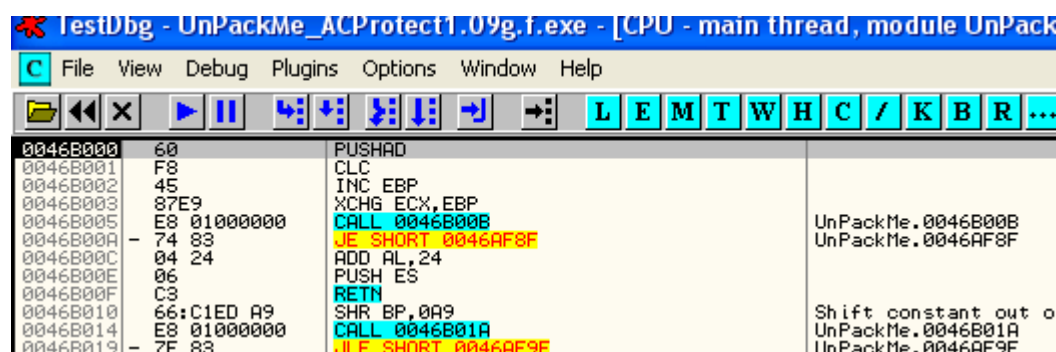


## INTRODUCCION AL CRACKING CON OLLYDBG PARTE 42

Bueno iremos introduciéndonos lentamente en packers mas difíciles y verán que los métodos que utilizaremos son muy variables según el caso, por lo cual hay que tener la mochila bien llena de trucos y practicar cada vez mas, el que nos ocupa ahora es el ACPROTECT 1.09 f el unpackme con todas las protecciones habilitadas.

Dicho unpackme tiene un poco de todo, por lo cual iremos despacio y analizando cada parte de la protección en varias entregas, no nos alcanzara con una sola parte para poder ir despacio y claramente.

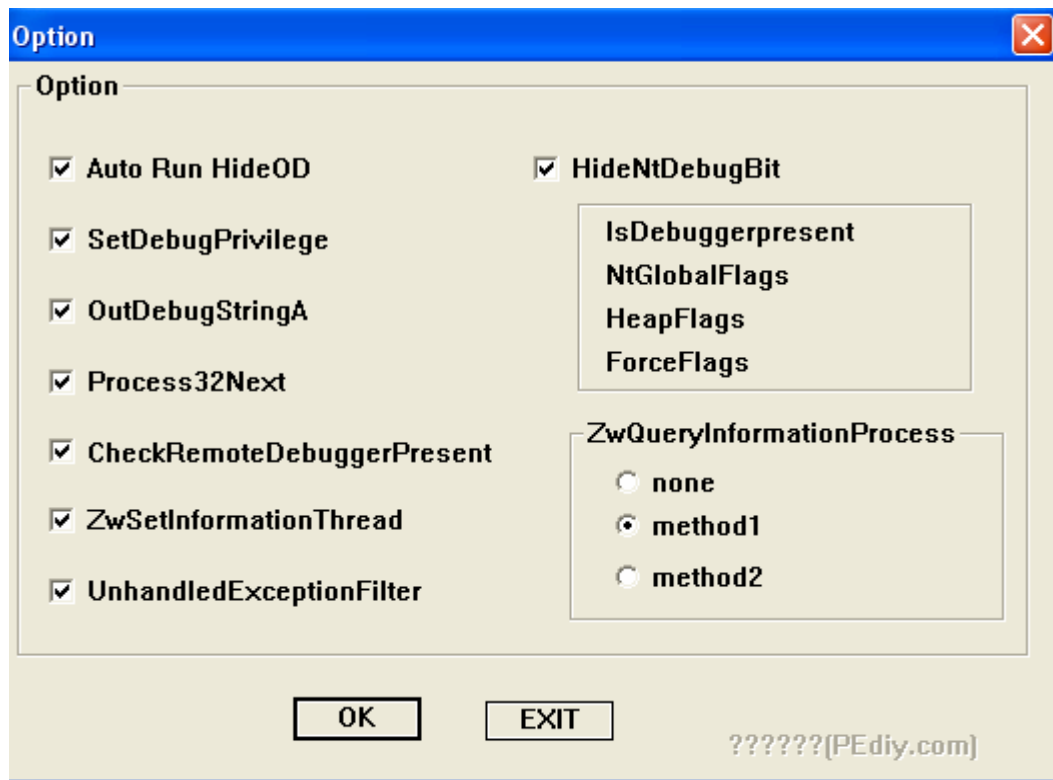
El unpackme esta adjunto a este tutorial, así que no hay problema, abramoslo en nuestro OLLYDBG parcheado y con los plugins para ocultarlo.



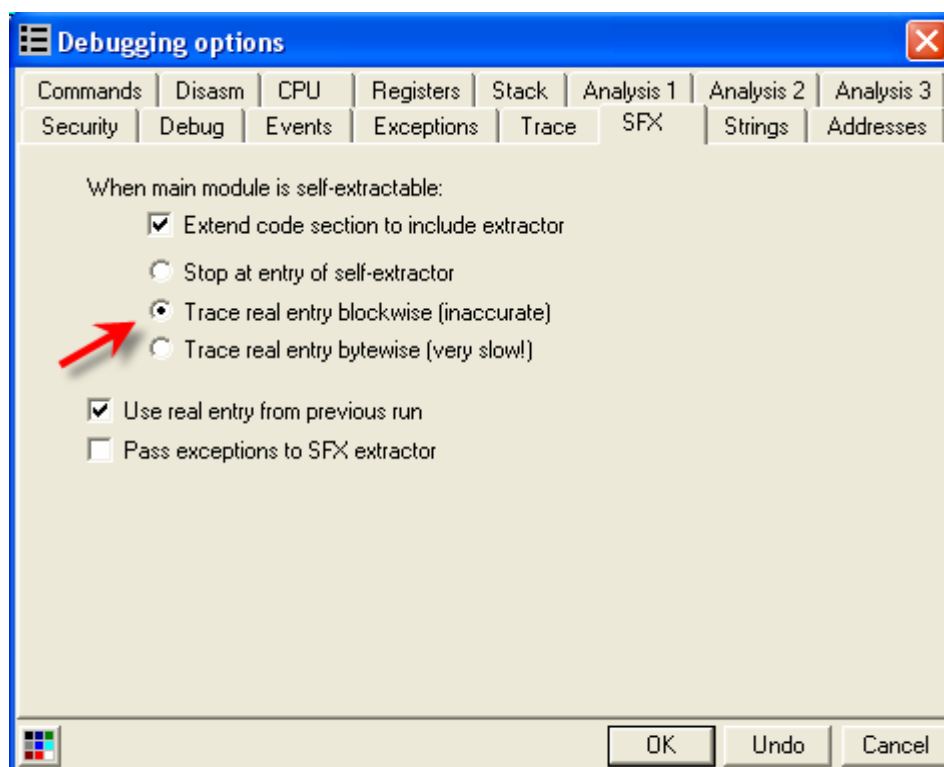
Allí esta abierto en OLLYDBG, y ya que tiene de tiene de todas las protecciones, chequeemos si corre en OLLYDBG, ya que si no lo hace debemos antes que nada ver como solucionar eso.



Pues aquí en mi parcheado4 ,que adjunte en la parte anterior,con todas las excepciones marcadas y usando el plugin HideOD 0.12 con la configuración que vemos abajo, no tiene ningún problema y corre perfectamente, recuerdo que este packer verificaba mediante las apis Process32Next cual era el proceso que lo había abierto y si descubría que no se habia abierto por medio de una ejecución de doble click, o sea si lo había cargado un loader o un debugger, no corría, pero el plugin HideOD, trae protección contra ese truco, por lo cual el antidebugger del mismo queda completamente anulado sin ningun dolor de cabeza.

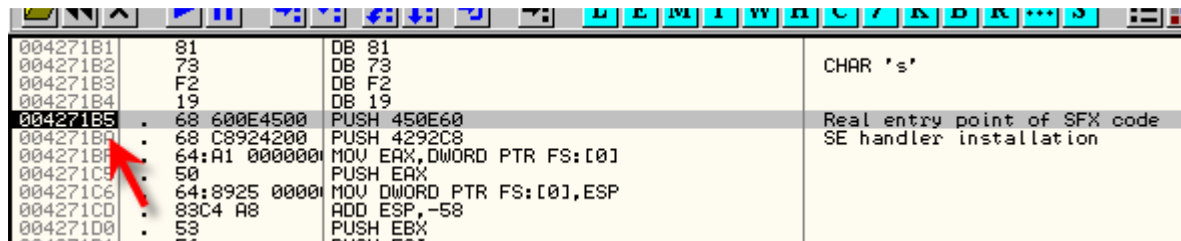


Bueno así que no debemos preocuparnos por eso, por lo cual nos concentraremos en llegar al OEP.



De los varios métodos que vimos tanto usar el OLLYDBG para hallar OEPs, como utilizar el buscador de OEPs incorporado que trae el OLLYDBG va perfecto, así que vamos a la

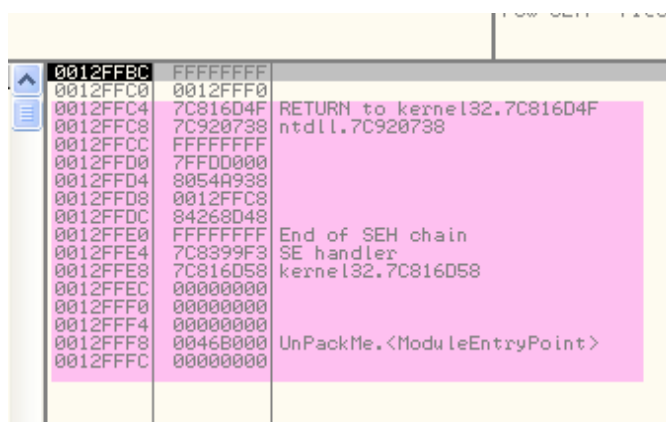
pestaña SFX, y ponemos la tilde en TRACE REAL ENTRY BLOCKWISE y aceptamos y reiniciamos el OLLYDBG el cual luego de un rato parara aquí.



004271B1	81	DB 81	
004271B2	73	DB 73	
004271B3	F2	DB F2	CHAR 's'
004271B4	19	DB 19	
004271B5	68 600E4500	PUSH 450E60	Real entry point of SFX code SE handler installation
004271B9	68 C8924200	PUSH 4292C8	
004271BF	64:A1 000000	MOV EAX,DWORD PTR FS:[0]	
004271C9	50	PUSH EAX	
004271C6	64:8925 0000	MOV DWORD PTR FS:[0],ESP	
004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	

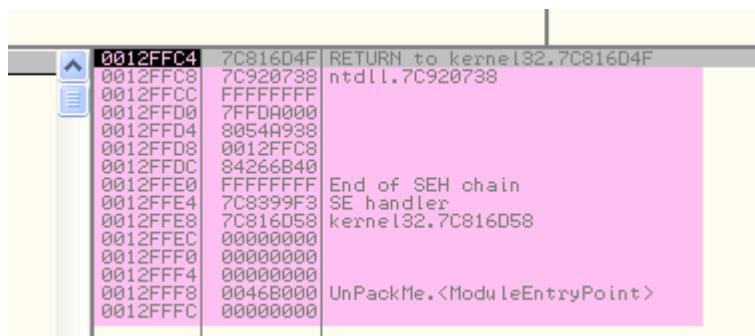
Ahora la cuestión es, es este el OEP real o tiene stolen bytes.

Miremos un poco el stack en este momento.



0012FFBC	FFFFFFFF	
0012FFC0	0012FFF0	
0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFD0000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	84268D48	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	0046B000	UnPackMe.<ModuleEntryPoint>
0012FFFC	00000000	

Si quitamos la tilde de buscar el OEP y reiniciamos



0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFD0000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	84268D48	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	0046B000	UnPackMe.<ModuleEntryPoint>
0012FFFC	00000000	

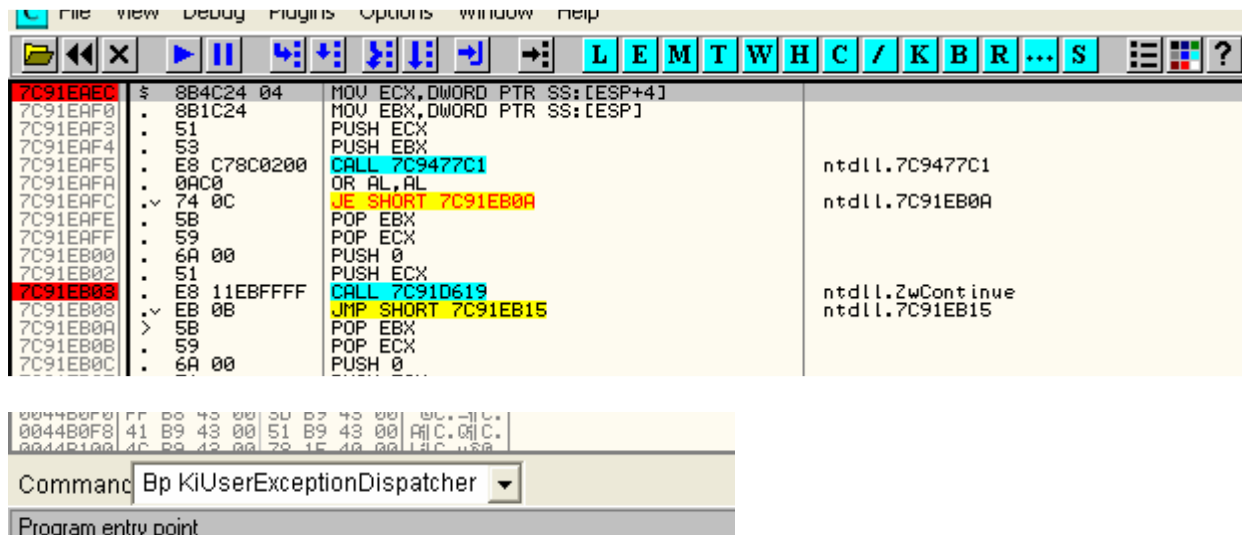
Vemos que el stack cuando esta detenido en el OEP, tiene dos valores mas de los que tenia cuando arranca en el inicio, y salvo raros casos eso es un indicio de que hay stolen bytes, así que nos prepararemos para hallarlos.

Pero antes debemos corregir el script para evitar que nos anulen los Hardware Breakpoints que les enseñe en las partes anteriores ya que tiene un pequeño bug y lo necesitaremos en este y en muchos packers mas.

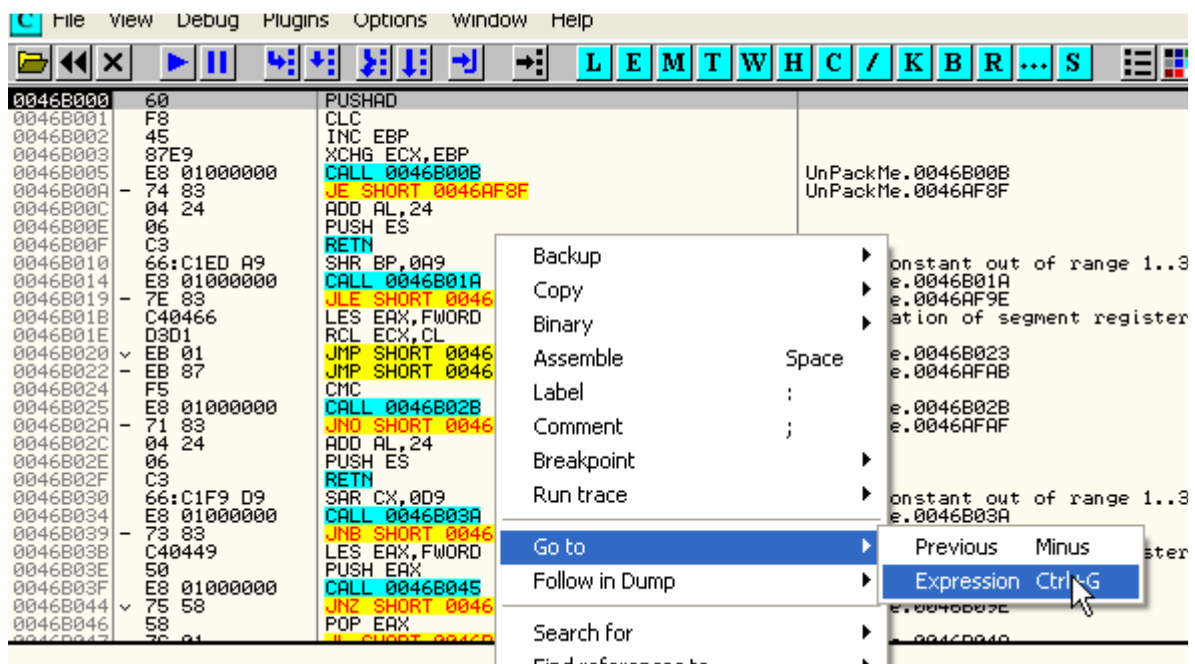
Aquí lo podemos probar ya que si ahora queremos parar en el OEP que ya conocemos y ponemos un HBP ON EXECUTION en el, vemos que no para ya que el programa lo deshabilita, inclusive usando el script que les di, tampoco para, ya que este tiene un bug.

Antes que nada explicare donde estaba el error, para que se entienda como quedara el script y porque hubo que modificarlo.

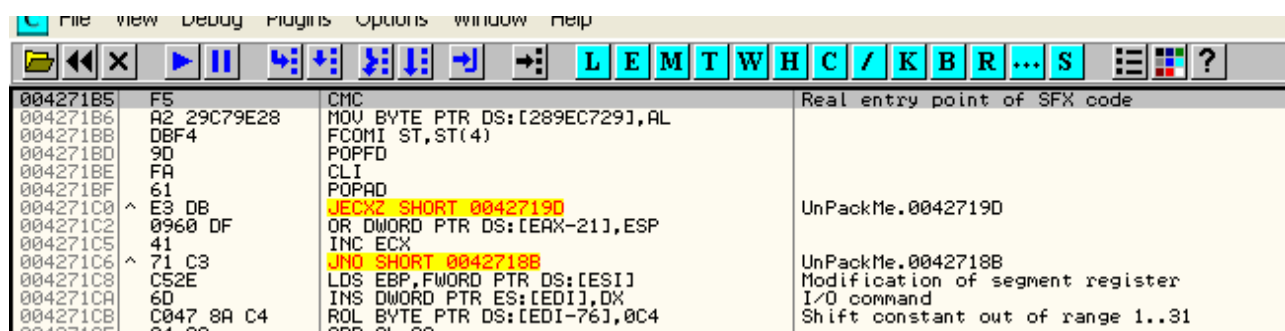
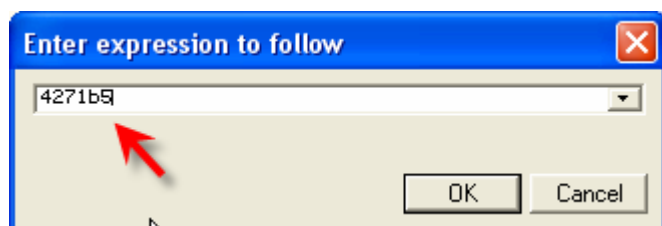
Arranco nuevamente el unpackme en el OLLYDBG, y le coloco los dos BP tanto en KiUserExceptionDispatcher como en el CALL que nos lleva a ZwContinue como se ve en la imagen.



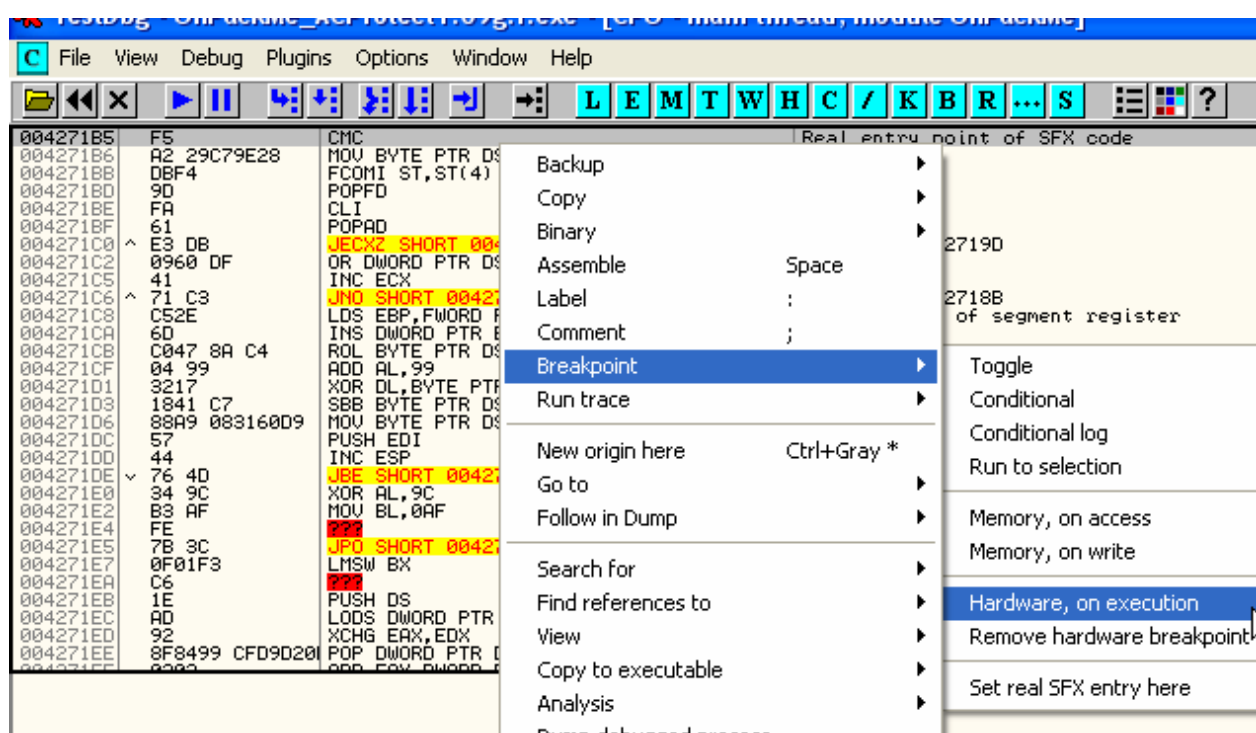
Ahora colocare un Hardware BP on execution en la direcci3n del OEP, para probar si lo borra o no, podr3a ser cualquier otra direcci3n, pero como sabemos que por all3 pasara seguro y llegara despu3s de haber hecho todas las trapisondas del packer, pues que mejor que usar esa direcci3n para probar el script.



All3 en el inicio, hago GOTO EXPRESSION y coloco la direcci3n del OEP que era 4271B5



Allí esta, le colocaremos el HBP ON EXECUTION.



Marco la primera linea y le coloco el HARDWARE BPX ON EXECUTION.

Ahora doy RUN.

Address	Disassembly	Comment
7C91E9EC	MOV ECX, DWORD PTR SS:[ESP+4]	
7C91E9F0	MOV EBX, DWORD PTR SS:[ESP]	
7C91E9F3	PUSH ECX	
7C91E9F4	PUSH EBX	
7C91E9F5	CALL 7C9477C1	ntdll.7C9477C1
7C91E9FA	OR AL, AL	
7C91E9FC	JE SHORT 7C91EB0A	ntdll.7C91EB0A
7C91E9FE	POP EBX	
7C91EA00	POP ECX	
7C91EA02	PUSH 0	
7C91EA04	PUSH ECX	
7C91EA06	CALL 7C91D619	ntdll.ZwContinue
7C91EA08	JMP SHORT 7C91EB15	ntdll.7C91EB15
7C91EA0A	POP EBX	
7C91EA0C	POP ECX	
7C91EA0E	PUSH 0	

Para en el BP y ahora miro el stack

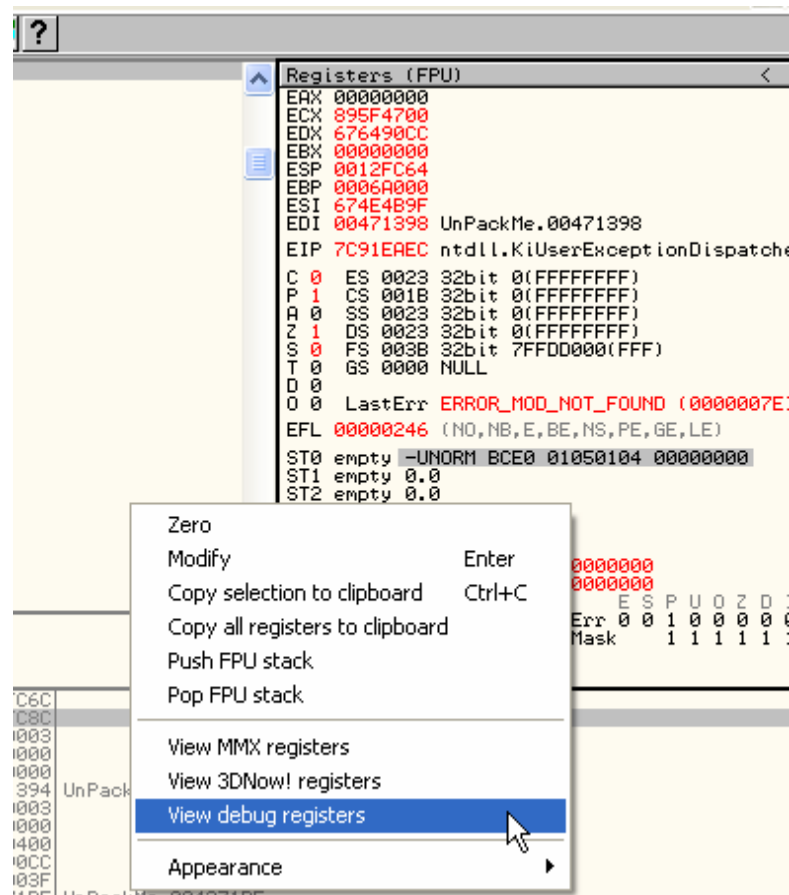
Address	Hex	Comment
0012FC64	0012FC6C	
0012FC68	0012FC8C	
0012FC6C	00000000	
0012FC70	00000000	
0012FC74	00000000	
0012FC78	00471394	UnPackMe.00471394
0012FC7C	00000003	
0012FC80	00000000	
0012FC84	00000400	
0012FC88	676490CC	
0012FC8C	0001003F	
0012FC90	004271C6	UnPackMe.004271C6
0012FC94	004271B5	UnPackMe.004271B5
0012FC98	00000000	
0012FC9C	00000000	
0012FCA0	FFFFFFF0	
0012FCA4	00000405	
0012FCA8	FFFF027F	
0012FCAC	FFFF4020	
0012FCB0	FFFFFFFF	
0012FCB4	02734312	
0012FCB8	0609001B	
0012FCBC	02CEBA60	
0012FCC0	FFFF0023	
0012FCC4	00000000	

**PUNTERO AL CONTEXT**

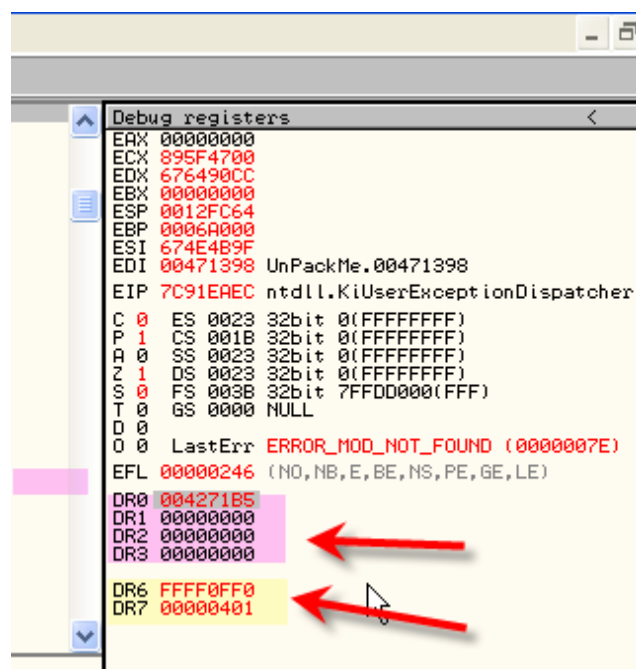
Cuando para en KiUserExceptionDispatcher el puntero al context esta en el segundo lugar del stack, miremos el CONTEXT en el dump.

Address	Hex	dump	ASCII
0012FC8C	3F 00 01 00 B5 71 42 00		?.0.âqB.
0012FC94	00 00 00 00 00 00 00 00		.....
0012FC9C	00 00 00 00 F0 0F FF FF		....-*
0012FCA4	01 04 00 00 7F 02 FF FF		@+..â0
0012FCAC	20 40 FF FF FF FF FF FF		@
0012FCB4	12 43 73 02 1B 00 09 06		¢Cs0+.J+
0012FCBC	60 BA CE 02 23 00 FF FF		'  #0#.
0012FCC4	00 00 00 00 04 01 05 01		...+0+0
0012FCCC	E0 BC 00 00 00 00 00 00		0^.....
0012FCD4	00 00 00 00 00 00 00 00		.....
0012FCDC	00 00 00 00 00 00 00 00		.....
0012FCE4	00 00 00 00 00 00 00 00		.....
0012FCEC	00 00 00 00 00 00 00 00		.....
0012FCF4	00 00 00 00 00 00 00 00		.....
0012FCFC	00 00 00 00 00 00 00 00		.....
0012FD04	00 00 00 80 FF 3F 00 00		...Ç ?
0012FD0C	00 00 00 00 00 80 FF 3F		...Ç ?
0012FD14	00 00 00 00 00 00 00 00		.....
0012FD1C	3B 00 00 00 23 00 00 00		...#...
0012FD24	23 00 00 00 98 13 47 00		#...ÿ!!G.
0012FD2C	9F 4B 4E 67 00 00 00 00		fKHg....
0012FD34	CC 90 64 67 00 47 5F 89		lFEdg.G_ê
0012FD3C	00 00 00 00 00 A0 06 00		....â+.
0012FD44	94 13 47 00 1B 00 00 00		ô!!G.+...
0012FD4C	46 02 00 00 58 FF 12 00		F0..X +.
0012FD54	23 00 00 00 7F 02 20 40		#...â0 @
0012FD5C	00 00 00 00 00 00 00 00		.....
0012FD64	00 00 FF FF 80 1F 00 00		...Ç▼..
0012FD74	00 00 00 00 00 00 00 00		.....
0012FD7C	04 01 05 01 E0 BC 00 00		+0+00^..
0012FD84	00 00 00 00 00 00 00 00		.....
0012FD8C	00 00 00 00 00 00 00 00		.....

Por supuesto no explicare todos los valores de la estructura CONTEXT, pero si los que utilizaremos por ahora.



En los registros, cambio la vista para ver los DEBUG REGISTERS, si no aparece la opción para cambiar es porque ya esta en la vista correcta.



Esos son los famosos DEBUG REGISTERS van desde Dr0 a dr7 , los correspondientes a los HBP son Dr0, Dr1, Dr2 y Dr3.

Cuando el programa para en un BP, en ese momento los DEBUG REGISTERS del OLLYDBG se actualizan con los valores de los HARDWARE BPX que estén vigentes en ese momento, como vemos allí, paro en un BP y nos muestra los HBP actuales, en Dr0 esta 4271B5 que es el valor del hardware BPX que colocamos, Dr4 y Dr5 no se utilizan, el Dr6 no nos interesa por ahora y el Dr7 indicara el tipo de hardware bpx que esta colocado segun una tablita que veremos mas adelante, lo importante es que vemos que Dr0 tiene nuestro HARDWARE BPX, ahora miremos el CONTEXT.

ntdll.KiUserApcDispatcher+2C

Address	Hex dump	ASCII
0012FC8C	3F 00 01 00 B5 71 42 00	... ÁqB.
0012FC94	00 00 00 00 00 00 00 00	.....
0012FC9C	00 00 00 00 F0 0F FF FF	...*...
0012FCA4	01 04 00 00 7F 02 FF FF	@...Δ@
0012FCAC	20 40 FF FF FF FF FF FF	@.....
0012FCB4	12 40 73 02 18 00 09 06	+Cs@+.~
0012FCBC	60 BF CE 02 23 00 FF FF	'  f@#.
0012FCC4	00 00 00 00 04 01 05 01	....@#@
0012FCCC	E0 B0 00 00 00 00 00 00	0.....
0012FCD4	00 00 00 00 00 00 00 00	.....
0012FCDC	00 00 00 00 00 00 00 00	.....
0012FCE4	00 00 00 00 00 00 00 00	.....
0012FCEC	00 00 00 00 00 00 00 00	.....
0012FCF4	00 00 00 00 00 00 00 00	.....
0012FCFC	00 00 00 00 00 00 00 00	.....
0012FD04	00 00 00 80 FF 3F 00 00	...Ç ?..
0012FD0C	00 00 00 00 00 80 FF 3F	...Ç ?
0012FD14	00 00 00 00 00 00 00 00	.....
0012FD1C	3B 00 00 00 23 00 00 00	;...#...
0012FD24	23 00 00 00 98 13 47 00	#...y!G.
0012FD2C	9F 4B 4E 67 00 00 00 00	fKNg....
0012FD34	CC 90 64 67 00 47 5F 89	lfedg.G_è
0012FD3C	00 00 00 00 00 A0 06 00	...@.
0012FD44	94 13 47 00 1B 00 00 00	ö!G.+...
0012FD4C	46 02 00 00 58 FF 12 00	F@.X@.
0012FD54	23 00 00 00 7F 02 20 40	#...Δ@ @
0012FD5C	00 00 00 00 00 00 00 00	.....
0012FD64	00 00 00 00 00 00 00 00	.....
0012FD6C	00 00 FF FF 80 1F 00 00	.. ÇV..
0012FD74	00 00 00 00 00 00 00 00	.....
0012FD7C	04 01 05 01 E0 BC 00 00	@#000...
0012FD84	00 00 00 00 00 00 00 00	.....
0012FD8C	00 00 00 00 00 00 00 00	.....

Ahí vemos la posición en el context de los registros Dr0 a Dr3, en amarillo esta la dirección 4271b5 que corresponde al HBP que colocamos y que esta activo, los otros tres en rosado están a cero pues solo colocamos un HBP los otros están vacíos.

O sea cuando el programa llega al manejador de excepciones como dijimos, pondrá a cero estos valores del CONTEXT, si nos fijamos, damos RUN para que llegue al 2do BP.



ntdll.KiUserApcDispatcher+43			
Address	Hex dump	ASCII	
0012FC8C	3F 00 01 00 00 00 00 00	?..@.....	
0012FC94	00 00 00 00 00 00 00 00	.....	
0012FC9C	00 00 00 00 F0 0F FF FF	.....-*	
0012FCA4	01 04 00 00 7F 02 FF FF	@...Δ@	
0012FCAC	20 40 FF FF FF FF FF FF	@.....	
0012FCB4	12 43 73 02 1B 00 D9 06	¢C\$@+...@	
0012FCBC	60 BA CE 02 23 00 FF FF	'  ir@#.	
0012FCC4	00 00 00 00 04 01 05 01	....@#@	
0012FCCC	E0 BC 00 00 00 00 00 00	0.....	
0012FCD4	00 00 00 00 00 00 00 00	.....	
0012FCDC	00 00 00 00 00 00 00 00	.....	
0012FCE4	00 00 00 00 00 00 00 00	.....	
0012FCEC	00 00 00 00 00 00 00 00	.....	
0012FCF4	00 00 00 00 00 00 00 00	.....	
0012FCFC	00 00 00 00 00 00 00 00	.....	
0012FD04	00 00 00 80 FF 3F 00 00	...Ç ?..	
0012FD0C	00 00 00 00 00 80 FF 3F	...Ç ?	
0012FD14	00 00 00 00 00 00 00 00	.....	
0012FD1C	3B 00 00 00 23 00 00 00	...#...	
0012FD24	23 00 00 00 98 13 47 00	#...ÿ!!G.	
0012FD2C	9F 4B 4E 67 00 00 00 00	fKNg....	
0012FD34	CC 90 64 67 00 47 5F 89	lrfdg.G_e	
0012FD3C	00 00 00 00 00 A0 06 00	.....â@.	
0012FD44	95 13 47 00 1B 00 00 00	o!!G.+...	
0012FD4C	46 02 00 00 58 FF 12 00	F@..X @.	
0012FD54	23 00 00 00 7F 02 20 40	#...Δ@ @	
0012FD5C	00 00 00 00 00 00 00 00	.....	
0012FD64	00 00 00 00 00 00 00 00	.....	

Vemos como al pasar por el manejador de excepciones puso a cero Dr0 y el error mio fue en el script, volver a restaurarlo aquí, ya que si recuerdan al llegar a este 2do BP, el script iba a restaurarlos, pero he aquí el error ya que lo que borro Dr0 es solo la preparación ya que coloca el CONTEXT en la forma que QUIERE que quede, la verdadera actualización de los HBP ocurre al salir de ZwContinue y entrar a RING0, antes de volver al programa, o sea que por mas que yo aquí vuelva a colocar los HBP en OLLYDBG, no sirve, pues estando preparados a cero en el CONTEXT, al salir de ZwContinue lo que hará sera borrarlos sin mas.

Entonces aquí hay dos posibilidades, o bien cuando para el primer BP guardar los valores que leo en el CONTEXT de los DEBUG REGISTERS y cuando llega al 2do BP, volverlos a copiar al CONTEXT evitando lo que ha hecho de ponerlos a cero, poniendo en Dr0..Dr3, los valores que tenían, los cuales actualizara normalmente al volver al programa.

Otra opción es hallar en el CONTEXT la dirección de retorno al programa, poner un BP allí, y ahí si cuando para, restaurar los HBP ya que ya fueron borrados y ya volvió al programa, así que es como ponerlos nuevamente, y luego borrar el BP de la dirección de retorno.

Esta segunda opción me pareció mas breve, pero donde esta ubicada en el CONTEXT la DIRECCION DE RETORNO al programa?

ntdll.KiUserApcDispatcher+43			
Address	Hex dump	ASCII	
0012FC8C	3F 00 01 00 00 00 00 00	?..0.....	
0012FC94	00 00 00 00 00 00 00 00	.....	
0012FC9C	00 00 00 00 F0 0F FF FF	....-*	
0012FCA4	01 04 00 00 7F 02 FF FF	@...00	
0012FCAC	20 40 FF FF FF FF FF FF	@.....	
0012FCB4	12 43 73 02 1B 00 09 06	+Cs@+.J	
0012FCBC	60 BA CE 02 23 00 FF FF	*  ir@#.	
0012FCC4	00 00 00 00 04 01 05 01	....@#0	
0012FCCC	E0 BC 00 00 00 00 00 00	0.....	
0012FCD4	00 00 00 00 00 00 00 00	.....	
0012FCD4	00 00 00 00 00 00 00 00	.....	
0012FCE4	00 00 00 00 00 00 00 00	.....	
0012FCEC	00 00 00 00 00 00 00 00	.....	
0012FCF4	00 00 00 00 00 00 00 00	.....	
0012FCFC	00 00 00 00 00 00 00 00	.....	
0012FD04	00 00 00 80 FF 3F 00 00	...C ?..	
0012FD0C	00 00 00 00 00 80 FF 3F	....C ?	
0012FD14	00 00 00 00 00 00 00 00	.....	
0012FD1C	3B 00 00 00 23 00 00 00	;...#...	
0012FD24	23 00 00 00 98 13 47 00	#...y!!G.	
0012FD2C	9F 4B 4E 67 00 00 00 00	fKNg....	
0012FD34	CC 90 64 67 00 47 5F 89	lfEdg.G_e	
0012FD3C	00 00 00 00 00 00 06 00	.....a.	
0012FD44	95 13 47 00 1B 00 00 00	0!!G.+...	
0012FD4C	46 02 00 00 58 FF 12 00	F@..X +.	
0012FD54	23 00 00 00 7F 02 20 40	#...00 @	
0012FD5C	00 00 00 00 00 00 00 00	.....	
0012FD64	00 00 00 00 00 00 00 00	.....	
0012FD6C	00 00 FF FF 80 1F 00 00	..Cv...	

Si a la dirección de inicio del CONTEXT le sumo 0B8 obtengo el puntero a la DIRECCION DE RETORNO, en este caso en mi maquina.

$$12FC8c + 0b8 = 12FD44$$

0012FD84	00 00 00 00 00 00 00 00	.....	
0012FD8C	00 00 00 00 00 00 00 00	.....	
Command	12FC8c + 0b8		0012FD44

O sea que 12FD44 en mi maquina apunta a la dirección de retorno al programa, si en el script coloco un BP allí, cuando pare ya habrá vuelto al programa, luego de borrar definitivamente los HBP y podre volverlos a colocar, veamos como quedo el script.

```
var aux
inicio:
```

```
bphws 4271b5, "x"
```

```
trabajo:
```

```
eob pirulo
run
```

```
pirulo:
log eip
cmp eip, 7c91eaec
je quitar
cmp eip, 7c91eb03
je restaurar
cmp eip,aux
je restaurar2
```

```
jmp final
```

```
quitar:  
bphwc 4271b5  
jmp trabajo
```

```
restaurar:  
mov aux,esp  
mov aux,[aux]  
add aux,0b8  
mov aux,[aux]  
log aux  
bp aux  
jmp inicio
```

```
restaurar2:  
bc aux  
jmp inicio
```

```
final:  
MSGYN "Continuar?"  
cmp $RESULT,1  
je inicio  
ret
```

-----  
-----

Estrictamente es el mismo script que antes veremos que cambia, al inicio

### **var aux**

var es el comando para declarar una variable, declaro la variable aux, que me serviré de auxiliar para guardar y calcular la dirección de retorno al programa desde la excepción.

```
restaurar:  
mov aux,esp  
mov aux,[aux]  
add aux,0b8  
mov aux,[aux]  
log aux  
bp aux  
jmp inicio
```

Allí vemos la parte donde antes se restauraban los HBPs, ahora paso a aux, el valor de ESP.

```
mov aux,[aux]
```

Lo que hace es, ya que aux apunta a ESP, lo que necesitamos es el contenido de aux ya que en el momento que para en el CALL, el contenido de ESP apunta al inicio de la estructura context, o sea que luego de esta sentencia, aux queda con la dirección de inicio del context en mi caso 12fc8c, luego le sumamos 0b8 con lo cual obtendremos 12fd44 o en sus maquinas el puntero a la dirección de retorno y finalmente

```
mov aux,[aux]
```

Mueve a aux su propio contenido, quedando la dirección de retorno finalmente en aux y en la linea siguiente Bp aux, coloca el Bp en la dirección de retorno al volver de la excepción y luego vuelve a correr el programa y finalmente

```
pirulo:  
log eip  
cmp eip, 7c91eaec  
je quitar  
cmp eip, 7c91eb03  
je restaurar  
cmp eip,aux  
je restaurar2
```

al evaluar las excepciones ve si se produce alguna en la dirección de retorno con lo cual sabemos que esta en el punto de retorno al programa.

```
restaurar2:  
bc aux  
jmp inicio
```

Por lo tanto al haber vuelto ya al programa, con el comando BC borra el breakpoint que colocamos en la dirección de retorno y luego vuelve a inicio, donde vuelve a colocar los HBP borrados y a dar RUN nuevamente con todo restaurado.

Bueno ya vimos los cambios realizados no son gran cosa, veamos ahora si funciona, reiniciemos el programa.

Editamos el script para que ponga el HBP en la dirección del OEP

```
Archivo Edición Formato Ver Ayuda
var aux
inicio:
bphws 4271b5, "x"
trabajo:
leob pirulo
run
pirulo:
log eip
cmp eip, 7c91eaed
je quitar
cmp eip, 7c91eb03
je restaurar
cmp eip, aux
je restaurar2
jmp final

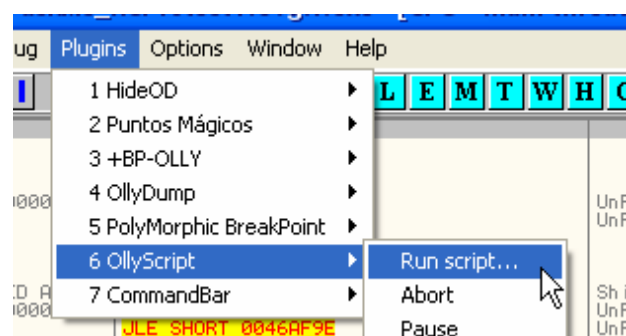
quitar:
bphwc 4271b5
jmp trabajo

restaurar:
mov aux, esp
mov aux, [aux]
add aux, 0b8
mov aux, [aux]
log aux
bp aux
jmp inicio

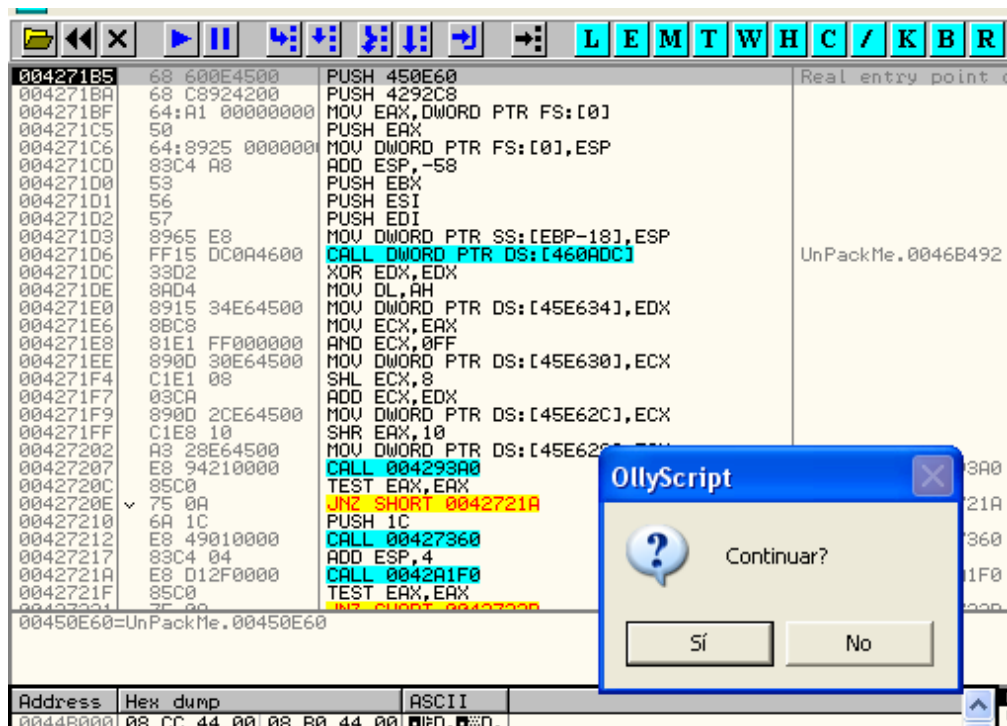
restaurar2:
bc aux
jmp inicio

final:
MSGYN "Continuar?"
cmp $RESULT, 1
je inicio
ret
```

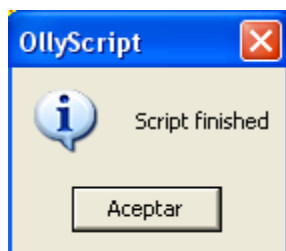
Allí editamos la dirección del HBP en el script que queremos que no nos borre, ponemos los dos BPs a mano necesarios para su funcionamiento, borramos todos los HBP anteriores que habia.



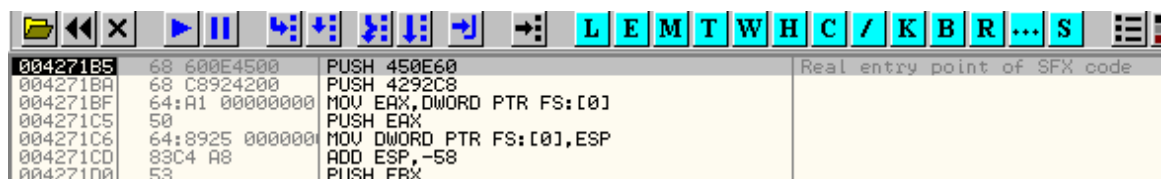
Corremos el script, no olvidemos de poner los dos Bps y poner todas las tildes en exceptions.



Allí nos avisa que se disparo nuestro HBP, le damos a que no continúe apretando NO.



Y evitamos que nos borre el HBP que colocamos y estamos detenidos en el OEP usando HBPs.



Es muy importante en estos nuevos packers tener la forma de evitar el borrado de los HBP, y ya lo hemos conseguido ya que para cualquier salto mágico o HBP que necesitamos colocar, tenemos mas armas contra ellos ya que los podremos usar.

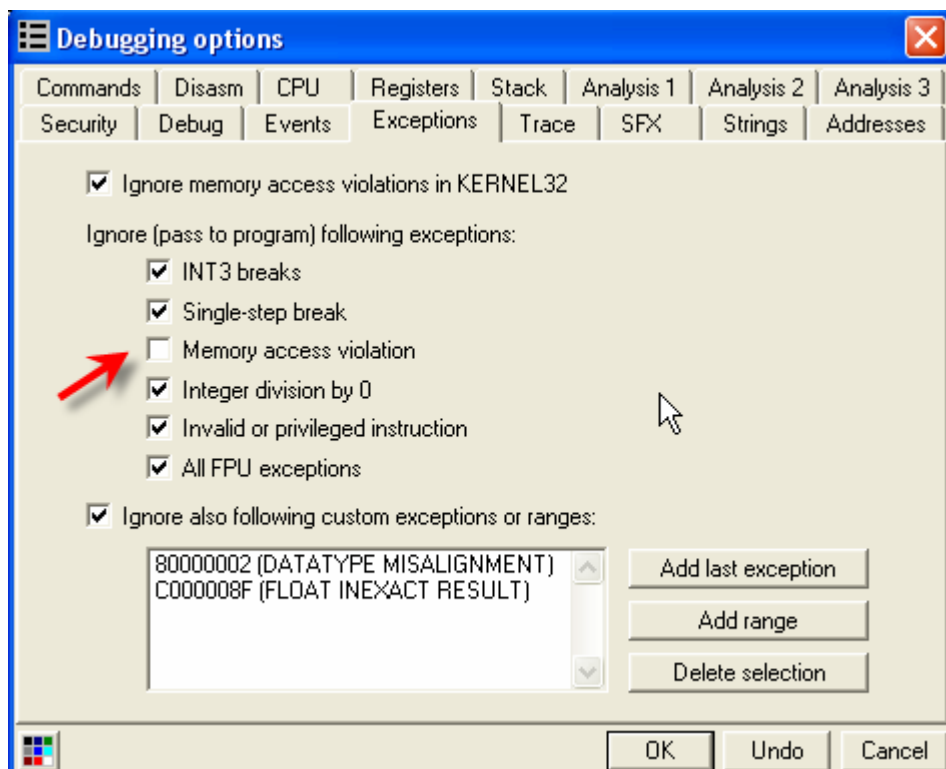
Ahora ya que podemos manejar perfectamente los HBP pasaremos a los stolen bytes.

Veamos si podemos llegar hasta la ultima exceptions y tracear desde allí.

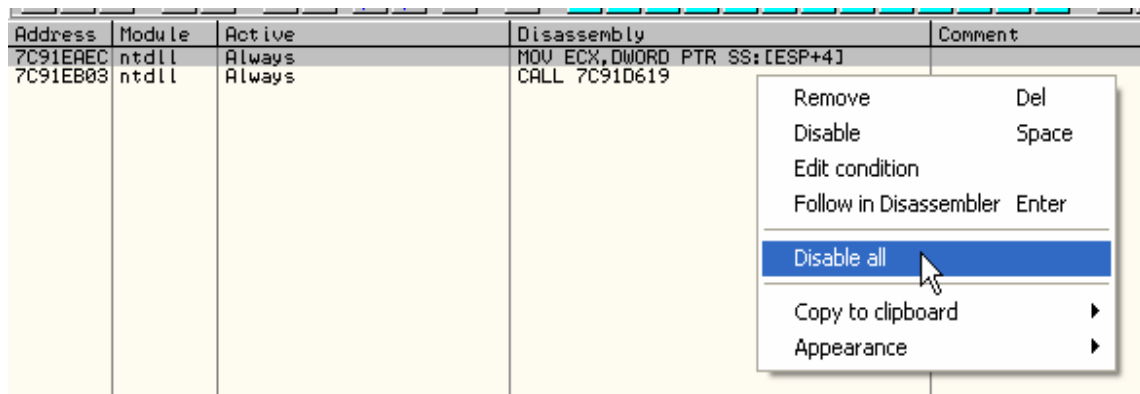
```
7b5b0000 Module C:\WINDOWS\system32\comctl32.dll
77f40000 Module C:\WINDOWS\system32\SHLWAPI.dll
77be0000 Module C:\WINDOWS\system32\msvcrt.dll
58c30000 Module C:\WINDOWS\system32\COMCTL32.dll
7c9d0000 Module C:\WINDOWS\system32\SHELL32.dll
773a0000 Module C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.26
72f80000 Module C:\WINDOWS\system32\WINSPool.DRV
74cc0000 Module C:\WINDOWS\system32\oledlg.dll
774b0000 Module C:\WINDOWS\system32\ole32.dll
00471394 INT3 command at UnPackMe.00471394
7c91e9ec Breakpoint at ntdll.KiUserExceptionDispatcher (KiUserAppDispatcher+2C)
eip = 7c91e9ec ; ntdll.KiUserExceptionDispatcher
7c91eb03 Breakpoint at ntdll.7c91eb03 (KiUserAppDispatcher+43)
eip = 7c91eb03
aux = 0012fc64
aux = 0012fc8c
aux = 0012fd44
aux = 00471395
00471395 Breakpoint at UnPackMe.00471395
eip = 00471395
0047108e Access violation when reading [FFFFFFFF]
7c91e9ec Breakpoint at ntdll.KiUserExceptionDispatcher (KiUserAppDispatcher+2C)
eip = 7c91e9ec ; ntdll.KiUserExceptionDispatcher
7c91eb03 Breakpoint at ntdll.7c91eb03 (KiUserAppDispatcher+43)
eip = 7c91eb03
aux = 0012fc68
aux = 0012fc8c
aux = 0012fd44
aux = 00471090
00471090 Breakpoint at UnPackMe.00471090
eip = 00471090
004271b5 Hardware breakpoint 1 at UnPackMe.004271b5
eip = 004271b5

Command [v]
Hardware breakpoint 1 at UnPackMe.004271b5
```

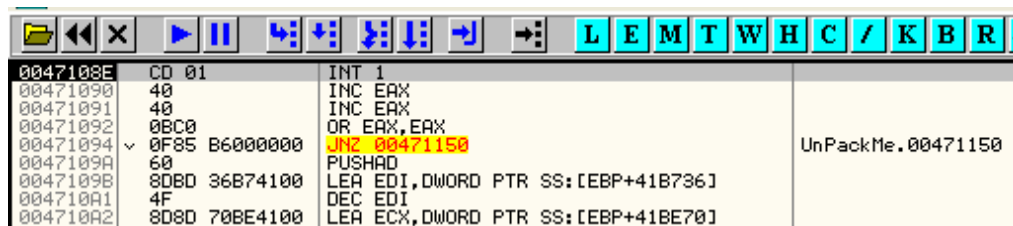
En el LOG veo que esa es la ultima excepci3n antes de llegar al OEP, as3 que como veo que es una ACCESS VIOLATION WHEN READING, quitare esa tilde para que pare en dicha exceptions.



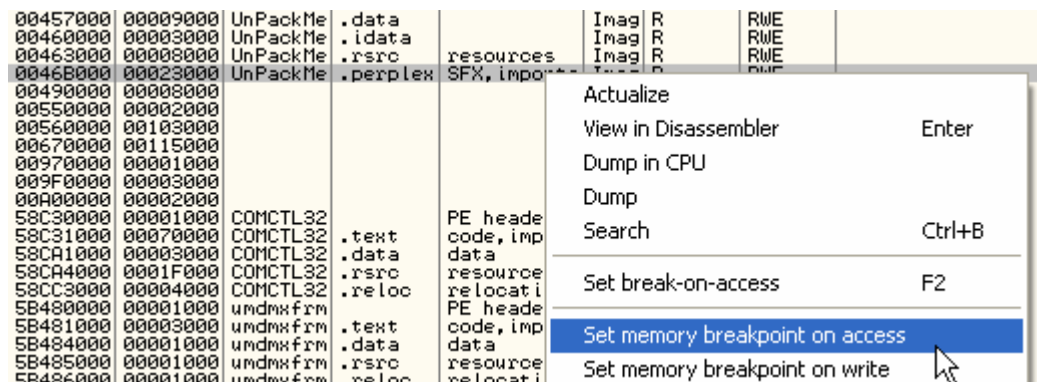
deshabilitare los Bps que coloque para usar el script.



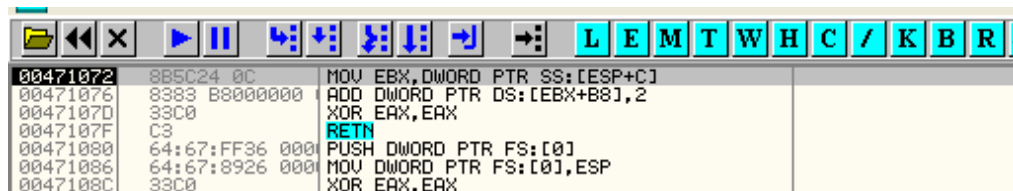
Y ahora reinicio y doy RUN



Allí para en la ultima exceptions, pongo un BPM ON ACCESS en la sección que esta ejecutándose, para que pare seguramente en el manejador de excepciones.



Paso la excepción con SHIFT +f9



Ahora traceo con f7 estas líneas del manejador de excepciones llegando al RET y al apretar RUN volvemos al programa en

00471090 40 INC EAX



Address	Disassembly	Comment
00471090	40	INC EAX
00471091	40	INC EAX
00471092	0BC0	OR EAX,EAX
00471094	0F85 B6000000	JNZ 00471150
0047109A	60	PUSHAD
0047109B	80BD 36B74100	LEA EDI,DWORD PTR SS:[EBP+41B736]
004710A1	4F	DEC EDI
004710A2	808D 70BE4100	LEA ECX,DWORD PTR SS:[EBP+41BE70]
004710A8	83C1 02	ADD ECX,2
004710AB	2BCF	SUB ECX,EDI
004710AD	C1E9 02	SHR ECX,2
004710B0	E8 49D8FFFF	CALL 0046E8FE
004710B5	AB	STOS DWORD PTR ES:[EDI]
004710B6	E2 F8	LOOPD SHORT 004710B0
004710B8	61	POPAD

Ahora si ya pasada la excepción, pongo el BP en el OEP ya que esta ya desempacado allá y me servirá para que pare cuando termine de tracear.

Address	Disassembly	Comment
004271B5	68 600E4500	PUSH 450E60
004271BA	68 C8924200	PUSH 4292C8
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	50	PUSH EAX
004271C6	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
004271CD	83C4 A8	ADD ESP,-58
004271D0	53	PUSH EBX
004271D1	56	PUSH ESI
004271D2	57	PUSH EDI

y desde 471090, iniciemos el traceo y vemos que tarda bastante ya que no saltea nada y pasa por todas las instrucciones una a una.

Allí paro al fin en el OEP así que puedo mirar el listado de la fila txt.

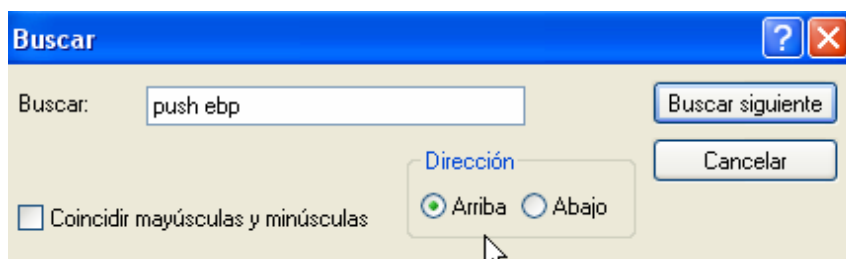
Address	Disassembly	Comment
004271B5	68 600E4500	PUSH 450E60
004271BA	68 C8924200	PUSH 4292C8
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	50	PUSH EAX
004271C6	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
004271CD	83C4 A8	ADD ESP,-58
004271D0	53	PUSH EBX
004271D1	56	PUSH ESI
004271D2	57	PUSH EDI
004271D3	8965 F8	MOV DWORD PTR SS:[EBP-18],ESP

```

00485EAE Main    JL SHORT 00485ECA
00485ECA Main    POPAD                                ; EAX=00000000, ECX=0012FFB0,
EDX=7C91EB94, EBX=7FFDF000, ESI=FFFFFFF
00485ECB Main    JMP SHORT 00485ECE
00485ECE Main    JMP DWORD PTR DS:[485ED4]
Breakpoint at UnPackMe.004271B5

```

allí están las ultimas lineas del txt y si busco de allí hacia arriba PUSH EBP, que es generalmente la primera instrucción de un programa.

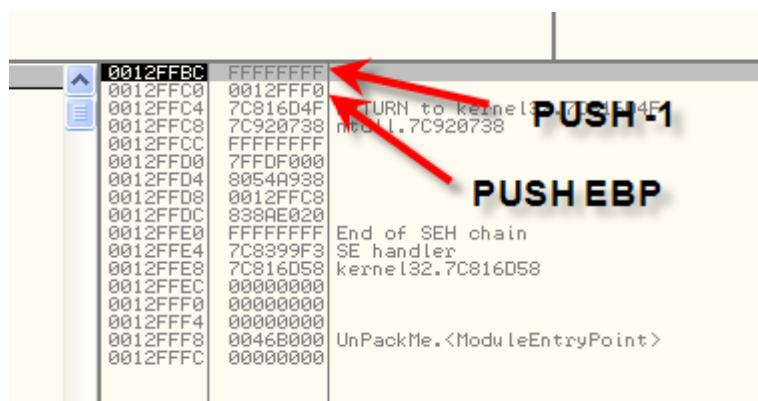


<b>00485AF3 Main</b>	<b>PUSH EBP</b>	
<b>00485AF4 Main</b>	<b>MOV EBP,ESP</b>	<b>; EBP=0012FFC0</b>
<b>00485AF6 Main</b>	<b>PUSH -1</b>	
00485AF8 Main	NOP	
00485AF9 Main	PUSHAD	
00485AFA Main	PUSHAD	
00485AFB Main	CALL 00485B00	
00485B00 Main	POP ESI	; ESI=00485B00
00485B01 Main	SUB ESI,6	; ESI=00485AFA
00485B04 Main	MOV ECX,35	; ECX=00000035
00485B09 Main	SUB ESI,ECX	; ESI=00485AC5
00485B0B Main	MOV EDX,E3D6D5FD	; EDX=E3D6D5FD
00485B10 Main	SHR ECX,2	; ECX=0000000D
00485B13 Main	SUB ECX,2	; ECX=0000000B
00485B16 Main	CMP ECX,0	

Allí vemos los stolen bytes y luego empieza a hacer pavadas pero antes de hacerlas guarda con PUSHAD el valor correcto de los registros que recuperara con POPAD justo antes de saltar al falso OEP.

00485ECA Main	<b>POPAD</b>	; EAX=00000000, ECX=0012FFB0, EDX=7C91EB94, EBX=7FFDF000, ESI=FFFFFFFF
00485ECB Main	JMP SHORT 00485ECE	
<b>00485ECE Main</b>	<b>JMP DWORD PTR DS:[485ED4]</b>	
<b>Breakpoint at UnPackMe.004271B5</b>		

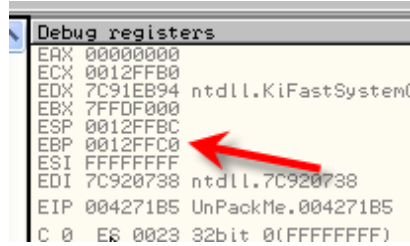
Por supuesto vemos que los stolen bytes coinciden con los valores que nos sobraban en el stack



Además el otro registro diferente al inicio es casualmente es EBP que varía al hacer el mov ebp,esp

<b>00485AF3 Main</b>	<b>PUSH EBP</b>	
<b>00485AF4 Main</b>	<b>MOV EBP,ESP</b>	<b>; EBP=0012FFC0</b>

## 00485AF6 Main PUSH -1



Debug registers	
EAX	00000000
ECX	0012FFB0
EDX	7C91EB94 ntdll.KiFastSystem
EBX	7FFDF000
ESP	0012FFBC
EBP	0012FFC0
ESI	FFFFFFFF
EDI	7C920738 ntdll.7C920738
EIP	004271B5 UnPackMe.004271B5
C 0 E6 0023 32bit 0(FFFFFFFF)	

Vemos que el listado del txt nos muestra que EBP tomo el valor 12FFc0, que es el mismo valor que tiene al llegar al falso OEP, todo lo que viene a continuación son solo volteretas para despistar y podrían no existir perfectamente, ya vemos que luego de tanta vuelta, EBP vuelve a valer 12ffc0, lo que demuestra que esos eran los stolen bytes correctos y todo lo intermedio es pura cascara.

Bueno con eso obtuvimos los famosos stolen bytes, hicimos funcionar el script que evita que nos borren los HBP y creo que por esta parte es suficiente seguiremos en la parte 43.

Hasta la parte 43

Ricardo Narvaja

miércoles, abril 26, 2006