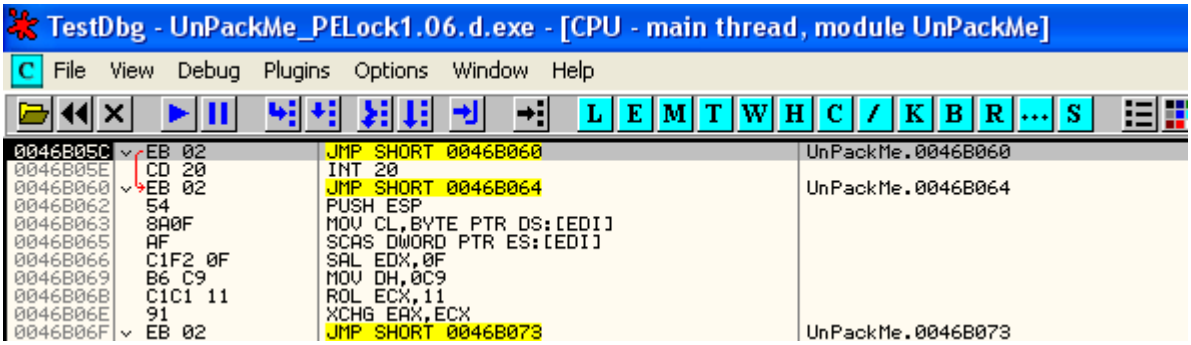
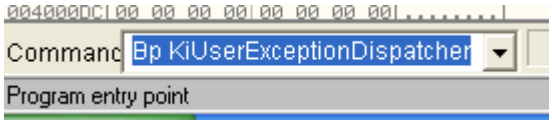
# INTRODUCCION AL CRACKING CON OLLYDBG PARTE 39

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 regenial..

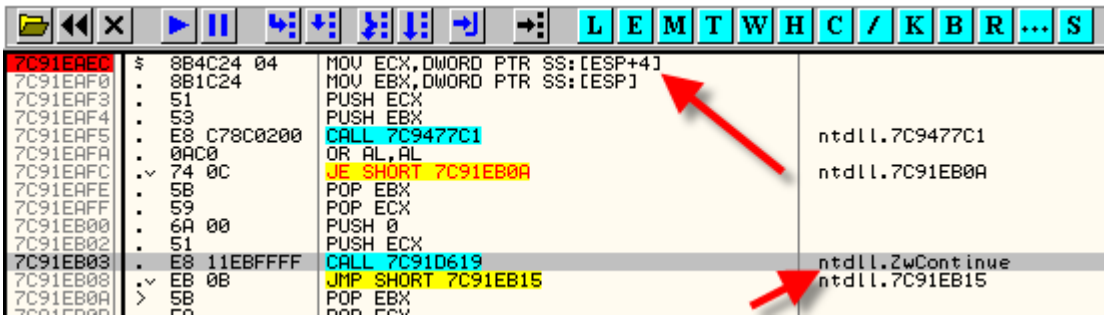
Aquí estamos en el inicio del programa paraditos en el OLLYDBG



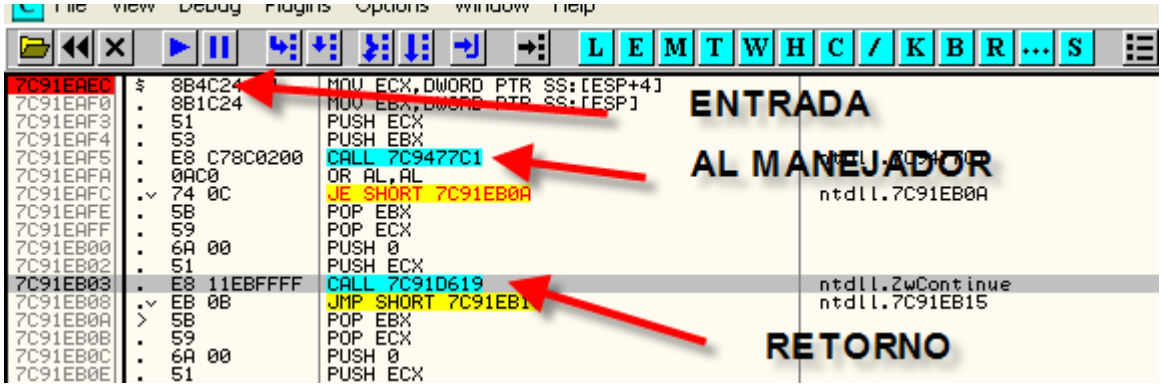

Usaremos una variacion del metodo de las excepciones mas adelante veran porque, en este caso en vez de confiar en el LOG de OLLYDBG haremos nuestro propio LOG de excepciones, porque en estos packers una excepcion que se te pasa o no sale en el LOG o que es camuflada nos puede hacer mucho lio, lo cual 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.



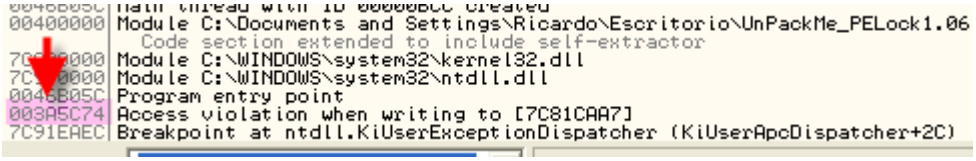

Esta rutina la podemos analizar facilmente de esta forma.




O sea el punto marcado con entrada es el inicio de la rutina que maneja las excepciones en la ntdll.dll, el call marcado con AL MANEJADOR, en un call que detro del mismo, luego de muchas vueltas salta al manejador de excepciones que nos indica VIEW-SEH CHAIN, luego de ejecutar el manejador de excepciones, retorna a esta rutina y llega hasta RETORNO donde dentro de ese call vuelve a continuar la ejecucion del programa.

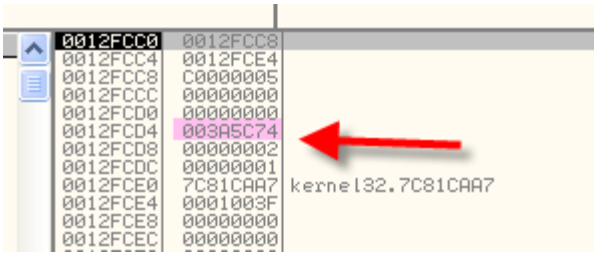
Aclaro por si alguien se pone a tracear, que cuando salto de aquí al programa, no podemos llegar traceando, pues hay rutinas RING0 intermedias que OLLYDBG no puede tracear, en ese caso si queremos parar en el programa desde aquí debemos poner un BP en el manejador o retorno o tambien BPM ON ACCESS en toda la seccion donde se produjo la excepcion, para que OLLYDBG despues de que termine de ejecutarse la parte RING0, pare en el programa nuevamente, digo esto porque no faltara el curioso que quiera tracear desde aquí hasta el programa, y no se puede con OLLYDBG se llega un punto donde se salta a RING0 donde no se puede continuar traceando.

Bueno igual con los puntos que tenemos, ya es suficiente, si damos RUN y para en la primera excepcion.

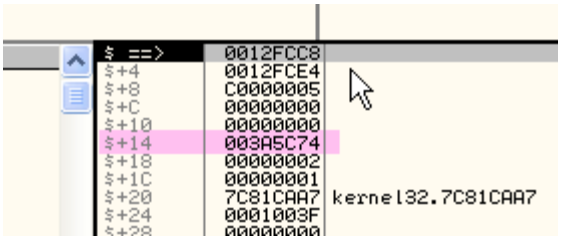



Vemos en el LOG que la misma se produjo en mi maquina en 3a5c74 y luego de ella como teniamos todas las tildes marcadas en DEBUGGING OPTIONS-EXCEPTIONS, paro directamente en el BP que colocamos.

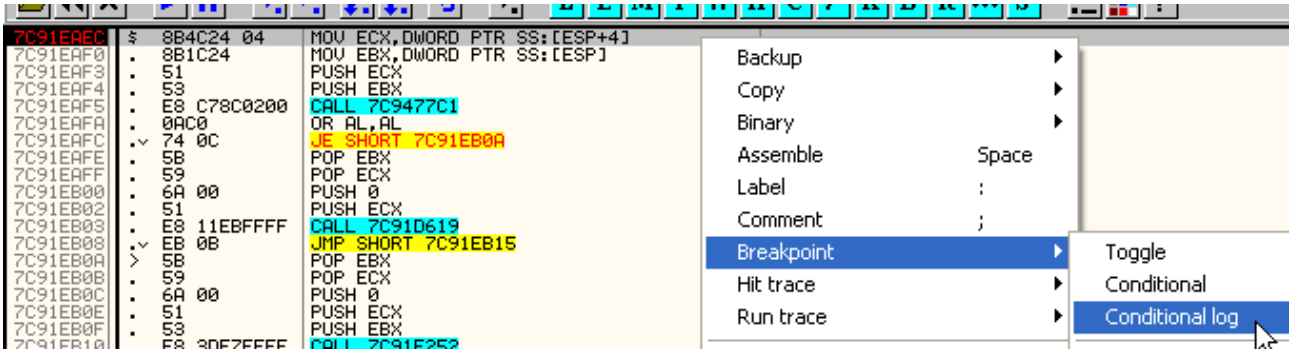
Ahora si miramos el stack



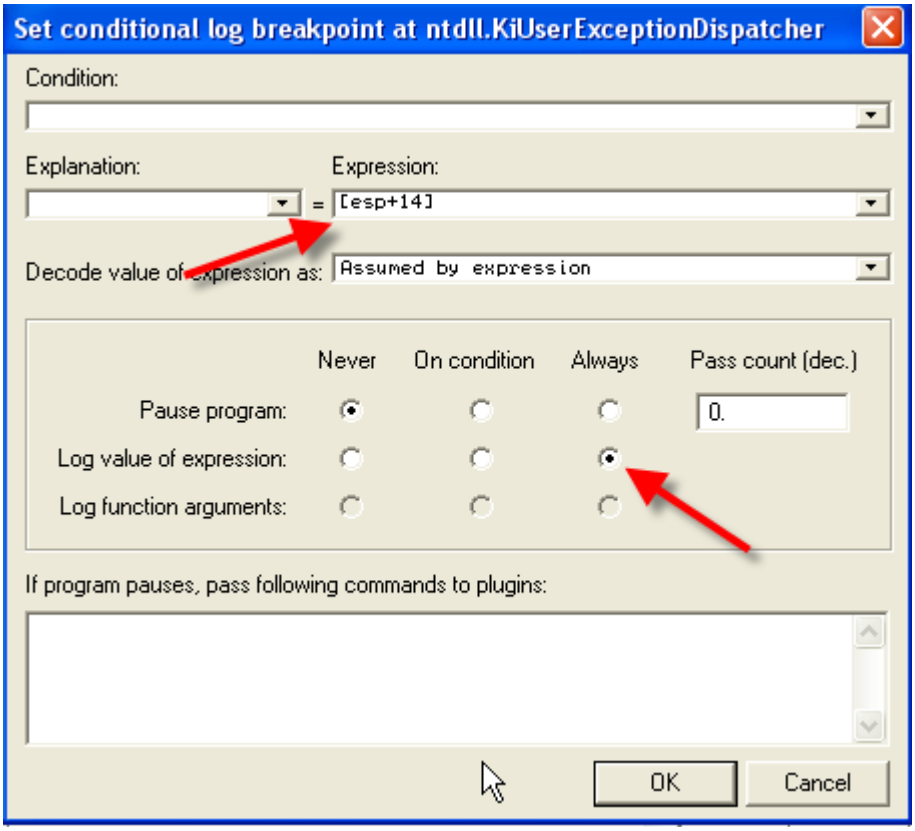

Vemos que unas lineas mas abajo esta la direccion donde se produjo la excepcion, exactamente en [esp +14]



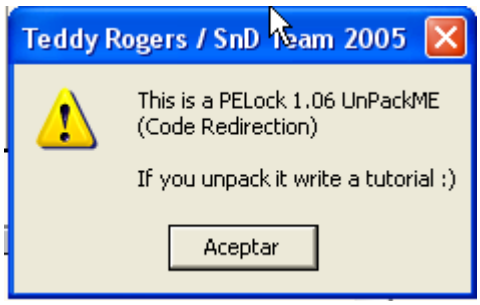

Por lo tanto si coloco un BREAKPOINT CONDICIONAL LOG en vez de un BP y hago que loguee [ESP+14] tendre mi logueador propio jeje.




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.

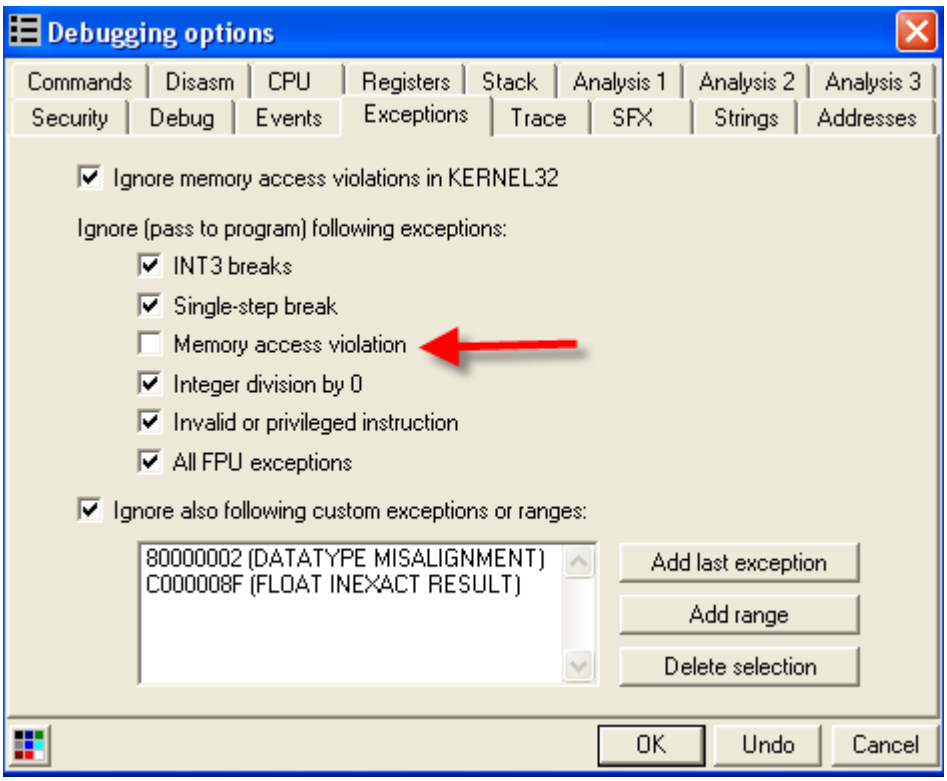
Bueno miremos nuestra obra en el LOG.

```
Address  | Message
0046B05C | Program entry point
003A5C74 | Access violation when writing to [7C81CAA7]
7C91EAEC | Breakpoint at ntdll.KiUserExceptionDispatcher (KiUserApcDispatcher+2C)
003A1CED | Integer division by zero
7C91EAEC | COND: 003A1CED
003A24EB | Access violation when writing to [00000000]
7C91EAEC | COND: 003A24EB
003A24EE | Integer division by zero
7C91EAEC | COND: 003A24EE
003A2698 | Access violation when writing to [00000000]
7C91EAEC | COND: 003A2698
003A269B | Integer division by zero
7C91EAEC | COND: 003A269B
003A2851 | Access violation when writing to [00000000]
7C91EAEC | COND: 003A2851
003A2854 | Integer division by zero
7C91EAEC | COND: 003A2854
003A3374 | Access violation when reading [FFFFFFFF]
7C91EAEC | COND: 003A3374
003A33CB | Illegal instruction
7C91EAEC | COND: 003A33CB
003A3374 | Access violation when reading [FFFFFFFF]
7C91EAEC | COND: 003A3374
003A33CB | Illegal instruction
7C91EAEC | COND: 003A33CB
003A3374 | Access violation when reading [FFFFFFFF]
7C91EAEC | COND: 003A3374
003A33CB | Illegal instruction
7C91EAEC | COND: 003A33CB
003A3374 | Access violation when reading [FFFFFFFF]
7C91EAEC | COND: 003A3374
003A33CB | Illegal instruction
7C91EAEC | COND: 003A33CB
003A3652 | Access violation when writing to [00000000]
7C91EAEC | COND: 003A3652
003A3655 | Integer division by zero
7C91EAEC | COND: 003A3655
003A37DE | Access violation when writing to [00000000]
7C91EAEC | COND: 003A37DE
003A37E1 | Integer division by zero
7C91EAEC | COND: 003A37E1
003A3873 | Access violation when writing to [00000000]
7C91EAEC | COND: 003A3873
003A3876 | Integer division by zero
7C91EAEC | COND: 003A3876
76B00000 | Module C:\WINDOWS\system32\WINMM.dll
77D10000 | Module C:\WINDOWS\system32\USER32.dll
```

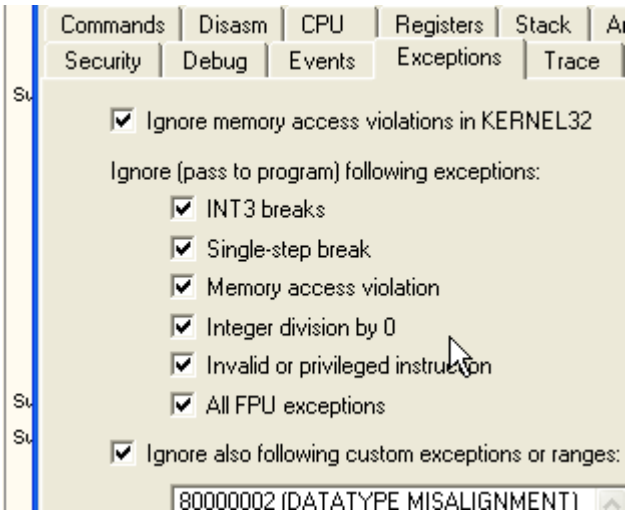
Alli se ven y OLLYDBG logueo bien las que se ven aquí, la ultima esta un poco mas abajo.

```
774B0000 | Module C:\WINDOWS\system32\ole32.dll
7C81084E | Breakpoint at kernel32.7C81084E
003A6744 | Access violation when writing to [00000000]
7C91EAEC | 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
```

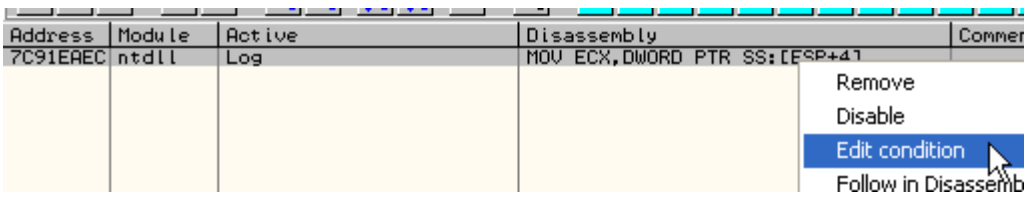
Vemos que la ultima excepcion es una ACCESS VIOLATION en 3A6744, pero alli no podemos poner BP ni HE estos ultimos son detectados por el packer y no corre el programa, veamos

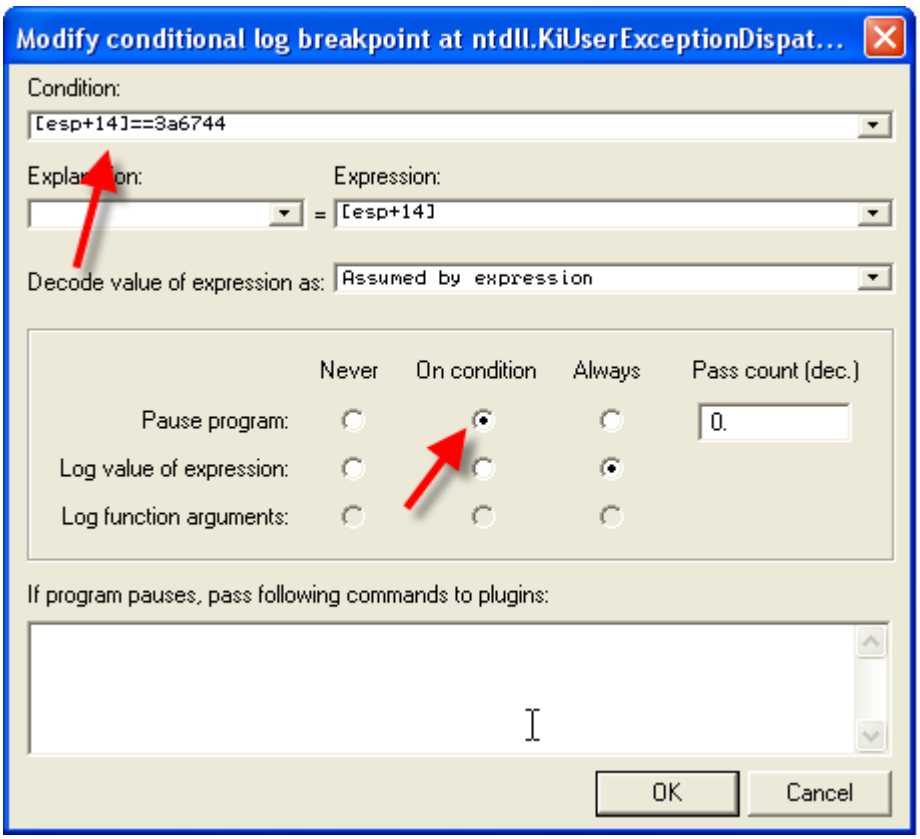



Una opcion que tengo es llegar al punto quitando la tilde de MEMORY ACCESS VIOLATIONS y parar en todas hasta que lleguemos a la ultima, este metodo si bien funciona, pensemos en algo mas generico y rapido y que sirva para todo tipo de excepciones, asi que vuelvo a colocar esta tilde.




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

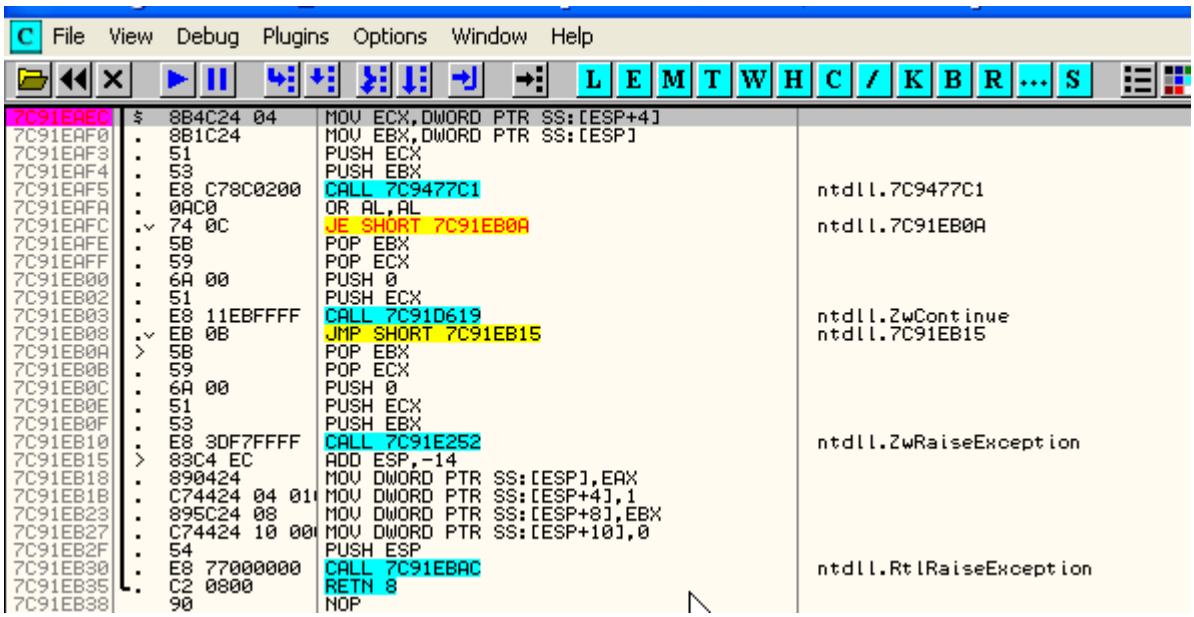





Solamente debo agregar la condicion para que pare, si logueabamos [esp+14] y en el momento que queremos que se detenga ese valor era 3a7644, ya que era la intruccion donde se genero la excepcion que nos mostraba nuestro LOG, pues entonces que pare cuando [esp+14]==3a6744 y listo.

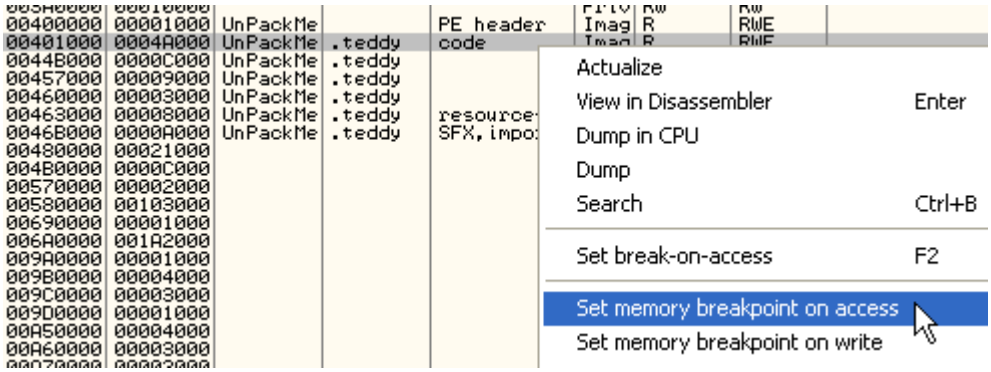
```
774B0000 Module C:\WINDOWS\system32\ole32.dll
7C81084E Breakpoint at kernel32.7C81084E
003A6744 Access violation when writing to [00000000]
7C91EAEC COND: 003A6744
7C810856 New thread with ID 00000110 created
770F0000 Module C:\WINDOWS\system32\OLEAUT32.dll
5B150000 Module C:\WINDOWS\system32\uxtheme.dll
746B0000 Module C:\WINDOWS\system32\MSCTF.dll
```

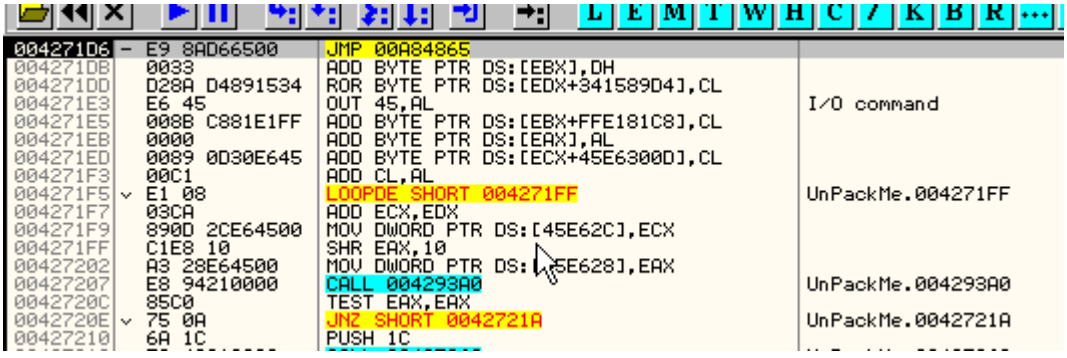
Cambio la tilde a que PAUSE PROGRAM-ON CONDITION para que pare cuando se cumpla la comdicion que colocamos y damos RUN.




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)




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 lienas, que cuando corrio con el packer, el maldito la ejecuto antes en su propia seccion y luego salto aquí, por lo cual en el dumepado, esas lineas no se ejecutaran.

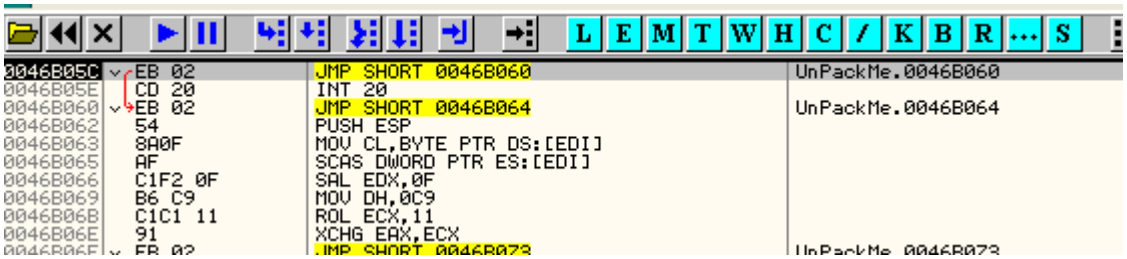
Que debo hacer?

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.

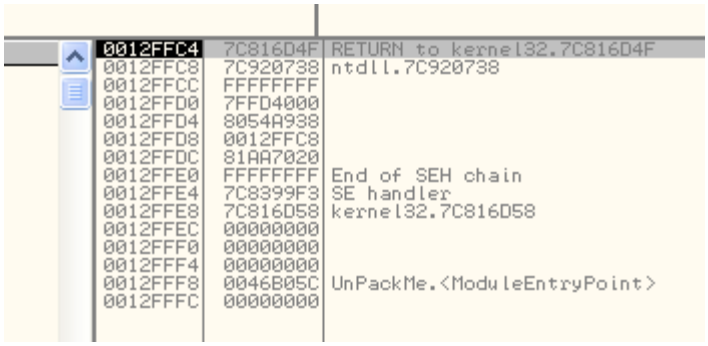
Por eso realice el metodo de las excepciones de esa forma, ya que muchas veces el traceo en OLLYDBG tiene unas cuantas posibilidades, tildes y opciones que intentar, y si cada vez que tengo que llegar a la ultima excepcion tengo que contarlas una a una, me volvere loco, por lo cual, tengo la forma de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

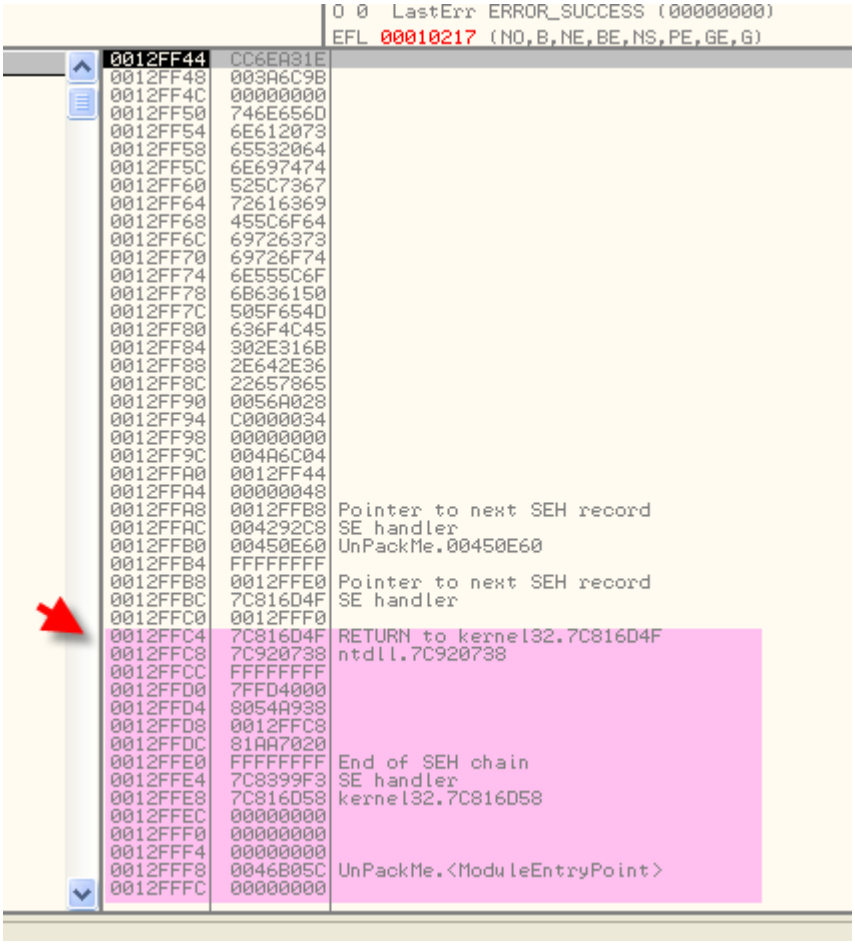



Si veo el stack al inicio.

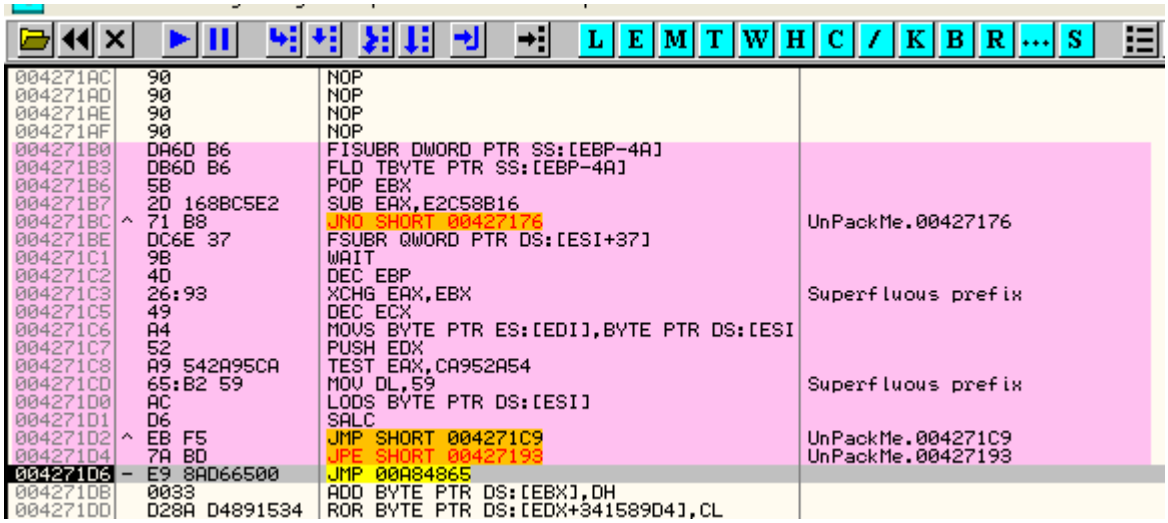



Esta en 12FFc4 en mi maquina, y sea cual sea el valor en la suya, en el OEP salvo casos muy estramboticos, el programa se debe iniciar con el stack en 12FFc4 o cercano, todo lo que haya en el falso OEP, arriba de esta direccion sera codigo que ya se ejecuto desde el OEP VERDADERO hasta el FALSO

Veamos lleguemos nuevamente al falso OEP.

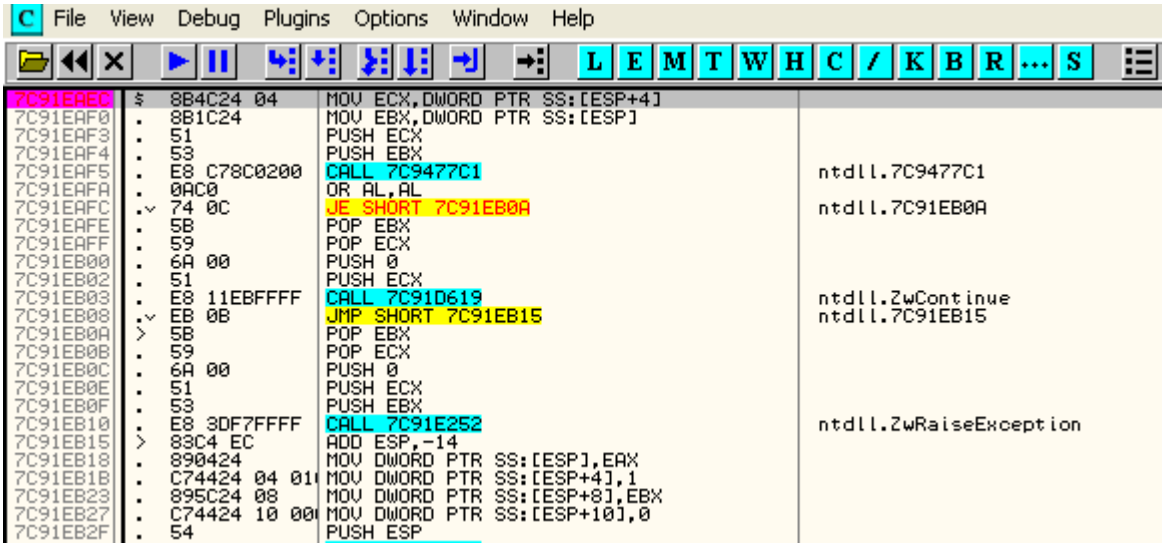



Vemos que si este fuera el verdadero OEP, el stack deberia estar en 12FFc4 o cerca, todo lo que hay arriba son instrucciones que ya se ejecutaron del programa, unas cuantas lineas que desde el verdadero OEP que el packer quito del programa y lo translado a otra seccion, se ejecutaron instrucciones que deberian 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.

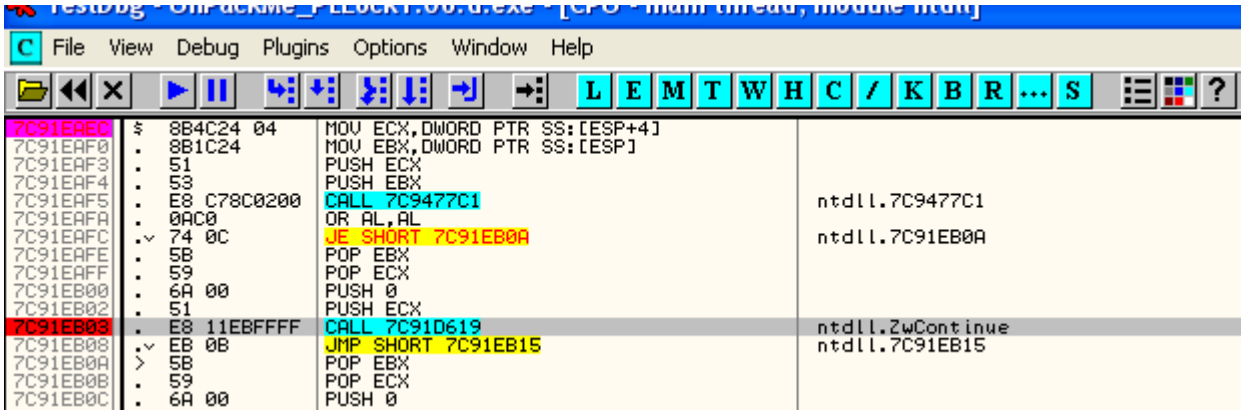


Vemos que el packer reemplazo los bytes originales del OEP e instrucciones siguientes con basura, que ademas nunca se ejecuto, pues con el BPM ON ACCESS paramos en la primera linea de codigo ejecutada en esta seccion y no paro en esos bytes sino mas abajo en 4271d6.

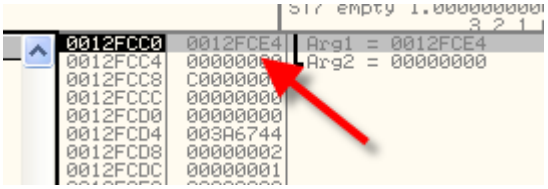
Pues cuales son los metodos para hallar los stolen bytes, pues hay muchos realmente, el mas clasico es tracear luego de regresar de la ultima excepcion, probemos ese reiniciemos y volvamos a la ultima excepcion.



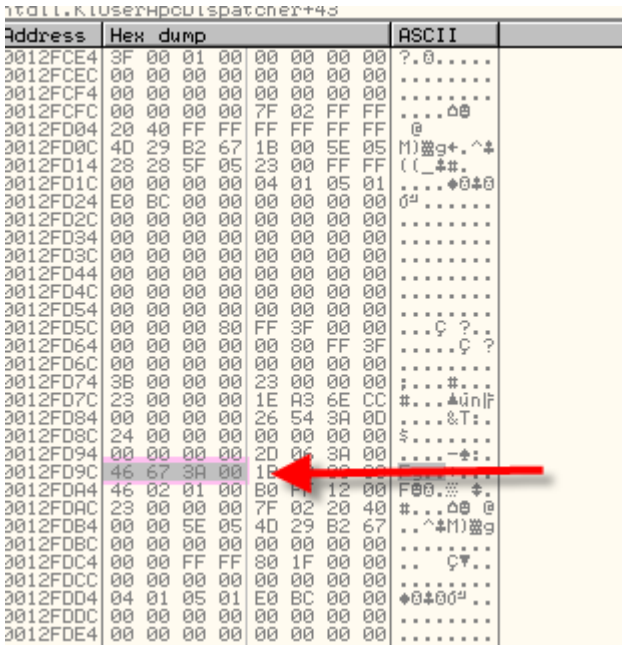

La misma se habia generado en 3A6744, por lo cual no me interesa en este caso el manejador de excepciones, asi que pondre un BP en el retorno, asi salteo el mismo.



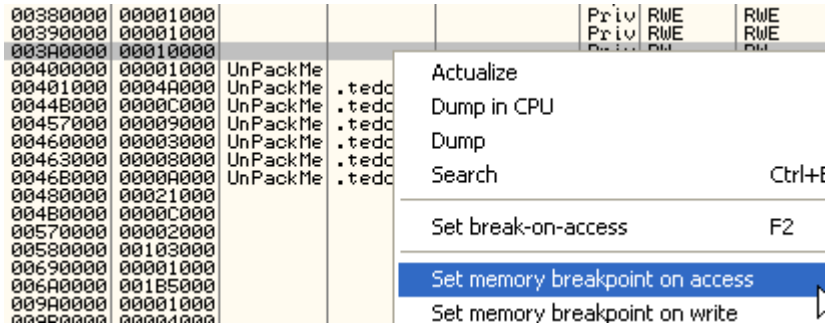

Ahora doy RUN, si alguien quiere saber donde retornara al programa para poner directamente un BP alli, pues veamos en el stack.



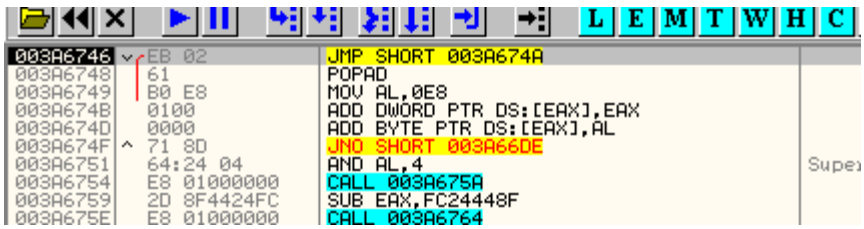

Tenemos el parametro que nos marca aquí el inicio de la estructura CONTEXT, miremosla en el DUMP.

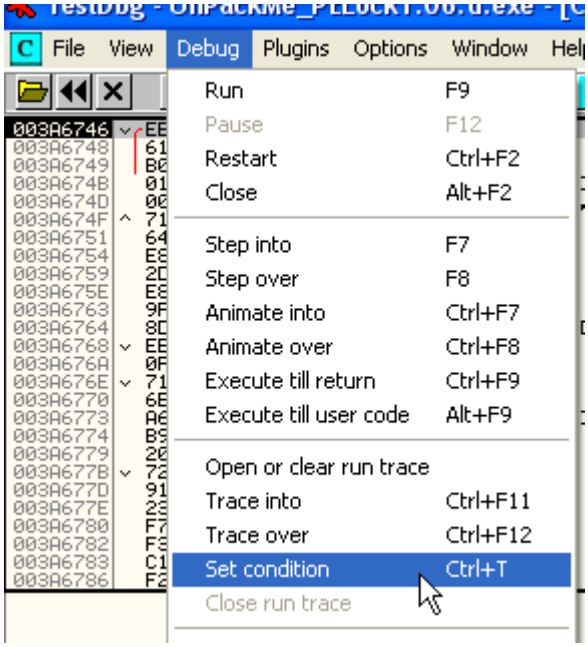



Los que ya conocen la estructura CONTEXT saben que ese es el EIP de la misma , entre los valores de los restantes registros que tendra el programa al regresar de la excepcion, como ya saben mas adelante le dedicaremos un estudio detallado de la estructura CONTEXT por ahora, solo mirando alli saben donde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

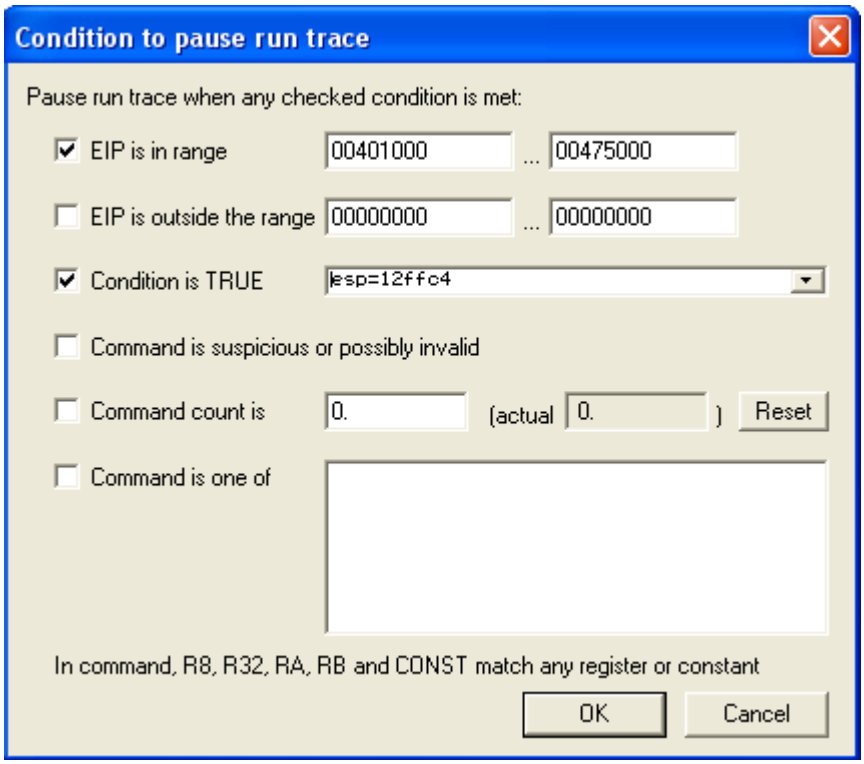



Y dando RUN vemos que para en el mismo lugar

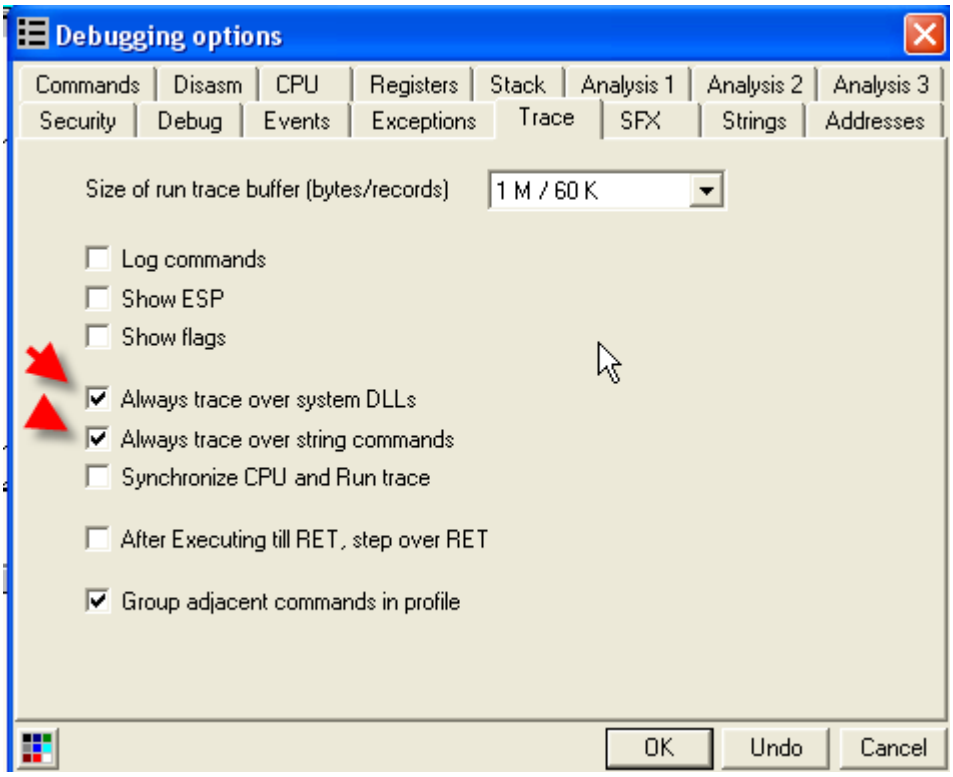




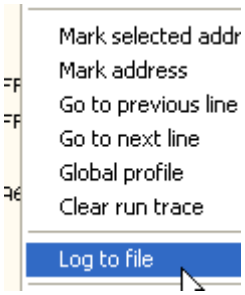

Alli colocaremos las condiciones del traceo, hay miles posibles, ya que puedo colocar que chequee los valores iniciales de los registros y cuando se cumplan que todos estan igual que antes de correr el programa, estare en el OEP, por ejemplo, pero miremos si funciona estas dos.



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.

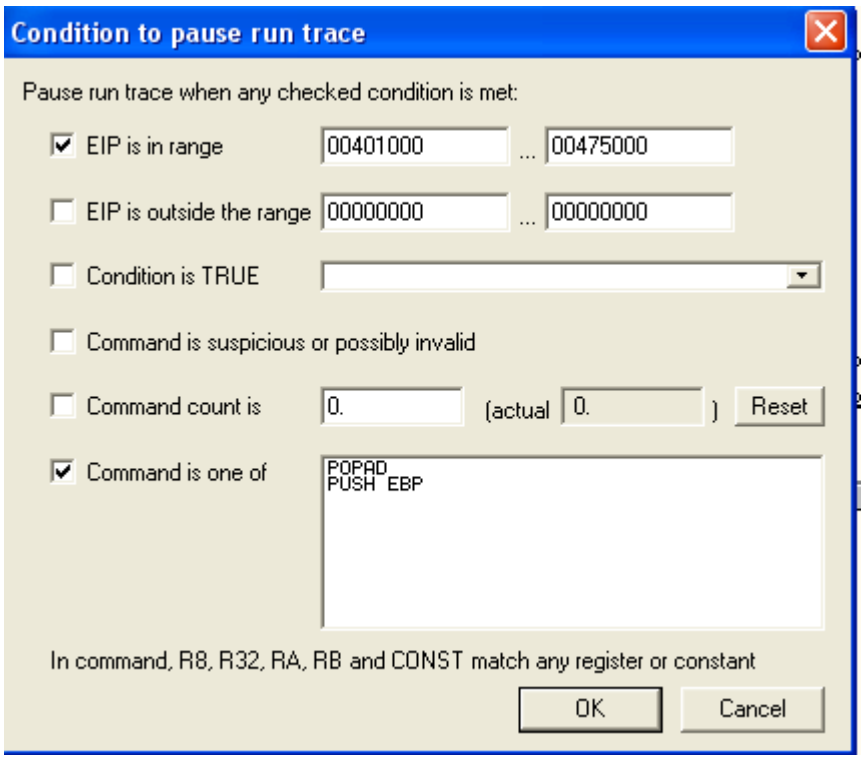



Ponemos las dos tildes en la configuracion para intentar si funciona con ellas, lo cual es lo mas probable, si no, probaremos quitandolas.

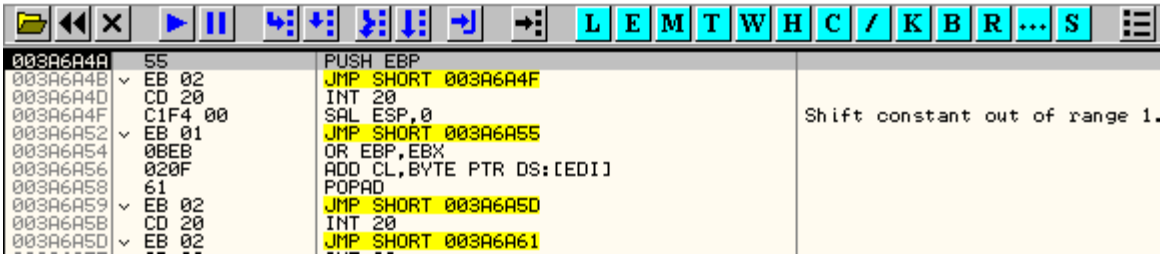



Tambien hacemos que loguee a una fila.

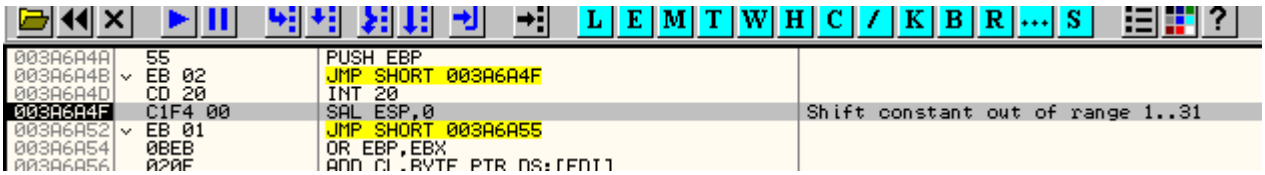
Apretamos TRACE INTO y me molesta la condicion de ESP que me hace parar muchas veces, si la quito y sigo con el trace into, para en el falso OEP, y en la fila de texto tengo todas las instrucciones que ejecuto antes, pero no me funciono lo de parar en el VERDADERO OEP, bueno mala suerte.



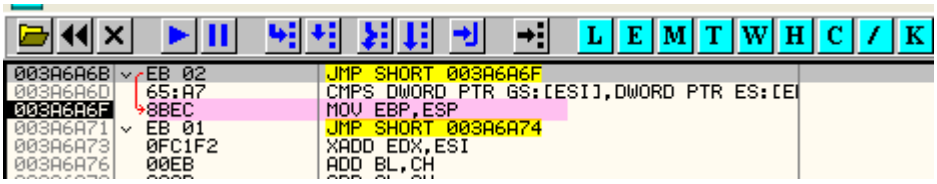

Una de las posibilidades que se puede intentar es poner que pare en cualquier PUSH EBP o POPAD que generalmente o son el OEP en el primer caso, o la recuperacion de registros en otros casos, esto puede funcionar o no, ya que muchas veces los push EBP son emulados, pero bueno veamos, pongamos a tracear.




Podria ser el verdadero OEP, traceemos un poco.



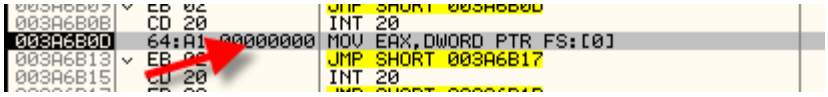

Por supuesto como instrucciones verdaderas no debemos considerar los saltos ni los SAL o cualquier otra instrucción basura.

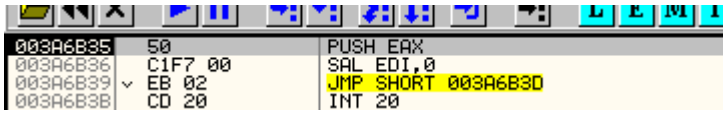


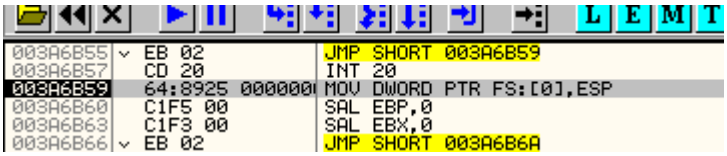
Esta parece ser la segunda instrucción faltante, al no ser emuladas y estar copiadas directamente del original es facil detectarlas.

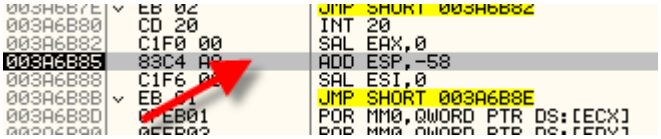

Y asi seguimos traceando y copiando las buenas instrucciones que va ejecutando hasta llegar al falso OEP.

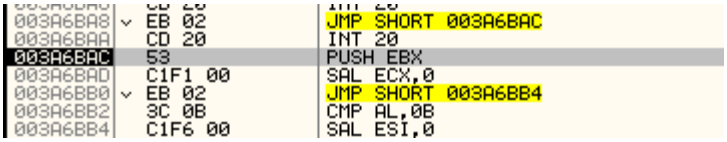


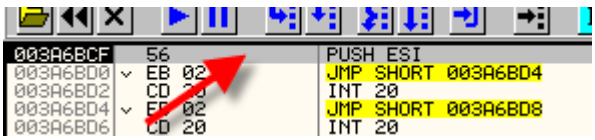


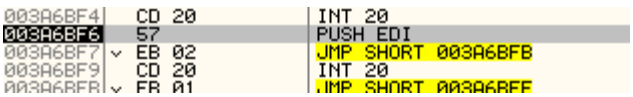



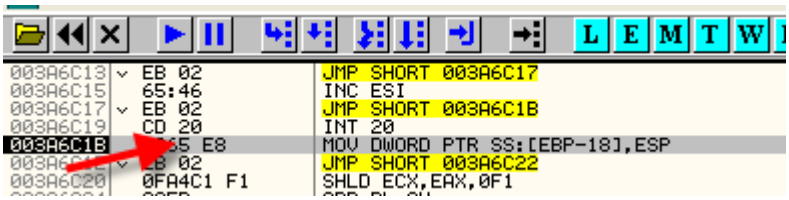



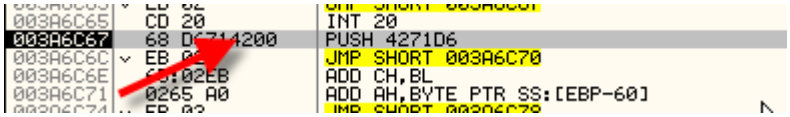



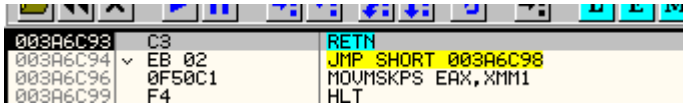



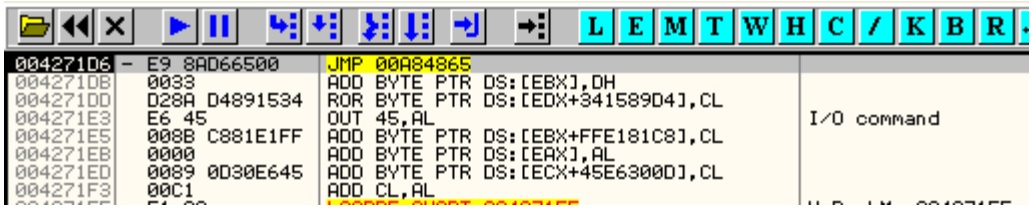





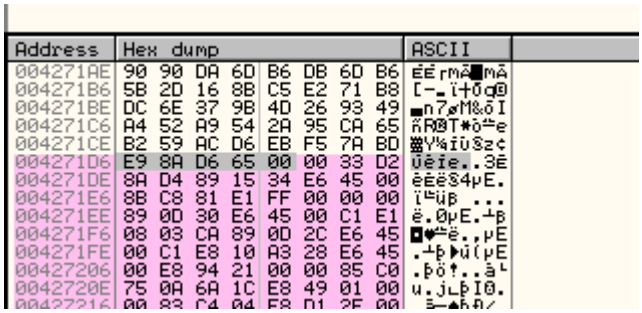




Y aquí llegamos al falso OEP ya que hizo PUSH 4271D6 que era la direccion del falso OEP y luego RET asi que salta alli.

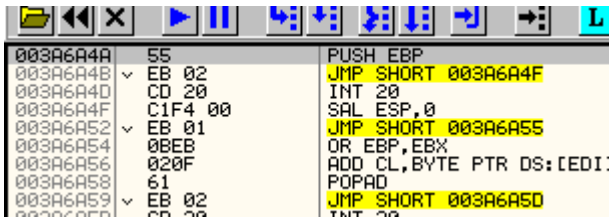


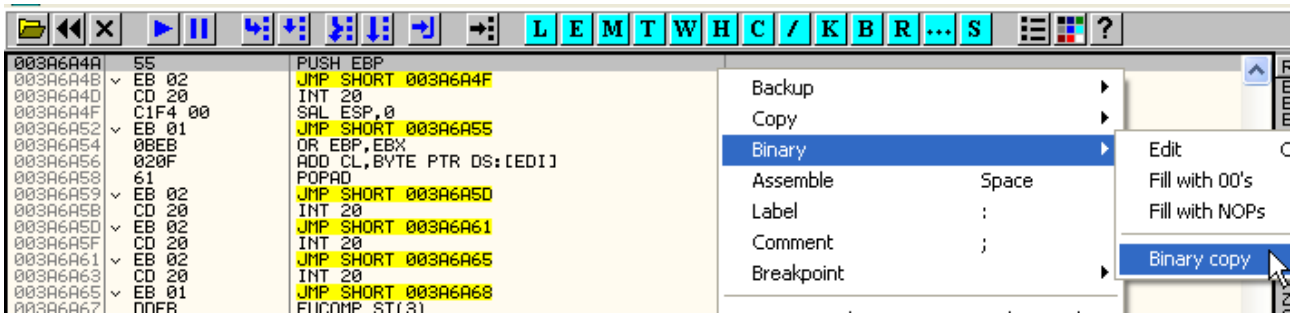

Por supuesto ahora hay que copiar todos los bytes de las instrucciones faltantes fijarse cuantos son y pegarlos antes del OEP FALSO veamos en el DUMP:

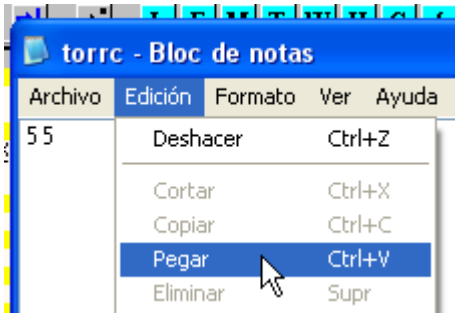



Alli vemos la zona del falso OEP, y arriba donde deben ir los bytes faltantes, como ya verificamos que es todo correcto volvemos para atrás apretando el menos hasta el verdadero OEP




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.



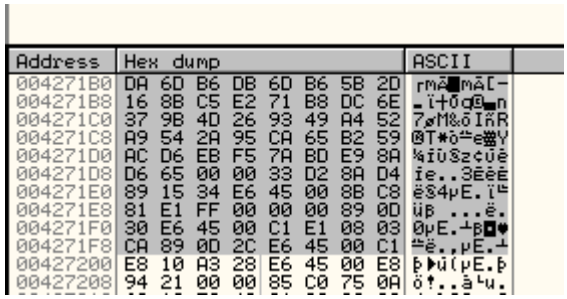

55 8B EC 6A FF 68 60 0E 45 00 68 C8 92 42 00 64 A1 00 00 00 00 50 64 89 25 00 00 00 00 83 C4 A8 53 56 57 89 65 E8

Son 38 stolen bytes que en hexadecimal es 26


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

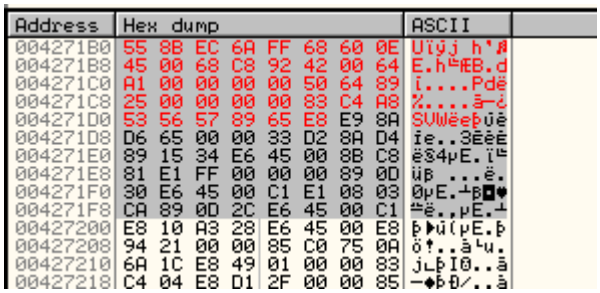

Asi hallamos la direccion de donde deberia haber estado el verdadero OEP si el packer no lo hubiera quitado.

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



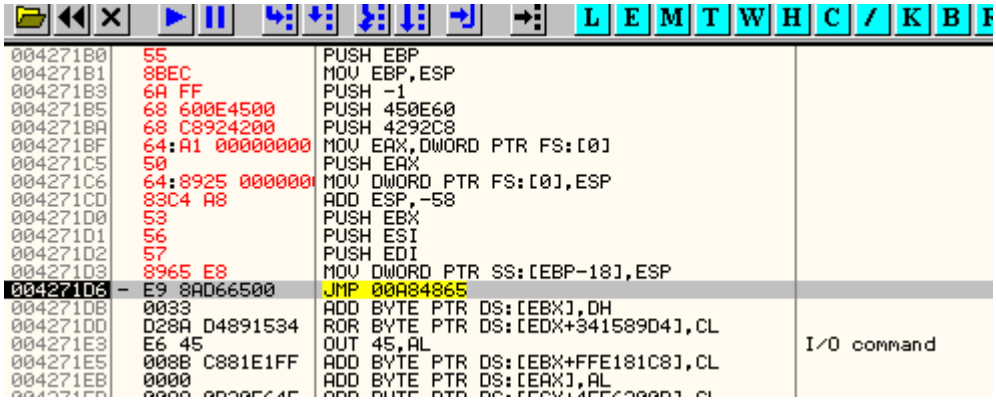
| Address | Hex dump | ASCII |
|---|---|---|
| 004271B0 | DA 6D B6 DB 6D B6 5B 2D | ┌mÂ▀mÂ[- |
| 004271B8 | 16 8B C5 E2 71 B8 DC 6E | ▬ï┼ôq◘▄n |
| 004271C0 | 37 9B 4D 26 93 49 A4 52 | 7¢M&ôIñR |
| 004271C8 | A9 54 2A 95 CA 65 B2 59 | ⌐T*ò╩e▒Y |
| 004271D0 | AC D6 EB F5 7A BD E9 8A | ¼ïÜ§z¢úè |
| 004271D8 | D6 65 00 00 33 D2 8A D4 | ie..3Ëèè |
| 004271E0 | 89 15 34 E6 45 00 8B C8 | ë§4µE.ï╚ |
| 004271E8 | 81 E1 FF 00 00 00 89 0D | üß ...ë. |
| 004271F0 | 30 E6 45 00 C1 E1 08 03 | 0µE.┴ß◘♥ |
| 004271F8 | CA 89 0D 2C E6 45 00 C1 | ╩ë..µE.┴ |
| 00427200 | E8 10 A3 28 E6 45 00 E8 | þ▶ú(µE.þ |
| 00427208 | 94 21 00 00 85 C0 75 0A | ö!..à└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 | Uïìÿjÿh`♫ |
| 004271B8 | 45 00 68 C8 92 42 00 64 | E.h╚Æ B.d |
| 004271C0 | A1 00 00 00 00 50 64 89 | í....Pdë |
| 004271C8 | 25 00 00 00 00 83 C4 A8 | %....â─¿ |
| 004271D0 | 53 56 57 89 65 E8 E9 8A | SVWëeèüè |
| 004271D8 | D6 65 00 00 33 D2 8A D4 | ie..3Ëèè |
| 004271E0 | 89 15 34 E6 45 00 8B C8 | ë§4µE.ï╚ |
| 004271E8 | 81 E1 FF 00 00 00 89 0D | üß ...ë. |
| 004271F0 | 30 E6 45 00 C1 E1 08 03 | 0µE.┴ß◘♥ |
| 004271F8 | CA 89 0D 2C E6 45 00 C1 | ╩ë..µE.┴ |
| 00427200 | E8 10 A3 28 E6 45 00 E8 | þ▶ú(µE.þ |
| 00427208 | 94 21 00 00 85 C0 75 0A | ö!..à└u. |
| 00427210 | 6A 1C E8 49 01 00 00 83 | j∟þI0..â |
| 00427218 | C4 04 E8 D1 2F 00 00 85 | ─♦þÐ/..à |

Alli se copiaron miremos como quedo el codigo



| | | | |
|---|---|---|---|
| 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 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