

Structured Exception Handling – SEH

(Manejo Estructurado de Excepciones)

Introducción

Bueno, ya que casi seguro este verano no esté muy activo (ingeniando inversamente) pues voy a escribir todo lo que he aprendido sobre el tema de las excepciones. Todo comenzó al intentar proteger un código en un programa en MASM que estoy haciendo. Así empezaron las dudas y más dudas. Con *zeroPad* analizamos muchas estructuras y códigos y al final llegamos a unas conclusiones y sé que si no las escribo pues las olvidaré. Y por eso este tute pienso que puede ser muy interesante. Además la mayoría de los tutoriales están en Inglés.

No es un tute extenso sobre SEH (no voy a hablar tampoco de anidar estructuras) sino un tute básico para entender ciertas estructuras y que pueden ayudar tanto al programador como al cracker.

Bibliografía

Introducción al OllyDBG desde cero – parte 25 (Ricardo Narvaja)

SEH - AIRBAG PARA TU CÓDIGO (MR Silver)

Win32 Exception handling for assembler programmers (Jeremy Gordon)

Leí (aunque estuviese en chino) y compilé TODO código e información que encontré por la red.

29 de julio de 2007

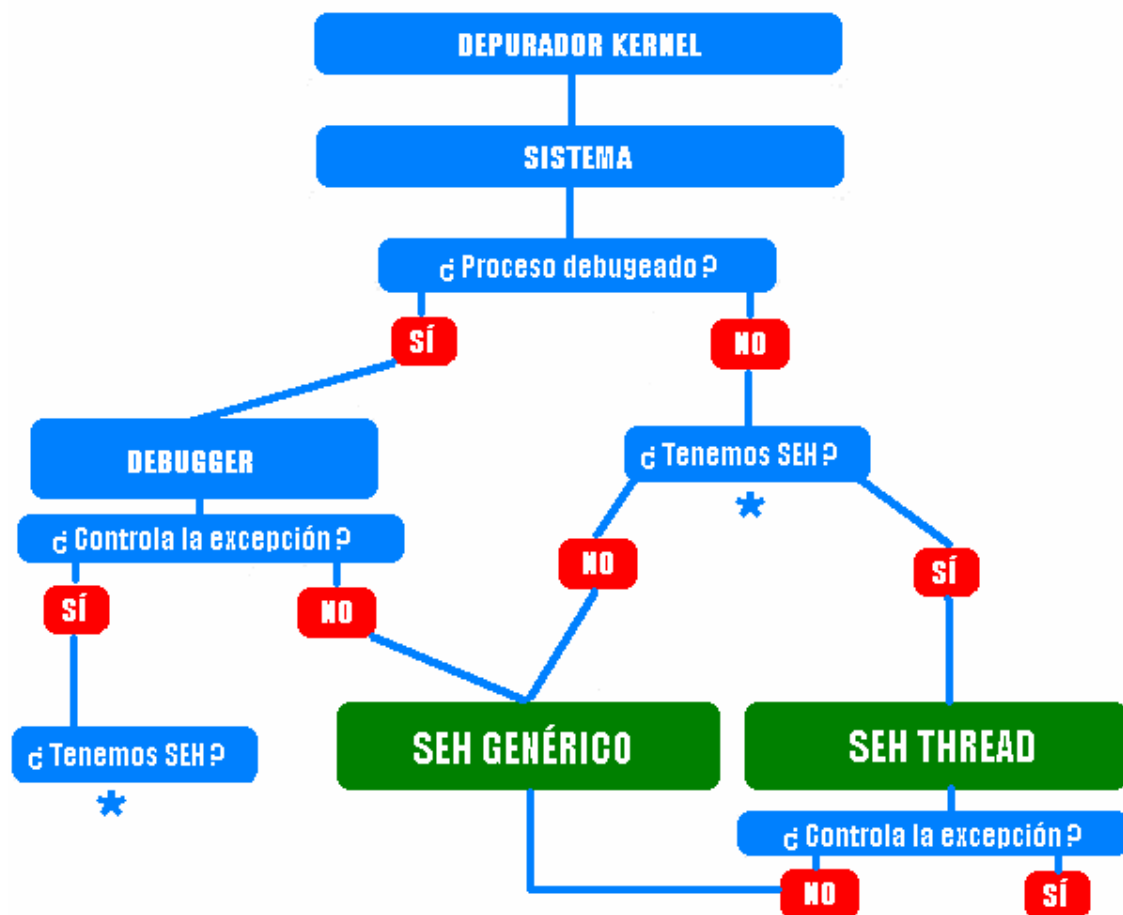
karmany

Iniciando

Structured Exception Handling (SEH - Manejo Estructurado de Excepciones) es un manejador que lo instalamos cuando por cualquier motivo queremos manejar nosotros mismos la excepción producida, si no la excepción será manejada por el SEH genérico del sistema. Normalmente se suelen utilizar para dar mayor información del error al usuario o para manejar algún error grave. De este modo, al producirse la excepción, podremos corregir el error o no y seguir la ejecución del programa por donde queramos, teniendo presentes algunas cosas que luego veremos.

Una cosa muy importante es que el manejador que pongamos sólo funcionará en el hilo (thread) en el que lo pongamos, no en los demás.

Voy a poner el gráfico de Silver (con un poco de colorido para que se vea bien):



Se observa que la primera pregunta es si el proceso está siendo debugeado.

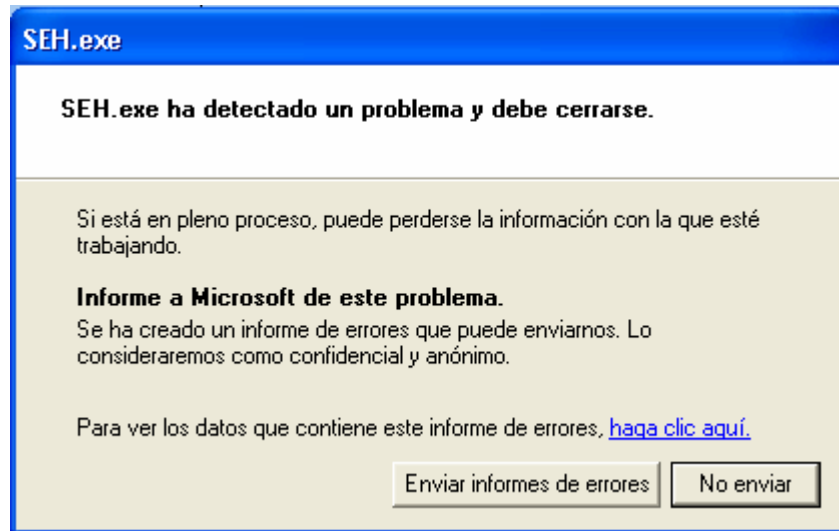
-Si el debugger la controla vemos que seguimos en “¿Tenemos SEH?”, que he puesto un asterisco para ver la continuación. (Todo esto está explicado por Silver y Ricardo en sus dos tutes respectivos). Sé que voy a decir al principio cosas muy similares ya vistas.

Lo principal que destaco es si tenemos un SEH propio, que si nuestro manejador de excepciones va a ser o no capaz de controlar la excepción.

SEH genérico

Es el manejador que se instala por defecto. Es controlado por la API **UnhandledExceptionFilter**.

Si nuestro programa comete algún error grave que no puede controlar terminará la ejecución con un cartelito muy conocido:



En otros casos, el manejador controla la excepción y el programa continúa perfectamente.

Antes de saber más sobre él creo que es conveniente saber cuáles son las excepciones que nos hacen saltar a nuestro SEH o al SEH genérico.

Excepciones

“Las excepciones se producen cuando el procesador realiza una operación no válida bajo el contexto del thread actual.” – Silver

He visto que hay muchas excepciones, más abajo pongo unas cuantas con el código que luego se explicará.

Para los recién iniciados en este tema voy a poner un par de ejemplos de excepciones que nos mandarán a nuestro SEH o al genérico:

*EXCEPTION_ACCESS_VIOLATION

Podemos encontrarnos esta excepción cuando intentamos acceder a la memoria y no está permitido. Por ejemplo:

```
XOR EAX, EAX ; EAX = 0
MOV DWORD PTR DS:[EAX],1 ; Excepción al intentar leer memoria 0.
```

*EXCEPTION_INTEGER_DIVIDE_BY_ZERO

Como su nombre indica se produce al hacer una división por cero, por ejemplo:

```
XOR EAX, EAX
DIV EAX
```

Todas estas excepciones, tienen un código (Dword) llamado: **ExceptionCode**

Con este código y la tabla que sigue podremos saber de qué excepción se trata, por ejemplo, para la excepción: *EXCEPTION_INTEGER_DIVIDE_BY_ZERO* que acabo de mostrar el ExceptionCode es: C0000094

Las excepciones más usuales y sus respectivos códigos de excepción están en una tabla más abajo.

¿Dónde está el SEH genérico?

Para ir entrando poco a poco en este tema, voy a abrir en el OllyDBG el programa que acabo de hacer y que viene con este tutorial: *SEH.exe*.

```
00401000  ES 11000000  CALL 00401016
00401005  B8 08000000  MOV EAX,8
0040100A  B8 08000000  MOV EAX,8
0040100F  6A 00        PUSH 0
00401011  E8 32000000  CALL 00401048
00401016  68 42104000  PUSH 401042
0040101B  64:FF35 0000  PUSH DWORD PTR FS:[0]
00401022  64:8925 0000  MOV DWORD PTR FS:[0],ESP
00401029  33C0        XOR EAX,EAX
0040102B  F7F0        DIV EAX
0040102D  BB 03000000  MOV EBX,3
00401032  BB 02000000  MOV EBX,2
00401037  64:8F05 0000  POP DWORD PTR FS:[0]
0040103E  83C4 04     ADD ESP,4
00401041  C3         RETN
00401042  B8 00000000  MOV EAX,0
00401047  C3         RETN
00401048  FF25 00204000  JMP NEAR DWORD PTR DS:[402000]
```

Muy sencillo, para entenderlo.

Observación: No ejecutar el programa porque hace un bucle infinito y habrá que cerrarlo pulsando Crt+Alt+Supr

Se ejecuta todo en la subrutina 00401016. Se puede observar que hago una división por cero: 0040102B, con lo cuál ya sabemos que se producirá una excepción conocida:

EXCEPTION_INTEGER_DIVIDE_BY_ZERO - 0C0000094H

De momento, porque quiero hacerlo entendible a todo newbie, lo dejamos aquí y ponemos la mirada en la pila:

```
0012FFC4  7C816FD7  RETURN to kernel32.7C816FD7
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDA000
0012FFD4  8054AB38
0012FFD8  0012FFC8
0012FFDC  81543CB8
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C839AA8  SE handler
0012FFE8  7C816FE0  kernel32.7C816FE0
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  SEH.<ModuleEntryPoint>
0012FFFC  00000000
```

Como se puede observar ahí tenemos instalado nuestro SEH genérico (aunque de momento no se entienda):

```
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C839AA8  SE handler
```

Las excepciones que se produzcan irán (de momento) al manejador que está en mi ordenador en 7C839AA8 como se puede ver. Además es el final de la cadena de manejadores de excepciones porque está con FFFFFFFF (-1). Aunque lo he adelantado (y puede no entenderse) después veremos qué significan esos dos Dword cuando instalemos nuestros propios SEH.

Si pulsamos en el OllyDBG: View → SEH chain veremos efectivamente lo siguiente:

SEH chain of main thread	
Address	SE handler
0012FFE0	kernel32.7C839AA8

Donde nos muestra que tenemos un solo SEH instalado y que el manejador se encuentra en la dirección 7C839AA8 que como sabemos es el SEH genérico y está en kernel32.dll

Y.. ¿Cómo sabemos que el SEH genérico se encuentra exactamente en 0012FFE0?¿No podría ser otra dirección?

No porque esa información también esta bien guardada.

Como sabemos (o deberíamos saber), el registro **FS** es un puntero a una estructura que se llama **Thread InformationBlock (TIB)**.

En este caso por ejemplo:

```
FS 0038 32bit 7FFDF000(FFF)
```

El puntero es a la dirección 7FFDF000.

Pues bien, de la estructura **TIB** (compleja y que tiene más punteros a otras diferentes y complicadas estructuras) podemos obtener información muy importante.

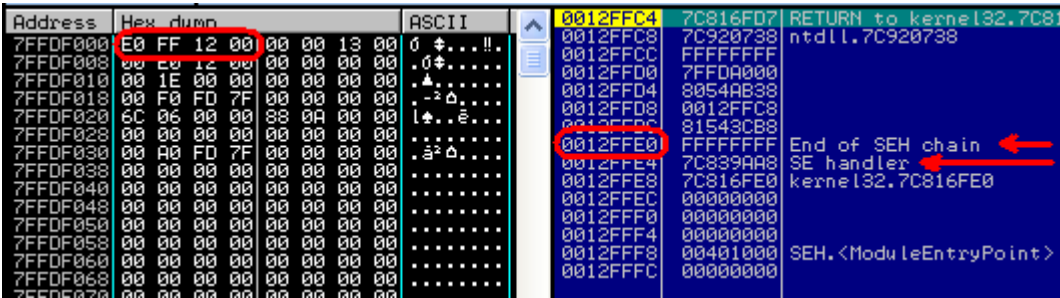
Para este tema de excepciones sólo nos vamos a fijar en el primer Dword que tiene la estructura **TIB**:

Estructura TIB

1º dword +0	Puntero a la estructura: pvExcept (EXCEPTION_REGISTRATION_RECORD)	Head of exception record list
etc...		

O sea, que el primer valor que encontramos en la estructura TIB es otro puntero a una estructura de tipo EXCEPTION_REGISTRATION_RECORD (que sólo contiene 2 punteros).

Vayamos ahora en el OllyDBG a la estructura TIB que como dijimos está en la dirección (en mi ordenador solamente): 7FFDF000



Vemos que el puntero a la estructura de tipo EXCEPTION_REGISTRATION_RECORD corresponde justamente a la dirección en la pila “End of SEH chain”.

Por lo tanto, los 2 dword que señalo con dos flechas son los 2 punteros de la estructura que acabo de comentar. He visto que en algunos sitios es también llamada estructura **ERR** (las siglas, para acortar).

Estructura ERR

1° dword +0	Puntero a la siguiente estructura ERR	*Next
2° dword +4	Puntero al manejador de la excepción	PVOID Handler

Vamos a analizar los 2 dword que aparecen:

1° *FFFFFFFF*

Esto indica que es la última estructura ERR. Ya veremos cuando instalemos nuestro propio SEH cómo se modificará este valor.

2° *7C839AA8*

Efectivamente es la dirección del manejador. Sabemos que es el genérico.

Ha llegado el momento de ponerle más eslabones a la cadena.

Establecer nuestros propios SEH

Hay 2 tipos de manejadores de excepciones diferentes:

Tipo1 o de "por proceso":* Éste se establece utilizando el API **SetUnhandledExceptionFilter y no se encadena como el tipo 2, es decir, que sólo puede haber una para todo un proceso (incluidos sus threads). Se ejecuta cuando no hay un manejador de tipo 2 y cuando no hay debugger.

**Tipo2 o de "por Thread":* Éste es en el que me detendré a explicar con más detenimiento que es el que utiliza la arquitectura SEH.

Tipo1, “por proceso”

Para establecer un manejador de excepciones de este tipo, lo podemos hacer de la siguiente forma. Pongo el ejemplo y lo explico:

```

                                Entry_Point:
                                PUSH Manejador_Final
                                CALL SetUnhandledExceptionFilter
                                {
Aquí está el código que      i ...
queremos proteger           i ...
                                i ...
                                CALL ExitProcess

                                ;*****

                                Manejador_Final:
                                {
Instrucciones del            i ...
manejador.                  i ...
                                i ...
                                MOV EAX,1
                                RET

```

Es muy sencillo de entender. Llamamos a la API **SetUnhandledExceptionFilter** y le pasamos la dirección de nuestro manejador.

Seguidamente se ejecuta el código y si existiera alguna excepción saltaríamos a nuestro **Manejador_Final**, donde se intentará reparar la excepción.

El retorno desde el manejador tiene 3 posibles valores para EAX:

***EXCEPTION_EXECUTE_HANDLER (1):** Retorna de **UnhandledExceptionFilter** y ejecuta el manejador asociado. Normalmente suele terminar el proceso.

***EXCEPTION_CONTINUE_EXECUTION (-1):**

Retorna de **UnhandledExceptionFilter** y continua la ejecución del thread en el punto donde sucedió la excepción, suponiendo que no se altere el Eip desde el manejador de excepciones, si se alterara el Eip la ejecución continuaria desde el nuevo eip establecido por el manejador.

***EXCEPTION_CONTINUE_SEARCH (0):** Proceder con una ejecución normal de **UnhandledExceptionFilter**. Esto significa obedecer los flags de la API **SetErrorMode**, o invocar el cuadro de dialogo de error de aplicación.

Tipo2, “por thread”

Como se explicó para establecer nuestro SEH propio, deberemos hacerlo utilizando una estructura ERR correcta y por la definición de ERR tendremos que darle dos punteros:

1° a la siguiente estructura ERR

2° al nuestro manejador de excepciones.

Entendido esto vamos a ir de nuevo al OEP del programa que esta junto a este tute: *SEH.exe*. Nos vamos a fijar en la pila:

0012FFC4	7C816FD7	RETURN to kernel32.7C816FD7
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDA000	
0012FFD4	8054AB38	
0012FFD8	0012FFC8	
0012FFDC	81543CB8	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C839AA8	SE handler
0012FFE8	7C816FE0	kernel32.7C816FE0
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00401000	SEH.<ModuleEntryPoint>
0012FFFC	00000000	

Ahí vemos el final de la cadena SEH. Si quisiéramos añadir nuestro SEH propio el primer dato de nuestra estructura ERR tendría que ser:

1° **0012FFE0**

y el segundo dato una dirección donde pondríamos nuestro manejador, en este caso:

2° **00401042**

Parece sencillo no?

Pues voy a poner esos dos valores directamente en la pila a ver qué ocurre:

0012FFBC	0012FFE0	
0012FFC0	00401042	Entry address
0012FFC4	7C816FD7	RETURN to kernel32.7C816FD7
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDE000	
0012FFD4	8054AB38	
0012FFD8	0012FFC8	
0012FFDC	82253B20	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C839AA8	SE handler
0012FFE8	7C816FE0	kernel32.7C816FE0
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00401000	SEH.<ModuleEntryPoint>
0012FFFC	00000000	

¿Cómo he puesto esos valores en la pila?

Pues muy sencillo: el primer valor que hay que poner lo he hecho tal cual:

PUSH 00401042

Mientras que el segundo valor si recordamos, nos lo daba un puntero de una estructura ERR que estaba contenida en la estructura TIB. Por lo tanto como sé que el puntero a la estructura ERR (que es lo que me interesa) corresponde al contenido de la dirección del registro FS, pues simplemente hay que hacer así:

PUSH DWORD PTR FS:[0]

Tiene que quedar bien claro este último párrafo.

¿Y así ya está todo?

Pues no. Solamente hemos puesto dos valores en la pila.

Ahora solamente hay que modificar el puntero de la primera estructura ERR, para que en vez de que “señale” en **0012FFE0**, pues que lo haga en **0012FFBC**. Y con eso ya estará todo.

Hacer esto es también muy sencillo porque el valor de la pila es ESP que corresponde con **0012FFBC**, y es el valor que nos interesa.

Por lo tanto lo siguiente que deberíamos hacer es:

MOV DWORD PTR FS:[0],ESP

Y ya está nuestro SEH propio. Además si miramos de nuevo la pila veremos cómo ha cambiado:



0012FFBC	0012FFE0	Pointer to next SEH record
0012FFC0	00401042	SE handler
0012FFC0	7C816FD7	RETURN to kernel32.7C816FD7
0012FFC0	7C920738	ntdll.7C920738
0012FFC0	FFFFFFFF	
0012FFD0	7FFDE000	
0012FFD0	8054AB38	
0012FFD0	0012FFC8	
0012FFD0	82253B20	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C839AA8	SE handler
0012FFE8	7C816FE0	kernel32.7C816FE0
0012FFEC	00000000	
0012FFFA	00000000	

Vemos que el primer **dword** de la estructura ERR:

0012FFBC 0012FFE0 Pointer to next SEH record, es el puntero a la siguiente estructura ERR, como ya se vió.

Mientras que el segundo **dword**:

0012FFC0 00401042 SE handler, corresponde por definición de la estructura ERR: “puntero al manejador de la excepción”.

Además si vamos a View → SEH chain vemos lo siguiente:

Address	SE handler
0012FFBC	SEH.00401042
0012FFE0	kernel32.7C839AA8

Se observa, por lo tanto, si utilizamos nuestro OllyDBG, que el manejador activo es el primero de toda la lista.

Pienso que no hay que explicar mucho más.

Desinstalar ahora nuestro SEH correctamente es sencillo. Como ya he explicado antes todos los valores pues ahora pongo solamente el código:

POP DWORD PTR FS:[0]

ADD ESP,4

Muy sencillo de entender.

Programando podemos hacer un código sencillo de esta forma:

```

        PUSH Nuestro_Manejador
        PUSH DWORD PTR FS:[0]
        MOV DWORD PTR FS:[0],ESP
Aquí está el código que } i...
queremos proteger      } i...
                        } i...
        POP FS:[0]
        ADD ESP,4h
        RET

        ;*****

        Nuestro_Manejador:
Instrucciones del } i...
manejador         } i...
                  } i...
        MOV EAX,1
        RET
```

Así cuando se produzca una excepción se ejecutará Nuestro_Manejador.
Podemos retornar desde nuestro manejador dando tres posibles valores a EAX:

EAX == -1 --> EXCEPTION_CONTINUE_EXECUTION

-Action: Returns an "invalid disposition" exception, indicating that this is not a valid return value!!!

-Acción: ¡¡Devuelve una "disposición inválida" de la excepción, indicando que no es un valor de retorno válido!!!

EAX == 1 --> EXCEPTION_EXECUTE_HANDLER

Action: Passes the exception to the win32 default handler (error box)

Acción: Pasa la excepción al manejador por defecto de win32 (diálogo de error).

EAX == 0 --> EXCEPTION_CONTINUE_SEARCH

Acción: Retorna al punto exacto donde se produjo el error, reejecutándolo otra vez haciendo por lo tanto un bucle infinito, si no se repara el error obviamente.

Observación importante: Como se puede observar, en los manejadores “por proceso” o en los manejadores “por thread”, los valores de EXCEPTION_CONTINUE_EXECUTION, EXCEPTION_EXECUTE_HANDLER y EXCEPTION_CONTINUE_SEARCH son diferentes. Por este motivo en estos últimos, hay que fijarse en el valor numérico (0, 1 o -1) y la explicación que se da. En realidad como se puede suponer, y es lógico, por ejemplo EXCEPTION_CONTINUE_EXECUTION se producirá cuando retorne al punto exacto donde se produjo el error y realmente tendría que ser EAX = 0, pero observamos que EAX = -1 (con EXCEPTION_CONTINUE_EXECUTION – “por thread”). Esto hay que tenerlo en cuenta porque en muchos ejemplos que hay en la red, es posible que veamos una variable de este tipo por ejemplo EXCEPTION_CONTINUE_EXECUTION y realmente el programa no retorne al punto donde se produjo la excepción. También cuando compilamos con MASM los valores de estas variables son los indicados en este tute. Sabiendo todo esto, ya cada uno en sus programas sabe lo que tiene que hacer, yo por ejemplo me creo mis propias variables.

Como ejemplo de este último caso con bucle infinito es el programa adjunto. Si observamos el código ahora ya debemos saber todo su funcionamiento.

```

00401000  ES:11000000 CALL 00401016
00401005  B8 08000000 MOV EAX,8
0040100A  B8 08000000 MOV EAX,8
0040100F  6A 00      PUSH 0
00401011  E8 32000000 CALL 00401048
00401016  68 42104000 PUSH 401042
0040101B  64:FF35 0000 PUSH DWORD PTR FS:[0]
00401022  64:8925 0000 MOV DWORD PTR FS:[0],ESP
00401029  33C0      XOR EAX,EAX
0040102B  F7F0      DIV EAX
0040102D  BB 03000000 MOV EBX,3
00401032  BB 02000000 MOV EBX,2
00401037  64:8F05 0000 POP DWORD PTR FS:[0]
0040103E  83C4 04    ADD ESP,4
00401041  C3        RETN
00401042  B8 00000000 MOV EAX,0
00401047  C3        RETN
00401048  FF25 00204000 JMP NEAR DWORD PTR DS:[402000]

```

Ahora queda la duda: ¿Cómo podemos salir de ese bucle infinito?

Sobre este tema busqué muchísima información, junto con zeroPad y al final llegamos a esta conclusión:

- O bien reparamos el error, como vimos en varios códigos y después retornamos con $EAX = 0$, que vuelve a ejecutar la misma instrucción pero esta vez bien.
- O bien, ponemos una etiqueta donde continuar y pasar y salvar la excepción.

Estas son las dos formas que vimos de evitar el bucle infinito. Al final pongo un código sencillo utilizando macros para usar en MASM.

En este momento, alguien puede pensar. Sí pero parado en mi SEH, ¿Cómo sé que tipo de excepción ha ocurrido? ¿o dónde ha ocurrido? O más datos sobre la excepción..porque sino, ¿Cómo voy a reparar la excepción?

Bien, vamos a ello.

La información enviada a los manejadores

En un artículo Matt Pietrek explica el mecanismo de las excepciones en Windows. Cuando Windows detecta una excepción mira a ver si hay manejadores buscando en el **TIB** y si es así, mete en la pila cuatro punteros:

[ExceptionRecord](#)

[EstablisherFrame](#)

[ContextRecord](#)

[DispatcherContext](#)

La información que debe ser enviada a los manejadores tiene que ser lo suficiente clara para que éstos puedan ser capaces de intentar reparar la excepción, hacer logs de error, o de informar usuario. Como veremos, cuando se llama a los manejadores, esta información es enviada por el sistema y la guarda en la pila. Además de esto, podemos enviar nuestra propia información a los manejadores ampliando la estructura **ERR**(ya vista) para que contenga más información.

Cuando vimos los 2 tipos de manejadores vimos que para el Tipo1 utilizabamos la API **UnhandledExceptionFilter**, pues bien, esta API recibe un solo parámetro y es un puntero a una estructura **EXCEPTION_POINTERS**. Estas estructuras que voy a explicar nos van a valer también después para el Tipo2.

Esta estructura se define así:

Estructura EXCEPTION_POINTERS

EXCEPTION_POINTERS +0	Puntero a la estructura EXCEPTION_RECORD
+4	Puntero a la estructura CONTEXT record

Como se observa tenemos que definir dos nuevas estructuras:

Estructura EXCEPTION_RECORD

EXCEPTION_RECORD +0	ExceptionCode
+4	ExceptionFlag
+8	Puntero a NestedExceptionRecord
+C	ExceptionAddress
+10	NumberParameters
+14	AdditionalData

Voy a explicar esos campos:

ExceptionCode:

Esto ya fue explicado cuando hablé de las excepciones. Es un código que nos indica el tipo de excepción que ha ocurrido.

Las más usuales son las siguientes:

EXCEPCIÓN	ExceptionCode
EXCEPTION_WAIT_0	00000000H
EXCEPTION_ABANDONED_WAIT_0	00000080H
EXCEPTION_USER_APC	000000C0H
EXCEPTION_TIMEOUT	00000102H
EXCEPTION_PENDING	00000103H
EXCEPTION_SEGMENT_NOTIFICATION	04000005H
EXCEPTION_GUARD_PAGE_VIOLATION	08000001H
EXCEPTION_DATATYPE_MISALIGNMENT	08000002H
EXCEPTION_BREAKPOINT	08000003H
EXCEPTION_SINGLE_STEP	08000004H
EXCEPTION_ACCESS_VIOLATION	0C0000005H
EXCEPTION_IN_PAGE_ERROR	0C0000006H
EXCEPTION_NO_MEMORY	0C0000017H
EXCEPTION_ILLEGAL_INSTRUCTION	0C000001DH
EXCEPTION_NONCONTINUABLE_EXCEPTION	0C0000025H
EXCEPTION_INVALID_DISPOSITION	0C0000026H
EXCEPTION_ARRAY_BOUNDS_EXCEEDED	0C000008CH
EXCEPTION_FLOAT_DENORMAL_OPERAND	0C000008DH
EXCEPTION_FLOAT_DIVIDE_BY_ZERO	0C000008EH
EXCEPTION_FLOAT_INEXACT_RESULT	0C000008FH
EXCEPTION_FLOAT_INVALID_OPERATION	0C0000090H
EXCEPTION_FLOAT_OVERFLOW	0C0000091H
EXCEPTION_FLOAT_STACK_CHECK	0C0000092H
EXCEPTION_FLOAT_UNDERFLOW	0C0000093H
EXCEPTION_INTEGER_DIVIDE_BY_ZERO	0C0000094H
EXCEPTION_INTEGER_OVERFLOW	0C0000095H
EXCEPTION_PRIVILEGED_INSTRUCTION	0C0000096H
EXCEPTION_STACK_OVERFLOW	0C00000FDH
EXCEPTION_CONTROL_C_EXIT	0C000013AH

ExceptionFlag:

Con los siguientes posibles valores:

0 - una excepción continuable (puede ser reparada) - **EXCEPTION_CONTINUABLE**

1 - Excepción no continuable (no puede ser reparada) - **EXCEPTION_NONCONTINUABLE**

(Cualquier intento de continuación de ejecución después de una excepción no continuable causará una excepción de tipo **EXCEPTION_NONCONTINUABLE_EXCEPTION** (0xc0000025))

2 - the stack is unwinding - no tratar de reparar

NestedExceptionRecord:

Es un puntero a otra estructura **EXCEPTION_RECORD**, si el manejador ha causado otra excepción.

ExceptionAddress:

Es la dirección donde se produjo la excepción..

NumberParameters

Especifica el número de parámetros asociados a la excepción. Este es el numero de elementos definidos en el array ExceptionInformation.

AdditionalData (o ExceptionInformation)

Un array de argumentos adicionales (4 bytes cada elemento) que describen la excepción. La API **RaiseException** puede especificar este array de elementos aunque para la mayoría de excepciones estos argumentos no están definidos, tan solo para **EXCEPTION_ACCESS_VIOLATION**.

Para este último caso tenemos los siguientes argumentos en el array:

1° dword	Flag indicando el tipo de operación que causó la violación de acceso.	Si es 0 → Intentó leer datos inaccesibles Si es 1 → Intentó escribir datos inaccesibles
2° dword	Indica la dirección virtual de los datos inaccesibles	

Y aquí termina la estructura **EXCEPTION_RECORD**.

Estructura CONTEXT_RECORD

CONTEXT_RECORD +0	ContextFlags	
+4	Registro DR0	REGISTROS DE DEPURACIÓN
+8	Registro DR1	
+C	Registro DR2	
+10	Registro DR3	
+14	Registro DR6	
+18	Registro DR7	
+1C	ControlWord	ESTRUCTURA FLOATING_SAVE_AREA
+20	StatusWord	
+24	TagWord	
+28	ErrorSelector	
+30	DataOffset	
+34	DataSelector	
+38	FP registers x 8 (10 bytes cada uno)	
+88	CrONpxState	
+8C	Registro GS	REGISTROS DE SEGMENTACIÓN
+90	Registro FS	
+94	Registro ES	
+98	Registro DS	

+9C	Registro EDI	REGISTROS
+A0	Registro ESI	
+A4	Registro EBX	
+A8	Registro EDX	
+AC	Registro ECX	
+B0	Registro EAX	
+B4	Registro EBP	REGISTROS DE CONTROL
+B8	Registro EIP	
+BC	Registro CS	
+C0	Registro EFlags	
+C4	Registro ESP	
+C8	Registro SS	

Como se puede ver, esta estructura da una buena información sobre la excepción que se produjo.

Podemos modificar cualquiera de estos valores pero según qué valores se modifiquen hay que cambiar también el valor de **ContextFlags** de este modo:

CONTEXT_CONTROL: Si se modificó alguno de los registros de control.

CONTEXT_INTEGER: Si se modificó alguno de los registros: EDI, ESI, EBX, EDX, ECX, EAX

CONTEXT_SEGMENTS: Si se modificó alguno de los registros de segmentación.

CONTEXT_FLOATING_POINT: Si se modificó algún dato en la estructura **_FLOATING_SAVE_AREA**.

La información enviada a los manejadores “por thread” (Tipo 2)

Como se explicó en este anterior punto al inicio del todo, dijimos que los manejadores de Tipo1 reciben un puntero a esta estructura **EXCEPTION_POINTERS**.

Y ahora bien, ¿qué ocurre cuando utilizamos los manejadores “por thread”? ¿Cómo se envía la información?

En un principio la encontré indigando por mi cuenta y de casualidad, así que busqué información y como dije en el punto anterior: Matt Pietrek en un artículo suyo, explica el mecanismo de las excepciones y toda esta información la mete en la pila.

Pues bien, vamos a buscarla.

Si nos detenemos justo en nuestro propio SEH, y miramos la pila observaremos lo siguiente que pongo en la tabla:

ESP+4	Puntero a la estructura EXCEPTION_RECORD
ESP+8	Puntero a nuestra estructura ERR
ESP+C	Puntero a la estructura CONTEXT_RECORD

Vamos a hacerlo con un ejemplo.

Nos vamos al OEP del programa de este tute *SEH.exe*. Veremos lo siguiente:

```
00401000 75 11000000 CALL 00401016
00401005 . B8 08000000 MOV EAX,8
0040100A . B8 08000000 MOV EAX,8
0040100F . 6A 00 PUSH 0
00401011 . E8 32000000 CALL 00401048
00401016 . 68 42104000 PUSH 401042
0040101B . 64:FF35 0000 PUSH DWORD PTR FS:[0]
00401022 . 64:8925 0000 MOV DWORD PTR FS:[0],ESP
00401029 . 33C0 XOR EAX,EAX
0040102B . F7F0 DIV EAX
0040102D . BB 03000000 MOV EBX,3
00401032 . BB 02000000 MOV EBX,2
00401037 . 64:8F05 0000 POP DWORD PTR FS:[0]
0040103E . 83C4 04 ADD ESP,4
00401041 . C3 RETN
00401042 . B8 00000000 MOV EAX,0
00401047 . C3 RETN
00401048 .- FF25 00204000 JMP NEAR DWORD PTR DS:[402000]
```

Ahora ya sabemos perfectamente su funcionamiento así que si observamos el código, sabemos que va a instalar un manejador de excepciones en la dirección **00401042** y es de tipo 2. Es muy sencillo. También se observa que voy a hacer una división por cero. Siguiendo lo que se acaba de explicar voy a poner un BP al inicio del manejador, en **00401042** y voy a pulsar F9 para que pare ahí.

```
00401041 . C3 RETN
00401042 . B8 00000000 MOV EAX,0 Structured exception handler
00401047 . C3 RETN
00401048 .- FF25 00204000 JMP NEAR DWORD PTR DS:[402000] kernel32.ExitProcess
```

Además ahora ya nos indica el Olly que se trata de un SEH.

Vamos a ver la pila:

```

0012FBF0 7C9137BF RETURN to ntdll.7C9137BF
0012FBF4 0012FCD8
0012FBF8 0012FFB8
0012FBFC 0012FCEC
0012FC00 0012FCAC
0012FC04 0012FFB8 Pointer to next SEH record
0012FC08 7C9137D8 SE handler
0012FC0C 0012FFB8

```

Voy a poner los datos para que salgan respecto a ESP para verlo mejor:

```

ESP => 7C9137BF RETURN to ntdll.7C9137BF
ESP+4 0012FCD8
ESP+8 0012FFB8
ESP+C 0012FCEC
ESP+10 0012FCAC
ESP+14 0012FFB8 Pointer to next SEH record
ESP+18 7C9137D8 SE handler
ESP+1C 0012FFB8

```

Y ahora ya es sencillo, sé que en **ESP+4** hay una estructura **EXCEPTION_RECORD**, voy a ella:

Address	Hex dump	ASCII
0012FCD8	94 00 00 C0 00 00 00 00	...L...
0012FCE0	2B 10 40 00	...+0...
0012FCE8	00 00 00 00 00 00 00 00	...?..
0012FCF0	00 00 00 00 00 00 00 00
0012FCF8	00 00 00 00 00 00 00 00
ExceptionCode 00 00 ExceptionAddress		
	FF FF	
0012FD10	FF FF FF FF
0012FD18	00 00 31 00 00 00 00 00	...1....
0012FD20	00 00 FF FF 6F 00 6C 00	...o.l...
0012FD28	04 01 05 01 B0 BB 69 00	...i...
0012FD30	6E 00 69 00 00 00 00 00	n.i....
0012FD38	00 00 00 00 00 00 00 00

He señalado solamente dos datos:

ExceptionCode = 0C000094 que si miramos la tabla que puse corresponde a **EXCEPTION_INTEGER_DIVIDE_BY_ZERO**

ExceptionAddress = 0040102B que efectivamente corresponde a la dirección en la cual se produjo la excepción.

Sigamos, sé que en **ESP+8** hay un puntero a nuestra estructura **ERR**:

```

0012FFB8 0012FFE0 Pointer to next SEH record
0012FFBC 00401042 SE handler

```

Y por último, hacia la estructura más completa y con más información, en **ESP+C** tenemos un puntero a una estructura **CONTEXT_RECORD**.

Por ej., voy a buscar un dato: la dirección donde se produjo la excepción que corresponde al valor del registro **EIP**. En la tabla que puse el registro **EIP** se encontraba en **+B8** así que tiene que estar en **0012FDA4**. Vamos a comprobarlo:

Address	Hex dump
0012FCEC	3F 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FCFC	00 00 00 00 00 00 00 00 00 00 00 00 7F 02 FF FF
0012FD0C	00 00 FF FF FF FF FF FF 00 00 00 00 00 00 31 00
0012FD1C	00 00 00 00 00 00 FF FF 6F 00 6C 00 04 01 05 01
0012FD2C	00 BB 69 00 6E 00 69 00 00 00 00 00 00 00 00 00
0012FD3C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FD4C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FD5C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FD6C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FD7C	3B 00 00 00 23 00 00 00 23 00 00 00 38 07 92 7C
0012FD8C	FF FF FF FF 00 F0 FD 7F 04 5B 91 7C B0 FF 12 00
0012FD9C	00 00 00 00 F0 FF 12 00 2B 10 40 00 18 00 00 00
0012FDA4	46 02 01 00 B8 FF 12 00 23 00 00 00 7F 02 00 00
0012FDBC	00 00 31 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FDCC	00 00 FF FF 80 1F 00 00 00 00 00 00 00 00 00 00
0012FDDC	04 01 05 01 B0 BB 00 00 00 00 00 00 00 00 00 00
0012FDEC	69 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FDFC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FE0C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FE1C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FE2C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Anexo

Para los programadores en MASM, encontré y es la que utilizo, una macro bien sencilla para establecer o no nuestras propias SEH. Pongo el código junto a este tute.

Voy a explicarla al detalle.

Antes de nada decir que el original fue escrito por **yoda** y que yo lo he adaptado a mi forma para que todo el mundo lo pueda entender.

Archivo SEH.inc

Lo primero de todo, voy a utilizar una estructura propia muy sencilla que la voy a usar para guardar los valores de los registros **ESP, EBP, EIP**.

```
sSEH STRUCT
    OriginalEsp    dd ?
    OriginalEbp    dd ?
    GuardarEip     dd ?
sSEH ENDS
```

La idea para este ejemplo es la siguiente:

Si queremos proteger un determinado código, antes del mismo instalaremos nuestro SEH. Si se produjera una excepción iríamos a nuestro SEH y de ahí al final del código que queremos proteger. Por lo tanto al final del código, pondremos una etiqueta.

De eso se trata, antes del código llamaremos a la macro que nos creará nuestra SEH y le pasaremos como dato la etiqueta donde queremos que vaya tras la excepción.

Al finalizar el código protegido, deberemos desinstalar nuestro SEH, llamando a la macro.

Es muy sencillo sería algo así:

```
Código protegido {
    InstalarSEH <Offset Final_del_código>
    .....
    .....
    .....
    Final_del_código:
    DesinstalarSEH
}
```

Como se observa, con la ayuda de la macro lo que vamos a conseguir es que nos olvidemos de nuestro manejador, y de este modo hacer las cosas mucho más sencillas.

Para hacer todo esto necesitaremos realizar lo siguiente:

1 MACRO: para instalar nuestro SEH

1 MACRO: para desinstalar nuestro SEH

1 PROCEDIMIENTO: para ejecutar nuestro manejador.

Vamos por el más fácil

MACRO para desinstalar nuestro SEH

Como expliqué en este tutorial al hablar del Tipo2 de manejadores, para desinstalarlos podemos hacer lo siguiente:

```
POP DWORD PTR FS:[0]
ADD ESP,4
```

Pues ya tenemos nuestra MACRO para desinstalar:

```
DesinstalarSEH MACRO
    POP DWORD PTR FS:[0]
    ADD ESP,4
ENDM
```

MACRO para instalar nuestro SEH

Una de las formas para instalar un nuevo SEH, como ya sabemos es la siguiente:

```
PUSH Offset ManejadorSEH
PUSH FS:[0]
MOV FS:[0],ESP
```

Lo primero cuando trabajamos en MASM es:

```
ASSUME FS: NOTHING
```

Vamos a empezar con la macro desde el principio:

```
InstalarSEH MACRO Dirección_a_continuar
    ASSUME FS: NOTHING

    ;La estructura SEH no está todavía definida, por lo tanto, para
    ;poder utilizarla habrá que definirla previamente, así:

    IFNDEF Estructura_sSEH
        Estructura_sSEH EQU 1
    .data
    SEH sSEH <>
    ENDIF

    ;Ahora sólo nos queda poner el código muy sencillo que iré
    ;explicando:
    .code
    MOV SEH.GuardarEip, Dirección_a_continuar ;guardamos en
    SEH.GuardarEip, la dirección en la que el manejador deberá
    continuar tras la excepción.
    MOV SEH.OriginalEbp, EBP ;guardamos el valor del registro
    EBP.
    PUSH OFFSET ManejadorSEH
    PUSH FS:[0]
    MOV SEH.OriginalEsp, ESP ;guardamos el valor original de ESP
    MOV FS:[0], ESP
ENDM
```

PROCEDIMIENTO de nuestro manejador

En este punto recordaremos unas de las cosas que puse (y no es mio):

-“En un artículo Matt Pietrek explica el mecanismo de las excepciones en Windows. Cuando Windows detecta una excepción mira a ver si hay manejadores buscando en el **TIB** y si es así, mete en la pila cuatro punteros:

ExceptionRecord

EstablisherFrame

ContextRecord

DispatcherContext”

Por este motivo, vamos a llamar al procedimiento tal que así:

ManejadorSEH **PROC C** pExcept:**DWORD**,pFrame:**DWORD**,pContext:**DWORD**,pDispatch:**DWORD**

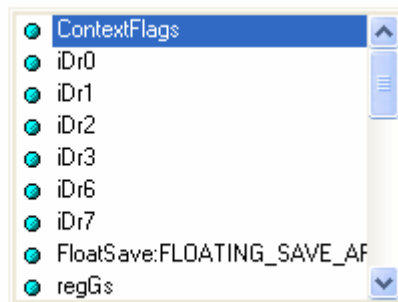
Además otra cosa importante, que ya veréis si lo hacéis desde RadASM la información que da. Haremos lo siguiente:

ASSUME EAX : PTR CONTEXT

De este modo haremos que eax sea un puntero a una estructura reconocida como:

CONTEXT_RECORD, que ya ha sido explicada.

De este modo, si por ejemplo utilizamos RadASM, veremos que si ponemos [EAX] y ponemos un punto, nos aparece toda la estructura **CONTEXT_RECORD**, muy útil:



MOV [EAX] .

Voy a poner ya todo el código:

ManejadorSEH **PROC C** pExcept:**DWORD**,pFrame:**DWORD**,pContext:**DWORD**,pDispatch:**DWORD**

MOV EAX, pContext ;EAX apuntará a una estructura CONTEXT_RECORD
ASSUME EAX : PTR CONTEXT ;ya explicado

PUSH SEH.GuardarEip ;pone en la pila la dirección a continuar.
POP [EAX].regEip ;pone la dirección a continuar en la estructura CONTEXT_RECORD, modificando el registro EIP.
PUSH SEH.OriginalEsp
POP [EAX].regEsp ;lo mismo con el registro ESP
PUSH SEH.OriginalEbp
POP [EAX].regEbp ;lo mismo con el registro EBP

MOV EAX, ExceptionContinueExecution
RET

ManejadorSEH **ENDP**

Como se ha visto, el manejador lo que hace es lo siguiente:

- En la estructura **CONTEXT_RECORD**, modifica el valor del registro EIP y lo pone igual al valor de la etiqueta **Dirección_a_continuar**, por lo tanto ahí continuará tras la excepción.
- Modifica la pila (ESP) y el registro (EBP) devolviéndolos a su estado original antes de la excepción.

Conclusiones

Sé que sobre el tema de las excepciones se puede hablar muchísimo más, pero bueno de momento para este tute no es mi intención, ya veré si continúo más adelante. Mi idea era recopilar toda la información que pude, leer, traducir(a veces de locos), probar, analizar y después de todo juntar lo más importante y escribirlo todo en este tute. Creo que es un tute que puede ser muy interesante.

Es posible que pueda haber algún error o alguna excepción ☺. Ya sabéis dónde encontrarme.

Un tute dedicado a **Solid**, gran persona y cracker.