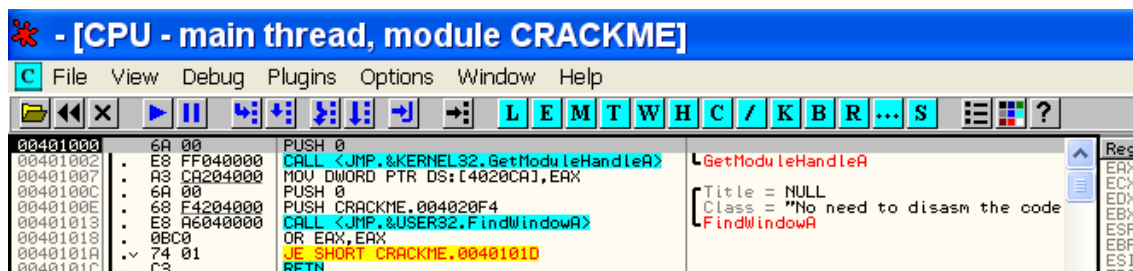


INTRODUCCION AL CRACKING CON OLLYDBG PARTE 25

EXCEPCIONES

Veremos en esta parte 25 el manejo de excepciones, que suele ser un problema para los newbies, pero en si no es un tema difícil si leemos un poquito sobre el.

Una excepción se produce en un programa cuando el procesador ejecuta una operación no valida, veamos algunos ejemplos en el mismo OLLYDBG, abramos el crackme de cruehead para escribir algunos ejemplos de excepción.

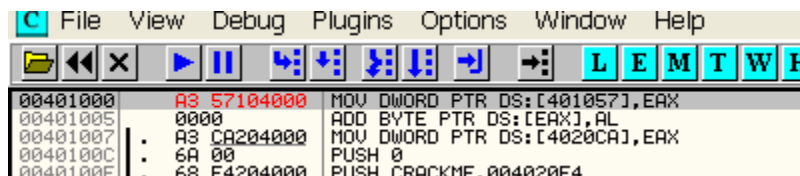


Allí tenemos el crackme de cruehead 1, parado en el Entry Point y escribiremos en la primera línea como hacíamos cuando veíamos la instrucciones de assembler, algunas instrucciones que al ejecutarlas nos producirán una excepción.

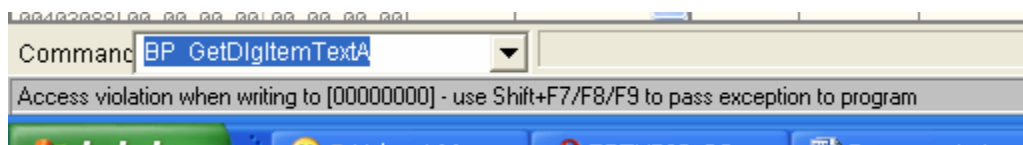
Usaremos algunas definiciones del tute de Mr Silver de excepciones y las graficaremos aquí con el ejemplo.

Acceso a memoria no válida: Se producen cuando un thread intenta acceder en un modo no permitido a una posición de memoria a la cual no tiene acceso. Por ejemplo se puede producir este tipo de excepción si el thread intenta escribir a una posición de memoria de solo lectura.

Escribo en el OLLYDBG la siguiente linea.



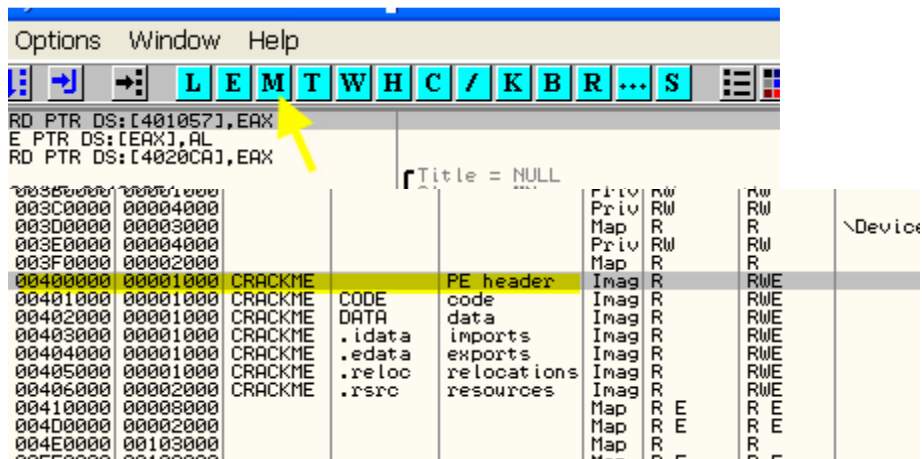
Esto lo vimos cuando explicamos la sentencia MOV, en este caso 401057 tiene solo permiso de lectura y ejecución, pero no de escritura, por lo tanto al escribir en dicha posición de memoria dará excepción, apretamos f8.



Las secciones tienen permisos iniciales, que están guardados en el header, hasta que el programa no los cambie mientras corre con alguna api como por ejemplo VirtualProtect que sirve para cambiar permisos en tiempo de ejecución, tendrá el permiso inicial.

Donde podemos ver los permisos iniciales de cada sección y modificarlos si queremos en el OLLYDBG?

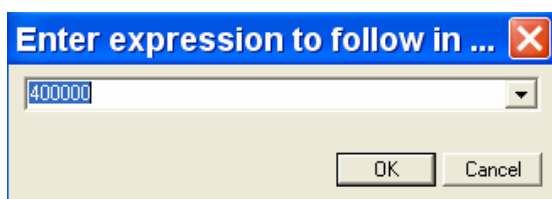
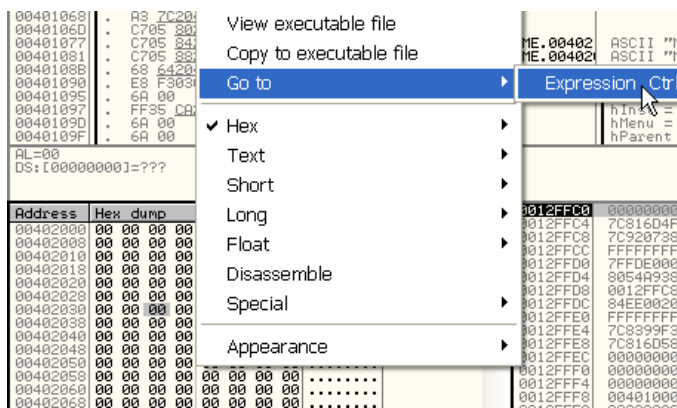
Vayamos a ver las secciones apretando el botón M



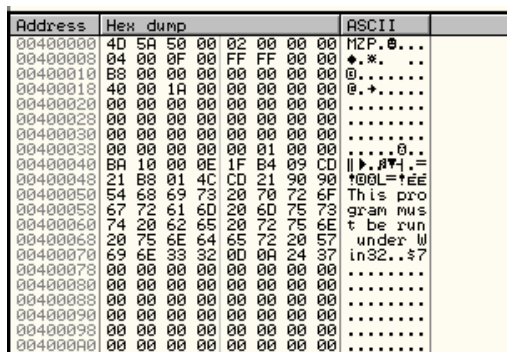
Vemos que la sección que empieza en 400000 o sea la primera realmente es el PE HEADER, es una pequeña sección de 1000 bytes que guarda los datos de las secciones, nombres, largo, todo sobre el archivo para arrancarlo.

Por supuesto el PEHEADER se puede editar con cuidado y siempre en una copia del archivo ya que si metemos mal la mano el programa no arrancará.

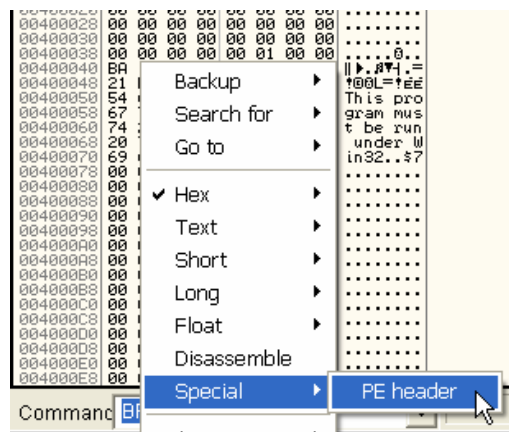
Veamos en el dump el header



Ponemos 400000 ya que el header, ya vimos que comienza allí.



Ahora el OLLYDBG tiene una opción para interpretar mejor los parámetros del header, haciendo en el dump, click derecho.



DS: [00000000]=???

Address	Hex dump	Data	Comment
00400000	4D 5A	ASCII "MZ"	DOS EXE Signature
00400002	5000	DW 0050	DOS_PartPag = 50 (80.)
00400004	0200	DW 0002	DOS_PageCnt = 2
00400006	0000	DW 0000	DOS_ReloCnt = 0
00400008	0400	DW 0004	DOS_HdrSize = 4
0040000A	0F00	DW 000F	DOS_MinMem = F (15.)
0040000C	FFFF	DW FFFF	DOS_MaxMem = FFFF (65535.)
0040000E	0000	DW 0000	DOS_ReloSS = 0
00400010	B800	DW 00B8	DOS_ExeSP = B8
00400012	0000	DW 0000	DOS_ChkSum = 0
00400014	0000	DW 0000	DOS_ExeIP = 0
00400016	0000	DW 0000	DOS_ReloCS = 0
00400018	4000	DW 0040	DOS_Tabloff = 40
0040001A	1A00	DW 001A	DOS_Overlay = 1A
0040001C	00	DB 00	
0040001D	00	DB 00	
0040001E	00	DB 00	
0040001F	00	DB 00	
00400020	0A	DB 0A	

Vemos que el OLLYDBG nos interpreta que es cada cosa

00400039	00	DB 00	
0040003A	00	DB 00	
0040003B	00	DB 00	
0040003C	00010000	DD 00000100	Offset to PE signature
00400040	BA	DB BA	
00400041	10	DB 10	
00400042	00	DB 00	
00400043	0E	DB 0E	

Si vamos bajando lo primero que hallamos es el offset a la PE SIGNATURE que es un valor que nos dice donde esta ubicada en el Header la información realmente, vemos que el puntero es 100, así que debemos sumarle los 400000 del inicio del header y nos dará 400100, bajemos hasta allí.

004000FE	00	DB 00	
004000FF	00	DB 00	
00400100	50 45 00 00	ASCII "PE"	PE signature (PE)
00400104	4C01	DW 014C	Machine = IMAGE_FILE_MACHINE_I386
00400106	0600	DW 0006	NumberOfSections = 6
00400108	2924D90A	DD 0A242929	TimeDateStamp = AD92429
0040010C	00000000	DD 00000000	PointerToSymbolTable = 0
00400110	00000000	DD 00000000	NumberOfSymbols = 0
00400114	E000	DW 00E0	SizeOfOptionalHeader = E0 (224.)
00400116	8E81	DW 818E	Characteristics = EXECUTABLE_IMAGE 32BIT_MF
00400118	0B01	DW 010B	MagicNumber = PE32
0040011A	02	DB 02	MajorLinkerVersion = 2
0040011B	19	DB 19	MinorLinkerVersion = 19 (25.)
0040011C	00060000	DD 00000600	SizeOfCode = 600 (1536.)
00400120	00220000	DD 00002200	SizeOfInitializedData = 2200 (8704.)
00400124	00000000	DD 00000000	SizeOfUninitializedData = 0
00400128	00100000	DD 00001000	AddressOfEntryPoint = 1000
0040012C	00100000	DD 00001000	BaseOfCode = 1000
00400130	00200000	DD 00002000	BaseOfData = 2000
00400134	00004000	DD 00400000	ImageBase = 400000
00400138	00100000	DD 00001000	SectionAlignment = 1000
0040013C	00020000	DD 00000200	FileAlignment = 200
00400140	0100	DW 0001	MajorOSVersion = 1

Como vemos no estábamos equivocados, aquí empieza la info importante sobre el programa.

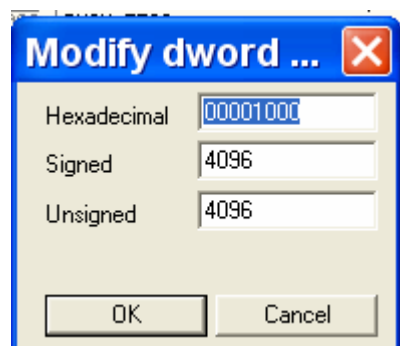
Ya explicaremos mas detalladamente muchos de estos valores solo mencionemos al pasar

0040011C	00060000	DD 00000600	SizeOfCode = 600 (1536.)
00400120	00220000	DD 00002200	SizeOfInitializedData = 2200 (8704.)
00400124	00000000	DD 00000000	SizeOfUninitializedData = 0
00400128	00100000	DD 00001000	AddressOfEntryPoint = 1000
0040012C	00100000	DD 00001000	BaseOfCode = 1000
00400130	00200000	DD 00002000	BaseOfData = 2000
00400134	00004000	DD 00400000	ImageBase = 400000

O sea ese puntero que es 1000 le sumamos los 400000 y tenemos el entry point del programa, si lo queremos cambiar, por ejemplo a 2000, hacemos click derecho

The screenshot shows a debugger window with assembly code. A right-click context menu is open over the assembly code, displaying various options. The 'Special' option is checked. The assembly code includes instructions like PUSH, CALL, MOV, and JMP. The menu options include Binary, Modify integer, Label, Breakpoint, Search for, Find references, View executable file, Copy to executable file, Go to, Hex, Text, Short, Long, Float, Disassemble, Special, and Appearance.

Y podemos escribir el valor que queramos por ejemplo si queremos que el programa empiece de 402000, escribiremos allí 2000 ya que siempre hay que restarle la imagebase 400000 o sea donde realmente empieza el header que es la primera sección del programa.



Luego para que quede definitivo guardado en el archivo, hacemos CLICK DERECHO-COPY TO EXECUTABLE y en la ventana que aparece CLICK DERECHO - SAVE FILE como guardamos los cambios normalmente, igual esto no lo haremos ahora solo quería mostrar otra posibilidad.

Bueno sigamos bajando en el header

004001F4	00000000	DD 00000000	Reserved
004001F8	43 4F 44 41	ASCII "CODE"	SECTION
00400200	00100000	DD 00001000	VirtualSize = 1000 (4096.)
00400204	00100000	DD 00001000	VirtualAddress = 1000
00400208	00060000	DD 00000600	SizeOfRawData = 600 (1536.)
0040020C	00060000	DD 00000600	PointerToRawData = 600
00400210	00000000	DD 00000000	PointerToRelocations = 0
00400214	00000000	DD 00000000	PointerToLineNumbers = 0
00400218	0000	DW 0000	NumberOfRelocations = 0
0040021A	0000	DW 0000	NumberOfLineNumbers = 0
0040021C	20000060	DD 60000020	Characteristics = CODE!EXECUTE!READ

Allí empiezan los datos de las secciones vemos que la sección que comenzara en la posición de memoria 1000 o sea 401000, y su características son CODE, EXECUTE, READ.

Si quisiéramos que esta sección tuviera permiso de escritura deberíamos cambiar ese 60000020 por E0000020 que es el valor que deja la sección con permiso para todo, jeje, probemos.

Modify dword ...

Hexadecimal

E0000020

Signed

-536870880

Unsigned

3758096416

OK

Cancel

004001F4	00000000	DD 00000000	Reserved
004001F8	43 4F 44 41	ASCII "CODE"	SECTION
00400200	00100000	DD 00001000	VirtualSize = 1000 (4096.)
00400204	00100000	DD 00001000	VirtualAddress = 1000
00400208	00060000	DD 00000600	SizeOfRawData = 600 (1536.)
0040020C	00060000	DD 00000600	PointerToRawData = 600
00400210	00000000	DD 00000000	PointerToRelocations = 0
00400214	00000000	DD 00000000	PointerToLineNumbers = 0
00400218	0000	DW 0000	NumberOfRelocations = 0
0040021A	0000	DW 0000	NumberOfLineNumbers = 0
0040021C	200000E0	DD E0000020	Characteristics = CODE!EXECUTE!READ!WRITE
00400220	44 41 54 41	ASCII "DATA"	SECTION
00400228	00100000	DD 00001000	VirtualSize = 1000 (4096.)
0040022C	00200000	DD 00002000	VirtualAddress = 2000
00400230	00020000	DD 00000200	SizeOfRawData = 200 (512.)

Ahora guardamos los cambios con el procedimiento usual

00401095

6A 00

PUSH

00401097

FF35 C8204000

PUSH

0040109D

6A 00

PUSH

0040109F

6A 00

PUSH

AL=00

DS:[00000000]=???

Find references

Ctrl+R

View executable file

Copy to executable file

Go to

Hex

Text

Short

Long

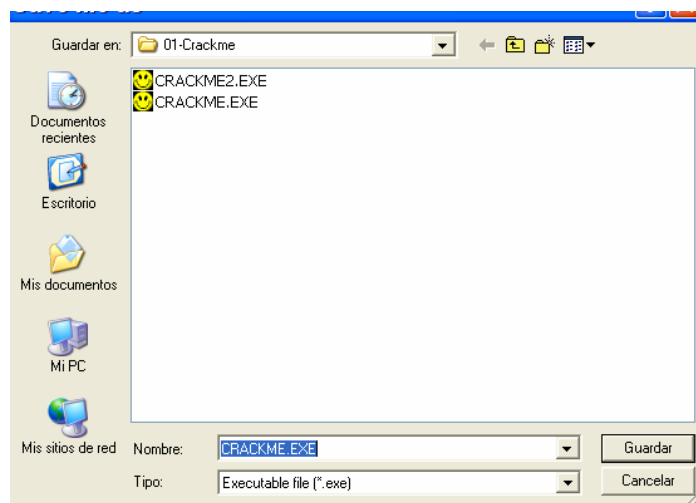
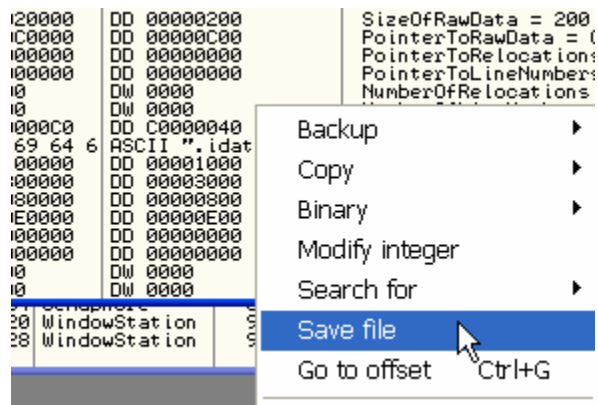
Float

Disassemble

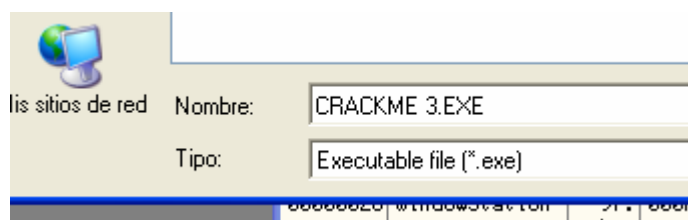
Special

Appearance

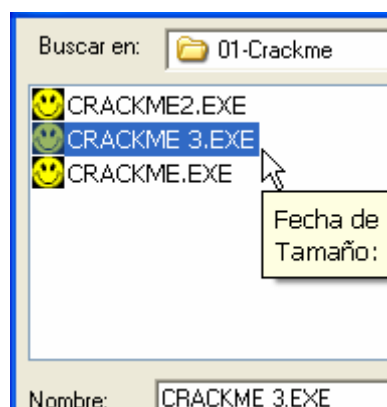
Address	Hex dump	Data
004001D4	00000000	DD 00000000
004001D8	00000000	DD 00000000
004001DC	00000000	DD 00000000
004001E0	00000000	DD 00000000
004001E4	00000000	DD 00000000
004001E8	00000000	DD 00000000
004001EC	00000000	DD 00000000
004001F0	00000000	DD 00000000
004001F4	00000000	DD 00000000
004001F8	43 4F 44 41	ASCII "CODE"
00400200	00100000	DD 00001000
00400204	00100000	DD 00001000
00400208	00060000	DD 00000600
0040020C	00060000	DD 00000600
00400210	00000000	DD 00000000
00400214	00000000	DD 00000000
00400218	0000	DW 0000
0040021A	0000	DW 0000
0040021C	200000E0	DD E0000020
00400220	44 41 54 41	ASCII "DATA"
00400228	00100000	DD 00001000

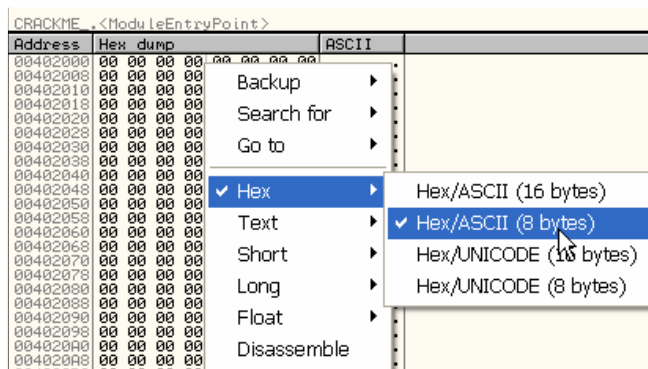


Le cambiare el nombre a CRACKME 3 para saber que este es el que modifique.



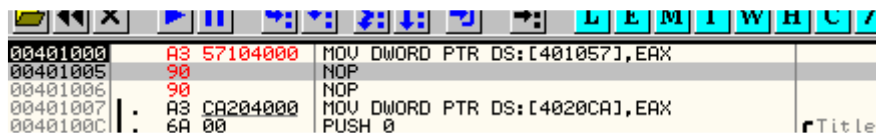
Ahora busquemos ese crackme 3 y abrámoselo en OLLYDBG





Volvamos a la visualización normal del dump

Probemos ahora la instrucción que hicimos antes



Apretemos F8

Vemos que no se produjo excepción y guardo el valor de EAX en el contenido de 401057 perfectamente.

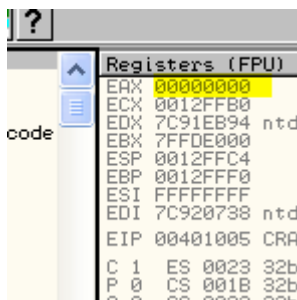
Otro tipo de excepción es

División entre 0: Se produce cuando se intenta dividir un número con 0.

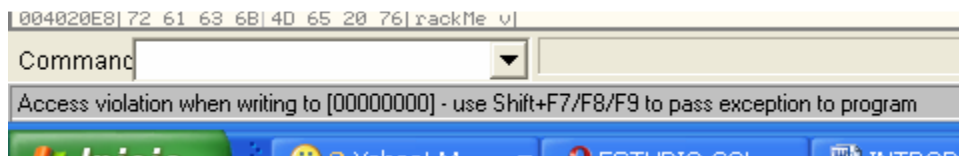
Por ejemplo escribo en OLLYDBG



Eso dividira ECX por EAX, si EAX es cero se producirá una excepción.



Apreto f8



No siempre OLLYDBG aclara bien las excepciones, no me dice que fue una división por cero, pero salto la excepción.

Instrucción no valida intento de ejecución de instrucción privilegiada: Se produce cuando el procesador intenta ejecutar una instrucción que no pertenece a su juego de instrucciones, es decir al encontrar un código de operación desconocido

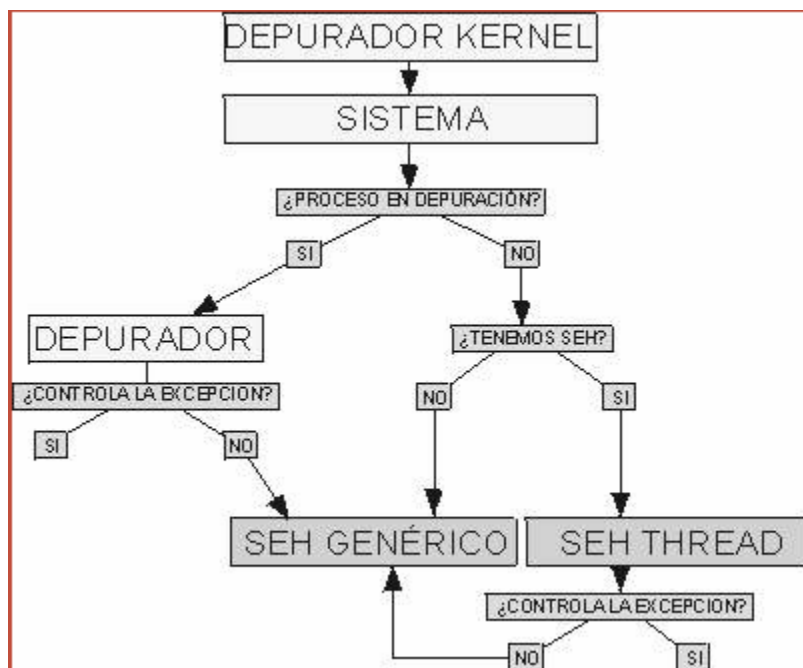
Eso no lo podemos probar porque OLLYDBG no nos deja escribir instrucciones que no existen para el procesador, pero el programador puede haber deslizado alguna instrucción inválida que no corresponde al juego de instrucciones que el procesador puede interpretar y lógico dará error al no saber que hacer.

La otra muy conocida es el INT 3 que provoca una excepción y es usada por el debugger cuando ponemos un BPX común para detener el programa y tomar el mando del mismo en el momento que se ejecuta el INT 3.

También algunos programas colocan INT 3 directamente escribiendo la sentencia, lo cual es una excepción más de todas las posibles.

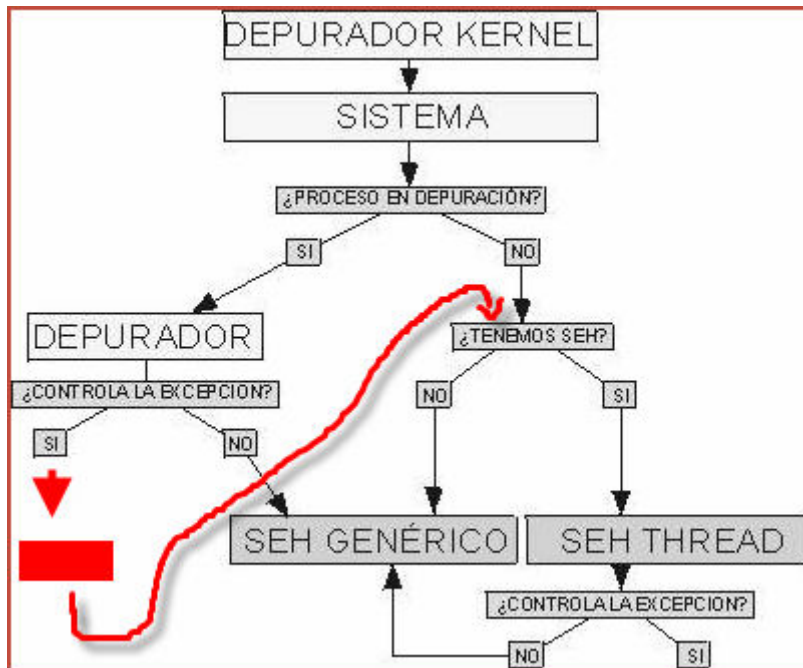
Hay muchísimas excepciones por supuesto no veremos todas esto es un simple ejemplo.

Ahora sabemos que en el programa se pueden producir excepciones, pero que ocurre cuando se genera una, veamos el siguiente esquema.



Aquí hay un grafico que le afanamos al tute de SILVER, allí vemos que al producirse una excepción, lo primero que ve el sistema es si el proceso esta siendo debuggeado.

En el caso que este siendo debuggeado, toma el control el DEBUGGER O DEPURADOR el cual en la imagen vemos que vemos que puede controlar la excepción o no, casi siempre CONTROLA LA EXCEPCION, y en el caso del OLLYDBG se fija si esta puesta la tilde en DEBUGGING OPTIONS-EXCEPTIONS para saltar ese tipo de excepción con lo cual si esta marcada, le devuelve el control al programa, de no estar la tilde espera que apretes SHIFT mas f9 para devolver el control al programa, o sea allí en el grafico donde dice CONTROLA LA EXCEPCION-SI, falta que una vez que toma por ese camino decide parar o continuar según haya tilde o no, y luego continua según la imagen de abajo.



Allí vemos el trabajo completo, luego de CONTROLAR LA EXCEPCION decide si para o continua si esta la tilde o no, si para, espera que apretes SHIFT mas F9 para volver donde muestra la flecha roja que agregue, y si no para, vuelve adonde muestra la flecha roja directamente.

Allí vemos que cuando retorna del OLLYDBG pregunta si tenemos SEH instalado, si tenemos va a el, y si no va al SEH genérico, expliquemos un poco esto que parece difícil pero no lo es.

QUE ES EL SEH

El SEH o Structured Exception Handling (Manejo Estructurado de Excepciones), es un manejador que se instala para poder recuperar a un programa de un error, ya que si no tenemos instalado un SEH propio, al encontrar un ERROR, se ejecuta el SEH GENÉRICO del sistema que en un 90% de los casos termina en el cartel que nos muestra que ha habido un error en la aplicación y se debe cerrar el programa y chau.

Con nuestro propio SEH, podemos interceptar un error, arreglarlo, y volver al programa y que continúe corriendo sin que salga el molesto cartel del sistema y nos cierre la aplicación.

También debemos decir que aunque aun no hemos visto que son los threads a fondo, una mínima definición de threads es que son hilos del programa o partes que se pueden ejecutar simultáneamente, hago esta definición porque cada thread tiene su propio manejo de excepciones, si a un thread le colocas un manejador de excepciones solo funcionara en el thread que fue colocado y no en los otros.

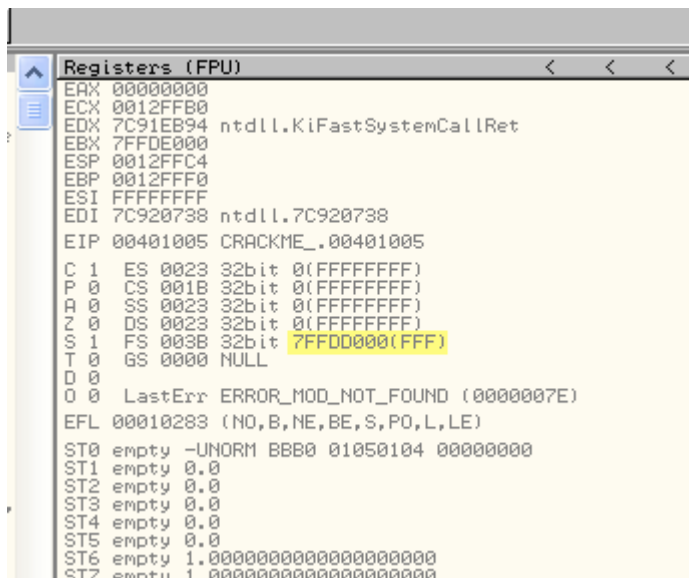
Como se coloca un Manejador de excepciones

Bueno veamos el crackme de cruehead nuevamente en el stack vemos

0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDE000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	8445FA58	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00401000	CRACKME_.<ModuleEntryPoint>
0012FFFC	00000000	

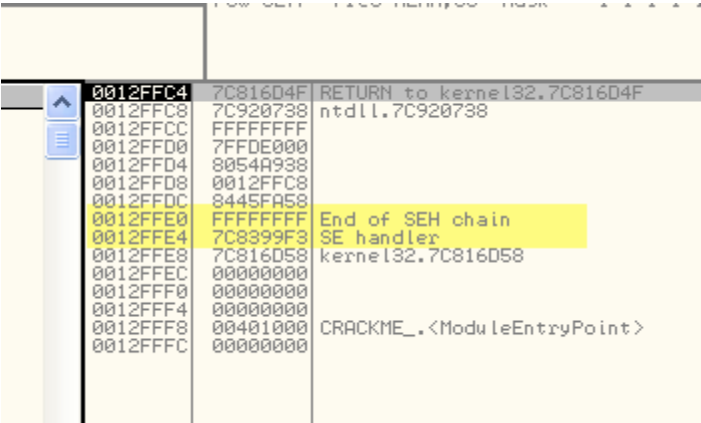
Ese es el SEH GENERICO que instala el sistema, mientras no se instale uno propio, cualquier error ira directo a este manejador que nos lleva casi siempre, al fatídico cartelito de error, veamos la ubicación de los punteros del SEH GENERICO.

Como habíamos visto que en FS :[0] estaba el puntero al manejador de excepciones que esta vigente, podemos ir en el dump allí, también mirando en OLLYDBG



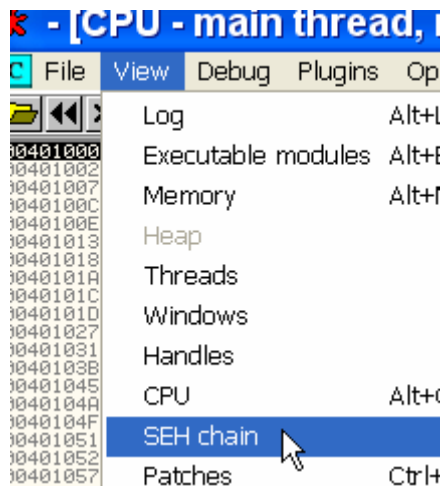
Address	Hex dump	ASCII
7FFD0000	E0 FF 12 00 00 00 13 00	ó ¤...!!.
7FFD0008	00 D0 12 00 00 00 00 00	.S ¤....
7FFD0010	00 1E 00 00 00 00 00 00	. ¤....
7FFD0018	00 D0 FD 7F 00 00 00 00	.S ¤ ¤....
7FFD0020	C8 08 00 00 4C 0A 00 00	¤ ¤.L....
7FFD0028	00 00 00 00 00 00 00 00
7FFD0030	00 E0 FD 7F 7E 00 00 00	. ¤ ¤ ¤....
7FFD0038	00 00 00 00 00 00 00 00
7FFD0040	38 A2 F5 E1 00 00 00 00	8 ¤ ¤ ¤....
7FFD0048	00 00 00 00 00 00 00 00
7FFD0050	00 00 00 00 00 00 00 00
7FFD0058	00 00 00 00 00 00 00 00
7FFD0060	00 00 00 00 00 00 00 00
7FFD0068	00 00 00 00 00 00 00 00
7FFD0070	00 00 00 00 00 00 00 00
7FFD0078	00 00 00 00 00 00 00 00

Como habíamos visto Fs:[0] es el contenido de la dirección de memoria que me marca el OLLYDBG puede variar en cada maquina, pero en la mía es entonces FS: [0] es 12ffe0

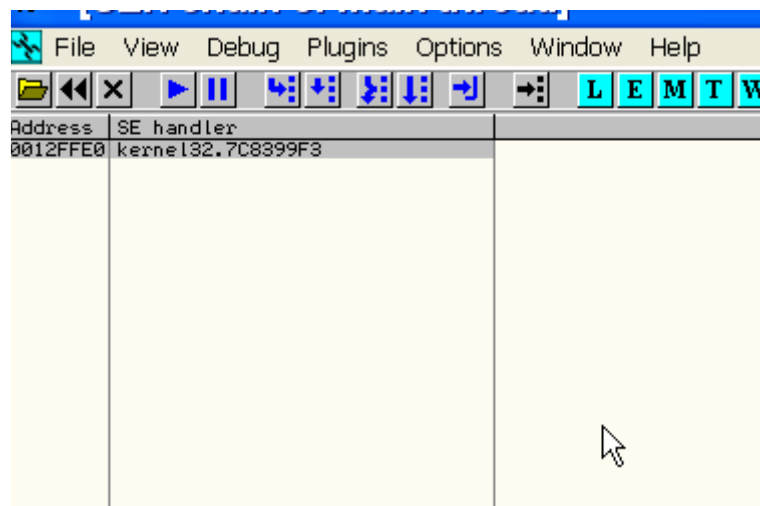


Como vemos en el stack 12ffe0 apunta al final de la cadena de manejadores, o sea al manejador que se encuentra en funcionamiento en este thread actualmente, una posición mas abajo esta la dirección donde

saltara al encontrar una excepción, es este caso como es el manejador genérico saltara a una dirección del sistema que terminara mostrándonos el famoso cartelito de error.



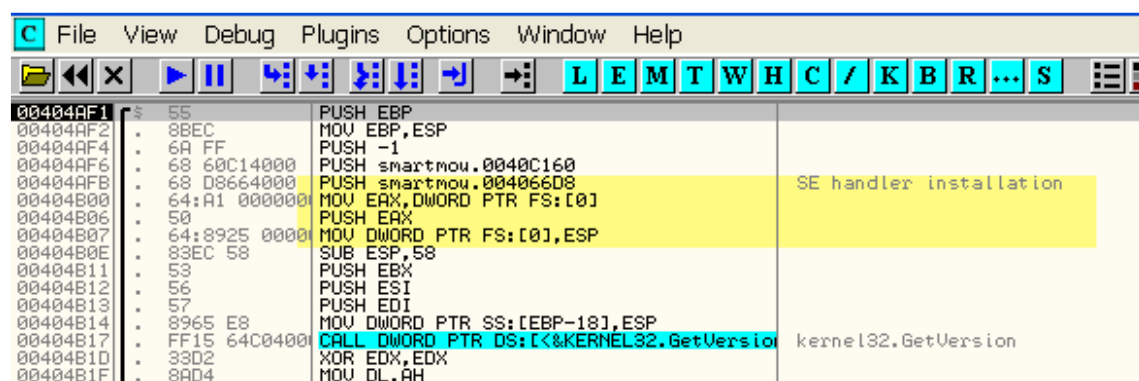
Y si vamos a VIEW-SEH CHAIN



Vemos la dirección del manejador genérico que es el único instalado, si hubiera aquí mas de uno, el que funciona al encontrar una excepción es el superior de toda la lista.

Ya que el crackme de cruehead no instala manejadores de excepción propios, utilizaremos un programita llamado SMARTMOUSE que adjunto al tutorial.

Lo arranco en OLLYDBG



Ali nomás veo que al inicio el programa va a instalar su manejador de excepciones propio, OLLYDBG nos lo indica en el comentario.

Como se hace esto traceemos y expliquémoslo paso a paso.

00404AF1	55	PUSH EBP	
00404AF2	8BEC	MOV EBP,ESP	
00404AF4	6A FF	PUSH -1	
00404AF6	68 60C14000	PUSH smartmou.0040C160	
00404AF8	68 08664000	PUSH smartmou.004066D8	SE handler installation
00404B00	64:A1 000000	MOV EAX,DWORD PTR FS:[0]	
00404B06	50	PUSH EAX	
00404B07	64:8925 0000	MOV DWORD PTR FS:[0],ESP	
00404B0E	83EC 58	SUB ESP,58	
00404B11	53	PUSH EBX	
00404B12	56	PUSH ESI	
00404B13	57	PUSH EDI	
00404B14	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
00404B17	FF15 64C04000	CALL DWORD PTR DS:[<&KERNEL32.GetVersion	kernel32.GetVersion
00404B1D	33D2	XOR EDX,EDX	

Llegamos traceando a la instrucción que el OLLYDBG nos indica que empieza la instalación del manejador de excepciones, lo primero es hacer un PUSH con la dirección a la cual queremos que salte el programa, cuando halle una excepción, en este caso, será 4066d8, ejecutamos el push.

0012FFB4	004066D8	Entry address
0012FFB8	0040C160	smartmou.0040C160
0012FFBC	FFFFFFFF	
0012FFC0	0012FFF0	
0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDF000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	84E80918	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00404AF1	smartmou.<ModuleEntryPoint
0012FFFC	00000000	

Allí coloco la dirección en el stack

00404AF6	68 60C14000	PUSH smartmou.0040C160	
00404AF8	68 08664000	PUSH smartmou.004066D8	SE handler installation
00404B00	64:A1 000000	MOV EAX,DWORD PTR FS:[0]	
00404B06	50	PUSH EAX	
00404B07	64:8925 0000	MOV DWORD PTR FS:[0],ESP	
00404B0E	83EC 58	SUB ESP,58	
00404B11	53	PUSH EBX	
00404B13	56	PUSH ESI	

La siguiente línea mueve el valor actual de fs : [0] a EAX vayamos a ver cuanto vale Fs [0] en el dump

```

Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 ntdll.KiFastSystemCallRet
EBX 7FFDF000
ESP 0012FFB4
EBP 0012FFC0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C920738
EIP 00404B00 smartmov.00404B00
C 1 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 1 FS 003B 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_MOD_NOT_FOUND (00000000)
EFL 00000283 (NO,B,NE,BE,S,PO,L,LE)
ST0 empty -UNORM BCE0 01050104 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.00000000000000000000
ST7 empty 1.00000000000000000000
3 2 1 0 E S P U O Z

```

Vamos allí

smartmov.<ModuleEntryPoint>+0F			
Address	Hex	dump	ASCII
7FFDE000	E0 FF 12 00	00 00 13 00	0 0...!!
7FFDE008	00 00 12 00	00 00 00 00	.S+....
7FFDE010	00 1E 00 00	00 00 00 00	.A.....
7FFDE018	00 E0 FD 7F	00 00 00 00	.0^Δ....
7FFDE020	00 02 00 00	04 0F 00 00	\$0..ë%..
7FFDE028	00 00 00 00	00 00 00 00
7FFDE030	00 F0 FD 7F	7E 00 00 00	..-2Δ....
7FFDE038	00 00 00 00	00 00 00 00
7FFDE040	B0 24 4E E1	00 00 00 00	;\$Np....
7FFDE048	00 00 00 00	00 00 00 00
7FFDE050	00 00 00 00	00 00 00 00
7FFDE058	00 00 00 00	00 00 00 00
7FFDE060	00 00 00 00	00 00 00 00
7FFDE068	00 00 00 00	00 00 00 00
7FFDE070	00 00 00 00	00 00 00 00
7FFDE078	00 00 00 00	00 00 00 00

Allí vemos que fs: [0] vale 12ffe0, podemos ver que el olly nos lo aclara también

00404B53	.v 75 08	JNZ SHORT smart
00404B55	. 6A 1C	PUSH 1C
FS:[00000000]=[7FFDE000]=0012FFE0		
EAX=00000000		
smartmov.<ModuleEntryPoint>+0F		
Address	Hex	dump
ASCII		

Ahora ejecutamos esta línea

```

Registers (FPU)
EAX 0012FFE0
ECX 0012FFB0
EDX 7C91EB94 ntdll.K
EBX 7FFDF000
ESP 0012FFB4
EBP 0012FFC0
ESI FFFFFFFF
EDI 7C920738 ntdll.7
EIP 00404B06 smartmo
C 1 ES 0023 32bit 0
P 0 CS 001B 32bit 0
A 0 SS 0023 32bit 0
Z 0 DS 0023 32bit 0
S 1 FS 003B 32bit 7
T 0 GS 0000 NULL

```

Movió a EAX ese valor

Ahora hace un PUSH con ese valor

0012FFB0	0012FFE0	
0012FFB4	004066D8	Entry address
0012FFB8	0040C160	smartmou.0040C160
0012FFBC	FFFFFFFF	
0012FFC0	0012FFF0	
0012FFC4	7C816D4F	RETURN to kernel32
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDF000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	84E80918	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00404AF1	smartmou.<ModuleEn
0012FFFC	00000000	

Vemos porque se le dice SEH CHAIN que significa cadena ya que este valor que guardo, apunta al manejador anterior que por ahora esta activo.

00404800	. 64:A1 000000	PUSH smartmou.004066D8	
00404806	. 50	MOV EAX,DWORD PTR FS:[0]	
00404807	. 64:8925 0000	PUSH EAX	
0040480E	. 83EC 58	MOV DWORD PTR FS:[0],ESP	
00404811	. 53	SUB ESP,58	
00404812	. 56	PUSH EBX	
00404813	. 57	PUSH ESI	
00404814	. 8965 E8	PUSH EDI	
		MOV DWORD PTR SS:[EBP-18],ESP	

La ultima línea cambia fs:[0], le coloca el valor de ESP o sea apunta a donde esta ubicada esta estructura nueva que escribimos

Registers (FPU)	
EAX	0012FFE0
ECX	0012FFB0
EDX	7C91EB94 ntdll
EBX	7FFDF000
ESP	0012FFB0
ESI	FFFFFFFF
EDI	7C920738 ntdll
EIP	00404807 smartmou
C 1	ES 0023 32b
P 0	CS 001B 32b
D 0	SS 0023 32b

0012FFB0	0012FFE0	
0012FFB4	004066D8	Entry address
0012FFB8	0040C160	smartmou.0040C160
0012FFBC	FFFFFFFF	
0012FFC0	0012FFF0	
0012FFC4	7C816D4F	RETURN to ke:
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDF000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	84E80918	
0012FFE0	FFFFFFFF	End of SEH cl
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C8
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00404AF1	smartmou.<Mo
0012FFFC	00000000	

Al ejecutar la línea

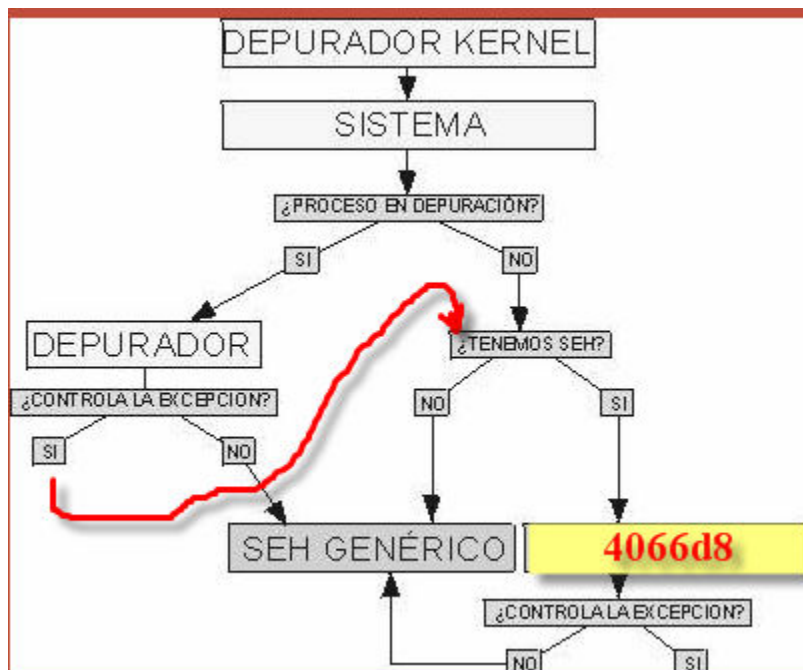
smartmou.<ModuleEntryPoint>+1D			
Address	Hex	dump	ASCII
7FFDE000	00 FF 12 00	00 00 13 00	...
7FFDE008	00 D0 12 00	00 00 00 00	...
7FFDE010	00 1E 00 00	00 00 00 00	...
7FFDE018	00 E0 FD 7F	00 00 00 00	...
7FFDE020	00 02 00 00	04 0F 00 00	...
7FFDE028	00 00 00 00	00 00 00 00	...
7FFDE030	00 F0 FD 7F	7E 00 00 00	...
7FFDE038	00 00 00 00	00 00 00 00	...
7FFDE040	00 24 4E E1	00 00 00 00	...
7FFDE048	00 00 00 00	00 00 00 00	...
7FFDE050	00 00 00 00	00 00 00 00	...
7FFDE058	00 00 00 00	00 00 00 00	...
7FFDE060	00 00 00 00	00 00 00 00	...

Tenemos nuestro manejador instalado en 12ffb0, como siempre vemos que fs:[0] queda apuntando al manejador actual

0012FFB0	0012FFE0	Pointer to next SEH record
0012FFB4	004066D8	SE handler
0012FFB8	0040C160	smartmou.0040C160
0012FFBC	FFFFFFFF	
0012FFC0	0012FFF0	
0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDF000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	84E80918	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00404AF1	smartmou.<ModuleEntryPoint>
0012FFFC	00000000	

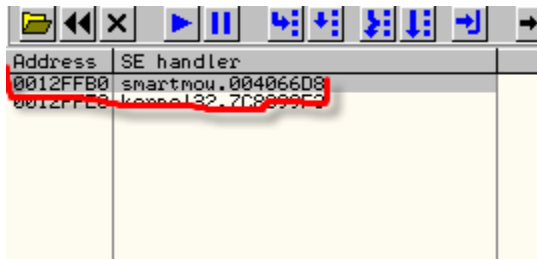
El cual OLLYDBG nos indica que es un SEH, el primer valor apunta al viejo SEH, y el segundo valor es la dirección que cuando halle una excepción el programa saltara a tratar de arreglar las cosas.

O sea que cuando ocurra una excepción en este ejemplo



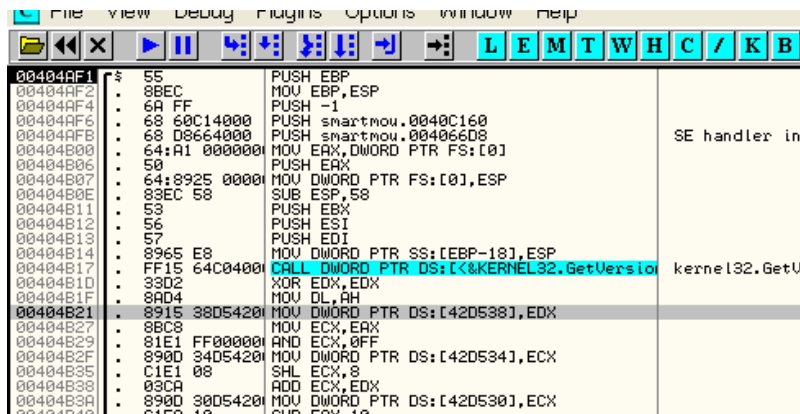
El sistema le dará control al debugger el cual parara o no según este la tilde de ese tipo de excepción marcada, y volverá al programa, directamente o apretando SHIFT +F9, adonde muestra el dibujo y como ahora tenemos SEH instalado, pues continuara en 4066d8.

Si vemos el menú SEH CHAIN ahora

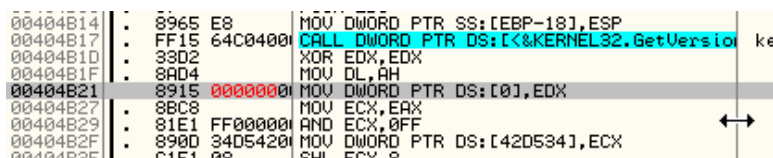


Vemos que quedo como manejador activo el que instalamos y el genérico quedo debajo sin funcionar.

Forcemos una excepción en el programa

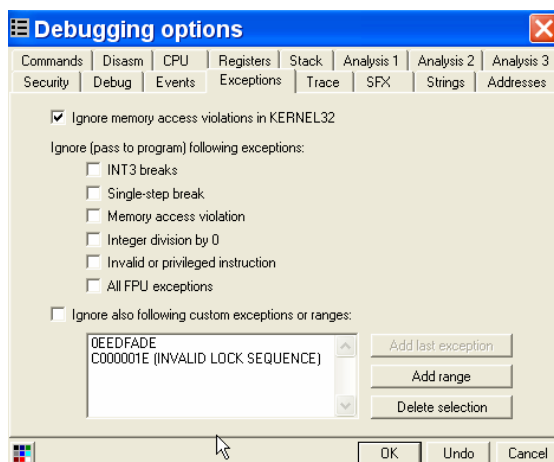


Veamos allí, cambiemos esa línea por



Lo cual dará error pues no se puede escribir en la dirección 0.

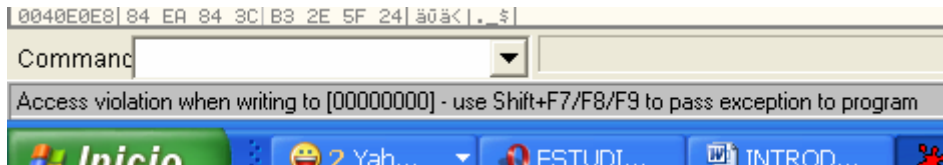
Quitemos todas las tildes en debugging option-exceptions, salvo la 1ra.



Demos RUN

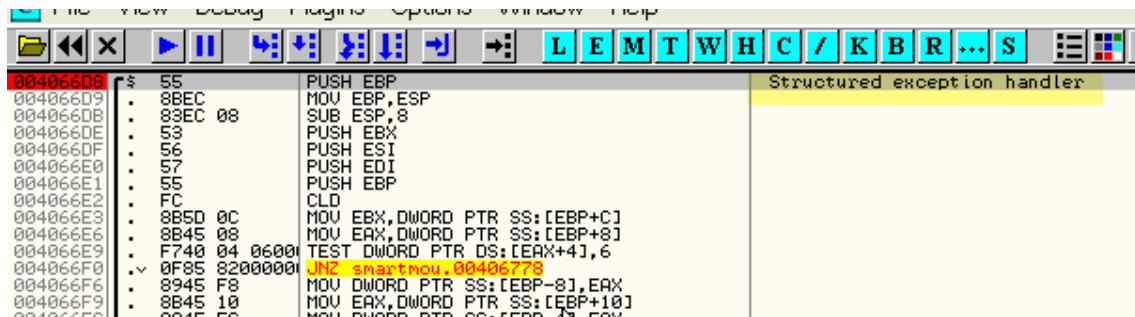
00404B10	33D2	XOR EDX,EDX
00404B1F	8AD4	MOV DL,AH
00404B21	8915 00000000	MOV DWORD PTR DS:[0],EDX
00404B27	8BC8	MOV ECX,EAX
00404B29	81E1 FF000000	AND ECX,0FF
00404B2F	890D 34D54200	MOV DWORD PTR DS:[42D534],ECX
00404B35	C1E1 08	SHL ECX,8

Para en el error mostrando

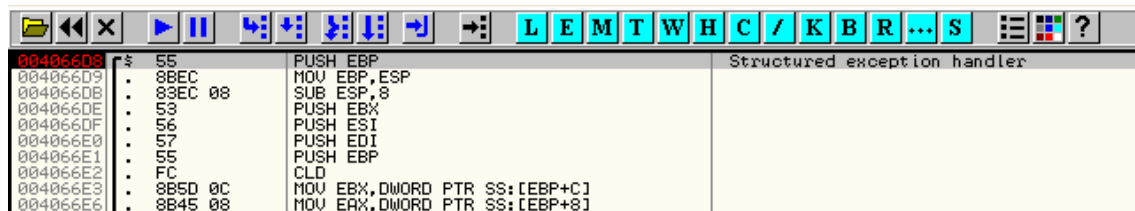


Entonces según lo que vimos el programa debería ahora continuar ejecutándose en el manejador para tratar de recuperar el error.

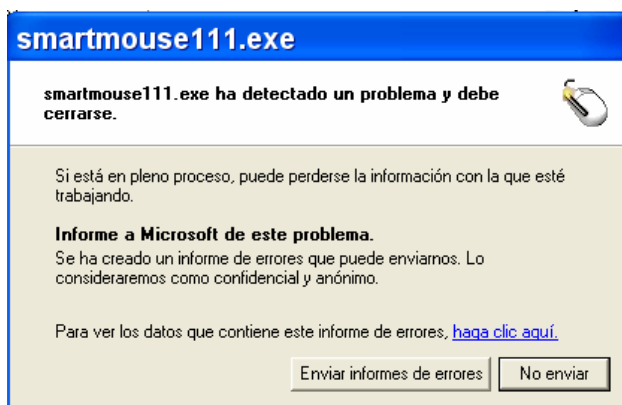
El manejador estaba en 4066d8 vayamos allí y pongamos un BP,



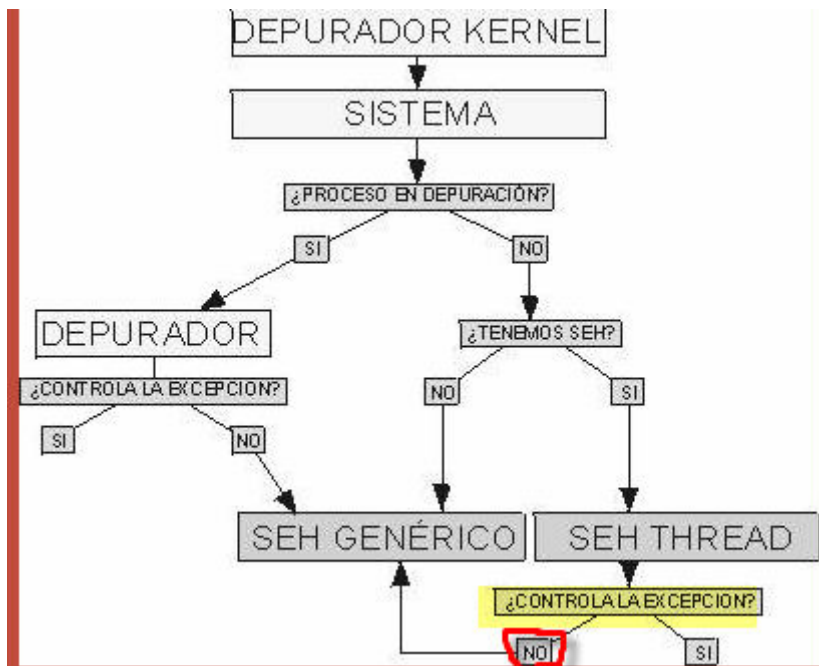
Como vemos OLLYDBG siempre nos aclara todo jeje, apretemos SHIFT + f9 para pasar la excepción



Como vemos paro en el manejador, veremos que hace el mismo y si es capaz de recuperarse de un error tan grave ya que altere el código del programa para generarlo.



Vemos que no pudo continuar y salto al manejador genérico

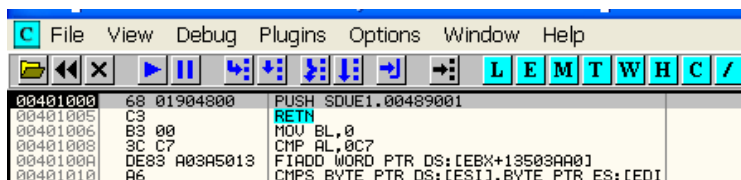


Ocurrió este caso como el error es muy grave no lo pudo recuperar y salto al manejador genérico que saco el cartel fatídico y cierra el programa.

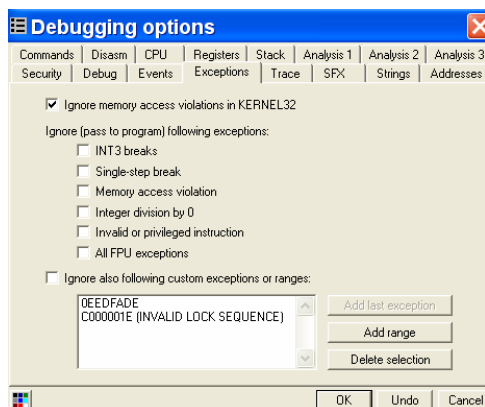
Obviamente el manejador de excepciones esta diseñado para otro tipo de excepciones las cuales espera, no esta que es una alteración del código lisa y llana, pero vemos igual como funciona en el caso de no controlar la excepción.

Para ver la otra rama del grafico que es cuando un manejador controla las excepciones en forma exitosa, veamos este crackme SDUE que adjunto.

Lo arranco en OLLYDBG y veo que esta empacado por el mensaje del OLLYDBG de puede ser automodificable etc etc.



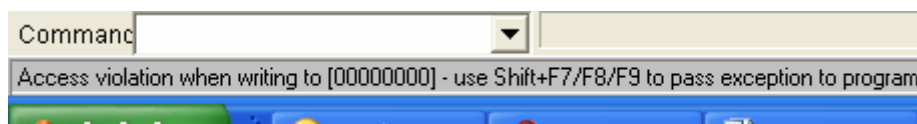
Quito todas las marcas en debugging options- exceptions, menos la 1ra



Y doy RUN

Address	Disassembly	Comment
00000090	XOR DWORD PTR DS:[EAX],EAX	
0000009F	JMP SHORT 000000A2	
000000A1	PUSH 58F64	
000000A6	ADD BYTE PTR DS:[EAX],AL	
000000A8	ADD BL,CH	
000000AA	ADD CH,AL	
000000AF	ORN DWORD PTR DS:[EAX+68],EBX	

Vemos que para en una excepción veamos



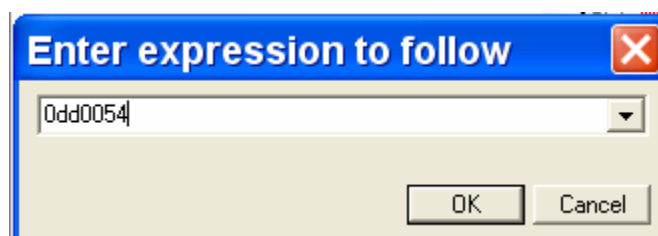
Donde se encuentra el manejador

Address	SE handler
0012FF90	00000054
0012FFE0	kernel32.7C8399F3

En sus maquinas esta dirección puede variar ya que es una sección creada mientras se ejecuta el programa.

Pongamos un BP allí

0012FF90	0012FFE0	Pointer to next SEH record
0012FF94	00000054	SE handler
0012FF98	00489030	RETURN to SDUE1.00489030 from SDUE1.00489031
0012FF9C	00000000	
0012FFA0	004892B9	SDUE1.004892B9



Address	Disassembly	Comment
00000054	PUSH 0000005D	
00000059	INC DWORD PTR SS:[ESP]	
0000005C	RETN	
0000005D	MOV ESP,0C24448B	
00000062	JMP SHORT 00000065	

Pongamos UN BPX allí

Address	Disassembly	Comment
00000054	PUSH 0000005D	
00000059	INC DWORD PTR SS:[ESP]	
0000005C	RETN	
0000005D	MOV ESP,0C24448B	
00000062	JMP SHORT 00000065	
00000064	XCHG BYTE PTR DS:[EBX+B880],AL	
0000006A	ADD BYTE PTR DS:[EDX],AL	
0000006C	JMP SHORT 00000068	

Apretemos SHIFT mas f9

00000054	68 50000000	PUSH 0000005D
00000055	FF0424	INC DWORD PTR SS:[ESP]
0000005C	C3	RETN
0000005D	BC 8B44240C	MOV ESP,0C24448B
00000062	EB 01	JMP SHORT 00000065
00000064	8683 80B80000	XCHG BYTE PTR DS:[EBX+B880],AL

Para en el manejador, el cual debe retornar si no hace cambios raros a la línea siguiente de donde se provoco la excepción para continuar el programa.

0000009D	3100	XOR DWORD PTR DS:[EAX],EAX
0000009F	EB 01	JMP SHORT 000000A2
000000A1	68 648F0500	PUSH 58F64
000000A6	0000	ADD BYTE PTR DS:[EAX],AL
000000A8	00EB	ADD BL,CH
000000AA	00FA	ADD CH,CL

Pongo UN BP en la línea siguiente donde se genero al excepción y doy RUN

0000009D	3100	XOR DWORD PTR DS:[EAX],EAX
0000009F	EB 01	JMP SHORT 000000A2
000000A1	68 648F0500	PUSH 58F64
000000A6	0000	ADD BYTE PTR DS:[EAX],AL
000000A8	00EB	ADD BL,CH
000000AA	00FA	ADD CH,CL

Vemos que para y el error fue salvado y el programa continuara su ejecución perfectamente sin cartel de error.

Como yo vi que era una rutina simple, y no hay modificaciones asumo que retornara a la línea siguiente de donde se provoco la excepción, este es el comportamiento normal, aunque la dirección de retorno puede ser modificada en el mismo manejador, en ese caso lo mejor es poner un BPM ON ACCESS en la sección donde se provoco la excepción e ir apretando F9, lo cual nos hará saltar línea a línea, hasta que termine la rutina del manejador y vuelva al programa, allí quitamos el BPM ON ACCESS y continuamos ejecutando y vemos claramente donde retorno.

Por supuesto si queremos parar en el momento que el programa coloca el manejador de excepciones lo mejor es poner un HARDWARE BPX ON WRITE en Fs:[0] aunque para muchísimas veces ya que las dll también colocan manejadores luego los quitan y al volver al programa dejan el que estaba activo antes, pero es una forma de ver como se van colocando y quitando manejadores a medida que corre el programa.

La otra forma de colocar un manejador de excepción es por medio de la api SetUnhandledExceptionFilter, la cual ya vimos su funcionamiento, el parámetro es la dirección del manejador donde continuara la excepción siempre y cuando no haya debugger como vimos anteriormente.

Bueno creo que es un primer pantallazo sobre excepciones que nos permitirá ir manejándonos con mas profundidad en el cracking, a medida que veamos ejemplos profundizaremos el tema para no ser tediosos.