



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 PEID v0.95 x64dbg (snapshot_2018-12-30_) Calculator v.1.7 by cybuit</i>
<i>Sistema Operativo</i>	<i>Windows 10 Home (64bits)</i>
<i>Cracker</i>	<i>QwErTy</i>
<i>Dedicado a</i>	<i>Ángel Zarza - Ricardo Narvaja - CrackSLatinoS</i>
<i>Descargar Crackme</i>	https://mega.nz/#!fIUQWKgJlyfnALEkQkP61gZX5RfI0IJ86FIWDbVBmXYUVr-u7MS8

Este entrañable CrackMe es más que conocido por todos los lectores aficionados al cracking, y quizás para muchos de nosotros fue uno de los primeros con el que empezamos a adentrarnos en el maravilloso arte de la ingeniería inversa.

Con todos mis respetos para el resto de compañeros y maestros que dedican parte de su tiempo a compartir conocimientos con todo el grupo CrackSLatinoS, hoy quiero dedicar muy especialmente este tutorial a ÁNGEL ZARZA, y aprovecho para animarlo a que continúe con su fabuloso y didáctico curso x64dbg desde cero.

Por otra parte, quién sabe si a lo mejor este trabajo puede servir de ayuda a los lectores más nóveles en la comprensión de algunas instrucciones de código ensamblador muy básicas.

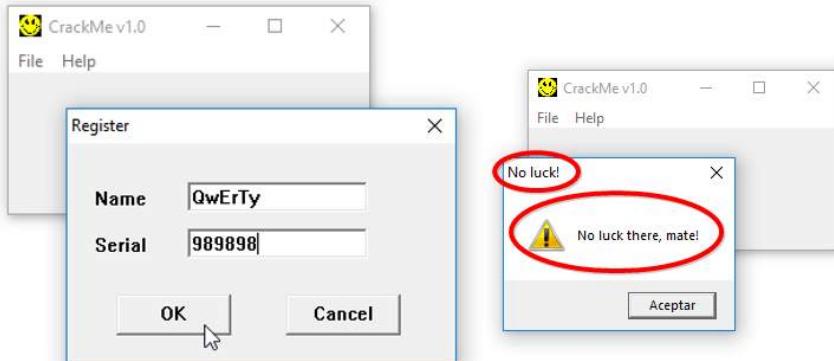
Para los más avanzados en cracking, no esperen encontrar nada nuevo, pero seguro que pasarán un rato entretenido con su lectura, al menos eso espero.

ESTUDIANDO LA VÍCTIMA



Que maja es, que agradable sonrisa, da la impresión de que no tiene malicia ninguna.....vamos a por ella

La ejecutamos y nos pide un "Name" y un "Serial", rellenamos datos, le damos a "OK" para validar y como de costumbreno hemos tenido suerte..... y nos salta el mensaje de "Chico Malo"



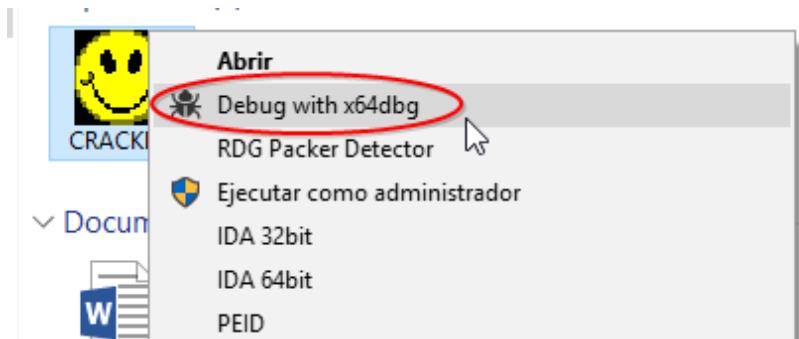
Aceptamos que hemos fallado en nuestro primer intento.

Pasamos los detectores de ejecutables, y nos revelan que estamos ante un compilado en "MASM/TASM", y de la inexistencia de "packed" alguno.



AL ATAQUE

Cargamos el "Crackme", con la tool "x64dbg"



Al detectar que se trata de una aplicación de 32bits, automáticamente ya nos la abre con el "x32dbg", y aparecemos en el "Original Entry Point" (OEP)

Immunity Debugger - PID: F48 - Module: crackme.exe - Thread: Main Thread 22AC x32dbg

File View Debug Trace Plugins Favourites Options Help Dec 30 2018

CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Snowman Handles Trace INFD

EIP [00401000]

Assembly:

```

    . . .
    PUSH 0
    CALL < JMP,AGETModuleHandleA>
    MOV DWORD PTR DS:[4020CA],EAX
    PUSH 0
    PUSH crackme.4020F4
    CALL < JMP,FindWindowA>
    OR EAX,EAX
    XOR EAX,EAX
    CALL crackme.40101D
    RET
    . . .
    C705 64204000 03400000 MOV DWORD PTR DS:[402064],4003
    C705 68204000 28114000 MOV DWORD PTR DS:[402068],<crackme.WndProc>
    C705 6C204000 00000000 MOV DWORD PTR DS:[40206C],0
    C705 70204000 00000000 MOV DWORD PTR DS:[402070],0
    A1 2A204000 00000000 MOV EAX,DWORD PTR DS:[4020CA]
    MOV DWORD PTR DS:[402074],EAX
    A3 74204000 6A 60 PUSH 64
    A3 74204000 50 00 PUSH EAX
    A3 74204000 48 00 CALL < JMP,LoadIconA>
    A3 78204000 40 00 MOV DWORD PTR DS:[402078],EAX
    68 007FF000 PUSH 7E00
    68 007FF000 PUSH 7E00
    A3 72C04000 40 00 CALL < JMP,LoadCursorA>
    MOV DWORD PTR DS:[40207E],EAX
    C705 80204000 05000000 MOV DWORD PTR DS:[402080],5
    C705 82204000 10214000 MOV DWORD PTR DS:[402084],crackme.402110
    C705 84204000 E4204000 MOV DWORD PTR DS:[402088],crackme.4020F4
    68 6A204000 PUSH crackme.4020F4
    . . .
    E8 F3030000 CALL < JMP,RegisterClassA>
    PUSH 0
    . . .
    F7 35 CA204000 PUSH 0
    6A 60 PUSH 0
    68 00 PUSH 0
    68 00 PUSH 8000
  
```

Registers:

EAX	13D171FE
EBX	00220000
ECX	00000000
EDX	00401000
EBP	0019FF94
ESP	0019FF84
ESI	00401000
EDI	00401000
EIP	00401000

Stack:

EFLAGS	00000244
ZF	1 PF 1 AF 0
OF	0 SF 0 DF 0
CF	0 TF 0 IF 1

LastError: 00000000 (ERROR_SUCCESS)

LastStatus: 00000034 (STATUS_OBJECT_NAME_NOT_FOUND)

GS: 0028 FS: 0053

ES: 0028 DS: 0028

CS: 0023 SS: 0028

ST(0) 00000000000000000000000000000000 x87r4 Empty 0.00

ST(1) 00000000000000000000000000000000 x87r4 Empty 0.00

ST(2) 00000000000000000000000000000000 x87r4 Empty 0.00

ST(3) 00000000000000000000000000000000 x87r4 Empty 0.00

ST(4) 00000000000000000000000000000000 x87r4 Empty 0.00

Default (stdcall)

1: [esp+4] 00220000
2: [esp+8] 74268460 <kernel32.BaseThreadInit>
3: [esp+C] 13D171FE
4: [esp+10] 0019FFDC
5: [esp+14] 77723AB8 nt!n...77723AB8

CODE:00401000 crackme.exe:\$1000 #600 <EntryPoint>

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct

Address	Hex	ASCII
776C1000	22 00 24 00 40 77 6C 77	18 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
776C1010	8C 10 40 00 00 00 00 00 00 00 00 00 00 00 00 00	.Tw... .
776C1020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.Tw... .
776C1030	28 00 24 00 08 79 6C 77	34 00 36 00 00 00 00 00 00 00 00 00 00 00 00 00
776C1040	1E 00 20 00 78 6C 77	14 00 1C 00 94 78 6C 77
776C1050	18 00 1A 00 78 6C 77	20 00 22 00 00 00 00 00 00 00 00 00 00 00 00 00
776C1060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. .
776C1070	20 00 22 00 EC 77 6C 77	18 00 1A 00 00 00 00 00 00 00 00 00 00 00 00 00
776C1080	10 00 12 00 9C 77 6C 77	B9 4C 6E 77 90 4B 6E 77
776C1090	80 3A 6E 77 30 3A 6E 77	00 00 00 00 B0 6E 77 B7
776C10A0	30 88 6E 77 20 3A 6E 77	00 00 00 00 B0 6E 77 B7
776C10B0	4C 6E 77 90 4B 6E 77	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Registers:

0019FF88	00220000
0019FF8C	74268460 kernel32._74268460
0019FF94	00000000
0019FF98	00220000
0019FF9C	77723AB8
0019FFA0	00200000
0019FFA4	00000000
0019FFA8	00000000
0019FFAC	00200000
0019FFB0	00000000
0019FFB4	00000000
0019FFB8	00000000

Command: Paused

[xAnalyzer]: Analysis completed in 3.508000 secs

Time Wasted Debugging: 0s

Miramos las "Strings references" y nos centramos en las que corresponden a "**Chico Bueno**" y "**Chico Malo**"

The screenshot shows the x32dbg debugger interface for the CRACKME.EXE binary. The assembly view displays several pushes of the address 402169 onto the stack. The strings view shows various hardcoded strings, with the string "Great work, mate!\rNow try the next crackMe!" highlighted with a green box and the strings "No luck!" highlighted with red boxes.

Address	Disassembly	string
0040100E	push crackme.4020F4	"No need to disasm the code!"
00401077	mov dword ptr ds:[402084],crackme.402110	"MENU"
00401081	mov dword ptr ds:[402088],crackme.4020F4	"No need to disasm the code!"
004010B7	push crackme.4020E7	"CrackMe v1.0"
004010BC	push crackme.4020F4	"No need to disasm the code!"
004011F7	push crackme.40211F	"DLG_ABOUT"
00401213	push crackme.402115	"DLG_PEGTS"
0040134F	push crackme.402129	"Good work!"
00401354	push crackme.402134	"Great work, mate!\rNow try the next crackMe!"
0040136B	push crackme.402160	"No luck!"
00401370	push crackme.402169	"No luck there, mate!"
004013AF	push crackme.402160	"No luck!"
004013B4	push crackme.402169	"No luck there, mate!"

Nos posicionamos sobre la de "Chico Malo", address "00401370", dos Clicks Izq de ratón y apareceremos aquí, donde vemos los dos "MessageBoxA"

The screenshot shows the x32dbg debugger interface with the assembly view active. The assembly pane displays several calls to `JMP crackme.401325`, which corresponds to the UI automation code shown in the right-hand code editor. The code uses `MessageBoxA` to display messages like "Good work!" and "No luck there, mate!". The UI automation code is highlighted with green and red boxes.

```
INT_PTR nResult = NULL
HWND hWnd;
EndDialog

UINT uType = MB_OK | MB_ICONEXCLAMATION | MB_ICONQUESTION | MB_APPLMODAL
LPCSTR lpcaption = "Good work!";
LPCSTR lptext = "Great work, mate! Now try the next CRACKME!";
HWND hWnd;
MessageBoxA

UINT uType = MB_OK
MessageBoxBeep

UINT uType = MB_OK | MB_ICONEXCLAMATION | MB_ICONQUESTION | MB_APPLMODAL
LPCSTR lpcaption = "No luck!";
LPCSTR lptext = "No luck there, mate!";
HWND hWnd;
MessageBoxA
```

Hacemos "scroll" hacia arriba y observamos dos Apis interesantes "GetDlgItemTextA"

```

004012A1:    PUSH 0
004012A2:    PUSH DWORD PTR SS:[EBP+8]
004012A3:    CALL <JMP.&InvalidateRect>
004012A4:    PUSH DWORD PTR SS:[EBP+8]
004012A5:    CALL <JMP.&SetFocus>
004012A6:    JMP crackme.401284
004012A7:    XOR EAX, EAX
004012A8:    CMP DWORD PTR SS:[EBP+10], 3EB
004012A9:    JE crackme.4012F7
004012AC:    CMP DWORD PTR SS:[EBP+10], 3EA
004012B0:    JNE crackme.4012B0
004012B1:    PUSH B
004012B2:    PUSH crackme.40218E
004012B3:    PUSH 3E8
004012B4:    PUSH DWORD PTR SS:[EBP+8]
004012B5:    CALL <JMP.&GetDlgItemTextA>
004012B6:    CMP EAX, 1
004012B7:    MOV DWORD PTR SS:[EBP+10], 3EB
004012B8:    JB crackme.4012A1
004012B9:    PUSH B
004012C0:    PUSH crackme.40217E
004012C1:    PUSH 3E9
004012C2:    PUSH DWORD PTR SS:[EBP+8]
004012C3:    CALL <JMP.&GetDlgItemTextA>
004012C4:    CMP EAX, 1
004012C5:    MOV DWORD PTR SS:[EBP+10], 3EB
004012C6:    JB crackme.4012A1
004012C7:    PUSH B
004012C8:    PUSH crackme.40217E
004012C9:    PUSH 3E9
004012CA:    PUSH DWORD PTR SS:[EBP+8]
004012CB:    CALL <JMP.&GetDlgItemTextA>
004012CC:    CMP EAX, 1
004012CD:    MOV DWORD PTR SS:[EBP+10], 3EB
004012CE:    JB crackme.4012A1
004012CF:    PUSH B
004012D0:    PUSH crackme.40218E
004012D1:    PUSH 3E8
004012D2:    PUSH DWORD PTR SS:[EBP+8]
004012D3:    CALL <JMP.&GetDlgItemTextA>
004012D4:    CMP EAX, 1
004012D5:    MOV DWORD PTR SS:[EBP+10], 3EB
004012D6:    JB crackme.4012A1
004012D7:    PUSH B
004012D8:    PUSH crackme.40217E
004012D9:    PUSH 3E9
004012DA:    PUSH DWORD PTR SS:[EBP+8]
004012DB:    CALL <JMP.&GetDlgItemTextA>
004012DC:    CMP EAX, 1
004012DD:    MOV DWORD PTR SS:[EBP+10], 3EB
004012DE:    JB crackme.4012A1
004012DF:    PUSH B
004012E0:    PUSH crackme.40218E
004012E1:    PUSH 3E8
004012E2:    PUSH DWORD PTR SS:[EBP+8]
004012E3:    CALL <JMP.&GetDlgItemTextA>
004012E4:    CMP EAX, 1
004012E5:    MOV DWORD PTR SS:[EBP+10], 3EB
004012E6:    JB crackme.4012A1
004012E7:    PUSH B
004012E8:    PUSH crackme.40217E
004012E9:    PUSH 3E9
004012EA:    PUSH DWORD PTR SS:[EBP+8]
004012EB:    CALL <JMP.&GetDlgItemTextA>
004012EC:    CMP EAX, 1
004012ED:    MOV DWORD PTR SS:[EBP+10], 3EB
004012EE:    JB crackme.4012A1
004012EF:    PUSH B
004012F0:    PUSH crackme.40218E
004012F1:    PUSH 3E8
004012F2:    PUSH DWORD PTR SS:[EBP+8]
004012F3:    CALL <JMP.&EndDialog>
004012F4:    MOV EAX, 1
004012F5:    MOV EAX, 0
004012F6:    JMP crackme.401284
004012F7:    PUSH EAX
004012F8:    PUSH DWORD PTR SS:[EBP+8]
004012F9:    CALL <JMP.&EndDialog>
004012FA:    MOV EAX, 1

```

Y si continuamos haciendo "scroll" hacia arriba, encontraremos una comparación sospechosa, address "00401241 CMP EAX,EBX" precedida de dos "CALL"

```

00401209:    > 6A 00
0040120B:    PUSH 0
0040120C:    PUSH <Crackme_sub_401253>
0040120D:    PUSH DWORD PTR SS:[EBP+8]
0040120E:    PUSH crackme.402115
0040120F:    PUSH DWORD PTR DS:[4020CA]
00401210:    CALL <KMP.&DialogBoxParamA>
00401211:    CMP EAX, 0
00401212:    JE crackme.4011E6
00401213:    PUSH crackme.40218E
00401214:    CALL <Crackme.sub_40137E>
00401215:    PUSH EAX
00401216:    PUSH crackme.40217E
00401217:    PUSH crackme.402115
00401218:    PUSH DWORD PTR DS:[4020CA]
00401219:    CALL <KMP.&DialogBoxParamA>
0040121A:    CMP EAX, 0
0040121B:    JE crackme.4011E6
0040121C:    PUSH crackme.40218E
0040121D:    CALL <Crackme.sub_4013D8>
0040121E:    ADD ESP, 4
0040121F:    POP EAX
00401220:    CMP EAX, EBX
00401221:    JE crackme.40124C
00401222:    CALL <Crackme.sub_401362>
00401223:    JMP crackme.4011E6
00401224:    CALL <Crackme.sub_40134D>
00401225:    JMP crackme.4011E6
00401226:    ENTER 0, 0
00401227:    PUSH EBX

```

Me posiciono sobre la "CALL" que se encuentra en la address "0040122D", le damos a "Enter" para ver lo que contiene dentro, y de entrada ya vemos algo interesante que nos llama la atención.....je,je,je...

```

0040137E:    $ 887424 04
00401382:    MOV ESI, DWORD PTR SS:[ESI+4]
00401383:    PUSH ESI
00401384:    MOV AL, BYTE PTR DS:[ESI]
00401385:    TEST AL, AL
00401386:    JE crackme.40139C
00401387:    CMP AL, 41
00401388:    JB crackme.4013AC
00401389:    CMP AL, 5A
0040138A:    JA crackme.401394
0040138B:    INC ESI
0040138C:    JMP crackme.401383
0040138D:    CALL <Crackme.sub_4013D2>
0040138E:    INC ESI
0040138F:    JMP crackme.401383
00401390:    POP ESI
00401391:    CALL <Crackme.sub_4013C2>
00401392:    XOR EDI, 5678
00401393:    MOV EAX, EDI
00401394:    JMP crackme.4013C1
00401395:    POP ESI
00401396:    PUSH 30
00401397:    PUSH crackme.402160
00401398:    PUSH crackme.402169
00401399:    PUSH DWORD PTR SS:[EBP+8]
004013A0:    CALL <JMP.&MessageBoxA>
004013A1:    RET
004013A2:    XOR EDI, EDI
004013A3:    PUSH 30
004013A4:    PUSH crackme.402160
004013A5:    PUSH crackme.402169
004013A6:    PUSH DWORD PTR SS:[EBP+8]
004013A7:    CALL <JMP.&MessageBoxA>
004013A8:    RET
004013A9:    PUSH 30
004013AA:    PUSH crackme.402160
004013AB:    PUSH crackme.402169
004013AC:    PUSH DWORD PTR SS:[EBP+8]
004013AD:    CALL <JMP.&MessageBoxA>
004013AE:    RET
004013AF:    PUSH 30
004013B0:    PUSH crackme.402160
004013B1:    PUSH crackme.402169
004013B2:    PUSH DWORD PTR SS:[EBP+8]
004013B3:    CALL <JMP.&MessageBoxA>
004013B4:    RET
004013B5:    PUSH 30
004013B6:    PUSH crackme.402160
004013B7:    PUSH crackme.402169
004013B8:    PUSH DWORD PTR SS:[EBP+8]
004013B9:    CALL <JMP.&MessageBoxA>
004013BA:    RET
004013C1:    C3
004013C2:    $ 33FF

```

pero antes de estudiar con detalle las instrucciones que estamos viendo, salimos de las entrañas de este "CALL" dándole a "Click derecho de ratón>Go to>Previous"

Assembly code at address 0040137E:

```

MOV ESI, DWORD PTR SS:[ESP+4]
PUSH ESI
MOV AL, BYTE PTR DS:[ESI]
TEST AL, AL
JE crackme.40139C
CMP AL, 41
JB crackme.4013AC
CMP AL, 5A
JG crackme.401394
INC ESI
JMP crackme.401383
CALL <crackme.sub_4013D2>
INC ESI
JMP crackme.401383
POP ESI
CALL <crackme.sub_4013C2>
XOR EDI, EDI
MOV EAX, EDI
JMP crackme.4013C1
POP ESI
PUSH 30
PUSH crackme.402160
PUSH crackme.402169
PUSH DWORD PTR SS:[EBP-8]
CALL <JMP.&MessageBoxA>
XOR EDI, EDI
XOR EBX, EBX
MOV BL, BYTE PTR DS:[ESI]
TEST BL, BL
JB crackme.401301
ADD EDI, EBX
INC ESI

```

Registers:

- EAX: 0040137E
- ECX: 0040137E
- EDX: 00401370
- EDI: 00401370
- EBX: 00401370
- ESP: 00401370
- EBP: 00401370
- CS: 0002
- DS: 0002
- SS: 0002
- ES: 0002
- FS: 0002
- GS: 0002

Stack dump (00401370-004013C0):

Address	Hex	ASCII
776C1000	22 00 24 00	40 77 6C 77 18 00 00 00 00 00 00 00 00 00 00 00 00 00
776C1010	8C 17 6C 77	40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
776C1020	2A 00 2C 00	64 77 6C 77 1C 00 01 E0 34 79 6C 77 *., dwl,...,4ylw
776C1030	2B 00 2A 00	08 79 6C 77 34 00 36 00 D0 78 6C 77 (. *.ylw. 6.Dxlw
776C1040	1E 00 20 00	B0 78 6C 77 1A 00 01 C0 94 78 6C 77 .*, xlw... , xlw
776C1050	18 00 1A 00	78 78 6C 77 20 00 22 00 54 78 6C 77 .*, xlw... , xlw
776C1060	30 00 32 00	20 78 6C 77 2C 00 02 E0 F0 77 6C 77 0, 2, xlw... , xlw
776C1070	20 00 22 00	CC 77 6C 77 18 00 1A 00 B0 77 6C 77 .*.lw... ,wlw
776C1080	10 00 12 00	9C 77 6C 77 B0 4C 6E 77 90 4E 6E 77 .*,lw!Lw, Nnw
776C1090	B0 3A 6E 77	70 3A 6E 77 00 00 00 00 B0 GF / B7 77 :nwp:nw... ,o{w
776C10A0	30 88 6E 77	70 3A 6E 77 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Y le ponemos un "BP"

Assembly code at address 00401238:

```

CMP EAX, 0
JE crackme.4011E6
PUSH crackme.40218E
CALL <crackme.sub_40137E>
PUSH EAX
PUSH crackme.40217E
CALL <crackme.sub_4013D8>
ADD ESP, 4
POP EAX
CMP EAX, EBX
JE crackme.40124C
CALL <crackme.sub_401362>
JMP crackme.4011E6
CALL <crackme.sub_40134D>
JMP crackme.4011E6
ENTER 0,0

```

Registers:

- EAX: 00401238
- ECX: 00401238
- EDX: 00401230
- EDI: 00401230
- EBX: 00401230
- ESP: 00401230
- EBP: 00401230
- CS: 0002
- DS: 0002
- SS: 0002
- ES: 0002
- FS: 0002
- GS: 0002

Ahora, miramos en la segunda "CALL" que se encuentra en la address "00401238", antes de la comparación "CMP EAX,EBX", nos posicionamos sobre ella

Assembly code at address 00401238:

```

CMP EAX, 0
JE crackme.4011E6
PUSH crackme.40218E
CALL <crackme.sub_40137E>
PUSH EAX
PUSH crackme.40217E
CALL <crackme.sub_4013D8>
ADD ESP, 4
POP EAX
CMP EAX, EBX
JE crackme.40124C
CALL <crackme.sub_401362>
JMP crackme.4011E6
CALL <crackme.sub_40134D>
JMP crackme.4011E6
ENTER 0,0

```

Registers:

- EAX: 00401238
- ECX: 00401238
- EDX: 00401230
- EDI: 00401230
- EBX: 00401230
- ESP: 00401230
- EBP: 00401230
- CS: 0002
- DS: 0002
- SS: 0002
- ES: 0002
- FS: 0002
- GS: 0002

nuevamente le damos a "Enter" para ver que parte de código esconde en su interior, y también intuimos que va a hacer otros cálculos que seguro que nos interesará saber antes de llegar a la comparación.

```

004013D8: $ 33C0 XOR EAX,EAX
    . 33FF XOR EDI,EDI
    . 33DB XOR EBX,EBX
    . 8B7424 04 MOV ESI,DWORD PTR SS:[ESP+4]
    . B0 0A MOV AL,A
    . 8A1E MOV BL,BYTE PTR DS:[ESI]
    . 84DB TEST BL,BL
    > 74 0B JE crackme.4013F5
    . 80EB 30 SUB BL,30
    . 0FAFF8 IMUL EDI,EAX
    . 03FB ADD EDI,EBX
    . 46 INC ESI
    ^ EB ED JMP crackme.4013E2
    > 81F7 34120000 XOR EDI,1234
    . 8BDF MOV EBX,EDI
    C3 RET
    . FF25 84314000 JMP DWORD PTR DS:<&KillTimer>
    . FF25 88314000 JMP DWORD PTR DS:<&GetSystemMetrics>
    $ FF25 8C314000 JMP DWORD PTR DS:<&LoadCursorA>
    $ FF25 90314000 JMP DWORD PTR DS:<&LoadAcceleratorsA>
    $ FF25 94314000 JMP DWORD PTR DS:<&MessageBeep>
    $ FF25 98314000 JMP DWORD PTR DS:<&GetWindowRect>
    > FF25 9C314000 JMP DWORD PTR DS:<&LoadStringA>

```

Bien, con que ya sabemos cómo salir de esta "CALL", (recordemos que todavía no hemos ejecutado el crackme, y simplemente estamos curioseando), salimos, le ponemos otro "BP" y nos queda así:

```

00401223: . 83F8 00 CMP EAX,0
    . ^ 74 BE JE crackme.4011E6
    . 68 8E214000 PUSH crackme.40218E
    . E8 4C010000 CALL <crackme.sub_40137E>
    . 50
    . 68 7E214000 PUSH EAX
    . E8 98010000 PUSH crackme.40217E
    > 68 9B010000 CALL <crackme.sub_4013D8>
    . 83C4 04 ADD ESP,4
    . 58 POP EAX
    . 3BC3 CMP EAX,EBX
    . ^ 74 07 JE crackme.40124C
    . E8 18010000 CALL <crackme.sub_401362>
    . ^ EB 9A JMP crackme.4011E6
    > E8 FC000000 CALL <crackme.sub_40134D>
    . ^ EB 93 JMP crackme.4011E6
    F. C8 0000 00 ENTER 0,0

```

Con nuestros dos "BP" colocados, damos "Run" para que arranque el Crackme

```

00401223: . 83F8 00 CMP EAX,0
    . ^ 74 BE JE crackme.4011E6
    . 68 8E214000 PUSH crackme.40218E
    . E8 4C010000 CALL <crackme.sub_40137E>
    . 50
    . 68 7E214000 PUSH EAX
    . E8 98010000 PUSH crackme.40217E
    > 68 9B010000 CALL <crackme.sub_4013D8>
    . 83C4 04 ADD ESP,4
    . 58 POP EAX
    . 3BC3 CMP EAX,EBX
    . ^ 74 07 JE crackme.40124C

```

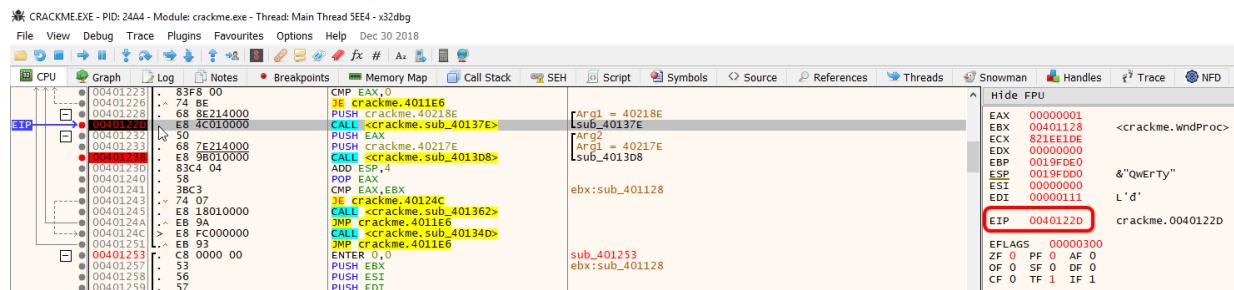
Rellenamos datos:

Register

Name	OwErTy
Serial	989898

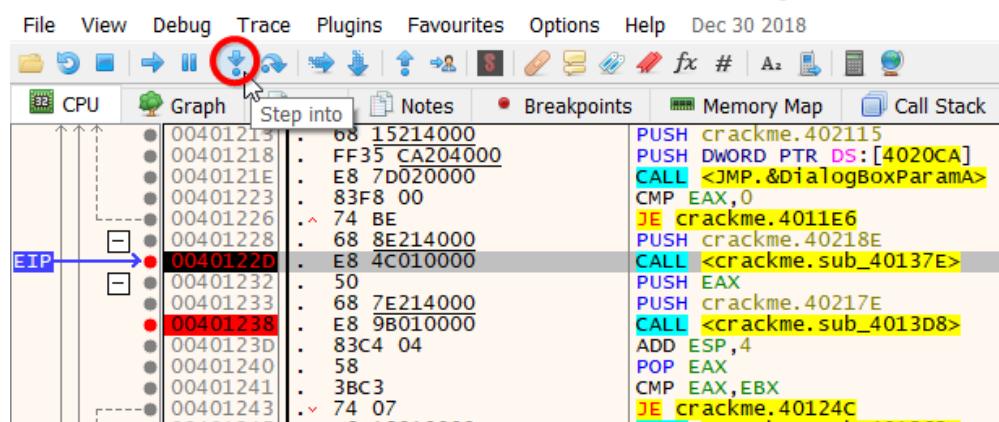
OK Cancel

Le damos a "OK" y nos encontramos en ejecución, parados en nuestro primer "BP"



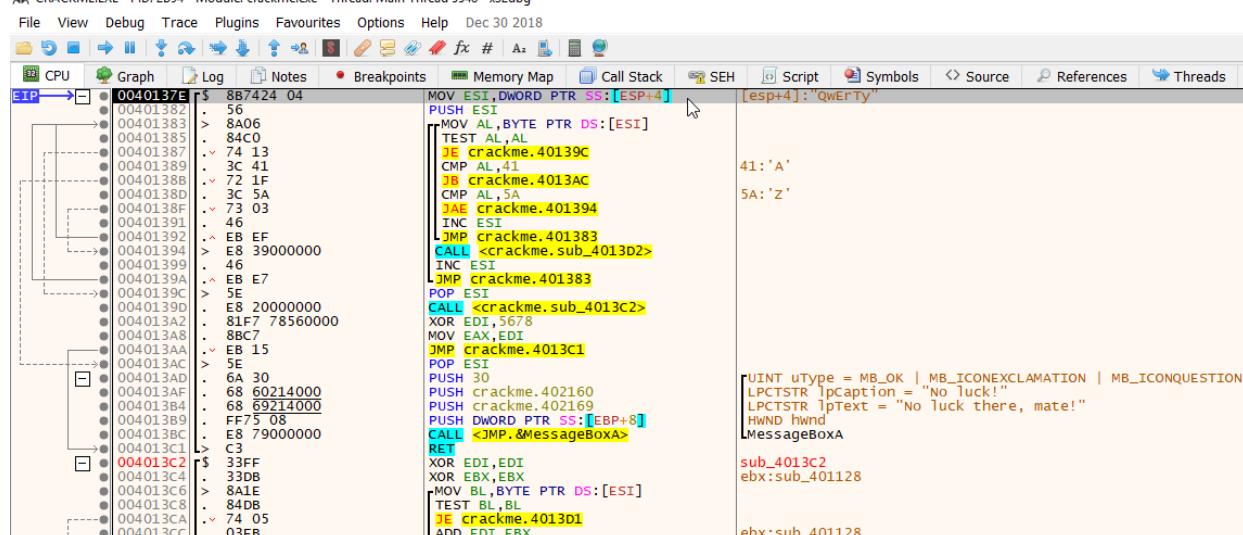
Entramos dentro del "CALL", esta vez con "Step into" o "F7"

CRACKME.EXE - PID: 2B94 - Module: crackme.exe - Thread: Main Thread 5548 - x32dbg



Vemos que esta llamada se inicia en la address "0040137E" y termina en el "RET", address "004013C1"

CRACKME.EXE - PID: 2B94 - Module: crackme.exe - Thread: Main Thread 5548 - x32dbg



Ahora vamos a interpretar las instrucciones que se van a ejecutar, cuyo comentario he agregado antes del trazeo, para facilitar su comprensión.

CRACKME.EXE - PID: 2808 - Module: crackme.exe - Thread: Main Thread 8A4 - x32dbg

File View Debug Trace Plugins Favourites Options Help Dec 30 2018

CPU

```

0040137E $ 887424 04
    MOV ESI,DWORD PTR SS:[ESP+4]
    PUSH ESI
    MOV AL,BYTE PTR DS:[ESI]
    TEST AL,AL
    JZ crackme.40139C
    CMP AL,5A
    JAE crackme.401394
    INC ESI
    JMP crackme.401383
    CALL <crackme.sub_4013D2>
    POP ESI
    JMP crackme.401383
    POP ESI
    CALL <crackme.sub_4013C2>
    XOR EDI,5678
    MOV EAX,EDI
    JMP crackme.4013C1
    POP ESI
    PUSH 30
    PUSH crackme.402160
    PUSH crackme.402169
    PUSH DWORD PTR SS:[EBP+8]
    CALL <JMP.&MessageBoxA>
    RET
004013C2 $ 33FF
    XOR EDI,EDI
    XOR EBX,EBX
    MOV BL,BYTE PTR DS:[ESI]
    TEST BL,BL
    JZ crackme.4013D1
    ADD EDI,EBX
    INC ESI
    JMP crackme.4013C6
    RET
004013D1 C3

```

es1=0
 DWORD PTR [ESP+4]=[0019FDD0 &"QwErTy"] =0040218E "QwErTy"
 Call from WindProc+105
 CODE:0040137E crackme.exe:\$137E #97E <sub_40137E>

Bien, ahora iremos traceando paso a paso y comprobando los resultados.

Empecemos....

MOV ESI,DWORD PTR SS:[ESP+4]

Va a mover al Registro "ESI" el contenido que se guarda en [ESP+4],

[ESP+4] = "QwErTy"

CRACKME.EXE - PID: 2590 - Module: crackme.exe - Thread: Main Thread B84 - x32dbg

File View Debug Trace Plugins Favourites Options Help Dec 30 2018

CPU

```

0040137E $ 887424 04
    MOV ESI,DWORD PTR SS:[ESP+4]
    PUSH ESI
    MOV AL,BYTE PTR DS:[ESI]
    TEST AL,AL
    JZ crackme.40139C
    CMP AL,5A
    JAE crackme.401394
    INC ESI
    JMP crackme.401383
    CALL <crackme.sub_4013D2>
    POP ESI
    JMP crackme.401383
    POP ESI
    CALL <crackme.sub_4013C2>
    XOR EDI,5678
    MOV EAX,EDI
    JMP crackme.4013C1
    POP ESI
    PUSH 30
    PUSH crackme.402160
    PUSH crackme.402169
    PUSH DWORD PTR SS:[EBP+8]
    CALL <JMP.&MessageBoxA>
    RET
004013C2 $ 33FF
    XOR EDI,EDI
    XOR EBX,EBX
    MOV BL,BYTE PTR DS:[ESI]
    TEST BL,BL
    JZ crackme.4013D1
    ADD EDI,EBX
    INC ESI
    JMP crackme.4013C6
    RET
004013D1 C3

```

es1=0
 DWORD PTR [ESP+4]=[0019FDD0 &"QwErTy"] =0040218E "QwErTy"
 Call from WindProc+105
 CODE:0040137E crackme.exe:\$137E #97E <sub_40137E>

EAX 00000001 <> crackme.wndProc
 EBX 00401128
 ECX 25158F27
 EDX 00000000
 EDI 00000000
 ESP 0019E9C0 &"Ph...ia"
 ESI 0040218E "QwErTy"
 EDI 00000011 L d
 EIP 00401382 crackme.00401382
 EFLAGS 00000020
 ZF 0 PF 0 AF 0
 OF 0 SF 0 DF 0
 CF 0 TF 0 IF 1
 LastError 00000000 (ERROR_SUCCESS)
 SetLastError C0000034 (STATUS_OBJECT_N
 GS 0028 FS 0053
 ES 0028 DS 0028
 CS 0023 > 0028
 ST(0) 00000000000000000000000000000000 x87r0 Emp
 ST(1) 00000000000000000000000000000000 x87r1 Emp
 ST(2) 00000000000000000000000000000000 x87r2 Emp
 ST(3) 00000000000000000000000000000000 x87r3 Emp
 ST(4) 3FFF8000000000000000000000000000 x87r4 Emp
 ST(5) 3FEF8000000000000000000000000000 x87r5 Emp

PUSH ESI

Va a meter el valor de "ESI" (nuestro Name tipeado) al "Stack" (primera carta del mazo)

MOV AL,BYTE PTR DS:[ESI]

Mueve a "AL" el primer byte del contenido de "ESI" si "ESI" = QwErTy el primer byte será 51h = "Q"

TEST AL, AL

Testea AL, AL por tanto en nuestro caso $AL = 51h = "Q"$

con que el resultado del Testeo entre AL,AL es 51h = "Q" el salto condicional "**JE**"

Por el contrario si en "AL" tuviéramos el valor "00" el Flag ZF levantaría bandera con el valor 1 y el salto condicional "**JE**" nos llevaría a la address "0040139C" que es la dirección de "main".

(Eso quedaré cuando ya no queden más caracteres para leer de nuestro Name típico.)

```

MOV_ESI_DWORD_PTR_SS:[ESP+4] [esp+4]: "Qwerty"
PUSH_ESI
MOV_AL_BYT_PTR_DS:[ESI]
TEST_AL_AL
JMP crackme.4013AC
    
```

CMP AL, 41

Aquí va a comparar 51h "Q" con 41h "A"

JB crackme.4013AC

Y ahora con el "**JB**" va a decidir que si el resultado de la comparación anterior es menor, saltaremos a la address "004013AC" donde vemos que se trata de una de las zonas de "**Chico malo**". Si por el contrario el resultado de la comparación es mayor, entonces nos dejará continuar.

En otras palabras:

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 (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, pues traceamos y seguimos

```

MOV_ESI_DWORD_PTR_SS:[ESP+4] [esp+4]: "Qwerty"
PUSH_ESI
MOV_AL_BYT_PTR_DS:[ESI]
TEST_AL_AL
JMP crackme.4013AC
    
```

Chico Malo

CMP AL, 5A

Ahora va a comparar 51h "Q" con 5Ah "Z"

JAE crackme.401394

Con el “**JAE**” va a decidir que si el resultado de la comparación anterior es mayor o igual, saltaremos a la address “00401394” donde vemos que se trata de otra llamada “**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 la address "00401394" donde se encuentra otra "CALL" cuando encuentre una letra minúscula. (en nuestro Name tipeado, esto sucederá cuando haga la comparación con los caracteres "w,r,y")

INC ESI

Esta instrucción va a INCREMENTAR en "1h" el valor del registro "ESI"

Como podemos observar en la imagen, el valor de "ESI" es "0040218E"

	EAX	00000051
)	EBX	00401128
	ECX	BB16B7C0
	EDX	00000000
	EBP	0019FDE0
	ESP	0019FDC8
	ESI	0040218E
	EDI	00000011

Ejecutamos la instrucción y ahora ESI vale 0040218F (0040218E+1)

Lo que ha hecho es que si antes de ejecutar la instrucción, en "ESI" había guardado "QwErTy", una vez ejecutada, al sumarle 1h pues pasa al siguiente byte, quedando en "wErTy"

```

    MOV ESI, DWORD PTR SS:[ESP+4] [esp+4;"QwErTy"] ----> Mueve a ESI nuestro Name tipeado
    es1:"QwErTy" -----> Mueve a AL el primer Byte del contenido de ESI
    es1:"QwErTy" -----> Testea AL con AL
    es1:"QwErTy" -----> Resultado del testeo AL,AL es 0 salta a 40139C (sal del Loop)
    41;"A" -----> Compara 41h con el valor Hex de AL
    -----> Salta a 4013AC si el resultado de la CMP es menor
    SA;"Z" -----> Compara SAh con el valor de AL
    -----> Salta a 00401383 si el resultado de la CMP es mayor o igual
    es1:"wErTy" -----> Pasa al siguiente byte de ESI
    es1:"wErTy" -----> Salta a 00401383 (Inicio del loop)
    es1:"wErTy" -----> convierte minúsculas a Mayúsculas
    es1:"wErTy" -----> Salta a 00401383 (Inicio del loop)
    es1:"wErTy" -----> Pasa la primera carta del stack a ESI (Nuestro Name convertido a MAYÚSCULAS)
    -----> Suma los bytes (hex) de nuestro Name y el resultado lo guarda en EDI
    -----> Corre el contenido de EDI con 5678 y guarda el resultado en EDI
    -----> Mueve el valor de EDI a EAX
    -----> Mueve el valor de EDI a EDI
    INC ESI
    JMP crackme.401383
    INC ESI
    CALL crackme_sub_4013D2
    INC ESI
    JMP crackme.401383
    POP ESI
    INC ESI
    CALL crackme_sub_4013C2>
    INC ESI
    XOR EDI, EDI
    MOV EAX, EDI
    JMP crackme.4013C1

```

JMP crackme.401383

Ahora el salto incondicional "JMP" nos va a llevar directos a la address 00401383 (inicio del "Loop1").

Ejecutamos la instrucción y aquí estamos de nuevo, con la diferencia que va hacer todas las comprobaciones realizadas hasta el momento con el segundo byte de nuestro Name tipeado, por lo que ahora le toca el turno a "w" (minúscula)

```

    MOV ESI, DWORD PTR SS:[ESP+4] [esp+4;"QwErTy"] ----> Mueve a ESI nuestro Name tipeado
    es1:"QwErTy" -----> Mueve a AL el primer Byte del contenido de ESI
    es1:"QwErTy" -----> Testea AL con AL
    es1:"QwErTy" -----> Resultado del testeo AL,AL es 0 salta a 40139C (sal del Loop)
    41;"A" -----> Compara 41h con el valor Hex de AL
    -----> Salta a 4013AC si el resultado de la CMP es menor
    SA;"Z" -----> Compara SAh con el valor de AL
    -----> Salta a 00401383 si el resultado de la CMP es mayor o igual
    es1:"wErTy" -----> Pasa al siguiente byte de ESI
    es1:"wErTy" -----> Salta a 00401383 (Inicio del loop)
    es1:"wErTy" -----> convierte minúsculas a Mayúsculas
    es1:"wErTy" -----> Salta a 00401383 (Inicio del loop)
    es1:"wErTy" -----> Pasa la primera carta del stack a ESI (Nuestro Name convertido a MAYÚSCULAS)
    -----> Suma los bytes (hex) de nuestro Name y el resultado lo guarda en EDI
    -----> Corre el contenido de EDI con 5678 y guarda el resultado en EDI
    -----> Mueve el valor de EDI a EAX
    -----> Mueve el valor de EDI a EDI
    INC ESI
    CALL crackme_sub_4013D2
    INC ESI
    JMP crackme.401383
    INC ESI
    CALL crackme_sub_4013C2>
    INC ESI
    XOR EDI, EDI
    MOV EAX, EDI
    JMP crackme.4013C1

```

MOV AL, BYTE PTR DS:[ESI]

Mueve a "AL" el primer Byte del contenido de "ESI"
si ahora "ESI" = "wErTy" el primer byte será "77h" = "w"

Vamos a comprobarlo

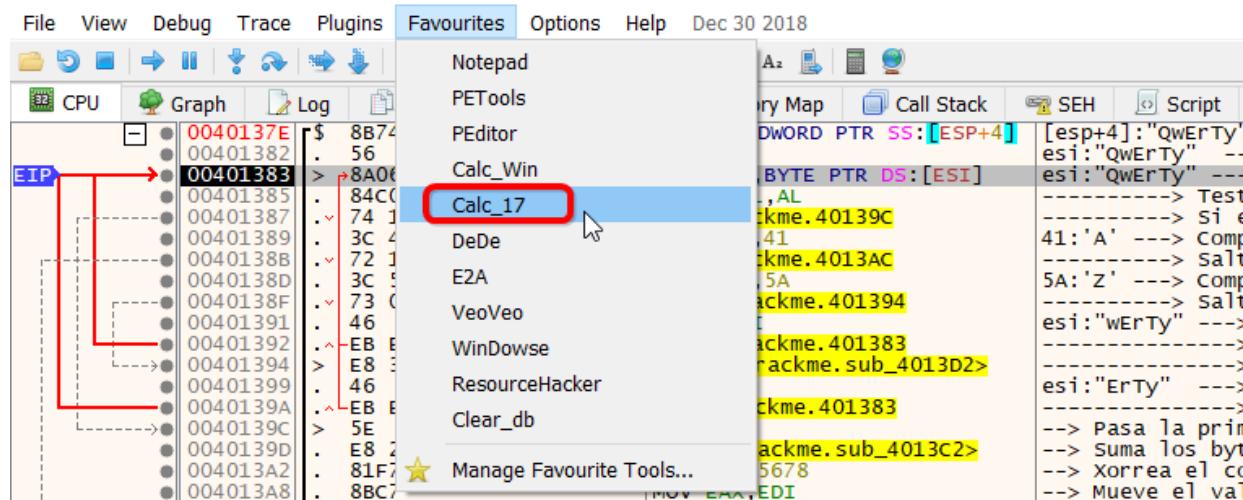
Abrimos la calculadora que trae por defecto X64dbg, y en ASCII tipeamos la "w" obteniendo el resultado de conversión en diferentes formatos, de los cuales y lógicamente el que nos interesa ahora es el Hexadecimal "77h"

Calculator window:

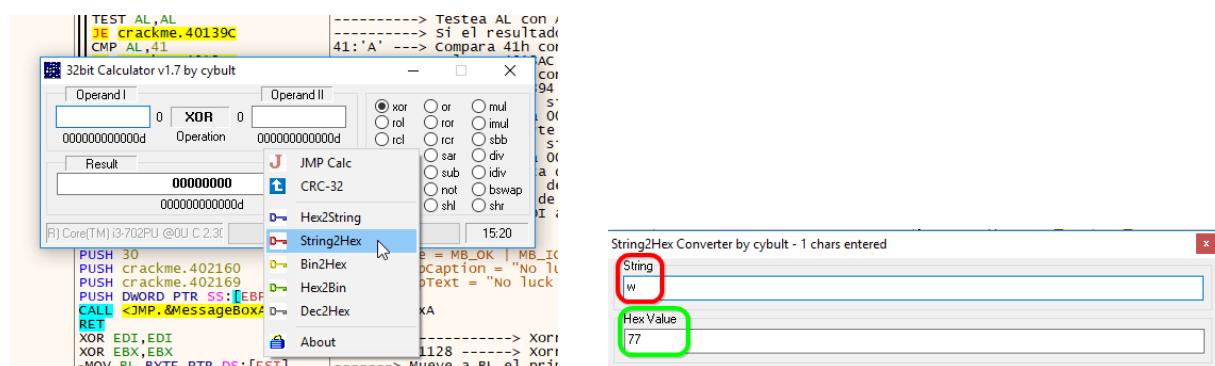
Expression:	77
Hexadecimal:	77
Signed:	119
Unsigned:	119
Octal:	167
Binary:	0000 0000 0000 0000 0000 0111 0111
ASCII:	w
Unicode:	w

También lo podemos hacer con la pequeña y para mí muy entrañable tool "Calc_17" que de momento veo que también funciona bien con Windows 10 Home (aunque solo sirve para calcular o convertir valores para aplicaciones de 32 bits, y aprovecho para decir que el convertidor "Hex2String" que lleva integrado no funciona).

CRACKME.EXE - PID: 5100 - Module: crackme.exe - Thread: Main Thread 4B2C - x32dbg



La abrimos, damos Click derecho de ratón sobre ella y escogemos la conversión "String2Hex", tipeamos la "String" a convertir y obtenemos el mismo resultado

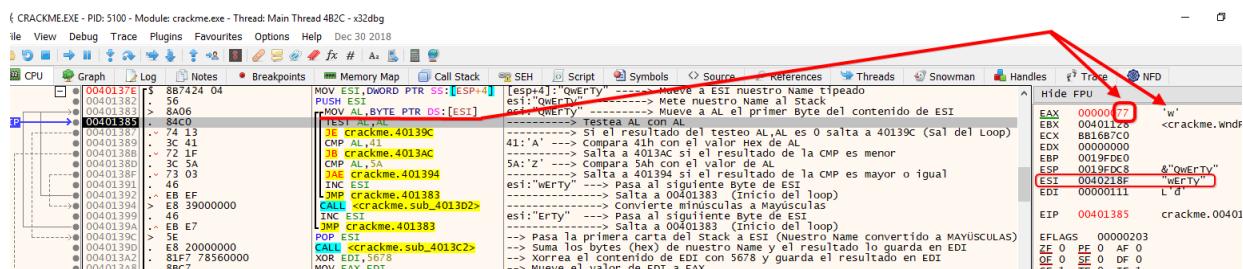


Sigamos....

Una vez ejecutada la instrucción

MOV AL, BYTE PTR DS:[ESI]

Efectivamente ha movido a "AL" el primer byte del contenido de "ESI"



TEST AL,AL

Testea AL,AL por tanto ahora AL = 77h = "w"

```

MOV ESI, DWORD PTR SS:[ESP+4] [esp+4]: "QwErTy" -----> Mueve a ESI nuestro Name tipiado
PUSH ESI
MOV AL, BYTE PTR DS:[ESI]
TEST AL, AL
JE crackme.40139C
CMP AL, 41
JB crackme.4013AC
JAE crackme.401394
INC ESI
JMP crackme.401383

```

Registers (right side):

	EAX	EBX	ECX	EDX	EBP	ESP	ESI	EDI
	00000077	00401128	BB16B7C0	00000000	0019FDE0	0019FDC8	"QwErTy"	L'd'
	'w'							

JE crackme.40139C

Con que el resultado del Testeo entre AL,AL es 77h = "w" , el salto condicional "**JE**" nos deja continuar.

CMP AL,41

Compara 77h "w" con 41h "A"

JB crackme.4013AC

Con que el resultado de la comparación (77h es mayor que 41h) entonces el "**JB**" nos deja continuar

CMP AL,5A

Compara 77h "w" con 5Ah "Z"

```

004013E: 8B42 04          ; mov eax, [esi+4]
004013F: 48                ; push rax
004013F: 8406              ; test al, al
004013F: 4C                ; push rbp
004013F: 84C0              ; mov rbp, rbp
004013F: 74 13              ; je crackme.40139C
004013F: 3C 41              ; cmp al, 41h
004013F: 73 03              ; jbe crackme.401394
004013F: 46                ; inc esi
004013F: EB EF              ; jmp crackme.401383
004013F: 39 00000000        ; call crackme.sub_4013D2
004013F: 46                ; inc esi
004013F: EB E7              ; jmp crackme.401383

```

Registers (right side):

	EAX	EBX	ECX	EDX	EBP	ESP	ESI	EDI	EIP
	00000077	00401128	BB16B7C0	00000000	0019FDE0	0019FDC8	"QwErTy"	L'd'	0040138D
	'w'								crackme.004013

JAE crackme.401394

Y con el "**JAE**" 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 la address "00401394"

Recordemos que:

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

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

Está claro que aquí, siempre que encuentre una letra minúscula (o cualquier otro carácter que esté por encima de 5Ah) nos mandará directos a la address "00401394" donde se encuentra la "**CALL**".

Pues en nuestro caso ya sabemos que iremos directos al "**CALL**" y así nos lo avanza X32dbg

Bien, un solo traceo más, y efectivamente nos encontramos parados en la address "00401394 CALL <crackme.sub 4013D2>"

Nos posicionamos con el cursor sobre la “CALL” Entramos con “Intro” para ver que se va a cocer ahí a dentro, y aparecemos aquí, en la address “004013D2”

Lo que va a hacer con la instrucción "SUB AL,20" es restar el segundo operando al primero, y guardar el resultado en el primer operando.

Con lo cual 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"

The screenshot shows the Immunity Debugger interface with assembly code and a calculator window.

Assembly View:

```
004013A2 . 81F7 78560000 XOR EDI,5678
004013A8 . 8BC7 MOV ECX,ED
004013AA . EB 15 Jmp 32bit Calculator v1.7 by cybult
004013AC > 5E
004013AD . 6A 30 XOR ECX,ECX
004013AF . 68 60 SUB ECX,40
004013B4 . 68 69 SUB ECX,5
004013B9 . FF75 C0 SUB ECX,[ECX]
004013BC . E8 79 00 00 CALL sub_4013C6
004013C1 > C3 HLT
004013C2 $ 33FF ADD ECX,ECX
004013C4 . 33DB ADD ECX,1
004013C6 > 8A1E MOV AL,[ECX]
004013C8 . 84DB TEST AL,AL
004013CA . 74 05 JNE 4013D2
004013CC . 03FB ADD ECX,ECX
004013CE . 46 INC ECX
004013CF . EB F5 JMP crackme.4013C6
004013D1 > C3 HLT
004013D2 $ 2C 20 SUB AL,20
004013D4 . 8806 MOV BYTE PTR DS:[ESI],AL
004013D6 . C3 RET
```

Calculator Window:

32bit Calculator v1.7 by cybult

Operand I	77	2	SUB	2	Operand II
0000000000119d			Operation	0000000000032d	
Result			00000057		
			000000000087d		

Right Panel:

- xor or mul
- rol ror imul
- rcl rcr sbb
- adc sar div
- add sub idiv
- neg bswap
- and shr

Core(TM) i3-702PU @ 2.3GHz 19:20

IL28 -----> Jueve a BL el resultado
esteña BL con el resultado
i el resultado
suma el valor
Pasa al siguiente
Salta a 004013D2

Pues ya podemos apuntar el siguiente comentario

```
TEST BL,BL  
JE crackme.4013D1  
ADD EDI,EBX  
INC ESI  
JMP crackme.4013C6  
RET  
SUB AL,20  
MOV BYTE PTR DS:[ESI],AL  
RET  
RET  
XOR EAX,EAX
```

-----> Testea BL con BL
-----> Si el resultado del testeo de BL,BL es 0 salta
-----> Suma el valor de EBX v EDI v quárdalo en EDI

Add comment at 004013D2 X

-----> Restará 20h al valor de AL y el resultado lo guarda en AL

OK Cancel

En la próxima instrucción que es "MOV BYTE PTR DS:[ESI], AL" lo que va a hacer es MOVer el valor de "AL" (ahora es "57h" = "W" mayúscula), y va a guardar/substituir ese byte en el contenido de "ESI"

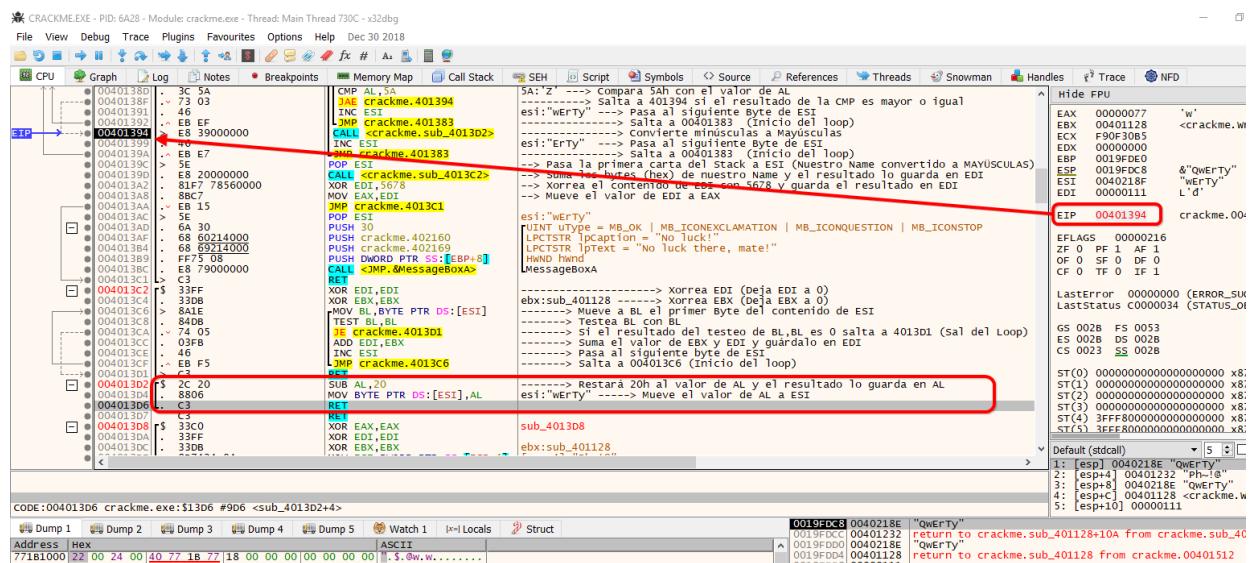
Y agregamos el comentario

esi:"wErTy" -----> Mueve el valor de AL a ESI

-----> Restará 20h al valor de AL y el resultado lo guarda en AL

Y nos queda así

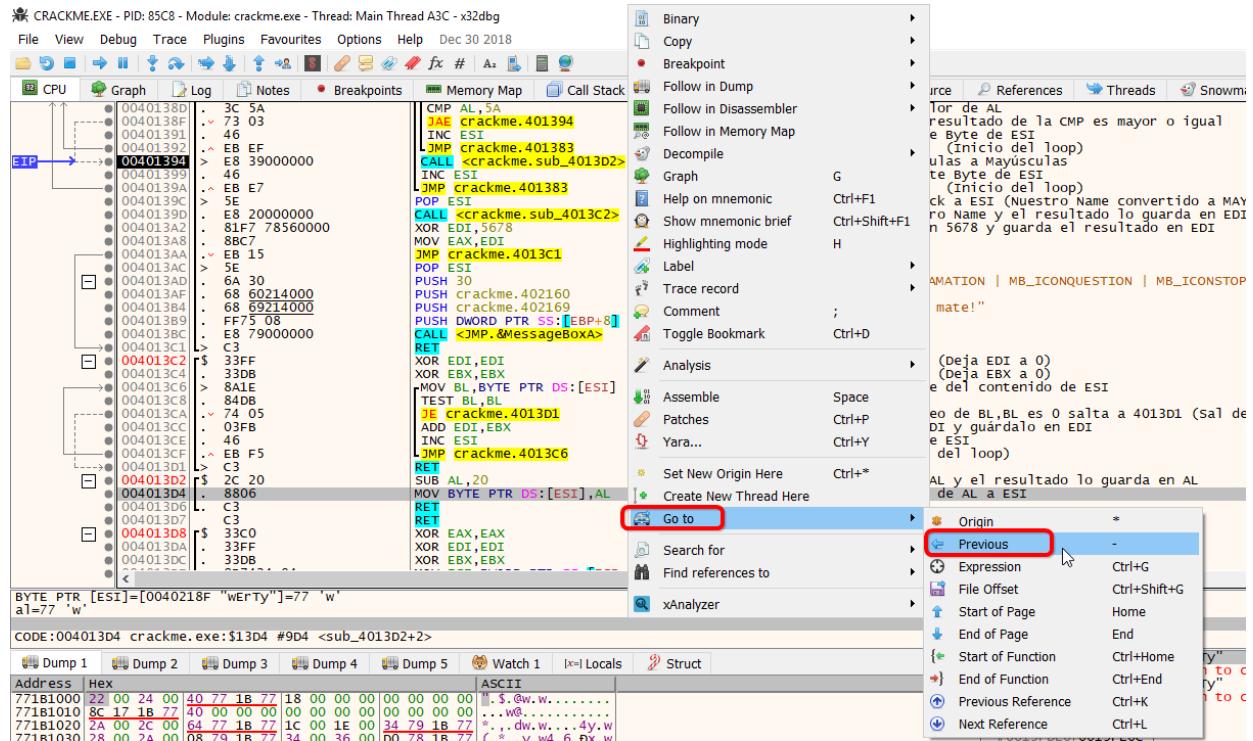
Recordemos que solo hemos entrado en este "CALL" con "Intro" para inspeccionar y descifrar código de lo que va a hacer su contenido, ya que en realidad en el desensamblador estamos parados en ejecución, en la address "00401394", que es la puerta de la propia "CALL"



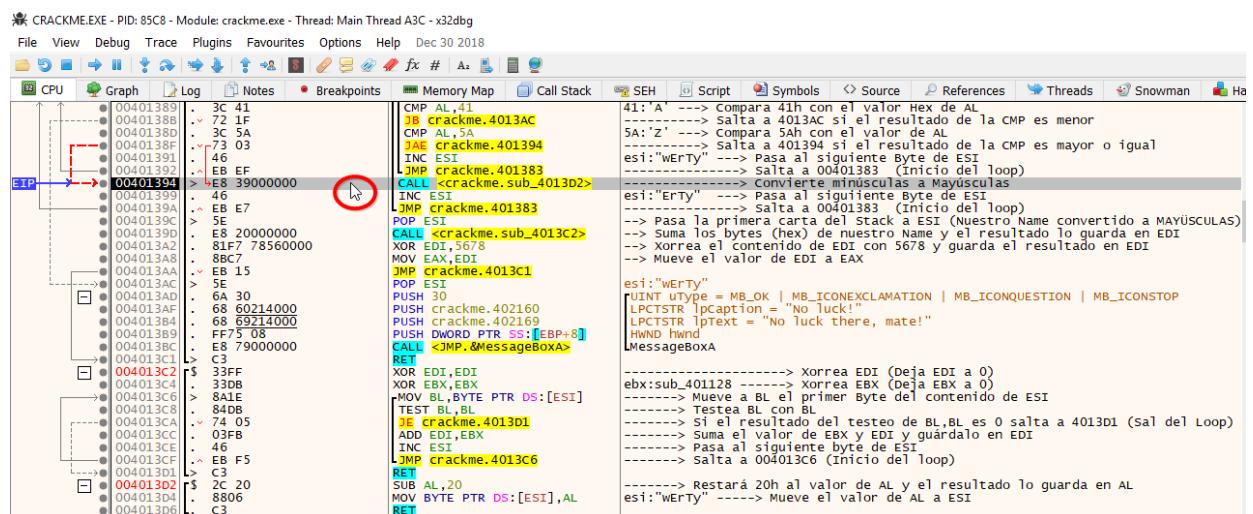
Con que la siguiente instrucción address "004013D6" es un "RET" , ja sabemos que cuando la pasemos en ejecución, su función será sacarnos de las entrañas de la "CALL".

Pues volvamos a la puerta de la "CALL".

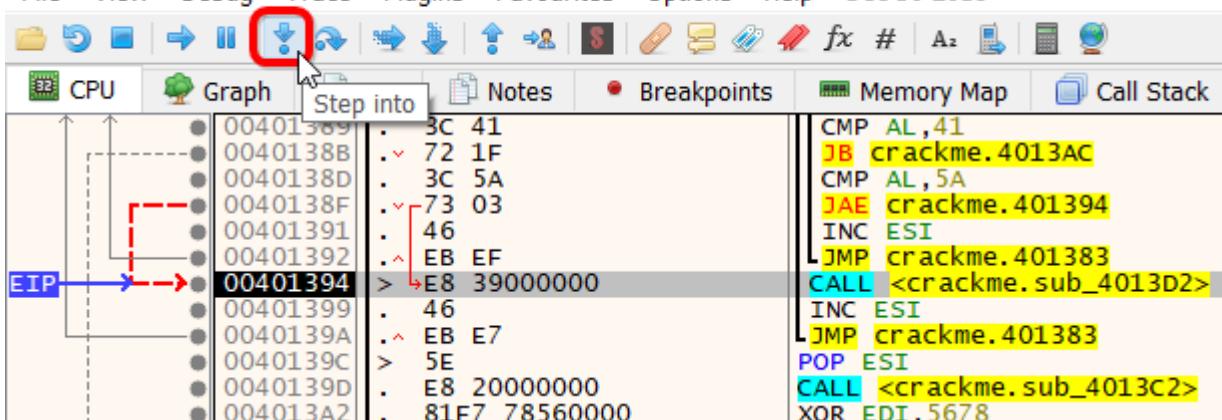
Nos posicionamos con el cursor en cualquier address del interior de la "CALL" en la que estamos, Click Derecho de ratón y "Go to > Previous"



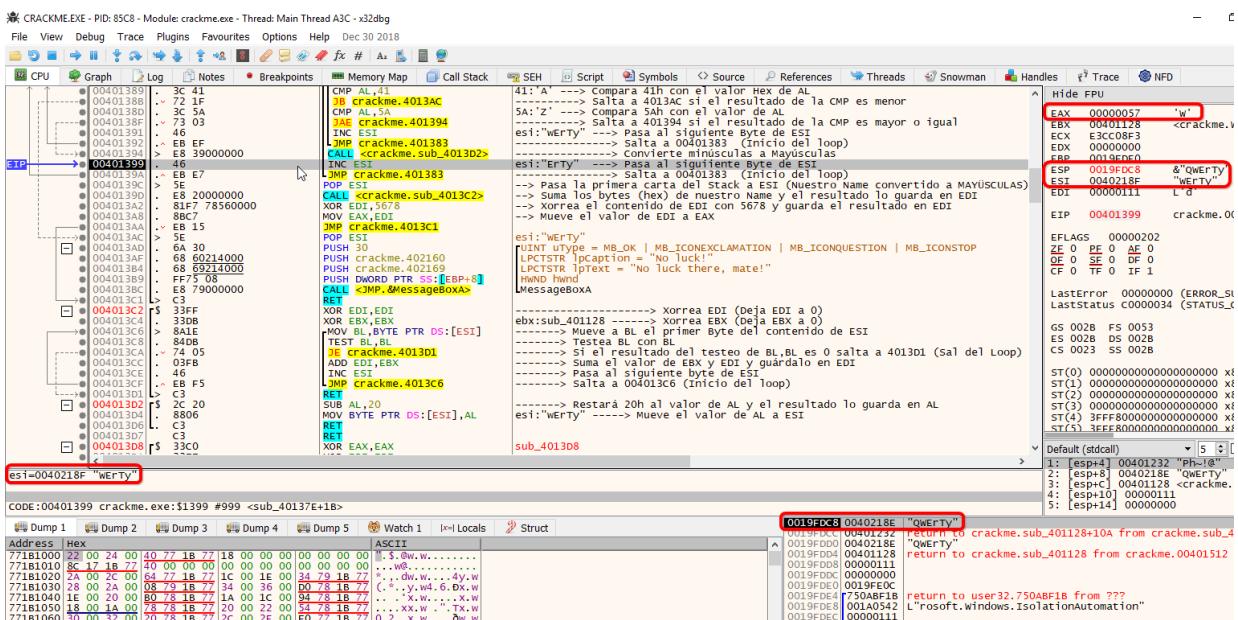
y "x32dbg" nos vuelve a la address en la que nos encontrábamos, justamente aquí:



Ahora, con que ya sabemos los cálculos que va a hacer, sí que podemos entrar en ejecución, y lo hacemos con "F7" o con "Step into"



Una vez dentro de la "CALL", traceamos, y comprobamos que la interpretación que hemos dado a las instrucciones efectivamente han sido las correctas, ya que al salir del "RET" aparecemos en la address "00401399", y observamos que la "CALL" de la que venimos se encarga de convertir las letras minúsculas a MAYÚSCULAS de nuestro Name tipeado.



Es decir:

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

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

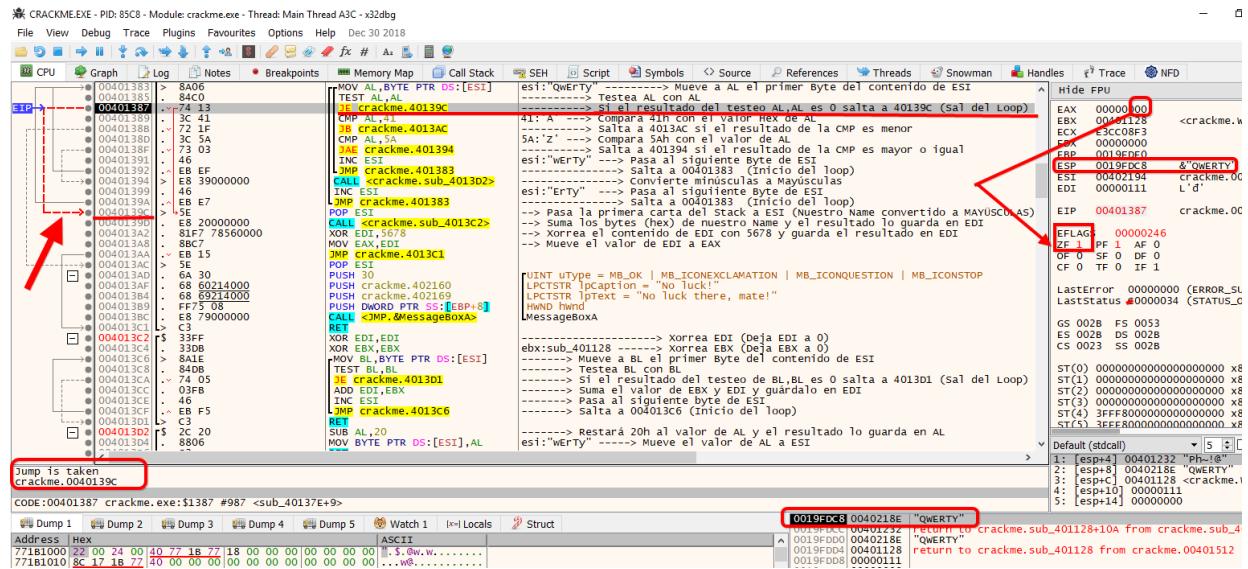
Está claro que si restamos 20h a cualquier valor Hexadecimal que represente una letra minúscula, cuando pasemos por la address "0040138D CMP AL,5A" y por su siguiente instrucción de nuestro Crackme, address "0040138F JAE crackme.401394", el resultado de la comparación nunca será mayor o igual a 5Ah

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

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

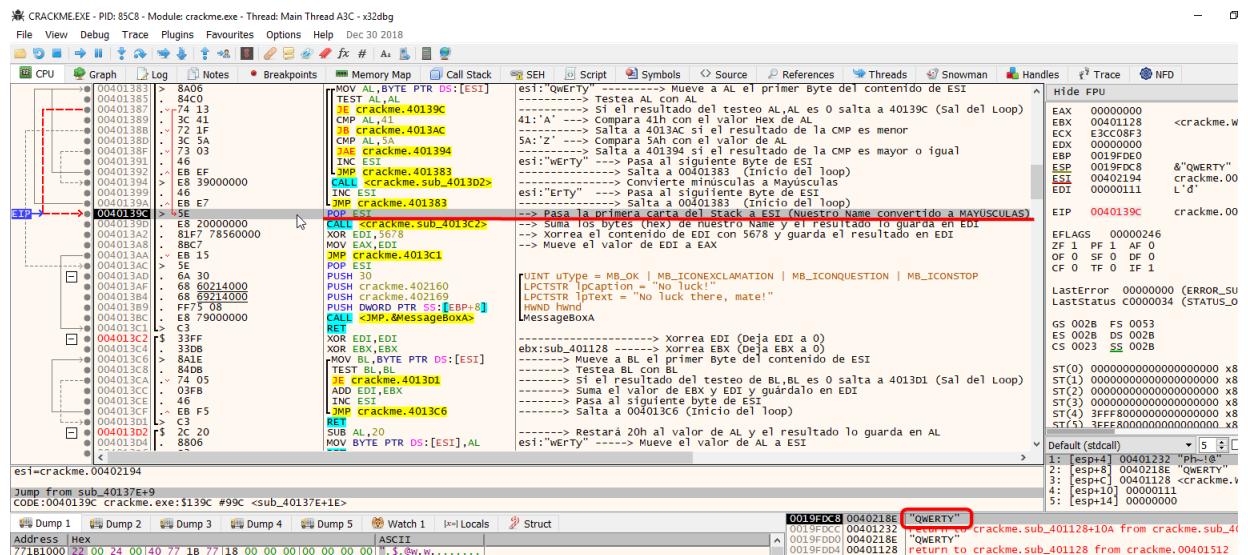
$$y = 79h - 20h = 59h = Y$$

Bien...., traceamos y traceamoshasta que queden convertidas las tres letras minúsculas de nuestro Name tipeado y tendremos esta pantalla:



Al haber terminado con todos los caracteres tipeados de nuestro Name, ahora el resultado del testeo "TEST AL,AL" es "00", el FLAG ZF levanta bandera pasando de "0" a "1" con lo cual el salto condicional "**JE**" se encarga de llevarnos a la address "0040139C"

Damos un trazo y ahora nos encontramos en la address anteriormente indicada:



POP ESI

Lo que va a hacer es coger el contenido de la primera carta del mazo del "Stack" (En nuestro caso "QWERTY", y lo va a pasar al valor del registro "ESI"

Traceamos, y eso es lo que ha hecho.

Ahora estamos en otra "CALL" que se encuentra en la address "0040139D", pues también nos metemos dentro con "Intro" para observar que se va a cocer esta vez en esa subrutina, y vemos que se inicia en la address "004013C2" y termina en el "RET" que está en la address "004013D1".

También observamos un pequeño “Loop2” que va desde la address “004013C6” y la address “004013CF”

Pues vamos a estudiar esas instrucciones una a una, cuyo comentario también he agregado para su mejor compresión.

Volvamos a nuestro punto de partida (salimos de la "CALL" con "Go to > Previous")

Y salimos aquí

CRACKME.EXE - PID: 7F0 - Module: crackme.exe - Thread: Main Thread 7E8 - x32dbg

File View Debug Trace Plugins Favourites Options Help Dec 30 2018

CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace NFD

```

0040137E 3 887424 04 MOV ESI, DWORD PTR SS:[ESP+4] [esp+4]: "QWERTY" --> Mueve a ESI nuestro Name tipado
00401382 > 8A06 PUSH ESI [esi] esti:"QWERTY" -----> Mete nuestro Name al Stack
00401385 > 84C0 MOV AL, BYTE PTR DS:[ESI]
00401389 > 74 13 TEST AL, AL [esp+4]: "QWERTY" -----> Testea AL con AL
0040138B > 3C 41 JE crackme_40139C CMP AL, A1 [esp+4]: "QWERTY" -----> Si el resultado del testeo AL,AL es 0 salta a 40139C (Sal del Loop)
0040138D > 72 1F JE crackme_4013AC CMP AL, SA [esp+4]: "QWERTY" -----> Compara 41h con el valor Hex de AL
0040138F > 73 03 JE crackme_401394 CMP AL, SA [esp+4]: "QWERTY" -----> Salta a 4013AC si el resultado de la CMP es menor
00401391 > 46 INC ESI [esp+4]: "QWERTY" -----> Sume el resultado de la CMP con el valor de AL
00401392 > EB EF JMP crackme_401383 CALL crackme_4013D2 [esp+4]: "QWERTY" -----> Pasa al siguiente byte de ESI
00401394 > E8 39000000 INC ESI [esp+4]: "QWERTY" -----> Convierte minúsculas a Mayúsculas
00401395 > 46 JMP crackme_401383 [esp+4]: "QWERTY" -----> Salta a 00401383 (Inicio del loop)
00401396 > EB E7 JMP crackme_401383 [esp+4]: "QWERTY" -----> Salta a 00401383 (Inicio del loop)
00401397 > E9 00000000 CALL crackme_sub_4013C2 [esp+4]: "QWERTY" -----> Pasa la primera carta del stack a ESI (Nuestro Name convertido a MAYÚSCULAS)
004013A8 > 88C7 MOV EAX, EDI [esp+4]: "QWERTY" -----> Suma los bytes (hex) de nuestro Name y el resultado lo guarda en EDI
004013A9 > EB 15 JMP crackme_4013C1 [esp+4]: "QWERTY" -----> Xorrea el contenido de EDI con 5678 y guarda el resultado en EDI
004013AC > 5E POP ESI [esp+4]: "QWERTY" -----> Mueve el valor de EDI a EAX
004013AD > 6A 30 PUSH crackme_4012160 [esp+4]: "QWERTY" ----->
004013AE > 88C7 XOR EDI, EDI [esp+4]: "QWERTY" ----->
004013AF > 68 60214000 PUSH crackme_402160 [esp+4]: "QWERTY" ----->
004013B0 > 68 69214000 PUSH crackme_402169 [esp+4]: "QWERTY" ----->
004013B1 > 4C 08 PUSH DWORD PTR DS:[EBP+8] [esp+4]: "QWERTY" ----->
004013B2 > 88C7 CALL <MP> &MessageBoxA [esp+4]: "QWERTY" -----> MessageboxA
004013B3 > C3 RET [esp+4]: "QWERTY" ----->
004013C1 > 33FF XOR EDI, EDI [esp+4]: "QWERTY" -----> Xorrea EDI (Deja EDI a 0)
004013C6 > 8A1E XOR EBX, EBX [esp+4]: "QWERTY" -----> Xorrea EBX (Deja EBX a 0)
004013C8 > 84DB MOV BL, BYTE PTR DS:[ESI] [esp+4]: "QWERTY" -----> Mueve a BL el primer Byte del contenido de ESI
004013CA > 70 05 TEST BL, BL [esp+4]: "QWERTY" -----> Testea BL con BL
004013CB > 3CFB JE crackme_4013D1 [esp+4]: "QWERTY" -----> Si el resultado del testeo de BL,BL es 0 salta a 4013D1 (Sal del Loop)
004013CC > 46 ADD EDI, EBX [esp+4]: "QWERTY" -----> Sume el valor de EBX y EDI y guardalo en EDI
004013CE > 46 INC ESI [esp+4]: "QWERTY" -----> Pasa al siguiente byte de ESI
004013CF > EB F5 JMP crackme_4013C6 [esp+4]: "QWERTY" -----> Salta a 004013C6 (Inicio del loop)
004013D1 > C3 RET [esp+4]: "QWERTY" ----->

```

EIP 00401390 crackme.

LastError 00000000 (ERROR)
LastStatus C0000034 (STATUS)

Ahora entramos en ejecución en la "CALL" con "F7"

CRACKME.EXE - PID: 7F0 - Module: crackme.exe - Thread: Main Thread 7E8 - x32dbg

File View Debug Trace Plugins Favourites Options Help Dec 30 2018

CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace NFD

```

0040137E 3 887424 04 MOV ESI, DWORD PTR SS:[ESP+4] [esp+4]: "QWERTY" --> Mueve a ESI nuestro Name tipado
00401382 > 8A06 PUSH ESI [esi] esti:"QWERTY" -----> Mete nuestro Name al Stack
00401385 > 84C0 MOV AL, BYTE PTR DS:[ESI]
00401389 > 74 13 TEST AL, AL [esp+4]: "QWERTY" -----> Testea AL con AL
0040138B > 3C 41 JE crackme_40139C CMP AL, A1 [esp+4]: "QWERTY" -----> Si el resultado del testeo AL,AL es 0 salta a 40139C (Sal del Loop)
0040138D > 72 1F JE crackme_4013AC CMP AL, SA [esp+4]: "QWERTY" -----> Compara 41h con el valor Hex de AL
0040138F > 73 03 JE crackme_401394 CMP AL, SA [esp+4]: "QWERTY" -----> Salta a 401394 si el resultado de la CMP es menor
00401391 > 46 INC ESI [esp+4]: "QWERTY" -----> Sume el resultado de la CMP con el valor de AL
00401392 > EB EF JMP crackme_401383 CALL crackme_4013D2 [esp+4]: "QWERTY" -----> Pasa al siguiente byte de ESI
00401394 > E8 39000000 INC ESI [esp+4]: "QWERTY" -----> Convierte minúsculas a Mayúsculas
00401395 > 46 JMP crackme_401383 [esp+4]: "QWERTY" -----> Salta a 00401383 (Inicio del loop)
00401396 > EB E7 JMP crackme_401383 [esp+4]: "QWERTY" -----> Salta a 00401383 (Inicio del loop)
00401397 > E9 00000000 CALL crackme_sub_4013C2 [esp+4]: "QWERTY" -----> Pasa la primera carta del stack a ESI (Nuestro Name convertido a MAYÚSCULAS)
004013A8 > 88C7 MOV EAX, EDI [esp+4]: "QWERTY" -----> Suma los bytes (hex) de nuestro Name y el resultado lo guarda en EDI
004013A9 > EB 15 JMP crackme_4013C1 [esp+4]: "QWERTY" -----> Xorrea el contenido de EDI con 5678 y guarda el resultado en EDI
004013AC > 5E POP ESI [esp+4]: "QWERTY" -----> Mueve el valor de EDI a EAX
004013AD > 6A 30 PUSH crackme_4012160 [esp+4]: "QWERTY" ----->
004013AE > 88C7 XOR EDI, EDI [esp+4]: "QWERTY" ----->
004013AF > 68 60214000 PUSH crackme_402160 [esp+4]: "QWERTY" ----->
004013B0 > 68 69214000 PUSH crackme_402169 [esp+4]: "QWERTY" ----->
004013B1 > 4C 08 PUSH DWORD PTR DS:[EBP+8] [esp+4]: "QWERTY" ----->
004013B2 > 88C7 CALL <MP> &MessageBoxA [esp+4]: "QWERTY" -----> MessageboxA
004013B3 > C3 RET [esp+4]: "QWERTY" ----->
004013C1 > 33FF XOR EDI, EDI [esp+4]: "QWERTY" -----> Xorrea EDI (Deja EDI a 0)
004013C6 > 8A1E XOR EBX, EBX [esp+4]: "QWERTY" -----> Xorrea EBX (Deja EBX a 0)
004013C8 > 84DB MOV BL, BYTE PTR DS:[ESI] [esp+4]: "QWERTY" -----> Mueve a BL el primer Byte del contenido de ESI
004013CA > 70 05 TEST BL, BL [esp+4]: "QWERTY" -----> Testea BL con BL
004013CB > 3CFB JE crackme_4013D1 [esp+4]: "QWERTY" -----> Si el resultado del testeo de BL,BL es 0 salta a 4013D1 (Sal del Loop)
004013CC > 46 ADD EDI, EBX [esp+4]: "QWERTY" -----> Sume el valor de EBX y EDI y guardalo en EDI
004013CE > 46 INC ESI [esp+4]: "QWERTY" -----> Pasa al siguiente byte de ESI
004013CF > EB F5 JMP crackme_4013C6 [esp+4]: "QWERTY" -----> Salta a 004013C6 (Inicio del loop)
004013D1 > C3 RET [esp+4]: "QWERTY" ----->

```

EIP 004013C2 <crackme>

LastError 00000000 (ERROR)
LastStatus C0000034 (STATUS)

XOR EDI,EDI

Xorrea EDI,EDI y va a dejar el Registro "EDI" con valor "0"

Traceamos para ejecutar la instrucción y efectivamente ahora "EDI" vale "00000000"

CRACKME.EXE - PID: 7F0 - Module: crackme.exe - Thread: Main Thread 7E8 - x32dbg

File View Debug Trace Plugins Favourites Options Help Dec 30 2018

CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace NFD

```

0040137E 3 887424 04 MOV ESI, DWORD PTR SS:[ESP+4] [esp+4]: "QWERTY" --> Mueve a ESI nuestro Name tipado
00401382 > 8A06 PUSH ESI [esi] esti:"QWERTY" -----> Mete nuestro Name al Stack
00401385 > 84C0 MOV AL, BYTE PTR DS:[ESI]
00401389 > 74 13 TEST AL, AL [esp+4]: "QWERTY" -----> Testea AL con AL
0040138B > 3C 41 JE crackme_40139C CMP AL, A1 [esp+4]: "QWERTY" -----> Si el resultado del testeo AL,AL es 0 salta a 40139C (Sal del Loop)
0040138D > 72 1F JE crackme_4013AC CMP AL, SA [esp+4]: "QWERTY" -----> Compara 41h con el valor Hex de AL
0040138F > 73 03 JE crackme_401394 CMP AL, SA [esp+4]: "QWERTY" -----> Salta a 401394 si el resultado de la CMP es menor
00401391 > 46 INC ESI [esp+4]: "QWERTY" -----> Sume el resultado de la CMP con el valor de AL
00401392 > EB EF JMP crackme_401383 CALL crackme_4013D2 [esp+4]: "QWERTY" -----> Pasa al siguiente byte de ESI
00401394 > E8 39000000 INC ESI [esp+4]: "QWERTY" -----> Convierte minúsculas a Mayúsculas
00401395 > 46 JMP crackme_401383 [esp+4]: "QWERTY" -----> Salta a 00401383 (Inicio del loop)
00401396 > EB E7 JMP crackme_401383 [esp+4]: "QWERTY" -----> Salta a 00401383 (Inicio del loop)
00401397 > E9 00000000 CALL crackme_sub_4013C2 [esp+4]: "QWERTY" -----> Pasa la primera carta del stack a ESI (Nuestro Name convertido a MAYÚSCULAS)
004013A8 > 88C7 MOV EAX, EDI [esp+4]: "QWERTY" -----> Suma los bytes (hex) de nuestro Name y el resultado lo guarda en EDI
004013A9 > EB 15 JMP crackme_4013C1 [esp+4]: "QWERTY" -----> Xorrea el contenido de EDI con 5678 y guarda el resultado en EDI
004013AC > 5E POP ESI [esp+4]: "QWERTY" -----> Mueve el valor de EDI a EAX
004013AD > 6A 30 PUSH crackme_4012160 [esp+4]: "QWERTY" ----->
004013AE > 88C7 XOR EDI, EDI [esp+4]: "QWERTY" -----> Xorrea EDI (Deja EDI a 0)
004013AF > 68 60214000 PUSH crackme_402160 [esp+4]: "QWERTY" ----->
004013B0 > 68 69214000 PUSH crackme_402169 [esp+4]: "QWERTY" ----->
004013B1 > 4C 08 PUSH DWORD PTR DS:[EBP+8] [esp+4]: "QWERTY" ----->
004013B2 > 88C7 CALL <MP> &MessageBoxA [esp+4]: "QWERTY" -----> MessageboxA
004013B3 > C3 RET [esp+4]: "QWERTY" ----->
004013C1 > 33FF XOR EDI, EDI [esp+4]: "QWERTY" -----> Xorrea EDI (Deja EDI a 0)
004013C6 > 8A1E XOR EBX, EBX [esp+4]: "QWERTY" -----> Xorrea EBX (Deja EBX a 0)
004013C8 > 84DB MOV BL, BYTE PTR DS:[ESI] [esp+4]: "QWERTY" -----> Mueve a BL el primer Byte del contenido de ESI
004013CA > 70 05 TEST BL, BL [esp+4]: "QWERTY" -----> Testea BL con BL
004013CB > 3CFB JE crackme_4013D1 [esp+4]: "QWERTY" -----> Si el resultado del testeo de BL,BL es 0 salta a 4013D1 (Sal del Loop)
004013CC > 46 ADD EDI, EBX [esp+4]: "QWERTY" -----> Sume el valor de EBX y EDI y guardalo en EDI
004013CE > 46 INC ESI [esp+4]: "QWERTY" -----> Pasa al siguiente byte de ESI
004013CF > EB F5 JMP crackme_4013C6 [esp+4]: "QWERTY" -----> Salta a 004013C6 (Inicio del loop)
004013D1 > C3 RET [esp+4]: "QWERTY" ----->

```

EIP 004013C4 <crackme>

LastError 00000000 (ERROR)
LastStatus C0000034 (STATUS)

XOR EBX,EBX

Lo mismo va a hacer con el registro "EBX", lo Xorrea y lo deja a "00000000"

```

CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Snowman Handles Trace NFD
0040138B > 72 1F
0040138D > 3C 5A
0040138E > 73 03
00401391 > 46
00401394 > EB EF
00401399 > 46
0040139A > 46
0040139B > EB E7
0040139D > 20 00
0040139E > 80 00
0040139F > 68 39000000
004013A0 > 46
004013A1 > EB 15
004013A2 > 3C 5F
004013A3 > 6A 30
004013A4 > 68 60214000
004013A5 > 68 60214000
004013A6 > 68 60214000
004013A7 > 68 60214000
004013A8 > 68 7 78560000
004013A9 > 80 00
004013AA > 80 00
004013AB > 80 00
004013AC > 80 00
004013AD > 80 00
004013AE > 80 05
004013AF > 3C 5F
004013B0 > 46
004013B1 > EB F5
004013C1 > C3
004013C2 > EB 0A
004013C3 > MOV BL,BYTE PTR DS:[ESI]
004013C4 > TEST BL,BL
004013C5 > JE crackme.4013D1
004013C6 > ADD EDI,EBX
004013C7 > INC EDI
004013C8 > JMP crackme.4013C6
004013D1 > C3

```

Registers pane (right side):

- EAX = 00000000
- EBX = 00000000 (highlighted with a red box)
- ECX = F14EA15B
- EDX = 00000000
- ESP = 0019FC08
- ESI = 0040218E "QWERTY"
- EDI = 00000000
- EIP = 004013C6 crackme.0

Registers pane (bottom):

- EFlags = 00000246
- ZF = 1 PF = 1 AF = 0
- OF = 0 SF = 0 DF = 0
- CF = 0 TF = 0 IF = 1

Registers pane (bottom right):

- LastError = 00000000 (ERROR_S)
- LastStatus = C0000034 (STATUS_)
- GS = 002B FS = 0053
- ES = 002B DS = 002B
- CS = 0023 SS = 002B
- ST(0) = 00000000000000000000000000000000
- ST(1) = 00000000000000000000000000000000

MOV BL,BYTE PTR DS:[ESI]

Esta instrucción va a MOVer el primer BYTE del contenido de "ESI" y lo va a guardar en "BL" (ESI = QwErTy primer Byte = 51h = Q)

```

CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Snowman Handles Trace NFD
0040138B > 72 1F
0040138D > 3C 5A
0040138E > 73 03
00401391 > 46
00401394 > EB EF
00401399 > 46
0040139A > 46
0040139B > EB E7
0040139D > 20 00
0040139E > 80 00
0040139F > 68 39000000
004013A0 > 46
004013A1 > EB 15
004013A2 > 3C 5F
004013A3 > 6A 30
004013A4 > 68 60214000
004013A5 > 68 60214000
004013A6 > 68 60214000
004013A7 > 68 60214000
004013A8 > 68 7 78560000
004013A9 > 80 00
004013AA > 80 00
004013AB > 80 00
004013AC > 80 00
004013AD > 80 00
004013AE > 80 05
004013AF > 3C 5F
004013B0 > 46
004013C1 > C3
004013C2 > EB 0A
004013C3 > MOV BL,BYTE PTR DS:[ESI]
004013C4 > TEST BL,BL
004013C5 > JE crackme.4013D1
004013C6 > ADD EDI,EBX
004013C7 > INC EDI
004013C8 > JMP crackme.4013C6
004013D1 > C3

```

Registers pane (right side):

- EAX = 00000000
- EBX = 00000000 (highlighted with a red box)
- ECX = F14EA15B
- EDX = 00000000
- ESP = 0019FC08
- ESI = 0040218E "QWERTY"
- EDI = 00000000
- EIP = 004013C8 crackme.0

Registers pane (bottom):

- EFlags = 00000246
- ZF = 1 PE = 1 AF = 0
- OF = 0 SF = 0 DF = 0
- CF = 0 TF = 1 IF = 1

Registers pane (bottom right):

- LastError = 00000000 (ERROR_S)
- LastStatus = C0000034 (STATUS_)
- GS = 002B FS = 0053
- ES = 002B DS = 002B
- CS = 0023 SS = 002B
- ST(0) = 00000000000000000000000000000000
- ST(1) = 00000000000000000000000000000000

TEST BL,BL

Testea BL,BL por tanto en nuestro caso "BL" = "51h" = "Q"

JE crackme.4013D1

Con que el resultado del Testeo entre BL,BL es "51h" , el salto condicional "JE" nos deja continuar. Por el contrario si en "BL" tuviéramos el valor "00" el Flag ZF levantaría bandera y el salto condicional "JE" nos llevaría a la address "004013D1" donde se encuentra el "RET" que es la salida del "Loop" en el que nos encontramos.

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

La siguiente instrucción es un

ADD EDI,EBX

Aquí va a ADDicionar (sumar) los valores de "EBX" + "EDI" , y el resultado lo va a guardar en "EDI"

Registers:

- EAX: 00000000
- EBX: 00000051 'Q'
- ECX: F14EA156
- EDX: 00000000
- EBP: 0019FDE0
- ESP: 0019FDC8
- ESI: 0040218E
- EDI: 00000000
- EIP: 004013CE crackme.0

Stack:

... 004013C8 004013C9 004013CA 004013CB 004013CC 004013CD 004013CE 004013CF 004013D0 004013D1 ...

Traceamos y con que "EBX" = "00000051h" + "EDI" = "00000000h" obtenemos que ahora "EDI" = "51h"

Registers:

- EAX: 00000000
- EBX: 00000051 'Q'
- ECX: F14EA156
- EDX: 00000000
- EBP: 0019FDE0
- ESP: 0019FDC8
- ESI: 0040218E
- EDI: 00000000
- EIP: 004013CE crackme.0

Stack:

... 004013C8 004013C9 004013CA 004013CB 004013CC 004013CD 004013CE 004013CF 004013D0 004013D1 ...

INC ESI

Esta instrucción va a INCREMENTAR en "1" el valor del registro "ESI"
Si ahora el valor de "ESI" es "0040218E"

Registers:

- EAX: 00000000
- EBX: 00000051 'Q'
- ECX: F14EA156
- EDX: 00000000
- EBP: 0019FDE0
- ESP: 0019FDC8
- ESI: 0040218E**
- EDI: 00000000
- EIP: 004013CE

Stack:

... 004013C8 004013C9 004013CA 004013CB 004013CC 004013CD 004013CE 004013CF 004013D0 004013D1 ...

Ejecutamos la instrucción y "ESI" vale 0040218F (0040218E+1)

Lo que ha hecho es que si antes de ejecutar la instrucción, en "ESI" guardaba "QwErTy", una vez ejecutada, al sumarle 1h pasa al siguiente byte, quedando en "wErTy"

Registers:

- EAX: 00000000
- EBX: 00000051 'Q'
- ECX: F14EA156
- EDX: 00000000
- EBP: 0019FDE0
- ESP: 0019FDC8
- ESI: 0040218F "WERTY"
- EDI: 00000001 'Q'
- EIP: 004013CF

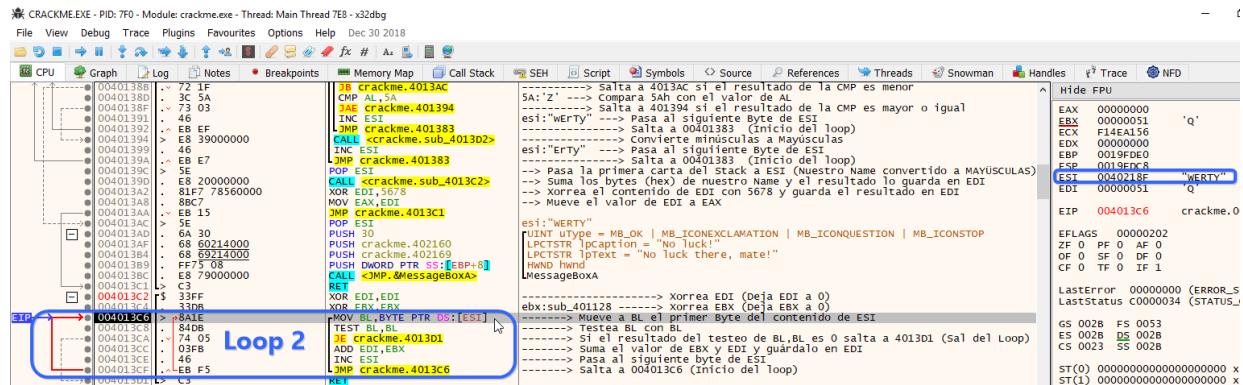
Stack:

... 004013C8 004013C9 004013CA 004013CB 004013CC 004013CD 004013CE 004013CF 004013D0 004013D1 ...

JMP crackme.4013C6

Este salto incondicional "JMP" nos llevará directos a la address "004013C6" que es el inicio del "Loop 2"

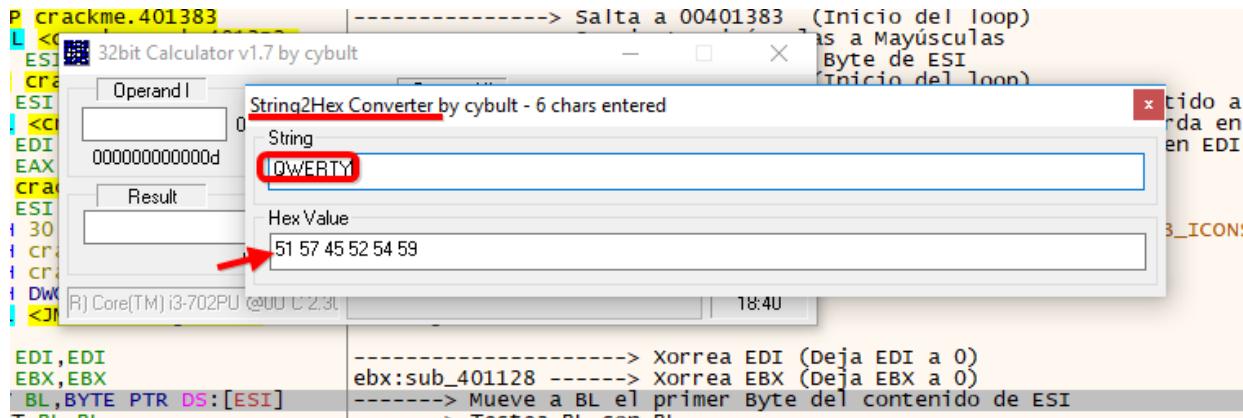
Lo ejecutamos y aparecemos aquí



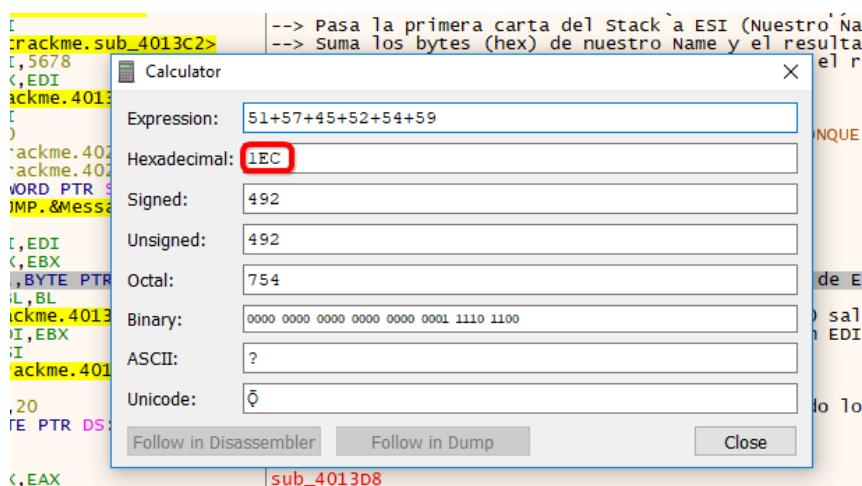
Y nuevamente se va repetir el mismo proceso con los demás bytes de nuestro Name tipeado,

En resumen, de lo que se encarga este "Loop 2" es sumar los bytes hexadecimales de nuestro Name

Nos vamos directos a la tool "Calc_17" y esta vez en la casilla "String de String2Hex", tipeamos directamente nuestro Name entero y obtenemos que su conversión a Hexadecimal es "51 57 45 52 54 59"



Por tanto si sumamos, el resultado obtenido es "1EC"

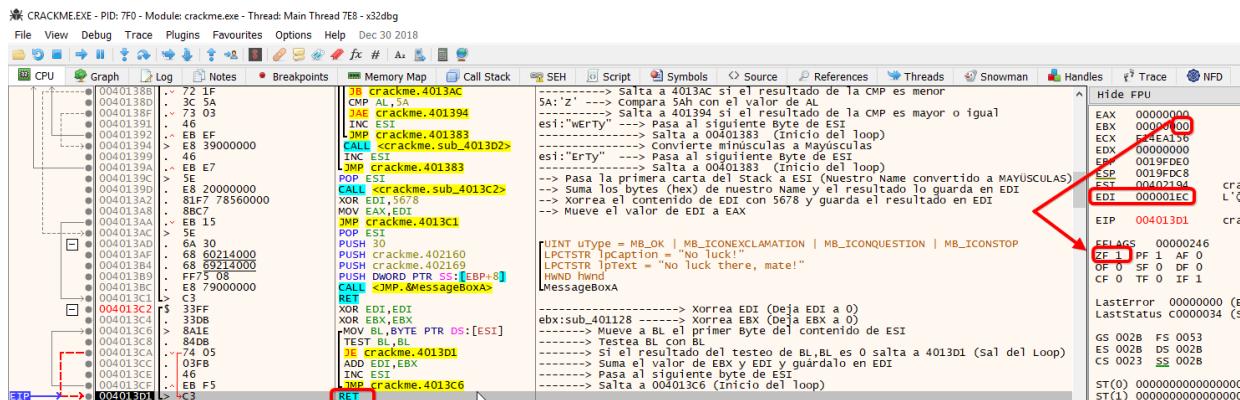


Q W E R T Y

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

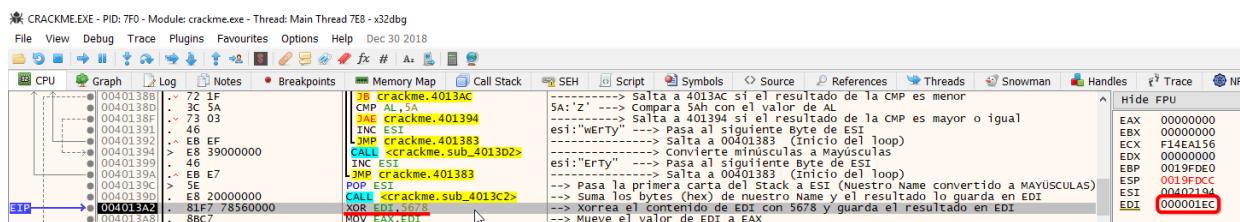
Vamos a comprobarlo, traceamos y traceamos y observamos como va trabajando el "Loop 2" hasta que no quede ningún carácter de nuestro Name tipeado y aparecemos en la address "004013D1", donde ahora "EDI" vale 1EC.

También observamos que con que el valor de "BL" es "00" ya que las instrucciones han terminado de leer los caracteres de nuestro Name, se activa el FLAG ZF levantando bandera "1", y el salto condicional "JE" de la address "004013CA" nos manda directos al "RET" de la address "004013D1"



RET

Ejecutamos el "RET" que ya sabemos que nos va a sacar de la "CALL"

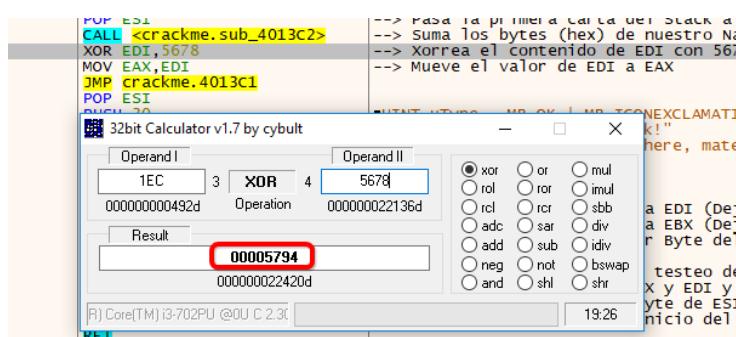


y ahora la siguiente instrucción es :

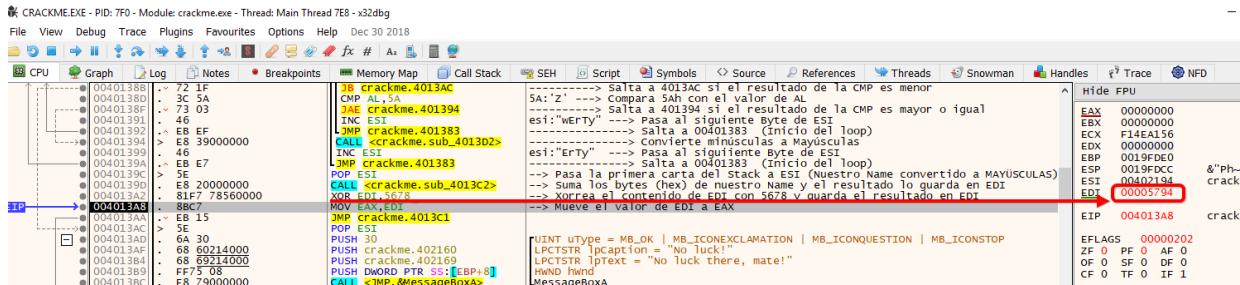
XOR EDI,5678

O sea que va a hacer un "XOR 5678" con el valor de "EDI" y el resultado lo va a guardar en "EDI"

Volvemos a nuestra "CAL_17", rellenamos datos y obtenemos el siguiente resultado



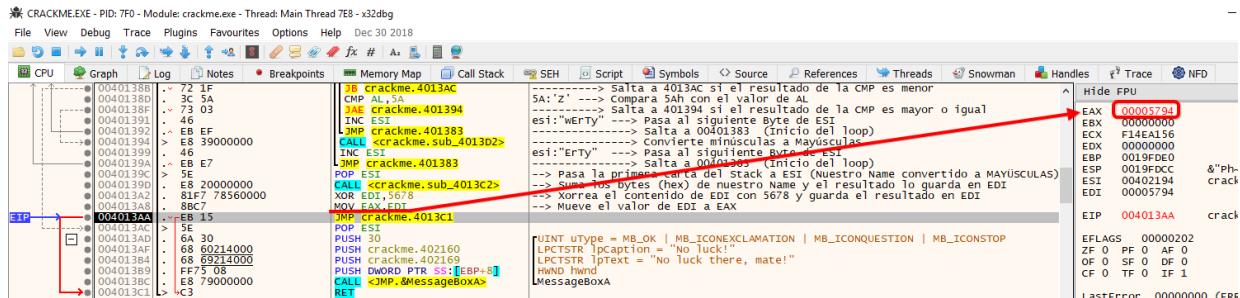
Tracemos para comprobar, y efectivamente ahora "EDI" vale "00005794h"



La siguiente instrucción es un

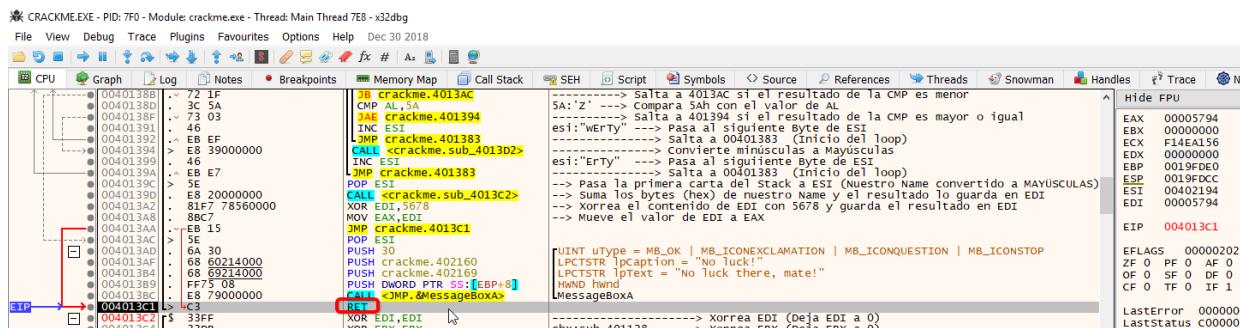
MOV EAX,EDI

Va a MOVer el contenido de "EDI" a "EAX"

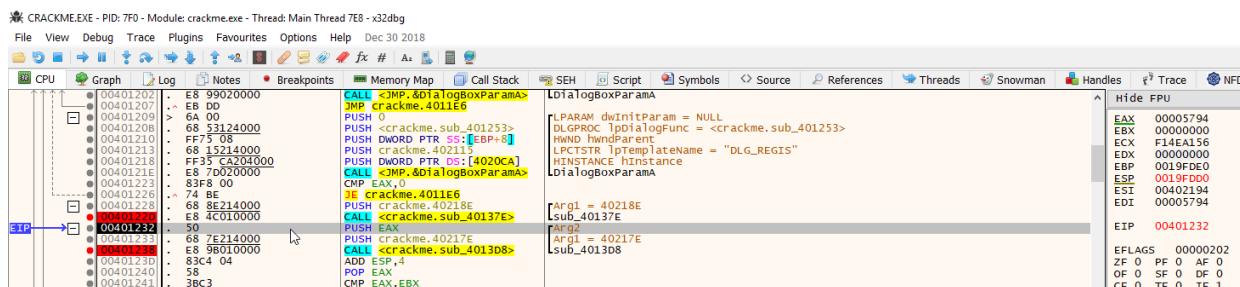


Y ahora el "JMP" nos llevará directos al "RET" que está en la address "004013C1"

Lo ejecutamos y aquí estamos



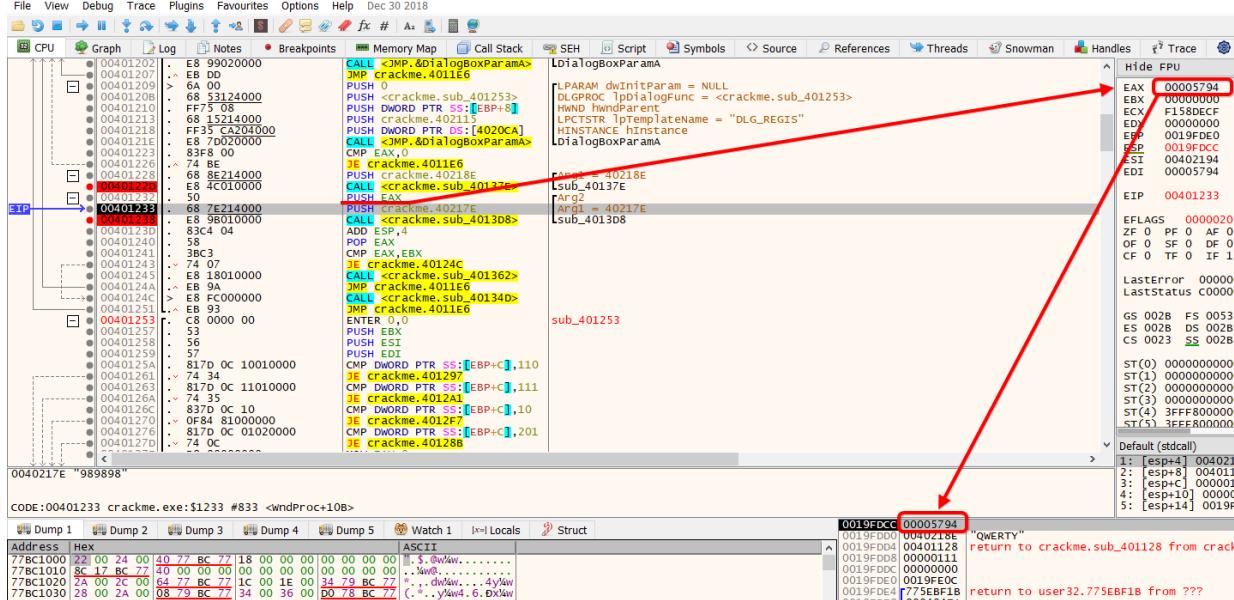
Pasamos también el "RET" y por fin salimos del PRIMER "CALL" de donde veníamos al principio



Este primer "CALL" lo que ha hecho hasta ahora es trabajar únicamente con el Name tipeado.

Seguimos....

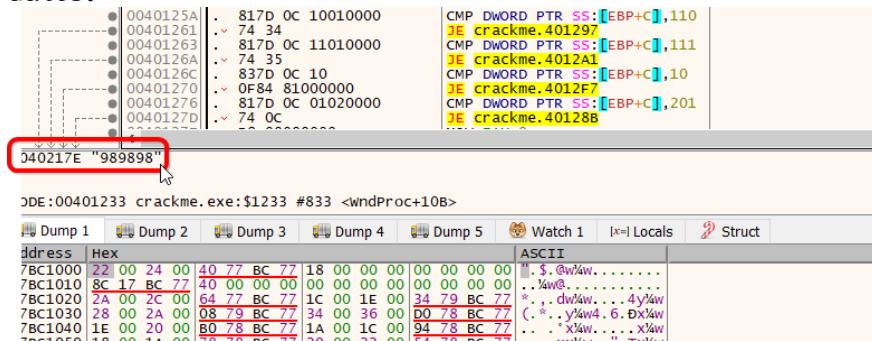
PUSH EAX



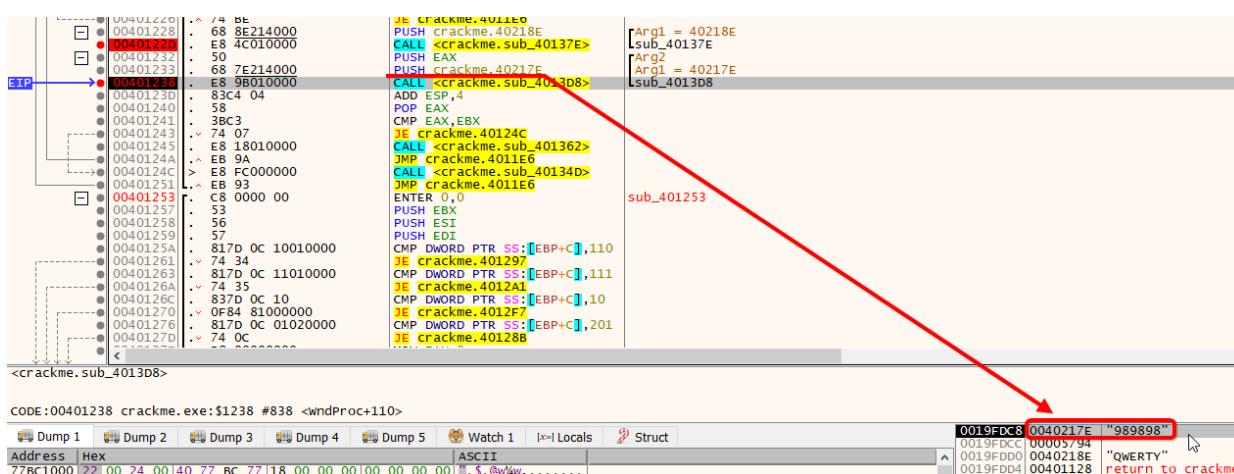
PUSH crackme.40217E

Mete a Stack el contenido de "0040217E"

(Ya vemos directamente que el contenido de `0040217E = "989898"`, nuestro Serial truco), por lo que todo parece indicar que ahora el código va a trabajar con esos datos.



Ejecutamos la instrucción y ahora la primera carta del mazo en el Stack es nuestro serial truco



Y nos encontramos en la puerta del segundo "CALL" donde tenemos colocado el segundo "BP"

Entramos directamente dentro de la "CALL" con un solo "F7" y ahora, vamos a estudiar las siguientes instrucciones del código:

```

004013D8: 33C0          XOR EAX,EAX
004013DA: 33FF          XOR EDI,EDI
004013DC: 33DB          XOR EBX,EBX
004013DE: BB7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
004013E2: B0 0A          MOV AL,A
004013E4: 8A1E          MOV BL,BYTE PTR DS:[ESI]
004013E6: 84DB          TEST BL,BL
004013E8: 74 0B          JE crackme.4013F5
004013E9: 80EB 30        SUB BL,30
004013ED: 0FAFF8        IMUL EDI,EAX
004013F0: 03FB          ADD EDI,EBX
004013F2: 46             INC ESI
004013F3: EB ED          JMP crackme.4013E2
004013F5: 81F7 34120000   XOR EDI,1234
004013F8: 8BDF          MOV EBX,EDI
004013FD: C3             RET
004013FE: FF25 84314000   JMP DWORD PTR DS:<&K11Timer>
00401404: FF25 88314000   JMP DWORD PTR DS:<&GetSystemMetrics>

```

Este "CALL" empieza en la address "004013D8" y termina en "004013FD", con esta simple ojeada del código que queda entre esas dos direcciones (y que ya vimos al principio del tutorial), nuestro instinto de cracker nos dice que estamos dentro de otra parte importante que debemos descifrar si queremos llegar a nuestra pretensión. Aquí va a trabajar con la parte del Serial.

También he agregado antes del traceo, los correspondientes comentarios para facilitar su comprensión.

```

004013D8: 33C0          XOR EAX,EAX
004013DA: 33FF          XOR EDI,EDI
004013DC: 33DB          XOR EBX,EBX
004013DE: BB7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
004013E2: B0 0A          MOV AL,A
004013E4: 8A1E          MOV BL,BYTE PTR DS:[ESI]
004013E6: 84DB          TEST BL,BL
004013E8: 74 0B          JE crackme.4013F5
004013E9: 80EB 30        SUB BL,30
004013ED: 0FAFF8        IMUL EDI,EAX
004013F0: 03FB          ADD EDI,EBX
004013F2: 46             INC ESI
004013F3: EB ED          JMP crackme.4013E2
004013F5: 81F7 34120000   XOR EDI,1234
004013F8: 8BDF          MOV EBX,EDI
004013FD: C3             RET
004013FE: FF25 84314000   JMP DWORD PTR DS:<&K11Timer>
00401404: FF25 88314000   JMP DWORD PTR DS:<&GetSystemMetrics>

```

XOR EAX,EAX

XOR EDI,EDI

XOR EBX,EBX

Los Registros EAX,EDI y ESI serán "XORreados" con ellos mismos, por tanto sus respectivos valores quedarán en "00000000"

Traceamos para que se ejecuten los "XOR" y vemos como los tres registros han quedado a "0"

```

004013D8: 33C0          XOR EAX,EAX
004013DA: 33FF          XOR EDI,EDI
004013DC: 33DB          XOR EBX,EBX
004013DE: BB7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
004013E2: B0 0A          MOV AL,A
004013E4: 8A1E          MOV BL,BYTE PTR DS:[ESI]
004013E6: 84DB          TEST BL,BL
004013E8: 74 0B          JE crackme.4013F5
004013E9: 80EB 30        SUB BL,30
004013ED: 0FAFF8        IMUL EDI,EAX
004013F0: 03FB          ADD EDI,EBX
004013F2: 46             INC ESI
004013F3: EB ED          JMP crackme.4013E2
004013F5: 81F7 34120000   XOR EDI,1234
004013F8: 8BDF          MOV EBX,EDI
004013FD: C3             RET
004013FE: FF25 84314000   JMP DWORD PTR DS:<&K11Timer>
00401404: FF25 88314000   JMP DWORD PTR DS:<&GetSystemMetrics>

```

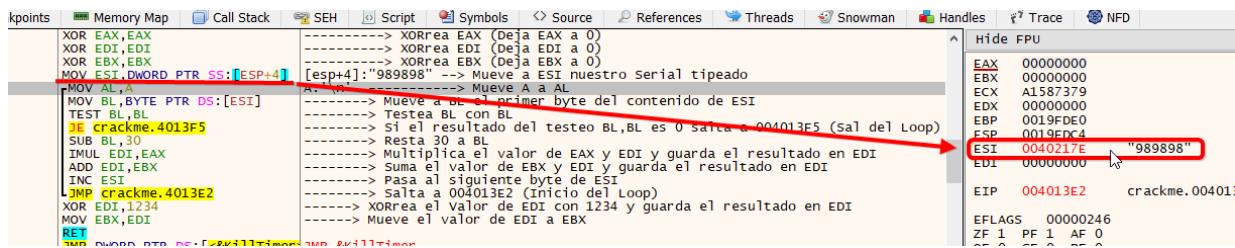
Después la instrucción

MOV ESI,DWORD PTR SS:[ESP+4]

Va a MOVer el contenido de [ESP+4] a "ESI"

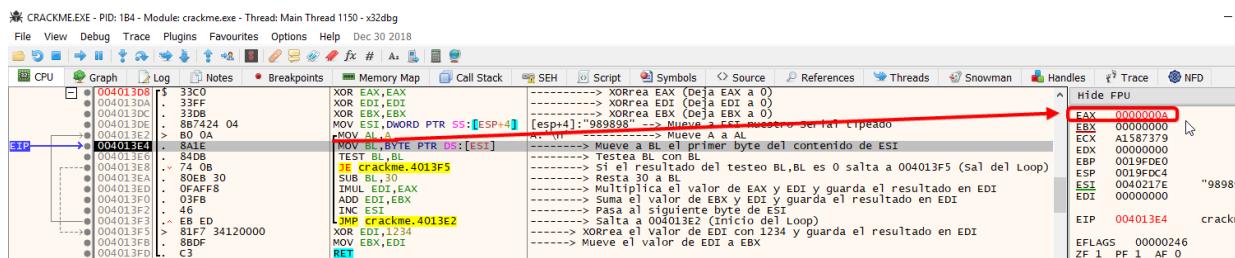
[ESP+4] = "989898"

La ejecutamos y efectivamente aquí vemos el movimiento



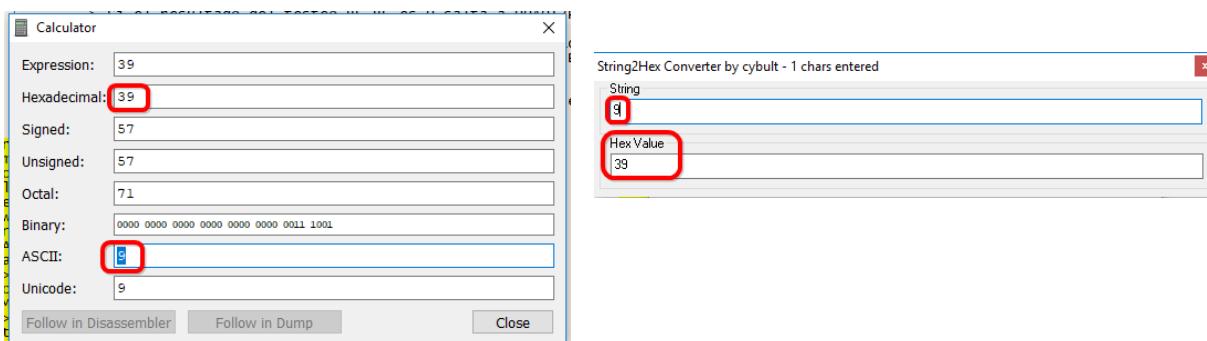
MOV AL,A

Aquí va a MOVer "A" a "AL"

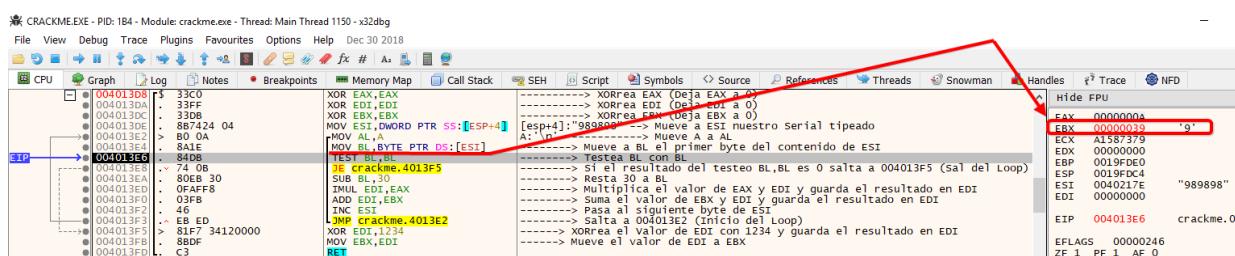


MOV BL,BYTE PTR DS:[ESI]

Ahora va a MOVer a "BL" el primer byte del contenido de "ESI".
ESI = "989898" el primer byte es 9 = "39h"



Traceamos y, aquí lo tenemos



TEST BL,BL

Testea BL,BL en nuestro caso "BL" = "39h" = "9"

```

int32_t crackme_4013F5()
{
    XOR EAX, EAX
    XOR EDI, EDI
    XOR EBX, EBX
    MOV ESI, DWORD PTR SS:[ESP+4]
    MOV AL, A
    MOV BL, BYTE PTR DS:[ESI]
    TEST BL, BL
    JE crackme_4013F5
    SUB BL, 30
    IMUL EDI, EAX
    ADD EDI, EBX
    INC ESI
    JMP crackme_4013E2
    XOR EDI, 1234
    MOV EBX, EDI
    RET
}

// Disassembly pane
xor eax, eax
xor edi, edi
xor ebx, ebx
mov esi, dword ptr ss:[esp+4]
mov al, a
mov bl, byte ptr ds:[esi]
test bl, bl
je crackme.4013F5
sub bl, 30
imul edi, eax
add edi, ebx
inc esi
jmp crackme.4013E2
xor edi, 1234
mov ebx, edi
ret

// Registers pane
EAX 0000000A
EBX 00000039 '9'
ECX A1587379
EDX 00000000
EBP 0019FDE0
ESP 0019FDCA
ESI 0040217E "989898"
EDI 00000000
EIP 004013E8 crackme.004013E8
EFLAGS 00000026
ZF 0 PF 1 AF 0

```

JE crackme.4013F5

Y con que el resultado del Testeo entre BL,BL es "39h" = "9" el salto condicional "JE" nos deja continuar.

Cuando en "BL" tengamos el valor "00" el Flag ZF levantará bandera y el salto condicional "JE" nos llevará a la address "004013F5" que es la salida del "Loop3" en el que nos encontramos.

(Eso sucederá cuando ya no quedan más caracteres para leer de nuestro Serial tipeado)

SUB BL, 30

Resta el segundo operando del primero y el resultado lo guarda en el primero
Pues si "BL" = "39h" lo que hará es "39h" - "30h" = "9h" por lo que "BL" = "09h"

```

int32_t crackme_4013ED()
{
    XOR EAX, EAX
    XOR EDI, EDI
    XOR EBX, EBX
    MOV ESI, DWORD PTR SS:[ESP+4]
    A: '\n'
    > BO OA
    MOV AL, A
    MOV BL, BYTE PTR DS:[ESI]
    TEST BL, BL
    JE crackme_4013F5
    SUB BL, 30
    OFAFF8
    IMUL EDI, EAX
    ADD EDI, EBX
    INC ESI
    JMP crackme_4013E2
    XOR EDI, 1234
    RET
}

// Disassembly pane
xor eax, eax
xor edi, edi
xor ebx, ebx
mov esi, dword ptr ss:[esp+4]
a: '\n'
> bo oa
mov al, a
mov bl, byte ptr ds:[esi]
test bl, bl
je crackme.4013F5
sub bl, 30
ofaff8
imul edi, eax
add edi, ebx
inc esi
jmp crackme.4013E2
xor edi, 1234
ret

// Registers pane
EAX 0000000A
EBX 00000009
ECX A1587379
EDX 00000000
EBP 0019FDE0
ESP 0019FDCA
ESI 0040217E
EDI 00000000
EIP 004013ED

```

Ahora

IMUL EDI,EAX

Va a multiplicar el valor de los dos operandos y el resultado lo va a guardar en el primero

Si "EDI" = "00000000h" y "EAX" = "0000000Ah" pues "Ah" * "0h" = "0h"

Calculator	32bit Calculator v1.7 by cybult
Expression: A*0	Operand I: A 1 IMUL 1 Operand II: 0
Hexadecimal: 0	000000000000d Operation: 000000000000d
Signed: 0	Result: 00000000
Unsigned: 0	000000000000d
Octal: 0	
Binary: 0000 0000 0000 0000 0000 0000 0000	
ASCII: ?	
Unicode: ?	
Follow in Disassembler	Follow in Dump
Close	Close

Traceamos y

```

    XOR EAX,EAX           ---> XORrea EAX (Deja EAX a 0)
    XOR EDI,EDI           ---> XORrea EDI (Deja EDI a 0)
    XOR EBX,EBX           ---> XORrea EBX (Deja EBX a 0)
    MOV ESI,WORD PTR SS:[ESP+4] [esp+4]: "989898" --> Mueve a ESI nuestro Serial tipiado
    MOV AL,A              A:\n
    MOV BL,BYTE PTR DS:[ESI] ---> Mueve a BL el primer byte del contenido de ESI
    TEST BL,BL             ---> Testea BL con BL
    JE crackme.4013F5     si el resultado del testeo BL,BL es 0 salta a 004013F5 (Sal del Loop)
    SUB BL,30               Resta 30 a BL
    IMUL EDI,EAX           ---> Multiplica el valor de EAX y EDI y guarda el resultado en EDI
    ADD EDI,EBX             ---> Suma el valor de EBX y EDI y guarda el resultado en EDI
    INC ESI                ---> Pasa al siguiente byte de ESI
    JMP crackme.4013E2     Salta a 004013E2 (Inicio del Loop)
    MOV EDI,1234             XORrea el valor de EDI con 1234 y guarda el resultado en EDI
    MOV EBX,EDI             Mueve el valor de EDI a EBX
    RET

```

Registers pane:

EAX	0000000A
EBX	00000009
ECX	A1587379
EDX	00000000
EBP	0019FDE0
ESP	0019FDC4
EST	0040217E "989898"
EDI	00000000

Status bar: EIP 004013F0 crackme.00

La operación siguiente es

ADD EDI,EBX

Va a ADDicionar los registros "EBX" y "EDI" y el resultado lo guarda en "EDI"

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

Help Dec 30 2018

```

    XOR EAX,EAX           ---> XORrea EAX (Deja EAX a 0)
    XOR EDI,EDI           ---> XORrea EDI (Deja EDI a 0)
    XOR EBX,EBX           ---> XORrea EBX (Deja EBX a 0)
    MOV ESI,WORD PTR SS:[ESP+4] [esp+4]: "989898" --> Mueve a ESI nuestro Serial tipiado
    MOV AL,A              A:\n
    MOV BL,BYTE PTR DS:[ESI] ---> Mueve a BL el primer byte del contenido de ESI
    TEST BL,BL             ---> Testea BL con BL
    JE crackme.4013F5     si el resultado del testeo BL,BL es 0 salta a 004013F5 (Sal del Loop)
    SUB BL,30               Resta 30 a BL
    IMUL EDI,EAX           ---> Multiplica el valor de EAX y EDI y guarda el resultado en EDI
    ADD EDI,EBX             ---> Suma el valor de EBX y EDI y guarda el resultado en EDI
    INC ESI                ---> Pasa al siguiente byte de ESI
    JMP crackme.4013E2     Salta a 004013E2 (Inicio del Loop)
    MOV EDI,1234             XORrea el valor de EDI con 1234 y guarda el resultado en EDI
    MOV EBX,EDI             Mueve el valor de EDI a EBX
    RET

```

Registers pane:

EAX	0000000A
EBX	00000009
ECX	A1587379
EDX	00000000
EBP	0019FDE0
ESP	0019FDC4
EST	0040217E "989898"
EDI	00000009

Status bar: EIP 004013F2 crackme.0

INC ESI

Va a INCrementar "1h" al valor de "ESI"

si "ESI" = "0040217E"

Ejecutamos la instrucción y ahora ESI vale 0040217F (0040217E+1)

Lo que ha hecho es que si antes de ejecutar la instrucción, en "ESI" había guardado "989898", una vez ejecutada, al sumarle 1h pues pasa al siguiente byte, quedando en "89898"

```

    XOR EAX,EAX           ---> XORrea EAX (Deja EAX a 0)
    XOR EDI,EDI           ---> XORrea EDI (Deja EDI a 0)
    XOR EBX,EBX           ---> XORrea EBX (Deja EBX a 0)
    MOV ESI,WORD PTR SS:[ESP+4] [esp+4]: "89898" --> Mueve a ESI nuestro Serial tipiado
    MOV AL,A              A:\n
    MOV BL,BYTE PTR DS:[ESI] ---> Mueve a BL el primer byte del contenido de ESI
    TEST BL,BL             ---> Testea BL con BL
    JE crackme.4013F5     si el resultado del testeo BL,BL es 0 salta a 004013F5 (Sal del Loop)
    SUB BL,30               Resta 30 a BL
    IMUL EDI,EAX           ---> Multiplica el valor de EAX y EDI y guarda el resultado en EDI
    ADD EDI,EBX             ---> Suma el valor de EBX y EDI y guarda el resultado en EDI
    INC ESI                ---> Pasa al siguiente byte de ESI
    JMP crackme.4013E2     Salta a 004013E2 (Inicio del Loop)
    MOV EDI,1234             XORrea el valor de EDI con 1234 y guarda el resultado en EDI
    MOV EBX,EDI             Mueve el valor de EDI a EBX
    RET

```

Registers pane:

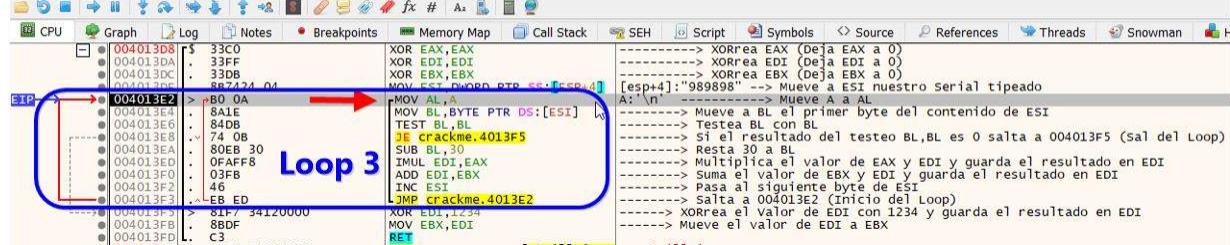
EAX	0000000A
EBX	00000009
ECX	A1587379
EDX	00000000
EBP	0019FDE0
ESP	0019FDC4
EST	0040217F "89898"
EDI	00000009

Status bar: EIP 004013F3 crackme.0

JMP crackme.4013E2

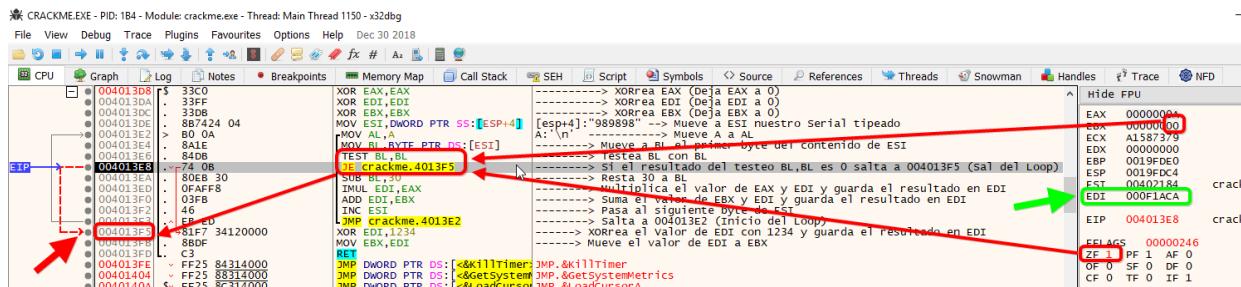
Y este salto incondicional "JMP" nos lleva de nuevo al inicio del "Loop3" que está en la address "004013E2"

Traceamos y apareceremos en esa address para repetir el proceso con los demás caracteres de nuestro Name tipiado.

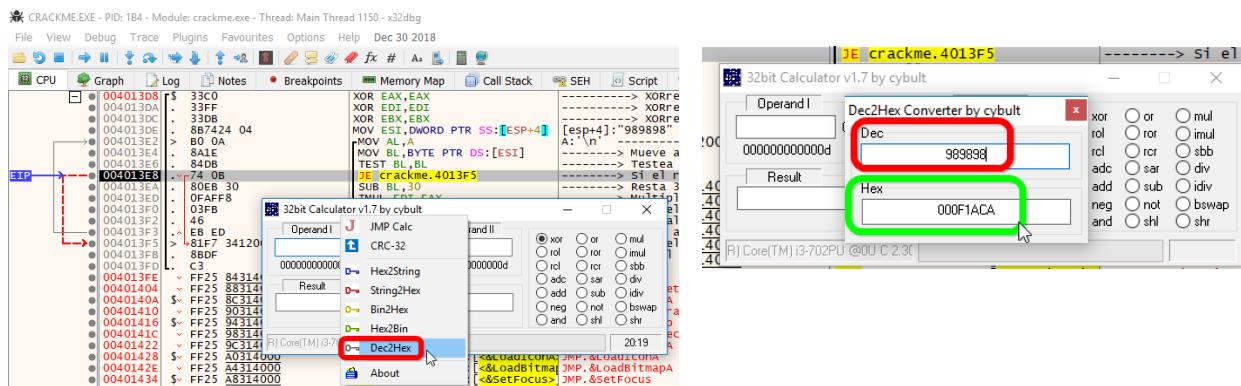


Traceamos y traceamos... hasta que no queden más caracteres en nuestro Name y cuando lleguemos a la address "004013E8" vemos que ahora "BL" vale "00", el Flag ZF levanta bandera "1" por lo que el salto condicional "JE" nos va a sacar del "Loop 3".

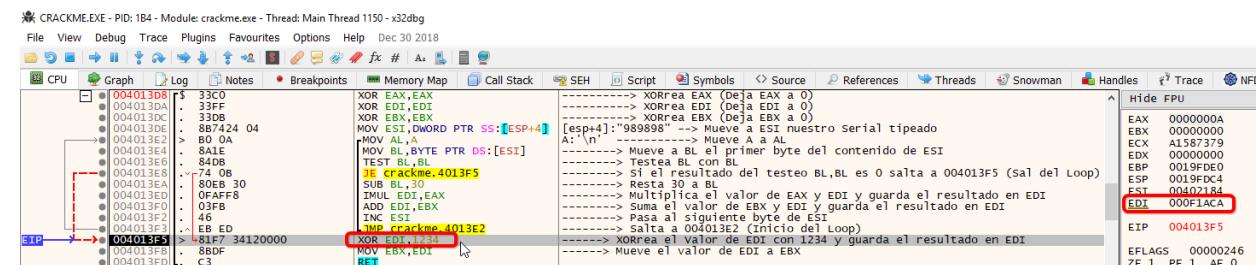
Los más observadores también se habrán dado cuenta que el registro "EDI" ahora su valor es "F1ACA" por lo que llegamos a la conclusión de que este "Loop 3" se encarga de pasar los caracteres del Serial tipeado a caracteres Hexadecimales



Lo comprobamos antes de ejecutar la instrucción y vemos que es cierto



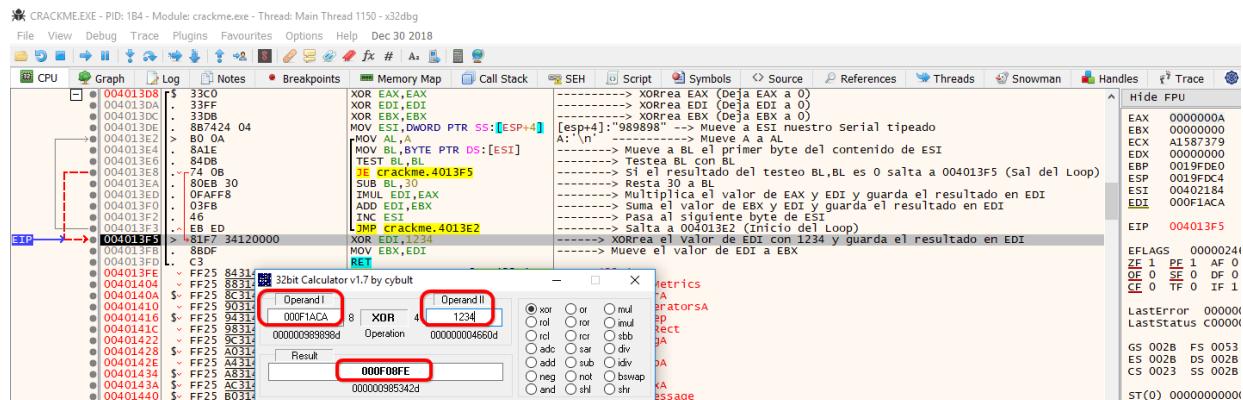
Ahora sí que podemos ejecutar la instrucción con toda seguridad ya que sabemos de antemano su resultado, salimos del "Loop 3" y nos encontramos en la address "004013F5"



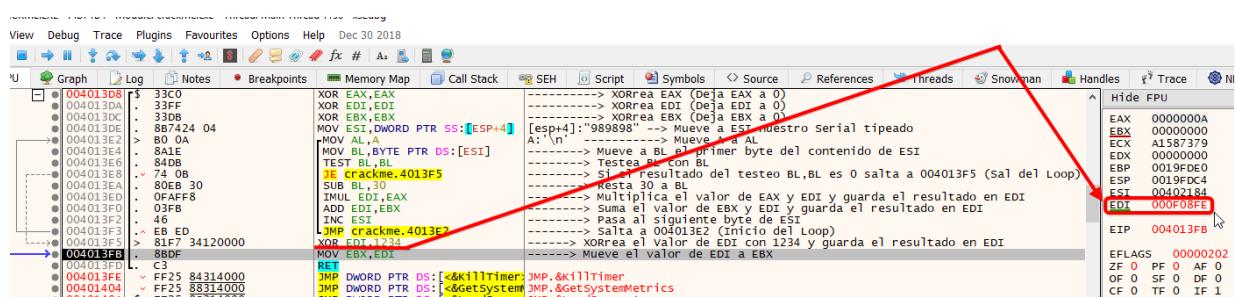
Ya solo nos quedan dos instrucciones antes de llegar al "RET" (Mi garganta está seca y empiezo a tener sed)

Va a hacerle un "XOR 1234" al valor de "EDI" y el resultado lo guardará en "EDI"

Nos vamos de nuevo a nuestra Calculadora, rellenamos datos y obtenemos que el resultado es "000F08FE"



Traceamos para ejecutar la instrucción y vemos que lo ha realizado correctamente



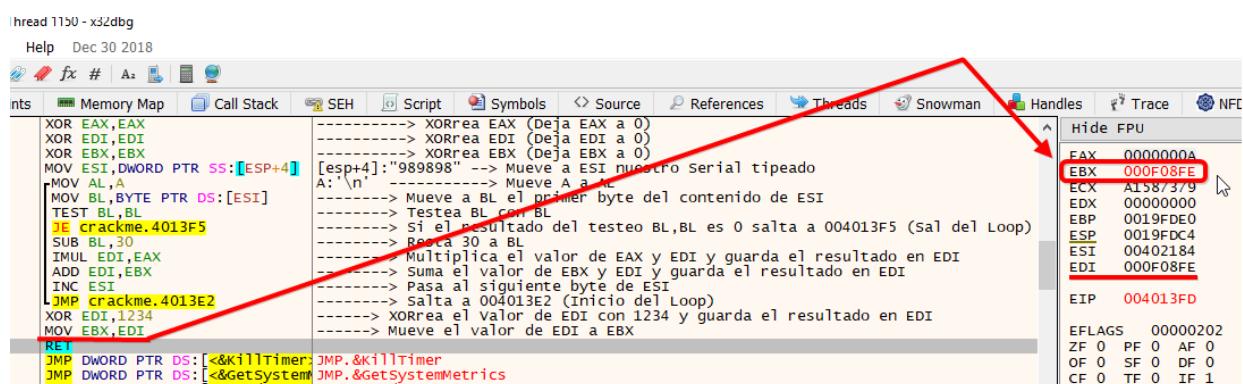
Ahora nos espera un

MOV EBX, EDI

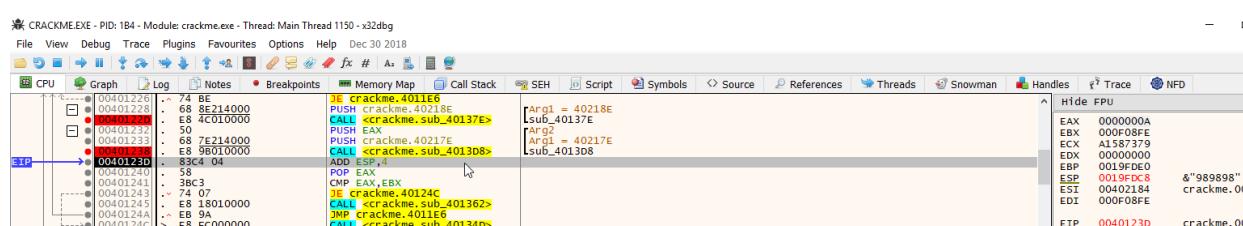
Va a MOVer el contenido de "EDI" a "EBX"

Si "EDI" = "000F08FE" pues "EBX" = "000F08FE"

Ejecutamos y eso es lo que ha hecho



Un trceo más para pasar el "RET" y salmos del "CALL"

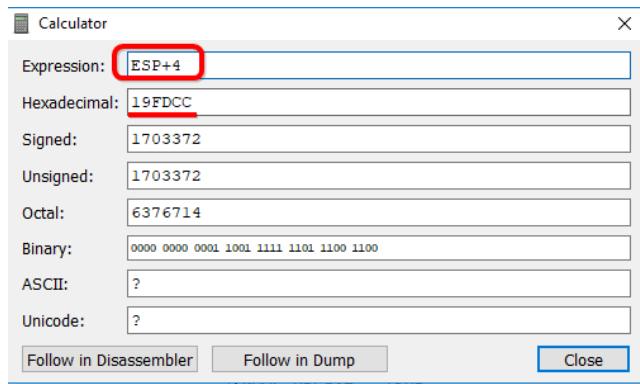


Parece que ya vemos el finalje,je,je.....

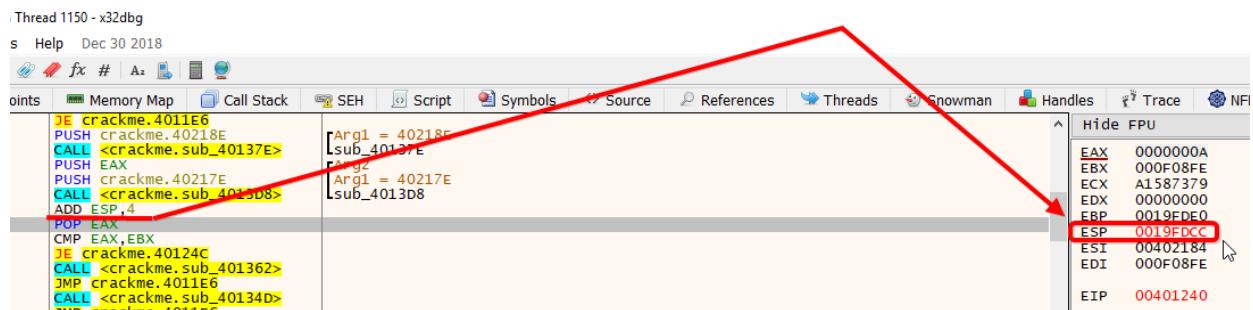
ADD ESP,4

Va a ADDicionar "4h" al valor del registro "ESP"

"ESP" = "0019FDCC" + "4" = "0019FDCC"

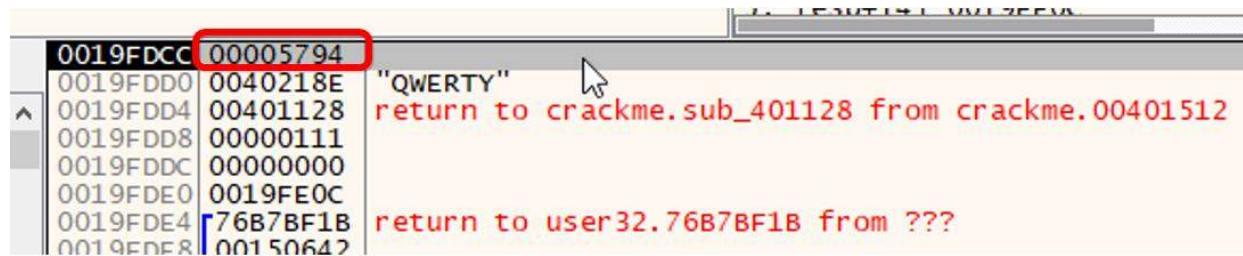


Traceamos y

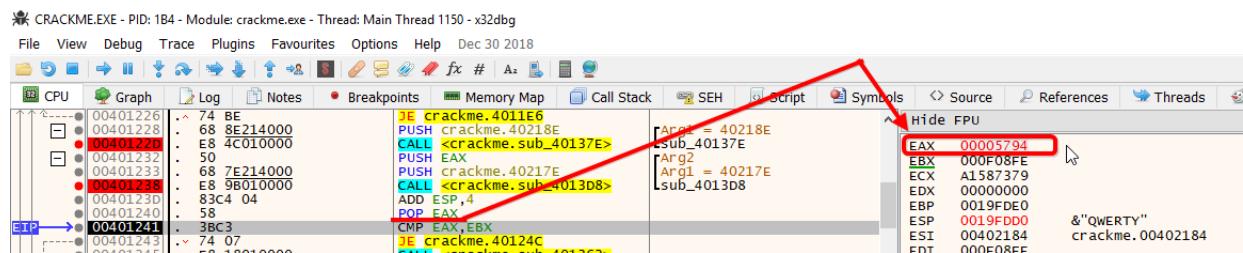


POP EAX

Va a coger el contenido de la primera carta del mazo del "Stack" (En nuestro caso "00005794", y lo va a pasar al valor del registro "ESI"



Traceamos, y eso es lo que ha hecho



Por fin llegamos a la Comparación

CMP EAX,EBX

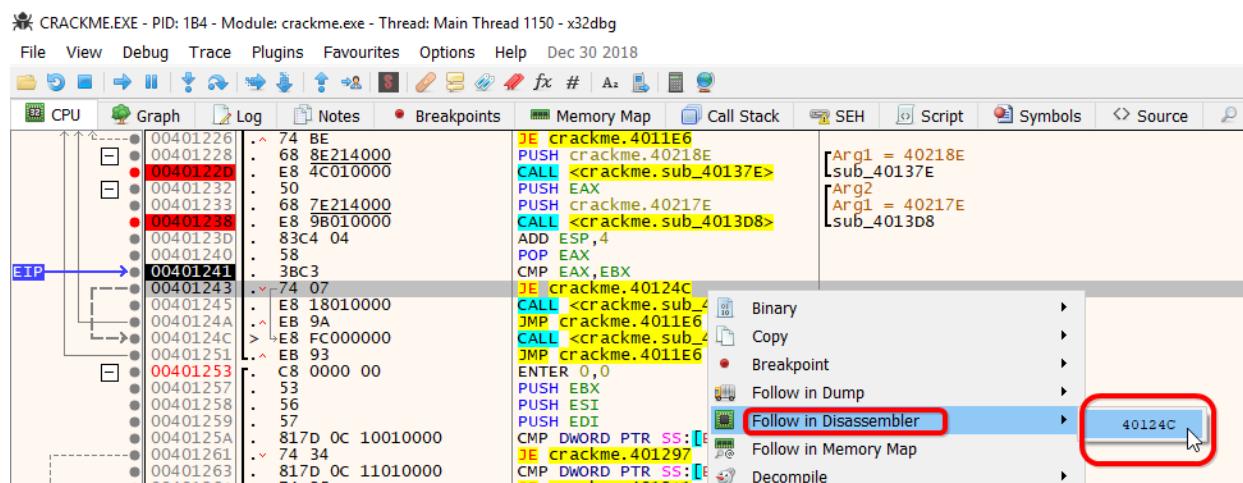
Donde va a comparar los registros "EAX" y "EBX" , y en la siguiente instrucción que es un salto condicional

JE crackme.40124C

Va a decidir que si el resultado de la comparación "EAX" y "EBX" es igual, nos mandará a la address "0040124C"

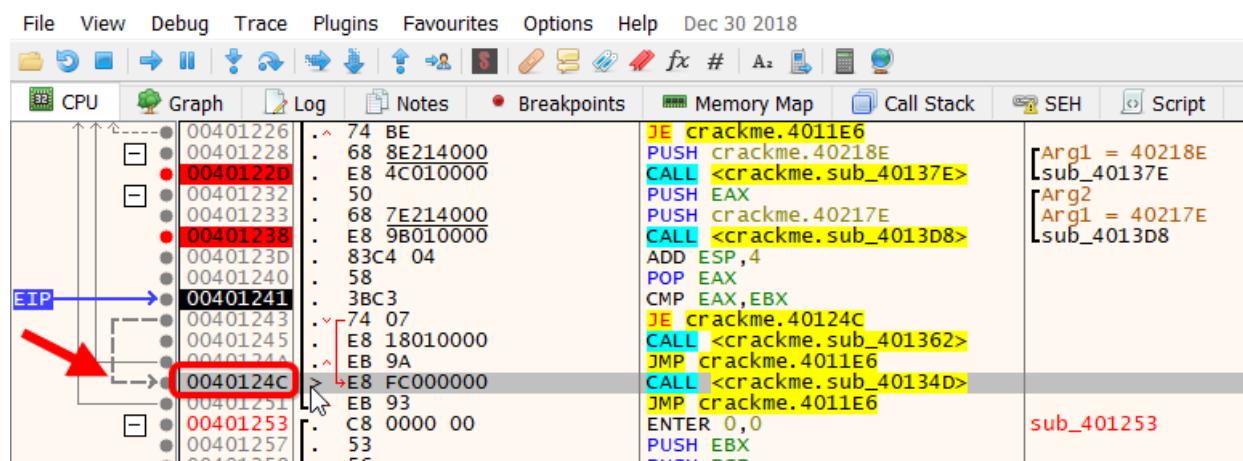
Ya sabemos de antemano que el resultado del "CMP" es diferente, pero ahora nos interesa saber que hay dentro de la address "0040124C" para cuando el resultado de la comparación sea igual.

Para ello nos posicionamos con el cursor sobre la instrucción **JE crackme.40124C**
Le damos click derecho de ratón y "**Follow in Disassembler>40124C**"



Y aparecemos en esa address

CRACKME.EXE - PID: 1B4 - Module: crackme.exe - Thread: Main Thread 1150 - x32dbg



Entramos dentro del "CALL" con Intro para curiosear y vemos el mensaje de "Chico Bueno" y un poco más abajo el de "Chico Malo"

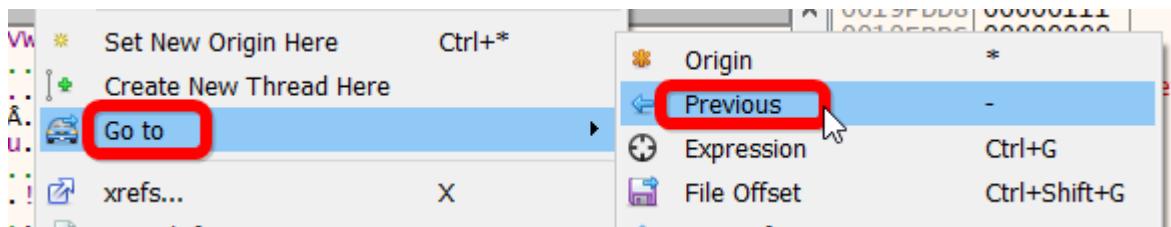
```

00401340: > 81D 10 F2030000 CMP DWORD PTR SS:[EBP+10], 3F2
    JNE crackme.401346
    PUSH 0
    PUSH DWORD PTR SS:[EBP+8]
    CALL KJMP.&SendDialog>
    MOV EAX, 0
    DMP crackme.401325
    MOV EAX, 0
    DMP crackme.401325
    PUSH 30
    CALL KJMP.&MessageBoxA>
    INT utype = MB_OK | MB_ICONEXCLAMATION | MB_ICONQUESTION | MB_ICONSTOP
    LPCTSTR lpcaption = "Good work!"
    LPCTSTR lpText = "Great work, mate! Now try the next crackMe!"

00401362: $ 6A 30
    PUSH 0
    CALL KJMP.&MessageBeep>
    INT utype = MB_OK
    MessageBeep
    LPCTSTR lpcaption = "No luck!"
    LPCTSTR lpText = "No luck there, mate!"
    PUSH DWORD PTR SS:[EBP+8]
    CALL KJMP.&MessageBoxA>
    RET
    INT utype = MB_OK | MB_ICONEXCLAMATION | MB_ICONQUESTION | MB_ICONSTOP
    LPCTSTR lpcaption = "No luck!"
    LPCTSTR lpText = "No luck there, mate!"
    HWND hwnd
    MessageBoxA
    RET
    C3

```

Salimos de la "CALL"



Y estamos de nuevo aquí

```

00401241: JE crackme.4011E6
    PUSH crackme.40218E
    CALL <crackme.sub_40137E>
    PUSH EAX
    PUSH crackme.40217E
    CALL <crackme.sub_4013D8>
    ADD ESP, 4
    POP EAX
    CMP EAX, EBX
    JE crackme.40124C
    CALL <crackme.sub_401362>
    JMP crackme.4011E6
    CALL <crackme.sub_40134D>
    JMP crackme.4011E6
    ENTER 0, 0
    PUSH EBX
    sub_401253

```

Y la conclusión a la que llegamos es que el salto condicional "JE" que se encuentra en la address "00401243" es el salto DECISIVO, donde si el resultado de la comparación de los registros "EAX" y "EBX" es igual, nos mandará a la address "0040124C" que ahora ya sabemos que es la zona de "Chico Bueno", de lo contrario, si el resultado de la comparación es diferente nos mandará a "Chico Malo".

O sea que esa sería la instrucción a invertir en el caso de que decidiéramos parchear, pero a estas alturas (llevamos 36 páginas de tute) con todo lo que hemos descubierto, creo que lo más elegante es descifrar y encontrar el Serial correcto para nuestro Name, y eso es lo que vamos a hacer.

Pero antes, levanto la cabeza del ordenador, me doy cuenta que llevo demasiado rato sentado, casi no me queda saliva para tragar, voy al frigorífico a por una cerveza Woll-Damm, (pequeña), le pego un buen trago, clavo mi mirada a la lámpara del techo y me digo a mí mismo:

CON EL NAME

- El Name tipeado no admite caracteres numéricos
- Si encuentra caracteres alfa (letras) minúsculas, las pasa a MAYÚSCULAS.

- Despues hace la suma con sus valores hexadecimales
- Al resultado de la suma lo Xorrea con 5678

CON EL SERIAL

- Convierte el Serial tipeado a hexadecimal
- Al resultado de la conversión lo Xorrea con 1234
- Al resultado le suma 4

LOS RESULTADOS DE AMBAS OPERACIONES "NAME" y "SERIAL" DEBEN SER IGUALES.

Por lo que :

Name = QwErTy

```

Q   W   E   R   T   Y
51h 57h 45h 52h 54h 59h
51h+57h+45h+52h+54h+59h=1ECh
1ECh Xor 5678 = 5794h Xor 1234 = 45A0h
45A0h + 4h = 45A4h = 17828d Serial

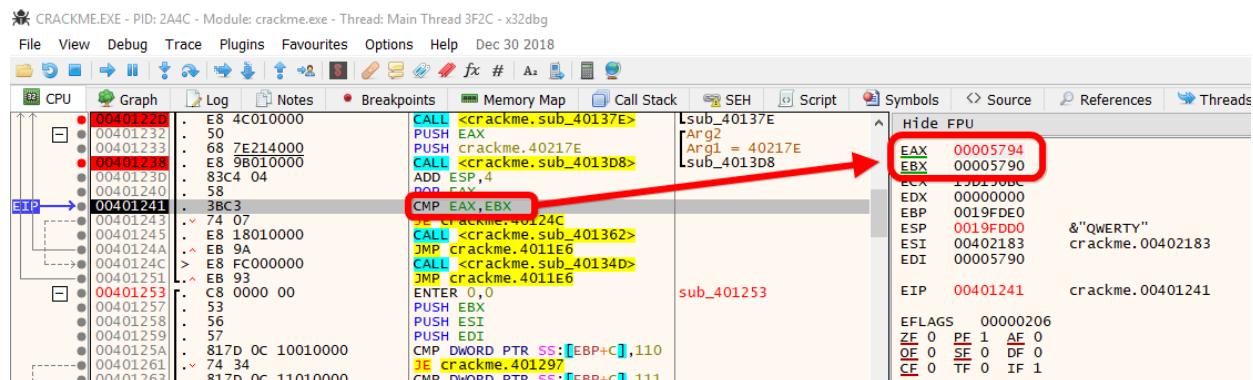
```

Probamos con

Name "QwErTy"

Serial "17828"

Y cuando llegamos a la comparación vemos que los resultados desafortunadamente NO son iguales, ya que la diferencia de los valores de "EAX" y "EBX" es de "4" (QUE CARAJO HE HECHO MAL....)



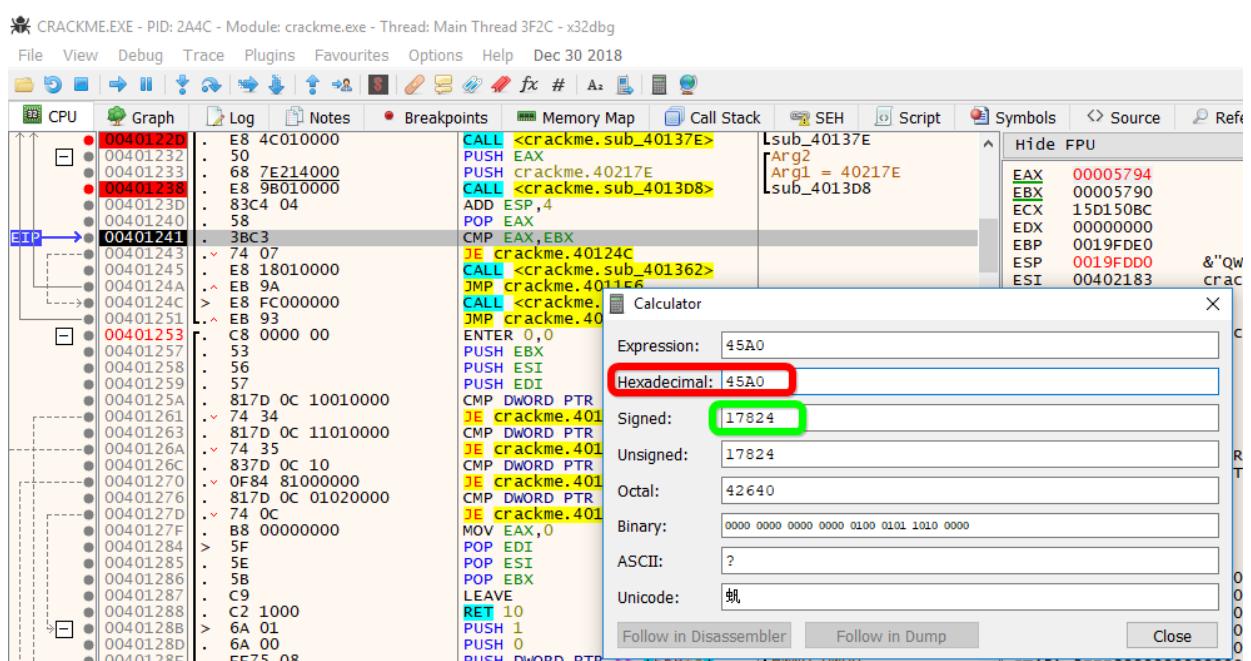
Pues parece que la solución está en que NO LE SUMAMOS "4" y listo.

Name = QwErTy

```

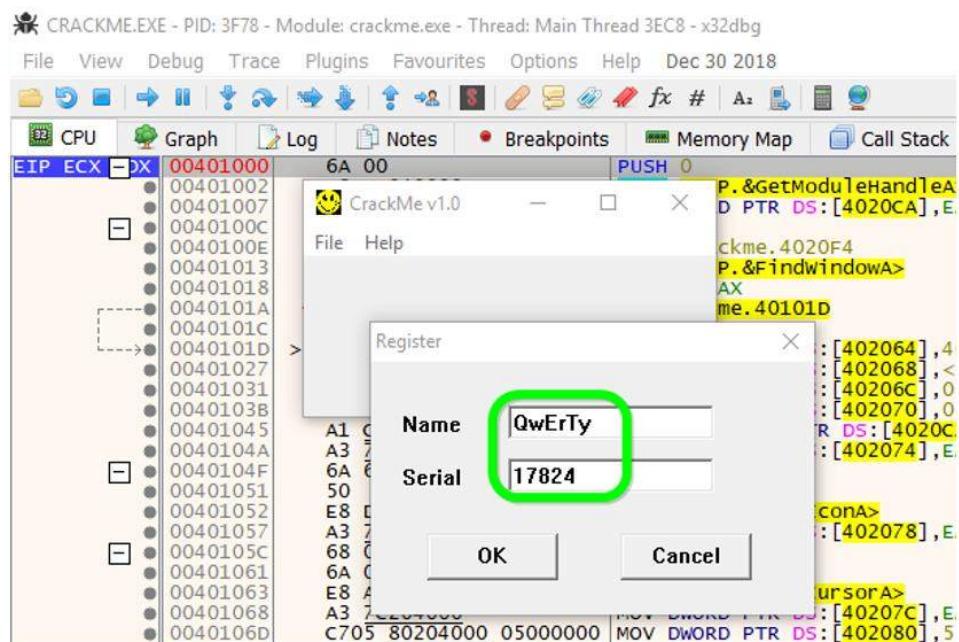
Q   W   E   R   T   Y
51h 57h 45h 52h 54h 59h
51h+57h+45h+52h+54h+59h=1ECh Xor 5678 = 5794h Xor 1234 = 45A0h
45A0h = 17824d = Serial

```

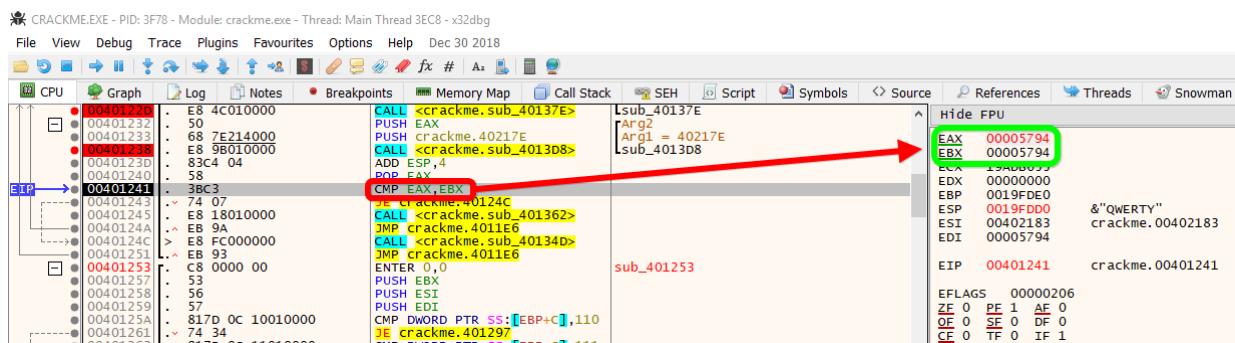


Reiniciamos y ahora probamos con

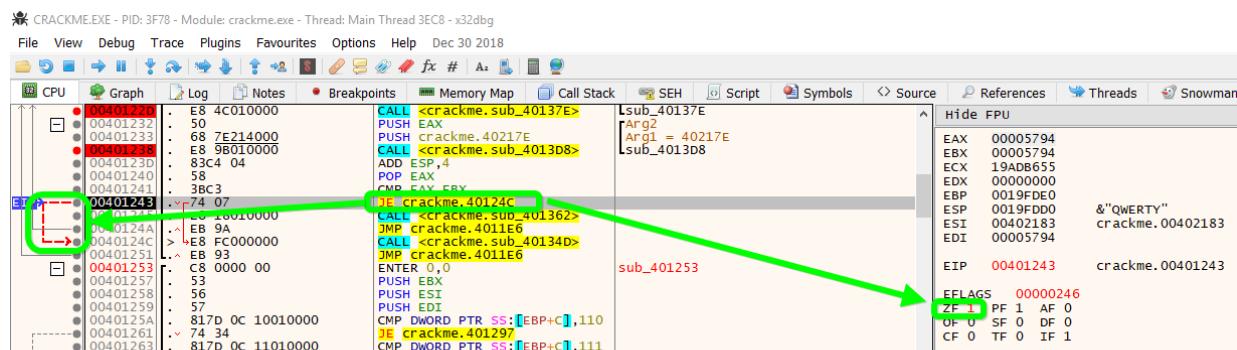
Name "QwErTy"
Serial "17824"



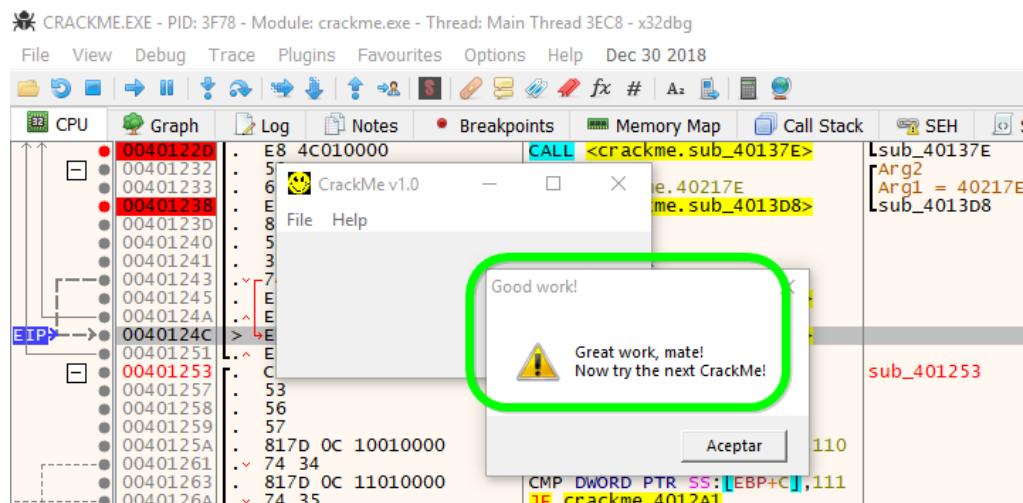
Llegamos a la comparación y por fin..... nuestro rostro presenta una agradable sonrisa que nos llega de oreja a oreja al ver que los resultados son iguales



Un trazo más para ejecutar la instrucción y vemos como el Flag ZF levanta bandera



Ejecutamos también el salto condicional "JE" y el "CALL" al que nos lleva a la address "0040124C", y felizmente nos salta el mensaje de "Chico Bueno"



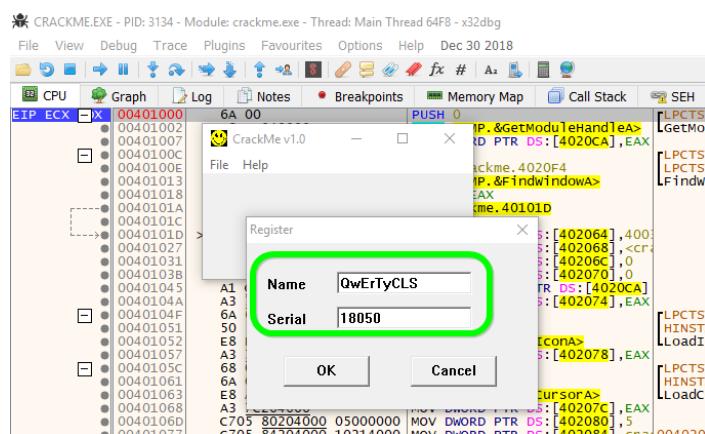
O sea que la solución del algoritmo está en hacer un Xor 5678 a la suma de los valores Hexadecimales de los caracteres alfa (una vez pasados a mayúsculas) de nuestro Name, y a su resultado le hacemos un Xor 1234, por último pasamos la conversión de Hexadecimal a Decimal y obtenemos el Serial correcto

Probamos ahora con

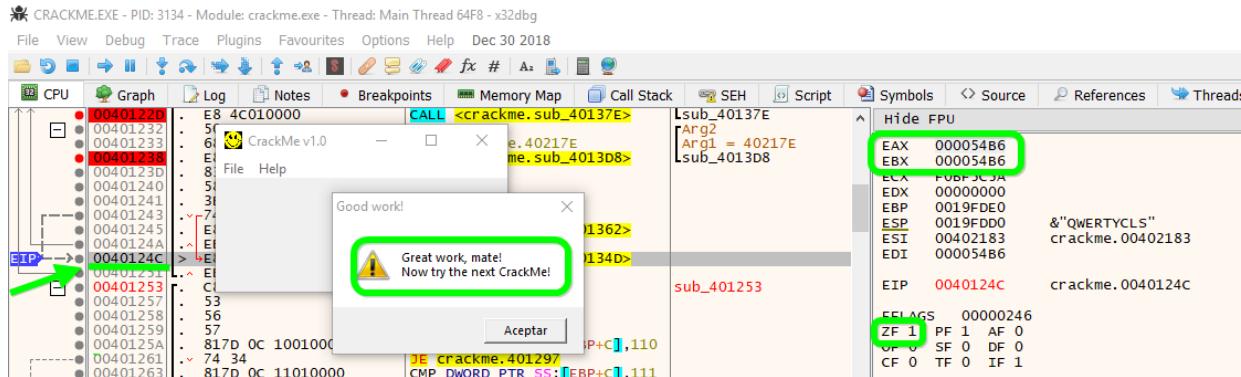
Name = "QwErTyCLS"

```

Q W E R T Y C L S
51h 57h 45h 52h 54h 59h 43h 4Ch 53h
51h+57h+45h+52h+54h+59h+43h+4Ch+53h= 2CEh
2CEh Xor 5678 = 54B6h Xor 1234 = 4682h
4682h = 18050d Serial
  
```



Y



Me encantaaaaaa

Hasta el próximo tute.

.....MISIÓN CUMPLIDA.....



Mis agradecimientos infinitos a

CracksLatinoS

Dime y lo olvido, enséñame y lo recuerdo, involúcrame y lo aprendo

Benjamin Franklin

Salu2

QwErTy CLS

2 de Febrero de 2019