Introducción

Hola a todos.

En esta entrega vamos a realizar ingeniería inversa pura y dura. Tenemos un programa compilado en Fortran 77, es un Fortran para Windows el cual no arranca en sistemas operativos NT como por ejemplo Window 2000, WinXP o Vista.

Buscando en la red he visto que este fallo es común en programas antiguos hechos en Fortran así que me he decido a realizar este tutorial por si puede ayudar a alguien más.

En este caso el problema está en el propio Fortran, el cual para la petición de memoria, movimiento de zonas de memoria, … usa APIS no documentadas de la ntdll.dll. Precisamente esas APIs han cambiado de nombre en las nuevas versiones de los sistemas operativos Windows y esta es una de las causas por la cual el programa no funciona.

Las herramientas usadas en este tutorial han sido:

- OllyDBG, personalmente uso una de las versiones de SND
- IDA PRO 5.2 versión 32 bits
- LordPE
- MSDN bien OnLine, bien instadados en su PC.

Manos a la obra.

Reparando la importación

Si intentamos arrancar el programa nos encontramos con un cartelito como este:

Acto seguido lo siguiente que hago es buscar esa API en el MSDN y evidentemente no la encuentro pero si esta:

RtlAllocateHeap

¿Curioso verdad? Enseguida se me viene a la cabeza APIs como VirtualAlloc o VirtualAllocEx cuya diferencia es que VirtualAllocEx recibe entre sus argumentos un Handle del proceso.

```
LPVOID VirtualAlloc(

  LPVOID lpAddress,          // address of region to reserve or commit
  DWORD dwSize,              // size of region
  DWORD flAllocationType,    // type of allocation
  DWORD flProtect            // type of access protection
 );
```

```
LPVOID VirtualAllocEx(

  HANDLE hProcess,           // process within which to allocate memory
  LPVOID lpAddress,          // desired starting address of allocation
  DWORD dwSize,              // size, in bytes, of region to allocate
  DWORD flAllocationType,    // type of allocation
  DWORD flProtect            // type of access protection
 );
```

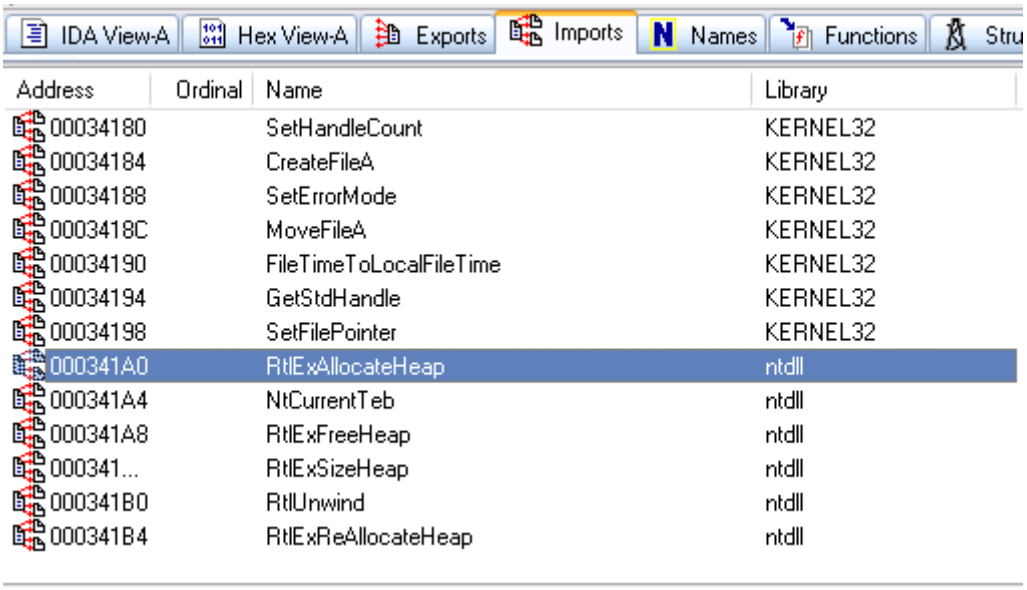
Curiosamente en este caso si hProcess es 0 se comporta igual que VirtualAlloc. Asi que en principio me esperaba algo así que con unos cuantos NOP bien puesto se podría arreglar, pero veamos que la cosa fue un poco más simple.

La API que tenemos en nuestro WinXP es la siguiente RtlAllocateHeap que según el MSDN

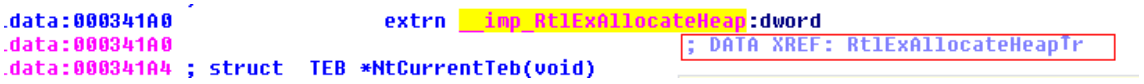
```
PVOID RtlAllocateHeap(
    IN PVOID  HeapHandle,
    IN ULONG  Flags,
    IN SIZE_T  Size
    );
```

Por desgracia no pude encontrar la declaración de RtlExAllocateHeap, y como no puedo cargarlo en el depurador porque la sección Import es incorrecta veamos que nos dice el IDA PRO.

Abrimos el IDA, cargamos el programa y seleccionamos la pestaña Import

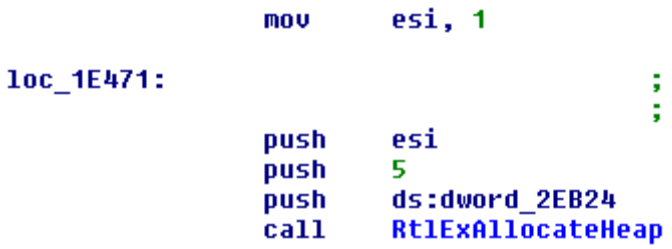


Buscamos nuestra API y le damos doble click a ver donde nos lleva



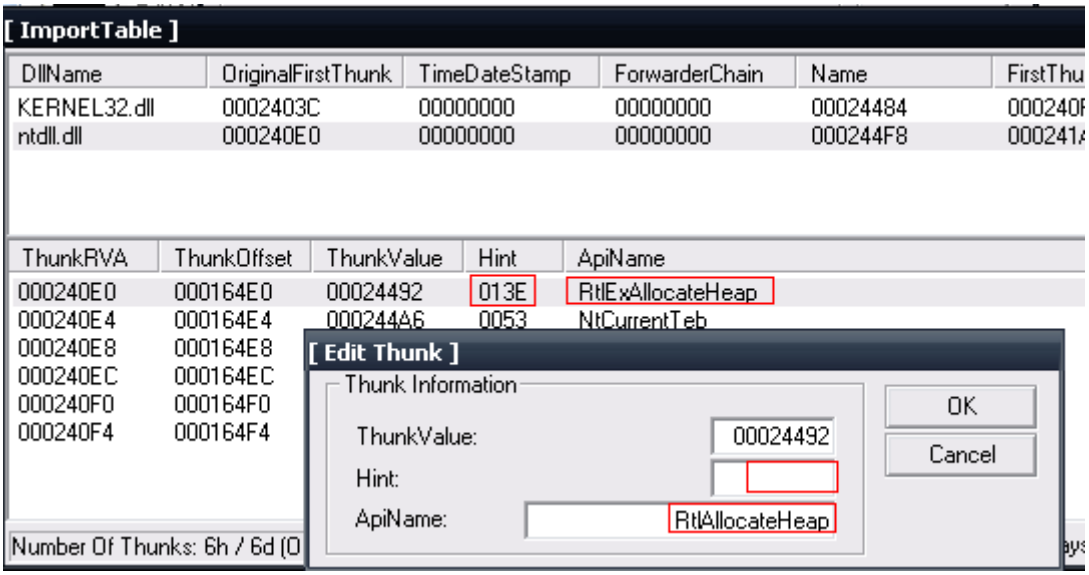
Hacemos doble click en la zona que he remarcado

Vemos que se llama desde un único sito pinchemos en la zona recuadrada



Pues aquí lo tenemos, parece que RtlExAllocateHeap también usa 3 parámetros.

Llegamos a este punto realizamos una copia con otro nombre del programa en cuestión y dejamos este abierto con en el IDA. La copia del programa la abrimos con LordPE para modificar la sección de Importación.



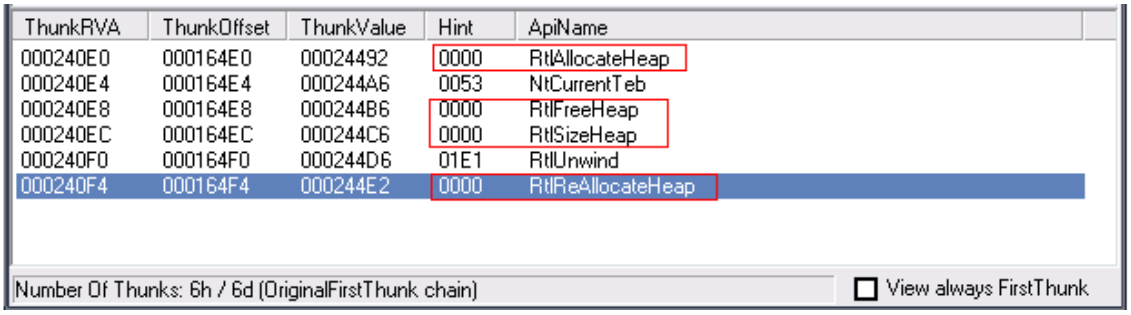
Una vez seleccionada la dll buscamos las APIs problemáticas y las editamos.
Los cambios son:

En ApiName simplemente le quitamos el **Ex**

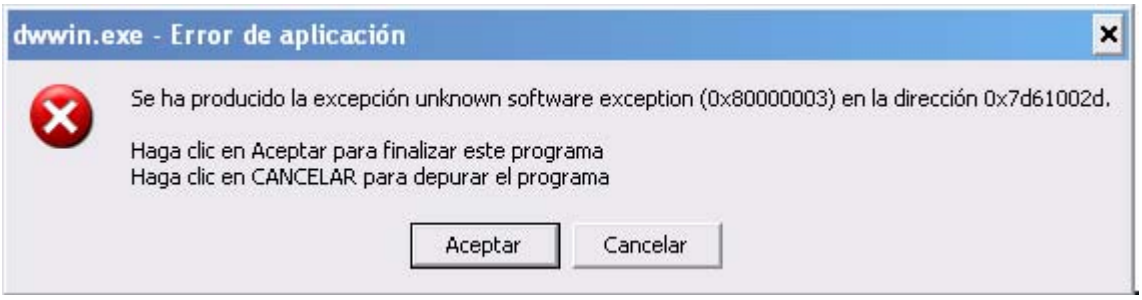
En Hint lo dejamos vacío.

Para el que no lo sepa el Hint es el dato a buscar cuando se carga la API por ordinal y no por nombre, como sabemos el nombre de la API a poner pero no su ordinal simplemente borramos ese dato.

Tras el retoque la importación queda así:



Salvamos todo y lo intentamos ejecutar a ver si tenemos suerte.

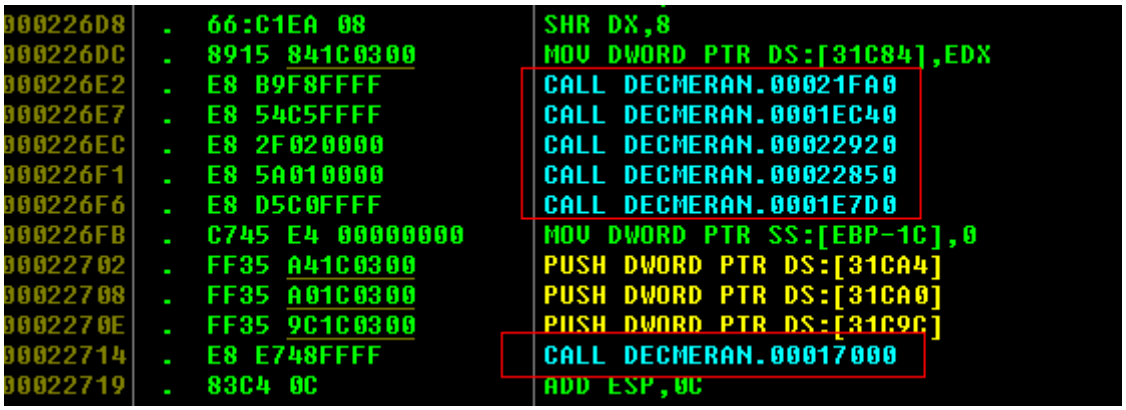


Vemos que no, pero por lo meno parece que ya carga.

Parcheando Actualizando Estructuras de Windows

Añadamos el programa a OllyDBG y vamos comparando que vemos con lo que nos muestra IDA PRO

Estamos en el EP y si miramos un poco con la rueda vemos esto



Los primeros call que estan recuadrados son en los que Fortran obtiene datos como por ejemplo, número de argumentos, punteros a los argumentos, las variables del Enviroment, …

Y el call que está recuadrado solo sería la llamada a la función main tipica de C que en Fortran no se como se llamaría, pero para entendernos todo la parafernalia hasta llegar a ese cal no es más que inicializaciones que hace Fortran y donde nos encontramos el problema.

Veamos un poco las notas que tengo ya en IDA

```
mov     dword_31C84, edx ; Hasta aqui va tomando datos
                         ; para ver la version de Windows
call    GetProcessHeap   ; Devuelve el puntero a la estructura Heaps
                         ; del proceso usando el TEB y el PEB
call    GetStdHandles    ; Función que toma los Standares de:
                         ;      - Entrada (teclado)
                         ;      - Salida  (Pantalla)
                         ;      - Error   (Suele ser pantalla)
call    sub_22920        ; Copia el Numero de argumentos pasados
                         ; en la variable global NumeroDeArgumentos
                         ;
                         ; Copia el string de ComandLine en la variable
                         ; global PArgumentos
call    sub_22850        ; Lee las variables indicadas en el Enviroment
                         ; y aloja una lista de punteros a las variables
                         ; en la variable global ListaPunteroEnviroment
call    sub_1E7D0        ; Inicializa algo ¿? al estilo de Borland
mov     [ebp+var_1C], 0
push    ListaPunteroEnviorment ; puntero a una lista de puntero a cada v
                         ; indicada por pEnviroment
push    PArgumentos
push    NumeroDeArgumentos
call    sub_17000        ; El programa en si
```

El problema

Antes de analizar la call remarcada en el Olly sería **CALL 0001EC40** veamos la 1º call que he llamado GetProcessHeap

```
.text:00021FA0 GetProcessHeap  proc near               ; CODE XREF: start+D2↓p
.text:00021FA0                 call    NtCurrentTeb
.text:00021FA5                 mov     eax, [eax+30h]  ; eax = Puntero a PEB
.text:00021FA8                 mov     eax, [eax+18h]  ; eax puntero a Heaps
.text:00021FAB                 mov     ds:dword_2EB24, eax
.text:00021FB0                 retn
.text:00021FB0 GetProcessHeap  endp
```

Esta función es la que se encarga de obtener el puntero a la estructura de los Heaps del programa para ir pidiendo, moviendo, borrando memoria.

En vez de crear un Heap lo que hace es llamar a la API NTCurrenTeb de la ntdll.dll. Esta API devuelve la dirección TEB del programa, luego toma el puntero a PEB y de la estructura PEB toma dirección de la estructura Heaps que la guarda en una variable global.

Ante este tipo de cosas pues es normal que con el cambio de SO los programas dejen de funcionar por no usar las APIs de alto nivel que ofrece Windows las cuales no han variado, al menos en su nombre y número de parámetros la verdad que un 0 para los desarrolladores del compilador de Fortran ☹

Bueno sigamos con la call problemática, la que en IDA le di el nombre de GetStdHandles

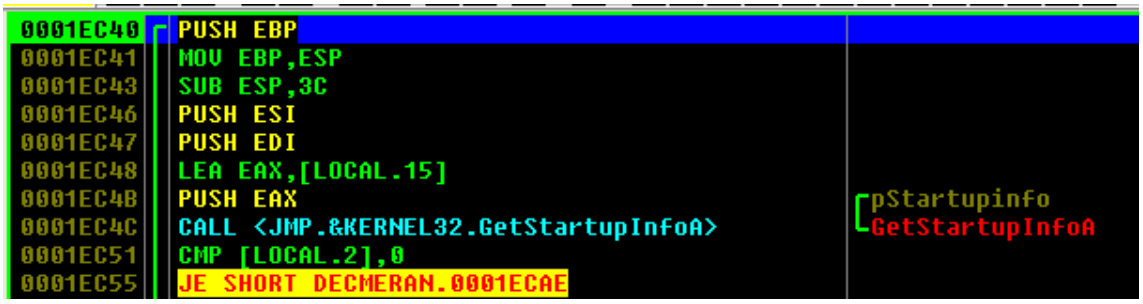
Esta función se encarga de obtener los Handles al stdin, stdout y stderr
En cristiano:

- stdin: Estandar de entrada, tipicamente el teclado
- stdout: Estandar de salida tipicamente la pantalla
- stderr: Estandar de error, tipicamente la pantalla

Vemos un estracto de la función según IDA

```
.text:0001EC40 GetStdHandles   proc near               ; CODE XREF: start+D7↓p
.text:0001EC40
.text:0001EC40 StartupInfo     = _STARTUPINFOA ptr -3Ch
.text:0001EC40
.text:0001EC40                 push    ebp
.text:0001EC41                 mov     ebp, esp
.text:0001EC43                 sub     esp, 3Ch
.text:0001EC46                 push    esi
.text:0001EC47                 push    edi
.text:0001EC48                 lea     eax, [ebp+StartupInfo]
.text:0001EC4B                 push    eax             ; lpStartupInfo
.text:0001EC4C                 call    GetStartupInfoA
.text:0001EC51                 cmp     [ebp+StartupInfo.lpReserved2], 0
.text:0001EC55                 jz      short loc_1ECAE
.text:0001EC57                 lea     edi, [ebp+StartupInfo.hStdInput]
.text:0001EC5A                 mov     esi, [ebp+StartupInfo.lpReserved2]
.text:0001EC5D                 movsd
.text:0001EC5E                 mov     edx, [ebp+StartupInfo.hStdInput]
.text:0001EC61                 mov     eax, 40h
.text:0001EC66                 cmp     edx, eax
.text:0001EC68                 jl      short loc_1EC6C
.text:0001EC6A                 mov     edx, eax
```

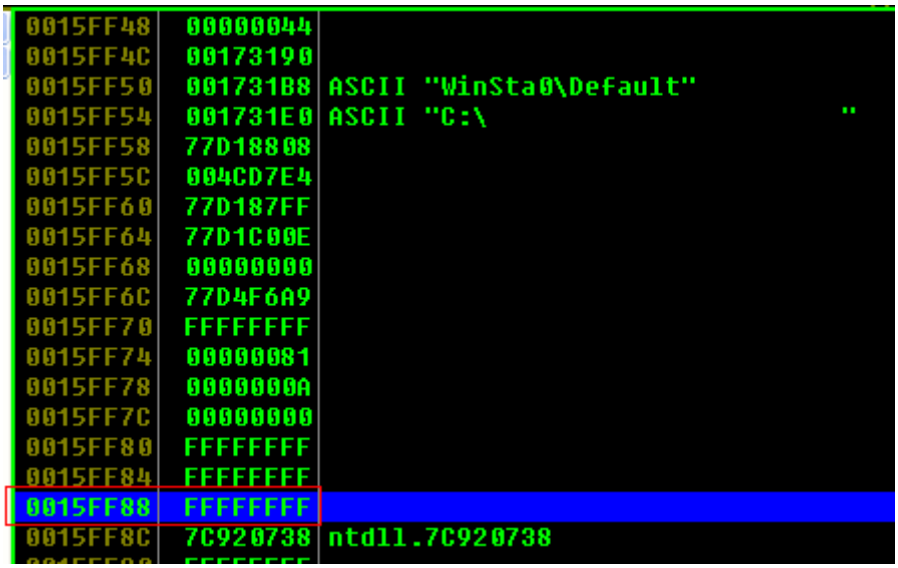
Parece que solo tiene una variable global y es la que se usa obtener los datos del encendido del proceso aunque lo veremos en Olly lo más importante es ese -3Ch, ahora lo veremos.



Estamos parados en el PUSH EBP, veamos la PILA

Es decir en la posición 15FF88 tenemos la dirección de retorno, la cual no se puede pisar sin la liamos.

Pues vamos ejecutando con F8 paso a paso y nos paramos en la dirección 1EC51, es decir justo después de la call GetStartupInfoA y vemos la pila



Pues el valor se está machacando y ¿eso porque?

Vemos la estructura STARTUPINFO

```
typedef struct _STARTUPINFO { // si
    DWORD   cb;
    LPTSTR  lpReserved;
    LPTSTR  lpDesktop;
    LPTSTR  lpTitle;
    DWORD   dwX;
    DWORD   dwY;
    DWORD   dwXSize;
    DWORD   dwYSize;
    DWORD   dwXCountChars;
    DWORD   dwYCountChars;
    DWORD   dwFillAttribute;
    DWORD   dwFlags;
    WORD    wShowWindow;
    WORD    cbReserved2;
    LPBYTE  lpReserved2;
    HANDLE  hStdInput;
    HANDLE  hStdOutput;
    HANDLE  hStdError;

} STARTUPINFO, *LPSTARTUPINFO;
```

Según la información de MSDN la variable **cb** tiene el tamaño en Byte de la estructura y según lo que vemos en la pila este dato se encuentra en **0015FF48** y tiene un valor de **44h**

Si recordamos, en el principio de la función tenemos

```
PUSH EBP
MOV EBP,ESP
```

SUB ESP,3C

El Sub ESP,3C lo que está haciendo es reservar espacio para las variables locales de la función, que en este caso como vimos en el IDA solo tenemos la estructura, por lo que tendremos que cambiar el 3C por el 44 que hemos visto en el OLLY

¿Y este cambio porqué? Bueno se ve que los de Microsoft en un momento dado ampliaron la estructura y añadieron más campos. Actualmente este tipo de cosas sigue pasando pero lo han mejorado ya que antes de llamar a las APIs que rellenan las estructuras es obligatorio indicar el tamaño que hemos reservado para ellas, de esta forma los de Microsoft saben si usar la estructura antigua o la nueva, en este caso se olvidaron de la compatibilidad hacia atrás y se jode todo.

¿Y con ese cambio que hemos hecho ya está todo?

Pues va a ser que no, con cambiar el SUB ESP,3C por SUB ESP,44 hacemos que no se desborde la pila pero como vemos en IDA en esta función se hace uso de los campos de la estructura y al ampliar la estructura los Offset que utilizan son incorrectos y hay que actualizarlos.

```
text:0001EC4C                   call    GetStartupInfoA
text:0001EC51                   cmp     [ebp+StartupInfo.lpReserved2], 0
text:0001EC55                   jz      short loc_1ECAE
text:0001EC57                   lea     edi, [ebp+StartupInfo.hStdInput]
text:0001EC5A                   mov     esi, [ebp+StartupInfo.lpReserved2]
```

Los cambios para mantener bien los punteros, según nos indica IDA, son los siguientes

```
0001EC40  PUSH EBP
0001EC41  MOV EBP,ESP
0001EC43  SUB ESP,44
0001EC46  PUSH ESI
0001EC47  PUSH EDI
0001EC48  LEA EAX,DWORD PTR SS:[EBP-44]
0001EC4B  PUSH EAX
0001EC4C  CALL <JMP.&KERNEL32.GetStartupInfoA>
0001EC51  CMP DWORD PTR SS:[EBP-10],0
0001EC55  JE SHORT DECMERAN.0001ECAE
0001EC57  LEA EDI,DWORD PTR SS:[EBP-C]
0001EC5A  MOV ESI,DWORD PTR SS:[EBP-10]
0001EC5D  MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0001EC5E  MOV EDX,DWORD PTR SS:[EBP-C]
```

```
0001EC6C  MOV ESI,DWORD PTR SS:[EBP-10]
0001EC6F  MOV EDI,DECMERAN.00031CB4
0001EC74  ADD ESI,4
0001EC77  MOV ECX,EDX
0001EC79  SHR ECX,2
0001EC7C  REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0001EC7E  MOV ECX,EDX
0001EC80  AND ECX,3
0001EC83  REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
0001EC85  MOV EAX,DWORD PTR SS:[EBP-C]
```

Con estos cambios conseguimos hacer andar el programa.

Como dije al inicio del tutorial esto no se puede optimizar sobre todo porque al igual que se ha detectado una estructura que ha cambiado al evolucionar el SO, esto mismo podría pasar dentro del “código del programador” y no en la “cabecera” que nos crea Fortran al compilar el ejecutable. En este caso el programa lo que hace es leer unos ficheros para generar otros, nada visual y por eso hemos terminado rápido, con otros programas igual los parches son más profundos.

Espero que haya quedado claro los pasos que hemos realizado en esta aventurilla en el que hemos vuelto a hacer funcionar un programa un poco viejito pero vital para mi padre en su curro

Saludos a todos los miembros de CracksLatinos