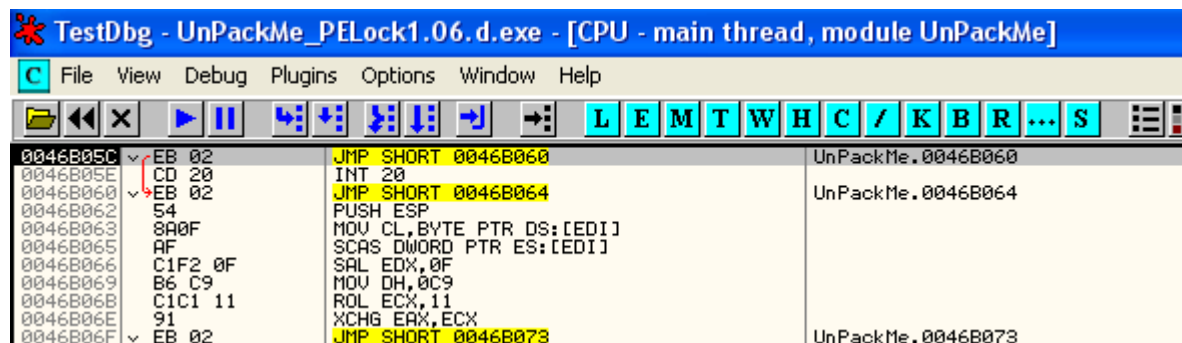


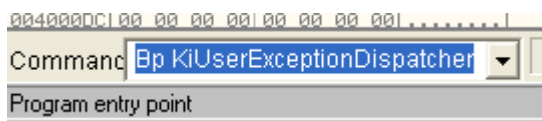
INTRODUCCION AL CRACKING CON OLLYDBG PARTE 39

Bueno en esta parte y la 40 empezaremos el tema de stolen bytes y scripts para ello usaremos uno unpackme conocido el UnPackMe_PELock1.06.d.exe que tiene su miga, y que del mismo o variaciones del mismo han hecho esta semana grandes tutes en la lista como el de Otup que esta genial..

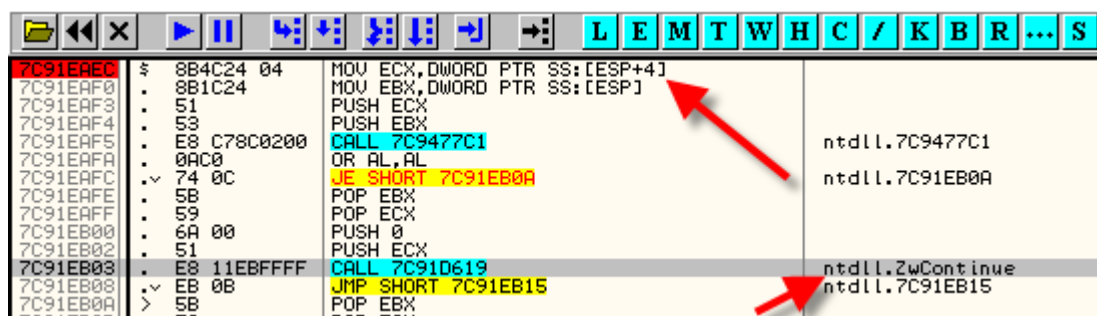
Aquí estamos en el inicio del programa parados en el OLLYDBG



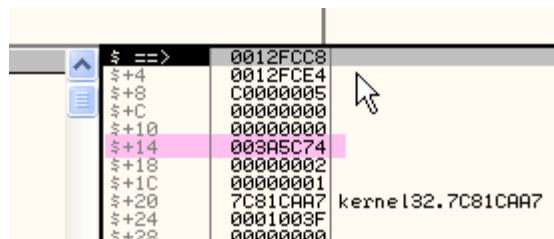
Usaremos una variacion del metodo de las excepciones mas adelante veran porque, en este caso en vez de confiar en el LOG de OLLYDBG haremos nuestro propio LOG de excepciones, porque en estos packers una excepcion que se te pasa o no sale en el LOG o que es camuflada nos puede hacer mucho lio, lo cual puede ser posible ya que OLLYDBG no es perfecto y ya me ha ocurrido en packers raros y asi terminamos muertos, entonces estudiemos este BP.



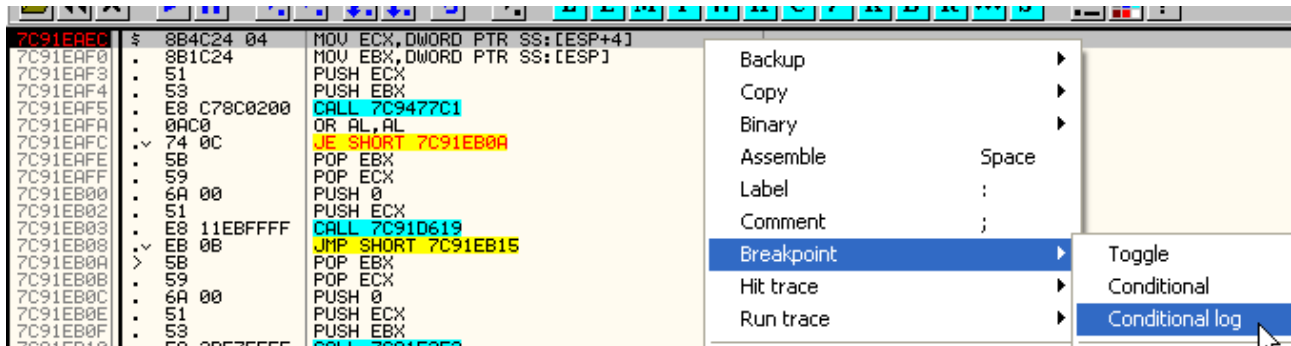
Este BP sera nuestro LOGUEADOR de excepciones, por ahi OLLYDBG loguea las mismas pero yo cuando me enfrento a packers con muchas excepciones y mas de un Thread como en este caso que crea un thread, antes de llegar al OEP, siempre confio en el logueo directo hecho por mi y no en el que hace OLLYDBG, el tema es que todas las excepciones deben pasar por este punto en el cual acabamos de poner un BP, si lo vemos en OLLYDBG.



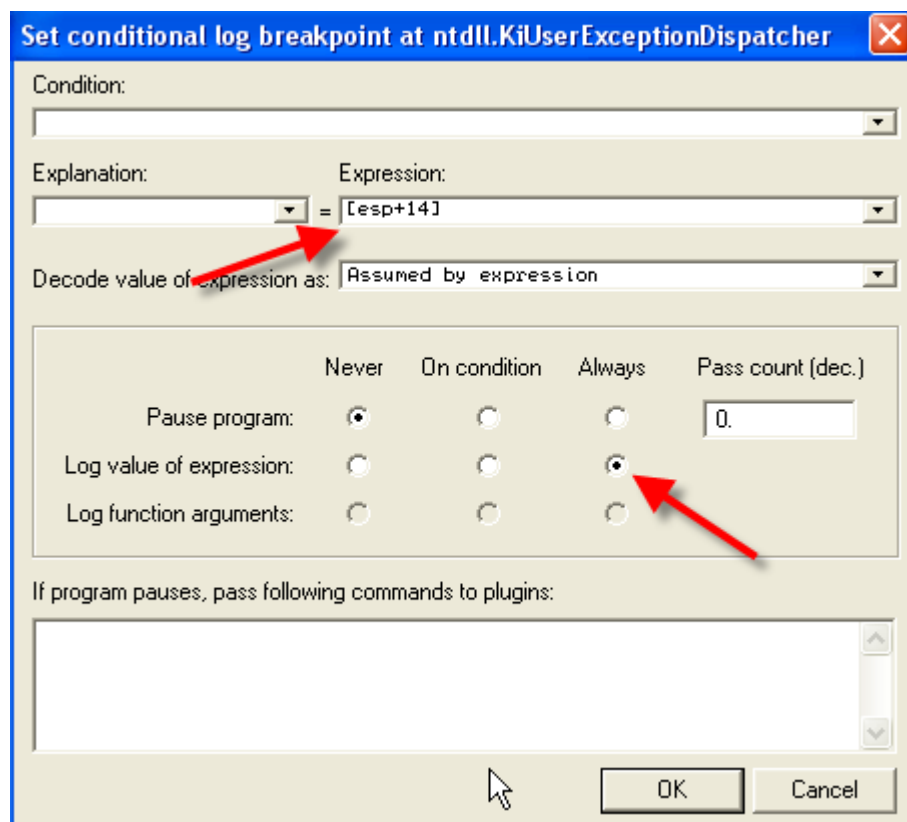
Esta rutina la podemos analizar facilmente de esta forma.



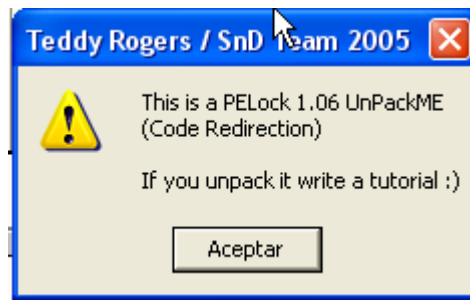
Por lo tanto si coloco un BREAKPOINT CONDICIONAL LOG en vez de un BP y hago que loguee [ESP+14] tendre mi logeador propio jeje.



Por lo tanto hago click derecho, y cambio el BP que habia colocado por un CONDITIONAL LOG.



Pongo la tilde para que siempre loguee el valor de [esp+14] cada vez que pase por alli y que no pare, solamente loguee, y doy RUN.



Vemos que el programa arranca, extrañamente casi ningun packer detecta los BP o BP CONDICIONALES en KiUserExceptionDispatcher.

Bueno miremos nuestra obra en el LOG.

Address	Message
0046B05C	Program entry point
003A5C74	Access violation when writing to [7C81CAA7]
7C91EAE0	Breakpoint at ntdll.KiUserExceptionDispatcher (KiUserApcDispatcher+2C)
003A1CED	Integer division by zero
7C91EAE0	COND: 003A1CED
003A24EB	Access violation when writing to [00000000]
7C91EAE0	COND: 003A24EB
003A24EE	Integer division by zero
7C91EAE0	COND: 003A24EE
003A2698	Access violation when writing to [00000000]
7C91EAE0	COND: 003A2698
003A269B	Integer division by zero
7C91EAE0	COND: 003A269B
003A2851	Access violation when writing to [00000000]
7C91EAE0	COND: 003A2851
003A2854	Integer division by zero
7C91EAE0	COND: 003A2854
003A3374	Access violation when reading [FFFFFFFF]
7C91EAE0	COND: 003A3374
003A33CB	Illegal instruction
7C91EAE0	COND: 003A33CB
003A3374	Access violation when reading [FFFFFFFF]
7C91EAE0	COND: 003A3374
003A33CB	Illegal instruction
7C91EAE0	COND: 003A33CB
003A3374	Access violation when reading [FFFFFFFF]
7C91EAE0	COND: 003A3374
003A33CB	Illegal instruction
7C91EAE0	COND: 003A33CB
003A3652	Access violation when writing to [00000000]
7C91EAE0	COND: 003A3652
003A3655	Integer division by zero
7C91EAE0	COND: 003A3655
003A37DE	Access violation when writing to [00000000]
7C91EAE0	COND: 003A37DE
003A37E1	Integer division by zero
7C91EAE0	COND: 003A37E1
003A3873	Access violation when writing to [00000000]
7C91EAE0	COND: 003A3873
003A3876	Integer division by zero
7C91EAE0	COND: 003A3876
76B00000	Module C:\WINDOWS\system32\WINMM.dll
77D10000	Module C:\WINDOWS\system32\USER32.dll

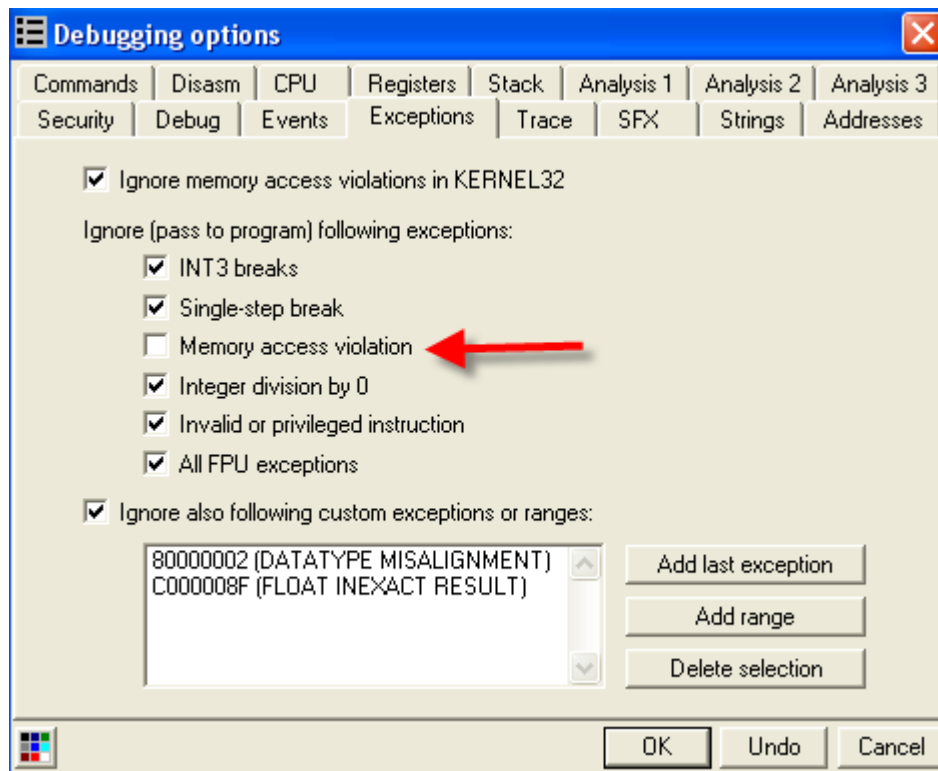
Alli se ven y OLLYDBG logueo bien las que se ven aquí, la ultima esta un poco mas abajo.

```

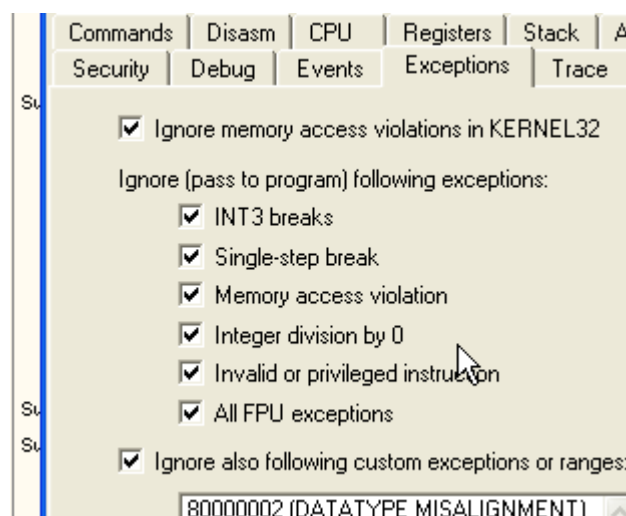
774B0000 Module C:\WINDOWS\system32\ole32.dll
7C81084E Breakpoint at kernel32.7C81084E
003A6744 Access violation when writing to [00000000]
7C91EAE0 COND: 003A6744
7C810856 New thread with ID 00000110 created
770F0000 Module C:\WINDOWS\system32\OLEAUT32.dll
5B150000 Module C:\WINDOWS\system32\uxtheme.dll
746B0000 Module C:\WINDOWS\system32\MSCTF.dll

```

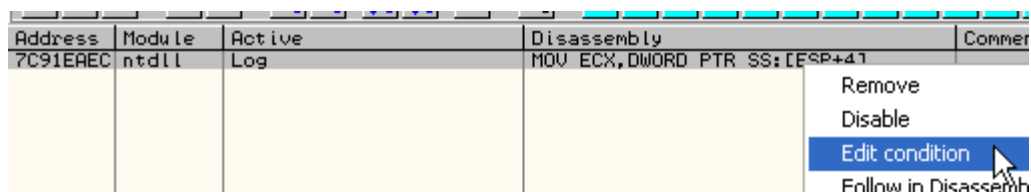
Vemos que la ultima excepcion es una ACCESS VIOLATION en 3A6744, pero alli no podemos poner BP ni HE estos ultimos son detectados por el packer y no corre el programa, veamos

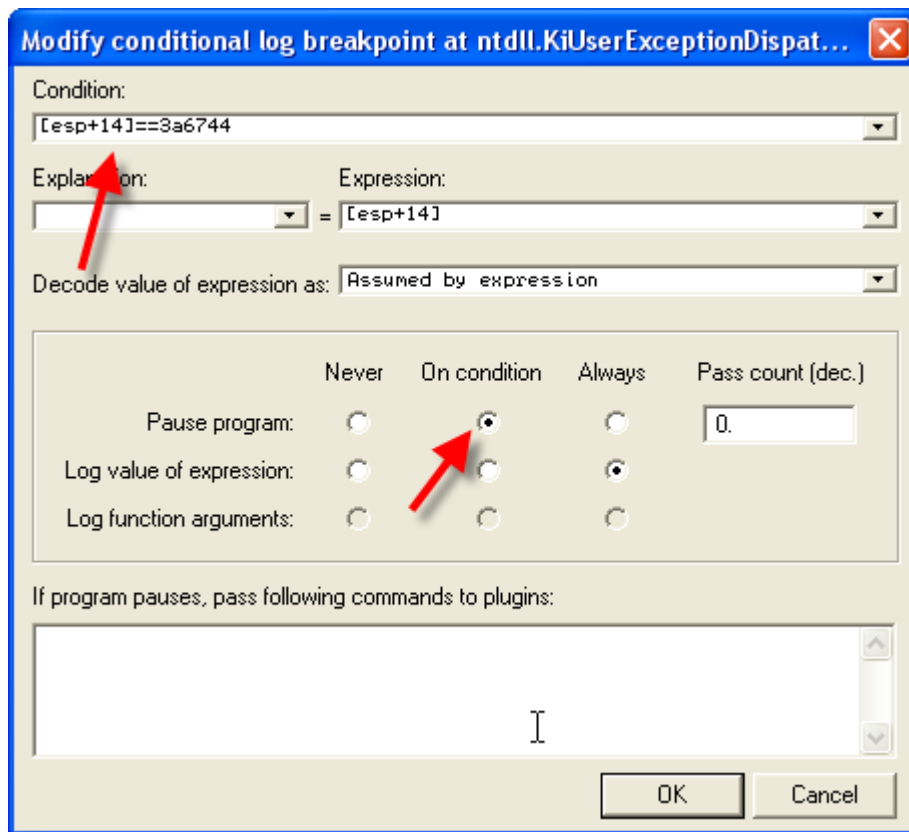


Una opción que tengo es llegar al punto quitando la tilde de MEMORY ACCESS VIOLATIONS y parar en todas hasta que lleguemos a la última, este método si bien funciona, pensemos en algo más genérico y rápido y que sirva para todo tipo de excepciones, así que vuelvo a colocar esta tilde.



Y sin matarme mucho voy a B donde está la lista de breakpoints y aun está mi BP condicional allí, hago click derecho- EDIT CONDITION.





Solamente debo agregar la condicion para que pare, si logueabamos [esp+14] y en el momento que queremos que se detenga ese valor era 3a7644, ya que era la instruccion donde se genero la excepcion que nos mostraba nuestro LOG, pues entonces que pare cuando [esp+14]==3a6744 y listo.

```

77480000| Module C:\WINDOWS\system32\ole32.dll
7C81084E| Breakpoint at kernel32.7C81084E
003A6744| Access violation when writing to [00000000]
7C91EAE0| COND: 003A6744
7C810856| New thread with ID 00000110 created
770F0000| Module C:\WINDOWS\system32\OLEAUT32.dll
5B150000| Module C:\WINDOWS\system32\uxtheme.dll
746B0000| Module C:\WINDOWS\system32\MSCTF.dll

```

Cambio la tilde a que PAUSE PROGRAM-ON CONDITION para que pare cuando se cumpla la condicion que colocamos y damos RUN.

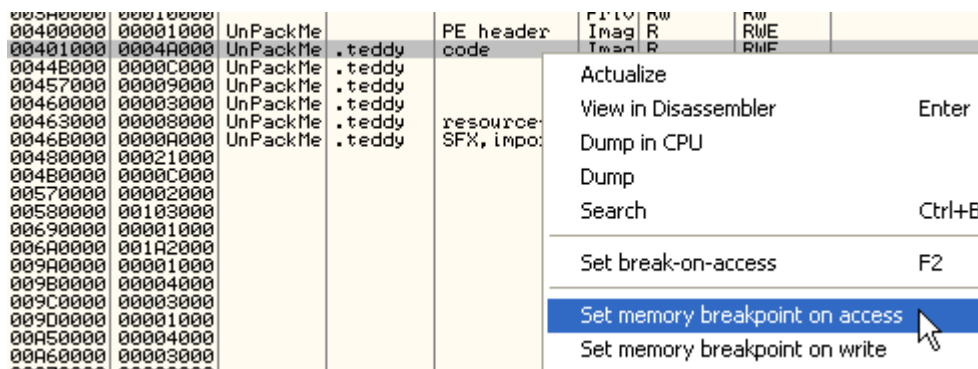
```

7C91EAB0 8B4C24 04 MOV ECX,DWORD PTR SS:[ESP+4]
7C91EAB3 8B1C24 MOV EBX,DWORD PTR SS:[ESP]
7C91EAB4 51 PUSH ECX
7C91EAB5 53 PUSH EBX
7C91EAB6 E8 C78C0200 CALL 7C9477C1
7C91EAB7 0AC0 OR AL,AL
7C91EAB8 74 0C JE SHORT 7C91EB0A
7C91EAB9 5B POP EBX
7C91EABA 59 POP ECX
7C91EABB 6A 00 PUSH 0
7C91EABD 51 PUSH ECX
7C91EABE E8 11EBFFFF CALL 7C91D619
7C91EABF EB 0B JMP SHORT 7C91EB15
7C91EAC0 5B POP EBX
7C91EAC1 59 POP ECX
7C91EAC2 6A 00 PUSH 0
7C91EAC4 51 PUSH ECX
7C91EAC5 53 PUSH EBX
7C91EAC6 E8 3DF7FFFF CALL 7C91E252
7C91EAC7 83C4 EC ADD ESP,-14
7C91EAC8 890424 MOV DWORD PTR SS:[ESP],EAX
7C91EAC9 C74424 04 01 MOV DWORD PTR SS:[ESP+4],1
7C91EACB 895C24 08 MOV DWORD PTR SS:[ESP+8],EBX
7C91EACD C74424 10 00 MOV DWORD PTR SS:[ESP+10],0
7C91EAE2 54 PUSH ESP
7C91EAE3 E8 77000000 CALL 7C91E8AC
7C91EAE4 C2 0800 RETN 8
7C91EAE5 90 NOP

```

Alli paro el programa en la ultima excepcion y sin tener que estar contando una a una, ademas si queremos volver a este punto, pues reiniciamos damos RUN de nuevo y parara aquí las veces que queramos sin tener que estar contando excepciones.

Por supuesto si desde aquí coloco un BPM ON ACCESS en la primera seccion pararemos en el OEP (sera asi? Jeje hagamoslo)



```

00427106 E9 8AD66500 JMP 00A84865
0042710B 0033 ADD BYTE PTR DS:[EBX],DH
0042710D 028A D4891534 ROR BYTE PTR DS:[EDX+341589D4],CL
0042710E E6 45 OUT 45,AL
0042710F 008B C881E1FF ADD BYTE PTR DS:[EBX+FFE181C8],CL
00427110 0000 ADD BYTE PTR DS:[EAX],AL
00427111 0089 0D30E645 ADD BYTE PTR DS:[ECX+45E6300D],CL
00427112 00C1 ADD CL,AL
00427113 E1 08 LOOPDE SHORT 004271FF
00427114 03CA ADD ECX,EDX
00427115 890D 2CE64500 MOV DWORD PTR DS:[45E62C],ECX
00427116 C1E8 10 SHR EAX,10
00427117 A3 28E64500 MOV DWORD PTR DS:[45E628],EAX
00427118 E8 94210000 CALL 004293A0
00427119 85C0 TEST EAX,EAX
0042711A 75 0A JNZ SHORT 0042721A
0042711B 6A 1C PUSH 1C

```

Vemos que ahi para en la primera seccion en el supuesto OEP, pero que pasa aquí?

En este punto comenzaremos a estudiar los stolen bytes.

STOLEN BYTES son bytes del programa que son ejecutados por el packer, lo mismo que STOLEN CODE es codigo del programa ejecutado por el packer.

Los stolen bytes generalmente son las primeras instrucciones del programa original a partir del OEP que el packer borra de la primera seccion y los ejecuta en otra seccion propia, y luego en vez de saltar al OEP salta a la 5 o 6ta linea por ejemplo habiendo ejecutado las anteriores en una seccion propia, generalmente fuera de las secciones del ejecutable.

Para que se hace esto?, muy simple si yo dumpeo aquí y reparo la IAT y coloco como OEP 4271D6, el programa no arranca pues le faltan las primeras lineas, que cuando corrio con el packer, el maldito la ejecuto antes en su propia seccion y luego salto aquí, por lo cual en el dumpeado, esas lineas no se ejecutaran.

Que debo hacer?

Un metodo es tratar de tracear la rutina del packer, luego de la ultima excepcion y llegar al OEP FALSO traceando y guardando en un txt todo lo que ejecuto, cosa de poder analizar que fue lo que ejecuto antes de saltar al ahora llamando falso OEP de 4271d6.

Por eso realice el metodo de las excepciones de esa forma, ya que muchas veces el traceo en OLLYDBG tiene unas cuantas posibilidades, tildes y opciones que intentar, y si cada vez que tengo que llegar a la ultima excepcion tengo que contarlas una a una, me volvere loco, por lo cual, tengo la forma de que pare en el punto solo dando RUN, y ya estamos nuevamente en la ultima excepcion.

Miremos antes de reiniciar un poco el panorama, cual es la forma de sospechar que hay stolen bytes?, pues mirando el stack.

Reiniciemos el programa.

Si veo el stack al inicio.

Esta en 12FFc4 en mi maquina, y sea cual sea el valor en la suya, en el OEP salvo casos muy estramboticos, el programa se debe iniciar con el stack en 12FFc4 o cercano, todo lo que haya en el falso OEP, arriba de esta direccion sera codigo que ya se ejecuto desde el OEP VERDADERO hasta el FALSO

Veamos lleguemos nuevamente al falso OEP.

```

0 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010217 (NO,B,NE,BE,NS,PE,GE,G)

0012FF44 CC6EA31E
0012FF48 003A6C9B
0012FF4C 00000000
0012FF50 746E656D
0012FF54 6E612073
0012FF58 65532064
0012FF5C 6E697474
0012FF60 525C7367
0012FF64 72616369
0012FF68 455C6F64
0012FF6C 69726373
0012FF70 69726F74
0012FF74 6E555C6F
0012FF78 6B636150
0012FF7C 505F654D
0012FF80 636F4C45
0012FF84 302E316B
0012FF88 2E642E36
0012FF8C 22657865
0012FF90 0056A028
0012FF94 C0000034
0012FF98 00000000
0012FF9C 004A6C04
0012FFA0 0012FF44
0012FFA4 00000048
0012FFA8 0012FFB8 Pointer to next SEH record
0012FFAC 004292C8 SE handler
0012FFB0 00450E60 UnPackMe.00450E60
0012FFB4 FFFFFFFF
0012FFB8 0012FFE0 Pointer to next SEH record
0012FFBC 7C816D4F SE handler
0012FFC0 0012FFF0
0012FFC4 7C816D4F RETURN to kernel32.7C816D4F
0012FFC8 7C920738 ntdll.7C920738
0012FFCC FFFFFFFF
0012FFD0 7FFD4000
0012FFD4 8054A938
0012FFD8 0012FFC8
0012FFDC 81A7020
0012FFE0 FFFFFFFF End of SEH chain
0012FFE4 7C8399F3 SE handler
0012FFE8 7C816D58 kernel32.7C816D58
0012FFEC 00000000
0012FFF0 00000000
0012FFF4 00000000
0012FFF8 0046B05C UnPackMe.<ModuleEntryPoint>
0012FFFC 00000000

```

Vemos que si este fuera el verdadero OEP, el stack debería estar en 12FFc4 o cerca, todo lo que hay arriba son instrucciones que ya se ejecutaron del programa, unas cuantas líneas que desde el verdadero OEP que el packer quitó del programa y lo trasladó a otra sección, se ejecutaron instrucciones que deberían estar arriba del falso OEP, y que fueron borradas, si recordamos este crackme que es el mismo que siempre venimos estudiando el OEP estaba en 4271b0, si miramos esa zona aquí.

```

004271AC 90 NOP
004271AD 90 NOP
004271AE 90 NOP
004271AF 90 NOP
004271B0 DA6D B6 FISUBR DWORD PTR SS:[EBP-4A]
004271B3 DB6D B6 FLD TBYTE PTR SS:[EBP-4A]
004271B6 5B POP EBX
004271B7 2D 168BC5E2 SUB EAX,E2C58B16
004271BC ^ 71 B8 JNO SHORT 00427176 UnPackMe.00427176
004271BE DC6E 37 FSUBR QWORD PTR DS:[ESI+37]
004271C1 9B WAIT
004271C2 4D DEC EBP
004271C3 26:93 XCHG EAX,EBX
004271C5 49 DEC ECX
004271C6 A4 MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004271C7 52 PUSH EDX
004271C8 A9 542A95CA TEST EAX,CA952A54
004271CD 65:B2 59 MOV DL,59 Superfluous prefix
004271D0 AC LODS BYTE PTR DS:[ESI]
004271D1 D6 SALC
004271D2 ^ EB F5 JMP SHORT 004271C9 UnPackMe.004271C9
004271D4 7A BD JPE SHORT 00427193 UnPackMe.00427193
004271D6 - E9 8AD66500 JMP 00A84865
004271D8 0033 ADD BYTE PTR DS:[EBX],DH
004271DA D28A D4891534 ROR BYTE PTR DS:[EDX+341589D4],CL

```

Vemos que el packer reemplazo los bytes originales del OEP e instrucciones siguientes con basura, que ademas nunca se ejecuto, pues con el BPM ON ACCESS paramos en la primera linea de codigo ejecutada en esta seccion y no paro en esos bytes sino mas abajo en 4271d6.

Pues cuales son los metodos para hallar los stolen bytes, pues hay muchos realmente, el mas clasico es tracear luego de regresar de la ultima excepcion, probemos ese reiniciemos y volvamos a la ultima excepcion.

La misma se habia generado en 3A6744, por lo cual no me interesa en este caso el manejador de excepciones, asi que pondre un BP en el retorno, asi salteo el mismo.

Ahora doy RUN, si alguien quiere saber donde retornara al programa para poner directamente un BP alli, pues veamos en el stack.

Tenemos el parametro que nos marca aquí el inicio de la estructura CONTEXT, miremosla en el DUMP.

1t011.K1USERHPCUIspatcher+43

Address	Hex dump	ASCII
0012FCE4	3F 00 01 00 00 00 00 00	? . 0
0012FCEC	00 00 00 00 00 00 00 00
0012FCF4	00 00 00 00 00 00 00 00
0012FCFC	00 00 00 00 7F 02 FF FF Δ @
0012FD04	20 40 FF FF FF FF FF FF	@
0012FD0C	4D 29 B2 67 1B 00 5E 05	M) 9g+. ^ *
0012FD14	28 28 5F 05 23 00 FF FF	((_ * #.
0012FD1C	00 00 00 00 04 01 05 01 ♦ @ * @
0012FD24	E0 BC 00 00 00 00 00 00	0 ^
0012FD2C	00 00 00 00 00 00 00 00
0012FD34	00 00 00 00 00 00 00 00
0012FD3C	00 00 00 00 00 00 00 00
0012FD44	00 00 00 00 00 00 00 00
0012FD4C	00 00 00 00 00 00 00 00
0012FD54	00 00 00 00 00 00 00 00
0012FD5C	00 00 00 80 FF 3F 00 00 C ?
0012FD64	00 00 00 00 00 80 FF 3F C ?
0012FD6C	00 00 00 00 00 00 00 00
0012FD74	3B 00 00 00 23 00 00 00	; . . . # . .
0012FD7C	23 00 00 00 1E A3 6E CC	# . . . Δ un f
0012FD84	00 00 00 00 26 54 3A 0D & T : .
0012FD8C	24 00 00 00 00 00 00 00	\$
0012FD94	00 00 00 00 2D 06 3A 00 - * .
0012FD9C	46 67 3A 00 1B 00 00 00	F 6 7 Δ 0 0 1 B
0012FDA4	46 02 01 00 B0 F4 12 00	F 0 2 . 0 1 0 0 B 0 F 4 1 2
0012FDAC	23 00 00 00 7F 02 20 40	# . . . Δ @ @
0012FDB4	00 00 5E 05 4D 29 B2 67	. . ^ * M) 9g
0012FDBC	00 00 00 00 00 00 00 00
0012FDC4	00 00 FF FF 80 1F 00 00 C ^
0012FDCC	00 00 00 00 00 00 00 00
0012FDD4	04 01 05 01 E0 BC 00 00	♦ @ * 0 0 ^
0012FDDC	00 00 00 00 00 00 00 00
0012FDE4	00 00 00 00 00 00 00 00

Los que ya conocen la estructura CONTEXT saben que ese es el EIP de la misma , entre los valores de los restantes registros que tendra el programa al regresar de la excepcion, como ya saben mas adelante le dedicaremos un estudio detallado de la estructura CONTEXT por ahora, solo mirando alli saben donde volvera, si no quieren complicarse la vida buscando en el CONTEXT, poniendo un BPM ON ACCESS en la seccion donde se produjo la excepcion.

00380000	00001000					Priv	RWE	RWE
00390000	00001000					Priv	RWE	RWE
003A0000	00010000					Priv	RWE	RWE
00400000	00001000	UnPackMe	.tedc					
00401000	0004A000	UnPackMe	.tedc					
0044B000	0000C000	UnPackMe	.tedc					
00457000	00009000	UnPackMe	.tedc					
00460000	00003000	UnPackMe	.tedc					
00463000	00008000	UnPackMe	.tedc					
0046B000	0000A000	UnPackMe	.tedc					
00480000	00021000							
00480000	0000C000							
00570000	00002000							
00580000	00103000							
00690000	00001000							
006A0000	001B5000							
009A0000	00001000							
009B0000	00001000							

Actualize

Dump in CPU

Dump

Search Ctrl+F

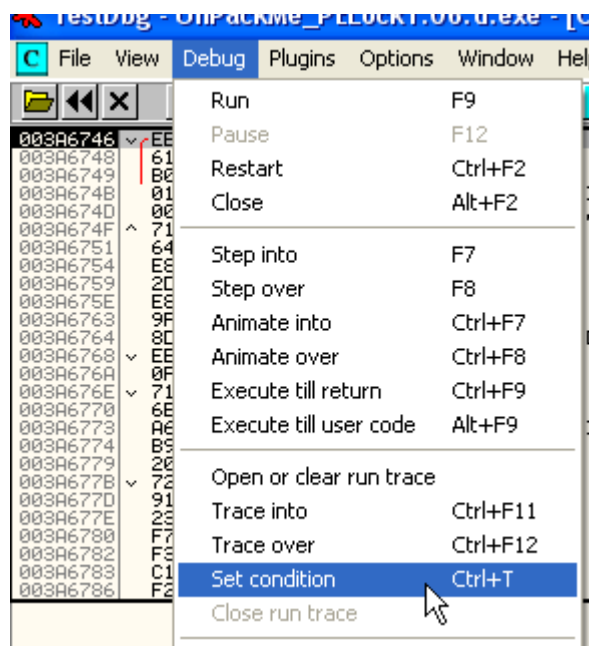
Set break-on-access F2

Set memory breakpoint on access

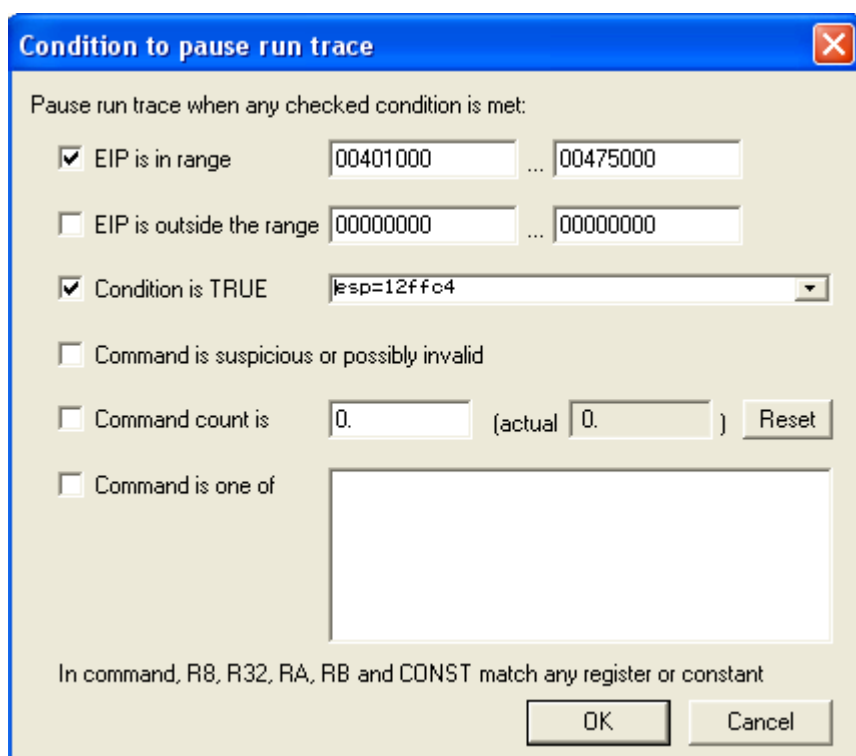
Set memory breakpoint on write

Y dando RUN vemos que para en el mismo lugar

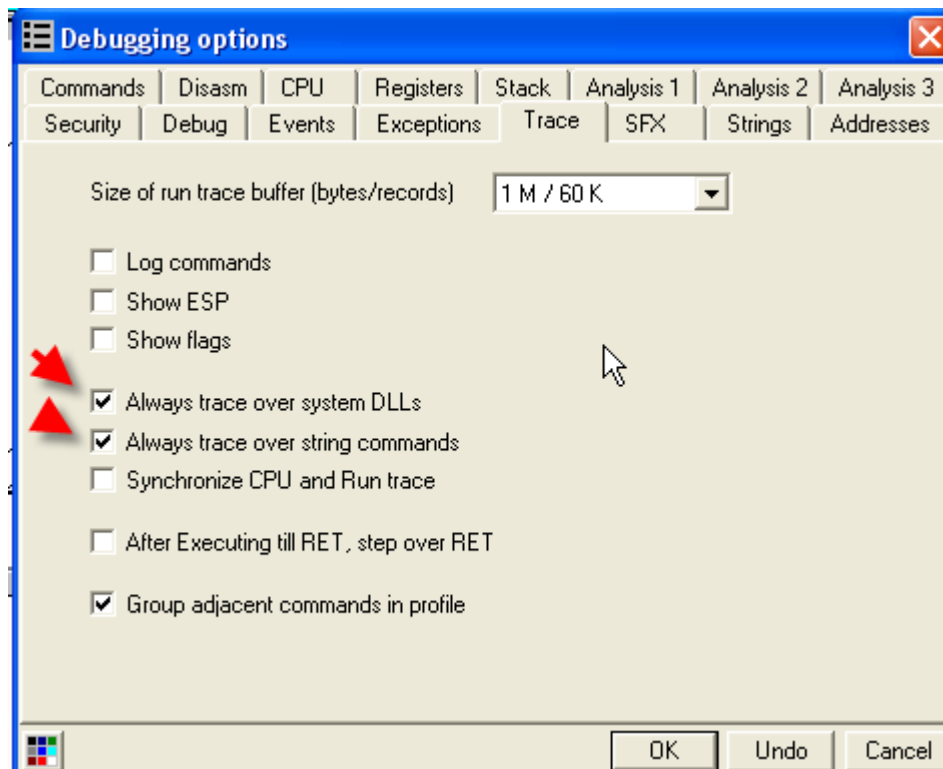
L E M T W H C			
003A6746	EB 02	JMP SHORT 003A674A	
003A6748	61	POPAD	
003A6749	B0 E8	MOV AL,0E8	
003A674B	0100	ADD DWORD PTR DS:[EAX],EAX	
003A674D	0000	ADD BYTE PTR DS:[EAX],AL	
003A674F	71 8D	JNO SHORT 003A66DE	
003A6751	64 24 04	AND AL,4	
003A6754	E8 01000000	CALL 003A675A	Super
003A6759	2D 8F4424FC	SUB EAX,FC24448F	
003A675E	E8 01000000	CALL 003A6764	



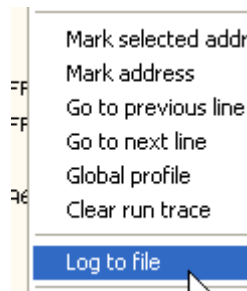
Allí colocaremos las condiciones del traceo, hay miles posibles, ya que puedo colocar que chequee los valores iniciales de los registros y cuando se cumplan que todos están igual que antes de correr el programa, estare en el OEP, por ejemplo, pero miremos si funciona estas dos.



La primera parara en el falso OEP pero traceando, lo cual puede dejarme ver en el LOGUEO del traceo, las instrucciones que ejecutó antes de llegar al mismo, la segunda tilde, es para ver si paramos en el momento del VERDADERO OEP ya que alli vimos que esp= 12ffc4.

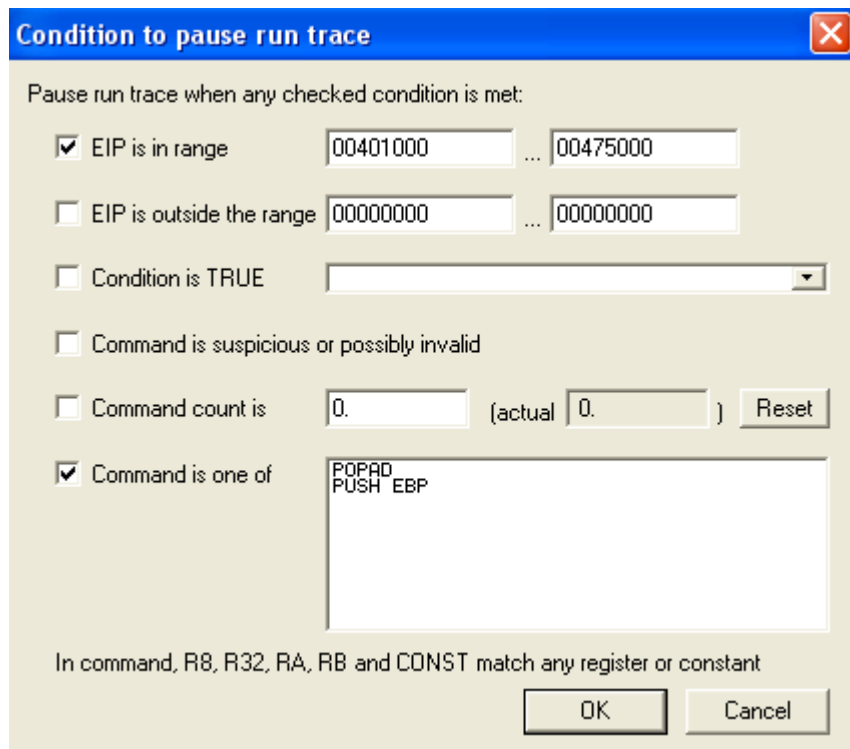


Ponemos las dos tildes en la configuracion para intentar si funciona con ellas, lo cual es lo mas probable, si no, probaremos quitandolas.

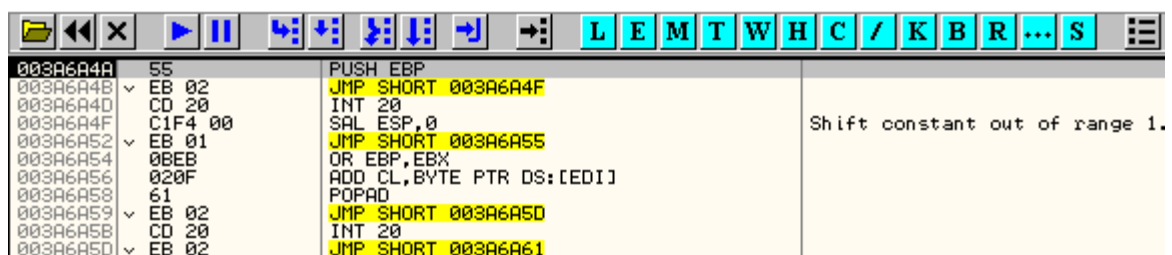


Tambien hacemos que loguee a una fila.

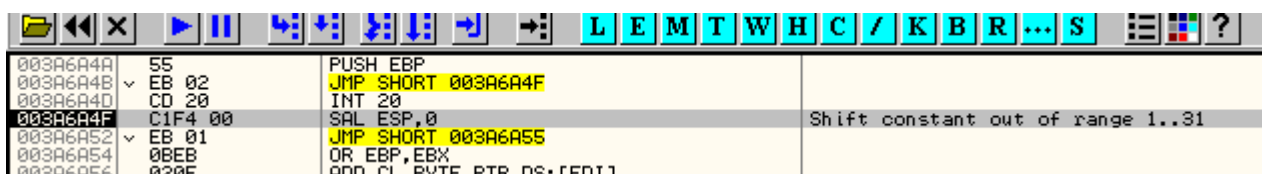
Apertamos TRACE INTO y me molesta la condicion de ESP que me hace parar muchas veces, si la quito y sigo con el trace into, para en el falso OEP, y en la fila de texto tengo todas las instrucciones que ejecuto antes, pero no me funciona lo de parar en el VERDADERO OEP, bueno mala suerte.



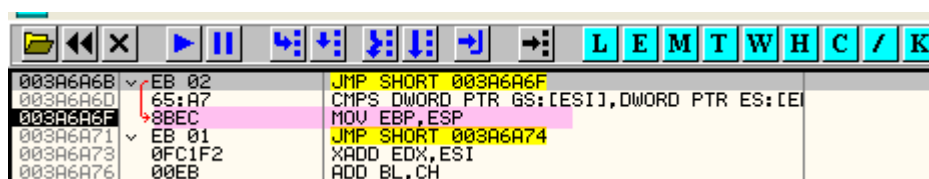
Una de las posibilidades que se puede intentar es poner que pare en cualquier PUSH EBP o POPAD que generalmente o son el OEP en el primer caso, o la recuperacion de registros en otros casos, esto puede funcionar o no, ya que muchas veces los push EBP son emulados, pero bueno veamos, pongamos a tracear.



Podria ser el verdadero OEP, traceemos un poco.



Por supuesto como instrucciones verdaderas no debemos considerar los saltos ni los SAL o cualquier otra instrucción basura.



Esta parece ser la segunda instrucción faltante, al no ser emuladas y estar copiadas directamente del original es facil detectarlas.

003A6A8E	CD 20	INT 20	
003A6A90	C1F4 00	SAL ESP,0	
003A6A93	6A FF	PUSH -1	Shift
003A6A95	C1F4 00	SAL ECX,0	Shift
003A6A98	EB 02	JMP SHORT 003A6A9C	

Y asi seguimos traceando y copiando las buenas instrucciones que va ejecutando hasta llegar al falso OEP.

003A6AB5	EB 02	JMP SHORT 003A6AB9	
003A6AB7	CD 20	INT 20	
003A6AB9	68 600E4500	PUSH 450E60	
003A6ABE	EB 02	JMP SHORT 003A6AC2	
003A6AC0	04 72	ADD AL,72	
003A6AC2	EB 01	JMP SHORT 003A6AC5	

003A6AE0	EB 01	JMP SHORT 003A6AE3	
003A6AE2	40	INC EAX	
003A6AE3	68 C8924200	PUSH 4292C8	
003A6AE8	EB 02	JMP SHORT 003A6AEC	
003A6AEA	BB F0EB02CD	MOV EBX,CD02EBFD	
003A6AEF	20EB	AND BL,CH	

003A6B07	EB 02	JMP SHORT 003A6B0D	
003A6B08	CD 20	INT 20	
003A6B0D	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
003A6B13	EB 02	JMP SHORT 003A6B17	
003A6B15	CD 20	INT 20	

003A6B35	50	PUSH EAX	
003A6B36	C1F7 00	SAL EDI,0	
003A6B39	EB 02	JMP SHORT 003A6B3D	
003A6B3B	CD 20	INT 20	

003A6B55	EB 02	JMP SHORT 003A6B59	
003A6B57	CD 20	INT 20	
003A6B59	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
003A6B60	C1F5 00	SAL EBP,0	
003A6B63	C1F3 00	SAL EBX,0	
003A6B66	EB 02	JMP SHORT 003A6B6A	

003A6B7E	EB 02	JMP SHORT 003A6B82	
003A6B80	CD 20	INT 20	
003A6B82	C1F0 00	SAL EAX,0	
003A6B85	83C4 00	ADD ESP,-58	
003A6B88	C1F6 00	SAL ESI,0	
003A6B8B	EB 01	JMP SHORT 003A6B8E	
003A6B8D	0EB01	POR MM0,QWORD PTR DS:[ECX]	
003A6B8F	0EB02	POR MM0,QWORD PTR DS:[ECX+1]	

003A6BA0	CD 20	INT 20	
003A6BA8	EB 02	JMP SHORT 003A6BAC	
003A6BAA	CD 20	INT 20	
003A6BAC	53	PUSH EBX	
003A6BAD	C1F1 00	SAL ECX,0	
003A6BB0	EB 02	JMP SHORT 003A6BB4	
003A6BB2	3C 0B	CMP AL,0B	
003A6BB4	C1F6 00	SAL ESI,0	

003A6BCF	56	PUSH ESI	
003A6BD0	EB 02	JMP SHORT 003A6BD4	
003A6BD2	CD 20	INT 20	
003A6BD4	EB 02	JMP SHORT 003A6BD8	
003A6BD6	CD 20	INT 20	

003A6BF4	CD 20	INT 20	
003A6BF6	57	PUSH EDI	
003A6BF7	EB 02	JMP SHORT 003A6BFB	
003A6BF9	CD 20	INT 20	
003A6BFB	FA 01	JMP SHORT 003A6BFF	

003A6C13	EB 02	JMP SHORT 003A6C17
003A6C15	65:46	INC ESI
003A6C17	EB 02	JMP SHORT 003A6C1B
003A6C19	CD 20	INT 20
003A6C1B	55 E8	MOV DWORD PTR SS:[EBP-18],ESP
003A6C1D	EB 02	JMP SHORT 003A6C22
003A6C20	0FA4C1 F1	SHLD ECX,EAX,0F1

003A6C65	CD 20	INT 20
003A6C67	68 D6714200	PUSH 4271D6
003A6C6C	EB 02	JMP SHORT 003A6C70
003A6C6E	CD 02EB	ADD CH,BL
003A6C71	0265 A0	ADD AH,BYTE PTR SS:[EBP-60]

Y aquí llegamos al falso OEP ya que hizo PUSH 4271D6 que era la direccion del falso OEP y luego RET así que salta allí.

003A6C93	C3	RETN
003A6C94	EB 02	JMP SHORT 003A6C98
003A6C96	0F50C1	MOVMSKPS EAX,XMM1
003A6C99	F4	HLT

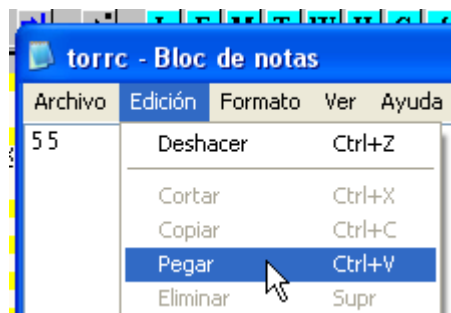
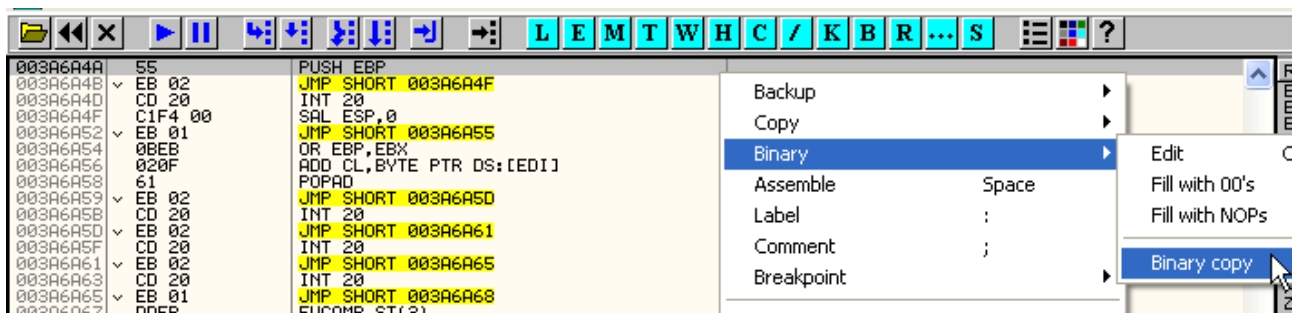
004271D6	E9 8AD66500	JMP 00A84865	
004271D8	0033	ADD BYTE PTR DS:[EBX],DH	
004271DB	D28A D4891534	ROR BYTE PTR DS:[EDX+341589D4],CL	
004271E3	E6 45	OUT 45,AL	I/O command
004271E5	008B C881E1FF	ADD BYTE PTR DS:[EBX+FFE181C8],CL	
004271EB	0000	ADD BYTE PTR DS:[EAX],AL	
004271ED	0089 0D30E645	ADD BYTE PTR DS:[ECX+45E6300D],CL	
004271F3	00C1	ADD CL,AL	

Por supuesto ahora hay que copiar todos los bytes de las instrucciones faltantes fijarse cuantos son y pegarlos antes del OEP FALSO veamos en el DUMP:

Address	Hex dump	ASCII
004271AE	90 90 DA 6D B6 DB 6D B6	ÉÉrmā mā
004271B6	5B 2D 16 8B C5 E2 71 B8	[_ i+0q0
004271BE	DC 6E 37 9B 4D 26 93 49	an7x1%oI
004271C6	A4 52 A9 54 2A 95 CA 65	ARBT*o=e
004271CE	B2 59 AC D6 EB F5 7A BD	W%iuSzC
004271D6	E9 8A D6 65 00 00 33 D2	Ueie.3E
004271DE	8A D4 89 15 34 E6 45 00	ééé34vE.
004271E6	8B C8 81 E1 FF 00 00 00	iüüß ...
004271EE	89 0D 30 E6 45 00 C1 E1	é.0vE.1ß
004271F6	08 03 CA 89 0D 2C E6 45	üüé.üE.
004271FE	00 C1 E8 10 A3 28 E6 45	.1ßüüüE
00427206	00 E8 94 21 00 00 85 C0	.üüü...äL
0042720E	75 0A 6A 1C E8 49 01 00	u.jLüüü.
00427216	00 83 C4 04 F8 D1 2F 00	ü-üüüü

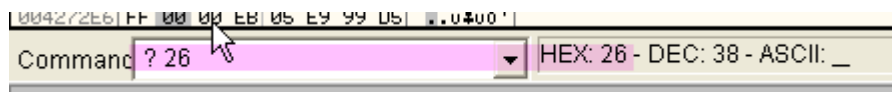
Allí vemos la zona del falso OEP, y arriba donde deben ir los bytes faltantes, como ya verificamos que es todo correcto volvemos para atrás apretando el menos hasta el verdadero OEP

Y ahora si, hare binary copy de cada linea que ejecuto y voy copiando los bytes en un bloc de notas o notepad sin colocar el push final y el ret pues esos son un salto al OEP falso no son stolen bytes.



55 8B EC 6A FF 68 60 0E 45 00 68 C8 92 42 00 64 A1 00 00 00 00 50 64 89 25 00 00 00 00 83 C4 A8 53 56 57 89 65 E8

Son 38 stolen bytes que en hexadecimal es 26



asi que le resto a la direccion del falso OEP esos 26 bytes faltantes.



Asi hallamos la direccion de donde deberia haber estado el verdadero OEP si el packer no lo hubiera quitado.

O sea que a partir de 4271b0 debemos pegar los bytes que hallamos.

Address	Hex dump	ASCII
004271B0	DA 6D B6 DB 6D B6 5B 2D	rmâma[-
004271B8	16 8B C5 E2 71 B8 DC 6E	i+0d n
004271C0	37 9B 4D 26 93 49 A4 52	7M%ôIR
004271C8	A9 54 2A 95 CA 65 B2 59	BT*ôeV
004271D0	AC D6 EB F5 7A BD E9 8A	%iû3æüê
004271D8	D6 65 00 00 33 D2 8A D4	ie..3ëëë
004271E0	89 15 34 E6 45 00 8B C8	ë34pE.i
004271E8	81 E1 FF 00 00 00 89 0D	ûp...ë.
004271F0	30 E6 45 00 C1 E1 08 03	0pE.-p
004271F8	CA 89 0D 2C E6 45 00 C1	æë..pE.+
00427200	E8 10 A3 28 E6 45 00 E8	pü(pE.p
00427208	94 21 00 00 85 C0 75 0A	ô!..â'u.

Marco los bytes del notepad y hago COPY y aquí hago click derecho BINARY PASTE.

Address	Hex dump	ASCII
004271B0	55 8B EC 6A FF 68 60 0E	Uiyj h'A
004271B8	45 00 68 C8 92 42 00 64	E.hEB.d
004271C0	A1 00 00 00 00 50 64 89	i....Pdë
004271C8	25 00 00 00 00 83 C4 A8	%....â-c
004271D0	53 56 57 89 65 E8 E9 8A	SUWëepüê
004271D8	D6 65 00 00 33 D2 8A D4	ie..3ëëë
004271E0	89 15 34 E6 45 00 8B C8	ë34pE.i
004271E8	81 E1 FF 00 00 00 89 0D	ûp...ë.
004271F0	30 E6 45 00 C1 E1 08 03	0pE.-p
004271F8	CA 89 0D 2C E6 45 00 C1	æë..pE.+
00427200	E8 10 A3 28 E6 45 00 E8	pü(pE.p
00427208	94 21 00 00 85 C0 75 0A	ô!..â'u.
00427210	6A 1C E8 49 01 00 00 83	jLpI0..â
00427218	C4 04 E8 D1 2F 00 00 85	-p0/..â

Alli se copiaron miremos como quedo el codigo

Address	Hex dump	Disassembly	Comment
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	E9 8AD66500	JMP 00A84865	
004271DB	0033	ADD BYTE PTR DS:[EBX],DH	
004271DD	D28A D4891534	ROR BYTE PTR DS:[EDX+341589D4],CL	
004271E3	E6 45	OUT 45,AL	I/O command
004271E5	008B C881E1FF	ADD BYTE PTR DS:[EBX+FFE181C8],CL	
004271EB	0000	ADD BYTE PTR DS:[EAX],AL	

Ahora si quedo mejor, podria dumpear desde aquí y cambiar el OEP a 4271B0 con lo cual estarian solucionados los stolen bytes.

En la parte siguiente vamos a comenzar con scripts los cuales son necesarios para reparar esta IAT y demas lios que cometio el packer.

Hasta la parte 40

Ricardo Narvaja

04/04/06