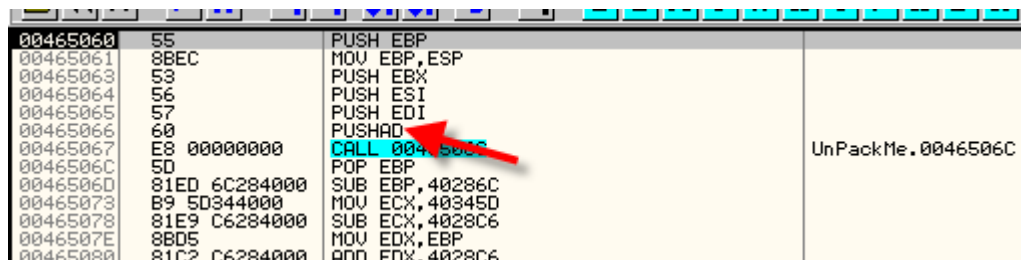


INTRODUCCION AL CRACKING CON OLLYDBG PARTE 38

En la ultima entrega vimos nuestro primer programa con entradas redireccionadas y como repararlas y continuaremos levemente aumentando la dificultad de los packers para ir aumentando el nivel, asimismo, empezaremos con los ejercicios para que practiquen un poco.

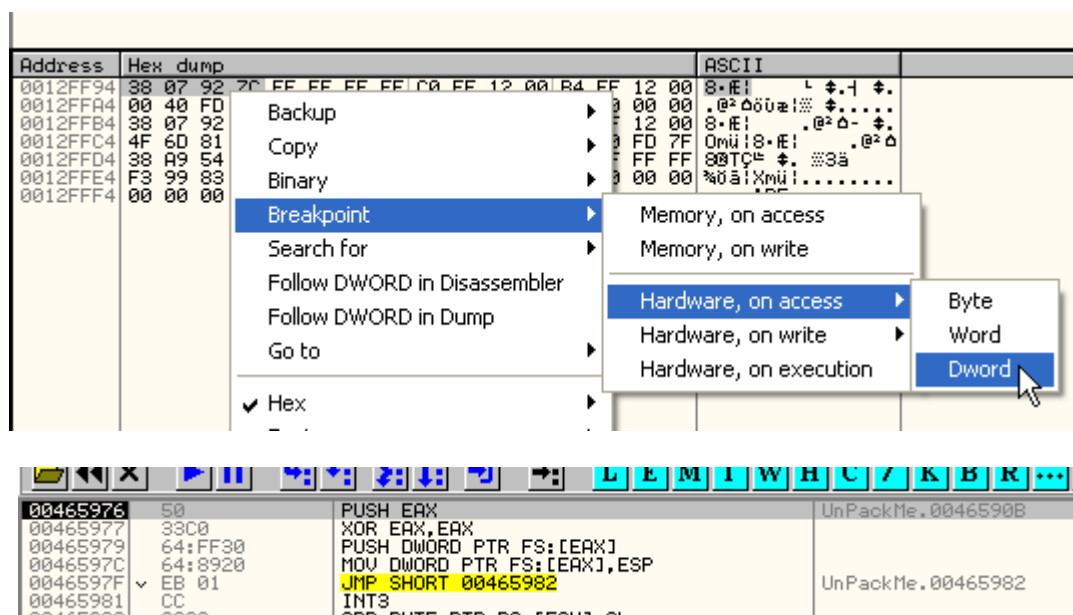
Aquí tenemos un unpackme de Yoda Crypter 1.3 para practicar un poco, por supuesto el OLLYDBG debe tener los plugins necesarios para no ser detectado por el antidebugging como vimos en partes anteriores.

Aquí estamos en el Entry Point



Vemos un PUSHAD funcionara? Probemos el metodo, lleguemos hasta alli traceando y pasemos con F7 el PUSHAD.

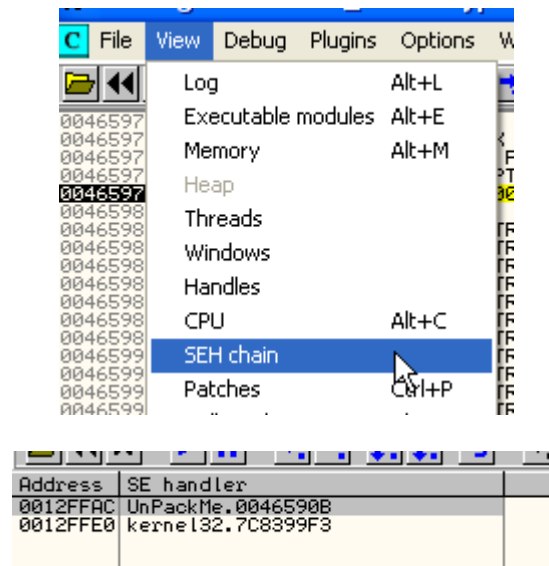
Hacemos ESP-FOLLOW IN DUMP y luego marcamos los primeros 4 bytes como vimos anteriormente y colocamos un HARDWARE BPX ON ACCESS.



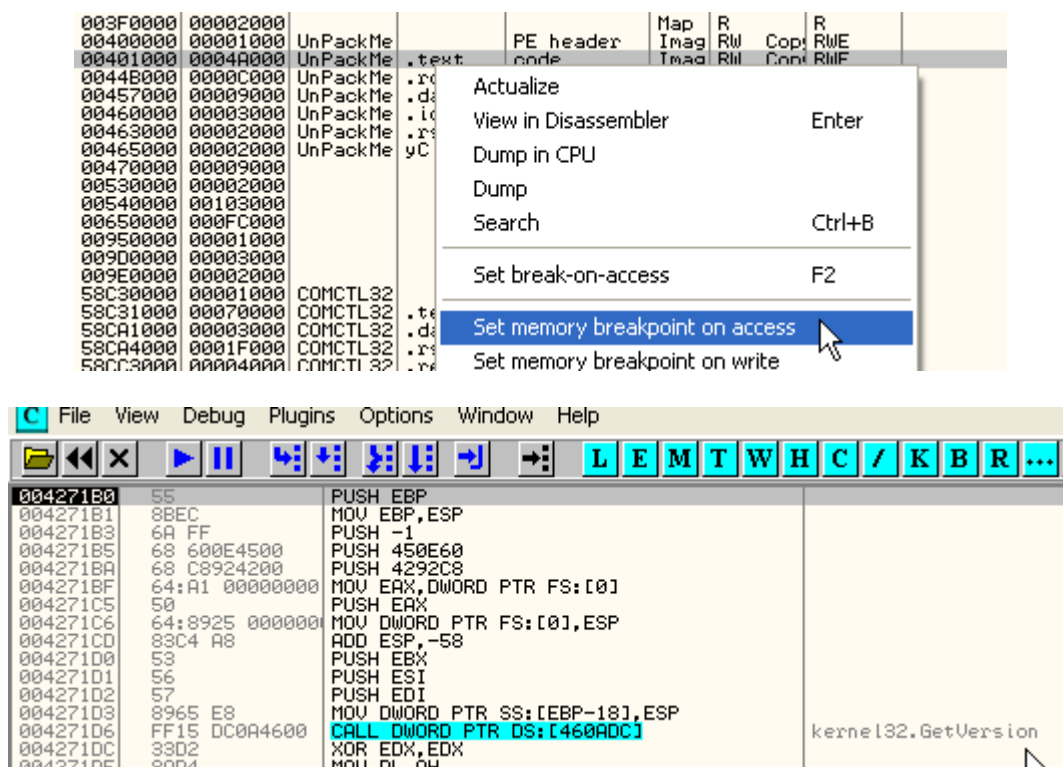
Vemos una rutina que creara un manejador de excepciones y luego como salta con un JMP a una zona de ceros para provocar un error, asi que si traceamos y llegamos al JMP.

00465976	50	PUSH EAX	
00465977	33C0	XOR EAX,EAX	
00465979	64:FF30	PUSH DWORD PTR FS:[EAX]	
0046597C	64:8920	MOV DWORD PTR FS:[EAX],ESP	
0046597F	EB 01	JMP SHORT 00465982	UnPackMe.00465982
00465981	CC	INT3	
00465982	0000	ADD BYTE PTR DS:[EAX],AL	
00465984	0000	ADD BYTE PTR DS:[EAX],AL	
00465986	0000	ADD BYTE PTR DS:[EAX],AL	

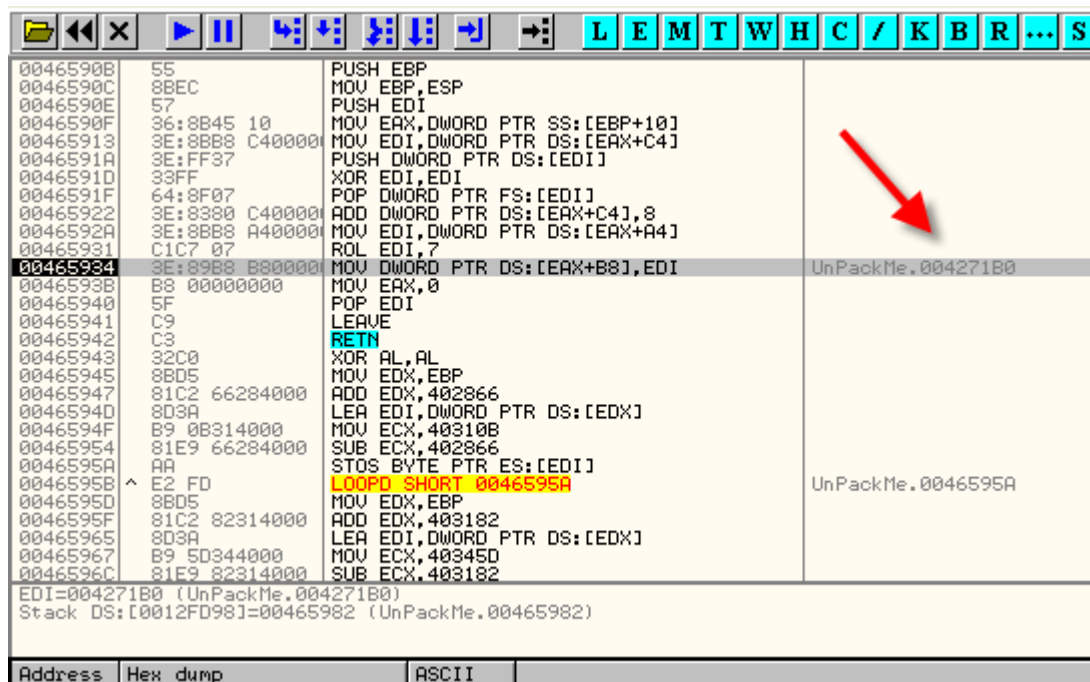
Alli va a saltar al error, miremos adonde saltara en el manejador de excepciones.



El mismo esta en 46590b, el que no quiere complicarse mucho la vida y llegar al OEP, aquí se puede poner un BPM ON ACCESS en la primera seccion y al pasar la excepcion llegaremos al OEP.



Alli esta el OEP es 4271B0 ya que esta empackado siempre con el mismo crackme, jeje.

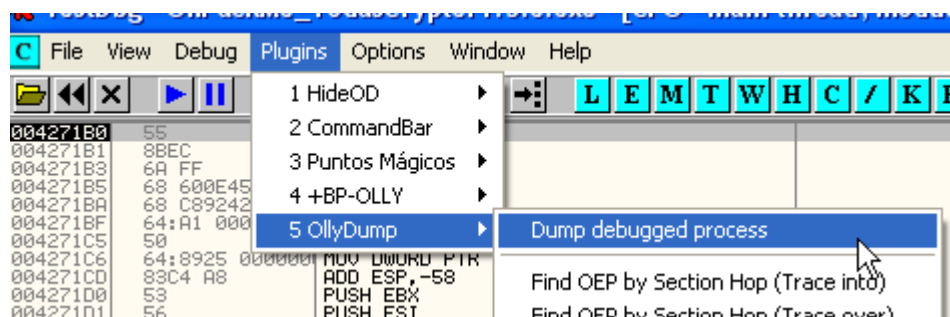


Address	Hex dump	ASCII
00465908	55	PUSH EBP
0046590C	8BEC	MOV EBP,ESP
0046590E	57	PUSH EDI
0046590F	36:8B45 10	MOV EAX,DWORD PTR SS:[EBP+10]
00465913	3E:8BB8 C40000	MOV EDI,DWORD PTR DS:[EAX+C4]
0046591A	3E:FF37	PUSH DWORD PTR DS:[EDI]
0046591D	33FF	XOR EDI,EDI
0046591F	64:8F07	POP DWORD PTR FS:[EDI]
00465922	3E:8380 C40000	ADD DWORD PTR DS:[EAX+C4],8
0046592A	3E:8BB8 A40000	MOV EDI,DWORD PTR DS:[EAX+A4]
00465931	C1C7 07	ROL EDI,7
00465934	3E:89B8 B80000	MOV DWORD PTR DS:[EAX+B8],EDI
00465938	B8 00000000	MOV EAX,0
00465940	5F	POP EDI
00465941	C9	LEAVE
00465942	C3	RETN
00465943	32C0	XOR AL,AL
00465945	8BD5	MOV EDX,EBP
00465947	81C2 66284000	ADD EDX,402866
0046594D	8D3A	LEA EDI,DWORD PTR DS:[EDX]
0046594F	B9 0B314000	MOV ECX,40310B
00465954	81E9 66284000	SUB ECX,402866
0046595A	AA	STOS BYTE PTR ES:[EDI]
0046595B	E2 FD	LOOPD SHORT 0046595A
0046595D	8BD5	MOV EDX,EBP
0046595F	81C2 82314000	ADD EDX,403182
00465965	8D3A	LEA EDI,DWORD PTR DS:[EDX]
00465967	B9 5D344000	MOV ECX,40345D
0046596C	81E9 82314000	SUB ECX,403182

EDI=004271B0 (UnPackMe.004271B0)
Stack DS:[0012FD98]=00465982 (UnPackMe.00465982)

Los que quieren profundizar un poco mas ya pueden ver que en el manejador de excepciones se manipula el context para modificar EIP y por lo tanto la direccion de la excepcion, que era normalmente 465982, o sea donde se produjo la excepcion, y es sobrescrita con la direccion del OEP o sea 4271B0 para que retorne de la excepcion directo al OEP, aunque esto no importa mucho por ahora, lo dejamos anotado para futuros estudios que hagamos de la estructura CONTEXT.

Bueno la cuestion es que ya estamos en el OEP, podemos DUMPEAR.



OllyDump - UnPackMe_YodasCrypter1.3.e.exe

Start Address: 400000 Size: 67000 Dump

Entry Point: 65060 -> Modify: 271B0 Get EIP as OEP Cancel

Base of Code: 1000 Base of Data: 4B000

☒ Fix Raw Size & Offset of Dump Image

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	0004917F	00001000	0004917F	00001000	E0000020
.rdata	0000BED8	0004B000	0000BED8	0004B000	C0000040
.data	00008BE4	00057000	00008BE4	00057000	C0000040
.idata	00002BEC	00060000	00002BEC	00060000	C0000040
.rsrc	0000140C	00063000	0000140C	00063000	40000040
yC	00002000	00065000	00002000	00065000	E00000E0

☐ Rebuild Import

☒ Method1: Search JMP[API] | CALL[API] in memory image

☐ Method2: Search DLL & API name string in dumped file

Le quitamos la tilde para que no intente reparar la IAT y DUMPEAMOS.

Mis sitios de red

Nombre: yodadump

Tipo: Executable file(*.exe)

Guardar Cancelar

0012FD28 00 00 00 00 00 00 00 00
 0012FD30 00 00 00 00 00 00 00 00
 0012FD38 00 00 00 00 00 00 00 00

Por supuesto si ejecutamos el yodadump.exe nos da error, nos podriamos haber dado por muy afortunados si corriera solo en nuestra maquina, pero ya veremos que la IAT tiene entradas redireccionadas lo cual hace que no pueda correr para nada, al tratar de acceder a esas direcciones inexistentes en el dump.

Bueno comencemos a analizar la IAT, busquemos una llamada a una api, alli debajo vemos una llamada a GetVersion la misma de siempre jeje.

Address	Disassembly	Comment
004271B0	PUSH EBP	
004271B1	MOV EBP,ESP	
004271B3	PUSH -1	
004271B5	PUSH 450E60	
004271B8	PUSH 4292C8	
004271BF	MOV EAX,DWORD PTR FS:[0]	
004271C5	PUSH EAX	
004271C6	MOV DWORD PTR FS:[0],ESP	
004271CD	ADD ESP,-58	
004271D0	PUSH EBX	
004271D1	PUSH ESI	
004271D2	PUSH EDI	
004271D3	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	CALL DWORD PTR DS:[460ADC]	kernel32.GetVersion
004271DC	XOR EDX,EDX	
004271DE	MOV DL,AH	
004271E0	MOV DWORD PTR DS:[45E634],EDX	
004271E6	MOV ECX,EAX	
004271E8	AND ECX,0FF	
004271EE	MOV DWORD PTR DS:[45E630],ECX	
004271F4	SHL ECX,8	

Bueno vemos que dicho CALL toma valores de 460ADC para ver donde saltara o sea que esta es una entrada de la IAT, vayamos a verla en el DUMP.

Address	Hex dump	ASCII
00460ADC	AB 14 81 7C F4 97 80 7C	! ! ! ! ! ! ! !
00460AE4	01 B0 85 7C 19 62 82 7C	! ! ! ! ! ! ! !
00460AEC	5C E8 81 7C 53 00 83 7C	! ! ! ! ! ! ! !
00460AF4	19 3C 87 7C CB 08 81 7C	! ! ! ! ! ! ! !
00460AFC	C1 0F 87 7C 5B B2 81 7C	! ! ! ! ! ! ! !
00460B04	E9 06 87 7C 4E 99 80 7C	! ! ! ! ! ! ! !
00460B0C	AC 92 80 7C 11 07 87 7C	! ! ! ! ! ! ! !
00460B14	42 24 80 7C F3 B8 81 7C	! ! ! ! ! ! ! !
00460B1C	A9 2C 87 7C F4 2C 87 7C	! ! ! ! ! ! ! !
00460B24	BA 38 87 7C 0C 6E 82 7C	! ! ! ! ! ! ! !
00460B2C	F1 BA 80 7C 3D 31 87 7C	! ! ! ! ! ! ! !
00460B34	83 31 87 7C CC 37 87 7C	! ! ! ! ! ! ! !
00460B3C	77 1D 80 7C 28 AC 80 7C	! ! ! ! ! ! ! !
00460B44	66 AA 80 7C A9 2C 81 7C	! ! ! ! ! ! ! !
00460B4C	ED CB 81 7C 3D 0D 87 7C	! ! ! ! ! ! ! !
00460B54	19 90 83 7C 59 35 81 7C	! ! ! ! ! ! ! !
00460B5C	31 03 92 7C 40 03 92 7C	! ! ! ! ! ! ! !
00460B64	D7 EF 80 7C 2D FF 80 7C	! ! ! ! ! ! ! !
00460B6C	2F FE 80 7C 51 28 81 7C	! ! ! ! ! ! ! !
00460B74	11 03 81 7C B1 C7 80 7C	! ! ! ! ! ! ! !
00460B7C	65 A0 80 7C CF C6 80 7C	! ! ! ! ! ! ! !
00460B84	21 2E 82 7C BD 99 80 7C	! ! ! ! ! ! ! !
00460B8C	88 2D 82 7C 5D 99 80 7C	! ! ! ! ! ! ! !
00460B94	94 97 80 7C 7B 97 80 7C	! ! ! ! ! ! ! !
00460B9C	29 B5 80 7C CF C6 80 7C	! ! ! ! ! ! ! !
00460BA4	00 00 00 00 48 33 15 00	! ! ! ! ! ! ! !
00460BAC	4D 33 15 00 52 33 15 00	! ! ! ! ! ! ! !

Pues se ve esta parte de la IAT como correcta ya que si miro en el mapa de memoria donde van las entradas, todas van a 7C8XXXXX o cercanas y esas direcciones caen en la seccion code de kernel32.dll o sea que estas entradas serian correctas.

77FAE000	00002000	SHLWAPI	.rsrc	resources	Image	R	RWE
77FB0000	00006000	SHLWAPI	.reloc	relocations	Image	R	RWE
7C800000	00001000	kernel32		PE header	Image	R	RWE
7C801000	00082000	kernel32	.text	code,import	Image	R	RWE
7C803000	00005000	kernel32	.data	data	Image	R	RWE
7C80B000	00073000	kernel32	.rsrc	resources	Image	R	RWE
7C80B000	00006000	kernel32	.reloc	relocations	Image	R	RWE
7C910000	00001000	ntdll		PE header	Image	R	RWE

Por lo demas si busco referencias en cualquiera de ellas al azar y haciendo click derecho FIND REFERENCES hay referencias.

Address	Disassembly	Comment
0040AEE0	CALL DWORD PTR DS:[460B34]	kernel32.ReadConsoleInputA

Si sigo bajando hasta la separacion

Address	Hex dump	ASCII
00460B8C	88 2D 82 7C 5D 99 80 7C	e-e!J0C!
00460B94	94 97 80 7C 7B 97 80 7C	duC!CuC!
00460B9C	29 B5 80 7C CF C6 80 7C	JAQ!08C!
00460BA4	00 00 00 00 48 33 15 00	...H3\$.
00460BAC	4D 33 15 00 52 33 15 00	M3\$.R3\$.
00460BB4	57 33 15 00 5C 33 15 00	W3\$.\3\$.
00460BBC	61 33 15 00 66 33 15 00	a3\$.f3\$.
00460BC4	6B 33 15 00 70 33 15 00	k3\$.p3\$.
00460BCC	75 33 15 00 7A 33 15 00	u3\$.z3\$.
00460BD4	7F 33 15 00 84 33 15 00	03\$.33\$.
00460BDC	89 33 15 00 8E 33 15 00	83\$.A3\$.
00460BE4	93 33 15 00 00 00 00 00	83\$.....
00460BEC	FE 69 A8 7C ED 69 A8 7C	!i!i!i!
00460BF4	80 0E A5 7C 00 00 00 00	C#N!....
00460BFC	05 2D 15 00 DA 2D 15 00	'-\$.r-\$.
00460C04	0F 2D 15 00 E4 2D 15 00	■-\$.6-\$.
00460C0C	E9 2D 15 00 EE 2D 15 00	U-\$.--\$.
00460C14	F3 2D 15 00 F8 2D 15 00	%-\$.o-\$.
00460C1C	FD 2D 15 00 02 2E 15 00	2-\$.0.\$.
00460C24	07 2E 15 00 0C 2E 15 00	.\$....\$.
00460C2C	11 2E 15 00 16 2E 15 00	4\$....\$.
00460C34	1B 2E 15 00 20 2E 15 00	+\$....\$.
00460C3C	25 2E 15 00 2A 2E 15 00	%\$.*.\$.
00460C44	2F 2E 15 00 34 2E 15 00	/\$.4.\$.
00460C4C	39 2E 15 00 3E 2E 15 00	9\$.>.\$.
00460C54	43 2E 15 00 48 2E 15 00	C\$.H.\$.
00460C5C	4D 2E 15 00 52 2E 15 00	H\$.R.\$.

Veo que el siguiente grupo va a una zona de memoria 15XXXX veamos que hay alli.

00130000	00002000				Priv	RWE	RWE	
00140000	00003000				Map	R	R	
00150000	00029000				Priv	RW	RW	
00250000	00006000				Priv	RW	RW	
00270000	00003000				Map	RW	RW	
00290000	00017000				Map	R	R	

Vemos una seccion que no hay dll asi que seguramente esa seccion fue creada por el packer, si reinicio para comprobarlo.

00130000	00002000				Priv	RWE	RWE	
00140000	00003000				Map	R	R	
00150000	00003000				Priv	RW	RW	
00250000	00006000				Priv	RW	RW	
00260000	00003000				Map	RW	RW	
00270000	00016000				Map	R	R	\Device\H:
00290000	0003D000				Map	R	R	\Device\H:
002D0000	00041000				Map	R	R	\Device\H:
00320000	00006000				Map	R	R	\Device\H:

Vemos que hay una seccion creada por el sistema de 3000 bytes, pero la que usa el packer es de 29000 bytes mucho mas larga asi que el packer agrando esa seccion para su uso.

Por lo tanto estas son entradas redireccionadas por el packer a esa seccion, veamos una de ellas a ver como trabaja.

Address	Hex dump	ASCII
00460B9C	29 B5 80 7C CF C6 80 7C	JAQ!08C!
00460BA4	00 00 00 00 48 33 15 00	...H3\$.
00460BAC	4D 33 15 00 52 33 15 00	M3\$.R3\$.
00460BB4	57 33 15 00 5C 33 15 00	W3\$.\3\$.
00460BBC	61 33 15 00 66 33 15 00	a3\$.f3\$.
00460BC4	6B 33 15 00 70 33 15 00	k3\$.p3\$.
00460BCC	75 33 15 00 7A 33 15 00	u3\$.z3\$.
00460BD4	7F 33 15 00 84 33 15 00	03\$.33\$.
00460BDC	89 33 15 00 8E 33 15 00	83\$.A3\$.
00460BE4	93 33 15 00 00 00 00 00	83\$.....
00460BEC	FE 69 A8 7C ED 69 A8 7C	!i!i!i!

Tomare una de ellas, por ejemplo esta, y buscare la referencia, haciendo click derecho FIND REFERENCES.

Address	Disassembly	Comment
00446685	CALL DWORD PTR DS:[460BAC]	DS:[00460BAC]=0015334D
004466CB	CALL DWORD PTR DS:[460BAC]	DS:[00460BAC]=0015334D

Hay dos calls que toman valores de dicha entrada, mirare el primero haciendo doble click en el, me llevara en el listado adonde esta ubicado.

00446683	57	PUSH EDI	
00446684	53	PUSH EBX	
00446685	FF15 AC0B4600	CALL DWORD PTR DS:[460BAC]	
00446688	33F6	XOR ESI,ESI	
0044668D	8B3D C00A4600	MOV EDI,DWORD PTR DS:[460AC0]	kernel32.WideCharToMultiByte
00446693	56	PUSH ESI	
00446694	56	PUSH ESI	
00446695	8BE8	MOV EBP,EAX	
00446697	56	PUSH ESI	
00446698	56	PUSH ESI	
00446699	55	PUSH EBP	
0044669A	53	PUSH EBX	
0044669B	56	PUSH ESI	
0044669C	56	PUSH ESI	
0044669D	FFD7	CALL EDI	
0044669F	50	PUSH EAX	
004466A0	8B4C24 18	MOV ECX,DWORD PTR SS:[ESP+18]	
004466A4	894424 1C	MOV DWORD PTR SS:[ESP+1C],EAX	
004466A8	E8 B259FFFF	CALL 0043C05F	UnPackMe.0043C05F
004466AD	56	PUSH ESI	

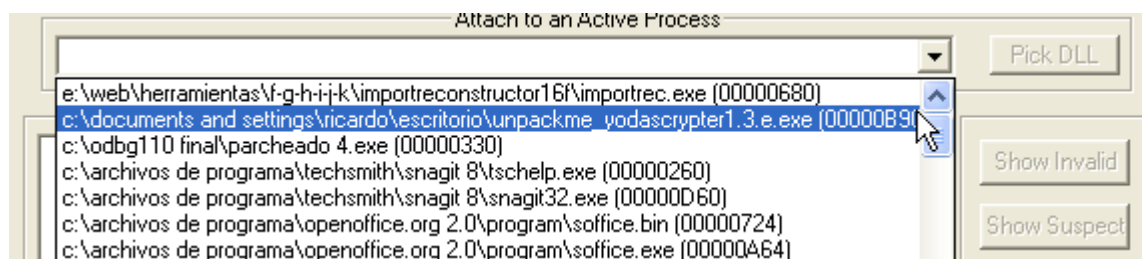
Alli vemos el CALL, hagamos click derecho FOLLOW a ver que hace en la seccion redireccionada.

0015334D	- E9 E918FA76	JMP 770F4C3B	OLEAUT32.SysStringLen
00153352	- E9 3D72FC76	JMP 7711A594	OLEAUT32.OleCreateFontIndirect
00153357	- E9 FD17FA76	JMP 770F4B59	OLEAUT32.SysAllocStringLen
0015335C	- E9 2118FA76	JMP 770F4E82	OLEAUT32.SafeArrayGetDim
00153361	- E9 32A1FC76	JMP 7711D498	OLEAUT32.SafeArrayGetElemsize
00153366	- E9 301DFA76	JMP 770F509B	OLEAUT32.SafeArrayGetLBound
0015336B	- E9 DF1CFA76	JMP 770F504F	OLEAUT32.SafeArrayGetUBound
00153370	- E9 9B1CFA76	JMP 770F5010	OLEAUT32.SafeArrayAccessData
00153375	- E9 C51CFA76	JMP 770F503F	OLEAUT32.SafeArrayUnaccessData
0015337A	- E9 5A33FA76	JMP 770F66D9	OLEAUT32.VariantChangeType
0015337F	- E9 CC14FA76	JMP 770F4850	OLEAUT32.SysFreeString
00153384	- E9 CC18FA76	JMP 770F4C55	OLEAUT32.SysAllocStringByteLen
00153389	- E9 3418FA76	JMP 770F4BC2	OLEAUT32.SysAllocString
0015338E	- E9 029FFC76	JMP 7711D295	OLEAUT32.VariantCopy
00153393	- E9 E8290077	JMP 7715D080	OLEAUT32.OleLoadPicture
00153398	0000	ADD BYTE PTR DS:[EAX],AL	
0015339A	0000	ADD BYTE PTR DS:[EAX],AL	
0015339C	0000	ADD BYTE PTR DS:[EAX],AL	
0015339E	0000	ADD BYTE PTR DS:[EAX],AL	

Vemos que sin muchos prolegomenos salta a la api SysStringLen directamente,por lo cual si lo traceamos a mano ya sabemos cual es la api correcta para esta entrada.

Vemos que la redireccion es muy sencilla, podra el IMP REC con sus traceadores repararla sin necesidad de hallar el magico?

Abramos el IMP REC.



Alli esta el proceso detenido en el OEP, lo elegimos en el menu desplegable.

No olvidemos que al IMP REC debemos darle 3 datos OEP, INICIO DE IAT y LARGO.

OEP=4271B0 al cual restandole la imagebase da **271B0**

El inicio de la IAT si subimos en la misma.

van a seccion code de dlls o aredireccionadas a la seccion 15XXXX.

Address	Hex dump	ASCII
00460ECC	9E 32 15 00 A3 32 15 00	x23.u23.
00460ED4	A8 32 15 00 AD 32 15 00	23.423.
00460EDC	00 00 00 00 F8 32 15 00	...23.
00460EE4	FD 32 15 00 02 33 15 00	23.033.
00460EEC	07 33 15 00 0C 33 15 00	33..33.
00460EF4	11 33 15 00 16 33 15 00	433..33.
00460EFC	1B 33 15 00 20 33 15 00	433..33.
00460F04	25 33 15 00 2A 33 15 00	433.*33.
00460F0C	2F 33 15 00 34 33 15 00	/33.433.
00460F14	39 33 15 00 3E 33 15 00	933.>33.
00460F1C	43 33 15 00 00 00 00 00	C33.....
00460F24	F3 32 15 00 00 00 00 00	%23.....
00460F2C	0B 00 50 6C 61 79 53 6F	3.PlaySo
00460F34	75 6E 64 41 00 00 57 49	undA..WI
00460F3C	4E 4D 4D 2E 64 6C 6C 00	NH1.dll.
00460F44	FE 00 47 65 74 4D 6F 64	■.GetMod
00460F4C	75 6C 65 48 61 6E 64 6C	uleHandl
00460F54	65 41 00 00 7E 01 49 6E	eA..0In
00460F5C	74 65 72 6C 6F 63 68 65	terlocke
00460F64	64 49 6E 63 72 65 6D 65	dIncreme
00460F6C	6E 74 00 00 7E 01 49 6E	nt..t0In

Alli vemos la ultima entrada de la IAT es 460f24 mas abajo si busco referencias no hay para ninguna entrada, asi que para que la ultima entrada quede dentro, el final de la IAT es 460f28, ahora debemos hallar el largo de la misma.

LARGO=FINAL -INICIO= 460f28 - 460818= 710



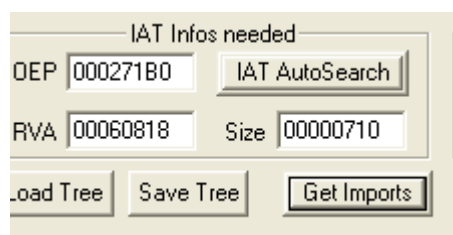
Asi que poniendo en limpio

OEP=271B0

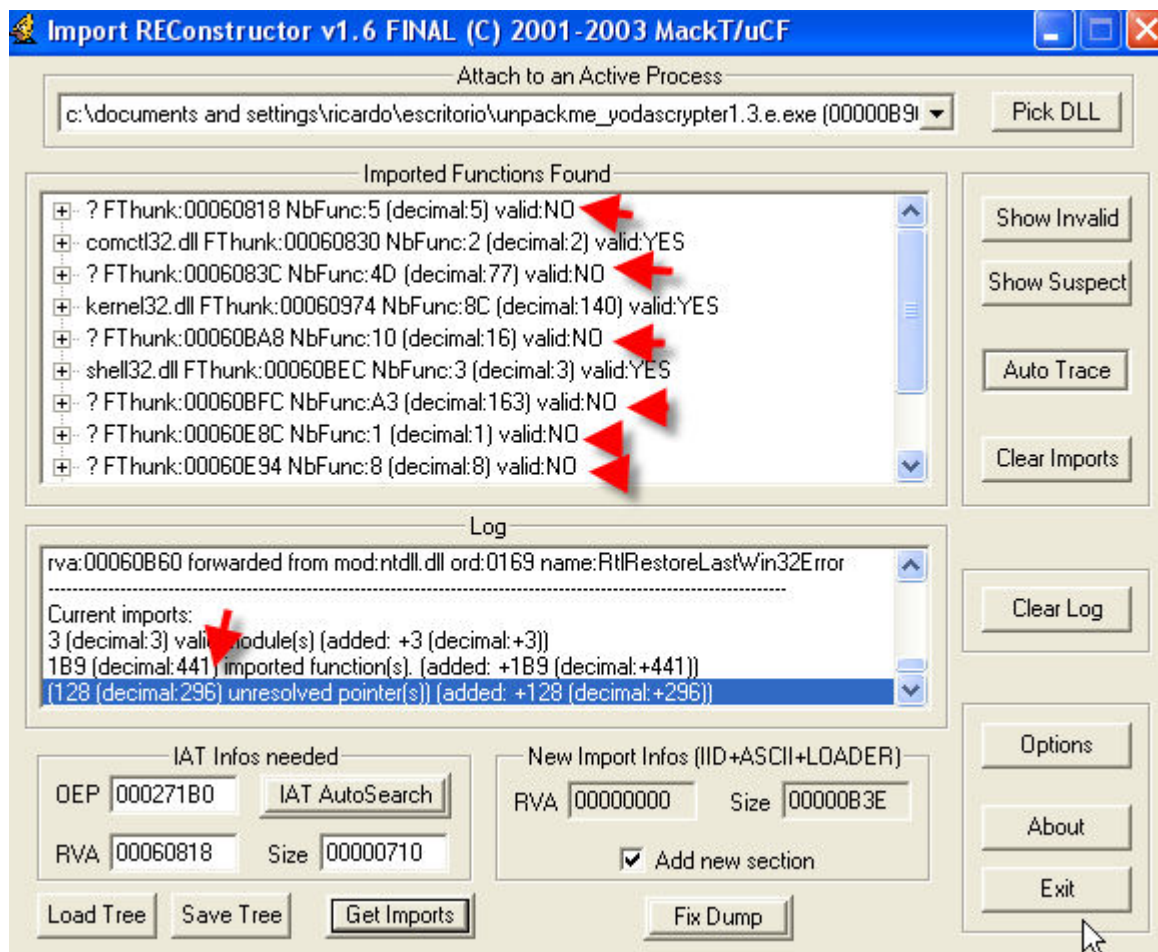
RVA o INICIO=60818

SIZE O LARGO=710

Pongamos estos valores en el IMP REC.

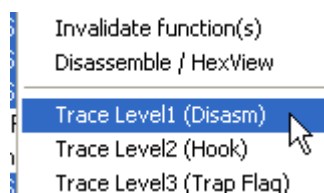


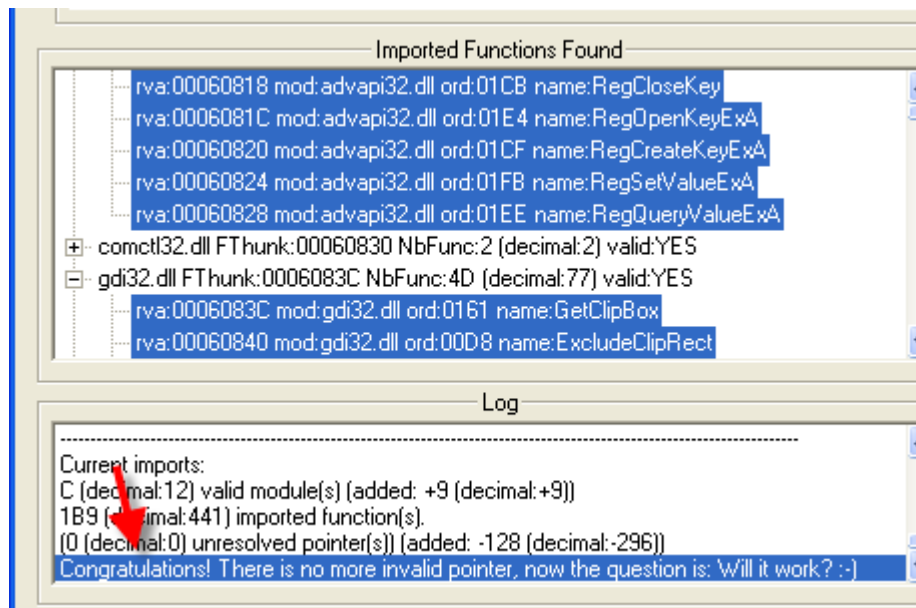
Al apretar GET IMPORTS



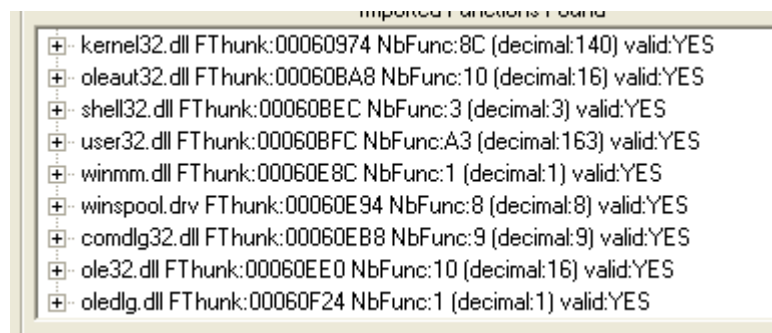
Vemos que hay 296 entradas sin resolver, lo podra resolver con algun traceador?

Si apreto el boton AUTO TRACE se cuelga, probemos con los otros traceadores, apreto SHOW INVALIDS y en las entradas marcadas hago click derecho TRACE LEVEL 1.

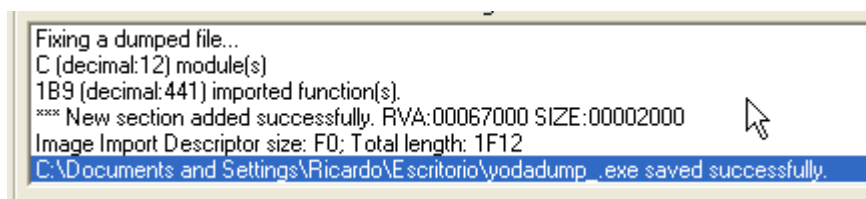




Dice que resolvió todo y que no hay más entradas inválidas, le creemos? Jeje. Si apreto SHOW INVALID nuevamente veo que todas están YES.



Probemos reparar el yodadump.exe apreto FIX DUMP



Allí me creo el yodadump.exe que supuestamente está reparado, probemoslo.



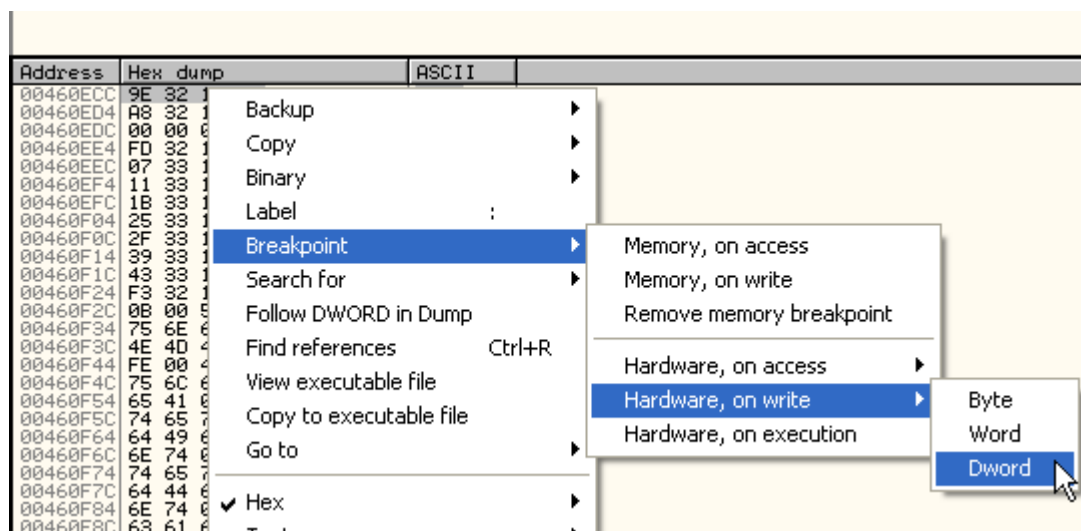
Jeje esta vez se portó el IMP REC y me ahorro mucho trabajo.

Ahora además de esto y para jorobar un poco, tiene salto mágico este packer?

Busquemos una mala entrada en la IAT en el programa que esta detenido en el OEP.

Address	Hex dump	ASCII
00460ECC	9E 32 15 00 A3 32 15 00	%2\$.u2\$.
00460ED4	A8 32 15 00 AD 32 15 00	%2\$.u2\$.
00460EDC	00 00 00 00 F8 32 15 00	...%2\$.
00460EE4	FD 32 15 00 02 33 15 00	%2\$.03\$.
00460EEC	07 33 15 00 0C 33 15 00	%3\$.3\$.
00460EF4	11 33 15 00 16 33 15 00	%3\$.3\$.
00460EFC	1B 33 15 00 20 33 15 00	%3\$.3\$.
00460F04	25 33 15 00 2A 33 15 00	%3\$.3\$.
00460F0C	2F 33 15 00 34 33 15 00	%3\$.43\$.
00460F14	39 33 15 00 3E 33 15 00	%3\$.>3\$.
00460F1C	43 33 15 00 00 00 00 00	C3\$.....
00460F24	F3 32 15 00 00 00 00 00	%2\$.....
00460F2C	0B 00 50 6C 61 79 53 6F	ø.PlaySo
00460F34	75 6E 64 41 00 00 57 49	undA..WI
00460F3C	4E 4D 4D 2E 64 6C 6C 00	NMM.dil.

Le colocare un HARDWARE BPX ON WRITE antes de reiniciar, ya que los HBP se mantienen, aun despues de reiniciar y ya vimos con el metodo del PUSHAD que el packer no es alergico a los mismos.



Por supuesto el metodo es tratar de parar cuando guarda el valor malo 15XXXX en la entrada, para ello reinicio el OLLYDBG y doy RUN.

Address	Hex dump	Disassembly	Comment
0046572F	5A	POP EDX	
00465730	8902	MOV DWORD PTR DS:[EDX],EAX	
00465732	EB 10	JMP SHORT 00465751	UnPackMe.00465751
00465734	52	PUSH EDX	
00465735	51	PUSH ECX	
00465736	8B01	MOV EAX,DWORD PTR DS:[ECX]	

Alli paro y guardo el valor bueno en la entrada de la IAT vemos que EAX tiene el valor de una API.

Register	Value	Comment
EAX	76377CD8	comdlg32.GetSaveFileNameA
ECX	004607B8	UnPackMe.004607B8
EDX	00460ECC	UnPackMe.00460ECC
EBX	76360000	comdlg32.76360000
ESP	0012FF94	
EBP	00062800	UnPackMe.00062800
ESI	00465A53	UnPackMe.00465A53
EDI	0046288E	ASCII:"GetSaveFileNameA"
EIP	00465732	UnPackMe.00465732
C	0	ES 0023 32bit 0(FFFFFFFF)
P	1	CS 001B 32bit 0(FFFFFFFF)
A	0	SS 0023 32bit 0(FFFFFFFF)

Y lo guarda en la entrada la cual esta siendo apuntada por EDX, quiere decir que en este caso,

primero guarda el valor bueno y luego lo modifica por el malo, sigamos traceando a ver cuando sucede esto.

0046578F	8BCD	MOV ECX,EBP	
00465791	81C1 79344000	ADD ECX,403479	
00465797	8D39	LEA EDI,DWORD PTR DS:[ECX]	
00465799	3E:8B77 04	MOV ESI,DWORD PTR DS:[EDI+4]	
0046579D	8932	MOV DWORD PTR DS:[EDX],ESI	
0046579F	2BC6	SUB EAX,ESI	
004657A1	83E8 05	SUB EAX,5	
004657A4	C606 E9	MOV BYTE PTR DS:[ESI],0E9	

Vemos que apenas una cuantas lineas mas abajo, ESI toma el valor malo y lo va a sobrescribir en la entrada.

Registers (FPU)		
EAX	76377CD8	comdlg32.GetSaveFileNameA
ECX	00465C79	UnPackMe.00465C79
EDX	00460ECC	UnPackMe.00460ECC
EBX	76360000	comdlg32.76360000
ESP	0012FF88	ASCII "SZF"
EBP	00062800	
ESI	0015329E	
EDI	00465C79	UnPackMe.00465C79
EIP	0046579D	UnPackMe.0046579D

Quiere decir que las entradas buenas solo escribieran la primera vez y no llegaran a esta segunda parte donde sobrescribe con el valor malo.

Repitamos el proceso ahora con una entrada buena, la de GetVersion que estaba unas lineas debajo del OEP estaba correcta asi que usemosla era 460ADC pongo un HBP ON WRITE en ella.

0046572E	61	PUPHU	
0046572F	5A	POP EDX	
00465730	8902	MOV DWORD PTR DS:[EDX],EAX	
00465732	EB 1D	JMP SHORT 00465751	UnPac
00465734	52	PUSH EDX	
00465735	51	PUSH ECX	
00465736	8B01	MOV EAX,DWORD PTR DS:[ECX]	
00465738	2D 00000000	SUB EAX,80000000	
0046573D	50	PUSH EAX	
0046573E	53	PUSH EBX	
0046573F	8B05	MOV EDX,EBP	
00465741	81C2 9B334000	ADD EDX,40339B	
00465747	FF12	CALL DWORD PTR DS:[EDX]	ASCII
00465749	85C0	TEST EAX,EAX	
0046574B	74 7B	JE SHORT 004657C8	UnPac
0046574D	59	POP ECX	
0046574E	5A	POP EDX	
0046574F	8902	MOV DWORD PTR DS:[EDX],EAX	
00465751	51	PUSH ECX	

Vemos que para alli, al guardar la direccion correcta que esta en EAX.

Registers (FPU)		
EAX	7C8114AB	kernel32.GetVersion
ECX	004603C8	UnPackMe.004603C8
EDX	00460ADC	UnPackMe.00460ADC
EBX	7C800000	kernel32.7C800000
ESP	0012FF94	
EBP	00062800	
ESI	00465A2F	UnPackMe.00465A2F
EDI	004612A2	ASCII "GetVersion"
EIP	00465732	UnPackMe.00465732
C	0	ES 0023 32bit 0(FFFFFFFF)
P	1	CS 001B 32bit 0(FFFFFFFF)
A	0	SS 0023 32bit 0(FFFFFFFF)

Traceemos un poco.

Address	Hex dump	Assembly	Comment
0046575A	F701 20000000	TEST DWORD PTR DS:[ECX],20	
00465760	74 4F	JE SHORT 004657B1	UnPackMe.004657B1
00465762	8BCD	MOV ECX,EBP	
00465764	81C1 1F324000	ADD ECX,40321F	
0046576A	8339 00	CMP DWORD PTR DS:[ECX],0	
0046576D	74 14	JE SHORT 00465783	UnPackMe.00465783
0046576F	81FB 00000070	CMP EBX,70000000	
00465775	72 08	JB SHORT 0046577F	UnPackMe.0046577F
00465777	81FB FFFFFFF7	CMP EBX,77FFFFFF	
0046577D	76 0E	JBE SHORT 0046578D	UnPackMe.0046578D
0046577F	EB 30	JMP SHORT 004657B1	UnPackMe.004657B1
00465781	EB 0A	JMP SHORT 0046578D	UnPackMe.0046578D
00465783	81FB 00000080	CMP EBX,80000000	
00465789	73 02	JNB SHORT 0046578D	UnPackMe.0046578D
0046578B	EB 24	JMP SHORT 004657B1	UnPackMe.004657B1
0046578D	57	PUSH EDI	
0046578E	56	PUSH ESI	
0046578F	8BCD	MOV ECX,EBP	
00465791	81C1 79344000	ADD ECX,403479	
00465793	8D39	LEA EDI,DWORD PTR DS:[ECX]	
00465795	3E:8B77 04	MOV ESI,DWORD PTR DS:[EDI+4]	
00465797	8932	MOV DWORD PTR DS:[EDX],ESI	
00465799	2BC6	SUB EAX,ESI	
0046579B	83E8 05	SUB EAX,5	
0046579D	C606 E9	MOV BYTE PTR DS:[ESI],0E9	
0046579F	8946 01	MOV DWORD PTR DS:[ESI+1],EAX	
004657A1	3E:8347 04 05	ADD DWORD PTR DS:[EDI+4],5	
004657A3	5E	POP ESI	
004657A5	5F	POP EDI	
004657A7	59	POP ECX	
004657A9	83C1 04	ADD ECX,4	

Vemos que llega al JMP que evita la zona marcada con la flecha roja donde se cambia por el valor malo, así que el tema es evitar que salte este JMP, algunos de los saltos anteriores que evitan el JMP es el salto magico, veamos cual es ya que hay varios saltos condicionales, delante del JMP.

También podríamos intentar como solución NOPEAR la línea donde guarda los valores malos así no lo guarda y queda toda la tabla bien, eso sería también correcto, podemos probar si funciona.

Address	Hex dump	Assembly
00465797	8D39	LEA EDI,DWORD PTR DS:[ECX]
00465799	3E:8B77 04	MOV ESI,DWORD PTR DS:[EDI+4]
0046579B	8932	MOV DWORD PTR DS:[EDX],ESI
0046579D	2BC6	SUB EAX,ESI
0046579F	83E8 05	SUB EAX,5
004657A1	C606 E9	MOV BYTE PTR DS:[ESI],0E9
004657A3	8946 01	MOV DWORD PTR DS:[ESI+1],EAX
004657A5	3E:8347 04 05	ADD DWORD PTR DS:[EDI+4],5
004657A7	5E	POP ESI
004657A9	5F	POP EDI
004657AB	59	POP ECX
004657AD	83C1 04	ADD ECX,4
004657AF	83C2 04	ADD EDX,4
004657B1	E9 09FFFFFF	JMP 004656C6
004657B3	83C6 0C	ADD ESI,0C
004657B5	E9 6BFFFFFF	JMP 00465630
004657B7	33C0	XOR EAX,EAX

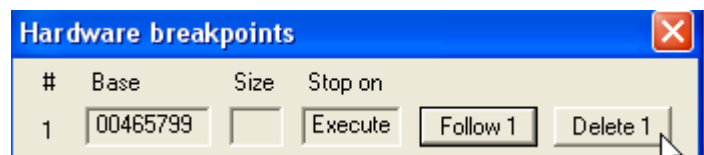
Para ello marco la línea anterior a la que guarda los valores malos y le coloqué un HBP ON EXECUTION y reinicié hasta que pare allí por primera vez, lo coloqué en la línea anterior porque el HBP para una instrucción después de donde lo coloqué y no quiero que me guarde el valor malo.

Address	Disassembly	Comment
0046578B	JMP SHORT 004657B1	UnPackMe.004657B1
0046578D	PUSH EDI	
0046578E	PUSH ESI	
0046578F	MOV ECX,EBP	
00465791	ADD ECX,403479	
00465797	LEA EDI,DWORD PTR DS:[ECX]	
00465799	MOV ESI,DWORD PTR DS:[EDI+4]	
0046579D	MOV DWORD PTR DS:[EDX],ESI	
0046579F	SUB EAX,ESI	
004657A1	SUB EAX,5	
004657A4	MOV BYTE PTR DS:[ESI],0E9	
004657A7	MOV DWORD PTR DS:[ESI+1],EAX	
004657AA	ADD DWORD PTR DS:[EDI+4],5	

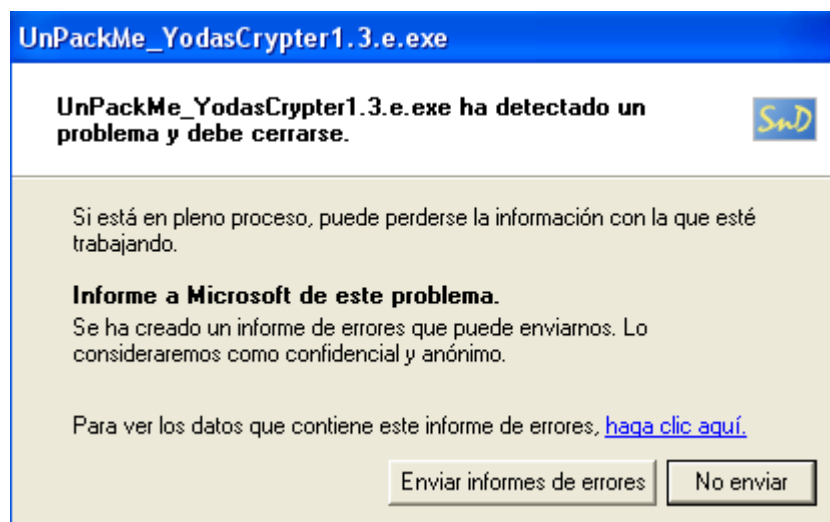
Alli paro, probemos nopeando la linea donde guarda el valor malo.

Address	Disassembly	Comment
0046579F	MOV ECX,EBP	
00465791	ADD ECX,403479	
00465797	LEA EDI,DWORD PTR DS:[ECX]	
00465799	MOV ESI,DWORD PTR DS:[EDI+4]	
0046579D	NOP	
0046579E	NOP	
0046579F	SUB EAX,ESI	
004657A1	SUB EAX,5	
004657A4	MOV BYTE PTR DS:[ESI],0E9	

Ahora sigamos a ver si no detecta el nopeo que hice y llega al OEP.



Quito el HBP y doy RUN

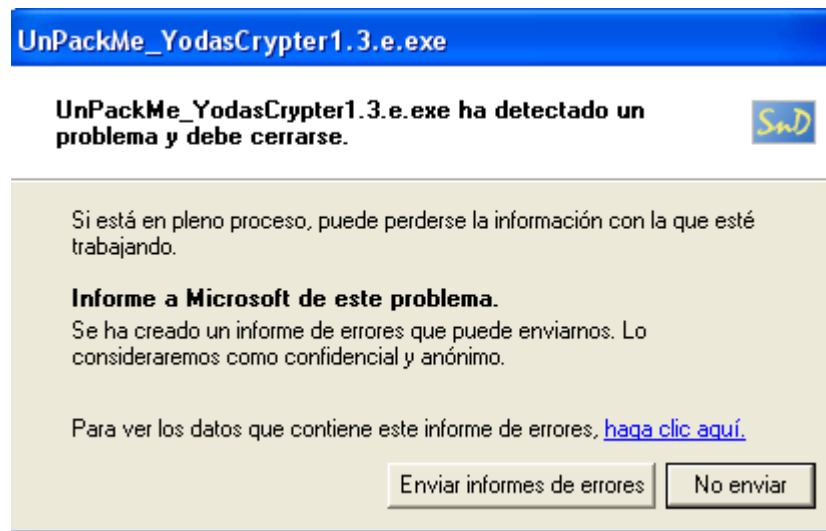


Bueno el maldito detecta los cambios que hice y da error, asi que el nopeo no va, continuemos con el salto magico.

Address	Disassembly	Comment
0046576F	CMP EBX,70000000	
00465775	JB SHORT 0046577F	UnPackMe.0046577F
00465777	CMP EBX,7FFFFFFF	
0046577D	JBE SHORT 0046578D	UnPackMe.0046578D
0046577F	JMP SHORT 004657B1	UnPackMe.004657B1
00465781	JMP SHORT 0046578D	UnPackMe.0046578D
00465783	CMP EBX,80000000	
00465789	JNB SHORT 0046578D	UnPackMe.0046578D
0046578B	JMP SHORT 004657B1	UnPackMe.004657B1
0046578D	PUSH EDI	
0046578E	PUSH ESI	

Traceando cualquier entrada mala, veo que es este el salto magico, el que evita el JMP y me lleva a

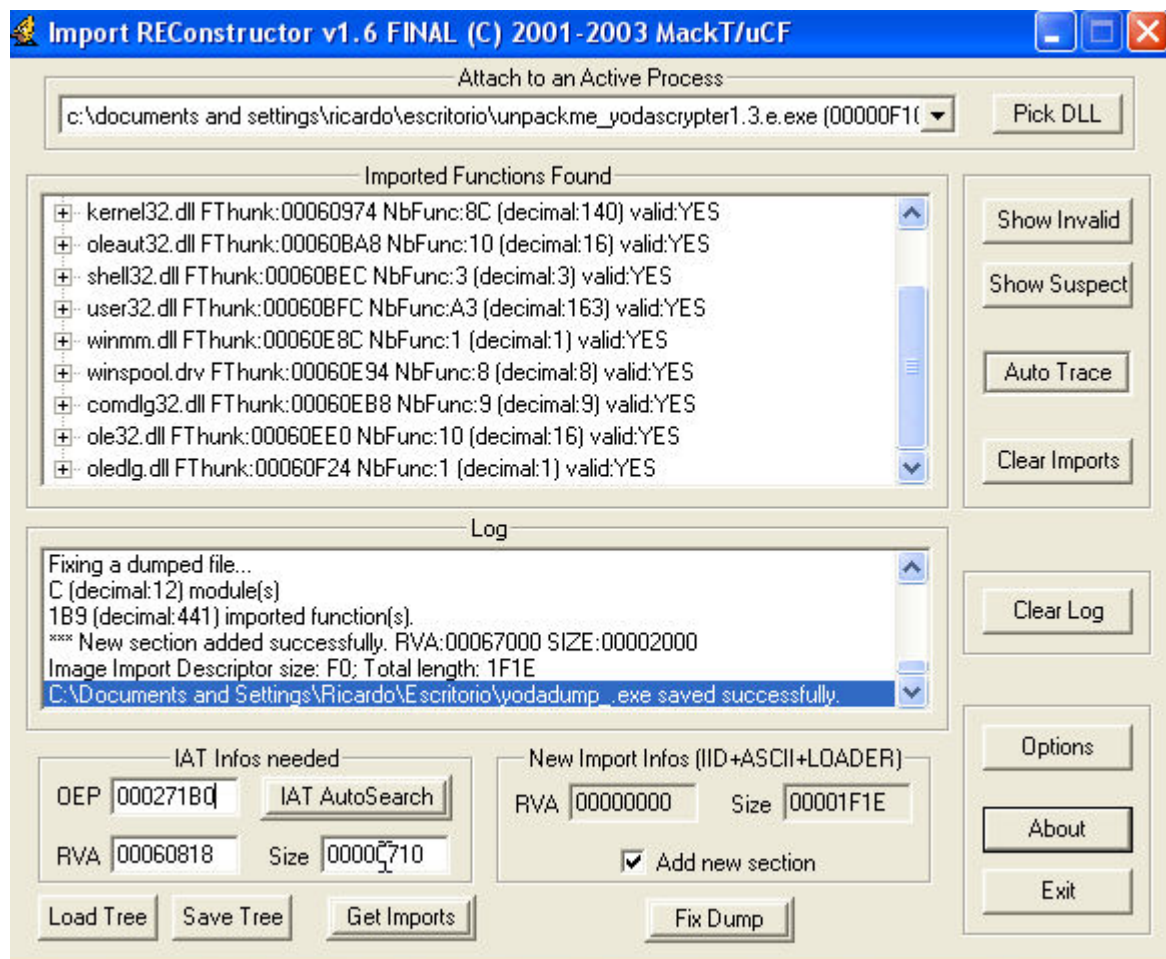
la zona donde guarda el valor malo, así que habría que nopearlo para llegar al JMP, aunque dudo que no detecte el nopeo, igual probemos.



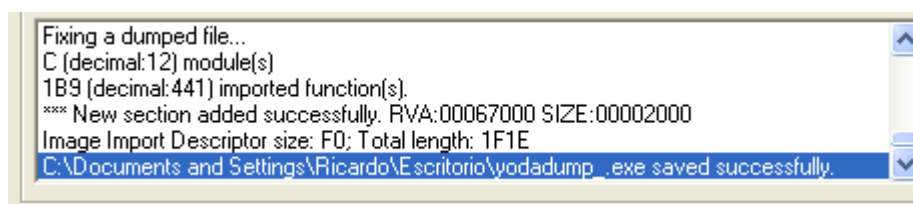
Igual en ambos metodos el programa nos da error pero luego de haber reparado correctamente toda la tabla si miramos la misma, en el programa que se detuvo por el error.

Address	Hex dump	ASCII
004607FC	78 2B 06 00 60 2B 06 00	x+±. ' +±.
00460804	4C 2B 06 00 2E 2B 06 00	L+±. . +±.
0046080C	00 00 00 00 08 00 00 80 0. . 0
00460814	00 00 00 00 F0 6B DA 77 -k rw
0046081C	1B 76 DA 77 F4 EA DA 77	+v rw 00 rw
00460824	E7 EB DA 77 83 78 DA 77	00 rw 00 rw
0046082C	00 00 00 00 DD 15 C5 58 !S+X
00460834	2E BD C3 58 00 00 00 00	.c t%. . . .
0046083C	04 6A EF 77 66 95 EF 77	éj' w f0' w
00460844	89 6A EF 77 F3 AD EF 77	éj' w 0i' w
0046084C	ED 09 EF 77 99 8B EF 77	Y' w 0i' w
00460854	00 B5 EF 77 2A 7D EF 77	' 0' w *j' w
0046085C	B2 7C EF 77 77 53 F2 77	00' w w S= w
00460864	1E C9 F1 77 0C BC EF 77	00' w 0' w
0046086C	52 04 EF 77 FA 8D EF 77	Ré' w. ' i' w
00460874	F1 DD EF 77 51 B2 EF 77	±! ' w 00' w
0046087C	26 05 EF 77 2A E3 EF 77	&' w *0' w
00460884	5F 39 F2 77 71 B4 EF 77	-9= w q1' w
0046088C	2E AD EF 77 E1 61 EF 77	.i' w B a' w
00460894	B8 85 EF 77 CC 02 EF 77	0a' w l fE' w
0046089C	43 70 EF 77 FB EA F0 77	Cp' w' 0 - w
004608A4	12 83 EF 77 01 72 F0 77	0a' w 0x- w
004608AC	A9 34 F0 77 D5 93 EF 77	04- w' 0' w

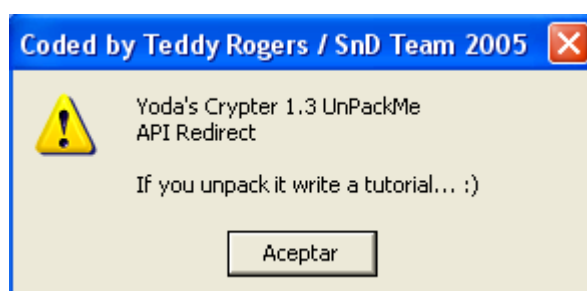
Veo que la IAT esta toda correcta así que no hay problema ninguno, tengo dos posibilidades aquí, o bien abro en otro OLLYDBG otra instancia del crackme, y sin modificar nada, llego al OEP, y le copio y pego la tabla correcta encima de la mala, con BYNARY COPY y BINARY PASTE lo cual es muy sencillo, también puedo utilizar directamente este proceso para el IMP REC que aunque no haya llegado al OEP ya tiene toda la IAT correcta y eso es con lo cual trabaja el IMP REC, el resto no importa, así que si lo abro en el IMP REC y le coloco los valores correctos de OEP; RVA y SIZE, y apreto GET IMPORTS.



Veo que me salen todos los valores correctos, y aunque el proceso que tengo detenido no me sirva para dumpear ya que no llego al OEP, no importa ya que he dumpado previamente en uno que si llego al OEP, y a ese dumpado solo le falta la tabla que este si tiene correcta, por lo cual puedo reparar perfectamente con este proceso el dumpado que ya tenia hecho, sin problemas y funcionara correctamente, si borro el anterior yodadump_.exe que estaba ya reparado y busco el dumpado anterior yodadump.exe, y apreto fix dump y lo reparo.



Me crea un nuevo yodadump_.exe veamos si funciona.



Si funciona perfectamente quiere decir que aprendimos que el dumpeado se necesita hacer desde el OEP ya que alli esta todo el programa desempacado en memoria, pero la tabla puede arreglarse desde un proceso que la tenga correcta, aunque no haya llegado hasta el OEP, porque el IMP REC no cambia nada del DUMPEADO, salvo la IAT la cual esta correcta, por eso el dumpeado funciona perfectamente.

Como ejercicio les dejo el archivo adjunto para desempacar, es muy sencillo si que espero que lo puedan hacer sin problemas.

Hasta la parte 39
Ricardo Narvaja
27/03/06