INTRODUCCION AL CRACKING CON OLLYDBG Parte 7

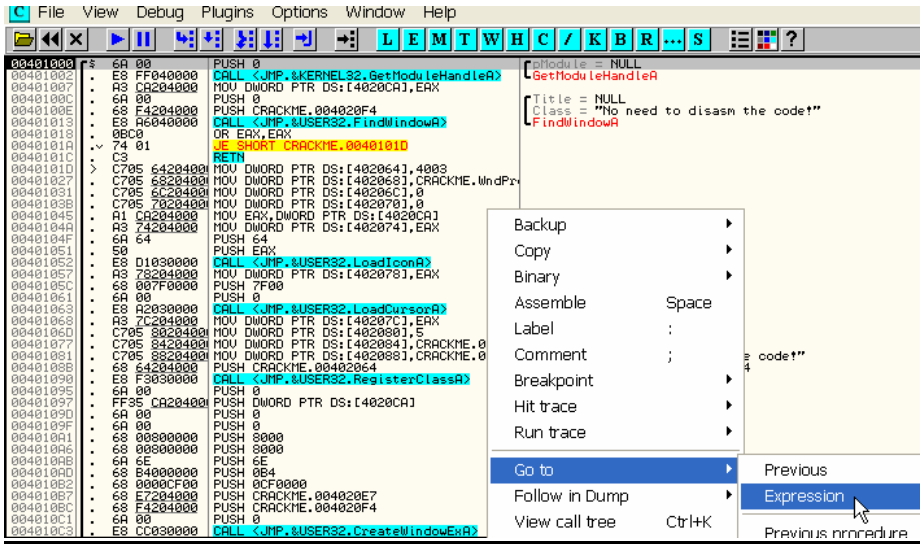
Los CALL y RET

He dejado algunas instrucciones para esta ultima parte porque creo que a esta altura ya tienen claro algunas cosas, y es mejor explicar estas ultimas instrucciones ya teniendo una pequeña base, ahora explicaremos los CALL y RET.

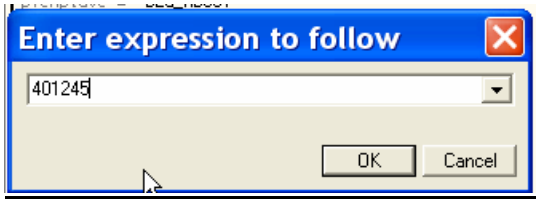
Esta instrucción aunque parezca su funcionamiento sencillo, muchos de los newbies no comprenden realmente su funcionamiento, por eso quería dedicarle un espacio importante e insistir en repasar toldo lo anterior, pero ademas mirar de entender muy bien el funcionamiento de los CALL y RET yo creo que es algo muy importante para el cracking.

Carguemos nuevamente nuestro crackme de ejemplo el CRACKME DE CRUEHEAD en OLLYDBG,

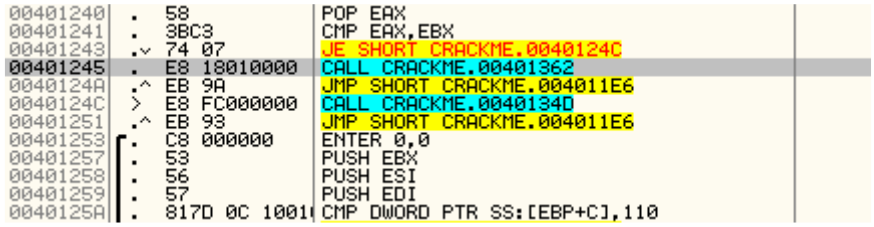
Para practicar hagamos click derecho en el listado desensamblado en cualquier línea y elijamos GO TO – EXPRESSION



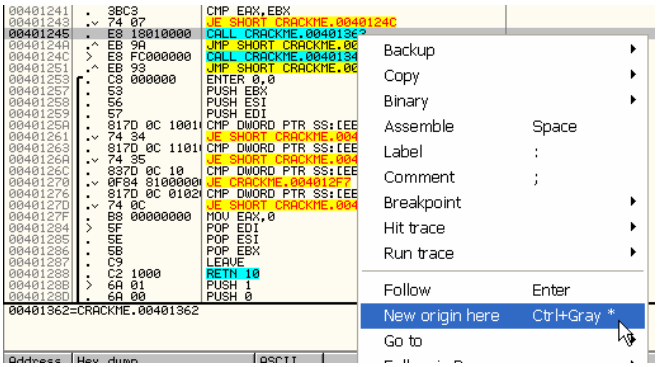
Y en la ventana que aparece tipeemos 401245



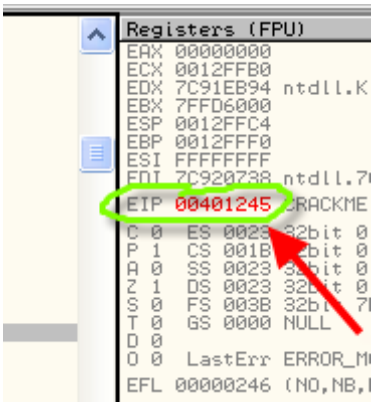
Nos llevara a dicha dirección en el listado desensamblado, donde hay un CALL para practicar.



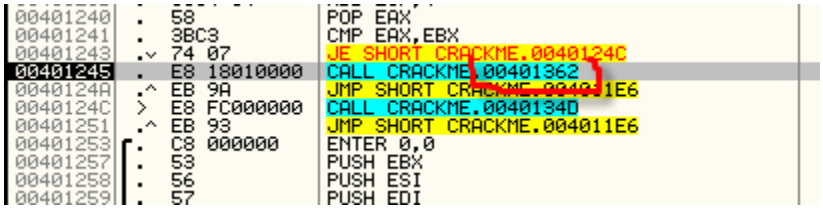
Allí vemos el CALL que elegí para practicar, para poder ejecutarlo hago CLICK DERECHO-NEW ORIGIN HERE con lo cual EIP apuntara a 401245 y dicha dirección será la próxima que se ejecutará.



Allí vemos como cambiamos EIP



Volvamos a nuestro CALL



La instrucción CALL lo que hace es ir a ejecutar una subrutina o si quieren parte del programa cuya dirección esta dada por el valor del operando o sea en este caso del ejemplo:

CALL 401362 significa que la próxima línea a ejecutarse será 401362 y cuando se termine de ejecutar la rutina que esta allí dentro, volverá a la instrucción siguiente a continuación del CALL que la llamo.

En este ejemplo luego de ejecutarse el contenido del CALL 401362, volverá a 40124A y seguirá desde allí.

Ahora hagamos unos ciertos movimientos con el OLLYDBG, que nos ayudan en los casos que estamos encima de un CALL como este.

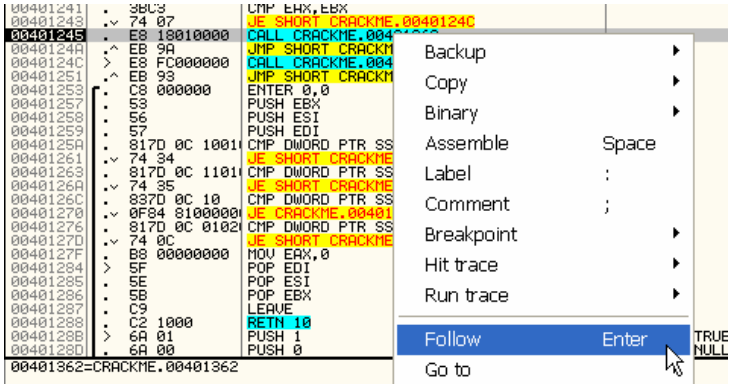
Si yo quiero mirar el contenido del CALL podría entrar al mismo con F7 y tracearlo por dentro, pero si quiero echar un vistazo a ver si el contenido me interesa para ser traceado o no, ya que

también tengo como recuerdan la opción de tracear con la tecla F8, la cual en este caso ejecutara el CALL sin entrar al mismo y seguirá en 40124A sin ni siquiera enterarnos de lo que hizo el programa dentro del CALL.

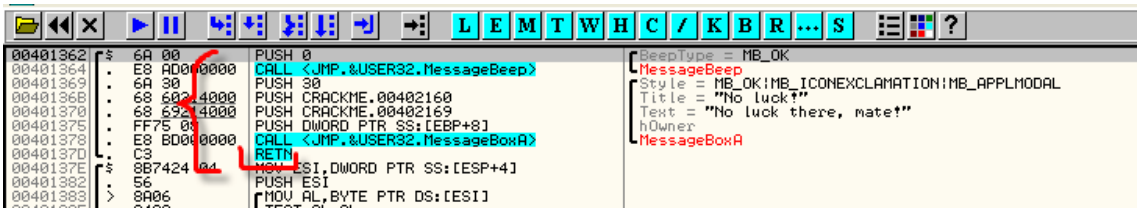
Entonces cada vez que llegamos a un CALL y estamos traceando un programa se plantea una disyuntiva, será importante para entrar a tracearlo con F7? O lo salteo con F8 porque es un call secundario que no hace nada que me importe?

Bueno, lo que OLLYDBG nos permite hacer es mirar sin ejecutar, a ver que veo dentro del CALL y si me interesa.

Para ello hago click derecho en el CALL y elijo FOLLOW.



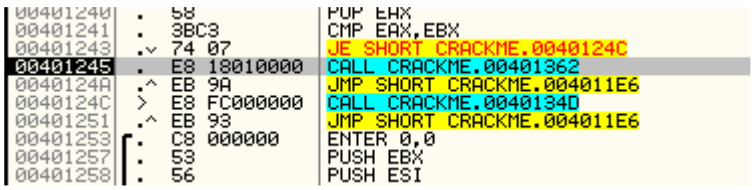
FOLLOW no ejecuta ninguna línea solo va a mostrarnos la próxima línea a ejecutarse, pero no cambia nada EIP seguirá en 401245, a la espera de nuestra decisión.



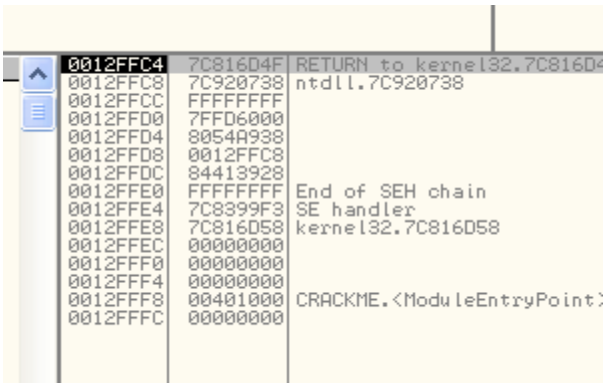
Cuando entro a mirar veo la rutina interna que obviamente empieza en 401362 que era el operando del CALL y donde terminará, pues en el primer RET que vea en este caso OLLYDBG escribe los RET como RETN, pero es lo mismo dicha instrucción es la finalización de la rutina y la vuelta a 40124A a la instrucción siguiente del CALL que nos trajo aquí.

Es muy importante entender esto, ahora que vimos como podemos mirar dentro sin ejecutar volvamos a la línea actual presionando la tecla - del teclado numérico o sea la tecla MENOS, esto siempre nos lleva al paso anterior sin ejecutar nada también.

Pues estamos en el CALL

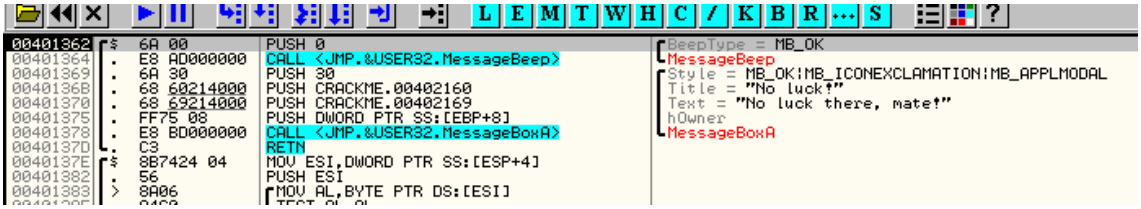


Ahora si entraremos con F7, pero antes de entrar el CALL veamos el stack, esto es muy importante pues allí se almacenan los datos para que el programa sepa donde regresar cuando llega al RET.



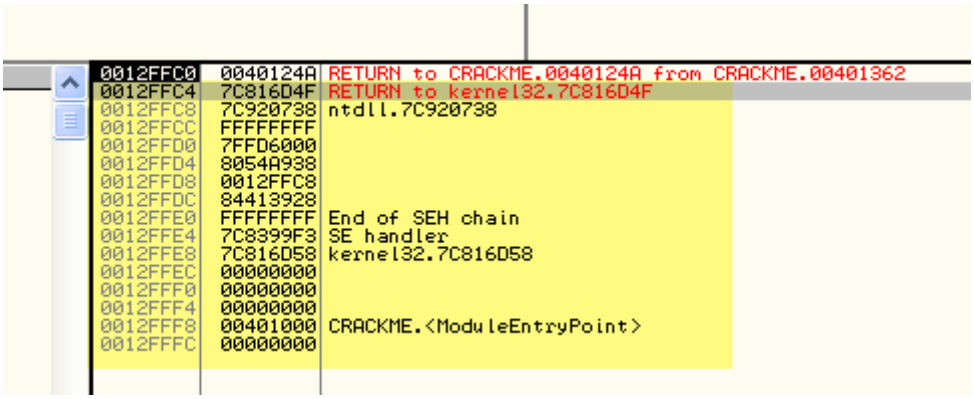
En mi maquina este es el stack, en la suya los valores pueden variar pero el mecanismo será similar.

Apreto F7 ahora



Allí entre en el CALL y estoy en 401362 pero a diferencia de la vez anterior que entre con FOLLOW ahora EIP cambio a 401362, lo cual quiere decir que estamos ejecutando esta rutina.

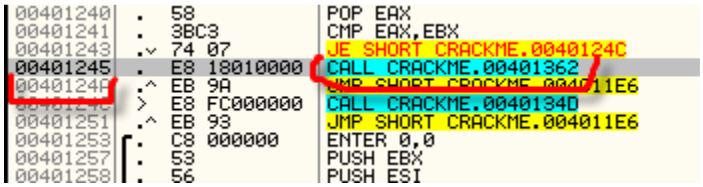
Veamos que paso en el stack



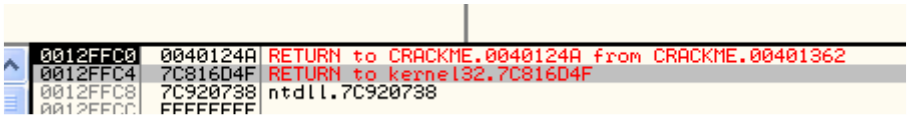
Allí en la imagen resalte el stack como estaba antes y veo que se agrego una nueva carta arriba o sea al entrar en un CALL el sistema automáticamente hace un PUSH con la dirección de retorno, o sea donde volverá al llegar al RET y salir de la rutina.

Vemos que la línea tiene el valor 40124A que es como sabemos la dirección de retorno.

Por si alguno lo olvido recordemos que es la dirección siguiente al CALL inicial, allí se ve.



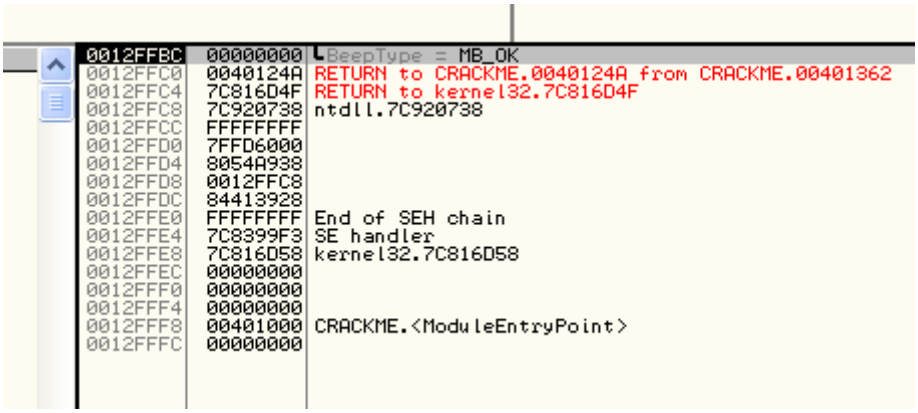
Bueno vemos que OLLY nos aclara aun mas el tema agregándonos información.



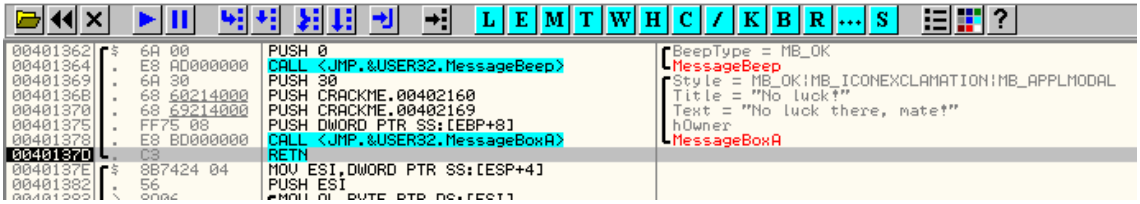
Nos dice RETORNO A 40124A desde 401362

O sea OLLY aun no sabe donde estará el RET que nos devolverá, pero sabe que la rutina empieza en 401362 y lo marca como que aquí empieza y terminara en un RET y volverá a 40124A.

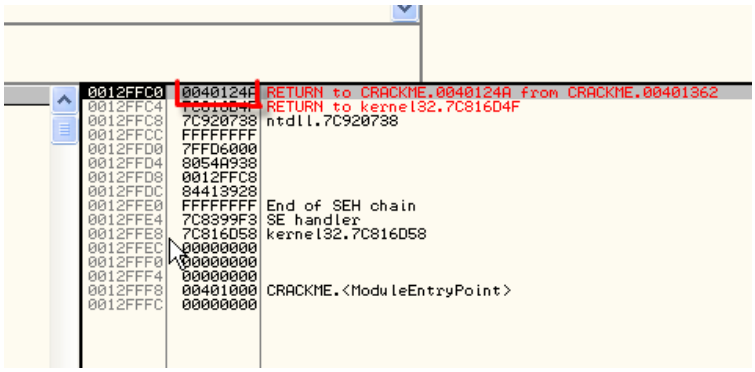
Apretemos F7 una vez vemos que ejecuta un PUSH 0 lo cual pone un CERO en el stack y abajo del mismo queda el indicador para el RET de la dirección de retorno.



El programa puede hacer mil operaciones dentro de la rutina hacer miles de PUSH, POPS lo que quiera, pero al llegar al RET deberá dejar arriba en el stack nuevamente el valor de la dirección de retorno, sigamos ejecutando con F8 para no entrar en los CALL y llegar al RET.



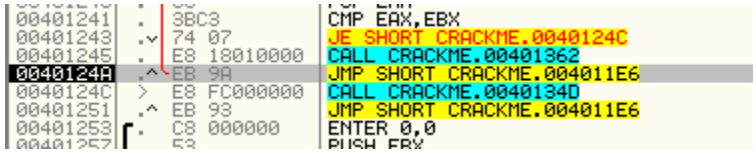
Allí llegue al RET y veo que como dije el stack tiene arriba nuevamente el valor donde retornara.



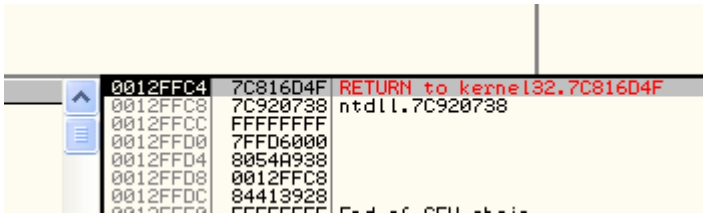
Como habíamos mencionado el RET es el final de la rutina y contrapartida de la instrucción CALL, si CALL nos trajo aquí, RET nos devolverá a la zona donde estábamos antes de entrar en esta rutina.

Pero además RET quitara la dirección de retorno del stack que es un valor que al sistema ya no le interesa pues ya volvemos y luego la dirección ya no es útil más.

Apreto F7



Y vuelvo a 40124A y el stack quedo como estaba antes de ejecutar el CALL en mi maquina en 12FFc4.



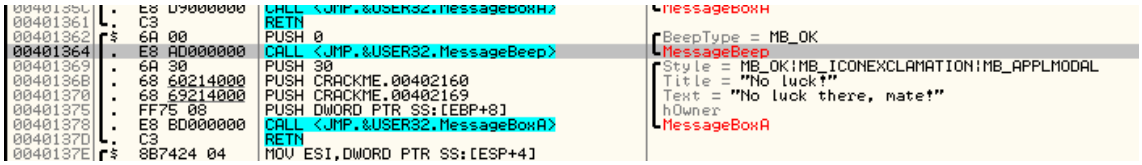
Es de mencionar que si uno ejecuta un RET sin haber entrado en ningún call por ejemplo

PUSH 401256
RET

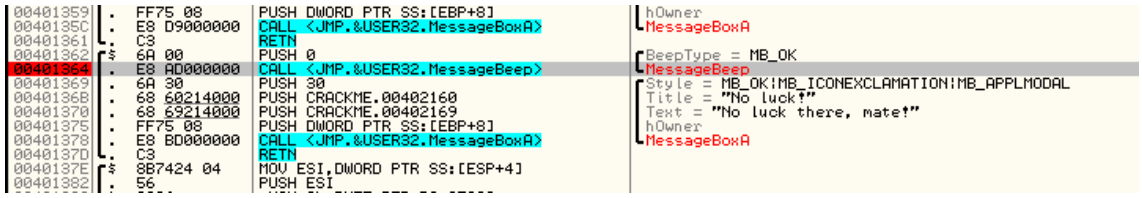
Lo que hará esto es poner en el primer lugar del stack el valor 401256, y como la siguiente instrucción es un RET, pues ella interpreta que el primer lugar del stack es una dirección de retorno de algún call anterior y aunque ello no haya ocurrido, al ejecutarlo nos llevara allí.

Ahora reinicio el CRACKME DE CRUEHEAD

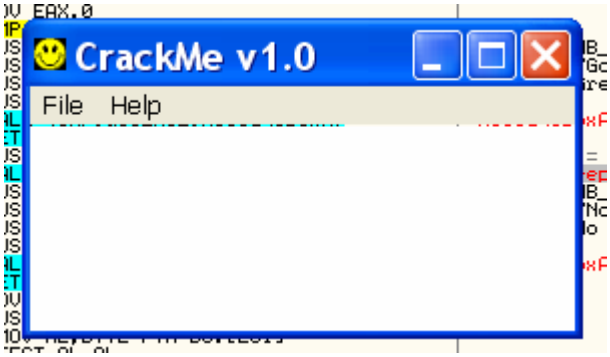
Ahora haré otro ejemplo voy en el listado con GO TO – EXPRESSION a 401364 es una dirección que elegí para mostrar algo ya verán.



Ahora no cambiare EIP ni nada si no que apretare F2 que es un BREAKPOINT los cuales ya explicaremos mas adelante en detalle, lo importante es que cuando el programa ejecute esa instrucción OLLYDBG parara allí.

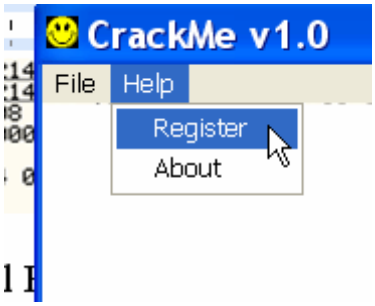


Allí esta puesto el BREAKPOINT ahora apreto F9 que es RUN para que el programa corra.

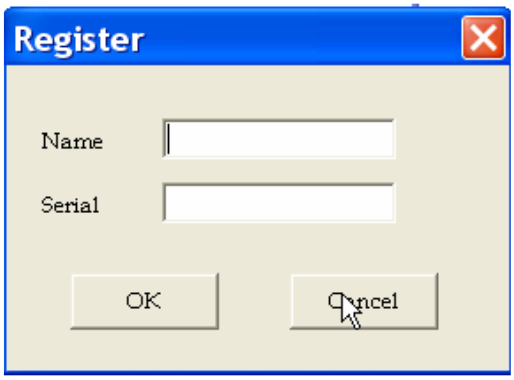


Vemos que nos sale la ventana del crackme, si no la ven, pues búsquenla con ALT mas TAB entre los programas que están corriendo.

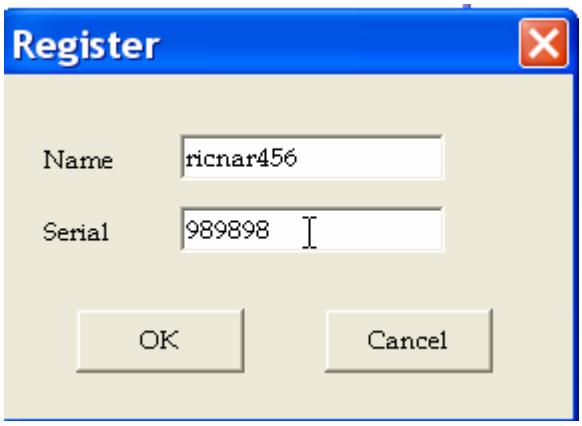
Allí se ve, el programa no paso aun por nuestro BREAKPOINT, vayamos en dicha ventanita a HELP - REGISTER



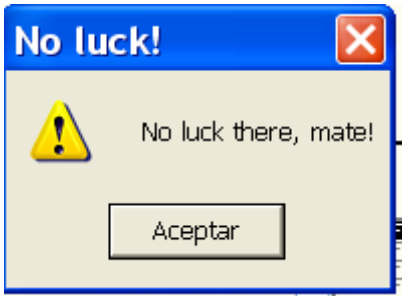
Nos sale la ventana para poner un NOMBRE Y UN SERIAL



Pongamos cualquiera



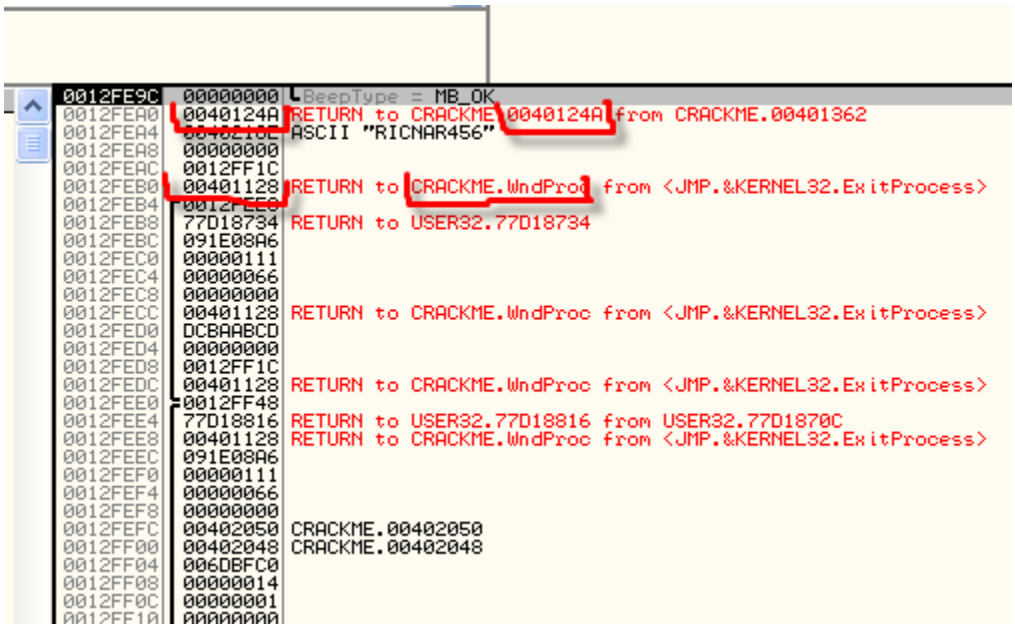
Y apreto OK



Nos sale una ventanita diciendo que no tuvimos suerte o sea que tipeamos el user y serial incorrecto (lo increíble seria que fuera el correcto jeje), y al aceptar dicha ventana para en nuestro BREAKPOINT.

Si no les para prueben con el user y serial que puse yo.

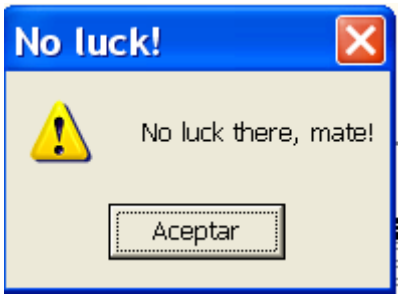
Allí estamos en el medio de la ejecución del programa pero algo de información tenemos



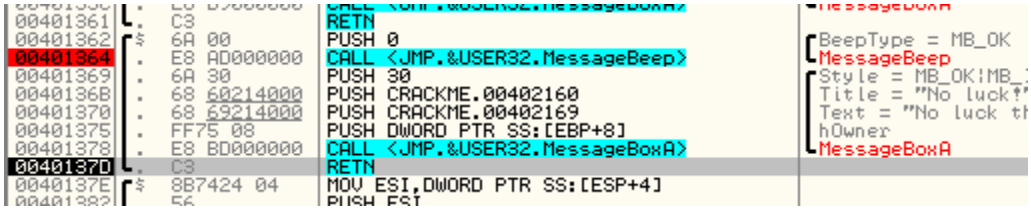
Vemos allí unos cuantos RETURN TO …. Que el stack tiene almacenados allí, así que podemos pensar que el superior de todos, será donde volverá el programa al llegar a un RET ya que suponemos estamos dentro de un CALL (porque vemos que hay RETURN TO …… en el stack) y al llegar a un RET ese RETURN quedara en la línea superior y volverá a 40124A .

Como vemos también estamos en la misma rutina que analizamos antes cambiando el EIP, pero ahora dentro de la ejecución el programa, no suelta.

Tratamos de llegar al RET apretando F8 como antes, en el call anterior al RET se para ya que nos debe mostrar algo que hace el CALL dentro que es una mensaje y que debemos aceptar para seguir.

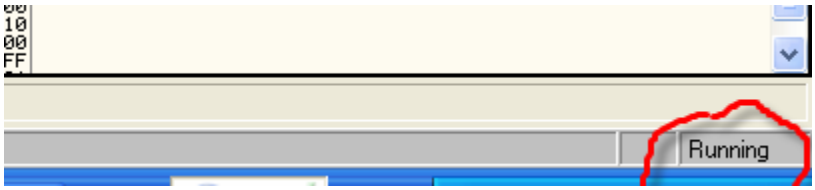


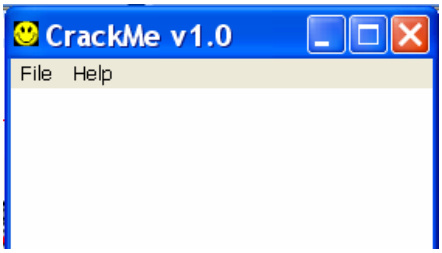
Presionamos ACEPTAR



Y llegamos al RET y como supusimos el valor superior del stack es

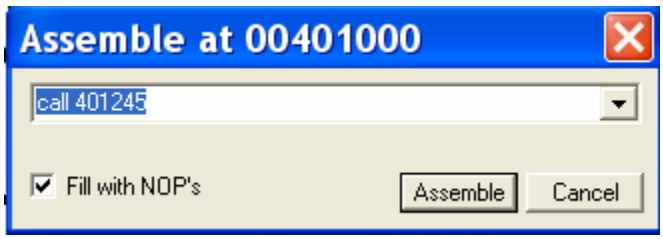
La diferencia entre la vez anterior y esta es que la vez anterior al cambiar EIP e ir directamente allí, ejecutamos el CALL solo aislado, no el resto del programa, ahora al poner un BREAKPOINT el programa se ejecuto normalmente, y al pasar por allí paro, y si apreto F9 seguirá corriendo como si nada.

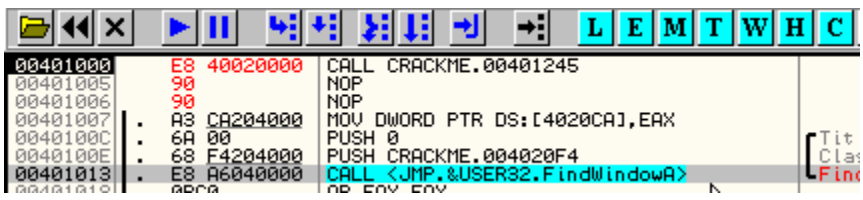




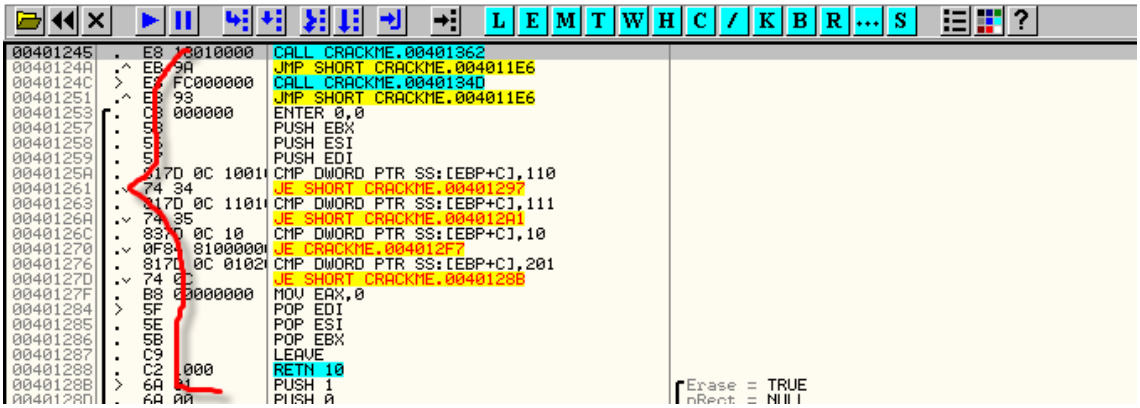
Además lo que quise mostrar es que a veces cuando estamos en el medio de la ejecución de un programa y paramos por algún motivo, la información del stack nos sirve para saber de donde fue llamado la rutina en que estamos y donde volverá, y si hay mas RETURN TO hacia abajo, también sabremos que son CALLS unos dentro de otros, o sea anidados, y que estamos dentro de un CALL que al llegar al primer RETURN TO, saldremos y al llegar al segundo pues saldremos de otro y así.

Creo que esta claro, igual como es muy importante que entiendan esto aclararemos con otro ejemplo reiniciemos el OLLY y apretamos la barra espaciadora y escribamos como primera línea solo para entender esto CALL 401245



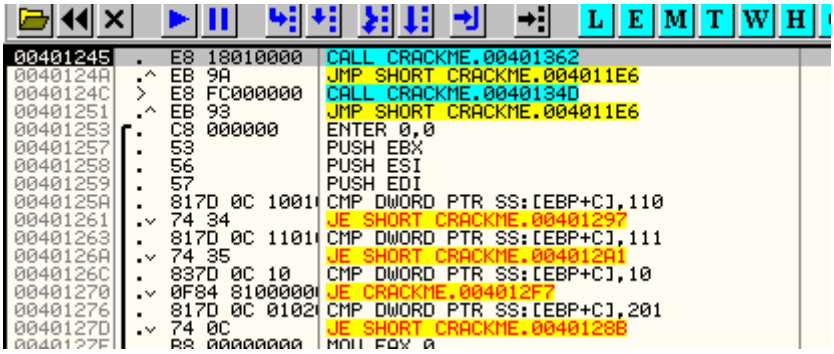


Alli esta ahora podemos practicar hacer FOLLOW para la rutina por dentro

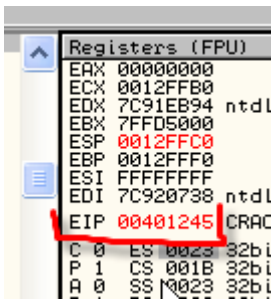


Vemos que como modificamos el programa la rutina ahora empieza en 401245 y terminara en el RET de 401288 (que es un RETN10 un poco diferente a un RET común, pero no es eso el tema de esta explicación ya lo veremos mas adelante lo que quiero que vean es lo que pasa cuando entramos en un call como en este caso y dentro de la rutina entramos en un segundo CALL.

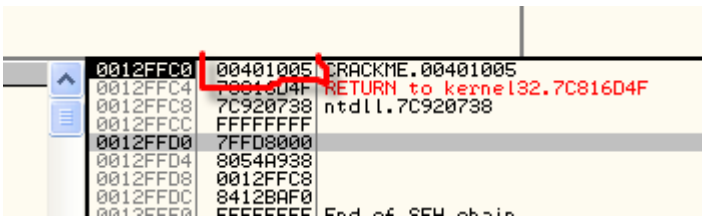
Bueno ya hicimos FOLLOW y miramos ahora apretemos MENOS para volver y ahora si apretemos F7 para entrar ejecutando.



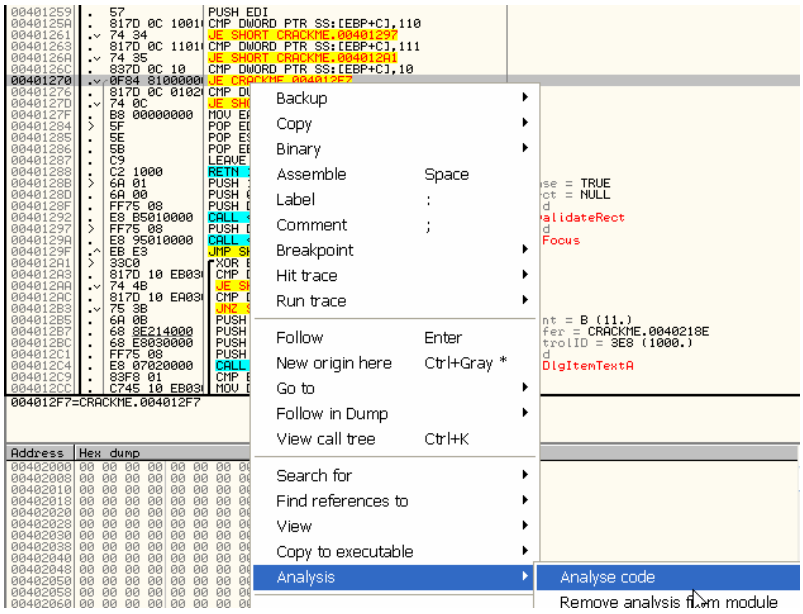
Allí estamos y EIP apunta a 401245 que es la próxima instrucción a ejecutarse



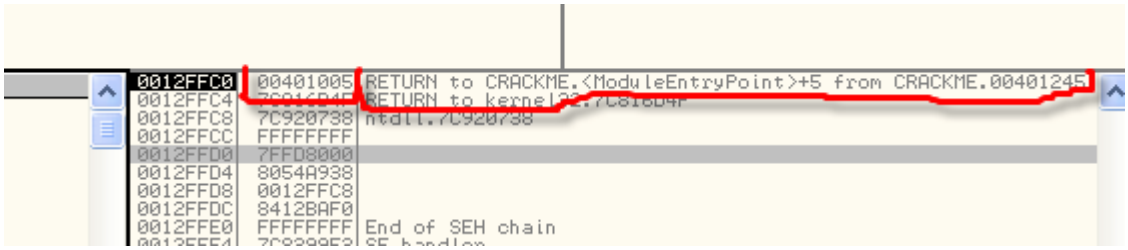
Y en el stack vemos que en el primer lugar esta el valor de retorno a la línea siguiente del call que escribimos a mano.



Vemos que el valor esta, pero OLLYDBG no nos aclaro RETURN TO 401005, porque ocurre esto, es bueno entenderlo para saber como funciona OLLYDBG, como nosotros agregamos el CALL después del análisis inicial que el OLLYDBG había hecho, pues de esta forma, le hemos cambiado el caballo en el medio del río y lo hicimos ahogar jeje, por lo cual si queremos arreglarlo, debemos hacer en el listado en cualquier línea CLICK DERECHO- ANALICE CODE, con lo cual lo volverá a pensar después de los cambios que hemos introducido.



Vemos que después de reanalizar el código

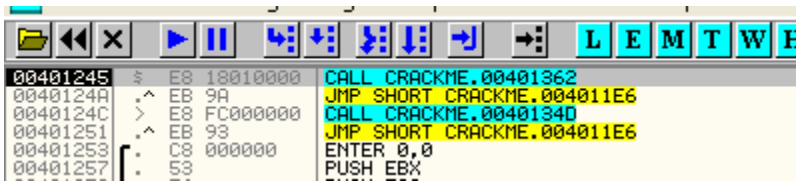


Bueno nos dijo que volverá a MODULE ENTRY POINT + 5

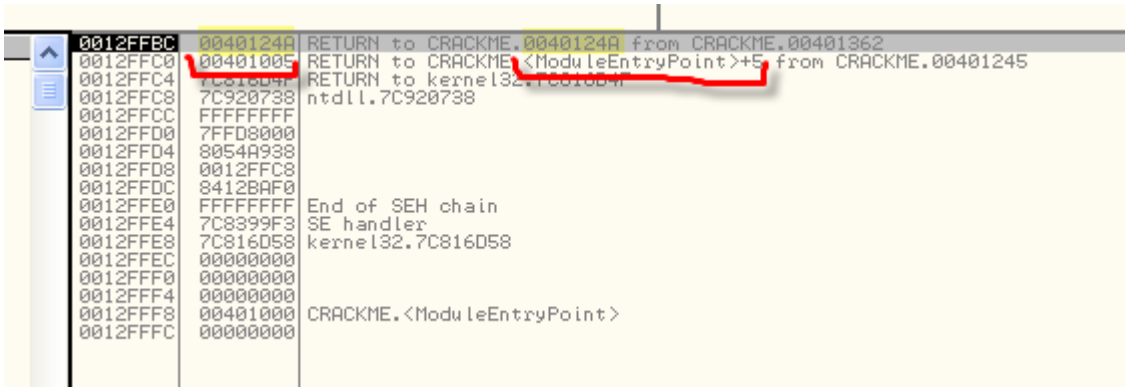
Dicho valor es 401000 que era el ENTRY POINT mas 5 =401005

Es muy importante esto porque muchas veces uno le dice a una persona que le pide consejo, fíjate los RETURN TO …… en el stack, pero resulta que uno mismo ha modificado cosas o el mismo programa se ha auto modificado al ejecutarse entonces el análisis inicial del OLLYDBG fallara en las aclaraciones y debemos actualizarlo al detenernos, o quitar el análisis como vimos si nos trae problemas en la parte 1.

Bueno aclarado esto volvemos a donde estábamos



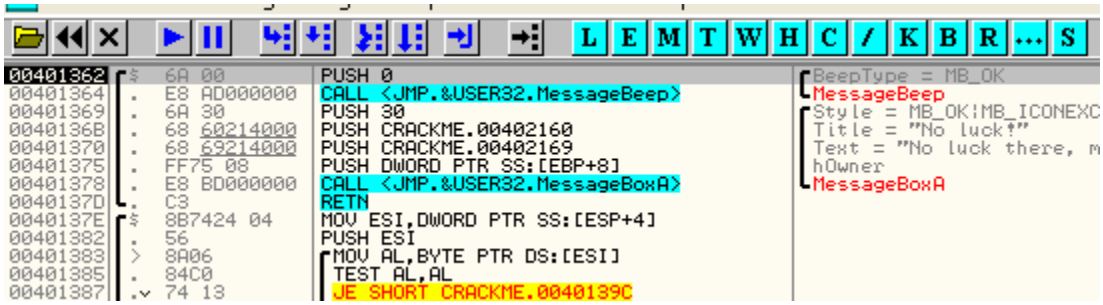
Apreto F7 para entrar en el segundo CALL



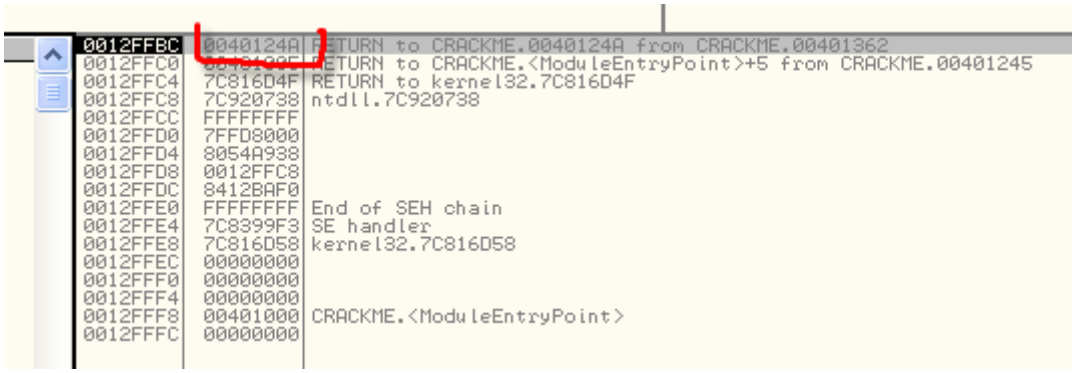
Vemos que arriba de donde guardo la dirección de retorno del primer call ahora guarda la segunda dirección de retorno de este, en este caso están consecutivos, pero podría haber valores numéricos intermedios productos de PUSH o operaciones diversas, lo importante es que el primer RETURN TO .. que hallamos de arriba hacia abajo es la dirección de retorno del ultimo CALL que entramos y la segunda que hallemos bajando será la dirección de retorno del call anterior que es el inicial que escribimos.

Esta es la idea de calls anidados o uno dentro de otro, si yo por poner un BREAKPOINT o por algún motivo parara aquí en el medio de la ejecución del programa, aunque no haya venido traceando ni tuviera información de cómo se vino ejecutando el programa al ver el stack , puedo sacar como conclusión

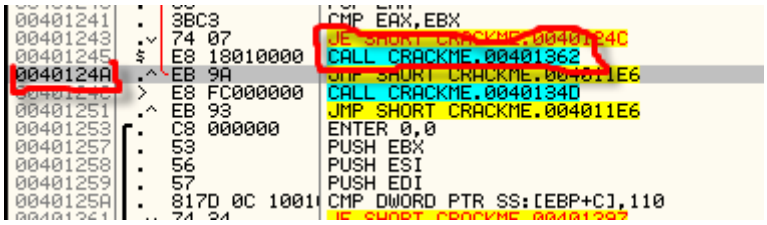
1) CAI AQUÍ Y VEO UN RET un poco mas abajo, supongo que estaré dentro de un CALL para confirmarlo miro el stack



2) MIRO EL STACK

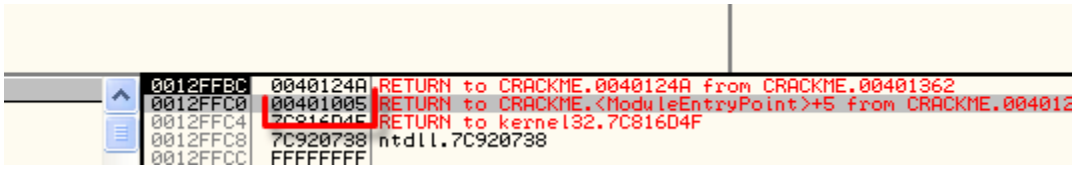


Al mirarlo y buscar desde arriba del stack hacia abajo y hallar el primer RETURN TO se que el programa al llegar al RET volverá a 40124A.

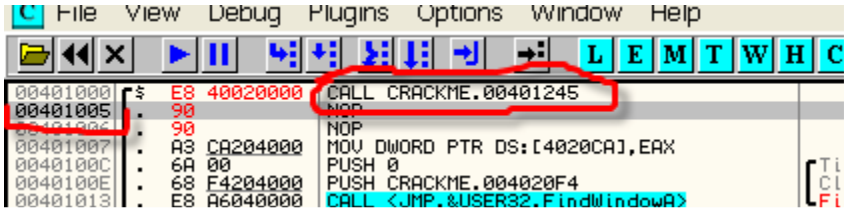


Y además de saber donde volverá se que estoy dentro de la ejecución del CALL ANTERIOR al punto de retorno, o sea que el programa llamo a la zona donde estoy desde 401245 usando un CALL que llama a la rutina de 401362.

Y no solo se eso si no que se también que antes de llegar a ese CALL, había entrado antes en otro ya que hay otro RETURN TO …. Mas abajo



Ese me informa que luego de ejecutar todo saldrá a 401005 y además se que el programa provenía del CALL de la línea anterior o sea



Solo parando y mirando el stack ya determine que el programa provino de aquí que entro en ese call, luego fue a 401245, allí había otro call que me llevo a 401362 y de esa forma llegue a la zona donde estoy.

Esa forma de pensar en el cracking es muy útil porque me hace reconstruir la forma que el programa fue llegando a cierto punto, que muchas veces no son dos calls uno dentro de otro si no que hay 30 calls uno dentro de otro y uno no puede tracearlos todos, así que si caigo en un punto, puedo hacer un análisis personal, y llegar a la conclusión de cómo el programa arribo al punto donde me encuentro en este momento.

Espero que haya quedado claro, lo de los CALL Y RETS les sugiero practicarlos repasarlo, si tienen dudas preguntar porque esto es muy importante, iba a terminar a continuación con los métodos de direccionamientos y algunas instrucciones que quedaron en el tintero, pero preferiría que le den buena importancia a entender esto y en la próxima parte continuamos con los temas pendientes.

Hasta la parte 8
Ricardo Narvaja
18 de noviembre de 2005