Depurando "Hola Mundo"

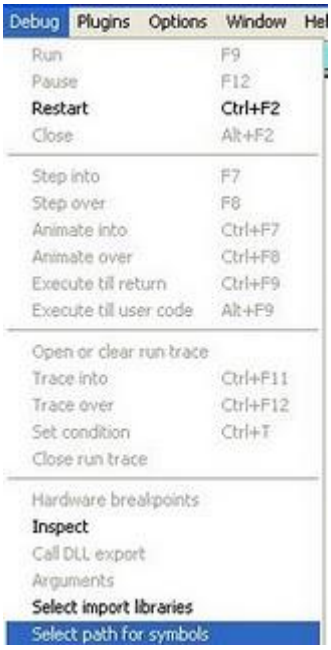
Hoy vamos a hablar de cómo se depura un programa, lo que nos servirá como excusa para hablar del stack, las secciones de un programa, etc, etc ...

DEBUG SYMBOLS

Antes de nada, va a venir muy bien tener los debug symbols e incorporarlos a Olly. Los debug symbols contienen información generada en precompilación de las librerías del sistema, y esto permitirá a Olly decodificar todas las llamadas al sistema con mucha más precisón.

Si no los tenemos, Olly sólo puede meterle mano a un conjunto predefinido de llamadas al sistema, las más típicas.

Podemos bajar los debug symbols, en mi caso para *WinXP SP3*, desde este link. Luego tenemos que decirle a Olly el path del directorio donde los hemos guardad, en principio *c:\windows\system32*



INTRODUCCIÓN A LA INFORMACIÓN MOSTRADA POR OLLY

Ejecutamos OllyDbg y desde él abrimos nuestro pequeño *hola mundo*. Olly nos presenta el programa desensamblado y presto para ser depurado:

El debugger ha parado el programa justo en su *entry point*. En este momento, todas las

DLLs han sido cargadas, los datos, código, etc están ya puestos en memoria, ... Es decir, el programa ha sido cargado y es el momento en el que el sistema operativo le va a ceder control.

Como veis, Olly ha sido capaz de reconocer las llamadas al sistema (MessageBox y ExitProcess), así como sus parámetros. Olly hace un análisis automático de todo el código del programa antes de presentárnoslo. No obstante, siempre tenemos la posibilidad de forzar dicho análisis pulsando *ctrl+a*.

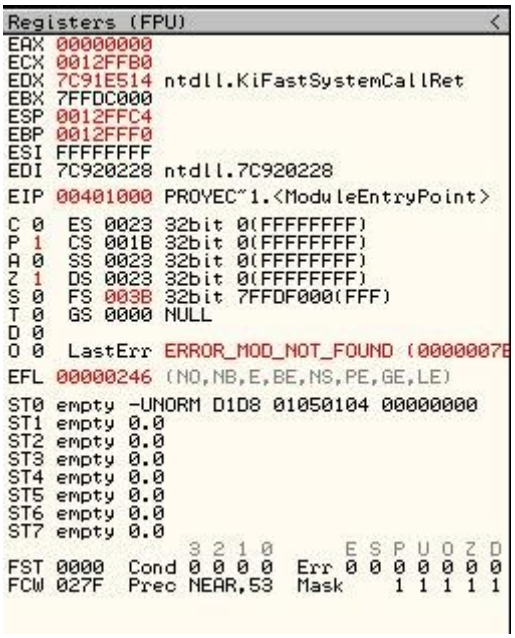
Podéis ver que los parámetros se envían al stack en orden inverso al que los tenemos definidos en msdn .Esto es porque sigue una de las posibles convenciones de llamada a funciones, *stdcall*, que es la más común en lenguajes como C y similares.

Si miráis la consola de Olly, veréis que nos está dando la información completa del estado de la CPU, la memoria, etc ... Démosle un repasillo.

A la derecha, tenemos todos los registros, incluídos los FPU, y flags(que corresponden al registro EFLAGS). Podemos hacer doble click sobre cualquiera de ellos y modificarlo a nuestro antojo.

Los registros, para que nos entendamos, son como nuestras variables "nativas" de la CPU. En asm por supuesto podemos definir las variables que nos de la gana, strings y de todo, pero los registros son de acceso mucho más rápido que la RAM, y por lo tanto son los que se usan mayoritariamente para leer/escribir datos, hacer operaciones, etc ...

EFLAGS contiene información sobre el status de la CPU. Por ejemplo, tiene un flag que nos indica si el resultado de la última operación ha sido cero. Si hacemos *mov eax, 0* dicho flag se pone a TRUE.

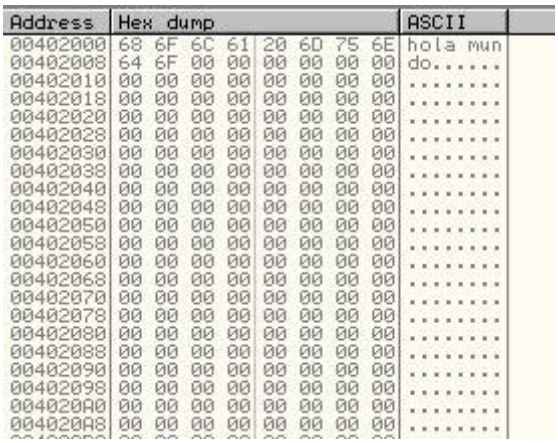


Debajo a la izda Olly nos ha hecho un dump de la *data section*. En ella tenemos definido nuestro string "Hola Mundo" y, si os fijáis, este string está referenciado en nuestro código en, por ejemplo, la posición *0x00401007*, donde se envía al stack un

puntero a dicho string:

00401007 |. 68 00204000 PUSH PROYEC~1.00402000 ; |Text = "hola mundo"

Esta es la data section:



Por supuesto, es posible, como con todo lo demás, sobre escribir directamente los datos en Olly, aunque estos datos son hechos en RAM, y no en el disco duro, por lo que al volver a ejecutar el programa todo permanecerá igual.

Finalmente, abajo a la derecha tenemos el stack. La CPU usa uno de los registros para apuntar a la posición actual del stack, *esp* (fijaos en que el valor de esp es *0x12FFF4*):

El stack no es ni más ni menos que un trozo de RAM que el sistema operativo reserva a nuestro programa, en principio, para pasar parámetros cuando se llama a las funciones. Para entender como funciona el stack vamos a depurar el programa y a ver qué es lo que pasa.

Primero, vamos a fijarnos en el contenido actual del stack:

0012FFC4 7C817077 ***RETURN to kernel32.7C817077***

Cuando el programa termine y haga un *ret* volverá a esta dirección (*ret* coge la dirección que hay en el stack, salta a ella y quita dicho elemento del stack). Si hacemos click en el stack y le damos a "follow in disassembler", Olly nos mostrará a qué está apuntando dicha dirección de memoria, 0x7C817077:

7C817077 . 50 PUSH EAX ; /ExitCode
7C817078 > E8 7B50FFFF CALL kernel32.ExitThread ; \ExitThread

Es decir, al terminar la ejecución de nuestro programa el sistema operativo recupera el control. Esto encaja con lo que comentábamos ayer de que el sistema operativo llama a "main" (realmente, crea un thread para ejecutar "main").

DEPURANDO CON OLLY

Olly nos permite ejecutar instrucción a instrucción en varios modos. Podemos ir paso a paso metiéndonos en cada una de las funciones llamadas o podemos ir paso a paso pasando por encima de ellas.

Como nosotros estaremos llamando a funciones del sistema operativo, y ya sabemos lo que hacen, vamos a elegir el segundo modo. Para ejecutar una instrucción le daremos a *F8*. Olly ejecutará dicha instrucción, mostrará todos los valores actualizados, tanto del stack como de los registros, etc, y parará el programa en espera de que ejecutemos la siguiente.

La primera instrucción es *push 0*, que envía un 0 al stack. Tras esto, podéis ver que el único registro que ha cambiado es esp, que apunta al último dato metido en el stack.

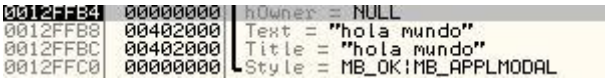
En este pantallazo podéis ver el cero metido en el stack y también (importante) que el valor de *esp* ha bajado en 4. Cuando enviamos un valor al stack el valor de esp baja 4 y cuando lo quitamos del stack, que se hace con la instrucción pop, aumenta en 4. Un poco contraintuitivo, pero es como funciona.

Insistimos ...

push eax; mete eax en el stack. Es decir, resta 4 al valor esp y copia el valor de eax en el stack
pop eax; lee eax del stack. Es decir, lee en eax lo que hay ahora mismo en el stack, que es donde apunta *esp*, y suma 4 al valor de esp

Si seguís depurando el programa, veréis cómo funciona el stack. Es sencillo, pero un tanto lioso al principio.

Además, justo antes de llamar a MessageBox veréis que aparecen todos los parámetros de modo "legible" en el stack:



De hecho, en cuanto Olly encuentra algo reconocible (un parámetro, un puntero a un string, ...) nos lo muestra allí para una mejor comprensión del programa.

Cuando llegamos a la llamada a MessageBox (aquí)

0040100E |. E8 F33F0000 CALL <jmp.&user32.messageboxa> ; \MessageBoxA</jmp.&user32.messageboxa>

Le damos a F8 y vemos el mensajito en pantalla:

Y pasamos a la siguiente instrucción:

00401013 |. 6A 00 PUSH 0 ; /ExitCode = 0

Finalmente, un par de F8 más y el programa termina con un exit code 0, que es el que hemos mandado al stack antes de llamar a *ExitProcess*.

Y con esto acabamos nuestra introducción a Olly y a la depuración de procesos.