI (Deja EDI a 0)
004013C4 xor ebx, ebx ; Xorrea EBX (Deja EBX a 0)

004013C6 loc_4013C6: ; Mueve el primer byte del contenido de ESI a BL
004013C6 mov bl, [esi]
004013C8 test bl, bl ; Testea BL con BL
004013C8 jz short locret_4013D1; Si el resultado del testeo de BL con BL es 0 salta a 4013D1 (salida del Loop)

004013CC add edi, ebx ; Suma los valores de EBX y EDI y guárdate en EDI
004013CE inc esi ; Pasa al siguiente byte de ESI
004013CF jmp short loc_4013C6 ; Salta a loc_4013C6 (Inicio del Loop)

004013D1 locret_4013D1:
004013D1 retn
004013D1 sub_4013C2 endp
004013D1

```

Registers pane:

- EAX 00000000
- EBX 00000051
- ECX 43E433E4
- EDX 00000000
- ESI 0040218E DATA:String
- EDI 00000051
- EBP 0019FDE0 Stack[00002FA4]
- ESP 0019FDC8 Stack[00002FA4]
- EIP 004013CE sub_4013C2+C
- EFL 00000002

Hex View pane:

00401362 6A 00 E8 AD 00 00 00 6A 30 68 60 21 40 00 68 69 j.b...j0h`@.hi	0019FDC8 004013A2 sub_4013E+24
00401372 21 40 00 FF 75 08 E8 BD 00 00 00 C3 88 74 04 @.u.b...+it\$.	0019FDC8 00401232 VndProc+0@
00401382 56 8A 06 84 C0 74 13 3C 41 72 1F 3C 5A 73 03 46 @.3t.<Ar.<Zs.F	0019FDD0 0040121E DATA:String

La próxima instrucción es "inc esi", por lo que va a incrementar en "1h" el valor del registro "ESI"

Si ahora el valor de "ESI" es "0040218E", le sumamos "1h" y se convertirá en "0040218F". Lo que realmente hace es que si antes "ESI" tenía el valor "QWERTY", una vez ejecutada, al sumarle "1h" pasa al siguiente byte, por lo que si ahora trabajábamos con 51h "Q" pasaremos a trabajar con 57h "W" (QWERTY)

Esto IDA nos lo señala en el registro "ESI" como "DATA:String+1"

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures Enums

IDA View-EIP Breakpoints

General registers

EAX 00000000
EBX 00000051
ECX 43E433E4
EDX 00000000
ESI 0040218F DATA:String+1
EDI 00000051
EBP 0019FDE0 Stack[00002FA4]:
ESP 0019FD08 Stack[00002FA4]:
EIP 004013CF sub_4013C2+B
EFL 00000002

004013C2
004013C2
004013C2
004013C2 sub_4013C2 proc near
004013C2 xor edi, edi ; Xorrea EDI (Deja EDI a 0)
004013C4 xor ebx, ebx ; Xorrea EBX (Deja EBX a 0)

004013C6 loc_4013C6:
004013C6 mov bl, [esi] ; Mueve el primer byte del contenido de ESI a BL
004013C8 test bl, bl ; Testea BL con BL
004013CA jz short locret_4013D1 ; Si el resultado del testeo de BL con BL es 0 salta a 4013D1 (salida del Loop)

004013CC add edi, ebx ; Suma los valores de EBX y EDI y guárdalo en EDI
004013CE inc esi ; Pasa al siguiente byte de ESI
004013CF jmp short loc_4013C6 ; Salta a loc_4013C6 (Inicio del Loop)

004013D1 locret_4013D1:
004013D1 retn
004013D1 sub_4013C2 endp
004013D1

100.00% (-46,-18) (1097,421) 000009CF 004013CF: sub_4013C2+B (Synchronized with EIP)

Ahora este salto incondicional "jmp short loc_4013C6" nos llevará directos a esa dirección que es el inicio del "Loop"

Lo ejecutamos y aparecemos aquí

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures Enums

IDA View-EIP Breakpoints

General registers

EAX 00000000
EBX 00000051
ECX 43E433E4
EDX 00000000
ESI 0040218F DATA:String+1
EDI 00000051
EBP 0019FDE0 Stack[00002FA4]
ESP 0019FD08 Stack[00002FA4]
EIP 004013C6 sub_4013C2:loc
EFL 00000002

004013C2
004013C2
004013C2
004013C2 sub_4013C2 proc near
004013C2 xor edi, edi ; Xorrea EDI (Deja EDI a 0)
004013C4 xor ebx, ebx ; Xorrea EBX (Deja EBX a 0)

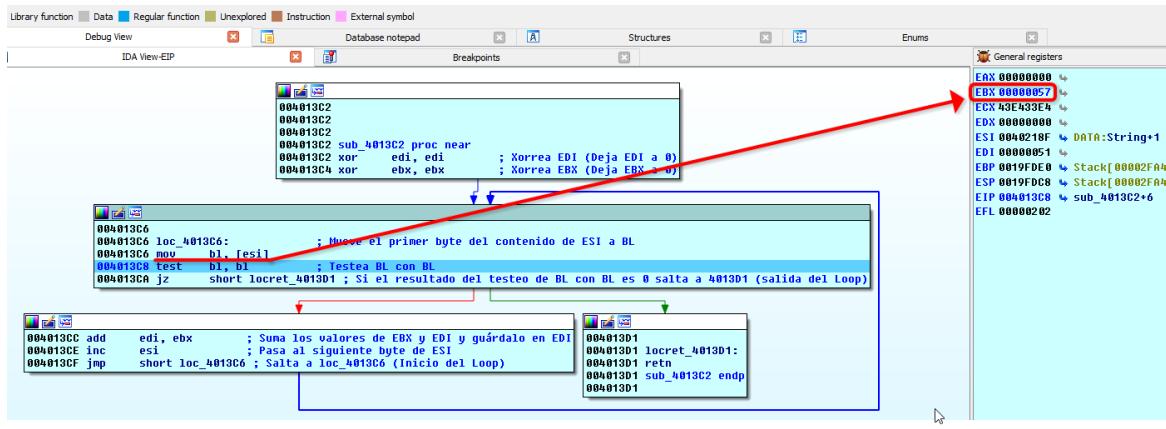
004013C6 loc_4013C6:
004013C6 mov bl, [esi] ; Mueve el primer byte del contenido de ESI a BL
004013C8 test bl, bl ; Testea BL con BL
004013CA jz short locret_4013D1 ; Si el resultado del testeo de BL con BL es 0 salta a 4013D1 (salida del Loop)

004013CC add edi, ebx ; Suma los valores de EBX y EDI y guárdalo en EDI
004013CE inc esi ; Pasa al siguiente byte de ESI
004013CF jmp short loc_4013C6 ; Salta a loc_4013C6 (Inicio del Loop)

004013D1 locret_4013D1:
004013D1 retn
004013D1 sub_4013C2 endp
004013D1

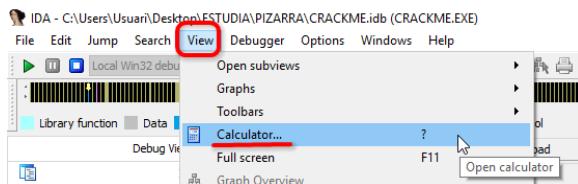
100.00% (-46,-18) (348,214) 000009C6 004013C6: sub_4013C2:loc_4013C6 (Synchronized with EIP)

Volvemos a ejecutar "mov bl, [esi]" donde moverá el byte 57h "W" a "BL"

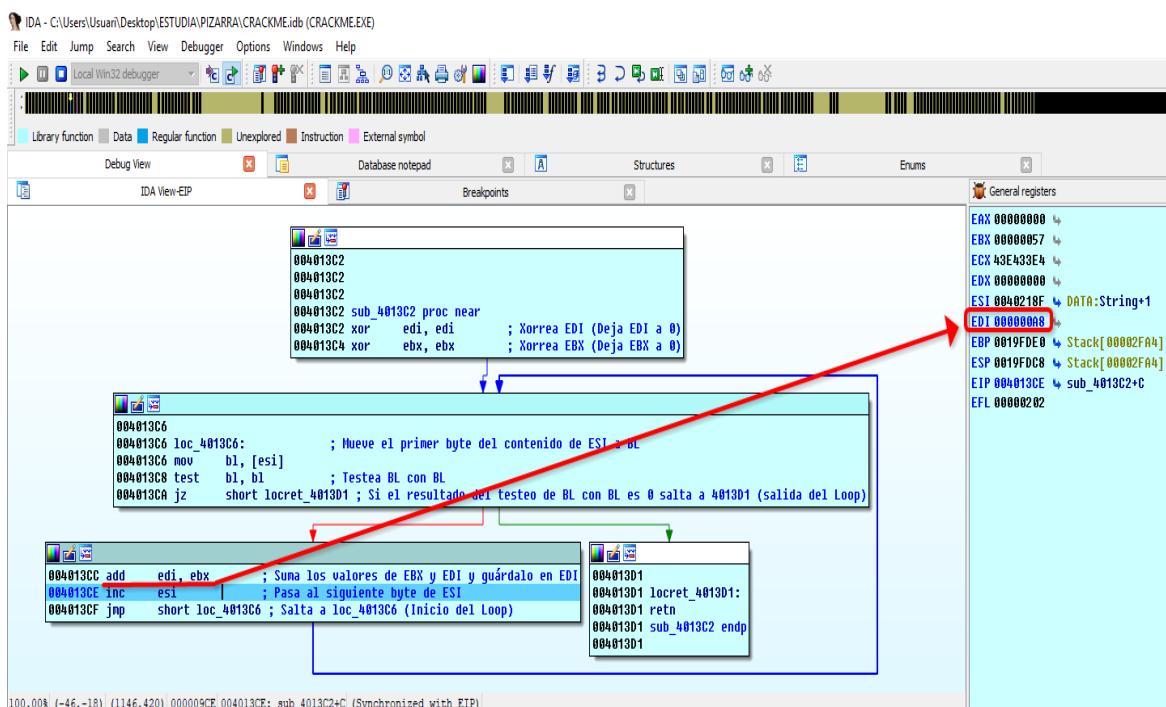
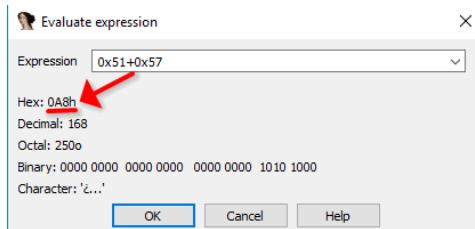


Pasamos las instrucciones traceando "test bl,bl" , "jz short locret_4013D1" y "add edi, ebx" (que ya sabemos lo que hacen), y vemos como ahora "EDI" vale "A8" que es la suma de 51h "Q" + 57h "W"

Nos vamos a la Calculadora que trae incorporada IDA



Rellenamos con "0x51+0x57" y ...verificamos que la suma es correcta = "A8h"

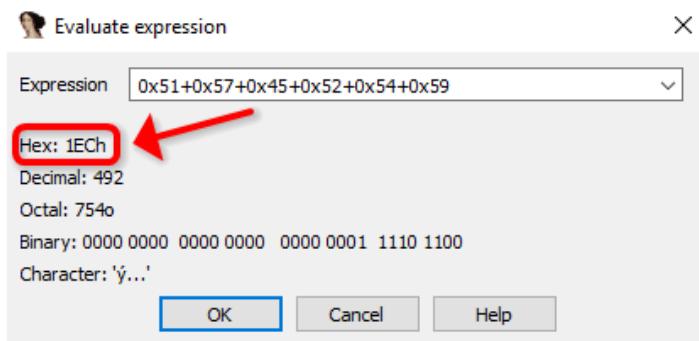


Y nuevamente se va repetir el mismo proceso con los demás caracteres de nuestro Name tipeado.

Resumiendo....., de lo que se encarga este “**Loop**” es sumar los valores hexadecimales de nuestro Name

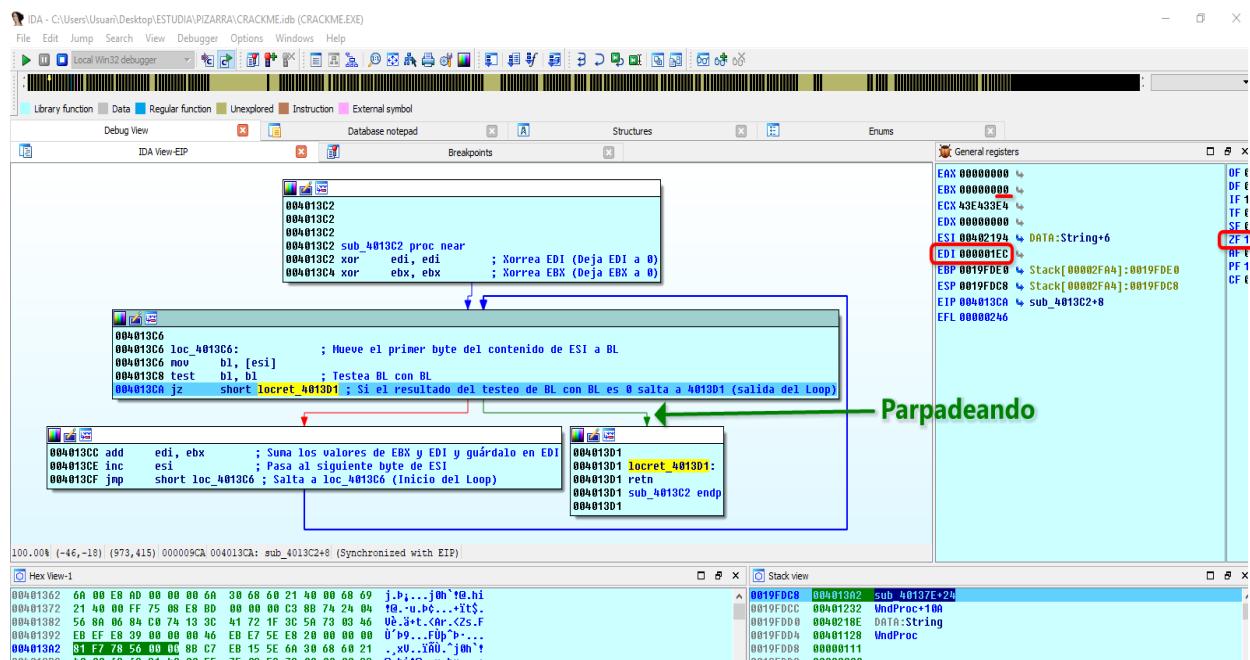
$$Q \quad W \quad E \quad R \quad T \quad Y$$

$$51h + 57h + 45h + 52h + 54h + 59h = 1ECh$$



Traceamos y traceamos y observamos como va trabajando el "Loop" hasta que no quede ningún carácter de nuestro Name y aparecemos en "004013D1", donde ahora comprobamos que efectivamente "EDI" vale "1ECh".

También observamos que el valor de "BL" es "0" (ya que las instrucciones han terminado de leer los caracteres de nuestro Name), se activa el FLAG ZF levantando bandera "1", y ésta vez el salto condicional "jz short_locret_4013D1" nos manda directos al "retn" de la address "004013D1", donde saldremos de las entrañas de la "call" en la que nos encontramos.



Volvamos al modo vista desensamblador y parecemos aquí, donde vemos exactamente lo mismo que estábamos viendo en el modo gráfico.

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

```

File Edit Jump Search View Debugger Options Windows Help
Local Win32 debugger
Library function Data Regular function Unexplored Instruction External symbol
Debug View Database notepad Structures Enums
IDA View-EIP Breakpoints General registers
CODE:004013C6 ; CODE XREF: sub_4013C2+D↓j
CODE:004013C6 loc_4013C6: ; Mueve el primer byte del contenido de ESI a BL
CODE:004013C6 mov bl, [esi]
CODE:004013C8 test bl, bl ; Testea BL con BL
CODE:004013CA jz short locret_4013D1 ; Si el resultado del testeo de BL con BL es 0 salta a 4013D1 (salida del Loop)
CODE:004013CC add edi, ebx ; Suma los valores de EBX y EDI y guárdalo en EDI
CODE:004013CE inc esi ; Pasa al siguiente byte de ESI
CODE:004013CF jmp short loc_4013C6 ; Salta a loc_4013C6 (Inicio del Loop)
CODE:004013D1 ; -----
CODE:004013D1 locret_4013D1: ; CODE XREF: sub_4013C2+8↑j
CODE:004013D1 retn
CODE:004013D1 sub_4013C2 endp
CODE:004013D1
CODE:004013D2 ; ===== SUB ROUTINE =====
CODE:004013D2
CODE:004013D2 sub_4013D2 proc near ; CODE XREF: sub_40137E:loc_401394↑p
CODE:004013D2 sub_4013D2 al, 20h ; Resta 20h al valor de AL y el resultado lo guarda en AL
CODE:004013D4 mov [esi], al ; Mueve el valor de AL a ESI
CODE:004013D6 retn ; Salida del CALL
CODE:004013D6 sub_4013D2 endp
CODE:004013D6
CODE:004013D7 ; -----
CODE:004013D7 retn
CODE:004013D8
000009CA 004013CA: sub_4013C2+8 (Synchronized with EIP)

```

Un trazo más y estamos en el "retn"

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

```

File Edit Jump Search View Debugger Options Windows Help
Local Win32 debugger
Library function Data Regular function Unexplored Instruction External symbol
Debug View Database notepad Structures Enums
IDA View-EIP Breakpoints General registers
CODE:004013C6
CODE:004013C6 loc_4013C6: ; CODE XREF: sub_4013C2+D↓j
CODE:004013C6 mov bl, [esi] ; Mueve el primer byte del contenido de ESI a BL
CODE:004013C8 test bl, bl ; Testea BL con BL
CODE:004013CA jz short locret_4013D1 ; Si el resultado del testeo de BL con BL es 0 salta a 4013D1 (salida del Loop)
CODE:004013CC add edi, ebx ; Suma los valores de EBX y EDI y guárdalo en EDI
CODE:004013CE inc esi ; Pasa al siguiente byte de ESI
CODE:004013CF jmp short loc_4013C6 ; Salta a loc_4013C6 (Inicio del Loop)
CODE:004013D1 ; -----
CODE:004013D1 locret_4013D1: ; CODE XREF: sub_4013C2+8↑j
CODE:004013D1 retn | ; CODE XREF: sub_4013C2+8↑j
CODE:004013D1 sub_4013C2 endp
CODE:004013D1
CODE:004013D2

```

Y otro más, y ya estamos fuera de la "call" como habíamos pronosticado

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

```

File Edit Jump Search View Debugger Options Windows Help
Local Win32 debugger
Library function Data Regular function Unexplored Instruction External symbol
Debug View Database notepad Structures Enums
IDA View-EIP Breakpoints General registers
CODE:0040139C ; CODE XREF: sub_40137E+9↑j
CODE:0040139C loc_40139C: ; Pasa la primera carta del Stack a "ESI" (nuestro Name convertido a MAYÚSCULAS)
CODE:0040139C pop esi
CODE:0040139C call sub_4013C2
CODE:004013A0 xor edi, 5678h
CODE:004013A0 mov eax, edi
CODE:004013A0 jmp short locret_4013C1
CODE:004013A0 ; -----
CODE:004013A0 loc_4013A0: ; CODE XREF: sub_40137E+D↑j
CODE:004013A0 pop esi
CODE:004013A0 push 30h ; uType
CODE:004013A0F push offset aNoLuck ; "No luck!"
CODE:004013B4 push offset aNoLuckThereHat ; "No luck there, mate!"
CODE:004013B9 push dword ptr [ebp+8] ; bMd
CODE:004013B9 call MessageBoxA
CODE:004013C1
CODE:004013C1 locret_4013C1: ; CODE XREF: sub_40137E+2C↑j
CODE:004013C1 retn
CODE:004013C1 sub_40137E endp
CODE:004013C1
CODE:004013C2

```

Por lo tanto ya podemos agregar el siguiente comentario a esa "call"

The screenshot shows the assembly view in IDA Pro. The instruction at address 0040139C is XOR EDI, 5678h. The assembly pane shows the instruction and its comments. The registers pane on the right shows the state of registers like EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP, and EFL.

La siguiente instrucción es un "xor edi, 5678h", está claro que va a hacer un "xor 5678h" con el valor de "EDI" y el resultado lo va a guardar en "EDI"

Hoy me encuentro con ganas de escribir, y vamos a hacer las operaciones a mano utilizando la siguiente tabla de conversión:

HEXADECIMAL	0	1	2	3	4	5	6	7
BINARIO	0000	0001	0010	0011	0100	0101	0110	0111

HEXADECIMAL	8	9	A	B	C	D	E	F
BINARIO	1000	1001	1010	1011	1100	1101	1110	1111

Si el valor de "EDI" en nuestro caso es "0000001EC" lo pasamos a binario con la tabla

0000 0000 0000 0000 0000 0001 1110 1100

Quitamos todos los ceros de la izquierda y nos queda así

111101100

Hacemos lo mismo con el valor "5678"

0101 0110 0111 1000

101011001111000

Ahora debemos saber que el operador "XOR" es una instrucción lógica, y tiene como salida un 1 (enabled) siempre que las entradas sean diferentes, y un 0 (disabled) siempre que sean iguales, por lo tanto se basa en la siguiente tabla:

$$\begin{aligned} 0 \text{ XOR } 0 &= 0 \\ 0 \text{ XOR } 1 &= 1 \\ 1 \text{ XOR } 0 &= 1 \\ 1 \text{ XOR } 1 &= 0 \end{aligned}$$

Si lo aplicamos a nuestros datos:

Nota: (las entradas vacías cuentan como 0)

$$\begin{array}{r} 1\ 01\ 01\ 10\ 01\ 11\ 10\ 00 \\ 1\ 11\ 10\ 11\ 00 \\ \hline \end{array}$$

Obtenemos el siguiente resultado

$$1\ 01\ 01\ 11\ 10\ 01\ 01\ 00$$

Y para convertir este binario a hexadecimal volvemos a utilizar la misma tabla, pero antes debemos ordenarlo en "Nibble" (grupos de 4 bits binarios) empezando de derecha a izquierda, y agregando el valor vacío con un "0" y nos queda así:

0101 0111 1001 0100

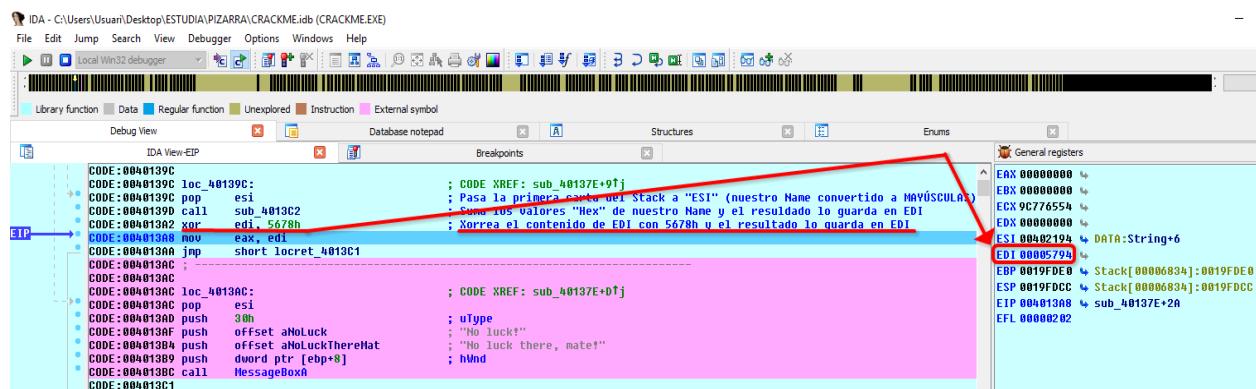
HEXADECIMAL	0	1	2	3	4	5	6	7
BINARIO	0000	0001	0010	0011	0100	0101	0110	0111

HEXADECIMAL	8	9	A	B	C	D	E	F
BINARIO	1000	1001	1010	1011	1100	1101	1110	1111

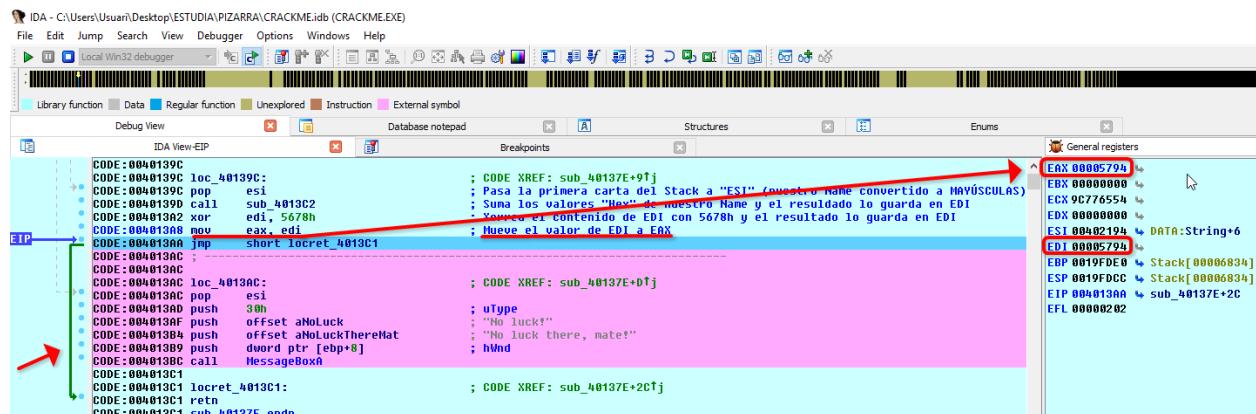
Con lo cual

0101 0111 1001 0100
↓ ↓ ↓ ↓
Obtenemos 5 7 9 4

Traceamos y verificamos que nuestro resultado obtenido manualmente es correcto, "EDI 00005794" y agregamos el correspondiente comentario



La siguiente instrucción es un "mov eax, edi", pues va a mover todo el contenido de "EDI" a "EAX", traceamos y agregamos comentario



En la imagen superior también vemos como la siguiente instrucción "jmp short locret_4013C1", el salto incondicional nos va a llevar directos a la salida del "call" CODE:004013C1 donde se encuentra un "retn" tal y como nos indica IDA con la flecha verde. También la ejecutamos, agregamos comentario y aquí estamos

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures Enums

IDA View-EIP Breakpoints

```

CODE:00401394 ; CODE XREF: sub_40137E+11†j
CODE:00401394 loc_401394: ; Convierte letras minúsculas a MAYÚSCULAS
CODE:00401394 call sub_4013D2 ; Pasa al siguiente byte de ESI
CODE:00401394 inc esi ; Pasa la primera carta del Stack a "ESI" (nuestro Name convertido a MAYÚSCULAS)
CODE:00401394 jmp short loc_401383 ; Salta a CODE:00401383 (Inicio del Loop)
CODE:0040139C ; -----
CODE:0040139C loc_40139C: ; CODE XREF: sub_40137E+9†j
CODE:0040139C pop esi ; Pasa la primera carta del Stack a "ESI" (nuestro Name convertido a MAYÚSCULAS)
CODE:0040139D call sub_4013C2 ; Suma los valores "Hex" de nuestro Name y el resultado lo guarda en EDI
CODE:004013A2 xor edi, 5678h ; Xorrea el contenido de EDI con 5678h y el resultado lo guarda en EDI
CODE:00401388 mov eax, edi ; Mueve el valor de EDI a EAX
CODE:00401388 jmp short locret_4013C1 ; Salta a 4013C1

CODE:004013AC ; CODE XREF: sub_40137E+D†j
CODE:004013AC loc_4013AC: ; CODE XREF: sub_40137E+D†j
CODE:004013AC pop esi ; uType
CODE:004013AD push 30h ; "No luck?""
CODE:004013AF push offset aNoLuck ; "No luck there, mate!""
CODE:004013B4 push offset aNoLuckThereMat ; hWnd
CODE:004013B9 push dword ptr [ebp+8]
CODE:004013BC call MessageBoxA ; -----
CODE:004013C1 locret_4013C1: ; CODE XREF: sub_40137E+20†j
CODE:004013C1 retn ; -----
CODE:004013C1 sub_40137E endp ; -----
CODE:004013C1

000009C1 004013C1: sub_40137E:locret_4013C1 (Synchronized with EIP)

```

Ejecutamos el "retn" con otro trazo más y efectivamente nos encontramos en "CODE:00401232" justo en la salida de las entrañas de nuestro "PRIMER CALL" al que le habíamos colocado el primer "BP"

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures Enums

IDA View-EIP Breakpoints

EIP

```

CODE:00401223 cmp eax, 0
CODE:00401226 jz short loc_4011E6
CODE:00401228 push offset String
CODE:0040122D call sub_40137E
CODE:0040122F push eax
CODE:00401230 push offset byte 40217E
CODE:00401233 push offset loc_4013C0
CODE:0040123D add esp, 4
CODE:00401240 pop eax
CODE:00401241 cmp eax, ebx
CODE:00401243 jz short loc_40124C
CODE:00401245 call sub_401362
CODE:0040124A jmp short loc_4011E6

```

General registers
EAX 00005794
EBX 00000000
ECX 9C776554
EDX 00000000
ESI 00402194 DATA:String+6
EDI 00005794
EBP 0019FDE0 Stack[00006834]:0019
ESP 0019FD00 Stack[00006834]:0019
EIP 00401232 UndProc+100
EFL 00000202

Este "PRIMER CALL" ha trabajado únicamente con nuestro Name tipeado.

Continuamos..... ahora tenemos un "push eax" con lo cual va a meter al "Stack" el valor de "EAX", que en este momento es "00005794"

Traceamos, y efectivamente aquí lo tenemos y agregamos comentario

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Debug View Database notepad Structures Enums

IDA View-EIP Breakpoints

EIP

```

CODE:00401223 cmp eax, 0
CODE:00401226 jz short loc_4011E6
CODE:00401228 push offset String
CODE:0040122D call sub_40137E
CODE:0040122F push eax ; Mete al Stack el contenido de EAX
CODE:00401230 push offset byte 40217E
CODE:00401233 push offset loc_4013C0
CODE:0040123D add esp, 4
CODE:00401240 pop eax
CODE:00401241 cmp eax, ebx
CODE:00401243 jz short loc_40124C
CODE:00401245 call sub_401362
CODE:0040124A jmp short loc_4011E6
CODE:0040124C
CODE:0040124D
CODE:0040124E loc_40124C: ; CODE XREF: UndProc+11B†j
CODE:0040124E call sub_401340
CODE:00401251 jmp short loc_4011E6
CODE:00401251 UndProc_endp ; sp-analysis failed
CODE:00401253
CODE:00401253 ; ===== SUB ROUTINE =====
CODE:00401253 ; Attributes: bp-based frame
CODE:00401253
CODE:00401253 ; INT_PTR __stdcall sub_401253(HWND, UINT, WPARAM, LPARAM)
CODE:00401253 sub_401253 proc near ; DATA XREF: UndProc+E3†o
CODE:00401252 00000832 00401232: WndProc+10A (Synchronized with EIP)

```

General registers
EAX 00005794
EBX 00000000
ECX 9C776554
EDX 00000000
ESI 00402194 DATA:String+6
EDI 00005794
EBP 0019FDE0 Stack[00006834]:0019
ESP 0019FDCC Stack[00006834]:0019
EIP 00401233 UndProc+100
EFL 00000202

Hex View-1 Stack view

```

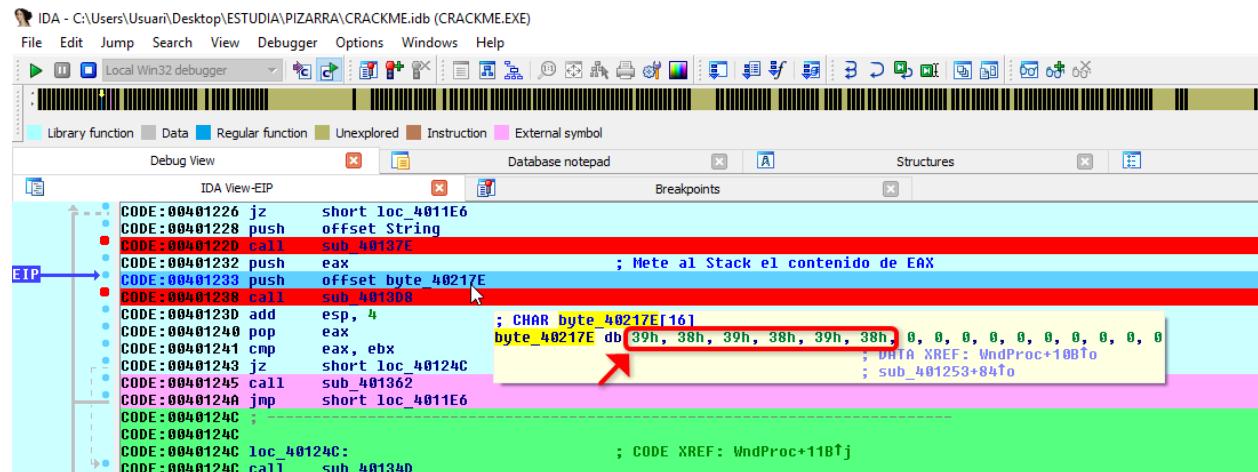
00401382 21 40 00 FF 75 08 E8 BD 00 00 00 C3 88 74 24 04 10..u.Pc...+Yt$.
00401382 56 8A 06 84 C0 74 13 3C A1 72 F1 3C 5A 73 03 46 06..8t.<Ar...<2s.F
00401382 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

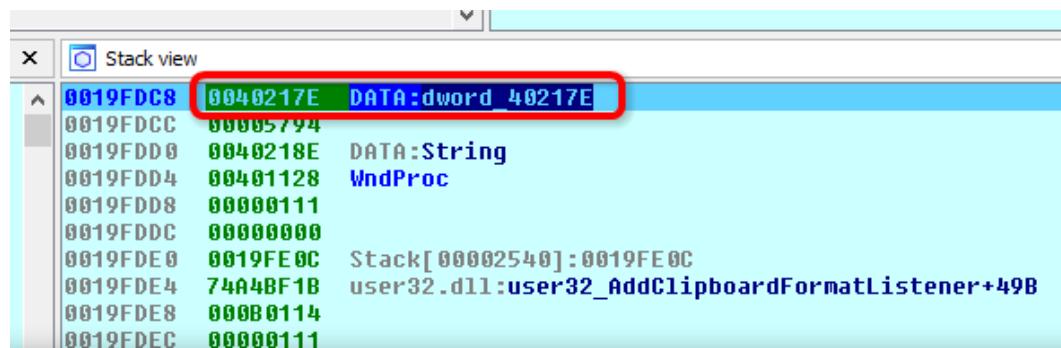
Stack view
0019FD00 00005794 DATA:String
0019FDE0 00006834

La siguiente instrucción del código es un "push offset byte_40217E" o sea que ahora va a meter al "Stack" el contenido de "0040217E".

Nos posicionamos con el cursor sobre "offset byte_40217E" y vemos que contiene nuestro serial trucho "39h 38h 39h 38h 39h 38h" (este también llevo tapeándolo desde muchoooos años je.je.je "989898"



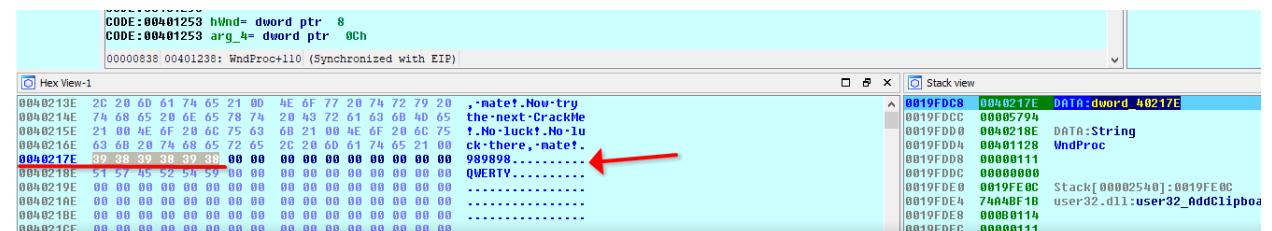
Traceamos para ejecutar la instrucción y aquí lo tenemos guardado en la primera carta del mazo del "Stack"



También podemos ver su contenido posicionándonos en la ventana "Stack" con el cursor sobre "dword_40217E" click derecho de ratón y "Follow in hex dump"



Y aquí está



Un trazo más y llegamos a la "SEGUNDA CALL" CODE:00401238, donde también pusimos un punto de parada "Breakpoint" "BP"

```

IDA - C:\Users\Usuari\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)
File Edit Jump Search View Debugger Options Windows Help
Local Win32 debugger
IDA View-EIP Database notepad Breakpoints Structures
EIP → CODE:00401238 call sub_4013D8 ; CODE XREF: WndProc+110↑p
CODE:00401239 add esp, 4
CODE:00401240 pop eax
CODE:00401241 cmp eax, ebx sub_4013D8=[sub_4013D8]
CODE:00401243 jz short loc_4013D8 ; ====== S U B R O U T I N E ======
CODE:00401245 call sub_401362
CODE:00401246 jmp short loc_4013D8 ; CODE XREF: sub_4013D8+1B↓j
CODE:0040124C ; sub_4013D8 proc near ; CODE XREF: WndProc+110↑p
CODE:0040124C loc_40124C:
CODE:0040124C call sub_40134D arg_0= dword ptr 4
CODE:00401251 jmp short loc_4013D8 ; CODE XREF: sub_4013D8+1B↓j
CODE:00401251 WndProc endp ; sp-ana
CODE:00401251 xor ebx, ebx
CODE:00401253 mov esi, [esp+arg_0]
CODE:00401253 ; ====== S U B R O U T I N E ======
CODE:00401253 loc_4013E2:
CODE:00401253 mov al, 0Ah ; CODE XREF: sub_4013D8+1B↓j
CODE:00401253 mov bl, [esi]
CODE:00401253 test bl, bl
CODE:00401253 sub_401253 proc near jz short loc_4013F5
CODE:00401253 xor ebx, ebx
CODE:00401253 hWnd= dword ptr 8 sub bl, 30h
CODE:00401253 imul edi, eax
CODE:00401253 arg_4= dword ptr 0Ch add edi, ebx
CODE:00401253 arg_8= dword ptr 10h inc esi
CODE:00401253 xor ebx, ebx
CODE:00401253 enter 0, 0 jmp short loc_4013E2
CODE:00401257 push ebx ; CODE XREF: sub_4013D8+10↓j
CODE:00401257 push ebx
00000838:00401238: WndProc+110 (Synchronous) loc_4013F5:
CODE:00401257 xor edi, 1234h ; CODE XREF: sub_4013D8+10↓j
CODE:00401257 mov ebx, edi
CODE:00401257 retn
Hex View-1
00402143 65 21 0D 4E 6F 77 20 74 72 79 20 74 sub_4013D8 endp
00402153 65 78 74 20 43 72 61 63 6B 4D 65 21
00402163 6C 75 63 6B 21 0B AF 6F 20 6C 75 63

```

Entramos en la "call", y observamos que empieza en "CODE:004013D8", finaliza en "CODE:004013FD" y en medio tenemos otro "loop" que también estudiaremos con detalle los cálculos que va a hacer. Esta "call" va a trabajar únicamente con la parte del "Serial".

```

IDA - C:\Users\Usuari\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)
File Edit Jump Search View Debugger Options Windows Help
Local Win32 debugger
IDA View-EIP Database notepad Breakpoints Structures
EIP → CODE:004013D8 sub_4013D8 proc near ; CODE XREF: WndProc+110↑p
CODE:004013D8 arg_0= dword ptr 4
CODE:004013D8
CODE:004013D8
CODE:004013D8 xor eax, eax Inicio de la call
CODE:004013D8 xor edi, edi
CODE:004013D8 xor ebx, ebx
CODE:004013D8 mov esi, [esp+arg_0]
CODE:004013E2 loc_4013E2: ; CODE XREF: sub_4013D8+1B↓j
CODE:004013E2 mov al, 0Ah
CODE:004013E2 mov bl, [esi]
CODE:004013E2 test bl, bl
CODE:004013E8 jz short loc_4013F5 Loop
CODE:004013EA sub bl, 30h
CODE:004013ED imul edi, eax
CODE:004013F0 add edi, ebx
CODE:004013F2 inc esi
CODE:004013F3 jmp short loc_4013E2
CODE:004013F5:
CODE:004013F5 loc_4013F5: ; CODE XREF: sub_4013D8+10↓j
CODE:004013F5 xor edi, 1234h
CODE:004013F5 mov ebx, edi Salida de la call
CODE:004013F5 retn
CODE:004013FD sub_4013D8 endp
CODE:004013FD
CODE:004013FE ;

```

Después de agregar todos los comentarios a las instrucciones que nos vamos a encontrar, queda así:

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

```

CODE:004013D8 sub_4013D8 proc near ; CODE XREF: WndProc+110↑p
CODE:004013D8 arg_0= dword ptr 4
CODE:004013D8
CODE:004013D8 xor eax, eax ; XORrea EAX con EAX (Deja EAX a 0)
CODE:004013D8 xor edi, edi ; XORrea EDI con EDI (Deja EDI a 0)
CODE:004013D8 xor ebx, ebx ; XORrea EBX con EBX (Deja EBX a 0)
CODE:004013D8 mov esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipeado
CODE:004013D8
CODE:004013D8 loc_4013E2: ; CODE XREF: sub_4013D8+1B↓j
CODE:004013D8 mov al, 0Ah ; Mueve Ah a AL
CODE:004013D8 mov bl, [esi] ; Mueve a BL el primer byte del contenido de ESI
CODE:004013D8 test bl, bl ; Testea BL con BL
CODE:004013D8 jz short loc_4013F5 ; Si el resultado del testeо BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013D8 sub bl, 30h ; Resta 30h a BL
CODE:004013D8 imul edi, eax ; Multiplica EAX con EDI y el resultado lo guarda en EDI
CODE:004013D8 add edi, ebx ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013D8 inc esi ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013D8 jmp short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013D8
CODE:004013D8 loc_4013F5: ; CODE XREF: sub_4013D8+10↓j
CODE:004013D8 xor edi, 1234h ; Xorrea el valor de EDI con 1234h y guarda el resultado en EDI
CODE:004013D8 mov ebx, edi ; Mueve el valor de EDI a EBX
CODE:004013D8 ret ; Salida de la CALL
CODE:004013D8 endp
CODE:004013D8
CODE:004013D8

```

Las tres primeras instrucciones "xor" van a dejar los registros "EAX, EDI y ESI" con valor "0", (cualquier valor xorreado por sí mismo el resultado siempre es cero).

Ejecutamos los tres "xor" y vemos que los tres registros han quedado a cero.

IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

```

CODE:004013D8 sub_4013D8 proc near ; CODE XREF: WndProc+110↑p
CODE:004013D8 arg_0= dword ptr 4
CODE:004013D8
CODE:004013D8 xor eax, eax ; XORrea EAX con EAX (Deja EAX a 0)
CODE:004013D8 xor edi, edi ; XORrea EDI con EDI (Deja EDI a 0)
CODE:004013D8 xor ebx, ebx ; XORrea EBX con EBX (Deja EBX a 0)
CODE:004013D8 mov esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipeado
CODE:004013D8
CODE:004013D8 loc_4013E2: ; CODE XREF: sub_4013D8+1B↓j
CODE:004013D8 mov al, 0Ah ; Mueve Ah a AL
CODE:004013D8 mov bl, [esi] ; Mueve a BL el primer byte del contenido de ESI
CODE:004013D8 test bl, bl ; Testea BL con BL
CODE:004013D8 jz short loc_4013F5 ; Si el resultado del testeо BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013D8 sub bl, 30h ; Resta 30h a BL
CODE:004013D8 imul edi, eax ; Multiplica EAX con EDI y el resultado lo guarda en EDI
CODE:004013D8 add edi, ebx ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013D8 inc esi ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013D8 jmp short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013D8
CODE:004013D8 loc_4013F5: ; CODE XREF: sub_4013D8+10↓j
CODE:004013D8 xor edi, 1234h ; Xorrea el valor de EDI con 1234h y guarda el resultado en EDI
CODE:004013D8 mov ebx, edi ; Mueve el valor de EDI a EBX
CODE:004013D8 ret ; Salida de la CALL
CODE:004013D8
CODE:004013D8

```

Ahora "mov esi, [esp+arg_0]" va a mover nuestro Serial tipeado a "ESI", traceamos y eso es lo que ha hecho

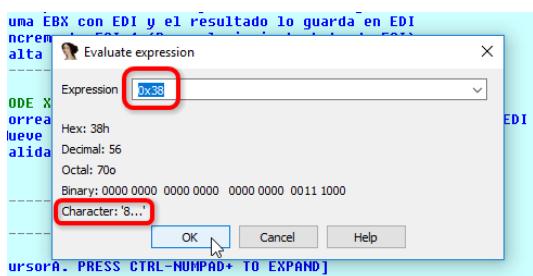
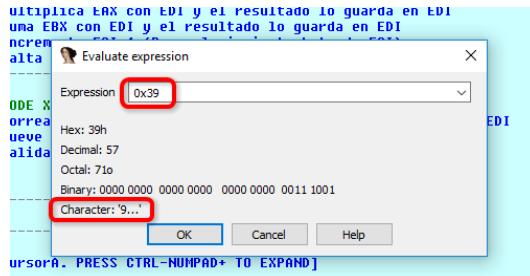
IDA - C:\Users\Usuar\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)

```

CODE:004013D8 sub_4013D8 proc near ; CODE XREF: WndProc+110↑p
CODE:004013D8 arg_0= dword ptr 4
CODE:004013D8
CODE:004013D8 xor eax, eax ; XORrea EAX con EAX (Deja EAX a 0)
CODE:004013D8 xor edi, edi ; XORrea EDI con EDI (Deja EDI a 0)
CODE:004013D8 xor ebx, ebx ; XORrea EBX con EBX (Deja EBX a 0)
CODE:004013D8 mov esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipeado
CODE:004013D8
CODE:004013D8 loc_4013E2: ; CODE XREF: sub_4013D8+1B↓j
CODE:004013D8 mov al, 0Ah ; Mueve Ah a AL
CODE:004013D8 mov bl, [esi] ; Mueve a BL el primer byte del contenido de ESI
CODE:004013D8 test bl, bl ; Testea BL con BL
CODE:004013D8 jz short loc_4013F5 ; Si el resultado del testeо BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013D8 sub bl, 30h ; Resta 30h a BL
CODE:004013D8 imul edi, eax ; Multiplica EAX con EDI y el resultado lo guarda en EDI
CODE:004013D8 add edi, ebx ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013D8 inc esi ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013D8 jmp short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013D8
CODE:004013D8 loc_4013F5: ; CODE XREF: sub_4013D8+10↓j
CODE:004013D8 xor edi, 1234h ; Xorrea el valor de EDI con 1234h y guarda el resultado en EDI
CODE:004013D8 mov ebx, edi ; Mueve el valor de EDI a EBX
CODE:004013D8 ret ; Salida de la CALL
CODE:004013D8
CODE:004013D8

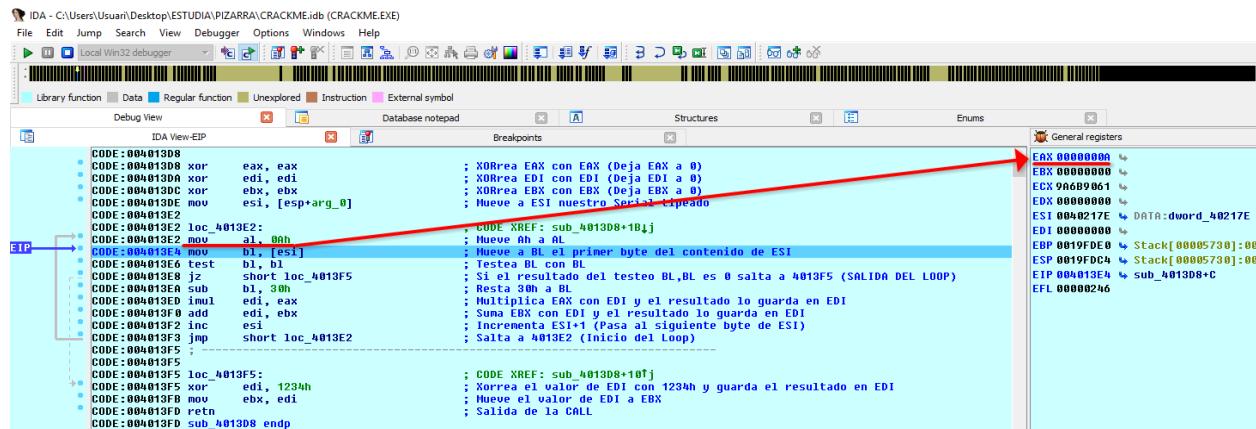
```

$$"39h" = 9 \text{ y } "38h" = 8 \rightarrow 39h \ 38h \ 39h \ 38h \ 39h \ 38h = 989898$$

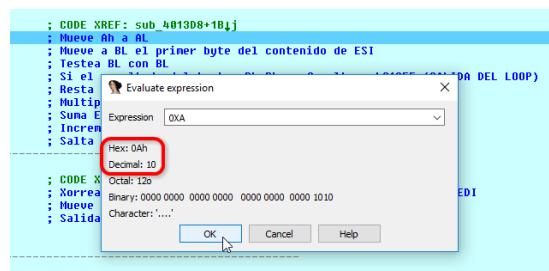


La siguiente instrucción es un "mov al, 0Ah", va a mover "Ah" a "AL"

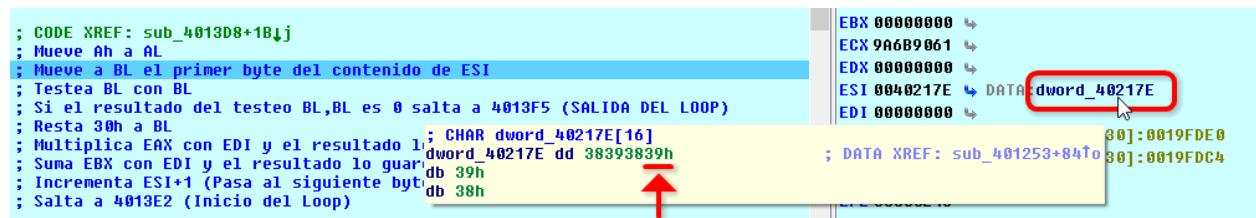
Traceamos y aquí lo vemos



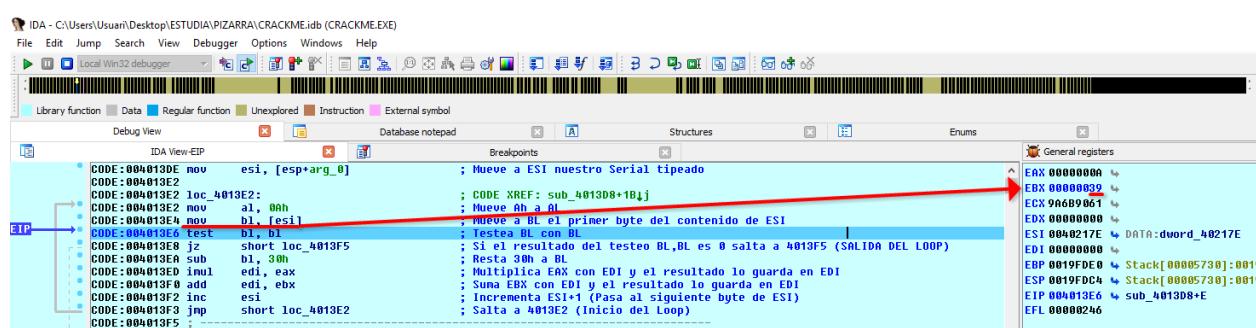
Ah = 10dec



Ahora con el "mov bl, [esi]" moverá el primer "byte" del contenido de "ESI" a "BL"



Ejecutamos la instrucción y eso es lo que ha hecho



Con "test bl,bl" va a testear "BL" con "BL" ("BL" = "39h" = "9")

Y con "jz short loc_4013F5" el salto condicional "jz" nos deja continuar ya que el resultado del testeo es "39h" = "9".

Cuando en "BL" encuentre valor "0" el Flag ZF levantará bandera y el salto condicional nos llevará a "CODE:004013F5" que es la salida del "Loop" en el que nos encontramos inmersos ahora.

(En nuestro caso, esto sucederá cuando ya no queden más caracteres para leer del Serial tipeado)

```

CODE:004013DE mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipeado
CODE:004013E2 loc_4013E2:           ; CODE XREF: sub_4013D8+18j
CODE:004013E2 mov    al, 0Ah          ; Mueve Ah a AL
CODE:004013E4 mov    bl, [esi]        ; Mueve a BL el primer byte del contenido de ESI
CODE:004013E6 test   bl, bl          ; Testea BL con BL
CODE:004013E8 jz    short loc_4013F5 ; Si el resultado del testeo BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013E9 sub   bl, 30h          ; Resta 30h a BL
CODE:004013ED imul   edi, eax        ; Multiplica EAX con EDI y el resultado lo guarda en EDI
CODE:004013F0 add    edi, ebx        ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013F2 inc    esi             ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013F3 jmp    short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013F5 ; -----
CODE:004013F5 loc_4013F5:           ; CODE XREF: sub_4013D8+10fj
CODE:004013F5 xor    edi, 1234h       ; Xorra el valor de EDI con 1234h y guarda el resultado en EDI
CODE:004013F6 mov    ebx, edi        ; Mueve el valor de EDI a EBX
CODE:004013F7 retn                   ; Salida de la CALL
CODE:004013FD sub   _4013D8 endp      ; Salida a 4013D8 (Final del Loop)
CODE:004013FD

```

Traceamos, y ahora lo que va a hacer la instrucción "sub bl, 30h" es restar "bl" = "39h" - "30h" y el resultado lo va a guardar en "BL"

```

CODE:004013DE mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipeado
CODE:004013E2 loc_4013E2:           ; CODE XREF: sub_4013D8+18j
CODE:004013E2 mov    al, 0Ah          ; Mueve Ah a AL
CODE:004013E4 mov    bl, [esi]        ; Mueve a BL el primer byte del contenido de ESI
CODE:004013E6 test   bl, bl          ; Testea BL con BL
CODE:004013E8 jz    short loc_4013F5 ; Si el resultado del testeo BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013E9 sub   bl, 30h          ; Resta 30h a BL
CODE:004013ED imul   edi, eax        ; Multiplica EAX con EDI y el resultado lo guarda en EDI
CODE:004013F0 add    edi, ebx        ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013F2 inc    esi             ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013F3 jmp    short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013F5 ; -----
CODE:004013F5 loc_4013F5:           ; CODE XREF: sub_4013D8+10fj
CODE:004013F5 xor    edi, 1234h       ; Xorra el valor de EDI con 1234h y guarda el resultado en EDI
CODE:004013F6 mov    ebx, edi        ; Mueve el valor de EDI a EBX
CODE:004013F7 retn                   ; Salida de la CALL
CODE:004013FD sub   _4013D8 endp      ; Salida a 4013D8 (Final del Loop)
CODE:004013FD

```

Con "imul edi, eax" va a multiplicar los valores de ambos registros y el resultado la va a guardar en "EDI", pues está claro que va a poner "EDI" a cero, ya que cualquier numero multiplicado por "0" el resultado siempre es "0"

"EAX" = Ah y "EDI" = 0h (Ah x 0h = 0h) "EDI" = 0h

Traceamos y

```

CODE:004013DE mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipeado
CODE:004013E2 loc_4013E2:           ; CODE XREF: sub_4013D8+18j
CODE:004013E2 mov    al, 0Ah          ; Mueve Ah a AL
CODE:004013E4 mov    bl, [esi]        ; Mueve a BL el primer byte del contenido de ESI
CODE:004013E6 test   bl, bl          ; Testea BL con BL
CODE:004013E8 jz    short loc_4013F5 ; Si el resultado del testeo BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013E9 sub   bl, 30h          ; Resta 30h a BL
CODE:004013ED imul   edi, eax        ; Multiplica EAX con EDI y el resultado lo guarda en EDI
CODE:004013F0 add    edi, ebx        ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013F2 inc    esi             ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013F3 jmp    short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013F5 ; -----
CODE:004013F5 loc_4013F5:           ; CODE XREF: sub_4013D8+10fj
CODE:004013F5 xor    edi, 1234h       ; Xorra el valor de EDI con 1234h y guarda el resultado en EDI
CODE:004013F6 mov    ebx, edi        ; Mueve el valor de EDI a EBX
CODE:004013F7 retn                   ; Salida de la CALL
CODE:004013FD sub   _4013D8 endp      ; Salida a 4013D8 (Final del Loop)
CODE:004013FD

```

Ahora con "add edi, ebx" va a sumar (adicionar) los valores de "EBX" y "EDI" y el resultado lo guardará en "EDI", por lo que

"EBX" = "00000009h" + "EDI" = "00000000h" = "000000009h" con lo que "EDI" = "00000009h"

Ejecutamos la instrucción y comprobamos que ahora "EDI" vale "9h"

```

CODE:004013D0 mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipiado
CODE:004013E2 loc_4013E2:
CODE:004013E2 mov    al, 00h ; Mueve Ah a AL
CODE:004013E5 mov    bl, [esi] ; Mueve a BL el primer byte del contenido de ESI
CODE:004013E6 test   bl, bl ; Testea BL con BL
CODE:004013E8 jz     short loc_4013F5 ; Si el resultado del testeo BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013E9 sub    bl, 30h ; Resta 30h a BL
CODE:004013ED imul   edi, eax ; Multiplica EBX con EDI y el resultado lo guarda en EDI
CODE:004013F0 add    edi, ebx ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013F2 inc    esi    ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013F3 jmp    short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013F5 ; 
CODE:004013F5 loc_4013F5:
CODE:004013F5 xor    edi, 1234h ; Xorrea el valor de EDI con 1234h y guarda el resultado en EDI
CODE:004013FB mov    ebx, edi ; Mueve el valor de EDI a EBX
CODE:004013FD retn   ; Salida de la CALL
CODE:004013FD sub_4013D8 endp

```

Ahora viene un "inc esi", va a incrementar "1h" el valor de "ESI"

Si "ESI" ahora mismo vale "0040217E" (ver foto superior), al sumarle "1h" valdrá "0040217F" con lo cual está preparando el segundo "byte" de nuestro "Serial" para trabajar con él, y por lo tanto le tocará el turno a "38"

39 38 39 38 39 38 = 989898

Traceamos

```

CODE:004013D0 mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipiado
CODE:004013E2 loc_4013E2:
CODE:004013E2 mov    al, 00h ; Mueve Ah a AL
CODE:004013E5 mov    bl, [esi] ; Mueve a BL el primer byte del contenido de ESI
CODE:004013E6 test   bl, bl ; Testea BL con BL
CODE:004013E8 jz     short loc_4013F5 ; Si el resultado del testeo BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013E9 sub    bl, 30h ; Resta 30h a BL
CODE:004013ED imul   edi, eax ; Multiplica EBX con EDI y el resultado lo guarda db 39h ; CHAR dword_40217E[16]
CODE:004013F0 add    edi, ebx ; Suma EBX con EDI y el resultado lo guarda db 39h ; CHAR dword_40217E[16]
CODE:004013F2 inc    esi    ; Incrementa ESI+1 (Pasa al siguiente byte db 38h )
CODE:004013F3 jmp    short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013F5 ; 
CODE:004013F5 loc_4013F5:
CODE:004013F5 xor    edi, 1234h ; Xorrea el valor de EDI con 1234h y guarda db 0
CODE:004013FB mov    ebx, edi ; Mueve el valor de EDI a EBX db 0
CODE:004013FD retn   ; Salida de la CALL db 0
CODE:004013FD sub_4013D8 endp

```

La siguiente es un salto incondicional "jmp short loc_4013E2", está claro que nos lleva al inicio del "Loop" y así nos lo marca IDA con la flecha verde.

Traceamos y aquí estamos, en "CODE:004013E2"

```

CODE:004013D0 mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipiado
CODE:004013E2 loc_4013E2:
CODE:004013E2 mov    al, 00h ; Mueve Ah a AL
CODE:004013E5 mov    bl, [esi] ; Mueve a BL el primer byte del contenido de ESI
CODE:004013E6 test   bl, bl ; Testea BL con BL
CODE:004013E8 jz     short loc_4013F5 ; Si el resultado del testeo BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013E9 sub    bl, 30h ; Resta 30h a BL
CODE:004013ED imul   edi, eax ; Multiplica EBX con EDI y el resultado lo guarda en EDI
CODE:004013F0 add    edi, ebx ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013F2 inc    esi    ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013F3 jmp    short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013F5 ; 
CODE:004013F5 loc_4013F5:
CODE:004013F5 xor    edi, 1234h ; Xorrea el valor de EDI con 1234h y guarda el resultado en EDI
CODE:004013FB mov    ebx, edi ; Mueve el valor de EDI a EBX
CODE:004013FD retn   ; Salida de la CALL
CODE:004013FD sub_4013D8 endp

```

La siguiente instrucción vuelve a ser "mov al, 0Ah", por lo que moverá "Ah" a "AL", pero "EAX" continuará valiendo lo mismo, porque ya vale "0000000A"

La pasamos con otro trazado y aquí lo vemos

```

CODE:004013DE mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipado
CODE:004013E2 loc_4013E2:           ; CODE XREF: sub_4013D8+104j
CODE:004013E2 mov    al, 0Ah          ; Mueve Ah a AL
CODE:004013E4 mov    bl, [esi]        ; Mueve a BL el primer byte del contenido de ESI
CODE:004013E6 test   bl, bl          ; Testea BL con BL
CODE:004013E8 jz    short loc_4013F5 ; Si el resultado del testeo BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013E9 sub    bl, 30h          ; Resta 30h a BL
CODE:004013ED imul   edi, eax        ; Multiplica EAX con EDI y el resultado lo guarda en EDI
CODE:004013F0 add    edi, ebx        ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013F2 inc    esi             ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013F3 jmp    short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013F5 ;
CODE:004013F5

```

Ahora con el "mov bl, [esi]", va a pasar el segundo "byte" del contenido de "ESI" (como ya vimos antes que se había preparado en "ESI" "dword_40217E+1", (39 38 39 38 = 989898) y lo va a meter a "BL"

Traceamos y efectivamente aquí lo vemos

```

CODE:004013DE mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipado
CODE:004013E2 loc_4013E2:           ; CODE XREF: sub_4013D8+1B4j
CODE:004013E2 mov    al, 0Ah          ; Mueve Ah a AL
CODE:004013E4 mov    bl, [esi]        ; Mueve a BL el primer byte del contenido de ESI
CODE:004013E6 test   bl, bl          ; Testea BL con BL
CODE:004013E8 jz    short loc_4013F5 ; Si el resultado del testeo BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013E9 sub    bl, 30h          ; Resta 30h a BL
CODE:004013ED imul   edi, eax        ; Multiplica EAX con EDI y el resultado lo guarda en EDI
CODE:004013F0 add    edi, ebx        ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013F2 inc    esi             ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013F3 jmp    short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013F5 ;
CODE:004013F5

```

Traceamos, traceamos y traceamos ... hasta que no queden más caracteres de nuestro "Serial" y cuando lleguemos a "CODE:004013E8" "BL" vale "0", el Flag ZF se activa "1" y el salto condicional "jz" nos sacará del "Loop".

También observamos que ahora el registro "EDI" vale "F1ACA", concluyendo que este "Loop" convierte los caracteres del Serial al sistema hexadecimal.

```

CODE:004013DE mov    esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipado
CODE:004013E2 loc_4013E2:           ; CODE XREF: sub_4013D8+1B4j
CODE:004013E2 mov    al, 0Ah          ; Mueve Ah a AL
CODE:004013E4 mov    bl, [esi]        ; Mueve a BL el primer byte del contenido de ESI
CODE:004013E6 test   bl, bl          ; Testea BL con BL
CODE:004013E8 jz    short loc_4013F5 ; Si el resultado del testeo BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013E9 sub    bl, 30h          ; Resta 30h a BL
CODE:004013ED imul   edi, eax        ; Multiplica EAX con EDI y el resultado lo guarda en EDI
CODE:004013F0 add    edi, ebx        ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013F2 inc    esi             ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013F3 jmp    short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013F5 ;
CODE:004013F5 loc_4013F5:           ; CODE XREF: sub_4013D8+10fj
CODE:004013F5 mov    edi, 1234h        ; Guarda el valor de EDI 1234h y guarda el resultado en EDI
CODE:004013F8 mov    ebx, edi        ; Muestra el valor de EDI a EBX
CODE:004013FB retf   ; Salida de la CALL
CODE:004013FD sub    _4013D8 endp
CODE:004013F8
CODE:00401400 :
CODE:00401400 jmp    ds:_KillTimer
CODE:00401404 mov    eax, 1
CODE:00401404 xor    eax, 1
CODE:00401404 jne    ds:GetSystemMetrics
CODE:00401404 ; [00000000 BYTES: COLLAPSED FUNCTION LoadCursorA. PRESS CTRL+NUMPAD+ TO EXPAND]
000000E5 004013E8: sub_4013D8+10 (Synchronized with EIP)

```

Otro trazado para salir del "Loop" y aquí, la instrucción "xor edi, 1234h" va a hacer un "xor" con 1234h al contenido de "EDI" y el resultado lo va a guardar en "EDI". Vamos a calcularlo

HEXADECIMAL	0	1	2	3	4	5	6	7
BINARIO	0000	0001	0010	0011	0100	0101	0110	0111

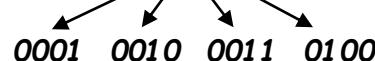
HEXADECIMAL	8	9	A	B	C	D	E	F
BINARIO	1000	1001	1010	1011	1100	1101	1110	1111

Si el valor de "EDI" en nuestro caso es "000F1ACA" lo pasamos a binario con la tabla



11110001101011001010

Hacemos lo mismo con el valor "1234"



0001001000110100

"XORreamos" los dos valores obtenidos teniendo en cuenta:

$$0 \text{ XOR } 0 = 0$$

$$0 \text{ XOR } 1 = 1$$

$$1 \text{ XOR } 0 = 1$$

$$1 \text{ XOR } 1 = 0$$

$$\begin{array}{cccccccc} 00 & 01 & 00 & 10 & 00 & 11 & 01 & 00 \\ 11 & 11 & 00 & 01 & 10 & 10 & 11 & 00 & 10 & 10 \end{array}$$

Resultado 11 11 00 00 10 00 11 11 11 10

Ordenamos en "Nibble":

1111 0000 1000 1111 1110

Convertimos este binario a hexadecimal volviendo a utilizar la tabla

HEXADECIMAL	0	1	2	3	4	5	6	7
BINARIO	0000	0001	0010	0011	0100	0101	0110	0111

HEXADECIMAL	8	9	A	B	C	D	E	F
BINARIO	1000	1001	1010	1011	1100	1101	1110	1111

Con lo cual

1111 0000 1000 1111 1110

Obtenemos

↓ ↓ ↓ ↓ ↓
F 0 8 F E

Traceamos para verificar que nuestro resultado obtenido es realmente correcto, "EDI 000F08FE"Bienn....

```

IDA - C:\Users\Usuan\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)
File Edit Jump Search View Debugger Options Windows Help
Local Win32 debugger
Library function Data Regular function Unexplored Instruction External symbol
Debug View Database notepad Structures Enums
IDA View-EIP Breakpoints General registers
CODE:004013DE mov esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipeado
CODE:004013E2
CODE:004013E2 loc_4013E2: ; CODE XREF: sub_4013D8+1B1j
CODE:004013E2 mov al, 0Ah ; Mueve Ah a AL
CODE:004013E4 mov bl, [esi] ; Mueve a BL el primer byte del contenido de ESI
CODE:004013E6 test bl, bl ; Testea BL con BL
CODE:004013E8 jz short loc_4013F5 ; Si el resultado del testeо BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013E9 sub bl, 30h ; Resta 30h a BL
CODE:004013ED imul edi, eax ; Multiplica EAX con EDI y el resultado lo guarda en EDI
CODE:004013F0 add edi, ebx ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013F2 inc esi ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013F3 jmp short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013F5
CODE:004013F5
CODE:004013F5 loc_4013F5: ; CODE XREF: sub_4013D8+10fj
CODE:004013F5 xor edi, 1234h ; Xorrea el valor de EDI con 1234h y guarda el resultado en EDI
CODE:004013F8 mov ebx, edi ; Mueve el valor de EDI a EBX
CODE:004013FD retn ; Salida de la CALL
CODE:004013FD sub_4013D8 endp
CODE:004013FD
CODE:004013FE ; -----
CODE:004013FE jmp ds:KillTimer
CODE:00401404 ; -----
CODE:00401404 jmp ds:GetSystemMetrics
CODE:0040140A ; [ 00000006 BYTES: COLLAPSED FUNCTION LoadCursorA. PRESS CTRL-NUMPAD+ TO EXPAND]
000009FB 004013FB: sub_4013D8+23 (Synchronized with EIP)

```

Ahora "mov ebx, edi" va mover el valor de "EDI" a "EBX", Traceamos y vemos el movimiento

```

IDA - C:\Users\Usuan\Desktop\ESTUDIA\PIZARRA\CRACKME.idb (CRACKME.EXE)
File Edit Jump Search View Debugger Options Windows Help
Local Win32 debugger
Library function Data Regular function Unexplored Instruction External symbol
Debug View Database notepad Structures Enums
IDA View-EIP Breakpoints General registers
CODE:004013DE mov esi, [esp+arg_0] ; Mueve a ESI nuestro Serial tipeado
CODE:004013E2
CODE:004013E2 loc_4013E2: ; CODE XREF: sub_4013D8+1B1j
CODE:004013E2 mov al, 0Ah ; Mueve Ah a AL
CODE:004013E4 mov bl, [esi] ; Mueve a BL el primer byte del contenido de ESI
CODE:004013E6 test bl, bl ; Testea BL con BL
CODE:004013E8 jz short loc_4013F5 ; Si el resultado del testeо BL,BL es 0 salta a 4013F5 (SALIDA DEL LOOP)
CODE:004013E9 sub bl, 30h ; Resta 30h a BL
CODE:004013ED imul edi, eax ; Multiplica EAX con EDI y el resultado lo guarda en EDI
CODE:004013F0 add edi, ebx ; Suma EBX con EDI y el resultado lo guarda en EDI
CODE:004013F2 inc esi ; Incrementa ESI+1 (Pasa al siguiente byte de ESI)
CODE:004013F3 jmp short loc_4013E2 ; Salta a 4013E2 (Inicio del Loop)
CODE:004013F5
CODE:004013F5
CODE:004013F5 loc_4013F5: ; CODE XREF: sub_4013D8+10fj
CODE:004013F5 xor edi, 1234h ; Xorrea el valor de EDI con 1234h y guarda el resultado en EDI
CODE:004013F8 mov ebx, edi ; Mueve el valor de EDI a EBX
CODE:004013FD retn ; Salida de la CALL
CODE:004013FD sub_4013D8 endp
CODE:004013FD
CODE:004013FE ; -----
CODE:004013FE jmp ds:KillTimer
CODE:00401404 ; -----
CODE:00401404 jmp ds:GetSystemMetrics
CODE:0040140A ; [ 00000006 BYTES: COLLAPSED FUNCTION LoadCursorA. PRESS CTRL-NUMPAD+ TO EXPAND]
000009FD 004013FD: sub_4013D8+25 (Synchronized with EIP)

```

Llegamos al "retn" , que nos sacará de esta "call" en la que nos encontramos y con que ya hemos descubierto lo que hace, traceamos para que se ejecute, aparecemos en "CODE:0040123D" y agregamos el siguiente comentario

```

CODE:0040121E call sub_40137E ; Mete al Stack el contenido de EAX
CODE:00401223 cmp eax, 0
CODE:00401226 jz short loc_4011E6
CODE:00401226 push offset String
CODE:00401229 call sub_40137E
CODE:0040122D push eax ; Mete al Stack el contenido de EAX
CODE:0040122E cmp eax, 0x217Eh
CODE:0040122F call sub_40120B ; Convierte Serial a Hexadecimal
CODE:00401230 add esp, 4
CODE:00401240 pop eax
CODE:00401241 cmp eax, ebx
CODE:00401243 jz short loc_40124C
CODE:00401245 call sub_401362
CODE:00401246 jmp short loc_4011E6
CODE:0040124C
CODE:0040124C loc_40124C: ; CODE XREF: WndProc+118†j
CODE:0040124C call sub_40134D
CODE:00401251 jmp short loc_4011E6
CODE:00401251 WndProc endp ; sp-analysis Failed
CODE:00401253
CODE:00401253 ; ===== S U
CODE:00401253 ; Attributes: bp-based Frame

```

Ahora "add esp, 4" sumará 4 al valor de "ESP" con lo cual "ESP" pasará a valer "0019FDCC" (0019FDC8 + 4)

```

233 push 40217Eh
238 call sub_40130B ; Convierte Serial a Hexadecimal
23D add esp, 4
240 pop eax
241 cmp eax, ebx
243 jz short loc_40124C
245 call sub_401362
246 jmp short loc_4011E6
24C
24C loc_40124C:
24C call sub_40134D
251 jmp short loc_4011E6
251 WndProc endp ; sp-analysis Failed
251
253 ; ===== S U
253 ; Attributes: bp-based Frame

```

Hex: 19FDCh
Decimal: 1703372
Octal: 6376714
Binary: 0000 0000 0001 1001 1111 1101 1100 1100
Character: ';'.

Y justamente en este puñetero punto.... es donde en mi anterior tutorial con este mismo "crackme".... con la "Tool x63dbg"..... no entendí muy bien lo que hacía esto de sumarle 4 al registro "ESP" , por lo que van a disculparme por unos instantes
Y antes de ejecutar la instrucción "add esp,4", me levanto de mi silla, estiro un poco las piernas, me dirijo a la nevera abro una cerveza "Woll-Damm" (pequeña), bien fresquita, le pego un buen trago....., vuelvo a sentarme delante del ordenador y con sumo detenimiento y muy concentrado observo lo siguiente:

```

CODE:00401229 call sub_40137E ; Mete al Stack el contenido de EAX
CODE:0040122D push eax ; Mete al Stack el contenido de EAX
CODE:0040122E cmp eax, 0x217Eh
CODE:0040122F call sub_40120B ; Convierte Serial a Hexadecimal
CODE:00401230 add esp, 4
CODE:00401240 pop eax
CODE:00401241 cmp eax, ebx
CODE:00401243 jz short loc_40124C
CODE:00401245 call sub_401362
CODE:00401246 jmp short loc_4011E6
CODE:0040124C
CODE:0040124C loc_40124C: ; CODE XREF: WndProc+118†j
CODE:0040124C call sub_40134D
CODE:00401251 jmp short loc_4011E6
CODE:00401251 WndProc endp ; sp-analysis Failed
CODE:00401253
CODE:00401253 ; ===== S U
CODE:00401253 ; Attributes: bp-based Frame

```

Hex View-1

00401218 1F 21 40 00 FF 35 CA 28 40 00 EB 99 02 00 00 .10.-5--0.80...0	0040121E 00402184 DATA:dword 40217E
00401228 DD 60 00 68 53 12 40 00 FF 75 08 68 15 21 40 00 .11.HS.0.-u.h..10.	00401219 00005794
0040122E FF 35 CA 28 40 00 EB 7D 02 00 00 83 F8 00 74 BE -5--0.b)...3*..tV	0040121A 0040218E DATA:String
00401228 68 BE 21 40 00 EB 4C 01 00 00 50 68 7E 21 40 00 hñññ.bl...Ph*10.	0040121B 00000000
00401238 EB 98 01 00 00 83 C4 04 58 38 C3 74 07 EB 18 01 þæ...å-X:tþ..þ	0040121C 00000011
00401248 00 00 EB 9A EB FC 00 00 00 EB 93 C8 00 00 00 53 ..ññb...ññb+...S	0040121D 00000000
00401258 56 57 81 7D 1C 01 00 00 74 34 81 7D 0C 11 01 UW)...t4...-	0040121E Stack[00006C1C]:0019FE0C
00401268 00 00 74 35 83 7D 0C 10 0F 84 81 00 00 00 81 7D ..ñññ...ñññ...ñ	0040121F 74A4BF18 user32.dll:user32_AddClipboardFormatListener+49B

La primera carta del "Stack" es "0019FDC8" y justamente la que le sigue es "0019FDCC" (es el resultado de la suma del dichoso 4 a "ESP"). Además veo que ahí guarda "00005794", que si recordamos era el resultado de hacer el "xor 5678h" a la suma de los valores hexadecimales de nuestro "Name" previamente pasados a MAYÚSCULAS.

Bien, ahora miro en el "Dump" lo que guarda el "dword_40217E"

0019FDC8	0040217E	DATA:dword_40217E	Synchronize with ▾
0019FDCC	00005794		Jump to ESP
0019FDD0	0040218E	DATA:String	Jump to EBP
0019FDD4	00401128	WndProc	Follow in disassembly
0019FDD8	00000011		Follow in hex dump
0019FDDC	00000000		
0019FDE0	0019FE0C	Stack[00006C1C]:0	
0019FDE4	74A4BF1B	user32.dll:user32	
0019FDE8	002606CC		
0019FDEC	00000011		

Y nos muestra el Serial tipeado (truco)

The screenshot shows the Immunity Debugger interface. The left pane, titled 'Hex View-1', displays memory dump data from address 3040213E to 304021CE. The right pane, titled 'Stack view', shows the stack starting at address 0019FD08. A red box highlights the memory dump area, and another red box highlights the stack area. A cursor points to the string 'QWERTY.....' in the stack dump.

Con lo cual lo que va a hacer la suma del dichoso 4 es que si en este momento la primera carta del "Stack" es la de nuestro Serial trucho, cuando se ejecute la instrucción "add esp, 4" lo que hará es pasar la segunda carta del mazo a la primera posición, y efectivamente el resultado de la instrucción es "0019FDCC" (0019FDC8 + 4)

Traceo, y aquí lo tenemos

The screenshot shows the Immunity Debugger interface with two main panes: the assembly pane on the left and the memory dump pane on the right. The assembly pane displays assembly code for the `VndProc+118` function, including instructions like `add esp, 4`, `pop eax`, and `jmp short loc_4011E6`. A red arrow points from the assembly pane to the memory dump pane, which shows the stack frame at address `0019FDE0`. The memory dump pane also shows other memory regions and registers like EIP, EDI, and ESI.

Ahora le toca el turno a "pop eax", va a pasar el contenido de la primera carta del mazo del "Stack" que ya sabemos que vale "00005794" al registro "EAX"

Traceamos y eso es lo que ha hecho

El **"cmp eax,ebx"** se encarga de comparar los contenidos de los dos registros **"EAX"** y **"EBX"**.

The screenshot shows the assembly view in IDA Pro. The assembly code is as follows:

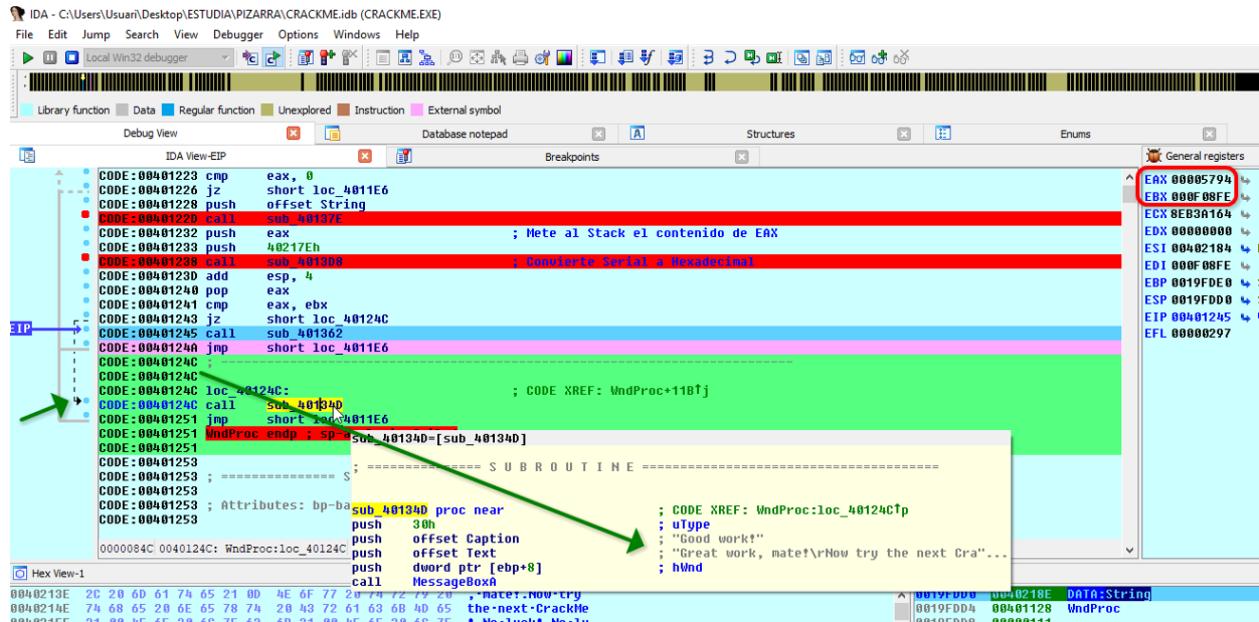
```
CODE:00401220 call sub_40187E
CODE:00401223 push eax ; Mete al Stack el contenido de EAX
CODE:00401223 push 40217Eh ; Convierte Serial a Hexadecimal
CODE:00401228 call sub_401808
CODE:0040122B add esp, 4
CODE:00401240 pop eax
CODE:00401241 cmp eax, ebx
CODE:00401243 jz short loc_40124C
CODE:00401245 call sub_401362
CODE:00401246 jmp short loc_4011E6
CODE:0040124C
CODE:0040124D loc_40124C: ; CODE XREF: WndProc+11B↑j
CODE:0040124D call sub_40134D
CODE:00401251 jmp short loc_4011E6
CODE:00401251 WndProc endp : sp-analysis failed
CODE:00401251
CODE:00401253
```

The assembly pane has several annotations:

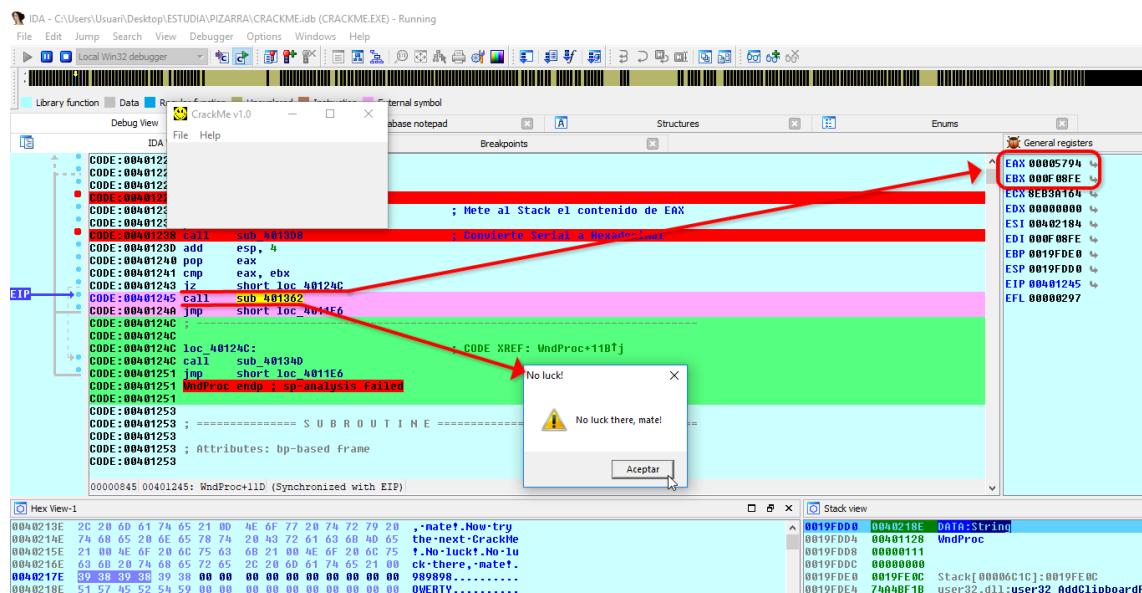
- A red box highlights the assembly code from `sub_40187E` to `sub_401808`, specifically the stack pushes and the call to `sub_401808`.
- A red arrow points from the highlighted assembly code to the "General registers" pane on the right.
- The "General registers" pane lists the following registers:
 - EBX 00000579h
 - ECX 000F08FE
 - EDX 00000000h
 - ESI 00401218h
 - EDI 000F08FE
 - EIP 0019FDE0
 - ESP 0019FD00
 - EIP 00401243
 - EFL 00000297

y por último el salto condicional "jz short loc_40124C" va a decidir que si el resultado de la comparación no es igual), nos deja continuar llevándonos a CODE:00401245 donde la "Call sub_401362 nos saltará el "Chico Malo",

Si por el contrario el resultado de la comparación es igual, saltaremos a "Short loc_40124C donde a su vez la "call sub_40134D nos saltará el "Chico Bueno"



Bien pues con que los valores de los registros "EAX" y "EBX" son diferentes, traceamos y nos manda a "Chico Malo"....Grrrrrr....



La solución del algoritmo estriba en hacer un "Xor 5678" a la suma de los valores Hexadecimales de los caracteres alfa una vez pasados a MAYÚSCULAS de nuestro Name tipeado, y a su resultado le hacemos un "Xor 1234", finalmente pasaremos la conversión de Hexadecimal a Decimal y obtendremos el Serial correcto.

Manos a la obra...

Name: QwErTy = 51h 77h 45h 72h 54h 79h
QWERTY = 51h 57h 45h 52h 54h 59h
 $51h + 57h + 45h + 52h + 54h + 59h = 1EC$
 $1EC \text{ xor } 5678 = 5794$ (ya lo obtuvimos manualmente)
 $5794 \text{ xor } 1234 = \text{Ahora vamos a por este resultado}$

HEXADECIMAL	0	1	2	3	4	5	6	7
BINARIO	0000	0001	0010	0011	0100	0101	0110	0111
HEXADECIMAL	8	9	A	B	C	D	E	F
BINARIO	1000	1001	1010	1011	1100	1101	1110	1111

Si el valor de "EDI" en nuestro caso es "**5794**" lo pasamos a binario con la tabla

0101 0111 1001 0100

Quitamos el cero de la izquierda y nos queda así

10101110010100

Hacemos lo mismo con el valor "**1234**"

0001 0010 0011 0100

1001000110100

"XORreamos" los dos valores obtenidos teniendo en cuenta:

$$0 \text{ XOR } 0 = 0$$

$$0 \text{ XOR } 1 = 1$$

$$1 \text{ XOR } 0 = 1$$

$$1 \text{ XOR } 1 = 0$$

1 00 10 00 11 01 00
1 01 01 11 10 01 01 00

Resultado

01 00 01 01 10 10 00 00

Ordenamos en "Nibble":

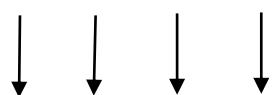
0100 0101 1010 0000

Convertimos este binario a hexadecimal volviendo a utilizar la tabla

HEXADECIMAL	0	1	2	3	4	5	6	7
BINARIO	0000	0001	0010	0011	0100	0101	0110	0111
HEXADECIMAL	8	9	A	B	C	D	E	F
BINARIO	1000	1001	1010	1011	1100	1101	1110	1111

Con lo cual

0100 0101 1010 0000



Obtenemos

4 5 A 0 (Hexadecimal)

Ahora pasamos este valor Hexadecimal a Decimal de la siguiente forma

45A0h

Primero ponemos base 16 elevada a las potencias 0,1,2,3... de Derecha a Izquierda

$$\begin{array}{cccc} 16^3 & 16^2 & 16^1 & 16^0 \\ 4 & 5 & A & 0 \end{array}$$

Desarrollamos las potencias y ponemos sus resultados, y también ponemos la equivalencia de los caracteres individuales hexadecimales a decimales.

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$$\begin{array}{ccccc} 4096 & 256 & 16 & 1 & (\text{Resultado desarrollo potencias}) \\ 16^3 & 16^2 & 16^1 & 16^0 & (\text{Base 16 elevada a potencias}) \\ 4h & 5h & Ah & 0h & (\text{Hexadecimal}) \\ 4 & 5 & 10 & 0 & (\text{Equivalencias hex a decimal}) \end{array}$$

Ahora multiplicamos el resultado del desarrollo de cada potencia por el valor de la conversión hexadecimal a decimal, y obtenemos los siguientes resultados:

$$4096 \times 4 = 16384 \quad 256 \times 5 = 1280 \quad 16 \times 10 = 160 \quad 1 \times 0 = 0$$

Por último sumamos los resultados

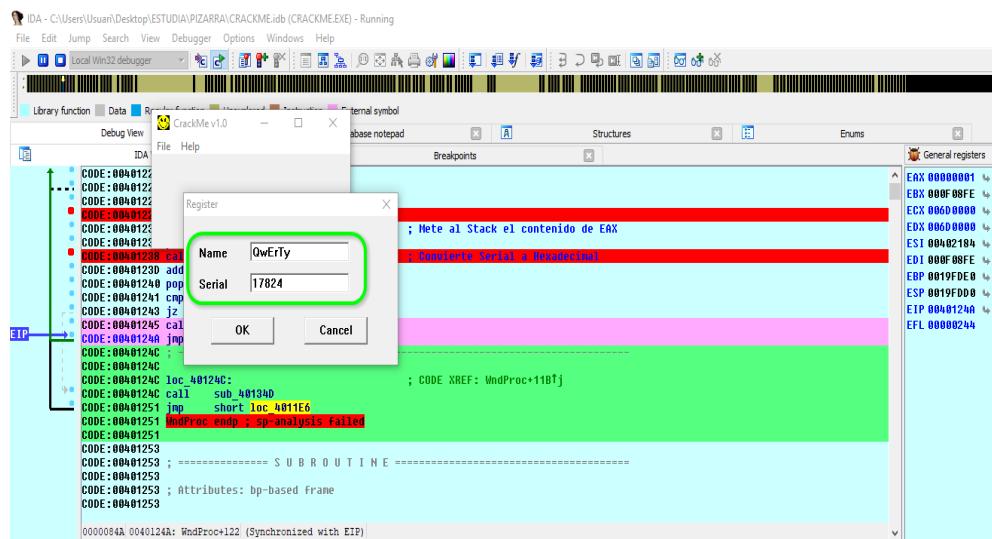
$$16384 + 1280 + 160 + 0 = 17824$$

Y obtenemos que

$$45A0\text{hexadecimal} = 17824\text{Decimal}$$

Con lo cual el serial calculado para nuestro Name "QwErTy" seria "17824"

Vamos a comprobarlo:



Llegamos hasta la zona caliente y vemos como ahora los registros "EAX" y "EBX" valen lo mismo, por lo que esta vez hemos acertado ya que el salto condicional nos va a llevar directos a la zona de "Chico Bueno"

```

CODE:00401223 cmp    eax, 0
CODE:00401226 jz    offset String
CODE:00401228 push   offset String
CODE:0040122B call   sub_40137E
CODE:00401232 push   eax ; Mete al Stack el contenido de EAX
CODE:00401233 push   40217Eh
CODE:00401238 call   sub_401308 ; Convierte Serial a Hexadecimal
CODE:0040123D add    esp, 4
CODE:00401240 pop    eax
CODE:00401241 cmp    eax, ebx
CODE:00401243 jz    short loc_40124C
CODE:00401245 call   sub_40130E
CODE:00401248 jmp    short loc_4011E6
CODE:0040124C ;
CODE:0040124C loc_40124C: ; CODE XREF: WndProc+11B↑j
CODE:0040124C call   sub_401340
CODE:00401251 jmp    short loc_4011E6
CODE:00401251 WndProc endp ; sp-analysis failed
CODE:00401251
CODE:00401253
CODE:00401253 ; ===== SUB ROUTINE =====
CODE:00401253
CODE:00401253 ; Attributes: bp-based frame
CODE:00401253
0000084C 0040124C: WndProc:loc_40124C (Synchronized with EIP)

```

General registers
EAX 00005794
EBX 00005794
ECX 0E83A164
EDX 00000000
ESI 00402183
EDI 00005794
EBP 0019FDE8
ESP 0019FD00
EIP 00401243
EFL 00000246

Stack view
0019FDD0 0040218E DATA:String
0019FDD4 00401128 WndProc
0019FDD8 00000111
0019FDDC 00000000
0019FDE0 0019FE0C Stack[00006C1C]:0019FDE0
0019FDE4 74A4BF1B user32.dll:user32_AddClipboardFormatListener+49B
0019FDE8 002606CC
0019FDEC 00000111
0019FDF0 00000066

Seguimos..... y por finnnnnn salta el deseado y esperado mensaje de "Chico Bueno", dando con ello nuestro trabajo por concluido.

```

CODE:00401223
CODE:00401226
CODE:00401228
CODE:0040122B
CODE:00401232
CODE:00401233
CODE:00401238 call   sub_40137E
CODE:0040123D add    esp, 4
CODE:00401240 pop    eax
CODE:00401241 cmp    eax, ebx
CODE:00401243 jz    short loc_40124C
CODE:00401245 call   sub_401362
CODE:00401248 jmp    short loc_4011E6
CODE:0040124C ;
CODE:0040124C loc_40124C: ; CODE XREF: WndProc+11B↑j
CODE:0040124C call   sub_401340
CODE:00401251 jmp    short loc_4011E6
CODE:00401251 WndProc endp ; sp-analysis failed
CODE:00401251
CODE:00401253
CODE:00401253 ; ===== SUB ROUTINE =====
CODE:00401253
CODE:00401253 ; Attributes: bp-based frame
CODE:00401253
0000084C 0040124C: WndProc:loc_40124C (Synchronized with EIP)

```

Good work!

Great work, mate!
Now try the next CrackMe!

Aceptar

Stack view
0019FDD0 0040218E DATA:String
0019FDD4 00401128 WndProc
0019FDD8 00000111
0019FDDC 00000000
0019FDE0 0019FE0C Stack[00006C1C]:0019FDE0
0019FDE4 74A4BF1B user32.dll:user32_AddClipboardFormatListener+49B

Y me viene a la cabeza, ojalá algún día la tecnología junto con la ciencia y la medicina permitan convertir las **células malas** a binario, y hacerles un "Xor" entre ellas mismas para que queden revertidas a **células buenas**.....

No dejen de crackear nunca....y un fuerte abrazo para todo el grupo.

Hasta el próximo tute.

.....**MISIÓN CUMPLIDA**.....



Mis agradecimientos infinitos a

*Ricardo Narvaja
CracksLatinoS*

Cuando hablas, solo repites lo que ya sabes, pero cuando escuchas quizás aprendas algo nuevo.

Dalai Lama

Salu2

QwErTy

1 de Junio de 2019