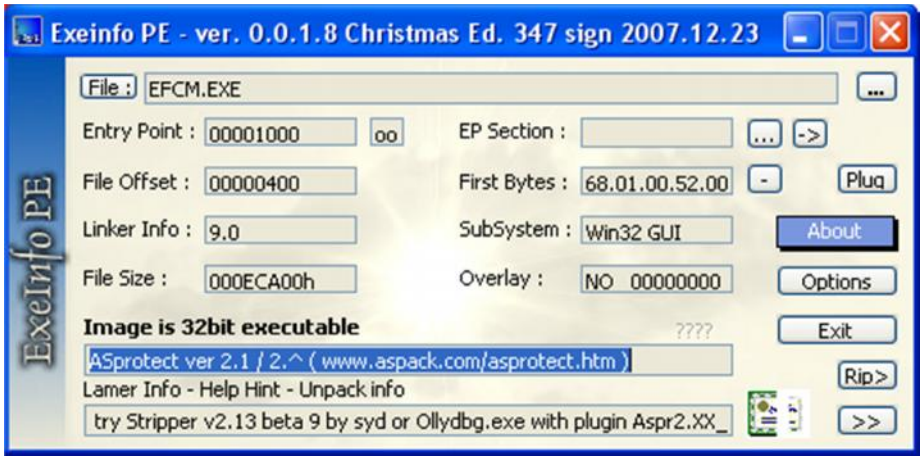
Atacando a EF CheckSum Manager

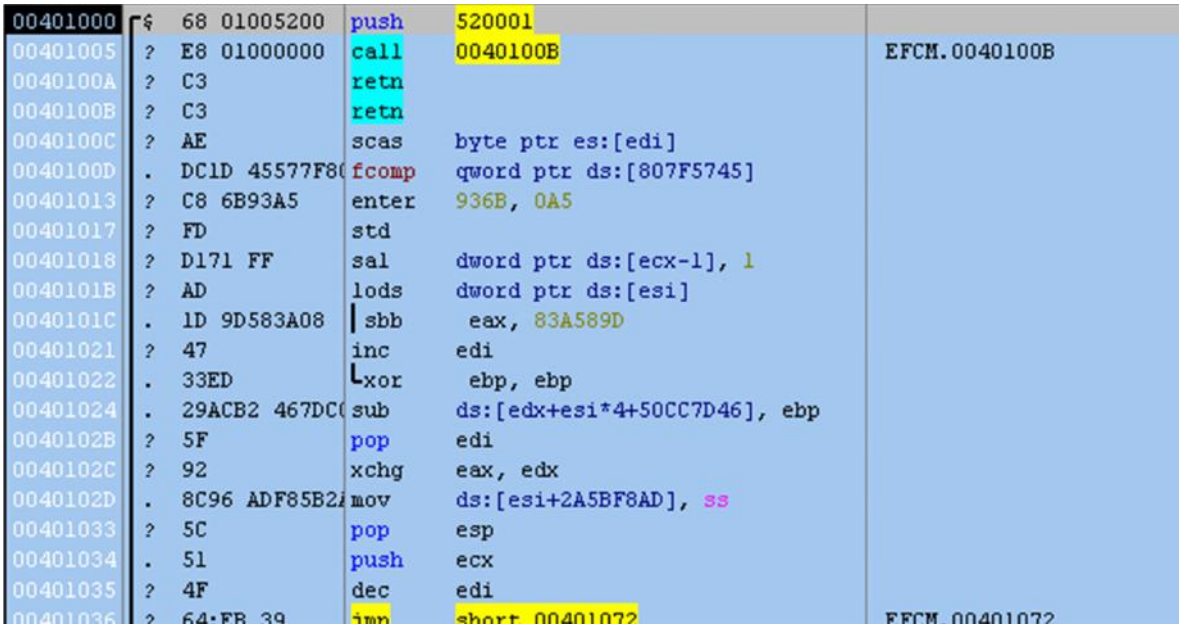
Fecha	**13 de octubre de 2016**
Victima	**EF Checksum Manager v8.0 Portable**
URL de descarga	**http://www.efsoftware.com/dw/wzp.cgi?cm**
MD5 del instalador	**15B41ABC629479576A6B89E05FDDFEC3**
Protección	**ASprotect ver 2.1 / 2.^ (www.aspack.com/asprotect.htm)**
Herramientas	**Exeinfo PE, OllyDbg, Borland C++Builder, Componente DbgCLS**
Objetivo	**Hacer que funcione como registrado**
Dificultad	**Media/Alta**
Cracker	**Aguml**

Analizando a la victima

Lo primero es analizar el ejecutable para ver con que está hecho y veo esto:

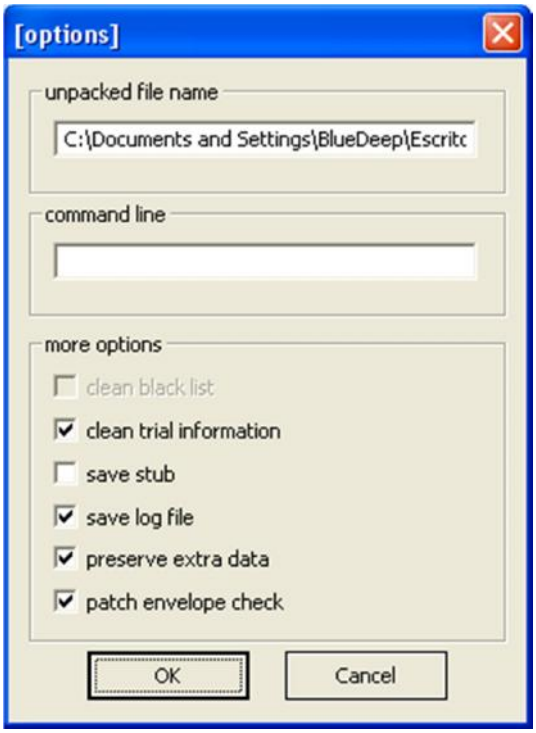


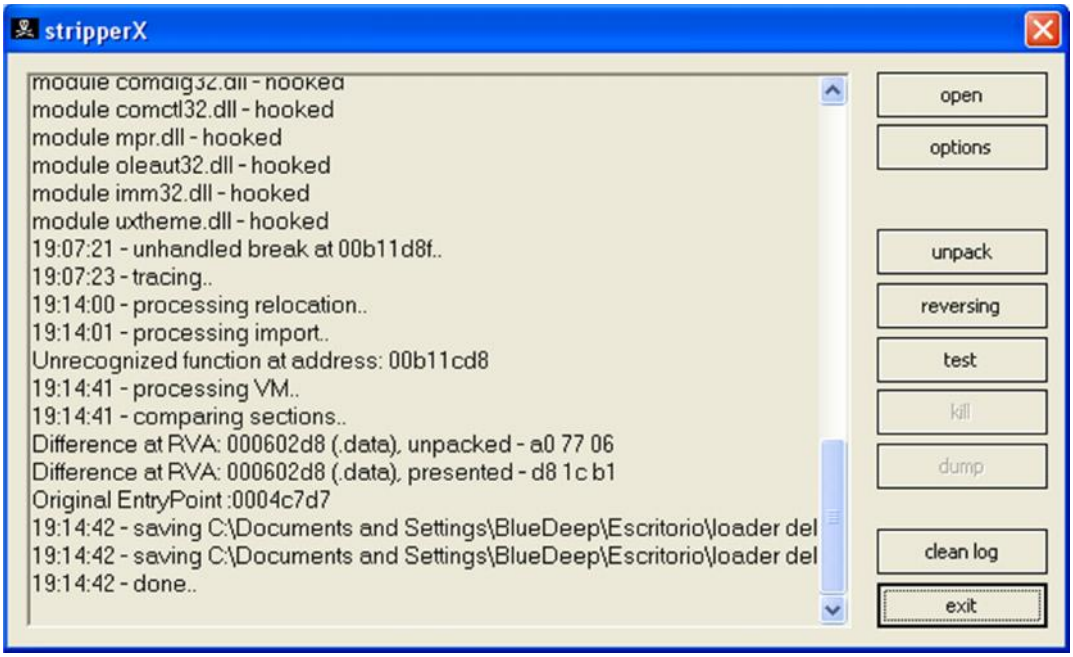
Lo abro en Olly y veo que efectivamente es un asprotect:



Comienza el ataque

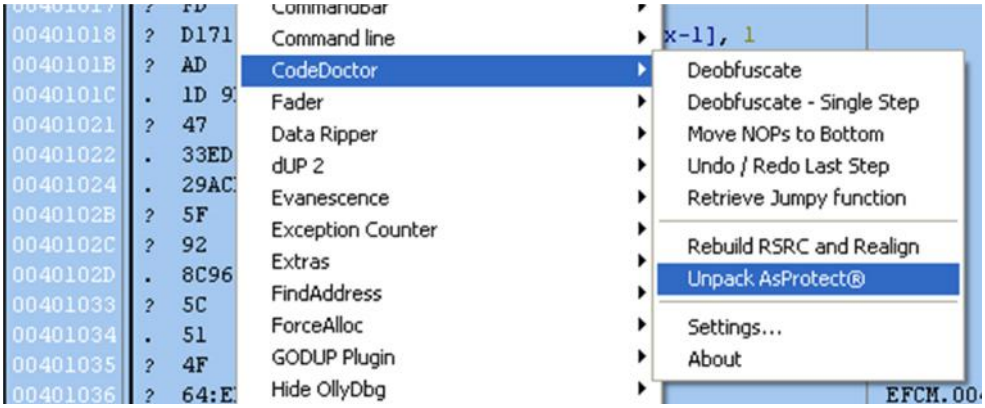
Pruebo con algunos desempacadores.
Primero con el que me recomienda el analizador:





Algo extraño pasa porque el desempacado se cierra al ejecutarse. Si cambio el nombre del ejecutable original por otro y cambio el del dumpeado por el del original ya arranca con lo que hay una zona donde comprueba que no hayamos cambiado el nombre del ejecutable.

Y luego con el plugin de Olly llamado CodeDoctor:

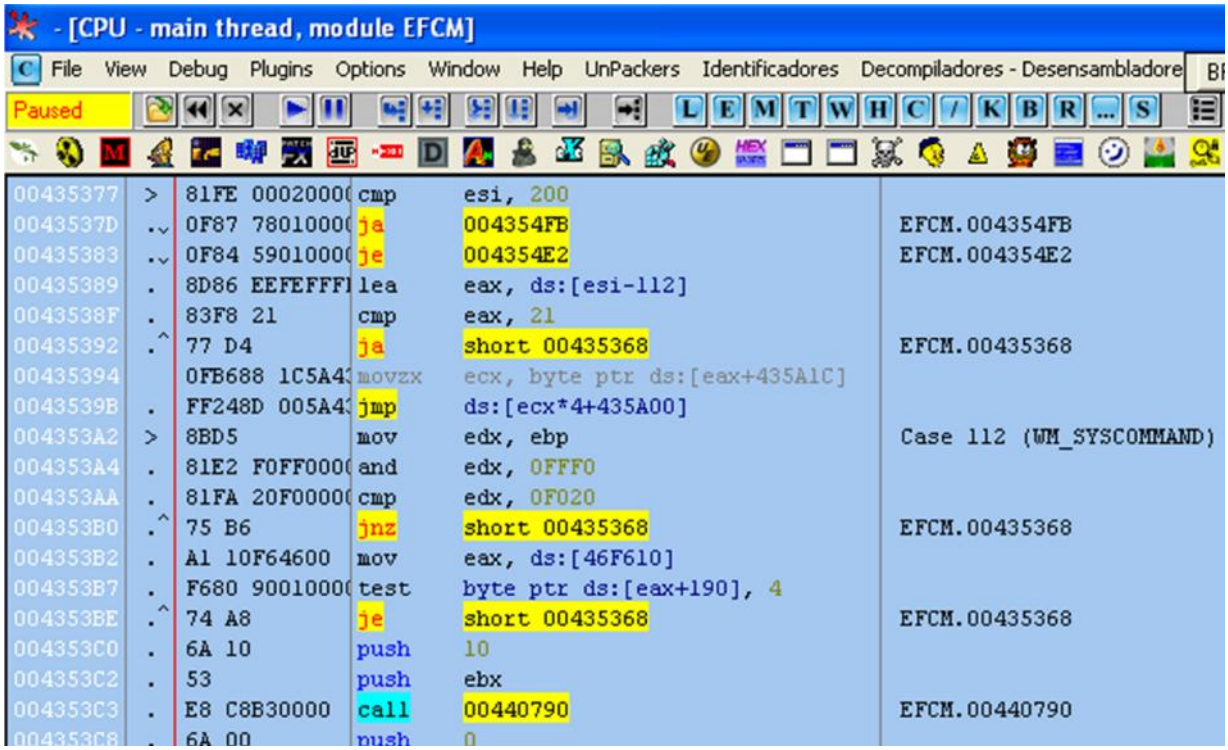


En este caso me encuentro con el mismo problema, el desempacado da error y se cierra de forma silenciosa sin avisar de ningún tipo de error.

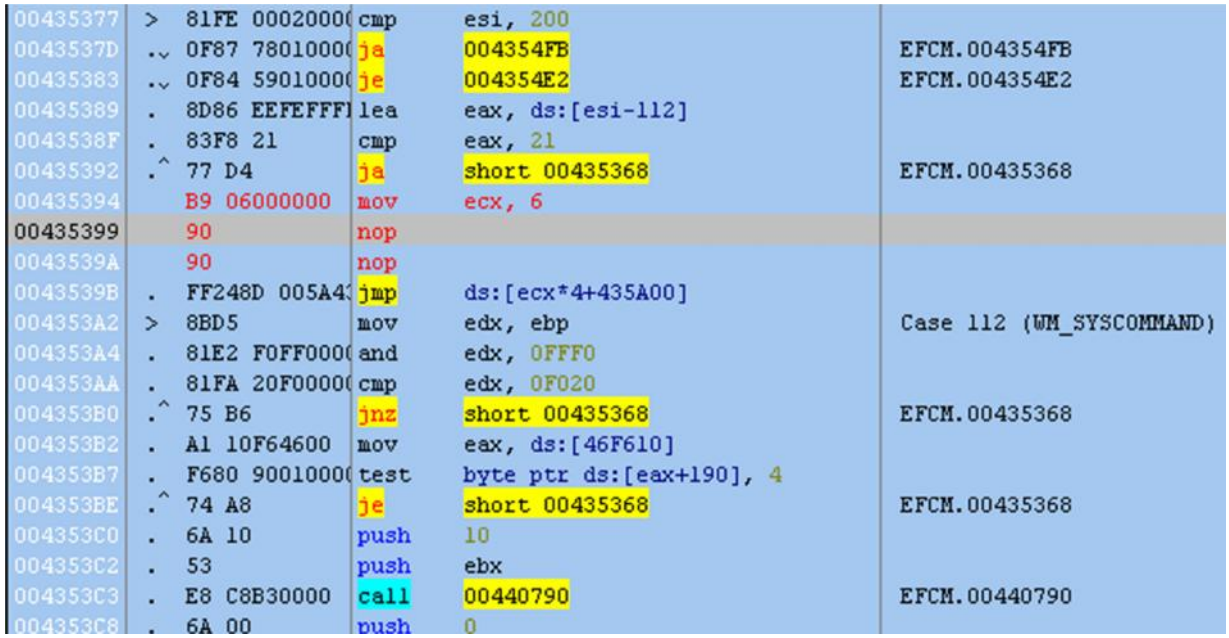
Ismael Perez me dijo esto:

Este ASProtect usa como siempre VAR para todo, el Unpack no es muy difícil. Lo malo la versión de este Protector que hay que fijar un poco casi de todo para que corra, yo más bien me decanto por un simple Loader, y listo, Parchando lo que has puesto está bien, pero con un parcheo queda "Registrado" y todo, lo único queda otra VAR de salir de la APP que es el MessageBox de "Trial" solo hay que fijarla. El tema de la Licencia por lo poco que he visto es bastante simple, si tengo más tiempo lo miraré.

Te pego el código a Parchar, si te fijas es solo un "MOV ECX, 6"

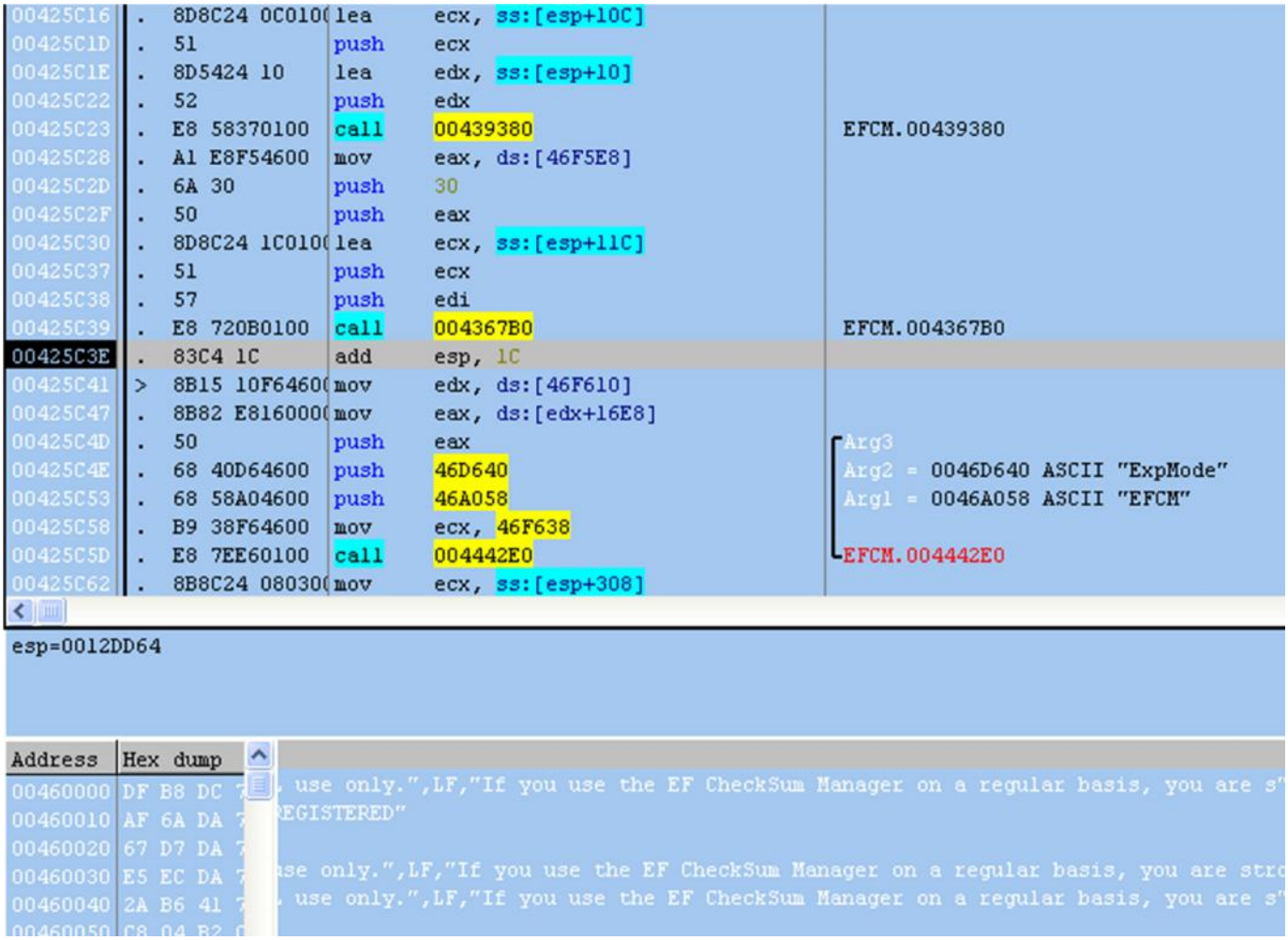


Por :

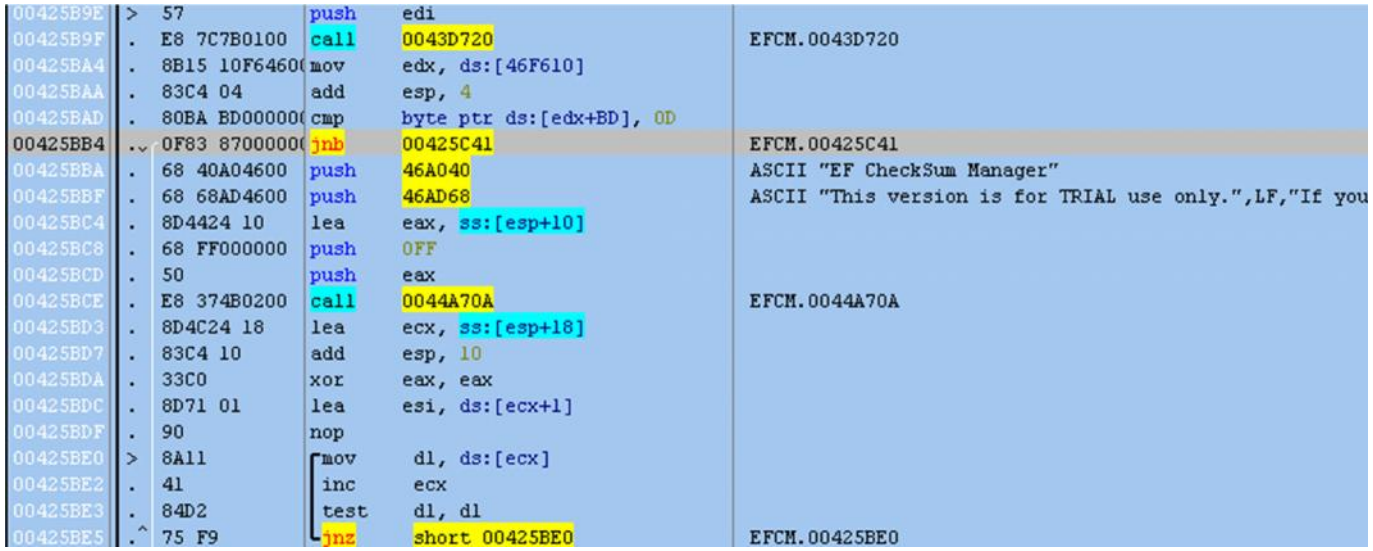


Espero que te ayude aguml en algo.

Ciertamente ese cambio no me aporta mucho y no me funciona todo lo bien que quisiera ya que en realidad no se registra y no muestra los iconos del menú Ayuda.
Sigamos por otro camino. Ataquemos por el mensaje que muestra al cerrarlo. Doy a cerrar y cuando salga el mensaje pauso Olly y voy con Ctrl+F9 y F7 hasta que el depurador se quede a la espera y entonces pulso el botón Aceptar del mensaje y sigo traceando hasta que llego aquí:



Subo un poco y veo esto:



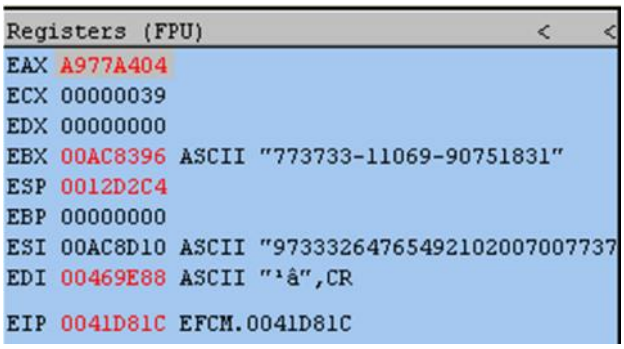
…
…
…

```
00425C11  >  68 00010000  push    100
00425C16  .  8D8C24 0C010( lea     ecx, ss:[esp+10C]
00425C1D  .  51           push    ecx
00425C1E  .  8D5424 10    lea     edx, ss:[esp+10]
00425C22  .  52           push    edx
00425C23  .  E8 58370100  call    00439380                  EFCM.00439380
00425C28  .  A1 E8F54600  mov     eax, ds:[46F5E8]
00425C2D  .  6A 30        push    30
00425C2F  .  50           push    eax
00425C30  .  8D8C24 1C010( lea     ecx, ss:[esp+11C]
00425C37  .  51           push    ecx
00425C38  .  57           push    edi
00425C39  .  E8 720B0100  call    004367B0                  EFCM.004367B0
00425C3E  .  83C4 1C      add     esp, 1C
00425C41  >  8B15 10F6460( mov     edx, ds:[46F610]
00425C47  .  8B82 E816000( mov     eax, ds:[edx+16E8]
00425C4D  .  50           push    eax                       Arg3
00425C4E  .  68 40D64600  push    46D640                    Arg2 = 0046D640 ASCII "ExpMode"
00425C53  .  68 58A04600  push    46A058                    Arg1 = 0046A058 ASCII "EFCM"
00425C58  .  B9 38F64600  mov     ecx, 46F638
00425C5D  .  E8 7EE60100  call    004442E0                  EFCM.004442E0
```

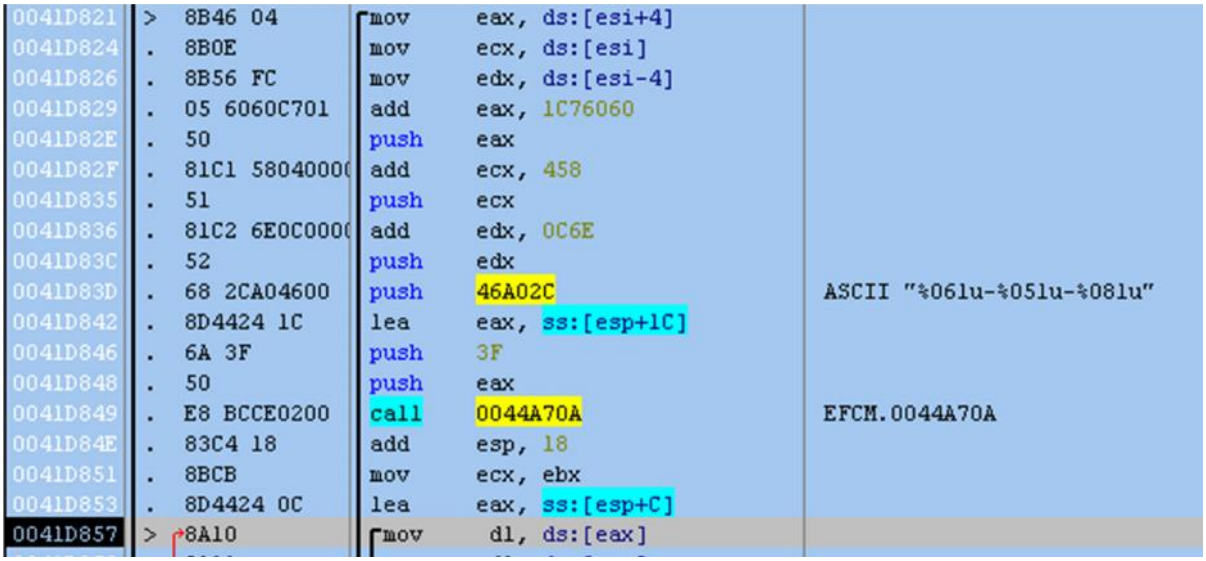
Se puede apreciar como ese salto condicional evita el que nos salga el mensaje de Trial así que lo examino mejor lo que se hace antes para ver si por ahí puedo encontrar como registrarlo pero no doy con el sitio que busco aunque al menos ya sé que al cambiar ese salto por un JMP evito ese molesto mensaje. Aquí me quedé estancado y salió al rescate nuestro amigo Apuromafo quien lo analizó y creo un tuto de cómo crear un loader usando Dup para ello. En mi caso no me funciona pero sí que me vino bien la licencia de otra versión ya que me muestra el siguiente mensaje al colocarla en el directorio:

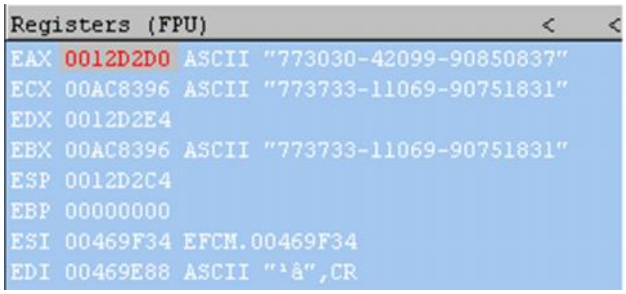

```
0041D9B2  .  E8 C6CC0200  call    0044A67D                  EFCM.0044A67D
0041D9B7  .  81C4 4408000( add     esp, 844
0041D9BD  .  C3           retn
0041D9BE  >  A1 10F64600  mov     eax, ds:[46F610]
0041D9C3  .  FE88 C700000( dec     byte ptr ds:[eax+C7]
0041D9C9  .  68 00040000  push    400
0041D9CE  .  8D5424 50    lea     edx, ss:[esp+50]
0041D9D2  .  52           push    edx
0041D9D3  .  68 44AD4600  push    46AD44                    ASCII "You try to start a ilegal c
0041D9D8  .  E8 A3B90100  call    00439380                  EFCM.00439380
0041D9DD  .  A1 E8F54600  mov     eax, ds:[46F5E8]
0041D9E2  .  8B15 E0F5460( mov     edx, ds:[46F5E0]
0041D9E8  .  6A 40        push    40
0041D9EA  .  50           push    eax
0041D9EB  .  8D4C24 60    lea     ecx, ss:[esp+60]
0041D9EF  .  51           push    ecx
0041D9F0  .  52           push    edx
0041D9F1  .  E8 BA8D0100  call    004367B0                  EFCM.004367B0
0041D9F6  .  A1 E0F54600  mov     eax, ds:[46F5E0]
0041D9FB  .  50           push    eax
0041D9FC  .  E8 6FFC0100  call    0043D670                  EFCM.0043D670
```

Me voy al principio de la función y pongo un HBP on Execution en la primera línea y reinicio y al parar ahí veo esto:

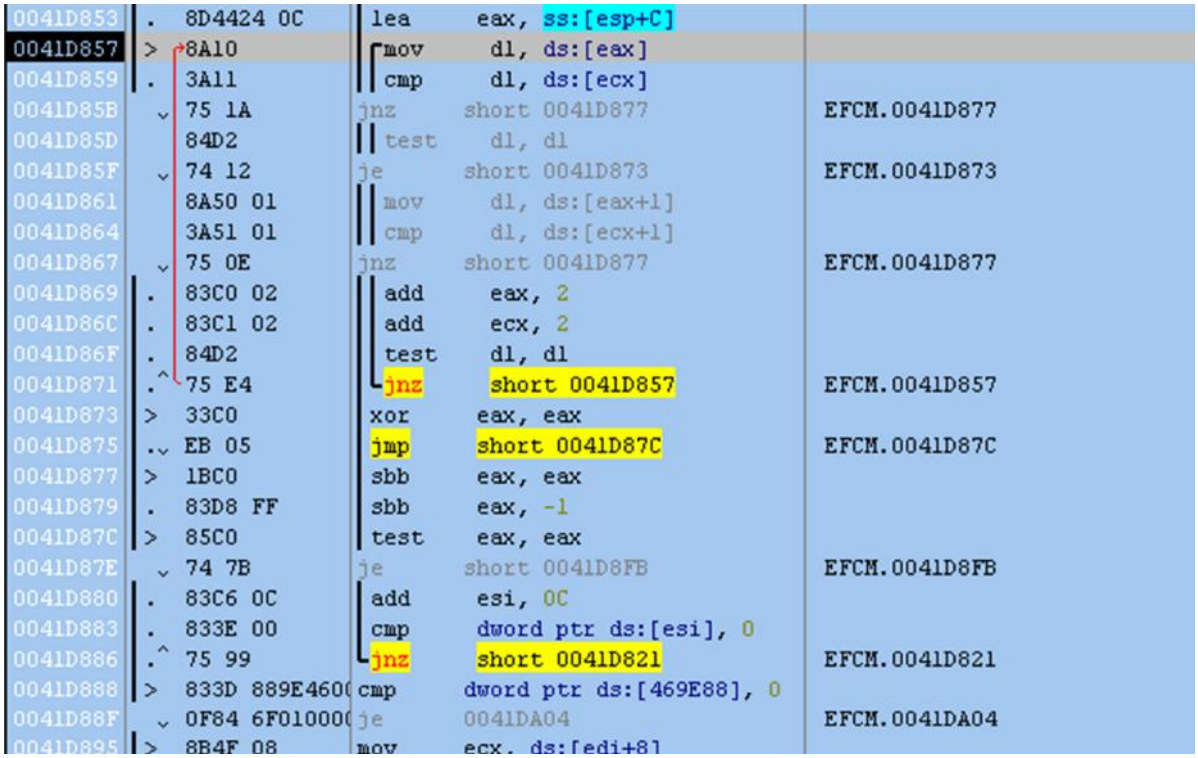


En EAX veo el serial que está en el archivo de licencia y en ESI veo una cadena que no sé de donde sale así que voy traceando para ver que hace:

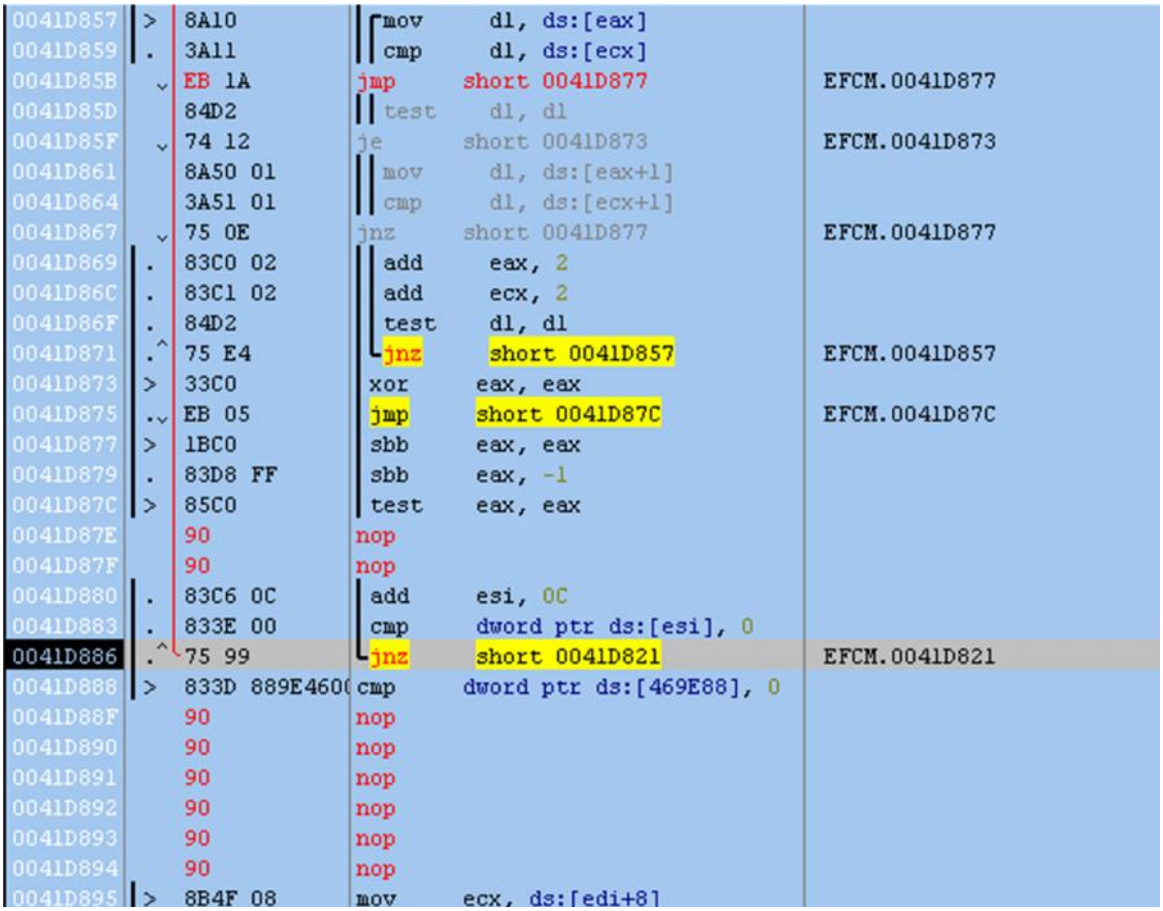




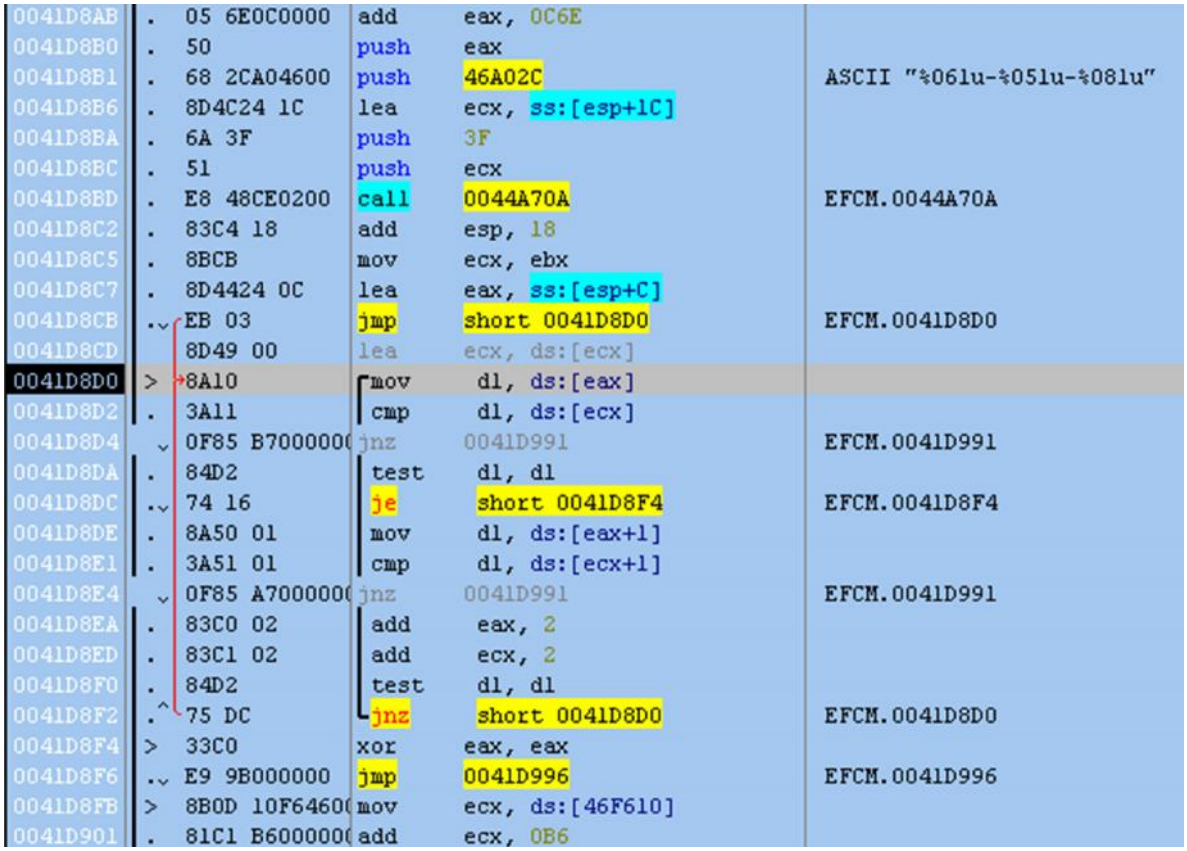
¿Ese serial de dónde sale? Si bajo un poco veo esto:



Después de tracear ese bucle lo que descubro es que compara mi serial con unos pocos de seriales.
En las dos primeras comparaciones va comparando de dos en dos los caracteres de ambas cadenas y el JE es para salir cuando llegue al final de la cadena del serial en caso de llegar antes de lo esperado.
En la línea 0041D87C solo llegará con EAX=0 si ambos seriales son idénticos con lo que se cumpliría el salto de la siguiente línea.
El salto de 41D886 es para que vaya comparando con todos los seriales de la lista con lo que si hubo una coincidencia sale antes de comparar con todos.
El último salto comprueba si se han comparado todos los seriales o se ha salido antes.
Después de muchas vueltas me di cuenta que es una lista negra ya que el serial que viene en el archivo de licencia está en dicha lista y ya vimos el bonito cartel que nos muestra.
Lo que haré será lo siguiente:



Con eso hago que cada vez que vaya a comparar el primer carácter de ambos seriales piense que son diferentes, y que no salte a la zona de chico malo donde muestra el cartelito.
Sigo traceando y llego aquí:



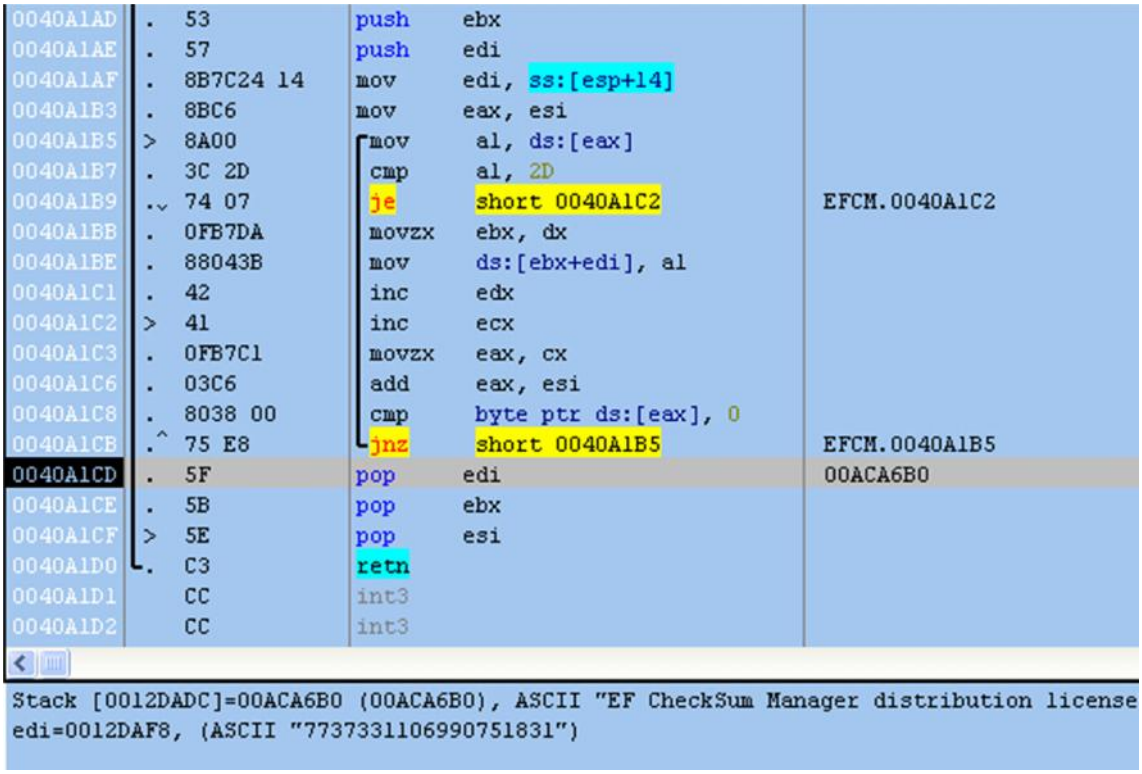
Pues es otra zona donde compara nuestro serial con otra lista negra así que la dejo así:

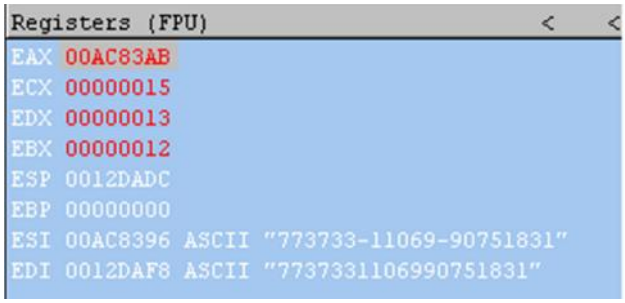
```
0041D8AB  .  05 6E0C0000   add     eax, 0C6E
0041D8B0  .  50            push    eax
0041D8B1  .  68 2CA04600   push    46A02C                   ASCII "%06lu-%05lu-%08lu"
0041D8B6  .  8D4C24 1C     lea     ecx, ss:[esp+1C]
0041D8BA  .  6A 3F         push    3F
0041D8BC  .  51            push    ecx
0041D8BD  .  E8 48CE0200   call    0044A70A                 EFCM.0044A70A
0041D8C2  .  83C4 18       add     esp, 18
0041D8C5  .  8BCB          mov     ecx, ebx
0041D8C7  .  8D4424 0C     lea     eax, ss:[esp+C]
0041D8CB  .v EB 03         jmp     short 0041D8D0           EFCM.0041D8D0
0041D8CD     8D49 00       lea     ecx, ds:[ecx]
0041D8D0  >  8A10          mov     dl, ds:[eax]
0041D8D2  .  3A11          cmp     dl, ds:[ecx]
0041D8D4     E9 B8000000   jmp     0041D991                 EFCM.0041D991
0041D8D9     90            nop
0041D8DA  .  84D2          test    dl, dl
0041D8DC  .v 74 16         je      short 0041D8F4           EFCM.0041D8F4
0041D8DE  .  8A50 01       mov     dl, ds:[eax+1]
0041D8E1  .  3A51 01       cmp     dl, ds:[ecx+1]
0041D8E4     0F85 A700000( jnz     0041D991                 EFCM.0041D991
0041D8EA  .  83C0 02       add     eax, 2
0041D8ED  .  83C1 02       add     ecx, 2
0041D8F0  .  84D2          test    dl, dl
0041D8F2  .^ 75 DC         jnz     short 0041D8D0           EFCM.0041D8D0
0041D8F4  >  33C0          xor     eax, eax
0041D8F6  .v E9 9B000000   jmp     0041D996                 EFCM.0041D996
0041D8FB  >  8B0D 10F6460( mov     ecx, ds:[46F610]
0041D901  .  81C1 B600000( add     ecx, 0B6
```

Sigo traceando y llego aquí:

```
0041D985  .  E8 F3CC0200   call    0044A67D                 EFCM.0044A67D
0041D98A  .  81C4 4408000( add     esp, 844
0041D990  .  C3            retn
0041D991  >  1BC0          sbb     eax, eax
0041D993  .  83D8 FF       sbb     eax, -1
0041D996  >  85C0          test    eax, eax
0041D998     74 24         je      short 0041D9BE           EFCM.0041D9BE
0041D99A  .  83C7 0C       add     edi, 0C
0041D99D  .  833F 00       cmp     dword ptr ds:[edi], 0
0041D9A0     0F85 EFFEFFF( jnz     0041D895                 EFCM.0041D895
0041D9A6  .  5F            pop     edi                      00ACA6B0
0041D9A7  .  5E            pop     esi
0041D9A8  .  5B            pop     ebx
0041D9A9  .  8B8C24 40080( mov     ecx, ss:[esp+840]
0041D9B0  .  33CC          xor     ecx, esp
0041D9B2  .  E8 C6CC0200   call    0044A67D                 EFCM.0044A67D
0041D9B7  .  81C4 4408000( add     esp, 844
0041D9BD  .  C3            retn
0041D9BE  >  A1 10F64600   mov     eax, ds:[46F610]
0041D9C3  .  FE88 C700000( dec     byte ptr ds:[eax+C7]
0041D9C9  .  68 00040000   push    400
0041D9CE  .  8D5424 50     lea     edx, ss:[esp+50]
0041D9D2  .  52            push    edx
0041D9D3  .  68 44AD4600   push    46AD44                   ASCII "You try to start a ilegal
0041D9D8  .  E8 A3B90100   call    00439380                 EFCM.00439380
```

Como se puede ver hemos evitado el salto JE de más arriba que nos lleva directo al mensaje de chico malo con lo que vamos bien aunque si nopeamos ese JE nos curamos en salud jejeje.
Sigo traceando y después de un ratito llego aquí:



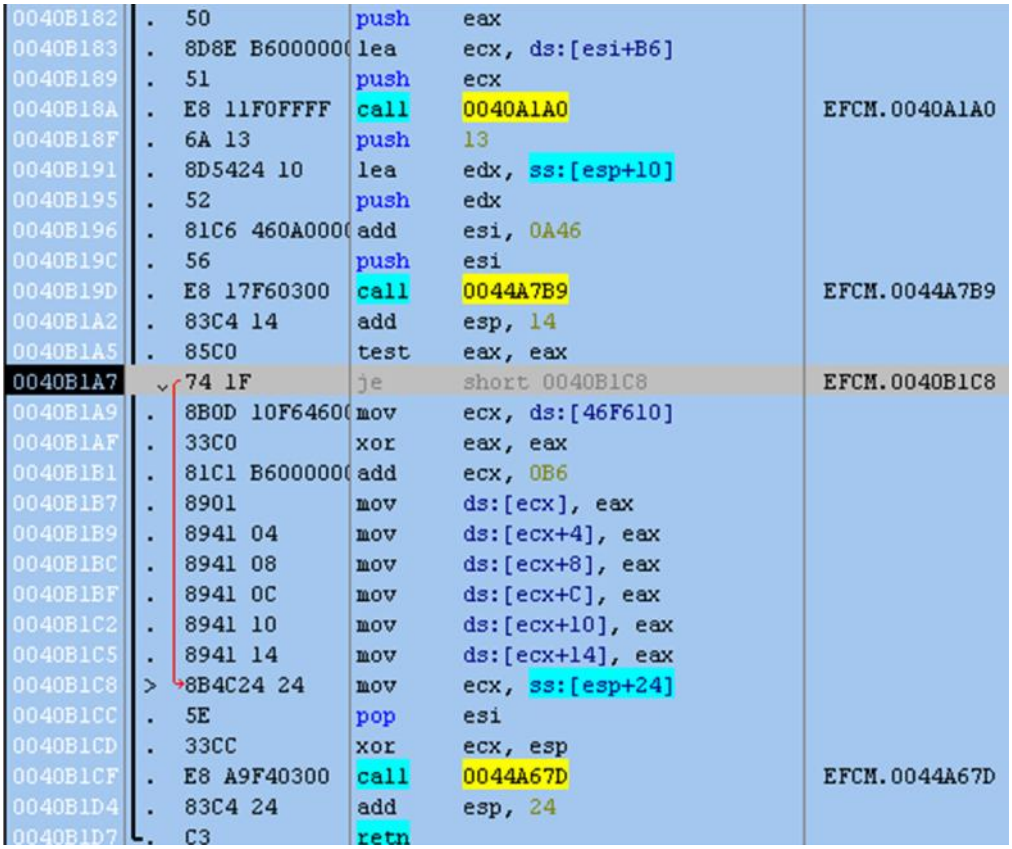


Se puede apreciar que le ha quitado todos los guiones a nuestro serial y lo ha guardado en otro sitio.
Sigo traceando y llego aquí:

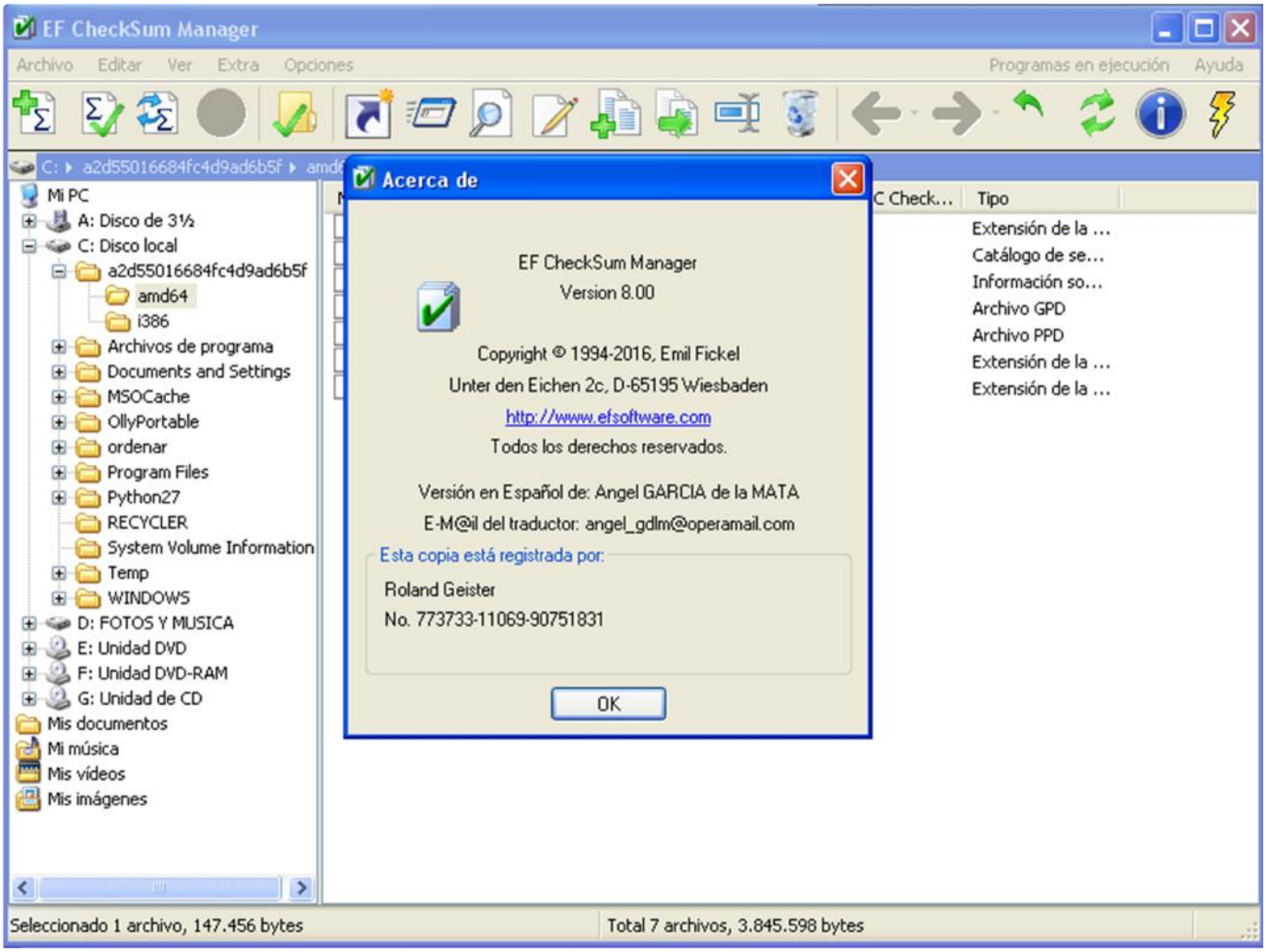
```
0040B18F  .  6A 13         push    13
0040B191  .  8D5424 10     lea     edx, ss:[esp+10]
0040B195  .  52            push    edx
0040B196  .  81C6 460A000( add     esi, 0A46
0040B19C  .  56            push    esi
0040B19D  .  E8 17F60300   call    0044A7B9                 EFCM.0044A7B9
0040B1A2  .  83C4 14       add     esp, 14
0040B1A5  .  85C0          test    eax, eax
0040B1A7  v  74 1F         je      short 0040B1C8           EFCM.0040B1C8
0040B1A9  .  8B0D 10F6460( mov     ecx, ds:[46F610]
0040B1AF  .  33C0          xor     eax, eax
0040B1B1  .  81C1 B6000000( add    ecx, 0B6
0040B1B7  .  8901          mov     ds:[ecx], eax
0040B1B9  .  8941 04       mov     ds:[ecx+4], eax
```

```
EAX 00AC83AB
ECX 00000015
EDX 00FDF73C ASCII "77373311069907518"
EBX 00000000
ESP 00FDF724
EBP 00000000
ESI 00AC8D26 ASCII "773733110699075183188062310160113 93"
EDI 00AC82E0
```

Va a entrar a ese CALL y veo en EDX mi serial sin guiones y en ESI algo que parece mi serial sin guiones pero más largo. Después de analizar lo que se hace dentro veo que es algo así como el strncmp de C donde lo que se hace es comparar los *n* caracteres de ambas cadenas. En este caso 0x13 caracteres que es justo el largo de mi serial sin guiones y que coinciden todos los caracteres en ambas cadenas con lo que no habrá problemas.
Al salir llego aquí:



Ese salto se cumple porque ambas cadenas han coincidido pero por si acaso quedémonos con la dirección de ese salto.
Si doy a F9 veo que ya no aparece el NOT REGISTERED y si voy al About veo esto:



Con estos cambios ya está registrado aunque con el nombre de otra persona. Intentemos darle más duro, intentemos que aparezca registrado con cualquier nombre y con cualquier serial.
Empecemos por cambiar solo el nombre en el archivo de licencia. A continuación reiniciamos Olly y, como ya tenemos un HBP que parará justo antes de revisar la listas negras y es la primera comprobación que hemos visto que hace, damos a F9 y cuando pare hacemos todos los cambios que hemos visto y pongo un BP en todos para ver que hace diferente y pasa todo correctamente pero algo se nos escapa porque tiene alguna comprobación para el nombre y no es ninguna de las que hemos visto.
Empiezo de nuevo con el proceso y esta vez iré más despacio fijándome en todo lo que pueda. Esta vez voy traceando, entrando en CALLs sospechosos con F7, saliendo de ellos… pero no hay nada que no hayamos visto en ninguno de esos CALLs pero al ir saliendo de las distintas funciones en la que está la comprobación de la lista negra llego hasta aquí:

```
004329F0  .  8B4424 14    mov     eax, ss:[esp+14]
004329F4  .  52           push    edx
004329F5  .  50           push    eax
004329F6  .  56           push    esi
004329F7  .  C641 16 49   mov     byte ptr ds:[ecx+16], 49
004329FB  .  E8 30CFFFFF  call    0042F930                  EFCM.0042F930
00432A00  .  83C4 0C      add     esp, 0C
00432A03  .  6A 00        push    0                         Timerproc = NULL
00432A05  .  68 D0070000  push    7D0                       Timeout = 2000. ms
00432A0A  .  6A 01        push    1                         TimerID = 1
00432A0C  .  56           push    esi                       hWnd
00432A0D  .  FF15 B804460 call    ds:[4604B8]               SetTimer
00432A13  .  A1 10F64600  mov     eax, ds:[46F610]
00432A18  .  0FB788 F6160 movzx   ecx, word ptr ds:[eax+16F6]
00432A1F  .  8B50 08      mov     edx, ds:[eax+8]
00432A22  .  6A 00        push    0
00432A24  .  51           push    ecx
00432A25  .  68 C89D4600  push    469DC8                    UNICODE "eeffgghhiij
00432A2A  .  6A 00        push    0
00432A2C  .  52           push    edx
00432A2D  .  E8 3EFD0000  call    00442770                  EFCM.00442770
```

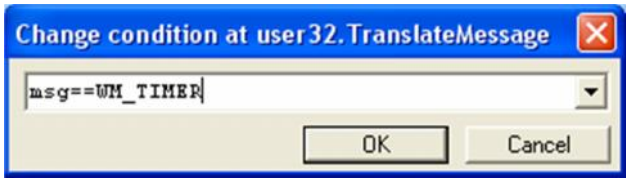
```
0012EB30  00150670  hWnd = 00150670 ('EF CheckSum Manager',class='
0012EB34  00000001  TimerID = 1
0012EB38  000007D0  Timeout = 2000. ms
0012EB3C  00000000  Timerproc = NULL
0012EB40  0012F4C8
```

Esta parte me trajo loco ya que ¿para qué quiere crear un Timer de este modo y que el lapso sea de 2 segundos? La curiosidad me podía así que busqué información y, de todo lo que explica la MSDN acerca de esta función, lo más interesante es esto:

lpTimerFunc [in, optional]

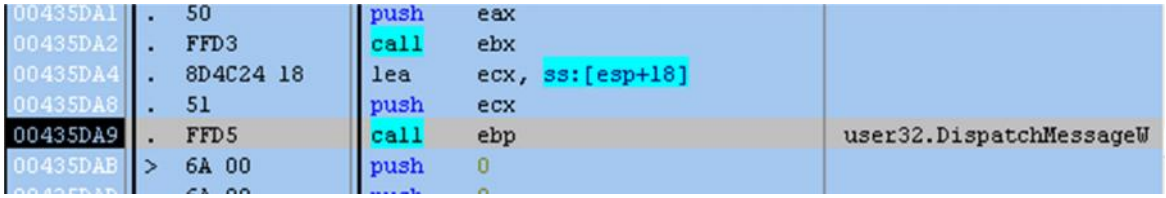
> Type: **TIMERPROC**
>
> A pointer to the function to be notified when the time-out value elapses. For more information about the function, see **TimerProc**. If *lpTimerFunc* is **NULL**, the system posts a **WM_TIMER** message to the application queue. The **hwnd** member of the message's **MSG** structure contains the value of the *hWnd* parameter.

Lo que viene a decir es que si Timerproc es NULL se enviará un mensaje de tipo WM_TIMER a la cola de mensajes del proceso para que el proceso haga lo que le venga en gana con este mensaje. Entonces se me ocurrió poner un BP Condicional en TranslateMessage:



Doy a OK y al dar a F9 para y veo esto en es stack:

Salgo con Ctrl+F9 y F7 y caigo justo en el lugar del código donde se controlan los mensajes así que traceo unas líneas más y veo esto:



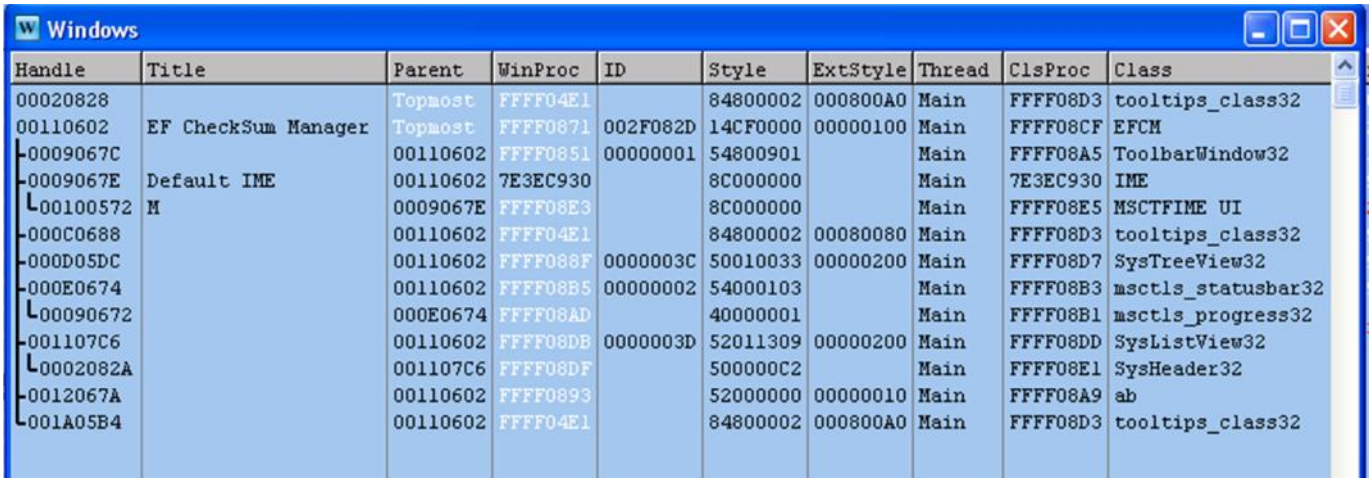
Aquí me vuelvo a quedar atascado ya que según la MSDN:

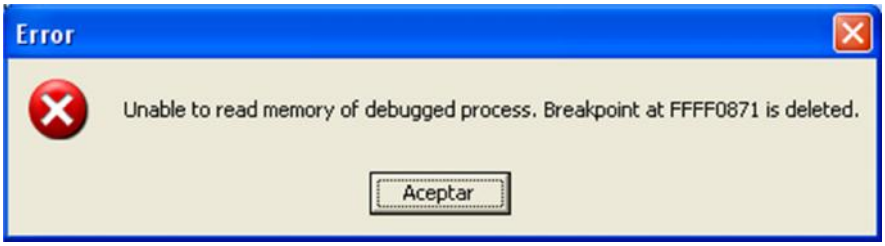
*Dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the **GetMessage** function.*

Que el traductor de Google lo deja como:

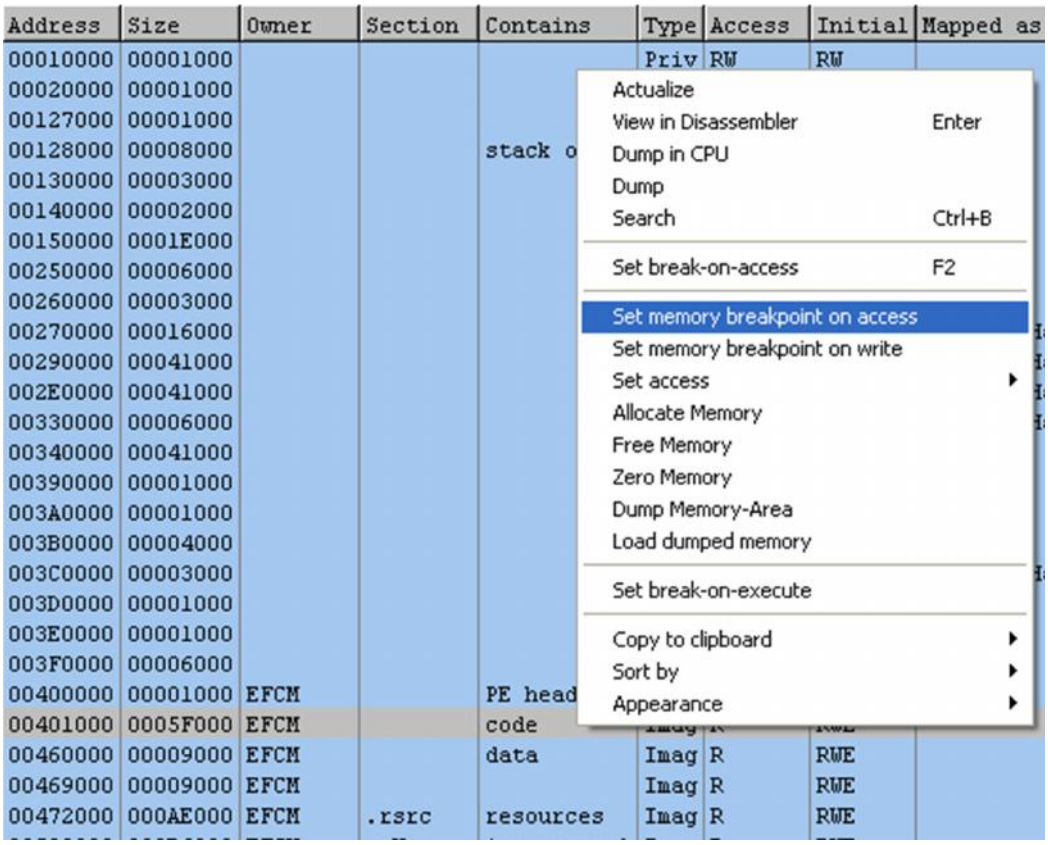
Distribuye un mensaje a un procedimiento de ventana. Por lo general se utiliza para enviar un mensaje recuperado por la función GetMessage.

¿Y ahora? No sé el motivo pero si doy a W no me deja poner ningún tipo de BP:

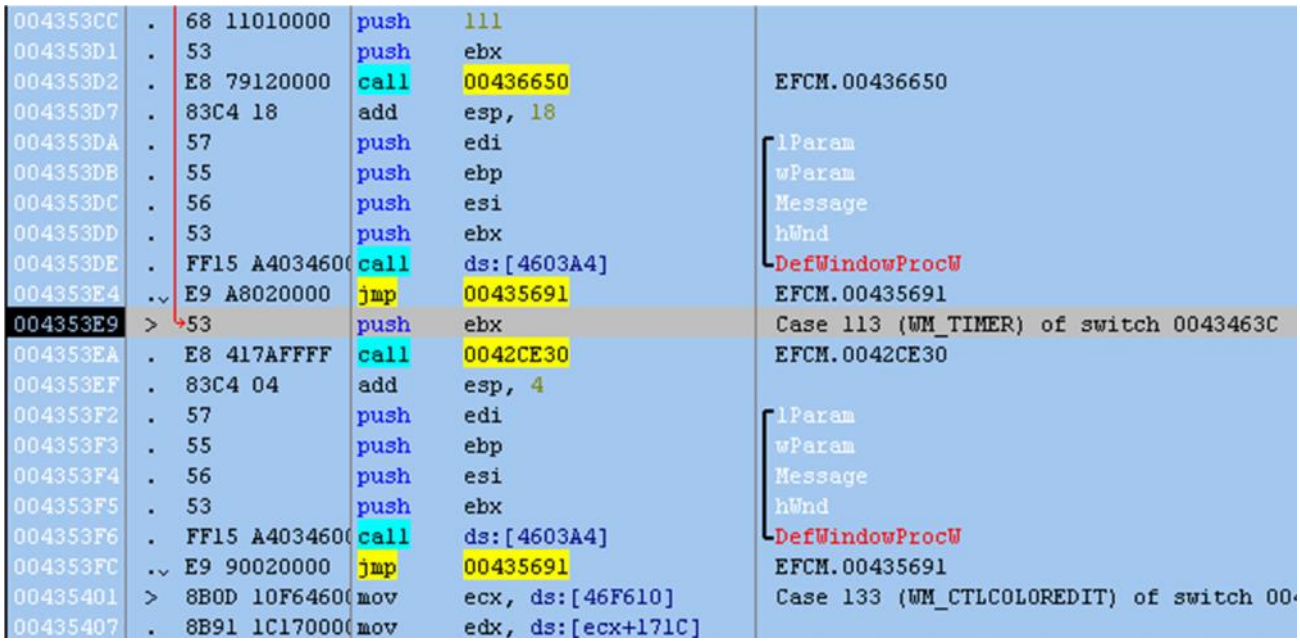




No sé porque las direcciones de los procedimientos son todas incorrectos.
Se me ocurrió otro modo, como lo que va a hacer lo hará en la sección .CODE del proceso ¿Y si pongo un BP en toda la sección para ver dónde caigo?



Después voy dando a F9 hasta que por fin caigo aquí:



Entro en el CALL y voy traceando y llego aquí:

```
0042CF83  .v 74 52        je      short 0042CFD7          EFCM.0042CFD7
0042CF85  .  8B35 E8F5460(mov     esi, ds:[46F5E8]
0042CF8B  .  05 CE000000  add     eax, 0CE
0042CF90  .  50           push    eax
0042CF91  .  56           push    esi
0042CF92  .  E8 A9B7FEFF  call    00418740                EFCM.00418740
0042CF97  .  83C4 08      add     esp, 8
0042CF9A  .  85C0         test    eax, eax
0042CF9C  .v 75 39        jnz     short 0042CFD7          EFCM.0042CFD7
0042CF9E  .  B8 40A04600  mov     eax, 46A040             ASCII "EF CheckSum Manager"
0042CFA3  .  8D50 01      lea     edx, ds:[eax+1]
```

```
Registers (FPU)                                    <
EAX 00AC83AE ASCII " NOT REGISTERED"
ECX 00000001
EDX 00000022
EBX 000E0844
ESP 0012F528
EBP 00000001
ESI 00AC83EE UNICODE "EF CheckSum Manager"
EDI 00000000
```

Entro en el CALL y luego de tracearlo me doy cuenta que lo que hace es buscar la cadena de EAX en la cadena de ESI, o sea que es algo como el strstr de C pero con la diferencia de que en este caso una cadena es UNICODE y la otra es ASCII y strstr no admitiría una búsqueda así. En este caso no la encuentra con lo que quiere decir que hemos pasado esta prueba sin ningún problema gracias a los cambios ya realizados y el salto que está unas líneas más abajo del CALL no se realizará. Sigo traceando hasta que llego aquí:

```
0042CFC8  .  51           push    ecx
0042CFC9  .  05 CE000000  add     eax, 0CE
0042CFCE  .  50           push    eax
0042CFCF  .  E8 ECC60000  call    004396C0                EFCM.004396C0
0042CFD4  .  83C4 0C      add     esp, 0C
0042CFD7  >  E8 8480FEFF  call    00415060                EFCM.00415060
0042CFDC  .  8B0D 10F6460(mov     ecx, ds:[46F610]
0042CFE2  .  8A81 0617000(mov     al, ds:[ecx+1706]
0042CFE8  .  5E           pop     esi
0042CFE9  .  3C 01        cmp     al, 1
0042CFEB  .v 76 19        jbe     short 0042D006          no salta
```

Entro y traceo hasta llegar aquí:

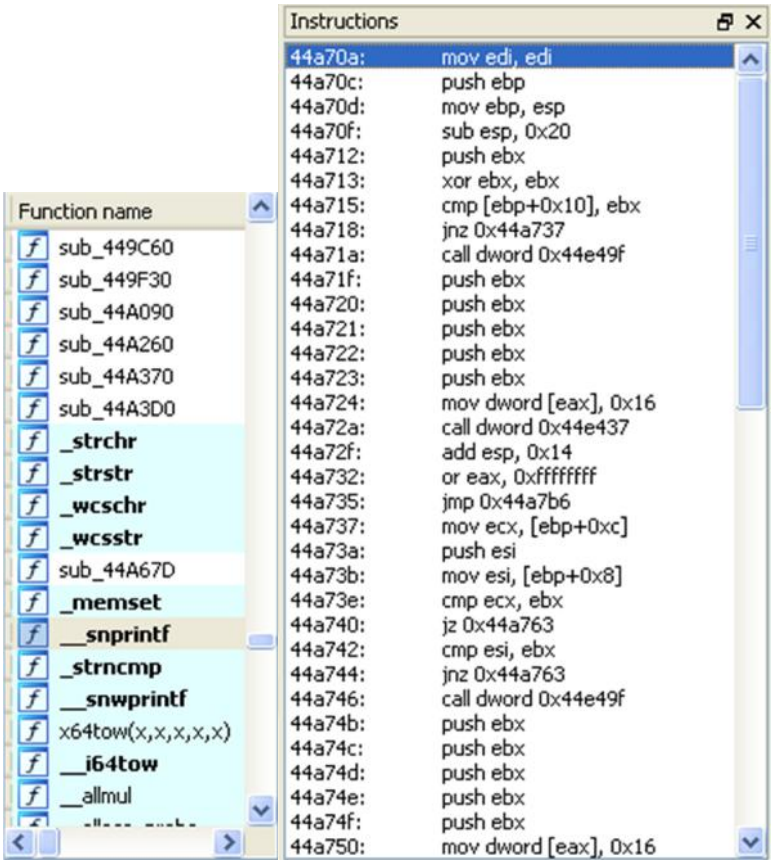
```
004150E8  >  52           push    edx
004150E9  .  68 04A04600  push    46A004                    ASCII "%05lu"
004150EE  .  8D45 00      lea     eax, ss:[ebp]
004150F1  .  6A 7F        push    7F
004150F3  .  50           push    eax
004150F4  .  E8 11560300  call    0044A70A                  EFCM.0044A70A
004150F9  .  A1 10F64600  mov     eax, ds:[46F610]
004150FE  .  8A4D 01      mov     cl, ss:[ebp+1]
00415101  .  83C4 10      add     esp, 10
00415104  .  3A88 310A000(cmp     cl, ds:[eax+A31]
0041510A  .v 75 16        jnz     short 00415122            no salta
0041510C  .  8A55 02      mov     dl, ss:[ebp+2]
0041510F  .  3A90 320A000(cmp     dl, ds:[eax+A32]
00415115  .v 75 0B        jnz     short 00415122            no salta
00415117  .  8A4D 03      mov     cl, ss:[ebp+3]
0041511A  .  3A88 330A000(cmp     cl, ds:[eax+A33]
00415120  .v 74 1B        je      short 0041513D            salta
00415122  >  8A88 330A000(mov     cl, ds:[eax+A33]
00415128  .  84C9         test    cl, cl
0041512A  .v 74 11        je      short 0041513D            EFCM.0041513D
```

```
Registers (FPU)                                    <
EAX 0012F4A4 ASCII "Agustin"
ECX 00000007
EDX 0000BD57
EBX 000E0844
ESP 0012F274
EBP 0012F4A4 ASCII "Agustin"
ESI 0000006E
EDI 00000000
```

Entro pero no entiendo lo que hace aunque al salir obtengo esto:

```
Registers (FPU)                                    <
EAX 00000005
ECX 191DEE39
EDX 0012F4A8
EBX 000E0844
ESP 0012F274
EBP 0012F4A4 ASCII "48471"
ESI 0000006E
EDI 00000000
```

Entonces me da por abrir el desempacado en IDA y ver que muestra y veo esto:

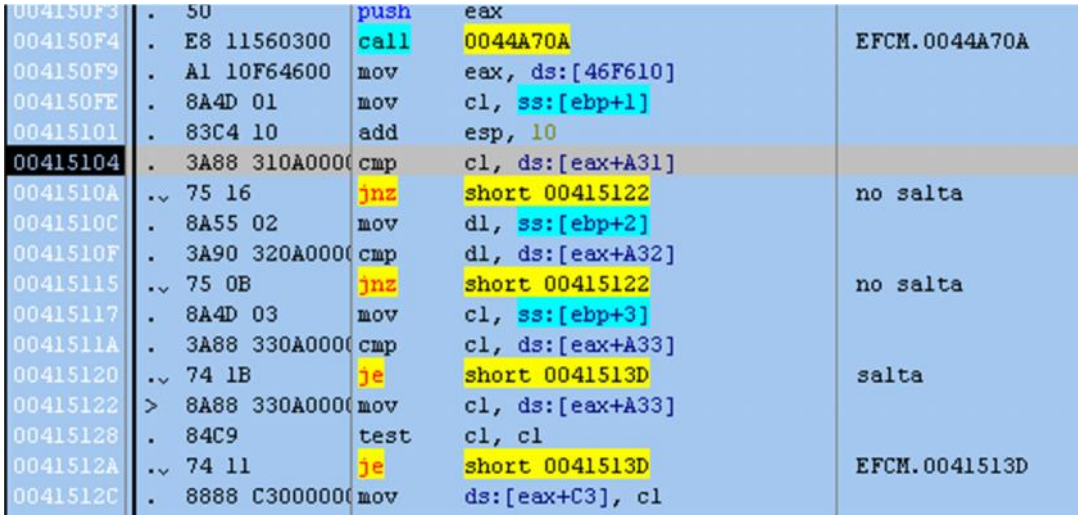


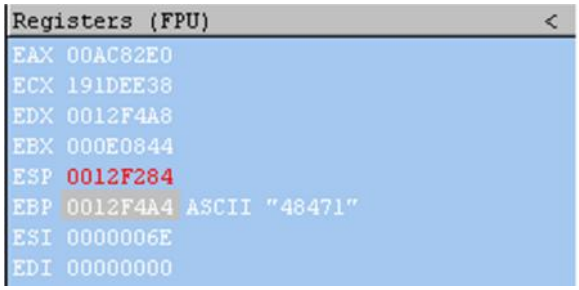
C++

```
uint32_t __output_l(void* a1, signed char* a2, int32_t a3, void* a4);

void __flsbuf();

int32_t* __errno();

uint32_t __invalid_parameter(int32_t a1, int32_t a2, int32_t a3, int32_t a4, int32_t a5);

uint32_t __snprintf(signed char* a1, signed char* a2, signed char* a3) {
    signed char* v4;
    uint32_t eax5;
    int32_t* eax6;
    int32_t* eax7;

    if (a3) {
        if (!a2 || a1) {
            v4 = reinterpret_cast<signed char*>(0x7fffffff);
            if (reinterpret_cast<uint32_t>(a2) <= 0x7fffffff) {
                v4 = a2;
            }
            eax5 = __output_l(reinterpret_cast<int32_t>(__zero_stack_offset()) - 4 - 32, a3, 0, reinterpret_cast<int3
            if (a1) {
                if (reinterpret_cast<int32_t>(v4 - 1) < reinterpret_cast<int32_t>(0)) {
                    __flsbuf();
                } else {
                    *a1 = 0;
                }
                eax5 = eax5;
            }
        } else {
```

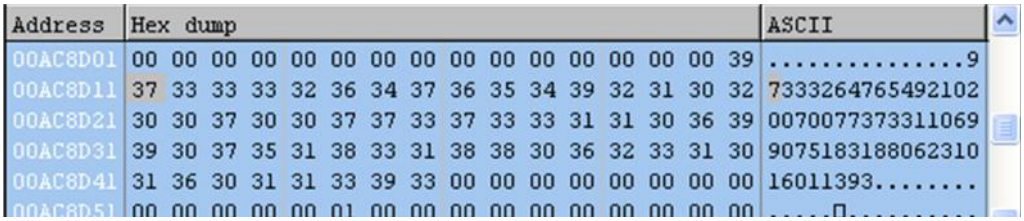
Según Microsoft, esta función escribe datos con formato en una cadena. Es parecida al sprintf de C pero con la peculiaridad que permite indicar un valor de largo máximo de cadena que en este código el valor es 0x7F. En este caso el formato es “%5lu”.

Traceo hasta aquí:





Si voy a la dirección con la que comparará la cadena de EBP:



Compara con algunos de los caracteres de la cadena que usó antes para comparar con el serial sin guiones y que no tengo ni idea de donde la ha sacado. Cambio esos saltos para que piense que coincide todo:

```
004150F3  .  50            push    eax
004150F4  .  E8 11560300   call    0044A70A                 EFCM.0044A70A
004150F9  .  A1 10F64600   mov     eax, ds:[46F610]
004150FE  .  8A4D 01       mov     cl, ss:[ebp+1]
00415101  .  83C4 10       add     esp, 10
00415104  .  3A88 310A000( cmp     cl, ds:[eax+A31]
0041510A     90            nop                              no salta
0041510B     90            nop
0041510C     8A55 02       mov     dl, ss:[ebp+2]
0041510F     3A90 320A000( cmp     dl, ds:[eax+A32]
00415115     90            nop                              no salta
00415116     90            nop
00415117     8A4D 03       mov     cl, ss:[ebp+3]
0041511A     3A88 330A000( cmp     cl, ds:[eax+A33]
00415120     EB 1B         jmp     short 0041513D           salta
00415122  >  8A88 330A000( mov     cl, ds:[eax+A33]
00415128  .  84C9          test    cl, cl
0041512A  .  74 11         je      short 0041513D           EFCM.0041513D
0041512C  .  8888 C300000( mov     ds:[eax+C3], cl
```

Sigo traceando y llego aquí:

```
00415120     EB 1B         jmp     short 0041513D           salta
00415122  >  8A88 330A000( mov     cl, ds:[eax+A33]
00415128  .  84C9          test    cl, cl
0041512A  .  74 11         je      short 0041513D           EFCM.0041513D
0041512C  .  8888 C300000( mov     ds:[eax+C3], cl
00415132  .  A1 10F64600   mov     eax, ds:[46F610]
00415137  .  FE80 B9000000 inc     byte ptr ds:[eax+B9]
0041513D  >  8B15 E0F5460( mov     edx, ds:[46F5E0]
00415143  .  52            push    edx
00415144  .  E8 E7150200   call    00436730                 EFCM.00436730
00415149  .  83C4 04       add     esp, 4
0041514C  .  85C0          test    eax, eax
0041514E  .  74 7F         je      short 004151CF           no salta
00415150  .  A1 10F64600   mov     eax, ds:[46F610]
00415155  .  83B8 EC160000 cmp     dword ptr ds:[eax+16EC], 0
0041515C  .  74 71         je      short 004151CF           no salta
```

Entro y veo esto:

```
00436730  $  8B4424 04     mov     eax, ss:[esp+4]
00436734  .  85C0          test    eax, eax
00436736  .  74 08         je      short 00436740           EFCM.00436740
00436738  .  50            push    eax                      hWnd
00436739  .  FF15 F4034600 call    ds:[4603F4]              IsWindow
0043673F  .  C3            retn
00436740  >  33C0          xor     eax, eax
00436742  .  C3            retn
00436743     CC            int3
```

```
0012F278  0013085A  hWnd = 0013085A ('EF CheckSum Manager',class='EFCM')
0012F27C  00415149  RETURN to EFCM.00415149 from EFCM.00436730
```

O sea que va a buscar la ventana de la aplicación y como la va a encontrar no se van a realizar los saltos que hay debajo del CALL de 415144 así que sigo y llego aquí:

```
00415164  .  68 00010000   push    100
00415169  .  8D8D F0FDFFF  lea     ecx, [local.132]
0041516F  .  51            push    ecx
00415170  .  52            push    edx
00415171  .  E8 CA130200   call    00436540                    EFCM.00436540
00415176  .  A1 10F64600   mov     eax, ds:[46F610]
0041517B  .  8BB0 EC16000  mov     esi, ds:[eax+16EC]
```

Entro y veo esto:

```
0043653F     CC            int3
00436540  $  8B4424 04     mov     eax, ss:[esp+4]
00436544  .  85C0          test    eax, eax
00436546  .v 74 12         je      short 0043655A              EFCM.0043655A
00436548  .  8B4C24 0C     mov     ecx, ss:[esp+C]
0043654C  .  8B5424 08     mov     edx, ss:[esp+8]
00436550  .  51            push    ecx                         Count
00436551  .  52            push    edx                         Buffer
00436552  .  50            push    eax                         hWnd
00436553  .  FF15 D8034600 call    ds:[4603D8]                 GetWindowTextW
00436559  .  C3            retn
0043655A  >  8B4424 08     mov     eax, ss:[esp+8]
0043655E  .  85C0          test    eax, eax
00436560  .v 74 05         je      short 00436567              EFCM.00436567
00436562  .  33C9          xor     ecx, ecx
00436564  .  66:8908       mov     ds:[eax], cx
00436567  >  33C0          xor     eax, eax
00436569  L. C3            retn
```

Lo que hace es obtener el título de la ventana así que salgo de la función y sigo traceando y llego aquí:

```
00415171  .  E8 CA130200    call    00436540                 EFCM.00436540
00415176  .  A1 10F64600    mov     eax, ds:[46F610]
0041517B  .  8BB0 EC16000(  mov     esi, ds:[eax+16EC]
00415181  .  83C4 0C        add     esp, 0C
00415184  .  8BCE           mov     ecx, esi
00415186  .  8D85 F0FDFFF(  lea     eax, [local.132]
0041518C  .  8D6424 00      lea     esp, ss:[esp]
00415190  >  66:8B10        mov     dx, ds:[eax]
00415193  .  66:3B11        cmp     dx, ds:[ecx]
00415196  .v 75 1E          jnz     short 004151B6           EFCM.004151B6
00415198  .  66:85D2        test    dx, dx
0041519B  .v 74 15          je      short 004151B2           EFCM.004151B2
0041519D  .  66:8B50 02     mov     dx, ds:[eax+2]
004151A1  .  66:3B51 02     cmp     dx, ds:[ecx+2]
004151A5  .v 75 0F          jnz     short 004151B6           EFCM.004151B6
004151A7  .  83C0 04        add     eax, 4
004151AA  .  83C1 04        add     ecx, 4
004151AD  .  66:85D2        test    dx, dx
004151B0  .^ 75 DE          jnz     short 00415190           EFCM.00415190
004151B2  >  33C0           xor     eax, eax
004151B4  .v EB 05          jmp     short 004151BB           EFCM.004151BB
004151B6  >  1BC0           sbb     eax, eax
004151B8  .  83D8 FF        sbb     eax, -1
004151BB  >  85C0           test    eax, eax
004151BD  .v 74 10          je      short 004151CF           salta
004151BF  .  8B0D E0F5460(  mov     ecx, ds:[46F5E0]
```

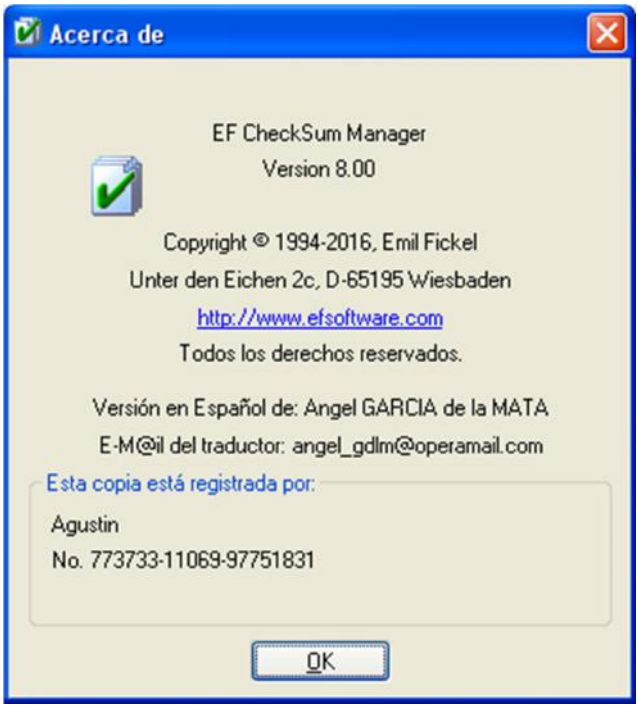
Lo que hace ahí en ese bucle es comparar el título de la ventana con el título que tendría que tener si está registrado y en este caso, como no pone nada de NOT REGISTERED coinciden y pasamos ese control.
Si llegásemos aquí:

```
004151B6  >  1BC0           sbb     eax, eax
004151B8  .  83D8 FF        sbb     eax, -1
004151BB  >  85C0           test    eax, eax
004151BD  .v 74 10          je      short 004151CF           salta
004151BF  .  8B0D E0F5460(  mov     ecx, ds:[46F5E0]
004151C5  .  56             push    esi
004151C6  .  51             push    ecx
004151C7  .  E8 D4120200    call    004364A0                 EFCM.004364A0
004151CC  .  83C4 08        add     esp, 8
004151CF  >  8B4D F4        mov     ecx, [local.3]
```

Y no saltase entraríamos en ese CALL:

```
0043649F     CC             int3
004364A0  r$ 8B4424 04      mov     eax, ss:[esp+4]
004364A4  .  85C0           test    eax, eax
004364A6  .v 74 0C          je      short 004364B4           EFCM.004364B4
004364A8  .  8B4C24 08      mov     ecx, ss:[esp+8]
004364AC  .  51             push    ecx                      Text
004364AD  .  50             push    eax                      hWnd
004364AE  .  FF15 D0034600( call    ds:[4603D0]              SetWindowTextW
004364B4  L> C3             retn
004364B5     CC             int3
```

Ahí realmente lo que hace es cambiar el título de la ventana con lo que nos viene fenomenal que no entre ya que no queremos que cambie el título. En este caso, como se puede ver, pasamos el control y el salto se cumple con lo que este código no se va a ejecutar. Sigo traceando y ya salgo de la función y llego a la parte donde se controla el mensaje WM_TIMER y no hay más código de importancia. Doy a F9 y…



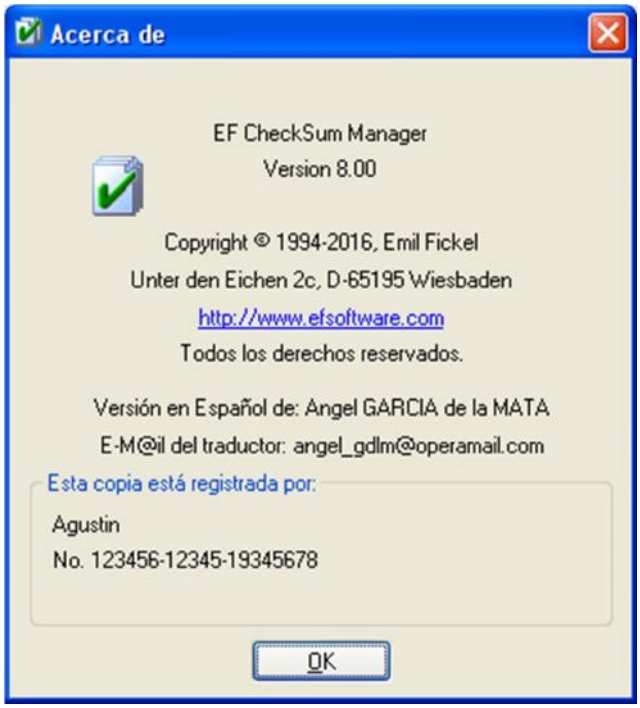
Hay algo extraño ya que el serial tendría que ser:

“773733-11069-90751831”

Y muestra:

“773733-11069-97751831”

Pero la verdad es que no sé porque pasa eso pero solo pasa si el nombre no es el adecuado para el serial.
Esto de momento se queda así y ahora veamos si se traga cualquier serial así que cambio el serial en el .LIC y vuelvo a arrancar haciendo todos los cambios que he visto hasta ahora:



El serial que puse es:

“123456-12345-12345678”

Y aparece:

“123456-12345-19345678”

O sea que suma 7 al carácter del serial que está mal y es exactamente lo que sumaba en el caso anterior.
Esta parte me tenía loco así que al final decidí iniciar Olly y repetir todo el proceso nuevamente, y mientras iba analizando, vi que, donde comparaba el serial sin guiones con la cadena esa que no sé de donde sale, usaba una función similar a strncmp de C:

```
0040B18F  .  6A 13          push    13
0040B191  .  8D5424 10      lea     edx, ss:[esp+10]
0040B195  .  52             push    edx
0040B196  .  81C6 460A000(  add     esi, 0A46
0040B19C  .  56             push    esi
0040B19D  .  E8 17F60300    call    0044A7B9                EFCM.0044A7B9
0040B1A2  .  83C4 14        add     esp, 14
0040B1A5  .  85C0           test    eax, eax
0040B1A7  .  EB 1F          jmp     short 0040B1C8          EFCM.0040B1C8
0040B1A9  .  8B0D 10F6460(  mov     ecx, ds:[46F610]
0040B1AF  .  33C0           xor     eax, eax
```

En la parte donde trabajaba con nuestro nombre y obtenía una cadena con __snprintf y luego comparaba esos caracteres con la cadena que no sé de dónde la saca:

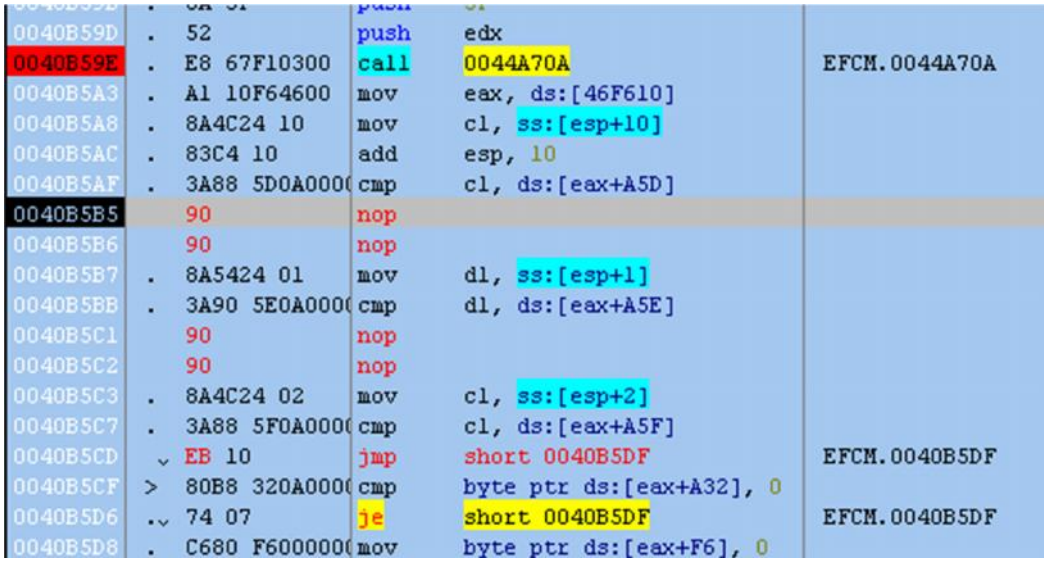
```
004150E9  .  68 04A04600   push    46A004                   ASCII "%05lu"
004150EE  .  8D45 00       lea     eax, ss:[ebp]
004150F1  .  6A 7F         push    7F
004150F3  .  50            push    eax
004150F4  .  E8 11560300   call    0044A70A                 EFCM.0044A70A
004150F9  .  A1 10F64600   mov     eax, ds:[46F610]
004150FE  .  8A4D 01       mov     cl, ss:[ebp+1]
00415101  .  83C4 10       add     esp, 10
00415104  .  3A88 310A000( cmp     cl, ds:[eax+A31]
0041510A     90            nop                              no salta
0041510B     90            nop
0041510C     8A55 02       mov     dl, ss:[ebp+2]
0041510F     3A90 320A000( cmp     dl, ds:[eax+A32]
00415115     90            nop                              no salta
00415116     90            nop
00415117     8A4D 03       mov     cl, ss:[ebp+3]
```

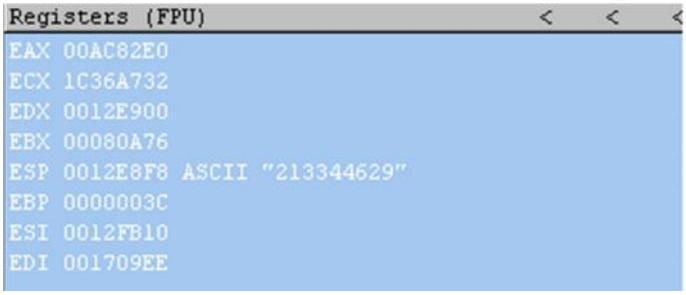
¿Y si la usa para más comprobaciones? Pongo un BP en todas las referencias de ambas y voy viendo que compara y que convierte y al final llego aquí:

```
0040B592  .  68 FC9F4600   push    469FFC                   ASCII "%04lu"
0040B597  .  8D5424 08     lea     edx, ss:[esp+8]
0040B59B  .  6A 3F         push    3F
0040B59D  .  52            push    edx
0040B59E  .  E8 67F10300   call    0044A70A                 EFCM.0044A70A
0040B5A3  .  A1 10F64600   mov     eax, ds:[46F610]
0040B5A8  .  8A4C24 10     mov     cl, ss:[esp+10]
0040B5AC  .  83C4 10       add     esp, 10
0040B5AF  .  3A88 5D0A000( cmp     cl, ds:[eax+A5D]
0040B5B5  .v 75 18         jnz     short 0040B5CF           EFCM.0040B5CF
0040B5B7  .  8A5424 01     mov     dl, ss:[esp+1]
0040B5BB  .  3A90 5E0A000( cmp     dl, ds:[eax+A5E]
0040B5C1  .v 75 0C         jnz     short 0040B5CF           EFCM.0040B5CF
0040B5C3  .  8A4C24 02     mov     cl, ss:[esp+2]
0040B5C7  .  3A88 5F0A000( cmp     cl, ds:[eax+A5F]
0040B5CD  .v 74 10         je      short 0040B5DF           EFCM.0040B5DF
0040B5CF  >  80B8 320A000( cmp     byte ptr ds:[eax+A32], 0
0040B5D6  .v 74 07         je      short 0040B5DF           EFCM.0040B5DF
0040B5D8  .  C680 F600000( mov     byte ptr ds:[eax+F6], 0
0040B5DF  >  8B4C24 40     mov     ecx, ss:[esp+40]
0040B5E3  .  33CC          xor     ecx, esp
0040B5E5  .  E8 93F00300   call    0044A67D                 EFCM.0044A67D
```

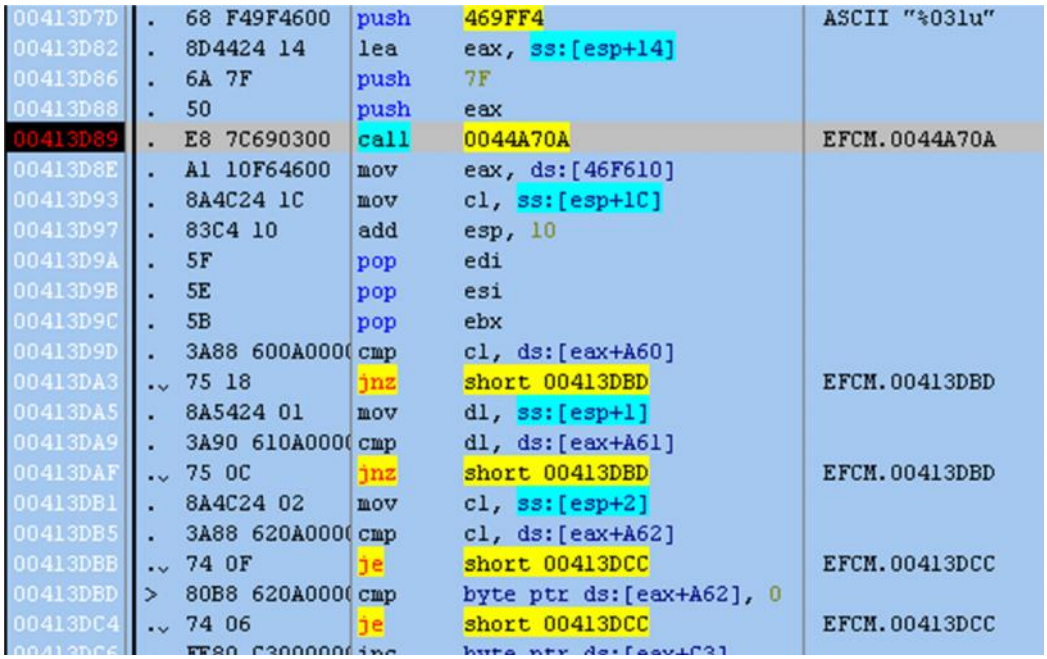
```
Registers (FPU)                          <    <    <
EAX 00000015
ECX 0CB76175
EDX 0012E8F8 ASCII "123456-12345-12345678"
EBX 00080A76
ESP 0012E8E8
EBP 0000003C
ESI 0012FB10
EDI 001709EE
```

Cambio los saltos para que cuele aunque no coincidan:

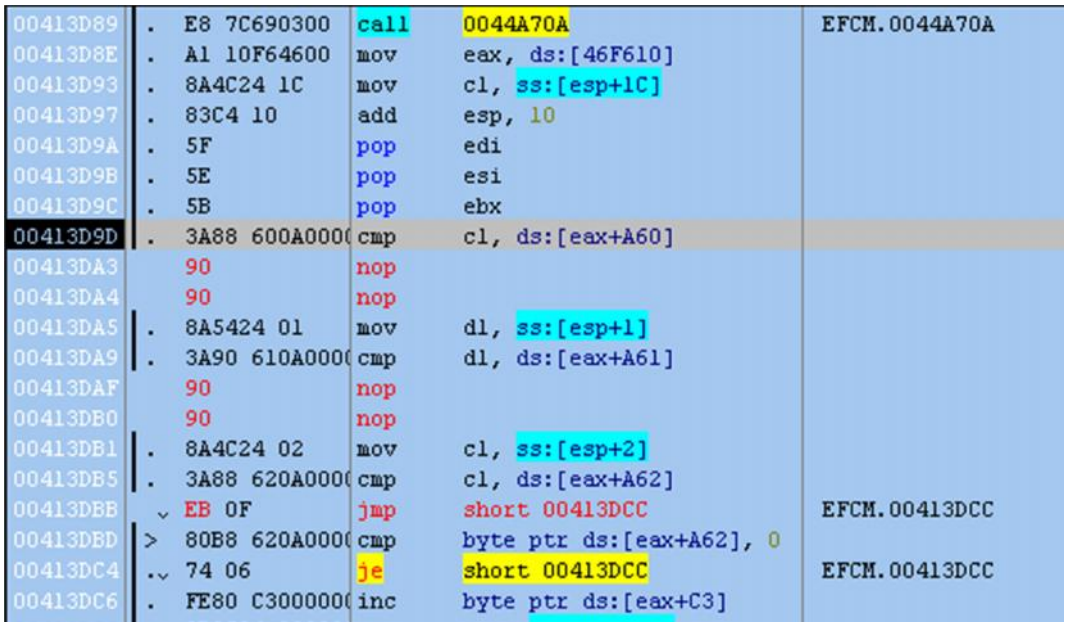




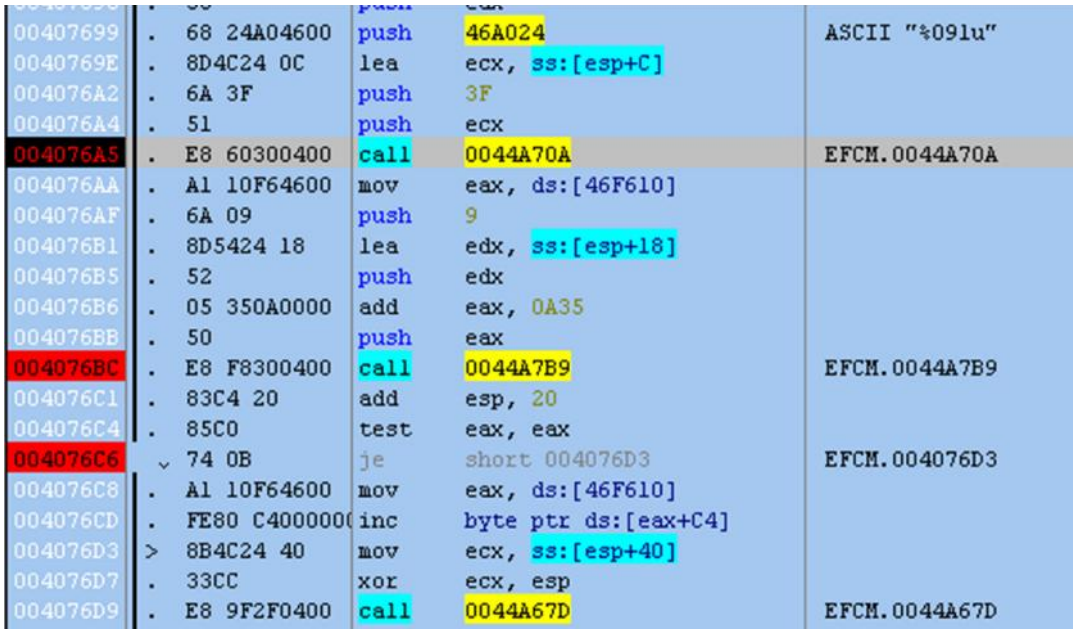
Compara esos caracteres con los de la cadena misteriosa y pasamos esta comprobación. Quito el BP, doy a F9 y vuelvo a parar en otro sitio:



Es el mismo caso de antes con otra parte de la cadena misteriosa así que realizo los cambios oportunos:



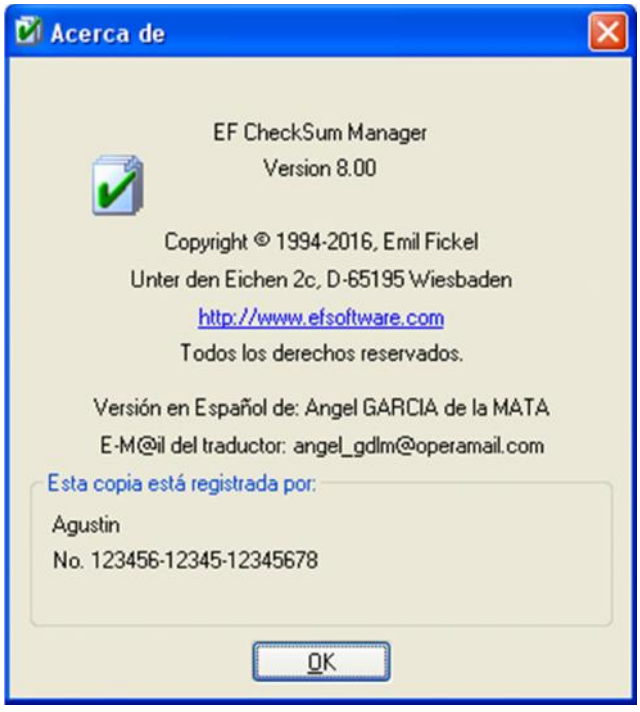
Quito el BP y doy a F9:



Ahí tenemos los dos CALLs, o sea, primero crea una cadena, luego usa la otra para comparar entre dos cadenas y después hay un salto condicional así que cambio el condicional por un salto incondicional, quito el BP y doy a F9:

```
00407699  .  68 24A04600    push    46A024                   ASCII "%09lu"
0040769E  .  8D4C24 0C      lea     ecx, ss:[esp+C]
004076A2  .  6A 3F          push    3F
004076A4  .  51             push    ecx
004076A5  .  E8 60300400    call    0044A70A                 EFCM.0044A70A
004076AA  .  A1 10F64600    mov     eax, ds:[46F610]
004076AF  .  6A 09          push    9
004076B1  .  8D5424 18      lea     edx, ss:[esp+18]
004076B5  .  52             push    edx
004076B6  .  05 350A0000    add     eax, 0A35
004076BB  .  50             push    eax
004076BC  .  E8 F8300400    call    0044A7B9                 EFCM.0044A7B9
004076C1  .  83C4 20        add     esp, 20
004076C4  .  85C0           test    eax, eax
004076C6     EB 0B          jmp     short 004076D3           EFCM.004076D3
004076C8  .  A1 10F64600    mov     eax, ds:[46F610]
004076CD  .  FE80 C4000000  inc     byte ptr ds:[eax+C4]
004076D3  >  8B4C24 40      mov     ecx, ss:[esp+40]
004076D7  .  33CC           xor     ecx, esp
004076D9  .  E8 9F2F0400    call    0044A67D                 EFCM.0044A67D
```

Ya no para más en ninguno así que abro el About a ver que veo:



Y al cerrar el About vuelve a parar:

```
0041B847  .  68 EC9F4600     push    469FEC                  ASCII "%02lu"
0041B84C  .  8D5424 4C       lea     edx, ss:[esp+4C]
0041B850  .  6A 3F           push    3F
0041B852  .  52              push    edx
0041B853  .  E8 B2EE0200     call    0044A70A                EFCM.0044A70A
0041B858  .  A1 10F64600     mov     eax, ds:[46F610]
0041B85D  .  8A4C24 54       mov     cl, ss:[esp+54]
0041B861  .  83C4 10         add     esp, 10
0041B864  .  5F              pop     edi
0041B865  .  5E              pop     esi
0041B866  .  3A88 300A000(   cmp     cl, ds:[eax+A30]
0041B86C  .v 74 06           je      short 0041B874          EFCM.0041B874
0041B86E  .  FE88 B6000000(  dec     byte ptr ds:[eax+B6]
0041B874  >  8B4C24 7C       mov     ecx, ss:[esp+7C]
0041B878  .  33CC            xor     ecx, esp
0041B87A  .  E8 FEED0200     call    0044A67D                EFCM.0044A67D
```

Compara el primer carácter de la cadena obtenida en ese CALL con el primer carácter de la cadena “mágica” y en este caso coinciden pero mejor asegurarse haciendo que ese condicional salte siempre ya que no sé de qué depende la creación de ambas cadenas y podría ser que cambiasen:

```
0041B837  .  B8 1F85EB51     mov     eax, 51EB851F
0041B83C  .  F7E1            mul     ecx
0041B83E  .  C1EA 05         shr     edx, 5
0041B841  .  6BD2 64         imul    edx, edx, 64
0041B844  .  2BCA            sub     ecx, edx
0041B846  .  51              push    ecx
0041B847  .  68 EC9F4600     push    469FEC                  ASCII "%02lu"
0041B84C  .  8D5424 4C       lea     edx, ss:[esp+4C]
0041B850  .  6A 3F           push    3F
0041B852  .  52              push    edx
0041B853  .  E8 B2EE0200     call    0044A70A                EFCM.0044A70A
0041B858  .  A1 10F64600     mov     eax, ds:[46F610]
0041B85D  .  8A4C24 54       mov     cl, ss:[esp+54]
0041B861  .  83C4 10         add     esp, 10
0041B864  .  5F              pop     edi
0041B865  .  5E              pop     esi
0041B866  .  3A88 300A000(   cmp     cl, ds:[eax+A30]
0041B86C     EB 06           jmp     short 0041B874          EFCM.0041B874
0041B86E  .  FE88 B6000000(  dec     byte ptr ds:[eax+B6]
0041B874  >  8B4C24 7C       mov     ecx, ss:[esp+7C]
0041B878  .  33CC            xor     ecx, esp
```

Quito el BP y doy a F9 y ya no vuelve a parar más en ningún sitio.

A por el Loader Debugger

Ahora solo nos queda crear un loader debugger que pare en algún sitio donde ya tengamos todo el código desempacado en memoria y podamos realizar las modificaciones. Tengo dos opciones que suelo usar:
1. La clase TDebugger de Guan
2. El componente DbgCLS de stzwei

Decir que no sé por qué pero ambas opciones me dieron muchísimos problemas con los BPs de todos los tipos. La clase TDebugger no para en los HBPs, ni en los BPM, ni en los BPs ni siquiera en las APIs con lo que intenté con el componente de stzwei y tampoco me para ni en los HBPs, ni en los BPM pero si me para en los BPs en las APIs con lo que después de varias pruebas descubrí que al llamar por primera vez a SetTimer aún no ha hecho ninguna comprobación y ya tiene todo el código listo para poder parchearlo con lo que opté por esa opción.
Así quedó el código:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "DbgCLS"
#pragma link "trayicon"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{

}
//---------------------------------------------------------------------------
void __fastcall TForm1::DbgCLS1BreakBP()
{
    if(DbgCLS1->GetLastAddr() == DbgCLS1->GetAddrAPI("user32.dll","SetTimer"))
    {
        //Este salto evita la comparacion entre el serial bueno y el malo
        DbgCLS1->WriteMemoryB(0x40B1A7,0xEB);

        //Inicio de cambios que permiten que usemos cualquier Serial
        DbgCLS1->WriteMemoryB(0x40B5B5,0x90);
        DbgCLS1->WriteMemoryB(0x40B5B6,0x90);

        DbgCLS1->WriteMemoryB(0x40B5C1,0x90);
        DbgCLS1->WriteMemoryB(0x40B5C2,0x90);

        DbgCLS1->WriteMemoryB(0x40B5CD,0xEB);

        DbgCLS1->WriteMemoryB(0x413DA3,0x90);
        DbgCLS1->WriteMemoryB(0x413DA4,0x90);

        DbgCLS1->WriteMemoryB(0x413DAF,0x90);
        DbgCLS1->WriteMemoryB(0x413DB0,0x90);

        DbgCLS1->WriteMemoryB(0x413DBB,0xEB);

        DbgCLS1->WriteMemoryB(0x4076C6,0xEB);
```

```
        //Fin de cambios que permiten que usemos cualquier Serial

        //Inicio de cambios que permiten que usemos cualquier nombre
        DbgCLS1->WriteMemoryB(0x41510A,0x90);
        DbgCLS1->WriteMemoryB(0x41510B,0x90);

        DbgCLS1->WriteMemoryB(0x415115,0x90);
        DbgCLS1->WriteMemoryB(0x415116,0x90);

        DbgCLS1->WriteMemoryB(0x415120,0xEB);
        //Fin de cambios que permiten que usemos cualquier nombre

        //Inicio de cambios que evitan que nos detecte el serial en la lista negra
        DbgCLS1->WriteMemoryB(0x41D85B,0xEB);

        DbgCLS1->WriteMemoryB(0x41D87E,0x90);
        DbgCLS1->WriteMemoryB(0x41D87F,0x90);

        DbgCLS1->WriteMemoryB(0x41D88F,0x90);
        DbgCLS1->WriteMemoryB(0x41D890,0x90);
        DbgCLS1->WriteMemoryB(0x41D891,0x90);
        DbgCLS1->WriteMemoryB(0x41D892,0x90);
        DbgCLS1->WriteMemoryB(0x41D893,0x90);
        DbgCLS1->WriteMemoryB(0x41D894,0x90);

        DbgCLS1->WriteMemoryB(0x41D8D4,0xE9);
        DbgCLS1->WriteMemoryB(0x41D8D5,0xB8);
        DbgCLS1->WriteMemoryB(0x41D8D6,0x00);
        DbgCLS1->WriteMemoryB(0x41D8D7,0x00);
        DbgCLS1->WriteMemoryB(0x41D8D8,0x00);
        DbgCLS1->WriteMemoryB(0x41D8D9,0x90);

        DbgCLS1->WriteMemoryB(0x41D998,0x90);
        DbgCLS1->WriteMemoryB(0x41D999,0x90);
        //Fin de cambios que evitan que nos detecte el serial en la lista negra

        DbgCLS1->WriteMemoryB(0x41B86C,0xEB);

        //Quito el BP en la api elegida porque ya se hicieron todas las
        //modificaciones y no lo necesitamos ya para nada
        DbgCLS1->ClearBP(DbgCLS1->GetAddrAPI("user32.dll","SetTimer"));
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormActivate(TObject *Sender)
{
    TrayIcon1->Minimize();
    if(!FileExists("EFCM.EXE")){
        ShowMessage("No se encuentra el archivo EFCM.EXE.\nColoca este ejecutable
junto al archivo EFCM.EXE y vuelve a intentarlo.");
    }else{
        DbgCLS1->InitDebug();
        DbgCLS1->DebugLoop();
    }
    Close();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::DbgCLS1BreakSystemBP()
{
```

```
    DbgCLS1->SetApiBP("user32.dll","SetTimer");
}
//---------------------------------------------------------------------------
```

Con esto ya tengo un loader que funciona perfectamente y me aparece como registrado con el nombre y serial que desee jejeje.

A por el Keygen

Después de todo esto se me ocurre analizar bien todo lo que hice para ver si puedo crear un keygen mejor que todo esto y lo primero es analizar que hace en cada sitio con _snprintf así que inicio y cuando pare en el HBP que teníamos en 41D815 voy analizando que hace:

```
0041D815   .  BF 889E4600   mov     edi, 469E88       ;  Mete esa dirección constante en
EDI
0041D81A   .  74 6C         je      short 0041D888    ;  EFCM.0041D888
0041D81C   .  BE 349F4600   mov     esi, 469F34       ;  Aquí inicializa ESI para trabajar
con la tabla que contiene los valores de la lista negra
0041D821   >  8B46 04       mov     eax, ds:[esi+4]   ;  El valor de esta dirección será
para la tercera parte del serial
0041D824   .  8B0E          mov     ecx, ds:[esi]     ;  El valor de esta dirección será
para la segunda parte del serial
0041D826   .  8B56 FC       mov     edx, ds:[esi-4]   ;  El valor de esta dirección será
para la primera parte del serial
0041D829   .  05 6060C701   add     eax, 1C76060      ;  Le suma esta constante al valor
de la tercera parte
0041D82E   .  50            push    eax               ;  Lo mete en la pila como parámetro
0041D82F   .  81C1 58040000 add     ecx, 458          ;  Le suma esta constante al segundo
valor del serial
0041D835   .  51            push    ecx               ;  Lo mete en la pila como parámetro
0041D836   .  81C2 6E0C0000 add     edx, 0C6E         ;  Le suma esta constante a la
primera parte del serial
0041D83C   .  52            push    edx               ;  Lo mete en la pila como parámetro
0041D83D   .  68 2CA04600   push    46A02C            ;  ASCII "%06lu-%05lu-%08lu" //Es la
máscara con la que saldrá el serial
0041D842   .  8D4424 1C     lea     eax, ss:[esp+1C]  ;  Obtiene la dirección del buffer
donde se guardará el serial obtenido
0041D846   .  6A 3F         push    3F                ;  Mete en la pila como parámetro el
tamaño máximo del serial
0041D848   .  50            push    eax               ;  Mete en la pila como parámetro el
puntero al buffer donde se guardará el serial
0041D849   .  E8 BCCE0200   call    0044A70A          ;  EFCM.0044A70A
```

Y aquí tenemos toda la tabla donde están los valores para estos seriales:

```
00469E84              B9 E2 0D 00 69 08 00 00 34 AD A1 03      ¹â..i..4- ¡
00469E94  F7 C1 0B 00 E5 26 00 00 F7 62 A1 03 4F BB 0D 00  ÷Á.å&..÷b¡ O».. 
00469EA4  7A 2F 00 00 34 AD A1 03 E9 1A 0F 00 8B 56 00 00  z/..4-¡é.‹V..
00469EB4  34 AD A1 03 4B FD 0A 00 30 67 01 00 1C 5F A1 03  4- ¡Ký..0g._¡
00469EC4  02 71 0B 00 E7 A3 00 00 15 5F A1 03 AD AD 0A 00  q.ç£.._¡- - ..
00469ED4  D6 78 00 00 BF E5 A2 03 AB D6 0A 00 E7 4D 00 00  Öx..¿å¢«Ö..çM..
00469EE4  B1 E5 A2 03 B1 1F 0D 00 14 00 00 00 23 86 A1 03  ±å¢±.....#†¡
00469EF4  46 26 0B 00 81 84 00 00 3F 5F A1 03 A1 0F 0C 00  F&.□ „..?_¡¡..
00469F04  21 15 01 00 0F 5F A1 03 BC 72 0B 00 C3 7C 00 00  !._¡ ¼r.Ã|..
00469F14  11 5F A1 03 00 00 00 00 00 00 00 00 00 00 00 00  _¡............
00469F24  25 30 32 6C 75 25 30 32 6C 75 00 00 38 BF 0B 00  %02lu%02lu..8¿.
00469F34  1B A0 00 00 B5 E5 A2 03 25 BC 04 00 FF F3 00 00   ..µå¢%¼.ÿó..
00469F44  FF 9A A0 03 B0 FD 0A 00 BE 2A 00 00 1B 5F A1 03  ÿš °ý..¾*..._¡
00469F54  A8 CD 0E 00 D7 15 01 00 32 AD A1 03 A2 A6 0E 00  ¨Í.×.2- ¡¢¦.
00469F64  DE 3C 01 00 32 AD A1 03 6C 57 0E 00 D2 C3 00 00  Þ<.2- ¡lW.ÒÃ..
00469F74  35 AD A1 03 9C 7F 0E 00 EF 63 01 00 32 AD A1 03  5-¡œ .ïc.2-¡
00469F84  EE 23 0B 00 27 15 01 00 13 5F A1 03 28 9A 0B 00  î#.'._¡ (š.
00469F94  C0 51 00 00 F9 62 A1 03 F0 75 00 00 C5 04 00 00  ÀQ..ùb¡ ðu..Å..
00469FA4  42 D4 A1 03 4B 38 01 00 F2 27 00 00 44 E6 1B 04  BÔ¡ K8.ò'..Dæ
00469FB4  10 95 0D 00 F9 2E 00 00 22 98 1B 04 EA 4E 00 00  •..ù..."˜êN..
00469FC4  D6 2B 00 00 42 D4 A1 03 0E 39 01 00 1A C8 00 00  Ö+..BÔ¡ 9.È..
```

```
00469FD4  42 D4 A1 03                                      BÔ¡
```

Todo esto tiene pinta de truco para ocultar la lista negra de ojos indiscretos jejeje.
Si seguimos llegamos aquí:

```
0041D895   > /8B4F 08       mov     ecx, ds:[edi+8]   ;  obtiene el valor de la tabla para
la tercera parte del serial
0041D898   . |8B57 04       mov     edx, ds:[edi+4]   ;  obtiene el valor de la tabla para
la segunda parte del serial
0041D89B   . |8B07          mov     eax, ds:[edi]     ;  obtiene el valor de la tabla para
la primera parte del serial
0041D89D   . |81C1 6060C701 add     ecx, 1C76060      ;  Le suma la constante al valor
para la tercera parte del serial
0041D8A3   . |51            push    ecx               ;  Lo mete en la pila como parámetro
0041D8A4   . |81C2 58040000 add     edx, 458          ;  Le suma la constante al valor
para la segunda parte del serial
0041D8AA   . |52            push    edx               ;  Lo mete en la pila como parámetro
0041D8AB   . |05 6E0C0000   add     eax, 0C6E         ;  Le suma la constante al valor
para la primera parte del serial
0041D8B0   . |50            push    eax               ;  Lo mete en la pila como parámetro
0041D8B1   . |68 2CA04600   push    46A02C            ;  ASCII "%06lu-%05lu-%08lu"
0041D8B6   . |8D4C24 1C     lea     ecx, ss:[esp+1C]  ;  Obtiene la dirección del buffer
donde se guardará el serial obtenido
0041D8BA   . |6A 3F         push    3F                ;  Mete en la pila como parámetro el
tamaño maximo del serial
0041D8BC   . |51            push    ecx               ;  Mete en la pila como parámetro el
puntero al buffer donde se guardará el serial
0041D8BD   . |E8 48CE0200   call    0044A70A          ;  EFCM.0044A70A
```

Es casi idéntico al código anterior pero con la diferencia de que coge los valores de la misma tabla pero en diferente orden y de diferente forma. En realidad todo esto no nos interesa mucho ya que no nos aporta nada para crear el keygen aunque es bueno ver cómo trabaja.

El siguiente sitio donde para es este:

```
00401C42  |.  50            push    eax               ;  Mete en la pila el valor
constante 0x2266
00401C43  |.  68 FC9F4600   push    469FFC            ;  ASCII "%04lu" Mascara de salida
00401C48  |.  8D4C24 18     lea     ecx, ss:[esp+18]  ;  Obtiene la direccion del buffer
00401C4C  |.  6A 3F         push    3F                ;  Mete en la pila el tamaño maximo
del buffer
00401C4E  |.  51            push    ecx               ;  Mete en la pila el puntero del
buffer
00401C4F  |.  E8 B68A0400   call    0044A70A          ;  EFCM.0044A70A
```

Da como resultado 0x8806 y lo compara con los siguientes caracteres de la cadena misteriosa:

```
00AC8D09                        39 37 33 33 33 32 36 34 37          973332647
00AC8D19  36 35 34 39 32 31 30 32 30 30 37 30 30 37 37 33  6549210200700773
00AC8D29  37 33 33 31 31 30 36 39 39 30 37 35 31 38 33 31  7331106990751831
00AC8D39  38 38 30 36 32 33 31 30 31 36 30 31 31 33 39 33  8806231016011393
```

Coincide pero de momento no sé de dónde sale esa cadena que usa para todas las verificaciones.

En la siguiente parte que para es esta:

```
0040B580   >  0FB67404 04   movzx   esi, byte ptr ss:[esp+ea> ; Coje el valor hexa del
caracter serial[contador]
0040B585   . |03F1          add     esi, ecx                  ; suma = suma +
serial[contador]
0040B587   . |8D4C70 01     lea     ecx, ds:[eax+esi*2+1]     ; total=contador +(suma*2)+1
0040B58B   . |40             inc     eax                       ; contador++
0040B58C   . |3BC2           cmp     eax, edx                  ; sale si contador es mayor
que 0x15 que es el largo del serial
0040B58E   .^\72 F0          jb      short 0040B580            ; EFCM.0040B580
0040B590   .  5E             pop     esi
0040B591   >  51             push    ecx                       ; mete en la pila como
parametro el valor obtenido
0040B592   .  68 FC9F4600    push    469FFC                    ; ASCII "%04lu"
0040B597   .  8D5424 08      lea     edx, ss:[esp+8]           ; Obtiene la direccion del
buffer
0040B59B   .  6A 3F          push    3F                        ; Mete en la pila el tamaño
maximo del buffer
0040B59D   .  52             push    edx                       ; Mete en la pila como
parametro el puntero al buffer
0040B59E   .  E8 67F10300    call    0044A70A                  ; EFCM.0044A70A
```

Compara la cadena obtenida con esta parte de la cadena de siempre:

```
00AC8D09                      39 37 33 33 33 32 36 34 37         973332647
00AC8D19  36 35 34 39 32 31 30 32 30 30 37 30 30 37 37 33  6549210200700773
00AC8D29  37 33 33 31 31 30 36 39 39 30 37 35 31 38 33 31  7331106990751831
00AC8D39  38 38 30 36 32 33 31 30 31 36 30 31 31 33 39 33  8806231016011393
```

En este caso no coincide ya que el serial es uno cualquiera.

En el siguiente sitio que para es aquí:

```
0041B810  |> /0FB67C04 08   /movzx   edi, byte ptr ss:[esp+eax+8]
0041B815  |. |8D4C79 2C     |lea     ecx, ds:[ecx+edi*2+2C]
0041B819  |. |0FB67C04 09   |movzx   edi, byte ptr ss:[esp+eax+9]
0041B81E  |. |8D547A 2C     |lea     edx, ds:[edx+edi*2+2C]
0041B822  |. |0FB67C04 0A   |movzx   edi, byte ptr ss:[esp+eax+A]
0041B827  |. |83C0 03       |add     eax, 3
0041B82A  |. |83F8 39       |cmp     eax, 39
0041B82D  |. |8D747E 2C     |lea     esi, ds:[esi+edi*2+2C]
0041B831  |.^\7C DD         \jl      short 0041B810                ;  EFCM.0041B810
0041B833  |.  03F2          add     esi, edx
0041B835  |.  03CE          add     ecx, esi
0041B837  |.  B8 1F85EB51   mov     eax, 51EB851F
0041B83C  |.  F7E1          mul     ecx
0041B83E  |.  C1EA 05       shr     edx, 5
0041B841  |.  6BD2 64       imul    edx, edx, 64
0041B844  |.  2BCA          sub     ecx, edx
0041B846  |.  51            push    ecx
0041B847  |.  68 EC9F4600   push    469FEC                        ;  ASCII "%02lu"
0041B84C  |.  8D5424 4C     lea     edx, ss:[esp+4C]
0041B850  |.  6A 3F         push    3F
0041B852  |.  52            push    edx
0041B853  |.  E8 B2EE0200   call    0044A70A                      ;  EFCM.0044A70A
```

Ahí lo que hace es coger la cadena de siempre y va haciendo operaciones con ella pero en este caso la cadena de siempre es diferente, el primer carácter en vez de ser 0x39 es 0x20:

```
0012F4B1                                                  20
0012F4C1  37 33 33 33 32 36 34 37 36 35 34 39 32 31 30 32  7333264765492102
0012F4D1  30 30 37 30 30 37 37 33 37 33 33 31 31 30 36 39  0070077373311069
0012F4E1  39 30 37 35 31 38 33 31 38 38 30 36 32 33 31 30  9075183188062310
0012F4F1  31 36 30 31 31 33 39 33                          16011393
```

Y con el valor obtenido vuelve a obtener una cadena a comparar con la cadena de siempre pero esta si tiene el 9 en la primera posición y lo compara con este:

```
00AC8D09                        39 37 33 33 33 32 36 34 37        973332647
00AC8D19  36 35 34 39 32 31 30 32 30 30 37 30 30 37 37 33  6549210200700773
00AC8D29  37 33 33 31 31 30 36 39 39 30 37 35 31 38 33 31  7331106990751831
00AC8D39  38 38 30 36 32 33 31 30 31 36 30 31 31 33 39 33  8806231016011393
```

En este caso obtiene la cadena “99” y solo compara el primer carácter así que lo pasa sin problemas.

En el siguiente lugar que para vemos esto:

```
004150C7  |> /8A08          /mov     cl, ds:[eax]
004150C9  |. |40            |inc     eax
004150CA  |. |84C9          |test    cl, cl
004150CC  |.^\75 F9         \jnz     short 004150C7             ;  En este bucle obtiene
el largo del nombre
004150CE  |.  2BC2          sub     eax, edx
004150D0  |.  33C9          xor     ecx, ecx
004150D2  |.  8D51 20       lea     edx, ds:[ecx+20]           ;  suma += 0x20 //empieza
valiendo 0
004150D5  |>  3BC8          /cmp     ecx, eax
004150D7  |.  73 0F         |jnb     short 004150E8             ;  Sale si ya se recorrio
todo el nombre
004150D9  |.  0FB6740D 00   |movzx   esi, byte ptr ss:[ebp+ecx] ;  Obtiene el caracter
nombre[contador]
004150DE  |.  8D1472        |lea     edx, ds:[edx+esi*2]        ;  total = suma +
(nombre[contador]*2)
004150E1  |.  8D5451 01     |lea     edx, ds:[ecx+edx*2+1]      ;  total = contador +
(total*2) + 1
004150E5  |.  41            |inc     ecx                        ;  contador++
004150E6  |.^ EB ED         \jmp     short 004150D5             ;  EFCM.004150D5
004150E8  |>  52            push    edx                        ;  Mete el resultado en la
pila como parametro
004150E9  |.  68 04A04600   push    46A004                     ;  ASCII "%05lu" //Mascara
004150EE  |.  8D45 00       lea     eax, ss:[ebp]              ;  Obtiene la direccion
del buffer
004150F1  |.  6A 7F         push    7F                         ;  Mete en la pila como
parametro el tamaño maximo para el buffer
004150F3  |.  50            push    eax                        ;  Mete en la pila como
parametro el puntero al buffer
004150F4  |.  E8 11560300   call    0044A70A                   ;  EFCM.0044A70A
```

Y los bytes que compara en la cadena de siempre son estos:

```
00AC8D09                        39 37 33 33 33 32 36 34 37        973332647
00AC8D19  36 35 34 39 32 31 30 32 30 30 37 30 30 37 37 33  6549210200700773
00AC8D29  37 33 33 31 31 30 36 39 39 30 37 35 31 38 33 31  7331106990751831
00AC8D39  38 38 30 36 32 33 31 30 31 36 30 31 31 33 39 33  8806231016011393
```

En este caso tampoco van a coincidir.

En el siguiente lugar donde para es este:

```
00413D28  |.  2BCA          sub     ecx, edx
00413D2A  |.  33C0          xor     eax, eax
00413D2C  |.  83F9 02       cmp     ecx, 2
00413D2F  |.  7C 26         jl      short 00413D57                  ;  En esta parte hace
las operaciones con la compañia
00413D31  |.  8D51 FF       lea     edx, ds:[ecx-1]
00413D34  |.  55            push    ebp
00413D35  |>  0FB66C04 10   /movzx   ebp, byte ptr ss:[esp+eax+10]
00413D3A  |.  8D2C68        |lea     ebp, ds:[eax+ebp*2]
00413D3D  |.  03E9          |add     ebp, ecx
00413D3F  |.  03DD          |add     ebx, ebp
00413D41  |.  0FB66C04 11   |movzx   ebp, byte ptr ss:[esp+eax+11]
00413D46  |.  8D2C68        |lea     ebp, ds:[eax+ebp*2]
00413D49  |.  03E9          |add     ebp, ecx
00413D4B  |.  83C0 02       |add     eax, 2
00413D4E  |.  8D742E 01     |lea     esi, ds:[esi+ebp+1]
00413D52  |.  3BC2          |cmp     eax, edx
00413D54  |.^ 72 DF         \jb      short 00413D35                  ;  EFCM.00413D35
00413D56  |.  5D            pop     ebp
00413D57  |>  3BC1          cmp     eax, ecx
00413D59  |.  73 0B         jnb     short 00413D66                  ;  EFCM.00413D66
00413D5B  |.  0FB65404 0C   movzx   edx, byte ptr ss:[esp+eax+C]    ;  edx=0
00413D60  |.  8D7C50 10     lea     edi, ds:[eax+edx*2+10]          ;  total = eax + (edx
* 2) + 0x10
00413D64  |.  03F9          add     edi, ecx                        ;  total = ecx
00413D66  |>  03F3          add     esi, ebx                        ;  esi += ebx
00413D68  |.  03FE          add     edi, esi                        ;  total += esi
00413D6A  |.  B8 5917B7D1   mov     eax, D1B71759
00413D6F  |.  F7E7          mul     edi                             ;  total *=
0xD1B71759
00413D71  |.  C1EA 0D       shr     edx, 0D                         ;  A la parte que
desborda de un DWORD le hace >> 0xD
00413D74  |.  69D2 10270000 imul    edx, edx, 2710                  ;  El resultado lo
multiplica por 0x2710
00413D7A  |.  2BFA          sub     edi, edx
00413D7C  |.  57            push    edi
00413D7D  |.  68 F49F4600   push    469FF4                          ;  ASCII "%03lu"
00413D82  |.  8D4424 14     lea     eax, ss:[esp+14]
00413D86  |.  6A 7F         push    7F
00413D88  |.  50            push    eax
00413D89  |.  E8 7C690300   call    0044A70A                        ;  EFCM.0044A70A
```

En el bucle de arriba lo que hace es coger la cadena del nombre de la compañía y hace unas operaciones con los caracteres impares menos el último y guarda el valor final en un registro. Hace lo mismo con los pares y guarda el resultado en otro registro, y por último, si el largo de la cadena era impar, hace otra serie de operaciones con el último carácter y guarda el resultado en un tercer registro y suma todo.
Si no hay cadena de compañía hace las operaciones de abajo pero el valor que usa es 0x10. Luego hace las operaciones que indico en los comentarios.
Esta parte al principio no terminaba de entenderla ya que siempre me daba 016, bueno, siempre hasta que me dio por poner una compañía y ahí la cosa cambió. Compara los siguientes caracteres de la cadena de siempre y en este caso sí coinciden:

```
00AC8D09                        39 37 33 33 33 32 36 34 37          973332647
00AC8D19  36 35 34 39 32 31 30 32 30 30 37 30 30 37 37 33  6549210200700773
00AC8D29  37 33 33 31 31 30 36 39 39 30 37 35 31 38 33 31  7331106990751831
00AC8D39  38 38 30 36 32 33 31 30 31 36 30 31 31 33 39 33  8806231016011393
```

Luego para aquí:

```
0040768C  |.  8B80 E4010000 mov     eax, ds:[eax+1E4]
00407692  |.  50            push    eax
00407693  |.  E8 78AEFFFF   call    00402510                        ;  EFCM.00402510
00407698  |.  50            push    eax
00407699  |.  68 24A04600   push    46A024                          ;  ASCII "%09lu"
0040769E  |.  8D4C24 0C     lea     ecx, ss:[esp+C]
004076A2  |.  6A 3F         push    3F
004076A4  |.  51            push    ecx
004076A5  |.  E8 60300400   call    0044A70A                        ;  EFCM.0044A70A
```

Si entro en el primer CALL tengo esto:

```
0040252F  |> \53            push    ebx                           ;  Calcula el tamaño del
archivo de licencia
00402530  |.  33C9          xor     ecx, ecx
00402532  |.  8BC7          mov     eax, edi
00402534  |.  33DB          xor     ebx, ebx
00402536  |.  894424 10     mov     ss:[esp+10], eax
0040253A  |.  85FF          test    edi, edi
0040253C  |.  76 45         jbe     short 00402583                ;  EFCM.00402583
0040253E  |.  55            push    ebp
0040253F  |.  8D6F 01       lea     ebp, ds:[edi+1]               ;  auxsizefilelic++
00402542  |>  0FB60431      /movzx   eax, byte ptr ds:[ecx+esi]   ;  Obtiene el caracter del
archivo buffer[contador]
00402546  |.  8D5408 03     |lea     edx, ds:[eax+ecx+3]          ;  valor =
buffer[contador] + contador + 3
0040254A  |.  33DA          |xor     ebx, edx                     ;  ebx = ebx ^
buffer[contador]
0040254C  |.  33D2          |xor     edx, edx                     ;  auxsizefilelic =
sizefilelic
0040254E  |.  8BC3          |mov     eax, ebx                     ;  auxsizefilelic++
00402550  |.  F7F5          |div     ebp                          ;  pos = valor %
auxsizefilelic
00402552  |.  83C2 02       |add     edx, 2                       ;  pos +=2
00402555  |.  3BD7          |cmp     edx, edi                     ;  if(pos >=
auxsizefilelic)
00402557  |.  76 05         |jbe     short 0040255E               ;  EFCM.0040255E
00402559  |.  BA 02000000   |mov     edx, 2                       ;  pos = 2
0040255E  |>  8B4424 14     |mov     eax, ss:[esp+14]
00402562  |.  0FB61432      |movzx   edx, byte ptr ds:[edx+esi]   ;  valor = buffer[pos]
00402566  |.  83C0 02       |add     eax, 2                       ;  auxsizefilelic += 2
00402569  |.  0FAFC3        |imul    eax, ebx                     ;  auxsizefilelic *= pos
0040256C  |.  03C2          |add     eax, edx                     ;  auxsizefilelic += valor
0040256E  |.  8BD0          |mov     edx, eax
00402570  |.  0FB60431      |movzx   eax, byte ptr ds:[ecx+esi]   ;  valor =
buffer[contador]
00402574  |.  8D0480        |lea     eax, ds:[eax+eax*4]          ;  valor = valor + (valor
* 4)
00402577  |.  03C2          |add     eax, edx                     ;  valor += auxsizefilelic
00402579  |.  41            |inc     ecx                          ;  contador++
0040257A  |.  894424 14     |mov     ss:[esp+14], eax
0040257E  |.  3BCF          |cmp     ecx, edi                     ;  while(sizefilelic >
contador)
00402580  |.^ 72 C0         \jb      short 00402542               ;  EFCM.00402542
```

No estoy del todo seguro de esta parte pero es algo así lo que hace. Luego llega al segundo CALL y usa ese valor del primer CALL para obtener la cadena y los caracteres de la cadena con los que compara la cadena obtenida son estos:

```
00AC8D09                        39 37 33 33 33 32 36 34 37          973332647
00AC8D19  36 35 34 39 32 31 30 32 30 30 37 30 30 37 37 33  6549210200700773
00AC8D29  37 33 33 31 31 30 36 39 39 30 37 35 31 38 33 31  7331106990751831
00AC8D39  38 38 30 36 32 33 31 30 31 36 30 31 31 33 39 33  8806231016011393
```

En este caso no coinciden las cadenas.

Hasta aquí he podido llegar. Estas partes son las que compara:

```
Parte que depende de las operaciones con la cadena de siempre pero que empieza con un
espacio.
Parte que depende del nombre.
Cadena que depende de las operaciones con todos los caracteres del archivo de licencia.
Esta parte tiene que coincidir con el serial sin guiones.
Parte que depende de las operaciones con la constante 0x2266 y que siempre me da “8806”.
Parte que depende de las operaciones con el serial.
Esta parte no termino de entender como calcula la cadena.
```

```
00AC8D09                        39 37 33 33 33 32 36 34 37          973332647
00AC8D19  36 35 34 39 32 31 30 32 30 30 37 30 30 37 37 33  6549210200700773
00AC8D29  37 33 33 31 31 30 36 39 39 30 37 35 31 38 33 31  7331106990751831
00AC8D39  38 38 30 36 32 33 31 30 31 36 30 31 31 33 39 33  8806231016011393
```

Entonces se me ocurrió poner un HBP on Write en esa dirección de memoria y reiniciar y después de varios clics a F9 llegué a esta zona donde lo que hace es una serie de operaciones carácter por carácter del ID del archivo para obtener la cadena que usa para todas las comparaciones:

```
00426C60  /$  56            push    esi
00426C61  |.  8B35 10F64600 mov     esi, ds:[46F610]
00426C67  |.  81C6 300A0000 add     esi, 0A30
00426C6D  |.  74 40         je      short 00426CAF          ;  EFCM.00426CAF
00426C6F  |.  53            push    ebx
00426C70  |.  33C9          xor     ecx, ecx
00426C72  |.  57            push    edi
00426C73  |.  8D79 39       lea     edi, ds:[ecx+39]
00426C76  |>  8A0431        /mov     al, ds:[ecx+esi]       ;  valor = ID[contador]
00426C79  |.  3C 40         |cmp     al, 40                 ;  if(valor <= 0x40)
00426C7B  |.  73 16         |jnb     short 00426C93         ;  {
00426C7D  |.  8BD1          |mov     edx, ecx               ;  aux = contador
00426C7F  |.  81E2 03000080 |and     edx, 80000003          ;  aux &= 0x80000003
00426C85  |.  79 05         |jns     short 00426C8C         ;  if(aux ...){
00426C87  |.  4A            |dec     edx
00426C88  |.  83CA FC       |or      edx, FFFFFFFC
00426C8B  |.  42            |inc     edx                    ;  }
00426C8C  |>  2AC2          |sub     al, dl                 ;  valor -= aux
00426C8E  |.  880431        |mov     ds:[ecx+esi], al       ;  ID[contador] = valor
00426C91  |.  EB 14         |jmp     short 00426CA7         ;  }
00426C93  |>  0FB6C0        |movzx   eax, al
00426C96  |.  83E8 41       |sub     eax, 41                ;  valor -= 0x41
00426C99  |.  99            |cdq                            ;  cdq no se que hace
00426C9A  |.  BB 0A000000   |mov     ebx, 0A
00426C9F  |.  F7FB          |idiv    ebx                    ;  result = valor % 0xA
00426CA1  |.  80C2 30       |add     dl, 30                 ;  result += 0x30
00426CA4  |.  881431        |mov     ds:[ecx+esi], dl
```

```
00426CA7  |>  41            |inc     ecx                    ;  contador++
00426CA8  |.  83EF 01       |sub     edi, 1
00426CAB  |.^ 75 C9         \jnz     short 00426C76         ;  EFCM.00426C76
00426CAD  |.  5F            pop     edi
00426CAE  |.  5B            pop     ebx
00426CAF  |>  A1 10F64600   mov     eax, ds:[46F610]
00426CB4  |.  05 B6000000   add     eax, 0B6
00426CB9  |.  50            push    eax
00426CBA  |.  E8 316BFFFF   call    0041D7F0               ;  EFCM.0041D7F0
00426CBF  |.  83C4 04       add     esp, 4
00426CC2  |.  5E            pop     esi
00426CC3  \.  C3            retn
```

Se me ocurre algo, ya sé dónde va cada parte y solo tendría que conseguir que cada parte me de el resultado y luego ir modificando el ID para que coincida, o sea que teniendo todas las cadenas y sabiendo en que parte va cada una solo necesito invertir la parte donde transforma el ID para que me de el ID encriptado. Las cadenas son estas:

1. La cadena que en mi caso se genera con las operaciones con mi serial es “213344629”.
2. La cadena que en mi caso se genera con las operaciones con mi nombre es “48471”.
3. La cadena que en mi caso se genera al poner mi serial sin guiones es “12345612345678”.
4. El primer carácter va cambiando mientras cambio el resto de caracteres ya que se obtiene con operaciones con los caracteres del ID. En mi caso el carácter bueno ha resultado ser el “8”.
5. La cadena “016” es constante con lo que tenemos que hacer que coincida.
6. La cadena resultante de todas las operaciones del archivo de licencia es “417327280“.
7. Y por último la cadena “8806” que es otra constante.

El ID desencriptado debería quedar así:

```
00AC8D11   38 34 37 32 34 31 37 33 32 37 32 38 30 30 2F 31  8472417327280 0/1
00AC8D21   30 37 2D 31 2F 31 32 33 34 35 36 31 32 33 34 35  07-1/12345612345
00AC8D31   31 32 33 34 35 36 37 38 38 38 30 36 32 31 33 30  1234567888062130
00AC8D41   31 36 36 32 39 33 39 33                          16629393
```

Y con este código que me he creado, el cual invierte el proceso de desencriptación, obtengo el ID válido:

```
AnsiString Encriptar(AnsiString cadena)
{
    DWORD valor;
    DWORD contador;
    DWORD aux;

    contador = 0;

    while(contador < cadena.Length()){
        valor = cadena[contador+1];
        if (valor >= 0x30) {
            valor += 0x41 - 0x30;
            cadena[contador+1]=(char)valor;
        } else {
            aux = contador;
            aux &= 0x80000003;
            if(aux < 0){
                aux--;
                aux |= 0xfffffffc;
                aux++;
            }
```

```
            valor += aux;
            cadena[contador+1] = (char)valor;
        }
        contador++;
    }
    return cadena;
}
```

Y estos son datos de registro correctos en este caso:

```
Name      : Agustin
Company   :
Serial No.: 123456-12345-12345678
Reg.ID.   : IIEHMEBHDCHCIAK21KH010BCDEFGBCDEFBCDEFGHIIIAGCBDABGGCJ5J3
```

Y después de codear mi keygen, añadiéndole todas las partes para que me genere el ID desencriptado y luego lo encripte, quedó así:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <stdio.h>
#include <fstream.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

const char cadenaEncabezado[] = "EF CheckSum Manager distribution
license\x0D\x0A\x0D\x0A"
                                "Copyright (c) 1994-2007, Emil Fickel\x0D\x0A"
                                "All rights reserved.\x0D\x0A\x0D\x0A"
                                "This copy is licensed to:\x0D\x0A";
const char cadenaNombre[] = "Name      : ";
const char cadenaCompania[] = "Company   : ";
const char cadenaSerial[] = "Serial No.: ";
const char cadenaID[] = "Reg.ID.   : ";
const char cadenaPie[] = "\x0D\x0AWarning: This product is licensed to you pursuant to
the terms of the\x0D\x0A"
                         "author license agreement included with the original
software.\x0D\x0A"
                         "It is protected by copyright law and international
treaties.\x0D\x0A"
                         "Unauthorized reproduction or distribution may result in severe
civil and\x0D\x0A"
                         "criminal penalties, and will be prosecuted to the maximum
extent possible\x0D\x0A"
                         "under the law.\x0D\x0A\x0D\x0A"
                         "One registered copy of this software may be dedicated to a
single\x0D\x0A"
                         "person who uses the software on one or more computers or to a
single\x0D\x0A"
                         "workstation used by multiple people.\x0D\x0A";
const char salto[]="\x0D\x0A";
//---------------------------------------------------------------------------
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

AnsiString Encriptar(AnsiString cadena)
{
    DWORD valor;
    DWORD contador;
    DWORD aux;

    contador = 0;

    while(contador < cadena.Length()){
        valor = cadena[contador+1];
        if (valor >= 0x30) {
            valor += 0x41 - 0x30;
            cadena[contador+1]=(char)valor;
        } else {
            aux = contador;
            aux &= 0x80000003;
            if(aux < 0){
                aux--;
                aux |= 0xfffffffc;
                aux++;
            }
            valor += aux;
            cadena[contador+1] = (char)valor;
        }
        contador++;
    }
    return cadena;
}
//---------------------------------------------------------------------------

AnsiString Desencriptar(AnsiString cadena)
{
    DWORD valor;
    DWORD contador;
    DWORD aux;

    contador = 0;

    while(contador < cadena.Length()){
        valor = cadena[contador+1];
        if (valor >= 0x40) {
            valor -= 0x41;
            valor = valor % 0xA;
            valor += 0x30;
            cadena[contador+1] = (char)valor;
        } else {
            aux = contador;
            aux &= 0x80000003;
            if(aux < 0){
                aux--;
                aux |= 0xfffffffc;
                aux++;
            }
            valor -= aux;
            cadena[contador+1] = (char)valor;
```

```
        }
        contador++;
    }
    return cadena;
}
//---------------------------------------------------------------------------

AnsiString CalcularValorSerial(AnsiString serial)
{
    unsigned long total = 0;
    int contador = 0;

    while(contador < serial.Length()){
        total += serial[contador + 1];
        total = contador + (total * 2) + 1;
        contador++;
    }
    return AnsiString().sprintf("%04lu", total);
}
//---------------------------------------------------------------------------

AnsiString CalcularValorNombre(AnsiString nombre)
{
    unsigned long total = 0;
    int contador = 0;

    total += 0x20; //empieza valiendo 0
    while(contador < nombre.Length()){
        total += (nombre[contador + 1] * 2);
        total = contador + (total * 2) + 1;
        contador++;
    }
    return AnsiString().sprintf("%05lu", total);
}
//---------------------------------------------------------------------------

AnsiString ObtenerValorCompania(AnsiString compania)
{
    unsigned long valor1=0,valor2=0,retval=0x10;
    int pos=0;
    unsigned __int64 aux;

    if(compania.Length() >= 2){
        while((compania.Length() - pos) > 1){
            valor1 += (compania[pos + 1] * 2) + pos + compania.Length();
            pos++;
            valor2 += (compania[pos + 1] * 2) + pos + compania.Length();
            pos++;
        }
    }

    if(compania.Length() - pos == 1)
        retval = (compania[pos + 1] * 2) + 0x10 + pos + compania.Length();

    retval += (valor1 + valor2);

    aux = (unsigned __int64)retval * 0xD1B71759;

    valor1 = aux >> 32;
    valor1 = valor1 >> 0xD;
```

```
    valor1 *= 0x2710;
    retval -= valor1;
    return AnsiString().sprintf("%03lu", retval);
}
//---------------------------------------------------------------------------

AnsiString CalcularValorLIC(char *cadena,int sizebuffer)
{
    unsigned long sizefilelic;
    unsigned long valor1, valor2, valor3 = 0, retval, contador = 0;
    unsigned long pos;
    int i;

    for(i=0;cadena[i] != '\0' & i < sizebuffer; i++);
    sizefilelic = i;

    retval = sizefilelic;
    while(sizefilelic > contador)
    {
        valor1 = cadena[contador] + contador + 3;
        valor3  ^= valor1;
        valor2 = valor3 % (sizefilelic + 1);
        pos = valor2 + 2;
        if(pos > sizefilelic)
            pos = 2;
        valor1 = cadena[pos];
        retval += 2;
        unsigned __int64 aux = (unsigned __int64)retval * (unsigned __int64)valor3;
        retval = aux & 0xFFFFFFFF;
        retval += valor1;
        valor1 = cadena[contador];
        valor1 = valor1 + (valor1 * 4);
        retval += valor1;
        contador++;
    }
    return AnsiString().sprintf("%09lu", retval);
}
//---------------------------------------------------------------------------

AnsiString CalcularValorID(AnsiString id)
{
    unsigned long valor1 = 0x2B, valor2 = 0, valor3 = 0;
    int caracter, pos = 1;

    id[1] = 0x20;

    while(pos < 57){
        caracter = id[pos];
        valor1 += (caracter*2) + 0x2C;
        pos++;
        caracter = id[pos];
        valor2 += (caracter*2) + 0x2C;
        pos++;
        caracter = id[pos];
        valor3 += (caracter*2) + 0x2C;
        pos++;
    }
    valor1 += (valor2 + valor3);

    unsigned __int64 aux = (unsigned __int64)valor1 * 0x51EB851F;
```

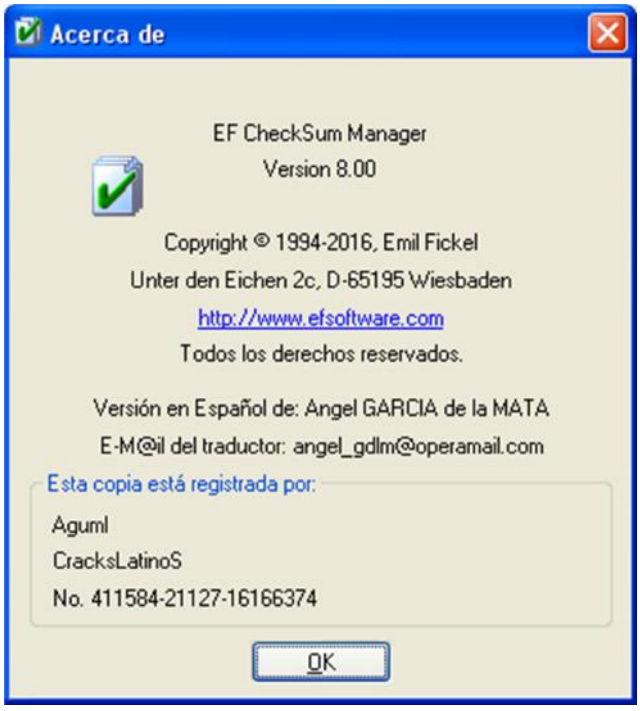
```
    valor1 -= (aux >> 37) * 0x64;

    return AnsiString().sprintf("%02lu", valor1);;
}
//---------------------------------------------------------------------------

//Limpiar buffer
void LimpiarBuffer(char *buffer,int sizeBuffer)
{
    for(int i = 0; i < sizeBuffer;buffer[i] = '\0', i++);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ButtonCalcularClick(TObject *Sender)
{
    AnsiString id, aux;
    unsigned long serialP1, serialP2, serialP3;

    //Si ya hay memoria asignada al buffer la libero antes de nada
    if(bufferLleno){
        delete bufferArchivo;
        bufferArchivo = NULL;
        bufferLleno = false;
    }

    do{
        //Obtengo las 3 partes del serial con valores aleatorios
        serialP1 = random(999999);
        serialP2 = random(99999);
        serialP3 = random(99999999);

        EditSerial->Text = AnsiString().sprintf("%06lu-%05lu-%08lu", serialP1, serialP2,
serialP3);
    }while(listaNegra->IndexOf(EditSerial->Text) != -1); //Si el serial esta en la lista
negra creo otro diferente

    //concateno las partes del serial sin guiones
    aux = AnsiString().sprintf("%06lu%05lu%08lu", serialP1, serialP2, serialP3);

    //Relleno el ID con una sucesion del serial pero empezando siempre por el segundo
    //caracter hasta llegar a 57 caracteres
    id="";
    for(int x=1, i=2; x <= 57; x++, i++){
        if(i > 19)
            i = 2;
        id += (char)aux[i];
    }

    //Obtengo el tama ue ocupar  odo el texto
    sizeBuffer = snprintf(NULL, 0, "%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s",
                          cadenaEncabezado,
                          cadenaNombre, EditNombre->Text.c_str(), salto,
                          cadenaCompania, EditCompania->Text.c_str(), salto,
                          cadenaSerial, EditSerial->Text.c_str(), salto,
                          cadenaID, id.c_str(), salto,
                          cadenaPie);
    sizeBuffer++;

    //Reservo la memoria necesaria para el buffer
    bufferArchivo = new char[sizeBuffer];
```

```
//Limpio el buffer
LimpiarBuffer(bufferArchivo, sizeBuffer);

//Concateno todo como lo requiere la operacion para obtener el valor sobre el texto
//de la licencia. Viene a ser como el original pero usando el ID con la concatenacion
//de seriales sin guiones
snprintf(bufferArchivo, sizeBuffer-1, "%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s",
         cadenaEncabezado,
         cadenaNombre, EditNombre->Text.c_str(), salto,
         cadenaCompania, EditCompania->Text.c_str(), salto,
         cadenaSerial, EditSerial->Text.c_str(), salto,
         cadenaID, id.c_str(), salto,
         cadenaPie);

//Creo un ID con caracteres aleatorios desde el caracter 0x21 que es '!' hasta
//el caracter 0x39 que es '9' y como 0x39 - 0x21 = 0x18 uso ese valor para crear
//el valor aleatorio y luego le sumo el valor del primero para obtener un caracter
//dentro de ese rango. Tiene que ser ese rango porque, al desencriptar el ID,
//cualquier ID no se sale de este rango y si ponemos un rango mayor no funcionará
id = "";
for(int i = 0; i < 57; i++){
    id += (char)(random(0x18) + 0x21);
}

//Desencripto el ID
id = Desencriptar(id);

//Concateno las partes del serial sin guiones
aux = AnsiString().sprintf("%06lu%05lu%08lu", serialP1, serialP2, serialP3);

//Coloco el serial sin guiones en el ID desencriptado
id = id.SubString(1,22) + aux + id.SubString(42, id.Length() - 41);

//Obtengo la cadena con el valor del nombre introducido
aux = CalcularValorNombre(EditNombre->Text);

//Coloco los caracteres 2, 3, y 4 de la cadena obtenida en el ID desencriptado
id = id.SubString(1, 1) + aux.SubString(2, 3) + id.SubString(5, id.Length() - 4);

//Obtengo la parte del "8806"
aux = AnsiString().sprintf("%04lu", 0x2266);

//La coloco en el ID
id = id.SubString(1, 41) + aux.SubString(1, 4) + id.SubString(46, id.Length() - 45);

//Obtengo el valor a partir del serial
aux = CalcularValorSerial(EditSerial->Text);

//Coloco los 3 primeros caracteres del valor obtenido en el ID
id = id.SubString(1, 45) + aux.SubString(1, 3) + id.SubString(49, id.Length() - 48);

//Obtengo la parte del 016 que no se bien como la obtiene pero bueno
aux = ObtenerValorCompania(EditCompania->Text);

//La coloco en el 0x16
id = id.SubString(1, 48) + aux.SubString(1, 3) + id.SubString(52, id.Length() - 51);

//Obtengo la cadena con el valor que se obtiene a partir del buffer de licencia
aux = CalcularValorLIC(bufferArchivo, sizeBuffer - 1);

//Coloco los 9 primeros caracteres de la cadena en el ID
```

```
    id = id.SubString(1, 5) + aux.SubString(1, 9) + id.SubString(15, id.Length() - 14);

    //Obtengo la cadena que se genera a partir del ID
    aux = CalcularValorID(id);

    //Coloco el que necesito del resultado en su posicion en el ID
    id = aux.SubString(1,1) + id.SubString(2, id.Length() - 1);

    //Encripto el ID y lo muestro
    id = Encriptar(id);

    //Como al desencriptar, si es mayor o igual a 0x40, va a dividir su valor entre 0xA
    //que es 10 y quedarse con el resto, cualquier valor que sea mayor o igual a 0x40
    //podemos multiplicarlo por 10, 20, o 30 sin temor de salirnos de los valores de la
    //tabla ASCII y así tener un ID que contenga más caracteres ya que si no solo
saldrian
    //los menores de 0x40 y el rango de la A a la J
    for(int i=1; i<=57; i++)
        if(id[i] >= 0x40)
            id[i] = id[i] + (random(3) * 10);

    //Limpio el buffer
    LimpiarBuffer(bufferArchivo, sizeBuffer);

    //Monto el buffer con la cadena de licencia final
    snprintf(bufferArchivo, sizeBuffer - 1, "%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s",
             cadenaEncabezado,
             cadenaNombre, EditNombre->Text.c_str(), salto,
             cadenaCompania, EditCompania->Text.c_str(), salto,
             cadenaSerial, EditSerial->Text.c_str(), salto,
             cadenaID, id.c_str(), salto,
             cadenaPie);
    bufferLleno = true;

    EditID->Text = id;
    ButtonCrearLIC->Enabled = true;
}
//---------------------------------------------------------------------------

AnsiString GetLastErrorAsString()
{
    //Get the error message, if any.
    DWORD errorMessageID = GetLastError();
    if(errorMessageID == 0)
        return NULL; //No error message has been recorded

    LPSTR messageBuffer = NULL;
    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_IGNORE_INSERTS,
                  NULL,
                  errorMessageID,
                  MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
                  (LPSTR)&messageBuffer,
                  0,
                  NULL);

    AnsiString message = messageBuffer;

    //Free the buffer.
    LocalFree(messageBuffer);
```

```
    return message;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ButtonCrearLICClick(TObject *Sender)
{
    ofstream archivoLIC;
    AnsiString error;

    //Abro el archivo de licencia
    archivoLIC.open("EFCM.LIC", ios::binary | ios::out);

    if(archivoLIC.fail()){
        Application->MessageBoxA(AnsiString("No se pudo crear el archivo de licencia.\n"
+ GetLastErrorAsString()).c_str(),
                                 "Error",
                                 MB_ICONERROR);
        return;
    }
    //Guardo la informacion en el
    archivoLIC.write(bufferArchivo, sizeBuffer - 1);

    //Cierro el archivo de licencia
    archivoLIC.close();
    Application->MessageBoxA("El archivo de licencia se ha creado satisfactoriamente.\nNo
olvide colocarla en el directorio del programa.",
                             "Enhorabuena",
                             MB_ICONINFORMATION);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
    if(bufferLleno)
        delete bufferArchivo;
        bufferArchivo != NULL;
        bufferLleno = false;
    delete listaNegra;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    bufferLleno = false;
    bufferArchivo = NULL;
    listaNegra = new TStringList;

    listaNegra->Add("773030-42099-90850837");
    listaNegra->Add("313491-63575-90700639");
    listaNegra->Add("723486-12054-90750843");
    listaNegra->Add("973334-72239-90770834");
    listaNegra->Add("963344-82230-90770834");
    listaNegra->Add("943066-51242-90770837");
    listaNegra->Add("953354-92231-90770834");
    listaNegra->Add("733276-72063-90750835");
    listaNegra->Add("763542-22040-90751833");
    listaNegra->Add("033374-02333-90780834");
    listaNegra->Add("083129-11338-98780836");
    listaNegra->Add("893310-13137-98760834");
    listaNegra->Add("023384-12334-90780834");
    listaNegra->Add("083324-52338-90780834");
```

```
    listaNegra->Add("913191-03265-90770836");
    listaNegra->Add("773733-11069-90751831");
    listaNegra->Add("903101-13266-90770836");
    listaNegra->Add("993111-23267-90770836");
    listaNegra->Add("723385-93064-90750844");
    listaNegra->Add("753008-43071-90750837");
    listaNegra->Add("703003-32046-90850847");
    listaNegra->Add("713497-21055-90850833");
    listaNegra->Add("863263-01132-90760835");
    listaNegra->Add("733876-35033-90750879");
    listaNegra->Add("793615-72057-90750831");
    listaNegra->Add("753450-33051-90750833");

    randomize();
}
//---------------------------------------------------------------------------
```



Agradecimientos a todos los CracksLatinoS y en especial a Apuromafo por la ayuda prestada.