INTRODUCCION AL CRACKING CON OLLYDBG PARTE 32

En la parte anterior tratamos de dejar claro el concepto de OEP, o sea la primera linea ejecutada del programa original que el 99 por ciento de las veces esta en la primera seccion (aunque aquí en esta parte he puesto uno que no esta en la primera sección de molesto que soy jeje)

Vimos que cuando llegamos alli, y mirabamos la memoria, el contenido es similar al del original, lo cual nos da la posibilidad de intentar dumpear y generar el archivo mas parecido posible al original para reconstruirlo.

El metodo clasico seria:

1)ENCONTRAR EL OEP
2)DUMPEAR
3)REPARAR LA IAT
4)VERIFICAR SI EL ARCHIVO TIENE ANTIDUMPS O CHEQUEOS QUE LE IMPIDAN CORRER Y REPARARLOS

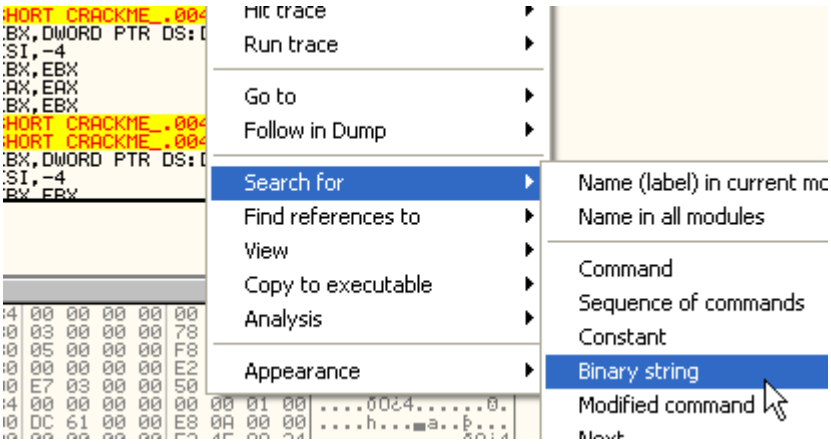
Este es el metodo clasico que con pequeñas variaciones según el packer, normalmente se utiliza como metodo de trabajo, nos enfocaremos en esta parte en los metodos que podemos utilizar para llegar al OEP, que pueden funcionar en diferentes packers.
Hay que decir que muchas veces es necesario probar e intentar, muchas veces estos metodos funcionan, otras veces hay packers que evitan que estos metodos funcionen, por lo cual hay que usar un poco de inventiva, pero teniendo la idea que cual es el punto a hallar (el OEP) veremos como podemos arreglarnos utilizando las herramientas que poseemos.

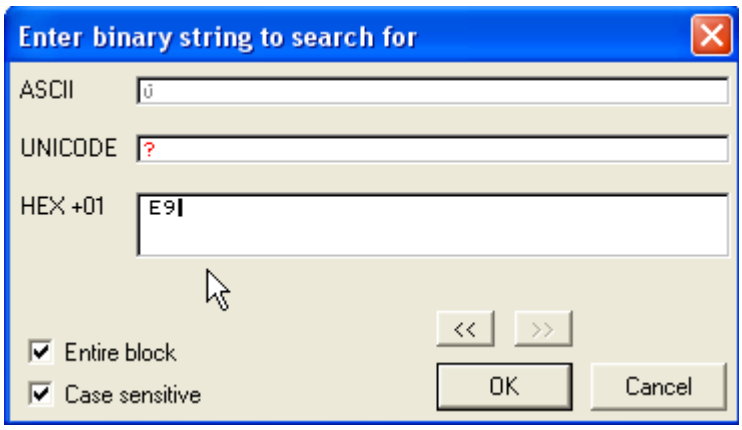
Por ahora para explicar utilizare el Crackme de Cruehead empacado, mas adelante veremos otros packers, pero en esta explicacion general, el mismo nos servira.

1) MIRAR O BUSCAR OPCODES EN EL LISTADO MUERTO DEL PACKER ANTES DE EJECUTAR

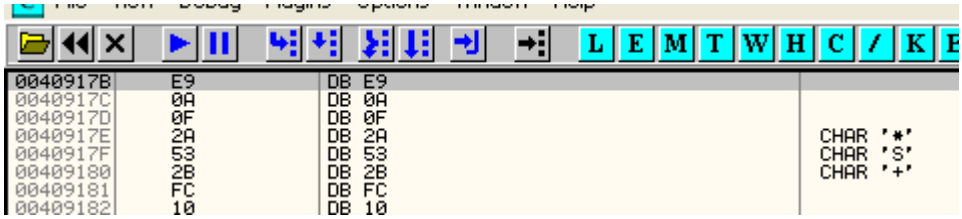
Esto solo puede funcionar en packers inocentes ni se me ocurriria intentarlo en un asprotect por ejemplo jeje, pero a veces sirve buscar los opcodes del JMP LARGO (0E9) o CALL LARGO (0E8), pensando que el packer necesitara un salto o call largo a la primera sección para llegar al OEP, y rezando que el mismo este presente en el inicio, o sea que no se automodifique el packer jeje.

Pues en este caso seria



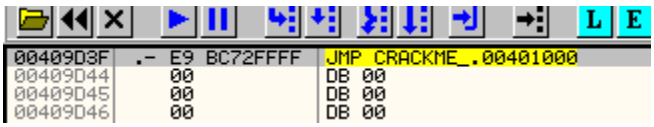


Cuando para miro a ver si es un salto a la primera sección y si no es, apreto CTRL+L para que busque el siguiente E9.



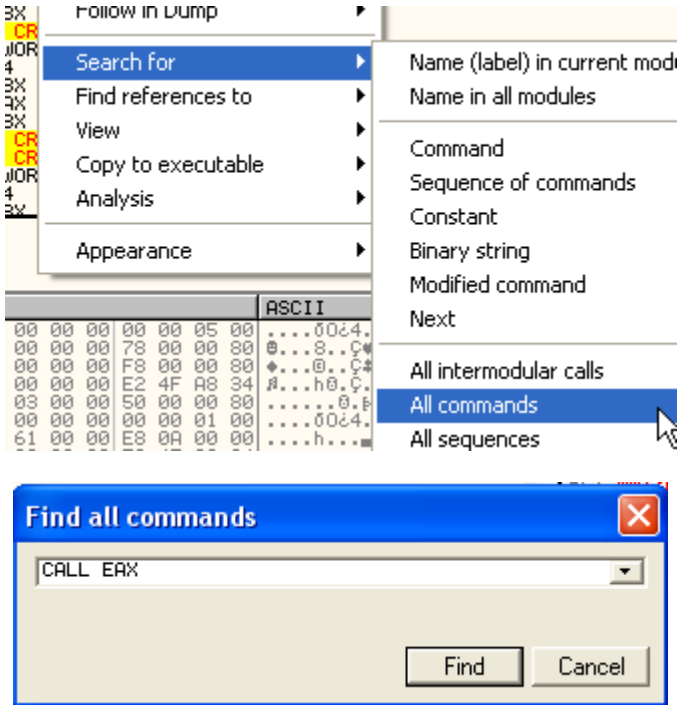
Hago CTRL+L

Empiezo a hallar salto largos que no van a la primera seccion, y asi hasta que llego a



El cual es un salto a la primera seccion al que le puedo poner un BPX y cuando para apretar f7 y estare en el OEP.

Tambien si uno tiene ganas de buscar en el codigo del packer deberia intentar CALL EAX, CALL; EBX, JMP EAX, porque muchos packers utlizan los registros para disimular el salto al OEP, en el caso de estos comandos lo bueno es que como son comandos completos los podemos buscar todos con SEARCH FOR - ALL COMMANDS y ponerle BPX a todos juntos, veamos un ejemplo.

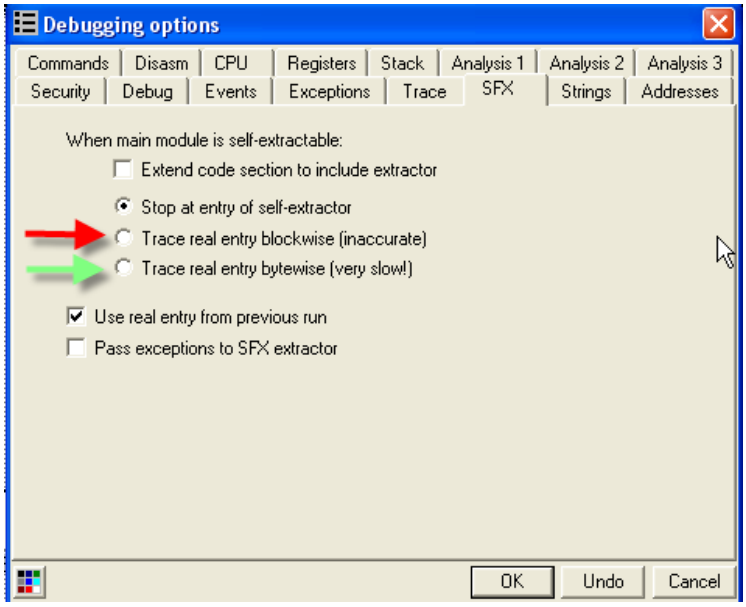


Y en este caso no sale ningun resultado, pero si hubiera varios resultados que me salen en la lista, puedo haces click derecho en los resultados y elegir la opcion de ponerle BPX a todos ellos, y con eso tendria la posibilidad de que pare en los mismos, y alli cuando para fijarme en este caso el valor que toma EAX y si vemos que es un CALL o JMP a la primera seccion, pues apreto f7 y llego.

Este metodo de buscar en el listado muerto no es muy utilizado, porque la mayoria de las rutinas desempacadoras modernas se automodifica, cuidando especialmente de que la parte del salto al OEP no este visible en el inicio, para evitar este tipo de busquedas, pero bueno, mencionamos la posibilidad que en un packer antiguo o malo, puede funcionar.

2)**<u>USAR EL BUSCADOR DE OEPs que trae incluido el OLLYDBG</u>**

Abramos el crackme UPX y vayamos a DEBUGGING OPTIONS-SFX



Alli vemos las dos opciones que en la pestaña SFX, tiene el OLLYDBG para hallar OEPs, la que esta señalada con rojo es la mas rapida, y la que esta señalada con verde es mas lenta aunque puede funcionar un poco

mejor a veces, probemos, pongamos la tilde en la de la flecha roja.

Al reiniciar veo que no funciona en este caso, porque es eso, veamos las instrucciones del caso.

Self-extracting (SFX) files

Self-extracting file consists of extracting routine and packed original program. When troubleshooting SFX, you usually want to skip extractor and stop on the entry point of original program ("real entry"). OllyDbg contains several functions that facilitate this task.

Usually extractor loads to address that is outside the executable section of the original program. In this case OllyDbg recognizes file as SFX.

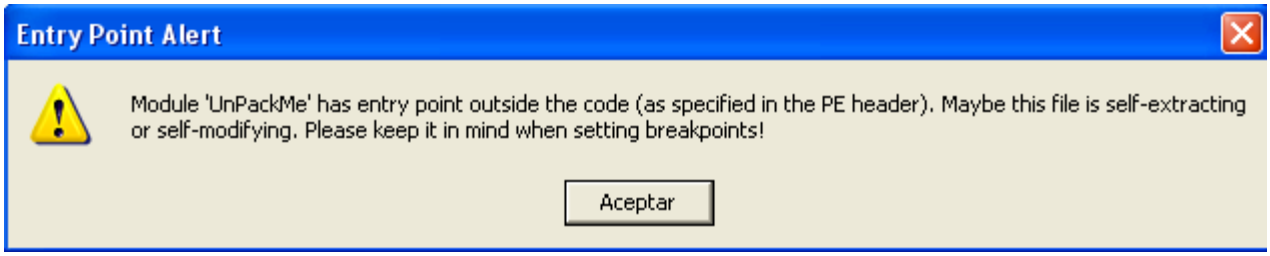
When SFX options request tracing of real entry, OllyDbg sets memory breakpoint on the whole code section. Initialy this is empty or contains compressed data. When program attempts to execute some command within protected area which is neither RET nor JMP, OllyDbg reports real entry. This is how bytewise extraction works.

This method is very slow. There is another, much faster method. Each time exception on data read occurs, OllyDbg enables reading from this 4-K memory block and disables previous read window. On each data write exception it enables writing to this block and disables previous write window. When program executes command within remaining protected area, OllyDbg reports real entry. However, when real entry is inside read or write window, its location will be reported incorrectly.

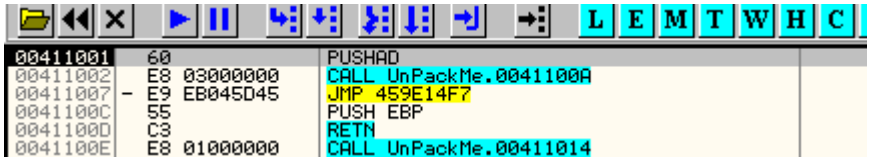
Veo en la ayuda del OLLYDBG que esto solo funciona cuando OLLYDBG detecta que el Entry point esta fuera de la seccion code como en la mayoria de los programas empacados, pero en este caso no nos advrtio OLLYDBG de ello, el problema es que UPX cambia la seccion CODE a esta en la cual se ejecuta el desempacador, por lo cual el EP esta en la misma seccion CODE y OLLYDBG no detecta como que es un empacado y en este caso el metodo no funciona, aunque es raro que un packer realice ese cambio, pues aquí no va.

Pues para poder demostrar como va este metodo, usare otro crackme empacado que adjunto, es el crackme empacado con aspack, otro sencillo packer.

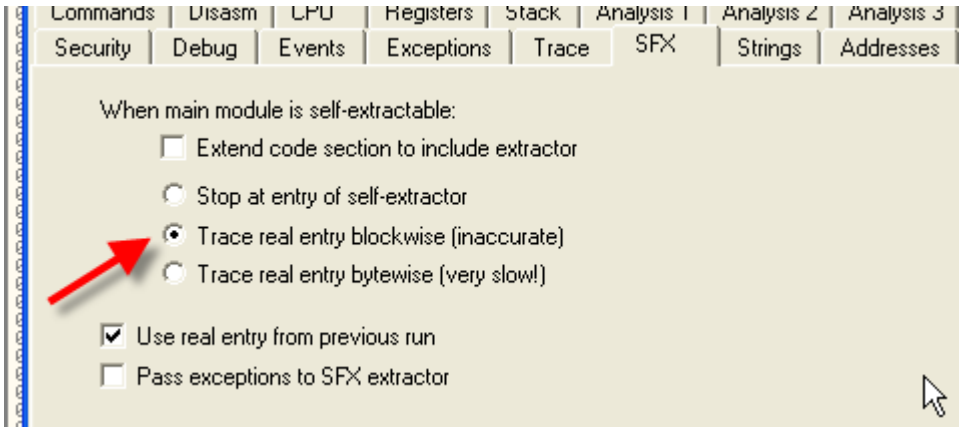
Primero coloco la tilde en su posicion nornal y veo que OLLYDBG me lo reconoce como packer según su metodo de ver si el EP esta dentro de la seccion CODE.

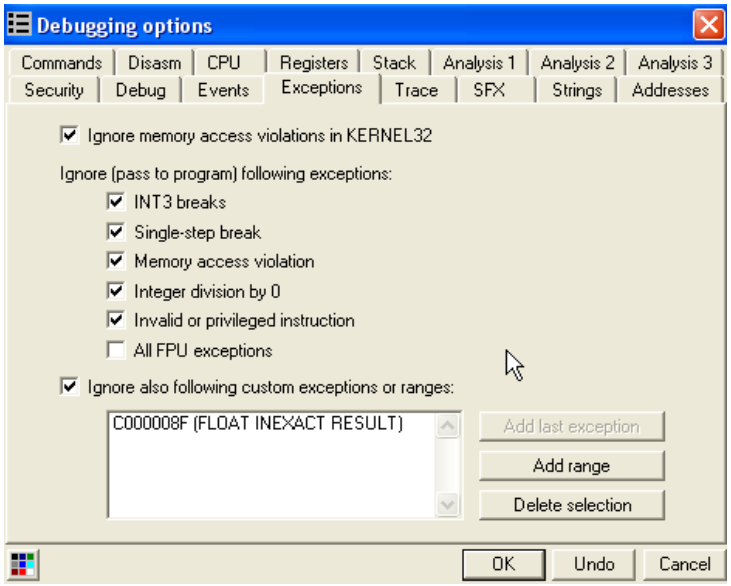


Y me muestra el cartelito de aviso y al aceptarlo llega al EP.



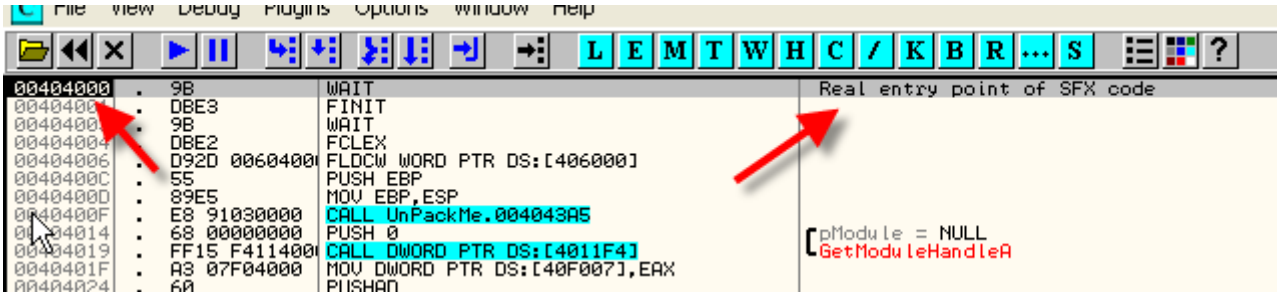
Ahora cambio la tilde por la de la flecha roja, me fijo que las tildes en EXCEPTIONS esten marcadas para que no pare por excepciones y renicio el programa en OLLYDBG.





Doy RUN.

Vemos que alli para en 404000 que me marca REAL ENTRY POINT OF SFX CODE (aunque no esta en la primera seccion ya veremos que este crackme es un caso especial que se desempaca en la tercera seccion algo inusual pero posible)



.

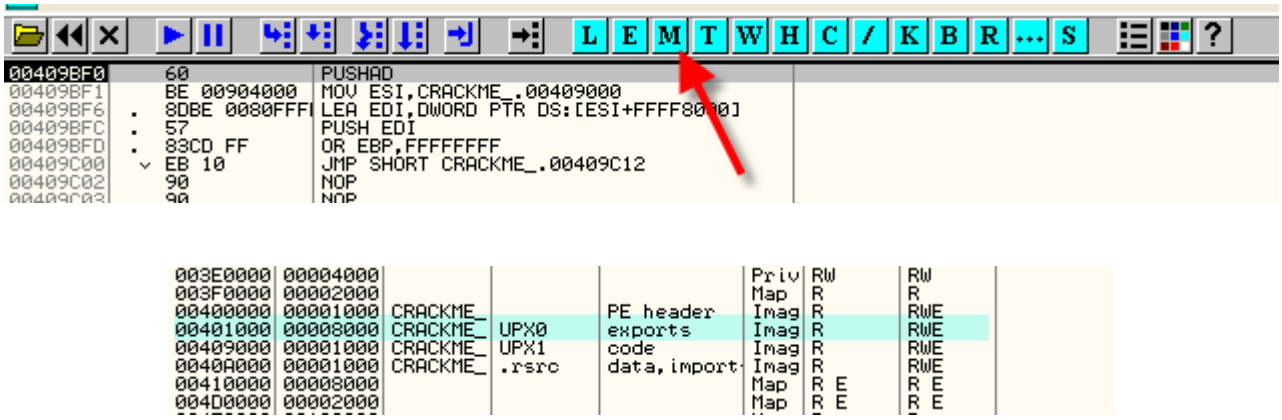
En este caso el metodo funciono, ese es el OEP, veamos si tarda mucho mas si hubieramos puesto la flecha verde en vez de la roja.

Reinicio y no tarda muchisimo mas ya que el codigo del desempacador es breve, en ambos casos funciono y me dio el OEP correcto.

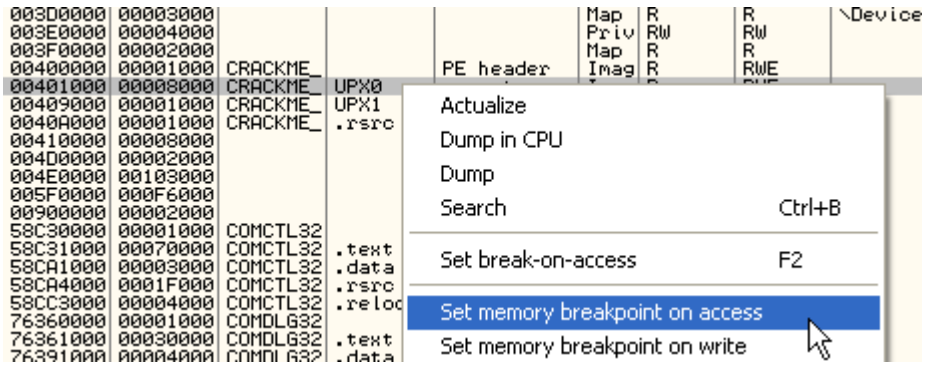
Siempre debemos recordar cuando termino de usar este metodo de ir y quitar la tilde y volverla a su posicion original, pues si no OLLYDBG no parara en los ENTRY POINTS comunes normalmente y siempre buscara parar en OEPs.

3)USANDO EL OLLYDBG PARCHEADO PARA BUSCAR OEPs

Este es el mismo OLLYDBG que usamos para las partes sobre Visual Basic, que cuando pones un BPM ON ACCESS, para solo por ejecucion y no cuando lee y escribe (ON READ o ON WRITE) y es ideal para hallar OEPS, veamos el caso del UPX, vayamos a M y miremos las secciones.



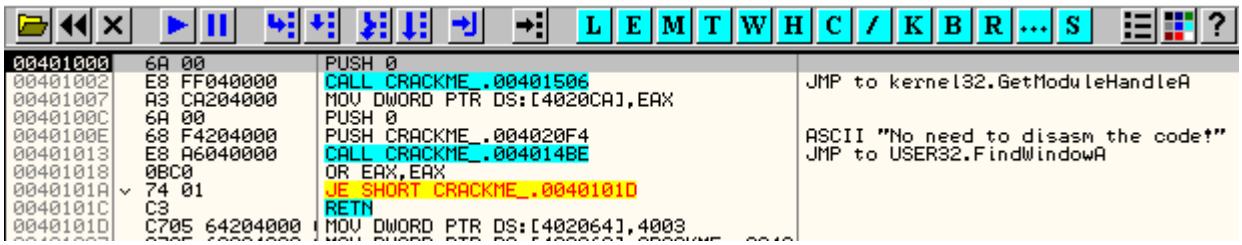
Alli esta la primera seccion, en ella el desempacador escribira mientras desencripta los bytes originales, y no queremos que pare mientras trabaja, ya que si no, parara miles de veces antes de llegar al OEP cuando lee y escribe, gracias a este OLLYDBG modificado, no para cuando lee y escribe en dicha seccion, si no solo cuando ejecuta, y eso es lo que queremos hallar nosotros, que pare en la primera linea que se ejecuta en la primera seccion la cual sera casi siempre el OEP.



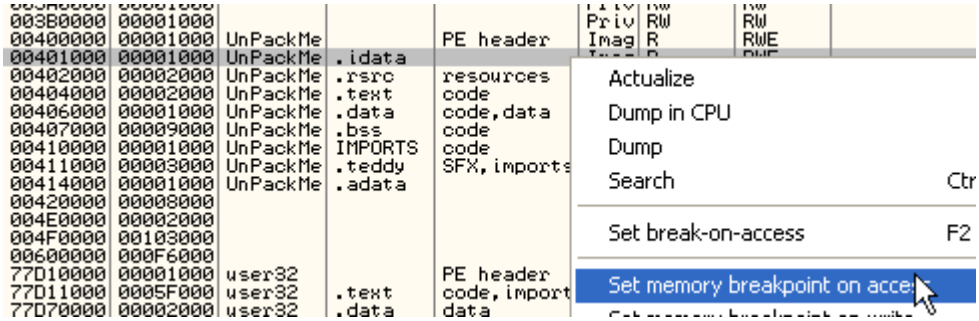
Me fijo en la pestaña EXCEPTIONS que esten todas las tildes marcadas para que no pare por EXCEPCIONES, y doy RUN y me voy a tomar unos mates, (café o te para los que no son argentinos jeje, aunque yo no tomo mate ju)

Por supuesto este metodo es un poco mas lento por eso le digo que se tomen unos mates, depende el packer puede tardar unos segundos hasta varios minutos.

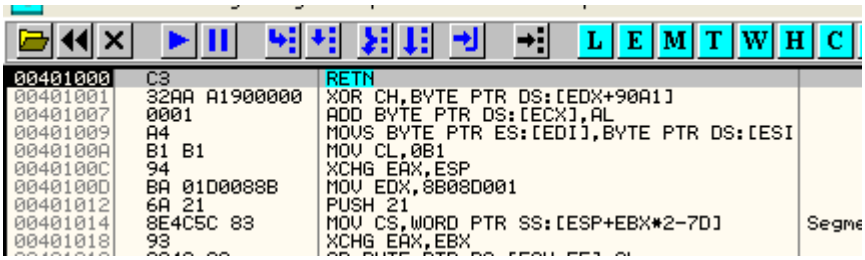
Cuando vuelvo de los mates esta detenido en el OEP



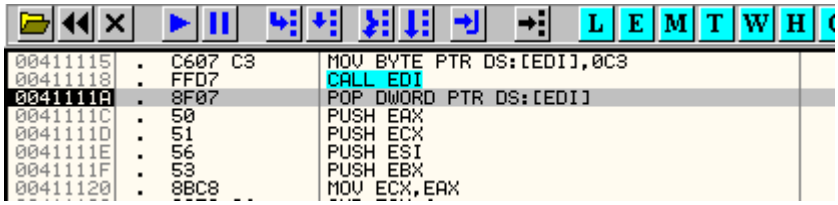
Si lo hago en el aspack



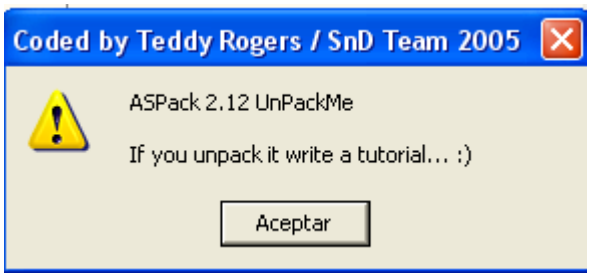
Doy RUN



Veo que la primera linea ejecutada es esta, veamos que pasa si apreto f7

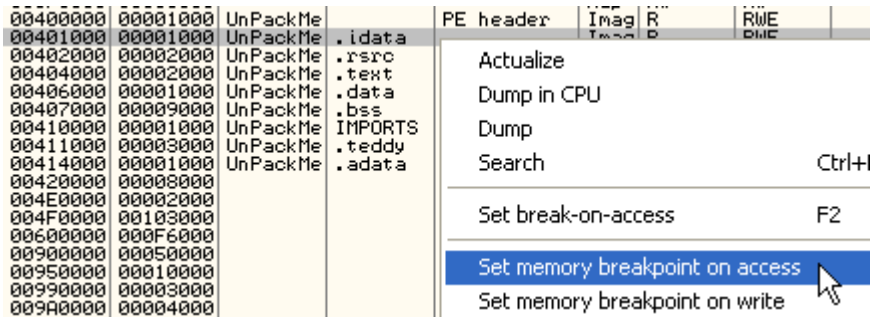


Vuelve a la rutina del packer, por lo cual le doy RUN de nuevo y vemos que el programa se ejecuta sin parar, porque ocurre esto?
Si nos fijamos, cuando halllamos el OEP con OLLYDBG en este caso el packer varia y no se desempaca en la primera seccion por lo cual hay que poner un BPM ON ACCESS en otra seccion y no en la primera, para determinar en cual, podemos correr el crackme sin poner BPM ni nada.

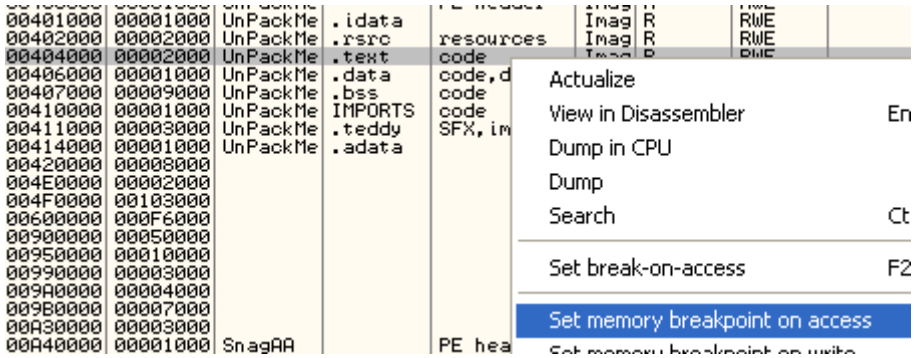


.

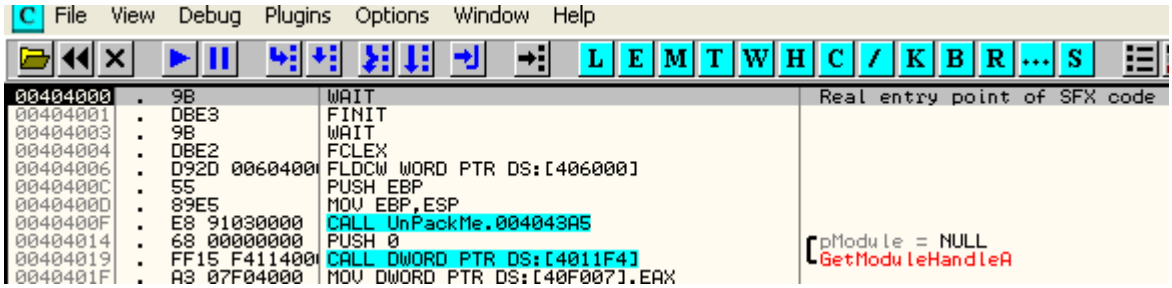
Una vez que aparece la ventana del mismo ya sabemos que esta desempacado en memoria, para saber en que seccion esta corriendo, probamos poner BPM ON ACCESS, en cada seccion, si el programa sigue coriendo y no para en OLLYDBG quiere decir que no esta corriendo en esa seccion y probamos la siguiente.



Alli pongo un BPM ON ACCESS en la primera y no pasa nada sigue como si nada, hago lo mismo en la segunda y nada, al poner en la tercera que empieza en 404000.



Veo que para en OLLYDBG, al tratar de ver la ventana del crackme, eso quiere decir que esa es la seccion que se esta ejecutando, asi que repitamos el proceso desde el inicio para buscar el OEP, pero en este caso, poniendo un BPM ON ACCESS en la tercera seccion.

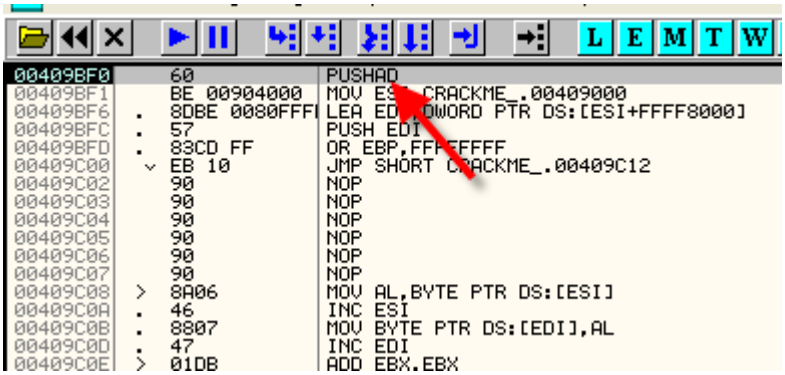


Ahi si se toman unos mates cafes lo que quieran y al volver esta parado en el OEP, jeje, otro metodo que suele funcionar bien en muchisimos packers.

4) EL METODO DEL PUSHAD

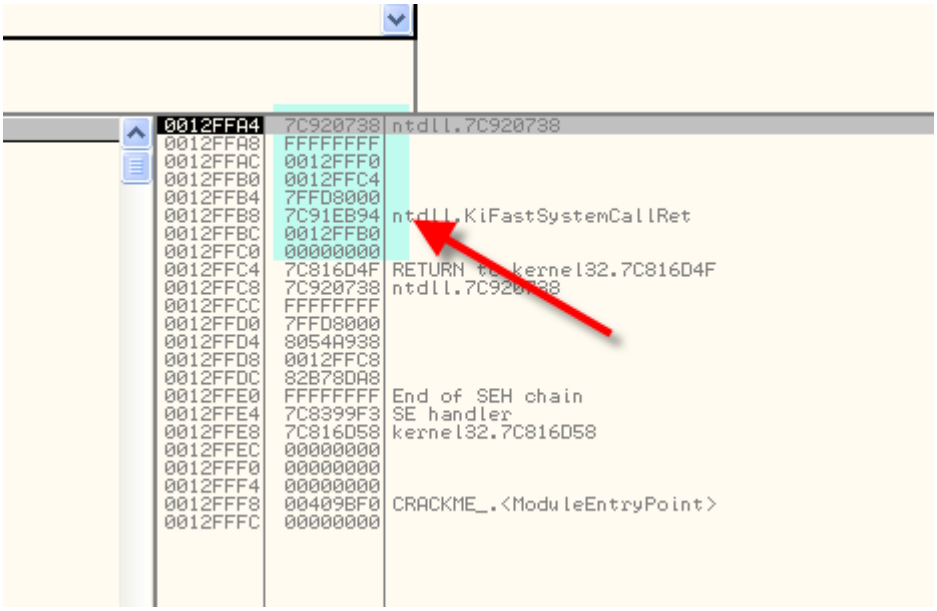
Este metodo funciona en una buena cantidad de packers y se basa en lo siguiente, muchos packers en sus primeras lineas ejecutan un PUSHAD para guardar los valores iniciales de los registros, luego desempacan, y antes de saltar al OEP, recuperan los valores iniciales de los registros con un POPAD.

Veamos el CRACKME UPX



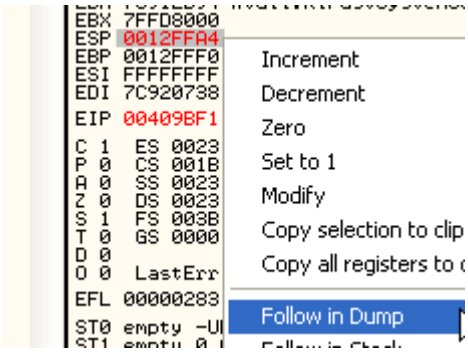
Vemos el PUSHAD alli en el inicio, a veces puede estar un poco mas adelante, otra veces hay packers que hacen PUSH de cada registro uno a uno, (PUSH EAX, PUSH EBX, etc) pero para el caso es lo mismo guardan en el stack los valores iniciales de los registros, los cuales recuperan antes de saltar al OEP.

Si nosotros pasamos el PUSHAD con f7

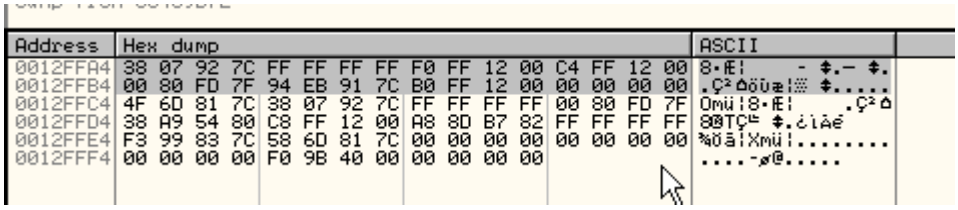


Vemos que alli estan guardados los valores iniciales de los registros, y si los lee antes de saltar al OEP podemos ponerle un HARDWARE BPX ON ACCESS en alguno de esos valores para que pare justo cuando lo lea, y estaremos justo antes de saltar al OEP.

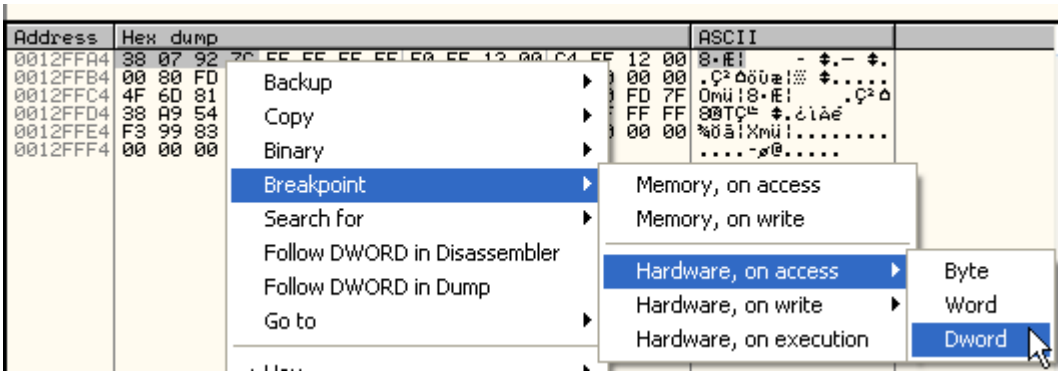
Busquemos estos valores en el DUMP, marcando el registro ESP y haciendo FOLLOW IN DUMP



Eso nos mostrara en el DUMP el contenido del stack



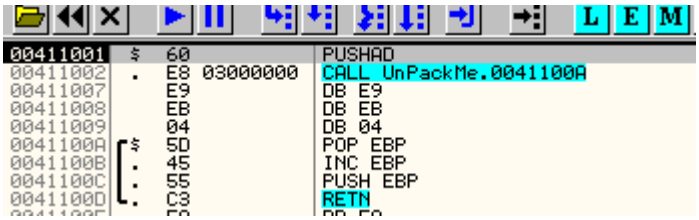
Alli vemos los valores que guardó, normalmente lo que se hace es marcar el primer byte o los primeros 4 bytes y colocarle un HARDWARE BPX ON ACCESS.



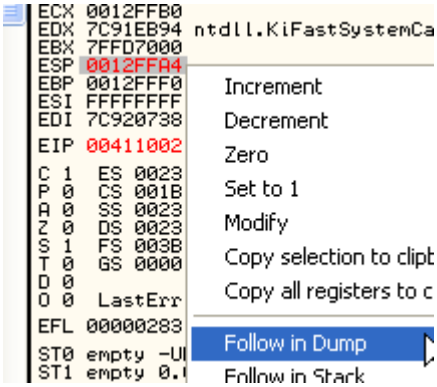
Es lo mismo que sea BYTE o WORD o DWORD, la cuestion es que pare cuando lea ese valor, demos RUN ahora.

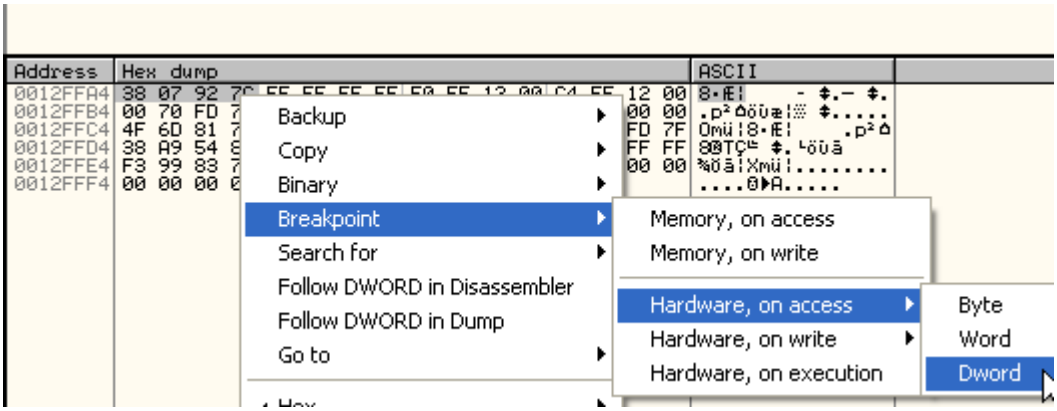
Vemos que al ejecutar el POPAD para restaurar esos valores guardados en el stack, los lee y para, y justo abajo tenemos el salto al OEP, asi que estamos de parabienes aquí este metodo funciono de maravilla.

Probemos en el aspack

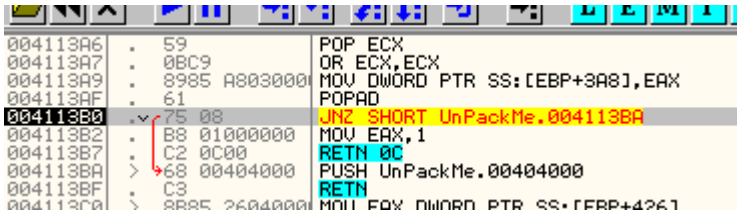


Alli veo un PUSHAD, lo paso con f7 y luego ESP-FOLLOW IN DUMP

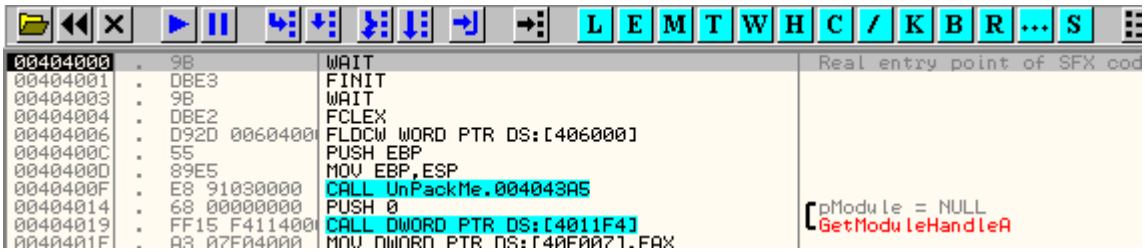




Y doy RUN.



Vemos que para alli justo antes del salto al OEP que en este caso es un PUSH 404000 y luego un RET si traceamos con f7.



Llegamos al OEP perfectamente

Debemos decir que muchos packers nuevos vienen protegidos contra este metodo, pero bueno, hay que conocer de todo, y intentarlo para saber si va o no.

Sigamos con mas metodos posibles.

5) PARA PROGRAMAS EN VISUAL BASIC (NATIVE O PCODE)

Bueno es muy sencillo hallar el OEP de programas de VB empacados, ya que como vimos siempre hay un PUSH y un CALL a la api de VB, al inicio, asi que utilizo el OLLYDBG para VB y cuando arranco el programa y estoy parado en el entry point, voy a M y busco la dll de visual, y en la seccion code de dicha dll, le coloco un BPM ON ACCESS.
De esta forma el programa se descomprimira y parara justo en la primera linea que ejecute la dll de visual basic, y en la primera linea del stack tendre la direccion de retorno del call inicial, con lo cual yendo a esa direccion, justo arriba estaran el push y el call incial que se encuentran en el OEP, con lo cual lo habre hallado.
Es cierto que el metodo del OLLYDBG modificado para VB tambien funcionara si pongo un BPM ON ACCESS en la primera seccion del programa directamente, pero hay packers que protegen esto muchisimo, por lo cual se pueden intentar ambos metodos y por eso lo dejo asentado aquí ya que ambos funcionan.

No veremos ejemplos en este caso pues es muy sencillo de entender, mas adelante si nos toca un crackme de VB empacado veremos el ejemplo en la practica.

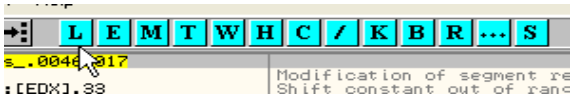
6)**METODO DE LAS EXCEPCIONES**

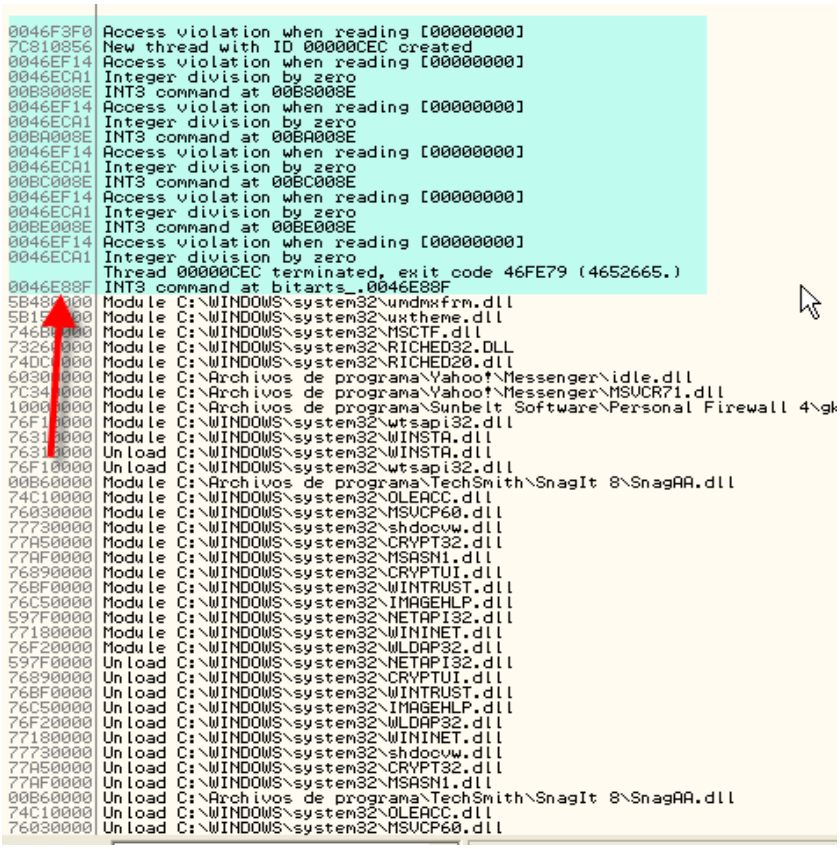
Si tenemos un packer que genera muchas excepciones al desempacarse, el metodo es el siguiente, usaremos el crackme bitarts que adjunto.

Lo abrimos en el OLLYDBG para VB, protegido con los plugins necesarios para no ser detectado.

Coloco todas las tildes en EXCEPTIONS y lo corro en OLLYDBG hasta que veo que arranca

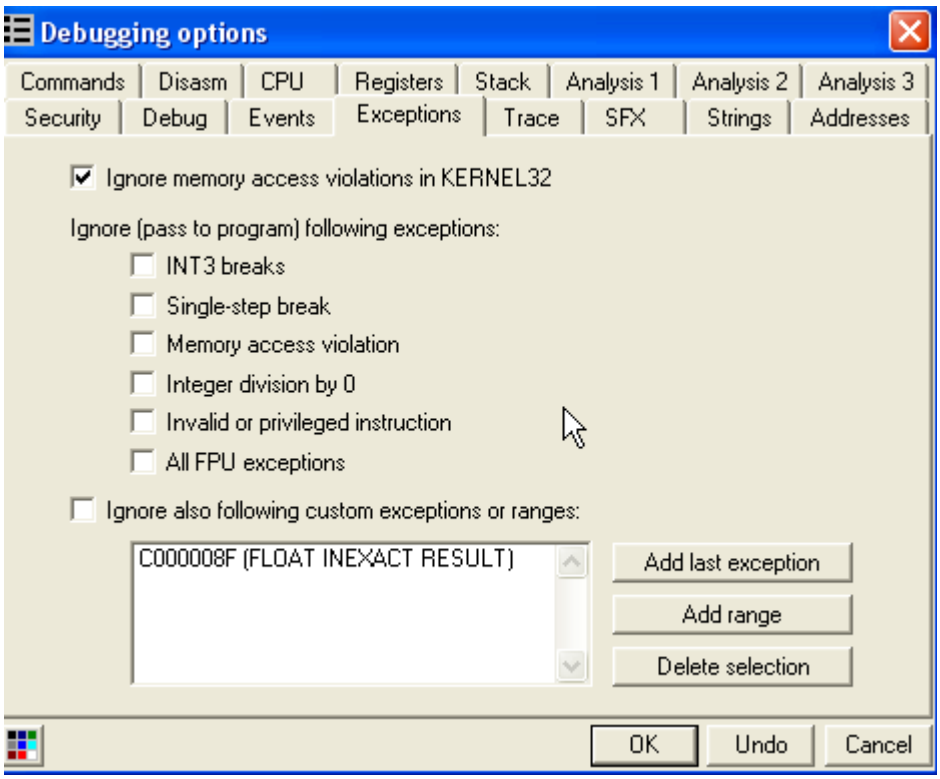
una vez que arranco me fijo en el LOG del OLLYDBG apretando L.



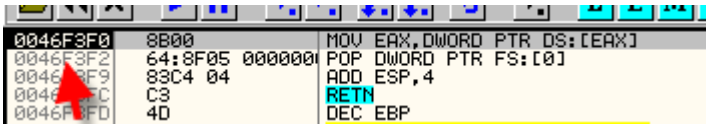


Me fijo la ultima excepcion que se produjo, que no sea producida en la primera seccion o sea que no se haya producido en la ejecucion del programa si no antes, en el desempacado, en este caso la ultima se produjo en 46e88f.

Ahora reinicio y quito todas las tildes en exceptions solo dejo la 1ra.



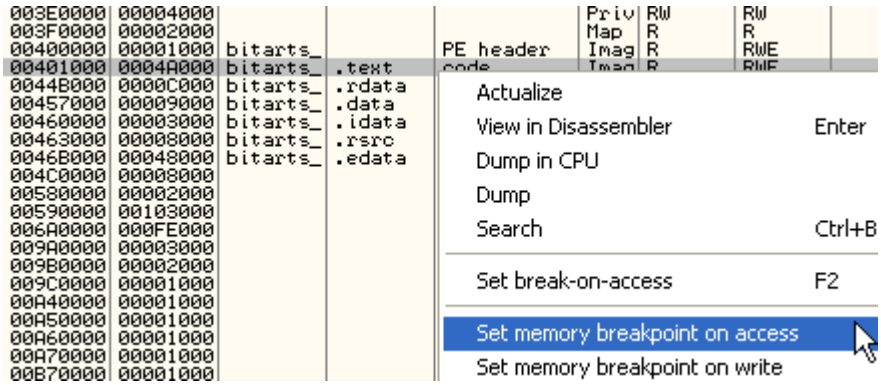
Y doy RUN y cada vez que para voy pasando con SHIFT+ f9 hasta que llego a la ultima que anote, en este caso 46e88f.



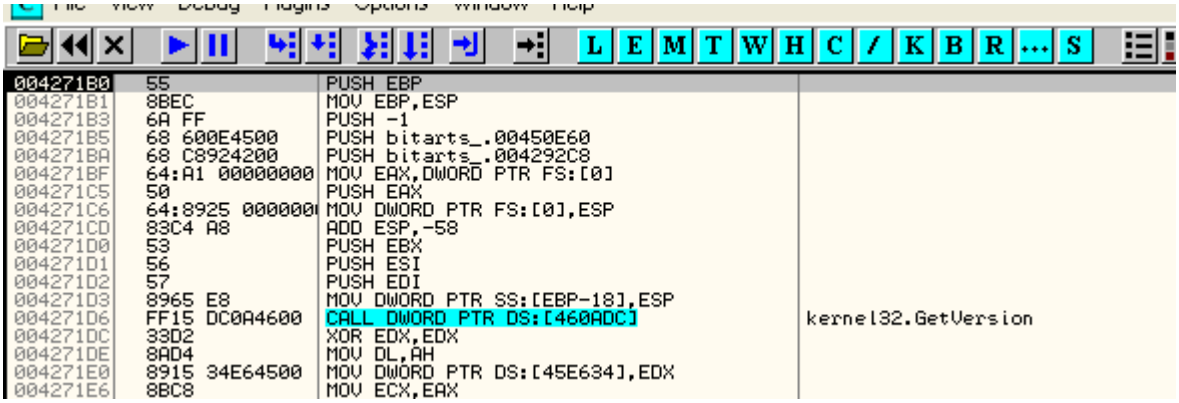
Alli paro como no es la que anote, hago shift + f9 para saltar la excepcion, recordemos que como es un INT3 OLLYDBG parara en la direccion justo siguiente o sea 46e890.

Alli estamos justo en la ultima excepcion generada por el desempacador, antes que arranque el programa.

Aquí tengo varias posibilidades, podemos poner un BPM ON ACCESS en la seccion code, y muchos se preguntaran porque no lo coloque en un inicio, y la respuesta es porque muchos packers pueden detectar el BPM si lo coloco desde el inicio, y al llegar aquí, quizas ya paso la detección, probemos si va.

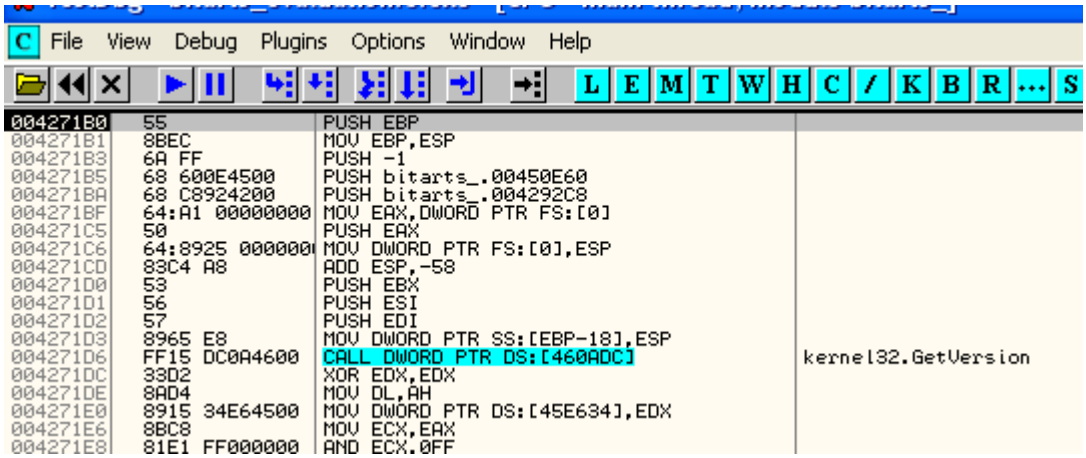


Ahora no olvidemos que debemos apretar SHIFT mas f9 ya que estamos en una excepcion.



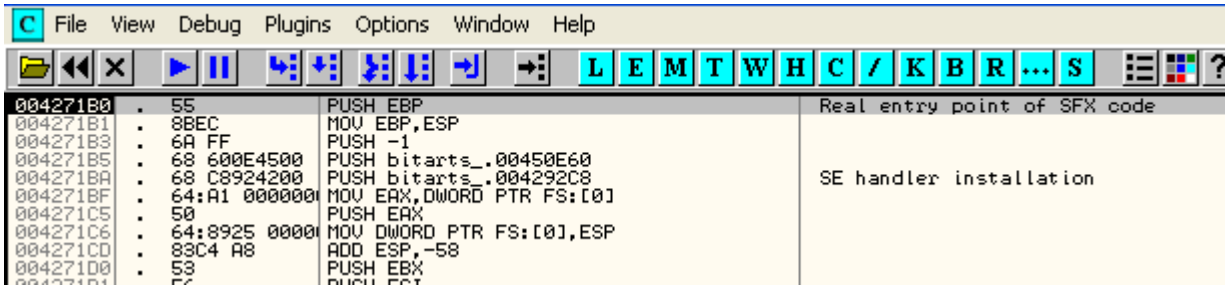
Alli paro en el OEP, podemos probar si el packer detectaba el BPM realizando esto mismo desde el inicio.

Reinicio y coloco todas las tildes en exceptions nuevamente y voy a M y le coloco un BPM ON ACCESS en la primera seccion y como estoy usando el OLLYDBG para detectar OEPs pues me voy a tomar unos mates jeje, ya que vemos que en este caso tardara unos cuantos largos minutos, en llegar al OEP.

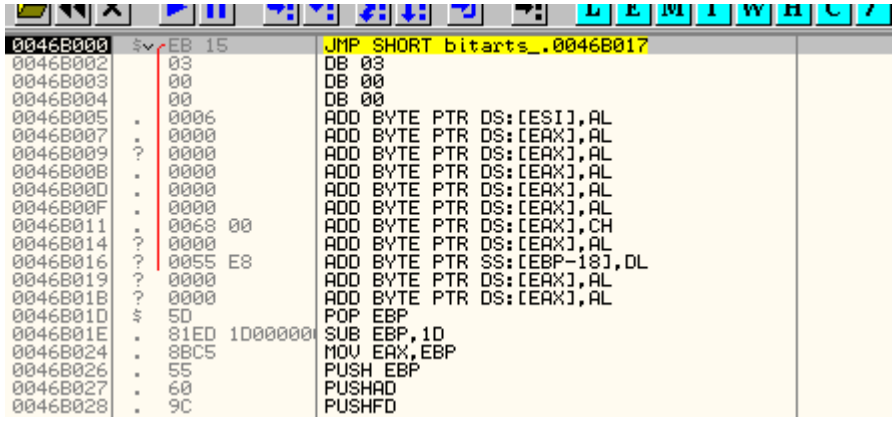


Vemos que llego perfectamente pero es bueno conocer el metodo de las excepciones para cuando el BPM ON ACCESS sea detectado por el packer y en ese caso hay que llegar lo mas cerca posible del OEP, para evitar la deteccion del mismo.

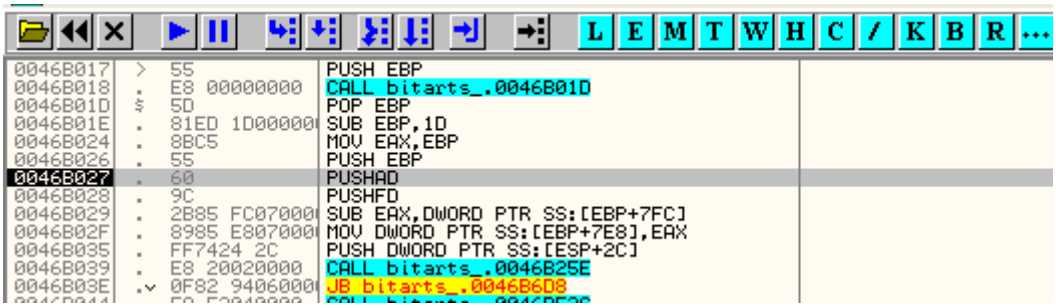
Vemos que en este packer incluso usando el metodo del buscador de OEPs que trae el OLLYDBG tambien para perfectamente en el mismo y rapido.



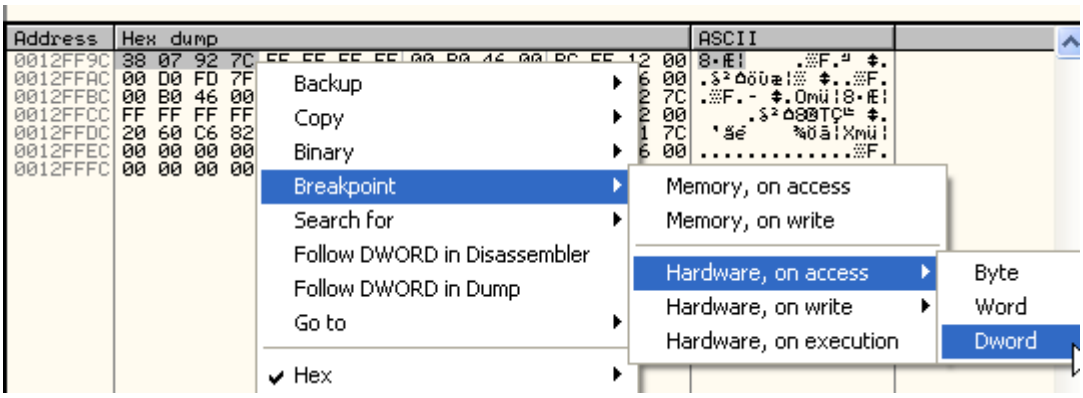
Hasta el metodo del PUSHAD funciona perfectamente, vemos en el inicio



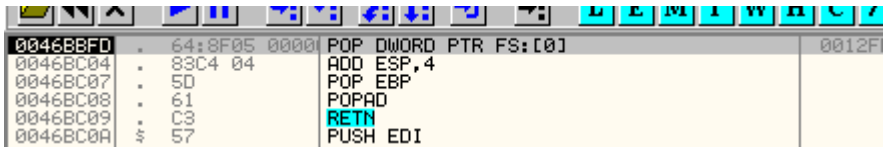
Que si traceamos con f7 unas lineas llegamos a un PUSHAD



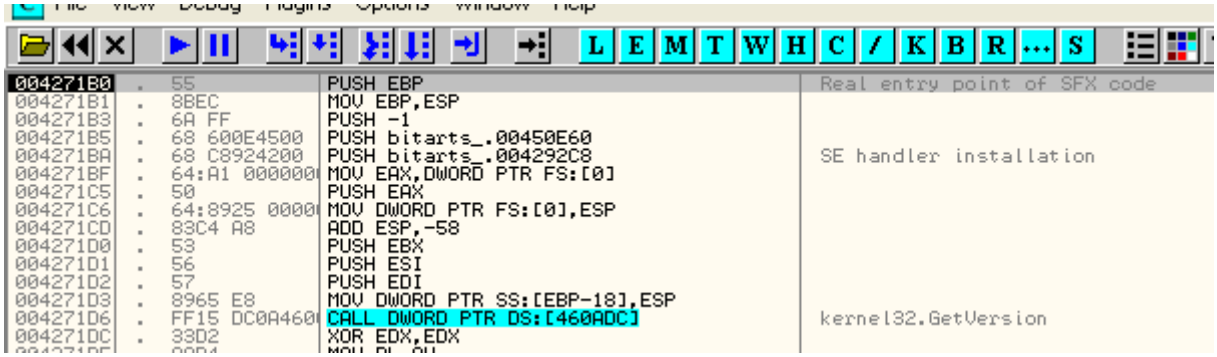
Lo pasamos con f7 y marcamos ESP-FOLLOW IN DUMP



Damos RUN

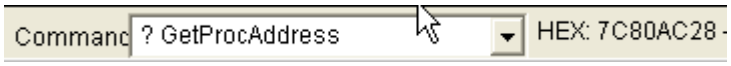


Y para cuando lee los valores guardados en ese POP, traceo con f7 hasta el ret al pasarlo llego al OEP.

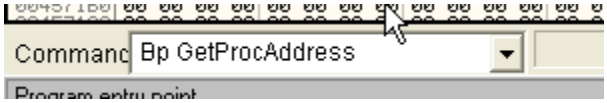


7)EL METODO DE UNA API MUY USADA POR EL DESEMPACADOR

Reinicio este BIT ARTS con el OLLYDBG para OEPs, con todas las excepciones marcadas y buscare una api muy usada normalmente puede ser GetProcAddress, Load Library tambien es muy usada, ExitThread, esto variara según el packer, probemos el metodo con GetProcAddress.

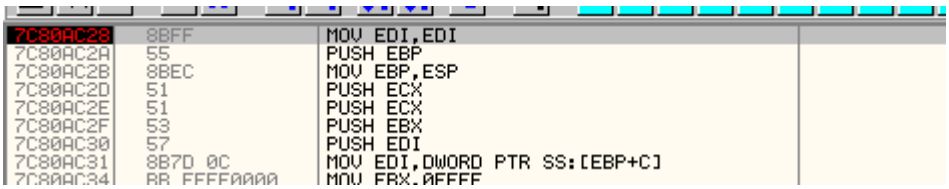


Alli veo en la commandbar la direccion de la api le pondre un BP

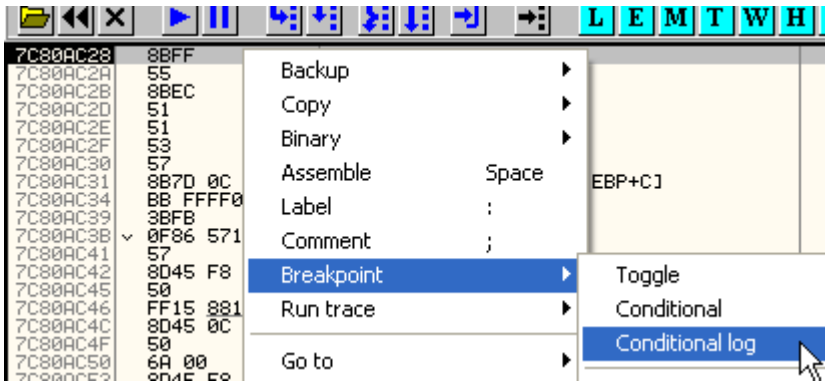


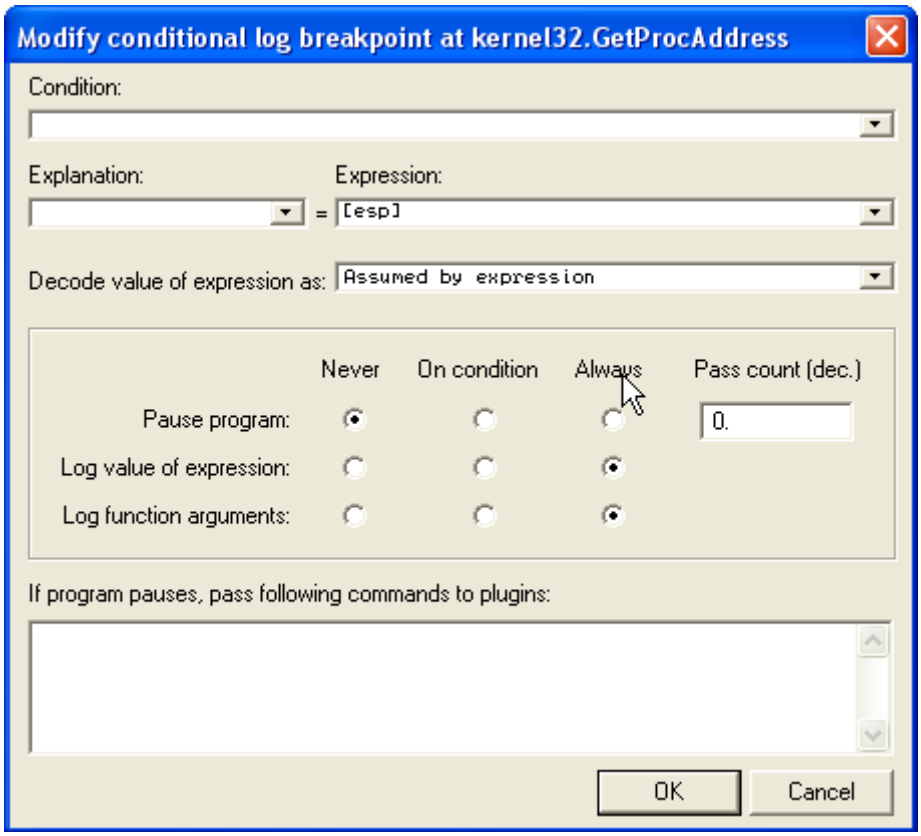
En la api elegida, al menos debe poder ponersele BP, si no se puede desde la comandbar habra que buscarla a mano y ponerselo directo.

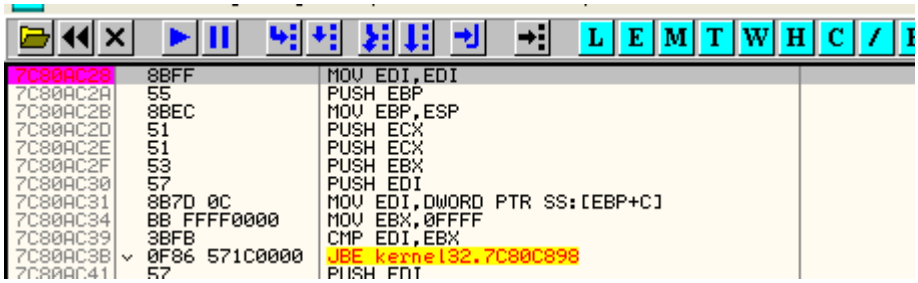
Demos RUN.



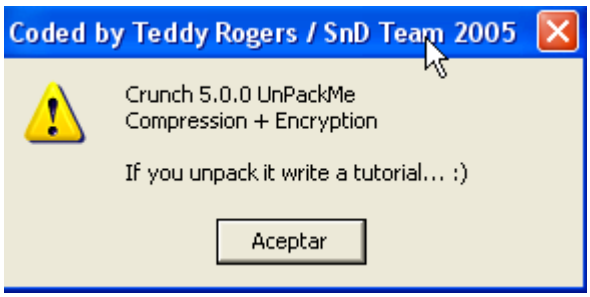
Ahora cambiare este BP por un BP CONDICIONAL LOG que no pare, solo LOGUEE.



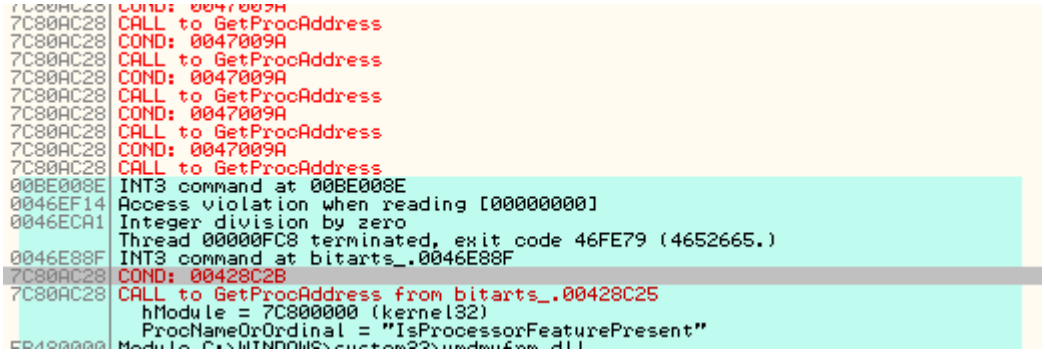




Ponemos que nunca pare, pero que loguee siempre y que ademas nos loguee el valor de [esp] o sea el valor de retorno, para poder saber desde donde fue llamada la api, limpiamos el LOG, haciendo click derecho-CLEAR LOG.

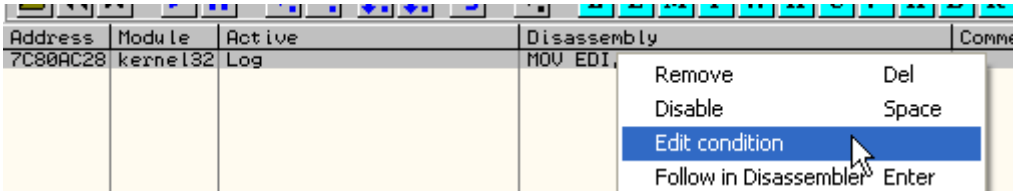


Damos Run hasta que arranque el programa y miraremos en el LOG la ultima llamada a la api que no sea hecha desde la primera seccion.

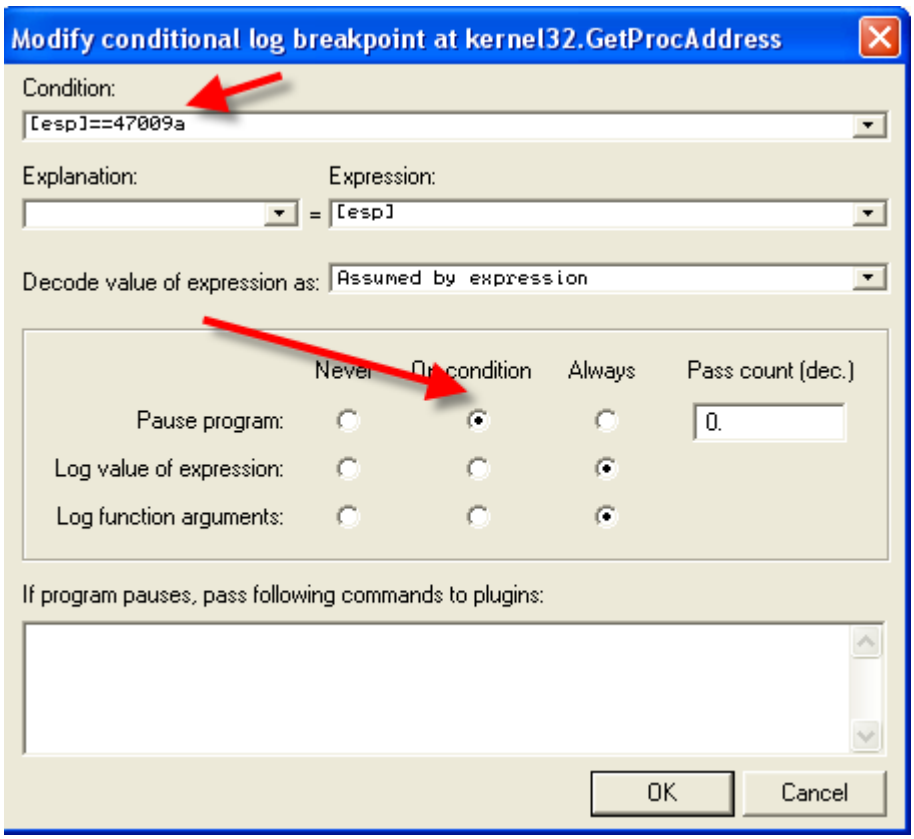


Vemos que hay todas llamadas desde el descompresor, hasta que la siguiente es desde 428c2B que ya es desde la primera seccion o sea que corresponde al programa, o sea que la ultima vez que usa el descompresor esta api, es cuando [esp]==47009A, asi que podemos poner como condicion que pare cuando se de eso.

Reiniciemos.

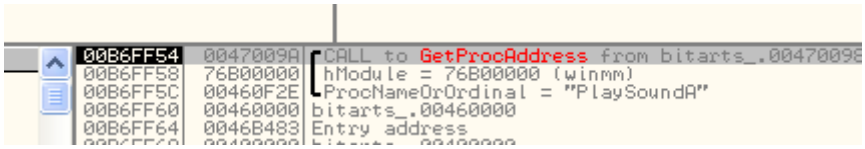


Editemos el BPX CONDICIONAL



Ahora le colocamos la condicion y ponemos la tilde para que pare ON CONDITION

Y damos RUN

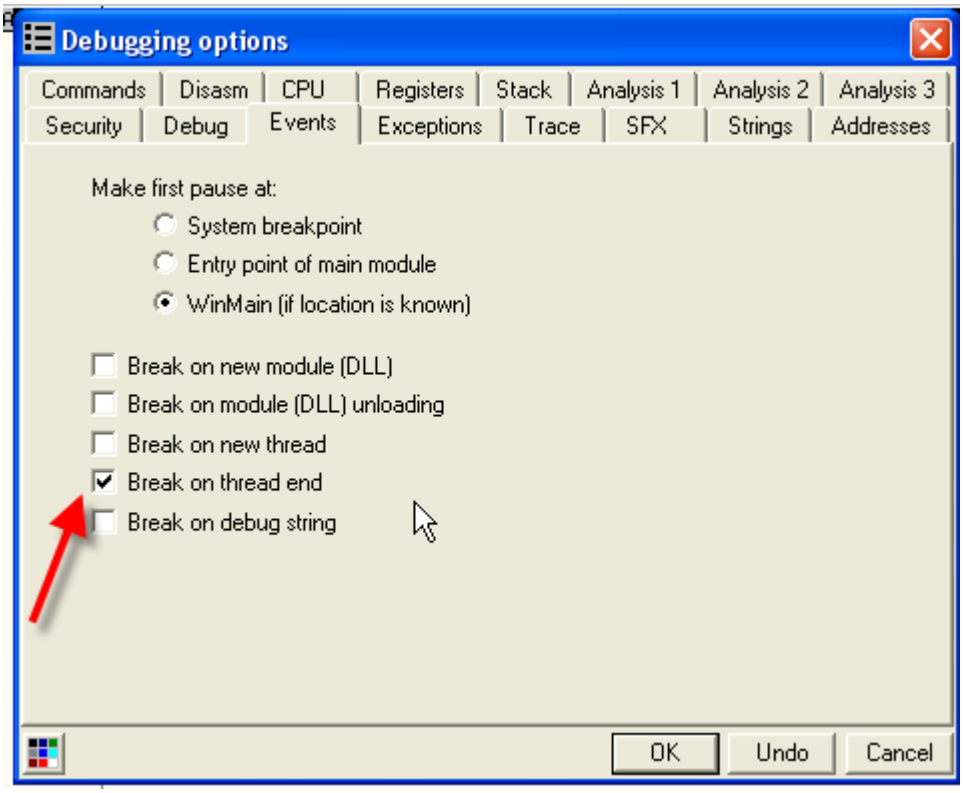


Alli vemos que paro, este metodo se puede completar como antes poniendo BPM ON ACCESS en la seccion CODE, para evitar la deteccion del BPM aunque tiene como problema que muchos nuevos packers detectan el BPX en las apis, por lo cual muchas veces conviene no hacerlo en la primera linea de la api si no en el RET de la misma, el metodo es similar, la idea es buscar que pare lo mas cerca posible antes del oep ya sea para poner un BPM ON ACCESS o ya sea para tracear desde aquí con el trazador automatico que tiene el OLLYDBG (TRACE INTO)

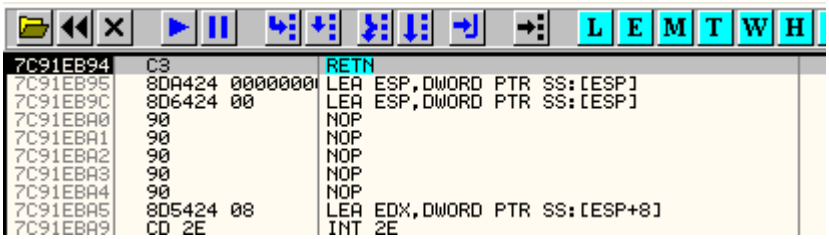
Si hemos llegado bastante cerca del OEP, tendremos mas suerte y habremos elegido bien la api, si no, pues deberemos cambiar de api, si miramos el LOG de este programa vemos que una de las cosas que hace antes de arrancar este, es terminar un thread

```
0046F3F0 Access violation when reading [00000000]
7C810856 New thread with ID 00000DB0 created
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00B8008E INT3 command at 00B8008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00BA008E INT3 command at 00BA008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00BC008E INT3 command at 00BC008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00BE008E INT3 command at 00BE008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
         Thread 00000DB0 terminated, exit code 46FE79 (465266
0046E88F INT3 command at bitarts_.0046E88F
5B480000 Module C:\WINDOWS\system32\umdmxfrm.dll
5B150000 Module C:\WINDOWS\system32\uxtheme.dll
746B0000 Module C:\WINDOWS\system32\MSCTF.dll
73260000 Module C:\WINDOWS\system32\RICHED32.DLL
```

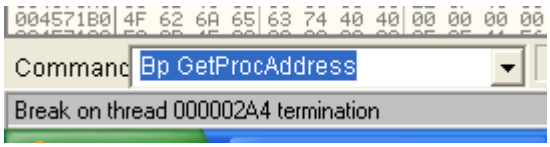
Por lo cual tanto poner un BP ExitThread como cambiar en el OLLY la configuracion para que pare cada vez que se cierra un thread es posible tambien.

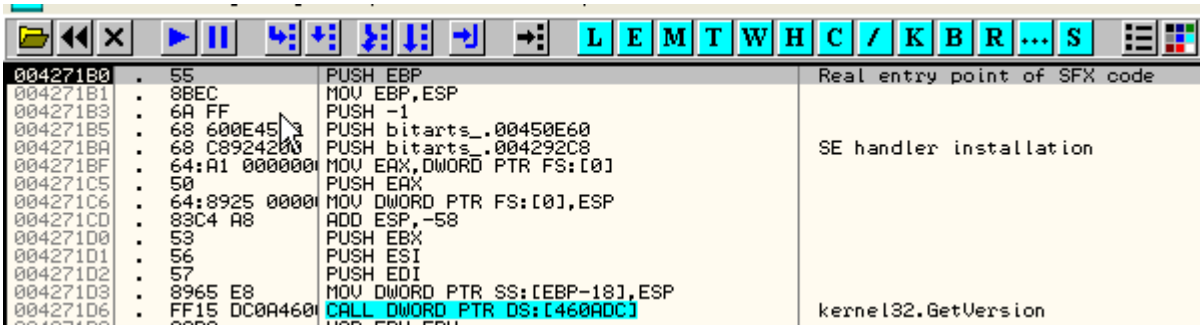


Demos RUN



para alli al terminar el thread



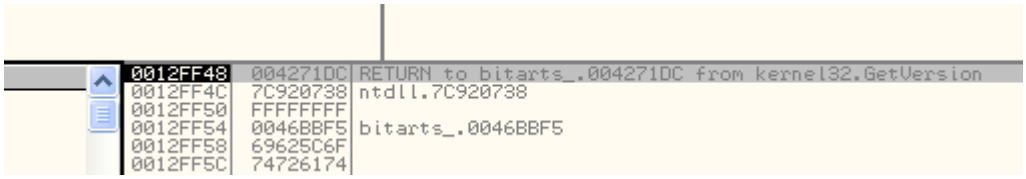


Alli poniendo el BPM ON ACCESS en la seccion CODE parara perfectamente tambien.

8)METODO DE LA PRIMERA API EJECUTADA POR EL PROGRAMA

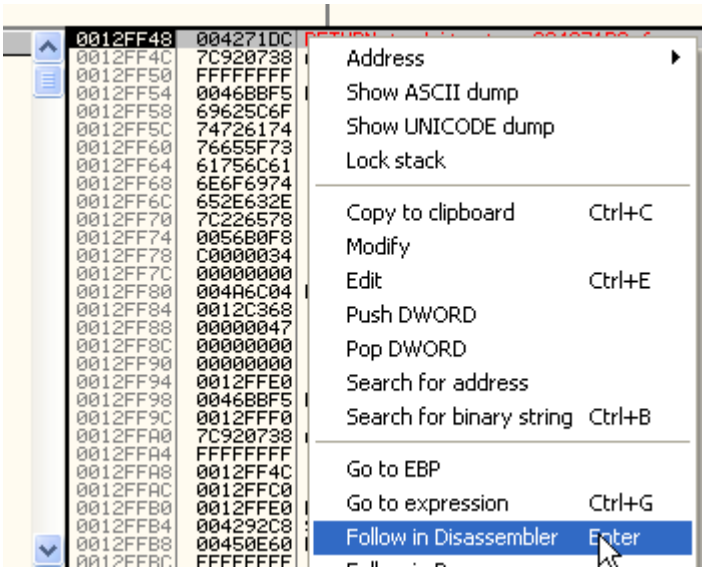
Este metodo es poner un BP directamente en una api sospechosa de ser la primera que ejecuta el programa, normalmente los programas ejecutan al inicio GetVersion, GetModuleHandleA, con mirar unos cuantos programas desempacados obtendremos una lista de las apis mas usadas al inicio no son muchas, en el caso del bitarts vemos que es GetVersion, en el Cruehead es GetModuleHandleA, por ejemplo probemos en el BIT ARTS con BP GetVersion.

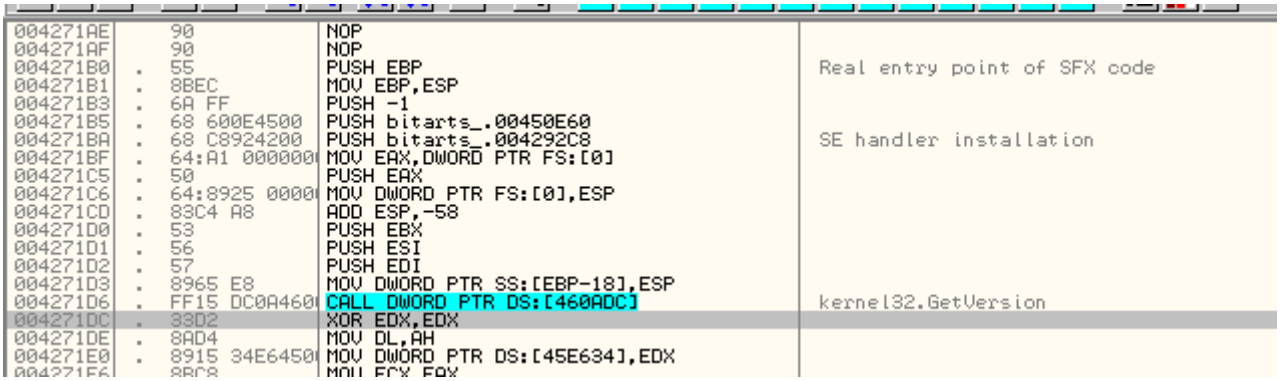
Doy RUN



Me debo asegurar que la llamada sea hecha por el programa o sea, desde la primera seccion.

Cuando para miro el primer valor del stack que es la direccion de retorno y voy alli.





Alli veo el OEP que lo obtuve con el metodo de la primera api utilizada por el programa, si el programa detecta el BP en GetVersion, se puede poner tambien en el RET de la api.

Creo que metodos hay como packers hay y son miles, estos son los ejemplos de los mas usados, ya veran cuando desempaquemos como estos metodos se pueden flexibilizar y adaptar al caso, pero es bueno que tengan una idea de los metodos generales, para poder tener una buena base de los mismos.

Adjunto un crackme para practicar que hallen el OEP el UnPackMe_tElock0.98.exe que tiene algunos trucos y que no le van todos los metodos anteriores, auqnue es bueno que practiquen y hallen el OEP por ustedes mismos.

Recuerden que si un desempacador detecta un BP o HBP, y no corre, deben revisar bien que no haya ningun BP mi HBP puesto, para que vuelva a correr, recuerden que el metodo del PUSHAD deja un HBP puesto que si no funciona el metodo, hay que borrar antes de intentar otro.

Por supuesto el proximo nivel sera un rar con clave, la clave para abrir el mismo sera el OEP del telock0.98 jeje, a practicar y a probar que tienen que ser expertos en hallar OEPs antes de pasar al dumpeo y reparacion de IATs.

Hasta la parte 33
Ricardo Narvaja
20 de febrero de 2006