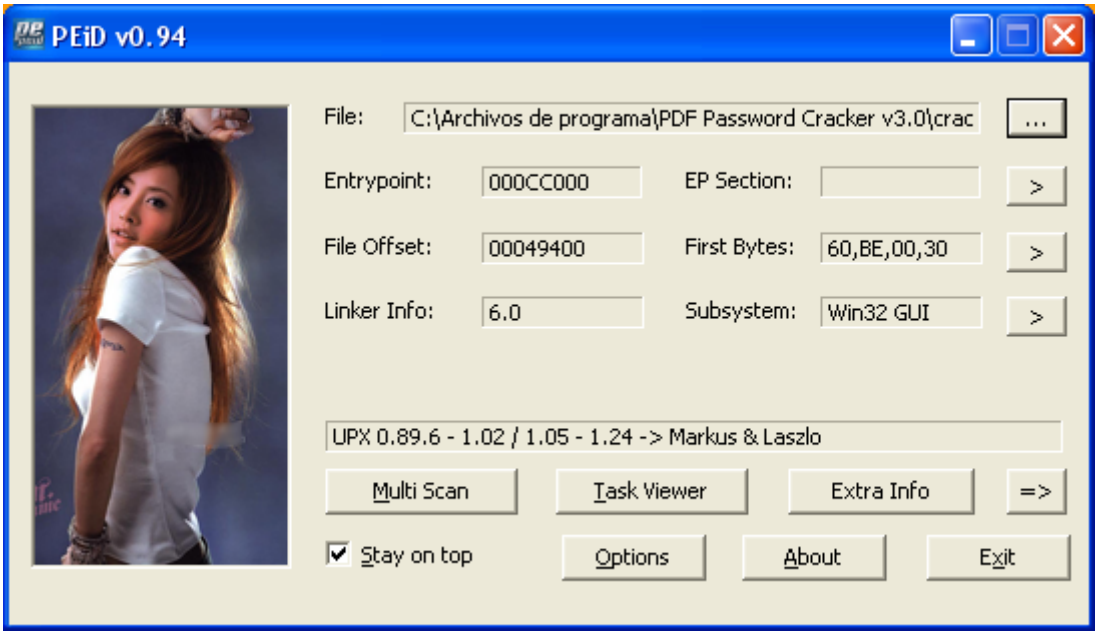
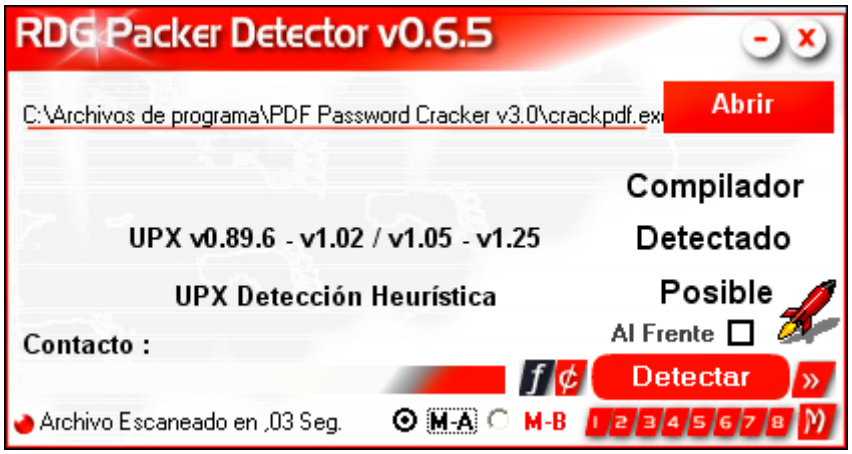
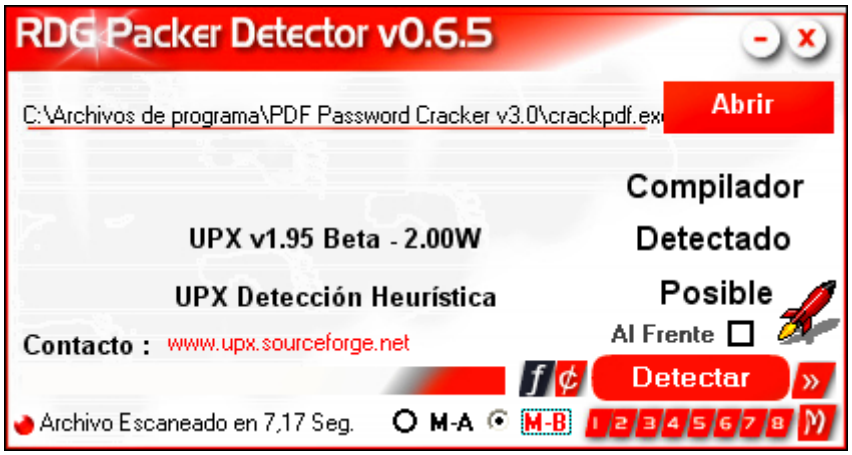
Autor	Aguml
Fecha	01-11-2008
Victima	PDF Password Cracker v3.0
Protección	UPX + Redirección IAT + Serial
Herramientas	Ollydbg + Import Rec
Objetivo	Desempacar y registrar
Dificultad	Muy Sencillo

Lo primero que hago es mirar con que está protegido:



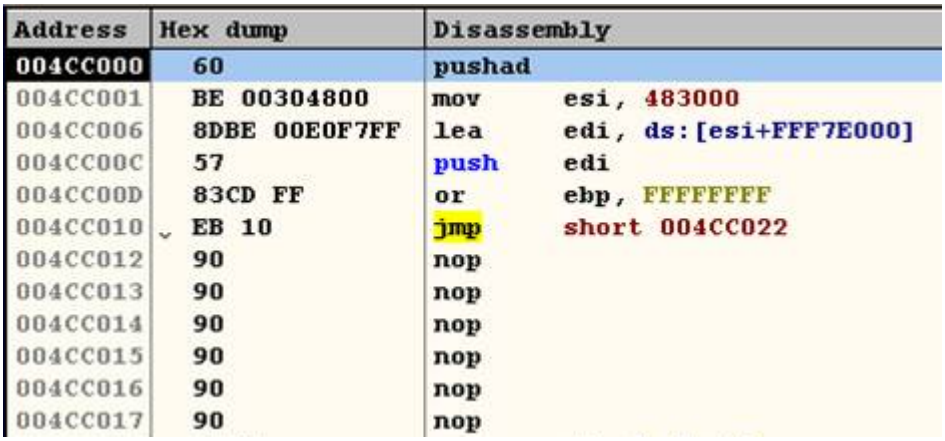




Vemos que ambos coinciden aunque en el modo M-B el RDG me dice que es otra versión pero parece que está claro que es UPX.
Abrimos el programa en olly y lo arrancamos con F9 y nos saldrá una ventana para registrarnos:

Metamos un serial cualquiera y probemos a ver si acertamos:

Lo normal, si hubiera acertado sería prácticamente milagroso jajaja. Bueno, por lo menos nos muestra una ventanita de chico malo así que atacaré por ahí. Así que vayamos hasta el OEP y dumpeemos y reparemos la IAT para que corra.
Lo primero será llegar al OEP así que reiniciamos la aplicación en olly y aparecemos aquí:

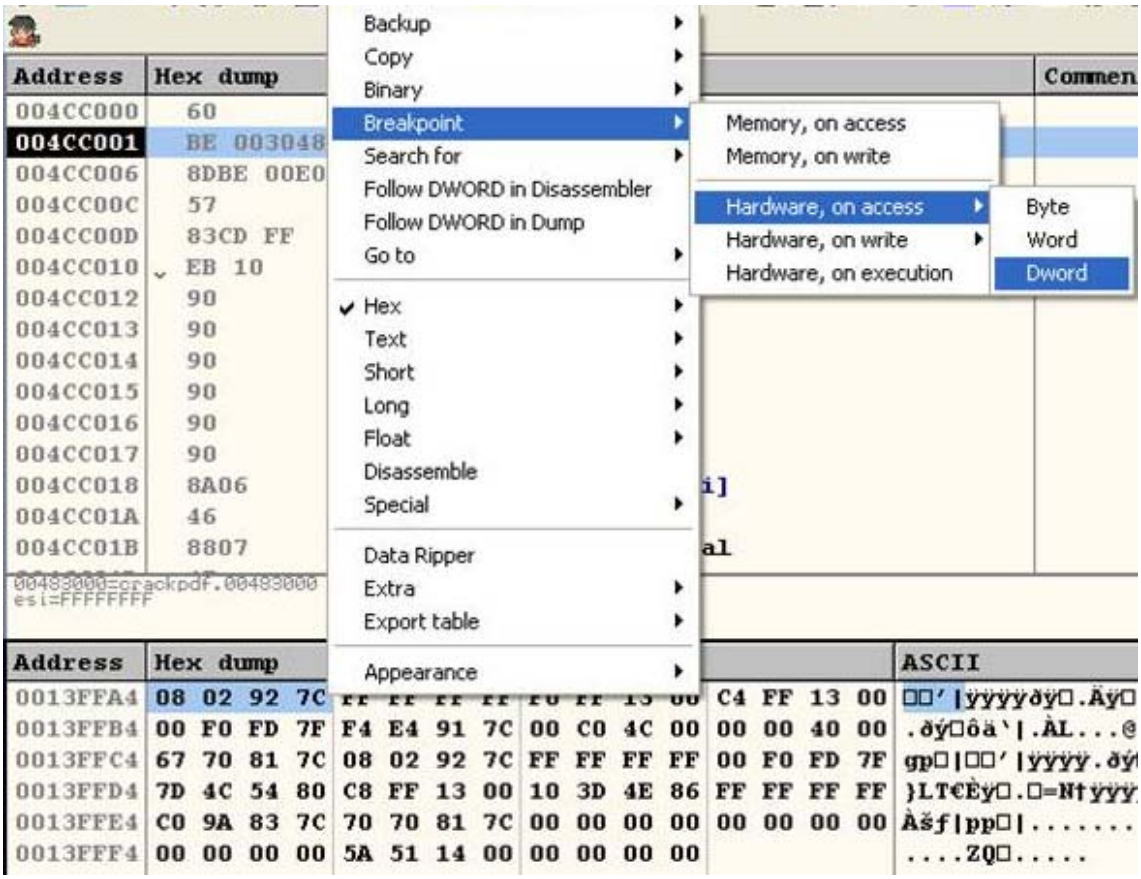


Address	Hex dump	Disassembly
004CC000	60	pushad
004CC001	BE 00304800	mov esi, 483000
004CC006	8DBE 00E0F7FF	lea edi, ds:[esi+FFF7E000]
004CC00C	57	push edi
004CC00D	83CD FF	or ebp, FFFFFFFF
004CC010	EB 10	jmp short 004CC022
004CC012	90	nop
004CC013	90	nop
004CC014	90	nop
004CC015	90	nop
004CC016	90	nop
004CC017	90	nop

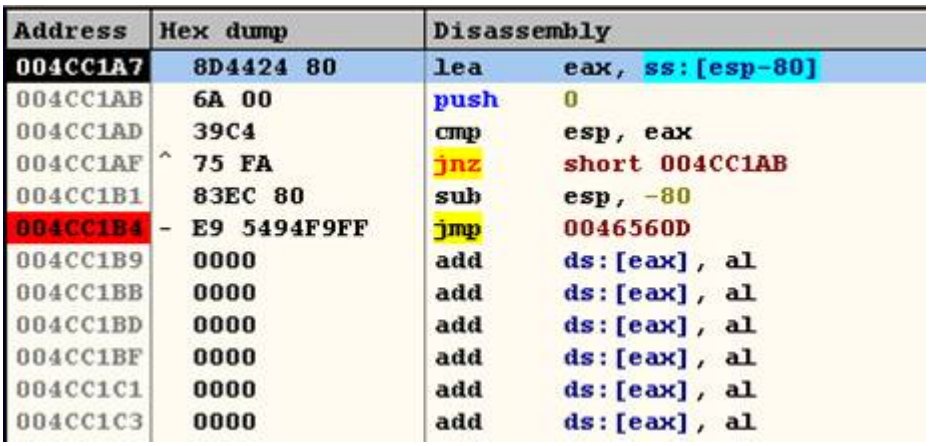
Pulsamos F7 para pasar el PUSHAD y ahora nos vamos a la Pila para ver donde introdujo el último valor de registro:

Address	Value	Comment
0013FFA4	7C920208	ntdll.7C920208
0013FFA8	FFFFFFFF	
0013FFAC	0013FFF0	
0013FFB0	0013FFC4	
0013FFB4	7FFDF000	
0013FFB8	7C91E4F4	ntdll.KiFastSystemCallRet
0013FFBC	004CC000	crackpdf.<ModuleEntryPoint>
0013FFC0	00400000	crackpdf.00400000
0013FFC4	7C817067	RETURN to kernel32.7C817067
0013FFC8	7C920208	ntdll.7C920208
0013FFCC	FFFFFFFF	
0013FFD0	7FFDF000	
0013FFD4	80544C7D	
0013FFD8	0013FFC8	
0013FFDC	864E3D10	
0013FFE0	FFFFFFFF	End of SEH chain
0013FFE4	7C839AC0	SE handler
0013FFE8	7C817070	kernel32.7C817070
0013FFEC	00000000	
0013FFF0	00000000	
0013FFF4	00000000	
0013FFF8	0014515A	
0013FFFC	00000000	

Vemos que el último que añadió esta en 0013FFA4 así que vayamos a esa dirección en el Dump y pongamos un HBP en él:



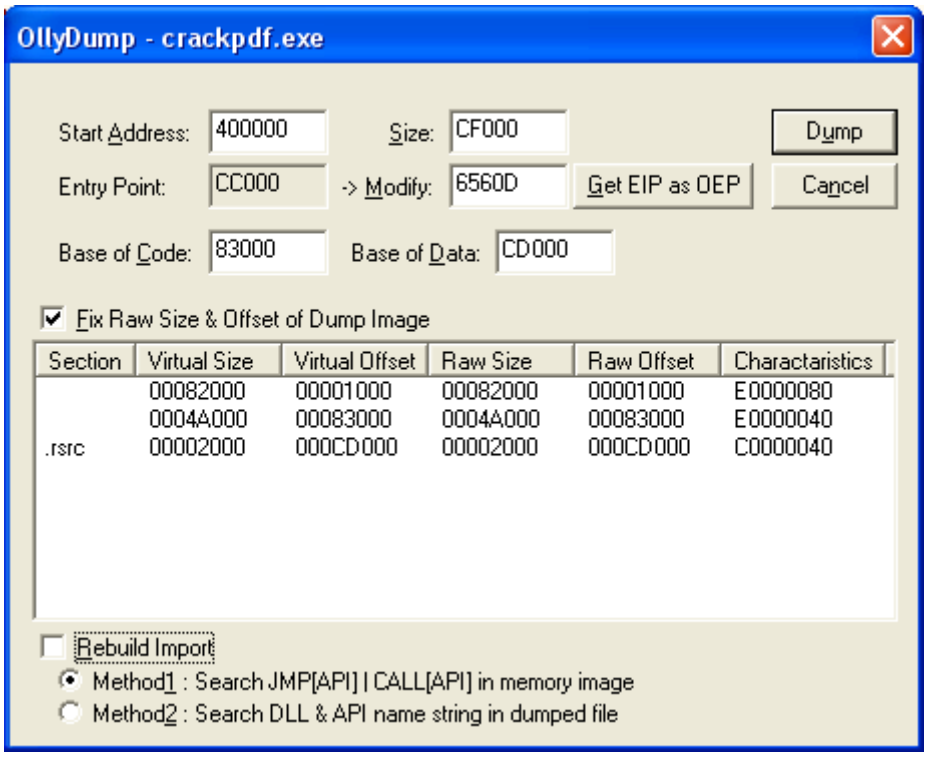
Ahora damos a F9 y vemos que para en:



Pues pongamos un BP en el JMP como se ve en la imagen y demos a F9 y cuando pare en el BP damos a F7 para pasarlo y llegamos aquí:

Address	Hex dump	Disassembly	Comment
0046560D	55	push ebp	
0046560E	8BEC	mov ebp, esp	
00465610	6A FF	push -1	
00465612	68 E8BC4800	push 48BCE8	
00465617	68 4C9D4600	push 469D4C	SE handler installation
0046561C	64:A1 00000000	mov eax, fs:[0]	
00465622	50	push eax	
00465623	64:8925 000000(	mov fs:[0], esp	
0046562A	83EC 58	sub esp, 58	
0046562D	53	push ebx	
0046562E	56	push esi	
0046562F	57	push edi	
00465630	8965 E8	mov ss:[ebp-18], esp	
00465633	FF15 38824800	call ds:[488238]	kernel32.GetVersion
00465639	33D2	xor edx, edx	
0046563B	8AD4	mov dl, ah	

Esto tiene toda la pinta de ser un OEP correcto así que lo doy por bueno y seguimos. Ahora lo siguiente es dumpearlo y para eso usamos por ejemplo OllyDump.



Ahora toca reparar la IAT y para ello necesitamos saber donde empieza y donde acaba esta. Para localizar la IAT nos colocamos encima de la siguiente CALL, ya que vemos que apunta a una API, y hacemos lo siguiente:

Aparecemos aquí:

Address	Hex dump															
00488238	6A	12	81	7C	B8	97	80	7C	93	C1	85	7C	31	BB	80	7C
00488248	C1	60	83	7C	DB	60	83	7C	AB	0B	83	7C	31	B7	80	7C
00488258	67	EE	80	7C	69	38	81	7C	CC	15	81	7C	8C	39	81	7C
00488268	16	50	83	7C	00	10	91	7C	E0	10	91	7C	30	AE	80	7C
00488278	5A	13	92	7C	81	9F	80	7C	6E	2B	81	7C	27	D8	81	7C
00488288	6B	23	80	7C	17	08	86	7C	12	18	80	7C	30	25	80	7C
00488298	46	24	80	7C	D7	9B	80	7C	7C	AC	85	7C	FA	CA	81	7C
004882A8	8D	1B	82	7C	7A	4F	81	7C	6E	2B	83	7C	6C	5D	83	7C
004882B8	5F	B5	80	7C	59	4D	83	7C	46	BE	80	7C	AD	23	86	7C

Subimos hasta encontrar el principio de la IAT y lo encontramos en 00487FFC:

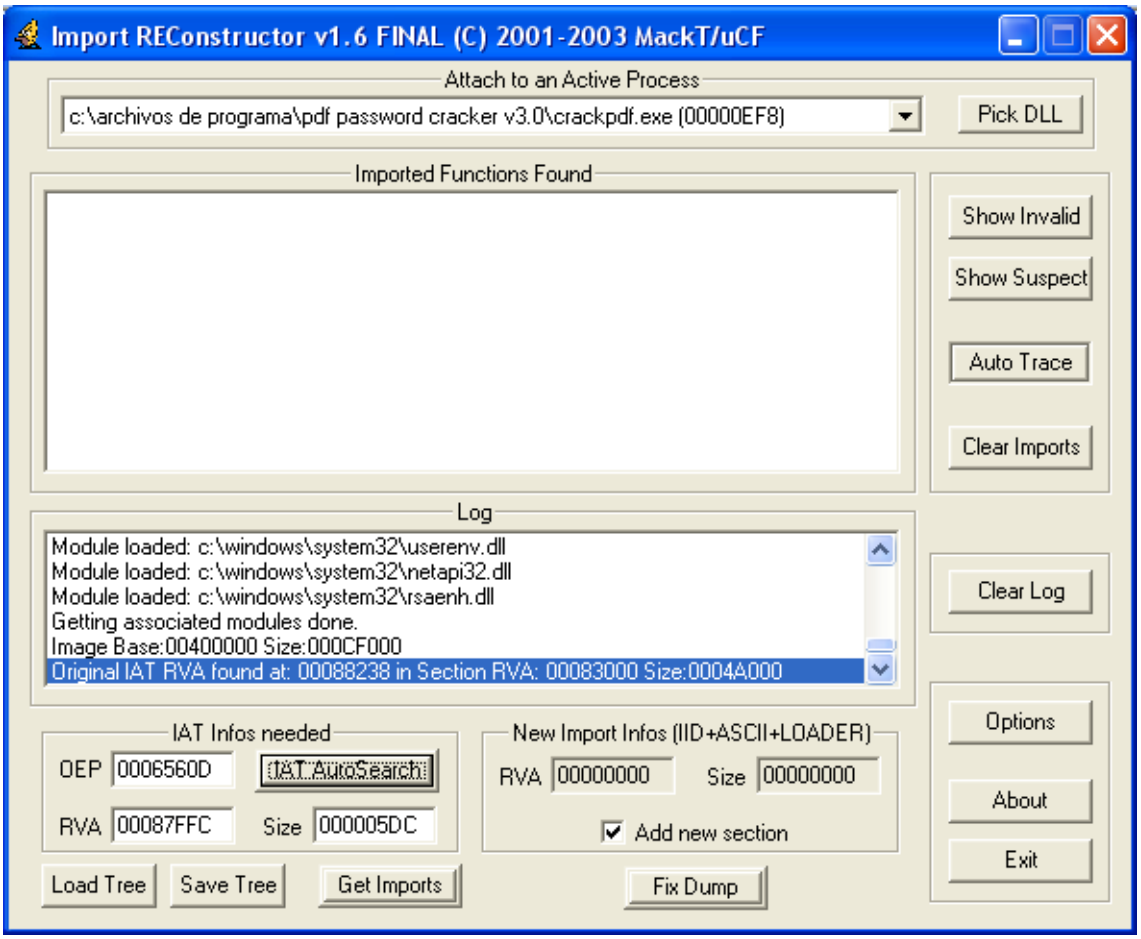
Address	Hex dump															
00487FE8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00487FF8	00	00	00	00	00	00	00	00	5D	BB	DC	77	42	78	DA	77
00488008	AB	7A	DA	77	E4	E9	DA	77	D7	EA	DA	77	D5	EC	DA	77
00488018	80	42	DB	77	17	6C	DA	77	00	00	00	00	F4	C7	C3	58
00488028	CF	65	C3	58	D8	03	C4	58	05	02	C4	58	00	00	00	00
00488038	AE	3A	F0	77	0E	D3	F1	77	05	3A	F0	77	EF	D3	F1	77
00488048	56	6A	EF	77	FA	6B	EF	77	06	C0	EF	77	F1	7C	EF	77
00488058	79	7C	EF	77	A5	61	EF	77	37	65	F2	77	1B	82	EF	77
00488068	3F	BA	EF	77	EA	D3	EF	77	7C	77	F0	77	9B	86	EF	77

Y ahora vamos para abajo hasta encontrar el final que lo vemos en 004885D8:

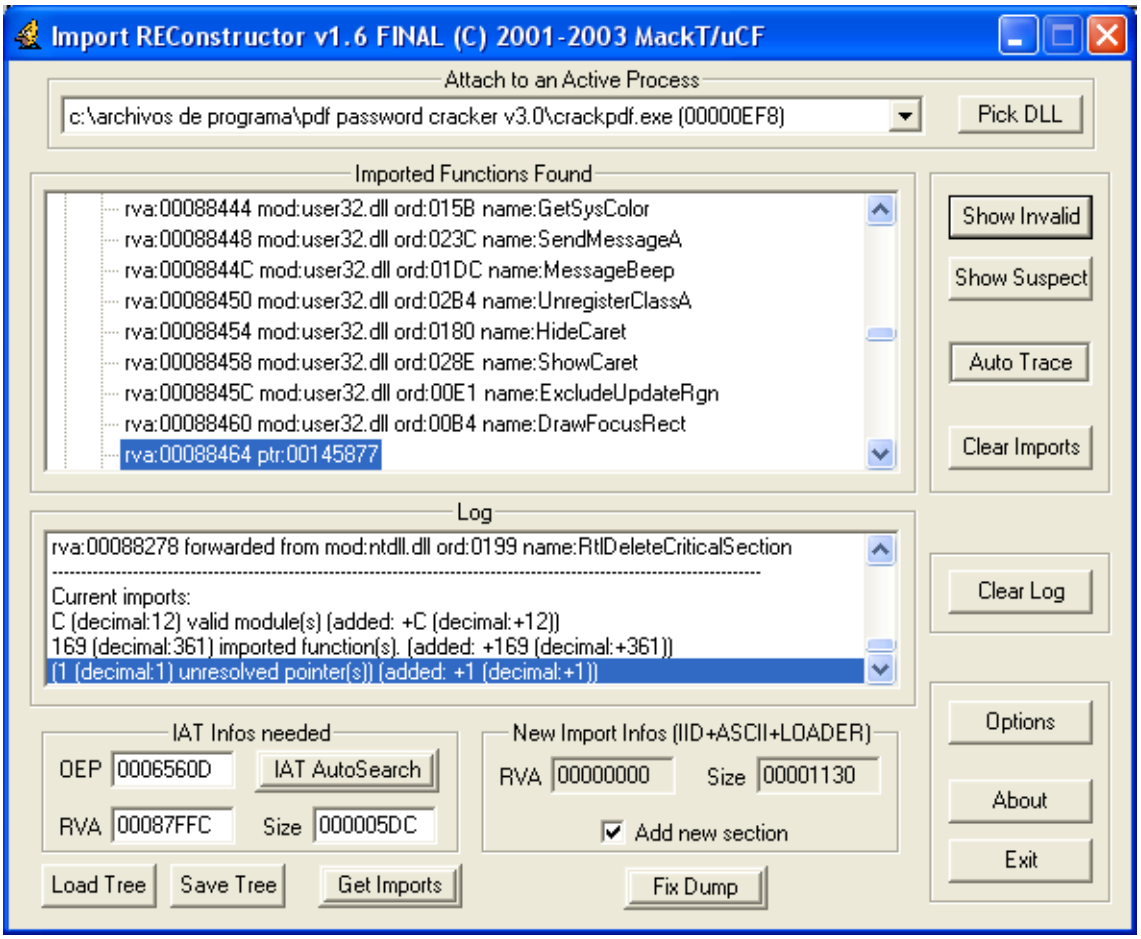
Address	Hex dump															
00488558	7A	97	3A	7E	9E	B2	3A	7E	89	C6	3B	7E	B2	DE	3A	7E
00488568	56	AF	3A	7E	00	00	00	00	40	4D	F8	72	5F	66	F9	72
00488578	57	37	F9	72	00	00	00	00	10	7C	37	76	9F	30	36	76
00488588	63	25	36	76	00	00	00	00	AC	00	50	77	EA	F6	4C	77
00488598	60	D0	4C	77	44	D0	4C	77	2A	56	51	77	5A	57	51	77
004885A8	1D	C8	5A	77	C5	56	4E	77	B6	FC	4F	77	F2	87	4E	77
004885B8	1F	5F	52	77	F3	A2	4F	77	71	AB	54	77	C1	A9	54	77
004885C8	E7	31	50	77	00	00	00	00	6A	09	1F	7E	00	00	00	00
004885D8	00	00	00	00	00	00	00	00	08	97	48	00	E8	85	48	00

Final IAT – Inicio IAT = Tamaño IAT
Tamaño IAT = 004885D8 - 00487FFC = 5DC

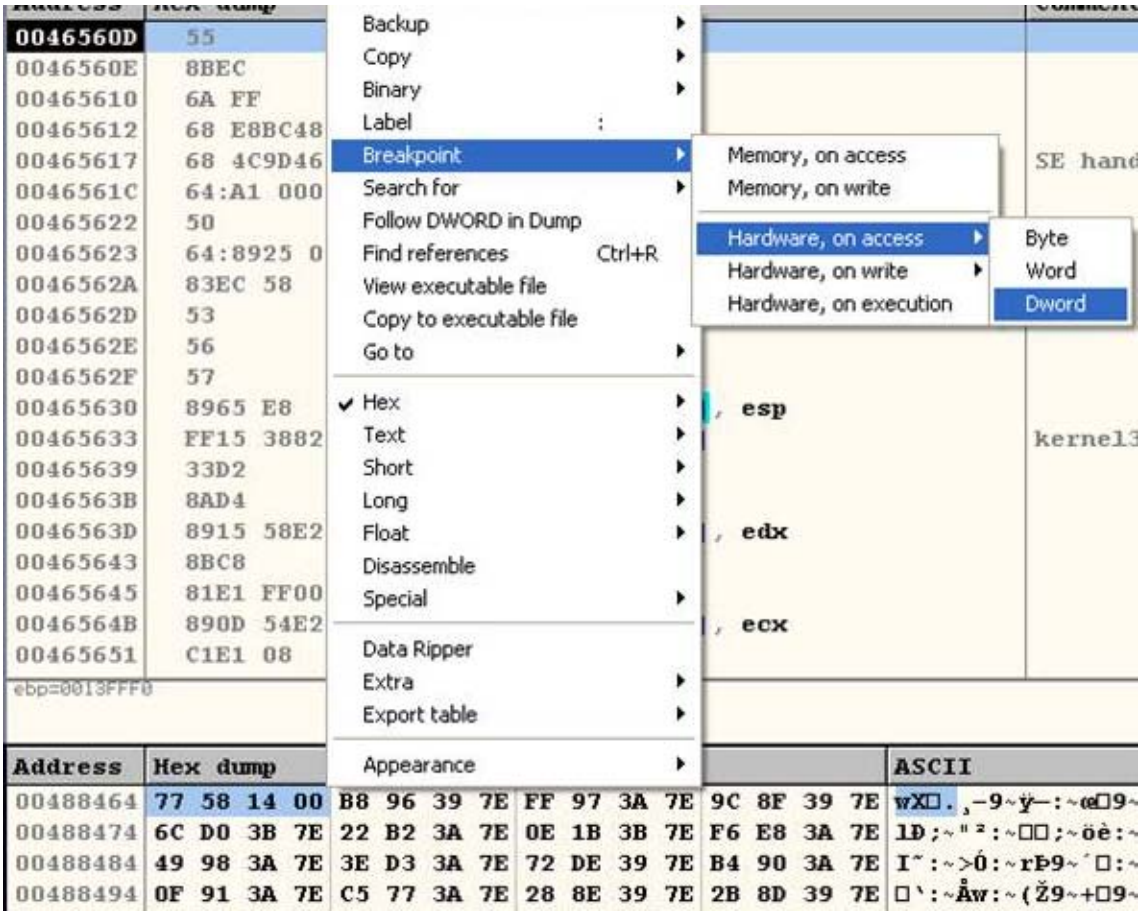
Bueno pues con estos valores y el OEP ya podemos intentar reparar la IAT así que abrimos Import Rec y seleccionamos a la víctima en la lista de procesos:



Como veis ya tiene puesto los valores necesarios que son el OEP (menos la ImageBase, el RVA (es el inicio de la IAT menos la ImageBase), y el Size (es el tamaño de la IAT).
Ahora hacemos clic en Get Imports y a continuación en Show Invalid, ya que en el log me está avisando de que existe una entrada mala en la IAT:



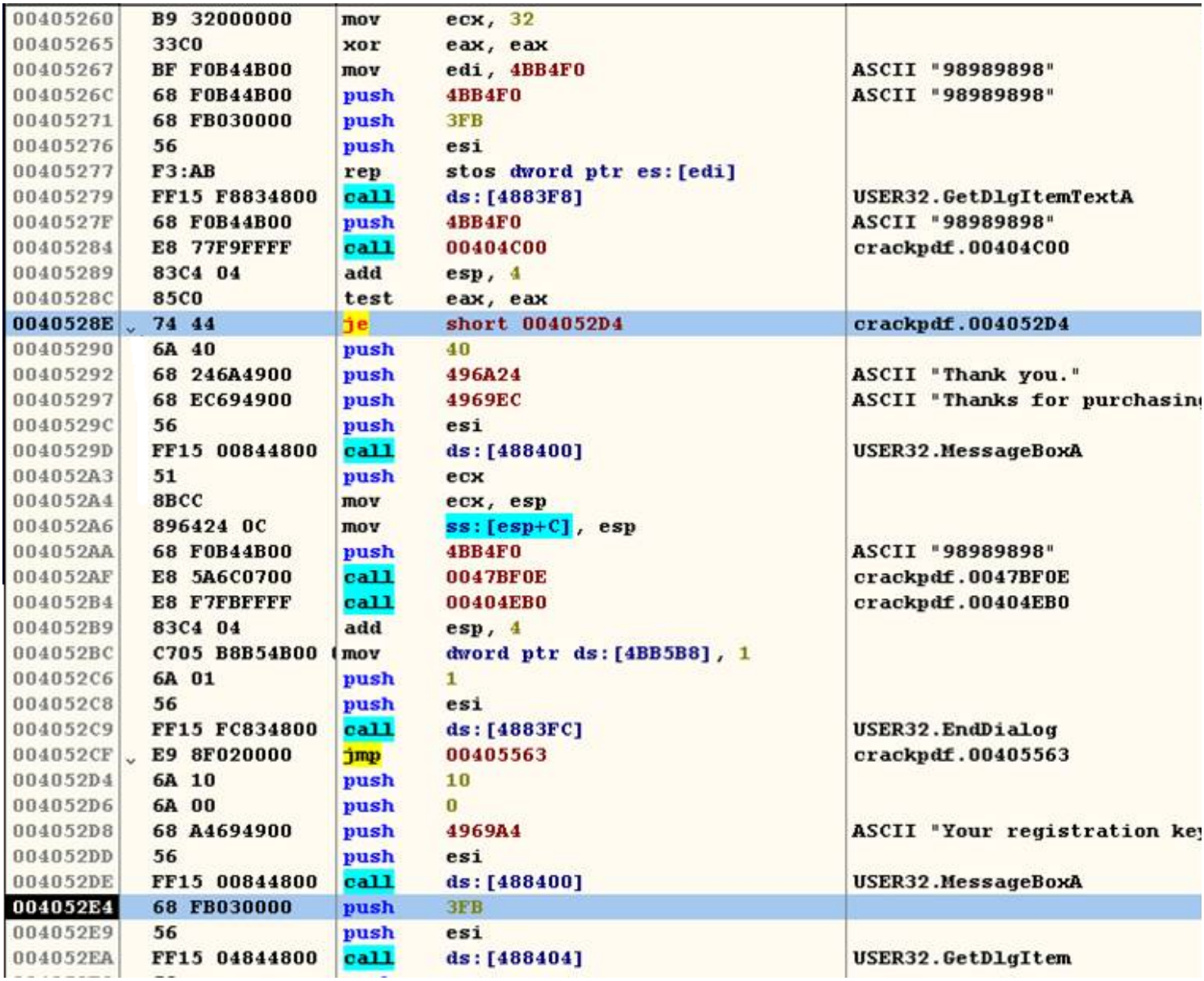
En la imagen vemos que la entrada mala está en 00088464 así que sumamos la ImageBase y nos da 00488464. Pues vayamos a esa dirección en el Dump para poner un HBP:



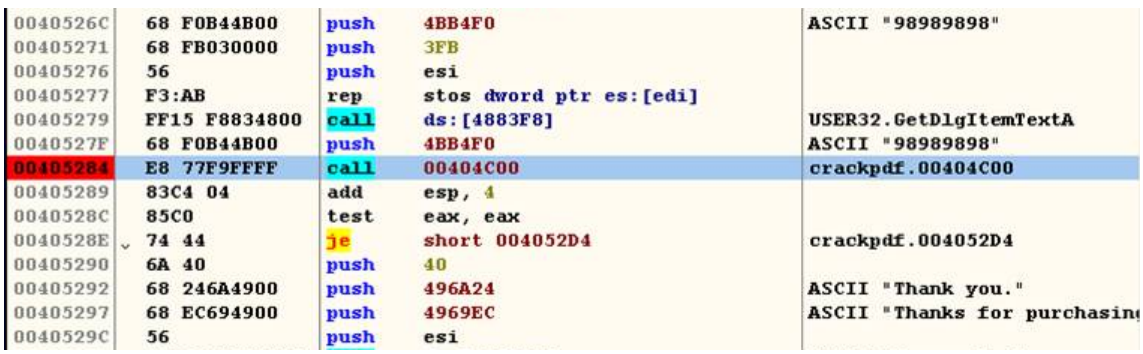
Damos a F9 y nos sale la ventana para registrarnos y no para así que tendremos que registrarlo para ver cuando usa esa dirección. Pues nada, reiniciamos la aplicación de nuevo y volvemos a llegar hasta el OEP verdadero. Lo primero que se me ocurre siempre es usar un BP MessageBoxA así que probemos a ponerlo desde la CommandBar y voy al lugar donde se puso y lo quito y lo coloco en el RETN de la API.
Doy a F9 y nos sale la ventanita que nos dice que el serial ese no vale así que aceptamos y…

```
7E3D07ED   8BEC             mov     ebp, esp
7E3D07EF   833D BC143F7E (  cmp     dword ptr ds:[7E3F14BC], 0
7E3D07F6   74 24            je      short 7E3D081C                  USER32.7E3D081C
7E3D07F8   64:A1 18000000   mov     eax, fs:[18]
7E3D07FE   6A 00            push    0
7E3D0800   FF70 24          push    dword ptr ds:[eax+24]
7E3D0803   68 241B3F7E      push    7E3F1B24
7E3D0808   FF15 C412397E    call    ds:[7E3912C4]                   kernel32.InterlockedCompareE
7E3D080E   85C0             test    eax, eax
7E3D0810   75 0A            jnz     short 7E3D081C                  USER32.7E3D081C
7E3D0812   C705 201B3F7E (  mov     dword ptr ds:[7E3F1B20], 1
7E3D081C   6A 00            push    0
7E3D081E   FF75 14          push    dword ptr ss:[ebp+14]
7E3D0821   FF75 10          push    dword ptr ss:[ebp+10]
7E3D0824   FF75 0C          push    dword ptr ss:[ebp+C]
7E3D0827   FF75 08          push    dword ptr ss:[ebp+8]
7E3D082A   E8 2D000000      call    7E3D085C                        USER32.MessageBoxExA
7E3D082F   5D               pop     ebp
7E3D0830   C2 1000          retn    10
7E3D0833   90               nop
7E3D0834   90               nop
```

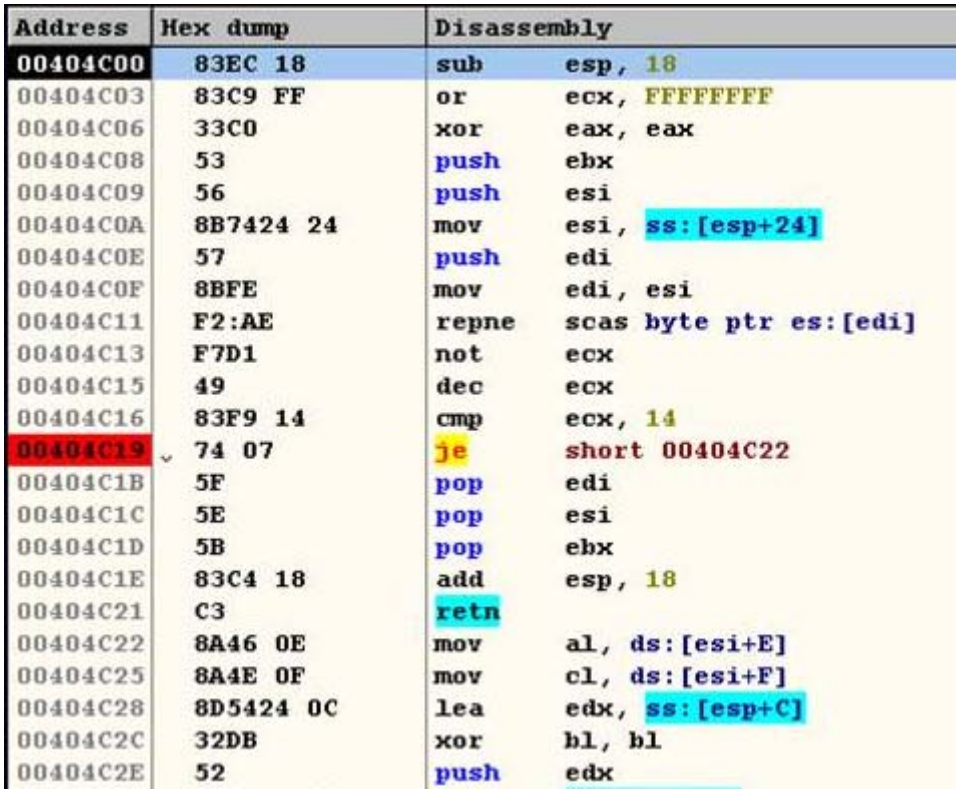
Vemos como ha parado donde queríamos así que demos a F7 para salir de la API y llegamos aquí:



Jojojo, vemos el MessageBox del chico malo y también el del chico bueno y justo encima del mensaje de chico bueno hay un salto condicional que si se cumple nos tira a la zona del chico malo y un poquito mas arriba hay una CALL que es la encargada de verificar el serial casi seguro así que pongamos un BP en esa CALL y demos a F9 para continuar y acto seguido intentamos registrarnos de nuevo para que pare en dicha CALL:

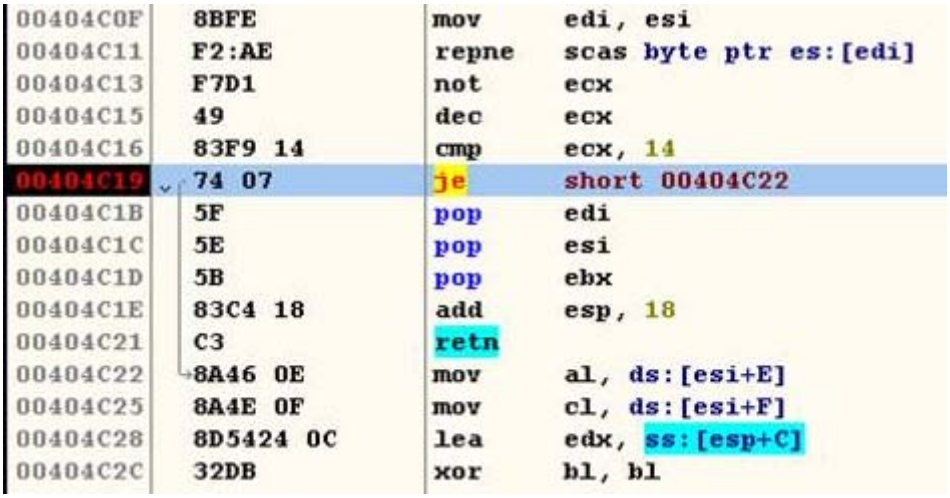


Una vez que pare en la CALL, damos a F7 y llegamos aquí:



Address	Hex dump	Disassembly
00404C00	83EC 18	sub esp, 18
00404C03	83C9 FF	or ecx, FFFFFFFF
00404C06	33C0	xor eax, eax
00404C08	53	push ebx
00404C09	56	push esi
00404C0A	8B7424 24	mov esi, ss:[esp+24]
00404C0E	57	push edi
00404C0F	8BFE	mov edi, esi
00404C11	F2:AE	repne scas byte ptr es:[edi]
00404C13	F7D1	not ecx
00404C15	49	dec ecx
00404C16	83F9 14	cmp ecx, 14
00404C19	74 07	je short 00404C22
00404C1B	5F	pop edi
00404C1C	5E	pop esi
00404C1D	5B	pop ebx
00404C1E	83C4 18	add esp, 18
00404C21	C3	retn
00404C22	8A46 0E	mov al, ds:[esi+E]
00404C25	8A4E 0F	mov cl, ds:[esi+F]
00404C28	8D5424 0C	lea edx, ss:[esp+C]
00404C2C	32DB	xor bl, bl
00404C2E	52	push edx

Si vamos traceando veremos que lo que hace es contar el largo de nuestro serial y justo antes del salto lo compara con 14 hexadecimal y como no es igual pues no saltara y nos sacara de la CALL y EAX valdrá 0 con lo cual nos llevará a la zona de chico malo así que tendremos que hacerlo saltar siempre:



00404C0F	8BFE	mov edi, esi
00404C11	F2:AE	repne scas byte ptr es:[edi]
00404C13	F7D1	not ecx
00404C15	49	dec ecx
00404C16	83F9 14	cmp ecx, 14
00404C19	74 07	je short 00404C22
00404C1B	5F	pop edi
00404C1C	5E	pop esi
00404C1D	5B	pop ebx
00404C1E	83C4 18	add esp, 18
00404C21	C3	retn
00404C22	8A46 0E	mov al, ds:[esi+E]
00404C25	8A4E 0F	mov cl, ds:[esi+F]
00404C28	8D5424 0C	lea edx, ss:[esp+C]
00404C2C	32DB	xor bl, bl

```
00404C0F   8BFE          mov     edi, esi
00404C11   F2:AE         repne   scas byte ptr es:[edi]
00404C13   F7D1          not     ecx
00404C15   49            dec     ecx
00404C16   83F9 14       cmp     ecx, 14
00404C19   EB 07         jmp     short 00404C22
00404C1B   5F            pop     edi
00404C1C   5E            pop     esi
00404C1D   5B            pop     ebx
00404C1E   83C4 18       add     esp, 18
00404C21   C3            retn
00404C22   8A46 0E       mov     al, ds:[esi+E]
00404C25   8A4E 0F       mov     cl, ds:[esi+F]
00404C28   8D5424 0C     lea     edx, ss:[esp+C]
00404C2C   32DB          xor     bl, bl
```

Seguimos traceando y llegamos a otro salto y si nos fijamos si no se cumple el salto xoreara a EAX contra si misma así que la pondrá a 0 y eso no nos interesa así que tiene que saltar aquí también siempre:

```
00404C44   8BF8          mov     edi, eax
00404C46   8D4424 1C     lea     eax, ss:[esp+1C]
00404C4A   50            push    eax
00404C4B   E8 60090600   call    004655B0
00404C50   03F8          add     edi, eax
00404C52   83C4 08       add     esp, 8
00404C55   83FF 0B       cmp     edi, 0B
00404C58   74 09         je      short 00404C63
00404C5A   5F            pop     edi
00404C5B   5E            pop     esi
00404C5C   33C0          xor     eax, eax
00404C5E   5B            pop     ebx
00404C5F   83C4 18       add     esp, 18
00404C62   C3            retn
00404C63   8A0E          mov     cl, ds:[esi]
00404C65   8A56 01       mov     dl, ds:[esi+1]
00404C68   8D4424 0C     lea     eax, ss:[esp+C]
```

```
00404C44   8BF8          mov     edi, eax
00404C46   8D4424 1C     lea     eax, ss:[esp+1C]
00404C4A   50            push    eax
00404C4B   E8 60090600   call    004655B0
00404C50   03F8          add     edi, eax
00404C52   83C4 08       add     esp, 8
00404C55   83FF 0B       cmp     edi, 0B
00404C58   EB 09         jmp     short 00404C63
00404C5A   5F            pop     edi
00404C5B   5E            pop     esi
00404C5C   33C0          xor     eax, eax
00404C5E   5B            pop     ebx
00404C5F   83C4 18       add     esp, 18
00404C62   C3            retn
00404C63   8A0E          mov     cl, ds:[esi]
00404C65   8A56 01       mov     dl, ds:[esi+1]
00404C68   8D4424 0C     lea     eax, ss:[esp+C]
```

Seguimos traceando y vemos que la historia se repite de nuevo así que aquí también tendrá que saltar siempre:

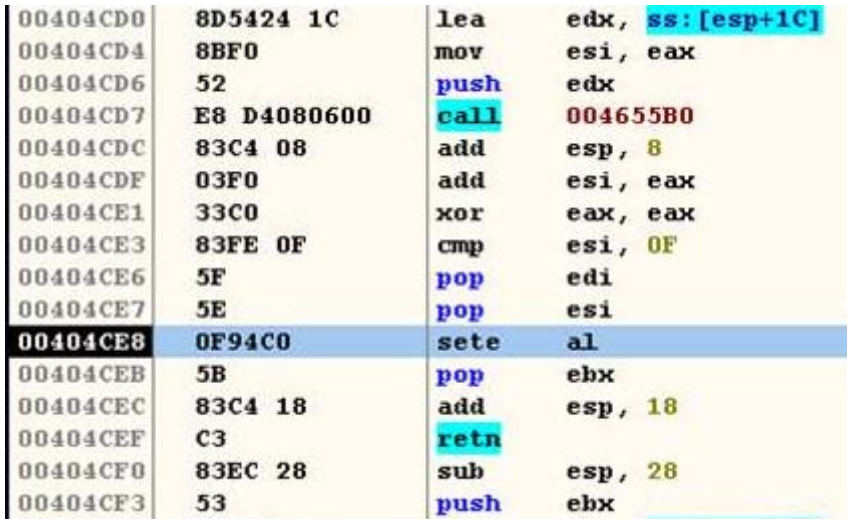
```
00404C86   8BF8            mov     edi, eax
00404C88   51              push    ecx
00404C89   E8 22090600     call    004655B0
00404C8E   03F8            add     edi, eax
00404C90   83C4 08         add     esp, 8
00404C93   83FF 0A         cmp     edi, 0A
00404C96   74 09           je      short 00404CA1
00404C98   5F              pop     edi
00404C99   5E              pop     esi
00404C9A   33C0            xor     eax, eax
00404C9C   5B              pop     ebx
00404C9D   83C4 18         add     esp, 18
00404CA0   C3              retn
00404CA1   807E 05 33      cmp     byte ptr ds:[esi+5], 33
00404CA5   74 09           je      short 00404CB0
00404CA7   5F              pop     edi
00404CA8   5E              pop     esi
```

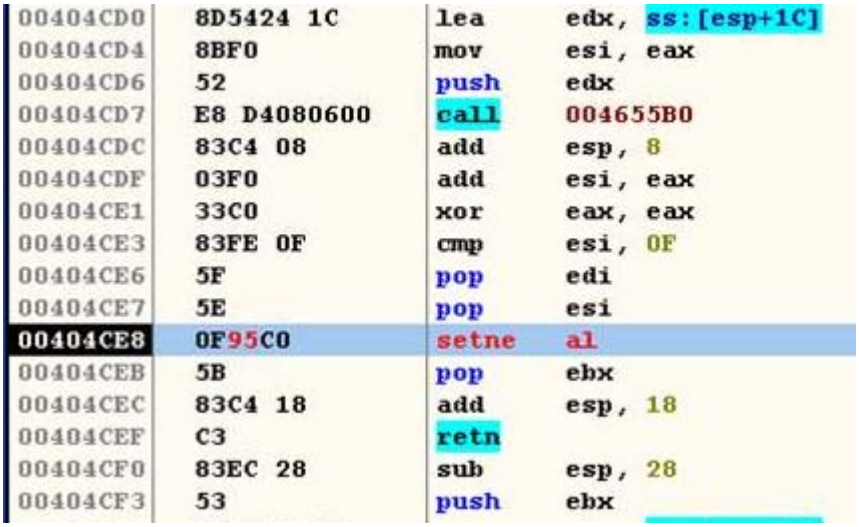
```
00404C86   8BF8            mov     edi, eax
00404C88   51              push    ecx
00404C89   E8 22090600     call    004655B0
00404C8E   03F8            add     edi, eax
00404C90   83C4 08         add     esp, 8
00404C93   83FF 0A         cmp     edi, 0A
00404C96   EB 09           jmp     short 00404CA1
00404C98   5F              pop     edi
00404C99   5E              pop     esi
00404C9A   33C0            xor     eax, eax
00404C9C   5B              pop     ebx
00404C9D   83C4 18         add     esp, 18
00404CA0   C3              retn
00404CA1   807E 05 33      cmp     byte ptr ds:[esi+5], 33
00404CA5   74 09           je      short 00404CB0
00404CA7   5F              pop     edi
00404CA8   5E              pop     esi
```

Justo después tenemos otro salto y lo mismo así que hay que tratarlo igual:

```
00404C93   83FF 0A         cmp     edi, 0A
00404C96   EB 09           jmp     short 00404CA1
00404C98   5F              pop     edi
00404C99   5E              pop     esi
00404C9A   33C0            xor     eax, eax
00404C9C   5B              pop     ebx
00404C9D   83C4 18         add     esp, 18
00404CA0   C3              retn
00404CA1   807E 05 33      cmp     byte ptr ds:[esi+5], 33
00404CA5   EB 09           jmp     short 00404CB0
00404CA7   5F              pop     edi
00404CA8   5E              pop     esi
00404CA9   33C0            xor     eax, eax
00404CAB   5B              pop     ebx
00404CAC   83C4 18         add     esp, 18
00404CAF   C3              retn
00404CB0   8A56 07         mov     dl, ds:[esi+7]
```

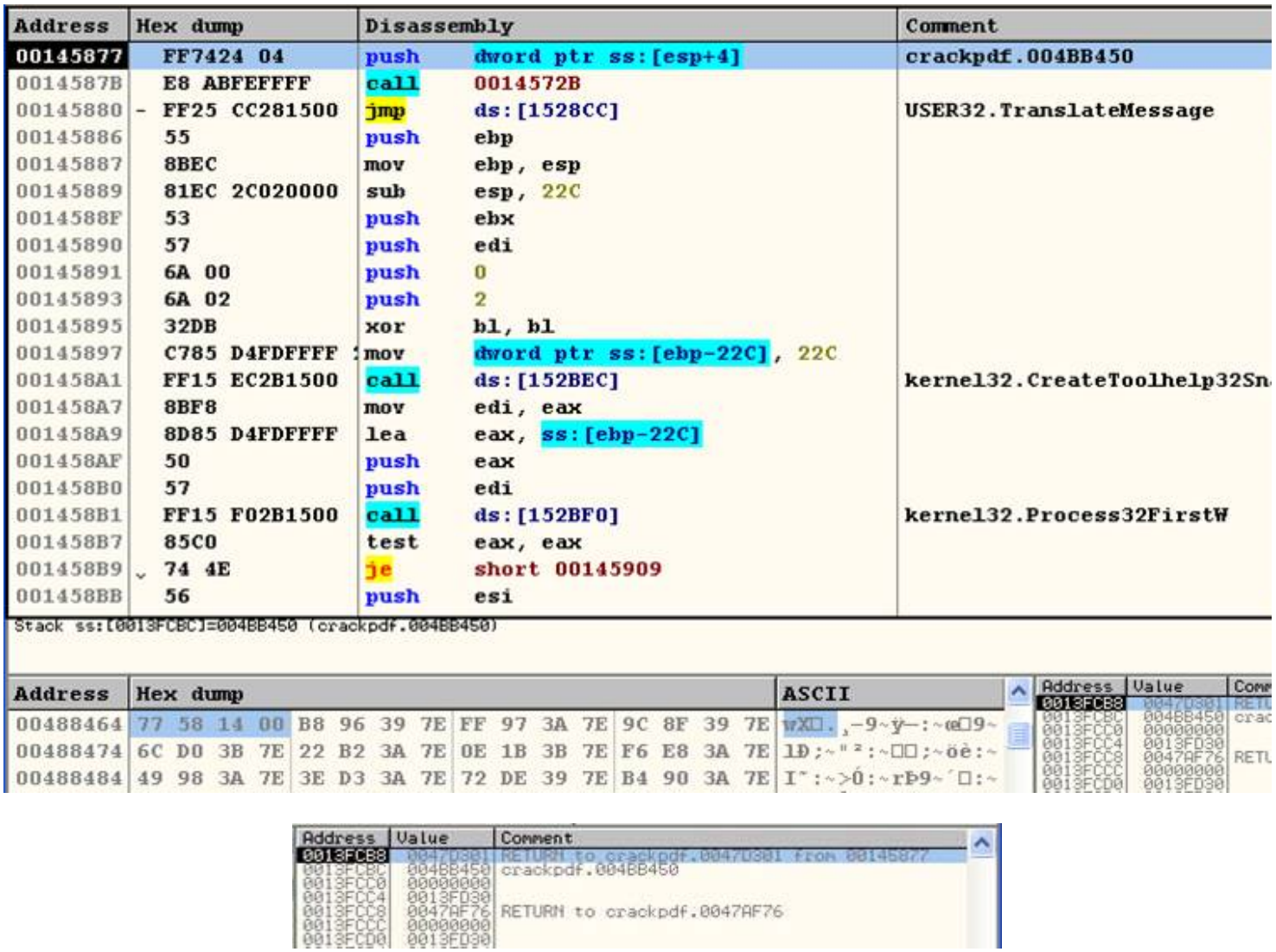
Seguimos traceando y llegamos a un SETE AL y poco después un RETN. Vemos que la condición no se cumple, con lo cual EAX seguirá siendo 0 así que sustituyamos el SETE por SETNE para hacer que se cumpla:





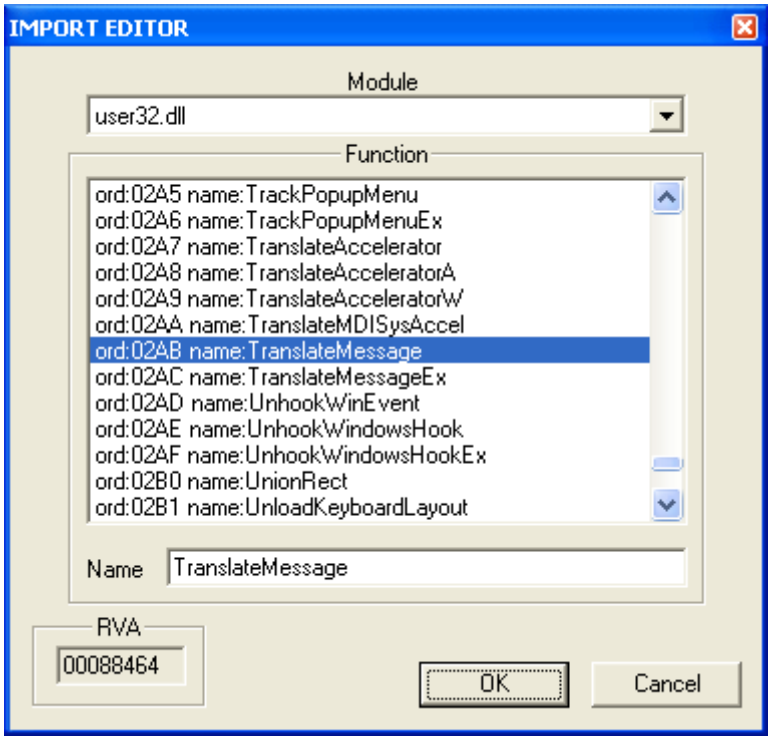
Seguimos traceando y salimos de la CALL y vemos como el salto no se cumple y nos aparece la ventana de chico bueno:

Ahora es el momento de ver cuando para en la entrada mala de la IAT así que, como tenemos un HBP puesto para que pare en cuanto intente hacer algo con ese valor, pues aceptemos la ventanita de chico bueno y demos a F9:

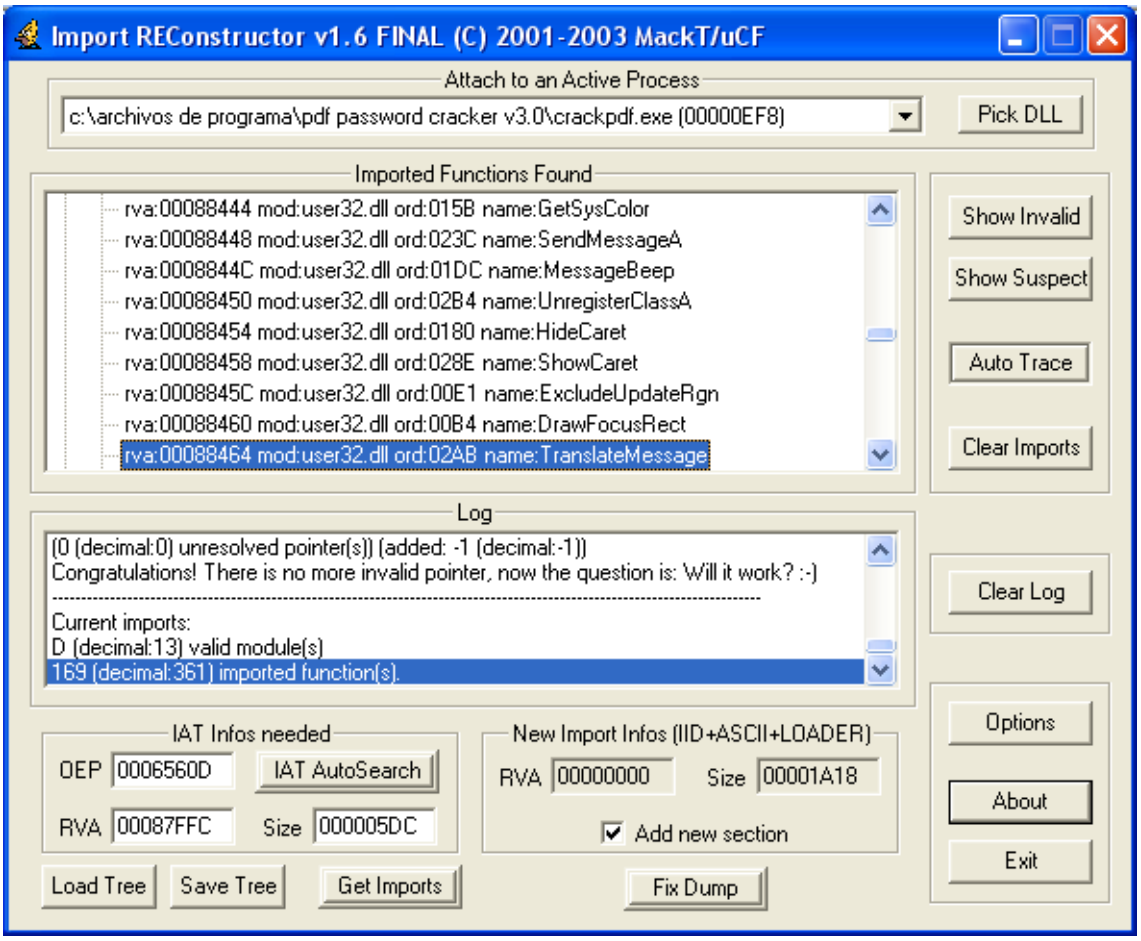


Vemos que esta vez si que paro y que fue desde una CALL, la cual es indirecta ya que no apareció al hacer una búsqueda de referencias.

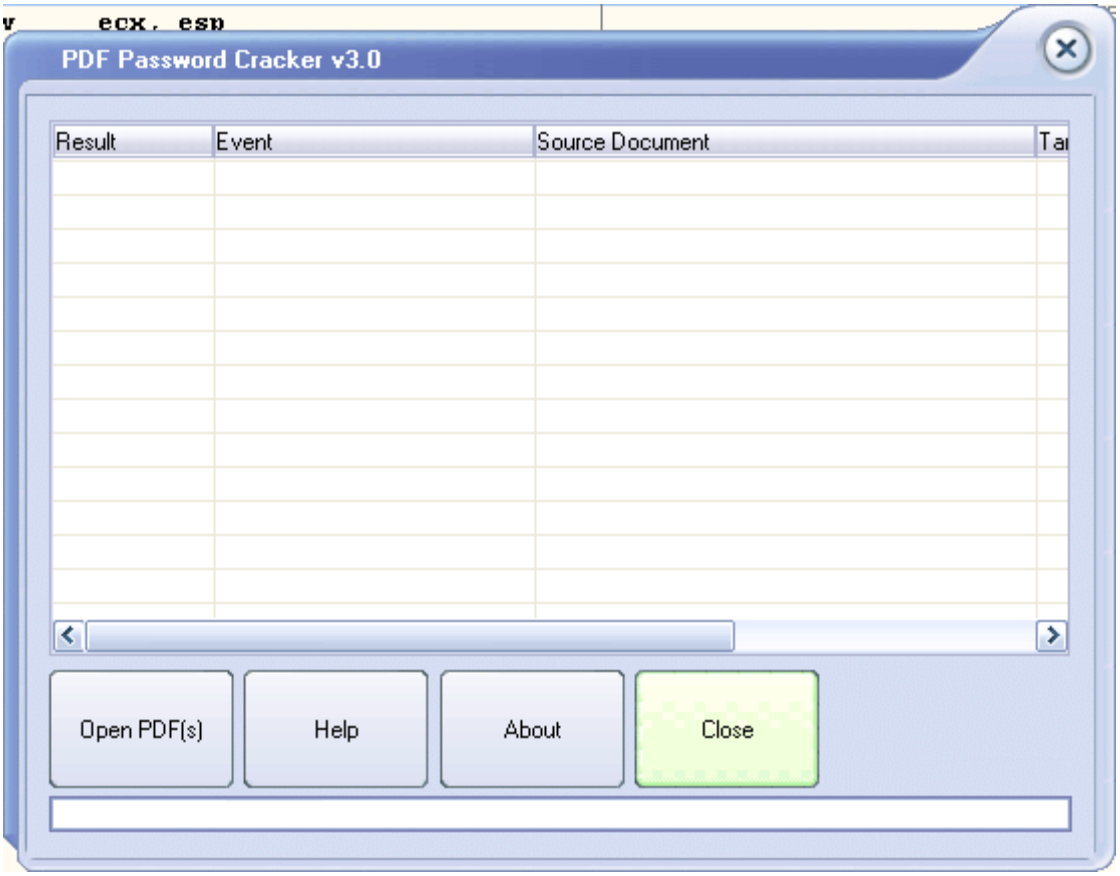
Pues yo diría que esa API es TranslateMessage así que ya tenemos todo lo que necesitamos para intentar sacar nuestro dumpeado.
Reiniciamos de nuevo la aplicación en olly y volvemos a parar en el OEP verdadero y esta vez cambiaremos todos los saltos y el SETE y lo dumpeamos.
Acto seguido hacemos igual que antes con Import Rec y nos colocamos encima de la entrada mala y hacemos doble clic encima y nos mostrara una ventana como esta:



Buscamos la API TranslateMessage y damos a OK:



Como vemos quedo reparada y nos indica que ya no hay más entradas malas así que damos en Fix Dump y elegimos el dumpeado que acabamos de crear y ya tendremos nuestro ejecutable con la IAT reparada.
Ahora toca ver si funciona así que hacemos doble clic en el y…



¡Funciona! Y no solo eso, sino que los mismos saltos que cambiamos son los responsables de comprobar el registro al iniciarse la aplicación, así que al cambiar los saltos conseguimos hacerle creer que estamos registrados.

Bueno, lo he probado y todo va perfecto y sin limitaciones ni nada así que objetivo cumplido jejeje.

Esto es todo y espero que os haya gustado.