

## INTRODUCCION AL CRACKING CON OLLYDBG PARTE 51

Bueno a partir de aca haremos un cambio, que creo que sera para bien, haremos los ultimos tutes de desempacado interactivamente, y como se puede hacer eso? Ya veran jeje.

Bueno eso no quiere decir que acaba la introduccion solo que pasaremos a otro tema, pero antes los hare trabajar un poquito, hoy desempacaremos interactivamente el asprotect ultima version hasta hoy, yo hare lo mas facil y ustedes lo mas difici, jeje que malo.

El archivo UnPackMe\_ASProtect.2.3.04.26.a.exe no tiene todas las protecciones asi que es el mas sencillo de la version esta, igua les digo que si en los intentos de llegar al OEP les empieza a decir ERROR DE PROTECCION, pues copiandolo a otro path funciona correctamente, lo mismo que si tienen en la carpeta que estan trabajando varios con diferentes nombres si alguno empieza a molestar con eso de error de proteccion y no quiere arrancar, pues usan el otro, luego el que usaban primero se desengancha del error solo, vaya a saber porque , jeje.

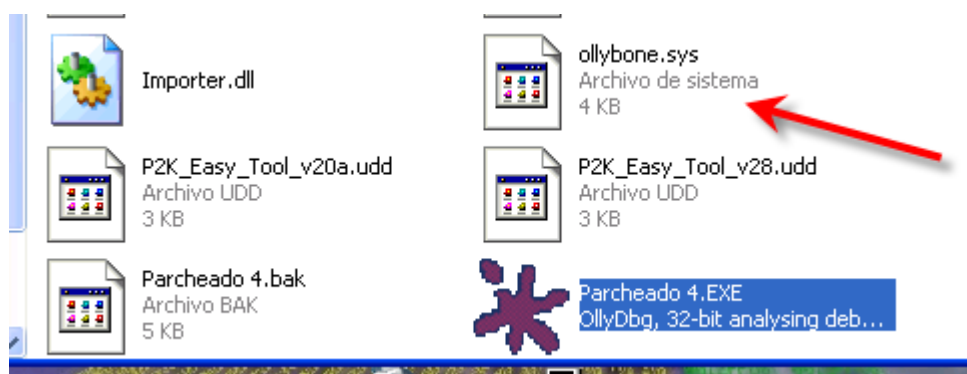
Ademas haremos la prueba de dos plugins que no hemos utilizado hasta ahora el OLLYBONE y el WEASLE.

La instalacion del OLLYBONE dice

Installation:

Copy ollybone.dll and i386/ollybone.sys to your OllyDbg directory.

Lo cual traducido al idioma humano quiere decir, copiar ollybone.dll a la carpeta donde tenes tus plugins y el ollybone.sys a la carpeta del OLLYDBG, si coinciden copias los dos a la carpeta del OLLYDBG.

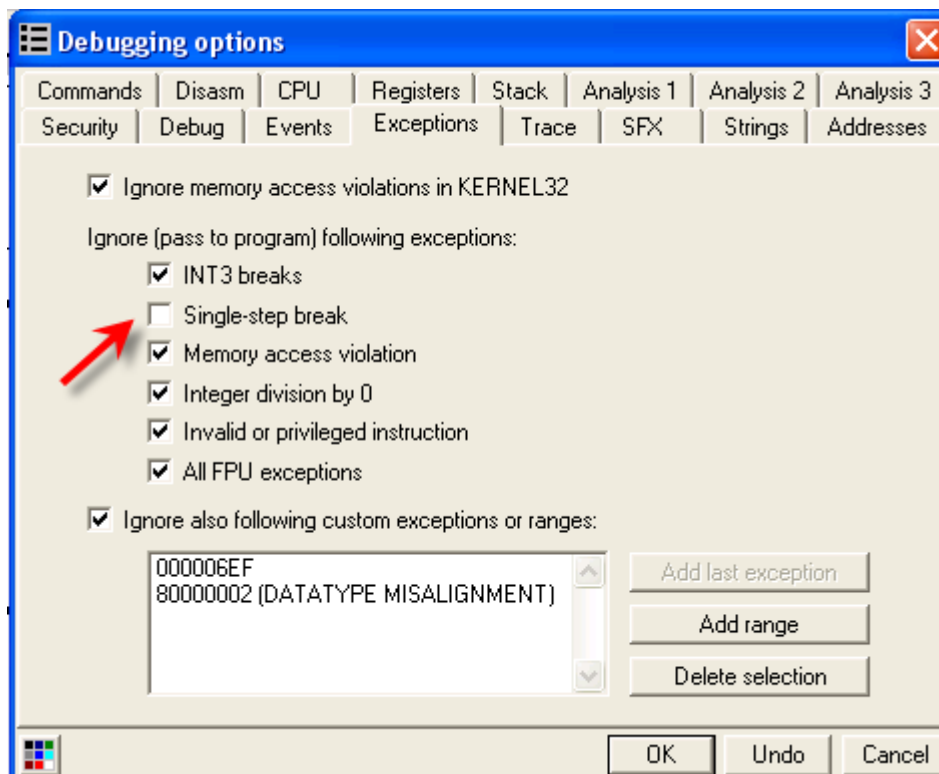


Alli esta el sys juanto al ejecutable del OLLY y el dll lo coloque en la carpeta de los plugins que en mi caso es [C:/PLUGINS](#).

Para el Weasle la cosa es que el paquete original nos pide que si usamos el weasle en un ollydbg modificado, desempaquemos la dll que esta empacado con upx, pero que la desempaquemos con el mismo UPX 2.01 original, ya que tambien desempaca como hemos visto, pues para ahorrarles ese trabajo yo ya lo he hecho y adjunto estan las dll ya listas para usar, la importer.dll va en la carpeta del OLLY como se ve tambien en la imagen anterior y RL!Weasle.dll en la carpeta de los plugins, bueno a trabajar.

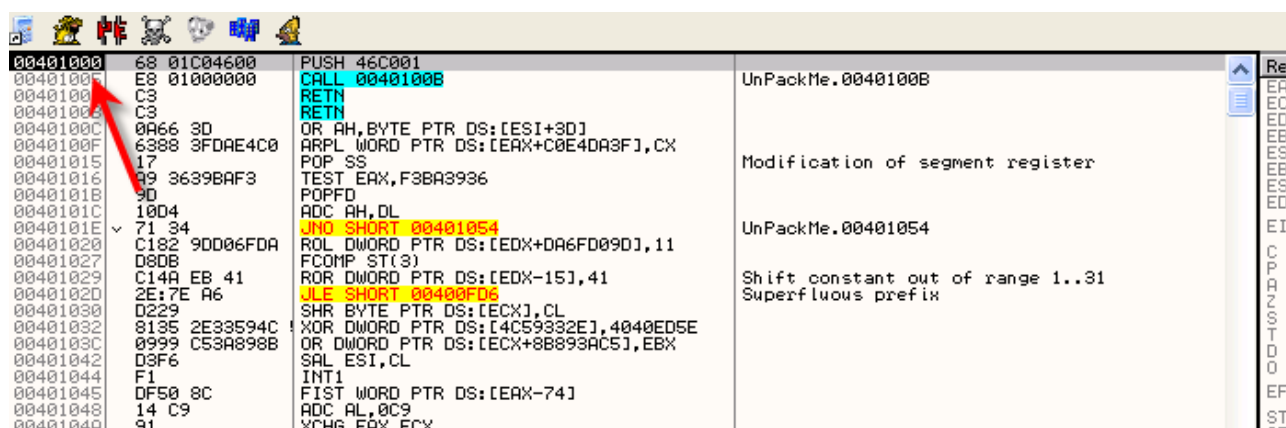
Antes que nada hay que tener mucho cuidado con este asprotect, pues muchas veces por poner BPs o HB los detecta y empieza con la cantinela del error de proteccion, y tenes que cambiar a otro exe, asi que usaremos el OLLYBONE para llegar al oep

Antes que nada expliquemos que es el OLLYBONE, es un plugin que por medio de un DRIVER sys, simula el BREAKPOINT ON EXECUTION, lo que hacíamos con el OLLYDBG PARCHEADO PARA BUSCAR OEPS, con la ventaja de que es instantaneo y no se pierde tiempo, la desventaja es que solo se puede colocar el break en las secciones del exe, o sea que para visual basic no sirve, pero bueno, lo han hecho para llegar a OEPs y lo usaremos para eso, antes que nada, debemos destildar para que funcione, la excepcion.



Si no, no funciona.

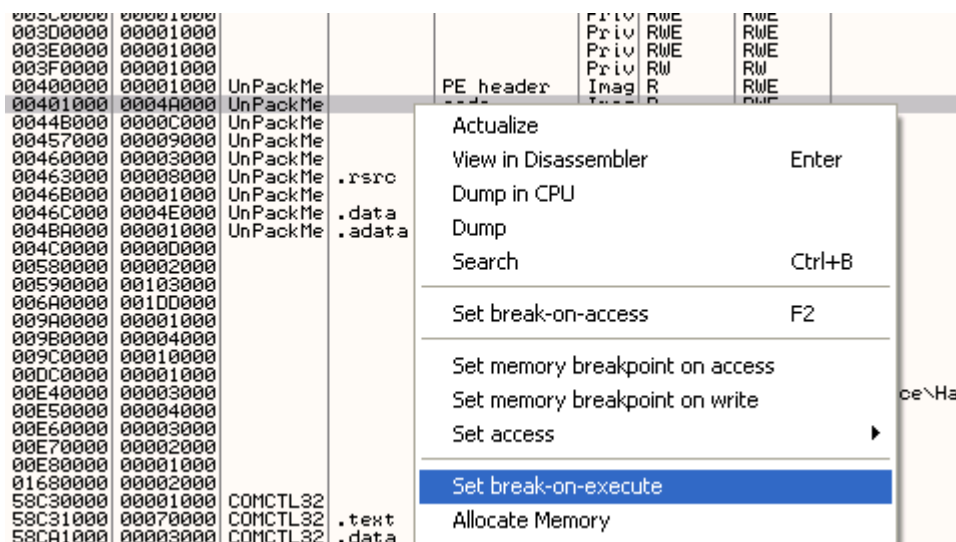
Las otras las marcamos todas, asi llegaremos a el facilmente.



Antes de poner el BREAK ON EXECUTE vemos que asprotect arranca desde la primera seccion, por lo cual tenemos que tracear unas lienas hasta que salgamos de ella.

0046C001	60	PUSHAD	
0046C002	E8 03000000	CALL 0046C00A	UnPackMe.0046C00A
0046C007	- E9 EB045D45	JMP 45A3C4F7	
0046C00C	55	PUSH EBP	
0046C00D	C3	RETN	
0046C00E	E8 01000000	CALL 0046C014	UnPackMe.0046C014
0046C013	EB 5D	JMP SHORT 0046C072	UnPackMe.0046C072
0046C015	BB EDFFFFFF	MOV EBX,-13	
0046C01A	030D	ADD EBX,EBP	
0046C01C	81FB 00C00600	SAR EBX,6C000	

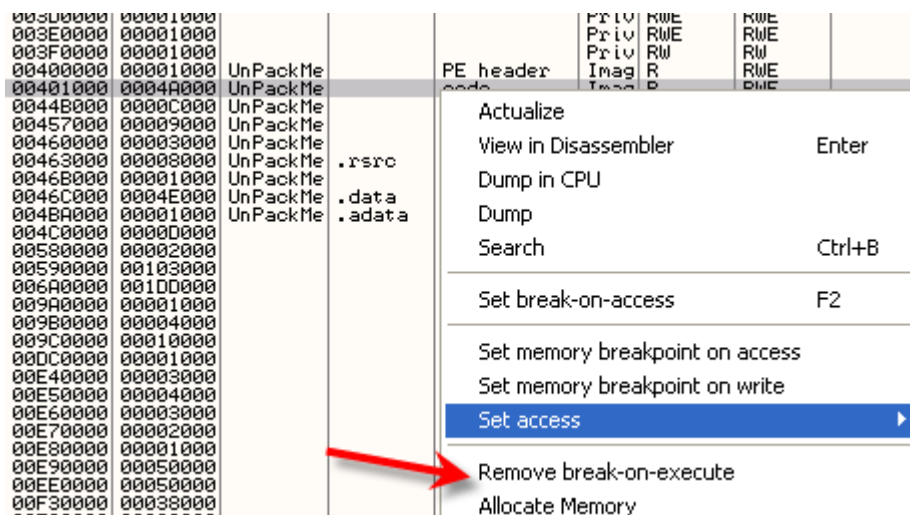
Ali esta despues de apretar 4 o 5 veces F7 ya estamos fuera de la primera seccion asi que podemos poner el BREAK ON EXECUTE vamos a M al mapa de memoria.



Para aquí

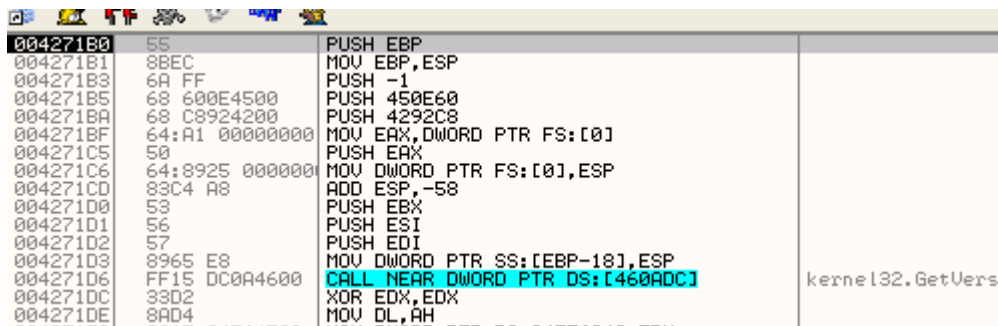
00401014	C3	RETN	
00401015	22D 2A0DB2D2	AND CH,BYTE PTR DS:[D2B20D2A]	
0040101B	2B0A	SUB ECX,DWORD PTR DS:[EDX]	
0040101D	11BB 18689190	ADC DWORD PTR DS:[EBX+90916818],EDI	
00401023	1D 26C0D0B0	SBB EAX,B0D0C026	
00401028	842F	TEST BYTE PTR DS:[EDI],CH	
0040102A	2C 4E	SUB AL,4E	

para cualquier otro packer este seria el OEP, pero asprotect siempre ejecuta un RET y vuelve al packer, asi que debemos quitar el BREAK ON EXECUTE.



Apreto f7 para salir de la seccion y lo coloco nuevamente y doy RUN.

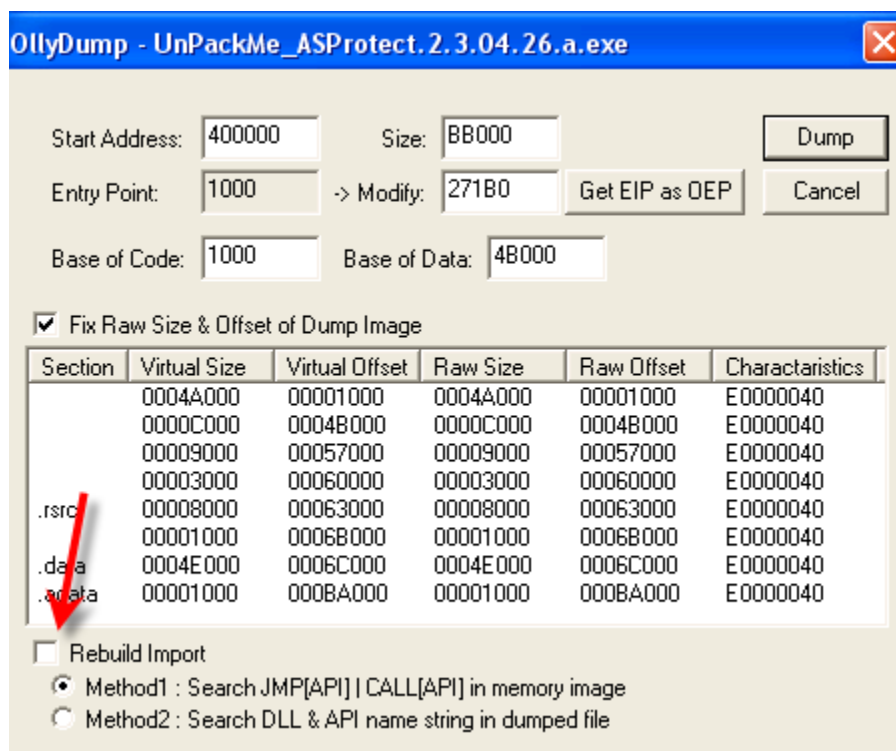
Ahora si ya para en el OEP, debemos recordar que todo este trabajo lo hace mediante un driver, asi que si queremos tracear o continuar ejecutando el programa debemos deshabilitar el break on execute si no dara un error y tendremos que repetir todo nuevamente.



```
004271B0 55      PUSH EBP
004271B1 8BEC    MOV EBP,ESP
004271B3 6A FF   PUSH -1
004271B5 68 60E45000  PUSH 450E60
004271B8 68 C8924200  PUSH 4292C8
004271BF 64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004271C5 50      PUSH EAX
004271C6 64:8925 00000000  MOV DWORD PTR FS:[0],ESP
004271CD 83C4 A8    ADD ESP,-58
004271D0 53      PUSH EBX
004271D1 56      PUSH ESI
004271D2 57      PUSH EDI
004271D3 8965 E8    MOV DWORD PTR SS:[EBP-18],ESP
004271D6 FF15 DC0A4600  CALL NEAR DWORD PTR DS:[460ADC]
004271DC 33D2     XOR EDX,EDX
004271DE 8AD4     MOV DL,AH
```

Asi que recordar remover el break on execute antes de continuar traceando o haciendo cualquier otra cosa.

Bueno ahora dumpeemos, con el OLLYDMP.

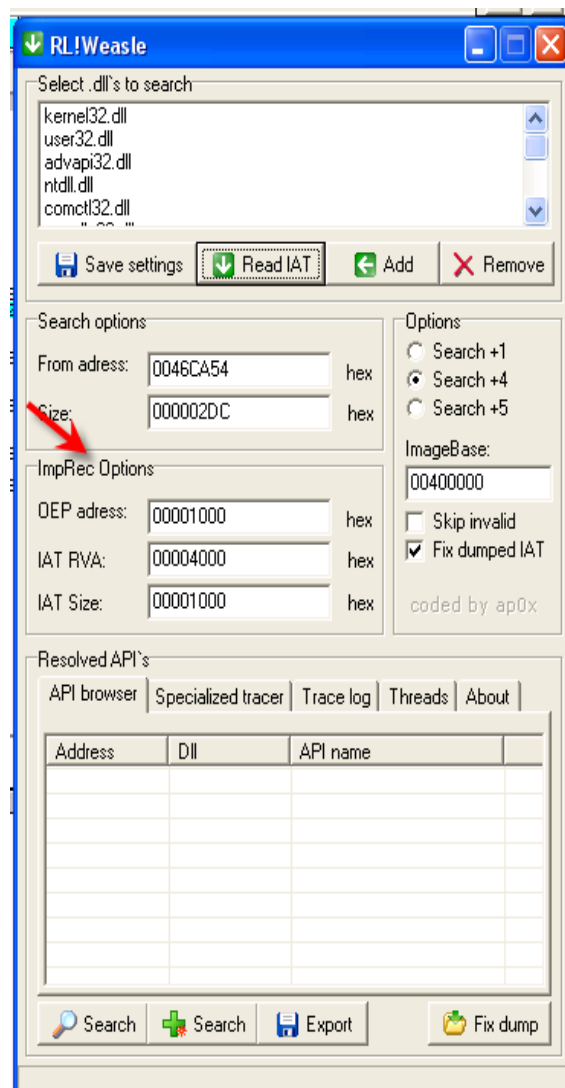
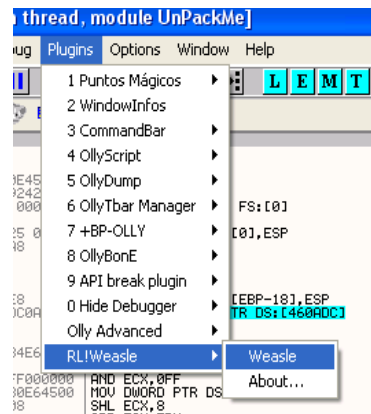


Le desmarcamos la tilde de rebuild imports, ahora busquemos el inicio y final de la tabla.

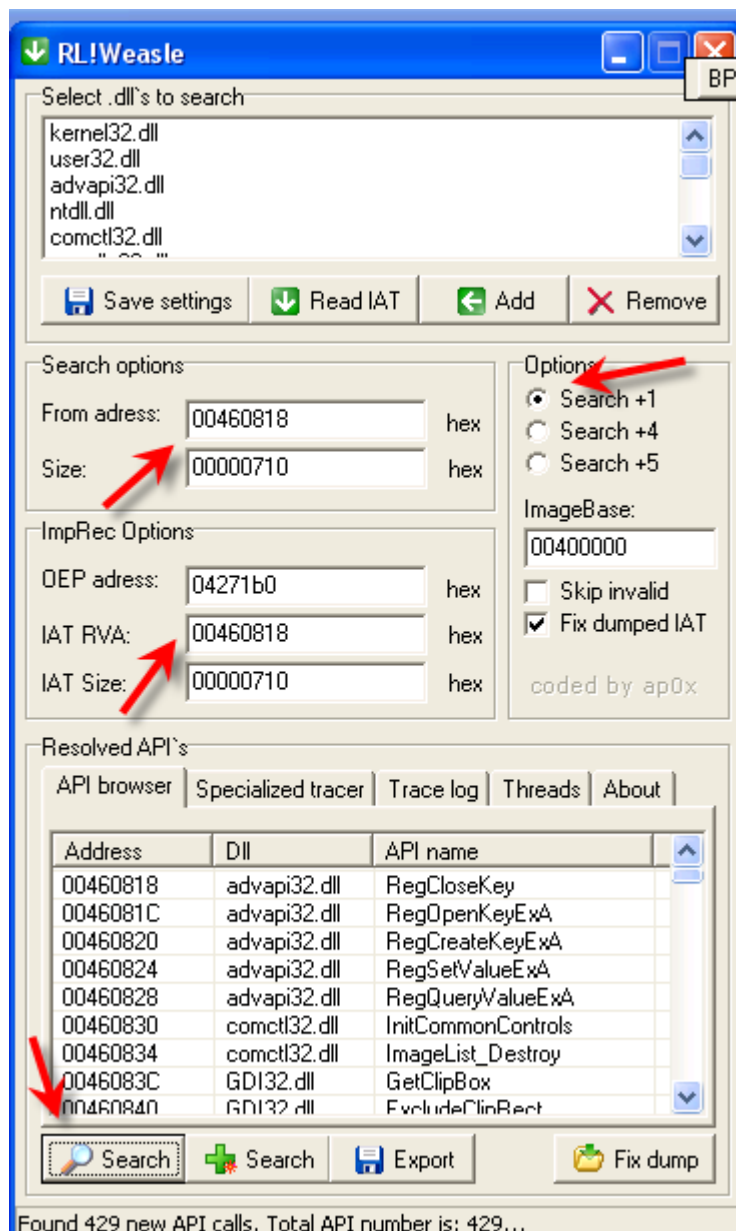


**OEP = 4271b0**  
**INICIO = 460818**  
**LARGO= 710**

Ahora en vez de utilizar el IMP REC trataremos de usar el Weasle que supuestamente sirve para reparar IATs veremos si sirve.



Bueno alli tenemos para llenar los valores que colocariamos en el IMP REC.

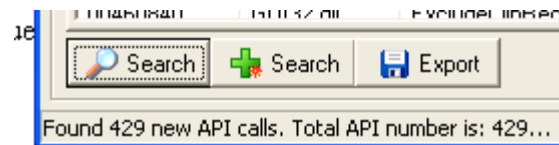


Igual tuve que copiar el inicio de IAT y el size en SEARCH OPTIONS- FROM ADDRESS y SIZE cambiar la tilde de OPTIONS a search +1, ahi si leyo bien todo, aunque algunas veces me ha faltado alguna dll, como la de la ultima entrada , que la agregue a mano con el boton ADD ya que me fije en la referencia del OLLY que era.

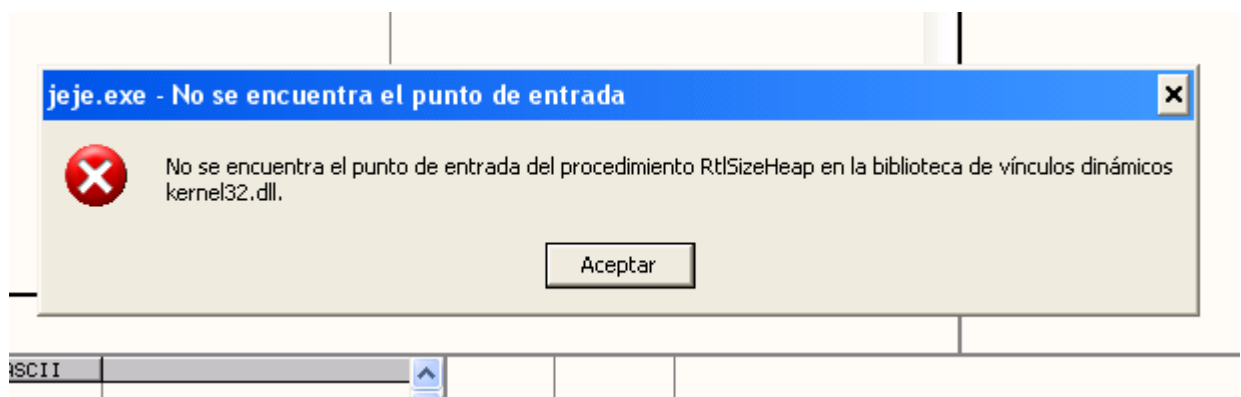
Address	Hex dump	ASCII
00460EFC	03 0E 52 77 33 0F 52 77	00Rw3*Rw
00460F04	40 A6 54 77 F1 A7 54 77	@3Tw:3Tw
00460F0C	92 9C 4F 77 6F 57 52 77	#E0w0wRw
00460F14	99 33 4E 77 B2 5D 4E 77	03Nw0INw
00460F1C	90 C0 5A 77 00 00 00 00	E'zw....
00460F24	F3 F0 CC 74 00 00 00 00	%-ft....
00460F2C	0B 00 A5 47 E8 D5 54 8C	0.0Gb'Ti
00460F34	E7 86 C4 B9 00 00 9D 19	f3-jl...0↓
00460F3C	BA 8C 90 EF B9 B2 8C 00	ll te'j8I.
00460F44	FE 00 91 7B FD B2 7E AB	■.at°8'z

Address	Disassembly	Comment
00435D38	JMP NEAR DWORD PTR DS:[460F24]	oledlg.OleUIBusyA

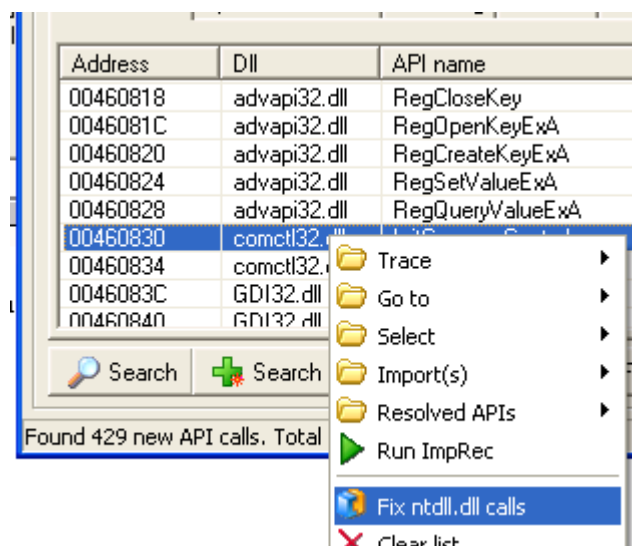
La oledlg.dll y que no estaba en la lista superior de las dll con las que trabaja el programa así que fui al botón ADD y tipee oledlg.dll y hice un search de nuevo y todo correcto, recordemos que es una beta aún y que puede tener alguna falla, veamos si repara bien, si no esperaremos la versión final jeje.



Ahí dice que de las 429 hay resueltas 429 así que veamos que ocurre.



Glup, ah me faltaba un detalle ya veo que está tomando las de ntdll mal, así que hay una opción para repararlas que siempre hay que usar.



Ahora sí, borro el exe que había reparado antes y el bak que me había creado lo vuelvo a exe, para quitar lo que había hecho mal y ahora sí FIX DUMP y todo correcto al menos arranca, lo que si casca en los antidumps que será el trabajo que realizarán ustedes y será de la siguiente forma.



004271B0	55	PUSH EBP	
004271B1	8BEC	MOV EBP,ESP	
004271B3	6A FF	PUSH -1	
004271B5	68 600E4500	PUSH 450E60	
004271BA	68 C8924200	PUSH 4292C8	
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
004271C5	50	PUSH EAX	
004271C6	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	
004271D1	56	PUSH ESI	
004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	kernel32.GetVersion
004271DC	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
004271E6	8BC8	MOV ECX,EAX	

Aquí tenemos el dumpeado que reparamos con el weasle, y quedo la iat perfecta todo correcto, el tema es que si hacemos search for intermodular calls vemos.

Address	Disassembly	Destination
00423310	CALL 019B0000	
00423A4F	CALL NEAR DWORD PTR DS:[460A00]	ntdll.RtlFreeHeap
00423B1C	CALL NEAR DWORD PTR DS:[4609FC]	ntdll.RtlAllocateHeap
00423C43	CALL NEAR DWORD PTR DS:[4609FC]	ntdll.RtlAllocateHeap
00423CA8	CALL NEAR DWORD PTR DS:[4609F8]	ntdll.RtlReAllocateHeap
00423E5C	CALL 019B0000	
00423E96	CALL NEAR DWORD PTR DS:[460B5C]	ntdll.RtlGetLastWin32Error
004249B6	CALL NEAR DWORD PTR DS:[4609F4]	ntdll.RtlSizeHeap
00425003	CALL NEAR DWORD PTR DS:[460B98]	kernel32.InterlockedIncrement
004251B3	CALL NEAR DWORD PTR DS:[460B98]	kernel32.InterlockedIncrement
004251C7	CALL NEAR DWORD PTR DS:[460B94]	kernel32.InterlockedDecrement
0042520B	CALL NEAR DWORD PTR DS:[460B94]	kernel32.InterlockedDecrement
004252D4	CALL NEAR DWORD PTR DS:[460B94]	kernel32.InterlockedDecrement
004252FB	CALL NEAR DWORD PTR DS:[460978]	kernel32.GetLocalTime
00425306	CALL 019B0000	
0042535E	CALL 019B0000	
004259D1	CALL NEAR DWORD PTR DS:[460A54]	kernel32.InitializeCriticalSection
004259E8	CALL NEAR DWORD PTR DS:[460A3C]	ntdll.RtlEnterCriticalSection
004259F5	CALL NEAR DWORD PTR DS:[460A44]	ntdll.RtlLeaveCriticalSection
00425B2B	CALL 019B0000	
00425B3F	CALL 019B0000	
00425B53	CALL 019B0000	
00425BF1	CALL 019B0000	
00425C9C	CALL 019B0000	
00425CA6	CALL NEAR DWORD PTR DS:[460B5C]	ntdll.RtlGetLastWin32Error
00425D71	CALL NEAR DWORD PTR DS:[460A54]	kernel32.InitializeCriticalSection
00425D8B	CALL NEAR DWORD PTR DS:[460A3C]	ntdll.RtlEnterCriticalSection
00425D8B	CALL NEAR DWORD PTR DS:[460A44]	ntdll.RtlLeaveCriticalSection
00425E13	CALL NEAR DWORD PTR DS:[460B98]	kernel32.InterlockedIncrement
00425E27	CALL NEAR DWORD PTR DS:[460B94]	kernel32.InterlockedDecrement
00425E6B	CALL NEAR DWORD PTR DS:[460B94]	kernel32.InterlockedDecrement
00425F34	CALL NEAR DWORD PTR DS:[460B94]	kernel32.InterlockedDecrement
004271B0	PUSH EBP	(Initial CPU selection)
004271D6	CALL NEAR DWORD PTR DS:[460ADC]	kernel32.GetVersion
0042723E	CALL 019B0000	
004272D5	CALL 019B0000	

Los antidumps de asprotect que en mi maquina son todos CALLs a la misma direccion o sea 019b0000 que es una direccion de asprotect que no existe en el dumpeado, la tarea para el hogar sera la siguiente y a no asustarse pues les dare algunos tips de ayuda, el tema es que los que quieran, tienen 15 dias para hacer un script que repare los antidumps, de todos los que yo reciba, usare el que yo crea que es el mejor mas simple y mas efectivo en la parte siguiente mencionando al autor, o sea el que yo piense que es el mas completo, simple y efectivo, a mi leal saber y entender, yo sere el juez.

Pueden usar como ayuda los desempacados que hicimos anteriormente para ver a que apis van estos calls en cada caso, pero el script debe funcionar en el original parado en el oep y reparar todos estos calls este es el trabajo del script, debe quedar todo perfecto para luego poder copiar a mano con BYNARY COPY la seccion entera ya reparada al dumpeado y que quede funcionando normalmente, el script debe funcionar en cualquier maquina, pues yo lo probare aqui, y debe funcionar, y si nadie lo soluciona en 15 dias me haran trabajar a mi jeje, espero que no ocurra eso, envien el script con su nombre o nick bien claro asi el ganador figura en la parte 52 y creo que lo merecera pues es un lindo trabajito.

O sea el script debe reparar esos calls en el original, el bynary copy paste no lo debe hacer el script

logicamente si no que lo hare yo a mano aqui, pero debe dejar todo correcto para que luego del copy paste funcione correcto el dumpeado.

ALGUNAS AYUDAS vayamos al original parado en el OEP.

004272CA	C745 D0 000000	MOV DWORD PTR SS:[EBP-30],0
004272D1	8D45 A4	LEA EAX,DWORD PTR SS:[EBP-5C]
004272D4	50	PUSH EAX
004272D5	E8 268D5801	CALL 019B0000
004272DA	D9F6	FDECSTP
004272DC	45	INC EBP
004272DD	BA01	RN RVTE PTR DS:[ECX] 1

Si comparamos con uno de los dumpeados que reparamos sabemos que algunos estan basados en el mismo programa, miremos en uno que sea similar la misma direccion a ver que hay alli.

004272D5	FF15 80094600	CALL NEAR DWORD PTR DS:[460980]	kernel32.GetStartupInfoA
004272DB	F645 D0 01	TEST BYTE PTR SS:[EBP-30],1	
004272DF	74 0A	JE SHORT 004272EB	UnPackMe.004272EB

Vemos que es un call a una api en este caso GetStartupInfoA y que al retornar, ya que es un comando de 6 bytes lo hace en 4272db, y en el que esta protegido con asprotect tambien debe retornar a la misma direccion lo que ocurre es que fue reemplazado por un comando de 5 bytes siendo el 6 basura, asi que si ponemos un BP en la direccion de retorno del call en el empackado con asprotect siempre debe ser un byte mas que el que indica la siguiente linea que se ve, en este caso la siguiente linea es

```
004272D5 E8 268D5801 CALL 019B0000
004272DA D9F6 FDECSTP
```

pues el BP para que pare al retornar de la api, debe ser BP 4272db y alli parara al retornar, es muy importante usar BPMs aquí o sea si yo quisiera resolver esto, pensaria que para tracear y no volverme viejo ya que es una rutina larguísima, en este caso en algun momento debe acceder a los bytes de la api, aunque sea como en este caso para leerlos para copiarlos a otro sitio, asi que se que en este caso a la api GetStartupInfoA le pongo un BPM ON ACCESS en las primeras lineas de la api, con eso descubri en mi maquina donde lee la api correcta, luego debo hallar el momento que cambia este CALL 019B0000 ya que luego de leer los bytes de la api los copia a otro lugar para ejecutarlos, asi que eso a mi no me interesa, solo cuando cambia el CALL 19b0000 por el call a ese nuevo lugar, eso tambien puede usarse un BPM ON WRITE y asi obtengo el lugar donde quiere modificar el call, con lo cual yo puedo alli accionar el script para que guarde lo que yo quiero para reparar el call y reemplazarlo por los bytes correctos y no lo que quiere guardar el jeje, esa seria la idea a ver quien la hace mejor.

Tendran que luchar y pensar que debe funcionar en cualquier maquina, asi que adelante y a trabajar espero que alguien lo resuelva tienen hasta el 17 de agosto de 2006 como fecha limite. Ademas de la mencion y el uso del script del ganador, se mencionaran los nombres de todos los que enviaron scripts que funcionan aunque no hayan ganado, como premio consuelo jeje.

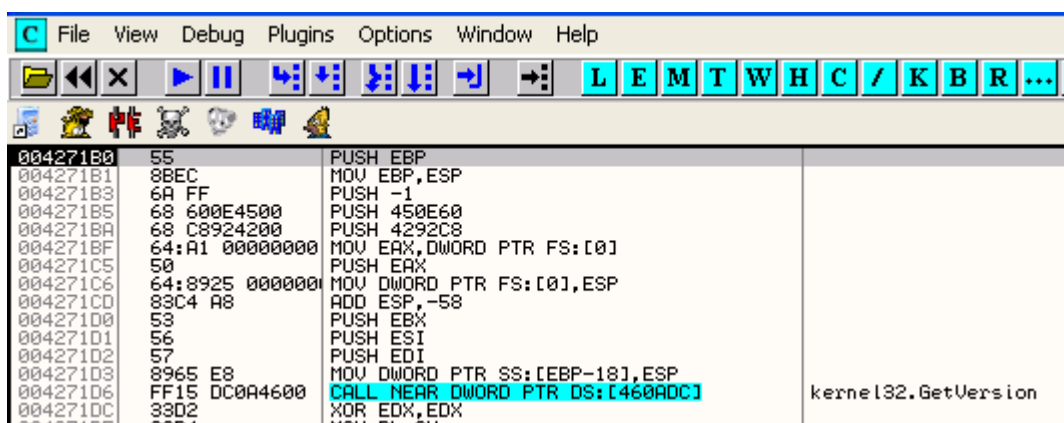
Suerte y good work  
Ricardo Narvaja  
02/08/06

## INTRODUCCION AL CRACKING CON OLLYDBG PARTE 52

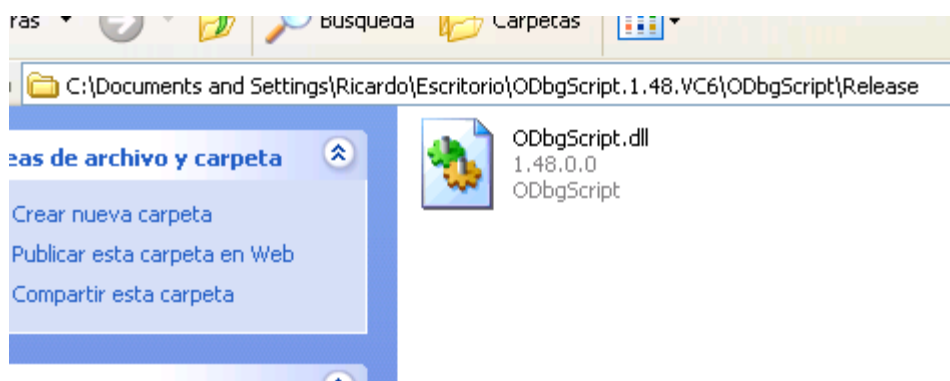
Bueno el concurso termno y creo que no tuvo mucha repercusion quizas porque era dificil, pero bueno el ganador del concurso ha sido HIEI que mando el script que repara los calls del crackme asprotect de la parte anterior.

De cualquier manera les quiero comentar que a muchos el unpackme asprotect no les corre a pesar de usar el plugin OLLYADVANCED ya que en el mismo, la tilde que crea el driver antiRDTSC no corre en todas las maquinas y sin eso no corre el unpackme, de cualquier manera, lo que he visto es que la ultima version de asprotect, en programas que he visto por ahi, no le han aplicado dicha proteccion y corren perfectamente en OLLYDBG, al menos por ahora no vi programas que la usen y que necesiten el antiRDTSC para correr, probablemente para no tener problemas de compatibilidad o la han quitado o han disminuido con lo cual no es problematica.

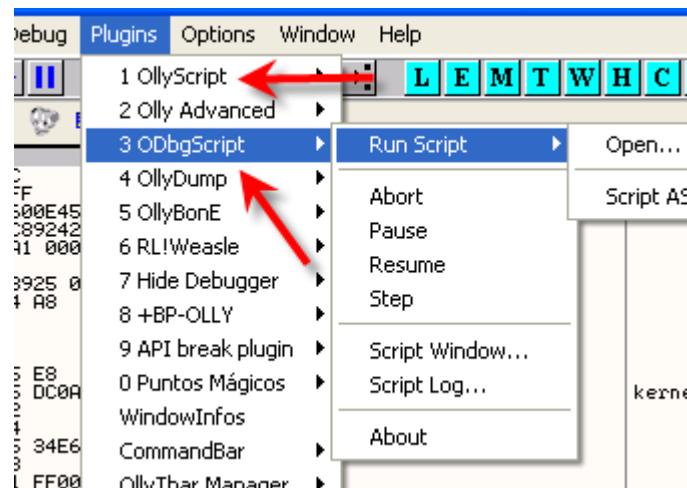
Pero a los que si les funciona el driver RDTSC del ollyadvanced pueden llegar al OEP facilmente, y una vez alli aplicar el script que esta adjunto.



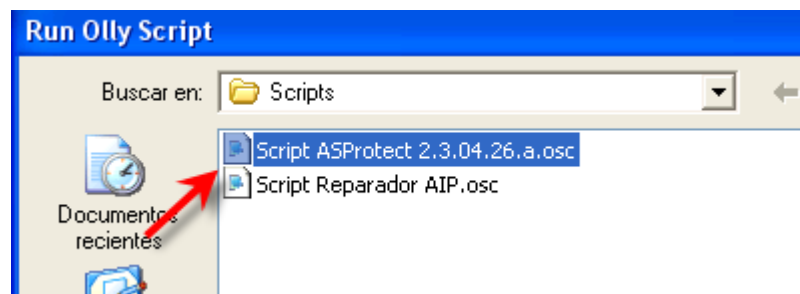
Alli estamos en el OEP, y cuando aplicamos el script de HIEI me dice que mi version de OLLYSCRIPT es muy antigua que me renueve, asi que busco la mas nueva que es la que esta adjunta y ahora se llama OdbgScript que continuo haciendo Epsilon ya que el autor original SHAG lo descontinuo, asi que colocho la dll en la carpeta plugins.



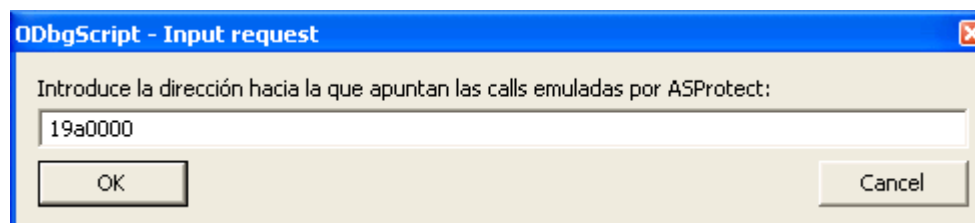
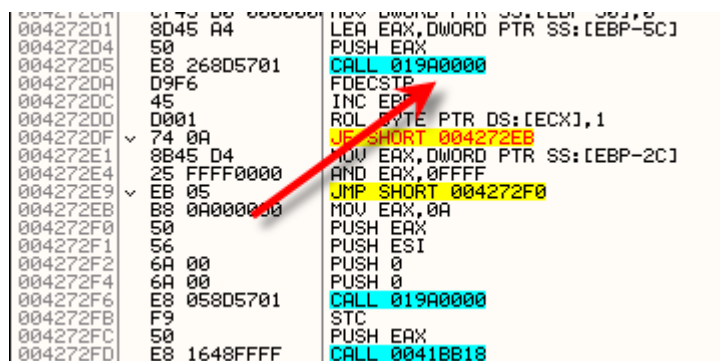
Y reinicio el OLLY y llego nuevamente al OEP, recordar desactivar el BREAK ON EXECUTE sino traera problemas.



Vemos que a pesar de ser la continuacion del OLLYSCRIPT ambos aparecen en el menu asi que elijo el nuevo y busco el script de HIEI.

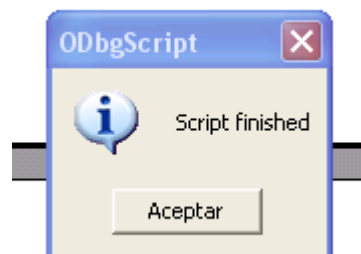
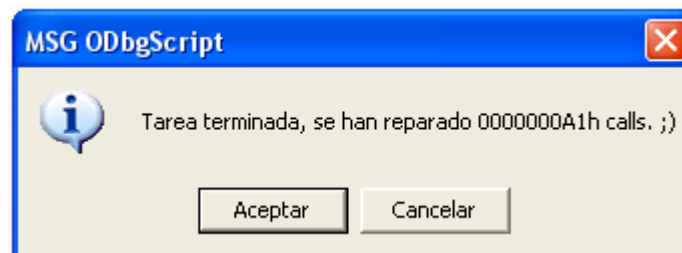


Y apenas arranca me pregunta la direccion donde van todos esos calls que en mi maquina es 19a0000.



Al aceptar empieza a trabajar.

Y luego de un rato



004272C4	C745 00 000000	MOV DWORD PTR SS:[EBP-30],0	
004272D1	8D45 A4	LEA EAX,DWORD PTR SS:[EBP-5C]	
004272D4	50	PUSH EAX	
004272D5	FF15 80094600	CALL NEAR DWORD PTR DS:[460980]	kernel32.GetStartupInfoA
004272DB	F645 D0 01	TEST BYTE PTR SS:[EBP-20],1	
004272DF	74 0A	JE SHORT 004272EB	UnPackMe.004272EB
004272E1	8B45 D4	MOV EAX,DWORD PTR SS:[EBP-2C]	
004272E4	25 FFFF0000	AND EAX,0FFFF	
004272E9	EB 05	JMP SHORT 004272F0	UnPackMe.004272F0
004272EB	B8 0A000000	MOV EAX,0A	
004272F0	50	PUSH EAX	
004272F1	56	PUSH ESI	
004272F2	6A 00	PUSH 0	
004272F4	6A 00	PUSH 0	
004272F6	FF15 9C0B4600	CALL NEAR DWORD PTR DS:[460B9C]	kernel32.GetModuleHandleA
004272FC	50	PUSH EAX	
004272FD	E8 1648FFFF	CALL 0041BB18	UnPackMe.0041BB18
00427302	8945 A0	MOV DWORD PTR SS:[EBP-60],EAX	
00427305	50	PUSH EAX	
00427306	E8 A50A0000	CALL 00427DB0	UnPackMe.00427DB0
0042730B	EB 21	JMP SHORT 0042732E	UnPackMe.0042732E

Allí vemos los call que antes se dirigían a 19a0000 en mi máquina ahora están reparados, perfecto script Hiei, felicitaciones.

Aquí está el script que está comentado por el propio autor, abajo realice mis comentarios.

/\*

-

.: [CracksLatinoS] .:

Script realizado por: Hiei.

Script para: Eliminar la protección AIP del ASProtect SKE v2.3

Objetivo: UnPackMe\_ASProtect.2.3.04.26.a.exe

Configuración: ODBGScript v1.3x o superior, ejecutar desde el OEP  
e Ignorar todas las excepciones.

Fecha: 05/AGOSTO/2006

**==[ Comentario del Script ]==**

**Agradecimientos a: 'Ricardo Narvaja' y a 'marciano' (porque usé un poquito de su lógica para el implementar el motor de búsqueda).**

---

**-  
\*/**

**var oep  
var codebase  
var codesize  
var base\_aspr  
var base\_aip  
var ini\_iat  
var dir  
var dir\_iat  
var sig  
var dest  
var api  
var cont**

**cmp \$VERSION,"1.30" // Consulto la versión del OllyScript.  
jb err\_version  
ask "Introduce la dirección hacia la que apuntan las calls emuladas por ASProtect:"  
cmp \$RESULT,0  
je salir  
mov base\_aip,\$RESULT // Guardo dirección introducida.  
mov oep,eip // Guardo el OEP.  
gmi eip,codebase // Busco la dirección de inicio de la sección Code.  
mov codebase,\$RESULT // Guardo la dirección de inicio.  
gmi eip,codesize // Busco el tamaño de la sección Code.  
mov codesize,\$RESULT // Guardo el tamaño de la sección.  
add codesize,codebase // Al tamaño le sumo la dir. de inicio.  
mov ini\_iat,460814 // Guardo la dirección de inicio de la IAT.  
mov base\_aspr,[46C048] // En la dir[46C048] se guarda la base de la sección donde  
se conoce a que api deben ir las calls.  
add base\_aspr,3B02E // A esa base le sumo una constante para obtener la  
dirección donde se ve la api.  
bphws base\_aspr,"x" // Pongo un HBP en la dirección donde se ve la api.  
jmp buscar**

**buscar:  
find codebase,#E8????????# // Busco calls a partir de la sección Code.  
cmp \$RESULT,0 // Si no se encuentran calls, termino el proceso.  
je no\_calls  
mov dir,\$RESULT // Si se encuentran calls, guardo la dirección.  
mov sig,dir // Muevo la dirección a otra variable.  
mov dest,dir // Muevo la dirección a otra variable.  
add sig,5 // sig contiene la dirección de la instrucción que sigue al  
call.**

```

    inc dest                                // Incremento dest, para obtener los opcodes después del
E8.                                         //
    mov dest,[dest]                        // Tomo el offset codificado en el call, luego del opcode E8.
    add dest,sig                           // Ahora dest tiene la dirección destino del call.
    cmp dest,base_aip                     // Es un call a ASProtect?.
    je ejecutar                            // Si no es, actualizo el puntero de búsqueda.
    inc dir                                // Muevo el puntero y busco de nuevo.
    mov codebase,dir
    jmp buscar

ejecutar:                                // Si estoy aca, entonces el call iba a ASPr.
    mov eip,dir                            // Muevo a eip la dirección donde encontré el Call a ASPr.
    run                                    // Y ejecuto.

eob comprobar                             // Si hay un bp, la etiqueta 'comprobar' toma el
control.

comprobar:                                // El BP es en la zona esperada?.
    cmp eip,base_aspr
    jne inesperado
    mov api,edx                            // Si el BP es en la zona esperada, guardo el valor de
la API que está en EDX.
    jmp buscar_api                        // Busco la API en la IAT.

buscar_api:
    cmp ini_iat, 460F28                    // Son el inicio y final de la IAT iguales?.
    je error                              // Si es así, no encontré la api en la IAT.
    cmp [ini_iat],api                     // Busco la API en el valor que lleva el puntero.
    je reparar                            // Si la encuentro la reparo.
    add ini_iat,4                          // Si no, le sumo 4 al puntero para evitar errores de
búsqueda.
    jmp buscar_api                        // Y sigo buscando.

reparar:
    mov dir_iat,ini_iat                    // Guardo la dirección de la IAT donde encontré la API.
    ref dir                               // Busco referencias a esa dirección.
    cmp $RESULT,0                         // Si encuentro ref. entonces debo ensamblar un jmp.
    jne reparar_jump
    eval "Call dword[{{dir_iat}}]"        // Si no encuentro ref. entonces debo ensamblar un
call.
    asm dir,$RESULT
    inc cont                              // Implemento un contador para reportar al final cuántas
calls se repararon.
    inc dir
    mov codebase,dir
    mov ini_iat,460814                    // Actualizo el puntero para que busque desde el inicio de la
IAT la próxima vez.

```

**jmp buscar**

```
reparar_jump:
    eval "Jmp dword[{dir_iat}]"           // Si estoy acá es porque tengo que ensamblar un
    jmp.
    asm dir,$RESULT
    inc cont
    inc dir
    mov codebase,dir
    mov ini_iat,460814                    // Actualizo el puntero para que busque desde el inicio de la
IAT la próxima vez.
    jmp buscar
```

```
inesperado:
    msg "Parada inesperada ¿Continúo?."
    cmp $RESULT,0
    je salir
    run
```

```
error:
    eval "Error. Por favor resolver a mano la call de la dirección: {dir}h."
    msg $RESULT
    run
```

```
no_calls:
    bphwc base_aspr
    eval "Tarea terminada, se han reparado {cont}h calls. ;)"
    msg $RESULT
    jmp salir
```

```
err_version:
    msg "Error. La versión de OllyScript es inferior a la versión solicitada."
    ret
```

```
salir:
    bphwc base_aspr
    mov eip,oep
    ret
```

---

Bueno creo que es claro con toda los comentarios que trae, lo que hace realmente es localizar para cualquier maquina, el lugar donde la rutina de asprotect, revela que api es la usada, y eso lo hace en vez de utilizar BPMs como se me ocurrio a mi, busca en que parte el programa guarda la base de dicha seccion que fue creada por el mismo asprotect

```
mov base_aspr,[46C048]           // En la dir[46C048] se guarda la base de la sección donde
se conoce a que api deben ir las calls.
```



```

add base_aspr,3B02E // A esa base le sumo una constante para obtener la
dirección donde se ve la api.
bphws base_aspr,"x" // Pongo un HBP en la dirección donde se ve la api.
jmp buscar

```

Allí se ve en 46c048 el programa tiene guardado la dirección de base de la sección donde revela la api, y una vez que uno sabe eso, sumándole una constante, pues hallará el punto para cualquier máquina, porque las secciones son iguales, solo cambian la dirección en cada máquina, así que al saber el inicio, sumándole una constante llegaremos a él, aquí el punto clave está 3B02e, más adelante del inicio, así que lo localiza y le coloca un HBP allí para que pare siempre que se ejecute.

Luego en la parte buscar lo que hace es verificar todos los calls desde el comienzo de la sección 401000 y fijarse si van a la zona de asprotect según el valor que nos pidió que le ingresemos.

```

ejecutar: // Si estoy acá, entonces el call iba a ASPr.
mov eip,dir // Muevo a eip la dirección donde encontré el Call a ASPr.
run // Y ejecuto.

```

Si es así, cambia el eip a la dirección del CALL A REPARAR y lo ejecuta y cuando salta una excepción salta a la etiqueta comprobar mediante el eob.

```

eob comprobar // Si hay un bp, la etiqueta 'comprobar' toma el
control.

```

Aquí comprueba si el Bp está en el lugar esperado, puede ocurrir que alguien se olvidó de quitar algún BP innecesario o de deshabilitar el BREAK ON EXECUTE y saltará una excepción molstando al script. Esta parte comprueba eso.

```

comprobar:
cmp eip,base_aspr // El BP es en la zona esperada?.
jne inesperado
mov api,edx // Si el BP es en la zona esperada, guardo el valor de
la API que está en EDX.
jmp buscar_api // Busco la API en la IAT.

```

Si estamos en el lugar esperando o sea en el punto que revela la IAT, pues va a ver en qué parte de la IAT está la entrada correspondiente a esta api que acaba de hallar.

```

buscar_api:
cmp ini_iat, 460F28 // Son el inicio y final de la IAT iguales?.
je error // Si es así, no encontré la api en la IAT.
cmp [ini_iat],api // Busco la API en el valor que lleva el puntero.
je reparar // Si la encuentro la reparo.
add ini_iat,4 // Si no, le sumo 4 al puntero para evitar errores de
búsqueda.
jmp buscar_api // Y sigo buscando.

```

Recorre toda la IAT fijándose cuál es la entrada correspondiente a la misma y cuando la halla salta a reparar.

Luego teniendo ya el call o jmp desde donde se llamo a la zona de asprotect, que hay que reparar, la api correcta, la entrada de la iat correspondiente, si es un call el que hay que reparar lo arregla cambiandolo por un CALL INDIRECTO que toma valores de dicha entrada de la IAT y si es un JMP hara un JMP INDIRECTO y repetira todo nuevamente hasta que no encuente mas nada que reparar, y deje todo listo.

Muy buen script muy claro y organizado por partes como a mi me gustan ademas de comentado lo cual pocos autores de scripts se toman el trabajo de hacer, y sirve para entender lo que esta haciendo el mismo en cada momento grande HIEI, te ganaste un viaje en el 60 a tigre, jeje pero eso si hay que venir aca, a que te den el premio, jajajajaja, un gran abrazo y gracias.

Bueno el nuevo concurso sera mas sencillo son dos partes , a ver si participan mas, la primera parte sera hacer un script para llegar al OEP del TPP PACK y que repare los Stolen bytes del mismo, es muy sencillo esta adjunto el unpackme y encima hay un tute de marciano del concurso 97 que explica como hacerlo a mano, asi que es coser y cantar, eso si solo esta permitido usar los plugins hide debugger 1.24 y HideOd y el Ollyscript aclarando que version usan del mismo preferentemente la ultima de ODbgScript, nada mas, asi unificamos todos en lo mismo.

La segunda parte sera hacer un script que repare la IAT del TPP pack ambos scripts son separados y se pueden anotar en la parte 1 o en la parte 2 o en ambas asi que queda a gusto de cada uno asi que.

PARTE1: Script que llegue al OEP y arregle los stolen bytes

PARTE2: Script que repare la IAT y la deje correcta.

Solo pueden usarse los 3 plugins mencionados.(bah no van a preguntar si tienen que quitar la command bar jeje)

Tienen hasta el dia 30 de agosto para enviar las soluciones recuerden que se pueden anotar en ambas partes o en 1 sola como quieran.

Espero soluciones, en un rato el tute de marciano estara en mi web aqui.

<http://storage.ricardonarvaja.com.ar/web/CONCURSOS%202004-2006/CONCURSO%2097/>

lo estoy por subir en un ratito.

Gracias por participar

Hasta la parte 53

Ricardo Narvaja

## INTRODUCCION AL CRACKING CON OLLYDBG PARTE 53

Bueno el ganador de ambas partes 1 y 2 del concurso de la parte anterior es Ularteck, que envio ambos scripts y una explicacion de la parte 1 en forma de tute que usaremos aquí el script ganador de la primera parte que sirve para reparar los stolen bytes es este, la explicacion esta debajo del mismo.

```
#####  
#####
```

<< CracksLatinoS - 2006 >>

Script hecho por: Ulaterck.

Descripción: Script realizado para el curso INTRODUCCIÓN AL CRACKING CON OLLYDBG DESDE CERO TOMO 52

por Ricardo Narvaja. La función de este script se lleva a cabo para la PARTE 1:

Encontrar el

OEP y reparar el Stolen Code.

Target: UnPackMe\_TPPpack.exe

Requisitos: ODBGScript 1.48 , HideDebugger 1.24 , HideOD. parados en el Entry Point destildar todas las

casillas de las excepciones menos la de KERNEL32, ya que el método utilizado es el de las excepciones de Ricardo.

- Antes de ejecutar el script anotar la ultima excepcion encontrada y anotarla ya que el script la pedirá.

En todo caso ver la explicacion que esta debajo del mismo.

```
#####  
#####
```

\*/

```
var dir_excep  
var Newoep  
var dir_JMP
```

```
var dir_CALL  
var oep  
var StartScan  
var Opcodes  
var temp  
var temp2  
var temp3
```

Datos:

```
mov Newoep, eip           // Guardamos la dirección del EntryPoint.  
                          // Anotando la ultima dirección del Script
```

```
ask "Introduzca la ultima excepción" // Sacamos la cajita de dialogo para introducir el
```

```

dato.
cmp $RESULT,0 // Comparamos si se ha introducido alguna
dirección.
je aviso // Si no nos dirigimos a la etiqueta aviso
mov dir_excep, $RESULT // Si introducimos una dirección la guardamos en
dir_excep.
jmp Inicio // Saltamos a la etiqueta Inicio para comenzar.

aviso:
msg "Ejecute de nuevo el script e introduzca una dirección válida."
jmp final

Inicio:
run // Ejecutamos el programa
eoe verifica // Si se produce una excepción nos dirigimos a la etiqueta verifica.

verifica:
cmp eip,dir_excep // una vez que estemos aquí por una excepción comprobamos que
sea la última.
je ultima // Si lo es nos vamos a la etiqueta ultima.
esto // Si no ejecuta SHIFT + F9 para pasar la excepción.
jmp Inicio: // Y saltamos a la etiqueta Inicio para buscar otra excepción.

ultima:
findop eip,#FFE0# // Al caer aquí por la última excepción buscamos el salto JMP
EAX al Stolen Code.
mov dir_JMP,$RESULT // Una vez encontrado aguardamos la dirección en
dir_JMP
bp dir_JMP // Ponemos un breakpoint (F2) en el salto JMP EAX.
esto // Pasamos la excepción con SHIFT + F9 para caer en el bp.
bc dir_JMP // Quitamos el bp del Salto JMP EAX.
sti // Ejecuta F7 para caer en el Stolen Code.

mov oep,eip // Guardamos la dirección del OEP robado.
mov StartScan,eip // Guardamos la dirección del oep también a StartScan que la
usaremos para buscar Calls.

BuscarCall: // Aquí empezaremos a buscar los calls directos para repararlos
para que a la hora de // de hacer binary copy y pegarlos al nuevo oep esos call directos
queden bien. // Nota ver MiniTuto.PARTE1.doc

findop StartScan,#E8# // Buscamos los calls directos que comienzan con su opcode E8
cmp $RESULT, 0 // Cuando no encuentre más $RESULT valdrá 0 y saltamos al
final del script.
je final
mov dir_CALL, $RESULT // Guardamos la dirección del primer call encontrado.
mov StartScan, $RESULT // Actualizamos desde donde queremos seguir buscando calls en
este caso desde el primer call encontrado.
add dir_CALL,1 // A la dirección del call encontrado le sumamos 1 esto para no

```

```

tomar el OPCODE E8.
mov Opcodes, [dir_CALL] // Movemos los opcodes de esa dirección a OPcodes.
add Opcodes, StartScan  // A esos Opcodes le Sumados la Dirección del Call.
add Opcodes, 5           // Al resultado le sumamos 5 y obtenemos la dirección que apunta
el call.

                        // Nota ver MIniTuto.PARTE1.doc para aclarar.

// Ahora le asignaremos un nuevo opcode a este call encontrado para que a la hora de hacer
binary paste en el nuevo
// oep ese call quede arreglado.

mov temp, StartScan    // Movemos la dirección que contiene StartScan a un temporal.
( StartScan = Dirección del call encontrado.)
sub temp, oep          // A la dirección de ese call a reparar le restamos la dirección
del oep del stolen code.
mov temp2, Newoep      // Movemos la direccion del nuevo oep que es el EntryPoint a un
temporal2.
add temp, temp2        // A ese temporal2 le sumamos el temporal1.
sub Opcodes, temp      // Ahora la dirección a la que apunta el CALL le restamos la
operación anterior
sub Opcodes, 5         // y a lo que quedó le sumamos 5 y obtenemos los opcodes nuevos
para ese call a reparar.

Editar:                // Empezaremos a Editar el call con esos nuevos opcodes.
mov temp3, StartScan   // Movemos la dirección del call a reparar contenida en
StartScan a un tercer temporal.
add temp3, 1           // A la direccion del call le sumamos 1 para no tamar en cuenta el
opcode E8.
mov [temp3], Opcodes   // Reparamos el call con los nuevos OPcodes.
jmp BuscarCall

final:
ret

```

---

Aquí a continuacion la explicacion de como funciona por el mismo Ularteck

PARTE 1: Encontrar el OEP y reparar el Stolen Code, por Ulaterck.

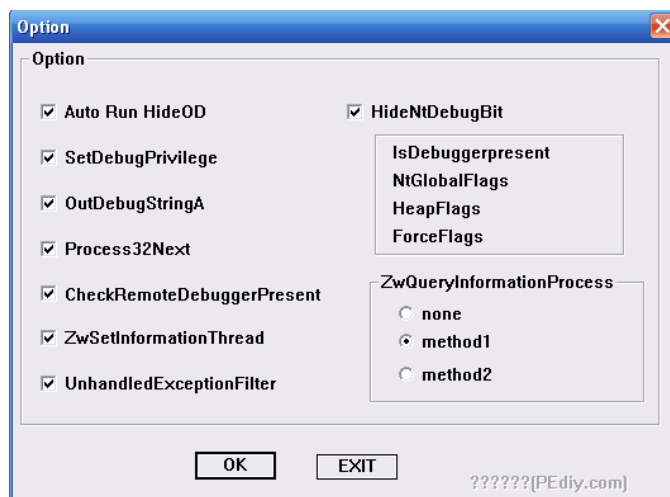
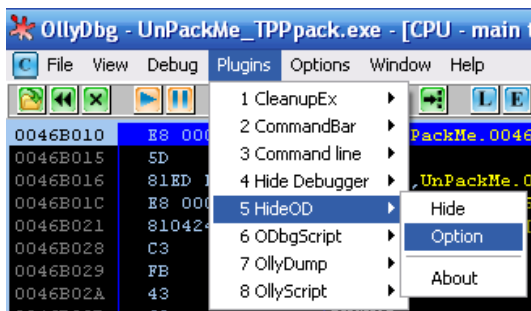
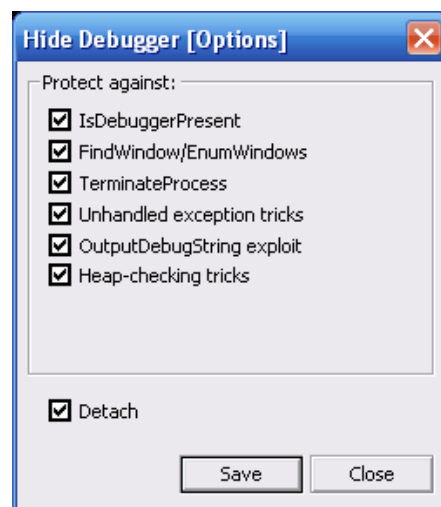
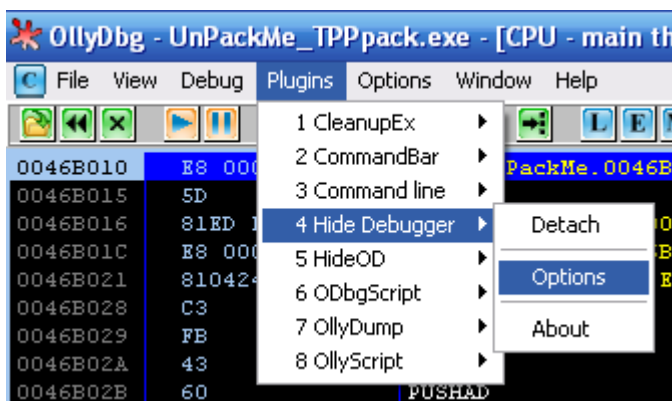
## INTRODUCCIÓN:

Aunque ya todo esta explicado en el Tutorial del Curso C97 N4 hecho por marciano, algunas cosas de las que el dice no me salen no se si es porque es otra versión del mismo archivo o no se, pero trataré de explicar brevemente el procedimiento que realiza el script y sea fácil de entender su funcionamiento.

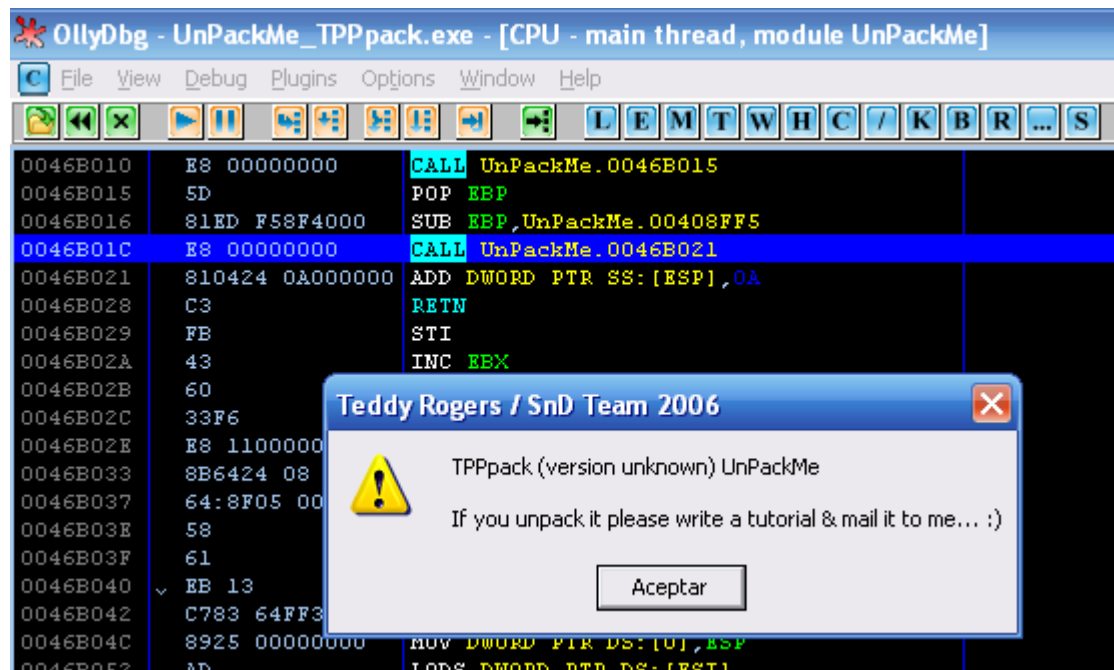
El método que use para encontrar el OEP es el de las excepciones ya que si a ustedes crackme les

da el error del que hablaba marciano mediante el antidebugging OpenProcess, entonces creo que este script no les funcionara pero bueno a mi no me sale ese error.

Los plugins que se valía usar los tengo configurados así.



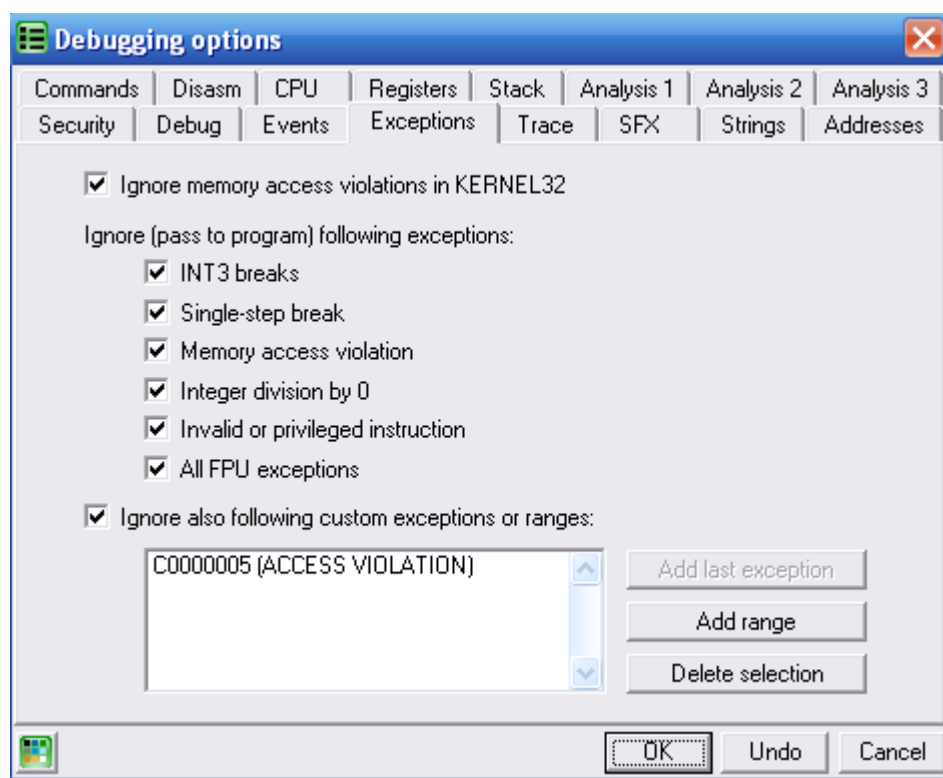
y Con eso el crackme me corre perfectamente.



El metodo utilizado para encontrar el OEP para este programa me agrada porque se parece mucho al de las nuevas versiones de Asprotect como la 2.1 SKE , 2.2 SKE , y la 2.3 SKE que tienen Stolen code

Así que nos puede ayudar mucho y la forma de reparar los calls es el 80% el metodo que usaremos cuando trabajemos con la máquina virtual de asprotect. así que de algo nos puede servir.

Configuramos la excepciones para que no pare en ninguna. ALT + O.



Ejecutamos el programa y una vez que arranca nos fijamos en el Log a ver cual fue la ultima

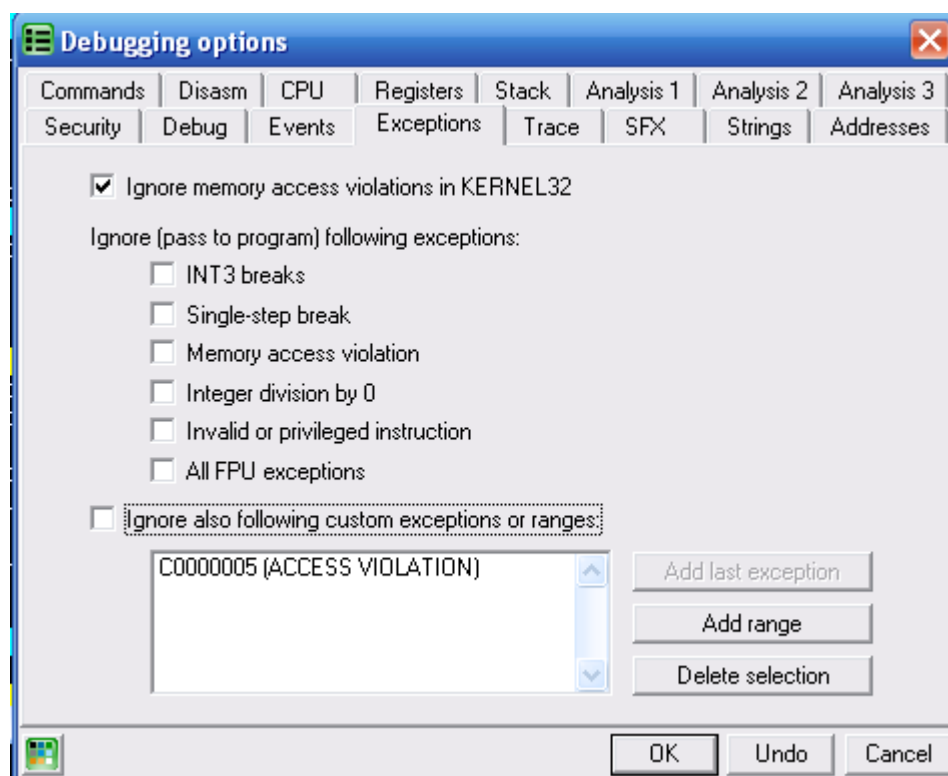
excepción.

```
74C00000 Module C:\WINDOWS\system32\oledlg.dll
774B0000 Module C:\WINDOWS\system32\ole32.dll
100064D8 Access violation when reading [00000000]
100065A2 Access violation when reading [00000000]
1000812F Access violation when reading [00000000]
10006B28 Access violation when reading [00000000]
100081D1 Access violation when reading [00000000]
10005F71 Access violation when reading [00000000]
10006014 Access violation when reading [00000000]
10008278 Access violation when reading [00000000]
10006D8E Access violation when reading [00000000]
10006E40 Access violation when reading [00000000]
100083D9 Access violation when reading [00000000]
1000847B Access violation when reading [00000000]
10008535 Access violation when reading [00000000]
0046D2CF Access violation when reading [00000000]
0046D36B Access violation when reading [00000000]
770F0000 Module C:\WINDOWS\system32\oleaut32.dll
5B150000 Module C:\WINDOWS\system32\uxtheme.dll
746B0000 Module C:\WINDOWS\system32\MSCTF.dll
73260000 Module C:\WINDOWS\system32\RICHED32.DLL
```

En mi caso fue **0046D36B** esta la usaremos para el script.  
Pero hagamos el procedimiento a mano para hallar el oep.

Reiniciamos

Presionamos ALT + O :



Y La configuramos como la imagen anterior .

Ejecutemos el programa con F9 y pasamos las excepciones con SHIFT + F9 hasta que estemos en el excepción en la dirección **0046D36B** o sea la ultima.



```

OllyDbg - UnPackMe_TPPack.exe - [CPU - main thread, module UnPackMe
File View Debug Plugins Options Window Help
[Icons] [L] [E] [M] [T] [W] [H] [C] [/] [K] [B]
0046D36B AD LODS DWORD PTR DS:[ESI]
0046D36C 90 NOP
0046D36D 2060 E8 AND BYTE PTR DS:[EAX-18], AH

```

Ahí estamos en la ultima así que presinamos ALT + M

Y ponemos un bp memory on acces en la sección code.

Address	Disassembly	Comment	Map	R	RWE
003B0000	00041000	UnPackMe	PE header		
00400000	00001000	UnPackMe			
00401000	0004A000	UnPackMe .text	code		
0044B000	0000C000	UnPackMe .rdata			
00457000	00009000	UnPackMe .data			
00460000	00003000	UnPackMe .idata			
00463000	00008000	UnPackMe .rsrc	resources		
0046B000	0001B000	UnPackMe .tt	SFX, imports		

Actualize  
 View in Disassembler Enter  
 Dump in CPU  
 Dump  
 Search Ctrl+B  
 Set break-on-access F2  
 Set memory breakpoint on access  
 Set memory breakpoint on write  
 Set access disk  
 Copy to clipboard disk  
 Sort by disk  
 Appearance disk

Y presionamos SHIFT + F9 para pasar la excepción y caemos aca.

```

[Icons] [L] [E] [M] [T] [W] [H] [C] [/] [K] [B]
004293A0 6A 00 PUSH 0
004293A2 68 00100000 PUSH 1000
004293A7 6A 00 PUSH 0
004293A9 FF15 A0094600 CALL DWORD PTR DS:[4609A0]
004293AF 85C0 TEST EAX,EAX
004293B1 A3 C4EB4500 MOV DWORD PTR DS:[45EBC4],EAX
004293B6 75 01 JNZ SHORT UnPackMe.004293B9
004293B8 C3 RETN
004293B9 E8 22000000 CALL UnPackMe.004293E0
004293BE 85C0 TEST EAX,EAX
004293C0 75 0F JNZ SHORT UnPackMe.004293D1
004293C2 A1 C4EB4500 MOV EAX,DWORD PTR DS:[45EBC4]
004293C7 50 PUSH EAX
004293C8 FF15 9C094600 CALL DWORD PTR DS:[46099C]
004293CE 33C0 XOR EAX,EAX
004293D0 C3 RETN
004293D1 B8 01000000 MOV EAX,1
004293D6 C3 RETN
004293D7 90 NOP
004293D8 90 NOP


```

Pero este no es el OEP y si nos fijamos en el stack:

```

0012FF48 008B0EA4 RETURN to 008B0EA4 from UnPackMe.004293A0
0012FF4C 00000002
0012FF50 00460613 ASCII "erm"
0012FF54 0040B5EE UnPackMe.0040B5EE
0012FF58 81B92558
0012FF5C 817CD020
0012FF60 00000202

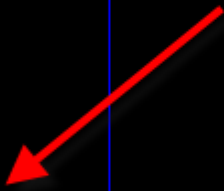
```



Vemos que ya se ha ejecutado código. Estoy ya lo hemos visto en las teorías que nos ha dado Ricardo

Vemos que la sección donde se ha ejecutado ya código está en 8B0EA4 que esta situada en la sección **8B0000**.

00460000	00003000	UnPackMe	.ldata		Imag	R		RWE	
00463000	00008000	UnPackMe	.rsrc	resources	Imag	R		RWE	
0046B000	0001B000	UnPackMe	.tt	SFX,imports	Imag	R		RWE	
00490000	00086000				Priv	RW		RW	
00520000	00001000				Priv	RW		RW	
00530000	00001000				Priv	RW		RW	
00540000	00004000				Priv	RW		RW	
00550000	00004000				Priv	RW		RW	
00560000	00002000				Map	R		R	
00570000	00003000				Priv	RW		RW	
00580000	00002000				Map	R		R	
00590000	00102000				Map	RW		RW	
006A0000	00016000				Map	R		R	\De
006C0000	0003D000				Map	R		R	\De
00700000	00041000				Map	R		R	\De
00750000	00006000				Map	R		R	
00760000	00041000				Map	R		R	
007B0000	00001000				Priv	RW		RW	
007E2000	00002000				Priv	RW		RW	
007FD000	0002C000				Priv	RW		RW	
008B0000	00001000				Priv	RW		RW	
00AB0000	00004000				Map	R E		R E	
00B70000	00002000				Map	R E		R E	
00B80000	00103000				Map	R		R	
00C90000	00092000				Map	R E		R E	
00F90000	00001000				Priv	RW		RW	
10000000	00028000				Priv	RWE		RWE	
58C30000	00001000	COMCTL32		PE header	Imag	R		RWE	
58C31000	00000000	COMCTL32	.text	code,imports	Imag	R		RWE	



Así que reiniciamos

Hacemos el mismo procedimiento anterior pero en vez de poner un bp memory on access en la sección code lo haremos en esta sección.

Empezaremos hacer el script para automatizar tareas.

```

var dir_excep
var Newoep

Datos:
mov Newoep, eip                                // Guardamos la dirección del EntryPoint.
                                                // Anotando la ultima dirección del Script
ask "Introduzca la ultima excepción"           // Sacamos la cajita de dialogo para introducir el dato.
cmp $RESULT,0                                  // Comparamos si se ha introducido alguna dirección.
je aviso                                       // Si no nos dirigimos a la etiqueta aviso
mov dir_excep, $RESULT                         // Si introducimos una dirección la guardamos en dir_excep.
jmp Inicio                                    // Saltamos a la etiqueta Inicio para comenzar.

aviso:
msg "Ejecute de nuevo el script e introduzca una dirección valida."
jmp final

Inicio:
run                                             // Ejecutamos el programa
eoe verifica                                  // Si se produce una excepción nos dirigimos a la etiqueta verifica.

```

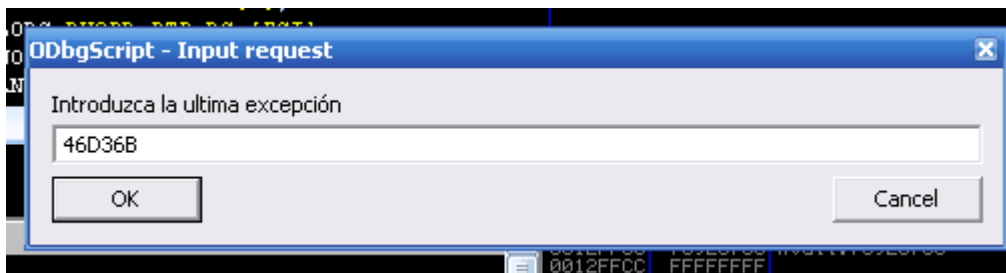
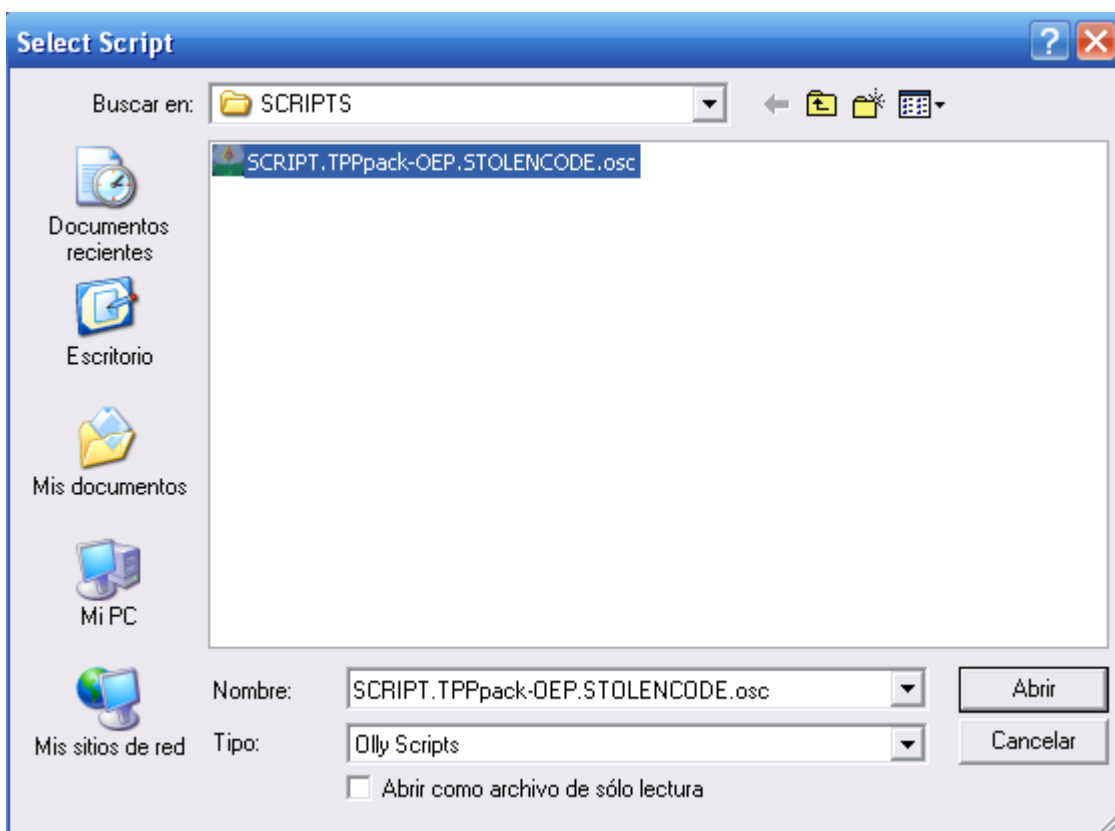
```

verifica:
cmp eip,dir_excep      // una vez que estemos aquí por una excepción comprobamos que sea la ultima.
je ultima             // Si lo es nos vamos a la etiqueta ultima.
esto                  // Si no ejecuta SHIFT + F9 para pasar la excepción.
jmp Inicio:           // Y saltamos a la etiqueta Inicio para buscar otra excepción.

ultima:
final:
ret

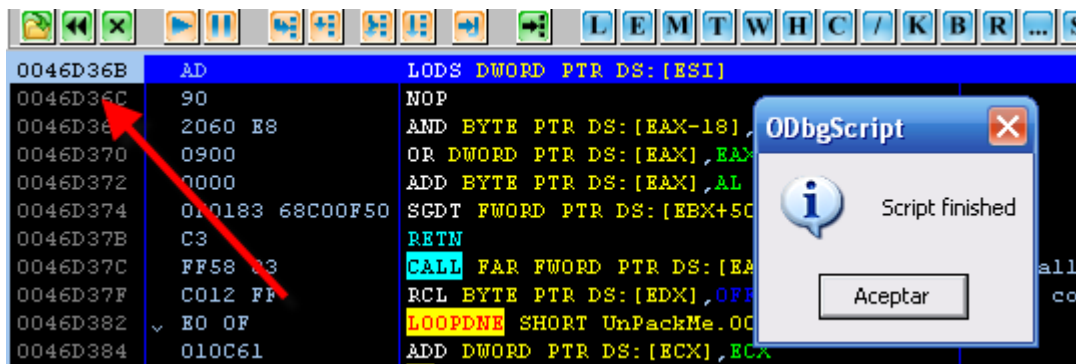
```

Ya con esto reiniciamos Olly y ejecutamos el script.

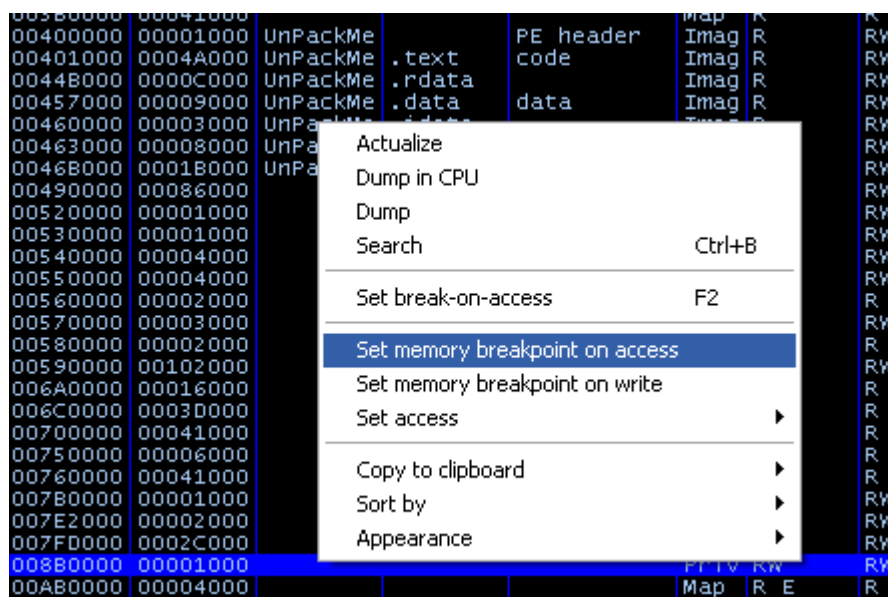


Con esto nos aparece un dialogo en el cual pondremos la ultima excepción que ya habíamos anotado en mi caso 46d36b.

Presionamos OK.



Y al momento termina cuando está en la ultima excepción.  
Presionamos ALT + M



Y en vez de poner un bp on access en la sección code lo haremos en la sección en la cual ya se había ejecutado código.  
Presionamos SHIFT + F9 y caemos en el Stolen Code.

008B0E48	55	PUSH EBP
008B0E49	8BEC	MOV EBP,ESP
008B0E4B	6A FF	PUSH -1
008B0E4D	68 600E4500	PUSH 450E60
008B0E52	68 C8924200	PUSH 4292C8
008B0E57	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
008B0E5D	50	PUSH EAX
008B0E5E	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
008B0E65	83C4 A8	ADD ESP,-58
008B0E68	53	PUSH EBX
008B0E69	56	PUSH ESI
008B0E6A	57	PUSH EDI
008B0E6B	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
008B0E6E	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
008B0E74	33D2	XOR EDX,EDX
008B0E76	8AD4	MOV DL,AH
008B0E78	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
008B0E7E	8BC8	MOV ECX,EAX
008B0E80	81E1 FF000000	AND ECX,0FF
008B0E86	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX
008B0E8C	C1E1 08	SHL ECX,8

Si presionamos la tecla (-) vemos que nos sitúa en la ultima excepción

0046D36E	AD	LDS DWORD PTR DS:[ESI]
0046D36C	90	NOP
0046D36D	2060 E8	AND BYTE PTR DS:[EAX-18],AH
0046D370	0900	OR DWORD PTR DS:[EAX],EAX
0046D372	0000	ADD BYTE PTR DS:[EAX],AL
0046D374	0F0183 68C00F50	SGDT FWORD PTR DS:[EBX+500FC068]
0046D37B	C3	RETN
0046D37C	FF58 83	CALL FAR FWORD PTR DS:[EAX-7D]
0046D37F	C012 FF	RCL BYTE PTR DS:[EDX],0FF
0046D382	✓ E0 0F	LOOPDNE SHORT UnPackMe.0046D393
0046D384	010C61	ADD DWORD PTR DS:[ECX],ECX
0046D387	✓ 74 04	JB SHORT UnPackMe.0046D38D
0046D389	✓ 75 02	JNZ SHORT UnPackMe.0046D38D
0046D38B	F66E EB	IMUL BYTE PTR DS:[ESI-15]
0046D38E	0BF2	OR ESI,EDX
0046D390	06	PUSH ES
0046D391	C6	???
0046D392	FD	STD
0046D393	^ E2 C3	LOOPD SHORT UnPackMe.0046D358
0046D395	E4 E5	IN AL,0E5
0046D397	A1 62B360E8	MOV EAX,DWORD PTR DS:[E860B362]
0046D39C	0900	OR DWORD PTR DS:[EAX],EAX
0046D39E	0000	ADD BYTE PTR DS:[EAX],AL
0046D3A0	0F0183 68C00F50	SGDT FWORD PTR DS:[EBX+500FC068]
0046D3A7	C3	RETN
0046D3A8	FF58 83	CALL FAR FWORD PTR DS:[EAX-7D]
0046D3AB	C012 FF	RCL BYTE PTR DS:[EDX],0FF
0046D3AE	✓ E0 0F	LOOPDNE SHORT UnPackMe.0046D3BF
0046D3B0	010C61	ADD DWORD PTR DS:[ECX],ECX
0046D3B3	E8 03000000	CALL UnPackMe.0046D3BB
0046D3B8	- E9 EBE483C4	JMP C4CAB8A8
0046D3BD	04 E8	ADD AL,0E8
0046D3BF	0000	ADD BYTE PTR DS:[EAX],AL
0046D3C1	0000	ADD BYTE PTR DS:[EAX],AL
0046D3C3	810424 08000000	ADD DWORD PTR SS:[ESP],8
0046D3CA	C3	RETN
0046D3CB	FFEO	JMP EAX
0046D3CD	8B5C24 04	MOV EBX,DWORD PTR SS:[ESP+4]

Y mas abajo el salto del que nos hablaba marciano que justamente nos manda al OEP que acabamos de encontrar entonces ahora modificaremos el script para que cuando este en la ultima excepción busque ese salto por medio de los opcodes **FFEO** nos ponga un breakpoint en el salto ejecute el programa y cuando estemos en el salto nos presione F7 para saltar al Stolen Code.

En la sección de variables del script le añadimos una más:

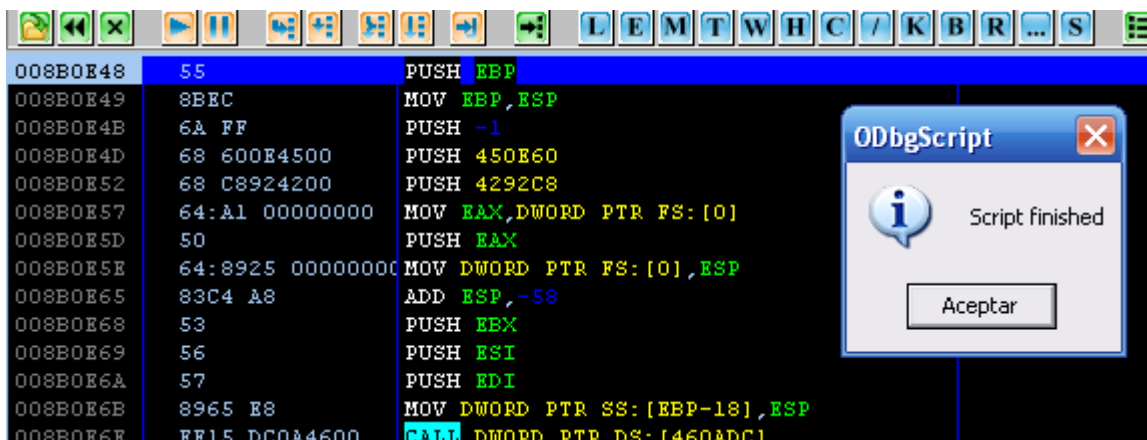
```
var dir_excep
var Newoep
var dir_JMP
Datos:
mov Newoep, eip
```

dir\_JMP ; que guardará la dirección del salto JMP EAX.

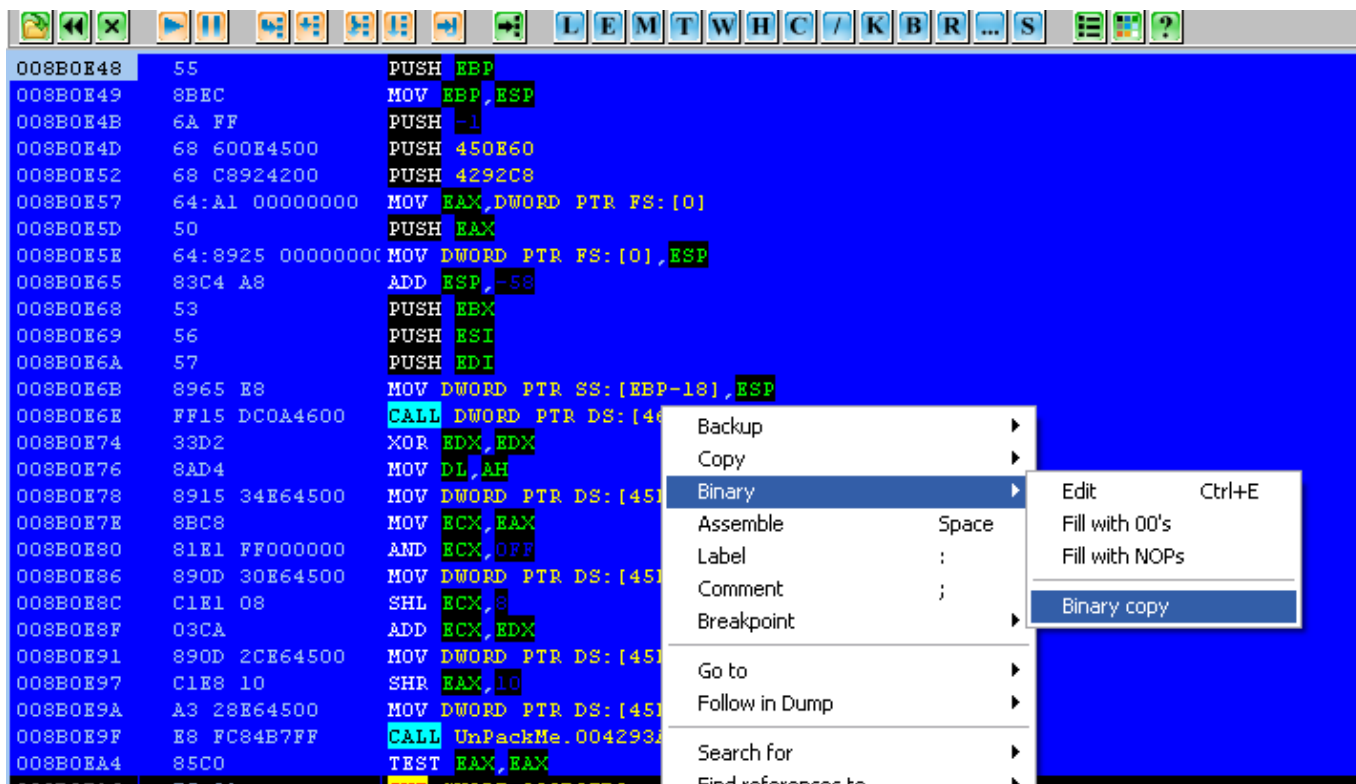
Y En la Etiqueta ultima le añadimos lo siguiente:

```
ultima:
findop eip,#FFE0#           // Al caer aquí por la ultima excepción buscamos el salto JMP EAX al Stolen Code.
mov dir_JMP,$RESULT         // Una vez encontrado aguardamos la dirección en dir_JMP
bp dir_JMP                  // Ponemos un breakpoint (F2) en el salto JMP EAX.
esto                        // Pasamos la excepción con SHIFT + F9 para caer en el bp.
bc dir_JMP                  // Quitamos el bp del Salto JMP EAX.
sti                         // Ejecuta F7 para caer en el Stolen Code.
```

Lo probamos.

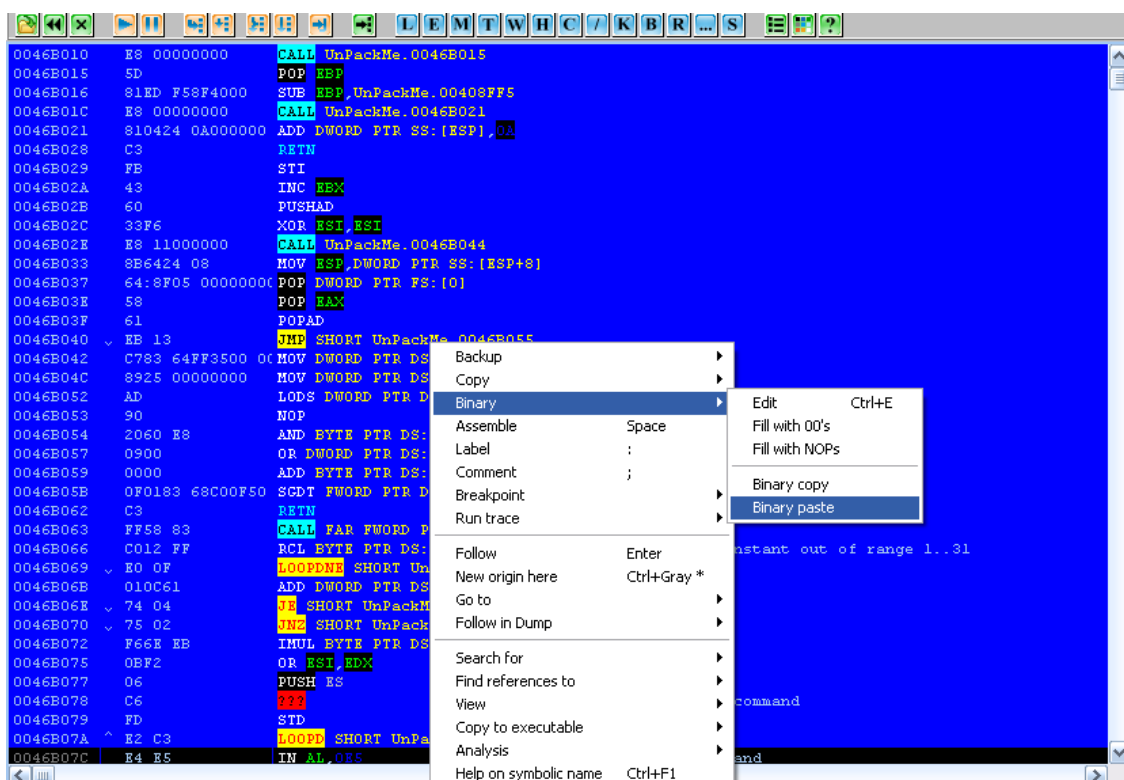


Y listo nos deja en el OEP ahora se complica el asunto como dijo marciano si copiamos este código al entrypoint:



Solo copiare un pedazo del Stolen Code hasta copiar por lo menos un call directo como en el que está en 008B0E9F.

Y lo pego en el ENTRY POINT:



Address	Hex	Assembly
0046B010	55	PUSH EBP
0046B011	8BEC	MOV EBP,ESP
0046B013	6A FF	PUSH -1
0046B015	68 600E4500	PUSH UnPackMe.00450E60
0046B01A	68 C8924200	PUSH UnPackMe.004292C8
0046B01F	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
0046B025	50	PUSH EAX
0046B026	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
0046B02D	83C4 A8	ADD ESP,-58
0046B030	53	PUSH EBX
0046B031	56	PUSH ESI
0046B032	57	PUSH EDI
0046B033	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
0046B036	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
0046B03C	33D2	XOR EDX,EDX
0046B03E	8AD4	MOV DL,AH
0046B040	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
0046B046	8BC8	MOV ECX,EAX
0046B048	81E1 FF000000	AND ECX,OFF
0046B04E	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX
0046B054	C1E1 08	SHL ECX,8
0046B057	03CA	ADD ECX,EDX
0046B059	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX
0046B05F	C1E8 10	SHR EAX,10
0046B062	A3 28E64500	MOV DWORD PTR DS:[45E628],EAX
0046B067	E8 FC84B7FF	CALL FFFE3568
0046B06C	85C0	TEST EAX,EAX
0046B06E	75 0A	JNZ SHORT UnPackMe.0046B07A
0046B070	75 02	JNZ SHORT UnPackMe.0046B074
0046B072	F66E EB	IMUL BYTE PTR DS:[ESI-15]
0046B075	0BF2	OR ESI,EDX
0046B077	06	PUSH ES
0046B078	C6	???

Ahí tenemos pegado un poco de código vemos que ese call indirecto que queda en la dirección 0046b067

Se estropeo todo veamos adonde tendría que apuntar, nos vamos a donde tenemos el OEP.

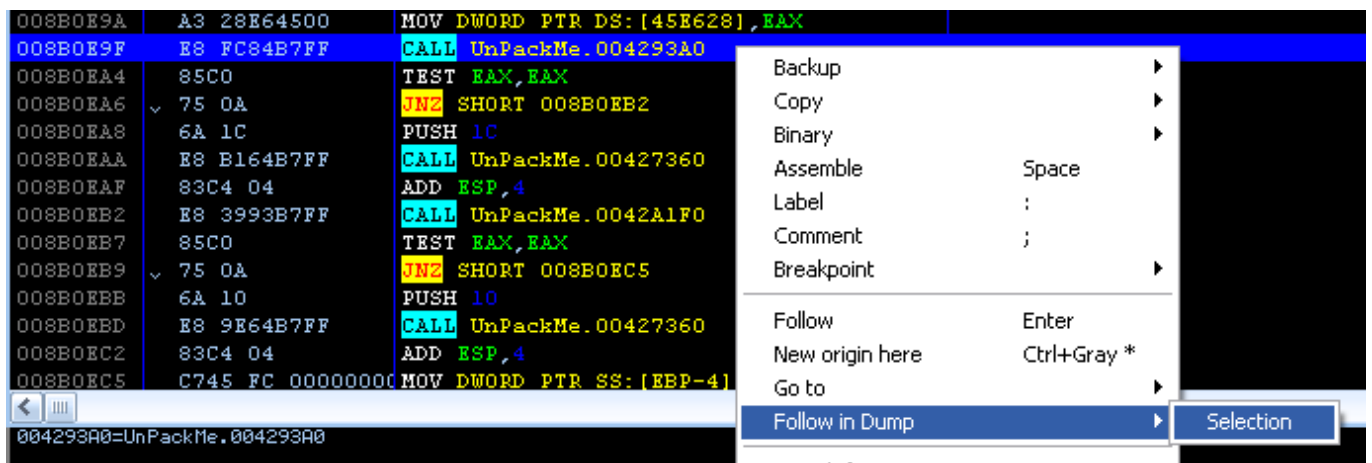
Address	Hex	Assembly
008B0E91	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX
008B0E97	C1E8 10	SHR EAX,10
008B0E9A	A3 28E64500	MOV DWORD PTR DS:[45E628],EAX
008B0E9F	E8 FC84B7FF	CALL UnPackMe.004293A0
008B0EA4	85C0	TEST EAX,EAX
008B0EA6	75 0A	JNZ SHORT 008B0EB2

Si vemos el original Stolen Code sería un **CALL 004293A0**

Entonces este call tendremos que arreglarlo para que cuando lo copiamos a otra parte apunte hacia esta misma zona y ¿Cómo hacemos esto? Pues bien .

Sigamos este call en el Dump:





Address	Hex dump
008B0E97	C1 E8 10 A3 28 E6
008B0E9F	E8 FC 84 B7 FF 85
008B0EA7	0A 6A 1C E8 B1 64
008B0EAF	83 C4 04 E8 39 93
008B0EB7	05 C8 75 0A C3 18

No tomamos en cuenta el E8 y usamos los cuatro bytes siguientes.

FF B7 84 FC → Que los llamaremos **OPCODES**

008B0E9A	A3 28E64500	MOV DWORD PTR DS:[45E628],EAX
008B0E9F	E8 FC84B7FF	CALL UnPackMe.004293A0
008B0EA4	85C0	TEST EAX,EAX

Tomamos la dirección del Call 008B0E9F → llamaremos **DIR\_CALL**

Ahora este Call apunta hacia **CALL 004293A0** y como obtenemos esa dirección

PUNTERO = **OPCODES + DIR\_CALL + 5**

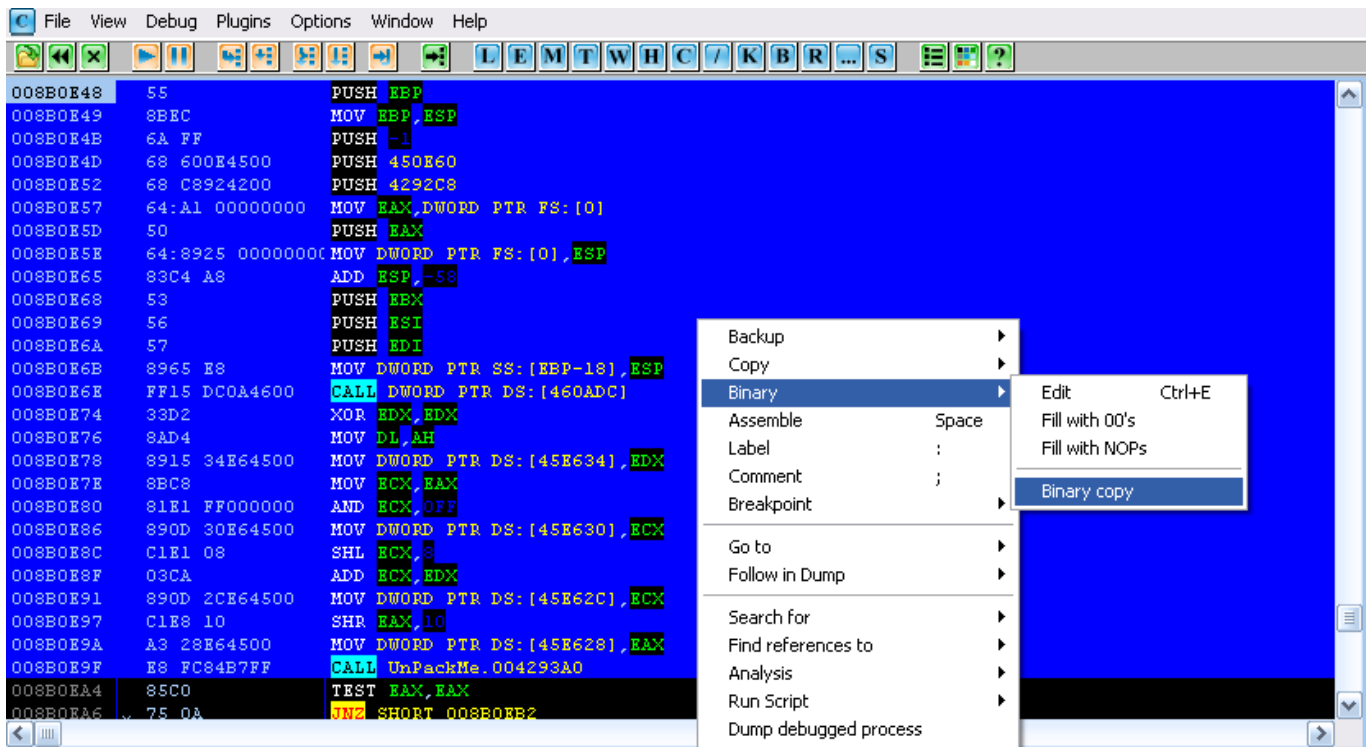
PUNTERO = FFB784FC + 008B0E9F + 5 = **004293A0**

Ahora bien ya tenemos la dirección hacia donde apunta el Call lo que nos falta es repararlo con los nuevos opcodes y estos nuevos opcodes los sacamos de la siguiente forma.

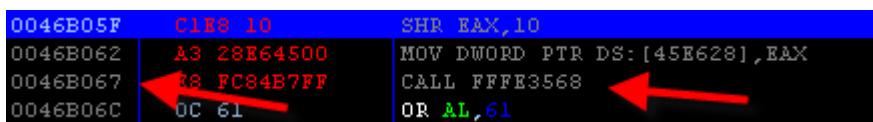
PUNTERO - DIRECCIÓN\_NUEVA - 5 = **NUEVOS OPCODES**

DIRECCIÓN\_NUEVA: la sacamos de esta forma.

Por ejemplo:



Si copiamos la primera linea del Stolen Code y lo pegamos en el ENTRY POINT.



Vemos que el primer call directo va a quedar en esta dirección en el entryptoint 0046B067  
Por lo tanto esta dirección la sacamos así .

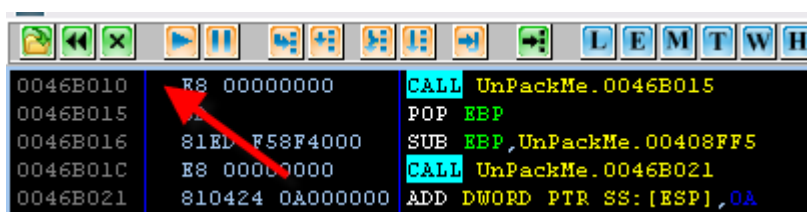
DIRECCIÓN ORIGINAL – DIRECCIÓN OEP(STOLEN CODE) + DIRECCIÓN ENTRY POINT  
Donde DIRECCIÓN ORIGINAL = 008B09EF



DIRECCIÓN OEP(STOLEN CODE) = 008B0E48



Y DIRECCION ENTRY POINT : 0046b010



8B0E9F - 8B0E48 + 46B010= 0046B067 , en esta dirección irá nuestro call reparado.

Con esto ya tenemos la DIRECCIÓN NUEVA = 0046B067

Ahora ya tenemos los datos para sacar los nuevos opcodes.

PUNTERO - DIRECCIÓN\_NUEVA - 5 = NUEVOS OPCODES

004293A0 - 0046B067 - 5 = FFFBE334

Ya tenemos los nuevos opcodes

Ahora solo que editarlos a mano para que vean.

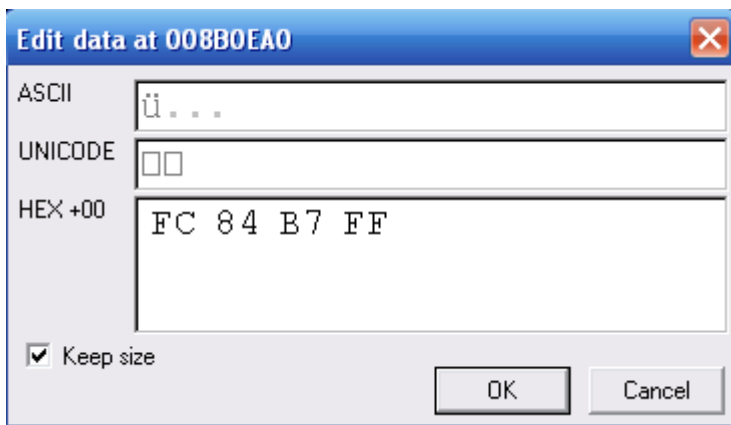
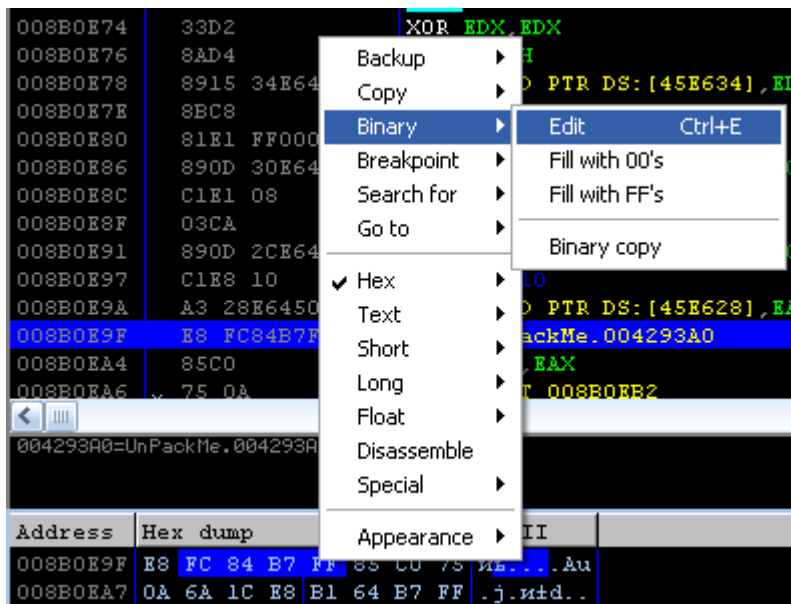
Nos vamos al primer call directo de los stolen code.

The screenshot shows the OllyDbg interface with the assembly window displaying code from address 008B0E48 to 008B0E9F. The instruction at 008B0E9F is highlighted: `E8 FC84B7FF CALL UnPackMe.004293A0`. A right-click context menu is open over this instruction, showing options like Backup, Copy, Binary, Assemble, Label, Comment, Breakpoint, Follow, New origin here, Go to, Follow in Dump, and Search for. The 'Follow in Dump' option is selected, and a 'Selection' button is visible at the bottom right of the menu.

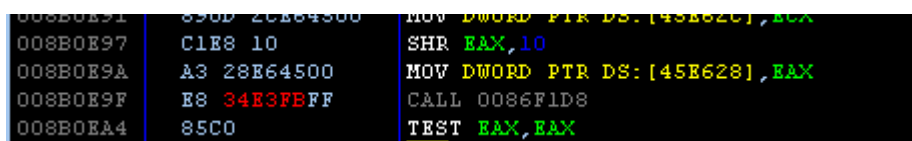
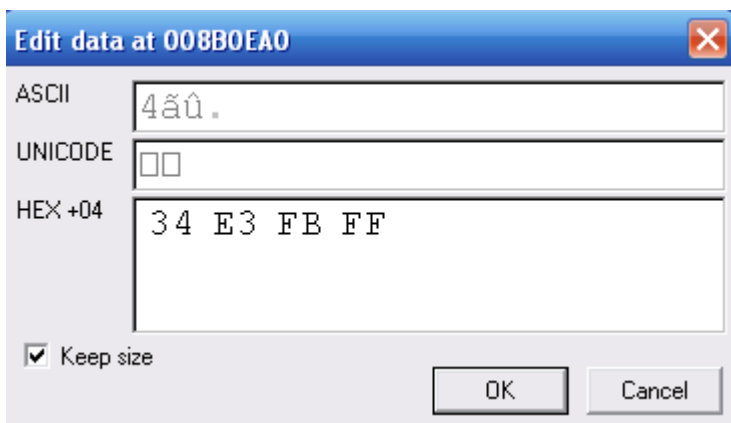
En el dump nos muestra esto:

Address	Hex dump	ASCII
008B0E9F	E8 FC 84 B7 FF 85 C0 75	иъ...Au
008B0EA7	0A 6A 1C E8 B1 64 B7 FF	.j.m+d..
008B0EAF	83 C4 04 E8 39 93 B7 FF	.Д.и9...

NO tomamos el E8 y modificamos los 4 bytes siguientes:



Y los reemplazamos por los nuevos opcodes.



Vemos que nuestro call quedó horrible pero si copiamos este pedazo de código y lo pegamos al ENTRY POINT.

```

008B0E48 55          PUSH EBP
008B0E49 8BEC       MOV EBP,ESP
008B0E4B 6A FF     PUSH -1
008B0E4D 68 600E4500 PUSH 450E60
008B0E52 68 C8924200 PUSH 4292C8
008B0E57 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
008B0E5D 50        PUSH EAX
008B0E5F 64:8925 00000000 MOV DWORD PTR FS:[0],ESP
008B0E65 83C4 A8    ADD ESP,-58
008B0E68 53        PUSH EBX
008B0E69 56        PUSH ESI
008B0E6A 57        PUSH EDI
008B0E6B 8965 E8    MOV DWORD PTR SS:[EBP-18],ESP
008B0E6E FF15 DC0A4600 CALL DWORD PTR DS:[460ADC]
008B0E74 33D2      XOR EDX,EDX
008B0E76 8AD4      MOV DL,AH
008B0E78 8915 34E64500 MOV DWORD PTR DS:[45E634],EDX
008B0E7E 8BC8      MOV ECX,EAX
008B0E80 81E1 FF000000 AND ECX,0FF
008B0E86 890D 30E64500 MOV DWORD PTR DS:[45E630],ECX
008B0E8C C1E1 08    SHL ECX,8
008B0E8F 03CA      ADD ECX,EDX
008B0E91 890D 2CE64500 MOV DWORD PTR DS:[45E62C],ECX
008B0E97 C1E8 10    SHR EAX,10
008B0E9A A3 28E64500 MOV DWORD PTR DS:[45E628],EAX
008B0E9F E8 34E3FBFF CALL UnPackMe.004293A0
008B0EA4 85C0      TEST EAX,EAX
  
```

```

0046B010 55          PUSH EBP
0046B011 8BEC       MOV EBP,ESP
0046B013 6A FF     PUSH -1
0046B015 68 600E4500 PUSH UnPackMe.00450E60
0046B01A 68 C8924200 PUSH UnPackMe.004292C8
0046B01F 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
0046B025 50        PUSH EAX
0046B026 64:8925 00000000 MOV DWORD PTR FS:[0],ESP
0046B02D 83C4 A8    ADD ESP,-58
0046B030 53        PUSH EBX
0046B031 56        PUSH ESI
0046B032 57        PUSH EDI
0046B033 8965 E8    MOV DWORD PTR SS:[EBP-18],ESP
0046B036 FF15 DC0A4600 CALL DWORD PTR DS:[460ADC]
0046B03C 33D2      XOR EDX,EDX
0046B03E 8AD4      MOV DL,AH
0046B040 8915 34E64500 MOV DWORD PTR DS:[45E634],EDX
0046B046 8BC8      MOV ECX,EAX
0046B048 81E1 FF000000 AND ECX,0FF
0046B04E 890D 30E64500 MOV DWORD PTR DS:[45E630],ECX
0046B054 C1E1 08    SHL ECX,8
0046B057 03CA      ADD ECX,EDX
0046B059 890D 2CE64500 MOV DWORD PTR DS:[45E62C],ECX
0046B05F C1E8 10    SHR EAX,10
0046B062 A3 28E64500 MOV DWORD PTR DS:[45E628],EAX
0046B067 E8 34E3FBFF CALL UnPackMe.004293A0
0046B06C 0C 61     OR AL,61
0046B06E 74 04     JB SHORT UnPackMe.0046B074
0046B070 75 02     JNZ SHORT UnPackMe.0046B074
  
```

Quedó arregladito justo como tenia que estar.

Ahora tenemos que hacer todo esto para los otros call directos pero esto se lo dejaremos al script Que le añadiremos esto.

```

var dir_CALL
var oep
var StartScan
var Opcodes
var temp
var temp2
var temp3

```

Y esto después

```

ultima:
findop eip,#FFE0#           // Al caer aquí por la ultima excepción buscamos el salto JMP EAX al Stolen Code.
mov dir_JMP,$RESULT         // Una vez encontrado aguardamos la dirección en dir_JMP
bp dir_JMP                  // Ponemos un breakpoint (F2) en el salto JMP EAX.
esto                        // Pasamos la excepción con SHIFT + F9 para caer en el bp.
bc dir_JMP                  // Quitamos el bp del Salto JMP EAX.
sti                         // Ejecuta F7 para caer en el Stolen Code.

mov oep,eip                 // Guardamos la dirección del OEP robado.
mov StartScan,eip          // Guardamos la dirección del oep tambien a StartScan que la usaremos para buscar Calls.

BuscarCall:                // Aquí empezaremos a buscar los calls directos para repararlos para que a la hora de
                           // de hacer binary copy y pegarlos al nuevo oep esos call directos queden bien.
                           // Nota ver MiniTuto.PARTE1.doc

findop StartScan,#E8#       // Buscamos los calls directos que comienzan con su opcode E8
cmp $RESULT, 0              // Cuando no encuentre mas $RESULT valdrá 0 y saltamos al final del script.
je final
mov dir_CALL, $RESULT       // Guardamos la dirección del primer call encontrado.
mov StartScan, $RESULT      // Actualizamos desde donde queremos seguir buscando calls en este caso desde el primer c
add dir_CALL,1              // A la dirección del call encontrado le sumamos 1 esto para no tomar el OPCODE E8.
mov Opcodes, [dir_CALL]     // Movemos los opcodes de esa dirección a Opcodes.
add Opcodes,StartScan       // A esos Opcodes le Sumados la Dirección del Call.
add Opcodes,5               // Al resultado le sumamos 5 y obtenemos la dirección que apunta el call.
                           // Nota ver MiniTuto.PARTE1.doc para aclarar.

```

```

// Ahora le asignaremos un nuevo opcode a este call encontrado para que a la hora de hacer bynary paste en el nuevo
// oep ese call quede arreglado.

```

```

// Ahora le asignaremos un nuevo opcode a este call encontrado para que a la hora de hacer bynary paste en el nuevo
// oep ese call quede arreglado.

```

```

mov temp, StartScan        // Movemos la dirección que contiene StartScan a un temporal. ( StartScan = Dirección del
sub temp, oep               // A la dirección de ese call a reparar le restamos la dirección del oep del stolen code.
mov temp2, Newoep           // Movemos la direccion del nuevo oep que es el EntryPoint a un temporal2.
add temp,temp2              // A ese temporal2 le sumamos el temporal1.
sub Opcodes, temp           //Ahora la dirección a la que apunta el CALL le restamos la operación anterior
sub Opcodes, 5              // y a lo que quedó le sumamos 5 y obtenemos los opcodes nuevos para ese call a reparar.

```

```

Editar:                     // Empezaremos a Editar el call con esos nuevos opcodes.
mov temp3, StartScan        // Movemos la dirección del call a reparar contenida en StartScan a un tercer temporal.
add temp3,1                 // A la direccion del call le sumamos 1 para no tamar en cuanta el opcode E8.
mov [temp3], Opcodes        // Reparamos el call con los nuevos Opcodes.
jmp BuscarCall

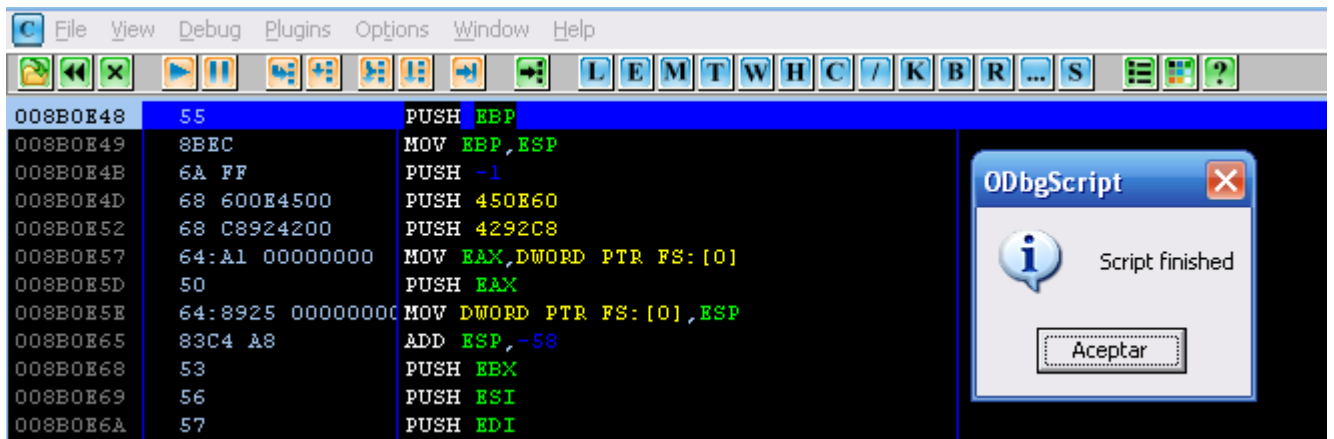
```

```

final:
ret

```

Ejecutamos el script.



The screenshot shows a debugger window with a menu bar (File, View, Debug, Plugins, Options, Window, Help) and a toolbar. The assembly list on the left shows instructions from address 008B0E48 to 008B0E6A. The instruction at 008B0E57 is highlighted. On the right, a dialog box titled 'ODbgScript' with a close button (X) displays an information icon (i) and the text 'Script finished'. Below the text is a button labeled 'Aceptar'.

Address	Hex	Assembly
008B0E48	55	PUSH EBP
008B0E49	8BEC	MOV EBP,ESP
008B0E4B	6A FF	PUSH -1
008B0E4D	68 600E4500	PUSH 450E60
008B0E52	68 C8924200	PUSH 4292C8
008B0E57	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
008B0E5D	50	PUSH EAX
008B0E5E	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
008B0E65	83C4 A8	ADD ESP,-58
008B0E68	53	PUSH EBX
008B0E69	56	PUSH ESI
008B0E6A	57	PUSH EDI

008B0E9A	A3 28E64500	MOV DWORD PTR DS:[45
008B0E9F	E8 34E3FBFF	CALL 0086F1D8
008B0EA4	85C0	TEST EAX,EAX

008B0EAA	E8 E9C2FBFF	CALL 0086D198
----------	-------------	---------------

008B0EBB	6A 10	PUSH 10
008B0EBD	E8 D6C2FBFF	CALL 0086D198
008B0EC2	83C4 04	ADD ESP,4
008B0EC5	C745 FC 00000000	MOV DWORD PTR SS:[EBP-4],0
008B0ECC	E8 27EDFBFF	CALL 0086FBF8
008B0ED1	E8 B2D2FBFF	CALL 0086E188
008B0ED6	FF15 84094600	CALL DWORD PTR DS:[460984]
008B0EDC	A3 D8EB4500	MOV DWORD PTR DS:[45EBD8],EAX
008B0EE1	E8 D255FCFF	CALL 008764B8

Vemos que nos ha arreglado todas las call que lo único que tenemos que hacer es

Seleccionar todo el Stolen Code: le damos Binary Copy

Y los pegamos en el Entry Point.

0046B010	55	PUSH EBP
0046B011	8BEC	MOV EBP,ESP
0046B013	6A FF	PUSH -1
0046B015	68 600E4500	PUSH UnPackMe.00450E60
0046B01A	68 C8924200	PUSH UnPackMe.004292C8
0046B01F	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
0046B025	50	PUSH EAX
0046B026	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
0046B02D	83C4 A8	ADD ESP,-58
0046B030	53	PUSH EBX
0046B031	56	PUSH ESI
0046B032	57	PUSH EDI
0046B033	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
0046B036	FF15 DCOA4600	CALL DWORD PTR DS:[460ADC]
0046B03C	33D2	XOR EDX,EDX
0046B03E	8AD4	MOV DL,AH
0046B040	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
0046B046	8BC8	MOV ECX,EAX
0046B048	81E1 FF000000	AND ECX,OFF
0046B04E	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX
0046B054	C1E1 08	SHL ECX,8
0046B057	03CA	ADD ECX,EDX
0046B059	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX
0046B05F	C1E8 10	SHR EAX,10
0046B062	A3 28E64500	MOV DWORD PTR DS:[45E628],EAX
0046B067	E8 34E3FBFF	CALL UnPackMe.004293A0
0046B06C	85C0	TEST EAX,EAX
0046B06E	75 0A	JNZ SHORT UnPackMe.0046B07A
0046B070	6A 1C	PUSH 1C
0046B072	E8 E9C2FBFF	CALL UnPackMe.00427360
0046B077	83C4 04	ADD ESP,4
0046B07A	E8 71F1FBFF	CALL UnPackMe.0042A1F0
0046B07F	85C0	TEST EAX,EAX
0046B081	75 0A	JNZ SHORT UnPackMe.0046B08D

Situados en el Entry Point damos cambios el eip.

0046B010	55	PUSH EBP
0046B011	8BEC	MOV EBP,ESP
0046B013	6A FF	PUSH -1
0046B015	68 600E4500	PUSH UnPackMe.00450E60
0046B01A	68 C8924200	PUSH UnPackMe.004292C8
0046B01F	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
0046B025	50	PUSH EAX
0046B026	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
0046B02D	83C4 A8	ADD ESP,-58
0046B030	53	PUSH EBX
0046B031	56	PUSH ESI
0046B032	57	PUSH EDI
0046B033	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
0046B036	FF15 DCOA4600	CALL DWORD PTR DS:[460ADC]
0046B03C	33D2	XOR EDX,EDX

Backup  
Copy  
Binary  
Undo selection Alt+BkSp  
Assemble Space  
Label :  
Comment ;  
Breakpoint  
Run trace  
New origin here Ctrl+Gray \*  
Go to

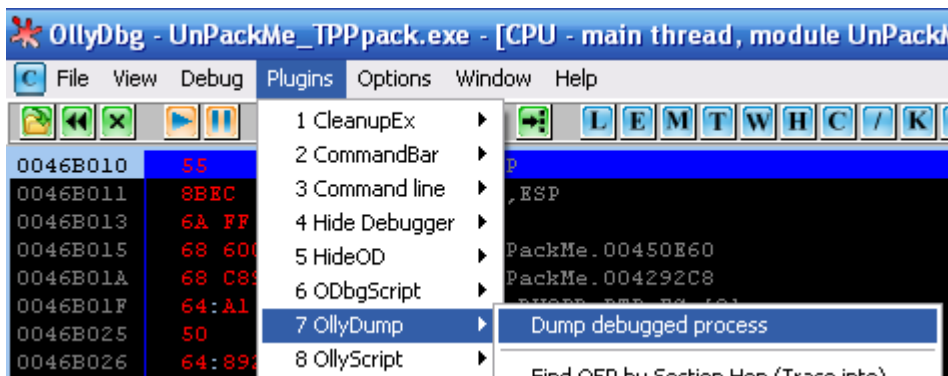
Please confirm suspicious EIP

The byte you are pointing at lies outside the executable code of any known module. Invalid EIP may have disastrous effects on the debugged program. Do you still want to change origin?

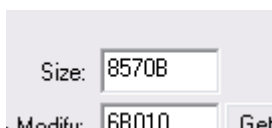
Sí No

Presionamos que SI y dumpeamos.

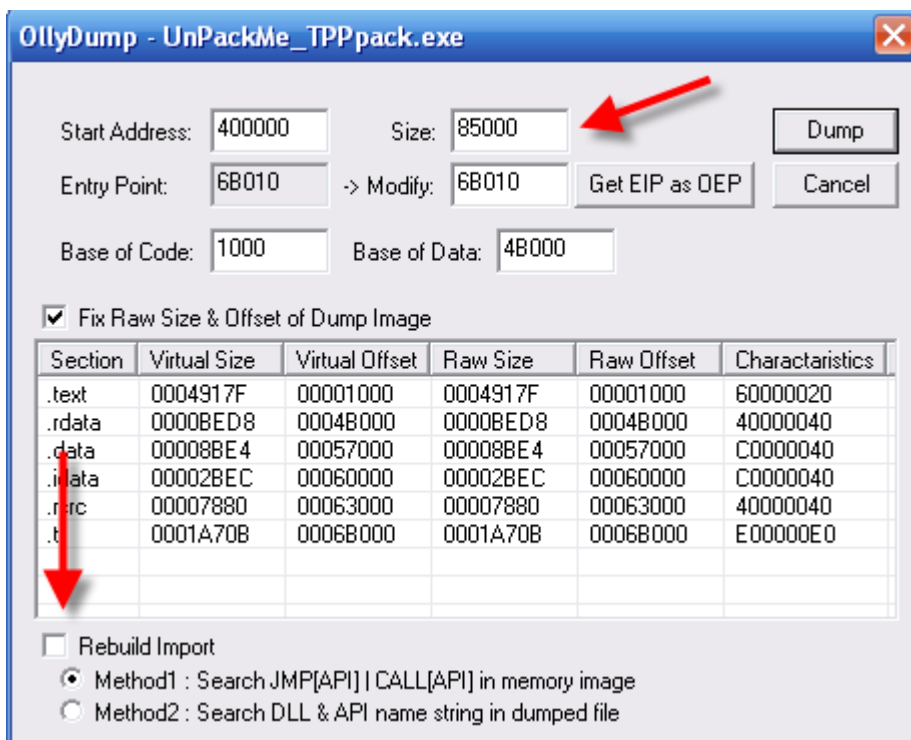




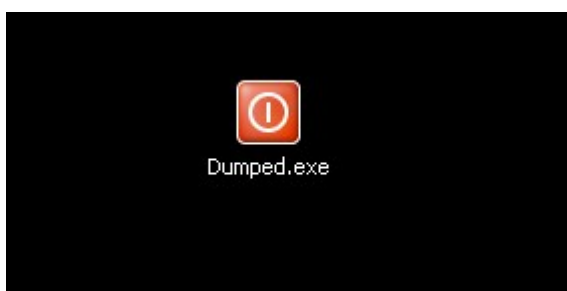
Acordemonos de lo que dijo marciano que tenemos que cambiar



Por

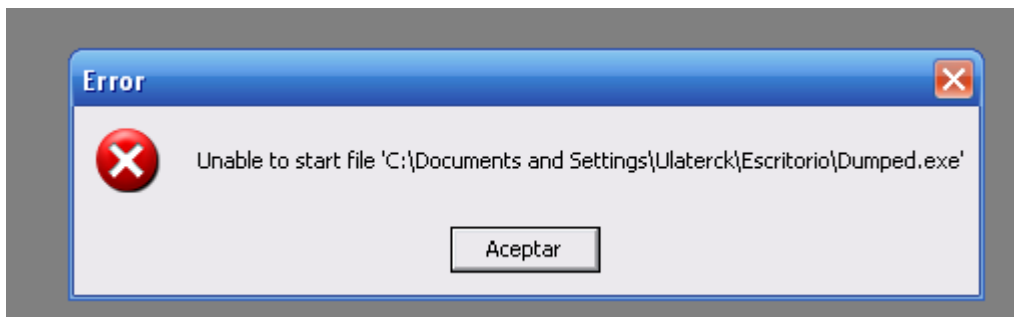


Destildamos la casilla Rebuild Import y presionamos Dump

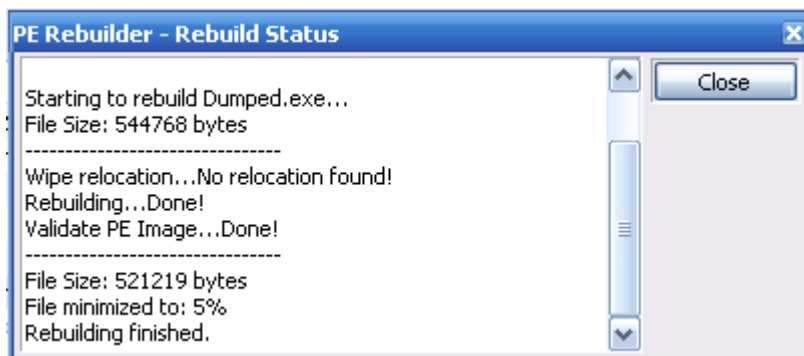
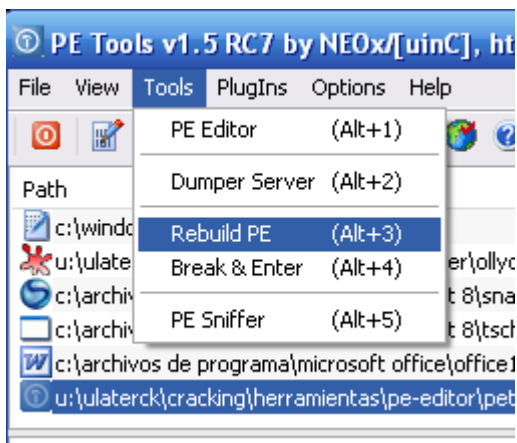


Ahí tenemos nuestro Dumpeado que le Falta la segunda PARTE Reparar la IAT.

Si lo intentamos abri con Olly



No podemos esto lo arreglamos con un PE. Editor Reconstruirlo.



Y listo

Address	Disassembly	Comment
0046B010	PUSH EBP	
0046B011	MOV EBP, ESP	
0046B013	PUSH -1	
0046B015	PUSH Dumped.00450E60	
0046B01A	PUSH Dumped.004292C8	SE han
0046B01F	MOV EAX, DWORD PTR FS:[0]	
0046B025	PUSH EAX	
0046B026	MOV DWORD PTR FS:[0], ESP	
0046B02D	ADD ESP, -58	
0046B030	PUSH EBX	
0046B031	PUSH ESI	
0046B032	PUSH EDI	
0046B033	MOV DWORD PTR SS:[EBP-18], ESP	
0046B036	CALL DWORD PTR DS:[460ADC]	
0046B03C	XOR EDX, EDX	
0046B03E	MOV DL, AH	
0046B040	MOV DWORD PTR DS:[45E634], EDX	
0046B046	MOV ECX, EAX	

Como aclaracion mia les digo que con el uso hemos descubierto muchos errores en el plugin HideOd que hace que ciertos programas no corran (mi idea es que a marciano le ocurrio esto, por ahi me equivoco pero a mi tambien el programa me corre normalmente como a Ularteck), asi que lo mejor es quitarlo y reemplazarlo por el HideDebugger y el OllyAdvanced.  
Bueno con esto queda terminada la explicacion del script de la parte 1 ahora vayamos al script de la parte 2.

---

\*/

```
var base
var dir_VirtualAlloc
var dir_VirtualProtect
VAR dir_mov
Inicio:
```

```
gpa "VirtualAlloc", "kernel32.dll" // Buscamos la direcci3n de la Funcion VirtualAlloc
mov dir_VirtualAlloc, $RESULT
log dir_VirtualAlloc
```

```
gpa "VirtualProtect", "kernel32.dll"
mov dir_VirtualProtect, $RESULT
```

```
bp dir_VirtualAlloc
run
eob info
```

info:

```
mov base, eax
log base
bc dir_VirtualAlloc
bp dir_VirtualProtect
```

```
Zona:
eob seccion
```

run

seccion:

cmp esi, 00460000

je retornar

jmp Zona

retornar:

bc dir\_VirtualProtect

mov Reg\_esp, [esp]

bp Reg\_esp

eob zona\_1

run

zona\_1:

bc Reg\_esp

find base, #897C24188B4424#

mov dir\_mov, \$RESULT

log dir\_mov

jmp nopear

nopear:

bp dir\_mov

eob nopear2

run

nopear2:

bc dir\_mov

fill dir\_mov, 4, 90

final:

msg "Mov nopeado, al terminar el script presiona run (F9)."

ret

El metodo para llegar arreglar la IAT es el clasico, que esta muy bien explicado en el tute de marciano, poner un BPM ON WRITE en una de las entradas malas de la IAT que localizamos previamente cuando estuvimos en el OEP, y luego tratar de llegar al punto donde guarda el valor malo, lamentablemente el programa mediante la api VirtualProtect borra el BPM ON WRITE asi que debemos poner un BP en dicha api para restaurarlo una vez que retorne de la misma.

00127A60	10007965	CALL to VirtualProtect from 1000795F
00127A64	00460000	Address = UnPackMe.00460000
00127A68	00002BEC	Size = 2BEC (11244.)
00127A6C	00000040	NewProtect = PAGE_EXECUTE_READWRITE
00127A70	00127BE0	pOldProtect = 00127BE0
00127A74	00000002	
00127A78	00460613	ASCII "erm"

Alli vemos la imagen extractada del tute de marciano cuando para en VirtualProtect para cambiar el

permiso de la zona de la IAT, así que en ese momento se debe llegar al RET y volver a colocar el BPM ON WRITE y seguir con run hasta que paramos en donde guarda el valor malo en la entrada,

```

10005CE9 894C37 02 MOV DWORD PTR DS:[EDI+ESI+2],ECX
10005CED 66:C74437 06 C MOV WORD PTR DS:[EDI+ESI+6],5C7
10005CF4 894C37 08 MOV DWORD PTR DS:[EDI+ESI+8],ECX
10005CF8 894437 0C MOV DWORD PTR DS:[EDI+ESI+C],EAX
10005CFC C64437 10 C3 MOV BYTE PTR DS:[EDI+ESI+10],0C3
10005D01 897C24 18 MOV DWORD PTR SS:[ESP+18],EDI
10005D05 8B4424 18 MOV EAX,DWORD PTR SS:[ESP+18]
10005D09 8945 00 MOV DWORD PTR SS:[EBP],EAX
10005D0C 8B4424 24 MOV EAX,DWORD PTR SS:[ESP+24]
10005D10 8B78 04 MOV EDI,DWORD PTR DS:[EAX+4]
10005D13 83C0 04 ADD EAX,4
10005D16 83C5 04 ADD EBP,4
10005D19 85FF TEST EDI,EDI
10005D1B 894424 24 MOV DWORD PTR SS:[ESP+24],EAX
10005D1F 896C24 48 MOV DWORD PTR SS:[ESP+48],EBP
10005D23 ^ 0F85 CAFDFFFF JNZ 10005AF3
10005D29 834424 2C 14 ADD DWORD PTR SS:[ESP+2C],14
10005D2E 8B6C24 2C MOV EBP,DWORD PTR SS:[ESP+2C]
10005D32 8B6C24 98030000 MOV EDI,DWORD PTR SS:[ESP+398]
10005D39 ^ E9 E5FCFFFF JMP 10005A23
10005D3E 5E POP ESI
10005D3F 5B POP EBX
10005D40 5F POP EDI
10005D41 33C0 XOR EAX,EAX
10005D43 5D POP EBP
10005D44 81C4 84030000 ADD ESP,384
10005D4A C2 0400 RETN 4

```

Allí para cuando lo guarda, en EAX está el valor malo, y EBP apunta a la entrada de la IAT que estábamos vigilando.

Luego coloca un BP unas líneas más arriba y cuando para, traceando ve que aquí

```

10005CF4 894C37 08 MOV DWORD PTR DS:[EDI+ESI+8],ECX
10005CF8 894437 0C MOV DWORD PTR DS:[EDI+ESI+C],EAX
10005CFC C64437 10 C3 MOV BYTE PTR DS:[EDI+ESI+10],0C3
10005D01 897C24 18 MOV DWORD PTR SS:[ESP+18],EDI ;Machaca valor bueno
10005D05 8B4424 18 MOV EAX,DWORD PTR SS:[ESP+18]
10005D09 8945 00 MOV DWORD PTR SS:[EBP],EAX ;Escribe en la IAT
10005D0C 8B4424 24 MOV EAX,DWORD PTR SS:[ESP+24]
10005D10 8B78 04 MOV EDI,DWORD PTR DS:[EAX+4]
10005D13 83C0 04 ADD EAX,4

```

Se reemplaza la dirección de la API correcta, por el valor malo que guarda unas líneas después así que el trabajo se resume en nopear esta línea ya que así guardara la dirección buena un poco más adelante.

Así que el script debe localizar esa instrucción y nopearla.

De esta forma el script lo primero que hace es buscar la dirección de la API VirtualAlloc ya que la primera vez que para en ella, obtiene la dirección de la zona donde estará la instrucción a nopear, ya que esta varía en cada máquina, hay que hallar este valor, para poder desde allí buscarla.

```

var base
var dir_VirtualAlloc
var dir_VirtualProtect
VAR dir_mov
Inicio:

```

```

gpa "VirtualAlloc", "kernel32.dll" // Buscamos la dirección de la Función VirtualAlloc
mov dir_VirtualAlloc, $RESULT
log dir_VirtualAlloc

```

Alli guarda en la variable **var dir\_VirtualAlloc**, la direccion de dicha api luego hace los mismo con la de la api VirtualProtect.

```
gpa "VirtualProtect", "kernel32.dll"  
mov dir_VirtualProtect, $RESULT
```

En la variable **var dir\_VirtualProtect** guarda la direccion de la api correspondiente.

```
bp dir_VirtualAlloc  
run  
eob info
```

**info:**

```
mov base, eax  
log base  
bc dir_VirtualAlloc  
bp dir_VirtualProtect
```

Luego vemos que coloca un BP en VirtualAlloc y da RUN y mediante eob, para en dicho bp alli y guarda la direccion donde el programa va a crear la zona donde estara la direccion a nopear, en la variable base.

Tambien coloca el bp en VirtualProtect

```
Zona:  
eob seccion  
run
```

```
seccion:  
cmp esi, 00460000  
je retornar  
jmp Zona
```

Alli cuando para por el Bp compara si ESI vale 460000 que es el inicio de la IAT ya que la api VirtualProtect lo usa como parametro para proteger dicha zona y borrar los BPMs que haya en ella y si es igual va a la etiqueta retornar.

```
retornar:  
bc dir_VirtualProtect  
mov Reg_esp, [esp]  
bp Reg_esp
```

Donde borra el BP en la api VirtualProtect y halla el valor donde retorna la misma que esta en [esp] y pone alli un BP para poder parar cuando apenas retorna de la api.

```
zona_1:  
bc Reg_esp  
find base, #897C24188B4424#  
mov dir_mov, $RESULT  
log dir_mov
```

## jmp nopear

Cuando retorna de la api, ya puede buscar la instrucción a nopear lo realiza buscando la cadena de bytes suficientemente larga para no encontrar instrucciones parecidas que no nos interesen nopear, **897C24188B4424**

10005CF4	894C37 08	MOV DWORD PTR DS:[EDI+ESI+8],ECX	
10005CF8	894437 0C	MOV DWORD PTR DS:[EDI+ESI+C],EAX	
10005CFC	8C4437 10 C3	MOV BYTE PTR DS:[EDI+ESI+10],0C3	
10005D01	897C24 18	MOV DWORD PTR SS:[ESP+18],EDI	;Machaca valor bueno
10005D05	8B4424 18	MOV EAX,DWORD PTR SS:[ESP+18]	
10005D09	8345 00	MOV DWORD PTR SS:[EBP],EAX	;Escribe en la IAT
10005D0D	8B4424 24	MOV EAX,DWORD PTR SS:[ESP+24]	
10005D11	8B78 04	MOV EDI,DWORD PTR DS:[EAX+4]	
10005D13	83C0 04	ADD EAX,4	

asi que luego de hallarla salta a nopearla

**nopear:**

**bp dir\_mov**

**eob nopear2**

**run**

**nopear2:**

**bc dir\_mov**

**fill dir\_mov, 4, 90**

**final:**

**msg "Mov nopeado, al terminar el script presiona run (F9)."**

**ret**

Bueno esa es la explicacion y felicitaciones a Ularteck por tan buen trabajo y gracias a marciano por su buen tute del cual extrajimos las imagenes para explicar la parte 2 del script.

Creo que con esto ya habra pocos que no dominen la tecnica de realizar scripts para ayudarse, creo que es algo muy poderoso y que no es para nada dificil, si uno se pone con ello e intenta.

En la parte 54 comenzaremos un nuevo tema nos veemos.

Hasta la 54

Ricardo Narvaja

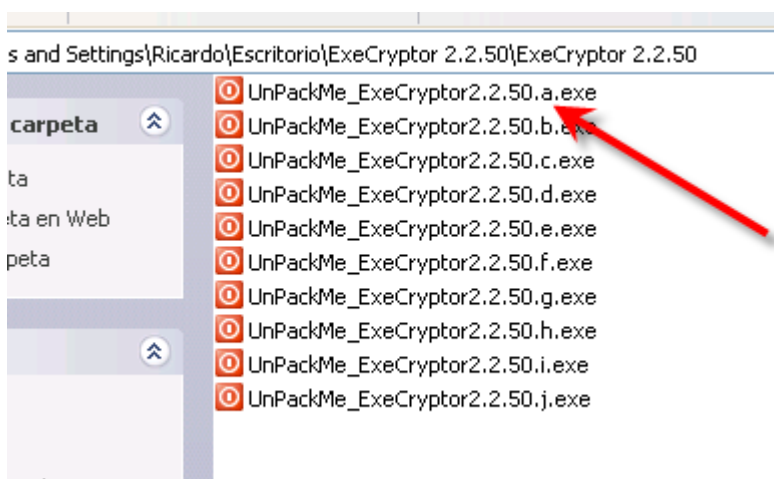
02/09/06

## INTRODUCCION AL CRACKING CON OLLYDBG PARTE 54

Cuando uno analiza un packer que no conoce, lo mejor es primero hallar si hay unpackmes conocidos o sea que tengamos el codigo desempaado del mismo, y si como en este caso hay unpackmes, tenemos el codigo del unpackme desempaado y encima tenemos varios unpackmes en los cuales se empieza por la proteccion mas simple del packer, y asi se va descubriendo cada truco lentamente hasta llegar al mas complicado y alli ya cuando conocemos los trucos del packer podemos saltar a un programa, de esta forma el conocimiento es gradual, y ademas si mas adelante salen nuevas versiones, al tener el funcionamiento entendido completo de la anterior, las modificaciones generalmente son minimas.

En este caso veremos una serie de unpackmes de execryptor que a pesar de que no es la ultima version es casi la ultima, y nos permitira ir conociendo al packer gradualmente, trataremos de desentrañar lo mas que podamos sabemos que es un packer maldito, pero haremos el esfuerzo por entenderlo, lo mas que podamos.

Vemos que tenemos varios niveles de unpackmes, en los cuales se va agregando dificultad y trucos.



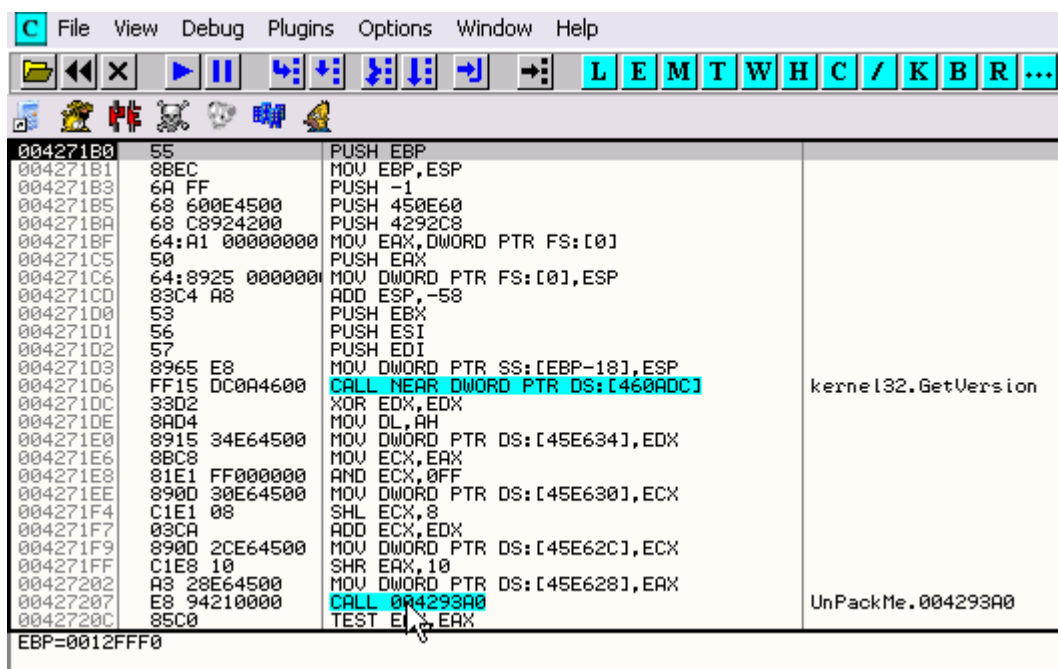
Pues el mas facil vemos que es el a, al correrlo nos dice cual es la proteccion que acusa



O sea que no tiene casi nada (jejeje), pero es un buen inicio, ademas tengo un desempaado de un unpackme upx, que adjunto al tutorial que esta realizado con el mismo codigo base, podemos usar ese crackme desempaado para comparar y sacar conclusiones, ya que vemos en el OEP la IAT, en este caso como es el mas sencillo, no es tan importante, pero mas adelante va a ser fundamental y una gran ayuda.

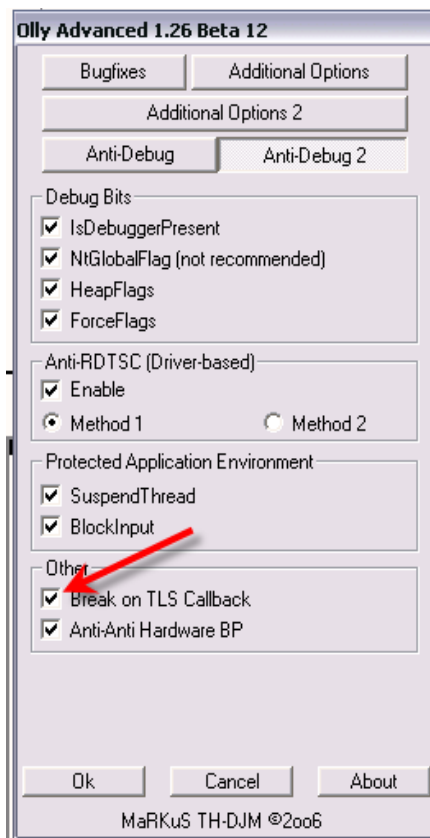
Alli vemos en el UPX desempaado su EP





Como vemos es el tipico unpackme de Teddy Rogers, ya hemos hecho varios en esta introduccion, y el de execryptor como esta hecho con el mismo codigo y luego empacado tendra el mismo OEP=4271B0, la misma IAT, la primera api que llama es GetVersion, etc, esto en packers muy complejos es una gran ayuda.

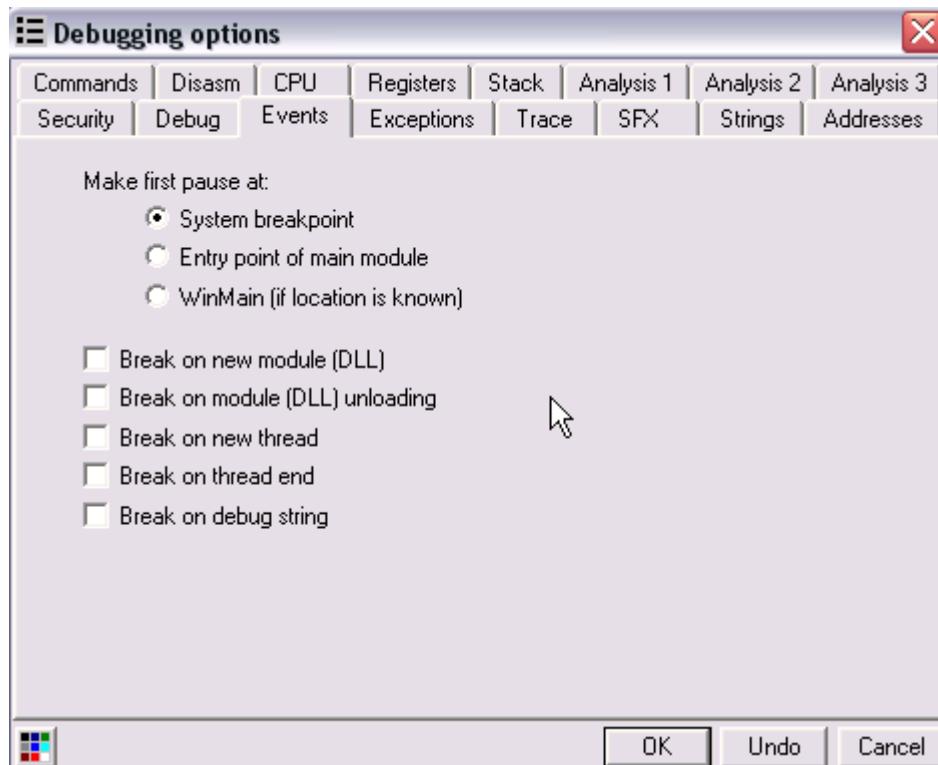
Bueno ahora empecemos con el mas facil execryptor.que es el A.  
Uso el plugin OLLYADVANCED beta 12



Utilizo el parcheado 4, que uso siempre y yo tengo muchas opciones marcadas en el advanced, la mayoría no son necesarias pero la que si es importante es la que esta en la imagen BREAK ON TLS CALLBACK.

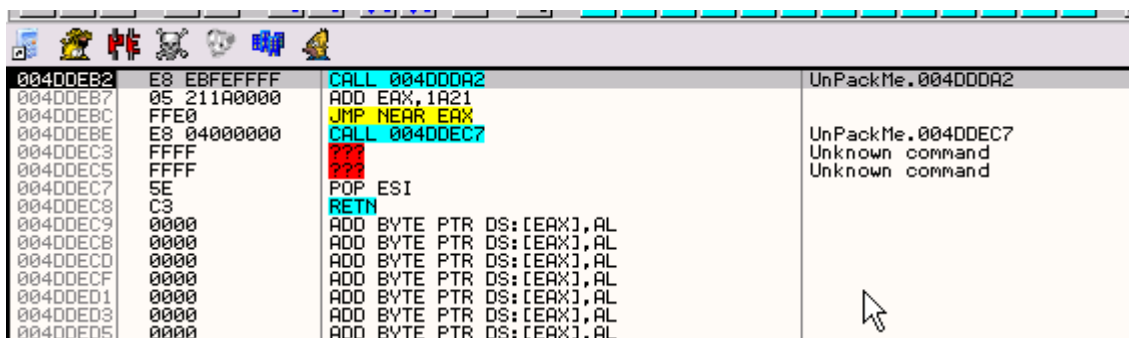
Creo que ya todos saben que el execryptor antes de llegar al EP, ejecuta codigo mediante un truco que ya ha sido explicado hasta el hartazgo, pero es que para en el TLS CALLBACK antes del EP.

Bueno la historia es que uno que hacia virus se dio cuenta que el sistema permitia ejecutar codigo antes del EP si se activaba dicha opcion, y ahi fue el autor de execryptor y copio la idea, por eso normalmente el execryptor se cierra antes de llegar al EP en OLLY, porque lo detecta antes de llegar al mismo.

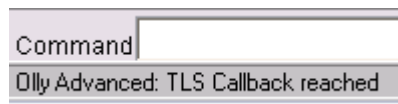


Hay que cambiar el OLLY para que pare en el SYSTEM BREAKPOINT.

Una vez que para alli si doy RUN me para en el TLS CALLBACK



Vemos abajo que el OLLY dice



Bueno el que quiere hallar el valor a mano va al header con GOTO EXPRESSION = 400000 y pone en modo SPECIAL-PE HEADER y si bajamos veremos.

00400148	00000000	DD 00000000	Global Ptr address = 0
0040014C	00000000	DD 00000000	Must be 0
00400150	10310900	DD 00093110	TLS Table address = 93110
00400154	18000000	DD 00000018	TLS Table size = 3 (24.)
00400158	00000000	DD 00000000	Load Config Table address = 0
0040015C	00000000	DD 00000000	Load Config Table size = 0

Vemos que dice 93110, así que sumándole la imagebase daría 493110, veamos en el DUMP que hay allí y es el inicio de la TLS TABLE, el valor que buscamos está allí, apenas más abajo..

The screenshot shows the OLLYMPIC interface with the assembly window and the hex dump window. The assembly window shows the following code:

```

0040DEB2  E8 EBFEFFFF  CALL 0040DDA2
0040DEB7  05 211A0000  ADD EAX,1A21
0040DEBE  FFE0        JMP NEAR EAX
0040DEC5  E8 04000000  CALL 0040DEC7
0040DEC6  FFFF
0040DEC8  FFFF
0040DEC9  5E          POP ESI
0040DECA  C3          RETN
0040DECB  0000        ADD BYTE PTR DS:[EAX],AL
0040DECD  0000        ADD BYTE PTR DS:[EAX],AL
0040DECE  0000        ADD BYTE PTR DS:[EAX],AL
0040DECF  0000        ADD BYTE PTR DS:[EAX],AL

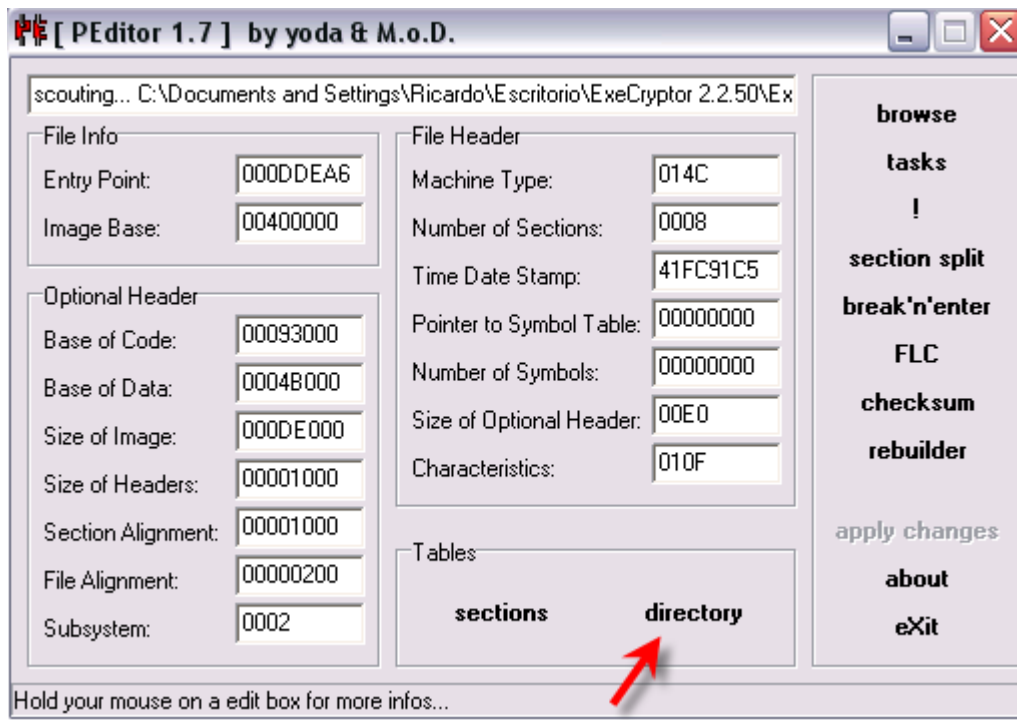
```

The hex dump window shows the following data:

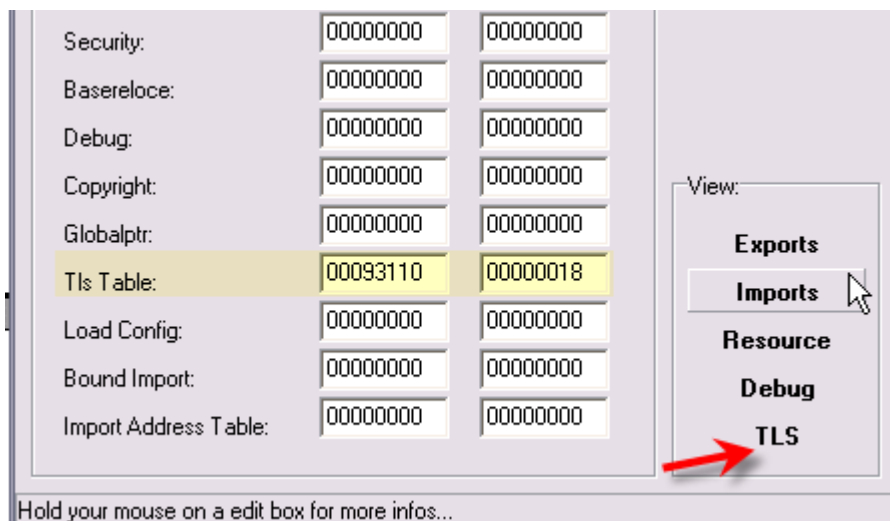
Address	Hex dump	ASCII
00493110	00 00 00 00 00 00 00 00 28 31 49 00 2C 31 49 00	.....(11,11.
00493120	00 00 00 00 00 00 00 00 00 00 00 00 B2 DE 4D 00	.....IM.
00493130	00 00 00 00 56 51 89 C6 89 D1 83 E9 04 FC AC 00	....VQ&g&0&u&?%\$
00493140	E8 80 F8 74 75 0E 8B 06 0F C8 01 17 89 06 83 C6	pC°tu&I&*%0&e&3&
00493150	04 83 E9 04 49 7F E7 59 5E C3 8B C6 00 10 40 00	4&u&I&Y^fLp0.
00493160	4A 43 00 92 04 00 00 00 00 00 F8 BA 05 20 FF A3	JC.#&.....?^' u

Allí localizamos donde comenzará a ejecutarse el execryptor, aunque el plugin OLLY ADVANCED lo hizo también por nosotros y paró correctamente.

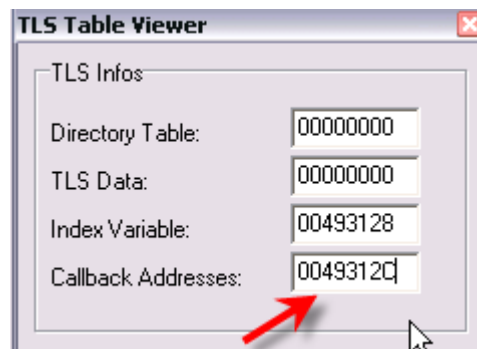
También si miramos el archivo en PEEDITOR



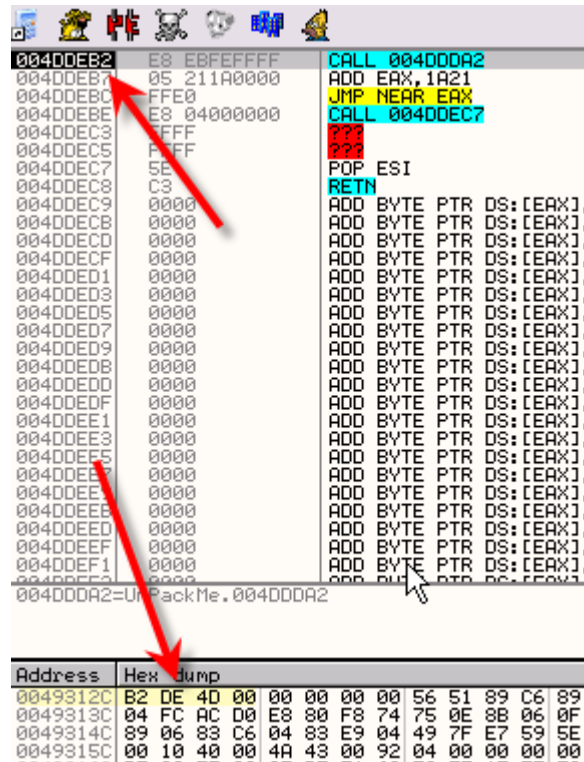
Alli en DIRECTORY



Alli vemos que nos dice que la TLS TABLE empieza en 493110, y el largo es 18, pero nos da mas precisiones si apretamos TLS nos dice donde hallar exactamente el valor en la tabla.

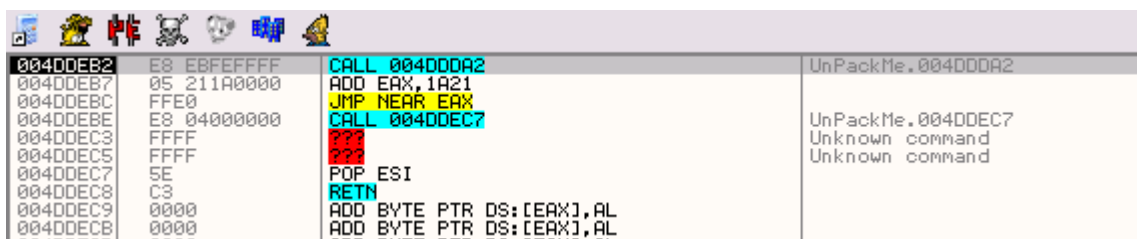


O sea que para mirar la direccion donde comienza a ejecutarse hay que mirar alli en 49312C, en el

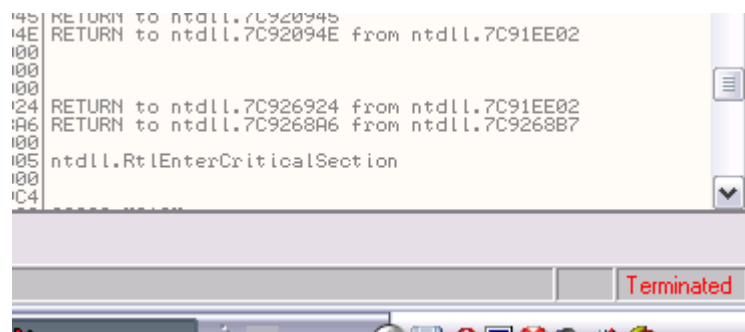


Que all nos indica donde empezará nomas, jeje, bueno ya sabemos como hallar el valor en OLLY, tambien con PEEDITOR, si tenemos el OLLY ADVANCED pues parara en el TLS CALLBACK, y no necesitaremos hallarlo a mano y si no lo tenemos cuando estamos en el system breakpoint ponemos un BP en el valor que acabamos de hallar y parara tambien.

La cuestion que estamos parados en la direccion donde comienza a ejecutarse.



Si doy RUN



Detecta algo y se cierra, yo tengo todas las opciones del advanced activadas, y ademas probe en otro OLLY parcheado con el AntiDetectOllly v2.2.4.exe, y lo mismo lo detecta igual, pero veamos

volvamos aqui.

004DDEB2	E8 EBFEBFFF	CALL 004DDDA2	UnPackMe.004DDDA2
004DDEB7	05 211A0000	ADD EAX,1A21	
004DDEBC	FFE0	JMP NEAR EAX	
004DDEBE	E8 04000000	CALL 004DDEC7	UnPackMe.004DDEC7
004DDEC3	FFFF	Unknown command	
004DDEC5	FFFF	Unknown command	
004DDEC7	5E	POP ESI	
004DDEC8	C3	RETN	
004DDEC9	0000	ADD BYTE PTR DS:[EAX],AL	
004DDECA	0000	ADD BYTE PTR DS:[EAX],AL	
004DDECB	0000	ADD BYTE PTR DS:[EAX],AL	

Si me fijo en la ventana de breakpoints aunque yo no coloque ninguno, hay uno

Address	Module	Active	Disassembly
004DDEA6	UnPackMe	One-shot	CALL 004DDDA2

Ah es el BP que pone el mismo OLLYDBG para que pare en el EP del programa, recordemos que execryptor se ejecuta antes, por lo tanto como es un BREAKPOINT que siempre coloca OLLYDBG, puede detectarlo ya que como sabemos es un CC colocado en dicha poiscion de memoria, borremoslo a ver que pasa y demos RUN.



Ah pillin detecta ese BP que el OLLY coloca y si esta puesto se termina, si lo borramos a mano, pues arranca, si no les arranca a pesar de borrar ese BP, deben agregar alguna opcion mas en la parte de antidebug del advanced, y recordar que estamos usando el OLLYDBG llamado PARCHEADO 4, no hubo necesidad hasta ahora de otra cosa.

La cuestion que ya sabemos como corre reiniciemos el OLLY.

004DDEB2	E8 EBFEBFFF	CALL 004DDDA2	UnPackMe.004DDDA2
004DDEB7	05 211A0000	ADD EAX,1A21	
004DDEBC	FFE0	JMP NEAR EAX	
004DDEBE	E8 04000000	CALL 004DDEC7	UnPackMe.004DDEC7
004DDEC3	FFFF	Unknown command	
004DDEC5	FFFF	Unknown command	
004DDEC7	5E	POP ESI	
004DDEC8	C3	RETN	
004DDEC9	0000	ADD BYTE PTR DS:[EAX],AL	
004DDECB	0000	ADD BYTE PTR DS:[EAX],AL	
004DDECD	0000	ADD BYTE PTR DS:[EAX],AL	

Coloquemos un BREAK ON EXECUTE en la seccion code, la que comienza en 401000, suponemos que se desempacara alli.

Address	Disassembly	Comment	Map	R	R/E
0E0000	00003000	UnPackMe	PE header	Image	R/W
400000	00001000	text			R/W
401000	00004000				R/W
44B000	0000C300	UnP.			R/W
457000	00009000	Actualize			R/W
460000	00003000	View in Disassembler		Enter	R/W
463000	00008000	Dump in CPU			R/W
468000	00001000	Dump			R/W
46C000	00027000	Search		Ctrl+B	R/W
493000	0004B000				R/W
4E0000	0000B000				R/W
5A0000	00002000				R/W
5B0000	00103000				R/W
6C0000	001A4000				R/W
D10000	00001000	use:		F2	R/W
D11000	0000F000				R/W
D70000	00002000	use:			R/W
D72000	00002B00	Set memory breakpoint on access			R/W
D90000	00003000	use:			R/W
EF0000	00001000	Set memory breakpoint on write			R/W
EF1000	00042000	GDI:			R/W
F33000	00001000	Set access			R/W
F34000	00001000	GDI:			R/W
F35000	00002000	Allocate Memory			R/W
800000	00001000	GDI:			R/W
801000	00002000	Free Memory			R/W
803000	00005000	kern:			R/W
808000	00073000	Zero Memory			R/W
8FB000	00006000	kern:			R/W
910000	00001000	Dump Memory-Area			R/W
911000	0007B000	ntd:			R/W
98C000	00005000	Load dumped memory			R/W
991000	00032000	ntd:			R/W
9C3000	00003000	ntd:			R/W
6F0000	00007000	Set break-on-execute			R/W
F60000	00001000				R/W
7F0000	00001000	Copy to clipboard			R/W

Volvamos a borrar el BP ya que se coloca cada vez que reiniciamos

Address	Module	Active	Disassembly	Comment
004000A6	UnPackMe	One-shot	CALL 004000A2	

Y ahora si luego de borrarlo demos RUN.

[illegible]

Vemos que para en el OEP conocido que era 4271B0, esta version A aun no tiene stolen bytes ni proteccion del oep ni nada, asi que estamos parados en el OEP.

Address	Disassembly	Comment
00427180	55	PUSH EBP
00427181	8BEC	MOV EBP,ESP
00427183	6A FF	PUSH -1
00427185	68 600E4500	PUSH 450E60
0042718A	68 C8924200	PUSH 4292C8
0042718F	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	50	PUSH EAX
004271C6	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
004271CD	83C4 A8	ADD ESP,-58
004271D0	53	PUSH EBX
004271D1	56	PUSH ESI
004271D2	57	PUSH EDI
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]
004271DC	33D2	XOR EDX,EDX
004271DE	8AD4	MOV DL,AH
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX



Veamos el inicio y final de la IAT

Empieza en 460818 y se puede comparar con el desempacado de UPX, por supuesto en el empieza tambien en la misma direccion, es conveniente verificar, para ver que no haya diferencia ni trucos raros.

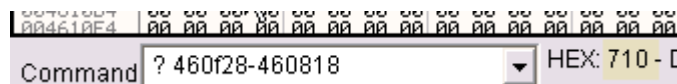
El final si bajamos es

Address	Hex	dump	ASCII
004600B8	40 64 48 00 37 62 48 00 06 B8 48 00 2C EE 48 00	@dH.7bH.7bH.7bH.	@dH.7bH.7bH.7bH.
004600C8	E1 0F 47 00 42 98 48 00 64 24 48 00 7D 44 47 00	88F.88F.88F.88F.	88F.88F.88F.88F.
004600D8	02 C3 46 00 42 04 46 00 0A 6E 48 00 53 C8 46 00	88F.88F.88F.88F.	88F.88F.88F.88F.
004600E8	CE 29 48 00 68 0E 47 00 84 83 47 00 82 69 48 00	88F.88F.88F.88F.	88F.88F.88F.88F.
004600F8	13 74 48 00 C0 0B 47 00 47 F5 46 00 03 F1 48 00	88F.88F.88F.88F.	88F.88F.88F.88F.
00460E08	07 FC 47 00 47 43 48 00 9A 54 47 00 0D 19 47 00	88F.88F.88F.88F.	88F.88F.88F.88F.
00460E18	E7 F9 46 00 15 45 47 00 E8 CF 46 00 CE C6 46 00	88F.88F.88F.88F.	88F.88F.88F.88F.
00460E28	92 AD 47 00 77 91 48 00 19 6B 47 00 E6 77 47 00	88F.88F.88F.88F.	88F.88F.88F.88F.
00460E38	FA C4 48 00 71 76 47 00 89 F9 47 00 E3 F3 48 00	88F.88F.88F.88F.	88F.88F.88F.88F.
00460E48	8D 05 46 00 06 17 47 00 AB 38 47 00 7D C0 47 00	88F.88F.88F.88F.	88F.88F.88F.88F.
00460E58	64 2A 48 00 F6 CD 48 00 2A 92 47 00 77 0B 47 00	88F.88F.88F.88F.	88F.88F.88F.88F.
00460E68	40 CE 47 00 05 29 48 00 41 D2 48 00 31 58 47 00	88F.88F.88F.88F.	88F.88F.88F.88F.
00460E78	0D 92 48 00 54 DA 46 00 76 0F 47 00 49 F7 47 00	88F.88F.88F.88F.	88F.88F.88F.88F.
00460E88	00 00 00 00 F7 A8 B1 76 00 00 00 00 C8 74 F8 72	88F.88F.88F.88F.	88F.88F.88F.88F.
00460E98	73 66 F9 72 87 72 F8 72 43 80 F8 72 67 37 F9 72	88F.88F.88F.88F.	88F.88F.88F.88F.
00460EA8	FB 41 F9 72 67 83 F8 72 90 53 F8 72 00 00 00 00	88F.88F.88F.88F.	88F.88F.88F.88F.
00460EB8	CE 00 37 76 7C 86 37 76 80 86 37 76 33 25 36 76	88F.88F.88F.88F.	88F.88F.88F.88F.
00460EC8	1E 31 36 76 D8 7C 37 76 89 C2 37 76 CD 46 38 76	88F.88F.88F.88F.	88F.88F.88F.88F.
00460ED8	CE EE 36 76 00 00 00 00 48 D0 4C 77 9C CB 4D 77	88F.88F.88F.88F.	88F.88F.88F.88F.
00460EE8	CC 42 4F 77 2C D0 4C 77 DA F6 4C 77 73 33 50 77	88F.88F.88F.88F.	88F.88F.88F.88F.
00460EF8	10 64 4D 77 03 0E 52 77 33 0F 52 77 40 A6 54 77	88F.88F.88F.88F.	88F.88F.88F.88F.
00460F08	F1 A7 54 77 92 9C 4F 77 6F 57 52 77 99 33 4E 77	88F.88F.88F.88F.	88F.88F.88F.88F.
00460F18	B2 5D 4E 77 90 C0 5A 77 00 00 00 00 F3 F0 C0 74	88F.88F.88F.88F.	88F.88F.88F.88F.
00460F28	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	88F.88F.88F.88F.	88F.88F.88F.88F.
00460F38	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	88F.88F.88F.88F.	88F.88F.88F.88F.



Address	Hex dump	ASCII
00460F04	40 A6 54 77 F1 A7 54 77 92 9C 4F 77 6F 57 52 77	@@Tw:0Tw#60w0WRw
00460F14	99 33 4E 77 B2 5D 4E 77 90 C0 5A 77 00 00 00 00	03Nw0INwE'Zw...
00460F24	F3 F0 CC 74 00 00 00 00 00 00 00 00 00 00 00	%-!ft.....
00460F34	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460F44	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

El final es 460f28 para que entre la ultima entrada, asi que el final menos el inicio nos da el largo.

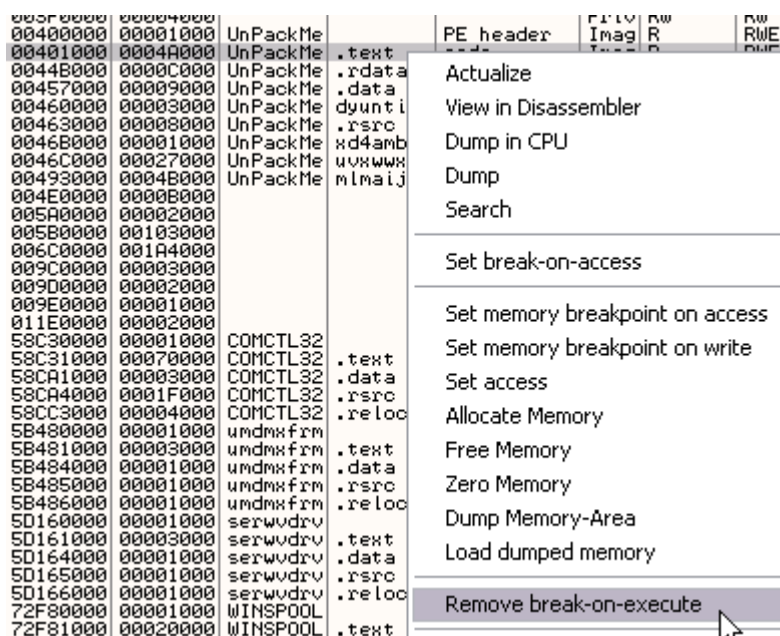


El largo es 710 asi que recopilando

**OEP:271b0**  
**RVA o INICIO:60818**  
**SIZE:710**

Alli ya tenemos los datos para el IMPORT RECONSTRUCTOR

El tema es ver como reparar la IAT aquí no hay salto magico ya que vamos a ver que execryptor resuelve la entrada y la guarda en la IAT con el valor correcto cada vez que ejecuta una api, si ponemos un BPM ON WRITE en la entrada de la IAT correspondiente a GetVersion.



Quitamos primero el BREAK ON EXECUTE si no dara error.

Ahora veamos la entrada correspondiente a la primera api llamada.

00427101	56	PUSH ESI	
00427102	57	PUSH EDI	
00427103	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
00427106	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	UnPackMe.00492493
0042710C	33D2	XOR EDX,EDX	
0042710E	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
004271E6	8BC8	MOV ECX,EAX	
004271E8	81E1 FF000000	AND ECX,0FF	
004271EE	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX	
004271F4	C1E1 08	SHL ECX,8	
004271F7	03CA	ADD ECX,EDX	
004271F9	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX	
004271FF	C1EB 10	SHR ECX,10	
DS:[00460ADC]=00492493 (UnPackMe.00492493)			
Address	Hex dump	ASCII	
00460ADC	93 24 49 00 98 F3 46 00 10 E4 46 00 9C 86 48 00	03I.0%F.0%F.6BH.	

Pongamosle un MEMORY BREAKPOINT ON WRITE alli a ver que pasa.

0046F387	8905 DC0A4600	MOV DWORD PTR DS:[460ADC],EAX	kernel32.GetVersion
0046F38D	8D05 30904700	LEA EAX,DWORD PTR DS:[479030]	
0046F393	C600 C3	MOV BYTE PTR DS:[EAX],0C3	
0046F396	E9 959C0000	JMP 00479030	UnPackMe.00479030
0046F39B	E8 EF380000	CALL 00472C8F	UnPackMe.00472C8F
0046F3A0	FF25 E00A4600	JMP NEAR DWORD PTR DS:[460AE0]	UnPackMe.0046F39B
0046F3A6	0F84 B70C0200	JE 00490063	UnPackMe.00490063
0046F3AC	E9 79B70000	JMP 0047AB2A	UnPackMe.0047AB2A
0046F3B1	0F85 1EDBFFFF	JNZ 0046CED5	UnPackMe.0046CED5
0046F3B7	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
0046F3BA	8B40 F8	MOV EAX,DWORD PTR DS:[EAX-8]	
0046F3BD	83C8 08	OR EAX,8	
0046F3C0	E9 ED900100	JMP 004884B2	UnPackMe.004884B2
0046F3C5	5B	POP EBX	
0046F3C6	81CB 8B3B989C	OR EBX,9C983B8B	
0046F3CC	F7C3 00000080	TEST EBX,80000080	
0046F3D2	E9 976F0100	JMP 0048636E	UnPackMe.0048636E
0046F3D7	81F2 05780920	XOR EAX,200978F5	

Alli guardara la direccion correcta de la api en la entrada correspondiente.

00492498	FF25 DC0A4600	JMP NEAR DWORD PTR DS:[460ADC]	kernel32.GetVersion
0049249E	81D0 B7D9C399	ADC EAX,99C3D9B7	
004924A4	0F86 13A4FFFF	JBE 0048C8BD	UnPackMe.0048C8BD
004924AA	3BD7	CMP EDX,EDI	

Una linea mas abajo salta a la direccion correcta, asi que cada vez que usa una api, repara la entrada correspondiente de la IAT, con lo cual, seria lo mas logico para reparar la IAT, parar cuando el programa ya se ejecuto y se va a cerrar, pero igual estudiaremos un poco mas.

0046F387	8905 DC0A4600	MOV DWORD PTR DS:[460ADC],EAX	
0046F38D	8D05 30904700	LEA EAX,DWORD PTR DS:[479030]	
0046F393	C600 C3	MOV BYTE PTR DS:[EAX],0C3	
0046F396	E9 959C0000	JMP 00479030	
0046F39B	E8 EF380000	CALL 00472C8F	

Vemos alli que luego de guardar la api, guarda un C3 en 479030 que carga con el LEA, veamos que hace realmente.

La direccion 479030 antes de esto es

00479030	0F89 5FDA0000	JNS 00486A95	UnPackMe.00486A95
00479036	871C24	XCHG DWORD PTR SS:[ESP],EBX	
00479039	8B03	MOV EDX,EBX	
0047903B	871424	XCHG DWORD PTR SS:[ESP],EDX	
0047903E	8BDA	MOV EBX,EDX	
00479040	E9 9190FFFF	JMP 004720D6	UnPackMe.004720D6
00479045	8905 F40D4600	MOV DWORD PTR DS:[460DF4],EAX	
00479048	8D05 AC464700	LEA EAX,DWORD PTR DS:[4746AC]	
00479051	C600 C3	MOV BYTE PTR DS:[EAX],0C3	
00479054	E9 6B270100	JMP 0048B7C4	UnPackMe.0048B7C4
00479059	5A	POP EDX	
0047905A	E9 99EAFFFF	JMP 00477AF8	UnPackMe.00477AF8
0047905F	81E9 09CFC88D	SUB ECX,8DC8CF09	
00479065	C1C1 12	ROL ECX,12	

Y cuando guarda el C3 queda

00479030	C3	RETN	
00479031	895F DA	MOV DWORD PTR DS:[EDI-26],EBX	
00479034	0000	ADD BYTE PTR DS:[EAX],AL	
00479036	871C24	XCHG DWORD PTR SS:[ESP],EBX	
00479039	8B03	MOV EDX,EBX	
0047903B	871424	XCHG DWORD PTR SS:[ESP],EDX	
0047903E	8BDA	MOV EBX,EDX	
00479040	E9 9190FFFF	JMP 004720D6	
00479045	8905 F40D4600	MOV DWORD PTR DS:[460DF4],EAX	

O sea que pone un RETN alli, se esta automodificando al arreglar una api cambio una parte del codigo , pero dicho codigo no es de la primera seccion si no del codigo de execryptor, hmmm veamos cuando accede a esta direccion reiniciemos lleguemos al OEP y pongamos un BPM ON ACCESS en dicha instrucción de 479030.

C File View Debug Plugins Options Window Help			
L E M T W H C / K B R ... S			
00479030	0F89 5FDA0000	JNS 00486A95	00486A95
00479036	871C24	XCHG DWORD PTR SS:[ESP],EBX	
00479039	8B03	MOV EDX,EBX	
0047903B	871424	XCHG DWORD PTR SS:[ESP],EDX	
0047903E	8BDA	MOV EBX,EDX	
00479040	E9 9190FFFF	JMP 004720D6	004720D6
00479045	8905 F40D4600	MOV DWORD PTR DS:[460DF4],EAX	
00479048	8D05 AC464700	LEA EAX,DWORD PTR DS:[4746AC]	
00479051	C600 C3	MOV BYTE PTR DS:[EAX],0C3	
00479054	E9 6B270100	JMP 0048B7C4	0048B7C4
00479059	5A	POP EDX	
0047905A	E9 99EAFFFF	JMP 00477AF8	00477AF8
0047905F	81E9 09CFC88D	SUB ECX,8DC8CF09	
00479065	C1C1 12	ROL ECX,12	
00479068	81C1 3C0C697E	ADD ECX,7E69DC3C	
0047906E	870C24	XCHG DWORD PTR SS:[ESP],EBX	
00479071	E9 3E4D0100	JMP 0048D0B4	
00479076	85DA	TEST EDX,EBX	
00479078	E9 7EE90000	JMP 004879FB	
0047907D	81F0 E99B22CB	XOR EAX,CB229BE9	
00479083	E8 B3500000	CALL 0047E138	
00479088	56	PUSH ESI	
00479089	81EE C2488074	SUB ESI,748048C2	
0047908F	8B5F 30000000	MOV EDI,DWORD PTR DS:[45F6341] EAX	

Lleguemos hasta la primera llamada a una api.

00427101	56	PUSH ESI	
00427102	57	PUSH EDI	
00427103	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
00427106	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	UnPackMe.00492493
0042710C	33D2	XOR EDX,EDX	
0042710E	8AD4	MOV DL,AH	
00427110	8915 34F64500	MOV DWORD PTR DS:[45F6341] EAX	

Pongamos un BP en el RETORNO o sea en 4271DC.

004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	UnPackMe.00492493
004271D8	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	

Ahora demos RUN

00479030	0F89 5FDA0000	JNS 00486A95	UnPackMe.00486A95
00479036	871C24	XCHG DWORD PTR SS:[ESP],EBX	
00479039	8BD3	MOV EDX,EBX	
0047903B	871424	XCHG DWORD PTR SS:[ESP],EDX	
0047903E	8BD3	MOV EBX,EDX	
00479040	E9 9190FFFF	JMP 004720D6	UnPackMe.004720D6
00479045	8905 F4004600	MOV DWORD PTR DS:[4600F4],EAX	

para cuando ejecuta ese salto, o sea que eso es parte de la rutina que reconoce la api, o sea que pasa primero por alli y luego pone un RET.

O sea que automodifica la rutina que detecta las apis, podemos reiniciar y ver si hace mas cambios, llegando al OEP, y poniendo un BPM ON WRITE en la seccion de excryptor adonde estan redireccionadas las apis y la rutina esa, veamos lleguemos al OEP.

004271C5	50	PUSH EAX	
004271C6	64:8925 000000	MOV DWORD PTR FS:[0],ESP	
004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	
004271D1	56	PUSH ESI	
004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	UnPackMe.00492493
004271D8	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
004271E6	8BC8	MOV ECX,EAX	
004271E8	81E1 FF000000	AND ECX,0FF	

Ahi estoy de nuevo en la api, veo que saltara a la redireccion que se encuentra en 492493, y que corresponde a la misma seccion del salto que cambia por RET en 479030, asi que pongamos un BPM ON WRITE en toda esa seccion y veremos que cambia en ella..

00460000	00003000	UnPackMe	dyuntioj		Imag	R	RI
00463000	00008000	UnPackMe	.rsrc	resources	Imag	R	RI
0046B000	00001000	UnPackMe	xd4ambdu		Imag	R	RI
0046C000	00027000	UnPackMe	xxxxxxis		Imag	R	RI
00493000	0004B000	UnPackMe					
004E0000	00009000						
005A0000	00002000						
005B0000	00103000						
006C0000	00117000						
009C0000	00003000						
009D0000	00002000						
009E0000	00001000						
011E0000	00002000						
58C30000	00001000	COMCTL3					
58C31000	00070000	COMCTL3					
58CA1000	00003000	COMCTL3					
58CA4000	0001F000	COMCTL3					
58CC3000	00004000	COMCTL3					
5B480000	00001000	undmxf					
5B481000	00003000	undmxf					
5B484000	00001000	undmxf					
5B485000	00001000	undmxf					
5B486000	00001000	undmxf					

004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	UnPackMe.00492493
004271D8	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	

Ahora si coloquemos el BP en el retorno y demos RUN y controlaremos los cambios que vaya realizando en la seccion y en la rutina de las apis especialmente en esta primera llamada.

004806D1	8802	MOV BYTE PTR DS:[EDX],AL
004806D3	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]
004806D6	8A00	MOV AL,BYTE PTR DS:[EAX]
004806D8	0045 F6	ADD BYTE PTR SS:[EBP-A],AL
004806DB	33C0	XOR EAX,EAX
004806DD	8A45 F6	MOV AL,BYTE PTR SS:[EBP-A]
004806E0	✓ E9 3E150000	JMP 00481C23
004806E5	81C1 F68EDF64	ADD ECX,64DF8EF6
004806EB	870C24	XCHG DWORD PTR SS:[ESP],ECX
004806EE	^ E9 9478FFFF	JMP 00477F87

AL=4B ('K')  
DS:[0047A9E8]=B4

Sigamos y copiemos

004806D1	8802	MOV BYTE PTR DS:[EDX],AL
004806D3	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]
004806D6	8A00	MOV AL,BYTE PTR DS:[EAX]
004806D8	0045 F6	ADD BYTE PTR SS:[EBP-A],AL
004806DB	33C0	XOR EAX,EAX
004806DD	8A45 F6	MOV AL,BYTE PTR SS:[EBP-A]
004806E0	✓ E9 3E150000	JMP 00481C23
004806E5	81C1 F68EDF64	ADD ECX,64DF8EF6
004806EB	870C24	XCHG DWORD PTR SS:[ESP],ECX
004806EE	^ E9 9478FFFF	JMP 00477F87

AL=45 ('E')  
DS:[0047A9E9]=45 ('E')

004806D1	8802	MOV BYTE PTR DS:[EDX],AL
004806D3	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]
004806D6	8A00	MOV AL,BYTE PTR DS:[EAX]
004806D8	0045 F6	ADD BYTE PTR SS:[EBP-A],AL
004806DB	33C0	XOR EAX,EAX
004806DD	8A45 F6	MOV AL,BYTE PTR SS:[EBP-A]
004806E0	✓ E9 3E150000	JMP 00481C23
004806E5	81C1 F68EDF64	ADD ECX,64DF8EF6
004806EB	870C24	XCHG DWORD PTR SS:[ESP],ECX
004806EE	^ E9 9478FFFF	JMP 00477F87

AL=52 ('R')  
DS:[0047A9EA]=94

Address	Hex dump	

Adelante

004806D1	8802	MOV BYTE PTR DS:[EDX],AL
004806D3	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]
004806D6	8A00	MOV AL,BYTE PTR DS:[EAX]
004806D8	0045 F6	ADD BYTE PTR SS:[EBP-A],AL
004806DB	33C0	XOR EAX,EAX
004806DD	8A45 F6	MOV AL,BYTE PTR SS:[EBP-A]
004806E0	✓ E9 3E150000	JMP 00481C23
004806E5	81C1 F68EDF64	ADD ECX,64DF8EF6
004806EB	870C24	XCHG DWORD PTR SS:[ESP],ECX

AL=4E ('N')  
DS:[0047A9EB]=9C

Vemos que esta cambiando bytes consecutivos, no los pegare todos, para no hacer tan grande el tute, sigamos vamos mirando en el DUMP la zona que esta cambiando.

Address	Hex dump	ASCII
0047A9E8	4B 45 52 4E 45 4C 33 32 2E 64 6C 6C 00 E9 A1 AF	KERNEL32.dll.üi>
0047A9F8	FF FF 0F 84 CF E7 FF FF 68 72 9D 07 4E 5A E9 4C	*âð hr0•NZüL
0047AA08	C2 00 00 0F 85 AF 96 00 00 C1 EB 03 F7 C3 B4 6F	т...*â»ü...+ü♦•Ho

Bueno alli realmente esta guardando el nombre de la dll en la cual buscará la api, asi que sigamos.

Address	Hex dump	ASCII
0047A9E8	4B 45 52 4E 45 4C 33 32 2E 64 6C 6C 00 E9 A1 AF	KERNEL32.dll.üi>
0047A9F8	FF FF 0F 84 CF E7 FF FF 68 72 9D 07 4E 5A E9 4C	*äð hr0.NZÜL

Alli guarda cero en la misma zona seguro sera el final del nombre.

Address	Hex dump	ASCII
0047A9E8	4B 45 52 4E 45 4C 33 32 2E 64 6C 6C 00 E9 A1 AF	KERNEL32.dll.üi>
0047A9F8	FF FF 0F 84 CF E7 FF FF 68 72 9D 07 4E 5A E9 4C	*äð hr0.NZÜL

Repite guardando B4 sobrescribiendo la K y asi hara con todo el nombre seguro ya lo habra usado y lo oculta.

Address	Hex dump	ASCII
00474954	00 00 00 00 0F 83 E1 4F 01 00 8B 15 48 EE 47 00	...*äð.üi>
00474964	09 02 0F 85 04 04 00 00 F9 A1 F2 FF FF 03 05 01	...*äð.üi>

Alli guarda la direccion base de la kernel32.dll que en mi maquina es 7c800000



0046F393	C600 C3	MOV BYTE PTR DS:[EAX],0C3	
0046F396	E9 959C0000	JMP 00479030	
0046F398	E8 EF380000	CALL 00472C8F	
0046F3A0	FF25 E00A4600	JMP NEAR DWORD PTR DS:[460AE0]	
0046F3A6	0F84 B70C0200	JE 00490063	
0046F3AC	E9 79B70000	JMP 0047AB2A	
0046F3B1	0F85 1EDBFFFF	JNZ 0046CED5	
0046F3B7	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
0046F3BA	8B40 F8	MOV EAX,DWORD PTR DS:[EAX-8]	
0046F3BD	83C8 08	OR EAX,8	
0046F3C0	E9 ED900100	JMP 004884B2	
0046F3C5	5B	POP EBX	
0046F3C6	81CB 8B3B989C	OR EBX,9C983B8B	
0046F3CC	F7C3 00000080	TEST EBX,80000080	
0046F3D2	E9 976F0100	JMP 0048636E	
0046F3D7	81F2 D57BD920	XOR EDX,20D97BD5	
0046F3DD	81EA 0E60598F	SUB EDX,8F59600E	
0046F3E3	81C2 16C01B80	ADD EDX,801BC016	
DS:[00479030]=0F			
Address	Hex dump	ASCII	
00474954	00 00 80 7C 0F 83 E1 1F 01 00 8B 15 48 EE 47 00	..C!*:	
00474964	09 D2 0F 85 D4 A4 00 00 E9 A1 E2 FF FF 03 D5 81	.E*ãEñ	

Y alli despues de guardar la direccion como vemos en la IAT pone el C3, entonces la cosa va tomando mejor color ya que seguramente mete ese RET alli para no pasar nuevamente dos veces por la misma rutina es una forma de cerrar el camino de entrada que ya no necesita porque ya paso por alli, y al estar la entrada de la IAT reparada no volvera a pasar por alli.

004271C0	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	
004271D1	56	PUSH ESI	
004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	kernel32.GetVersion
004271DC	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	
004271F0	8915 34F64500	MOV DWORD PTR DS:[45F6341],EDX	

Y llega sin mas cambios al BP, borremoslo, y traceemos con f7 hasta la siguiente api.

004293A0	6A 00	PUSH 0	
004293A2	68 00100000	PUSH 1000	
004293A7	6A 00	PUSH 0	
004293A9	FF15 A0094600	CALL NEAR DWORD PTR DS:[4609A0]	UnPackMe.00482308
004293AF	85C0	TEST EAX,EAX	
004293B1	A3 C4EB4500	MOV DWORD PTR DS:[45EBC4],EAX	
004293B6	75 01	JNZ SHORT 004293B9	UnPackMe.004293B9
004293B8	C3	RET	

Pongamos el BP en el retorno y demos RUN

004293A2	68 00100000	PUSH 1000	
004293A7	6A 00	PUSH 0	
004293A9	FF15 A0094600	CALL NEAR DWORD PTR DS:[4609A0]	UnPackMe.00482308
004293AF	85C0	TEST EAX,EAX	
004293B1	A3 C4EB4500	MOV DWORD PTR DS:[45EBC4],EAX	
004293B6	75 01	JNZ SHORT 004293B9	UnPackMe.004293B9

004808B7	C600 C3	MOV BYTE PTR DS:[EAX],0C3
004808BA	^ E9 B851FFFF	JMP 00482A77
004808BF	8B0424	MOV EAX,DWORD PTR SS:[ESP]
004808C2	50	PUSH EAX
004808C3	^ E9 45F2FFFF	JMP 0048CB00
004808C8	8005 67324700	LEA EAX,DWORD PTR DS:[473267]
004808CE	C600 C3	MOV BYTE PTR DS:[EAX],0C3
004808D1	^ E9 2826FFFF	JMP 0047FEFE
004808D6	8B0424	MOV EAX,DWORD PTR SS:[ESP]
004808D9	52	PUSH EDX
004808DA	E8 51E8FEFF	CALL 0047C130
004808DF	B8 FC862613	MOV EAX,132606FC
004808E4	53	PUSH EBX
004808E5	E8 8761FEFF	CALL 00473A71
004808EA	^ E9 8B41FFFF	JMP 00481A7A
004808EF	^ E9 BEFEFDFD	JMP 0046D7B2
004808F4	870C24	XCHG DWORD PTR SS:[ESP],ECX
004808F7	^ E9 FED0FEFF	JMP 0047A9FA
004808FC	^ 0F84 E55BFEFF	JE 004734E7

DS:[0047A0BC]=5A ('Z')

para donde coloca otro RET aquí en 47A0BC

0047A09E	C1C9 12	ROR ECX,12	
0047A0A1	^ 0F87 A554FFFF	JA 0046F540	UnPackMe.0046F54C
0047A0A7	^ E9 BF1F0000	JMP 0047C06B	UnPackMe.0047C06B
0047A0AC	- FF25 A0094600	JMP NEAR DWORD PTR DS:[4609A0]	kernel32.HeapCreat
0047A0B2	^ E9 9CE90000	JMP 00488A53	UnPackMe.00488A53
0047A0B7	^ E9 413D0100	JMP 0048DDFD	UnPackMe.0048DDFD
0047A0BC	C3	RETN	

Luego de eso ya llega al BP luego de cambiar la IAT, vemos que esta corresponde tambien a kernel32.dll igual que la primera uff.

Llegemos hasta una tercera api.

00429409	85ED	TEST EBP,EBP	
0042940B	^ 0F84 2B010000	JE 0042953C	UnPackMe.0042953C
00429411	8B3D A8094600	MOV EDI,DWORD PTR DS:[4609A8]	UnPackMe.00480D45
00429417	6A 04	PUSH 4	
00429419	68 00200000	PUSH 2000	
0042941E	68 00004000	PUSH 400000	
00429423	6A 00	PUSH 0	
00429425	FFD7	CALL NEAR EDI	
00429427	8BF0	MOV ESI,EAX	
00429429	85F6	TEST ESI,ESI	

Llegamos a esta parte veamos en el desempacado de upx la misma parte.

0042940B	^ 0F84 2B010000	JE 0042953C	UnPackMe.0042953C
00429411	8B3D A8094600	MOV EDI,DWORD PTR DS:[4609A8]	kernel32.VirtualAlloc
00429417	6A 04	PUSH 4	
00429419	68 00200000	PUSH 2000	
0042941E	68 00004000	PUSH 400000	
00429423	6A 00	PUSH 0	
00429425	FFD7	CALL NEAR EDI	
00429427	8BF0	MOV ESI,EAX	
00429429	85F6	TEST ESI,ESI	
0042942B	^ 0F84 F4000000	JE 00429525	UnPackMe.00429525
00429431	6A 04	PUSH 4	

Y si es un call a VirtualAlloc, pongamosle un BP en el retorno, y demos RUN.



0047E423	C600 C3	MOV BYTE PTR DS:[EAX],0C3
0047E426	E9 34290000	JMP 00480D5F
0047E42B	8B0424	MOV EAX,DWORD PTR SS:[ESP]
0047E42E	52	PUSH EDX
0047E42F	51	PUSH ECX
0047E430	E9 70360100	JMP 00491A95
0047E435	E8 87670000	CALL 00484BC1
0047E43A	B8 0CC084C6	MOV EAX,C684C00C
0047E43F	53	PUSH EBX
0047E440	68 F4A32B27	PUSH 272BA3F4
0047E445	E9 C62D0000	JMP 00481210
0047E44A	5A	POP EDX
0047E44B	8B0424	MOV EAX,DWORD PTR SS:[ESP]
0047E44E	50	PUSH EAX
0047E44F	8BC2	MOV EAX,EDX
0047E451	870424	XCHG DWORD PTR SS:[ESP],EAX
0047E454	E8 016F0000	CALL 0047535A
0047E459	B8 50A8F865	MOV EAX,65F8A850
0047E45E	53	PUSH EBX

DS:[00480D5F]=5A ('Z')

Alli guarda otro RET veamos como es esa parte antes de modificarla

00480D40	- FF25 A8094600	JMP NEAR DWORD PTR DS:[4609A8]	kernel32.VirtualAlloc
00480D50	E9 030E0000	JMP 00481B58	UnPackMe.00481B58
00480D55	E9 608FFFFF	JMP 00479CBA	UnPackMe.00479CBA
00480D5A	E9 CC73FFFF	JMP 0047812B	UnPackMe.0047812B
00480D5F	5A	POP EDX	
00480D60	0F82 C5D6FFFF	JB 0047E42B	UnPackMe.0047E42B

es un RET que coloca un poco mas abajo de uno de los JMP directo a la api reparada.

00480D40	E9 3A14FFFF	JMP 0047217F	UnPackMe.0047217F
00480D45	E8 15000000	CALL 00480D5F	UnPackMe.00480D5F
00480D4A	- FF25 A8094600	JMP NEAR DWORD PTR DS:[4609A8]	kernel32.VirtualAlloc
00480D50	E9 030E0000	JMP 00481B58	UnPackMe.00481B58
00480D55	E9 608FFFFF	JMP 00479CBA	UnPackMe.00479CBA
00480D5A	E9 CC73FFFF	JMP 0047812B	UnPackMe.0047812B
00480D5F	C3	RETN	
00480D60	0F82 C5D6FFFF	JB 0047E42B	UnPackMe.0047E42B

O sea que hizo lo mismo que con la api anterior cerrar la entrada a la rutina ya que antes de colocar el RET pasa por alli mismo, y luego de cambiar la IAT coloca un RET para que no ejecute nuevamente la misma rutina, es como cerrar el porton de entrada, ya que esa entrada queda clausurada.

Si repasamos lo que hace realmente traceando verificamos lo que dijimos:

00429425	FFD7	CALL NEAR EDI
00429427	8BF0	MOV ESI,EAX
00429429	85F6	TEST ESI,ESI

pasa dos veces por alli, la primera apenas entra y pasa por el POP EDX y luego va a la rutina y despues de que arregla la IAT, coloca el RET y cancela la entrada por aqui.

00480D40	E9 3A14FFFF	JMP 0047217F	UnPackMe.0047217F
00480D45	E8 15000000	CALL 00480D5F	UnPackMe.00480D5F
00480D4A	- FF25 A8094600	JMP NEAR DWORD PTR DS:[4609A8]	kernel32.VirtualAlloc
00480D50	E9 030E0000	JMP 00481B58	UnPackMe.00481B58
00480D55	E9 608FFFFF	JMP 00479CBA	UnPackMe.00479CBA
00480D5A	E9 CC73FFFF	JMP 0047812B	UnPackMe.0047812B
00480D5F	C3	RETN	
00480D60	0F82 C5D6FFFF	JB 0047E42B	UnPackMe.0047E42B

O sea que cuando la api ya esta reparada toda esta parte se autoanula, ya que el programa accede a la api directo de la IAT y si hay mas accesos, pues no ira a la rutina pues encontrara el RET en la entrada y terminara en el JMP a la api igual.

Lo que se ve es que halla el nombre de la dll la primera vez que la usa, halla la base posiblemente con LoadLibraryA y luego salta a hallar la api y luego la guarda en la IAT y hace alguna

modificacion en la seccion para cerrar es arutina ya usada y no necesaria.

Podriamos intentar un script, llegar a ExitProcess con el programa y antes que se correr el script que vaya saltando a los diferentes call a las apis, y los vaya arreglando sera una tarea dificil por lo cambiante del codigo, pero bueno sera el desafio para la siguiente parte de la introduccion, de cualquier manera el tema de la IAT hay que resolverlo para todos los excryptor que miremos, y una vez que lo hagamos en este primero, seran las IATs de lso restantes, todas iguales y sera ese tema ya superado (ese tema, otros veremos jejeje)

Hasta la parte siguiente donde intentaremos el script para reparar la IAT de nuestro excryptor.

03/10/06

Ricardo Narvaja

## **INTRODUCCION AL CRACKING CON OLLYDBG PARTE 55**

Pues en esta parte seguiremos reparando el excryptor de la parte anterior, debemos hacer el script para reparar la IAT , y se que muchos cuando escuchan la palabra script se le vuelan los pelos (si tienen jeje), pero hacer script es realmente muy sencillo, el tema es que cuando uno ve un script ya hecho y terminado parece redifcil, pero realmente no se hace de una asi, se comienza con una idea simple a la cual se le van agregando partes, por ejemplo para el excryptor que debemos arreglar la tabla, si estamos parados en el OEP, deberia establecer primero una idea generica de lo que hara el script y hacer una bse esquematica.

Aclaro que ni idea de como terminara el script o si funcionara, lo estoy haciendo a medida que escribo para que ustedes vean como se va pensando y como se va armando, si no sale bien, pues ya habra tiempo de arreglarlo o corregirlo.

La idea seria que el script vaya recorriendo la tabla y por cada valor de la misma, si ve que esta redireccionado, trate de arreglarlo esto que parece resimple se podria traducir en una primera estructura basica.

Si debo recorrer la tabla debe haber una variable que vaya tomando los valores de la misma, la

podemos llamar para ser originales “tabla” jeje.

**var tabla**

luego debo ponerle el valor inicial que es el inicio de tabla

**mov tabla,460818**

luego que tenemos ya la variable inicializada vamos a hacer un loop que recorra toda la iat, verifique si la entrada esta redireccionada y si es asi, vaya a repararla, y si no que saltee y vaya a buscar la siguiente entrada

**start:**

**cmp [tabla],50000000**

**ja saltar**

**log tabla**

**end**

**saltar:**

**add tabla,4**

**jmp start**



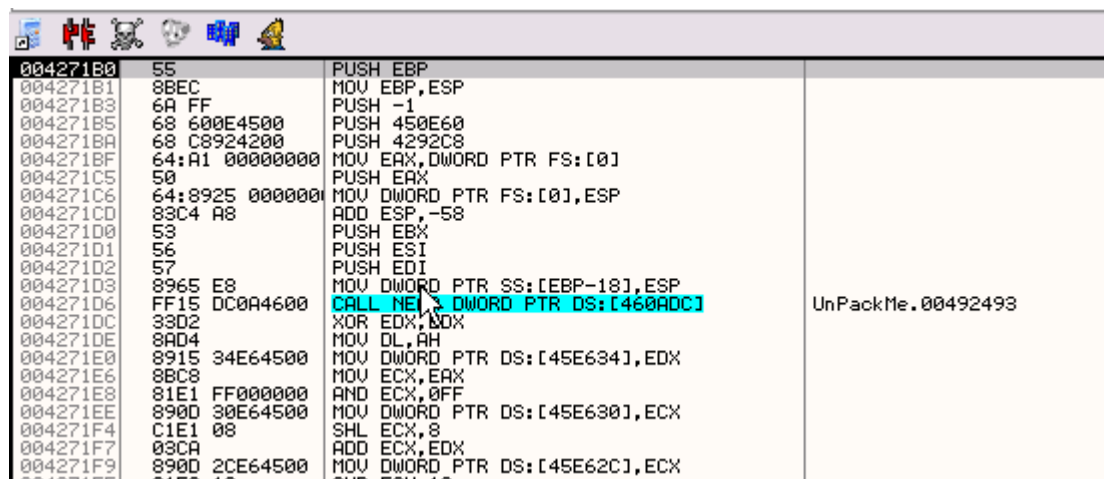
pues este es el esquema basico, verifica si la entrada es mayor a 50000000 porque a partir de esa direccion de memoria estan ubicadas las dlls en este caso, por lo tanto si una entrada tiene un valor superior, va directamente a una api y no esta redireccionada y mejor no tocarla y buscar la siguiente, es lo que hace, si no esta redireccionada va a saltar, incrementa la variable tabla para que busque la siguiente entrada de la IAT y se repita el proceso para reparar la 2da.

Como ven el esquema basico es muy sencillo, ustedes diran cuando esta redireccionada la api y no salta va aquí

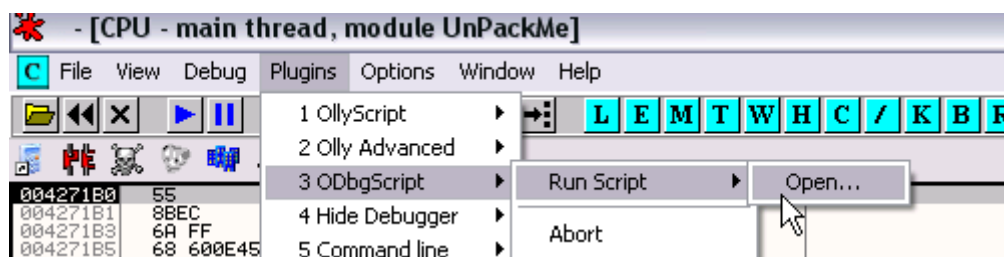
**log tabla**

**end**

y si alli debo escribir lo que hara realmente cuando halle una entrada para repararla, pero como antes de eso, me debo asegurar que encuentre bien las entradas de las apis redireccionadas, pues por ahora solo pruebo a ver si encuentra una api redireccionada, la loguea y termina, sin repetir el loop, por experiencia siempre es mejor ver si funciona de a poco antes de agregar mas cosas asi que ahora antes que nada debemos probar si funciona esto y me loguea la primera entrada de la tabla a reparar.



Pues alli estamos en el OEP, y ejecutemos el script a ver si funciona.



Bueno siempre alguna pavada se pasa, jeje el comando para que termine no es end si no ret, jeje a arreglar eso.

---

```
var tabla
```

```
mov tabla,460818
```

```
start:
```

```
cmp [tabla],50000000
ja saltar
```

```
log tabla
```

```
ret
```

```
saltear:
```

```
add tabla,4
```

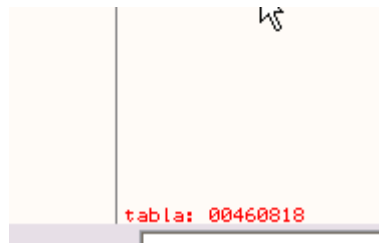
```
jmp start
```

---

Corregimos la pavada que hicimos y probamos de nuevo.



Bueno al menos no dio error, veamo si logueo el primer valor.



Veamos si es una entrada redireccionada, para ello podemos asignar otra variable que tenga el contenido de la entrada para poder loguear el valor, así que la llamare contenido jeje

---

```
var tabla
```

```
var contenido
```

```
mov tabla,460818
```

```
start:
```

```
cmp [tabla],50000000
```

ja saltar

log tabla

**mov contenido,[tabla]**

**log contenido**

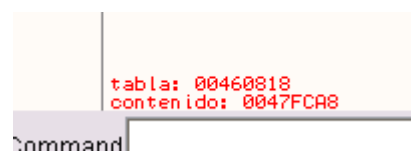
ret

saltar:

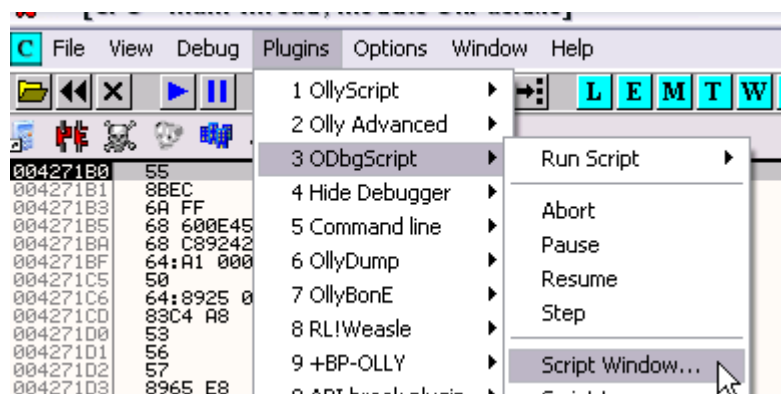
add tabla,4

jmp start

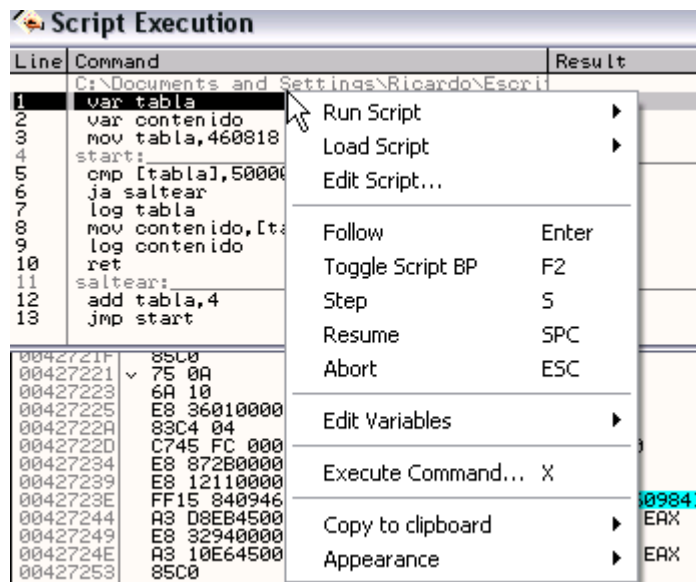
ahí está, le agregamos la variable contenido, que como su nombre lo indica, tiene el contenido la la entrada a la que apunta tabla, así que ahora si lo corro me debería loguear ambos, el valor de la entrada de la IAT y su contenido.



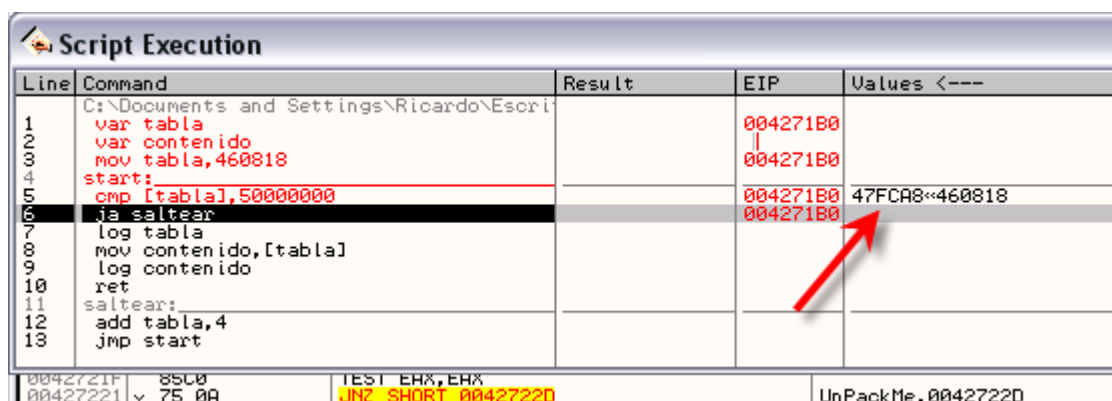
Allí está me logueo la entrada de la IAT a reparar y su contenido que es 47fcab, o sea que estamos hasta aquí correctos, el siguiente paso es mandar a loopear para que loguee todas las entradas, quitar el ret y probar para ver como va, pero este ODbgScript permite tracear el script y ver que hace, así que si uno en algún punto tiene alguna duda de como trabaja va a ...



Con eso abrimos el traceador de script



como vemos tenemos comandos para tracear con la letra S, para poner BPs en alguna linea con f2, y con eso nos alcanza para tracearlo y revisarlo si tuviera algun error, probemos tracearlo apretando S a ver que pasa.



Vemos que en una columna nos muestra el valor de EIP y en valores, si hay alguna operación nos muestra su contenido, en este caso el contenido, que es 47fcab, asi podriamos segur traceando o poner bps y ejecutarlo para chequear como va, por ahora como todo nos va bien, cerraremos esta ventana y continuaremos agregandole cosas al script.

Ahora debemos agregarle un opcion para que cuando termine de leer toda la tabla, salga y termine el script, ya que el final de tabla es 460f28

```
var tabla
var contenido
```

```
mov tabla,460818
```

```
start:
```

```
cmp tabla,460f28
```

```
ja final
```

```
cmp [tabla],50000000
```

```
ja saltar
```

```
log tabla
mov contenido,[tabla]
log contenido
ret
```

```
saltear:
add tabla,4
jmp start
```

```
final:
ret
```

---

asi que alli le agregue el verdadero final del script cuando tabla tome el valor del final de la misma o mayor, pues terminara, saltara a la etiqueta final y ira al verdadero fin del script..

Asi que ahora para probar si loguea todas las entradas redireccionadas a reparar y su contenido, y termina bien, debemos quitar el ret que paraba el script luego de encontrar una sola entrada.

```
var tabla
var contenido

mov tabla,460818

start:
cmp tabla,460f28
ja final
cmp [tabla],50000000
ja saltear

log tabla
mov contenido,[tabla]
log contenido

saltear:
add tabla,4
jmp start

final:
ret
```

---

alli vemos que le quitamos el ret que estaba debajo de los LOG, de esta forma luego de loguear, ira a saltear donde incrementara tabla en 4, y repetira el proceso hasta que tabla termine toda la IAT, probemoslo a ver que pasa.





Veamos si logueo todo bien



Vemos que loguea bien salvo que debemos agregar un chequeo para cuando el contenido sea cero, o sea las entradas de separacion de tabla, en esas no hay que trabajar, asi que le agregamos.

```
var tabla  
var contenido
```

```
mov tabla,460818
```

```
start:
```

```
cmp tabla,460f28
```

```
ja final
```

```
cmp [tabla],50000000
```

```
ja saltar
```

```
mov contenido,[tabla]
```

```
cmp contenido,0
```

```
je saltar
```

```
log contenido
```

```
log tabla
```

```
saltar:
```

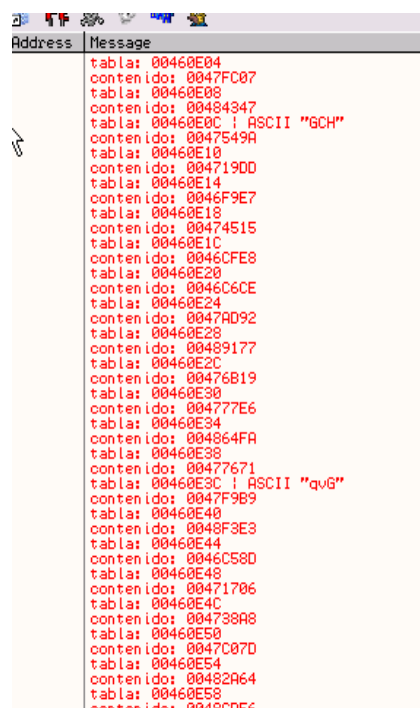
```
add tabla,4
```

```
jmp start
```

```
final:
```

```
ret
```

vemos que vamos agregando de a poco, nadie hace un script como se ven en los tutes y lo escribe de una, va poco a poco verificando y agregando cosas, como vemos ahora si el contenido es cero no logueara, saltara ambos logs que ahora ubique despues, probemos.



Address	Message
00460E04	tabla: 00460E04
0047FC07	contenido: 0047FC07
00460E08	tabla: 00460E08
00484347	contenido: 00484347
00460E0C	tabla: 00460E0C ; ASCII "GCH"
0047549A	contenido: 0047549A
00460E10	tabla: 00460E10
004719D0	contenido: 004719D0
00460E14	tabla: 00460E14
0046F9E7	contenido: 0046F9E7
00460E18	tabla: 00460E18
00474515	contenido: 00474515
00460E1C	tabla: 00460E1C
0046CFE8	contenido: 0046CFE8
00460E20	tabla: 00460E20
0046C6CE	contenido: 0046C6CE
00460E24	tabla: 00460E24
0047AD92	contenido: 0047AD92
00460E28	tabla: 00460E28
00489177	contenido: 00489177
00460E2C	tabla: 00460E2C
00476B19	contenido: 00476B19
00460E30	tabla: 00460E30
004777E6	contenido: 004777E6
00460E34	tabla: 00460E34
004864FA	contenido: 004864FA
00460E38	tabla: 00460E38
00477671	contenido: 00477671
00460E3C	tabla: 00460E3C ; ASCII "qvG"
0047F9B9	contenido: 0047F9B9
00460E40	tabla: 00460E40
0048F3E3	contenido: 0048F3E3
00460E44	tabla: 00460E44
0046C58D	contenido: 0046C58D
00460E48	tabla: 00460E48
00471706	contenido: 00471706
00460E4C	tabla: 00460E4C
004738A8	contenido: 004738A8
00460E50	tabla: 00460E50
0047C07D	contenido: 0047C07D
00460E54	tabla: 00460E54
00482A64	contenido: 00482A64
00460E58	tabla: 00460E58
0048C9FA	contenido: 0048C9FA

Vemos que ahora si loguea todas las entradas a reparar con su contenido, eso quiere decir que a

continuacion de los logs puedo escribir la parte en que arregla cada entrada (o el intento jeje)

Bueno ahora pensemos como podria trabajar esto, si en ese punto, cambio el eip y le pongo el valor del contenido, podria funcionar, pero deberia tener alguna forma de parar antes de que salte a la api, pues como no estamos colocando los parametros correctos daria error en algunas.

Por lo tanto como antes de saltar a la api guarda en valor en la entrada, podemos poner un BPM ON WRITE cosa de que cuando pare, retome el control el script y no lo deje saltar a la api y vaya a buscar la siguiente entrada.

BPWM addr, size

-----

Set memory breakpoint on write. Size is size of memory in bytes.

Example:

bpwm 401000, FF

esa es la instrucción que pone un BPM ON WRITE

**mov eip,contenido**

**bpwm tabla, 4**

podrian ser las instrucciones para comenzar a reparar una entrada redireccionada, mover el eip a la rutina redireccionada y poner un BPM ON WRITE en la entrada que arreglara.

Luego la instrucción COB que suplanta a la vieja EOB, salta a una rutina cuando encuentra un BP

COB

---

Makes script continue execution after a breakpoint has occurred (removes EOB)

Example:

COB

**mov eip,contenido**

**bpwm tabla, 4**

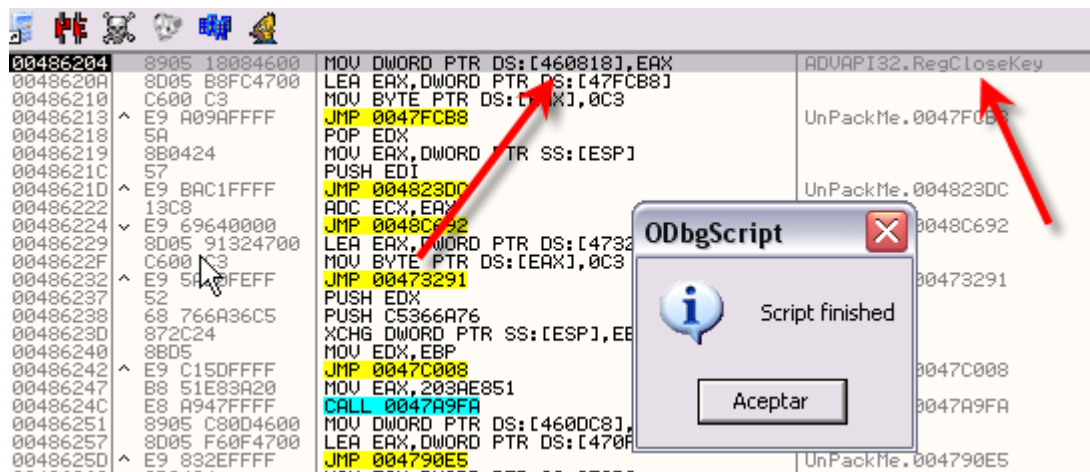
**cob reparar**

**run**

**reparar:**

**ret**

de esta forma luego de poner el BPM ON WRITE ponemos el programa a correr y cuando pare por una excepcion o breakpoint ira a reparar, veamos al igual que la primera vez pongamos un RET aquí en reparar para ver si funciona con la primera entrada.(tambien podriamos usar el traceandor y poner un bp alli es lo mismo)



Vemos que realmente puede funcionar, para justo cuando va a guardar el valor de la api, en la entrada que queremos arreglar, asi que para que la guarde debemos ejecutar la linea esa con

STI

---

Execute F7 in OllyDbg.

Example:

sti

con eso ejecutara solo una instrucción equivale a apretar f7.

mov eip,contenido  
bpwm tabla, 4  
cob reparar  
run

reparar:

**sti**

ret

vemos de esta forma que si lo ejecuto nuevamente, reparara la primera entrada y terminara.



427100	83C4 08	000 ESP -58		
<b>Script Execution</b>				
Line	Command	Result	EIP	Values <---
10	cmp contenido,0			
11	je saltar			
12	log contenido			
13	log tabla			
14	mov eip,contenido			
15	bpwm tabla, 4			
16	cob reparar			
17	run			
18	reparar:			
19	sti			
20	saltear:			
21	add tabla,4			
22	jmp start			
23	final:			
24	ret			
427212	E8 49010000	CALL 00427360		UnPackMe.00427360

Ponemos un bp en la linea esa con f2, luego de que el programa guarda la primera entrada, para tracear a partir de alli a ver que pasa, ejecuto el script.

<b>Script Execution</b>		
Line	Command	Result
10	cmp contenido,0	
11	je saltar	
12	log contenido	
13	log tabla	
14	mov eip,contenido	
15	bpwm tabla, 4	
16	cob reparar	
17	run	
18	reparar:	
19	sti	
20	saltear:	
21	add tabla,4	
22	jmp start	
23	final:	
24	ret	
427212	E8 49010000	CALL 00427360
427217	83C4 04	
42721A	F8 D12FA000	

Run Script	▶
Load Script	▶
Edit Script...	
Follow	Enter
Toggle Script BP	F2
Step	S
Resume	SPC
Abort	ESC

Al hacer resume se ejecuta y para en el sti

Line	Command	Result	EIP	Values <---
1	C:\Documents and Settings\Ricardo\Escri...		004271B1	
2	var tabla		004271B1	
3	var contenido			
4	mov tabla,460818			
5	start:			
6	cmp tabla,460f28		004271B1	460818
7	je final			
8	cmp [tabla],50000000			47FCA8<<460818
9	je saltar			47FCA8<<460818
10	mov contenido,[tabla]			47FCA8
11	cmp contenido,0			47FCA8
12	je saltar			460818
13	log contenido			460818
14	log tabla			460818
15	mov eip,contenido			460818
16	bpwm tabla, 4			
17	cob reparar			
18	run		004271B1	
19	reparar:			
20	sti		00486204	
21	saltear:			
22	add tabla,4			
23	jmp start			
24	final:			
25	ret			

en el olly puedo mirar a pesar de que aquí me dice el eip, si quiero puedo ver donde esta parado.

00486204	8905 18084600	MOV DWORD PTR DS:[460818],EAX	ADVAPI32.RegCloseKey
0048620A	8D05 B8FC4700	LEA EAX,DWORD PTR DS:[47FCB8]	
00486210	C600 C3	MOV BYTE PTR DS:[EAX],0C3	
00486213	E9 A09AFFFF	JMP 0047FCB8	UnPackMe.0047FCB8
00486218	5A	POP EDX	
00486219	8B0424	MOV EAX,DWORD PTR SS:[ESP]	
0048621C	57	PUSH EDI	
0048621D	E9 BAC1FFFF	JMP 004823DC	
00486222	13C8	ADC ECX,EAX	
00486224	E9 69640000	JMP 0048C692	
00486229	8D05 91324700	LEA EAX,DWORD PTR DS:[473291]	
0048622F	C600 C3	MOV BYTE PTR DS:[EAX],0C3	
00486232	E9 5AD0FEFF	JMP 00473291	
00486237	52	PUSH EDX	
00486238	68 766A96C5	PUSH C5366A76	
0048623D	872C24	XCHG DWORD PTR SS:[ESP],EBP	
00486240	8B05	MOV EDX,EBP	
00486242	E9 C15DFFFF	JMP 0047C008	
00486247	B8 51E83A20	MOV EAX,203AE851	
0048624C	E8 A947FFFF	CALL 0047A9FA	
00486251	8905 C80D4600	MOV DWORD PTR DS:[460DC8],EAX	
00486256	8D05 F60F4700	LEA EAX,DWORD PTR DS:[470FF6]	
0048625B	E9 832EFFFF	JMP 004790E5	
00486262	8B0424	MOV EAX,DWORD PTR SS:[ESP]	

Justo donde pensamos cuando va a guardar la primera entrada, así que apretamos la S para que ejecute la siguiente instrucción del script.

00486204	8905 18084600	MOV DWORD PTR DS:[460818],EAX	
0048620A	8D05 B8FC4700	LEA EAX,DWORD PTR DS:[47FCB8]	
00486210	C600 C3	MOV BYTE PTR DS:[EAX],0C3	
00486213	E9 A09AFFFF	JMP 0047FCB8	
00486218	5A	POP EDX	
00486219	8B0424	MOV EAX,DWORD PTR SS:[ESP]	
0048621C	57	PUSH EDI	
0048621D	E9 BAC1FFFF	JMP 004823DC	
00486222	13C8	ADC ECX,EAX	
00486224	E9 69640000	JMP 0048C692	
00486229	8D05 91324700	LEA EAX,DWORD PTR DS:[473291]	
0048622F	C600 C3	MOV BYTE PTR DS:[EAX],0C3	
00486232	E9 5AD0FEFF	JMP 00473291	
00486237	52	PUSH EDX	
00486238	68 766A96C5	PUSH C5366A76	
0048623D	872C24	XCHG DWORD PTR SS:[ESP],EBP	
00486240	8B05	MOV EDX,EBP	
00486242	E9 C15DFFFF	JMP 0047C008	
00486247	B8 51E83A20	MOV EAX,203AE851	
0048624C	E8 A947FFFF	CALL 0047A9FA	
00486251	8905 C80D4600	MOV DWORD PTR DS:[460DC8],EAX	
00486256	8D05 F60F4700	LEA EAX,DWORD PTR DS:[470FF6]	
0048625B	E9 832EFFFF	JMP 004790E5	
00486262	8B0424	MOV EAX,DWORD PTR SS:[ESP]	
DS:[00460818]=77DA6BF0 (ADVAPI32.RegCloseKey)			
Address	Hex dump	ASCII	
00460818	F0 6B DA 77 41 6F 48 00 98 00 48 00 F9 A4 47 00	-k rWAoH.ü.H.-RG.	
00460828	81 09 48 00 00 00 00 00 CF 65 C3 58 08 03 C4 58	ü.H.....e}Xi-X	

Hasta aquí vamos bien, el problema se da cuando va a arreglar la segunda entrada, sigamos apretando S a ver que pasa.

Line	Command	Result	EIP	Values <---
1	var tabla		004271B1	
2	var contenido		004271B1	
3	mov tabla,460818			
4	start:			
5	cmp tabla,460f28		00486204	46081C,460818
6	ja final		00486204	
7	cmp [tabla],50000000		004271B1	47FCA8<<460818
8	ja saltear			47FCA8<<460818
9	mov contenido,[tabla]			47FCA8
10	cmp contenido,0			47FCA8
11	je saltear			460818
12	log contenido			47FCA8
13	log tabla			460818
14	mov eip,contenido			47FCA8
15	bpmv tabla, 4			460818
16	cob reparar			
17	run		004271B1	
18	reparar:			
19	sti		00486204	
20	saltear:			
21	add tabla,4		00486204	460818
22	jmp start		00486204	
23	final:			
24	ret			

Vemos que tabla se actualizo y ahora apunta a la siguiente entrada sigamos traceando.

00479B7A=UnPackMe.00479B7A			
Address	Hex dump	ASCII	
00460818	F0 6B DA 77 41 6F 48 00 98 00 48 00 F9 A4 47 00	-k rWAoH.ü.H.-RG.	
00460828	81 09 48 00 00 00 00 00 CF 65 C3 58 08 03 C4 58	ü.H.....e}Xi-X	
00460838	00 00 00 00 04 6A EF 77 66 95 EF 77 89 6A EF 77	....ej'wfö'wej'w	
00460848	F3 AD EF 77 ED 09 EF 77 99 8B EF 77 C0 B5 EF 77	%i'wÿ'w0i'w'A'w	
00460858	2A 7D EF 77 B2 7C EF 77 77 53 F2 77 1E C9 F1 77	*)'wÿ!'wWS=wAf'w	
00460868	0C BC EF 77 52 D4 EF 77 FA 8D EF 77 F1 DD EF 77	.'wRE'w.i'w;i'w	

Vemos que esta trabajando con la segunda entrada y que contenido tomo el valor correspondiente.

Line	Command	Result	EIP	Values <---
1	C:\Documents and Settings\Ricardo\Escri		004271B1	
2	var tabla		004271B1	
3	var contenido		004271B1	
4	mov tabla,460818		004271B1	
5	start:		00486204	46081C,460818
6	cmp tabla,460F28		00486204	486F41<<46081C,47FCA8<<460818
7	ja final		00486204	486F41<<46081C,47FCA8<<460818
8	cmp [tabla],50000000		00486204	486F41<<46081C,47FCA8<<460818
9	ja saltar		00486204	486F41<<46081C,47FCA8<<460818
10	mov contenido,[tabla]		00486204	486F41<<46081C,47FCA8<<460818
11	cmp contenido,0		00486204	486F41<<46081C,47FCA8<<460818
12	je saltar		00486204	486F41<<46081C,47FCA8<<460818
13	log contenido		00486204	486F41<<46081C,47FCA8<<460818
14	log tabla		00486204	486F41<<46081C,47FCA8<<460818
15	mov eip,contenido		00486204	486F41<<46081C,47FCA8<<460818
16	bpmw tabla, 4		00486204	460818
17	cob reparar		004271B1	
18	run		004271B1	
19	reparar:		00486204	
20	sti		00486204	
21	saltear:		00486204	
22	add tabla,4		00486204	460818
23	jmp start		00486204	
24	final:			
25	ret			

ahora tiene que poner el BPM ON WRITE en la segunda entrada si seguimos traceando vemos que al apretar RUN da error asi que la maldita secuencia que vimos que hace mil vueltas, se ve que se corrompe al hacerlo de esta forma grr,

veamos donde provoco el error

Line	Command	Result	EIP	Values <---
1	C:\Documents and Settings\Ricardo\Escri		004271B0	
2	var tabla		004271B0	
3	var contenido		004271B0	
4	mov tabla,460818		004271B0	
5	start:		0048620A	46081C,460818
6	cmp tabla,460F28		0048620A	486F41<<46081C,47FCA8<<460818
7	ja final		0048620A	486F41<<46081C,47FCA8<<460818
8	cmp [tabla],50000000		0048620A	486F41<<46081C,47FCA8<<460818
9	ja saltar		0048620A	486F41<<46081C,47FCA8<<460818
10	mov contenido,[tabla]		0048620A	486F41<<46081C,47FCA8<<460818
11	cmp contenido,0		0048620A	486F41<<46081C,47FCA8<<460818
12	je saltar		0048620A	486F41<<46081C,47FCA8<<460818
13	log contenido		0048620A	486F41<<46081C,47FCA8<<460818
14	log tabla		0048620A	486F41<<46081C,47FCA8<<460818
15	mov eip,contenido		0048620A	486F41<<46081C,47FCA8<<460818
16	bpmw tabla, 4		00486F41	46081C,460818
17	cob reparar		00486F41	
18	run		00486F41	
19	reparar:		00486204	
20	sti		00486204	
21	saltear:		0048620A	
22	add tabla,4		0048620A	460818
23	jmp start		0048620A	
24	final:			
25	ret			

Cuando el script va a correr puedo cerrarlo y seguir traceando con ollydbg a ver el error, elijo ABORT para que se termine el script y sigo traceando en olly



00486F41	E8 342CFFFF	CALL 00479B7A	UnPackMe.00479B7A
00486F46	^ E9 59FAFEFF	JMP 004769A4	UnPackMe.004769A4
00486F4B	8105 77D1DE21	ADC EBP,210ED177	
00486F51	^ E9 A0250000	JMP 004894F6	UnPackMe.004894F6
00486F56	13F0	ADC ESI,EAX	
00486F58	5F	POP EDI	
00486F59	23D7	AND EDX,EDI	
00486F5B	03DD	ADD EBX,EBP	
00486F5D	81C2 94EF1087	ADD EDX,871DEF94	
00486F63	81E2 22185F2A	AND EDX,2A5F1822	
00486F69	81C2 23F8FB07	ADD EDX,D7FBF823	
00486F6F	^ E9 F9B0FEFF	JMP 0047206D	UnPackMe.0047206D
00486F74	C3	RETN	
00486F75	^ E9 B2B5FFFF	JMP 0048252C	UnPackMe.0048252C
00486F7A	E8 65FCFEFF	CALL 00476BE4	UnPackMe.00476BE4
00486F7F	^ E9 3248FFFF	JMP 0047B7B6	UnPackMe.0047B7B6
00486F84	EE	POP EBP	

alli estamos en el inicio de la rutina de la redireccion de la segunda entrada, traceemos

00479B7A	870424	XCHG DWORD PTR SS:[ESP],EAX	ADVAPI32.RegCloseKey
00479B7D	58	POP EAX	
00479B7E	^ 0F83 B2970000	JNB 00483336	UnPackMe.00483336
00479B84	8B05 94004800	MOV EAX,DWORD PTR DS:[480094]	
00479B8A	^ E9 D0BEFFFF	JMP 00475A6C	UnPackMe.00475A6C

alli intercambia el contenido de esp con eax, hmmm

00483336	8B05 94004800	MOV EAX,DWORD PTR DS:[480094]	
0048333C	E8 955E0000	CALL 004891D6	
00483341	^ E9 5B78FFFF	JMP 0047ABA1	
00483346	C3	RETN	

alli pasa a eax el valor cero que se halla en esa posicion de memoria

vemos que tracear terrible rutina es un trabajo de chinos, solo con trace into si la pongo a tracear tarda muchisimo tiempo creo que esto se esta poniendo de castaño a oscuro.

Tambien realizando una prueba en la segunda entrada que tira error, sin el script, voy a la entrada, busco una referencia a la misma un CALL que toma valores de la IAT de dicha entrada

0043FA2E	FF15 1C084600	CALL NEAR DWORD PTR DS:[46081C]	UnPackMe.00486F41
0043FA34	85C0	TEST EAX,EAX	
0043FA36	^ 75 6C	JNZ SHORT 0043FAA4	UnPackMe.0043FAA4
0043FA38	8D45 FC	LEA EAX,DWORD PTR SS:[EBP-4]	

Cambio el eip con new origin here en dicho CALL, le pongo el BPM ON WRITE a MANO en la entrada de la IAT y un BP al retornar del call, y veo que esta segunda entrada devuelve el valor correcto pero no se modifica a si misma, por eso me estaba dando error, quiere decir que no cumple la idea que nosotros tuvimos al solucionar la primera entrada, que es que pare en el BPM ON WRITE al guardar el valor correcto de la api, aquí en esta segunda entrada la cosa es diferente y habra que buscarle otra forma, ya veremos.

Bueno buscaremos otra solucion, no sera tan elegante pero bueno veamos, pongamos a tracear la primera entrada desde donde se inicia, poniendo el BPM ON WRITE en la misma que sabemos que en la primera entrada funciona, asi que el traceo es larguisimo, pero luego de como 5 minutos para aqui.

```

00483C91 Main  MOV AL,1                                ; EAX=77DA6C01
00483C93 Main  JMP 00483C78
00483C78 Main  MOV BYTE PTR SS:[EBP-5],AL
00483C7B Main  MOV AL,BYTE PTR SS:[EBP-5]
00483C7E Main  POP ECX                      ; ECX=01000001, ESP=0012FE6C
00483C7F Main  POP ECX                      ; ECX=77DA6C75, ESP=0012FE70
00483C80 Main  POP EBP                      ; ESP=0012FE74, EBP=0012FE80
00483C81 Main  RETN                        ; ESP=0012FE78
004833D5 Main  TEST AL,AL                    ; FL=0
004833D7 Main  JNZ 0047CC50
0047CC50 Main  POP ECX                      ; ECX=00000001, ESP=0012FE7C
0047CC51 Main  POP ECX                      ; ECX=77DA6C75, ESP=0012FE80
0047CC52 Main  POP EBP                      ; ESP=0012FE84, EBP=0012FFB0
0047CC53 Main  RETN                        ; ESP=0012FE88
0047691C Main  MOV EAX,DWORD PTR SS:[EBP-C]      ; EAX=77DA6BF0
0047691F Main  MOV ESP,EBP                  ; ESP=0012FFB0
00476921 Main  JMP 004765DC
004765DC Main  JMP 0047F15C
0047F15C Main  PUSH 47C9B5                  ; ESP=0012FFAC
0047F161 Main  JMP 00491A5F
00491A5F Main  JMP 004737D4
004737D4 Main  RETN                        ; ESP=0012FFB0
0047C9B5 Main  POP EBP                      ; ESP=0012FFB4, EBP=0012FFF0
0047C9B6 Main  RETN                        ; ESP=0012FFB8
0046E81D Main  RETN                        ; ESP=0012FFBC

```

Memory breakpoint when writing to [00460818]

estas son las ultimas lineas que ejecuta antes del memory breakpoint, asi que trataremos de buscar un punto donde pasen todas la entradas o la mayoria, porque si no es una lacra.

```

0047691C Main  MOV EAX,DWORD PTR SS:[EBP-C]      ; EAX=77DA6BF0

```

aquí realmente pasa a EAX el valor definitivo de la api, podemos poner un HE alli,para ver si para en la segunda entrada tambien.

Al igual que en la primera entrada para alli mostrando en EAX la api correcta, traceemos



vemos que si seguimos traceando y comparando con el traceo de la primera entrada aquí esta la diferencia

```

0046E81D C3      RETN

```

en la primera entrada salta a guardar la api correcta en la IAT, mientras que en la segunda entrada salta aquí

004868FF	50	PUSH EAX	ADVAPI32.RegOpenKeyExA
00486900	8BC2	MOV EAX, EDX	
00486902	✓ E9 371C0000	JMP 0048853E	UnPackMe.0048853E
00486907	C3	RETN	
00486908	^ F9 0261FFFF	JMP 0046C00F	UnPackMe.0046C00F

Bueno pero nosotros podemos tratar de aprovechar lo comun de ambas, esta instrucción esta buena pues aparentemente todas las entradas pasan por aqui.

0047691C	8B45 F4	MOV EAX, DWORD PTR SS:[EBP-C]	
0047691F	8BE5	MOV ESP, EBP	
00476921	^ E9 B6FCFFFF	JMP 004765DC	UnPack
00476926	8100 70F092B4	AND EAX, B492F070	

Aquí pasa a EAX el valor de la buena entrada, así que si modificamos el script para que cuando pare allí arregle la entrada de la IAT veremos que sale.

```
-----
var tabla
var contenido

mov tabla,460818

start:
cmp tabla,460f28
ja final
cmp [tabla],50000000
ja saltear

mov contenido,[tabla]
cmp contenido,0
je saltear
log contenido
log tabla

mov eip,contenido
bphws 47691f, "x"
cob reparar
run

reparar:
log eax
mov [tabla],eax

saltear:
add tabla,4
jmp start

final:
```

---

00407100 Single stop event at U-Park-M: 00407100

Vemos que empieza mejor va reparando unas cuantas y luego da error la ultima que intenta reparar es esta la de 460988

Hasta allí fuimos bien con nuestro script somos cabezaduras veremos que pasa en este caso reiniciemos jeje.

## Vayamos al inicio de la rutina

00480400	E8 0B000000	CALL 00480410	UnPackMe.00480410
00480405	^ FF25 88094600	JMP NEAR DWORD PTR DS:[460988]	UnPackMe.00480400
00480408	^ E9 8C61FFFF	JMP 0048359C	UnPackMe.0048359C
00480410	5A	POP EDX	
00480411	^ 0F89 E3030000	JNS 0048D7FA	UnPackMe.0048D7FA
00480417	^ E9 8566FFFF	JMP 00483AA1	UnPackMe.00483AA1
0048041C	81F5 4436DDE0	XOR EBP,E0D03644	
00480422	F7C5 0891CD72	TEST EBP,72CD9108	
00480428	^ E9 F5130000	JMP 0048E822	UnPackMe.0048E822
0048042D	870424	XCHG DWORD PTR SS:[ESP],EAX	
00480430	8BEC	MOV EBP,ESP	
00480432	52	PUSH EDX	
00480433	03D7	ADD EDX,EDI	
00480435	^ E9 262AFEFF	JMP 0046FE60	UnPackMe.0046FE60
0048043A	C1C8 0A	ROR EAX,0A	
0048043D	52	PUSH EDX	
0048043E	68 1F11C75B	PUSH 5BC7111F	
00480443	5A	POP EDX	
00480444	C1C2 0D	ROL EDX,0D	
00480447	01F2 1AF00044	ADD EDI,1AF00044	

veamos si para en el HE 47691f que utilizamos en el script ponemos el HE y damos run

00476917	E8 E2E60000	CALL 00484FFE	UnPackMe.00484FFE	Registers (FPU) EAX: 7C947A40 ntdll.RtlUnwind ECX: 7C947B62 ntdll.7C947B62 EDX: 7C947B62 ntdll.7C947B62 EBX: 7FFDF000
0047691C	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]		
0047691F	8BE5	MOV ESP,EBP		
00476921	^ E9 B6FCFFFF	JMP 0047650C	UnPackMe.0047650C	
00476925	81D0 70ED92B4	ADC EAX,B492ED7D		

pues si para y muestra en EAX la api, es posible que dada la complejidad de las rutinas al hacer todo de una sola vez se provoquen errores, igual podriamos hacerlo por partes, pero intentaremos hacerlo en una sola vez.

Lo que tengo que hacer ahora es empezar en la entrada que dio el error asi que en el script cambio el inicio de tabla a 460988

```
var tabla
var contenido
```

```
mov tabla,460988
```

```
start:
cmp tabla,460f28
ja final
cmp [tabla],50000000
ja saltar
```

```
mov contenido,[tabla]
cmp contenido,0
je saltar
log contenido
log tabla
```

```
mov eip,contenido
bphws 47691f, "x"
cob reparar
run
```

```
reparar:
log eax
mov [tabla],eax
```

```

saltear:
add tabla,4
jmp start

```

```

final:
ret

```

asi que podemos empezar desde aquí a ver que pasa.

```

76361000 Code size in header is 0002FE00, extending to size
004271B0 Single step event at UnPackMe.004271B0
74CC0000 Module C:\WINDOWS\system32\oledlg.dll
74CC1000 Code size in header is 00010400, extending to size
00401000 Sent virtual address to ollybone module for NX remap
          contenido: 0048D400
          tabla: 00460988
0047691F Hardware breakpoint 1 at UnPackMe.0047691F
          eax: 7C947A40 ! ntdll.RtlUnwind
          contenido: 00490116
          tabla: 0046098C
0047691F Hardware breakpoint 1 at UnPackMe.0047691F
          eax: 7C812A09 ! kernel32.RaiseException
          contenido: 004914B7
          tabla: 00460990
0047691F Hardware breakpoint 1 at UnPackMe.0047691F
          eax: 7C81CDDA ! kernel32.ExitProcess
          contenido: 0046F629
          tabla: 00460994
0047691F Hardware breakpoint 1 at UnPackMe.0047691F
          eax: 7C801E16 ! kernel32.TerminateProcess
          contenido: 0047E634
          tabla: 00460998
0047691F Hardware breakpoint 1 at UnPackMe.0047691F
          eax: 7C809915 ! kernel32.GetACP
          contenido: 0047C5F3
          tabla: 0046099C
0047691F Hardware breakpoint 1 at UnPackMe.0047691F
          eax: 7C810EF8 ! kernel32.HeapDestroy
          contenido: 00482308
          tabla: 004609A0
0047691F Hardware breakpoint 1 at UnPackMe.0047691F
          eax: 7C812BB6 ! kernel32.HeapCreate
          contenido: 00483086
          tabla: 004609A4
00481732 Access violation when reading [000000AD]
          Process terminated, exit code C0000005 (-107374181)

```

Jeje otras cinco o seis reparo y salto a error, jeje paciencia de santo se ve que la rutina es tan compleja que no da para arreglar mas de 4 o 5 por vez sin que se corrompa.

Esto esta reluchado vemos que luego de producir la excepcion termina el programa, a las patadas pero la voy a arreglar jeje.

El proximo paso es poner un HE ZwTerminateProcess para que cuando el programa salte a terminarse, el script lo recupere no lo deje terminar y continúe el script (truco sucisimo jeje)

```

reparar:
cmp eip,7c91e88e
je saltear
log eax
mov [tabla],eax
run

```

en mi maquina en 7c91e88e esta dicha api ZwTerminateProcess, asi que cuando el script cae alli de una excepcion, compara si el programa esta tratando de terminarse en dicha api y si es asi, saltea la entrada esa y va a la siguiente, al probar esto al menos la IAT se arregla bastante pero llega un punto que arregla una api si y una no, porque una produce una excepcion y luego de la excepcion la siguiente va bien pero saltea la misma entrada que se estaba arreglando, jeje que idea que se me

ocurrio jajajaja (si el es sucio yo sere mas jeje).

Si luego de arreglar una entrada provoco una excepcion y salto a la fuera a ZwTerminateProcess la siguiente entrada despues de la excepcion, aparentemente se arregla bien asi que puedo crear una excepcion luego de arreglar cada entrada, cosa de que provoque un error a proposito, y luego la siguiente entrada se arregle bien jeje

jeje de esta forma el script final quedaria asi

```
start:
cmp tabla,460f28
ja final
cmp [tabla],50000000
ja saltear
```

```
mov contenido,[tabla]
cmp contenido,0
je saltear
log contenido
log tabla
```

```
mov eip,contenido
bphws 47691f, "x"
mov [47691f],0
mov [476920],0
cob reparar
run
```

```
reparar:
cmp eip,7c91e88e
je saltear
log eax
mov [tabla],eax
run
```

```
saltear:
add tabla,4
jmp start
```

```
final:
ret
```

---

Hay que quitar la tilde en exceptions la de memory access, alli la parte crttica una vez que para en el hardware bpx pone a ceros los bytes siguientes para que se produzca una excepcion a proposito, de

esa forma al dar RUN, la excepcion la provoco yo y el programa se arregla y la entrada siguiente se arreglara bien, jeje que ideota.

Address	Hex dump	ASCII
004607EC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004607FC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0046080C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....-k rw
0046081C	1B 76 DA 77 F4 EA DA 77 E7 EB DA 77 83 78 DA 77	+Urw00rw00rwax rw
0046082C	00 00 00 00 CF 65 C3 58 D8 03 C4 58 00 00 00 00	...DeIXI-X...
0046083C	D4 6A EF 77 66 95 EF 77 89 6A EF 77 F3 AD EF 77	Ej'wfo'wej'wxi'w
0046084C	ED 09 EF 77 99 8B EF 77 C0 B5 EF 77 2A 7D EF 77	Y'w0i'w-A'w*)'w
0046085C	B2 7C EF 77 77 53 F2 77 1E C9 F1 77 0C BC EF 77	W!wWS=wAfzw.'w
0046086C	52 D4 EF 77 FA 8D EF 77 F1 D0 EF 77 51 B2 EF 77	Re'w.'i'wz!'w0w'w
0046087C	26 D5 EF 77 2A E3 EF 77 5F 39 F2 77 71 B4 EF 77	&'w*0'w_9=wqf'w
0046088C	2E AD EF 77 E1 61 EF 77 B8 85 EF 77 CC D2 EF 77	.i'wpa'w0a'wlf'w
0046089C	43 70 EF 77 FB EA F0 77 12 83 EF 77 01 72 F0 77	Cp'w'0-w+a'w0r-w
004608AC	A9 34 F0 77 D5 93 EF 77 68 EF EF 77 AA D2 EF 77	04-w'0'wh'wre'w
004608BC	B2 6F EF 77 3F 38 F2 77 D6 E8 EF 77 68 E0 EF 77	0o'w?8=wif'wh0'w
004608CC	00 60 EF 77 90 5B EF 77 6D AC EF 77 94 6C F0 77	.i'we['wm'w0l-w
004608DC	22 8D EF 77 3D C8 F1 77 3D 6D F0 77 6F C0 EF 77	"i'w=ztw=n-w0l'w
004608EC	85 7B EF 77 26 D9 EF 77 FB 5E EF 77 36 8A EF 77	ac'w&'w'w6e'w
004608FC	FC 8A EF 77 0F 62 EF 77 49 5E EF 77 97 5D EF 77	'e'w*b'wI^'wu'w
0046090C	1A 9A EF 77 68 FA EF 77 7B C9 F0 77 DA 98 F2 77	+u'wk.'wCf-wry=w
0046091C	1A 40 F2 77 55 EA EF 77 C5 61 EF 77 70 E6 EF 77	+0=wU0'wfa'wpy'w
0046092C	F0 81 EF 77 2D 6C EF 77 98 6E EF 77 4F 83 EF 77	-u'w-l'wgn'w0a'w
0046093C	09 ED EF 77 EB 8A EF 77 26 69 F0 77 B1 95 EF 77	.Y'wu-w&l-w0o'w
0046094C	6F B0 EF 77 8A 5A EF 77 E9 49 F2 77 26 F1 F0 77	o'wez'w'wz-w
0046095C	C9 D0 F0 77 51 E0 F0 77 33 8C EF 77 6C EC EF 77	f'w00-w3wly'w
0046096C	29 94 EF 77 00 00 00 00 68 17 80 7C D4 A7 80 7C	l0'w...k0e0C
0046097C	51 0E 81 7C EE 1E 80 7C 10 2F 81 7C 40 7A 94 7C	Q0u'-'A0'w'0e0
0046098C	09 2A 81 7C DA CD 81 7C 16 1E 80 7C 15 99 80 7C	.*u'='u'-'A0'80C
0046099C	F8 0E 81 7C B6 2B 81 7C E4 9A 80 7C 51 9A 80 7C	00u'-'A+u'-'A0'00C
004609AC	E3 14 82 7C BF 50 83 7C E8 8D 83 7C A8 CC 80 7C	00e'-'A+u'-'A0'00C
004609BC	62 2E 86 7C 77 DF 81 7C E7 4A 81 7C 5B CF 81 7C	b.'A'w'-'A0'00C
004609CC	08 2F 81 7C 9D 47 84 7C 0C 8A 83 7C 90 A4 80 7C	0'-'A0'00C'-'A0'00C
004609DC	CF BC 80 7C 62 D2 80 7C 62 15 81 7C 77 D0 80 7C	0'-'A0'00C'-'A0'00C
004609EC	5E A3 80 7C 78 34 83 7C ED 09 92 7C FD 79 92 7C	^u0'x4a'Y.E'2yE
004609FC	D4 05 92 7C 3D 04 92 7C A7 27 81 7C 76 2E 81 7C	E0E'-'A0'00C'-'A0'00C
00460A0C	8B B2 85 7C A9 60 83 7C 45 1C 83 7C 77 0A 81 7C	00a'0'-'A0'00C'-'A0'00C
00460A1C	3C 15 81 7C FE 4F 83 7C 54 5D 83 7C 23 2C 81 7C	<0u'-'A0'00C'-'A0'00C
00460A2C	72 67 83 7C 40 97 80 7C 27 09 83 7C C5 9B 80 7C	rga'00C'-'A0'00C
00460A3C	05 10 91 7C B9 23 81 7C ED 10 91 7C B9 4C 83 7C	00a'0'-'A0'00C'-'A0'00C
00460A4C	8A 18 92 7C 9F 2D 81 7C F1 9E 80 7C FC 38 81 7C	e0E'f-'A0'00C'-'A0'00C
00460A5C	A5 1B 82 7C 44 20 83 7C BC 22 83 7C 61 23 83 7C	n+e'ID a'-'A0'00C'-'A0'00C
00460A6C	47 9B 80 7C 41 26 81 7C 8E 0B 81 7C 87 0D 81 7C	G0C'-'A0'00C'-'A0'00C
00460A7C	0E 18 80 7C 24 1A 80 7C F5 D0 80 7C FE D0 80 7C	00C'-'A0'00C'-'A0'00C
00460A8C	01 BF 80 7C 0F AC 80 7C 00 F7 82 7C BB 0B 83 7C	00C'-'A0'00C'-'A0'00C

Vemos que ahi se arreglan todas y la IAT queda perfecta, veamos dumpeemos y repararemos el dump



004271B0 55 PUSH EBP  
 004271B1 8BEC MOV EBP,ESP  
 004271B3 6A FF PUSH -1  
 004271B5 68 60E4500 PUSH 450E60  
 004271B8 68 C8924200 PUSH 4292C8  
 004271BF 64:01 00000000 MOV EAX,DWORD PTR FS:[0]  
 004271C5 58 PUSH EAX  
 004271C6 64:8925 000000 MOV DWORD PTR FS:[0],ESP  
 004271CD 83C4 A8 ADD ESP,-58  
 004271D0 53 PUSH EBX  
 004271D1 56 PUSH ESI  
 004271D2 57 PUSH EDI  
 004271D3 8965 E8 MOV DWORD PTR SS:[EBP  
 004271D6 FF15 DC0A4600 CALL NEAR DWORD PTR D  
 004271DC 33D2 XOR EDX,EDX  
 004271DE 8AD4 MOV DL,AH  
 004271E0 8915 34E64500 MOV DWORD PTR DS:[45E  
 004271E6 8BC8 MOV ECX,EAX  
 004271E8 81E1 FF000000 AND ECX,0FF  
 004271EE 890D 30E64500 MOV DWORD PTR DS:[45E  
 004271F4 C1E1 08 SHL ECX,8  
 004271F7 03CA ADD ECX,EDX  
 004271F9 890D 2CE64500 MOV DWORD PTR DS:[45E  
 004271FF C1E8 10 SHR EAX,10  
 00427202 A3 28E64500 MOV DWORD PTR DS:[45E  
 00427207 E8 94210000 CALL 004293A0  
 0042720C 85C0 TEST EAX,EAX  
 0042720E 75 0A JNZ SHORT 0042721A  
 00427210 6A 1C PUSH 1C  
 00427212 E8 49010000 CALL 00427360  
 00427217 83C4 04 ADD ESP,4  
 0042721A E8 D12F0000 CALL 0042A1F0  
 0042721F 85C0 TEST EAX,EAX  
 00427221 75 0A JNZ SHORT 0042722D  
 00427223 6A 10 PUSH 10  
 EBP=0004C5F0

Registers (FPU)  
 EAX 0004C70C  
 ECX C0000005  
 EDX 00000000  
 EBX 00000004  
 ESP 0004BF54  
 EBP 0004C5F0  
 ESI 0004C618  
 EDI 00000000  
 EIP 004271B0 UnPackMe\_Execrptor2.2.50.a.exe

OllyDump - UnPackMe\_Execrptor2.2.50.a.exe

Start Address: 400000 Size: DE000 Dump  
 Entry Point: DDEA6 -> Modify: 271B0 Get EIP as OEP Cancel  
 Base of Code: 93000 Base of Data: 4B000

☒ Fix Raw Size & Offset of Dump Image

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	0004A000	00001000	0004A000	00001000	E0000020
.rdata	0000C000	00048000	0000C000	00048000	C0000040
.data	00009000	00057000	00009000	00057000	C0000040
.dyntioj	00003000	00060000	00003000	00060000	E0000060
.rsrc	00008000	00063000	00008000	00063000	40000040
xd4am...	00001000	00068000	00001000	00068000	C0000040
uvxww...	00027000	0006C000	00027000	0006C000	E0000020
mlmainz	00048000	00093000	00048000	00093000	E0000060

☐ Rebuild Import  
☒ Method1 : Search JMP[API] | CALL[API] in memory image  
☐ Method2 : Search DLL & API name string in dumped file

Ahora abrimos el IMP REC

Import REConstructor v1.6 FINAL (C) 2001-2003 MackT/uCF

Attach to an Active Process  
 c:\documents and settings\ricardo\escritorio\escritorio\execrptor 2.2.50\execrptor 2.2.50 Pick DLL

Imported Functions Found

- advapi32.dll FTThunk:00060818 NbFunc:5 (decimal:5) valid:YES
- comctl32.dll FTThunk:00060830 NbFunc:2 (decimal:2) valid:YES
- gdi32.dll FTThunk:0006083C NbFunc:4D (decimal:77) valid:YES
- kernel32.dll FTThunk:00060974 NbFunc:8C (decimal:140) valid:YES
- oleaut32.dll FTThunk:00060BA8 NbFunc:10 (decimal:16) valid:YES
- shell32.dll FTThunk:00060BEC NbFunc:3 (decimal:3) valid:YES
- user32.dll FTThunk:00060BFC NbFunc:A3 (decimal:163) valid:YES
- winmm.dll FTThunk:00060E8C NbFunc:1 (decimal:1) valid:YES
- winspool.drv FTThunk:00060E94 NbFunc:8 (decimal:8) valid:YES

Log

1B9 (decimal:441) imported function(s). (added: +1B9 (decimal:+441))  
 IAT read successfully.

Current imports:  
 C (decimal:12) valid module(s)  
 1B9 (decimal:441) imported function(s).

IAT Infos needed  
 OEP 000271B0 IAT AutoSearch  
 RVA 00060818 Size 00000710

New Import Infos (IID+ASCII+LOADER)  
 RVA 00000000 Size 00001F1E  
☒ Add new section

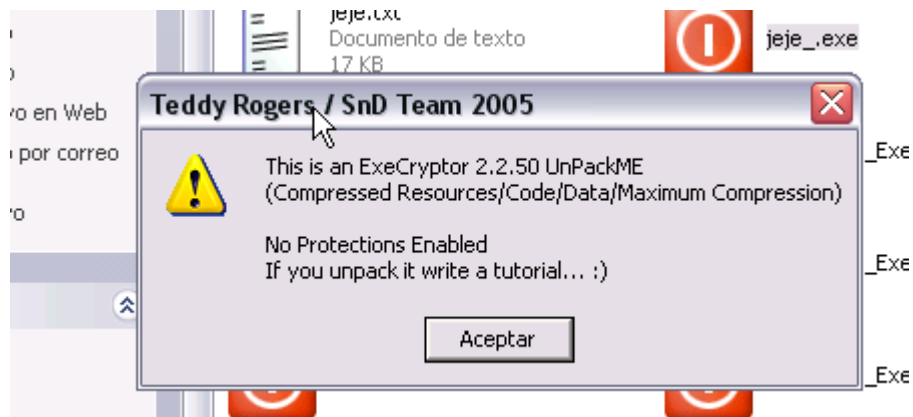
Load Tree Save Tree Get Imports Fix Dump

Show Invalid Show Suspect Auto Trace Clear Imports Clear Log Options About Exit

Ahora repararemos el dump y quitemosle el TLS poniendo a cero en el header

00400148	00000000	DD 00000000	Global Ptr address = 0
0040014C	00000000	DD 00000000	Must be 0
00400150	00000000	DD 00000000	TLS Table address = 0
00400154	00000000	DD 00000000	TLS Table size = 0
00400158	00000000	DD 00000000	Load Config Table address = 0
0040015C	00000000	DD 00000000	Load Config Table size = 0

y ahora si quedo bien, arranca del EP 4271b0 y la IAT y el codigo, estan identicos a la del UPX que usamos para comparar, incluso el codigo de la primera seccion es tambien identico no hay ni una sola diferencia.



Corro el dumpeado reparado y funciona perfectamente, esto demuestra que mas vale maña que fuerza y que todo se vale en el cracking y en el amor jajaja.(y que si juegan sucio podemos jugar peor nosotros jeje)

Hasta la parte siguiente

Ricardo Narvaja

14/10/06

## **INTRODUCCION AL CRACKING DESDE CERO CON OLLYDBG PARTE 56**

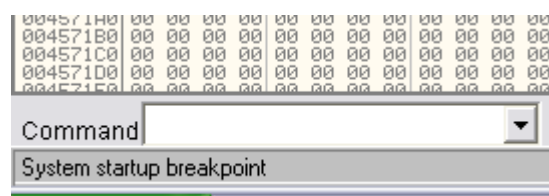
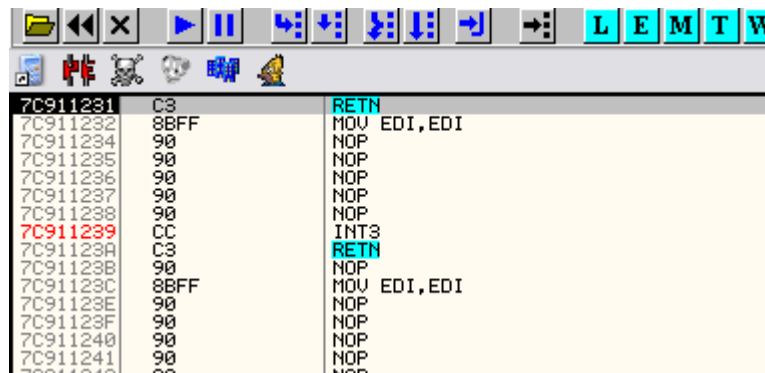
La verdad yo pensaba seguir con otro tema pero tuve muchos pedidos de que vayamos avanzando en el mismo execryptor en los crackmes b, c, etc a ver si podemos llegar hasta la maxima proteccion, y bueno haremos lo posible, primero que nada veamos que dice que tiene el b.

Al correrlo fuera de OLLYDBG nos dice.

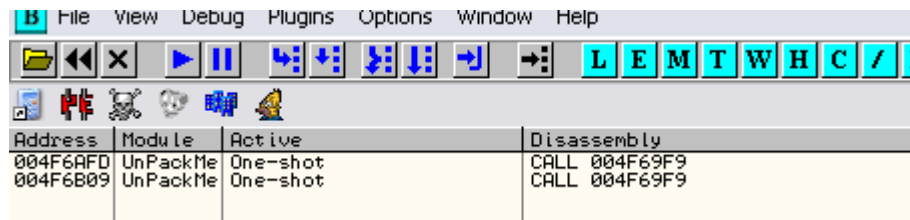


Bueno, alli esta dice que este el b es un poco mas dificil o sea modo agresivo, wow, estoy que tiemblo, jeje.

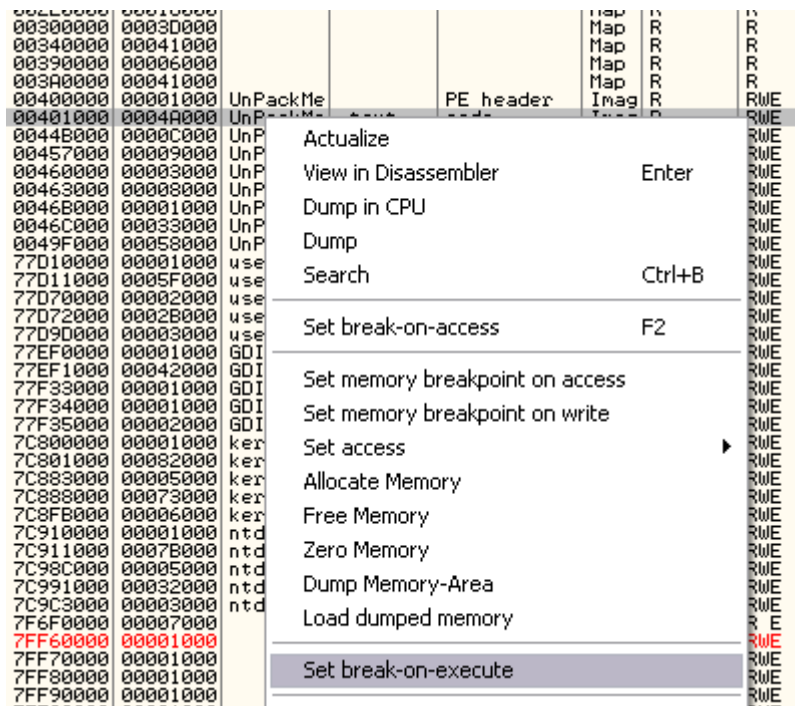
Veamos antes que nada si corre en OLLYDBG con la configuracion que usamos en el a, con el plugin OLLY ADVANCED y parando en system breakpoint para borrar el BP que coloca OLLYDBG para que pare en el Entry Point.



Alli estamos parados en el system breakpoint, y vamos a la ventana b para borrar el breakpoint que coloca OLLYDBG automaticamente.

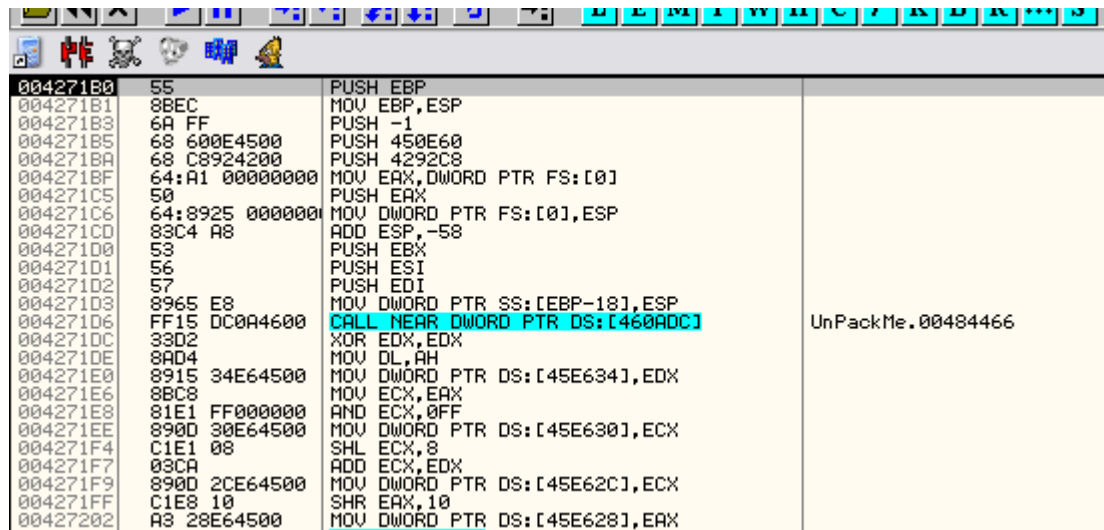


Bueno los borro y pongo un BREAK ON EXECUTE en la primera seccion como en el anterior.



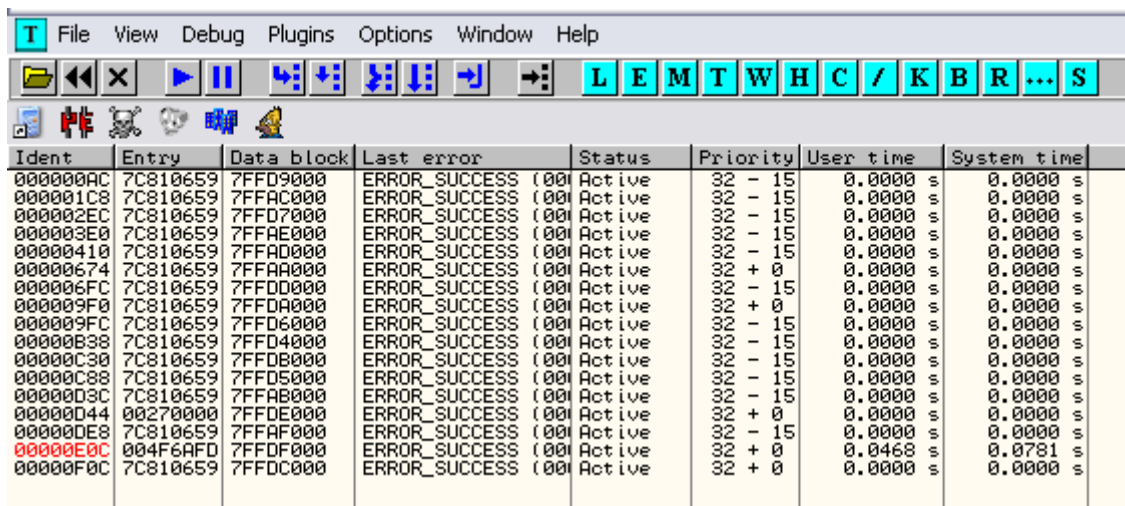
Recordemos que para esto estamos usando el plugin OLLYBONE.

Bueno demos RUN



Bueno hasta aquí no hay diferencias, quitemos el BREAK ON EXECUTE y sigamos.

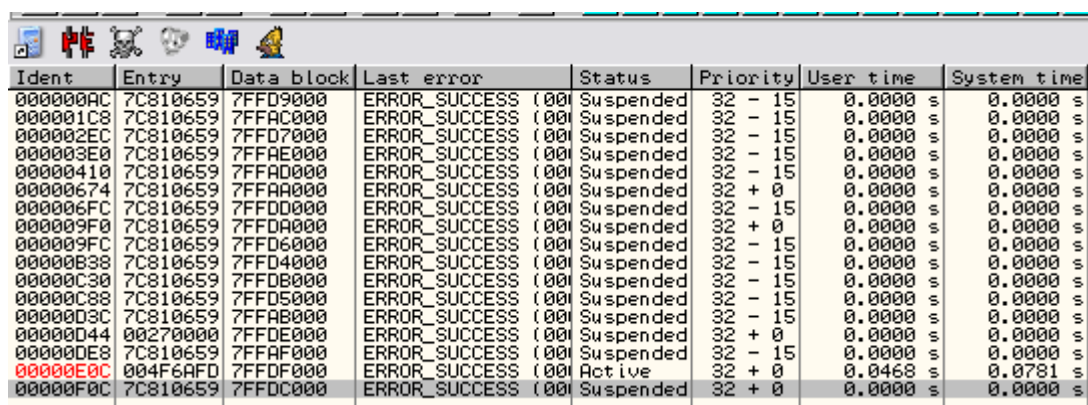
Lo que veo si miro un poco es que este tiene muchos threads a comparacion del a que tenia solo dos cuando llegaba al OEP.



Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000000AC	7C810659	7FFD9000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000001C8	7C810659	7FFAC000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000002EC	7C810659	7FFD7000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000003E0	7C810659	7FFAE000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
00000410	7C810659	7FFAD000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
00000674	7C810659	7FFAA000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s
000006FC	7C810659	7FFDD000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000009F0	7C810659	7FFDA000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s
000009FC	7C810659	7FFD6000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
00000B38	7C810659	7FFD4000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
00000C30	7C810659	7FFDB000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
00000C38	7C810659	7FFD5000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
00000D3C	7C810659	7FFAB000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
00000D44	00270000	7FFDE000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s
00000DE8	7C810659	7FFAF000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
00000E0C	004F6AFD	7FFDF000	ERROR_SUCCESS (000)	Active	32 + 0	0.0468 s	0.0781 s
00000F0C	7C810659	7FFDC000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s

Seguro estos threads se la pasaran vigilando si pasa algo raro para echarme fuera.

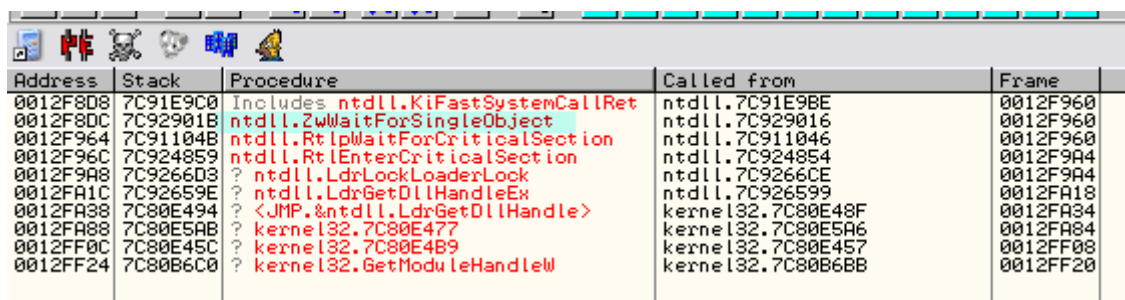
Vemos que si le doy RUN el crackme corre, pero hay que ver si en el funcionamiento del script estos threads vigilantes no me echan afuera al ver el tiempo que consume etc, veamos que pasa si suspendo todos los threads menos el actual o sea el que esta en rojo.



Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000000AC	7C810659	7FFD9000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
000001C8	7C810659	7FFAC000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
000002EC	7C810659	7FFD7000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
000003E0	7C810659	7FFAE000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000410	7C810659	7FFAD000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000674	7C810659	7FFAA000	ERROR_SUCCESS (000)	Suspended	32 + 0	0.0000 s	0.0000 s
000006FC	7C810659	7FFDD000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
000009F0	7C810659	7FFDA000	ERROR_SUCCESS (000)	Suspended	32 + 0	0.0000 s	0.0000 s
000009FC	7C810659	7FFD6000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000B38	7C810659	7FFD4000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C30	7C810659	7FFDB000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C38	7C810659	7FFD5000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D3C	7C810659	7FFAB000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D44	00270000	7FFDE000	ERROR_SUCCESS (000)	Suspended	32 + 0	0.0000 s	0.0000 s
00000DE8	7C810659	7FFAF000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E0C	004F6AFD	7FFDF000	ERROR_SUCCESS (000)	Active	32 + 0	0.0468 s	0.0781 s
00000F0C	7C810659	7FFDC000	ERROR_SUCCESS (000)	Suspended	32 + 0	0.0000 s	0.0000 s

Alli hice click derecho – SUSPENDED y suspendi todos los threads menos el principal que esta en rojo, correa igual? Veamos.

Vemos que queda RUNNING que no corre hmm esto se pone espeso.



Address	Stack	Procedure	Called from	Frame
0012F8D8	7C91E9C0	Includes ntdll.KiFastSystemCallRet	ntdll.7C91E9BE	0012F960
0012F8DC	7C92901B	ntdll.ZwWaitForSingleObject	ntdll.7C929016	0012F960
0012F964	7C91104B	ntdll.RtlpWaitForCriticalSection	ntdll.7C911046	0012F960
0012F96C	7C924859	ntdll.RtlEnterCriticalSection	ntdll.7C924854	0012F9A4
0012FA98	7C9266D3	? ntdll.LdrLockLoaderLock	ntdll.7C9266CE	0012F9A4
0012FA1C	7C92659E	? ntdll.LdrGetDllHandleEx	ntdll.7C926599	0012FA18
0012FA38	7C80E494	? <JMP.&ntdll.LdrGetDllHandle>	kernel32.7C80E48F	0012FA34
0012FA88	7C80E5AB	? kernel32.7C80E477	kernel32.7C80E5A6	0012FA84
0012FF0C	7C80E45C	? kernel32.7C80E4B9	kernel32.7C80E457	0012FF08
0012FF24	7C80B6C0	? kernel32.GetModuleHandleW	kernel32.7C80B6BB	0012FF20

Al pausar y mirar el call stack vemos que hay llamadas a WaitForSingleObject o sea que este thread



esta esperando algo y como los otros estan pausados, pues no sigue.

Enpecemos a probar de a uno a ver cual necesita para seguir demosle RESUME al primero y RUN al programa a ver si sigue.

Ident	Entry	Data block	Last error	Status	Priority	User time
000000AC	7C810659	7FFD9000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000
000001C8	7C810659	7FFAC000	ERROR_SUCCESS (00)	Su		
000002EC	7C810659	7FFD7000	ERROR_SUCCESS (00)	Su		
000003E0	7C810659	7FFAE000	ERROR_SUCCESS (00)	Su		
00000410	7C810659	7FFAD000	ERROR_SUCCESS (00)	Su		
00000674	7C810659	7FFA0000	ERROR_SUCCESS (00)	Su		

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000000AC	7C810659	7FFD9000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
000001C8	7C810659	7FFAC000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000002EC	7C810659	7FFD7000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000003E0	7C810659	7FFAE000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000410	7C810659	7FFAD000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000674	7C810659	7FFA0000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s
000006FC	7C810659	7FFD0000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000009F0	7C810659	7FFDA000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s
000009FC	7C810659	7FFD6000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000B38	7C810659	7FFD4000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C30	7C810659	7FFDB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C88	7C810659	7FFD5000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D3C	7C810659	7FFAB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D44	00270000	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000DE8	7C810659	7FFAF000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E0C	004F6AFD	7FFDF000	ERROR_SUCCESS (00)	Active	32 + 0	0.0468 s	0.0781 s
00000F0C	7C810659	7FFDC000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s

Vemos que queda RUNNING, ahora sin pausar el programa suspendamos el primero y pongamos a correr al segundo asi hasta que hallemos si hay uno solo que hace el programa correr.

00000C88	7C810659	7FFD5000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D3C	7C810659	7FFAB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D44	00270000	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000DE8	7C810659	7FFAF000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E0C	004F6AFD	7FFDF000	ERROR_SUCCESS (00)	Active	32 + 0	0.0781 s	0.1250 s
00000F0C	7C810659	7FFDC000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s



Cuando activo ese thread, el programa corre.

Vemos que el thread es una rutina que empieza en 270000 en mi maquina.

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000000AC	7C810659	7FFD9000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
000001C8	7C810659	7FFAC000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
000002EC	7C810659	7FFD7000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
000003E0	7C810659	7FFAE000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000418	7C810659	7FFAD000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000670	7C810659	7FFAA000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
000006FC	7C810659	7FFD0000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
000009F0	7C810659	7FFDA000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
000009FC	7C810659	7FFD6000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000B38	7C810659	7FFD4000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C30	7C810659	7FFDB000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C88	7C810659	7FFD5000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D3C	7C810659	7FFAB000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D44	00270000	7FFDE000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s
00000DE8	7C810659	7FFAF000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E0C	004F6AFD	7FFDF000	ERROR_SUCCESS (000)	Active	32 + 0	0.0937 s	0.2031 s
00000F0C	7C810659	7FFDC000	ERROR_SUCCESS (000)	Suspended	32 + 0	0.0000 s	0.0000 s

Vemos que es solo una rutinita que esta alli

00270000	55	PUSH EBP
00270001	56	PUSH ESI
00270002	57	PUSH EDI
00270003	53	PUSH EBX
00270004	E8 00000000	CALL 00270009
00270009	58	POP EAX
0027000A	83C0 10	ADD EAX,10
0027000D	50	PUSH EAX
0027000E	64:FF35 00000000	PUSH DWORD PTR FS:[0]
00270015	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
0027001C	53	PUSH EBX
0027001D	FFD7	CALL NEAR EDI
0027001F	EB 12	JMP SHORT 00270033
00270021	8B6424 08	MOV ESP,DWORD PTR SS:[ESP+8]
00270025	64:8F05 00000000	POP DWORD PTR FS:[0]
0027002C	B8 010000C0	MOV EAX,C0000001
00270031	EB 07	JMP SHORT 0027003A
00270033	64:8F05 00000000	POP DWORD PTR FS:[0]
0027003A	83C4 04	ADD ESP,4
0027003D	5B	POP EBX
0027003E	5F	POP EDI
0027003F	5E	POP ESI
00270040	5D	POP EBP
00270041	50	PUSH EAX
00270042	FFD6	CALL NEAR ESI
00270044	C3	RETN
00270045	CC	INT3

Pongamos un BPM ON ACCESS en dicha seccion



00150000	00057000				Priv	RW	RW
00250000	00066000				Priv	RW	RW
00260000	00001000				Priv	E	RW
00270000	00001000				Priv	F	RW
00280000	00001000						RW
002CE000	00002000			Actualize			RW
002D0000	00003000			Dump in CPU			RW
002E0000	00016000			Dump			R
00300000	0003D000			Dump			R
00340000	00041000			Dump			R
00390000	00066000			Search		Ctrl+B	R
003A0000	00041000			Search		Ctrl+B	R
003F0000	00001000			Search		Ctrl+B	R
00400000	00001000	UnPackMe		Set break-on-access		F2	RWE
00401000	0004A000	UnPackMe		Set break-on-access		F2	RWE
0044B000	0000C000	UnPackMe		Set break-on-access		F2	RWE
00457000	00009000	UnPackMe		Set memory breakpoint on access			RWE
00460000	00003000	UnPackMe		Set memory breakpoint on access			RWE
00463000	00008000	UnPackMe		Set memory breakpoint on write			RWE

Si doy Run no para así que cambiemos de estrategia.

El tema hay que ver como funciona con estos threads vigilando lo que reparamos la IAT, debemos tratar de que sean los menos posible, lleguemos de nuevo al OEP:

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000002E0	7C810659	7FFDD000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000304	7C810659	7FFAE000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000478	7C810659	7FFD8000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000480	7C810659	7FFAA000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000A88	7C810659	7FFAC000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000BC4	00270000	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000C14	7C810659	7FFD5000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000C48	7C810659	7FFD4000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000D10	7C810659	7FFDC000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000D64	7C810659	7FFD6000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000D70	7C810659	7FFAF000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000E64	7C810659	7FFAB000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000F20	7C810659	7FFDB000	ERROR_SUCCESS (00)	Active	32 - 15	0.0156 s	0.0000 s
00000F90	004F6AFD	7FFDF000	ERROR_SUCCESS (00)	Active	32 + 0	0.0468 s	0.0625 s
00000F94	7C810659	7FFAD000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000FC8	7C810659	7FFD9000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000FF4	7C810659	7FFD7000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s

Verifiquemos si solo con el tread actual y el que en mi maquina empieza en 270000 el unpackme corre ya que cuanto menos interferencia mejor, si en algun programa con dos threads solos no corre pues habra que buscar un tercero y tratar de que sean los menos posibles, otra posibilidad es llegar hasta el ExitProcess con un Bp en dicha api, ya que alli el programa mata todos los threads antes de cerrarse y ver si alli podemos correr el script.

Probemos si corre con esos dos solos suspendiendo todos los otros.

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000002E0	7C810659	7FFDD000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000304	7C810659	7FFAE000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000478	7C810659	7FFD8000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000480	7C810659	7FFAA000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000A88	7C810659	7FFAC000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000BC4	00270000	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000C14	7C810659	7FFD5000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C48	7C810659	7FFD4000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D10	7C810659	7FFDC000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D64	7C810659	7FFD6000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D70	7C810659	7FFAF000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E64	7C810659	7FFAB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000F20	7C810659	7FFDB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0156 s	0.0000 s
00000F90	004F6AFD	7FFDF000	ERROR_SUCCESS (00)	Active	32 + 0	0.0468 s	0.0625 s
00000F94	7C810659	7FFAD000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s
00000FC8	7C810659	7FFD9000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s
00000FF4	7C810659	7FFD7000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s

Ahi esta, una forma tambien que muchas veces sirve entre tanto thread es ver la prioridad, aquí vemos que este tiene la misma prioridad que el principal, mientras que hay muchos con menos prioridad.

Demos RUN a ver que pasa.



Bueno esos dos threads son lo minimo necesario que el unpackme necesita para correr.

Les recuerdo que en estos packers dificiles, nunca hay que llevarse por pasos exactos, ni leer un tute y pensar que todo va a ser identico, por eso este no es un tute sobre execryptor en si, si no sobre como pensar algo complicado, y como irse adaptando a las circunstancias adversas que se nos van planteando, como me va ocurriendo a mi a medida que lo vos solucionando y escribiendo a la vez. La idea es que el que lee aprenda a moverse de lo estrictamente automatico y aprenda a moverse siempre un poco del clasico apreta 5 veces f8, 3 veces f7 y ya esta, hay que pensar y adaptarse a las situaciones nuevas.

Puede ocurrir que en otra maquina mi solucion no corra o haya que hacerle algun ajuste, en casos como este donde las rutinas son tan largas y complejas que el OLLYDBG esta largo rato traceando y que no podemos controlar todo lo que hace el packer, pueden ocurrir variaciones, pues es alli donde el que lee debe moverse un poquito de la rigidez, y pensar como adaptarse en ese caso y es lo que debera hacer siempre, pensar por si mismo que es lo que queremos lograr que aprendan con este curso mas que un tute mas de execryptor que hay muchos.

Lo importante es aprender a improvisar y pensar y adaptarse a situaciones nuevas, execryptor puede cambiar y salir del esquema que estamos mostrando pero el que aprende a buscar soluciones propias siempre puede moverse en nuevas direcciones y solucionarlo.

Bueno como vi las veces anteriores que lo corri que la primera entrada de la IAT al igual que en la version "a", se automodifica y guarda el valor correcto en la misma, pues probare primero el mismo metodo por ahora que en la parte anterior, poner un BPM ON WRITE en dicha entrada y poner a tracear el OLLY hasta que pare cuando va a guardar en ella, luego de largo rato traceando para aqui. (el traceo esta adjunto al tute)

00483FE3	8905 DC0A4601	MOV DWORD PTR DS:[460ADC],EAX	kernel32.GetVersion
00483FE9	8D05 70464901	LEA EAX,DWORD PTR DS:[494670]	
00483FEF	68 DF8A4800	PUSH 488ADF	
00483FF4	E9 B0EFFFFF	JMP 00482FA9	UnPackMe.00482FA9
00483FF9	8B0424	MOV EAX,DWORD PTR SS:[ESP]	
00483FFC	E8 B3FA0000	CALL 00493AB4	UnPackMe.00493AB4

En el script anterior, unas lineas mas arriba en el traceo, habia una instrucci3n que movia a EAX la direcci3n correcta de la api y eso servia para cualquier entrada veamos si aqu3 ocurre igual.

8.	Main	UnPackMe 0048158C	POP ECX	ECX=7C81120A, ESP=0012FE08
7.	Main	UnPackMe 0048158D	POP EBP	ESP=0012FE0C, EBP=0012FF38
6.	Main	UnPackMe 0048158E	RETN	ESP=0012FE10
5.	Main	UnPackMe 00483420	MOV EAX,DWORD PTR SS:[EBP-C]	EAX=7C81110A
4.	Main	UnPackMe 00483423	MOV ESP,EBP	ESP=0012FF38
3.	Main	UnPackMe 00483425	POP EBP	ESP=0012FF3C, EBP=0012FFC0
2.	Main	UnPackMe 00483426	RETN	ESP=0012FF40
1.	Main	UnPackMe 00475937	RETN	ESP=0012FF44
0.	Main	UnPackMe 00483FE3	MOV DWORD PTR DS:[460ADC],EAX	

Vemos que ahora en 483420 es donde mueve a EAX el valor correcto de la api asi que siguiendo la logica del script anterior, pues el mismo deberia poner un HE on EXECUTION en 483423 que es la

El script final de la parte anterior era

```
final:
ret
```

[illegible]

```
00483420 > 8B45 F4    MOV EAX,DWORD PTR SS:[EBP-C]    ;
kernel32.GetVersion
00483423 . |8BE5    MOV ESP,EBP
```

asi que en el script debemos reemplazar 47691f que era la direccion donde el script ponia el HE ON EXECUTION para obtener el valor correcto de EAX por 483423 quedaria asi.

---

```
var tabla
```

```
var contenido
```

```
mov tabla,460818
```

```
start:
```

```
cmp tabla,460f28
```

```
ja final
```

```
cmp [tabla],50000000
```

```
ja saltar
```

```
mov contenido,[tabla]
```

```
cmp contenido,0
```

```
je saltar
```

```
log contenido
```

```
log tabla
```

```
mov eip,contenido
```

```
bphws 483423, "x"
```

```
mov [483423],0
```

```
mov [483424],0
```

```
cob reparar
```

```
run
```

```
reparar:
```

```
cmp eip,7c91e88e
```

```
je saltar
```

```
log eax
```

```
mov [tabla],eax
```

```
run
```

```
saltar:
```

```
add tabla,4
```

```
jmp start
```

```
final:
```

```
ret
```

---

Ahora recordemos que debemos suspender todos los threads menos los 2 que detectamos como necesarios antes de correr el script.

[illegible][illegible]

La primera entrada que fallo es la 46099c veamos en el LOG a ver si encontramos algo raro.

HL=00

Address	Hex dump	ASCII
0046097C	51 0E 81 7C EE 1E 80 7C 1D 2F 81 7C 00 00 00 00	Q&u!-!C!#/?u!...!
0046098C	09 2A 81 7C DA CD 81 7C 16 1E 80 7C 15 99 80 7C	.*u! =u!-!C!S0C!
0046099C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
004609AC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
004609BC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
004609CC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
004609DC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
004609EC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
004609FC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
00460A0C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
00460A1C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
00460A2C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
00460A3C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
00460A4C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E..E..E..E..E..E..
00460A5C	A5 18 82 7C 00 00 00 00 00 00 00 00 00 00 00 00	~+e!.....
00460A6C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460A7C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460A8C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460A9C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460AAC	00 00 00 00 00 00 00 00 94 F8 46 00 47 BD 48 00	.....8°F.GcH.
00460ABC	F9 64 47 00 CB FE 47 00 58 04 48 00 A1 40 48 00	..dG.πG.X♦H.i0H.
00460ACC	9D 98 48 00 7A 4C 48 00 26 87 49 00 7D 58 49 00	0xH.zLH.&C!.JXI.
00460ADC	66 44 48 00 F7 8A 48 00 F8 CA 46 00 FF 52 47 00	fNH.66H.°4F.■RA.

```

00483423 Access violation when writing to [7C809915]
          contenido: 00472EBD ; Entry address
          tabla: 0046099C
0047F5D8 Access violation when reading [00880045]
          eax: 00880045
0047F5D8 Access violation when reading [00880045]
          contenido: 00472719 ; Entry address
          tabla: 004609A0
004824D3 Access violation when reading [00880045]
          eax: 00880045
004824D3 Access violation when reading [00880045]
          contenido: 0047C259 ; Entry address
          tabla: 004609A4
0047B961 Access violation when reading [00880045]
          eax: 00880045
0047B961 Access violation when reading [00880045]
          contenido: 0048A03E ; Entry address
          tabla: 004609A8
004770E5 Access violation when reading [00880045]
          eax: 00880045
004770E5 Access violation when reading [00880045]
          contenido: 0048EF06 ; Entry address
          tabla: 004609AC
00476492 Access violation when reading [00880045]
          eax: 00880045
00476492 Access violation when reading [00880045]
          contenido: 0047DDE8 ; Entry address
          tabla: 004609B0
0048A42D Access violation when reading [00880045]
          eax: 00880045
0048A42D Access violation when reading [00880045]
          contenido: 0049E46F
          tabla: 004609B4
004872D2 Access violation when reading [00880045]
          eax: 00880045
004872D2 Access violation when reading [00880045]
          contenido: 00480313 ; Entry address
          tabla: 004609B8
00482C53 Access violation when reading [00880045]
          eax: 00880045
00482C53 Access violation when reading [00880045]

```

Alli vemos que a partir de esa entrada se produce una excepcion que antes no se producía en 880045 y que a partir de allí se pudre todo jeje

Si cambio el script así para que cuando se produzca cualquier excepción saltee a la entrada siguiente, pero sigue fallando en la misma entrada, solo que ahora saltea todas las malas y no las guarda mal.

```

reparar:
cmp eip,483423
jne saltar
log eax
mov [tabla],eax
run

```

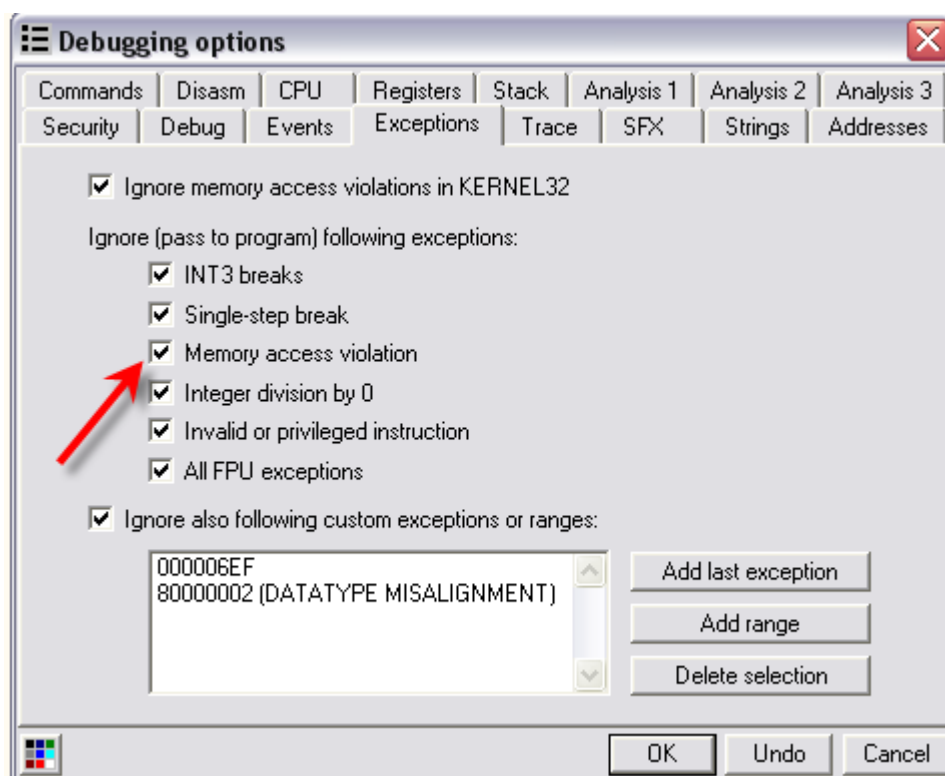


Que es lo que ocurre aquí ? tanto vario el programa del a al b?, algunos listeros que quisieron solucionar el a con el script les paso esto, pues entonces donde esta el error?, veamos

A mi cuando lo solucione al a, cuando detectaba algun error saltaba a cerrar el programa y aquí no, puedo ver en el log que no salta a ZwTerminateProcess,  
En el a yo le provocaba una excepcion para que en cada entrada se provoque un error a proposito, y salte a cerrarlo, interceptar ese error y volver con el HE puesto en ZwTerminateProcess, pero aquí nunca salta a terminar el programa sera por la tilde que quite en memory access exception?, veamos si la coloco y uso el script nuevamente.

Address	Message
00483423	tabla: 00460A30
00483423	Hardware breakpoint 3 at UnPackMe.00483423
00483423	eax: 7C809740 ! kernel32.TlsGetValue
00483423	Access violation when writing to [7C809740]
7C91E88E	Hardware breakpoint 2 at ntdll.ZwTerminateProcess
7C91E88E	contenido: 0049EE49
7C91E88E	tabla: 00460A34
00483423	Hardware breakpoint 3 at UnPackMe.00483423
00483423	eax: 7C830927 ! kernel32.LocalReAlloc
00483423	Access violation when writing to [7C830927]
7C91E88E	Hardware breakpoint 2 at ntdll.ZwTerminateProcess
7C91E88E	contenido: 00474D6A
7C91E88E	tabla: 00460A38 ! ASCII "jMG"
00483423	Hardware breakpoint 3 at UnPackMe.00483423
00483423	eax: 7C809BC5 ! kernel32.TlsSetValue
00483423	Access violation when writing to [7C809BC5]
7C91E88E	Hardware breakpoint 2 at ntdll.ZwTerminateProcess
7C91E88E	contenido: 004733CA
7C91E88E	tabla: 00460A3C
00483423	Hardware breakpoint 3 at UnPackMe.00483423
00483423	eax: 7C911005 ! ntdll.RtlEnterCriticalSection
00483423	Access violation when writing to [7C911005]
7C91E88E	Hardware breakpoint 2 at ntdll.ZwTerminateProcess
7C91E88E	contenido: 00498571
7C91E88E	tabla: 00460A40
00483423	Hardware breakpoint 3 at UnPackMe.00483423
00483423	eax: 7C8123B9 ! kernel32.GlobalReAlloc
00483423	Access violation when writing to [7C8123B9]

AHHHHHHHHHHHHH, ahora si funciona como el anterior, en cada entrada salta por el error que yo produje a cerrar el programa a ZwTerminateProcess y ahora si la tabla se arregla completa sin problema, el error era tener desmarcada la excepcion memory access habia que tenerla marcada asi saltaba a terminar el programa, disculpen el error.



Entonces debia estar asi colocada la tilde para que el script funcione ahora veamos como quedo la tabla.

Address	Hex dump	ASCII
00460818	F0 6B DA 77 1B 76 DA 77 F4 EA DA 77 E7 EB DA 77	-k r w + v r w q u r w f u r w
00460828	83 78 DA 77 00 00 00 00 CF 65 C3 58 D8 03 C4 58	ãk r w . . . ð e   x i - x
00460838	00 00 00 00 04 6A EF 77 66 95 EF 77 89 6A EF 77	. . . ë j ' w f ò ' w ë j ' w
00460848	F3 AD EF 77 ED D9 EF 77 99 8B EF 77 C0 B5 EF 77	% ã ' w j ' w ò i ' w l ã ' w
00460858	2A 7D EF 77 B2 7C EF 77 77 53 F2 77 1E C9 F1 77	* j ' w ë i ' w w s = w ã f ' w
00460868	0C BC EF 77 52 D4 EF 77 FA 8D EF 77 F1 D0 EF 77	' ã ' w R E ' w ' i ' w t ' w
00460878	51 B2 EF 77 26 D5 EF 77 2A E3 EF 77 5F 39 F2 77	Ö ë ' w & ' w * b ' w 9 = w
00460888	71 B4 EF 77 2E AD EF 77 E1 61 EF 77 B8 85 EF 77	q l ' w . ã ' w p a ' w ò ã ' w
00460898	CC D2 EF 77 43 70 EF 77 FB EA F0 77 12 83 EF 77	l f e ' w C p ' w ' ò - w f ã ' w
004608A8	01 72 F0 77 A9 34 F0 77 D5 93 EF 77 68 EF EF 77	ò r - w ò 4 - w ' ò ' w h ' w
004608B8	AA D2 EF 77 B2 6F EF 77 3F 38 F2 77 D6 E8 EF 77	- e ' w ë o ' w ? ò = w i p ' w
004608C8	68 E0 EF 77 00 60 EF 77 90 58 EF 77 6D AC EF 77	h ò ' w . ' w e l ' w m % ' w
004608D8	94 6C F0 77 22 8D EF 77 3D C8 F1 77 3D 6D F0 77	ò l - w ' l ' w = ã t = w = m - w
004608E8	6F C0 EF 77 85 7B EF 77 26 D9 EF 77 FB 5E EF 77	o ' w ã t ' w & ' w i ^ ' w
004608F8	36 8A EF 77 FC 8A EF 77 0F 62 EF 77 49 5E EF 77	6 è ' w ' e ' w * b ' w i ^ ' w
00460908	97 5D EF 77 1A 9A EF 77 6B FA EF 77 7B C9 F0 77	ù j ' w + ù ' w k ' w C l r - w
00460918	DA 98 F2 77 1A 40 F2 77 55 EA EF 77 C5 61 EF 77	r y = w + ò = w ù ò ' w t a ' w
00460928	70 E6 EF 77 F0 81 EF 77 2D 6C EF 77 98 6E EF 77	p v ' w - ù ' w - l ' w y n ' w
00460938	4F 83 EF 77 09 ED EF 77 EB AA EF 77 26 69 F0 77	ò ã ' w . y ' w ù - w & i - w
00460948	B1 95 EF 77 6F 8D EF 77 8A 5A EF 77 E9 49 F2 77	ë ò ' w c ' w e 2 ' w ù i = w
00460958	26 F1 F0 77 C9 D0 F0 77 51 E0 F0 77 33 8C EF 77	% t - w f i - w ò ò - w 3 i ' w
00460968	6C EC EF 77 29 94 EF 77 00 00 00 00 6B 17 80 7C	l y ' w ' ò ' w . . . k % ç !
00460978	D4 A7 80 7C 51 0E 81 7C EE 1E 80 7C 1D 2F 81 7C	e ç ç ! ö ã i ' ã ç ' ã i ' ã
00460988	40 7A 94 7C 09 2A 81 7C DA CD 81 7C 16 1E 80 7C	ò z ò . ' ã i ' ã ç ' ã i ' ã
00460998	15 99 80 7C F8 0E 81 7C B6 2B 81 7C E4 9A 80 7C	s ö ç ' ö ã i ' ã ç ' ã i ' ã
004609A8	51 9A 80 7C E3 14 82 7C BF 50 83 7C E8 80 83 7C	ò ö ç ' ö ã i ' ã ç ' ã i ' ã
004609B8	AB CC 80 7C 62 2E 86 7C 77 0F 81 7C E7 4A 81 7C	ç l f ç ' b . ã ' ã i ' ã ç ' ã i ' ã
004609C8	58 CF 81 7C 08 2F 81 7C 9D 47 84 7C 0C 0A 83 7C	l ç ù i ' ã i ' ã ç ' ã i ' ã
004609D8	90 A4 80 7C CF BC 80 7C 62 D2 80 7C 62 15 81 7C	e ã ç ' ç ç ' b e ç ' b s i ' ã
004609E8	77 D0 80 7C 5E A3 80 7C 78 34 83 7C ED 09 92 7C	w s ç ' ù ç ' x 4 ã ' ã i ' ã
004609F8	FD 79 92 7C D4 05 92 7C 3D 04 92 7C A7 27 81 7C	' y f e ' e ã f e ' ã f e ' ã i ' ã
00460A08	76 2E 81 7C 8B 82 85 7C A9 60 83 7C 45 1C 83 7C	v . ù i ' ã ã i ' ã ç ' ã i ' ã
00460A18	77 0A 81 7C 3C 15 81 7C FE 4F 83 7C 54 5D 83 7C	w . ù i ' < ã i ' ã ç ' ã i ' ã
00460A28	23 2C 81 7C 72 67 83 7C 40 97 80 7C 27 09 83 7C	# . ù i ' r g ã i ' ã ç ' ã i ' ã
00460A38	C5 9B 80 7C 05 10 91 7C B9 23 81 7C ED 10 91 7C	t + ç ç ' ã ã i ' ã ç ' ã i ' ã
00460A48	B9 4C 83 7C 8A 18 92 7C 9F 2D 81 7C F1 9E 80 7C	q l l ã i ' e f e ' f - ù i ' t x ç
00460A58	FC 38 81 7C A5 1B 82 7C 44 20 83 7C BC 22 83 7C	' s i ' ã i ' ã e d ã i ' ã i ' ã
00460A68	61 23 83 7C 47 9B 80 7C 41 26 81 7C 8E 0B 81 7C	a # ã i ' G a ç ' A ã i ' ã i ' ã
00460A78	87 0D 81 7C 0E 1B 80 7C 24 1A 80 7C F5 D0 80 7C	ç . ù i ' ã t ç ' s + ç ' s i ç
00460A88	FE D0 80 7C 01 BE 80 7C 0F AC 80 7C A0 F7 82 7C	# i ç ' ò ã ç ' * ã ç ' ã - e
00460A98	B8 0B 83 7C A1 B8 80 7C EB 98 80 7C 15 A4 80 7C	q l ã i ' l l ç ' ö y ç ' s ã ç
00460AA8	66 E8 80 7C EC E7 80 7C 01 9E 80 7C 79 9E 80 7C	f b ç ' y e ç ' ö x ç ' y x ç
00460AB8	74 0D 83 7C F8 9B 80 7C D4 A0 80 7C B6 BD 80 7C	t . ã i ' ç ç ' e ã ç ' ã ç ç
00460AC8	41 4D 83 7C 28 97 80 7C 19 FF 80 7C 82 FE 80 7C	A M ã i ' ö ç ' + ç ' e ã ç
00460AD8	CF B4 80 7C DA 11 81 7C C6 97 80 7C 19 B2 85 7C	ç H ç ' r ù ç ' ã i ' ã ç ' ã
00460AE8	AC 17 82 7C AB 1E 83 7C EE 86 82 7C 69 3F 87 7C	% f e ' % ã ã ' ã e ' l ? ç
00460AF8	03 DC 81 7C 11 13 87 7C 8B B5 81 7C 39 0A 87 7C	• ã i ' ã i ' ç ' l ã i ' 9 . ç
00460B08	20 99 80 7C 9C 92 80 7C 61 0A 87 7C 42 24 80 7C	ö ç ' ã f e ç ' ã . ç ' B s ç
00460B18	2B BC 81 7C F9 2F 87 7C 44 30 87 7C 0A 3C 87 7C	+ ã i ' / ç ' D ö ç ' < ç
00460B28	EE 61 83 7C 69 BC 80 7C 8D 34 87 7C D3 34 87 7C	' ã ã i ' ã ç ' l 4 ç ' e 4 ç
00460B38	1C 3B 87 7C 77 1D 80 7C A0 AD 80 7C DE A8 80 7C	L ; ç ' w # ç ' ã ç ç ' i % ç
00460B48	39 2F 81 7C 25 CF 81 7C 8D 10 87 7C B1 4E 83 7C	9 / ù i ' ã i ' ç ' l ç ' ã i ' ã
00460B58	D9 37 81 7C 31 03 92 7C 40 03 92 7C D7 ED 80 7C	+ 7 ù i ' l ö f e ' ö f e ' i y ç
00460B68	2D FD 80 7C 2F FC 80 7C DE 2A 81 7C 11 01 81 7C	- 2 ç ' / ç ' i * i ' ã i ' ã
00460B78	89 BE 80 7C B5 9F 80 7C 97 CC 80 7C B1 2E 83 7C	e # ç ' A f ç ' ö f ç ' ã i ' ã
00460B88	8D 99 80 7C 1D 2E 83 7C 2F 99 80 7C 7A 97 80 7C	l ö ç ' # . ã i ' ö ç ' z u ç
00460B98	66 97 80 7C A1 B6 80 7C 97 CC 80 7C 00 00 00 00	f u ç ' l ã ç ' ö f ç ' . . .
00460BA8	C0 48 0F 77 3B 4C 0F 77 94 A5 11 77 59 48 0F 77	' H * w ; L * w ö ã i ' w y k * w
00460BB8	82 4E 0F 77 98 D4 11 77 9B 50 0F 77 4F 50 0F 77	e N * w e i ' w p * w ö p * w
00460BC8	10 50 0F 77 3F 50 0F 77 D9 66 0F 77 50 48 0F 77	p * w ? p * w i ' f * w p H * w
00460BD8	55 4C 0F 77 C2 4B 0F 77 95 D2 11 77 80 5D 15 77	U L * w t k * w ö e i ' w ç i s w
00460BE8	00 00 00 00 C1 70 A8 7C B0 70 A8 7C B0 0E A5 7C	. . . - p e i ' p e i ' ã i ' ã
00460BF8	00 00 00 00 EA D6 D1 77 93 B6 D5 77 7D B5 D5 77	. . . ö i f w ö A ' w ö A ' w

Ahora si la arregla toda el script final entonces seria:

```

var tabla
var contenido

mov tabla,460818

start:
cmp tabla,460f28
ja final
cmp [tabla],50000000
ja saltear

```



```
mov contenido,[tabla]
cmp contenido,0
je saltar
log contenido
log tabla
```

```
mov eip,contenido
bphws 483423, "x"
mov [483423],0
mov [483424],0
cob reparar
run
```

```
reparar:
cmp eip,483423
jne saltar
log eax
mov [tabla],eax
run
```

```
saltar:
add tabla,4
jmp start
```

```
final:
ret
```

---

y hay que recordar poner el HE on EXECUTION a mano en ZwTerminateProcess antes de correrlo poner la tilde en MEMORY ACCESS EXCEPTION y suspender lso threads que estan de mas.

Entonces para recopilar llego al OEP, quito el BREAK ON EXECUTE, me fijo que esten marcadas todas las tildes de las excepciones, y pongo a mano el He ZwTerminateProcess, y arranco el script.

7C91E8B8	B8 03010000	MOV EAX,103
7C91E8BD	BA 0003FE7F	MOV EDX,7FFE0300
7C91E8C2	FF12	CALL NEAR DWORD PTR DS:[EDX]
7C91E8C4	C3	RETN
7C91E8C5	8D49 00	LEA ECX,DWORD PTR DS:[ECX]
7C91E8C9	90	NOP
7C91E8CA	90	NOP
7C91E8CB	90	NOP
7C91E8CC	90	NOP
7C91E8CD	B8 04010000	MOV EAX,104
7C91E8D0	BA 0003FE7F	MOV EDX,7FFE0300

EAX=0004C70C

Address	Hex dump	ASCII
00460D88	FE EC 03 77 83 F7 D4 77 DE F2 D2 77 DF 1A D3 77	uEw3-EwI-Ew+EW
00460D89	F6 F0 D4 77 9C F3 D4 77 33 F2 D2 77 6C C9 D1 77	+Ew3Ew3-EwlfW
00460D8A	F6 8B D1 77 88 96 D1 77 8C 94 D1 77 61 C6 D3 77	+iW00W.00wasEW
00460D8B	81 E5 D2 77 80 03 D3 77 55 E6 D1 77 AD A8 D1 77	iW0W0WUpWzW
00460D8C	EA 04 D4 77 24 13 D3 77 58 9F D1 77 33 89 D1 77	0W3EWW0W3iW
00460D8D	65 F6 D4 77 2F B7 D1 77 B4 F6 D4 77 6C BF D1 77	eEW/0WtEWiW
00460D8E	F5 B5 D1 77 24 15 D3 77 E2 C2 D1 77 29 69 D5 77	S0W\$3EW0WliW
00460D8F	DF BA D5 77 8C 14 D2 77 4C 1F D3 77 F9 D7 D1 77	WtEWLEW-iW
00460D90	F7 D6 D1 77 65 C4 D1 77 D4 B6 D1 77 C8 BD D1 77	-iW0E-0W0W0WcW
00460D91	AE B6 D1 77 CD 48 D2 77 3E 0B D2 77 C7 86 D1 77	<0W=Hew>0W0W0W
00460D92	9D 86 D1 77 26 BF D1 77 3F B5 D1 77 69 D8 D1 77	00W0W0W?0W0W0W
00460D93	85 C8 D1 77 71 BE D1 77 65 C6 D1 77 9D 8F D1 77	0W0W0W0W0W0W0W
00460D94	9D BE D1 77 31 B6 D1 77 17 E9 D3 77 00 EE D4 77	*0W10W0W0W-EW
00460D95	FA F3 D2 77 B5 37 D2 77 78 8E D1 77 88 EE D4 77	0W0W0W0W0W0W0W
00460D96	00 00 00 00 F7 A8 B1 76 00 00 00 00 C8 74 F8 72	.....tOr
00460D97	73 66 F9 72 87 72 F8 72 43 80 F8 72 67 37 F9 72	sf-rOrCCOr97-r
00460D98	FB 41 F9 72 67 83 F8 72 90 53 F8 72 00 00 00 00	'A-r00rESOr...
00460D99	CE 00 37 76 7C 86 37 76 80 56 37 76 33 25 36 76	fr7013703703260
00460D9A	1E 31 36 76 D8 7C 37 76 89 C2 37 76 CD 46 38 76	#16v117V07VrF0v
00460D9B	CC EE 36 76 00 00 00 00 48 D0 4C 77 9C CB 4D 77	fr6v...HSL00W
00460D9C	C2 42 4F 77 2C D0 4C 77 DA F6 4C 77 73 50 77	frB0w,SLwrLw3Pw
00460D9D	10 64 4D 77 03 0E 52 77 33 0F 52 77 40 A6 54 77	rdMw00Rw3Rw00W
00460D9E	F1 A7 54 77 92 9C 4F 77 6F 57 52 77 99 33 4E 77	:0TwE0W0W0Rw03W
00460D9F	B2 5D 4E 77 90 C0 5A 77 00 00 00 00 F3 F0 CC 74	WJNwE'Zw....%-lft
00460DA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460DA1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....



Ahi finalizo corectamente, volvemos a aclarar que si no pausamos lso scripts que estan de mas estos se dan cuenta que hay algo raro y va todo mal, asi que a hacer las cosas bien jeje.

Ahora si, dumpeo con el OLLYDMP y abro el IMP REC.

**Import REConstructor v1.6 FINAL (C) 2001-2003 MackT/uCF**

Attach to an Active Process

c:\documents and settings\ricardo\escritorio\unpackme\_execryptor2.2.50.b.exe (00000D1) Pick DLL

Imported Functions Found

- + advapi32.dll FTunk:00060818 NbFunc:5 (decimal:5) valid:YES
- + comctl32.dll FTunk:00060830 NbFunc:2 (decimal:2) valid:YES
- + gdi32.dll FTunk:0006083C NbFunc:4D (decimal:77) valid:YES
- + kernel32.dll FTunk:00060974 NbFunc:8C (decimal:140) valid:YES
- + oleaut32.dll FTunk:00060BA8 NbFunc:10 (decimal:16) valid:YES
- + shell32.dll FTunk:00060BEC NbFunc:3 (decimal:3) valid:YES
- + user32.dll FTunk:00060BFC NbFunc:A3 (decimal:163) valid:YES
- + winmm.dll FTunk:00060E8C NbFunc:1 (decimal:1) valid:YES
- + winspool.drv FTunk:00060E94 NbFunc:8 (decimal:8) valid:YES

Log

rva:00060B5C forwarded from mod:ntdll.dll ord:00D1 name:RtlGetLastWin32Error  
rva:00060B60 forwarded from mod:ntdll.dll ord:0169 name:RtlRestoreLastWin32Error

Current imports:

C (decimal:12) valid module(s). (added: +C (decimal:+12))  
1B9 (decimal:441) imported function(s). (added: +1B9 (decimal:+441))

IAT Infos needed

DEP 000271B0 IAT AutoSearch

RVA 00060818 Size 00000710

New Import Infos (IID+ASCII+LOADER)

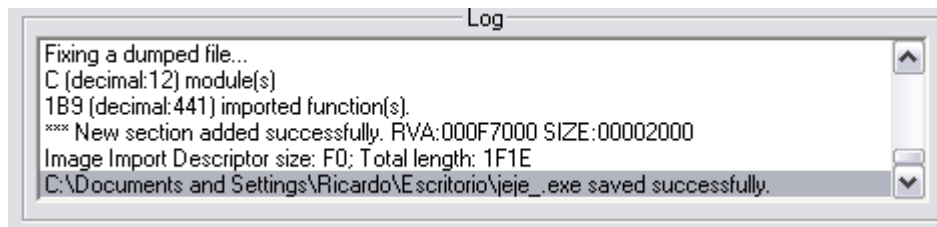
RVA 00000000 Size 00001F1E

☒ Add new section

Load Tree Save Tree Get Imports Fix Dump

Show Invalid Show Suspect Auto Trace Clear Imports Clear Log Options About Exit

Veo que esta todo correcto reparo el dumpeado



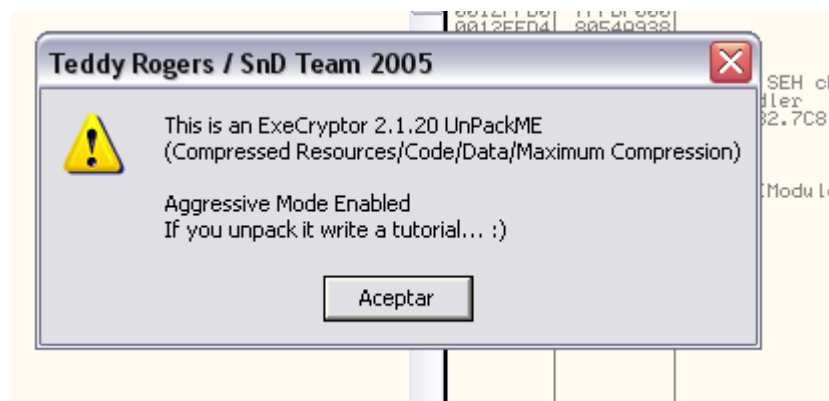
Ahora debo arreglar el TLS, abrimos el dumpeado en el OLLY y en el header buscamos

0040014C	00000000	DD 00000000	Must be 0
00400150	10F10900	DD 0009F110	TLS Table address = 9F110
00400154	18000000	DD 00000018	TLS Table size = 18 (24.)
00400158	00000000	DD 00000000	Load Config Table address = 0
0040015C	00000000	DD 00000000	Load Config Table size = 0
00400160	00000000	DD 00000000	Bound Import Table address = 0

y lo cambiamos a

0040014C	00000000	DD 00000000	Must be 0
00400150	00000000	DD 00000000	TLS Table address = 0
00400154	00000000	DD 00000000	TLS Table size = 0
00400158	00000000	DD 00000000	Load Config Table address = 0
0040015C	00000000	DD 00000000	Load Config Table size = 0
00400160	00000000	DD 00000000	Bound Import Table address = 0

Y guardamos los cambios



Y ya funciona correctamente, jeje la proxima parte seguiremos con el c, otro mas de esta saga vencido

Hasta la parte 57

Ricardo Narvaja  
10/11/06

## INTRODUCCION AL CRACKING CON OLLYDBG PARTE 57

### UNPACKME C:

Seguiremos avanzando en los unpackme de execryptor, el unpackme c es realmente similar al b, lo que dice cuando lo ejecutamos es que mata los file monitors y registry monitors.



Como yo no use ni monitors de filas, ni monitores de registro, realmente es repetir el mismo tute que el b, en este caso la instrucción donde aparece la api en mi maquina estaria aqui.

```
00486DF7  8B45 F4      MOV EAX,DWORD PTR SS:[EBP-C]
00486DFA  E8 61670100 CALL 0049D560      ; UnPackMe.0049D560
00486DFF  5B          POP EBX
00486E00  8B0424      MOV EAX,DWORD PTR SS:[ESP]
00486E03  52          PUSH EDX
```

Por lo tanto la direccion donde debemos poner el HARDWARE BPX ON EXECUTION y modificar el script para ello es **00486DFA** .

El script quedaria asi:

-----  
var tabla  
var contenido

mov tabla,460818

start:

```
cmp tabla,460f28
ja final
cmp [tabla],50000000
ja saltear
```

```
mov contenido,[tabla]
cmp contenido,0
je saltear
log contenido
log tabla
```

```
mov eip,contenido
bphws 486DFA, "x"
mov [486DFA],0
mov [486DFB],0
cob reparar
run
```

```
reparar:
cmp eip,486DFA
jne saltear
log eax
mov [tabla],eax
run
```

```
saltear:
add tabla,4
jmp start
```

```
final:
ret
```

---

y no hay que olvidar antes de correrlo ya que llegamos al OEP con el OLLYBONE, deshabilitar el BREAK ON EXECUTE que usamos para llegar allí, suspender los threads innecesarios y poner el `He ZwTerminateProcess` y ahí si correrlo, reparara toda la tabla perfectamente, y quedara para dumper y usar el IMP REC con los mismos valores que el b.

### UNPACKME D:

Por lo tanto pasemos al unpackme d cuando lo ejecutamos nos muestra la proteccion que trae:



Debug Messages Enabled, o sea que mostrara mensajes cuando detecta el debugger, hmm, veamos en que nos cambia esto el metodo que venimos usando, abramoslo en OLLYDBG y llegamos como siempre al System Breakpoint, y allí borramos los BP que colocho el OLLY, luego colocamos el BREAK ON EXECUTE y llegamos al OEP.

004271B0	55	PUSH EBP	
004271B1	8BEC	MOV EBP,ESP	
004271B3	6A FF	PUSH -1	
004271B5	68 600E4500	PUSH 450E60	
004271B8	68 C8924200	PUSH 4292C8	
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
004271C5	50	PUSH EAX	
004271C6	64:8925 000000	MOV DWORD PTR FS:[0],ESP	
004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	
004271D1	56	PUSH ESI	
004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	UnPackMe.0048AF52
004271DC	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	

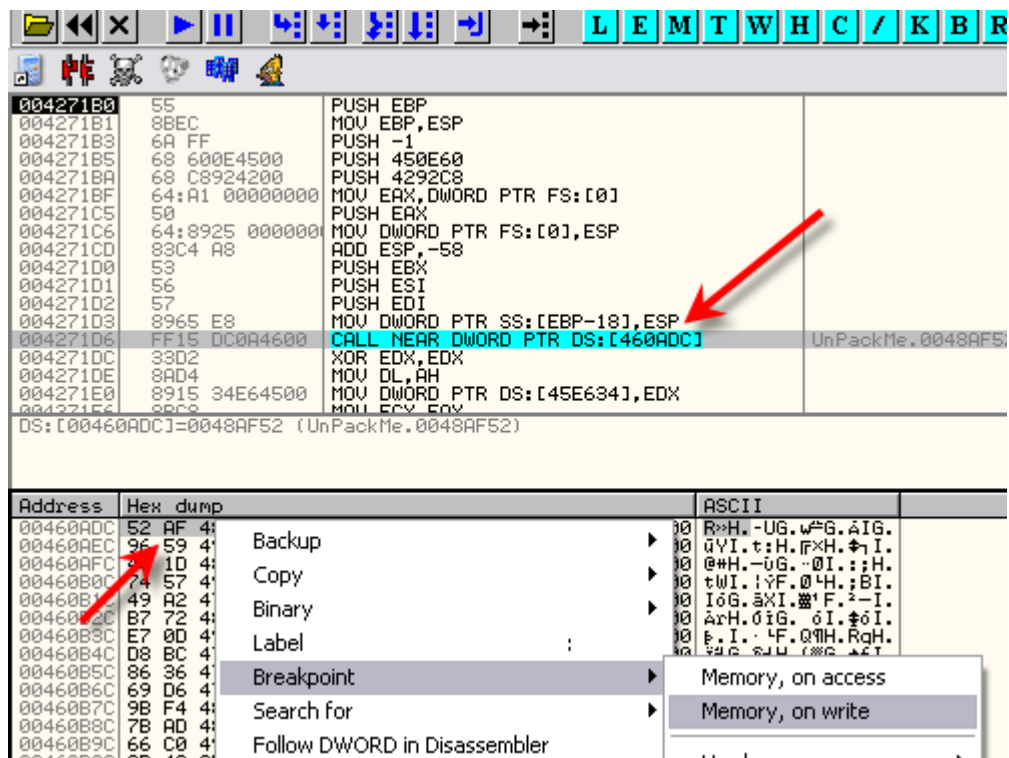
Hasta aquí no hay novedad miremos los Threads para suspenderlos

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000001C4	7C810659	7FFAA000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
000002C4	7C810659	7FFDC000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000428	7C810659	7FFAC000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
000005D4	7C810659	7FFD5000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000608	7C810659	7FFD4000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
0000079C	004F7085	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0468 s	0.0937 s
000007A8	7C810659	7FFD9000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000814	7C810659	7FFAB000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000850	7C810659	7FFD6000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000858	7C810659	7FFAD000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000920	7C810659	7FFDB000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000A7C	7C810659	7FFD8000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000DA4	7C810659	7FFAF000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000E78	7C810659	7FFDA000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000E7C	00270000	7FFDD000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000EA8	7C810659	7FFAE000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000F4C	7C810659	7FFD7000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s

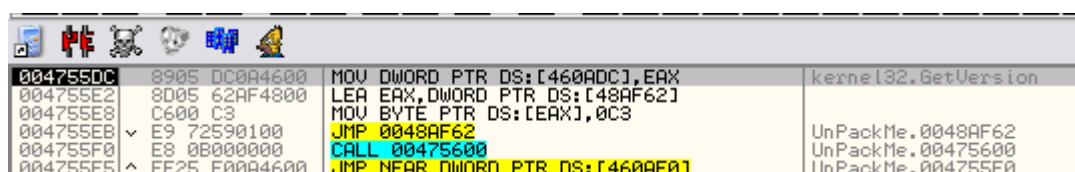
Se ve bastante similar, suspendamos todos menos los 2 que veníamos suspendiendo

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000001C4	7C810659	7FFAA000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000002C4	7C810659	7FFDC000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000428	7C810659	7FFAC000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000005D4	7C810659	7FFD5000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000608	7C810659	7FFD4000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
0000079C	004F7085	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0468 s	0.0937 s
000007A8	7C810659	7FFD9000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000814	7C810659	7FFAB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000850	7C810659	7FFD6000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000858	7C810659	7FFAD000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000920	7C810659	7FFDB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000A7C	7C810659	7FFD8000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000DA4	7C810659	7FFAF000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E78	7C810659	7FFDA000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E7C	00270000	7FFDD000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000EA8	7C810659	7FFAE000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000F4C	7C810659	7FFD7000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s

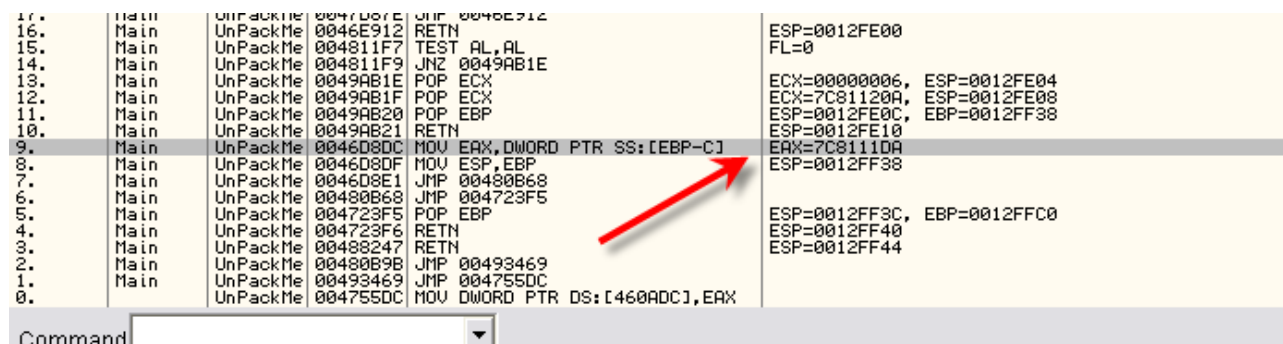
Como la vez anterior, dejo el rojo que es el actual y el que en mi maquina comienza en 270000 y suspendo todo el resto, ahora buscare la primera entrada debajo del OEP y le pondre un BPM ON WRITE alli y pondre a tracear el OLLY.



Y allí paro luego de tracear



Veamos en el trazo unas líneas antes cuando manda a EAX la dirección de la api.



```
mov tabla,460818
```

```
start:  
cmp tabla,460f28  
ja final  
cmp [tabla],50000000  
ja saltar
```

```
mov contenido,[tabla]  
cmp contenido,0  
je saltar  
log contenido  
log tabla
```

```
mov eip,contenido  
bphws 46D8DF, "x"  
mov [46D8DF],0  
mov [46D8DF],0  
cob reparar  
run
```

```
reparar:  
cmp eip,46D8DF  
jne saltar  
log eax  
mov [tabla],eax  
run
```

```
saltar:  
add tabla,4  
jmp start
```

```
final:  
ret
```

---

Bueno reinicio, llego al OEP, quito el break on execute, coloco el He ZwTerminateProcess y a ver si funciona el script.



EAX=0004C70C													
Address	Hex dump												ASCII
00460818	F0	6B	DA	77	1B	76	DA	77	F4	EA	DA	77	-k rw+u rW0 rW0i
00460828	83	78	DA	77	00	00	00	00	CF	65	C3	58	08 03 C4 58
00460838	00	00	00	00	D4	6A	EF	77	66	95	EF	77	...Ej'wfo'wEj'
00460848	F3	AD	EF	77	ED	D9	EF	77	99	8B	EF	77	%i'wY'w0i'wA'
00460858	2A	7D	EF	77	B2	7C	EF	77	77	53	F2	77	*j'wE'wS=wA'F'
00460868	0C	BC	EF	77	52	D4	EF	77	FA	8D	EF	77	.wRE'w.i'wz'
00460878	51	B2	EF	77	26	D5	EF	77	2A	E3	EF	77	Qw&'w*0'w_9c
00460888	71	B4	EF	77	2E	AD	EF	77	E1	61	EF	77	qI'w.i'wBa'w0a'
00460898	CC	D2	EF	77	43	70	EF	77	FB	EA	F0	77	IfE'wCp'w'0-wAa'
004608A8	01	72	F0	77	A9	34	F0	77	D5	93	EF	77	0x-w04-w'0'wh'
004608B8	AA	D2	EF	77	B2	6F	EF	77	3F	38	F2	77	-E'w0o'w?8=wIb'
004608C8	68	E0	EF	77	00	60	EF	77	90	5B	EF	77	h0'w.'wE['wmA'
004608D8	94	6C	F0	77	22	8D	EF	77	3D	C8	F1	77	0l-w''w=ztwm-
004608E8	6F	C0	EF	77	85	7B	EF	77	26	D9	EF	77	oL'wA0'w&'w^'
004608F8	36	8A	EF	77	FC	8A	EF	77	0F	62	EF	77	6e'wF'e'w*b'wI^'
00460908	97	5D	EF	77	1A	9A	EF	77	68	FA	EF	77	uJ'w+u'wk'wC'F'
00460918	DA	98	F2	77	1A	40	F2	77	55	EA	EF	77	rY=w+0=wU0'w+a'
00460928	70	E6	EF	77	F0	81	EF	77	2D	6C	EF	77	py'w-u'w-l'wYn'
00460938	4F	83	EF	77	09	ED	EF	77	EB	AA	EF	77	0a'w.Y'wU-w&i-
00460948	B1	95	EF	77	6F	B0	EF	77	8A	5A	EF	77	00'wo'wEz'wUic
00460958	26	F1	F0	77	C9	DD	F0	77	51	E0	F0	77	%z-wF'-w00-w3i'
00460968	6C	EC	EF	77	29	94	EF	77	00	00	00	00	lY'w)0'w...k0
00460978	D4	A7	80	7C	51	0E	81	7C	EE	1E	80	7C	E0C'Q0u'-'A0i'#/
00460988	40	7A	94	7C	09	2A	81	7C	DA	CD	81	7C	@z0'.*u'F-u'..A0
00460998	15	99	80	7C	F8	0E	81	7C	B6	2B	81	7C	S0C'00u'A+u'00C
004609A8	51	9A	80	7C	E3	14	82	7C	BF	50	83	7C	Q0C'00e'P0a'0i0
004609B8	A8	CC	80	7C	62	2E	86	7C	77	DF	81	7C	0F0'0b.0w'0'0eJ0
004609C8	5B	CF	81	7C	08	2F	81	7C	9D	47	84	7C	[0u'0/00G0'.e0
004609D8	90	A4	80	7C	CF	BC	80	7C	62	D2	80	7C	E0C'00C'0eC'0b00
004609E8	77	D0	80	7C	5E	A3	80	7C	78	34	83	7C	w0C'00C'0x40i'0
004609F8	FD	79	92	7C	D4	05	92	7C	3D	04	92	7C	zY0'E0E'0+0'0'0
00460A08	76	2E	81	7C	8B	B2	85	7C	A9	60	83	7C	v.u'000'0'0'0'0'0
00460A18	77	0A	81	7C	3C	15	81	7C	FE	4F	83	7C	w.u'<0u'0=00i'0
00460A28	23	2C	81	7C	72	67	83	7C	40	97	80	7C	#.u'rg0'00C'0'0
00460A38	C5	9B	80	7C	05	10	91	7C	B9	23	81	7C	+0C'000'0'0'0'0'0
00460A48	B9	4C	83	7C	8A	18	92	7C	9F	2D	81	7C	0L0'0e'0'f-u'0'0
00460A58	FC	38	81	7C	A5	18	82	7C	44	20	83	7C	00u'0+0'0'0'0'0
00460A68	61	23	83	7C	47	9B	80	7C	41	26	81	7C	a00'00C'0A0'0'0'0
00460A78	87	0D	81	7C	0E	18	80	7C	24	1A	80	7C	0.u'00C'0+0'0'0'0
00460A88	FE	DD	80	7C	01	BE	80	7C	0F	AC	80	7C	0'000'00C'0A0'0'0
00460A98	B8	0B	83	7C	A1	BA	80	7C	EB	98	80	7C	000'0'0'0'0'0'0'0
00460AA8	66	E8	80	7C	EC	E7	80	7C	01	9E	80	7C	f0C'00C'00C'0'0'0
00460AB8	74	0D	83	7C	F8	9B	80	7C	D4	A0	80	7C	t.0'0'0'0'0'0'0'0
00460AC8	41	4D	83	7C	28	97	80	7C	19	FF	80	7C	0M0'00C'0'0'0'0'0
00460AD8	CF	B4	80	7C	DA	11	81	7C	C6	97	80	7C	00C'0'0'0'0'0'0'0
00460AE8	AC	17	82	7C	AB	1E	83	7C	EE	86	82	7C	%00'00A'0'0'0'0'0
00460AF8	03	DC	81	7C	11	13	82	7C	8B	B5	81	7C	0'0'0'0'0'0'0'0'0

Vemos que arregla toda la tabla y que no hay diferencias con el anterior así que seguimos con el unpackme “e” a ver si encontramos algo diferente.

## UNPACKME E:



Active Watch enabled eso es lo que nos muestra el unpackme “e” cuando lo corremos veamos que diferencia tiene.

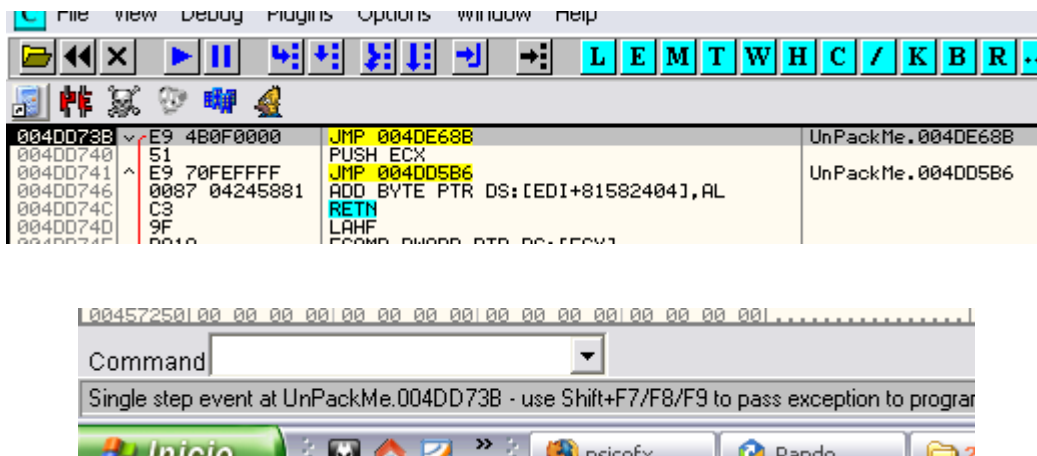
Realmente es similar a los anteriores por lo tanto lo saltaremos con el metodo visto se desempaca perfectamente seguiremos con el “g” saltaremos el f pues parece tambien similar y a lo s mo si tiene alguna novedad la encontraremos en el “g” tambien pues cada uno acumula las novedades de lso anteriores.

## UNPACKME G:

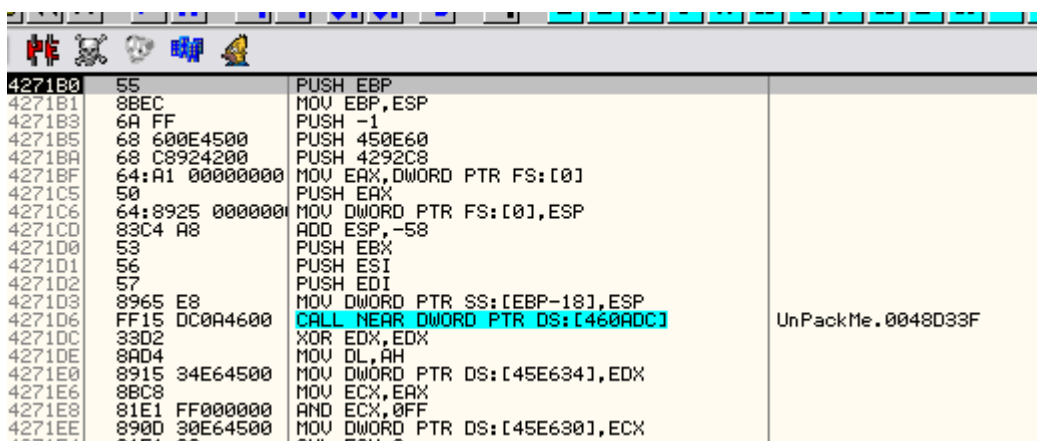


El “g” cuando lo corremos vemos algo que nos puede influir, dice que tiene opcion anti trazo y en nuestro metodo tenemos que trazar para hallar el punto donde poner el HE ON EXECUTION, hagamos este a ver si la opcion antitraceo es efectiva o no.

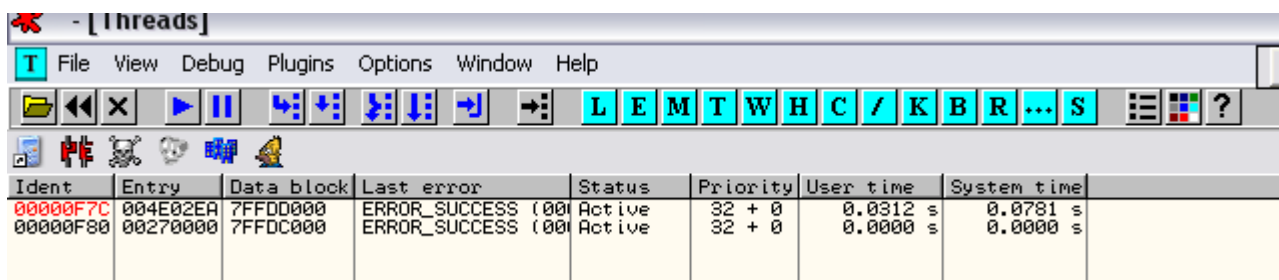
Bueno la primera diferencia es cuando llego al system breakpoint y borro los BP que coloca OLLYDBG, y coloco el BREAK ON EXECUTE para en 5 o 6 Single step exceptions antes de parar en el OEP



El tema es que las debo pasar a mano con SHIFT mas F9 si o si, porque si le coloco la tilde para que saltee la excepcion, no funciona el OLLYBONE, asi que saltamos esas excepciones con SHIFT mas f9 y llegamos al OEP.



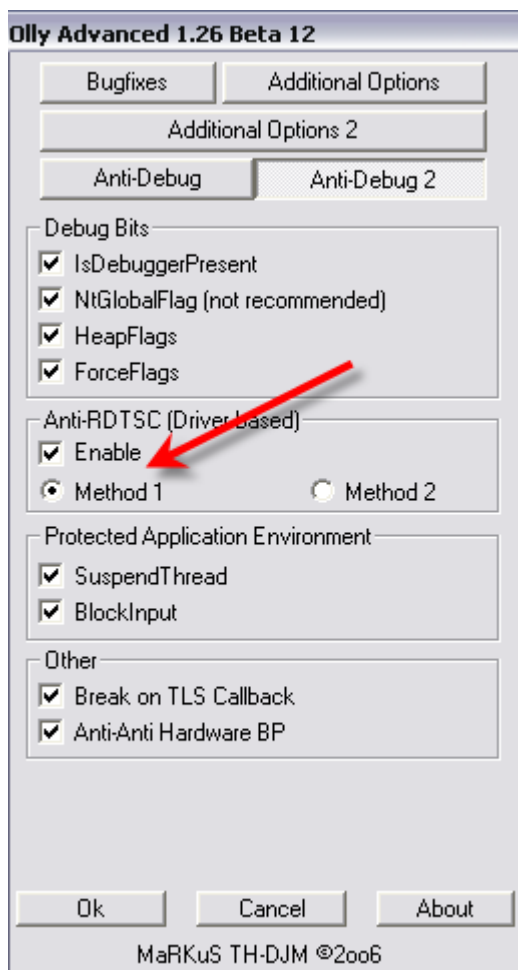
Otro detalle que vemos es que aun no creo threads quizas los vaya creando a medida que va funcionando asi que estaremos atentos al menos ya vemos algunas diferencias.



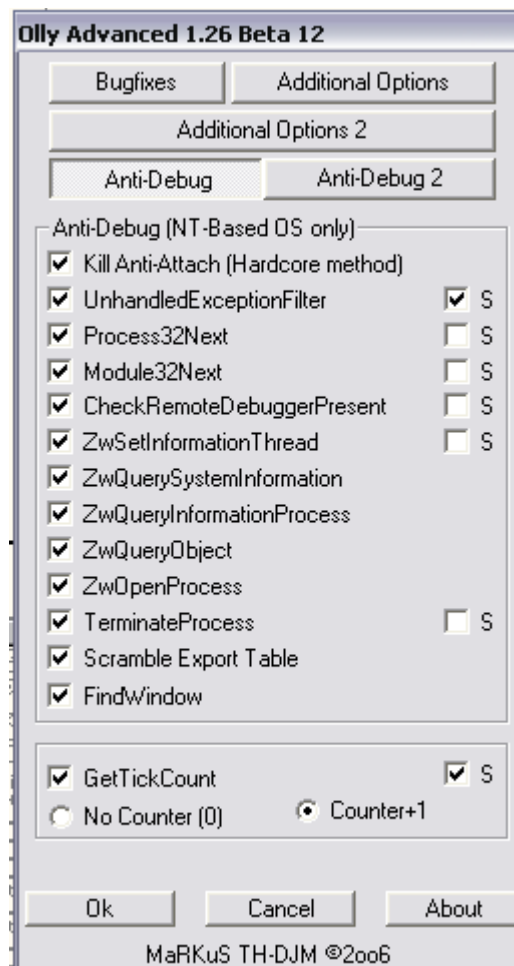
The screenshot shows the 'Threads' window in OllyDbg. It contains a table with the following data:

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
00000F7C	004E02EA	7FFDD000	ERROR_SUCCESS (00000000)	Active	32 + 0	0.0312 s	0.0781 s
00000F80	00270000	7FFDC000	ERROR_SUCCESS (00000000)	Active	32 + 0	0.0000 s	0.0000 s

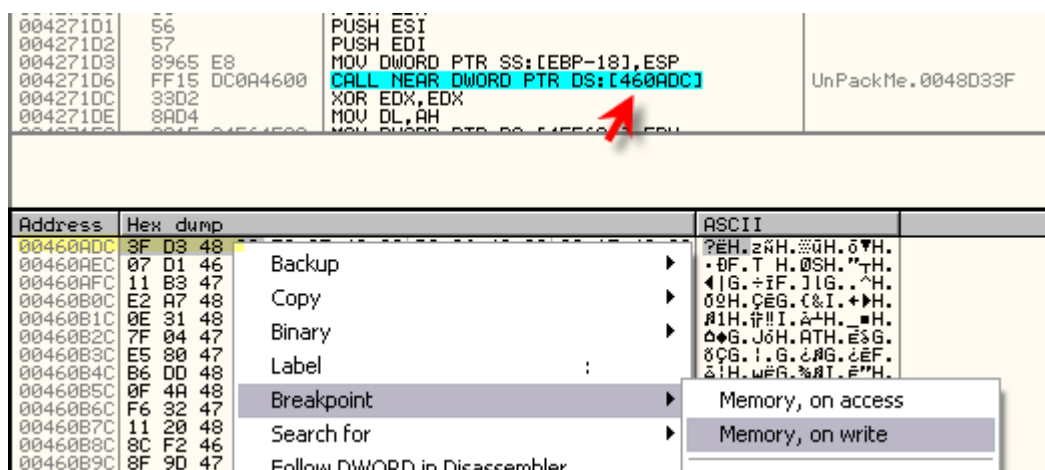
Como vemos solo estan presentes aquí los dos threads necesarios para correr, si hubiera mas ya saben que hay que suspenderlos y bueno veamos que tal es la opcion antitraceo, por si acaso use RDTSC hay que recordar que debi tener activado en el plugin OLLY ADVANCE la opcion ANTI RDTSC y GetTickCount.



Y



Bueno tambien cuando uno necesita si o si de las opciones del ANTI RDTSC, hay que fijarse cuando uno arranca el OLLYDBG porque muchas veces en la esquina inferior donde suele poner la informacion de los breakpoints nos muestra por unos segundos el mensaje COULDN'T START THE DRIVER, en ese caso el driver no se cargo correctamente y no funcionara, muchas veces eso ocurre en mi caso luego de un tiempo de usar la maquina y reiniciando ya carga nuevamente, si no es asi y siempre les sale ese cartel deben revisar bien y tratarde arreglar ese problema.



Le coloco un BPM ON WRITE en la entrada que sabemos que se va a reparar guardadndo alli la direccion de la api, como siempre, quitamos el BREAK ON EXECUTE y ponemos a tracear a ver que pasa.

0047D675	8905 DC0A4600	MOV DWORD PTR DS:[460ADC],EAX	kernel32.GetVersion
0047D67B	E9 ECCE0000	JMP 0048A56C	UnPackMe.0048A56C
0047D680	C1E8 18	SHR EAX,18	
0047D683	87FD	XCHG EBP,EDI	

Para aqui.

Veamos en el traceo si vemos la instrucción que necesitamos un poco mas arriba de esta.

11.	Main	UnPackMe	00478F25	POP EAX	EAX=7C81120H, ESP=0012FE08
10.	Main	UnPackMe	00478F26	POP EBP	ESP=0012FE0C, EBP=0012FF38
9.	Main	UnPackMe	00478F27	RETN	ESP=0012FE10
8.	Main	UnPackMe	00491E65	MOV EAX,DWORD PTR SS:[EBP-C]	EAX=7C81110A
7.	Main	UnPackMe	00491E68	MOV ESP,EBP	ESP=0012FF38
6.	Main	UnPackMe	00491E6A	PUSH 4842D4	ESP=0012FF34
5.	Main	UnPackMe	00491E6F	JMP 00476845	
4.	Main	UnPackMe	00476845	RETN	ESP=0012FF38
3.	Main	UnPackMe	004842D4	POP EBP	ESP=0012FF3C, EBP=0012FFC0
2.	Main	UnPackMe	004842D5	RETN	ESP=0012FF40
1.	Main	UnPackMe	00484BCD	RETN	ESP=0012FF44
0.	Main	UnPackMe	0047D675	MOV DWORD PTR DS:[460ADC],EAX	

Asi que la instrucción buscada es **491E68**

00491E65 8B45 F4 MOV EAX,DWORD PTR SS:[EBP-C] ; kernel32.GetVersion  
00**491E68** 8BE5 MOV ESP,EBP

Hasta aquí no hubo problemas con el traceo, si paso por algun RDTSC el plugin advanced dio cuenta de el perfectamente y los threads se mantuvieron siendo solo 2 en mi caso.

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
00000F7C	004E02EA	7FFDD000	ERROR_SUCCESS (00000000)	Active	32 + 0	0.0937 s	8.6718 s
00000F80	00270000	7FFDC000	ERROR_SUCCESS (00000000)	Paused	32 + 0	0.0000 s	1.3593 s

Bueno ya obtuvimos lo que queriamos ahora debemos ver si funciona el script, primero realicemos el dumpteo y guardemoslo para luego repararlo.

OllyDump - UnPackMe\_Execryptor2.2.50.g.exe

Start Address: 400000

Size: E1000

Dump

Entry Point: E02EA

-> Modify: 271B0

Get EIP as OEP

Cancel

Base of Code: 93000

Base of Data: 4B000

☒ Fix Raw Size & Offset of Dump Image

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	0004A000	00001000	0004A000	00001000	E0000020
.rdata	0000C000	0004B000	0000C000	0004B000	C0000040
.data	00009000	00057000	00009000	00057000	C0000040
u8ajidy	00003000	00060000	00003000	00060000	E0000060
.rsrc	00008000	00063000	00008000	00063000	40000040
txnbnds	00001000	0006B000	00001000	0006B000	C0000040
7vapk...	00027000	0006C000	00027000	0006C000	E0000020
i8kmfeug	0004E000	00093000	0004E000	00093000	E0000060

☐ Rebuild Import

☒ Method1 : Search JMP[API] | CALL[API] in memory image
☐ Method2 : Search DLL & API name string in dumped file

Reemplazamos en el script la dirección donde se coloca el HBP ON EXECUTION

```
var tabla  
var contenido
```

```
mov tabla,460818
```

```
start:  
cmp tabla,460f28  
ja final  
cmp [tabla],50000000  
ja saltar
```

```
mov contenido,[tabla]  
cmp contenido,0  
je saltar  
log contenido  
log tabla
```

```
mov eip,contenido  
bphws 491E68, "x"  
mov [491E68],0  
mov [491E68],0  
cob reparar  
run
```

```
reparar:  
cmp eip,491E68  
jne saltar  
log eax  
mov [tabla],eax  
run
```

```
saltar:  
add tabla,4  
jmp start
```

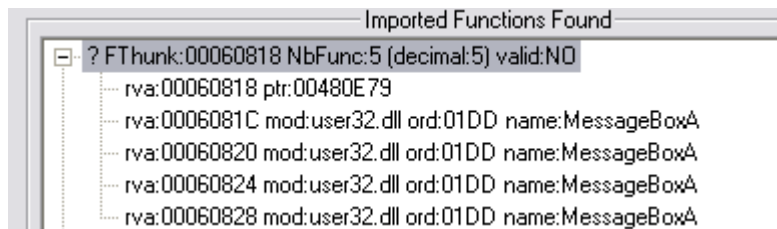
```
final:  
ret
```

---

Bueno coloqué el HE ZwTerminateProcess y corré el script y reparé toda la tabla







Vemos que la primera no la reparo el script y las siguientes las reparo mal investiguemos un poco que paso.

```
mov tabla,460818
```

```
start:
```

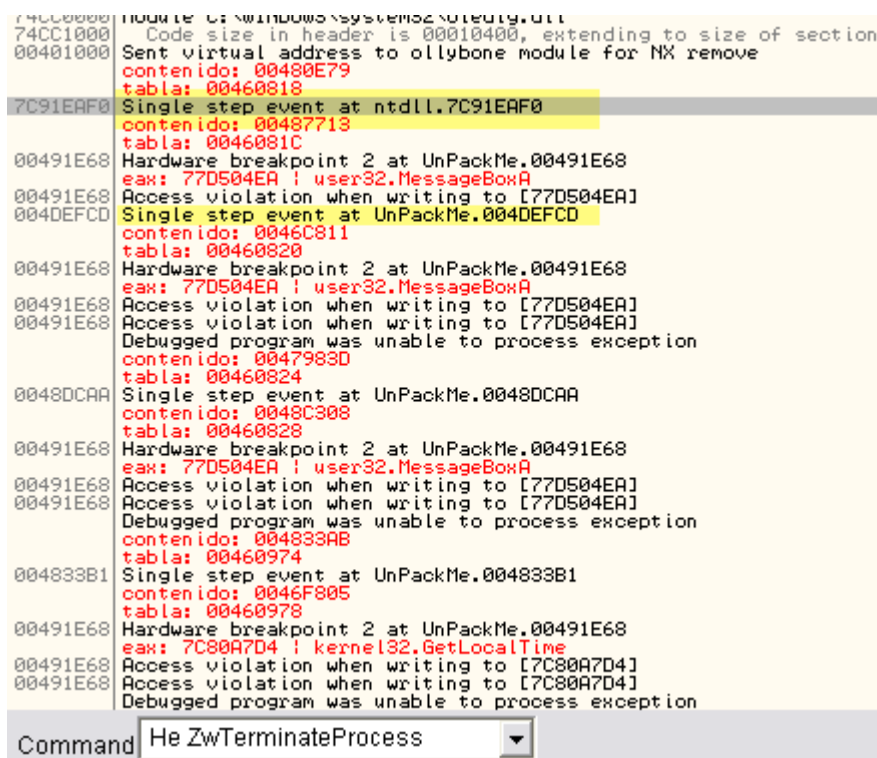
```
cmp tabla,460978
```

```
ja final
```

```
cmp [tabla],50000000
```

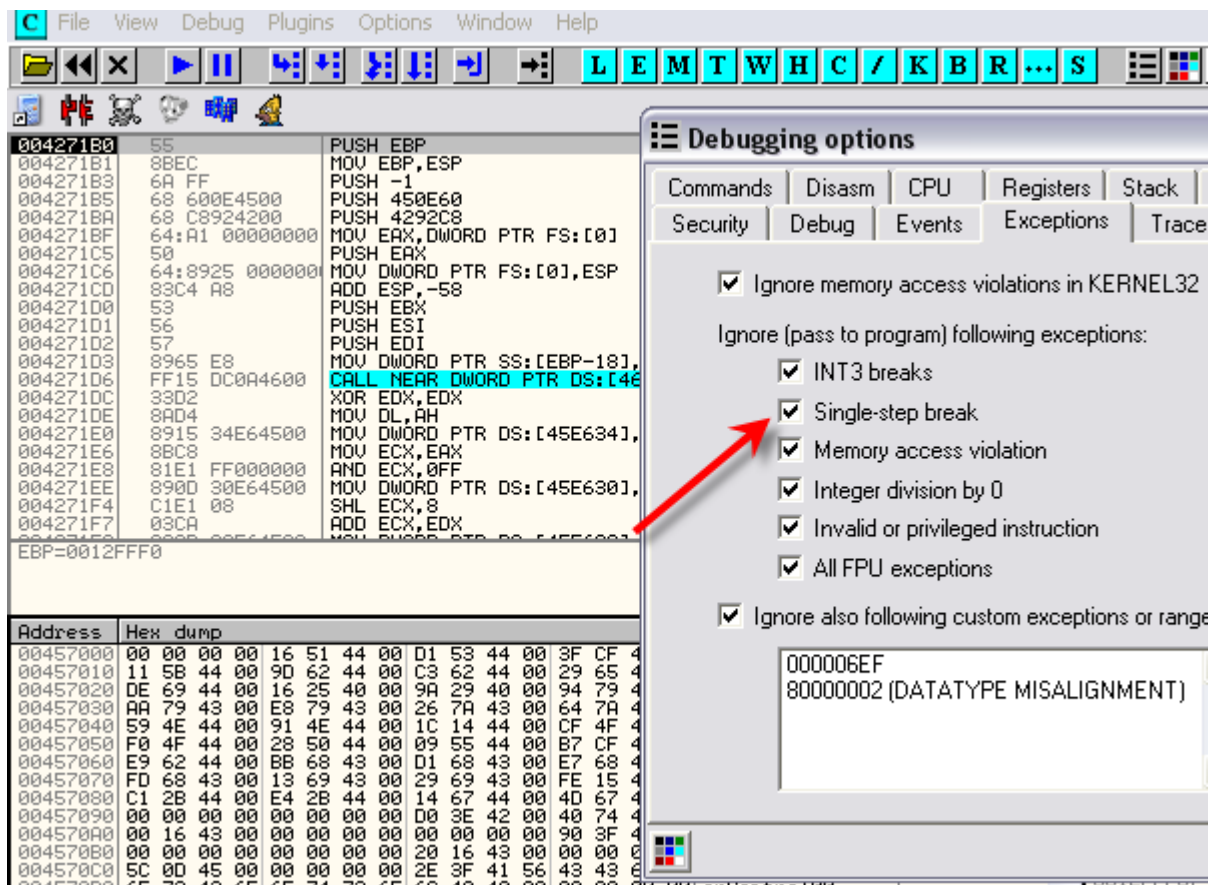
```
ja saltar
```

En el script cambio para que solo repare las primeras entradas a ver si investigo a fondo lo que ocurrio, usando el script y mirando el log.



Veo que hay unas excepciones SINGLE STEP que antes no se producian, asi que vuelvo a llegar al OEP y a repetir el proceso pero esta vez como ya el OLLYBONE no lo utilizo, coloco la tilde en la excepcion SINGLE STEP.





Vuelvo a colocar el HE ZwTerminateProcess y pruebo en las primeras entradas solo a ver si con la tilde trabaja bien.

Address	Hex dump	ASCII
00460818	F0 68 DA 77 18 76 DA 77 F4 EA DA 77 E7 EB DA 77	-k rw+vrw90 rwb0 rw
00460828	83 78 DA 77 00 00 00 00 CF 65 C3 58 08 03 C4 58	ax rw...de-xi-X
00460838	00 00 00 00 04 6A EF 77 66 95 EF 77 89 6A EF 77	...Ej'wfo'wEj'w
00460848	F3 AD EF 77 ED 09 EF 77 99 8B EF 77 C0 85 EF 77	%i'wy'w0i'wLA'w
00460858	2A 7D EF 77 B2 7C EF 77 77 53 F2 77 1E C9 F1 77	*j'wW! 'wWS=wA'F:w
00460868	0C 0C FF 77 C2 04 FF 77 F0 8D FF 77 F1 0D FF 77	#'uRE'w'w'w'w'w

Ahi las reparo correctamente miremos el LOG

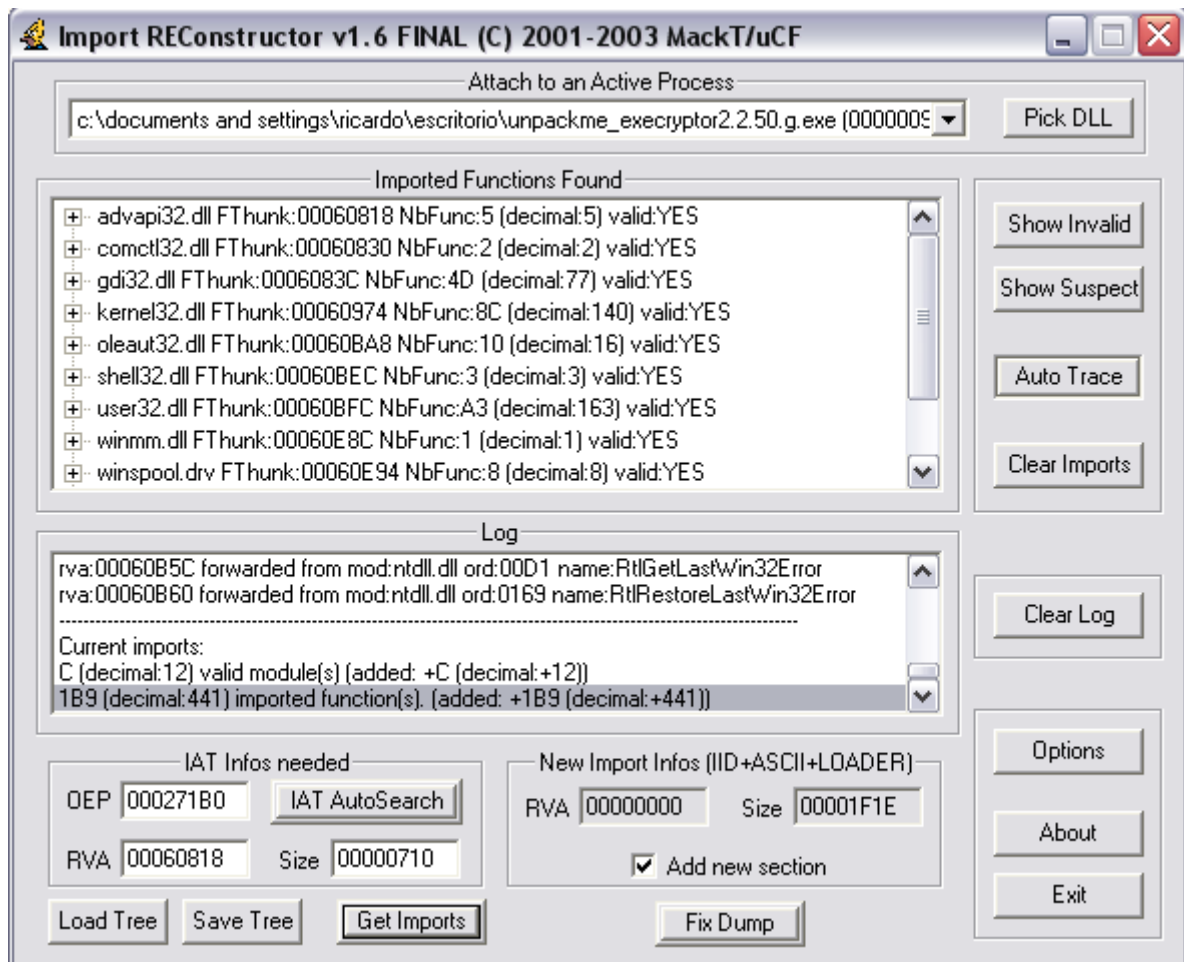
```

74CC0000 module C:\Windows\System32\nteddy.dll
74CC1000 Code size in header is 00010400, extending to size of :
00401000 Sent virtual address to ollybone module for NX remove
contenido: 00480E79
tabla: 00460818
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 77DA6BF0 ! ADVAPI32.RegCloseKey
00491E68 Access violation when writing to [77DA6BF0]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contenido: 00487713
tabla: 0046081C
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 77DA761B ! ADVAPI32.RegOpenKeyExA
00491E68 Access violation when writing to [77DA761B]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contenido: 0046C811
tabla: 00460820
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 77DAEAF4 ! ADVAPI32.RegCreateKeyExA
00491E68 Access violation when writing to [77DAEAF4]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contenido: 0047983D
tabla: 00460824
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 77DAEBE7 ! ADVAPI32.RegSetValueExA
00491E68 Access violation when writing to [77DAEBE7]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contenido: 0048C308
tabla: 00460828
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 77DA7883 ! ADVAPI32.RegQueryValueExA
00491E68 Access violation when writing to [77DA7883]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contenido: 004833AB
tabla: 00460974
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 7C80176B ! kernel32.GetSystemTime
00491E68 Access violation when writing to [7C80176B]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contenido: 0046F805
tabla: 00460978
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 7C80A7D4 ! kernel32.GetLocalTime
00491E68 Access violation when writing to [7C80A7D4]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess

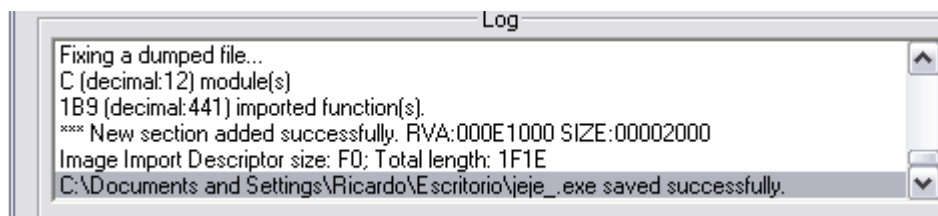
```

Ah ahora vemos las apis correctas y que salta a ZwTerminateProcess como las veces anteriores puedo cambiar en el script la direccion final de la tabla y reparara a continuacion hasta el final, ya que saltara las reparadas.

Vemos que ahora si repara todo correctamente



Reparo todo correctamente, realicemos el FIX DUMP.



Veamos si corre



Perfecto, aquí vamos viendo una simple moraleja ya ven que todos los unpackmes son de la misma versión de execryptor y sin embargo con el método que veníamos usando, la persona que sigue los tutes sin entender lo que estamos haciendo, este último no lo podría haber solucionado y diría, este método no sirve, solo por una tilde en exceptions que este necesita y los anteriores no, ya ven que un mínimo detalle nos hace volcar, y no hemos cambiado ni siquiera de versión del packer, así que hay que razonar cuando uno lee un tute y entender la idea de

lo que esta haciendo el autor, no ponerse con apreto 5 veces f7 luego 4 veces f8 mecanicamente ,porque en packers complejos terminas siempre mal, tienen muchas variantes dentro de una misma version o uno muchas veecs ni sabe realmente que version del packer es y hay que saber acomodarse un poco a ellas y saber remar un poco contra la corriente.

Hasta la parte siguiente donde veremos los 3 finales el h el i y el j de la serie de execryptor y terminaremos con el tema (si podemos jeje)

Ricardo Narvaja  
15/11/06

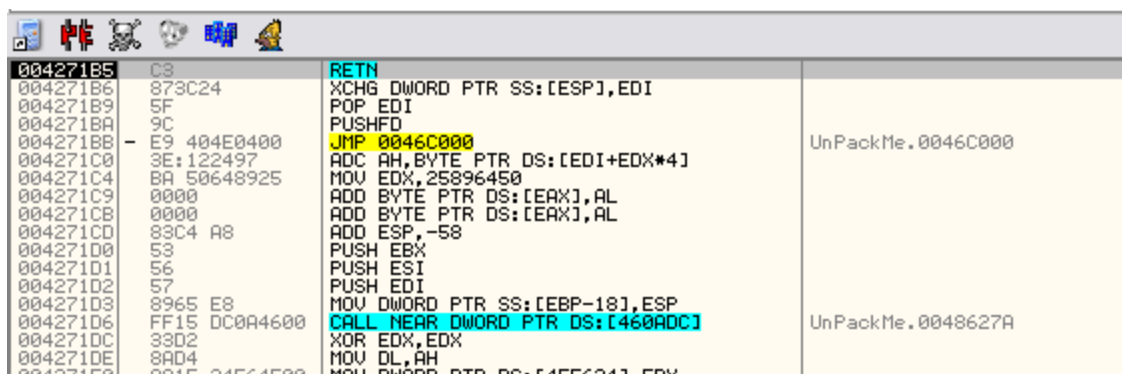
Bueno nos quedan los 3 execryptor mas dificiles vamos a ver si tenemos suerte en resolverlos y si no al menos quedaran en los tutes los intentos que hemos hecho lo cual también enseña ya que execryptor es uno de los packers mas dificiles de hoy día.

Cuando corremos el unpackme “h” vemos que este agrega la siguiente protección

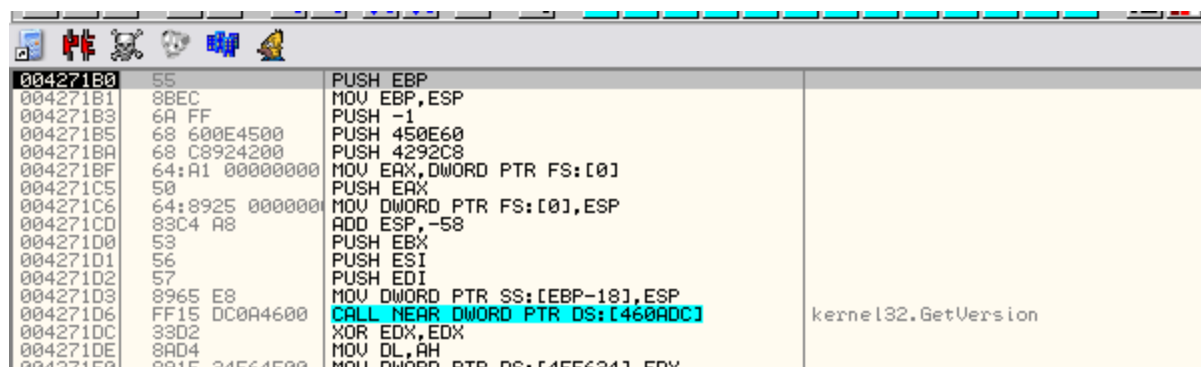


O sea tratara de esconder el entry point y nosotros con la ayuda del desempacado que tenemos trataremos de entender la forma en que lo hace para poder sortear esta protección en cualquier execryptor si tenemos éxito.

Bueno adelante, jeje si utilizamos el método de los anteriores unpackmes con el BREAK ON EXECUTE caemos aquí.



Se veo feo jeje si comparamos con el que esta bien reparado



Realmente se ve espantoso jeje.

Otra foto del original

0042719C	80CC 03	OR AH,3	
0042719F	F7C2 00000400	TEST EDX,40000	
004271A5	74 03	JE SHORT 004271AA	jeje_.004271AA
004271A7	80CC 10	OR AH,10	
004271AA	C3	RETN	
004271AB	90	NOP	
004271AC	90	NOP	
004271AD	90	NOP	
004271AE	90	NOP	
004271AF	90	NOP	
004271B0	55	PUSH EBP	
004271B1	8BEC	MOV EBP,ESP	
004271B3	6A FF	PUSH -1	
004271B5	68 600E4500	PUSH 450E60	
004271BA	68 C8924200	PUSH 4292C8	
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
004271C5	50	PUSH EAX	

y de la misma zona del h

004271A5	74 03	JE SHORT 004271AA	UnPackMe.004271AA
004271A7	80CC 10	OR AH,10	
004271AA	C3	RETN	
004271AB	90	NOP	
004271AC	90	NOP	
004271AD	90	NOP	
004271AE	90	NOP	
004271AF	90	NOP	
004271B0	E9 3A820500	JMP 0047F3EF	UnPackMe.0047F3EF
004271B5	C3	RETN	
004271B6	873C24	XCHG DWORD PTR SS:[ESP],EDI	
004271B9	5F	POP EDI	
004271BA	9C	PUSHFD	
004271BB	E9 404E0400	JMP 0046C000	UnPackMe.0046C000
004271C0	3E:122497	ADC AH,BYTE PTR DS:[EDI+EDX*4]	
004271C4	BA 50648925	MOV EDX,25896450	
004271C9	0000	ADD BYTE PTR DS:[EAX],AL	
004271CB	0000	ADD BYTE PTR DS:[EAX],AL	
004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	

Como se ve la cosa da para diferente jeje.

Comparemos los stacks este es el del unpackme h

0012FFB4	0046D7E0	UnPackMe.0046D7E0
0012FFB8	0047F3EF	UnPackMe.0047F3EF
0012FFBC	00472431	UnPackMe.00472431
0012FFC0	FFFFFFFF	
0012FFC4	7C816FD7	RETURN to kernel32.7C816FD7
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFD9000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	FE0D5260	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C839AA8	SE handler
0012FFE8	7C816FE0	kernel32.7C816FE0
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	004EFE34	UnPackMe.<ModuleEntryPoint>
0012FFFC	00000000	

y este es del desempacado

0012FFC4	7C816FD7	RETURN to kernel32.7C816FD7
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFD9000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	819A0020	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C839AA8	SE handler
0012FFE8	7C816FE0	kernel32.7C816FE0
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	004271B0	jeje_.<ModuleEntryPoint>
0012FFFC	00000000	

Como vemos en el desempacado cuando esta detenido el OEP en mi maquina esta en 12ffc4, en otras maquinas puede variar, pero es el valor que normalmente tiene el stack cuando arrancamos un programa en OLLY y para en su EP, si es un programa empacado, casi siempre luego de desempacarse en el OEP debería salvo trucos raros, estar detenido en el mismo valor de ESP, en el caso de los execryptor como tienen el famoso TLS en el arranque el stack arranca diferente pues tiene rutinas que se ejecutan antes de llegar al EP, pero podemos verificar que la mayoría de los empacados tienen un ESP de arranque en el EP y que cuando llegan al OEP, este valor es el mismo.

En este caso el ESP de arranque seria 12ffc4 en mi maquina, seguramente ustedes pueden hallar el suyo arrancando cualquier programa llegando al EP y fijándose el valor que tiene ESP en ese momento, y ese sera su ESP de arranque.

Luego volviendo al desempacado que esta detenido en su EP nos damos cuenta que luego en el stack si ejecutamos la primera linea se agregaría al mismo el valor de EBP, ya que la primera linea es PUSH EBP.

Al ejecutar el push ebp

The screenshot shows the OllyDbg interface with the assembly window displaying the following code:

```

004271B1 55      PUSH EBP
004271B1 8BEC    MOV EBP,ESP
004271B3 6A FF   PUSH -1
004271B5 68 600E4500 PUSH 450E4500
004271B8 68 C8924200 PUSH 4292C8
004271BF 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
004271C5 50      PUSH EAX
004271C6 64:8925 00000000 MOV DWORD PTR FS:[0],ESP
004271CD 83C4 A8  ADD ESP,-58
004271D0 53      PUSH EBX
004271D1 56      PUSH ESI
004271D2 57      PUSH EDI
004271D3 8965 E8  MOV DWORD PTR SS:[EBP-18],ESP
004271D6 FF15 DC0A4600 CALL NEAR DWORD PTR DS:[468AD0]
004271DC 33D2    XOR EDX,EDX

```

The registers window on the right shows the following values:

```

Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94
EBX 7FFDE000
ESP 0012FFC4
ESI FFFFFFFF
EDI 7C920738
EIP 004271B0

```

The stack window at the bottom shows the following memory dump:

```

Address Hex dump ASCII
00457000 00 00 00 00 16 51 44 00 D1 53 44 00 3F CF 41 00 .....0D.0SD.?0A.
00457010 11 5B 44 00 90 62 44 00 C3 62 44 00 29 65 44 00 4(D.0bD.hbD.)eD.
00457020 DE 69 44 00 16 25 40 00 9A 29 40 00 94 79 43 00 i(D.~%0.u)0.6yC.
00457030 AA 79 43 00 E8 79 43 00 26 7A 43 00 64 7A 43 00 ~yC.6yC.8zC.dzC.
00457040 59 4E 44 00 91 4E 44 00 1C 14 44 00 CF 4F 44 00 VHD.8HD.LND.R0D.

```

se coloca en el stack y quedaría el mismo así

The screenshot shows the OllyDbg interface with the assembly window displaying the following code:

```

004271B1 55      PUSH EBP
004271B1 8BEC    MOV EBP,ESP
004271B3 6A FF   PUSH -1
004271B5 68 600E4500 PUSH 450E4500
004271B8 68 C8924200 PUSH 4292C8
004271BF 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
004271C5 50      PUSH EAX
004271C6 64:8925 00000000 MOV DWORD PTR FS:[0],ESP
004271CD 83C4 A8  ADD ESP,-58
004271D0 53      PUSH EBX
004271D1 56      PUSH ESI
004271D2 57      PUSH EDI
004271D3 8965 E8  MOV DWORD PTR SS:[EBP-18],ESP
004271D6 FF15 DC0A4600 CALL NEAR DWORD PTR DS:[468AD0]
004271DC 33D2    XOR EDX,EDX

```

The registers window on the right shows the following values:

```

Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94
EBX 7FFDE000
ESP 0012FFC0
ESI FFFFFFFF
EDI 7C920738
EIP 004271B1

```

The stack window at the bottom shows the following memory dump:

```

Address Hex dump ASCII
00457000 00 00 00 00 16 51 44 00 D1 53 44 00 3F CF 41 00 .....0D.0SD.?0A.
00457010 11 5B 44 00 90 62 44 00 C3 62 44 00 29 65 44 00 4(D.0bD.hbD.)eD.
00457020 DE 69 44 00 16 25 40 00 9A 29 40 00 94 79 43 00 i(D.~%0.u)0.6yC.
00457030 AA 79 43 00 E8 79 43 00 26 7A 43 00 64 7A 43 00 ~yC.6yC.8zC.dzC.
00457040 59 4E 44 00 91 4E 44 00 1C 14 44 00 CF 4F 44 00 VHD.8HD.LND.R0D.
00457050 F0 4F 44 00 28 59 44 00 09 55 44 00 B7 CF 41 00 -DD.(PD..UD.A0A.
00457060 E9 62 44 00 B8 63 43 00 D1 68 43 00 E7 68 43 00 0bD.7hC.0hC.7hC.
00457070 FD 68 43 00 13 69 43 00 29 69 43 00 FE 15 44 00 ?hC.!!iC.!!iC.wSD.
00457080 C1 2B 44 00 E4 2B 44 00 14 67 44 00 40 67 44 00 +D.8+D.7gD.MgD.
00457090 00 00 00 00 00 00 00 00 D0 3E 42 00 40 74 42 00 .....>B.@tB.
004570A0 00 16 43 00 00 00 00 00 00 00 00 00 3F 42 00 .....C.....e?B.
004570B0 00 00 00 00 00 00 00 00 20 16 43 00 00 00 00 00 .....C.....
004570C0 5C 00 45 00 00 00 00 00 2E 3F 41 56 43 43 6F 6C \.E.....?AUCCol
004570D0 6F 72 43 6F 6E 74 72 6F 6C 40 00 00 00 00 00 orControl@0.....
004570E0 5C 00 45 00 00 00 00 00 2E 3F 41 56 43 43 6F 6C \.E.....?AUCCol
004570F0 6F 72 42 75 74 74 6F 6E 40 00 00 00 00 00 00 orButton@0.....

```

Vemos que el valor de EBP que en mi caso es 12ffc0 ahora lo coloca en el stack, esa es la primera instrucción ejecutada por el programa original, así que miremos en el unpackme h, si esa instrucción ya ha sido ejecutada fijándonos justo arriba de 12ffc4, si ya coloco el 12FFc0 o no.

The screenshot shows a debugger window with three main panes. The top pane displays assembly instructions from address 004271AC to 004271DE. The middle pane shows the state of various registers, including EAX, ECX, EDX, EBP, ESP, ESI, EDI, EIP, C, P, A, Z, S, T, D, O, and EFL. The bottom pane shows the stack memory addresses from 00457000 to 00457120. The registers window shows EBP at 0000004C and ESP at 0000004C. The stack window shows a return address at 00457000.

Miremos solo el stack

The screenshot shows a debugger window with the stack memory displayed. The stack window shows memory addresses from 0012FFB4 to 0012FFFC. The stack contains various values, including 0046D7E0, 0047F3EF, 00472431, FFFFFFFF, 7C816FD7, 7C920738, FFFD9000, 8054A938, FE0D5260, FFFFFFFF, 7C839AA8, 7C816FE0, 00000000, 00000000, 00000000, 004EFE34, and 00000000. The stack is used to store return addresses and other data.

Todos los valores que el programa comience a ejecutar a partir del OEP, deben ir justo encima de 12ffc4 y como vemos allí no esta aun el 12ffc0

Por lo tanto quiere decir que el unpackme emulara a partir de aquí, en una rutina fuera de la sección code las primeras instrucciones del programa, y en vez de hacer PUSH EBP, hará un montón de instrucciones basura que en resumidas cuentas tendrán el mismo efecto que el PUSH EBP.

Aquí realmente el que tiene fiaca podría poner un HE o un BP en esa zona



004271AF	90	NOP	
004271B0	E9 3A820500	JMP 0047F3EF	UnPackMe.0047F3EF
004271B5	C3	RETN	
004271B6	873C24	XCHG DWORD PTR SS:[ESP],EDI	
004271B9	5F	POP EDI	
004271BA	9C	PUSHFD	
004271BB	E9 404E0400	JMP 0046C000	UnPackMe.0046C000
004271C0	3E:122497	ADC AH,BYTE PTR DS:[EDI+EDX*4]	
004271C4	BA 50648925	MOV EDX,25896450	
004271C9	0000	ADD BYTE PTR DS:[EAX],AL	
004271CB	0000	ADD BYTE PTR DS:[EAX],AL	
004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	
004271D1	56	PUSH ESI	
004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	UnPackMe.0048627A
004271DC	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
004271E6	8BC8	MOV ECX,EAX	
004271E8	81E1 FF000000	AND ECX,0FF	
004271EE	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX	
004271F4	C1E1 08	SHL ECX,8	
004271F7	03CA	ADD ECX,EDX	
004271F9	890D 2CF64500	MOV DWORD PTR DS:[45F62C],ECX	

que es donde comienza a ejecutar la zona sin emulación que es similar al original y analizando con muchísima paciencia el stack allí se puede determinar fácilmente las instrucciones que fueron emuladas por ejemplo:

004271B8	E9 404E0400	JMP 0046C000	UnPackMe.0046C000
004271C0	3E:122497	ADC AH,BYTE PTR DS:[EDI+EDX*4]	
004271C4	BA 50648925	MOV EDX,25896450	
004271C9	0000	ADD BYTE PTR DS:[EAX],AL	
004271CB	0000	ADD BYTE PTR DS:[EAX],AL	
004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	
004271D1	56	PUSH ESI	
004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	UnPackMe.0048627A
004271DC	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
004271E6	8BC8	MOV ECX,EAX	
004271E8	81E1 FF000000	AND ECX,0FF	
004271EE	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX	
004271F4	C1E1 08	SHL ECX,8	

Allí paro y si vemos el stack

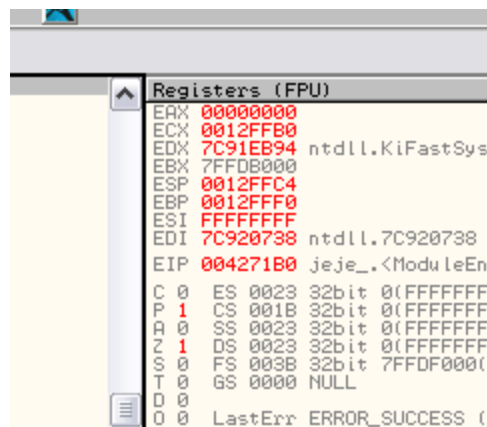
0012FFB0	0012FFE0	Pointer to next SEH record
0012FFB4	004292C8	SE handler
0012FFB8	00450E60	UnPackMe.00450E60
0012FFBC	FFFFFFFF	
0012FFC0	0012FFF0	
0012FFC4	7C816FD7	RETURN to kernel32.7C816FD7
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFD7000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	83A6F660	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C839AA8	SE handler
0012FFE8	7C816FE0	kernel32.7C816FE0
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	004EFE34	UnPackMe.<ModuleEntryPoint>
0012FFFC	00000000	

Vemos que el primer valor agregado al stack es 12fff0 que es el valor inicial de EBP así que por deducción la primera rutina emulada es PUSH EBP, la siguiente hacia arriba es FFFFFFFF que es igual a -1, así que la segunda instrucción que se ejecuto llevando valores al stack es PUSH -1, aunque seguro por conocimiento sabemos que en el medio de ambas hay un MOV EBP,ESP

y así con un poco de paciencia si seguimos subiendo podremos calcular todas las instrucciones iniciales, pero realmente esto no me conforma, dicen los autores de execryptor que su encriptación es imposible y no vi nadie que al menos haya intentado luchar con ella, y buscar el oro en medio de la basura, si lo intentamos y lo logramos nos anotamos un poroto grande jeje,. Crackslatinos siempre al frente luchando contra lo imposible, vamos a intentarlo y si morimos en el intento no nos dirán cobardes, jeje.

Yo creo que en este tipo de rutinas hay 2 tipos, las difíciles, y las dificilísimas, como estoy haciendo el tute aun no tengo ni la menor idea de cual de ambas es, aunque por el hecho que nadie la venció aun, supongo ya que es de las dificilísimas.

Luego de terminar si mi presentimiento se cumple les explicare la diferencia entre ambas, y si nos salio bien podrán entender la diferencia explicarlo ahora es imposible sin ver un poco antes como funciona allí al ver eso, podremos entender la diferencia, retomaremos esto al final.



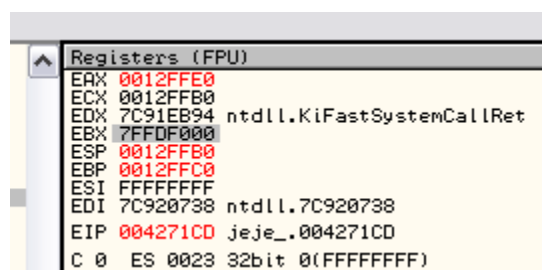
Estos son los registros iniciales de un programa común sin TLS en mi maquina, lo único que varia de uno a otro es el valor de EBX como ya vimos pero igual podemos siempre identificar que se trata del valor de EBX ya que siempre cambia poco puede ser 7FFDB000 o 7FFDF000 o sea que es fácilmente identificable con respecto a los otros registros iniciales, que no tienen valores ni parecidos.

Bueno lleguemos nuevamente a la zona del OEP y veamos que si llegamos nuevamente al

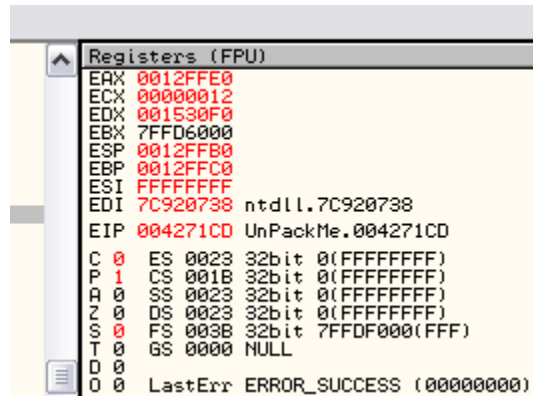
004271CD 83C4 A8 ADD ESP,-58

y comparemos los registros con los del desempacado en la misma posición, realmente

## **DESEMPACADO**

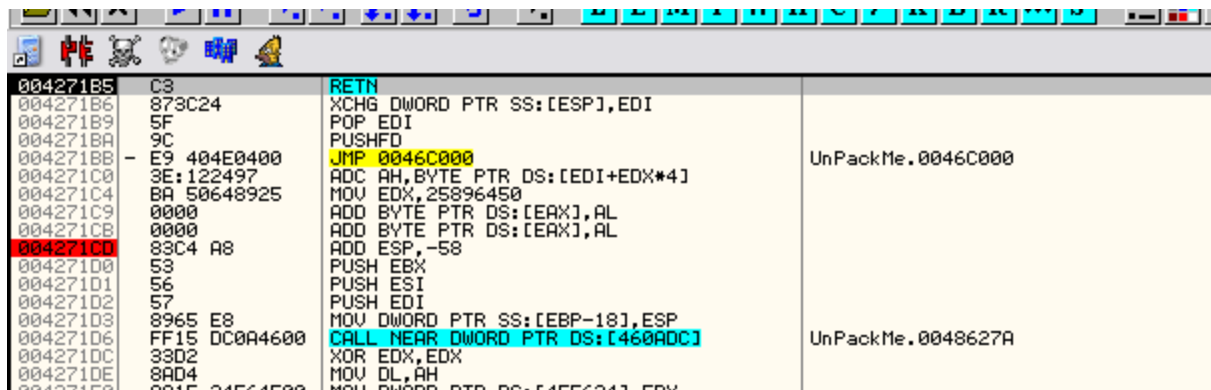


## EMPACADO:

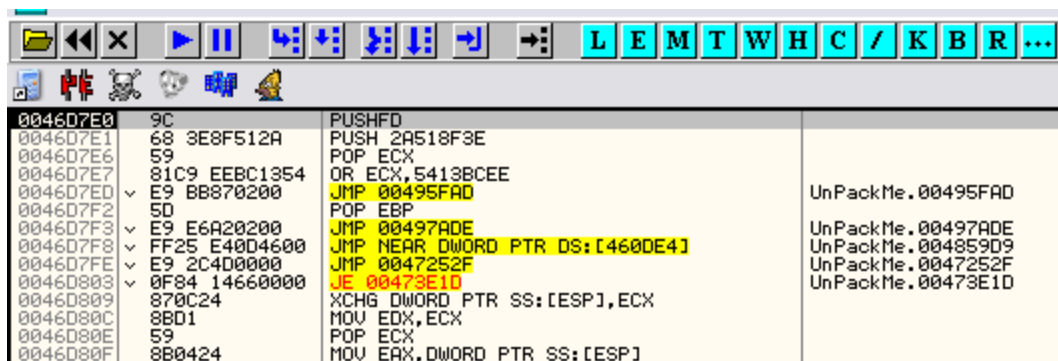


Vemos que luego de las rutinas de emulación ademas de como ya vimos arreglar el stack para ser similar, salvo el valor de ECX y EDX el resto se recupera al mismo valor que el desempacado tiene en esa misma posición, por lo cual debemos estudiar el stack y realmente los valores de los registros que si se restauran o sea EAX,EBX,ESP,EBP,ESI y EDI, obviamente el programa correrá con cualquier valor inicial de ECX y EDX pues no son similares, por lo tanto no interesan.

Bueno volvamos al inicio de la emulación podremos con ella?



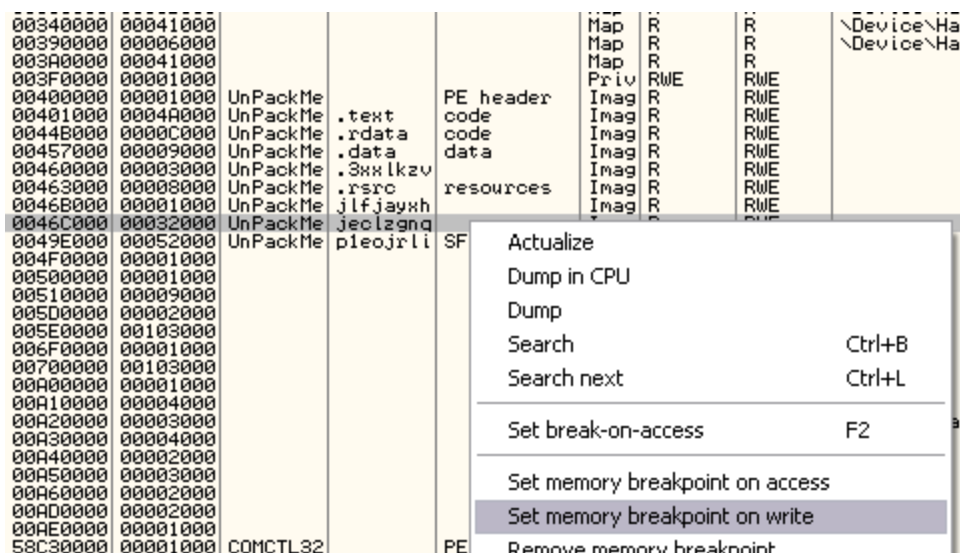
Si la ponemos a tracear antes de llegar al BP llena paginas y paginas de instrucciones basura, saltos al pedo y miles de patrañas si quieren hacer la prueba adelante pero no se los aconsejo jeje, es para amargarse antes de empezar la batalla, por 5 instrucciones de mierda, llena paginas y paginas de instrucciones basura.



Bueno aquí estamos en la entrada al infierno, jeje, tendremos salida?

Lo primero que debemos ver es tratar de establecer en donde la rutina maldita va guardando los valores con que opera, para eso si sirve el traceo también probando algunos BPM ON WRITE en

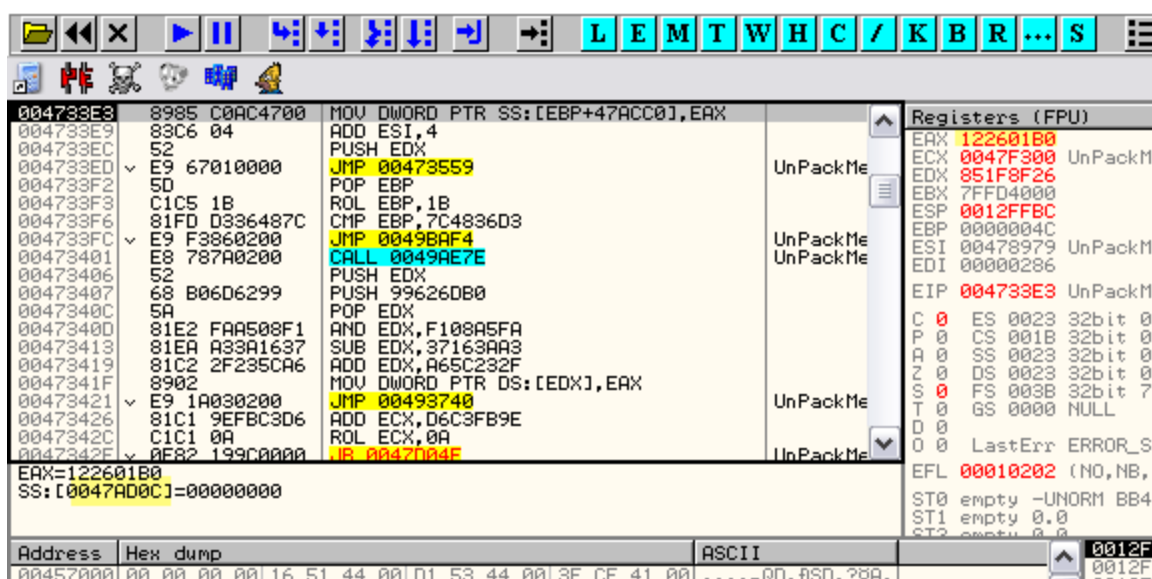
algunas secciones de execryptor enseguida vemos que siempre guarda valores en la misma sección que en este caso es la misma sección donde se esta ejecutando la rutina maldita y tiene sentido, pues generalmente estas rutinas quieren ser los menos dispersadoras de datos posibles y trabajar con todo en la misma sección, al menos en este caso y varios que conozco es así, habrá algunos que no sera así, sera cuestión de buscar donde guarda, pero bueno aquí fácilmente vemos que guarda en la misma sección si ponemos un BPM ON WRITE en la misma sección que se va a ejecutar.



Bueno allí lo colocamos y damos RUN a ver que hace no se asusten que yo estoy temblando jeje.

Agárrense que empiezan las pavadas jeje

Para aquí:



Lo importante es mirar QUE guarda y DONDE lo guarda vemos que en EAX esta un valor que no tiene ningún significado pues no es ninguno de los registros iniciales, ni valores conocidos, y que guarda en 47ad0c, pues haremos una pequeña listita con estos datos, que guarda y donde, nada mas

que eso.

EAX=122601B0 lo guarda en 47ad0c  
EAX=5A731601 “ 4815e0  
EBX= 5A731601 4815e0  
EAX=0046C5DE 47a90C  
00496CBB 01 lo pone a 00  
00496C9C 01 “ a 02  
00496CAB 00 a 01

Mientras hacemos esto de reojo miramos si aparece el 12fff0 en el stack justo donde debería, por ahora aun no apareció, sigamos haciendo la listita.

0048E2C2 8F85 C0A04700 POP DWORD PTR SS:[EBP+47A0C0] ; 0012FFF0

mediante este POP mueve 12fff0 a 47a0CC  
EDI=7C920738 lo guarda en 47a4cc

aquí

0047949B 89BD C0A44700 MOV DWORD PTR SS:[EBP+47A4C0],EDI

Resalto el valor de EDI pues es uno de los valores iniciales, de aquí en mas todo valor conocido lo resaltare.

ESI=0046C5DE lo guarda en 0047A8CC  
EDX=0047F3EF lo guarda en 0047BCF8  
ECX=0047F3EF lo guarda en 0047B8EC  
EAX=0047E97E lo guarda en 0047B4E4  
EBX=7FFDB000 lo guarda en 00481198

aquí

00498D03 899D 8C114800 MOV DWORD PTR SS:[EBP+48118C],EBX

EAX=0012FFC0 lo guarda en 0047B0D8

ojo pues 12ffc0 es el lugar donde debe guardar el siguiente valor en el stack sigamos

EAX=14F43E15 lo guarda en 0047ACCC  
00496CAB 01 lo pone a 00  
00496C9C 02 lo pone a 03  
00496D9A 00 lo pone a 01

Nuevamente el mismo POP que antes pone en

mediante este POP mueve 12fff0 a 0047A488  
EDI=7C920738 lo guarda en 0047A888

Aun después de todo este baile de valores sigue sin aparecer en 12ffc0 el valor que corresponde a PUSH EBP o sea 12fff0.

ESI=FFFFFFFF lo mueve a 0047AC88

aquí

```
00470F82 89B5 C0A84700 MOV DWORD PTR SS:[EBP+47A8C0],ESI
```

EDX=0047F3EF lo guarda en 0047C0B4

ECX=0047F3EF lo guarda en 0047BCA8

EAX=0047E97E lo guarda en 0047B8A0

EBX=7FFDB000 lo guarda en 00481554

en la misma instrucción que antes también guardo EBX aquí

```
00498D03 899D 8C114800 MOV DWORD PTR SS:[EBP+48118C],EBX
```

Sigamos con mucha paciencia solo estamos logueando lo que guarda entre miles de instrucciones basura y aun no aparece la primera instrucción real del programa ejecutada uff, paciencia adelante.

EAX=0012FFC4 lo guarda en 0047B494

aquí

```
00498D0B 8985 CCB04700 MOV DWORD PTR SS:[EBP+47B0CC],EAX
```

el resaltado es el valor inicial de ESP también es importante

sigamos

EAX=1E500000 lo guarda en 0047B088

```
0047B494 C4 FF 12 00
```

allí esta 12ffc4 que es el valor inicial de ESP y ahora le resta 4, que es lo que ocurre cuando hay un push el valor de ESP disminuye en 4, jeje.

```
0047711C 83AD CCB04700 0>SUB DWORD PTR SS:[EBP+47B0CC],4
```

O sea que en

```
0047B494 C0 FF 12 00      Ày#.
```

Ahora queda 12ffc0

00496D9A 01 lo pone a 00

00496C9C 03 lo pone a 04  
00496D8A 00 lo pone a 01

Otra vez la misma sentencia POP

0048E2C2 8F85 C0A04700 POP DWORD PTR SS:[EBP+47A0C0] ; 0012FFF0

que guarda en 0047A448 de nuevo 12fff0

Recordemos que aun no ejecuto ni usa sola instrucción real del programa jeje

EDI=7C920738 lo guarda en 0047A848

nuevamente en

0047949B 89BD C0A44700 MOV DWORD PTR SS:[EBP+47A4C0],EDI ;  
ntdll.7C920738

ESI=FFFFFFFF lo guarda en 0047AC48

en la misma instrucción que guardo ESI antes

00470F82 89B5 C0A84700 MOV DWORD PTR SS:[EBP+47A8C0],ESI

Paciencia adelante

EDX=0047F3EF lo guarda en 0047C074  
ECX=0047F3EF lo guarda en 0047BC68  
EAX=0047E97E lo guarda en 0047B860

EBX= 00478304 lo guarda en 00481514

aquí

00498D03 899D 8C114800 MOV DWORD PTR SS:[EBP+48118C],EBX ;  
UnPackMe.0047830

EAX=0012FFBC lo guarda 0047B454

ojo que 12FFbc es el segundo lugar del stack a rellenar

EAX=192082C0 lo guarda en 0047B048

vemos que comienza a repetirse el ciclo y realmente no vimos acción aun jeje sigamos

EAX=0048F082 lo guarda en 0048191C  
EBX=0048F082 lo guarda en 0048191C

00496D8A 01 lo cambia a 00  
00496C9C 04 a 05  
00496CFB 00 a 01

Con el POP nuevamente

0048E2C2 8F85 C0A04700 POP DWORD PTR SS:[EBP+47A0C0] ; 0012FFF0

mueve a 0047A20C el valor 12fff0.

EDI=7C920738 lo mueve a 0047A60C

en la misma instrucción que marcamos antes que guardaba EDI

ESI=FFFFFFFF lo mueve a 0047AA0C

Aun en el stack ni se ejecuto la primera instrucción real grrr

EDX= 0047F3EF lo mueve a 0047BE38  
ECX=0047F3EF a 0047BA2C  
EAX=0047E97E . a 0047B624

Bueno me convertirán en SAN Ricardo después de esto jeje

EBX=7FFDB000 lo mueve a .004812D8

nuevamente aquí

00498D03 899D 8C114800 MOV DWORD PTR SS:[EBP+48118C],EBX

EAX=0012FFC0 lo mueve a 0047B218 .

Observamos porque somos muy sagaces que la vez anterior que ejecuto el ciclo EAX valía 12ffc4 y ahora 12ffc0 en este momento, esos detalles son los que hay que ir viendo.

EAX=1B700602 lo mueve a 0047AE0C  
EAX=FFFFFFFF lo mueve a 0047B624 7E E9 47 00 ~ég.

Vemos que mueve a una dirección donde esta guardado el valor superior del stack y lo reemplaza por FFFFFFFF

00496CFB 01 lo pone a 00  
00472B9C 00 lo pone a 01  
00472B9C 01 00  
00496C9C 05 06  
00496CEB 00 01

Con el POP nuevamente

guarda allí 0047A1CC el valor 12fff0

Luego



EDI=7C920738 lo guarda en 0047A5CC  
ESI=FFFFFFFF lo guarda en 0047A9CC

EDX=00172CF0 lo guarda en 0047BDF8  
ECX=00000012 lo guarda en 0047B9EC  
EAX= 00472B00 lo guarda en 0047B5E4

El gol de la protección es cansar al que la esta tratando de entender pero yo soy muy cabeza dura adelante jeje, aun ni siquiera ejecuto la primera instrucción real

EBX= 7FFDB000 lo guarda en 00481298  
EAX= 0012FFC4 lo guarda en 0047B1D8

EAX=18000500 lo guarda en 0047ADCC

y al fin como puse un hardware breakpoint on write en 12ffc0 veo donde guarda el valor del PUSH EBP

004923C8 871C24 XCHG DWORD PTR SS:[ESP],EBX

The screenshot shows a debugger window with the following components:

- Assembly List:** Displays instructions at addresses 004923B9 to 004923D9. The instruction at 004923C8 is highlighted: `XCHG DWORD PTR SS:[ESP],EBX`. A red arrow points to this instruction.
- Registers (MMX):** A window on the right showing the current values of registers. EBP is 0012FFC0. A red arrow points to this register.
- Hex Dump:** A window at the bottom showing memory contents. The address 0047A1CC is highlighted, and the hex dump shows the value `FF 12 00 00 00 00 00 00`. A red arrow points to this address.

Como vemos unas lineas antes de guardar el 12fff0 lo lee de 47a1cc de allí saca el valor

Si miramos unas lineas atrás vemos que decía

Con el POP nuevamente

guarda allí 0047A1CC el valor 12fff0

o sea que le sentencia esa POP es muy importante pues es la que acomodo el valor de EBP que luego fue hecho PUSH por la instrucción XCHG donde estamos ahora.

De a poquito vamos encontrando ciertos patrones entre la maleza jeje

la siguiente instrucción real a ejecutarse es

```
004271B1 8BEC      MOV EBP,ESP
```

ESP vale ahora 12ffc0 y por lo tanto al moverlo quedaría EBP con el mismo valor 12FFc0, esto se puede verificar fácilmente ejecutando las primeras instrucciones del desempacado en el cual vemos como se mueven los valores del programa real, para luego ver como los emula aquí en la rutina lacra.

Vemos que va a actualizar restándole 4 a

```
0047711C 83AD CCB04700 0>SUB DWORD PTR SS:[EBP+47B0CC],4
```

```
0047B1D8 C4 FF 12 00      Äÿ#.
```

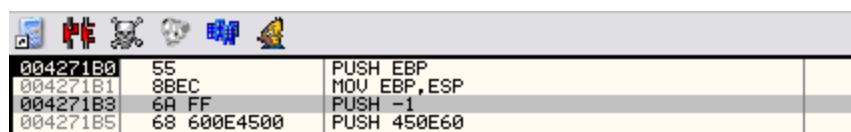
Y quedara en 12ffc0 por lo tanto podemos suponer que allí en 47b1d8 salvo error u omisión se guarda el valor de EBP, y ahora de esta forma haría el MOV EBP,ESP, la verdad no estoy seguro aun pero es una posibilidad, otra es que marque ESP y lo este actualizando pues ya veremos, por lo menos dejamos marcado lo que hace.

Queda

```
0047B1D8 C0 FF 12 00      Àÿ#.
```

Que como dijimos podría ser el valor de EBP o de ESP ya que la segunda instrucción los hace iguales.

Bueno por si acaso para que pare cuando guarde la siguiente instrucción coloco un HARDWARE BPX ON WRITE en el stack en la siguiente posición donde cambiara o sea 12FFBC allí al ejecutar un PUSH -1, guardara FFFFFFFF que es similar a -1.



004271B0	55	PUSH EBP
004271B1	8BEC	MOV EBP,ESP
004271B3	6A FF	PUSH -1
004271B5	68 600E4500	PUSH 450E60

Allí vemos las primeras instrucciones en el desempacado, que esta lacra esta emulando.

Bueno sigamos viendo donde guarda las cosas a ver si hallamos un patrón repetitivo y donde guarda cada cosa.

```
00496CEB 01      pone a 00
00496C9C 06      a 07
00496CDB 00      a 01
```

con el POP repite lo mismo que antes

0048E2C2 8F85 C0A04700 POP DWORD PTR SS:[EBP+47A0C0] ; 0012FFF0

0047A18C F0 FF 12 00 pone 12FFF0 allí

sigamos

mueve EDI=7C920738 a 0047A58C

mueve ESI=FFFFFFFF a 0047A98C

EDX=00172CF0 a 0047BDB8

ECX=00000012 a 0047B9AC

EAX= 00472B00 a 0047B5A4

Como conclusión que se me va ocurriendo mientras voy recopilando información, es algo curioso, la mayoría de los programas que emulan instrucciones casi siempre determinan un lugar para cada registro en la memoria, hay un lugar definido para EDI, otro para ESI y así, acá vemos que siempre vuelve a guardar en nuevos lugares vacíos, puede ocurrir dos cosas o bien hace eso porque es solo la emulación del OEP y tiene ya lugar asignado para meter tantos valores, o bien aun no comenzó a mostrar los lugares finales donde guarda cada cosa definitivamente, porque es cierto que esta emulando 5 líneas, pero en la versión final de execryptor se emulan muchísimas rutinas completas del programa y no creo que cada vez que guarda un valor lo haga siempre en lugares nuevos, no alcanzaría la memoria para guardar en lugares diferentes cada vez, creo que debería llegar un punto donde deberíamos localizar donde guarda cada registro en forma firme, al menos eso he visto yo en la mayoría de los emuladores que analice, si no verdaderamente estaremos antes una protección asombrosa, sigamos adelante recabando información.

EBX=7FFDB000 lo guarda en 00481258

EAX=12FFc0 lo guarda en 0047B198

empezamos de nuevo el ciclo

EAX=1590F683 lo guarda en 0047AD8C

EAX=004820F6 lo guarda en 00481660

EBX=004820F6 lo guarda en 00481660

Por lo demás como tenemos un HARDWARE BPX en el stack vemos que para mucho escribiendo constantes como acá

00478520 68 7F354700 PUSH 47357F

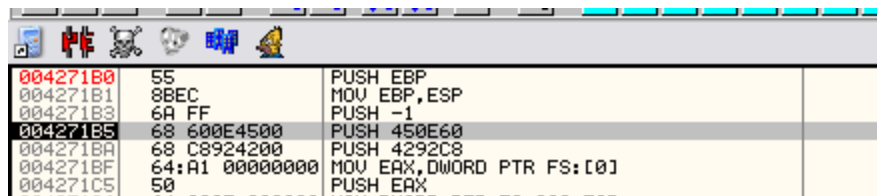
realmente sabemos que eso es basura el programa no puede actualizar y emular una instrucción guardando valores constantes, así que eso yo ni lo menciono se supone que no tiene importancia y es una mas de tanta basura acumulada (no pasara el basurero?), jeje

00496CDB 01 lo pone a 00

Bueno cuando abro la bocata ya sale atrás a desdecirme jeje

00495F2D 6A FF **PUSH -1**

guarda FFFFFFFF en 12FFBC jeje que lacra que es esto se ve que tenia guardada la instrucción maledetto jeje.



004271B0	55	PUSH EBP
004271B1	8BEC	MOV EBP, ESP
004271B3	6A FF	PUSH -1
004271B5	68 600E4500	PUSH 450E60
004271B8	68 C8924200	PUSH 4292C8
004271BF	64:A1 00000000	MOV EAX, DWORD PTR FS:[0]
004271C5	50	PUSH_EAX

Bueno ahora vendrian los dos push siguientes guardar 450e60 en 12FFb8 y 4292c8 en 12ffb4 así que pongo otro hardware bpx on write en 12ffb8.

00496C9C 07 lo cambia a 08

00496CCB 00 lo cambia a 01

luego otra vez el POP

0048E2C2 8F85 C0A04700 POP DWORD PTR SS:[EBP+47A0C0] ; 0012FFC0

Por si no se dieron cuenta cada vez que ejecuta un POP de estos lo marco en azul para si uno mira el tute buscar la repetición de cada ciclo entre ellos.

Realmente pareciera que cada vez que ejecuta el POP actualiza el valor de EBP en este caso guarda 12FFc0 que es el valor actual de EBP y si recordamos cuando emulo el primer PUSH EBP, antes había mediante un pop de estos guardado el 12FFF0 que luego mando al stack, así que siempre hay que estar atento, aunque terminemos en un zanjón jeje.

Es bueno ir sacando conclusiones a medida que uno va realizando la recopilación de información , como en este caso, yo pienso que siempre actualiza EBP, la siguiente vez si no llega a coincidir pues descarto eso, pero bueno vamos marcando las posibilidades, aparentemente esta emulación parece ser mas bien que guardar los registros en lugares fijos, cambiar los lugares pero siempre actualizar cada registro en la misma instrucción, o sea parecería ser que aquí lo importante es la instrucción que hace cada cosa y no donde los guarda, veremos si es así.

O sea en la sentencia POP es importante saber que ella actualiza EBP y no donde lo guarda pues siempre guarda en lugares diferentes, dado que la mayoría siempre busca un patrón y ver si puede hallar una posición de memoria para cada registro.

Si llegamos descubrir la forma que trabaja la emulación luego también nos servirá para la parte en que emula el resto del programa por eso tengo tanta paciencia y trato de ver la punta del hilo, se ve que la emulación es buena, la mayoría de las emulación guarda en 30 bytes contiguos los valores de cada registro y los ves todos continuados ahí (como en el CONTEXT) y se actualizan en el mismo lugar, con lo cual es fácil determinar que instrucción esta ejecutando al ver como cambian, aquí la cosa pinta mas oscura.

EDI=**7C920738** lo mueve a 0047A54C

este es el ejemplo mas visible EDI lo mueve a un lugar vacío siempre en la misma instrucción o sea aquí

0047949B 89BD C0A44700 MOV DWORD PTR SS:[EBP+47A4C0],EDI ;  
ntdll.7C920738

O sea que en cada ciclo esa rutina es la que muestra el valor actual de EDI, lo mismo hace ahora con ESI.

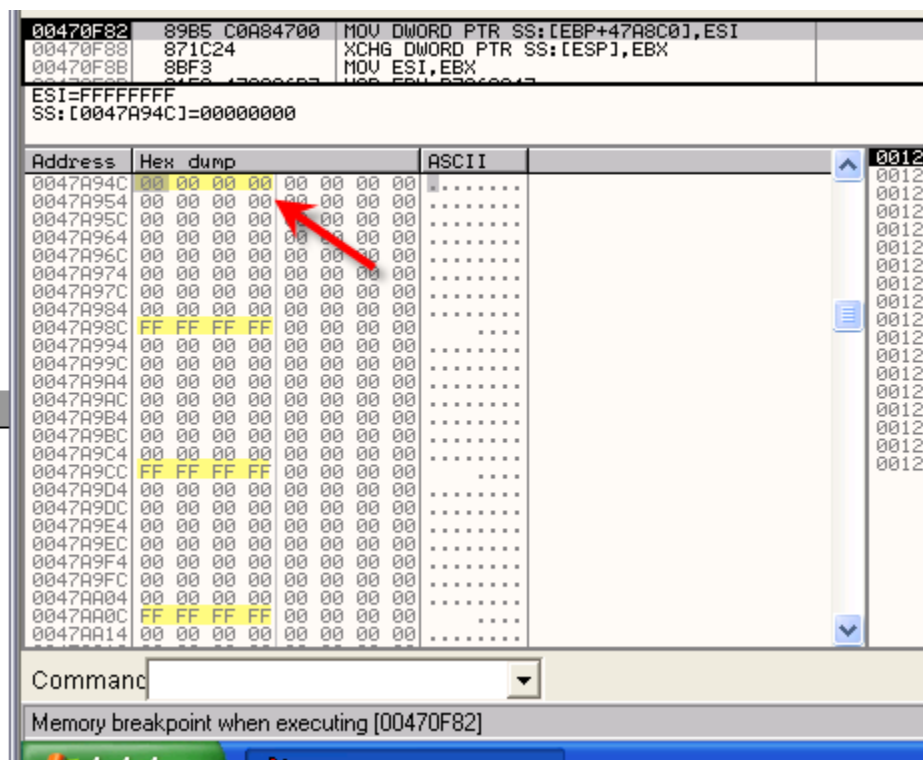
ESI=FFFFFF lo mueve a 0047A94C

que es un lugar vacío pero en la misma instrucción que siempre

00470F82 89B5 C0A84700 MOV DWORD PTR SS:[EBP+47A8C0],ESI

por supuesto esto en medio de miles de saltos, push al pedo, y vueltas para confundir, pero estamos tratando de encontrar la punta del ovillo, el esfuerzo lo estamos haciendo esperemos terminar bien jeje.

No nos olvidemos que si uno encuentra que el método es ese, puede hacer un script donde uno averigüe mirando rápidamente las instrucciones donde actualiza cada registro y de esa forma puede correr un script donde puede ir logueando el estado de los registros originales en cada ciclo, lo cual ayuda muchísimo a interpretar que esta haciendo el programa.



También vemos que siempre que actualiza un registro a un lugar vacío, se ve como un patrón aquí en la imagen, se ve las veces que se actualizo ESI antes, y que están formado un cierto dibujo simétrico, ahora se actualizara donde va la flecha.

EDX=00172CF0 lo mueve a 0047BD78

ECX=00000012 lo mueve a 0047B96C

Repito que siempre que guarda un valor en un lugar vacío se ve lo mismo en este caso el 12 que acaba de guardar

SS: [0047B564]=00000000

Address	Hex dump	ASCII
0047B96C	12 00 00 00 00 00 00 00	.....
0047B974	00 00 00 00 00 00 00 00	.....
0047B97C	00 00 00 00 00 00 00 00	.....
0047B984	00 00 00 00 00 00 00 00	.....
0047B98C	00 00 00 00 00 00 00 00	.....
0047B994	00 00 00 00 00 00 00 00	.....
0047B99C	00 00 00 00 00 00 00 00	.....
0047B9A4	00 00 00 00 00 00 00 00	.....
0047B9AC	12 00 00 00 00 00 00 00	.....
0047B9B4	00 00 00 00 00 00 00 00	.....
0047B9BC	00 00 00 00 00 00 00 00	.....
0047B9C4	00 00 00 00 00 00 00 00	.....
0047B9CC	00 00 00 00 00 00 00 00	.....
0047B9D4	00 00 00 00 00 00 00 00	.....
0047B9DC	00 00 00 00 00 00 00 00	.....
0047B9E4	00 00 00 00 00 00 00 00	.....
0047B9EC	12 00 00 00 00 00 00 00	.....
0047B9F4	00 00 00 00 00 00 00 00	.....
0047B9FC	00 00 00 00 00 00 00 00	.....
0047BA04	00 00 00 00 00 00 00 00	.....
0047BA0C	00 00 00 00 00 00 00 00	.....
0047BA14	00 00 00 00 00 00 00 00	.....
0047BA1C	00 00 00 00 00 00 00 00	.....
0047BA24	00 00 00 00 00 00 00 00	.....
0047BA2C	EF F3 47 00 00 00 00 00	.....
0047BA34	00 00 00 00 00 00 00 00	.....

Command

Sigamos ya vamos por 18 paginas de tute jeje, San Ricardo sigue al menos hasta que termine de emular todas las instrucciones y llegue al código del programa.

EAX=00472B00 lo mueve a 0047B564

EBX= 7FFDB000 lo mueve a 00481218

esta seria la instrucción que actualiza EBX

00498D03 899D 8C114800 MOV DWORD PTR SS:[EBP+48118C],EBX

y también vemos como los va guardando simétricamente

Address	Hex dump	ASCII
00481218	00 B0 FD 7F 00 00 00 00	.....
00481220	00 00 00 00 00 00 00 00	.....
00481228	00 00 00 00 00 00 00 00	.....
00481230	00 00 00 00 00 00 00 00	.....
00481238	00 00 00 00 00 00 00 00	.....
00481240	00 00 00 00 00 00 00 00	.....
00481248	00 00 00 00 00 00 00 00	.....
00481250	00 00 00 00 00 00 00 00	.....
00481258	00 B0 FD 7F 00 00 00 00	.....
00481260	00 00 00 00 00 00 00 00	.....
00481268	00 00 00 00 00 00 00 00	.....
00481270	00 00 00 00 00 00 00 00	.....
00481278	00 00 00 00 00 00 00 00	.....
00481280	00 00 00 00 00 00 00 00	.....
00481288	00 00 00 00 00 00 00 00	.....
00481290	00 00 00 00 00 00 00 00	.....
00481298	00 B0 FD 7F 00 00 00 00	.....
004812A0	00 00 00 00 00 00 00 00	.....
004812A8	00 00 00 00 00 00 00 00	.....
004812B0	00 00 00 00 00 00 00 00	.....
004812B8	00 00 00 00 00 00 00 00	.....
004812C0	00 00 00 00 00 00 00 00	.....
004812C8	00 00 00 00 00 00 00 00	.....
004812D0	00 00 00 00 00 00 00 00	.....
004812D8	00 B0 FD 7F 00 00 00 00	.....
004812E0	00 00 00 00 00 00 00 00	.....
004812E8	00 00 00 00 00 00 00 00	.....

EAX=12FFBC lo guarda en 0047B158

aquí

00498D0B 8985 CCB04700 MOV DWORD PTR SS:[EBP+47B0CC],EAX

en esta instrucción estaría guardando siempre el valor actual de ESP si miramos donde guarda vemos como fue cambiando

Address	Hex dump	ASCII
0047B158	BC FF 12 00 00 00 00 00	"\$.....
0047B160	00 00 00 00 00 00 00 00	.....
0047B168	00 00 00 00 00 00 00 00	.....
0047B170	00 00 00 00 00 00 00 00	.....
0047B178	00 00 00 00 00 00 00 00	.....
0047B180	00 00 00 00 00 00 00 00	.....
0047B188	00 00 00 00 00 00 00 00	.....
0047B190	00 00 00 00 00 00 00 00	.....
0047B198	C0 FF 12 00 00 00 00 00	l"\$.....
0047B1A0	00 00 00 00 00 00 00 00	.....
0047B1A8	00 00 00 00 00 00 00 00	.....
0047B1B0	00 00 00 00 00 00 00 00	.....
0047B1B8	00 00 00 00 00 00 00 00	.....
0047B1C0	00 00 00 00 00 00 00 00	.....
0047B1C8	00 00 00 00 00 00 00 00	.....
0047B1D0	00 00 00 00 00 00 00 00	.....
0047B1D8	C0 FF 12 00 00 00 00 00	l"\$.....
0047B1E0	00 00 00 00 00 00 00 00	.....
0047B1E8	00 00 00 00 00 00 00 00	.....
0047B1F0	00 00 00 00 00 00 00 00	.....
0047B1F8	00 00 00 00 00 00 00 00	.....
0047B200	00 00 00 00 00 00 00 00	.....

Allí vemos el valor actual 12ffBC y el valor anterior 12ffc0 bueno al menos vamos hallando coincidencias.

Sigamos comienza otro ciclo

EAX=1EF00200 lo mueve a 0047AD4C

Si, algo hallamos ahora vemos como la instrucción SUB le resta 4 al ultimo lugar donde dijimos que guardo el valor de ESP, se viene un PUSH jeje

0047711C 83AD CCB04700 0>SUB DWORD PTR SS:[EBP+47B0CC],4



Address	Hex dump	ASCII
0047B158	BC FF 12 00 00 00 00 00	"L.....
0047B160	00 00 00 00 00 00 00 00	.....
0047B168	00 00 00 00 00 00 00 00	.....
0047B170	00 00 00 00 00 00 00 00	.....
0047B178	00 00 00 00 00 00 00 00	.....
0047B180	00 00 00 00 00 00 00 00	.....
0047B188	00 00 00 00 00 00 00 00	.....
0047B190	00 00 00 00 00 00 00 00	.....
0047B198	C0 FF 12 00 00 00 00 00	"L.....
0047B1A0	00 00 00 00 00 00 00 00	.....
0047B1A8	00 00 00 00 00 00 00 00	.....
0047B1B0	00 00 00 00 00 00 00 00	.....
0047B1B8	00 00 00 00 00 00 00 00	.....
0047B1C0	00 00 00 00 00 00 00 00	.....
0047B1C8	00 00 00 00 00 00 00 00	.....
0047B1D0	00 00 00 00 00 00 00 00	.....
0047B1D8	C0 FF 12 00 00 00 00 00	"L.....
0047B1E0	00 00 00 00 00 00 00 00	.....

Command

Memory breakpoint when executing [0047711C]

Rabiamos dicho que allí se guardo el valor de ESP actual, ahora le resta 4, convirtiéndolo en 4 menos, o sea que cuando hace esto como la vez anterior es que parece que se viene un PUSH, jeje.

Queda así

Address	Hex dump	ASCII
0047B158	B8 FF 12 00 00 00 00 00	@L.....
0047B160	00 00 00 00 00 00 00 00	.....
0047B168	00 00 00 00 00 00 00 00	.....
0047B170	00 00 00 00 00 00 00 00	.....
0047B178	00 00 00 00 00 00 00 00	.....
0047B180	00 00 00 00 00 00 00 00	.....
0047B188	00 00 00 00 00 00 00 00	.....
0047B190	00 00 00 00 00 00 00 00	.....
0047B198	C0 FF 12 00 00 00 00 00	"L.....
0047B1A0	00 00 00 00 00 00 00 00	.....
0047B1A8	00 00 00 00 00 00 00 00	.....
0047B1B0	00 00 00 00 00 00 00 00	.....

Que seria el valor actual de ESP luego del PUSH que esta por realizar como es emulado lo hace en

varias instrucciones, primero actualiza el valor de ESP, luego guardara el valor en el stack, jeje.

```
00496CCB 01 lo pone a 00
00496C9C 08 a 09
00496D3B 00 a 01
```

el POP es la instrucción que actualiza el valor de EBP

```
0048E2C2 8F85 C0A04700 POP DWORD PTR SS:[EBP+47A0C0] ; 0012FFC0
```

que es 12FFc0 y lo guarda en 0047A30C

aquí me cambio un poco el esquema ya que EDI no esta teniendo el valor actual y lo esta actualizando en la misma instrucción que siempre lo actualizo, bueno sigamos

```
0047949B 89BD C0A44700 MOV DWORD PTR SS:[EBP+47A4C0],EDI ;
UnPackMe.0049B8C7
```

EDI=0049B8C7 lo mueve a 0047A70C

quizás sea una maniobra de despiste y luego vuelva al patrón anterior sigamos

ESI se actualiza bien

ESI=FFFFFFFF lo guarda en 0047AB0C

```
EDX=00172CF0 lo mueve a 0047BF38
ECX=00000012 a 00000012
EAX=00472B00 a 0047B724
```

Actualiza EBX

```
00498D03 899D 8C114800 MOV DWORD PTR SS:[EBP+48118C],EBX
```

EBX=7FFDB000 lo mueve a 004813D8

Luego actualiza como vimos el valor de ESP acá

```
00498D0B 8985 CCB04700 MOV DWORD PTR SS:[EBP+47B0CC],EAX
```

EAX=12ffb4 y lo guarda aquí 0047B318

aunque todavía no ejecuto el primer push pero ESP ya esta actualizado como vemos para dos push consecutivos y de esta forma iría a 12ffc4 como muestra allí, ya veremos si es así sigamos.

EAX=12A43F15 mueve a 0047AF0C

Parece que comienza otro ciclo

00496D3B 01 lo pone a 00  
00496C9C 09 0a  
00496D2B 00 01

0048E2C2 8F85 C0A04700 POP DWORD PTR SS:[EBP+47A0C0] ; 0012FFC0

otra vez el POP que actualiza el valor de EBP

0047A2CC C0 FF 12 00 Àÿ#.

Sigue siendo 12FFc0

Ahora actualiza EDI y como vemos lo anterior fue un método de distracción pues vuelve a su valor correcto

0047949B 89BD C0A44700 MOV DWORD PTR SS:[EBP+47A4C0],EDI ;  
ntdll.7C920738

EDI=7C920738 lo mueve a 0047A6CC  
ESI=FFFFFFFF lo mueve a 0047AACC

00470F82 89B5 C0A84700 MOV DWORD PTR SS:[EBP+47A8C0],ESI

siempre allí

EDX=00172CF0 lo mueve a 0047BEF8  
ECX=0047BEF8 lo mueve a 0047BAEC

es posible que estas sean las actualizaciones de EDX Y ECX también aunque como vimos no tenían importancia es bueno saber como se maneja en el caso de que este emulando una rutina de un programa y no solo las primeras instrucciones del OEP

estas instrucciones para ECX y EDX están aquí

00491254 8995 ECBC4700 MOV DWORD PTR SS:[EBP+47BCEC],EDX  
0049125A 898D E0B84700 MOV DWORD PTR SS:[EBP+47B8E0],ECX

aunque no le estamos dando importancia es bueno saber que también las podemos hallar

y EAX=00472B00 se mueve a 0047B6E4

ACTUALIZA EBX

00498D03 899D 8C114800 MOV DWORD PTR SS:[EBP+48118C],EBX

EBX=7FFDB000 se mueve a 00481398

Actualiza ESP

```
00498D0B 8985 CCB04700 MOV DWORD PTR SS:[EBP+47B0CC],EAX
```

EAX= 0012FFB8 lo mueve a 0047B2D8

vemos entonces que no hará dos push consecutivos porque volvió ESP a 12ffb8 o sea que ahora apunta correctamente al valor donde quedara luego del primer push solo fue su acostumbrado vuelterio.

Otro ciclo jeje

EAX=09AFCDC0 lo guarda en 0047AECC

EAX=00493FCD lo mueve a 004817A0

EBX=00493FCD lo mueve a 004817A0

Vemos que hay ocasiones que lee los valores de los registros por ejemplo en esta instrucción

```
004965A4 8BB5 C0A84700 MOV ESI,DWORD PTR SS:[EBP+47A8C0]
```

esta leyendo el ultimo valor de ESI guardado que estaba en

```
0047AACC FF FF FF FF      yyyÿ
```

quiere decir que nuestra teoría es cierta solo que los lugares donde guarda los registros se van moviendo, pero igual se comprueba, en estas veces que lee de los registros obviamente el programa no para porque solo tenemos un BPM ON WRITE y si pusiéramos por ACCESS pararía por ejecución en cada instrucción ya que es la misma sección así que bueno, sigamos como podemos.

```
00496D2B 01 lo pone a 00
```

```
00496C9C 0A lo pone a 0b
```

```
00496D1B 00          01
```

De nuevo el POP actualiza el valor de EBP

```
0048E2C2 8F85 C0A04700 POP DWORD PTR SS:[EBP+47A0C0] ; 0012FFC0
```

Actualiza EDI Y ESI con los valores correctos en las mismas instrucciones que antes.

```
00491254 8995 ECBC4700 MOV DWORD PTR SS:[EBP+47BCEC],EDX
```

```
0049125A 898D E0B84700 MOV DWORD PTR SS:[EBP+47B8E0],ECX
```

```
00491260 8985 D8B44700 MOV DWORD PTR SS:[EBP+47B4D8],EAX
```

allí actualiza EDX, ECX y mueve EAX como siempre

Registers (FPU)		
EAX	00472B00	UnPac
ECX	00000012	
EDX	00172CF0	
EBX	7FFDE000	
ESP	0012FFB8	
EBP	000001CC	
ESI	00493FD2	UnPac
EDI	00000203	
EIP	00491254	UnPac
C 0	ES 0023	32bit

Luego actualiza EBX=7FFDE000

00498D03 899D 8C114800 MOV DWORD PTR SS:[EBP+48118C],EBX

Y ESP aquí lo actualiza vale 0012FFB8

00498D0B 8985 CCB04700 MOV DWORD PTR SS:[EBP+47B0CC],EAX

Bueno otro ciclo uf esto es peor que ir a pie a Lujan.

A partir de aquí solo pondré cuando se actualizan registros a valores nuevos, si se repiten y se actualizan en las mismas instrucciones, manteniendo el mismo valor, ya no lo anotare pues ya sabemos donde lo hace y el valor que tiene.

Así que a partir de aquí lo que se repita ya no lo anotare solo lo nuevo o diferente.

Otra vez va a restarle 4 a ESP

0047711C 83AD CCB04700 0>SUB DWORD PTR SS:[EBP+47B0CC],4

0047B418 B4 FF 12 00 'y#.

Siguiendo veo que aquí esta ingresando el valor que debe hacer el PUSH que si recordamos era 450e60.

00491254 8995 ECBC4700 MOV DWORD PTR SS:[EBP+47BCEC],EDX

0049125A 898D E0B84700 MOV DWORD PTR SS:[EBP+47B8E0],ECX

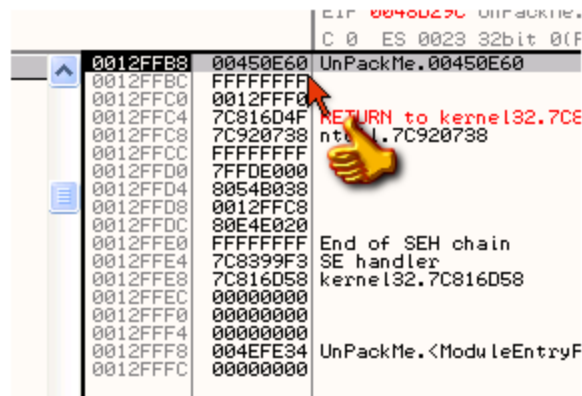
00491260 8985 D8B44700 MOV DWORD PTR SS:[EBP+47B4D8],EAX

Registers (FPU)		
EAX	00472B00	UnPackM
ECX	00450E60	UnPackM
EDX	00172CF0	
EBX	7FFDE000	
ESP	0012FFB8	
EBP	0000030C	
ESI	0048923C	UnPackM
EDI	00000206	
EIP	00491254	UnPackM
C 0	ES 0023	32bit 0

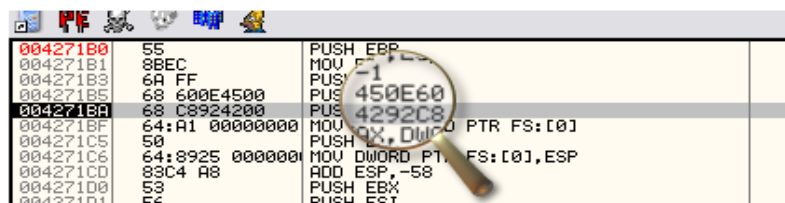
Ahora aquí guarda el valor esperado

0048D299 870C24 XCHG DWORD PTR SS:[ESP],ECX

como vimos había ingresado el valor a ECX y ahora intercambia el valor de ECX con el contenido de ESP y así emula el PUSH 450e60, entre miles de sentencias basura, la verdad una lacra terrible.

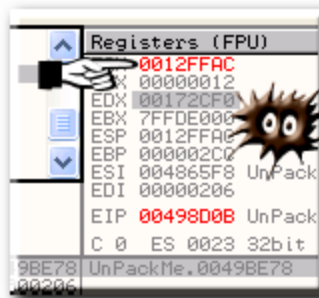


Bueno continuemos marcado solo lo importante y saltando lo repetitivo



Ahí esta el original ahora sigue el PUSH 4292c8, así que sigamos:

en EAX esta el valor de ESP que esta guardando como antes guarda un valor mas pequeño que el necesario pero luego lo corregirá como ya vimos.



Address	Hex dump	ASCII
0047B398	AC FF 12 00 00 00 00 00	% \$.....
0047B3A0	00 00 00 00 00 00 00 00	.....
0047B3A8	00 00 00 00 00 00 00 00	.....
0047B3B0	00 00 00 00 00 00 00 00	.....
0047B3B8	00 00 00 00 00 00 00 00	.....
0047B3C0	00 00 00 00 00 00 00 00	.....
0047B3C8	00 00 00 00 00 00 00 00	.....
0047B3D0	00 00 00 00 00 00 00 00	.....
0047B3D8	B8 FF 12 00 00 00 00 00	@ \$.....
0047B3E0	00 00 00 00 00 00 00 00	.....
0047B3E8	00 00 00 00 00 00 00 00	.....
0047B3F0	00 00 00 00 00 00 00 00	.....
0047B3F8	00 00 00 00 00 00 00 00	.....
0047B400	00 00 00 00 00 00 00 00	.....
0047B408	00 00 00 00 00 00 00 00	.....
0047B410	00 00 00 00 00 00 00 00	.....
0047B418	B4 FF 12 00 00 00 00 00	+ \$.....
0047B420	00 00 00 00 00 00 00 00	.....
0047B428	00 00 00 00 00 00 00 00	.....
0047B430	00 00 00 00 00 00 00 00	.....
0047B438	00 00 00 00 00 00 00 00	.....

Allí vemos como fue variando ESP, aquí varia un poco mas que en el programa original pues hace mil piruetas, pero estuvo es 12ffb8 cuando hizo el PUSH anterior como corresponde y ahora cambio a 12ffac pero seguro cambiara al valor correcto antes de hacer el otro push.

00496805	8385 CCB04700	ADD DWORD PTR SS:[EBP+47B0CC],4
0049680C	8B85 CCB04700	MOV EAX,DWORD PTR SS:[EBP+47B0CC]
00496812	8BE0	MOV ESP,EAX
00496814	68 B47F121F	PUSH 1F127FB4
00496819	5A	POP EDX
0049681A	81F2 A277F226	XOR EDX,26F277A2
00496820	E9 8001FFFF	JMP 004869A5
00496825	81F2 FE6176AD	XOR EDX,AD7661FE
0049682A	81F2 000041B0	SUB EDI,41B00000
SS:[0047B398]=0012FFAC		
Address	Hex dump	ASCII
0047B398	AC FF 12 00 00 00 00 00	% \$.....
0047B3A0	00 00 00 00 00 00 00 00	.....
0047B3A8	00 00 00 00 00 00 00 00	.....

Y si dicho y hecho ahora aumenta ESP sumándole 4, justo en la misma dirección que estaba guardado.

Comos siempre vueltero vuelve a restarle 4 jeje

00471718 83AD CCB04700 0>SUB DWORD PTR SS:[EBP+47B0CC],4

Y aquí tambero guarda un valor incorrecto de ESI para despistar, como ya hizo antes con EDI, luego lo restaurara.

y si traceo ya llego a la sección code donde ejecuta las instrucciones que faltan allí mismo



004271C5	50	PUSH EAX
004271C6	64:8925 000000	MOV DWORD PTR FS:[0],ESP
004271C0	83C4 A8	ADD ESP,-58
004271D0	53	PUSH EBX
004271D1	56	PUSH ESI
004271D2	57	PUSH EDI
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]
EAX=0012FFE0		

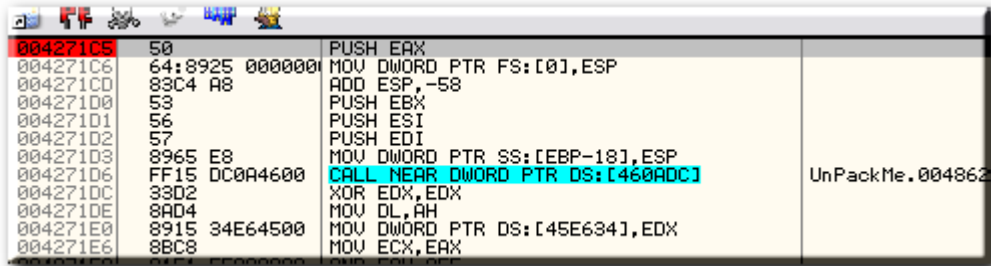
Y continua ya desde allí con el programa normal

Bueno comprendemos un poco mas ya como funciona el packer emulando las instrucciones, vemos que es una emulación bastante de las muy dificiles, porque esta diferencia es la que quería marcar en las emulaciones mas sencillas, los registros se guardan en posiciones de memoria que no cambian de lugar, o sea que uno puede ir testeando esas posiciones de memoria y ver cuando van cambiando para saber cuando se ejecuto una instrucción real del programa y no basura, otra importante es que muchos emuladores restauran en los registros EAX, EBX etc antes de ejecutar una instrucción real, todos los valores que tenían los reales hasta ese momento con lo cual con un simple traceo EAX==XXXX && EBX=YYYYY && etc parara cuanto todos los registros tomen los valores actuales, justo antes de ejecutar una instrucción, aquí vimos que no es así, los registros guardados del programa real nunca están todos a la vez a la vista, y están dispersos lo que hace mas difícil encontrar los momentos justos de que se ejecuta una instrucción.

Bueno con la información recopilada trataremos de avanzar un poco mas:

004271B5	C3	RETN
004271B6	873C24	XCHG DWORD PTR SS:[ESP],EDI
004271B9	5F	POP EDI
004271BA	9C	PUSHFD
004271BB	E9 404E0400	JMP 0046C000
004271C0	3E:122497	ADC AH,BYTE PTR DS:[EDI+EDX*4]
004271C4	BA 50648925	MOV EDX,25896450
004271C9	0000	ADD BYTE PTR DS:[EAX],AL
004271CB	0000	ADD BYTE PTR DS:[EAX],AL
004271CD	83C4 A8	ADD ESP,-58
004271D0	53	PUSH EBX
004271D1	56	PUSH ESI
004271D2	57	PUSH EDI
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]
004271DC	33D2	XOR EDX,EDX

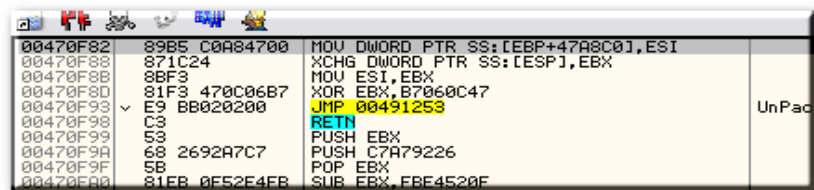
Ahí llegamos al OEP y vamos a poner un BP donde retorna de la rutina emulada al programa o sea aquí



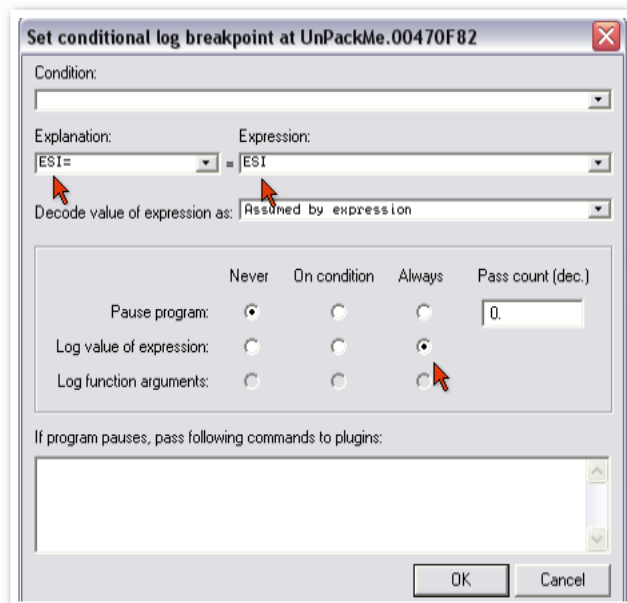
Vimos que volvía allí ahora tratemos de colocar BREAKPOINTS CONDICIONALES en las instrucciones que vimos que guardaban los valores de los registros reales.

00470F82 89B5 C0A84700 MOV DWORD PTR SS:[EBP+47A8C0],ESI

aquí guardaba ESI vamos allí



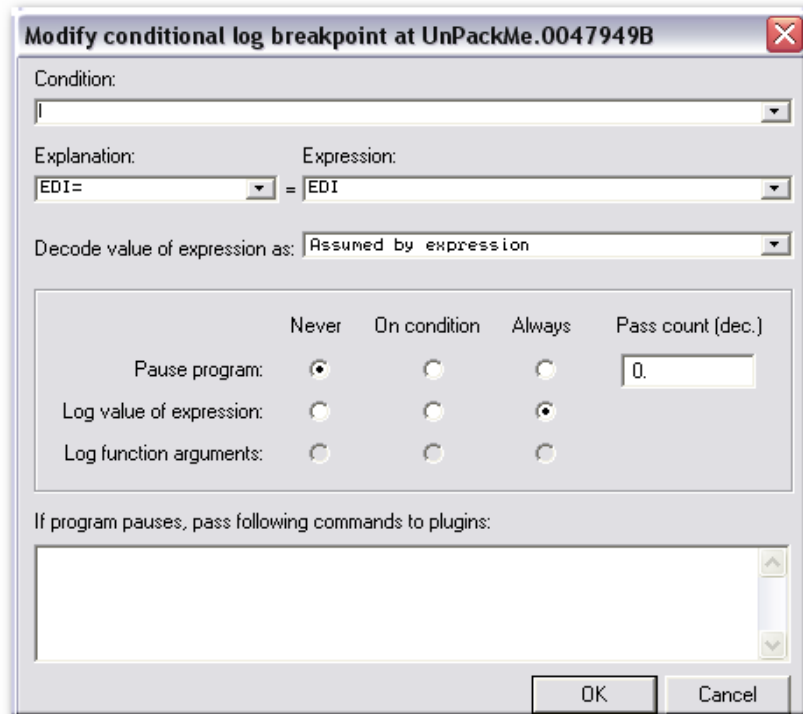
El BREAKPOINT CONDICIONAL seria



Bueno luego vamos a donde guarda **EDI**

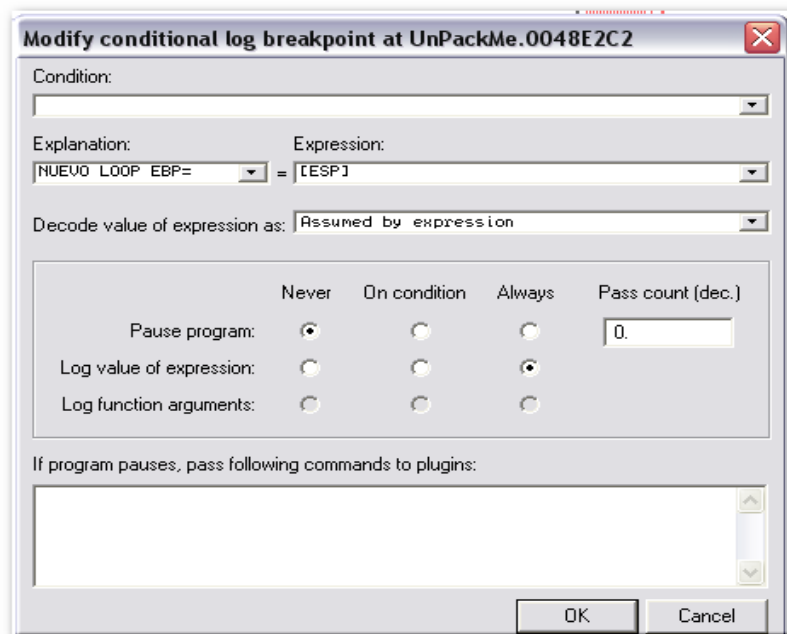
0047949B 89BD C0A44700 MOV DWORD PTR SS:[EBP+47A4C0],EDI

Y hacemos lo mismo



donde mediante el POP actualiza **EBP**

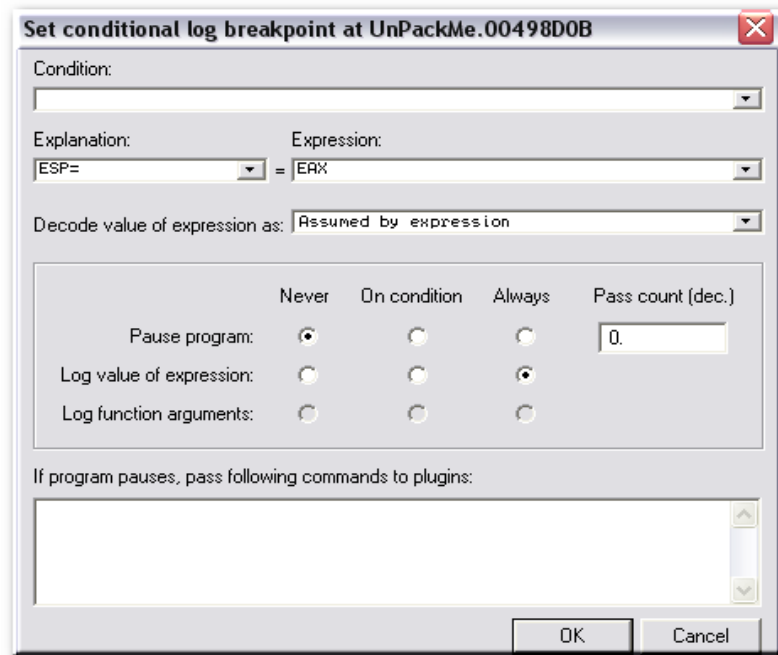
0048E2C2 8F85 C0A04700 POP DWORD PTR SS:[EBP+47A0C0]



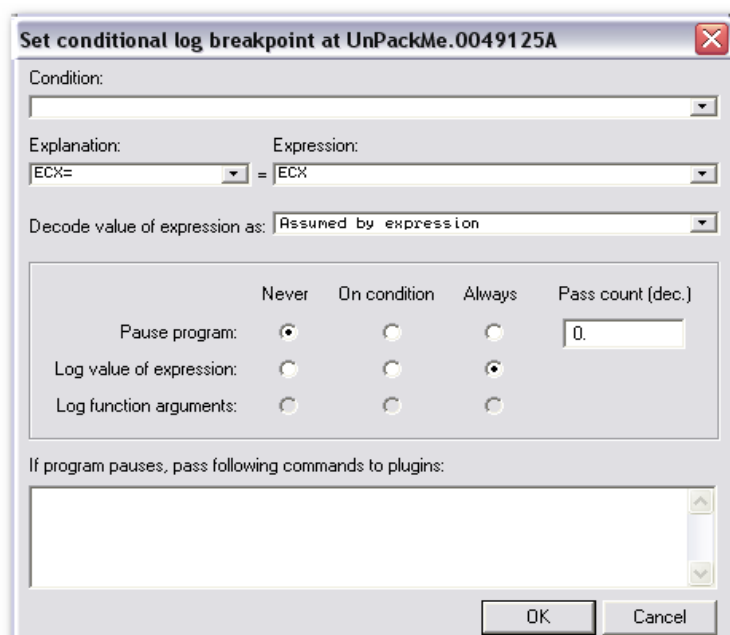
Ya que el valor de EBP del programa original emulado se encuentra en el contenido de ESP en ese momento y el POP lo saca de allí.

00498D0B 8985 CCB04700 MOV DWORD PTR SS:[EBP+47B0CC],EAX

En esta instrucción guardaba el valor de **ESP** que estaba en EAX así que ponemos también un BP CONDICIONAL ALLÍ.



0049125A aquí actualiza **ECX**



Aunque no respetaba el valor de **EDX** igual vimos que lo actualizaba aquí

```
00491254 8995 ECBC4700 MOV DWORD PTR SS:[EBP+47BCEC],EDX
```

pongamos por si acaso también allí un BP CONDICIONAL como los anteriores.

Y **EBX**

```
00498D03 899D 8C114800 MOV DWORD PTR SS:[EBP+48118C],EBX
```

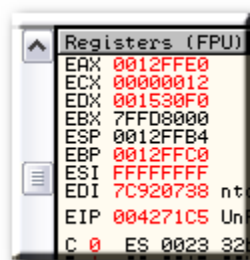
Nos quedaría ver EAX que no esta tan sencillo de determinar, pero si sigieramos la emulación en otra partes del unpackme siguiente seguramente ya lo sacaremos también al ver como va variando EAX.

Deberemos guardar a una fila el LOGUEO.

Para que el sistema sea perfecto al terminar el ultimo LOOP debería estar muy parecido los registros guardados con los registros reales

```
00498D0B COND: ESP= = 0012FFB0
0048E2C2 COND: NUEVO LOOP EBP= = 0012FFC0
0047949B COND: EDI= = 7C920738
00470F82 COND: ESI= = F2AFFFFF
00491254 COND: EDX= = 001530F0
0049125A COND: ECX= = 00000012
00498D03 COND: EBX= = 7FFD8000
00498D0B COND: ESP= = 0012FFA8
004271C5 Breakpoint at UnPackMe.004271C5
```

Vemos que no son exactos pero nos estamos acercando



EBP es similar, lo mismo que ECX, EDX,EBX,EDI nos quedaría ver bien el tema de ESP y de EAX y ESI que esta cerca pero varia demasiado.

Esto puede ser mejorado, si queremos investigar mejor el tema de ESI por ejemplo, traceemos con esta condición a ver si encontramos el momento en que ESI vale FFFFFFFF (no olvidar poner el 0 delante de FFFFFFFF) y luego ver si poniendo el CONDICIONAL BREAKPOINT se mantiene siempre en ese valor asta que llega de nuevo al programa luego de la emulación.

**Condition to pause run trace**

Pause run trace when any checked condition is met:

☐ EIP is in range    00401000 ... 0044AFFE

☐ EIP is outside the range    00000000 ... 00000000

☒ Condition is TRUE    ESI==0FFFFFFF

☐ Command is suspicious or possibly invalid

☐ Command count is    0.    (actual 1519. )    **Reset**

☐ Command is one of

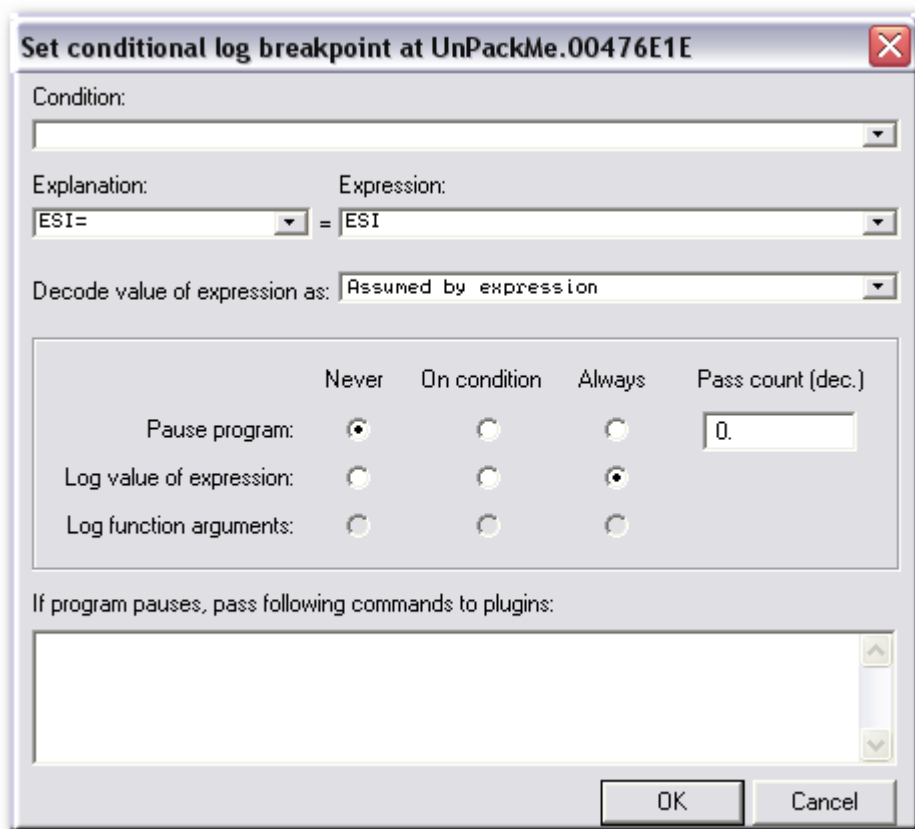
In command, R8, R32, RA, RB and CONST match any register or constant

**OK**    **Cancel**

Para aquí

00476E19	871C24	XCHG DWORD PTR SS:[ESP],EBX	
00476E1C	8BF3	MOV ESI,EBX	
00476E1E	5B	POP EBX	7F
00476E1F	E9 9FCB0000	JMP 004839C3	Un
00476E24	B8 F1203A47	MOV EAX,473A20F1	

y ESI vale FFFFFFFF pongamos el BP CONDICIONAL allí a ver si respeta que siempre actualiza el valor de ESI.



Address	Module	Active	Disassembly
004271C5	UnPackMe	Always	PUSH EAX
00470F82	UnPackMe	Disabled	MOV DWORD PTR SS:[EBP+47A8C0],ESI
00476E1E	UnPackMe	Log "ESI="	POP EBX
0047949B	UnPackMe	Log "EDI="	MOV DWORD PTR SS:[EBP+47A4C0],EDI
0048E2C2	UnPackMe	Log "NUEVO LOOP EBP="	POP DWORD PTR SS:[EBP+47A0C0]
00491254	UnPackMe	Log "EDX="	MOV DWORD PTR SS:[EBP+47BCE0],EDX
0049125A	UnPackMe	Log "ECX="	MOV DWORD PTR SS:[EBP+47B8E0],ECX
00498D03	UnPackMe	Log "EBX="	MOV DWORD PTR SS:[EBP+48118C],EBX
00498D0B	UnPackMe	Log "ESP="	MOV DWORD PTR SS:[EBP+47B0CC],EAX

Y deshabilito el otro que tenia para ESI

```

00401000 Sent virtual address to ollybone module for NX r
0048E2C2 COND: NUEVO LOOP EBP= = 0012FFF0
0047949B COND: EDI= = 7C920738
00491254 COND: EDX= = 0047F3EF
0049125A COND: ECX= = 0047F3EF
00498D03 COND: EBX= = 7FFDF000
00498D0B COND: ESP= = 0012FFC0
00476E1E COND: ESI= = FFFFFFFF
0048E2C2 COND: NUEVO LOOP EBP= = 0012FFF0
0047949B COND: EDI= = 7C920738
00491254 COND: EDX= = 0047F3EF
0049125A COND: ECX= = 0047F3EF
00498D03 COND: EBX= = 7FFDF000
00498D0B COND: ESP= = 0012FFC4
0048E2C2 COND: NUEVO LOOP EBP= = 0012FFF0
0047949B COND: EDI= = 7C920738
00491254 COND: EDX= = 0047F3EF
0049125A COND: ECX= = 0047F3EF
00498D03 COND: EBX= = 00478304
00498D0B COND: ESP= = 0012FFBC
0048E2C2 COND: NUEVO LOOP EBP= = 0012FFF0

```

Vemos que no se repite en cada loop por lo tanto solo pasa una vez por allí, sigamos buscando que con paciencia y saliva un elefante fuck hormiga, jeje.

Si traceamos desde 47ce1e cuando para vemos que unas lineas después llegamos aquí

0049188F	9C	PUSHFD	
00491890	56	PUSH ESI	
00491891	8BF0	MOV ESI,EAX	
00491893	873424	XCHG DWORD PTR SS:[ESP],ESI	
00491896	^ E9 9D56FFFF	JMP 00486F38	UnPackMe.00486F38
0049189B	^ 0F8D CE160000	JGE 00492F6F	UnPackMe.00492F6F
004918A1	^ E9 0EDAFFFF	JMP 0048F2B4	UnPackMe.0048F2B4
004918A6	^ E9 9EC5FFFF	JMP 0047DF49	UnPackMe.0047DF49

donde vuelve a recuperar a ESI el valor FFFFFFFF y que para si ponemos un BP repetidas veces y siempre recupera el FFFFFFFF, pongamos el BP CONDICIONAL aquí.

0049188F	9C	PUSHFD	
00491890	56	PUSH ESI	
00491891	8BF0	MOV ESI,EAX	
00491893	873424	XCHG DWORD PTR SS:[ESP],ESI	
00491896	^ E9 9D56FFFF	JMP 00486F38	Un
0049189B	^ 0F8D CE160000	JGE 00492F6F	Un
004918A1	^ E9 0EDAFFFF	JMP 0048F2B4	Un

En la linea siguiente ya que allí ya tiene ESI el valor correcto veamos sis e mantiene hasta el FIN cada vez que pasa por aquí.

00498D0B	COND: ESP=	= 0012FFB4	
0048E2C2	COND: NUEVO LOOP EBP=	= 0012FFC0	
0047949B	COND: EDI=	= 7C920738	
00491254	COND: EDX=	= 001530F0	
0049125A	COND: ECX=	= 00000012	
00498D03	COND: EBX=	= 7FFD6000	
00498D0B	COND: ESP=	= 0012FFB8	
00491896	COND: ESI=	= FFFFFFFF	
0048E2C2	COND: NUEVO LOOP EBP=	= 0012FFC0	
0047949B	COND: EDI=	= 7C920738	
00491254	COND: EDX=	= 001530F0	
0049125A	COND: ECX=	= 00000012	
00498D03	COND: EBX=	= 7FFD6000	
00498D0B	COND: ESP=	= 0012FFB4	
0048E2C2	COND: NUEVO LOOP EBP=	= 0012FFC0	
0047949B	COND: EDI=	= 7C920738	
00491254	COND: EDX=	= 001530F0	
0049125A	COND: ECX=	= 01B1CF8F	
00498D03	COND: EBX=	= 7FFD6000	
00498D0B	COND: ESP=	= 0012FFB8	
0048E2C2	COND: NUEVO LOOP EBP=	= 0012FFC0	
0047949B	COND: EDI=	= 7C920738	
00491254	COND: EDX=	= 001530F0	
0049125A	COND: ECX=	= 00450E60	
00498D03	COND: EBX=	= 7FFD6000	
00498D0B	COND: ESP=	= 0012FFB8	
0048E2C2	COND: NUEVO LOOP EBP=	= 0012FFC0	
0047949B	COND: EDI=	= 7C920738	
00491254	COND: EDX=	= 001530F0	
0049125A	COND: ECX=	= 00000012	
00498D03	COND: EBX=	= 7FFD6000	
00498D0B	COND: ESP=	= 0012FFAC	
00491896	COND: ESI=	= FFFFFFFF	
0048E2C2	COND: NUEVO LOOP EBP=	= 0012FFC0	
0047949B	COND: EDI=	= 7C920738	
00491254	COND: EDX=	= 001530F0	
0049125A	COND: ECX=	= 00000012	
00498D03	COND: EBX=	= 7FFD6000	
00498D0B	COND: ESP=	= 0012FFB0	
0048E2C2	COND: NUEVO LOOP EBP=	= 0012FFC0	
0047949B	COND: EDI=	= 7C920738	
00491254	COND: EDX=	= 001530F0	
0049125A	COND: ECX=	= 00000012	
00498D03	COND: EBX=	= 7FFD6000	
00498D0B	COND: ESP=	= 0012FFB0	
0048E2C2	COND: NUEVO LOOP EBP=	= 0012FFC0	
0047949B	COND: EDI=	= 7C920738	
00491254	COND: EDX=	= 001530F0	
0049125A	COND: ECX=	= 00000012	
00498D03	COND: EBX=	= 7FFD6000	
00491254	COND: EDX=	= 001530F0	



Si hago un traceo completo veo que solo al final recupera el valor correcto de ESI, así que por ahora mantendremos este como valor correcto de ESI aunque no aparezca en todos los LOOPS, seguro tendremos muchas posibilidades de mejorar la precisión cuando hagamos rutinas mas largas.

Creo que esta parte se extendió demasiado ya seguiremos estudiando y mejorando la comprensión de la emulación en la parte siguiente.

Hasta la 59  
Ricardo Narvaja  
22/11/06