

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 31

NOCIONES INICIALES SOBRE DESEMPACANDO

Ustedes diran este se volvio loco, anuncio una tercera parte de PCODE y ahora comienza con desempacado, lo que pasa es que estoy viendo que en nuestra lista y en muchos crackers que me consultan, estan ansiosos por mejorar su performance en el unpacking, y PCODE hay muchos tutes incluso con WKT, ademas que no hay tantos programas hechos con PCODE, que creo que mas vale empezar con el tema de desempacado, ya que packers hay miles, por supuesto no veremos todos aquí pero daremos ideas generales y ejemplos que nos ayudaran a pensar packers que no hemos visto, por nosotros mismos, sin desesperarnos en buscar un tute por alli, (bueno a veces una ayuda puede servir jeje)

En esta primera parte veremos algunos conceptos e ideas basicas que nos sirvan para trabajar en unpacking, luego en futuras partes, pondremos manos a la obra, desempacando ejemplos.

Bueno cual es la idea de empaacar un programa, antes que nada

Pues ya vimos que un programa desempacado es sencillo de modificar pues los bytes estan accesibles desde el inicio del mismo y no cambian (estan en el ejecutable), el listado es constante, y en cualquier momento podemos modificar cualquier byte sin problemas y guardar los cambios.

Si un programa se automodifica y va cambiando a medida que corre, es mas dificil de parchear pues , lo que debes parchear no esta al inicio, y va cambiando a medida que corre el mismo,

De esta manera, un programa empaado, al no tener el codigo original visible en un inicio, el codigo importante del programa original, no puede ser modificado facilmente, ya que no lo hallaremos, al estar encriptado inicialmente.

Pues el tema es que cuando empacas una aplicación con un determinado packer, este encripta y guarda el codigo original del programa, bien escondido, y le agrega generalmente una o mas secciones, y en ellas le agrega una especie de cargador y redirige el Entry Point hacia este cargador-desempacador.

Como resultado de esto el codigo del programa original no estara visible en el inicio, si arrancamos el programa en OLLYDBG y se detiene, lo hara en el ENTRY POINT del desempacador, desde donde comenzara a ejecutarse.

Este desempacador, buscara la informacion guardada del codigo original encriptado, lo desencryptara y guardara en la ubicación original, una vez que el proceso de desencryptado ha concluido, saltara el OEP o Original Entry Point que es el punto de entrada que tenia el programa antes de ser empaado, o sea seria la primera linea del codigo original ejecutada.

Buscaremos el empaador mas sencillo que existe, que es el UPX, bajaremos la version con GUI que es mas sencilla de usar se llama GUIPEX y se baja de:

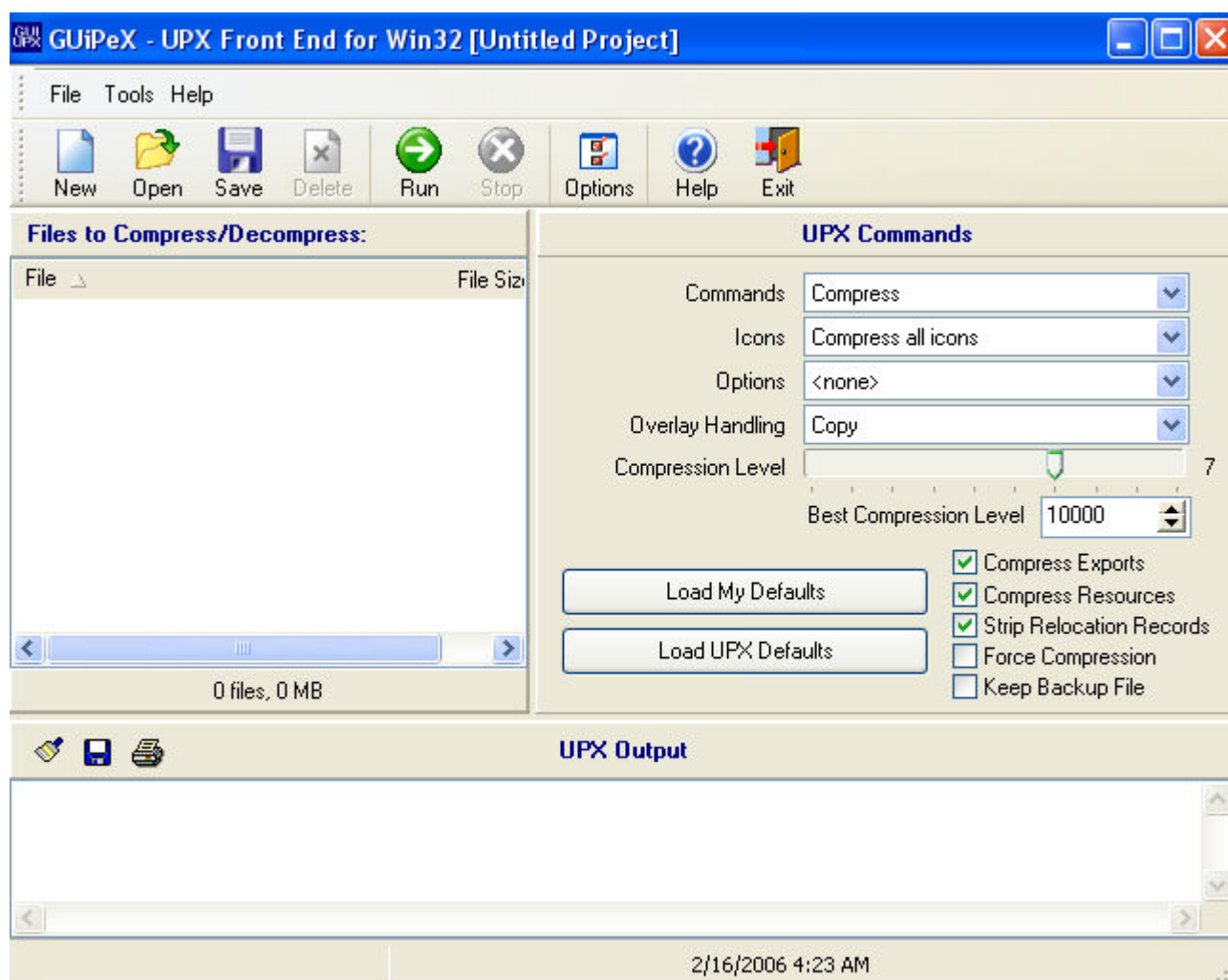
<http://www.blueorbsoft.com/guipeex/>

Installation



To install GUIPeX, simply [click here](#) to download the setup file (498KB). After downloading, launch the setup file and follow the simple on-screen instructions. Should you wish to remove GUIPeX from your system, use the "Uninstall" menu option from GUIPeX's Start Menu program group, or use the "Add/Remove Programs" Control Panel applet.

Bueno lo instalamos en nuestra maquina y lo corremos.



Bueno alli tenemos al empacador mas sencillo, pero que a algunos les ha hecho pasar un mal rato jeje, usaremos como victima el famoso CRACKME DE CRUEHEAD antes que nada abramoslo en OLLYDBG sin empacarlo aun.

```

00401000  6A 00      PUSH 0
00401002  E8 FF040000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007  A3 C8204000 MOV DWORD PTR DS:[4020C8],EAX
0040100C  6A 00      PUSH 0
0040100E  68 F4204000 PUSH CRACKME.004020F4
00401013  E8 A6040000 CALL <JMP.&USER32.FindWindowA>
00401018  0BC0      OR EAX,EAX
0040101A  74 01      JE SHORT CRACKME.0040101D
0040101C  C3        RETN
0040101D  C705 64204000 MOV DWORD PTR DS:[402064],4003
00401027  C705 68204000 MOV DWORD PTR DS:[402068],CRACKME.WndProc
00401031  C705 6C204000 MOV DWORD PTR DS:[40206C],0
0040103B  C705 70204000 MOV DWORD PTR DS:[402070],0
00401045  A1 CA204000 MOV EAX,DWORD PTR DS:[4020CA]
0040104A  A3 74204000 MOV DWORD PTR DS:[402074],EAX
0040104F  6A 64      PUSH 64
00401051  58        POP EAX
  
```

Alli lo tenemos abierto en OLLYDBG, el Entry Point es 401000 o sea que si lo corremos, esta sería la primera línea de código que se ejecutaría.

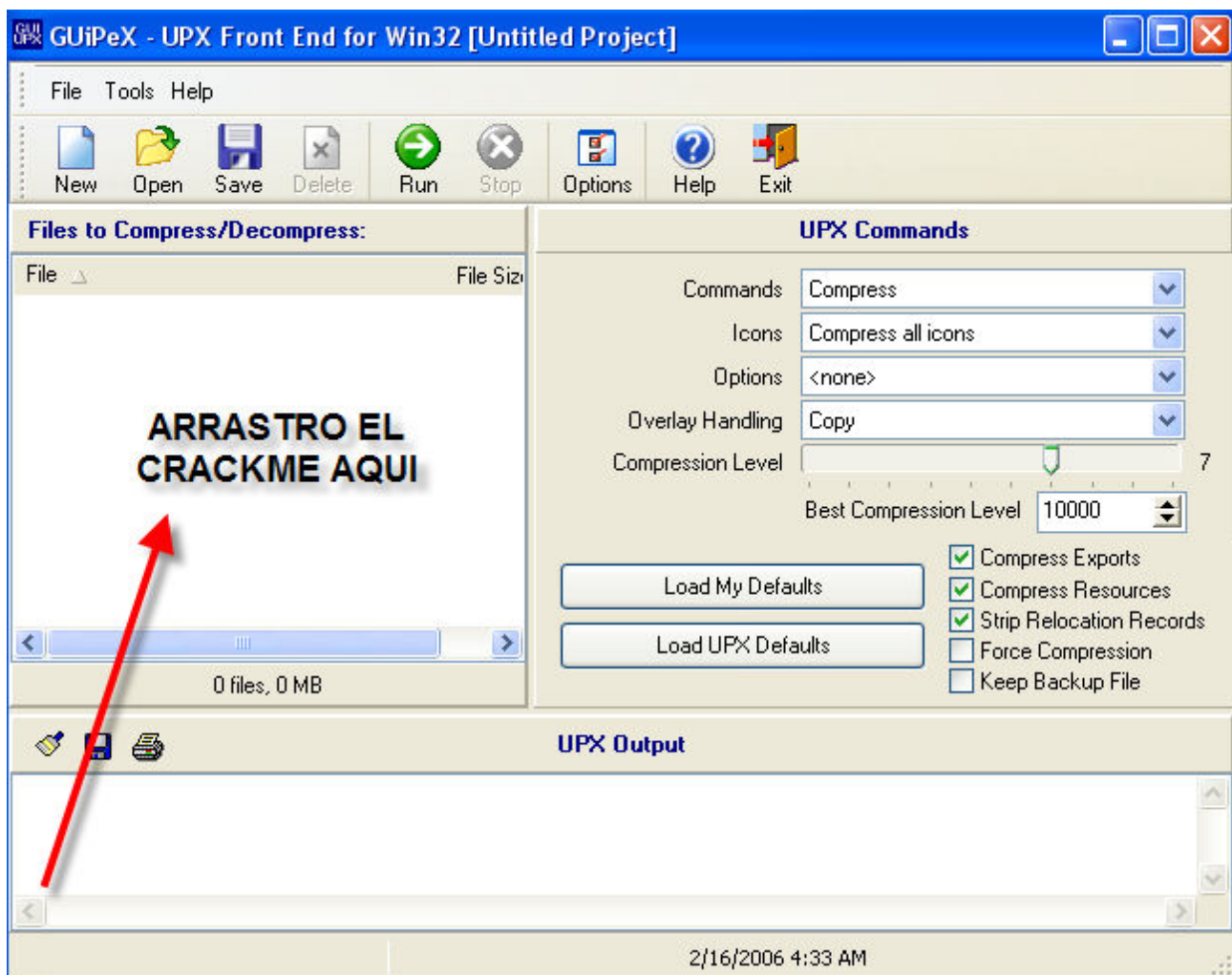
O sea que si lo empacamos el GUIPEX le agregara y cambiara las secciones, y encriptara el código original, y lo guardara en algún lugar, luego cambiara el EntryPoint para que apunte al cargador-desempacador y listo.

Entonces cuando corramos el programa empacado, el desempacador sacara del baul, el código original encriptado, lo desencriptara y ubicara en la primera sección y luego de terminar su trabajo saltara a la primera línea de código original que se va a ejecutar, que es el famoso OEP, o ENTRY POINT ORIGINAL, que en nuestro caso ya sabemos que estara en 401000, ya que el programa original empieza allí y esa sera la primera línea que ejecute del mismo.

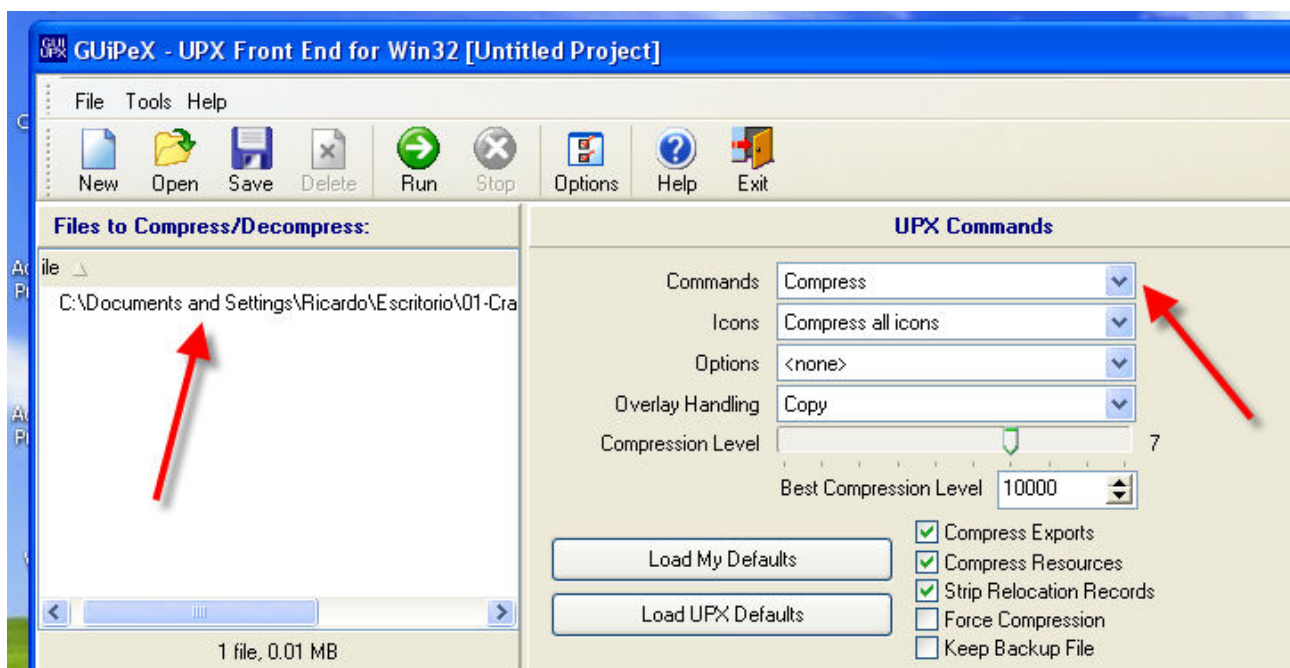
Logicamente cuando nosotros atacamos un programa empacado, no tenemos el original para comparar y saber cual es el OEP o primera línea del código original ejecutado, por lo cual debemos aprender diversas técnicas para hallarlo, antes que nada practiquemos con el CC (Crackme de Cruehead, jeje para abreviar)

Guardemos una copia del CC en un lugar seguro por ejemplo el escritorio ya que el UPX modificara el que tenemos y necesitamos el original también, para comparar.

Abro el GUIPEX



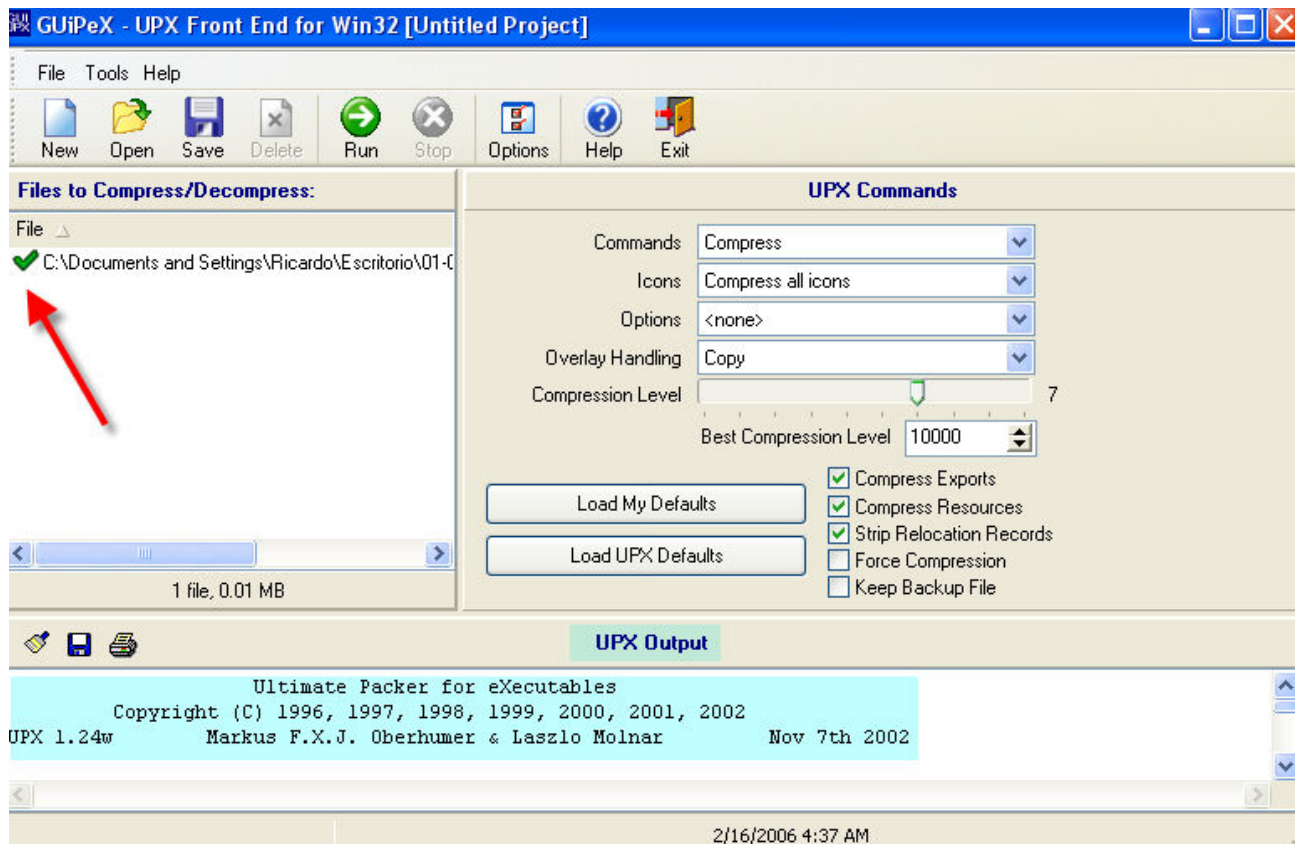
Pues eso, arrastro y suelto el crackme alli



Alli vemos que lo tomo ya que figura el path al crackme, y en COMMANDS elijo COMPRESS

para que comprima , pues tambien puedo descomprimir con esta tool, y el resto de las opciones las dejo como muestra la figura.

Una vez hecho eso apreto el boton RUN del GUIPEX para que haga el trabajo.

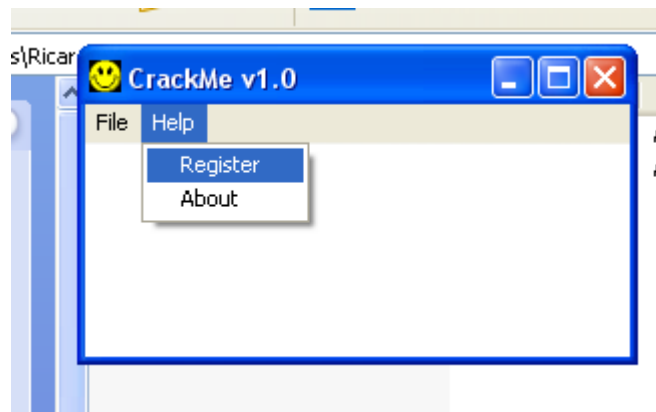


Alli vemos el OK de que la operación ha sido correcta y que el crackme en UPX OUTPUT ha sido empacado.

A este crackme le cambiare el nombre para diferenciarlo del original le pondre CRACKME UPX.exe

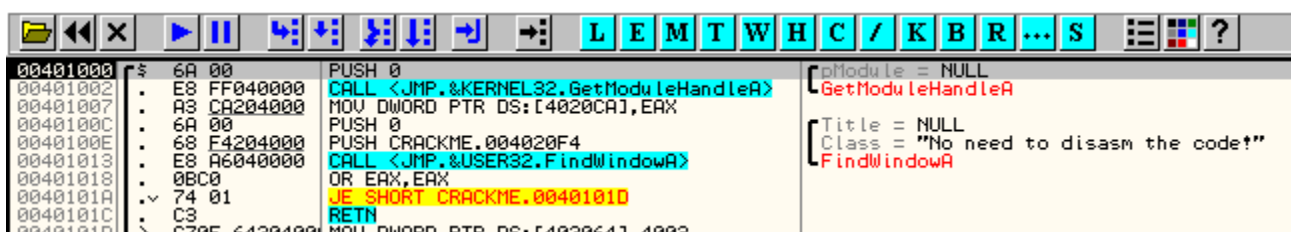
ardo\Escritorio\01-Crackme				
Nombre	Tamaño	Tipo	Fecha de modificación	
📁 CRACKME UPX.EXE	7 KB	Aplicación	09/01/1998 1:22	
📁 CRACKME.EXE	12 KB	Aplicación	09/01/1998 1:22	

Como vemos el crackme empacado es mas pequeño que el original, en una epoca esto era asi, ahora hay packer que agregan tanto codigo para proteger que son mucho mas grandes los empacados que el original, jeje.

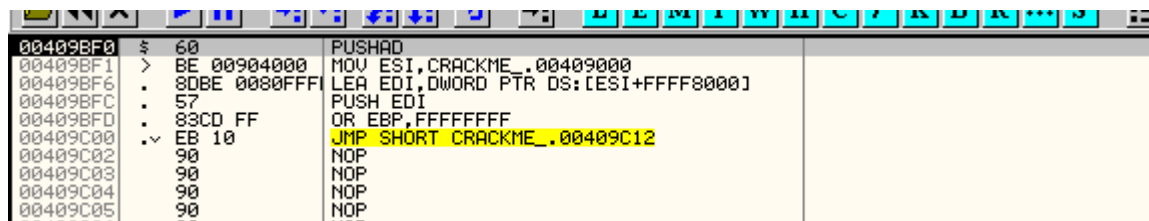


Si ejecutamos el empackado vemos que funciona igual que el original, ahora miremos un poco ambos, abramos dos OLLYDBG uno con el CRACKME UPX.exe y otro con el CRACKME.exe para comparar.

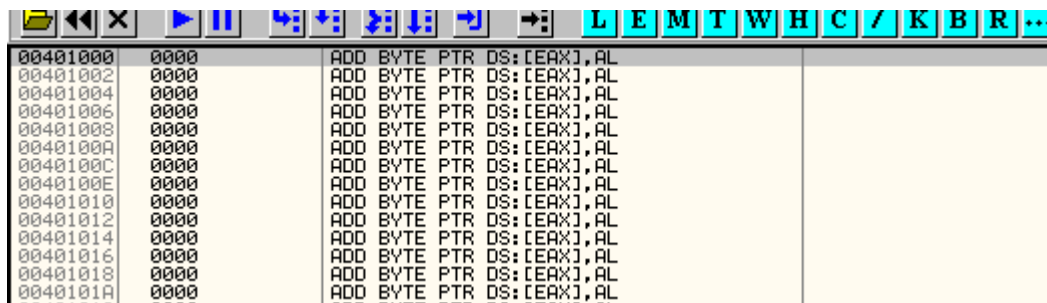
ENTRY POINT DEL CRACKME.exe



ENTRY POINT DEL CRACKME UPX.exe



Como vemos son diferentes el CRACKME UPX cambio el entry point por 409bf0, donde comenzara a ejecutarse el desempacador, y si en este ultimo voy a mirar que hay en la direccion 401000 por supuesto no encuentro ni rastros del codigo original.

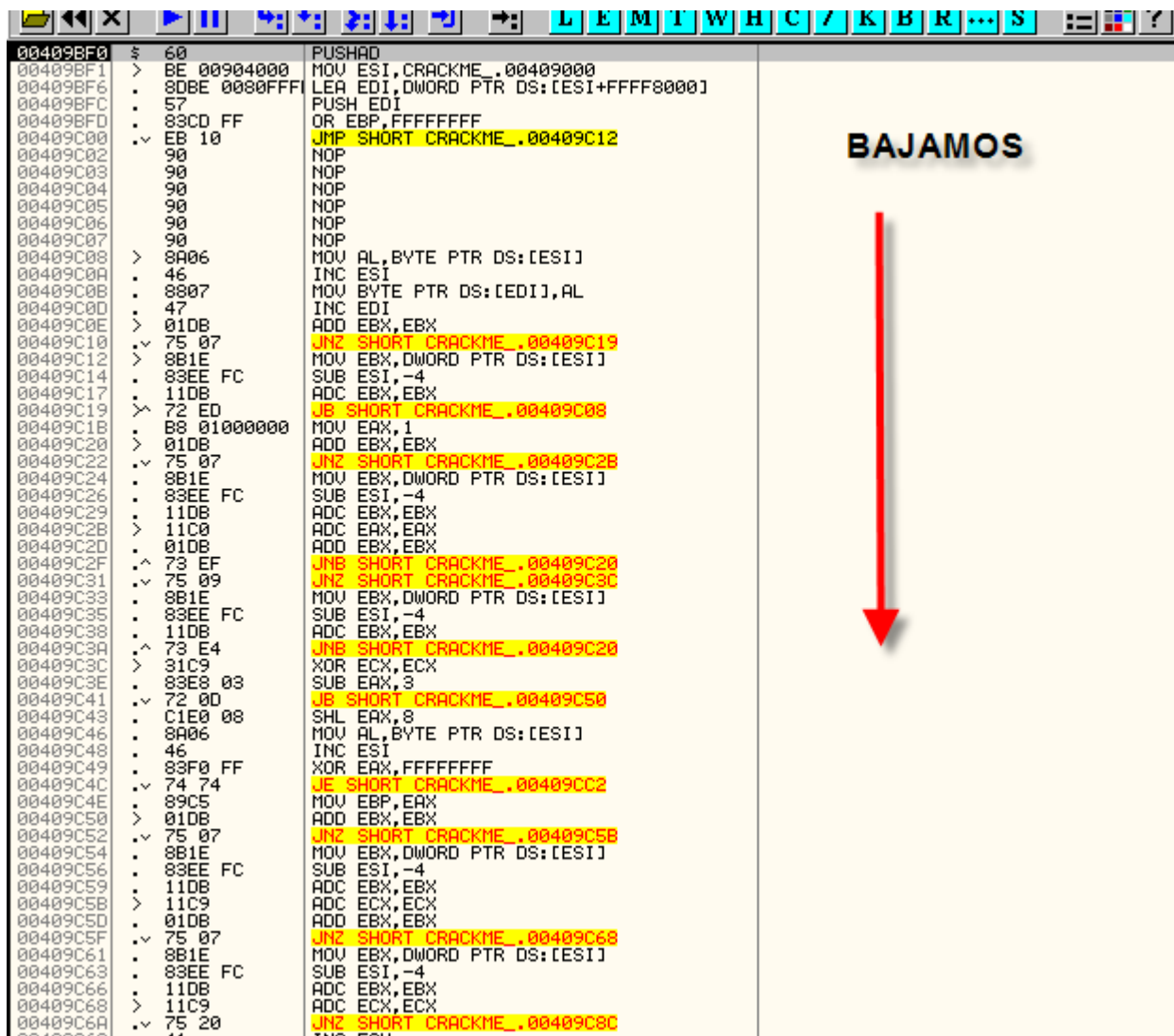


Como vemos la primera seccion esta completamente vacia, asi que el packer quito todos estos bytes los encripto y los guardo en alguna parte del codigo, ademas en este caso vacio la primera seccion completa.

En general la mayoría de los packers, crean una propia seccion y se ejecutan desde alli, para tomar

los bytes encriptados guardados del código original y arreglar la primera sección, descriptando el código original en ella.

Si miramos nuevamente la rutina del desempaquetador sin ejecutarla, y bajamos.



```
00409BF0 $ 60 PUSHAD
00409BF1 > BE 00904000 MOV ESI,CRACKME_.00409000
00409BF6 . 8DBE 0080FFF LEA EDI,DWORD PTR DS:[ESI+FFFF8000]
00409BFC . 57 PUSH EDI
00409BFD . 83CD FF OR EBP,FFFFFFFF
00409C00 . EB 10 JMP SHORT CRACKME_.00409C12
00409C02 . 90 NOP
00409C03 . 90 NOP
00409C04 . 90 NOP
00409C05 . 90 NOP
00409C06 . 90 NOP
00409C07 . 90 NOP
00409C08 > 8A06 MOV AL,BYTE PTR DS:[ESI]
00409C0A . 46 INC ESI
00409C0B . 8807 MOV BYTE PTR DS:[EDI],AL
00409C0D . 47 INC EDI
00409C0E > 010B ADD EBX,EBX
00409C10 . 75 07 JNZ SHORT CRACKME_.00409C19
00409C12 > 8B1E MOV EBX,DWORD PTR DS:[ESI]
00409C14 . 83EE FC SUB ESI,-4
00409C17 . 110B ADC EBX,EBX
00409C19 > 72 ED JB SHORT CRACKME_.00409C08
00409C1B . B8 01000000 MOV EAX,1
00409C1D > 010B ADD EBX,EBX
00409C22 . 75 07 JNZ SHORT CRACKME_.00409C2B
00409C24 . 8B1E MOV EBX,DWORD PTR DS:[ESI]
00409C26 . 83EE FC SUB ESI,-4
00409C29 . 110B ADC EBX,EBX
00409C2B > 11C0 ADC EAX,EAX
00409C2D . 010B ADD EBX,EBX
00409C2F . 73 EF JNB SHORT CRACKME_.00409C20
00409C31 . 75 09 JNZ SHORT CRACKME_.00409C3C
00409C33 . 8B1E MOV EBX,DWORD PTR DS:[ESI]
00409C35 . 83EE FC SUB ESI,-4
00409C38 . 110B ADC EBX,EBX
00409C3A . 73 E4 JNB SHORT CRACKME_.00409C20
00409C3C > 31C9 XOR ECX,ECX
00409C3E . 83E8 03 SUB EAX,3
00409C41 . 72 0D JB SHORT CRACKME_.00409C50
00409C43 . C1E0 08 SHL EAX,8
00409C46 . 8A06 MOV AL,BYTE PTR DS:[ESI]
00409C48 . 46 INC ESI
00409C49 . 83F0 FF XOR EAX,FFFFFFFF
00409C4C . 74 74 JE SHORT CRACKME_.00409CC2
00409C4E . 89C5 MOV EBP,EAX
00409C50 > 010B ADD EBX,EBX
00409C52 . 75 07 JNZ SHORT CRACKME_.00409C5B
00409C54 . 8B1E MOV EBX,DWORD PTR DS:[ESI]
00409C56 . 83EE FC SUB ESI,-4
00409C59 . 110B ADC EBX,EBX
00409C5B > 11C9 ADC ECX,ECX
00409C5D . 010B ADD EBX,EBX
00409C5F . 75 07 JNZ SHORT CRACKME_.00409C68
00409C61 . 8B1E MOV EBX,DWORD PTR DS:[ESI]
00409C63 . 83EE FC SUB ESI,-4
00409C66 . 110B ADC EBX,EBX
00409C68 > 11C9 ADC ECX,ECX
00409C6A . 75 20 JNZ SHORT CRACKME_.00409C8C
00409C6C . 41 INC ECX
```

Seguimos bajando hasta que vemos

00409CCF	> 3C 01	CMP AL,1
00409CD1	. ^ 77 F7	JA SHORT CRACKME_.00409CCA
00409CD3	. 803F 00	CMP BYTE PTR DS:[EDI],0
00409CD6	. ^ 75 F2	JNZ SHORT CRACKME_.00409CCA
00409CD8	. 8B07	MOV EAX,DWORD PTR DS:[EDI]
00409CDA	. 8A5F 04	MOV BL,BYTE PTR DS:[EDI+4]
00409CDD	. 66:C1E8 08	SHR AX,8
00409CE1	. C1C0 10	ROL EAX,10
00409CE4	. 86C4	XCHG AH,AL
00409CE6	. 29F8	SUB EAX,EDI
00409CE8	. 80EB E8	SUB BL,0E8
00409CEB	. 01F0	ADD EAX,ESI
00409CED	. 8907	MOV DWORD PTR DS:[EDI],EAX
00409CEF	. 83C7 05	ADD EDI,5
00409CF2	. 89D8	MOV EAX,EBX
00409CF4	. ^ E2 D9	LOOPE SHORT CRACKME_.00409CCF
00409CF6	. 8DBE 00700000	LEA EDI,DWORD PTR DS:[ESI+7000]
00409CFC	> 8B07	MOV EAX,DWORD PTR DS:[EDI]
00409CFE	. 09C0	OR EAX,EAX
00409D00	. ^ 74 3C	JE SHORT CRACKME_.00409D3E
00409D02	. 8B5F 04	MOV EBX,DWORD PTR DS:[EDI+4]
00409D05	. 8D8430 E0940	LEA EAX,DWORD PTR DS:[EAX+ESI+94E0]
00409D0C	. 01F3	ADD EBX,ESI
00409D0E	. 50	PUSH EAX
00409D0F	. 83C7 08	ADD EDI,8
00409D12	. FF96 58950000	CALL DWORD PTR DS:[ESI+9558]
00409D18	. 95	XCHG EAX,EBP
00409D19	> 8A07	MOV AL,BYTE PTR DS:[EDI]
00409D1B	. 47	INC EDI
00409D1C	. 08C0	OR AL,AL
00409D1E	. ^ 74 DC	JE SHORT CRACKME_.00409CFC
00409D20	. 89F9	MOV ECX,EDI
00409D22	. 57	PUSH EDI
00409D23	. 48	DEC EAX
00409D24	. F2:AE	REPNE SCAS BYTE PTR ES:[EDI]
00409D26	. 55	PUSH EBP
00409D27	. FF96 5C950000	CALL DWORD PTR DS:[ESI+955C]
00409D2D	. 09C0	OR EAX,EAX
00409D2F	. ^ 74 07	JE SHORT CRACKME_.00409D38
00409D31	. 8903	MOV DWORD PTR DS:[EBX],EAX
00409D33	. 83C3 04	ADD EBX,4
00409D36	. ^ EB E1	JMP SHORT CRACKME_.00409D19
00409D38	> FF96 60950000	CALL DWORD PTR DS:[ESI+9560]
00409D3E	> 61	POPAD
00409D3F	. ^ E9 BC72FFFF	JMP CRACKME_.00401000
00409D44	. 00	DB 00
00409D45	. 00	DB 00
00409D46	. 00	DB 00
00409D47	. 00	DB 00
00409D48	. 00	DB 00

SALTO AL OEP

Este es un packer muy inocente, vemos que arranca hace sus desenscriptaciones y cuando termina todo su trabajo salta al OEP, sin ni siquiera ocultarlo, al ver los packers actuales da un poco de risa ver esto aun, pero bueno, asi trabaja.

O sea que si pongo un BP en ese JMP

00409D31	. 8903	MOV DWORD PTR DS:[EBX],EAX
00409D33	. 83C3 04	ADD EBX,4
00409D36	. ^ EB E1	JMP SHORT CRACKME_.00409D19
00409D38	> FF96 60950000	CALL DWORD PTR DS:[ESI+9560]
00409D3E	> 61	POPAD
00409D3F	. ^ E9 BC72FFFF	JMP CRACKME_.00401000
00409D44	. 00	DB 00
00409D45	. 00	DB 00
00409D46	. 00	DB 00
00409D47	. 00	DB 00
00409D48	. 00	DB 00

Y doy RUN

00409D38	> FF96 60950000	CALL DWORD PTR DS:[ESI+9560]
00409D3E	> 61	POPAD
00409D3F	. ^ E9 BC72FFFF	JMP CRACKME_.00401000
00409D44	. 00	DB 00
00409D45	. 00	DB 00
00409D46	. 00	DB 00
00409D47	. 00	DB 00
00409D48	. 00	DB 00

Alli paro y como termino de arreglar la primera seccion, salta al OEP o la primera linea original de codigo apretamos F7.

00401000	6A 00	PUSH 0	
00401002	E8 FF040000	CALL CRACKME_00401506	JMP to kernel32.GetModuleHandleA
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA],EAX	
0040100C	6A 00	PUSH 0	
0040100E	68 F4204000	PUSH CRACKME_004020F4	ASCII "No need to disasm the code!"
00401013	E8 A6040000	CALL CRACKME_004014BE	JMP to USER32.FindWindowA
00401018	0BC0	OR EAX,EAX	
0040101A	74 01	JE SHORT CRACKME_0040101D	
0040101C	C3	RETN	
0040101D	C705 64204000	MOV DWORD PTR DS:[402064],4003	
00401027	C705 68204000	MOV DWORD PTR DS:[402068],CRACKME_0040	
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C],0	
0040103B	C705 70204000	MOV DWORD PTR DS:[402070],0	
00401045	A1 CA204000	MOV EAX,DWORD PTR DS:[4020CA]	
0040104A	A3 74204000	MOV DWORD PTR DS:[402074],EAX	
0040104F	6A 64	PUSH 64	
00401051	50	PUSH EAX	
00401052	E8 D1030000	CALL CRACKME_00401428	JMP to USER32.LoadIconA
00401057	A3 78204000	MOV DWORD PTR DS:[402078],EAX	
0040105C	68 00F00000	PUSH 7F00	
00401061	6A 00	PUSH 0	

Ahi vemos el OEP con el mismo codigo que el Cracme de Cruehead original, el desempacador termino su trabajo y desenscripto todo el codigo original, que como habiamos visto en un inicio no estaba alli, pues todo esto eran ceros, y una vez que termino salto al OEP, para empazar a correr el programa.

Este esquema de funcionamiento de un programa empackado

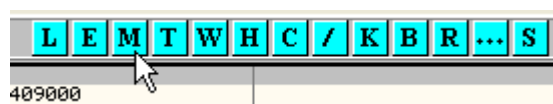
- 1)EJECUCION DEL DESEMPACADOR
- 2)REPARACION Y DESENCRIPTADO DE LA SECCION DONDE CORRE EL CODIGO ORIGINAL
- 3)SALTO AL OEP
- 4)EJECUCION DEL PROGRAMA

Ha sido un esquema que ha funcionado durante años y muchisimos packers trabajan asi, con el tiempo los programadores de packers se han dado cuenta que debian modificar un poco el esquema y han agregado ardidess para ocultar el OEP, y otro trucos que veremos mas adelante, pero normalmente el funcionamiento general es ese.

Reinicio el CRACKME UPX

Obviamente si la primera seccion estaba vacia como en este caso o tenia basura y alli se contruye el codigo original, debera escribir en dicha seccion, por lo cual si uso un BPM ON ACCESS en la misma, parara en la rutina que desenscripta el codigo veamos.

Vayamos a M



003E0000	00004000			PE header	File	RW	RW
003F0000	00002000			exports	Map	R	R
00400000	00001000	CRACKME_	UPX0	code	Image	R	RWE
00401000	00008000	CRACKME_	UPX1	data, import	Image	R	RWE
00409000	00001000	CRACKME_	.rsrc		Image	R	RWE
00410000	0000A000				Map	R E	R E
004D0000	00002000				Map	R E	R E
004F0000	00103000				Map	R	R

Vemos que las secciones han sido cambiadas en tamaño con respecto al original, el cual vemos sus secciones abajo.

File View Debug Plugins Options Window Help

LEMTWHC / KBR ...

```

00401000 6A 00 PUSH 0
00401002 8B FF 04 00 00 A3 CA 20 40 00 6A 00 68 F4 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 8B CA 20 40 00 MOV DWORD PTR DS:[4020CA],EAX
0040100C 6A 00 PUSH 0
0040100E 68 F4 20 40 00 PUSH CRACKME.004020F4
00401013 E8 A6 04 00 00 CALL <JMP.&USER32.FindWindowA>
00401018 0BC0 OR EAX,EAX
0040101A 74 01 JE SHORT CRACKME.0040101D
0040101C C3 RETN
0040101D C7 05 64 20 40 00 MOV DWORD PTR DS:[402064],4003
00401027 C7 05 68 20 40 00 MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031 C7 05 6C 20 40 00 MOV DWORD PTR DS:[40206C],0

```

CRACKME.<ModuleEntryPoint>

Address	Hex dump	ASCII
00401000	6A 00 E8 FF 04 00 00 A3 CA 20 40 00 6A 00 68 F4	j..b ..u# @.j.h
00401010	20 40 00 E8 A6 04 00 00 0B C0 74 01 C3 C7 05 64	@.p@ ..t@t@d
00401020	20 40 00 03 40 00 00 C7 05 68 20 40 00 28 11 40	@.p@ ..h @.(
00401030	00 C7 05 6C 20 40 00 00 00 00 00 C7 05 70 20 40	.A!l @.....Ap @

Vemos que esta guardando el primer byte 6A que el original posee en 401000, si apreto F9 de nuevo, debería guardar el byte siguiente o sea el 00 que esta a continuacion, veamos.

```

00409C0A 46 INC ESI
00409C0B 8B 07 MOV BYTE PTR DS:[EDI],AL
00409C0D 47 INC EDI
00409C0E 01 0B ADD EBX,EBX
00409C10 75 07 JNZ SHORT CRACKME.00409C19
00409C12 8B 1E MOV EBX,DWORD PTR DS:[ESI]
00409C14 83 EE FC SUB ESI,-4
00409C17 01 0B ADD EBX,EBX
00409C19 72 ED JB SHORT CRACKME.00409C08
00409C1B B8 01 00 00 00 MOV EAX,1
00409C1D 01 0B ADD EBX,EBX
00409C1F 75 07 JNZ SHORT CRACKME.00409C2B
00409C21 8B 1E MOV EBX,DWORD PTR DS:[ESI]
00409C23 83 EE FC SUB ESI,-4
00409C26 01 0B ADD EBX,EBX
00409C28 01 0B ADD EAX,EAX
00409C2A 01 0B ADD EBX,EBX
00409C2C 73 EF JNB SHORT CRACKME.00409C20
00409C2E 75 09 JNZ SHORT CRACKME.00409C3C
00409C30 8B 1E MOV EBX,DWORD PTR DS:[ESI]
00409C32 83 EE FC SUB ESI,-4
00409C35 01 0B ADD EBX,EBX
00409C37 73 E4 JNB SHORT CRACKME.00409C20
00409C39 31 C9 XOR ECX,ECX
00409C3B 83 E8 03 SUB EAX,3
00409C3D 72 0D JB SHORT CRACKME.00409C50
00409C3F C1 E0 08 SHL EAX,8
00409C41 8A 06 MOV AL,BYTE PTR DS:[ESI]

```

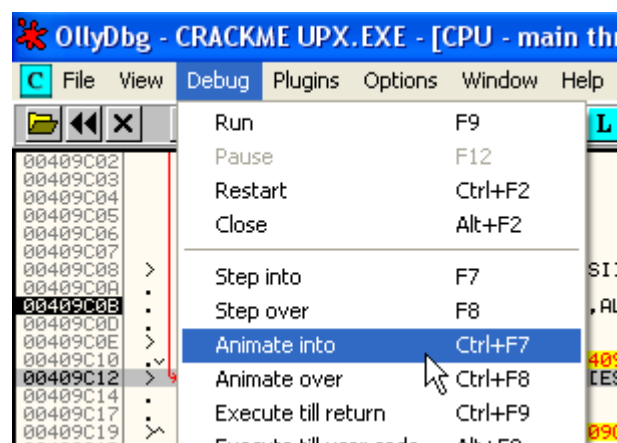
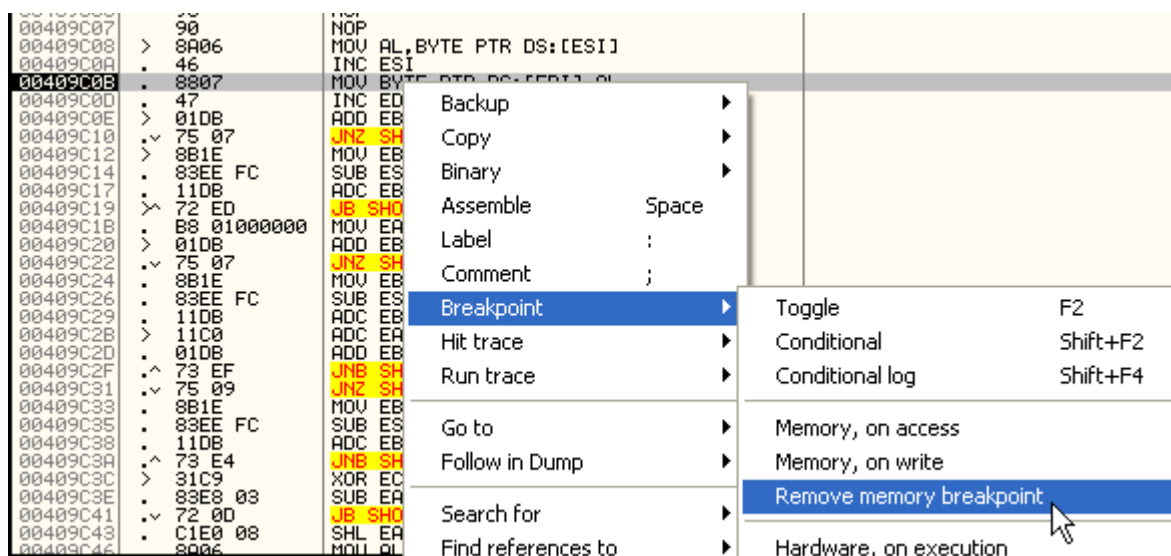
AL=00
DS:[00401001]=00

Address	Hex dump	ASCII
00401000	6A 00 00 00 00 00 00 00 00 00 00 00 00 00 00	j.....
00401010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Y así podemos seguir byte por byte, mientras guarda los bytes descriptados y reconstruye el código original, por supuesto no todos los packers desempacan en orden ni desde el inicio como este que es buenito, pero vemos el ejemplo de algo lo mas sano posible para empezar jeje.

Si nos tomamos el trabajo de tracear veremos que esto es un LOOP donde lee un byte de los guardados encriptados, le realiza las operaciones matematicas para descriptarlo, (sumas, multiplicaciones etc por ejemplo) y cuando obtiene el valor original, lo guarda a continuacion del anterior.

Si quitamos el BPM ON ACCESS, podemos si apretamos ANIMATE INTO, divertirnos viendo como se va llenando la primera seccion con los valores originales como si fuera una pelicula, hasta que llega al BPX que pusimos que es el JMP al OEP.



Vemos como trabaja el programa descriptando la seccion CODE

00409C0E	> 01DB	ADD EBX,EBX	
00409C10	> 75 07	JNZ SHORT CRACKME_.00409C19	
00409C12	> 8B1E	MOV EBX,DWORD PTR DS:[ESI]	
00409C14	> 83EE FC	SUB ESI,-4	
00409C17	> 11DB	ADC EBX,EBX	
00409C19	> 72 ED	JB SHORT CRACKME_.00409C08	
00409C1B	> B8 01000000	MOV EAX,1	
00409C1D	> 01DB	ADD EBX,EBX	
00409C1F	> 75 07	JNZ SHORT CRACKME_.00409C2B	
00409C21	> 8B1E	MOV EBX,DWORD PTR DS:[ESI]	
00409C23	> 83EE FC	SUB ESI,-4	
00409C25	> 11DB	ADC EBX,EBX	
00409C27	> 11C0	ADC EAX,EAX	
00409C29	> 01DB	ADD EBX,EBX	
00409C2B	> 73 EF	JNB SHORT CRACKME_.00409C20	
00409C2D	> 75 09	JNZ SHORT CRACKME_.00409C3C	
00409C2F	> 8B1E	MOV EBX,DWORD PTR DS:[ESI]	
00409C31	> 83EE FC	SUB ESI,-4	
00409C33	> 11DB	ADC EBX,EBX	
00409C35	> 73 E4	JNB SHORT CRACKME_.00409C20	
00409C37	> 31C9	XOR ECX,ECX	
00409C39	> 83E8 03	SUB EAX,3	
00409C3B	> 72 00	JB SHORT CRACKME_.00409C50	
00409C3D	> C1E0 08	SHL EAX,8	
00409C3F	> 8A06	MOV AL,BYTE PTR DS:[ESI]	
00409C41	> 46	INC ESI	
00409C43	> 83F0 FF	XOR EAX,FFFFFFFF	
00409C45	> 74 74	JE SHORT CRACKME_.00409CC2	
00409C47	> 89C5	MOV EBP,EAX	
00409C49	> 01DB	ADD EBX,EBX	
00409C4B	> 75 07	JNZ SHORT CRACKME_.00409C5B	
00409C4D	> 8B1E	MOV EBX,DWORD PTR DS:[ESI]	
00409C4F	> 83EE FC	SUB ESI,-4	
00409C51	> 11DB	ADC EBX,EBX	
00409C53	> 11C9	ADC ECX,ECX	

Jump is NOT taken
00409C50=CRACKME_.00409C50

Address	Hex dump	ASCII
00401000	6A 00 E8 00 00 05 02 A3 CA 20 40 00 6A 00 68 F4	J..@...@.J.h
00401010	20 40 00 E8 00 00 04 BA 08 C0 74 01 C3 C7 05 64	@.p... o^t@t&sd
00401020	20 40 00 03 40 00 00 C7 05 68 20 40 00 28 11 40	@.p@..&sh @.(4@
00401030	00 C7 05 6C 20 40 00 00 00 00 00 00 00 00 00	.&sl @.....
00401040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

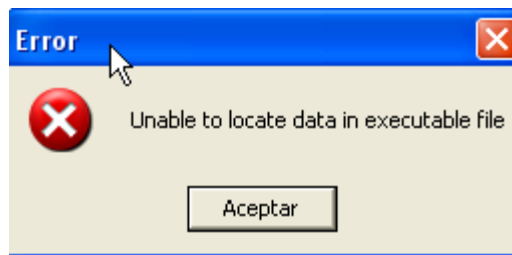
Arriba vemos moviendose y ejecutandose al desempacador transpirando la camiseta en un loop, mientras va llenando la primera seccion de valores originales, hasta que termina y llega al JMP al OEP.

Una vez que para en el JMP estamos nuevamente a un pasito del OEP apretamos F7

00401000	6A 00	PUSH 0	
00401002	E8 FF040000	CALL CRACKME_.00401506	JMP to kernel32.GetModuleHandleA
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA],EAX	
0040100C	6A 00	PUSH 0	
0040100E	68 F4204000	PUSH CRACKME_.004020F4	ASCII "No need to disasm the code!"
00401013	E8 A6040000	CALL CRACKME_.004014BE	JMP to USER32.FindWindowA
00401018	0BC0	OR EAX,EAX	
0040101A	< 74 01	JE SHORT CRACKME_.0040101D	
0040101C	C3	RETN	
0040101D	C705 64204000	MOV DWORD PTR DS:[402064],4003	
00401027	C705 68204000	MOV DWORD PTR DS:[402068],CRACKME_.0040	
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C],0	
0040103B	C705 70204000	MOV DWORD PTR DS:[402070],0	
00401045	A1 CA204000	MOV EAX,DWORD PTR DS:[4020CA]	
0040104A	A3 74204000	MOV DWORD PTR DS:[402074],EAX	
0040104F	6A 64	PUSH 64	
00401051	50	PUSH EAX	
00401053	EB D1030000	CALL CRACKME_.00401428	JMP to USER32.LoadIconA

Alli estamos en el OEP, porque decimos que no podemos cambiar con OLLYDBG el codigo y guardarlo como lo hacemos habitualmente, jeje.

Si yo por ejemplo agarro los dos primeros bytes 6A 00 y los quiero cambiar por ejemplo por 90 90, y trato de guardar los cambios OLLYDBG me despierta con un



O sea que no puedo hacer eso, porque OLLYDBG no encuentra el código en el ejecutable, para cambiarlo.

Pero si insisto y con un EDITOR HEXA pongo 90 90 en 401000 y abro en OLLY el crackme modificado, y voy a mirar que hay en 401000, tendré el 90 90 y a continuación todos ceros, y cuando el packer corra y guarde los valores originales en la primera sección, mis cambios serán sobrescritos, por los bytes 6A 00 nuevamente, que el desempaquetador guardara allí en ejecución y al llegar al OEP en 401000 tendré 6A 00, por más que en el archivo estén guardados 90 90, han sido sobrescritos en tiempo de ejecución por el desempaquetador.

O sea que si yo quisiera cambiar esos dos bytes realmente, debería buscar donde lee el desempaquetador los bytes originales encriptados, ver qué operaciones le aplica para transformarlos en originales, así poder calcular qué valor deberían tener uff, demasiado trabajo más fácil es desempaquetar el archivo así puedo cambiar lo que se me antoje cuando quiero, jeje.

Por supuesto hay otras técnicas para parchear en memoria mediante loaders, que no son motivo de esta parte, los loaders cargan el crackme, esperando que se desempaquete en memoria y luego hacen los cambios en allí mismo en la memoria, para que al correr, encuentre los bytes modificados, pero eso es otra historia vamos despacio que despacio se llega jeje ya eso lo veremos más adelante.

Bueno aquí termina esta breve y sencilla reseña inicial, en la parte dos continuaremos investigando el desempaquetado del famoso CC.

Hasta la parte 32
Ricardo Narvaja
16/02/06