.:[OllyDbg]:.

.· Análisis de sus posibilidades ·.
- II -

• El desensamblador

He aquí quizás la parte más importante, y con la que más tiempo vamos a trabajar mientras estemos depurando con OllyDbg. Una vez abrimos el módulo, la ventana principal y primera que abre el OllyDbg una vez analiza el código, es lo que él mismo llama “ventana de CPU”. Está dividida en varios fragmentos o subventanas, cada una de ellas con su función específica, y que en conjunto nos ofrecen el estado actual del proceso que queremos depurar. En el gráfico siguiente tienes una explicación de qué refleja cada parte de la ventana.

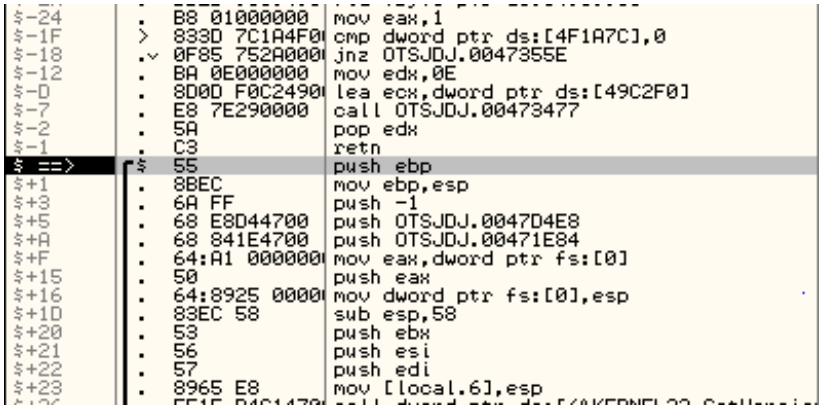
Según en qué ventana estemos, y sobre qué dato hagamos click con el botón derecho el menú contextual adoptará unas opciones u otras. Mi experiencia es que el OllyDbg a veces tiene tantas opciones distintas e intenta ser tan completo que se puede hacer un poco engorroso trabajar con él. Pero ahora veremos cómo es algo sencillo de comprender.

La ventana del desensamblador está dividida en 3 columnas: la de direcciones, el volcado en hexadecimal, el desensamblado propiamente dicho, y la columna de comentarios. En todo momento podemos hacer visible una pequeña barra descriptiva en la parte superior de las columnas seleccionando en el menú contextual: `Appearance|Show bar` (podremos ocultarla con `Appearance|Hide bar`).

No entraré en detalles que significa lo que muestra cada columna, pues considero que eso es la base del desensamblado, y no pertenece al contenido ni objetivo de este tutorial.

Tan solo comentar algunos aspectos generales de los menús contextuales: Cuando estamos sobre un volcado de datos (*dump*), habitualmente podemos hacer búsquedas binarias, copiar, editar, y visualizar en otros formatos. Cuando estamos sobre una dirección los comandos más comunes son

ver el volcado de esa dirección en la “ventana de dump”, ir a la dirección, buscar referencias a esa dirección. De igual modo cuando estamos sobre una columna de direcciones (veremos que en la ventana de la CPU la tienen las ventanas del desensamblador, de dump, y del stack) si hacemos doble click sobre una dirección concreta pasará de mostrar las direcciones de modo absoluto a modo relativo.



Aspecto de la columna de direcciones de la ventana del desensamblador en modo relativo.

Podríamos decir que en este sentido OllyDbg realmente muestra menús contextuales, ya que interpreta en cada momento qué queremos hacer, y las opciones de estos menús varían según qué y cómo podamos hacerlo. Citando el ejemplo anterior, es claro que sobre las direcciones habitualmente lo que queremos hacer es : ir a otra dirección, ver valores en relativo, y poco más. Si lo que queremos es actuar sobre el dump de la instrucción básicamente será para editarlo, o para buscar un valor concreto.

Veamos primero qué opciones tiene la configuración de la parte del desensamblador, para analizar después qué podemos hacer con ello.

• Configuración del desensamblador

Para configurar el modo en que ha de comportarse el desensamblador, seleccionamos: `Options|Debugging` options, y seleccionamos primero la pestaña `Dissasembler`.

Dissasembling sintax: Nos permite seleccionar qué sintaxis ha de usar a la hora de representar los modos de direccionamiento en algunas instrucciones. Aunque habitualmente se usa el de MASM, cuando nos acostumbramos a trabajar con otros compiladores puede ser útil seleccionar IDEAL o HLA[1].

Veamos un ejemplo de cada uno de los formatos:

MASM: `imul esp,dword ptr ds:[edi+ebp*2+20],6C20616C`
IDEAL: `imul esp,[dword ds:edi+ebp*2+20],6C20616C`
HLA: `intmul(6C20616C,[type dword ds:edi+ebp*2+20],esp);`

Disassemble in lowercase: Bien sencillo, los opcodes y los nombres de los registros aparecerán en minúsculas cuando tengamos esta opción marcada. Si está sin marcar aparecerán en mayúsculas (para gustos se hicieron los colores). Lo cierto es que es una opción necesaria para

1 Este formato está incluido en la última versión del OllyDbg (v. 1.09 step1).

poder depurar con comodidad.

Tab between mnemonics and arguments: habitualmente cuando escribimos códigos assembler tecleamos los argumentos del opcodes inmediatamente después del opcode, separándolo con un espacio, pero algunos editores de código fuente introducen ahí un tabulador, haciendo en ocasiones que nos acostumbremos a ver los argumentos separados del opcode por un tabulador. Con esta opción podemos ver el código desensamblado con un tabulador entre el opcode y los argumentos de este.

Extra space between arguments: Introduce un espacio después de la coma que separa los argumentos de un opcode que tenga más de un argumentos. Por ejemplo: `XOR EAX,EAX` lo mostraría como `XOR EAX, EAX`.

Show default segments: En windows, y en sistemas donde se utilizan estructuras de memoria FLAT no tiene mucho sentido usar explícitamente los registros de segmento. De todos modos existen y están ahí, y de hecho la CPU los usa. Algunos registros necesitan tener especificado qué registro de segmento deben usar, pero en otras ocasiones si no se especifica explícitamente usan unos por defecto (EBP usa SS, EAX,ABX,ACX,ADX usan DS, etc...). Si marcamos esta opción, al desensamblar nos mostrará siempre qué registro de segmento se usa en ese opcode.

Always show size of memory operands: Muestra siempre el tamaño del dato que se está moviendo, aunque sea obvio. Por ejemplo, `mov al,[eax]` es obvio que tiene que mover un byte. Si tenemos marcada esta opción, nos mostrará explícitamente que se mueve un byte, desensamblando la instrucción así: `mov al, byte ptr [eax]`.

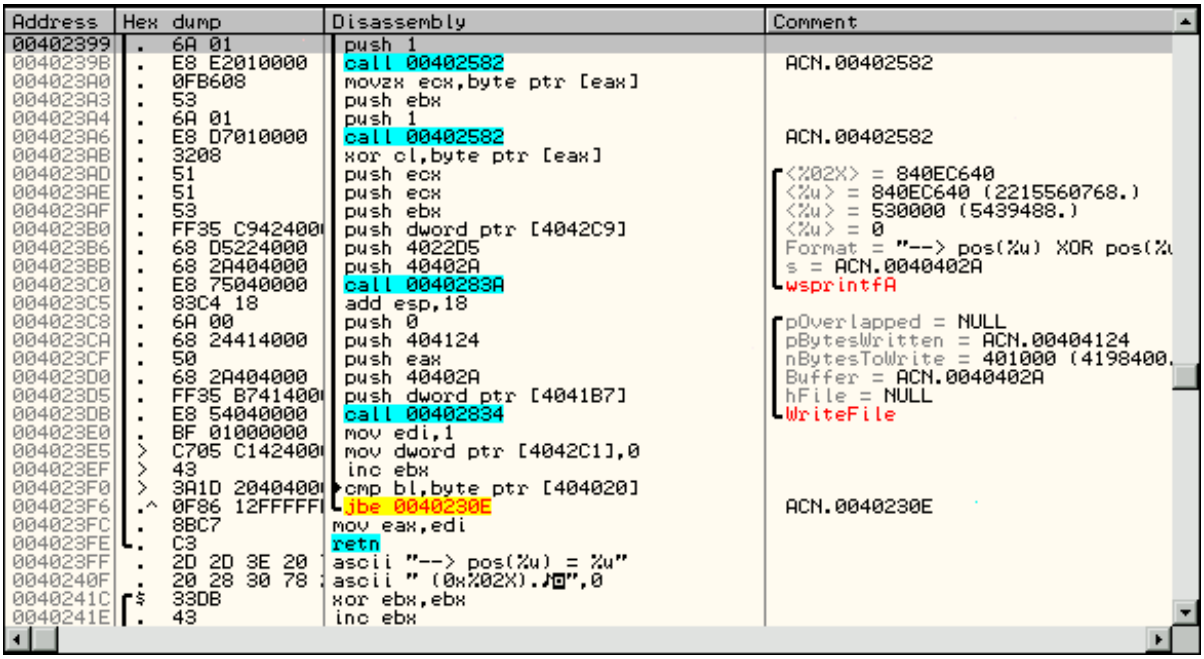
Show NEAR jump modifiers: Según la documentación del OllyDbg añade “NEAR” a la dirección de los saltos (`jmp`) indirectos y `call`'s que sean intramodulares. Mi experiencia es que aún teniéndolo marcado no he visto nunca ninguna variación :-? [2]

Show local module name: Cuando esta opción está marcada el desensamblador prefija cada dirección con el nombre del módulo. En ocasiones nos puede ayudar para saber si es una llamada a una función propia, o a una API externa.

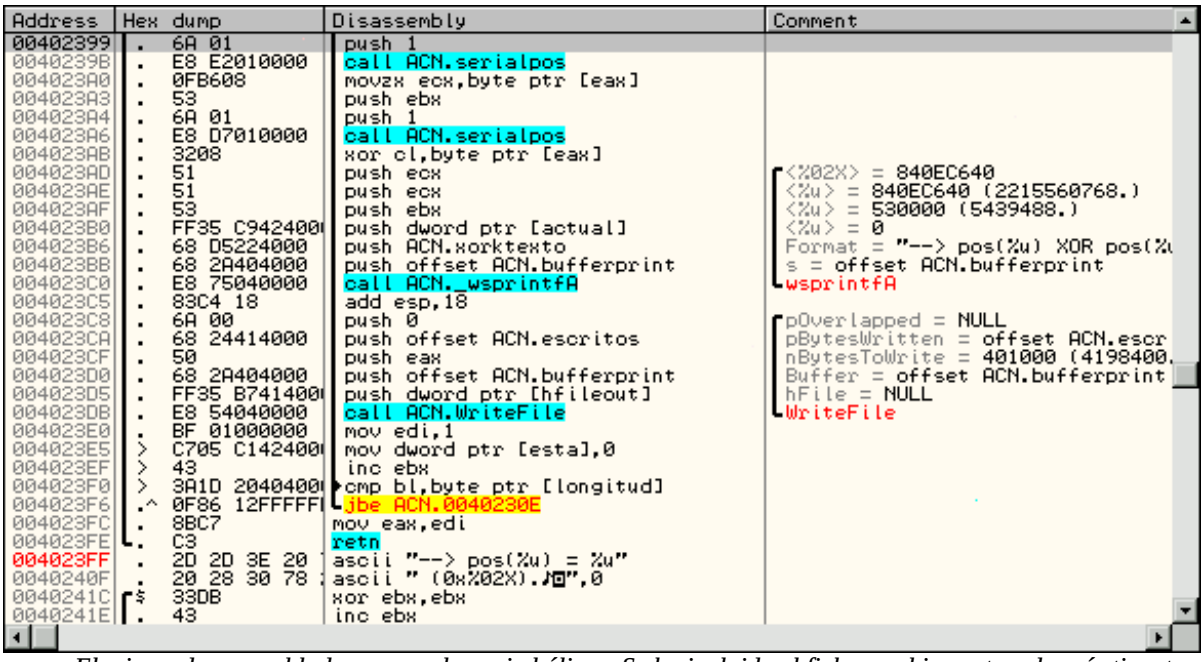
Show simbolic addresses: Si no está marcada las constantes y las direcciones son mostradas como números hexadecimales, si está marcada esta opción, y existe un nombre simbólico (etiqueta) para la dirección, se mostrará el nombre simbólico.

Atención aquí a todos los que usáis el debugger para depurar programas propios: si incluís los ficheros .obj en el analizador de código, **todas** las funciones y etiquetas que tengáis definidas en el código fuente aparecerán indicadas con su nombre simbólico.

2 Nótese, no obstante, que OllyDbg indica siempre cuándo un salto es de tipo corto o largo, indicando los saltos de tipoc corto con el indicador “short”. Un salto de tipo corto se codifica con 2 bytes, mientras que un salto de tipo largo necesita 5 bytes para codificarse.



Un desensamblado sin nombres simbólicos (etiquetas). El prefijo del nombre del módulo local se muestra en la zona de los comentarios.



El mismo desensamblado con nombres simbólicos. Se ha incluido el fichero .obj, mostrando así etiquetas propias del módulo como: serialpos, escritos, bufferprint, hfileout, xortexto, etc...

Veamos ahora otras opciones con las que podemos configurar el estado final de nuestros listados generados por el OllyDbg.

En `Options|Debugging options`, en la pestaña `Commands` podemos configurar cómo queremos que nos muestra unos opcodes determinados, especialmente los que se refieren al manejo de cadenas, los push globales, y el elemento por defecto en la pila del coprocesador matemático.

Use short form of string commands: Tanto `MOVS` como `STOS` usan siempre unos argumentos por defecto, que son `ESI` y `EDI`, como *source* y *destination* de la cadena a mover, o a escribir; por lo que podemos especificar que estos opcodes nos los muestre en forma abreviada, entendiéndose igualmente cual es su cometido. Para quienes no estén familiarizados con estas instrucciones es mejor no marcar esta opción, para ver los argumentos, pues puede ser más intuitivo.

Specify size of 16-bits SSE operands as: Sencillamanete es para indicar cómo queremos que OllyDbg interprete los argumentos de 128bits SSE.

Decode size-sensitive 16/32 bits mnemonics like: Similar a la opción anterior. Cuestión de estilo, y de sentirnos cómodos nosotros con el códigos desensamblado.

Decode top of stack as: Más de lo mismo. Podemos seleccionar que la parte alta de la pila del coprocesador nos la muestre como `ST` o como `ST(0)`. En el ensamblador ambas instrucciones son similares.

En la pestaña `CPU options` podemos indicar más opciones del desensamblador, que más que configurar cómo queremos que interprete o muestre determinadas instrucciones, lo que podremos será configurar cómo queremos que trate el código en tiempo real el propio OllyDbg. Recordemos que en la ventana del desensamblador podemos, además de ver cómo se actualiza mientras depuramos y hacemos un paso a paso, "navegar" a través de ella, moviéndonos con los cursores o con el ratón, viendo en la ventana de información adicional los valores analizados por el propio OllyDbg.

Veamos cómo podemos ayudarnos con el OllyDbg para prever el comportamiento del código que estamos desensamblando.

Syncronize source with CPU: actualiza la ventana del código fuente (*source window*) cada vez que se actualiza la ventana de CPU.

Underline fixups: Casi todas las librerías DLL y algunos ejecutables permiten ser cargados en direcciones de memoria diferentes de las establecidas cuando fueron compilados (en los exe habitualmente es 400000h). Para ello contienen una serie de tablas llamadas *relocs*, que actualizan ciertas constantes y valores en los opcodes al ser cargados por el *loader* de Windows.

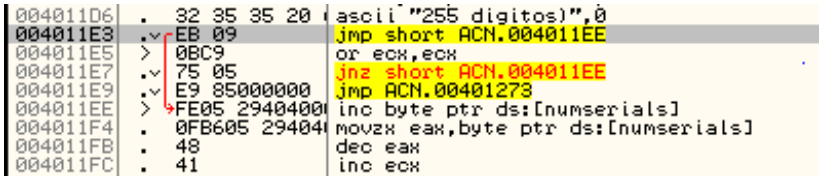
Cuando esta opción está marcada, OllyDbg subrayará los bytes que hayan sido modificados por las tablas de *relocs*. Atención a esta opción a todos los que trabajéis con volcados de memoria, pues os puede ahorar muchas horas de trabajo, y nos puede ayudar a la hora de modificar un fichero DLL, sabiendo así si también tenemos que actualizar la tabla de *relocs*. No es cometido de este pequeño artículo hacer un estudio de las tablas de *relocs*, pero os remito a mi tutorial acerca del análisis exhaustivo sobre el PeCompact, donde explico qué es y cómo se rellena una tabla de relocalizaciones[3].

Show direction of jumps: En la ventana del desensamblador justo antes de los bytes que representan la codificación de la instrucción, podemos reservar un pequeño espacio, para que

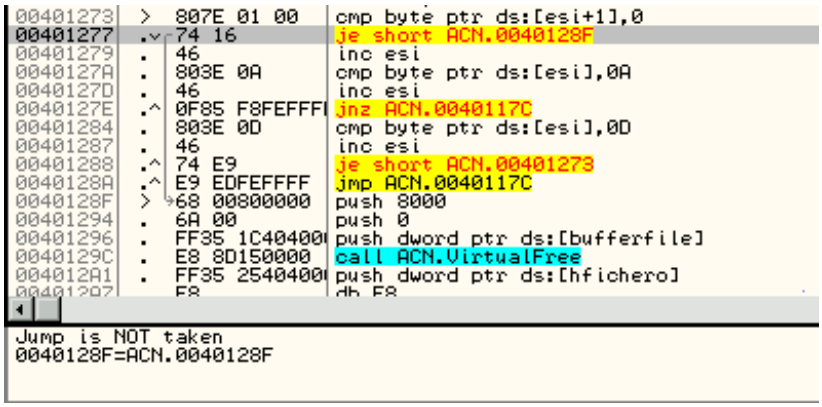
3 Todos los que alguna vez hayais trabajado en programas de 8 bits, recordaréis que había programas que eran realojables en diferentes partes de la memoria, no siendo necesario cargarlos en la direccion donde fueron compilados originalmente. Estos programas básicamente tenían una tabla de relocs, que actualizaban la primera vez que eran ejecutados, usando un sistema y una estructura muy similar al que usa el loader de Windows.

OllyDbg nos indique la dirección de un salto con una pequeña flecha hacia arriba o abajo. Si el salto es hacia una dirección que pertenece a otro módulo externo, nos lo indicará con un pequeño guión. Si copiamos al portapapeles[4] un fragmento de código sólo se copian las flechas hacia arriba (^).

Show jump path: Muestra una pequeña flecha a la izquierda del código, para ver gráficamente a donde salta la instrucción (sólo lo muestra para saltos de tipo `jmp`, condicionales y fijos). Mostrará una flecha de color rojo cuando el salto se va a producir. Si hemos seleccionado la opción *show grayed path is jump is not taken*, mostrará una flecha de color gris a donde iría el salto de haberse cumplido la condición para que se ejecutase. Atención: estas flechas se muestran también en el modo de navegación (cuando movemos el cursor sobre el código, sin estar depurando), y a la hora de decidir si se cumple o no una condición lo hará tomando los valores del registro de *flags* en ese momento. Recordar que en la ventana de información adicional nos indica si el salto se produce (*jump is taken*) o no (*jump is not taken*), independientemente de cómo tengamos marcada esta opción.



Un ejemplo de un salto incondicional marcado con la flecha que añade OllyDbg. Nótese también cómo se indican con pequeños indicadores las direcciones que tomarán los `jmp`, y con una flecha ">" las direcciones a donde hacen referencia los `jmp` (las direcciones referenciadas por `call` son interpretadas como puntos de entrada de funciones, y son marcadas con "$").



He aquí un ejemplo de una flecha que indica a dónde saltaría un jump condicional si se cumpliese. En la ventana de información adicional se indica que el salto no es realizado. Obviamente la siguiente instrucción que se ejecutará en este ejemplo es `inc esi`.

Center 'Follow'-ed command: Cuando en cualquier otra sección de la ventana principal de CPU seguimos alguna dirección se actualizará la ventana del desensamblador, centrando esa dirección. Si no tenemos esta opción activada la dirección seguida será mostrada en la parte superior de la ventana del desensamblador.

Gray commands that fill gaps between procedures: Normalmente los compiladores introducen código vacío después de una función, para que la siguiente empiece en una dirección

4 OllyDbg cuando copiamos un fragmento de código al portapapeles intenta mantener toda la ayuda que muestra en pantalla, tanto con los comentarios, como con las ayudas gráficas. No obstante en ocasiones no puede hacer la conversión a caracteres ANSI con la misma perfección que puede mostrarlo en pantalla gráfica.

"alienada". Al margen de explicar aquí qué sentido tiene esa alineación, con esta opción podemos pedir al OllyDbg que ese código nos lo muestre en color grisáceo.

Letter key in Disassembler starts: Cuando estamos en la ventana del desensamblador, con el cursor puesto sobre una instrucción y pulsamos una letra en el teclado, podemos estar queriendo hacer varias cosas: añadir un comentario, ensamblar una nueva instrucción ahí, o quizás añadir una etiqueta... En esta opción podemos definir qué es lo que solemos hacer cuando pulsamos una letra. Si hemos seleccionado, por ejemplo, *New label*, al colocarnos con el cursor sobre una instrucción y pulsar una letra, el desensamblador entiende que queremos añadir ahí una etiqueta, y nos lanzará el diálogo para que introduzcamos el nombre de la etiqueta.

• Otras opciones importantes

Existen otras dos pestañas en el panel de `Debugging options` que son importantes para la ventana de desensamblado, pues atañen al modo de representar y visualizar las cadenas de texto y las direcciones de memoria. Veamos las opciones de la pestaña `Strings`:

Decode Pascal-style string constants: Las cadenas (*strings*) creadas en Pascal van precedidas por un byte que indica su longitud, y por ello no necesitan ser finalizadas con un byte nulo (00h). Algunos lenguajes de alto nivel, como el Delphi, utilizan este mismo formato para las cadenas. Hay que advertir que esta opción cuando está activa incrementa las posibilidades de una mala interpretación.

Dump non-printable ASCII chars as dots: A la hora de mostrar en las ventanas de dump os contenidos de una cadena, podemos encontrarnos con caracteres ASCII que no son imprimibles. Con esta opción habilitada OllyDbg imprime esos caracteres no imprimibles como puntos. Hay que tener en cuenta que OllyDbg hace la discriminación entre imprimibles y *no-imprimibles* basándose también en la siguiente opción.

Allow diacritical symbols in strings: Se entiende que son caracteres ASCII standard los comprendidos en el rango 20h..7Eh, incluyendo TAB, CR, y LF. Sin embargo algunos alfabetos europeos hacen uso de caracteres especiales (aupa la ñ !!). Con esta opción marcada podemos decir a OllyDbg que interprete los caracteres por encima de 7Eh como caracteres especiales del alfabeto. Cuando esta opción está activada se incrementa también el número de cadenas erróneas reconocidas como tales.

Mode of string decode: Esta opción hace referencia a cómo son impresas las cadenas en la zona de comentarios de la ventana del desensamblador. En el modo *Plain* se muestra la cadena tal cual, sin hacer ninguna interpretación sobre ella. En el modo *Assembler* los caracteres que no son imprimibles se interpretan como valores hexadecimales o valores simbólicos (TAB, CR, LF). En el modo *C* se usan los constructores propios de este lenguaje \n , \r , \x01.

En la pestaña `Addresses` podemos definir cómo queremos que se interpreten las direcciones que suelen aparecer en la primera columna de la izquierda en las ventana del desensamblador y de volcados de memoria.

Demangle symbolic names: Algunos lenguajes orientados a objeto usan técnicas en las que añaden en el nombre de la función información del tipo de argumento que usan éstas. Con esta opción activada OllyDbg intenta interpretar la información que pueda estar contenida en el nombre

de la función, reconociendo los formatos propios de Borland y de Microsoft. Hay que advertir sin embargo que muchos nombres pueden ser (y suelen ser) ambiguos.

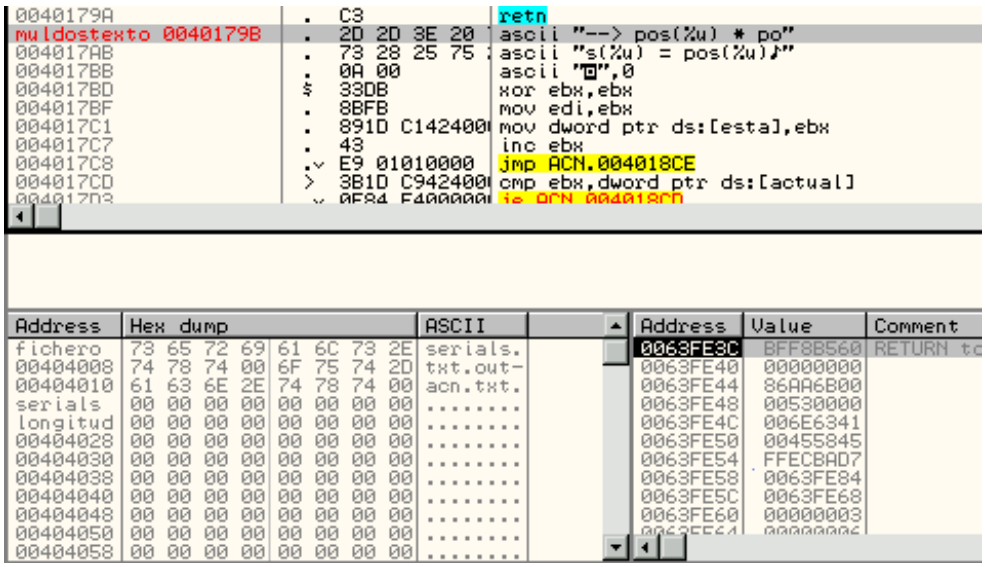
Prepend ordinals to IMPLIB names: Cuando OllyDbg encuentra la información de las funciones importadas por ordinales en las librerías IMPLIB, podemos indicarle que nos muestre el nombre simbólico, el número ordinal, o ambos simultáneamente.

Display adress in form of: Si una dirección no tiene nombre simbólico (etiqueta) establecido OllyDbg muestra su valor como un número hexadecimal de ocho dígitos. Pero si la dirección tiene un nombre simbólico aquí podemos definir cómo queremos que aparezca en los listados desensamblados.

Atención: esta opción deberíamos tener marcado que nos muestre (si existe) el nombre simbólico, pues nos ayudará mucho a la hora de depurar. Si ponemos, por ejemplo, en la dirección `402010` la etiqueta "n_ventanas", si hacemos un volcado de memoria, y en la columna de las direcciones (la primera por la izquierda) aparece `402010` OllyDbg automáticamente nos mostrará "n_ventanas", que obviamente será un buen recordatorio.

Las opciones disponibles son sencillas:

HEX, Symbol: Muestra el valor hexadecimal, y a continuación el nombre simbólico[5].
Symbol, HEX: Muestra la etiqueta, y a continuación el valor de la dirección.
Either HEX or symbol: Si existe etiqueta para esa dirección muestra solo la etiqueta, si no, muestra el valor hexadecimal de la dirección.



Tanto en la ventana del desensamblador, como en las ventanas de dump, cuando una dirección tiene nombre simbólico (etiqueta), podemos ver éste en lugar del valor hexadecimal de la dirección. Con la opción "highlight" nos marca en rojo las etiquetas de la ventana que tengamos activa en ese momento.

5 Debido a la anchura que por defecto establece OllyDbg para la columna de las direcciones, es posible que sólo se vea el valor hexadecimal de la dirección, teniendo que ampliar el ancho de la columna para poder ver el nombre simbólico.

Show name of local module: De este modo OllyDbg siempre añade el nombre del módulo a las direcciones simbólicas que son mostradas en la primera columna de las ventanas de desensamblado y en las ventanas de *dumps* (volcados)

Highlight symbolic names: Con esta opción se resaltan los nombres simbólicos (etiquetas) que están mostradas en la primera columna de la ventana del desensamblador y de las ventanas de *dumps*.

By default, sort contents of Names window by: En la ventana de nombres de funciones (*Names*), podemos ver no solo las funciones importadas y exportadas por el módulo, sino también las etiquetas de las funciones que nosotros vayamos añadiendo. Aquí podemos especificar cómo queremos que se ordenen: por dirección, o por nombre simbólico. Cuando ordenamos por nombre simbólico lo hace por el nombre de la función, no teniendo en cuenta a qué módulo pertenece la función (aunque el nombre del módulo aparezca indicado).

• La apariencia también importa

Un aspecto importante de OllyDbg es su apariencia. Quizás por ello no deberíamos pasar este apartado por encima, especialmente si las opciones que nos pueda ofrecer están encaminadas a ayudarnos a entender el código que estamos analizando.

Aunque quizás estamos acostumbrados a que un debugger tiene que ser una herramienta fuerte, robusta, pero a la vez ligera, y que por ello no debe tener especial cuidado en el aspecto gráfico. Pero también sabemos que bastante duro es tener que poner en marcha el debugger (alguien dijo duro? ;o) como para encima tener que rompernos la vista, o no poder seleccionar cómo queremos tener organizado todo. En este aspecto OllyDbg nos propone una serie de esquemas de color, de resaltado de código, y de fuentes, pero debemos saber que somos nosotros quienes podemos dar aspecto al debugger, para que sea como más cómodos nos encontremos.

Para ello tenemos un cuadro de diálogo destinado completamente a la apariencia de OllyDbg. Para acceder a él debemos seleccionar `Options|Appearance`.

De todas las pestañas que presenta el diálogo quizás las que más nos pueden ayudar a la hora de enfrentarnos con un módulo sean `Colour options` y `Code highlighting`.

En `Colour options` podemos indicar qué colores queremos para cada uno de los siguientes elementos:

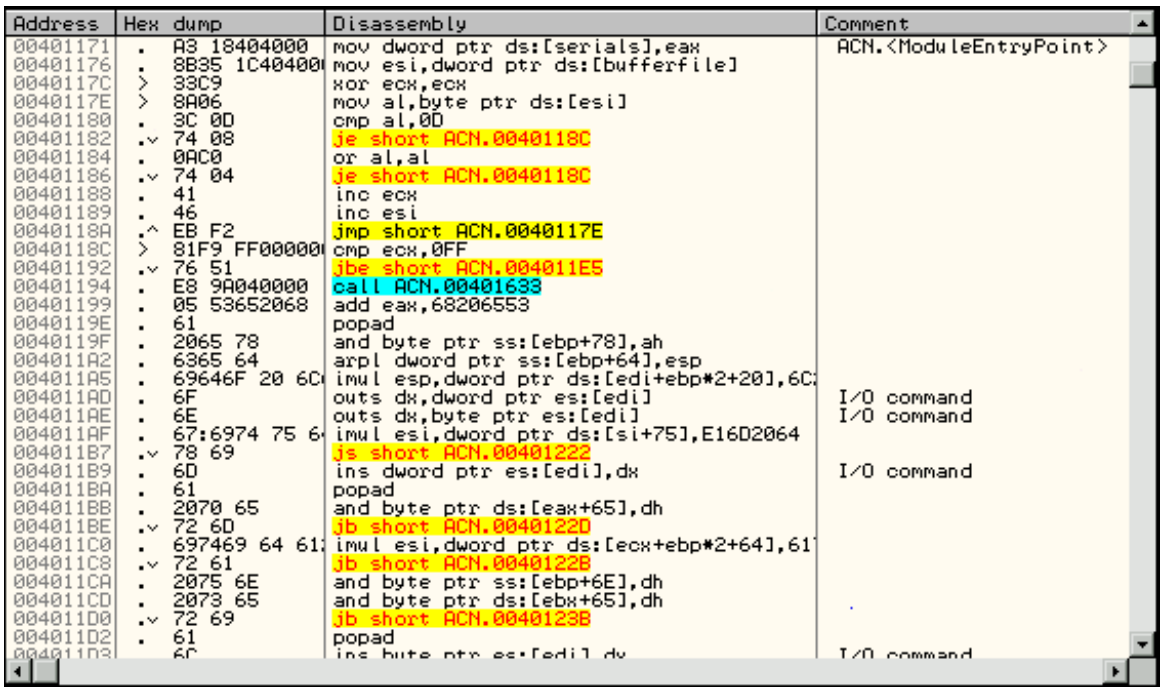
Plain text: se refiere al texto común, así como a las llaves que nos indican los funciones y los loops.
Highlighted text: se refiere al texto resaltado (funciones, etiquetas, breakpoints, flechas indicadoras de jmp).
Grayed text: texto ensombrecido (por ejemplo las instrucciones nulas que se insertan entre funciones para alinearlas en memoria, o las flechas de los saltos condicionales que no se cumplen).
Normal background: el fondo común de las ventanas.
Selected background: el color del fondo cuando seleccionamos un elemento.
Separating line: se refiere a las lineas verticales que separan las columnas de las que están compuestas cada ventana.

Auxiliary object: es el color de las lineas que inserta en las ventanas de volcado hexadecimal, para indicar las alineaciones (normalmente cada 4 bytes), también se usa para indicar las columnas que han sido minimizadas (un ancho nulo).

Conditional breakpoint: el color para indicar los breakpoints condicionales.

En lo que se refiere a la pestaña `code highlighting` debo decir que es una característica bastante interesante de OllyDbg. Si bien esta opción la poseen casi todos los editores de código fuente (¿quien no ha usado nunca el ultraedit?), en el caso que nos ocupa es especialmente importante, pues podemos pedir que se "pinte" el código desensamblado para resaltar ciertas instrucciones. Si bien OllyDbg incluye algunos esquemas predefinidos, podemos adaptar y crear los nuestros propios.

Básicamente indicar que podemos pedir que interprete cada instrucción como un todo, o que resalte la instrucción con un color y los argumentos (en función del tipo de argumento) con otro color distinto. Para poder activar el resaltado de código, debemos seleccionarlo en la ventana usando el menú contextual y seleccionando: `Appearance|Hightlighting` y el esquema de resaltado que queramos usar.



Una muestra de código resaltado con el esquema predefinido por OllyDbg "jmps'n'calls", donde se resaltan los opcodes `call` y `ret` de color azul, y los `jmp` de fondo amarillo y el texto en rojo o negro dependiendo si con condicionales o no.

Bien amigos, pues hasta aquí este artículo. En la próxima entrega nos meteremos ya de lleno con el debugger, viendo las posibilidades que nos brinda para poder depurar nuestras aplicaciones (bueno, vaaaale, y las que no son nuestras también ;o)