Capítulo 1

Preparando la mesa de trabajo

Bueno, acá empezamos.
La idea es armar una introducción a la programación de virus en ASM para Windows. Para desarrollar esto vamos a ir armando paso a paso un virus real para ir viendo las partes principales que lo componen y a la vez vamos a tocar temas fundamentales en la creación de estos especimenes como son la administración de memoria, el manejo estructurado de excepciones (SEH), formatos de los archivo portables ejecutables (PE), y un largo etc.

Si todo va bien, cuando terminemos vamos a tener desarrollado un virus funcional que luego servirá de punto de partida para que puedan realizar sus propias creaciones, de todas formas el tema principal es que se entienda cada parte del mismo en profundidad, no solamente la mecánica, sino para que se realiza y por que se hace de esa forma. Creo que lo esencial es comprender el funcionamiento interno de un virus, ya que si lo que queremos es obtener un código fuente lo podemos buscar en Internet y listo.

Tengo que aclarar algo, dentro de este curso vamos a ver cosas desde cero pero algunas no las vamos a poder tratar con demasiada profundidad ya que si no se haría demasiado extenso. Pero voy a ir dejando referencias para que puedan profundizar cada tema visto.

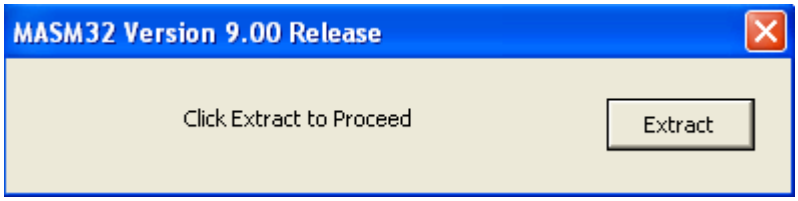
Bueno, vamos a la acción. Todas las herramientas que vamos a utilizar son de distribución gratuita.

Vamos a trabajar con algunas herramientas básicas las cuales les voy a contar como configurarlas para que puedan tener todo listo para que arranquemos, y más adelante veremos otras complementarias según las vayamos necesitando.

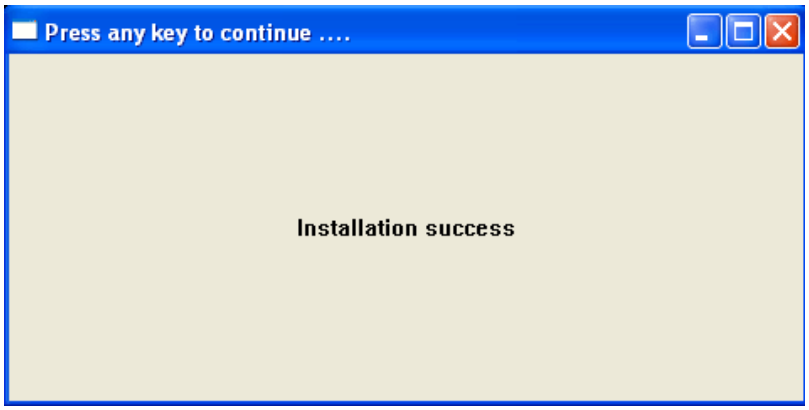
MASM

El assembler que vamos a utilizar es Microsoft Macro Assembler (MASM), el cual es un assembler de alto nivel creado por Microsoft, el cual se puede conseguir en forma gratuita desde la dirección http://www.masm32.com/. La elección de este lenguaje es por que hay mucha información del mismo (como los excelentes tutes de Iczelion) y además se lleva muy bien con la otra herramienta que vamos a utilizar (RadASM). De todas formas todo lo que vamos a ver es muy fácil portarlo a otro assembler como TASM, NASM, etc.
La instalación es muy sencilla, bajamos el instalador de la página que les mencionaba mas arriba, y luego lo descomprimimos. Se extrae un único archivo llamado "install.exe", el cual ejecutamos y nos aparece la siguiente pantalla:

Seleccionamos el disco donde lo vamos a instalar y hacemos clic en "Start".
Luego nos pide que confirmemos la instalación y nos avisa que no debe ejecutarse en background o con baja prioridad ya que necesita mucho procesador para compilar las librerías necesarias. Le damos aceptar y nos sale un mensaje para que se proceda a instalar el MASM:



Le damos a 'Extract' y comienza a descomprimirse en el disco rígido.
Luego aparecerá una ventana DOS para que confirmemos la compilación de unas librerías que necesita. Nos va a pedir varias confirmaciones hasta que muestra que la instalación se realizó con éxito:



OllyDBG

Para quien no lo conoce, el OllyDBG es un excelente debugger para programas en Windows de 32 bits. Este programa trabaja en Ring 3, que para empezar con un poco de teoría les comento de que se trata.

Los procesadores X86 de Intel y compatibles tienen 4 niveles de privilegio:

Ring 0: mayor privilegio (nivel kernel), sobre este se ejecuta el sistema operativo
Ring 1
Ring 2
Ring 3: menor privilegio (nivel usuario)

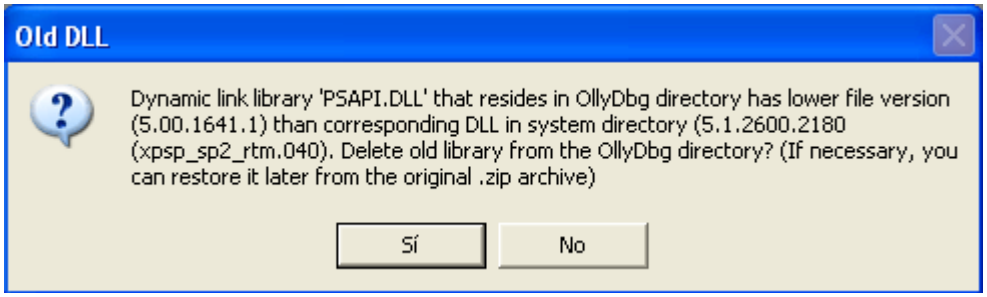
Win32 solamente soporta ring0 y ring3. En ring3 no tenemos acceso directo a los puertos, ni a zonas de memoria específicas, ni podemos controlar cualquier proceso, etc.

De todas formas nosotros nos vamos a basar en ring3, ya que es lo mas recomendable si queremos tener una mayor compatibilidad con los sistemas operativos. Los virus que logran correr sobre ring0 se basan en ciertos bugs de los SO (sistemas operativos), pero esto los limita para nuevas versiones del mismo o a parches de seguridad que corrijan dicho error.

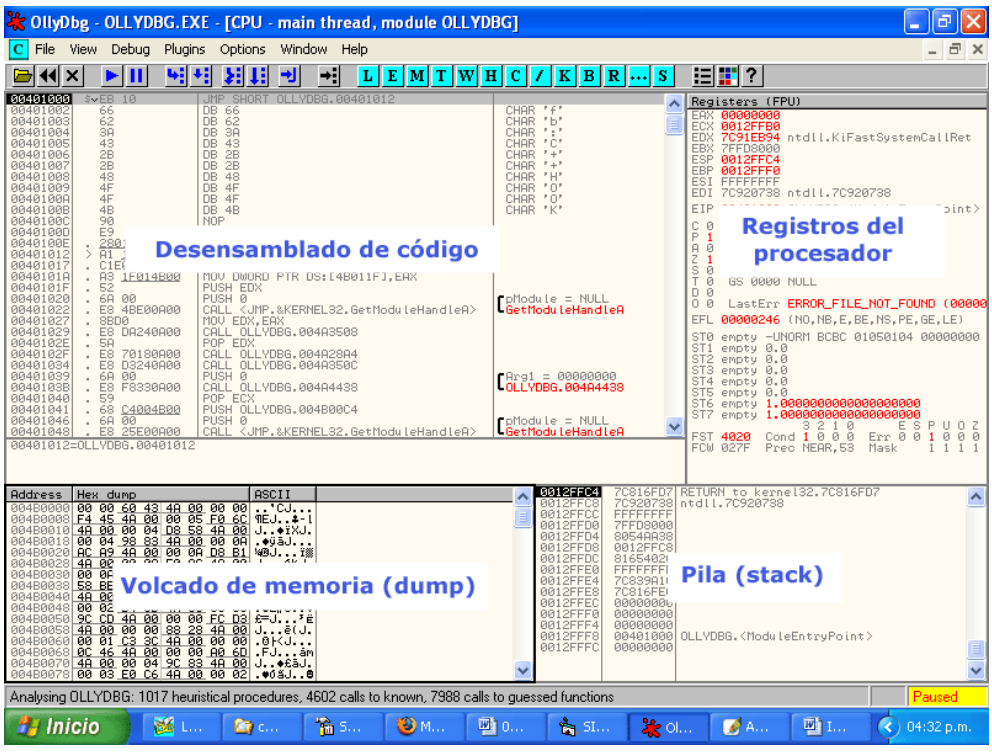
Veamos un poco mas de esto. Cuando nosotros estamos en ring3 y queremos acceder al disco rígido, llamamos a una API (que es una función que nos provee el SO, ya veremos mas detalladamente de que va esto). Esa API se ejecuta dentro del kernel, el cual entra en modo privilegiado, ejecuta el acceso al medio físico y luego vuelve al nivel usuario para devolverle el control al programa que lo solicitó. De esta forma el SO nos aisla de ring 0.

Bueno, volvamos al OllyDBG. Se puede bajar desde la dirección http://www.ollydbg.de/. Una vez descargado descomprímanlo en una carpeta, por defecto yo lo voy a poner en C:\OllyDBG.
Este programa tiene varios plugins, los cuales lo que hacen es agregarle funcionalidades al mismo para hacernos las cosas mas fáciles. Si buscan en Internet van a encontrar miles, de todas formas de momento lo vamos a dejar sin nada y luego vamos a ir agregándole algunos a medida que vayamos necesitándolos.

Para configurarlo entramos en el programa, el cual nos muestra un mensaje que nos indica que la DLL que esta en la carpeta de OLLYDBG es mas antigua que la de sistema, si apretamos 'No' va a dejar la del sistema. Elegimos esa opción y ya estamos en Olly.



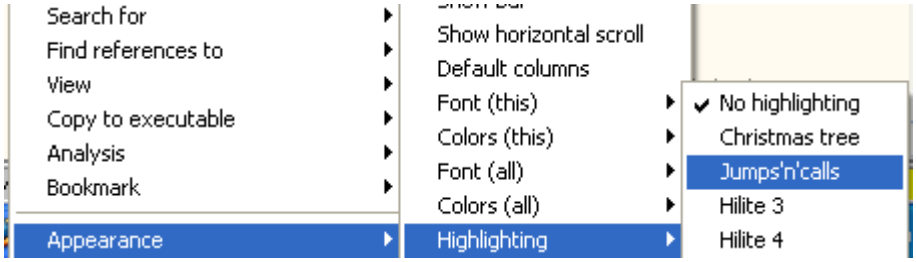
Para ver las distintas partes del programa abramos cualquier ejecutable, incluso puede ser el mismo Olly. Para esto vamos a File -> Open y seleccionamos el archivo a abrir. Nos mostrará una pantalla similar a la siguiente, donde veremos las siguientes partes:



Ya vamos a ir viendo cada cosa con más detalle a medida que avancemos.

Nota: para los que utilicen OllyDBG en un sistema de 64 bits, seguramente les va a dar varios problemas y van a quedar parados en varias excepciones y no les va a ejecutar el código. Para esto hay dos opciones, pueden bajar la versión 2 del OllyDBG de la misma página o pueden utilizar un plugin llamado Stealth64 de la siguiente dirección: http://tuts4you.com/download.php?view.2425. Se instala como un plugin cualquiera de OllyDBG (pueden ver el capítulo 3 para ver como se instala un plugin), y luego deben ingresar al OllyDBG, en la opción 'Plugins' -> ' Stealth64' -> 'Options' y marcar la opción 'x64 Compatibility mode'.

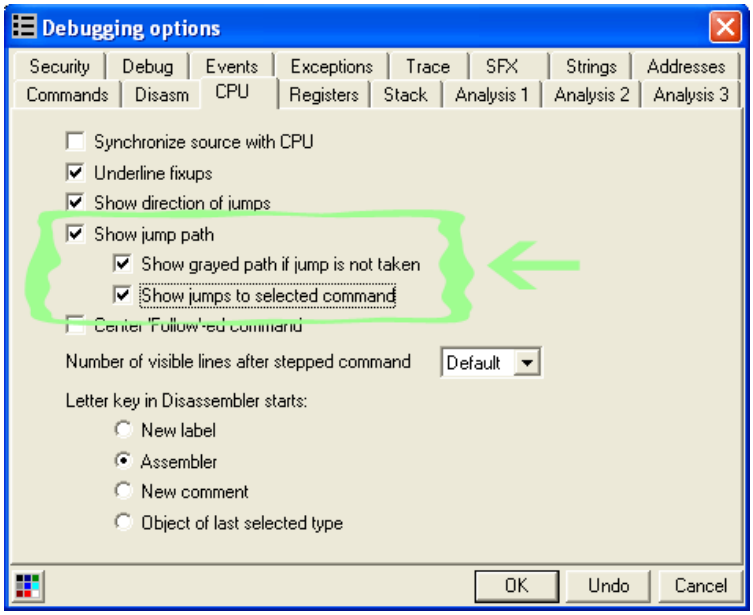
Continuamos, para darle mas claridad al código y así poder entenderlo mejor vamos a resaltar los saltos (jmp) y las llamadas a subrutinas (call). Para eso presionamos el clic derecho del mouse sobre la ventana de desensamblado y seleccionamos Appearance -> Highlighting -> Jumps´n´calls:



Ahora nos va a mostrar el código en forma mas clara, resaltando los saltos con distintos colores:

```
E9 AFC10A00   JMP OLLYDBG.004AD208
E9 26340A00   JMP OLLYDBG.004A4484
33C0          XOR EAX,EAX
A0 0D014B00   MOV AL,BYTE PTR DS:[4B010D]
C3            RETN
A1 23014B00   MOV EAX,DWORD PTR DS:[4B0123]
C3            RETN
60            PUSHAD
BB 0050B0BC   MOV EBX,BCB05000
53            PUSH EBX
68 AD0B0000   PUSH 0BAD
C3            RETN
B9 9C000000   MOV ECX,9C
0BC9          OR ECX,ECX
74 4D         JE SHORT OLLYDBG.004010CF
833D 1B014B00 CMP DWORD PTR DS:[4B011B],0
73 0A         JNB SHORT OLLYDBG.00401095
```

Los call y ret los pone en negro con fondo azul. Los saltos incondicionales (JMP) se ven en negro sobre amarillo y los saltos condicionales (JE, JB, JNB, JL, etc.) se ven en rojo sobre amarillo. Más adelante vamos a ver de qué se trata esto de los saltos y los call. Vamos a configurar algo más para ver más claramente los saltos. Para esto hay que ir a Options -> Debugging options, elegimos la solapa 'CPU' y tildamos 'Show jump path', 'Show grayed path if jump is not taken' y 'Show jumps to selected command'.



Ahora, cuando estemos parados sobre un salto nos va a mostrar con una flecha hacia donde está dirigido, y si el mismo es condicional lo va a mostrar en distintos colores según si va a ejecutar el salto o no.

Sobre la parte superior hay una barra de herramientas, las cuales vamos a ver rápidamente:

L: log data, muestra un detalle de lo que va haciendo el Olly (cuando arranca el programa, cuando genera un error, etc.)

E: executable modules, muestra todos los módulos que utiliza el programa debuggeado, el propio exe, las librerías que carga, etc.

M: memory map, como su nombre lo indica nos muestra un mapa de la memoria donde está nuestro programa, las dll que utiliza, etc.

T: threads, nos muestra los hilos de ejecución que utiliza nuestro proceso (esto lo vamos a ver luego con mas detalle)

W: windows, nos muestra las ventanas que tiene abiertas el programa

H: handles, son los manejadores que utiliza nuestro programa (también lo vamos a ver bien mas adelante)

C: cpu, la pantalla principal del Olly

/: patches, muestra los parches que se aplicaron al programa

K: call stack of main thread, muestra los distintos calls a los que vamos entrando

B: breakpoints, nos muestra los distintos breakpoints que hemos puesto en nuestro programa (lo que hacen es interrumpir la ejecución y darle el control al debugger)

R: references, nos muestra las referencias cuando realizamos alguna búsqueda

...: run trace, nos muestra el resultado de un traceo, esto es cuando elegimos la opción de que guarde todo lo que va ejecutando el Olly

S: source, muestra el código fuente, pero solo está disponible si el ejecutable contiene información de debuggin en formato Borland

Por ahora no vamos a ver nada mas de Olly, si más adelante a medida que lo vayamos necesitando.

RadASM

El RadASM es un IDE (entorno integrado de desarrollo) que facilita la programación ya que integra varias herramientas que nos van a ayudar a la hora de programar (compilador, debugger, ayudas, etc.).
Este IDE sirve para varios lenguajes pero lo vamos a instalar y adaptar a MASM.

El sitio para descargarlo es: http://www.radasm.com/.
Desde acá vamos a descargar varios archivos que nos van a servir para la instalación y la configuración del RadASM:

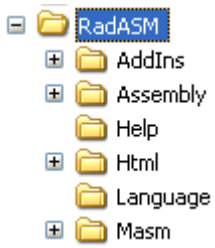
. RadASM IDE pack
. RadASM Assembly programming
. RadASM language pack
. Win32 help file

Lo primero que hay que hacer es descomprimir el archivo RadASM.zip dentro del disco C: en una carpeta que se llame RadASM (si bien se puede instalar en cualquier lugar -al igual que MASM y el OllyDBG- yo voy a mostrar las opciones por defecto).

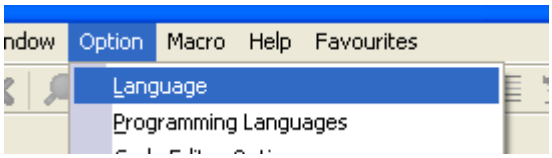
Una vez que tenemos los archivos descomprimidos vamos a instalar el paquete de idiomas para poder ponerlo en español. Para esto debemos descomprimir el archivo RadLNG.zip dentro de la carpeta C:\RadASM de tal forma que queden los archivos en C:\RadASM\Language.

Luego descomprimimos el archivo Assembly.zip dentro de C:\RadASM\ Assembly. Como lo que a nosotros nos interesa es MASM lo que vamos a hacer es pasar el archivo masm.ini y toda la carpeta Masm al directorio C:\RadASM.
Por último extraemos el archivo win32.zip en el directorio C:\RadASM.

Después de todo esto debe quedar algo así:

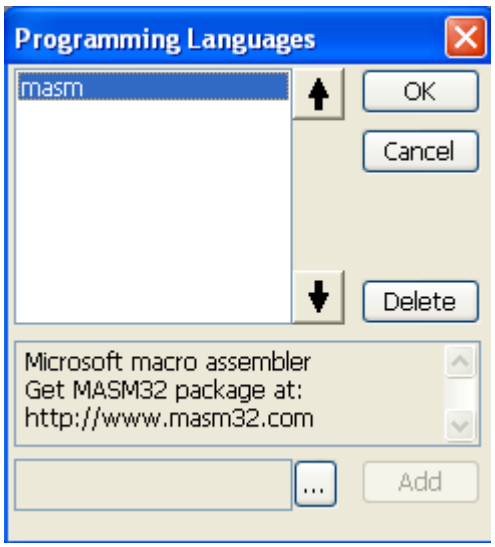


Ahora vamos a configurar el IDE. Ingresamos al programa RadASM.exe y vamos a la opción Option -> Language:



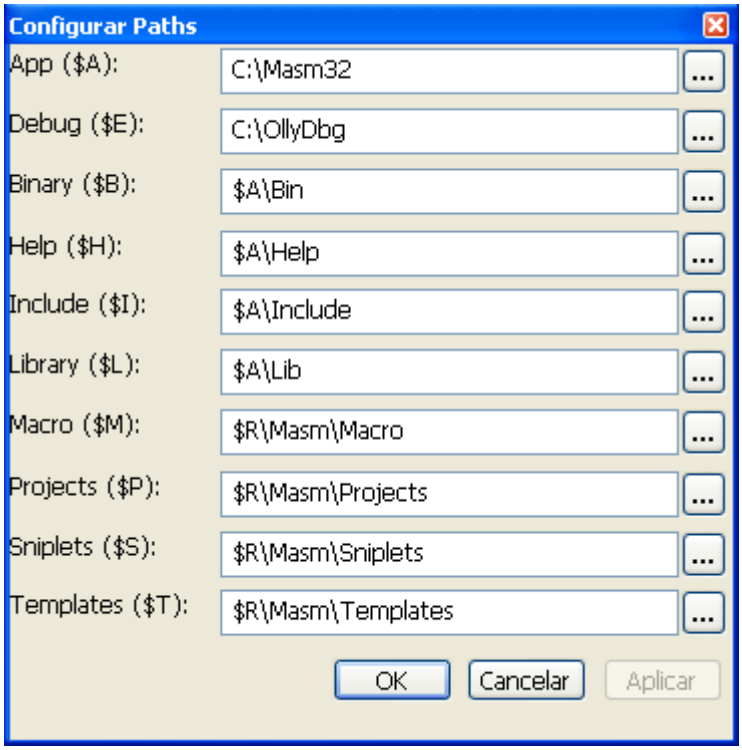
Elegimos la opción "Español" y le damos OK. Ahora veremos todas las opciones en español. Mucho mejor.
Ahora vamos a configurar el MASM. Para esto vamos a Opciones -> Programming Languages.

Por defecto nos muestra html, presionamos el botón Delete y lo eliminamos. Ahora hacemos clic sobre el botón que tiene los tres puntitos y nos aparece un cuadro de diálogo donde nos pide el lenguaje a configurar. Seleccionamos masm.ini y le damos Abrir.
Por último presionamos el botón Add y nos va a quedar algo así:



Le damos OK y ya tenemos configurado el MASM dentro de RadASM.
Luego de esto cerramos y volvemos a abrir el RadASM para que nos tome los cambios.

Ahora tenemos que decirle al IDE donde encontrar los programas que vamos a utilizar. Para esto vamos a Opciones -> Fijar Rutas. Si instalamos los programas en las rutas por defecto no vamos a tener que modificar nada, sino le decimos donde encontrar cada programa.



Por ahora esto es todo, luego utilizaremos otras herramientas como editores PE, editores hexadecimales y demás, pero las vamos a ir viendo a medida que las vayamos necesitando. Estas que vimos son las fundamentales y sería bueno para el que no tiene experiencia en las mismas que practique mucho para poder sacarles provecho.

Referencias

Acá voy a ir poniendo los documentos o links que me sirvieron en cada capítulo o que están relacionados con los temas tratados en el mismo para que puedan consultarlos e ir ampliando lo visto.

. Instalación del MASM32 – Configuración del RadASM como IDE de YuLSoft

. Introducción al cracking con OllyDBG desde cero de Ricardo Narvaja (parte 1)

. Programación con MASM+RADASM de Alan Moreno y RedH@wk (capítulo 1)

. Grupo MASM32-RadASM (http://groups.google.com.ar/group/MASM32-RadASM)

Bueno, eso es todo para esta primera entrega, nos vemos en la próxima.

:: zeroPad ::
Abril de 2007