

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 29

Creo que con lo que hemos visto sobre Visual Basic ya hemos completado lo básico, los que quieren profundizar en el tema pueden leer estos tutes de crackslatinos que completaran sus conocimientos y no valdría la pena repetirlos aquí pues están muy bien hechos.

<http://www.ricnar456.dyndns.org/CRACKING/NUEVO%20CURSO/TEORIAS/TEORIAS%20POR%20TEMA/VISUAL%20BASIC/>

user y pass: hola

En el cual encontraran muy buenos tutes de Visual Basic como los de COCO, los míos de los INVENCIBLES de La Calavera que son bastante difíciles y es una buena practica, los de ARAPUMK de los puntos mágicos de Visual Basic y muchos grandes tutes mas que creo innecesario volver a repetir en esta introducción pues ya fueron escritos y no descubriríamos nada nuevo, con esto les dejo el camino abierto a la lectura y profundización sobre el tema, y pasaremos al tema siguiente que es el P-CODE.

Los programas de VISUAL BASIC pueden ser de dos tipos, los NATIVE que son los que vimos hasta ahora y los P-CODE (PSEUDO CODIGO) que son los que veremos brevemente para introducirlos en el tema.

La diferencia principal radica en que los programas en NATIVE code, ejecutan líneas de código en la sección code del programa, mientras que los que son en P-CODE, si los abrimos con el OLLYDBG modificado para OEPs y VB y le ponemos un BPM ON ACCESS en la sección CODE, veremos que corren y que nunca para en la sección code en EJECUCION, salvo cuando ingresa a alguna api, y aun así es evidente que no tiene código ejecutable en dicha sección.

El desensamblado de un programa en P-CODE no sirve pues al no poseer código que se ejecute, no podemos interpretar nada.

Como ejemplo sencillo, les digo por ejemplo que siempre se ejecuta la dll de Visual y esta va leyendo valores de la sección code que le dicen lo que debe hacer, por ejemplo si lee un

le hará un Salto incondicional.

El le significara un salto condicional, y en la misma dll de visual Basic se ejecutara el mismo, o sea que los valores que va leyendo de la sección code le van indicando a la dll de Visual que debe hacer, aun sin ejecutar código en la sección code, solo leyendo valores de ella.

Ahora como somos guapos, agarraremos el cuchillo para luchar y acometeremos una tarea titánica que nadie ha hecho aun, trazar e interpretar un crackme en PCODE todo en OLLYDBG, opcode por opcode, jeje.

Veremos primero un crackme en el cual debemos hallar el serial, llamado clave1, y que se lo robamos al amigo JB DUC que tiene muy buenas teorías sobre el tema.

Por supuesto la mayoría del cracking en PCODE se realiza usando el excelente programa WKT debugger, el que quiere ver teorías con el mismo, puede ver las del mismo JB DUC o en el nuevo curso de crackslatinos también hay grandes teorías sobre P-CODE, aquí usaremos OLLYDBG y del EXDEC nos ayudaremos para ver los nombres de los OPCODES ya que la lista no es suministrada por nuestro amigo BILL GATES je.

```

00401030 $ 68 58124000 PUSH clave1.00401258
00401035 . E8 F0FFFFFF CALL <JMP.<MSUBUM50.#100>
0040103A . 0000 ADD BYTE PTR DS:[EAX],AL
0040103C . 0000 ADD BYTE PTR DS:[EAX],AL
0040103E . 0000 ADD BYTE PTR DS:[EAX],AL
00401040 . 3000 XOR BYTE PTR DS:[EAX],AL
00401042 . 0000 ADD BYTE PTR DS:[EAX],AL
00401044 . 40 INC EAX
00401045 . 0000 ADD BYTE PTR DS:[EAX],AL
00401047 . 0000 ADD BYTE PTR DS:[EAX],AL
00401049 . 0000 ADD BYTE PTR DS:[EAX],AL
0040104B . 0077 ADD BYTE PTR DS:[EAX],AL

```

Allí abrimos el crackme en un OLLYDBG común sin parchear y con los plugins para ocultarlo, como vimos en las partes anteriores.

A simple vista parece igual que un NATIVE inclusive el método del 4c se aplica y nos lleva a donde están las forms de la misma forma que en los natives (buena idea para quitar nags en P-CODE también, es usar el método del 4c cuando funcione)

Ahora que podemos ver de diferente:

Si vamos bajando desde el entry point no vemos líneas de código

```

00401000 . 66 FF0474 INC DWORD PTR FS:[ESP+ESI*2]
00401004 . FF21 JMP DWORD PTR DS:[ECX]
00401006 . 0F DB 0F
00401007 . 00 DB 00
00401008 . 03 DB 03
00401009 . 19 DB 19
0040100A . 78 DB 78
0040100B . FF DB FF
0040100C . 08 DB 08
0040100D . 78 DB 78
0040100E . FF DB FF
0040100F . 00 DB 00
00401010 . A0 DB A0
00401011 . 00 DB 00
00401012 . 00 DB 00
00401013 . 00 DB 00
00401014 . 6C DB 6C
00401015 . 74 DB 74
00401016 . FF DB FF
00401017 . 0A DB 0A
00401018 . 02 DB 02
00401019 . 00 DB 00
0040101A . 04 DB 04
0040101B . 00 DB 00
0040101C . FD DB FD
0040101D . 6B DB 6B
0040101E . 54 DB 54
0040101F . FF DB FF
00401020 . 5D DB 5D
00401021 . 04 DB 04

```

00401258=clave1.00401258

Address	Hex dump	ASCII
00401208	50 00 00 00 42 BD 48 72	P...BcHr
00401210	2C 43 D6 11 9C 10 A4 33	,Ci4&P&3
00401218	E1 D7 20 7A 00 00 00 00	pi z....
00401220	00 00 00 00 00 00 00 00	

Address	Hex dump	ASCII
0012FFC4	7C816D4F	RETURN
0012FFC8	7C920738	ntdll.
0012FFCC	FFFFFFFF	
0012FFD0	7FFDF000	
0012FFD4	8054A938	

Solo basura de ese estilo y que si la forzamos a analizar también da cosas sin sentido, recordemos que en un Visual Basic native si bajamos desde el entry point vemos algo así.

OllyDbg - Flipi1.exe - [CPU - main thread, module Flipi1]

```

00401D40 . 00 DB 00
00401D4E . 00 DB 00
00401D4F . 00 DB 00
00401D50 . FF DB FF
00401D51 . FF DB FF
00401D52 . FF DB FF
00401D53 . FF DB FF
00401D54 . 00 DB 00
00401D55 . 00 DB 00
00401D56 . 00 DB 00
00401D57 . 00 DB 00
00401D58 . 5C1A4000 DD Flipi1.00401A5C
00401D5C . EC194000 DD Flipi1.004019EC
00401D60 . D4324000 DD Flipi1.004032D4
00401D64 . 5C1A4000 DD Flipi1.00401A5C
00401D68 . 841A4000 DD Flipi1.00401A84
00401D6C . D8324000 DD Flipi1.004032D8
00401D70 . 5C1A4000 DD Flipi1.00401A5C
00401D74 . 9C1A4000 DD Flipi1.00401A9C
00401D78 . DC324000 DD Flipi1.004032DC
00401D7C . 5C1A4000 DD Flipi1.00401A5C
00401D80 . 841A4000 DD Flipi1.00401A84
00401D84 . E0324000 DD Flipi1.004032E0
00401D88 . CC INT3
00401D89 . CC INT3
00401D8A . CC INT3
00401D8B . CC INT3
00401D8C . CC INT3

```

Basura parecida pero si continuamos bajando

```
0040109B CC INT3
0040109C CC INT3
0040109D CC INT3
0040109E CC INT3
0040109F CC INT3
004010A0 > 55 PUSH EBP
004010A1 . 8BEC MOV EBP,ESP
004010A3 . 83EC 0C SUB ESP,0C
004010A6 . 68 B6104000 PUSH <JMP,&MSUBUM60.__vbaExceptionHandler>
004010AB . 64:A1 000000 MOV EAX,DWORD PTR FS:[0]
004010B1 . 50 PUSH EAX
004010B2 . 64:8925 0000 MOV DWORD PTR FS:[0],ESP
004010B9 . 81EC A0000000 SUB ESP,0A0
004010BF . 53 PUSH EBX
004010C0 . 56 PUSH ESI
004010C1 . 57 PUSH EDI
004010C2 . 8965 F4 MOV DWORD PTR SS:[EBP-C],ESP
004010C5 . C745 F8 9810 MOV DWORD PTR SS:[EBP-8],Flipi1.0040109
004010CC . 8B75 08 MOV ESI,DWORD PTR SS:[EBP+8]
004010CF . 8BC6 MOV EAX,ESI
004010D1 . 83E0 01 AND EAX,1
004010D4 . 8945 FC MOV DWORD PTR SS:[EBP-4],EAX
004010D7 . 83E6 FE AND ESI,FFFFFFFF
004010DA . 56 PUSH ESI
004010DB . 8975 08 MOV DWORD PTR SS:[EBP+8],ESI
004010DE . 8B0E MOV ECX,DWORD PTR DS:[ESI]
004010E0 . FF51 04 CALL DWORD PTR DS:[ECX+4]
004010E3 . 8B16 MOV EDX,DWORD PTR DS:[ESI]
004010E5 . 330B XOR EBX,EBX
004010E7 . 56 PUSH ESI
004010E8 . 895D E4 MOV DWORD PTR SS:[EBP-1C],EBX
004010EB . 895D E0 MOV DWORD PTR SS:[EBP-20],EBX
004010EE . 895D D0 MOV DWORD PTR SS:[EBP-30],EBX
004010F1 . 895D C0 MOV DWORD PTR SS:[EBP-40],EBX
004010F4 . 895D B0 MOV DWORD PTR SS:[EBP-50],EBX
004010F7 . 895D A0 MOV DWORD PTR SS:[EBP-60],EBX
004010FA . 895D 90 MOV DWORD PTR SS:[EBP-70],EBX
004010FD . 895D 80 MOV DWORD PTR SS:[EBP-80],EBX
004010E0 . FF32 00030000 CALL DWORD PTR DS:[EDX+300]
004010E6 . 50 PUSH EAX
004010E7 . 8D45 E0 LEA EAX,DWORD PTR SS:[EBP-20]
004010E8 . 50 PUSH EAX
004010E9 . FF15 20104000 CALL DWORD PTR DS:[<&MSUBUM60.__vbaObjSet]
004010E1 . 8BF8 MOV EDI,EAX
004010E3 . 8D55 E4 LEA EDX,DWORD PTR SS:[EBP-1C]
004010E6 . 52 PUSH EDX
004010E7 . 57 PUSH EDI
004010E8 . 8B0F MOV ECX,DWORD PTR DS:[EDI]
004010EA . FF91 A0000000 CALL DWORD PTR DS:[ECX+A0]
004010E0 . 3BC3 CMP EAX,EBX
004010E2 . DBE2 FCLEX
```

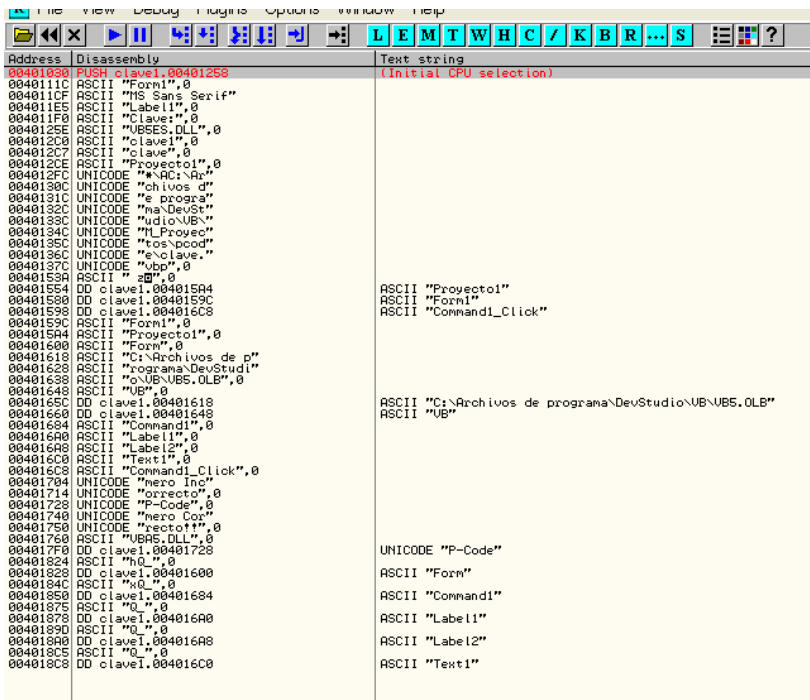
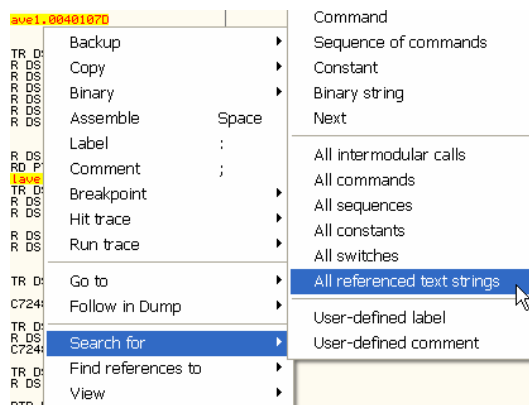
Encontramos código y es bastante largo continua hasta casi el fin de la sección todo puro código ejecutable, en cambio en el P-CODE salvo alguna línea suelta que de casualidad por la cercanía de bytes, OLLYDBG interpreta como alguna instrucción, es pura basura.

Volvamos al crackme de PCODE

```
00401000 .- FF25 68304000 JMP DWORD PTR DS:[<&MSUBUM50.#581>]
00401006 .- FF25 50304000 JMP DWORD PTR DS:[<&MSUBUM50.#595>]
0040100C .- FF25 60304000 JMP DWORD PTR DS:[<&MSUBUM50.__vbaExceptionHandler>]
00401012 .- FF25 5C304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK_QueryInterface>]
00401018 .- FF25 54304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK_AddRef>]
0040101E .- FF25 58304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK_Release>]
00401024 .- FF25 4C304000 JMP DWORD PTR DS:[<&MSUBUM50.MethCallEngine>]
0040102A .- FF25 64304000 JMP DWORD PTR DS:[<&MSUBUM50.#100>]
00401030 . 68 58124000 PUSH clave1.00401258
00401035 . E8 F0FFFFFF CALL <JMP,&MSUBUM50.#100>
0040103A . 0000 ADD BYTE PTR DS:[EAX],AL
0040103C . 0000 ADD BYTE PTR DS:[EAX],AL
0040103E . 0000 ADD BYTE PTR DS:[EAX],AL
```

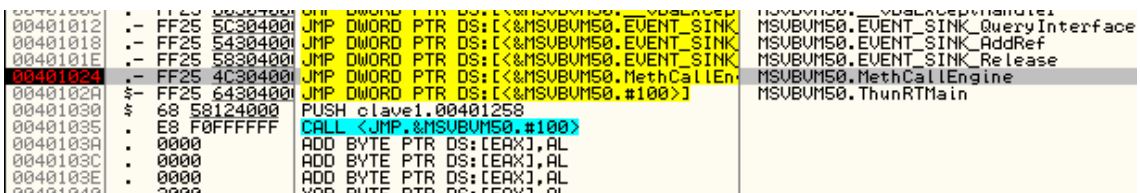
Otra característica es la api esa MethCallEngine la cual encontramos en los crackmes hechos en P-CODE, así que el primer paso, identificar si el programa es P-CODE o no, ya sabemos como hacerlo, tanto sea mirando si hay código ejecutable en la sección CODE o por la api que acabamos de mencionar.

Lo primero que se nos ocurre hacer es ver si hallamos STRINGS



No ayuda mucho eso en este caso, aunque en algún otro podría ayudar.

Bueno pongamos un BP tanto en el JMP de la api MethCallEngine como en la misma api directamente, por si es llamada sin usar el JMP en forma directa.

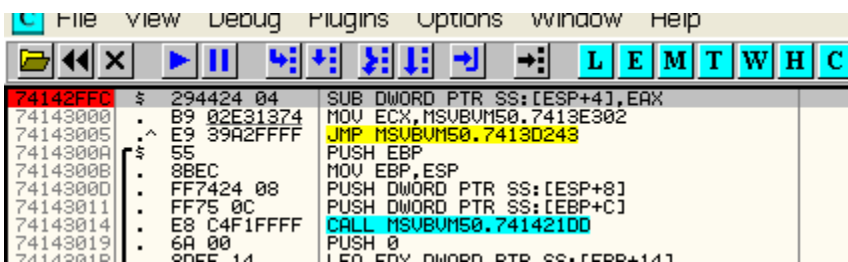


Buscamos arriba del Entry Point y encontramos rápidamente el JMP a la api MethCallEngine, y situándonos encima y haciendo click derecho FOLLOW vamos a la misma, donde también ponemos un BP.

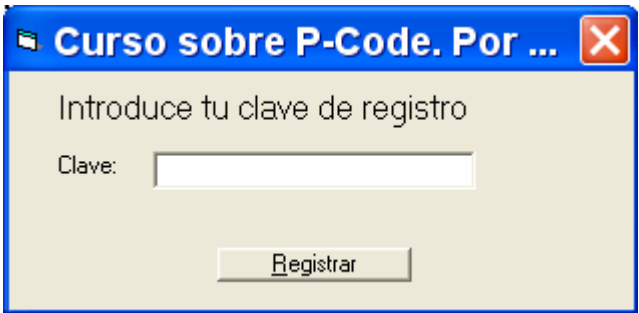
```

00401012 .- FF25 5C304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK] MSUBUM50.EVENT_SINK
00401016 .- FF25 54304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK] MSUBUM50.EVENT_SINK
0040101E .- FF25 58304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK] MSUBUM50.EVENT_SINK
00401024 .- FF25 4C304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK] MSUBUM50.EVENT_SINK
0040102A .- FF25 64304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK] MSUBUM50.EVENT_SINK
00401030 $ 68 58124000 PUSH clave1.0040125 Backup
00401035 . E8 F0FFFFFF CALL <JMP.&MSUBUM50> Copy
0040103A . 0000 ADD BYTE PTR DS:[EAX] Binary
0040103C . 0000 ADD BYTE PTR DS:[EAX] Space
0040103E . 0000 ADD BYTE PTR DS:[EAX] Label
00401040 . 0000 XOR BYTE PTR DS:[EAX] ;
00401042 . 0000 ADD BYTE PTR DS:[EAX] Comment
00401044 . 48 INC EAX Breakpoint
00401045 . 0000 ADD BYTE PTR DS:[EAX] :
00401047 . 0000 ADD BYTE PTR DS:[EAX] ;
00401049 . 0000 ADD BYTE PTR DS:[EAX] Hit trace
0040104B . 0077 BD ADD BYTE PTR DS:[EAX] Run trace
0040104E . 48 DEC EAX
0040104F . 72 2C JB SHORT clave1.004
00401051 . 43 INC EBX
00401052 . D6 SALC
00401053 . 119C10 A433E ADC DWORD PTR DS:[EAX]
0040105A . 207A 00 AND BYTE PTR DS:[EAX]
0040105D . 0000 ADD BYTE PTR DS:[EAX]
0040105F . 0000 ADD BYTE PTR DS:[EAX]
00401061 . 0001 ADD BYTE PTR DS:[EAX]
00401063 . 0000 ADD BYTE PTR DS:[EAX]
00401065 . 0000 ADD BYTE PTR DS:[EAX]
00401067 . 0000 ADD BYTE PTR DS:[EAX]

```

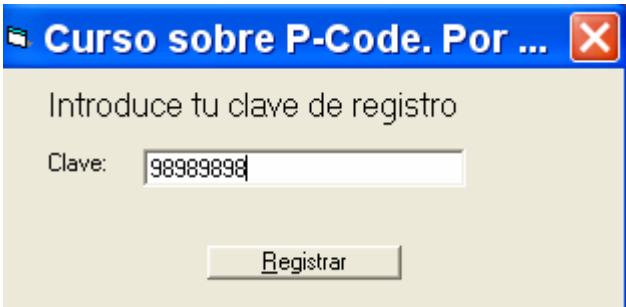


Ahora damos RUN.



Vemos que aparece la ventana para ingresar el serial falso antes de parar en la api, lo cual es lógico porque la creación de las ventanas y todo esto lo realiza en la misma forma que en VB nativo, la misma dll de visual sin ejecutar código del programa, al leer las forms que vemos con el método 4c.

Ahora ingresamos el serial falso.



Y apretamos REGISTRAR

```

00401024  FF25 4C304000 JMP DWORD PTR DS:[<&MSUBUM50.MethCallEn MSUBUM50.MethCallEngine
0040102A  FF25 64304000 JMP DWORD PTR DS:[<&MSUBUM50.#100>] MSUBUM50.ThunRTMain
00401030  68 58124000 PUSH c_lave1.00401258
00401035  E8 F0FFFFFF CALL <JMP.&MSUBUM50.#100>
0040103A  0000 ADD BYTE PTR DS:[EAX],AL
0040103C  0000 ADD BYTE PTR DS:[EAX],AL
0040103E  0000 ADD BYTE PTR DS:[EAX],AL
00401040  3000 XOR BYTE PTR DS:[EAX],AL
00401042  0000 AND BYTE PTR DS:[EAX],0

```

Para en el JMP que iniciara la parte verdadera de P-CODE.

```

74142FFC  294424 04 SUB DWORD PTR SS:[ESP+4],EAX
74143000  B9 02E31374 MOV ECX,MSUBUM50.7413E302
74143005  E9 39A2FFFF JMP MSUBUM50.7413D243
7414300A  55 PUSH EBP
7414300B  8BFC MOV EBP,ESP

```

Allí entra a la api veamos que va haciendo

```

7413D243  55 PUSH EBP
7413D244  8BEC MOV EBP,ESP
7413D246  83EC 78 SUB ESP,78
7413D249  53 PUSH EBX
7413D24A  56 PUSH ESI
7413D24B  57 PUSH EDI
7413D24C  8BDA MOV EBX,EDX
7413D24E  895D B0 MOV DWORD PTR SS:[EBP-50],EBX
7413D251  894D 94 MOV DWORD PTR SS:[EBP-6C],ECX
7413D254  8B15 64F0147 MOV EDX,DWORD PTR DS:[7414F064]
7413D25A  0BD2 OR EDX,EDX
7413D25C  0F85 09630000 JNZ MSUBUM50.74143568
7413D262  8B15 6CF0147 MOV EDX,DWORD PTR DS:[7414F06C]
7413D268  8B3B MOV EDI,DWORD PTR DS:[EBX]

```

Allí comienza

Ahora si abrimos el mismo crackme con el exdec que es un desensamblador de PCODE para ayudarnos, vemos que nos muestra esto.

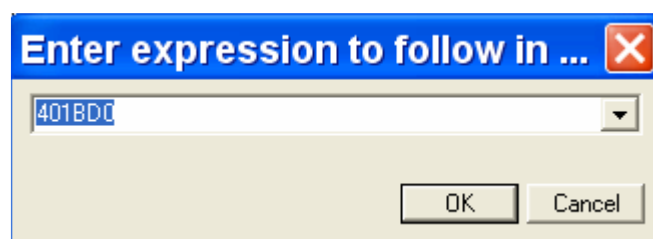
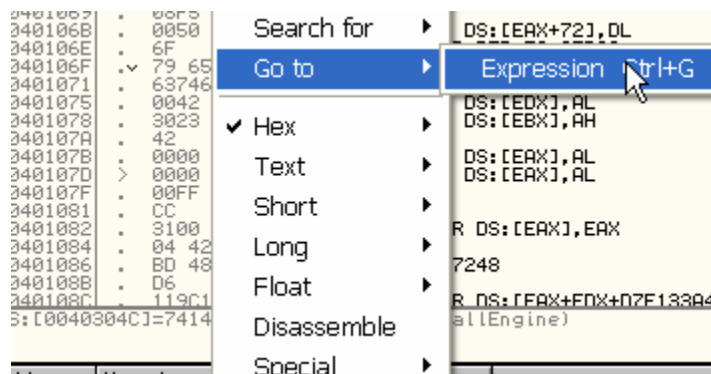
```

Email josephco_@hotmail.com with any errors or problemsa0dThis program do

Proc: 401c98
401BD0: 04 FLdRfVar      local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd        text
401BD7: 19 FStAdFunc      local_0088
401BDA: 08 FLdPr         local_0088
401BDD: 0d VCallHresult   get_ipropTEXTEDIT
401BE2: 6c ILdRf         local_008C
401BE5: 1b LitStr:        "
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str      local_008C
401BED: 1a FFree1Ad       local_0088
401BF0: 1c BranchF:       401BF6
401BF3: 1e Branch:        401c94
401BF6: Lead3/c1 LitVarl4: [ local_3BE500AC ] 0x3c41a (246810)
401BFE: Lead1/f6 FStVar   local_009C
401C02: 04 FLdRfVar      local_008C
401C05: 21 FLdPrThis
401C06: 0f VCallAd        text
401C09: 19 FStAdFunc      local_0088

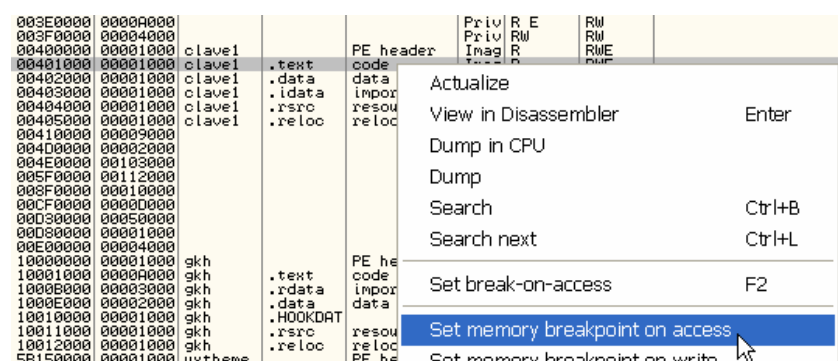
```

O sea el primer byte que lee es un 04 y esta en la posición 401BD0 ese sería el primer byte leído, eso debe ocurrir no muy lejos de aquí, así que pongamos un BPM ON ACCESS en dicho byte.



Address	Hex dump	ASCII
00401BD0	04 74 FF 21 0F 00 03 19	!t f%.#↓
00401BD8	78 FF 08 78 FF 0D A0 00	x 8x .a.
00401BE0	00 00 7C 74 FF 1B 01 00	..lt +0.
00401BE8	FB 30 7F 74 FF 1A 78 FF	!0/t +x
00401BF0	1C 26 00 1E C4 00 FE C1	L&.▲-.-↓
00401BF8	54 FF 1A C4 03 00 FC F6	T +->0.7÷
00401C00	54 FF 04 74 FF 21 0F 00	d ▲+ f%

Ese sería el primer byte que leería, cuando para, estaríamos en el inicio, de cualquier forma podemos llegar a él sin el EXDEC, una vez que para en el BP de la api MethCallEngine, ponemos un BPM ON ACCESS en la sección code.



Y damos RUN vemos que para varias veces

7413D254	. 8B15 64F0147	MOV EDX,DWORD PTR DS:[7414F064]	
7413D25A	. 0B02	OR EDX,EDX	
7413D25C	. 0F85 0963000	JNZ MSUBUM50.74143568	
7413D262	. 8B15 6CF0147	MOV EDX,DWORD PTR DS:[7414F06C]	
7413D268	. 8B38	MOV EDI,DWORD PTR DS:[EBX]	clave1.0040176C
7413D26A	. 8B77 34	MOV ESI,DWORD PTR DS:[EDI+34]	
7413D26D	. 8975 AC	MOV DWORD PTR SS:[EBP-54],ESI	
7413D270	. 8B77 04	MOV ESI,DWORD PTR DS:[EDI+4]	
7413D273	. 8B76 14	MOV ESI,DWORD PTR DS:[ESI+14]	
7413D276	. 8B76 0C	MOV ESI,DWORD PTR DS:[ESI+C]	
7413D279	. 8975 D4	MOV DWORD PTR SS:[EBP-2C],ESI	
7413D27C	. 8955 BC	MOV DWORD PTR SS:[EBP-44],EDX	
7413D27F	. C745 B8 0000	MOV DWORD PTR SS:[EBP-48],0	
7413D286	. 81F9 02E3137	CMP ECX,MSUBUM50.7414E302	
7413D28C	. 0F84 9400000	JE MSUBUM50.7414D326	

Pero la única que lee del contenido de ESI que apunta al byte susodicho, la hallaremos enseguida, luego de unas cuantas veces que pare. (en mi maquina conté 10 exactas)

```

7413031C . 8A06 MOV AL, BYTE PTR DS:[ESI]
7413031E . 46 INC ESI
7413031F . FF2485 94ED1 JUMP DWORD PTR DS:[EAX*4+7413ED94]
74130326 > 66:F743 0C 10 TEST WORD PTR DS:[EBX+C],10
7413032C > 0F85 5962000 JNZ MSUBUM50.7413558B

```

La primera vez que para y lee un byte de [ESI] y lo mueve a AL es donde comienza a leer el primer opcode de P-CODE, de esa forma podemos encontrar el primer byte sin siquiera usar el EXDEC.

Como vemos los siguientes opcodes que muestra el exdec están a continuación del anterior.

```

Proc: 401c98
401BD0: 04 FLdRfVar local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd text
401BD7: 19 FStAdFunc local_0088
401BDA: 08 FLdPr local_0088
401BDD: 0d VCallHresult get_ipropTEXTEDIT
401BE2: 6c ILdRf local_008C
401BE5: 1b LitStr: ""
401BE8: Lead0/30 EqStr

```

Address	Hex dump	ASCII
00401BD0	04 74 FF 21 0F 00 03 19	04 74 FF 21 0F 00 03 19
00401BD8	78 FF 08 78 FF 00 A0 00	78 FF 08 78 FF 00 A0 00
00401BE0	00 00 6C 74 FF 1B 01 00	00 00 6C 74 FF 1B 01 00
00401BE8	FB 30 2F 74 FF 1A 78 FF	FB 30 2F 74 FF 1A 78 FF
00401BF0	1C 26 00 1E C4 00 FE C1	1C 26 00 1E C4 00 FE C1
00401BF8	54 FF 1A C4 03 00 FC F6	54 FF 1A C4 03 00 FC F6
00401C00	64 FF 04 74 FF 21 0F 00	64 FF 04 74 FF 21 0F 00
00401C08	03 19 78 FF 08 78 FF 00	03 19 78 FF 08 78 FF 00
00401C10	A0 00 00 00 6C 74 FF 0A	A0 00 00 00 6C 74 FF 0A
00401C18	02 00 04 00 FD 68 54 FF	02 00 04 00 FD 68 54 FF
00401C20	5D 04 64 FF FB 40 2F 74	5D 04 64 FF FB 40 2F 74

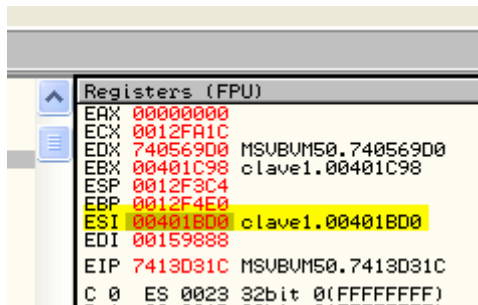
Como ven están en orden, aunque tengan en medio, los parámetros que necesita cada opcode para ejecutarse.

```

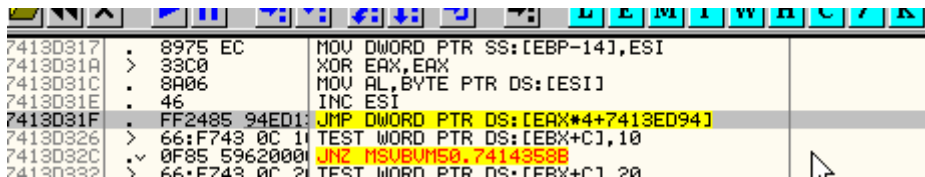
74130312 . 03F3 ADD ESI,EBX
74130314 . 8975 A8 MOV DWORD PTR SS:[EBP-58],ESI
74130317 . 8975 EC MOV DWORD PTR SS:[EBP-14],ESI
7413031A > 33C0 XOR EAX,EAX
7413031C . 8A06 MOV AL, BYTE PTR DS:[ESI]
7413031E . 46 INC ESI
7413031F . FF2485 94ED1 JUMP DWORD PTR DS:[EAX*4+7413ED94]
74130326 > 66:F743 0C 10 TEST WORD PTR DS:[EBX+C],10
7413032C > 0F85 5962000 JNZ MSUBUM50.7413558B
74130332 > 66:F743 0C 20 TEST WORD PTR DS:[EBX+C],20
74130338 > 75 1D JNZ SHORT MSUBUM50.7413D357
7413033A . 0BC0 OR EAX,EAX
7413033C . B8 00E00000 MOV EAX,0E0000
74130341 > 74 14 JE SHORT MSUBUM50.7413D357

```

Como vimos allí lee el primer byte



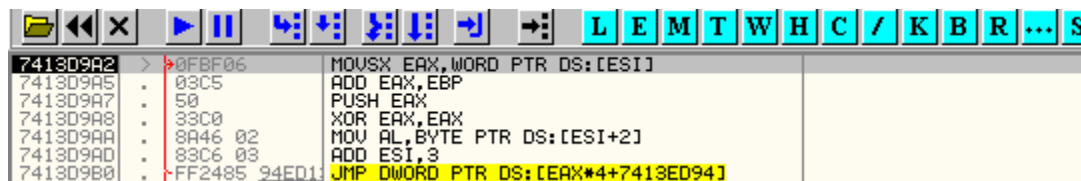
Lo que vemos es que ESI apunta al byte actual que se esta ejecutando, pero ya en la línea siguiente antes de hacer ninguna operación lo incrementa en 1, para leer los parámetros del opcode.



Luego llega siempre a un JMP indirecto que nos lleva a las líneas que ejecutarán el OPCODE, en este caso 04 como vimos en el EXDEC.

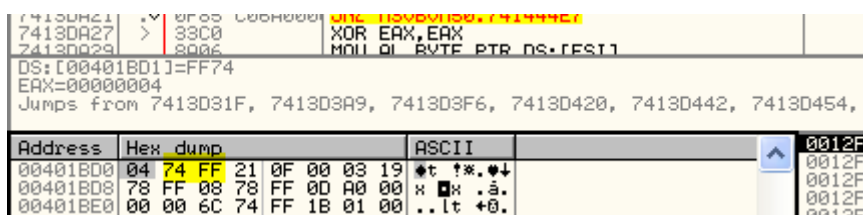
401BD0: 04 FLdRfVar local_008C

Veremos que hace este opcode tan misterioso entramos en el.



Allí vemos la ejecución del OPCODE 04 **FLdRfVar**, son unas pocas líneas de código, nada para asustarse jeje y siempre vemos que termina con un XOR EAX,EAX y a leer el siguiente opcode.

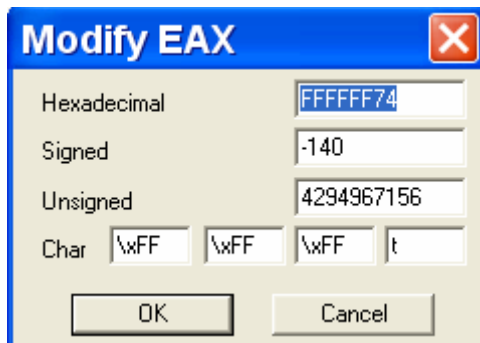
Lo primero que hace es cargar los parámetros del OPCODE que son los dos bytes siguientes al mismo.



Bueno los pasa a EAX y como es un MOVSX y el valor FF74 es negativo los completa con FFs como vimos en la parte de assembler, sigamos traceando.



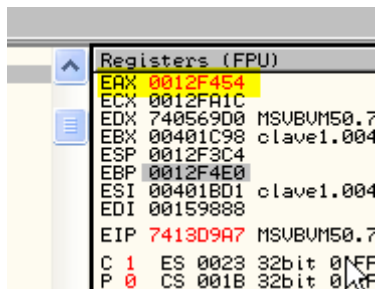
Ese valor es EAX es -8c, pues si hacemos doble click en el



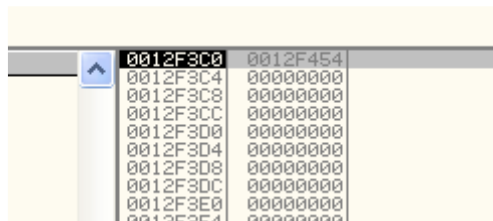
Nos muestra que vale -140 decimal que es -8c en hexa por lo tanto, como vemos el 8c que nos muestra el EXDEC aparece aquí.

401BD0: 04 FLdRfVar [local 008C](#)

En la siguiente línea suma ese valor que leyo de los parámetros con EBP



Y luego le hace PUSH a ese valor



O sea que esto es equivalente a un PUSH EBP- 8c esta mandando el stack una variable local, en mi maquina EBP es 12f4e0 si a eso le restamos 8c nos da 12f454 que es el valor que quedo en EAX y pushea.



Registers (FPU)	
EAX	0012F454
ECX	0012FA1C
EDX	740569D0 MSUBVM50.740569D0
EBX	00401C98 clave1.00401C98
ESP	0012F3C4
EBP	0012F4E0
ESI	00401BD1 clave1.00401BD1
EDI	00159888
EIP	7413D9A7 MSUBVM50.7413D9A7
C	1 ES 0023 32bit 0(FFFFFFFF)
P	0 CS 001B 32bit 0(FFFFFFFF)
A	0 SS 0023 32bit 0(FFFFFFFF)

Nada del otro mundo sigamos.

LEMIWH		
7413D9A2	> 0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413D9A5	. 03C5	ADD EAX,EBP
7413D9A7	. 50	PUSH EAX
7413D9A8	. 33C0	XOR EAX,EAX
7413D9AA	. 8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D9AD	. 83C6 03	ADD ESI,3
7413D9B0	. FF2485 34ED1	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9B7	> 8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]

Vemos que luego pone EAX a cero lo cual significa que la operación con este opcode ya finaliza y esta inicializando los registros para leer el siguiente, en la próxima línea ya lee el opcode siguiente.

Email josephco_@hotmail.com with any errors or prot

```

Proc: 401c98
401BD0: 04 FLdRfVar      local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd          text
401BD7: 19 FStAdFunc         local_0088
401BDA: 08 FLdPr             local_0088
401BDD: 0d VCallHresult      get_ipropTEXTEDIT
401BE2: 6c ILdRf             local_008C
401BE5: 1b LitStr:           ""
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str         local_008C
401BED: 1a FFree1Ad          local_0088
401BF0: 1c BranchF:          401BF6
401BF3: 1e Branch:           401c94
401BF6: Lead3/c1 LitVarl4:   [ local_3BE500AC ] 0
401BFE: Lead1/f6 FStVar      local_009C
401C02: 04 FLdRfVar          local_008C
401C05: 21 FLdPrThis
401C06: 0f VCallAd          text
401C09: 19 FStAdFunc         local_0088

```

El segundo opcode es el 21

Aquí lo lee

```

7413D9A7 . 50          PUSH EAX
7413D9A8 . 33C0       XOR EAX,EAX
7413D9AA . 8A46 02    MOV AL, BYTE PTR DS:[ESI+2]
7413D9AD . 83C6 03    ADD ESI,3
7413D9B0 . FF2485 24ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9B7 > 8B45 08    MOV EAX,DWORD PTR SS:[EBP+8]
7413D9BA . 8945 B4    MOV DWORD PTR SS:[EBP-4C],EAX
7413D9BD . 33C0       XOR EAX,EAX
7413D9BF . 8A06       MOV AL, BYTE PTR DS:[ESI]
7413D9C1 . 46         INC ESI
7413D9C2 . FF2485 24ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9C9 > 0FBF06    MOVSX EAX,WORD PTR DS:[ESI]
7413D9CC . 5B         POP EBX
7413D9CD . 66:891C28  MOV WORD PTR DS:[EAX+EBP],BX
7413D9D1 . 33C0       XOR EAX,EAX
7413D9D3 . 8A46 02    MOV AL, BYTE PTR DS:[ESI+2]
7413D9D6 . 83C6 03    ADD ESI,3
7413D9D9 . FF2485 24ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9E0 > 0FBF06    MOVSX EAX,WORD PTR DS:[ESI]
7413D9E3 . 5B         POP DWORD PTR DS:[EAX+EBP]
7413D9E6 . 33C0       XOR EAX,EAX
7413D9E8 . 8A46 02    MOV AL, BYTE PTR DS:[ESI+2]
7413D9EB . 83C6 03    ADD ESI,3
7413D9EE . FF2485 24ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9F5 > 0FBF06    MOVSX EAX,WORD PTR DS:[ESI]
7413D9F8 . D91C28     FSTP DWORD PTR DS:[EAX+EBP]
7413D9FB . DFE0       FSTSW AX
7413D9FD . A9 00      TEST AL,00
7413D9FF . 0F85 A1C700 JNZ MSUBUM50.7414A1A6
7413DA05 . 33C0       XOR EAX,EAX
7413DA07 . 8A46 02    MOV AL, BYTE PTR DS:[ESI+2]
7413DA0A . 83C6 03    ADD ESI,3
7413DA0D . FF2485 24ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413DA14 > 8B3C24     MOV EDI,DWORD PTR SS:[ESP]
7413DA17 . 66:8B07    MOV AX,WORD PTR DS:[EDI]
7413DA1A > 80E4 BF    AND AH,0BF
7413DA1D . 66:83F8 09 CMP AX,9
7413DA21 > 0F85 C0A000 JNZ MSUBUM50.741444E7
7413DA27 > 33C0       XOR EAX,EAX
7413DA29 . 8A06       MOV AL, BYTE PTR DS:[ESI]
DS:[00401BD3]=21 ('!')
AL=00

```

Address	Hex dump	ASCII
0012F3C0	0012F3C0	

Registers (FPU)	
EAX	00000021
ECX	0012FA1C
EDX	740569D0 MSUBU
EBX	00401C98 clave
ESP	0012F3C0
EBP	0012F4E0
ESI	00401BD1 clave
EDI	00159888
EIP	7413D9AD MSUBU
C 0	ES 0023 32bit

Lo mueve a AL como siempre.

```

File View Debug Plugins Options Window Help
[Icons] [L] [E] [M] [T] [W] [H] [C] [/] [K] [B] [R] ...

7413D9A2 > 0FBF06    MOVSX EAX,WORD PTR DS:[ESI]
7413D9A5 . 03C5      ADD EAX,EBP
7413D9A7 . 50        PUSH EAX
7413D9A8 . 33C0       XOR EAX,EAX
7413D9AA . 8A46 02    MOV AL, BYTE PTR DS:[ESI+2]
7413D9AD . 83C6 03    ADD ESI,3
7413D9B0 > FF2485 24ED1 JMP DWORD PTR DS:[EAX*4+7413ED94] MSUBUM50.7413D9B7
7413D9B7 > 8B45 08    MOV EAX,DWORD PTR SS:[EBP+8]
7413D9BA . 8945 B4    MOV DWORD PTR SS:[EBP-4C],EAX

```

Y ahora le suma a ESI el valor 3 para que quede apuntando a los parámetros de este opcode, luego llega al JMP indirecto, que va al código del OPCODE 21.

Veamos que hace buscando intensamente en google

'21, FLdPrThis

'Load reference pointer into item pointer.

Bueno hay aquí algunos punteros que no sabemos bien para que sirven, ni sirve mucho pero el tema es que carga un reference pointer y lo guarda en item pointer, en asm esto nos dice que lee un puntero del stack y lo mueve a otro lugar de nuestro stack.

Si seguimos traceando

```

7413D9B0 . FF2485 24ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9B7 > 8B45 08    MOV EAX,DWORD PTR SS:[EBP+8]
7413D9BA . 8945 B4    MOV DWORD PTR SS:[EBP-4C],EAX
7413D9BD . 33C0       XOR EAX,EAX
7413D9BF . 8A06       MOV AL, BYTE PTR DS:[ESI]
7413D9C1 . 46         INC ESI
7413D9C2 . FF2485 24ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]

```

Lo que vemos es que lee el contenido de EBP + 8 (reference pointer) y lo guarda en el contenido de EBP-4c (item pointer)

Bueno el valor que lee en mi maquina es 15b000 si vemos en el DUMP

Registers (FPU)	
EAX	0015B000
ECX	0012FA1C
EDX	740569D0 MSUBU
EBX	00401C98 clave
ESP	0012F3C0
EBP	0012F4E0
ESI	00401BD4 clave
EDI	00159888
EIP	7413D9BA MSUBU
C 0	ES 0023 32bit
P 1	CS 001B 32bit
A 0	SS 0023 32bit
Z 0	DS 0023 32bit

Address	Hex dump	ASCII
0015B000	E8 22 40 00 00 00 00 00
0015B008	1C B0 15 00 7C B1 CF 00
0015B010	0C B1 CF 00 00 00 00 00
0015B018	00 00 00 00 60 A8 06 74
0015B020	04 00 00 00 05 00 00 00
0015B028	00 00 00 00 0F 00 00 00
0015B030	00 00 00 00 E0 18 40 00
0015B038	74 19 40 00 D0 19 40 00
0015B040	30 1A 40 00 90 1A 40 00
0015B048	00 00 00 00 00 00 00 00
0015B050	AB AB AB AB AB AB AB AB
0015B058	00 00 00 00 00 00 00 00
0015B060	07 00 00 00 ED 07 18 00
0015B068	00 00 00 00 7C B8 CF 00
0015B070	68 BD 15 00 70 AE 15 00
0015B078	00 00 00 00 00 00 00 00
0015B080	0D F0 AD BA 0D F0 AD BA
0015B088	AB AB AB AB AB AB AB AB
0015B090	00 00 00 00 00 00 00 00
0015B098	19 00 07 00 F2 07 18 00

Vemos que alli hay un puntero a 4022e8, y alli si vemos en el DUMP

Address	Hex dump	ASCII
004022C8	00 00 00 00 00 00 00 00
004022D0	00 00 00 00 00 00 00 00
004022D8	00 00 00 00 00 00 00 00
004022E0	00 00 00 00 6C 17 40 00
004022E8	08 4C 05 74 FE 25 04 74
004022F0	6F DC 05 74 C1 F8 10 74
004022F8	D0 F8 10 74 A8 40 09 74
00402300	92 47 09 74 80 BD 14 74
00402308	88 BD 14 74 90 BD 14 74
00402310	98 BD 14 74 A0 BD 14 74
00402318	A8 BD 14 74 B0 BD 14 74
00402320	B8 BD 14 74 C0 BD 14 74
00402328	C8 BD 14 74 D0 BD 14 74
00402330	D8 BD 14 74 E0 BD 14 74
00402338	E8 BD 14 74 F0 BD 14 74
00402340	F8 BD 14 74 00 BE 14 74
00402348	08 BE 14 74 10 BE 14 74
00402350	18 BE 14 74 20 BE 14 74
00402358	28 BE 14 74 30 BE 14 74
00402360	38 BE 14 74 40 BE 14 74

Vemos que alli comienza una tabla asi que 15b000, el reference pointer apunta a algo que parece una tabla, aunque esto no nos ayuda mucho para el cracking es bueno siempre tratar de ir descifrando un poco lo que hace el programa, lo mas profundamente que podamos y de acuerdo a la poca información que tenemos.

Por lo demás este es un comando sin parámetros por lo cual, no hay variación en el mismo, seguramente al ejecutarlo lee siempre el reference pointer y lo guarda en el item pointer y chau, jeje.

Bueno continuemos

7413D9B0	. FF2485 94ED11	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D9B7	> 8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
7413D9BA	. 8345 B4	MOV DWORD PTR SS:[EBP-4],EAX	
7413D9BD	. 33C0	XOR EAX,EAX	
7413D9BF	. 8A06	MOV AL,BYTE PTR DS:[ESI]	
7413D9C1	. 46	INC ESI	
7413D9C2	. FF2485 94ED11	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D9C9	> 0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	

Luego ya borra EAX y en la próxima línea lee el tercer opcode

Address	Hex dump	ASCII
00401BCC	08 22 40 00 04 74 FF 21	i"0.♦t ↑
00401BD4	0F 00 03 19 78 FF 08 78	%.♦↓x □x
00401BDC	FF 00 A0 00 00 00 6C 74	.♦...lt
00401BE4	FF 01 00 FB 30 2F 74	+0.'0/t
00401BEC	FF 1A 78 FF 1C 26 00 1E	+x L&..▲
00401BF4	C4 00 FE C1 54 FF 1A C4	-.■-T ↑
00401BFC	03 00 FC F6 64 FF 04 74	♦.z÷d ♦t
00401C04	FF 21 0F 00 03 19 78 FF	↑%.♦↓x
00401C0C	08 78 FF 0D A0 00 00 00	□x .♦...
00401C14	6C 74 FF 07 02 00 04 00	lt .0.♦.
00401C1C	FD 68 54 FF 5D 04 64 FF	zkt l♦d
00401C24	FB 40 2F 74 FF 1A 78 FF	'0/t +x
00401C2C	1C 93 00 27 E1 FE 27 04	L0.'8■'♦
00401C34	FF 3A 34 FF 03 00 4E 24	:4 ♦.N\$
00401C3C	FF 04 24 FF F5 40 00 00	♦\$ 30..
00401C44	00 3A 54 FF 04 00 4E 44	..T ♦.ND
00401C4C	FF 04 44 FF 0A 05 00 14	♦D .♦.¶
00401C54	00 36 08 00 44 FF 24 FF	.6□.D \$
00401C5C	04 FF E4 FE 1E C4 00 27	♦ 8■▲-.'
00401C64	E4 FE 27 04 FF 3A 34 FF	8■'♦ :4
00401C6C	03 00 4E 24 FF 04 34 FF	♦.N\$ ♦\$

Command ? ebp-4c HEX: 12F4

Que es 0F en el EXDEC vemos que se trata de

```
Proc: 401c98
401BD0: 04 FLdRf local_008C
401BD3: 21 FLdPrThis
401BD4: 0F VCallAd text
401BD7: 19 FStAdFunc local_0088
401BDA: 08 FLdPr local_0088
401BDD: 0d VCallHresult get_ipropTEXTEDIT
401BE2: 6c lLdRf local_008C
401BE5: 1b LitStr: ""
401BE8: Lead0/30 EaStr
```

VCallAd

```
'0F, VCallAd, FC, 02
'Access an item's method.
'Parameter 1 = 2 bytes.
'Parameter 1 is offset into item's Descriptor table.
'Offset = &h2FC.
'Method at offset in item's Descriptor table is accessed.
'Method's return value is pushed onto stack.
'Stack operations: Method dependent + Push x1.
```

Bueno aquí vemos la idea de lo que hace el siguiente OPCODE 0F, vemos que tiene un solo parámetro de 2 bytes

Address	Hex dump	ASCII
00401BCC	08 22 40 00 04 74 FF 21	i"0.♦t ↑
00401BD4	0F 00 03 19 78 FF 08 78	*.♦↓x □x
00401BDC	FF 00 A0 00 00 00 6C 74	.a...lt
00401BE4	FF 1B 01 00 FB 30 2F 74	+0.'0/t
00401BEC	FF 1A 78 FF 1C 26 00 1E	+x L&.<
00401BF4	C4 00 FE C1 54 FF 1A C4	-..-T →
00401BFC	03 00 FC F6 64 FF 04 74	♦.z+d ♦t

El parámetro en mi caso es 0300 y dice que dicho valor es un offset en la item descriptor table, hmm veamos, entremos en el JMP indirecto así miramos si coincide con lo que dice.

Registers (FPU)		
7413E89B	> 8B5D B4	MOV EBX,DWORD PTR SS:[EBP-4C]
7413E89E	. 53	PUSH EBX
7413E89F	. 0FB706	MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2	. 0303	ADD EAX,DWORD PTR DS:[EBX]
7413E8A4	. FF10	CALL DWORD PTR DS:[EAX]
7413E8A6	. 50	PUSH EAX
7413E8A7	. 33C0	XOR EAX,EAX
7413E8A9	. 8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413E8AC	. 83C6 03	ADD ESI,3
7413E8AF	. FF2485 94ED1	JMP DWORD PTR DS:[EAX*4+7413ED94]

Allí esta lee el famoso contenido de EBP-4c y lo pasa a EBX

Registers (FPU)	
EAX	0000000F
ECX	0012FA1C
EDX	740569D0 MSUBU
EBX	0015B000
ESP	0012F3C0
EBP	0012F4E0
ESI	00401BD5 clave1
EDI	00159888
EIP	7413E89E MSUBU
C 0	ES 0023 32bit
P 0	CS 001B 32bit
A 0	SS 0023 32bit
Z 0	DS 0023 32bit

Allí esta el 15b000 dicho valor lo pushea

0012F3BC	0015B000
0012F3C0	0012F454
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000
0012F3D0	00000000
0012F3D4	00000000

Registers (FPU)		
7413E89B	> 8B5D B4	MOV EBX,DWORD PTR SS:[EBP-4C]
7413E89E	. 53	PUSH EBX
7413E89F	. 0FB706	MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2	. 0303	ADD EAX,DWORD PTR DS:[EBX]
7413E8A4	. FF10	CALL DWORD PTR DS:[EAX]
7413E8A6	. 50	PUSH EAX
7413E8A7	. 33C0	XOR EAX,EAX
7413E8A9	. 8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413E8AC	. 83C6 03	ADD ESI,3
7413E8AF	. FF2485 94ED1	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E8B2	. 8B5D B4	MOV EBX,DWORD PTR SS:[EBP-4C]

Luego lee los parámetros que en mi caso es 300

Registers (FPU)	
EAX	00000300
ECX	0012FA1C
EDX	740569D0 MSUB
EBX	0015B000
ESP	0012F3BC
EBP	0012F4E0
ESI	00401BD5 clav
EDI	00159888
EIP	7413E8A2 MSUB
C 0	ES 0023 32bit

7413E89E	53	PUSH EBX
7413E89F	0FB706	MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2	0303	ADD EAX,DWORD PTR DS:[EBX]
7413E8A4	FF10	CALL DWORD PTR DS:[EAX]
7413E8A6	50	PUSH EAX
7413E8A7	33C0	XOR EAX,EAX
7413E8A9	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413E8AB	83C6 03	ADD ESI,3

0012F3BC	00CFBCA4
0012F3C0	0012F454
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000
0012F3D0	00000000

O sea de todo ese chiste quedo el valor ese en el stack, que ya veremos para que sirve, por ahora sabemos como dice la definición, que es un valor que leyo de la tabla de ítems, según el parámetro 0300, y el retorno produjo ese valor en el stack.

```

Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd            text
401BD7: 19 FStAdFunc          local_0088
401BDA: 08 FLdPr              local_0088
401BDD: 0d VCallHresult       get__ipropTEXTEDIT

```

El próximo OPCODE es el 19, aquí trabajara con la variable local 88, que surgirá de los parámetros del opcode.

7413E89F	0FB706	MOVZX EAX,WORD PTR DS:[ESI]	
7413E8A2	0303	ADD EAX,DWORD PTR DS:[EBX]	
7413E8A4	FF10	CALL DWORD PTR DS:[EAX]	
7413E8A6	50	PUSH EAX	
7413E8A7	33C0	XOR EAX,EAX	
7413E8A9	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413E8AC	83C6 03	ADD ESI,3	
7413E8AF	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	MSUBUM50.7413E50A
7413E8B6	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]	
7413E8B9	0FBF46 02	MOVSX EAX,WORD PTR DS:[ESI+2]	
7413E8BD	83C6 04	ADD ESI,4	

Allí saltamos a la ejecución del mismo.

7413E50A	BB FFFFFFFF	MOV EBX,-1
7413E50F	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413E512	83C6 02	ADD ESI,2
7413E515	03C5	ADD EAX,EBP
7413E517	59	POP ECX
7413E518	53	PUSH EBX
7413E519	50	PUSH EAX
7413E51A	51	PUSH ECX
7413E51B	E8 301E0000	CALL MSUBUM50.74140350
7413E520	33C0	XOR EAX,EAX
7413E522	8A06	MOV AL,BYTE PTR DS:[ESI]
7413E524	46	INC ESI
7413E525	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]

Allí esta.

Address	Hex dump	ASCII
00401BD0	04 74 FF 21 0F 00 03 19	♦t ↑*.♦↓
00401BD8	78 FF 08 78 FF 0D A0 00	x □x .ä.
00401BE0	00 00 6C 74 FF 1B 01 00	..lt +0.
00401BE8	FB 30 2F 74 FF 1A 78 FF	*0/t +x.

Lee los parámetros con MOVXS y como es negativo los completa con FFs

Registers (FPU)	
EAX	FFFFFFF78
ECX	0012FA1C
EDX	00CF4BA0
EBX	FFFFFFFF
ESP	0012F3BC
EBP	0012F4E0
ESI	00401BD8 clau
EDI	00159888
EIP	7413E512 MSVE
C 0	ES 0023 32bi
D 4	CS 001D 32bi

Como ya adivinaron ese es el valor -88 en hexa como dice el EXDEC.

Modify EAX

Hexadecimal

FFFFFFF78

Signed

-136

Unsigned

4294967160

Char

\xFF

\xFF

\xFF

x

OK

Cancel

-136 decimal es igual a -88 en hexa

			L E M T W H C
7413E50A	BB FFFFFFFF	MOV EBX,-1	
7413E50F	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413E512	83C6 02	ADD ESI,2	
7413E515	03C5	ADD EAX,EBP	
7413E517	59	POP ECX	
7413E518	53	PUSH EBX	
7413E519	50	PUSH EAX	
7413E51A	51	PUSH ECX	
7413E51B	E8 301E0000	CALL MSUBUM50.74140350	
7413E520	33C0	XOR EAX,EAX	
7413E522	8A06	MOV AL,BYTE PTR DS:[ESI]	

Luego incrementa el puntero ESI en 2 y le suma a EBP el valor -88 que es lo mismo que hacer EBP-88 y el resultado queda en EAX.

Registers (FPU)	
EAX	0012F458
ECX	0012FA1C
EDX	00CF4BA0
EBX	FFFFFFFF
ESP	0012F3BC
EBP	0012F4E0
ESI	00401BD8 cla
EDI	00159888
EIP	7413E517 MSV
C 1	ES 0023 32b
D 6	CS 001D 32b

			L E M T W H C / K
7413E50A	BB FFFFFFFF	MOV EBX,-1	
7413E50F	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413E512	83C6 02	ADD ESI,2	
7413E515	03C5	ADD EAX,EBP	
7413E517	59	POP ECX	
7413E518	53	PUSH EBX	
7413E519	50	PUSH EAX	
7413E51A	51	PUSH ECX	
7413E51B	E8 301E0000	CALL MSUBUM50.74140350	
7413E520	33C0	XOR EAX,EAX	

Vemos que al llegar al call en el stack tenemos tres parámetros

0012F3B4	00CFBCA4	Arg1 = 00CFBCA4
0012F3B8	0012F458	Arg2 = 0012F458
0012F3BC	FFFFFFFF	Arg3 = FFFFFFFF
0012F3C0	0012F454	
0012F3C4	00000000	

Al primero es el valor que nos había encontrado y guardado en el stack el opcode anterior, y dos valores mas en mi caso 12f458 que es la variable local ebp-88 y un tercer parámetro que es -1, sin entrar en el CALL pasémoslo con f8 a ver que pasa.

Al ejecutarlo vemos que en ebp-88 se guardo el valor hallado por el opcode anterior

Address	Hex dump	ASCII
0012F458	A4 BC CF 00 EC F4 12 00	00000000
0012F460	C8 F5 12 00 3C BB CF 00	00000000
0012F468	F8 F4 12 00 FF FF FF FF	00000000

Lo demás quedo todo igual salvo que el stack cambio y que ECX volvió siendo cero, quizás para marcar que el proceso fue exitoso.

O sea que esto movió a EBP-88 el valor que obtuvo del opcode anterior.

```

Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd            text
401BD7: 19 FStAdFunc          local_0088
401BDA: 08 FLdPr              local_0088
401BDD: 0d VCallHresult       get_ipropTEXTEDIT
401BE2: 6c ILdRf              local_008C
401BE5: 1b LitStr:            "
401BF8: 1e add0/30 FnStr

```

Llegamos al otro opcode que es 08, que también se ve que trabaja con la misma variable local 88 o sea ebp-88.

Address	Hex dump	Disassembly	Comment
7413E512	83C6 02	ADD ESI,2	
7413E515	03C5	ADD EAX,EBP	
7413E517	59	POP ECX	
7413E518	53	PUSH EBX	
7413E519	50	PUSH EAX	
7413E51A	51	PUSH ECX	
7413E51B	E8 301E0000	CALL MSUBUM50.74140350	
7413E520	33C0	XOR EAX,EAX	
7413E522	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E524	46	INC ESI	
7413E525	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	MSUBUM50.7413E3D0
7413E52C	FF3424	PUSH DWORD PTR SS:[ESP]	

Entramos el OPCODE

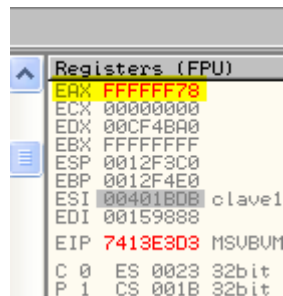
Address	Hex dump	Disassembly	Comment
7413E3D0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413E3D3	83C6 02	ADD ESI,2	
7413E3D6	8B0428	MOV EAX,DWORD PTR DS:[EAX+EBP]	
7413E3D9	0BC0	OR EAX,EAX	
7413E3DB	0F84 34760000	JE MSUBUM50.74145A15	
7413E3E1	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX	
7413E3E4	33C0	XOR EAX,EAX	
7413E3E6	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E3E8	46	INC ESI	
7413E3E9	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413E3F0	8B7D B4	MOV EDI,DWORD PTR SS:[EBP-4C]	

Este es cortito ahí abajo vemos que termina pues esta el siguiente XOR EAX,EAX que marca la finalización del mismo aunque hay un salto condicional en el medio veamos que hace.

Lo primero, pasa a EAX los parámetros del OPCODE

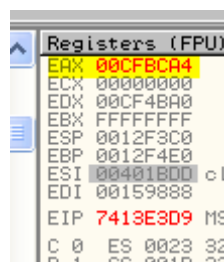
Address	Hex dump	ASCII
004018D3	21 0F 00 03 19 78 FF 08	00000000
004018DB	78 FF 00 A0 00 00 00 6C	00000000
004018E3	74 FF 1B 01 00 FB 30 2F	00000000
004018EB	74 FF 1A 78 FF 1C 26 00	00000000
004018F3	1E C4 00 FE C1 54 FF 1A	00000000

Al igual que antes es el valor FF78 que al moverlo con MOVSX lo completa con FFs al ser negativo y ya vimos que es el -88 hexa.



7413E3D0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413E3D3	83C6 02	ADD ESI,2	
7413E3D6	8B0428	MOV EAX,DWORD PTR DS:[EAX+EBP]	
7413E3D9	0BC0	OR EAX,EAX	
7413E3DB	0F84 34760000	JE MSUBUM50.74145A15	
7413E3E1	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX	
7413E3E4	33C0	XOR EAX,EAX	
7413E3E6	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E3E8	46	INC ESI	

En esa línea directamente suma EAX+EBP lo que le da EBP-88 y luego mueve el valor guardado allí que era el famoso, que obtuvimos de la tabla de ítems.



7413E3D0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413E3D3	83C6 02	ADD ESI,2	
7413E3D6	8B0428	MOV EAX,DWORD PTR DS:[EAX+EBP]	
7413E3D9	0BC0	OR EAX,EAX	
7413E3DB	0F84 34760000	JE MSUBUM50.74145A15	
7413E3E1	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX	
7413E3E4	33C0	XOR EAX,EAX	
7413E3E6	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E3E8	46	INC ESI	
7413E3E9	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413E3F0	8B7D B4	MOV EDI,DWORD PTR SS:[EBP-4C]	

Ahí testea si es cero, si fuera cero saltaría, pero como aquí no es cero continua.

7413E3D0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413E3D3	83C6 02	ADD ESI,2	
7413E3D6	8B0428	MOV EAX,DWORD PTR DS:[EAX+EBP]	
7413E3D9	0BC0	OR EAX,EAX	
7413E3DB	0F84 34760000	JE MSUBUM50.74145A15	
7413E3E1	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX	
7413E3E4	33C0	XOR EAX,EAX	

Guarda ese valor en EBP-4C

Aquí le vemos sentido a lo que decíamos en el inicio recordemos

Lo que vemos es que lee el contenido de EBP + 8 (reference pointer) y lo guarda en el contenido de EBP-4c (item pointer)

Como ebp-4c es el puntero de ítems, como este ya confirmo que es valido y que no es cero lo guarda alli, en la variable EBP-4c que se usa para eso, por eso se llama ITEM POINTER, ya que este valor lo hallo en relación a la tabla de ITEMS.

7413E3D0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413E3D3	83C6 02	ADD ESI,2	
7413E3D6	8B0428	MOV EAX,DWORD PTR DS:[EAX+EBP]	
7413E3D9	0BC0	OR EAX,EAX	
7413E3DB	0F84 34760000	JE MSUBUM50.74145A15	
7413E3E1	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX	
7413E3E4	33C0	XOR EAX,EAX	
7413E3E6	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E3E8	46	INC ESI	
7413E3E9	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413E3F0	8B7D B4	MOV EDI,DWORD PTR SS:[EBP-4C]	

Llegamos al siguiente opcode

```

Proc: 401c98
401BD0: 04 FLdRfVar      local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd       text
401BD7: 19 FStAdFunc     local_0088
401BDA: 08 FLdPr        local_0088
401BDD: 0d VCallHresult  get_ipropTEXTEDIT
401BE2: 6c ILdRf        local_008C
401BE5: 1b LitStr:      "
401BE8: 1e LdD030 FcStr

```

7413E3E6	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E3E8	46	INC ESI	
7413E3E9	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	MSUBUM50.7413E829
7413E3F0	8B7D B4	MOV EDI,DWORD PTR SS:[EBP-4C]	
7413E3F3	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	

Por allí en San Google leo que dicho OPCODE sirve para

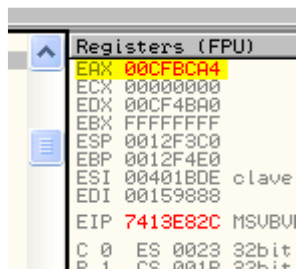
0d VCallHresult #get the text from textbox

O sea que leerá el serial falso que tipeamos en el textbox, veamos si es cierto, traceemos

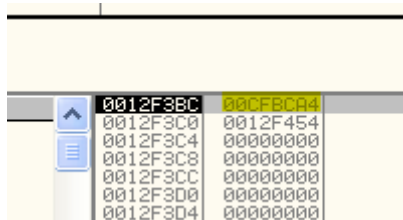
7413E829	8B45 B4	MOV EAX,DWORD PTR SS:[EBP-4C]	
7413E82C	50	PUSH EAX	
7413E82D	0FB73E	MOVZX EDI,WORD PTR DS:[ESI]	
7413E830	8B00	MOV EAX,DWORD PTR DS:[EAX]	
7413E832	03C7	ADD EAX,EDI	
7413E834	FF10	CALL DWORD PTR DS:[EAX]	
7413E836	8B55 BC	MOV EDX,DWORD PTR SS:[EBP-44]	
7413E839	66:F742 76 020	TEST WORD PTR DS:[EDX+76],2	
7413E83F	75 13	JNZ SHORT MSUBUM50.7413E854	
7413E841	0BC0	OR EAX,EAX	
7413E843	78 19	JS SHORT MSUBUM50.7413E85E	
7413E845	33C0	XOR EAX,EAX	
7413E847	8A46 04	MOV AL,BYTE PTR DS:[ESI+4]	
7413E849	83C6 05	ADD ESI,5	

Ahí vemos el OPCODE que termina en XOR EAX,EAX como siempre

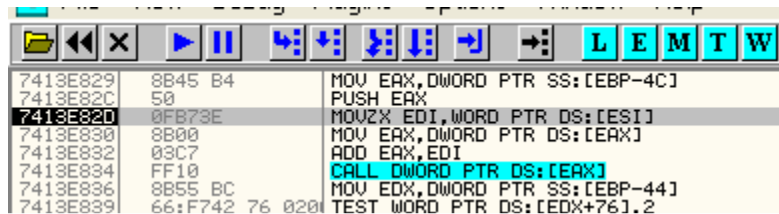
Lo primero que hace es leer del puntero EBP-4c, que como dijimos era el ITEM POINTER y lo mueve a EAX el archiconocido valor que leyó de la tabla de ITEMS



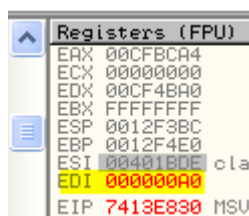
Luego lo PUSHEA



Luego lee un parámetro del OPCODE.

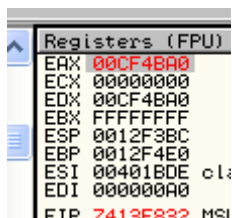


Address	Hex dump	ASCII
00401BD6	03 19 78 FF 08 78 FF 00	␣␣␣␣␣␣␣␣
00401BDE	00 00 00 00 6C 74 FF 1B	␣␣␣␣␣␣␣␣
00401BE6	01 00 FB 30 2F 74 FF 1A	␣␣␣␣␣␣␣␣
00401BEE	78 FF 1C 26 00 1E C4 00	␣␣␣␣␣␣␣␣
00401BF6	FE C1 54 FF 1A C4 03 00	␣␣␣␣␣␣␣␣



Y lo pasa a EDI

Luego lee el contenido de EAX que es el inicio de otra tabla



Address	Hex dump	ASCII
00CF4B88	AB AB AB AB EE FE EE FE	%%%~■
00CF4B90	00 00 00 00 00 00 00 00
00CF4B98	49 00 0F 00 41 07 18 00	I.*.A.†.
00CF4BA0	95 5A 06 74 A2 E2 05 74	0Z†t00†t
00CF4BA8	58 E2 05 74 4A 68 0E 74	X0†tJh†t
00CF4BB0	55 00 0A 74 80 43 06 74	U..tC†t
00CF4BB8	AB CD 06 74 47 26 09 74	%=†tG%.t
00CF4BC0	6F 37 07 74 09 04 06 74	o7..t.E†t
00CF4BC8	55 1E 08 74 20 38 10 74	U4†t ;†t
00CF4BD0	A4 94 06 74 BE F7 0C 74	K0†t#.t
00CF4BD8	C6 F7 0C 74 AC 40 09 74	g..t%0.t
00CF4BE0	7E 79 0D 74 8E 79 0D 74	~y..tAy..t
00CF4BE8	AA 79 0D 74 BC 79 0D 74	~y..tAy..t
00CF4BF0	D0 79 0D 74 E2 79 0D 74	sy..tOy..t
00CF4BF8	F6 79 0D 74 08 7A 0D 74	÷y..t0z..t
00CF4C00	1C 7A 0D 74 2E 7A 0D 74	Lz..t.z..t
00CF4C08	42 7A 0D 74 54 7A 0D 74	Bz..tTz..t
00CF4C10	68 7A 0D 74 7A 7A 0D 74	hz..tzz..t
00CF4C18	8E 7A 0D 74 A0 7A 0D 74	Az..tAz..t
00CF4C20	B4 7A 0D 74 C6 7A 0D 74	†z..tAz..t
00CF4C28	00 7A 0D 74 EC 7A 0D 74	..tAz..t

Command ? 88

Y en esa tabla le suma el parámetro 00a0 para buscar el ella.

Registers (FPU)	
EAX	00CF4C40
ECX	00000000
EDX	00CF4BA0
EBX	FFFFFFFF
ESP	0012F3BC
EBP	0012F4E0
ESI	00401BDE cl
EDI	000000A0
EIP	7413F834 MS

Address	Hex dump	ASCII
00CF4C30	00 7B 0D 74 12 7B 0D 74	.t.t#t.t
00CF4C38	26 7B 0D 74 38 7B 0D 74	&t.t8t.t
00CF4C40	B6 A5 09 74 32 0D 09 74	AN.t2..t
00CF4C48	4C 7B 0D 74 5E 7B 0D 74	L.t.^t.t
00CF4C50	72 7B 0D 74 84 7B 0D 74	r.t.t&t.t
00CF4C58	98 7B 0D 74 AA 7B 0D 74	y.t.t-t.t
00CF4C60	BE 7B 0D 74 D0 7B 0D 74	#t.t&t.t
00CF4C68	E4 7B 0D 74 F6 7B 0D 74	8t.t÷t.t
00CF4C70	0A 7C 0D 74 1C 7C 0D 74	.t.tL.t.t
00CF4C78	30 7C 0D 74 42 7C 0D 74	0!.tB!.t
00CF4C80	56 7C 0D 74 68 7C 0D 74	U!.th!.t
00CF4C88	7C 7C 0D 74 8E 7C 0D 74	!!..tA!.t
00CF4C90	A2 7C 0D 74 B4 7C 0D 74	o!.t†!.t
00CF4C98	C8 7C 0D 74 DA 7C 0D 74	=!.t.r†.t
00CF4CA0	EE 7C 0D 74 00 7D 0D 74	~!.t..t.t
00CF4CA8	14 7D 0D 74 26 7D 0D 74	¶).t&).t
00CF4CB0	3A 7D 0D 74 4C 7D 0D 74	~).tL).t
00CF4CB8	60 7D 0D 74 72 7D 0D 74	').t.r).t
00CF4CC0	86 7D 0D 74 98 7D 0D 74	g).t9).t
00CF4CC8	AC 7D 0D 74 BE 7D 0D 74	%).t#).t
00CF4CD0	00 7D 0D 74 EC 7D 0D 74	..t#).t

Command ? 88

Y el realiza un call a la dirección que obtiene de esa tabla, por supuesto no vamos a trazar ese call por dentro, vemos que en el stack continúan los valores que obtuvo de los opcodes anteriores

0012F3BC	00CFBCA4
0012F3C0	0012F454
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000

Ejecuto el CALL con f8

Luego mueve a EDX un valor que obtiene del contenido de EBP-44

Registers (FPU)		
EAX	00000000	
ECX	7C92056D nt	
EDX	00159888	
EBX	FFFFFFFF	
ESP	0012F3C4	
EBP	0012F4E0	
ESI	00401BDE cl	
EDI	000000A0	
EIP	7413E839 MS	
C 0	ES 0023 32	
P 1	CS 001B 32	

7413E834	FF10	CALL DWORD PTR DS:[EAX]
7413E836	8B55 BC	MOV EDX,DWORD PTR SS:[EBP-44]
7413E839	66:F742 76 020	TEST WORD PTR DS:[EDX+76],2
7413E83F	75 13	JNZ SHORT MSUBUM50.7413E854
7413E841	0BC0	OR EAX,EAX
7413E843	78 19	JS SHORT MSUBUM50.7413E85E
7413E845	33FA	XOR EAX,EAX

Y luego de testear un par de valores llega al próximo OP CODE, pero ustedes dirán, leyo lo que tipeamos o sea nuestro serial falso?, si lo leyó si vemos el EXDEC

```

Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd             text
401BD7: 19 FStAdFunc           local_0088
401BDA: 08 FLdPr              local_0088
401BDD: 0d VCallHresult        get_ipropTEXTEDIT
401BE2: 6c ILdRf               local_008C
401BE5: 1b LitStr:             "
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str           local_008C
401BED: 15 FFree1Str           local_008C

```

Vemos que a continuación trabaja con la variable local 8c que como sabemos esta en EBP-8c

Si buscamos el valor de EBP-8c

0015D444	EE FE EE FE EE FE EE FE	EE FE EE FE	EE FE EE FE
0015D44C	EE FE EE FE EE FE EE FE	EE FE EE FE	EE FE EE FE
0015D454	EE FE EE FE EE FE EE FE	EE FE EE FE	EE FE EE FE
0015D45C	EE FE EE FE EE FE EE FE	EE FE EE FE	EE FE EE FE

Command	? ebp-8c	HEX: 12F454 -
---------	----------	---------------

Es 12f454

Address	Hex dump	ASCII
0012F454	BC D3 15 00 A4 BC CF 00	"ES. A" B.
0012F45C	EC F4 12 00 C8 F5 12 00	U"t. "S.
0012F464	3C BB CF 00 E8 F4 12 00	<"B. b"t.
0012F46C	FF FF FF FF 15 20 00 00	S . .
0012F474	02 E3 13 74 E8 F4 12 00	00!!t b"t.
0012F47C	00 00 00 00 00 00 00 00
0012F484	00 00 00 00 D0 1B 40 00	...S+0.
0012F48C	E4 17 40 00 98 1C 40 00	S+0. yL0.
0012F494	A4 BC CF 00 00 E0 00 00	A" B. .0..
0012F49C	88 98 15 00 00 00 00 00	eyS.....
0012F4A4	00 00 00 00 00 00 04 00+
0012F4AC	00 00 00 00 D0 F4 12 00	...S"t.
0012F4B4	80 AF 15 00 1C FA 12 00	C>S. L. .
0012F4BC	D0 69 05 74 02 E3 13 74	S i t 00 !!t
0012F4C4	C4 F3 12 00 E0 F4 12 00	-%t. 0"t.
0012F4CC	D0 1B 40 00 88 98 15 00	S+0. eyS.
0012F4D4	98 1C 40 00 00 00 00 00	yL0.....
0012F4DC	00 00 00 00 EC F4 12 00	...y"t.
0012F4E4	A9 E5 05 74 00 B0 15 00	00 t. S.
0012F4EC	FC F4 12 00 F8 1A 40 00	"t. .+0.

Y ese es el puntero al serial falso que escribimos, o sea en 15d3bc

EAX=00000000			
Address	Hex dump	ASCII	
0015D3BC	39 00 38 00 39 00 38 00	9.8.9.8.	
0015D3C4	39 00 38 00 39 00 38 00	9.8.9.8.	
0015D3CC	00 00 AD BA 0D F0 AD BA	.- .-	
0015D3D4	0D F0 AD BA AB AB AB AB	.- %%%%	
0015D3DC	AB AB AB AB 00 00 00 00	%%%. . . .	
0015D3E4	00 00 00 00 2D 00 07 00 - . .	
0015D3EC	EE 04 EE 00 E0 02 15 00	◆-.00\$.	

Uff costo sangre pero llegamos al punto donde leyó el serial falso.

El siguiente OPCODE es **6c ILdRf**

Email josephco_@hotmail.com with any errors or problems

```

Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd              text
401BD7: 19 FStAdFunc            local_0088
401BDA: 08 FLdPr                local_0088
401BDD: 0d VCallHresult         get_ipropTEXTEDIT
401BE2: 6c ILdRf                local_008C
401BE5: 1b LitStr:              "
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str            local_008C

```

7413E843	78 19	JS SHORT MSUBUM50.7413E85E	
7413E845	33C0	XOR EAX,EAX	
7413E847	8A46 04	MOV AL,BYTE PTR DS:[ESI+4]	
7413E84A	83C6 05	ADD ESI,5	
7413E84D	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413E854	B8 689C0000	MOV EAX,9C68	
7413E859	E9 3AEBFFFF	JMP MSUBUM50.7413D398	
7413E85E	66:3D 689C	CMP AX,9C68	
7413E862	0F84 D57A0000	JE MSUBUM50.7414633D	
7413E868	E7	DISCU CONT	

'6C, ILdRf, 0C, 00
 'Load reference value.
 'Parameter 1 = 2 bytes.
 'Parameter 1 is offset into local Frame.
 'Offset = &hC.
 'Address pointer is retrieved from local Frame at offset.
 'Address pointer is pushed onto stack.
 'Stack operations: Push x1.

Allí dice que este OPCODE es LOAD REFERENCE VALUE veamos entremos en el OPCODE

7413E841	0BC0	OR EAX,EAX	
7413E843	78 19	JS SHORT MSUBUM50.7413E85E	
7413E845	33C0	XOR EAX,EAX	
7413E847	8A46 04	MOV AL,BYTE PTR DS:[ESI+4]	
7413E84A	83C6 05	ADD ESI,5	
7413E84D	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	MSUBUM50.7413D947
7413E854	B8 689C0000	MOV EAX,9C68	
7413E859	E9 3AEBFFFF	JMP MSUBUM50.7413D398	

Aquí esta

7413D947	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413D94A	FF3428	PUSH DWORD PTR DS:[EAX+EBP]
7413D94D	33C0	XOR EAX,EAX
7413D94F	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D952	83C6 03	ADD ESI,3
7413D955	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]

Lo primero que hace es mover el parámetro con MOVSX asi completa con FFs si es negativo.

DS:[00401BE3]=FF74		EAX=0000006C	
Address	Hex dump	ASCII	
00401BDB	78 FF 0D A0 00 00 00 6C	x .á...l	
00401BE3	74 FF 1B 01 00 FB 30 2F	t +0.10/	
00401BEB	74 FF 1A 78 FF 1C 26 00	t +x L&.	
00401BF3	1E C4 00 FE C1 54 FF 1A	▲-.-+T →	
00401BFB	C4 03 00 FC F6 64 FF 04	-0.3÷d ◆	
00401C03	74 FF 21 0F 00 03 19 78	t !*.◆+x	

Como el parámetro es FF74, pues al pasarlo a EAX quedara

Registers (FPU)
EAX FFFFFFF4
ECX 7C92056D nti
EDX 00159888
EBX FFFFFFFF
ESP 0012F3C4
EBP 0012F4E0
ESI 00401BE3 cl.
EDI 000000A0
EIP 7413D94A MS
C 0 ES 0023 32
P 0 CS 001B 32

Que es -8c en hexa

7413D947	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413D94A	FF3428	PUSH DWORD PTR DS:[EAX+EBP]
7413D94D	33C0	XOR EAX,EAX
7413D94F	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D952	83C6 03	ADD ESI,3
7413D955	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D95C	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]

En la próxima instrucción lo suma a EBP, con lo cual obtiene EBP-8c y pushea el contenido o sea que seria equivalente a PUSH [ebp-8c]

0012F3C0	0015D3BC	UNICODE "98989898"
0012F3C4	00000000	
0012F3C8	00000000	
0012F3CC	00000000	
0012F3D0	00000000	

Pero en el contenido de ebp-8c tenia un puntero al serial falso, por lo cual ahora tenemos el puntero alli en el stack.

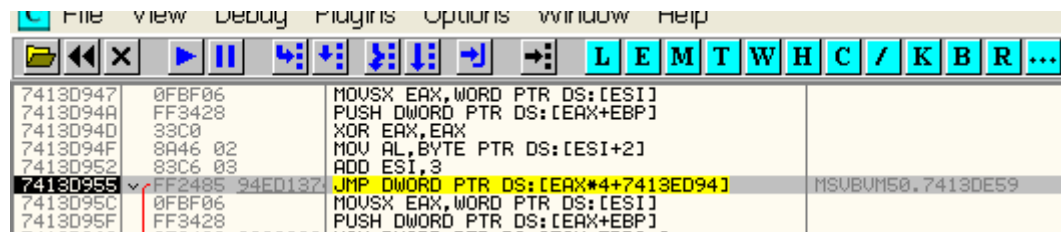
Address	Hex dump	ASCII
0015D3BC	39 00 38 00 39 00 38 00	9.8.9.8.
0015D3C4	39 00 38 00 39 00 38 00	9.8.9.8.
0015D3CC	00 00 AD BA 0D F0 AD BA+
0015D3D4	0D F0 AD BA 0D F0 AD BA	..+ %2%
0015D3DC	0E 0E 0E 0E 00 00 00 00	%2%....
0015D3E4	00 00 00 2D 00 07 00	...-..
0015D3EC	EE 04 EE 0A EA 02 1E 0A	▲-▲&S

Si lo vemos en el dump vemos claramente que apunta al serial falso, vemos porque llaman al OPCODE, LOAD REFERENCE VALUE, pues carga al stack un valor que vamos a usar desde una variable local.

El próximo opcode es

1b LitStr por allí leo que significa Literal String

Veamos que hace

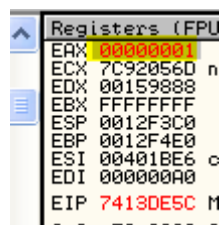


```
File View Debug Plugins Options Window Help
[Icons] [L] [E] [M] [T] [W] [H] [C] [/] [K] [B] [R] [...]
7413D947 0FBF06 MOVZX EAX,WORD PTR DS:[ESI]
7413D94A FF3428 PUSH DWORD PTR DS:[EAX+EBP]
7413D94D 33C0 XOR EAX,EAX
7413D94F 8A46 02 MOV AL,BYTE PTR DS:[ESI+2]
7413D952 83C6 03 ADD ESI,3
7413D955 FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94] MSUBUM50.7413DE59
7413D95C 0FBF06 MOVZX EAX,WORD PTR DS:[ESI]
7413D95F FF3428 PUSH DWORD PTR DS:[EAX+EBP]
```

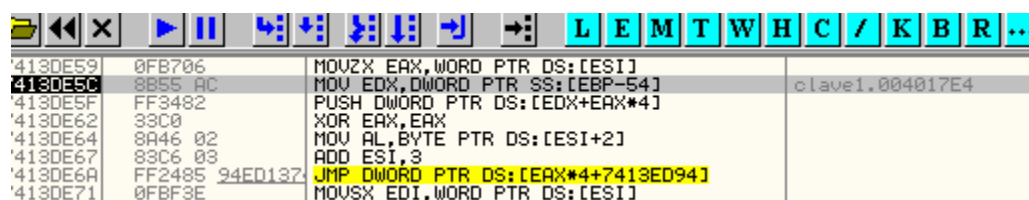
Entremos en el OPCODE

Address	Hex dump	ASCII
00401BDE	A0 00 00 00 6C 74 FF 1B	á...lt +
00401BE6	01 00 FB 30 2F 74 FF 1A	0.'0/t +
00401BEE	70 FF 1C 26 00 1E C4 00	x L%.+-.
00401BF6	FF C1 54 FF 1A C4 03 00	■+T +-.

Lo primero lee el parámetro que aquí es 0001 y lo mueve a EAX como es positivo no le agrega FFs

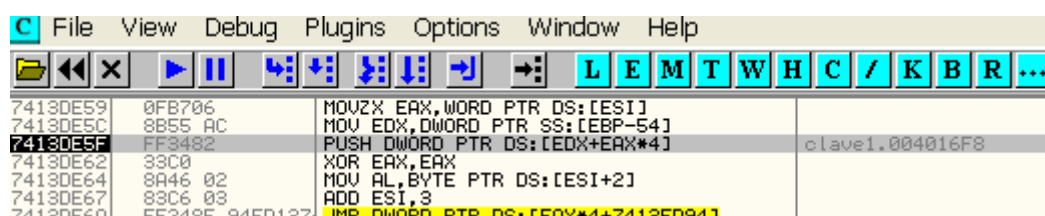


```
Registers (FPU)
EAX 00000001
ECX 7C92056D n
EDX 00159888
EBX FFFFFFFF
ESP 0012F3C0
EBP 0012F4E0
ESI 00401BE6 c
EDI 000000A0
EIP 7413DE5C M
```



```
[Icons] [L] [E] [M] [T] [W] [H] [C] [/] [K] [B] [R] [...]
7413DE59 0FB706 MOVZX EAX,WORD PTR DS:[ESI]
7413DE5C 8B55 AC MOV EDX,DWORD PTR SS:[EBP-54] clave1.004017E4
7413DE5F FF3482 PUSH DWORD PTR DS:[EDX+EAX*4]
7413DE62 33C0 XOR EAX,EAX
7413DE64 8A46 02 MOV AL,BYTE PTR DS:[ESI+2]
7413DE67 83C6 03 ADD ESI,3
7413DE6A FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413DE71 0FBF3E MOVZX EDI,WORD PTR DS:[ESI]
```

Vemos que en la próxima línea mueve a EDX un valor 4017E4 no sabemos que es sigamos.



```
File View Debug Plugins Options Window Help
[Icons] [L] [E] [M] [T] [W] [H] [C] [/] [K] [B] [R] [...]
7413DE59 0FB706 MOVZX EAX,WORD PTR DS:[ESI]
7413DE5C 8B55 AC MOV EDX,DWORD PTR SS:[EBP-54]
7413DE5F FF3482 PUSH DWORD PTR DS:[EDX+EAX*4] clave1.004016F8
7413DE62 33C0 XOR EAX,EAX
7413DE64 8A46 02 MOV AL,BYTE PTR DS:[ESI+2]
7413DE67 83C6 03 ADD ESI,3
7413DE6A FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94]
```

Y que pusha un valor 4016f8 que puede ser?

```

Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd             text
401BD7: 19 FStAdFunc             local_0088
401BDA: 08 FLdPr                local_0088
401BDD: 0d VCallHresult        get_ipropTEXTEDIT
401BE2: 6c ILdRf                local_008C
401BE5: 1b LitStr:               "
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str          local_008C

```

Vemos que la aclaración del EXDEC nos muestra unas comillas, o sea que este valor que movio al stack apunta a una string vacía.

Address	Hex dump	ASCII			
004016F8	00 00 00 00 22 00 00 00".		0012F3BC	004016F8
00401700	4E 00 FA 00 6D 00 65 00	N..m.e.		0012F3C0	0015D3BC
00401708	72 00 6F 00 20 00 49 00	r.o..I.		0012F3C4	00000000
00401710	6E 00 63 00 6F 00 72 00	n.c.o.r.		0012F3C8	00000000
00401718	72 00 65 00 63 00 74 00	r.e.c.t.		0012F3CC	00000000
00401720	6F 00 00 00 0C 00 00 00	o.....		0012F3D0	00000000
00401728	50 00 2D 00 43 00 6F 00	P.-C.o.		0012F3D4	00000000
00401730	64 00 65 00 00 00 00 00	d.e.....		0012F3D8	00000000
00401738	22 00 00 00 4E 00 FA 00	".N..		0012F3DC	00000000
00401740	6D 00 65 00 72 00 6F 00	m.e.r.o.		0012F3E0	00000000
00401748	20 00 43 00 6F 00 72 00	.C.o.r.		0012F3E4	00000000
00401750	72 00 65 00 63 00 74 00	r.e.c.t.		0012F3E8	00000000
00401758	6F 00 21 00 21 00 00 00	o.f.t...		0012F3EC	00000000
00401760	56 42 41 35 2E 44 4C 4C	VBAS.DLL		0012F3F0	00000000
00401768	00 00 00 00 00 00 00 00		0012F3F4	00000000
00401770	00 00 00 00 00 00 00 00		0012F3F8	00000000

Pues si vemos en el dump que apunta a una string vacia que termina en comillas, o sea lo que hará en el siguiente opcode será chequear si tipeamos algo, o si dejamos vacío el textbox y apretamos register.

File View Debug Plugins Options Window Help			
Address	Hex dump	Disassembly	Comment
7413DE59	0FB706	MOVZX EAX,WORD PTR DS:[ESI]	
7413DE5C	8B55 AC	MOV EDX,DWORD PTR SS:[EBP-54]	
7413DE5F	FF3482	PUSH DWORD PTR DS:[EDX+EAX*4]	
7413DE62	33C0	XOR EAX,EAX	
7413DE64	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413DE67	83C6 03	ADD ESI,3	
7413DE6A	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413DE71	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]	
7413DE74	83C6 02	ADD ESI,2	

Que es

```

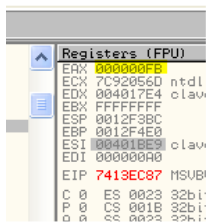
401BE2: 6c ILdRf          local_008C
401BE5: 1b LitStr:        "
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str      local_008C

```

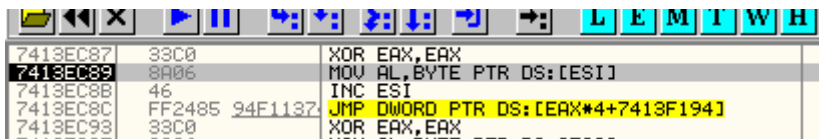
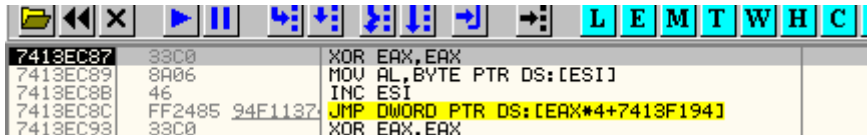
Bueno LEAD 0 es una operacion y según la segunda parte del OPCODE será la operación que realiza en este caso 30 Eq Str que vemos en San Google

Lead0/30 EqStr <---Compara dos strings

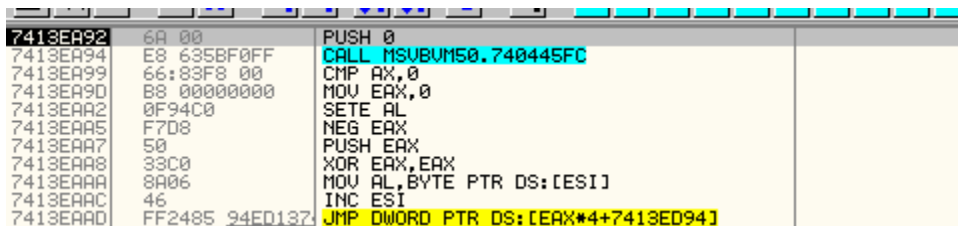
O sea que va a comparar esas dos strings este es un OPCODE doble o sea que primero lee el primero que es FB salta el JMP indirecto



Y allí nomás termina el primer OPCODE con XOR EAX,EAX sin hacer nada y lee el segundo OPCODE



Allí lee el segundo OPCODE 30, diferenciamos en este caso un OPCODE doble, de leer parámetros del un OPCODE, porque la diferencia esta en que en un OPCODE DOBLE se cierra el primer OPCODE con el XOR EAX.EAX y ahí recién se lee el segundo OPCODE, en el caso de lectura de parámetros el primer OPCODE queda abierto y no se cierra al leerlos.



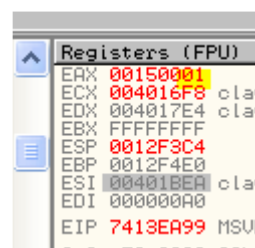
Allí esta el segundo OPCODE, hace un PUSH 0



Y quedan los tres argumentos al llegar al CALL pasémoslo con f8 y veamos lo que cambio.

Vemos que el stack se movió pero los valores continúan sin modificación un poco mas arriba.

Vemos que en la siguiente línea CMP AL,0 lo cual me dice que allí en AL, guarda el resultado en mi caso es



AL=01

Porque las strings no son iguales

Address	Hex	Assembly
7413EA92	6A 00	PUSH 0
7413EA94	E8 635BF0FF	CALL MSUBUM50.740445FC
7413EA99	66:83F8 00	CMP AX,0
7413EA9D	B8 00000000	MOV EAX,0
7413EAA2	0F94C0	SETL AL
7413EAA5	F7D8	NEG EAX
7413EAA7	50	PUSH EAX
7413EAA8	33C0	XOR EAX,EAX
7413EAAA	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EAAE	4C	INC ECX

Luego de la comparación mueve a EAX el valor cero y como resultado de todo esto termina PUSHEANDO el cero al stack, lo cual es el resultado del OPCODE, da cero en el stack si no son iguales, si hubieran sido iguales daría FFFFFFFF si quieren probar háganlo jeje.

Address	Hex	Assembly
7413EA92	6A 00	PUSH 0
7413EA94	E8 635BF0FF	CALL MSUBUM50.740445FC
7413EA99	66:83F8 00	CMP AX,0
7413EA9D	B8 00000000	MOV EAX,0
7413EAA2	0F94C0	SETL AL
7413EAA5	F7D8	NEG EAX
7413EAA7	50	PUSH EAX
7413EAA8	33C0	XOR EAX,EAX
7413EAAA	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EAAE	4C	INC ECX
7413EAB0	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413EAB4	5A	POP EDX
7413EAB5	58	POP EAX

Llegamos al siguiente OPCODE

```

401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str      local_008C
401BED: 1a FFree1Ad       local_0088
401BF0: 1c BranchF:       401BF6
  
```

San Google nos dice que es similar a SysFreeString que libera una string cuando ya no se usa, veamos en este caso liberaría la de ebp-8c (local 8c)

Address	Hex	Assembly
7413E6C7	BF 01000000	MOV EDI,1
7413E6CC	0FBF1E	MOVSX EBX,WORD PTR DS:[ESI]
7413E6CF	83C6 02	ADD ESI,2
7413E6D2	FF342B	PUSH DWORD PTR DS:[EBX+EBP]
7413E6D5	E8 793AF0FF	CALL <JMP.&OLEAUT32.#6>
7413E6DA	C7042B 00000000	MOV DWORD PTR DS:[EBX+EBP],0
7413E6E1	4F	DEC EDI
7413E6E2	7F E8	JG SHORT MSUBUM50.7413E6CC
7413E6E4	33C0	XOR EAX,EAX
7413E6E6	8A06	MOV AL,BYTE PTR DS:[ESI]
7413E6E8	4C	INC ECX
7413E6E9	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E6FA	0F8F3F	MOVSX EAX,WORD PTR DS:[ESI]

Veamos mueve a EDI el valor 1, luego el FF74 que es -8c en hexa como siempre con el MOVSX para llenar con FFs si es negativo.

Registers (FPU)

EAX	0000002F
ECX	004016F8 clave
EDX	004017E4 clave
EBX	FFFFFFFF74
ESP	0012F3BC
EBP	0012F4E0
ESI	004018B8 clave
EDI	00000001
EIP	7413E6CF MSUBUM50.740445FC
C 0	ES 0023 32bit
P 1	CS 001B 32bit

Address	Hex	Assembly
7413E6CC	0FBF1E	MOVSX EBX,WORD PTR DS:[ESI]
7413E6CF	83C6 02	ADD ESI,2
7413E6D2	FF342B	PUSH DWORD PTR DS:[EBX+EBP]
7413E6D5	E8 793AF0FF	CALL <JMP.&OLEAUT32.#6>
7413E6DA	C7042B 00000000	MOV DWORD PTR DS:[EBX+EBP],0
7413E6E1	4F	DEC EDI

EBX+EBP es EBP-8c, así que pushea el puntero al serial falso

Address	Hex dump	ASCII		0012F3B8	0015D3BC	Arg1 = 0015D3BC
0015D3B0	39 00 38 00 39 00 38 00	9.8.9.8.		0012F3B0	00000000	
0015D3C4	39 00 38 00 39 00 38 00	9.8.9.8.		0012F3C0	00000000	
0015D3CC	00 00 AD BA 0D F0 AD BA	..+ ..+		0012F3C4	00000000	
0015D3D4	0D F0 AD BA 0D F0 AD BA	..+ ..+		0012F3C8	00000000	
				0012F3CC	00000000	

Address	Hex dump	ASCII		0012F3B8	0015D3BC	Arg1 = 0015D3BC
74042153	FF25 88190474		JMP DWORD PTR DS:[<&OLEAUT32.#6>]			
74042159	E8 28000000		CALL MSUBUM50.74042186			
7404215E	833D 64F01474		CMP DWORD PTR DS:[7414F064],0			
74042165	74 18		JE SHORT MSUBUM50.7404217F			
74042167	A1 68F01474		MOV EAX,DWORD PTR DS:[7414F068]			
7404216C	50		PUSH EAX			
7404216D	FF15 84100474		CALL DWORD PTR DS:[<&KERNEL32.TlsGetValue			
74042173	8B48 18		MOV ECX,DWORD PTR DS:[EAX+18]			
74042176	8B41 14		MOV EAX,DWORD PTR DS:[ECX+14]			
74042179	8B10		MOV EDX,DWORD PTR DS:[EAX]			
7404217B	8951 14		MOV DWORD PTR DS:[ECX+14],EDX			
7404217E	C3		RETN			
7404217F	01 6CF01474		MOVL EAX,DWORD PTR DS:[7414F06C]			

Vemos que dentro de ese call va a la api SysFreeString , y al retornar

Address	Hex dump	ASCII		0012F3B8	0015D3BC	Arg1 = 0015D3BC
7413E6C7	BF 01000000		MOV EDI,1			
7413E6CC	0FBF1E		MOVSX EBX,WORD PTR DS:[ESI]			
7413E6CF	83C6 02		ADD ESI,2			
7413E6D2	FF342B		PUSH DWORD PTR DS:[EBX+EBP]			
7413E6D5	E8 793AF0FF		CALL <JMP.&OLEAUT32.#6>			
7413E6DA	C7042B 00000000		MOV DWORD PTR DS:[EBX+EBP],0			
7413E6E1	4F		DEC EDI			
7413E6E2	7F E8		JG SHORT MSUBUM50.7413E6CC			
7413E6F4	33FA		XOR EBX,EBX			

Machaca con cero el puntero a nuestro serial falso

Address	Hex dump	ASCII		0012F3B8	0015D3BC	Arg1 = 0015D3BC
0012F454	00 00 00 00	A4 BC CF 00A4BC			
0012F45C	EC F4 12 00	C8 F5 12 00	...C8F5			
0012F464	3C BB CF 00	E8 F4 12 00	...E8F4			
0012F46C	FF FF FF FF	15 20 00 00	...1520			
0012F474	02 E3 13 74	E8 F4 12 00	...E8F4			
0012F47C	00 00 00 00	00 00 00 000000			

Ojo no borro el serial falso, este sigue estando en 15d3bc lo que borro es el puntero al mismo que estaba en EBP-8c.

Address	Hex dump	ASCII		0012F3B8	0015D3BC	Arg1 = 0015D3BC
7413E6D5	E8 793AF0FF		CALL <JMP.&OLEAUT32.#6>			
7413E6DA	C7042B 00000000		MOV DWORD PTR DS:[EBX+EBP],0			
7413E6E1	4F		DEC EDI			
7413E6E2	7F E8		JG SHORT MSUBUM50.7413E6CC			
7413E6E4	33C0		XOR EAX,EAX			
7413E6E6	8A06		MOV AL,BYTE PTR DS:[ESI]			
7413E6E8	46		INC ESI			
7413E6E9	FF2485 94ED137		JMP DWORD PTR DS:[EAX*4+7413ED94]			
7413E6F0	0FBF3E		MOVSX EDI,WORD PTR DS:[ESI]			

Alli llega al siguiente OPCODE

401BED: 1a FF free1Ad local_0088
 401BF0: 1c BranchF: 401BF6
 401BF3: 1e Branch: 401c94
 401BFE: 1a FF free1Ad local_0088

Este seguro borrara el contenido de la variable local ebp-88

Que había allí?

Address	Hex dump	ASCII		0012F3B8	0015D3BC	Arg1 = 0015D3BC
00401C85	00 00 00 00	00 00 00 000000			
Command	? ebp - 88					

Address	Hex dump	ASCII
0012F458	A4 BC CF 00 EC F4 12 00	...000.
0012F460	C8 F5 12 00 3C BB CF 00	...<00.
0012F468	E8 F4 12 00 FF FF FF FF	...000.
0012F470	15 20 00 00 02 E3 13 74	...00!!t
0012F478	E8 F4 12 00 00 00 00 00	...000.
0012F480	00 00 00 00 00 00 00 00	...000.
0012F488	00 18 40 00 E4 17 40 00	...000.
0012F490	98 1C 40 00 A4 BC CF 00	...000.
0012F498	00 E0 00 00 88 98 15 00	...000.

Ah el valor que había hallado de la tabla de ítems, seguro lo borrara veamos, ya nos estamos haciendo mas prácticos jeje.

7413E726	BF 01000000	MOV EDI,1
7413E72B	0FBF1E	MOVSX EBX,WORD PTR DS:[ESI]
7413E72E	83C6 02	ADD ESI,2
7413E731	8B042B	MOV EAX,DWORD PTR DS:[EBX+EBP]
7413E734	0BC0	OR EAX,EAX
7413E736	74 0D	JE SHORT MSUBUM50.7413E745

Allí lee los parámetros del opcode que son

Address	Hex dump	ASCII
00401BE6	01 00 FB 30 2F 74 FF 1A	0.10/t +
00401BEE	78 FF 1C 26 00 1E C4 00	x L%.<.
00401BF6	FE C1 54 FF 1A C4 03 00	...T +.
00401BFE	FC F6 64 FF 04 74 FF 21	...d +t !
00401C06	0F 00 03 19 78 FF 08 78	*.0x 0x

FF78 que completara con FFs y sera el valor -88 hexa

7413E72E	83C6 02	ADD ESI,2
7413E731	8B042B	MOV EAX,DWORD PTR DS:[EBX+EBP]
7413E734	0BC0	OR EAX,EAX
7413E736	74 0D	JE SHORT MSUBUM50.7413E745
7413E738	50	PUSH EAX
7413E739	8B00	MOV EAX,DWORD PTR DS:[EAX]
7413E73B	FF50 08	CALL DWORD PTR DS:[EAX+8]

Luego mueve a EAX el contenido de EBP-88 testea si es cero, como no lo es sigue adelante.

7413E739	8B00	MOV EAX,DWORD PTR DS:[EAX]
7413E73B	FF50 08	CALL DWORD PTR DS:[EAX+8]
7413E73E	C7042B 00000000	MOV DWORD PTR DS:[EBX+EBP],0
7413E745	4F	DEC EDI
7413E746	7F E3	JG SHORT MSUBUM50.7413E72B
7413E748	33C0	XOR EAX,EAX

Y ira a un call que como antes liberara el valor este, y luego borrara el contenido de ebp-88.

Address	Hex dump	ASCII
0012F458	00 00 00 00 EC F4 12 00	...000.
0012F460	C8 F5 12 00 3C BB CF 00	...<00.
0012F468	E8 F4 12 00 FF FF FF FF	...000.

Allí esta puesto a cero.

7413E746	7F E3	JG SHORT MSUBUM50.7413E72B
7413E748	33C0	XOR EAX,EAX
7413E74A	8A06	MOV AL,BYTE PTR DS:[ESI]
7413E74C	46	INC ESI
7413E74D	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E754	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]
7413E757	83C6 02	ADD ESI,2
7413E75A	D1EF	SHR EDI,1

El próximo OPCODE es

```

401BEA: 2f FF free1Str      local_008C
401BED: 1a FF free1Ad      local_0088
401BF0: 1c BranchF:        401BF6
401BF3: 1e Branch:         401c94
401BF6: Lead3/c1 LitVar4: { local_3BE500AC

```

Es un salto condicional ya que todos los BRANCH son saltos

Instrucción	Opcode	Significado
Branch	1e	Salto incondicional.
BranchF	1c	Salta si False.
BranchT	1d	Salta si True.

O sea que es un salto si es falso y a continuación hay un JMP o sea que este salto al ver que tipeamos algo en el textbox salta a la comparación del serial, si no volvería a repetir el proceso con el JMP.

Veamos si es cierto, si esto salta, el próximo opcode debería ser

```

401BED: 1a FF000000 local_0088
401BF0: 1c BranchF: 401BF6
401BF3: 1e Branch: 401c94
401BF6: Lead3/c1 LitVar14: [local_3BE500AC] 0x3c41a [246810]
401BFF: Lead1/K6 FstVar local_009C

```

Ya que este salto condicional evita el JMP de 401bf3

C File View Debug Plugins Options Window Help			
7413D524	0FB736	MOVZX ESI,WORD PTR DS:[ESI]	
7413D527	0375 A8	ADD ESI,DWORD PTR SS:[EBP-58]	
7413D52A	33C0	XOR EAX,EAX	
7413D52C	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413D52E	46	INC ESI	
7413D52F	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D536	33C0	XOR EAX,EAX	
7413D538	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413D53B	83C6 03	ADD ESI,3	
7413D53E	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D545	59	POP ECX	
7413D546	66:0BC9	OR CX,CX	
7413D549	74 D9	JE SHORT MSUBUM50.7413D524	
7413D54B	33C0	XOR EAX,EAX	
7413D54D	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413D550	83C6 03	ADD ESI,3	
7413D553	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D55A	58	POP EAX	
7413D55B	0FBFD0	MOVSX EDX,AX	
7413D55E	3BC2	CMP EAX,EDX	
7413D560	0F85 5E050000	JNZ MSUBUM50.7413DAC4	
7413D566	50	PUSH EAX	
7413D567	2E:8BC0	MOV EAX,EAX	
7413D56A	33C0	XOR EAX,EAX	
7413D56C	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413D56E	46	INC ESI	
7413D56F	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D576	D9FC	FRNDINT	
7413D578	83EC 04	SUB ESP,4	
7413D57B	DF1C24	FISTP WORD PTR SS:[ESP]	
7413D57E	DFF0	FSTSW AX	
7413D580	A8 00	TEST AL,00	
7413D582	0F85 1ECC0000	JNZ MSUBUM50.7414A1A6	
7413D588	33C0	XOR EAX,EAX	
7413D58A	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413D58C	46	INC ESI	
7413D58D	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D594	E8 41E4F1FF	CALL MSUBUM50.vbaI2Var	

Allí termina el opcode y lee el próximo

7413D527	0375 A8	ADD ESI,DWORD PTR SS:[EBP-58]
7413D52A	33C0	XOR EAX,EAX
7413D52C	8A06	MOV AL,BYTE PTR DS:[ESI]
7413D52E	46	INC ESI
7413D52F	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D536	33C0	XOR EAX,EAX
7413D538	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D53B	83C6 03	ADD ESI,3
7413D53E	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D545	59	POP ECX
7413D546	66:0BC9	OR CX,CX
7413D549	74 D9	JE SHORT MSUBUM50.7413D524
7413D54B	33C0	XOR EAX,EAX
7413D54D	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D550	83C6 03	ADD ESI,3
7413D553	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D55A	58	POP EAX
7413D55B	0FBFD0	MOVSX EDX,AX
7413D55E	3BC2	CMP EAX,EDX
7413D560	0F85 5E050000	JNZ MSUBUM50.7413DAC4
7413D566	50	PUSH EAX
7413D567	2E:8BC0	MOV EAX,EAX
7413D56A	33C0	XOR EAX,EAX
7413D56C	8A06	MOV AL,BYTE PTR DS:[ESI]
7413D56E	46	INC ESI
7413D56F	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D576	D9FC	FRNDINT
7413D578	83EC 04	SUB ESP,4
7413D57B	DF1C24	FISTP WORD PTR SS:[ESP]
7413D57E	DFF0	FSTSW AX
7413D580	A8 00	TEST AL,00
7413D582	0F85 1ECC0000	JNZ MSUBUM50.7414A1A6
7413D588	33C0	XOR EAX,EAX
7413D58A	8A06	MOV AL,BYTE PTR DS:[ESI]
7413D58C	46	INC ESI
7413D58D	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D594	E8 41E4F1FF	CALL MSUBUM50.vbaI2Var

DS:[00401BF6]=FE
AL=00

Que como suponíamos es el FE de 401bf6, así que el salto condicional salto y evito el JMP, jeje.

Lead3/c1 LitVar14

Este es un OPCODE doble veamos que hace

7413ECAB	33C0	XOR EAX,EAX
7413ECAD	8A06	MOV AL,BYTE PTR DS:[ESI]
7413ECAF	46	INC ESI
7413ECB0	FF2485 94FD137	JMP DWORD PTR DS:[EAX*4+7413FD94]
7413ECB7	33C0	XOR EAX,EAX
7413ECB9	8A06	MOV AL,BYTE PTR DS:[ESI]

Allí cierra el primer OPCODE y lee el segundo

7413DEEF	BB 03000000	MOV EBX,3
7413DEF4	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]
7413DEF7	66:891C2F	MOV WORD PTR DS:[EDI+EBP],BX
7413DEFB	8B46 02	MOV EAX,DWORD PTR DS:[ESI+2]
7413DEFE	83C6 06	ADD ESI,6
7413DF01	89442F 08	MOV DWORD PTR DS:[EDI+EBP+8],EAX
7413DF05	EB D9	JMP SHORT MSUBUM50.7413DEE0
7413DF07	BB 05000000	MOV EBX,5

Que empieza aquí traceemos con paciencia

Address	Hex dump	ASCII
00401BF0	1C 26 00 1E C4 00 FE C1	L&.▲-.■±
00401BF8	54 FF 1A C4 03 00 FC F6	T →◆.3÷
00401C00	64 FF 04 74 FF 21 0F 00	d ◆t †*.
00401C08	03 19 78 FF 08 78 FF 00	◆+x 0x .
00401C10	A0 00 00 00 6C 74 FF 0A	á...lt.
00401C18	02 00 04 00 FD 6B 54 FF	0.◆.²kT

Lee los parámetros del opcode

Registers (FPU)	
EAX	000000C1
ECX	00000000
EDX	00000000
EBX	00000003
ESP	0012F3C4
EBP	0012F4E0
ESI	00401BF8
EDI	FFFFFF54
EIP	7413DEF7 MSUI
C 0	ES 0023 32b

Ya sabemos que los completa con FFs si es negativo como en este caso

7413DEEF	BB 03000000	MOV EBX,3
7413DEF4	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]
7413DEF7	66:891C2F	MOV WORD PTR DS:[EDI+EBP],BX
7413DEFB	8B46 02	MOV EAX,DWORD PTR DS:[ESI+2]
7413DEFE	83C6 06	ADD ESI,6
7413DF01	89442F 08	MOV DWORD PTR DS:[EDI+EBP+8],EAX
7413DF05	EB D9	JMP SHORT MSUBUM50.7413DEE0
7413DF07	BB 05000000	MOV EBX,5

Lee mas parámetros en este caso es un DWORD entero que pasa a EAX

Address	Hex dump	ASCII
00401BF0	1C 26 00 1E C4 00 FE C1	L&.▲-.■±
00401BF8	54 FF 1A C4 03 00 FC F6	T →◆.3÷
00401C00	64 FF 04 74 FF 21 0F 00	d ◆t †*.
00401C08	03 19 78 FF 08 78 FF 00	◆+x 0x .
00401C10	A0 00 00 00 6C 74 FF 0A	á...lt.
00401C18	02 00 04 00 FD 6B 54 FF	0.◆.²kT

Registers (FPU)	
EAX	0003C41A
ECX	00000000
EDX	00000000
EBX	00000003
ESP	0012F3C4
EBP	0012F4E0
ESI	00401BF8
EDI	FFFFFFFF
EIP	7413DEFE

Y ese valor lo guarda en una variable local que esta dada por la operación

7413DEEF	BB 03000000	MOV EBX,3
7413DEF4	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]
7413DEF7	66:891C2F	MOV WORD PTR DS:[EDI+EBP],BX
7413DEFB	8B46 02	MOV EAX,DWORD PTR DS:[ESI+2]
7413DEFE	83C6 06	ADD ESI,6
7413DF01	89442F 08	MOV DWORD PTR DS:[EDI+EBP+8],EAX
7413DF05	EB 09	JMP SHORT MSUBUM50.7413DEE0
7413DF07	BB 05000000	MOV EBX,5
7413DF0C	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]
7413DF0F	66:891C2F	MOV WORD PTR DS:[EDI+EBP],BX

EBP+ FFFFFFF54 + 8 o sea que le suma el primer parámetro y 8 esto da 12f43c, allí guarda el valor

7413DF61	58	POP EAX
7413DF62	66:290424	SUB WORD PTR DS:[ESI],EAX
7413DF66	0FBF3E	JMP MSUBUM50
EAX=0003C41A		
Stack DS:[0012F43C]=00000000		
Address	Hex dump	ASCII

Address	Hex dump	ASCII
0012F43C	1A C4 03 00 00 00 00 00	→.....
0012F444	00 00 00 00 00 00 00 00
0012F44C	00 00 00 00 00 00 00 00
0012F454	00 00 00 00 00 00 00 00
0012F45C	FC F4 12 00 00 00 00 00	↓.....

Y esperen que aun no termino que el jmp nos lleva a

7413DEE0	03FD	ADD EDI,EBP
7413DEE2	57	PUSH EDI
7413DEE3	33C0	XOR EAX,EAX
7413DEE5	8A06	MOV AL,BYTE PTR DS:[ESI]

Donde pushea 12f434

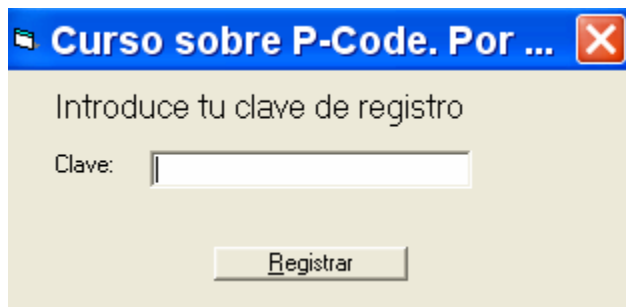
0012F3C0	0012F434
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000
0012F3D0	00000000

Que es el puntero a un estructura donde empieza con el 3 que había guardado, y mas abajo el valor guardado.

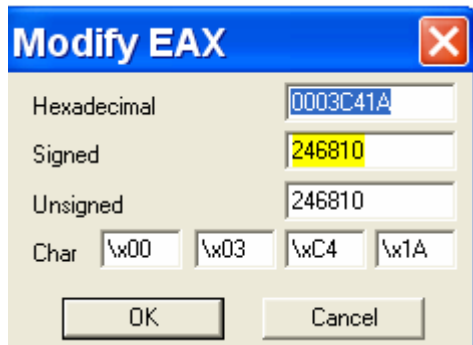
Address	Hex dump	ASCII
0012F434	03 00 00 00 00 00 00 00
0012F43C	1A C4 03 00 00 00 00 00	→.....
0012F444	00 00 00 00 00 00 00 00

Sigamos ya llegamos jeje

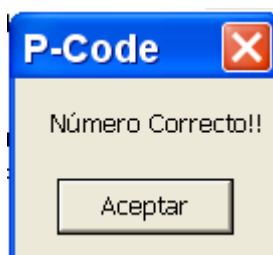
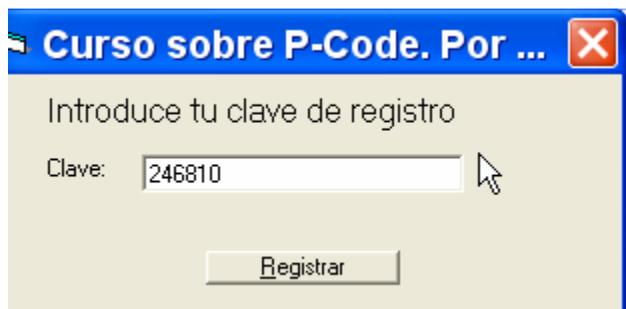
Cualquiera aquí ya que es un serial fijo probaría pasar este valor a decimal y intentar a ver si es el serial correcto, si abro otra instancia del crackme fuera de OLLYDBG



Miro en OLLY cuanto vale ese número en decimal



Vale 246810 tipeemoslo en el crackme

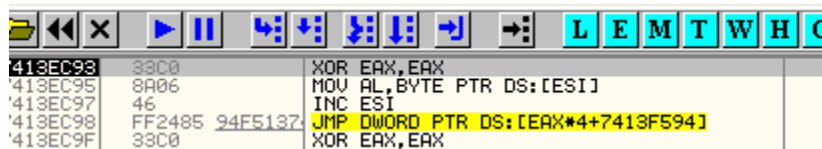


Jeje ya sabemos algo pero no se libran tan fácil de esto, lleguemos a la comparación.

401BF3: 1e Branch:	401c94
401BF6: Lead3/c1 LitVarI4:	{ local_3BE500AC } 0x3c41a [246810]
401BFE: Lead1/#6 FStVar	local_009C
401C02: 04 FLdRfVar	local_008C
401C05: 21 FLdPrThis	
401C06: 0f VCallAd	text
401C09: 19 FStAdFunc	local_0088

El próximo OP CODE también es doble

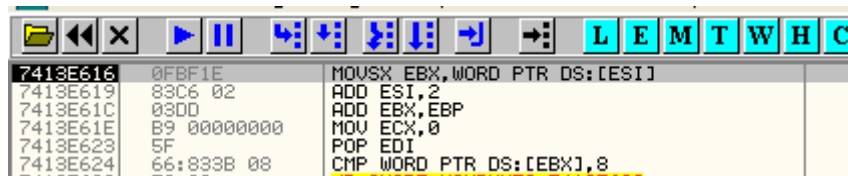
Es FC y enseguida lo cierra y carga el segundo



Address	Disassembly
413EC98	JMP DWORD PTR DS:[EAX*4+7413F594]

Registers (FPU): EAX: 000000F6, ECX: 00000000, EDX: 00000000, EBP: 0012F3C0, ESP: 0012F3C0

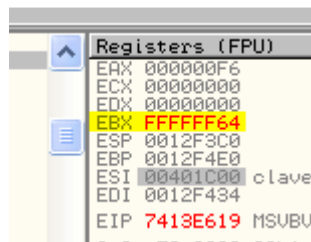
Que es F6 entremos en el sin miedo (ya a que podemos temer jeje), recordemos que el exdec nos dice que trabajara con la variable local 9c o sea ebp-9c.



Address	Disassembly
7413E616	MOV EBX, WORD PTR DS:[ESI]

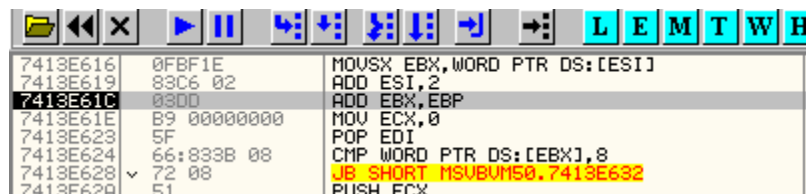
Registers (FPU): EAX: 000000F6, ECX: 00000000, EDX: 00000000, EBP: 0012F3C0, ESP: 0012F3C0

Allí lee los parámetros y los completa con FFs



Registers (FPU): EAX: 000000F6, ECX: 00000000, EDX: 00000000, EBX: FFFFFFFF64, ESP: 0012F3C0, EBP: 0012F4E0, ESI: 00401C00, EDI: 0012F434, EIP: 7413E619

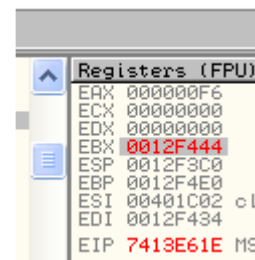
Por supuesto ese valor es -9c



Address	Disassembly
7413E61C	ADD EBX, EBP

Registers (FPU): EAX: 000000F6, ECX: 00000000, EDX: 00000000, EBP: 0012F4E0, ESP: 0012F3C0

Lo suma a EBP para hallar EBP-9c que es 12f444 que por ahora esta vacío



Registers (FPU): EAX: 000000F6, ECX: 00000000, EDX: 00000000, EBX: 0012F444, ESP: 0012F3C0, EBP: 0012F4E0, ESI: 00401C02, EDI: 0012F434, EIP: 7413E61E

Address	Hex dump	ASCII
0012F444	00 00 00 00 00 00 00 00
0012F44C	00 00 00 00 00 00 00 00
0012F454	00 00 00 00 00 00 00 00
0012F45C	EC F4 12 00 C8 F5 12 00
0012F464	3C BB CF 00 E8 F4 12 00
0012F46C	FF FF FF FF 15 20 00 00
0012F474	02 E3 13 74 E8 F4 12 00
0012F47C	00 00 00 00 00 00 00 00
0012F484	00 00 00 00 D0 1B 40 00

7413E61E	B9 00000000	MOV ECX,0
7413E623	5F	POP EDI
7413E624	66:833B 08	CMP WORD PTR DS:[EBX],8
7413E628	72 08	JB SHORT MSUBUM50.7413E632
7413E62A	51	PUSH ECX
7413E62B	53	PUSH EBX
7413E62C	E8 CFE9FFFF	CALL MSUBUM50.7413D000
7413E631	59	POP ECX
7413E632	8B47	MOV FAX.DWORD PTR DS:[EDI]

Luego compara si es mas bajo que 8 como es cierto salta.

413E632	8B07	MOV EAX,DWORD PTR DS:[EDI]
413E634	66:83F8 09	CMP AX,9
413E638	75 15	JNZ SHORT MSUBUM50.7413E64F
413E63A	8B07	MOV EDI,EDI
413E63C	8BCB	MOV ECX,EBX
413E63E	E8 E0270000	CALL MSUBUM50.7413D000
413E643	33C0	XOR EAX,EAX
413E645	8A06	MOV AL,BYTE PTR DS:[ESI]
413E647	46	INC ESI
413E648	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
413E64F	66:83F8 0D	CMP AX,0D
413E653	0F84 88750000	JE MSUBUM50.74145BE1
413E659	8903	MOV DWORD PTR DS:[EBX],EAX

Luego vuelve a saltar no entraremos en demasiado detalle, llega al final aquí donde se ven las últimas líneas del OPCODE.

7413E653	0F84 88750000	JE MSUBUM50.74145BE1
7413E659	8903	MOV DWORD PTR DS:[EBX],EAX
7413E65B	66:890F	MOV WORD PTR DS:[EDI],CX
7413E65E	8B4F 04	MOV ECX,DWORD PTR DS:[EDI+4]
7413E661	894B 04	MOV DWORD PTR DS:[EBX+4],ECX
7413E664	8B4F 08	MOV ECX,DWORD PTR DS:[EDI+8]
7413E667	894B 08	MOV DWORD PTR DS:[EBX+8],ECX
7413E66A	8B4F 0C	MOV ECX,DWORD PTR DS:[EDI+C]
7413E66D	894B 0C	MOV DWORD PTR DS:[EBX+C],ECX
7413E670	33C0	XOR EAX,EAX
7413E672	8A06	MOV AL,BYTE PTR DS:[ESI]
7413E674	46	INC ESI
7413E675	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]

Address	Hex dump	ASCII
0012F444	03 00 00 00 00 00 00 00
0012F44C	00 00 00 00 00 00 00 00
0012F454	00 00 00 00 00 00 00 00
0012F45C	EC F4 12 00 C8 F5 12 00
0012F464	3C BB CF 00 E8 F4 12 00

Mueve el 3 que esta en EAX a la variable EBP-9C

Y borrar el 3 de la estructura anterior o sea que esta copiando el número, desde donde estaba a una nueva dirección.

Aquí estaba y borra el 3

Address	Hex dump	ASCII
0012F434	03 00 00 00 00 00 00 00
0012F43C	1A C4 03 00 00 00 00 00
0012F444	03 00 00 00 00 00 00 00
0012F44C	00 00 00 00 00 00 00 00

Al ejecutar

ECX=00000000			
Address	Hex dump	ASCII	
0012F434	00 00 00 00 00 00 00 00	
0012F43C	1A C4 03 00 00 00 00 00	+-.....	
0012F444	03 00 00 00 00 00 00 00	*.....	
0012F44C	00 00 00 00 00 00 00 00	
0012F454	00 00 00 00 00 00 00 00	
0012F45C	EC F4 12 00 C8 F5 12 00	q4+.53+	

Luego las siguientes líneas copian todo lo que hay aquí, a la nueva dirección o sea a EBP-9c

Address	Hex dump	ASCII	
0012F444	03 00 00 00 00 00 00 00	*.....	
0012F44C	1A C4 03 00 00 00 00 00	+-.....	
0012F454	00 00 00 00 00 00 00 00	
0012F45C	EC F4 12 00 C8 F5 12 00	q4+.53+	

Allí lo vemos al numero que será el serial correcto en la estructura que comienza en EBP-9c

7413E660	894B 0C	MOV DWORD PTR DS:[EBX+C],ECX	
7413E670	33C0	XOR EAX,EAX	
7413E672	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E674	46	INC ESI	
7413E675	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413E67C	E8 2FEDFFFF	CALL MSUBUM50.7413D3B0	
7413E681	C1E0 04	SHL EAX,4	
7413E684	03D8	ADD EBX,EAX	
7413E686	EB 96	JMP SHORT MSUBUM50.7413E61E	
7413E688	0F0F0F0F	MOVSX EDI,WORD PTR DS:[ESI]	

Llegamos al siguiente OPCODE

401C02: 04 FLdRfVar local_008C

Vemos que repite todo lo que ya vimos hasta aquí al inicio

401C02: 04 FLdRfVar local_008C
401C05: 21 FLdPrThis
401C06: 0f VCallAd text
401C09: 19 FStAdFunc local_0088
401C0C: 08 FLdPr local_0088
401C0F: 0d VCallHresult get__ipropTEXTEDIT
401C14: 6c ILdRf local_008C

Es todo similar al inicio, el siguiente OPCODE es

401C17: 0a ImpAdCallFPR4: rtcR8ValFromBstr
401C1C: Lead2/6b CVarR8
401C20: 5d HardType

Así que para saltar todo lo anterior pongo un BPM ON ACCESS en 401c17, para que pare cuando lea el OPCODE.

7413D94F	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413D952	83C6 03	ADD ESI,3	
7413D955	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D95C	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413D95F	FF3428	PUSH DWORD PTR DS:[EAX+EBP]	
7413D962	C70428 00000000	MOV DWORD PTR DS:[EAX+EBP],0	
7413D968	33C0	XOR EAX,EAX	

Allí paro y lee el OPCODE 0A que vemos en el EXDEC

ImpAdCallFPR4 vemos que significa el llamado a la api que nos marca el EXDEC

Por ejemplo

4017F5: 0a ImpAdCallFPR4: _rtcMsgBox <---Llamada a la función

En este ejemplo seria un llamado a la api rtcMsgBox, en nuestro caso es un llamado a la api

401C17: 0a ImpAdCallFPR4: _rtcR8ValFromBstr

7413E7A6	0FB70E	MOVZX ECX,WORD PTR DS:[ESI]
7413E7A9	8B55 AC	MOV EDX,DWORD PTR SS:[EBP-54]
7413E7AC	8B048A	MOV EAX,DWORD PTR DS:[EDX+ECX*4]
7413E7AF	0BC0	OR EAX,EAX

Lee los parámetros del opcode

Address	Hex dump	ASCII
00401C17	0A 02 00 04 00 FD 6B 54	.0.♦.²kT
00401C1F	FF 5D 04 64 FF FB 40 2F]♦d '0/
00401C27	74 FF 1A 78 FF 1C 93 00	t *x L0.

Registers (FPU)	
EAX	00000000
ECX	00000002
EDX	00159888
EBX	FFFFFFFF
ESP	0012F3C0
EBP	0012F4E0
ESI	00401C18 clave
EDI	000000A0
EIP	7413E7A9 MSUBU
C 0	ES 0023 32bit

Los mueve a ECX

7413E7A6	0FB70E	MOVZX ECX,WORD PTR DS:[ESI]		
7413E7A9	8B55 AC	MOV EDX,DWORD PTR SS:[EBP-54]		
7413E7AC	8B048A	MOV EAX,DWORD PTR DS:[EDX+ECX*4]		
7413E7AF	0BC0	OR EAX,EAX	clave1.00401000	
7413E7B1	0F84 78790000	JL MSUBUMS0.7414612F		
7413E7B7	0FB77E 02	MOVZX EDI,WORD PTR DS:[ESI+2]		
7413E7BB	83C6 04	ADD ESI,4		
7413E7BE	03FC	ADD EDI,ESP		
7413E7C0	FFD0	CALL EAX		
7413E7C2	3BFC	CMP EDI,ESP		
7413E7C4	0F85 5B790000	JNZ MSUBUMS0.74146125		

Registers (FPU)	
EAX	00401000 <J
ECX	00000002
EDX	004017E4 c.l
EBX	FFFFFFFF
ESP	0012F3C0

Luego mueve a EAX el valor 401000 y lo testea si es cero

7413E7B7	0FB77E 02	MOVZX EDI,WORD PTR DS:[ESI+2]
7413E7BB	83C6 04	ADD ESI,4
7413E7BE	03FC	ADD EDI,ESP
7413E7C0	FFD0	CALL EAX
7413E7C2	3BFC	CMP EDI,ESP
7413E7C4	0F85 5B790000	JNZ MSUBUMS0.74146125

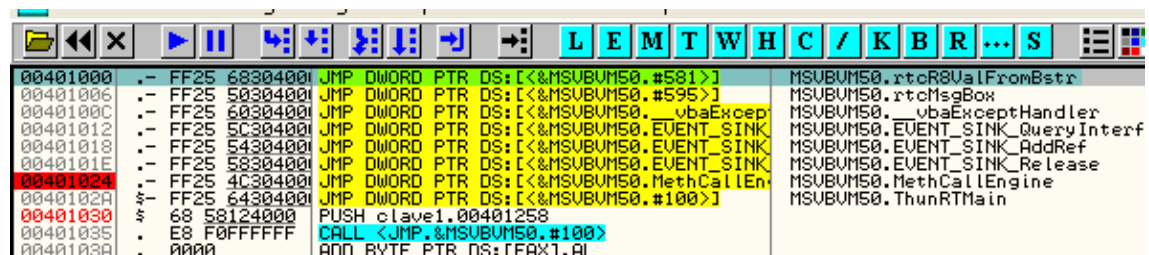
Luego lee un segundo parámetro

Address	Hex dump	ASCII
00401C17	0A 02 00 04 00 FD 6B 54	.0.♦.²kT
00401C1F	FF 5D 04 64 FF FB 40 2F]♦d '0/
00401C27	74 FF 1A 78 FF 1C 93 00	t *x L0.

Registers (FPU)	
EAX	00401000 <J
ECX	00000002
EDX	004017E4 c
EBX	FFFFFFFF
ESP	0012F3C0
EBP	0012F4E0
ESI	00401C18 c
EDI	00000004
EIP	7413E7BB MS

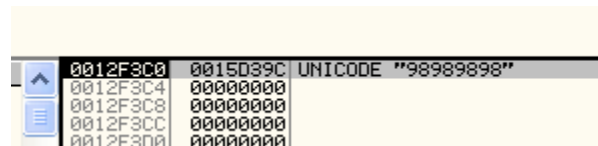
7413E7BB	83C6 04	ADD ESI,4
7413E7BE	03FC	ADD EDI,ESP
7413E7C0	FFD0	CALL EAX
7413E7C2	3BFC	CMP EDI,ESP
7413E7C4	0F85 5B790000	JNZ MSUBUMS0.74146125
7413E7C6	3BFC	CMP EDI,ESP

Y llega al call eax donde EAX vale 401000, veamos donde va, va a la api susodicha.



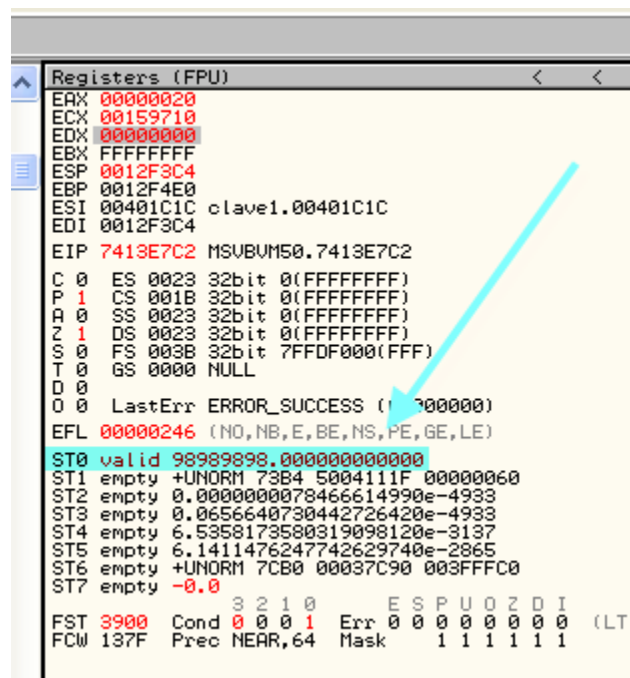
```
00401000  FF25 68304000 JMP DWORD PTR DS:[<&MSUBUM50.#581>] MSUBUM50.rtcR8ValFromBstr
00401006  FF25 50304000 JMP DWORD PTR DS:[<&MSUBUM50.#595>] MSUBUM50.rtcMsgBox
0040100C  FF25 60304000 JMP DWORD PTR DS:[<&MSUBUM50._vbaExcept MSUBUM50._vbaExceptionHandler
00401012  FF25 5C304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK MSUBUM50.EVENT_SINK_QueryInterf
00401018  FF25 54304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK MSUBUM50.EVENT_SINK_AddRef
0040101E  FF25 58304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK MSUBUM50.EVENT_SINK_Release
00401024  FF25 4C304000 JMP DWORD PTR DS:[<&MSUBUM50.MethCallEng MSUBUM50.MethCallEngine
0040102A  FF25 64304000 JMP DWORD PTR DS:[<&MSUBUM50.#100>] MSUBUM50.ThunRTMain
00401030  68 58124000 PUSH clave1.00401258
00401035  E8 F0FFFFFF CALL <JMP.&MSUBUM50.#100>
0040103A  AAAA AND BYTE PTR DS:[EAX],0
```

Y el parámetro que le pasamos en el stack a la api es



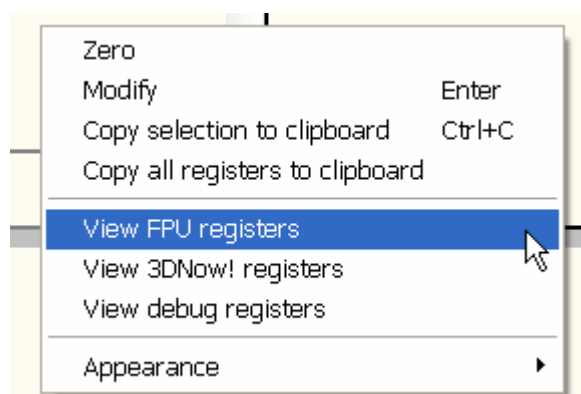
```
0012F3C0 0015D39C UNICODE "98989898"
0012F3C4 00000000
0012F3C8 00000000
0012F3CC 00000000
0012F3D0 00000000
```

Mi serial falso



```
Registers (FPU)
EAX 00000020
ECX 00159710
EDX 00000000
EBX FFFFFFFF
ESP 0012F3C4
EBP 0012F4E0
ESI 00401C1C clave1.00401C1C
EDI 0012F3C4
EIP 7413E7C2 MSUBUM50.7413E7C2
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
O 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 valid 98989898.000000000000
ST1 empty +UNORM 73B4 5004111F 00000060
ST2 empty 0.0000000078466614990e-4933
ST3 empty 0.0656640730442726420e-4933
ST4 empty 6.5358173580319098120e-3137
ST5 empty 6.1411476247742629740e-2865
ST6 empty +UNORM 7CB0 00037C90 003FFFC0
ST7 empty -0.0
FST 3900 Cond 0 0 0 1 Err 0 0 0 0 0 0 0 0 (LT)
FCW 137F Prec NEAR,64 Mask 1 1 1 1 1 1
```

El cual fue cargado en St0 que es una entrada del stack de punto flotante que aun no hemos explicado, pero bueno, allí lo ven esta debajo de los registros, si no le sale visible, pues tienen la vista en otra opción hacer click derecho



Allí esta el tipo cargo mi serial falso allí.

7413E7C2	3BFC	CMP EDI,ESP
7413E7C4	0F85 5B790000	JNZ MSUBUM50.74146125
7413E7C8	33C0	XOR EAX,EAX
7413E7CC	8A06	MOV AL,BYTE PTR DS:[ESI]
7413E7CE	46	INC ESI
7413E7CF	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E7D6	0FB70E	MOVZX ECX,WORD PTR DS:[ESI]
7413E7D9	8B55 AC	MOV EDX,DWORD PTR SS:[EBP-54]
7413E7DC	8B048A	MOV FAX,DWORD PTR DS:[EDX+ECX*4]

Llego al siguiente OPCODE

401C1C: Lead2/6b CVarR8

Pues allí estamos entremos en el opcode

7413EC9F	33C0	XOR EAX,EAX
7413ECA1	8A06	MOV AL,BYTE PTR DS:[ESI]
7413ECA3	46	INC ESI
7413ECA4	FF2485 94F9137	JMP DWORD PTR DS:[EAX*4+7413F994]

Como es un opcode doble borra el primero y carga el segundo

7413D804	BA 05000000	MOV EDX,5
7413D809	0FBF06	MOVZX EAX,WORD PTR DS:[ESI]
7413D80C	03C5	ADD EAX,EBP
7413D80E	DD58 08	FSTP QWORD PTR DS:[EAX+8]
7413D811	66:8910	MOV WORD PTR DS:[EAX],DX
7413D814	50	PUSH EAX
7413D815	DFF0	FSTSW AX
7413D817	A8 0D	TEST AL,0D
7413D819	0F85 87C90000	JNZ MSUBUM50.7414A1A6
7413D81F	33C0	XOR EAX,EAX

Bueno allí hay unos comandos de punto flotante que aun no hemos visto

Pero vemos que al inicio esta cargando parámetros

Address	Hex dump	ASCII
00401C16	FF 0A 02 00 04 00 FD 6B	.0.♦.²k
00401C1E	54 FF 5D 04 64 FF FB 40	T J♦d '0
00401C26	2F 74 FF 1A 78 FF 1C 93	/t ♦x L0
00401C2E	00 27 E4 FE 27 04 FF 3A	.'♦♦♦♦

En este caso

Registers (FPU)		
EAX	FFFFFFFF54	
ECX	00159710	
EDX	00000005	
EBX	FFFFFFFF	
ESP	0012F3C4	
EBP	0012F4E0	
ESI	00401C1E	clave1
EDI	0012F3C4	
EIP	7413D80C	MSUBUM

7413D804	BA 05000000	MOV EDX,5
7413D809	0FBF06	MOVZX EAX,WORD PTR DS:[ESI]
7413D80C	03C5	ADD EAX,EBP
7413D80E	DD58 08	FSTP QWORD PTR DS:[EAX+8]
7413D811	66:8910	MOV WORD PTR DS:[EAX],DX
7413D814	50	PUSH EAX

Lo suma a EBP quedando EAX en

Registers (FPU)	
EAX	0012F434
ECX	00159710
EDX	00000005
EBX	FFFFFFFF
ESP	0012F3C4
EBP	0012F4E0
ESI	00401C1E cl:
EDI	0012F3C4
EIP	7413D80E MSI

7413D809	0F BF 06	MOVSS EAX,WORD PTR DS:[ESI]
7413D80C	03 C5	ADD EAX,EBP
7413D80E	DD 58 08	FSTP QWORD PTR DS:[EAX+8]
7413D811	66:89 10	MOV WORD PTR DS:[EAX],DX
7413D814	50	PUSH EAX
7413D815	DF E0	FSTSW AX
7413D817	A8 0D	TEST AL,0D
7413D819	74 14 01 06	JNZ MSURUM50.74140106

Con FSTP guardara el primer valor del stack de punto flotante ST0, a la dirección de memoria en este caso [EAX+8] o sea 12f43c y hará POP el stack de punto flotante bueno eso ya lo explicaremos mas adelante.

Address	Hex dump	ASCII
0012F43C	1A C4 03 00 00 00 00 00
0012F444	03 00 00 00 00 00 00 00
0012F44C	1A C4 03 00 00 00 00 00
0012F454	9C D3 15 00 A4 BC CF 00	ES. #.
0012F45C	EC F4 12 00 C8 F5 12 00	g. #. #.

Al ejecutar

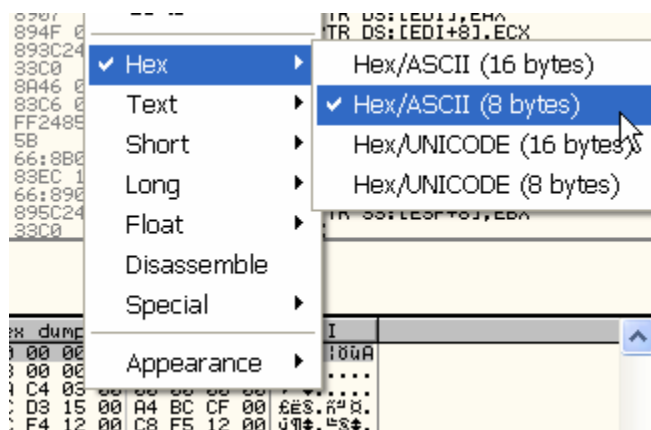
DX=0005		
Stack DS:[0012F434]=0000		
Address	Hex dump	ASCII
0012F43C	00 00 00 28 DD 99 97 41	... (100A
0012F444	03 00 00 00 00 00 00 00
0012F44C	1A C4 03 00 00 00 00 00
0012F454	9C D3 15 00 A4 BC CF 00	ES. #.
0012F45C	EC F4 12 00 C8 F5 12 00	g. #. #.

Ya se que ustedes pueden tener alguna duda que ese es nuestro serial falso, lo que pasa es que esta transformado a 64 bit doble, veamos si hago click derecho

3D869	FF2485 94ED1	Short	[EAX*4+7413ED94]
3D870	5B	Long	S:[ESI]
3D871	66:8B06	Float	FSTP QWORD PTR DS:[EAX+8]
3D874	83EC 10	Disassemble	
3D877	66:890424	Special	
3D87B	895C24 08	Appearance	
3D87F	33C0		
0005			
ck DS:[0012F434]=0000			
Address	Hex dump		
2F43C	00 00 00 28 DD 99 97 41		0012F
2F444	03 00 00 00 00 00 00 00		0012F
2F44C	1A C4 03 00 00 00 00 00		0012F
2F454	9C D3 15 00 A4 BC CF 00		0012F
2F45C	EC F4 12 00 C8 F5 12 00		0012F

Address	64-bit double	
0012F43C	98989898.00000000	1.482196937523740e-323
0012F44C	1.219403420500781e-318	9.038962733673929e-305
0012F45C	2.636724066601938e-308	2.636248745657113e-308
0012F46C	1.743007342698825e-310	2.636249701104218e-308
0012F47C	0.0	1.792145974831099e-307
0012F48C	1.792485495817880e-307	1.216837333634428e-309
0012F49C	6.992530848216996e-318	5.562684646268003e-309
0012F4AC	2.636197811031977e-308	2.639075238026684e-308
0012F4BC	1.423837248936062e+251	2.636231763578288e-308
0012F4CC	3.003269133012093e-308	2.075878075142002e-317

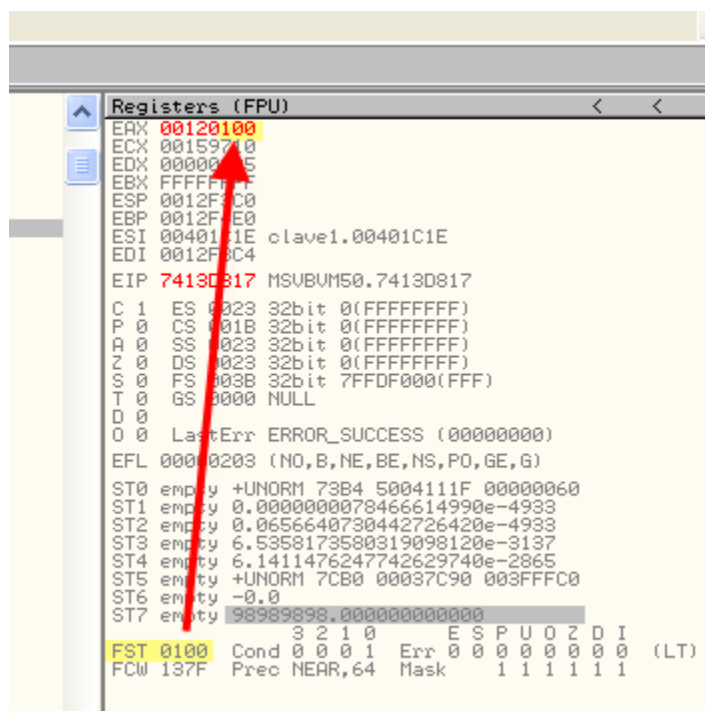
Veo que efectivamente es mi serial falso, solo que cambio de forma de representación al abarcar 8 bytes o sea doble de 64- bit, lógico de 4 bytes es 32 -bit, ahora es doble o sea un numero de 64-bit, por eso se ve diferente.



Volvamos a la visión normal

7413D814	50	PUSH EAX
7413D815	DF00	FSTSW AX
7413D817	A8 00	TEST AL,00
7413D819	0F85 87C90000	JNZ MSUBUM50.7414A1A6

Eso guarda el STATUS WORD de punto flotante a AX, ejecutemos



7413D819	0F85 87C90000	JNZ MSUBUM50.7414A1A6
7413D81F	33C0	XOR EAX,EAX
7413D821	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D824	83C6 03	ADD ESI,3
7413D827	FF2485 24ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D82E	8B0424	MOV EAX,DWORD PTR SS:[ESP]
7413D831	66:8108 0080	OR WORD PTR DS:[EAX],0000
7413D836	33C0	XOR EAX,EAX
7413D838	8A06	MOV AL,BYTE PTR DS:[ESI]
7413D83A	46	INC ESI

Allí termina el OPCODE

401C20: 5d HardType

Bueno este no tengo la más mínima idea de lo que es pero lo descubriremos

7413D827	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D82E	8B0424	MOV EAX,DWORD PTR SS:[ESP]
7413D831	66:8108 0000	OR WORD PTR DS:[EAX],8000
7413D836	33C0	XOR EAX,EAX
7413D838	8A06	MOV AL,BYTE PTR DS:[ESI]
7413D83A	46	INC ESI
7413D83B	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D842	0F0F0F	MOVSX EDI,WORD PTR DS:[ESI]

Bueno son solo dos líneas mueve el contenido de ESP a EAX

Registers (FPU)	
EAX	0012F434
ECX	00159710
EDX	00000005
EBX	FFFFFFFF
ESP	0012F3C0
EBP	0012F4E0
ESI	00401C21 cla
EDI	0012F3C4
EIP	7413D831 MSU
C 0	ES 0023 32b
P 1	CS 001B 32b
A 1	SS 0023 32b
Z 0	DS 0023 32b
S 0	FS 003B 32b

Address	Hex dump	ASCII
0012F434	05 00 00 00 00 00 00 00
0012F43C	00 00 00 28 DD 99 97 41	...(!00A
0012F444	03 00 00 00 00 00 00 00

Vemos que esta agregando al número que esta arriba de mi serial falso transformado, posiblemente ese número indique el formato en que se encuentra, y ahora lo ajusta.

Address	Hex dump	ASCII
0012F434	05 80 00 00 00 00 00 00
0012F43C	00 00 00 28 DD 99 97 41	...(!00A
0012F444	03 00 00 00 00 00 00 00
0012F44C	10 C4 00 00 00 00 00 00

Siguiente OPCODE

401C21: 04 FLdRfVar local_009C

Ese es el inicial o sea PUSH ebp-9c en este caso

Command ? ebp- 9c HEX: 12F444	
0012F3BC	0012F444
0012F3C0	0012F434
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000

Allí esta

El siguiente es un opcode doble

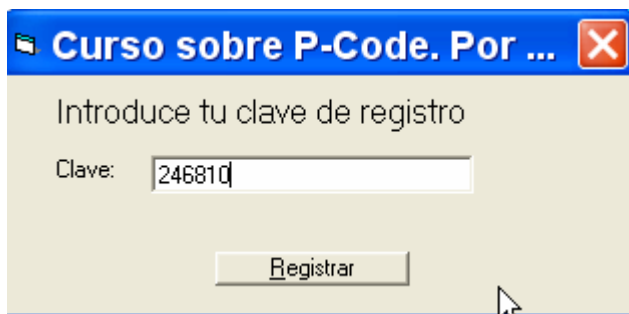
401C24: Lead0/40 NeVarBool

0012F3C0	FFFFFFFF
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000

Y que hace un PUSH al stack con el valor FFFFFFFF, es casi seguro que la comparación es aquí, así que reinicio el crackme total puse un BP, y pongo el serial correcto que obtuve de esta comparación

Address	Hex dump	ASCII
0012F444	03 00 00 00 00 00 00 00
0012F44C	1A C4 03 00 00 00 00 00
0012F454	9C D3 15 00 A4 BC CF 00	ES. "R.
0012F45C	EC F4 12 00 C8 F5 12 00	gT*. "S*.

03c41a pasado a decimal es 246810, pongámoslo y lleguemos a la comparación de nuevo



Al parar veo que compara

Address	64-bit double
0012F434	1.619201341115517e-319 246810.0000000000
0012F444	1.482196937523740e-323 1.219403420500781e-318
0012F454	9.038962733674010e-305 2.636724066601938e-308
0012F464	2.636248745657113e-308 1.743007342698825e-310

246810 con el valor hexa del mismo, la verdad que son diferentes formatos pero como arriba de cada uno hay un numero que identifica el formato es posible que dentro del mismo call los transforme y los compare, veamos que nos da ahora a la salida

Registers (FPU)	
EAX	00000000
ECX	00000003
EDX	0012F444
EBX	7413ED74 MS
ESP	0012F3C4
EBP	0012F4E0
ESI	00401C26 cl
EDI	0012F3C4
EIP	7413EC06 MS
C 0	ES 0023 32

Vemos que ahora nos da EAX=0 (antes daba 1)

0012F3C0	00000000
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000
0012F3D0	00000000

Y el push al stack también da cero

Lo cual quiere decir que esta es la comparación, además vemos justo abajo unos Branch, que son los saltos que decidirán si ir al cartel correcto o incorrecto, luego de liberar las strings y valores usados

401C21: 04 FLdRfVar	local_009C
401C24: Lead0/40 NeVarBool	
401C26: 2f FFree1Str	local_008C
401C29: 1a FFree1Ad	local_0088
401C2C: 1c BranchF:	401C63
401C2F: 27 LitVar_Missing	
401C32: 27 LitVar_Missing	

Ustedes dirán esto es mucho trabajo, pero si se dan cuenta el trabajo fuerte, es la primera vez que usas cada opcode pues una vez que descubriste que hace ya no necesitas repetirlo, por lo demás el WKT debugger te muestra el nombre de los OPCODES, pero no sabes que hace cada uno, salvo los mas conocidos y con OLLYDBG , podemos determinar que hace cada uno y tener una mejor idea de cómo funciona el programa.

En la próxima parte haré con el mismo método el crackme adjunto clave 2, me gustaría que siguiendo el mismo método hallaran la clave si pueden de ese clave2, si no, en la próxima parte lo haré yo, y lo podrán leer.

UFFFFFFFFF

Hasta la 30

Ricardo Narvaja