Registros del Microprocesador

1.- Tipos de Registros

Al usar ASM nos valemos directamente de los registro de nuestro microprocesador a diferencia de los lenguajes de alto nivel.

1.1.-Registros de uso General :

EAX (Acumulador) : Utilizado para operaciones aritmetica (suma,resta,multiplicacion,divisi?n) , la mayoria de funciones despues de ser utilizadas devuelven un valor a EAX .

EBX (Base) : Se utiliza para direccionar el acceso a datos , situados en la memoria ; tambien se puede utiliza para operaciones arim?ticas .

ECX (Contador) : Se utiliza como contador por algunas instrucciones ; tambien se puede utiliza para operaciones arim?ticas .

EDX (Datos) : Algunas operaciones de entrada/salida requieren su uso ; las operaciones de multiplicaci?n y divisi?n con cifras grandes suponen al EDX y EAX trabajando juntos .

1.2.-Registros de Indice :

ESIy **EDI** : Son requeridos por algunas operaciones con cadenas de caracteres .

1.3.-Registros de banderas :

Se usan para registrar la informaci?n de estado y de control de las operaciones del microprocesador, y son 9 :
CF, OF, ZF, SF, PF, AF, DF, IF, TF.

1.4.-Otros Registros Importantes :

EIP : Este registro contendr? la direcci?n de la siguiente instrucci?n que se ejecutar?.

EBP : Apunta a la Base de la pila.

ESP : Apunta a la parte Superior de la pila.

2.- Tama?os de los Registros

<table>
<tr><td colspan="3">EAX (4 bytes = 32 bits)</td></tr>
<tr><td rowspan="2">Sin nombre</td><td colspan="2">AX (2 bytes = 16 bits)</td></tr>
<tr><td>AH (1byte)</td><td>AL (1 byte)</td></tr>
</table>

Lo que representa la imagen expuesta no solo es aplicable al registro EAX sino que puede ser cambiado por los demas registros de uso general (EAX, EBX, ECX, EDX)

Ejemplo :

12345678 => EAX
1234**5678** => AX
1234**56**78 => AH
12345**78** => AL

Referencias :

DWORD -> EAX
WORD -> AX
BYTE -> AH y AL

La Pila

Seg?n la Wikipedia :
- T?cnicamente es una estructura de datos, del tipo LIFO (del ingl?s Last In First Out, ?ltimo en entrar, primero en salir)
Enlace : http://es.wikipedia.org/wiki/LIFO

?Que esto de una estructura?.Bien, imaginad que formamos una ventana, o por ejemplo, el t?pico mensaje de texto. En ?ste caso es la API MessageBoxA .

Y claro, aqu? es donde entra el papel de la pila, antes de invocar a la API MessageBoxA tenemos que indicarle todos estos datos para que nos la forme tal y como nosotros le hemos indicado.

PUSH **T?tulo**
PUSH **Cuerpo**
PUSH **TipodeBotones**
CALL **MessageBoxA**

MessageBoxA es la API encargada de lanzar ese mensaje tipo "caja de texto" que puse en la imagen.

En Ensamblador o en C esto se visualizaria as? :

Llamada a MessageBoxA en C:
MessageBoxA (0, "Hola mundo", "xD", 0);

Pila:

Código equivalente en ASM:

```
push 0
push Puntero_xD
push Puntero_HolaMundo
push 0
call[MessageBoxA]
```

Practicando el Lenguaje Ensamblador

1.- Instrucciones B?sicas del Microprocesador :

Comenzaremos por las instrucciones m?s b?sicas, con breves explicaciones y ejemplos:

MOV destino, origen :
Sirve para mover informaci?n del origen al destino. Por ejemplo de un registro a otro, de un registro a una variable, etc. Aunque no nos sirve para mover datos entre dos variables.

Ej: **MOV A, B** => El valor de A pasar?a a ser igual que el valor de B.

ADD/SUB Destino, Cantidad
ADD : Suma una determinada cantidad a un registro o variable (destino).

Ej: ADD A, B => Se le suma a A el valor de B y el resultado se guardar?a en A.

SUB : Resta una determinada cantidad a un registro o variable (destino).

Ej: **SUB A, B** => Se le resta a A el valor de B y el resultado se guardar?a en A.

- Recuerda : No est? permitido sumar o restar datos directamente entre variables.

INC/DEC Variable o registro
INC : Incrementa el valor de una variable o registro en 1.

Ej: **INC A** => Se incrementar?a el valor de A en 1.

DEC : Decrementa el valor de una variable o registro en 1.

Ej: **DEC A** => Se decrementar?a el valor de A en 1.

CMP :
Compara el valor de dos registros , en realidad es una instrucci?n SUB que no guarda el resultado en ningun lado sino que su funcion es cambiar los flags (Registros de banderas) , esta instrucci?n trabaja cuando a los saltos al igual que TEST.

Asumiendo A=B
Ej: **CMP A, B** => Se efect?a una comparaci?n entre A y B (la diferencia de A y B es cero por lo tanto el flag Z se activa)

TEST :
Compara el valor de dos registros , a diferencia de CMP este en realidad es una instrucci?n AND que no guarda el resultado en ningun lado sino que su funcion es cambiar los flags (Registros de banderas)

Ej: **TEST A, B** => Se efect?a una comparaci?n entre A y B

PUSH DWORD
PUSH mete un DWORD en la pila. La pila se utiliza por ejemplo para pasar par?metros a las funciones o para utilizar variables locales. Es importante que cada PUSH tenga un POP (ahora lo explico).

Ej: **push eax**

POP DWORD
POP al contrario que PUSH saca un DWORD de la pila.

Ej: push eax => Guarda a eax en la pila
pop ebx => Saca el ultimo valor metido en la pila (en este caso EAX) y lo guarda en EBX (en este caso)

RET
RET se usa generalmente para regresar de una funci?n. Lo que hace realmente es ?popear? un DWORD de la pila y saltar a ?l. De ah? que en las funciones se meta la direcci?n de retorno en la pila y con el RET contin?e la ejecuci?n del programa donde corresponde.
A los interesados en el Stack Overflow esto les puede interesar bastante

2.- Instrucciones L?gicas :
Este tipo de operaciones se realizan a nivel de bits (binario) , los resultados son reemplazados en el operando Destino, si deseas realizar los ejemplos manualmente utiliza la calculadora de Windows en modo cientifica para convertir las cantidades a binario.

AND Destino, Fuente

TABLA DE VERDAD AND		
A	B	Resultado
0	0	0
0	1	0
1	0	0
1	1	1

Ej: 1110 (14 en decimal o E hexadecimal).
1101 (13 en decimal o D hexadecimal).
1100 (12 en decimal o C hexadecimal).

OR Destino, Fuente

TABLA DE VERDAD OR		
A	B	Resultado
0	0	0
0	1	1
1	0	1
1	1	1

Ej: 1110 (14 en decimal o E hexadecimal).
1101 (13 en decimal o D hexadecimal).
1111 (15 en decimal o F hexadecimal).

XOR Destino, Fuente

TABLA DE VERDAD XOR		
A	B	Resultado
0	0	0
0	1	1
1	0	1
1	1	0

Ej: 1110 (14 en decimal o E hexadecimal).
1101 (13 en decimal o D hexadecimal).
0011 (3 en decimal).

NOT Destino

TABLA DE VERDAD NOT	
A	Resultado
1	0
0	1

Ej: 1101 (13 en decimal o D hexadecimal).
0010 (2 en decimal).

3.- Saltos Condicionales :
Simulan lo que ser?a la estructura IF de un lenguaje de alto nivel .
La forma de todo salto es :

cmp *Valor1, Valor2*
Salto *Direcci?n*

Hay muchos tipos, as? que solo citar? los m?s importantes :

JMP - Salta siempre
JE ? Salta si los n?meros comparados son iguales.
JNE ? Salta si no son iguales.
JG ? Salta si Valor1 es mayor que Valor2.
JGE ? Salta si Valor1 es mayor o igual que Valor2.
JB ? Salta si Valor1 es menor que Valor2.
JBE ? Salta si Valor1 es menor o igual que Valor2.

4.- Apis :
Son funciones que pueden ser utilizadas en cualquier programa , para tener que evitar repetir lo mismo en todos los programas.
Aqui pondre las mas usadas en el mundo del cracking :

En construcci?n

Aprendiendo con OllyDbg

1.- Introducci?n :

Habiendo una gran cantidad de debuggers en la actualidad, ?Por qu? vamos a hacer la introducci?n con OllyDBG?
Creo que la mayoria de expertos en el cracking que han utilizado cualquier debugger reconocen que es mas sencillo empezar con OllyDBG, ya que muestra mayor informaci?n y es mas c?modo para aprender, la idea es ingresar al mundo del cracking, por la puerta del OllyDBG, mas adelante cuando uno ya conoce mas sobre el tema del cracking, puede trasladar f?cilmente a cualquier debugger lo aprendido pues cambian las formas de usar de los programas, pero no la esencia.

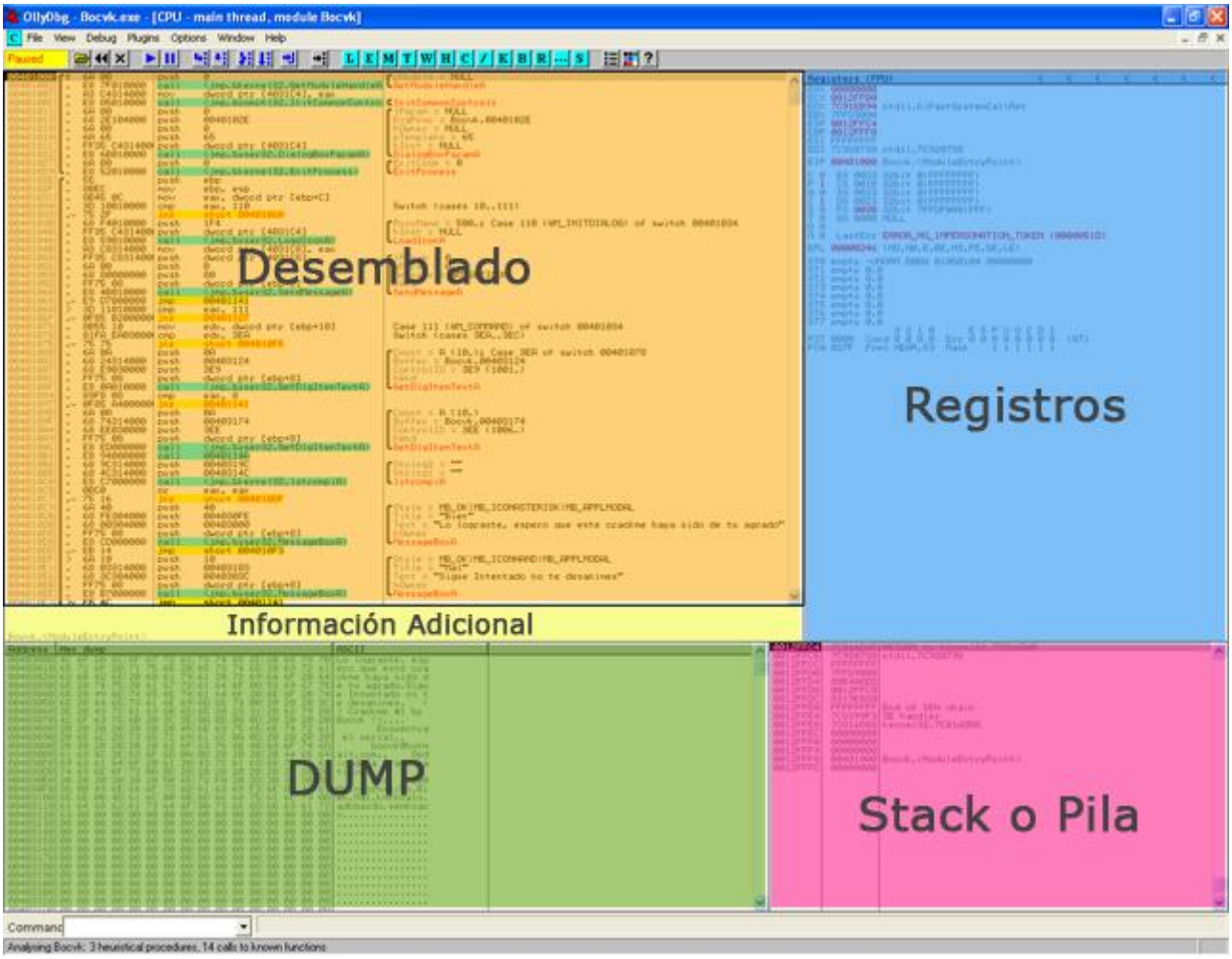
2.- Presentaci?n de OllyDBG :

OllyDBG es principalmente un debugger para aplicaciones a 32 bits en Microsoft Windows. Es un debugger en Ring3.
Tiene much?simas otras opciones e incluso es posible la utilizaci?n de plugins (o si eres programador hacerlos t? mismo) que hacen de ?l un programa verdaderamente potente.
Para conocer sus caracter?sticas podemos visitar su p?gina Web:
http://www.ollydbg.de/

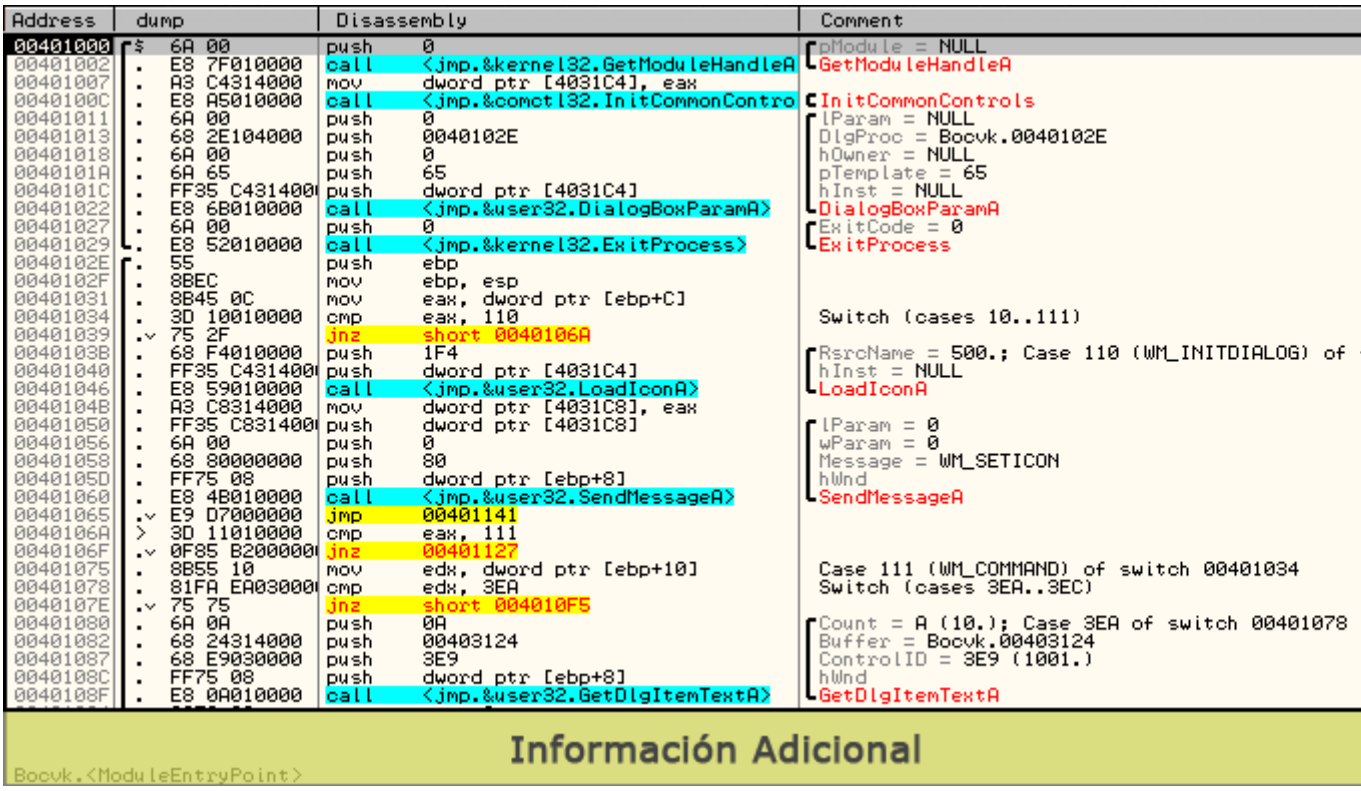
Para descargar el programa podemos descargar la versi?n 1.10 b?sica, sin ning?n plugin y ninguna modificaci?n desde su p?gina web, aqu?:

http://www.ollydbg.de/odbg110.zip

3.- Interfaz :

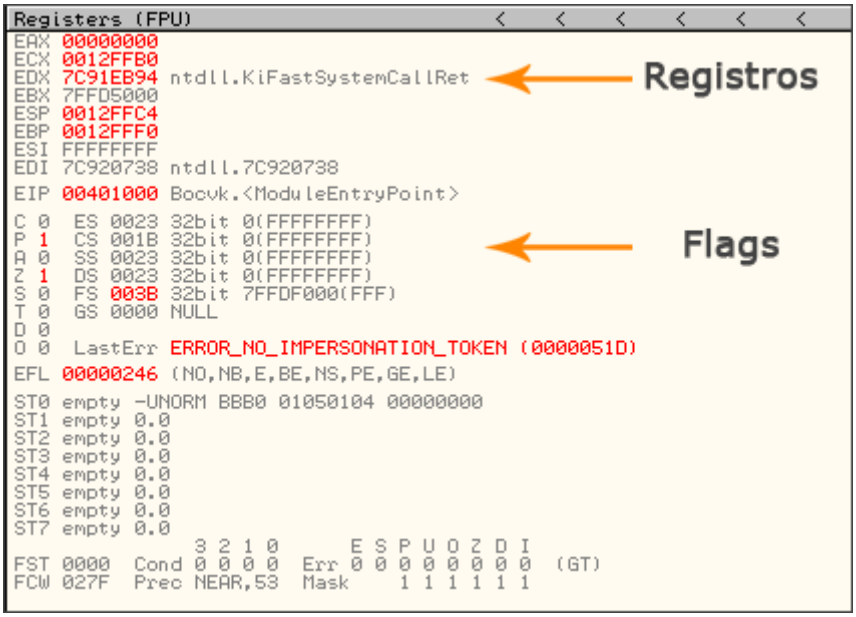


3.1.- Desamblado :

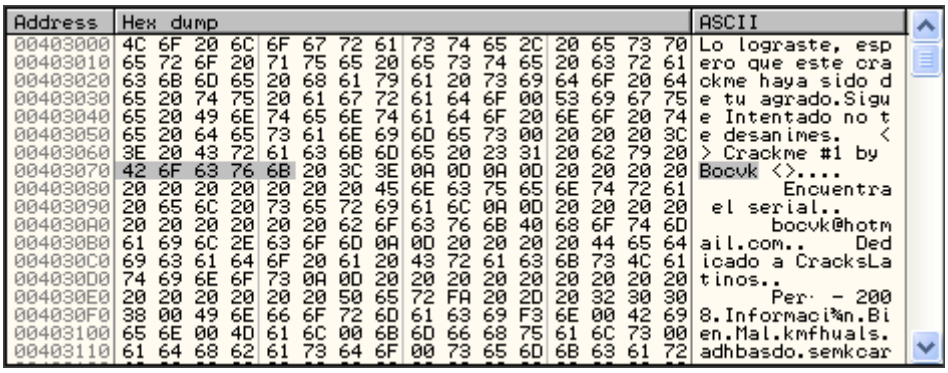


- La primera columna **"Address"** muestra la direcci?n donde se ejecutar? el c?digo.
- La segunda columna **"Hex dump"** muestra los bytes que forman la instrucci?n que hay a la derecha (columna "Disassembly).
- La tercera columna **"Disassembly"** indica el desensamblado propiamente dicho.
- La ?ltima columna **"Comment"** nos puede mostrar: comentarios propios, cadenas de texto, direcciones, nombres de funciones y direcciones, datos que vamos a insertar en la pila a cierta funci?n o API etc... que nos pueden ayudar much?simo.

3.2.- Registros :



3.3.- Dump :

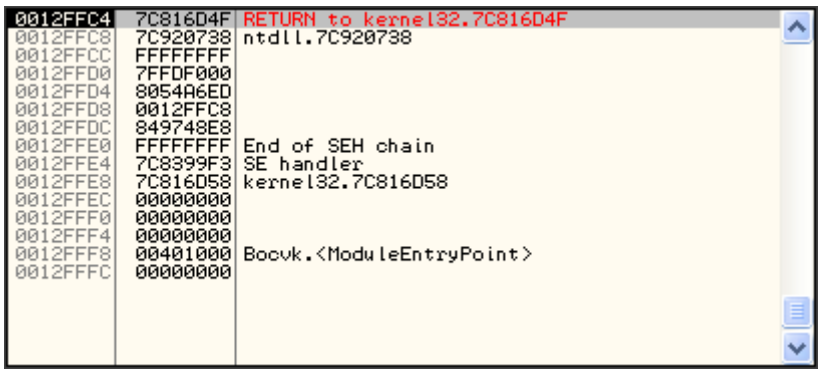


En esta subventana podemos representar los valores hexadecimales, de muchas formas diferentes:
- Podemos ver el car?cter ASCII que representa al valor hex.
- Podemos ver el c?digo UNICODE que representa a los valores hex.
- Podemos ver la representaci?n de punto flotante.
- Podemos ver las direcciones que pueden representar.
- Podemos ver incluso el c?digo desensamblado, aunque para esto ya tenemos nuestra ventana de desensamblado.
- Podemos ver un PE header con todos sus datos.

Y muchas opciones m?s. Una ventana muy ?til, que nos va a ayudar enormemente en nuestra labor.

Normalmente se suele utilizar la apariencia de la imagen anterior, que est? dividida en 3 columnas.
Address --> direcci?n donde se encuentran los bytes
Hex dump --> los valores hexadecimales
ASCII --> el car?cter ASCII del valor hexadecimal

3.4.- Pila :



4.- Lo mas usado en el OllyDbg:

4.1.- Conceptos :

Tracear : Ejecutar linea por linea de su codigo del programa ahi dos formas con F7 y F8

4.2.- Teclas :

F7 : Ejecuta una sola l?nea de c?digo (si estas en un CALL entra al mismo a ejecutarlo por dentro)

F8 : Ejecuta una sola l?nea de c?digo (si estas en un CALL no entra al mismo lo ejecuta completo sin entrar y sigue en la
siguiente l?nea luego del CALL)

Esas dos formas de tracear manualmente son verdaderamente diferentes y seg?n cada caso usaremos F7 o F8 .

F2 : Coloca un Breakpoint COMUN en la l?nea que marcas con el Mouse o esta grisada en el listado, para quitar el BP apretas
nuevamente F2.

F9 : Arrancar el programa (Run)

CTRL+G : Ir hacia un direccion determinada por nosotros

CTRL+N : Listado de Apis (funciones) utilizadas

En Construcci?n

Autores : karmany y Shaddy

Bibliografia :
Introducci?n al Cracking con OllyDBG Desde Cero por Ricardo Narvaja , Team CrackLatinos
Clases de MASM - RadASM por ^A|An M0r3N0^ y RedH@wk , RVLCN

Modificaciones : Bocvk

PD :
Con esto he terminado una parte de la Introducci?n al Cracking espero que les haya servido , pues me cansado de escribir nunca he escrito tutoriales (Papers), asi que espero que me ayuden a seguir con esto ,me gustaria saber quien esta interesado en escribir o apoyarme con temas de Cracking, y tambien quisiera saber quienes lo leyeron o mejor a quienes le importa porque no me gustaria escribir tanto si nadie le va a prestar la atenci?n ...
Por favor espero comentarios ,criticas y/o sugerencias ...