Test6



CONCURSO 1. OBJETIVO: Unpacking! PACKER: VMProtect v1.60 – 2.05 HERRAMIENTAS: OllyDBG v1.10, y PlugIns. PLUGINS FUNDAMENTALES:

> CommandBar v3.00.108 StrongOD v0.4.5.810 OllyDump v3.00.110

Break On Execution v1.1b

POR: NOX

PE: Test6.exe [VB 6.0]

Enero - 2012

Introducción

Haciendo este VMProtect para mi es un gran paso, pero no puedo decir que estoy satisfecho, es más dude mucho en mandarlo así y escribir este tute, pero bueno u.u, como dicen en la lista siempre es bueno que haya bastante tutes de algo que poco o ninguno.

El crédito por este tute se lo lleva karmany, ya que buscando información sobre VB, encontré un escrito de él y cómo reparaba la iat de un Themida creo que era, pero estoy seguro que no era un VMProtect. A esto se suma al tute de Guan de Dio que hizo sobre este packer, creo que si no hubiera leído esos escritos no lo pudiera haber resuelto. Muchas gracias a los dos! Un humilde Unpacking...

Configuración del PlugIn

Para este packer es necesario usar el StronOD, esta configuración es la que uso por defecto, así qué, si me funciona, no cambio nada.

StrongOL	0 v0.4.5.810			×
sO gOod,s ▼	sO sTroNg! HidePEB	~	!*Kill BadPE Bug	
	!*PatchFloat	•	AdvEnumModule	
	Adv Ctrl+G	Γ	Anti Anti_Attach	
•	*KernelMode	☑	Skip Some Exceptions	;
Γ	*ShowBar	7	Remove EP one-shot	
	Break On Tls	Γ	Break On Ldr	
v	Load Symbols	•	AutoUpdate	
_⊂ r e	eateProcess Optio	n —		
•	Normal			
C	CreateAsUser		Super Mode	
C	CreateAsRestrict			
	Save		Cancel ed by ????[CUG][RCT]	
			iccp://www.unpack.cn	

Analizando

Ejecutamos.



Analizamos con el PiD.



Nos sacó la versión.

Unpackeando

Aquí es donde comienza el laburo. Abrimos el proggie dentro del OllyDBG...

Address	Hex dump	Disassembly
005F9A05	E8 340A0000	CALL Test6.005FA43E
005F9A0A	3C F6	CMP AL, ØF6
005F9A0C	92	XCHG EAX, EDX
005F9A0D	06	PUSH ES
005F9A0E	F6	222
005F9A0F	8C2F	MOV WORD PTR DS:[EDI], GS
005F9A11	6D	INS DWORD PTR ES:[EDI], DX
005F9A12	1C F3	SBB AL. ØF3
005F9A14	^ E2 89	LOOPD SHORT Test6.005F999F
00000017	40	THE DUTE DTD FORFEDIA DV

Este packer usa la tabla TLS para empezar a desempacarse, entonces para la resolución de este packer, se debe tener cómo obligación los PlugIns que he mencionado.

El VMProtect detecta los BPs, BPMs, HBPs, para poder usarlos debemos saber en que momento deben ser puestos, siendo así, el packer no lo detectará.

Ahí parados en el TLS CallBack...

Como todo, debemos llegar al OEP y lo primero que haría, sería poner un BPM on Access (Execution) en la sección .CODE, pero como sabemos este packer detecta los BPM así que debemos saber dónde ponerlos.

Los BPMs no son más que cambios en los permisos de una terminada región, es decir quitar o agregar permisos de escritura, lectura o acceso + el flag PAGE_GUARD, cuando el progie accede a la región de memoria que queremos controlar, manda una notificación al depurador, generando que este pare, todo esto controlado nativamente por el Olly.

Para que el packer gestione los cambios que hemos hecho con estas banderas/flag, necesita obtener dicha información de las regiones de memoria, para esto puede usar la API VirtualQuery o también la API VirtualProtect u otra que desconozco o se me está pasando, pero estoy con un sueño ZzZzZzZz...

También sabemos que detecta los BPs, así que debemos ser cuidados al ponerlos, aparte de que este packer también emula las APIS, así que tenemos todo un espécimen.

Nos evitamos ese problema si ponemos un BP en VirtualProtectEx.



Damos F9 RUN.

Address	Hex	dump		Disasse	mbly					Comment
7C801A61	٢ŝ	8BFF		MOV EDI	, EDI					
7C801A63		55		PUSH EB	2					
7C801A64	H •	8BEC		MOV EBP	, ESP					
7C801A66	H •	56		PUSH ES	1					
7C801A67	1 ·	8B35	C412807C	MOV ESI	, DWORD	PTR	DS: E<&r	tdll.NtPr	otec	ntdll.ZwProtectVirtualMemory

Para por primera vez, si vemos el stack \rightarrow

Address	Value	Comment
0012F664	7C801AEC	CALL to VirtualProtectEx from kernel32.7C801AE7
0012F668	FFFFFFFF	hProcess = FFFFFFFF
0012F66C	005EB000	Address = Test6.005EB000
0012F670	000090E0	Size = 90E0 (37088.)
0012F674	00000004	NewProtect = PAGE_READWRITE
0012F678	0012FF98	<pre>p0ldProtect = 0012FF98</pre>

Nada interesante... Seguimos con F9 RUN.

Address	Value	Comment
0012F664	7C801AEC	CALL to VirtualProtectEx from kernel32.7C801AE7
0012F668	FFFFFFF	hProcess = FFFFFFFF
0012F66C	005EB000	Address = Test6.005EB000
0012F670	000090E0	Size = 90E0 (37088.)
0012F674	00000020	NewProtect = PAGE_EXECUTE_READ
0012F678	0012FF98	<pre>pOldProtect = 0012FF98</pre>

Nada tampoco seguimos con F9 RUN, hasta ver lo siguiente \rightarrow

Address	Value	Comment	
0012F5D0	7C801AEC	<pre>CALL to VirtualProtectEx from kernel32.7C801AE7</pre>	
0012F5D4	FFFFFFFF	hProcess = FFFFFFF	
0012F5D8	00401000	Address = Test6.00401000	
0012F5DC	000012B8	Size = 1288 (4792.)	
0012F5E0	00000004	NewProtect = PAGE READWRITE	
0012F5E4	0012FF04	pOldProtect = 0012FF04	

Si hubiéramos puesto un BPM en la sección .CODE que comienza en la dirección 00401000h, al pasar esta API el packer detectaría el cambio que hicimos y nos sacaría a patadas.

Entonces lo que debemos hacer es poner el BPM una vez que pase la comprobación del packer a la sección .CODE.

F9 RUN, otra vez.

Address	Value	Comment
0012F5D0	7C801AEC	CALL to VirtualProtectEx from kernel32.7C801AE7
0012F5D4	FFFFFFF	hProcess = FFFFFFFF
0012F5D8	00405000	Address = Test6.00405000
0012F5DC	0000F60E	Size = F60E (62990.)
0012F5E0	00000004	NewProtect = PAGE_READWRITE
0012F5E4	0012FF04	LpOldProtect = 0012FF04

Seguimos en la sección .CODE.

F9 RUN, otra vez.

Address	Value	Comment
0012F5D0	7C801AEC	CALL to VirtualProtectEx from kernel32.7C801AE7
0012F5D4	FFFFFFF	hProcess = FFFFFFFF
0012F5D8	00401000	EAddress = Test6.00401000
0012F5DC	000012B8	Size = 1288 (4792.)
0012F5E0	00000020	NewProtect = PAGE_EXECUTE_READ
0012F5E4	0012FF04	<pre>LpOldProtect = 0012FF04</pre>

F9 RUN de nuevo.

Address Value Comment	
OO12F5D0 7C801AEC CALL to VirtualProtectEx from kernel:	32.7C801AE7
0012F5D4 FFFFFFFF hProcess = FFFFFFFF	
0012F5D8 00405000 Address = Test6.00405000	
0012F5DC 0000F60E Size = F60E (62990.)	
0012F5E0 00000020 NewProtect = PAGE_EXECUTE_READ	
0012F5E4 0012FF04 Lp01dProtect = 0012FF04	

El packer sigue gestionando la protección a la determinada región de memoria.

F9 RUN.

Address	Value	Comment
0012FBB4	7C801AEC	<pre>_CALL to VirtualProtectEx from kernel32.7C801AE7</pre>
0012FBB8	FFFFFFFF	hProcess = FFFFFFFF
0012FBBC	02C70000	Address = 02070000
0012FBC0	000060000	Size = 6000 (24576.)
0012FBC4	00000020	NewProtect = PAGE_EXECUTE_READ
0012FBC8	0012FBF4	pOldProtect = 0012FBF4

De aquí hacia delante ya no vuelve a tocar la sección .CODE, entonces para intentar llegar al OEP lo hacemos de la siguiente manera:

Plugins	Op <u>t</u> ions	<u>W</u> indow	He	lp	
<u>1</u> Analyze This		۲	LEMTWH	C / K	
<u>2</u> Asm2Clipboard		•		Comment	
<u>3</u> Bookmarks		▶_			
<u>4</u> Break On Execution			•	Apply (for this session	n only)

De esta forma, al poner un BPM on Access, parará sólo por on Execution.

Y ponemos el BPM en la sección .CODE.

Test6 Test6		PE he	ader	Imag	R	RWE	
Test6 Test6 Test6 Test6 Test6 Test6 Test6 Test6	.rsrc .idata egemswqn lvwkkgjj .lol0 .lol1	code code code code code code	Actu View Dump Dump	alize in Disa p in CPI p		Enter	
			Sear	Ctrl+B			
			Set b	F2			
			Set n	nemory	/ breakpo	int on <u>a</u> cce	SS

Quitamos el BP del VirtualProtectEx, y damos F9 RUN.



Uppss...!!!

Algo pasó aquí! Listemos lo que observamos en estos momentos.

Paramos en un JMP indirecto de la IAT, y más abajo vemos la función de la VM de VB que nunca cambian la entrada o la emulan (al menos nunca lo he visto yo).

- EIP en un JMP indirecto que no es ThunRTMain.
- ThunRTMain, primera función a ejecutarse en un VB (normalmente).
- 0x0000, En todo OEP "normal" de VB, encontramos estos bytes después del JMP ThunRTMain, y antes del OEP, es decir en el medio.
- Normalmente antes del OEP se encuentra los bytes 0x0000, y los JMPs indirectos de la IAT.
- Stolen Bytes!, aquí debería ir la siguiente instrucción: PUSH ADDR
 - CALL ThunRTMain
 - Siendo ADDR un puntero a la cadena "VB5!" (Sin comillas).
- Luego de la instrucción CALL (del OEP) encontramos en un determinada longitud de bytes más abajo, el byte 0x30.

De todo esto podemos deducir: <u>Reconstruir el OEP!!!!</u>

Para poder reconstruir debemos estar parados en el JMP indirecto a la función ThunRTMain.

Reiniciamos.

Hacemos todos los pasos hasta llegar al Paso1 es decir poner un Bp En VirtualProtectEx y llegar hasta aquí \rightarrow

Address	Value	Comment	
0012F5D0	7C801AEC	CALL to VirtualProtectEx from kernel32.7C801AE7	
0012F5D4	FFFFFFFF	hProcess = FFFFFFFF	
0012F5D8	00401000	Address = Test6.00401000	
0012F5DC	000012B8	Size = 12B8 (4792.)	
0012F5E0	00000004	NewProtect = PAGE_READWRITE	
0012F5E4	0012FF04	pOldProtect = 0012FF04	

Nos vamos a la dirección JMP indirecto a la función ThunRTMain \rightarrow 004011E8h, y ponemos un HE.

Address	Hex dump	Disassembly		Comment	
004401146 004401162 004401182 004401182 004401182 004401104 004401104 004401104 004401104 004401100 004401102 004401203 004401203	- FF25 HC184000 - FF25 38184000 - FF25 70184000 - FF25 70184000 - FF25 20104000 - FF25 44104000 - FF25 44104000 - FF25 70104000 - FF25 781844000 - FF25 581844000 - FF25 58184400 - FF25 58184400 - FF25 58184400 - FF25 58184400 0000 0000 0000 0000 0000 0000 00	UHP DWURD PTR DS: [40104] UHP DWURD PTR DS: [40108] UHP DWURD PTR DS: [40108] UHP DWURD PTR DS: [401070] UHP DWURD PTR DS: [401060] UHP DWURD PTR DS: [401070] UHP DWURD PTR DS: [401070] UHP DWURD PTR DS: [401080] UHP DWURD	Backup Copy Binary Assemble Label Comment	msubum60.734E98E0 msubum60.734E99CF msubum60.734E99CF msubum60.734E99E2 59469564 59469564	* •
00401209 00401209 00401208 00401213 00401213 00401213 00401215 00401215 00401215 00401223 00401223 00401223 00401229 00401229 00401229	0000 0053 9E BA 128093F6 41 95 BD 512DD3A5 BD 512DD3A5 0000 0001 0000 000465 72 53 - 74 79 1098]=734235A4 (ms	HUD BYTE PTR DS:LEAAJ, F ADD BYTE PTR DS:LEAXJ, F ADD BYTE PTR DS:LEAXJ, F ADD EVTE PTR DS:LEAXJ, F NOV EDX, F693801E INC ECX, EBP MOU EBP, A5D32D51 TEST EAX, 8F ADD BYTE PTR DS:LEAXJ, F ADD BYTE PTR DS:LEAXJ, F ADD BYTE PTR DS:LEAXJ, F ADD BYTE PTR DS:LEAXJ, F ADD BYTE PTR SS:LEBP+721 PUSH EBX JF SHORT Test6,00401205 vbvm60.734235A4)	Breakpoint Run trace Eollow New origin here Go to Thread Follow in Dump	, Enter e Ctrl+Gray *	 <u>Ioggle</u> <u>C</u>onditional Conditional log <u>R</u>un to selection Memory, on <u>a</u>ccess Memory, on <u>w</u>rite <u>H</u>ardware, on execution

Hay 2 razones por lo que hacemos esto aquí.

- Primero al llegar al Punto1, ya fue desencriptado los bytes de la sección .CODE y la vemos como en la imagen anterior, si viéramos la sección antes del Punto1 encontraríamos basura.
- Cuando llegamos en el Punto1, ya pasamos la comprobación de los HBPs, y podemos poner uno sin ningún temor a que nos detecte.

F9 y RUN. Luego de terminar el último break de VirtualProtectEx haciendo referencia la sección .CODE (00405000h), rompe por el HE.

004011DC - FF25 48104000 004011E2 - FF25 50104000	JMP DWORD PTR DS:[401048] JMP DWORD PTR DS:[401050]	
004011E8 - FF25 98104000	JMP DWORD PTR DS:[401098]	msvbvm60.ThunRTMain
004011EE 0000	ADD BYTE PTR DS:[EAX], AL	
004011F0 895C24 40	MOV DWORD PTR SS:[ESP+40], EBX	
004011F4 v E9 CF300000	JMP Test6.004042C8	
004011F9 1800	SBB BYTE PTR DS:[EAX], AL	
004011FB 0000	ADD BYTE PTR DS:[EAX], AL	
004011FD 0000	ADD BYTE PTR DS:[EAX]. AL	
004011FF 0030	ADD BYTE PTR DS:[EAX], DH	

Si miramos el Stack

Address	Value	Commer	ıt		
0012FF28	004121FC	Test6.	.004121FC		
0012FF2C 0012FF30 0012FF34 0012FF38 0012FF3C 0012FF40 0012FF44	004013D4 73479E37 005126CA FFFE1000 004D97CD 9AAB76F4 F3342014	Test6. msvbvn Test6. Test6.	Address Show <u>A</u> SCII dump Show <u>U</u> NICODE dump	۶	
0012FF48 0012FF4C 0012FF50 0012FF54 0012FF58	0012FF5C 099E49A1 7C98E174 7C927559 004DB890	ntdll. <mark>RETURN</mark> Test6.	Copy to clipboard Modify	Ctrl+C	
0012FF5C 0012FF60 0012FF64 0012FF68 0012FF66 0012FF6C	004ACEDF 004ACEDB F3342014 0012FF7C 099E49A0 20996620	Test6. Test6.	Test6. Test6.	Edit Push DWORD Pop DWORD	Ctrl+E
0012FF74 0012FF78 0012FF78	0000006F 004D9667 008CCC6C	Test6.	Search for binary string	Ctrl+B	
0012FF80 0012FF84 0012FF88	00000000 F3342014 0012FF9C		Go to ESP <u>G</u> o to expression	* Ctrl+G	
			Eollow in Disassembler	Enter	
			Eollow in Dump		

Volcamos la dirección 004013D4h.

Comenzamos a reparar el OEP, haciendo lo siguiente.

004011E2 - FF25 50104000 004011E2 - FF25 98104000 004011E 0000 00401 95024 40 004011F0 895024 40 96401 9600 004011F4 E9 CF300000 96401 9600 004011F9 1800 9600 96401 9600 004011F9 0000 9600 96401 9600 004011F5 9000 9600 96401 9600 004011F5 9000 9600 96401 9600	JMP DWORD PTR DS:[401050] JMP DWORD PTR DS:[401098] ADD BYTE PTR DS:[EAX], AL MOV DWORD PTR SS:[ESP+40], JMP Test6.004042C8 SBB BYTE PTR DS:[EAX], AL ADD BYTE PTR DS:[EAX], AL ADD BYTE PTR DS:[EAX], AL ADD BYTE PTR DS:[EAX], AL ADD BYTE PTR DS:[EAX], AL	<u>B</u> ackup Copy Binary Assemble	msubum60.Ti	Space
004011FF 0030 00401201 0000 00401203 0038 00401205 0000 00401207 0000 00401207 0000 00401208 0053 9E 00401208 0053 9E 00401208 BA 1E8093F6 00401213 41	ADD BYTE PTR DS:[EAX], DH ADD BYTE PTR DS:[EAX], AL ADD BYTE PTR DS:[EBX], AL ADD BYTE PTR DS:[EBX-62], [MOV EDX, F693801E INC ECX	Assemble Label Comment Breakpoint Run trace		Space : ;
		New origin h	ere	Ctrl+Gra

Luego parchamos por estas instrucciones \rightarrow

004011D0 - FF25 481040	00 JMP DWORD PTR DS: [401048]	
004011E2 - FF25 501040 004011E8 - FF25 981040 004011EF 0000	00 JMP DWORD PTR DS:[401050] 00 JMP DWORD PTR DS:[401098] 00 BYTE PTR DS:[F9X], 0	msvbvm60.ThunRTMain
004011F0 68 D4134000 004011F5 E8 EEFFFFF	PUSH Test6.004013D4 CALL Test6.004011E8	JMP to msvbvm60.ThunRTMain

Luego hago un POP DWORD dos veces en el stack para que quede bonito ;)

Address	Value	Comm	ent		
0012FF28	004121FC	Test	C 00440456		1
0012FF2C	004013D4	Test	Address		í
0012FF30	73479E37	msvb			
0012FF34	005126CA	Test	Show <u>A</u> SCII dump		
0012FF38	FFFE1000		Show UNICODE dump		
0012FF3C	004D97CD	Test			
0012FF40	9AAB76F4		Lock stack		
0012FF44	F3342014				
D012FF48	0012FF5C		Construction of the base of	chile c	
0012FF4C	099E49A1		Copy to clipboard	Ctri+C	
0012FF50	7C98E174	ntdl	Modify		
0012FF54	7C927559	RETU	<u>H</u> odil y		7
0012FF58	004DB890	Test	Edit	Ctrl+E	
0012FF5C	004ACEDF	Test			
D012FF60	004ACEDB	Test	Push DWORD		
D012FF64	F3342014		Peo DWORD		
001 DEECO	00125520		FOP DWORD		

Quiten el BP a VirtualProtectEx, y para probar podemos hacer F9 RUN, y ver cómo corre pero NO lo haremos en estos momentos, primero le echaremos una miradita a los JMPs Indirectos.

Huditeps	Disassembly	Descritation
0040108A 0040109E 004011BE 004011F5 00401740	CALL E8417D91 CALL E84684A5 JMP DWORD PTR DS:[401044] CALL Test6.004011E8 CALL Test6.004011E8 CALL 02405755	(Initial CPU selection) msvbvm60.ThunRTMain
00401CEE	CALL DWORD PTR DS: [401028]	DS: F004010281=02F20892
00401CFF 00401D21 00401D21 00401D21 00401D20 00401D5D 00401D5D 00401D5D 00401D90 00401D97 00401D97 00401D92 00401D92 00401E79 00401E99 00401E99 00401E98 00401E98 00401E98 00401E98 00401E98 00401E98 00401E98	CALL DWORD PTR DS: [401028] CALL DWORD PTR DS: [40107C] CALL DWORD PTR DS: [401060] CALL DWORD PTR DS: [401044] CALL DWORD PTR DS: [40102C] CALL DWORD PTR DS: [401070] CALL DWORD PTR DS: [401070] CALL DWORD PTR DS: [40104C] CALL DWORD PTR DS: [401080] CALL DWORD PTR DS: [401082] CALL DWORD PTR DS: [401082] CALL DWORD PTR DS: [401083] CALL DWORD PTR DS: [401082] CALL DWORD PTR DS: [401082] CALL DWORD PTR DS: [401082] CALL DWORD PTR DS: [401083] CALL DWORD PTR D	DS: [00401028]=02E20892 DS: [0040107]=02E70000 DS: [00401060]=02E60000 DS: [00401064]=02E3073F DS: [0040102C]=02E208C0 DS: [0040102C]=02E208C0 DS: [00401070]=02E6095E DS: [00401070]=02E800FD DS: [00401080]=02E800FD DS: [00401080]=02E800FD DS: [00401080]=02E7088D DS: [00401080]=02E7088D DS: [00401080]=02E10408 DS: [00401080]=02E10408 DS: [00401080]=02E10409 DS: [00401080]=02E8016D DS: [00401080]=02E8016D DS: [0040108C]=02E8016D DS: [0040108C]=02E8016D DS: [0040108C]=02E8016D DS: [0040108C]=02E8016D DS: [0040108C]=02E8016D DS: [0040108C]=02E8016D DS: [0040108C]=02E8016D DS: [0040108C]=02E8016D DS: [0040108C]=02E8016D DS: [0040108C]=02E8002C DS: [00401080]=02E8002C
00401F07 00410FF7	CALL DWORD PTR DS:[401078] CALL 800AC093	DS:[00401078]=02E60ABC

Ufa!, si entramos a algún call vemos que esta aplicación usa CALLs indirectos de los que ocupan 6 bytes XD.

Bien con esta Info podemos darle F9 RUN, y vemos cómo hemos reconstruido de manera correcta el OEP \odot .



La debilidad que mencionaba antes es que al poner nosotros su VM es decir la DLL MSVBVM60.DLL en la misma carpeta que el proggie, le dará prioridad a esta y será cargada por el ejecutable en vez de la del sistema, lo podemos ver en la subventana Executables.

Para eso pongamos el proggie dentro de algún directorio, y dentro de ella la DLL MSVBVM60.DLL.

Cómo el packer emula las APIS, lo que haríamos es modificar el *head of the functions* de la DLL, haciendo un CALL hacia su misma dirección.

Hagamos una Prueba.



Una de las funciones muy muy usadas es la __vbaFreeStr.

Cargamos la DLL y hacemos la modificación a esa función.



Y ahí está, cambiamos, el mismo Olly nos da la referencia a la dirección que hemos hecho el CALL y el nombre de la función.

Guardamos los cambios y cargamos el proggie, llegando hasta el Punto2, no olvidándonos de sacar el HE por si no lo hemos hecho.

00101100	1160	11101000	0.0	Dwon D		00101010111		
004011C4 -	FF25	60104000	JMP	DWORD	PTR	DS: [401060]	_	
004011CA -	FF25	70104000	JMP	DWORD	PTR	DS: [40107C]		
00401100 -	FF25	28104000	JMP	DWORD	PTR	DS: [401028]		'
00401106 -	FF25	58104000	JMP	DWORD	PTR	DS: [401058]		
004011DC -	FF25	48104000	JMP	DWORD	PTR	DS: [401048]		
004011E2 -	FF25	50104000	JMP	DWORD	PTR	DS: [401050]		
004011E8 -	FF25	98104000	JMP	DWORD	PTR	DS: [401098]		ms
004011101		20101000	1.000	000110	1 111	00101010101		1.055

Ahora buscamos, en cual entrada está esa API jeje :P.

Address	Hex dump	Disassembly
004011A0	- FF25 AC104000	JMP DWORD PTR DS: [4010AC]
004011A6	- FF25 1C104000	JMP DWORD PTR DS: [40101C]
004011AC	- FF25 38104000	JMP DWORD PTR DS: [401038]
004011B2	- FF25 70104000	JMP DWORD PTR DS: [401038]

Bueno yo ahorro el trabajo, un FOLLOW

Address	Hex dump	Disassembly
02E80E64	E8 C75B6070	CALL msvbvm60vbaFreeStr
02E80E69	60	PUSHAD
02E80E6A	66:8BCA	MOV CX, DX
02E80E6D	B6 01	MOV DH, 1

Aja! Esto nos facilita todo, ahí podríamos irnos al ImportRec (previamente reparada el OEP) y usar la opción TraceLevel1 y obtendríamos que entrada es la correcta.

Ahora cómo no somos adivinos, no sabemos todas las APIs que son usadas en este proggie.

Para esto! Me cree un Script, que modifica el Head of Functions de la DLL según el rango que le ponga. ☺

Como la explicación de este Script escapa el objetivo de tute, ya que, el COMO HACERLO ya lo mencioné. Sólo haré unos pantallazos de lo que deben saber.

```
32 ADD AofName, bMSDLL
33 ADD AofFun, bMSDLL // Como es RVA.
34 SUB AofName, 4 // Rango FINAL?
35 MOV Val, OFF, 1
36
37 ADD AofFun, 50 //Rango de Inicio
```

Ahí tenemos el Rango de inicio y el final, no recomiendo tocar el rango final para nada, pero si el de inicio le diremos, que 50 bytes después de la primera función comience a remplazar el Head.

50h / 4h = 14h

Es decir las primeras 14 funciones no serán tocadas, todas las demás posteriores sí.

```
69 0xFFXX:
70 //MOV eip, Addy1
71 GN Addy1
72 LOG $RESULT_2
73 LOG Va1
74 MOV [Addy1], Va1, 1
75
76 DEC Va1
77
78 CMP AofFun, AofName
79 JZ Fina1
80 ADD AofFun, 4
81 JMP Bucle
```

No todas las funciones tienen cómo longitud de la misma 5 bytes, tamaño requerido para el CALL que rescribiremos, entonces lo que haremos es poner el valor de 0xFF, e irá disminuyendo según encuentre más funciones con menos de 5bytes. Para saber que funciones son, las logeo así que, nos vamos a esa ventana y ahí encontraremos esa información.

Sin más floro, abrimos la DLL MSVBVM60.DLL (que está en nuestro directorio), y mandamos el script!

Aceptamos el MSGBOX que aparece al terminar el script, y nos vamos a la ventana LOG.

COOHTHEO	Entry point of debugged bit
	<pre>\$RESULT_2:vbaVarVargNofree</pre>
	Val: 000000FF
	& DECHLT 2. Ebl (by south load
	AVESOFT_5: EDETDIaryon toad
	Val: 000000FE
	<pre>\$RESULT_2: VarPtr</pre>
	Val: 000000FD
	<pre>\$RESULT_2: _ad.j_fpatan</pre>
	Val: 000000FC
	<pre>\$RESULT_2: _ad.i_fptan</pre>
	Val: 000000FB

Y ahí están las funciones que se han escrito los valores indicados en la imagen.

Lo guardamos en un fichero de texto para no perderlo.

Ahora si miramos el directorio apareció la DLL que hemos modificado, la cambiamos al nombre original "MSVBVM60.DLL", y cargamos nuestro proggie.

Llegamos al Paso3 y hacemos el Paso4 por seguridad.



Ahí estamos si comenzamos a revisar los JMPs Indirectos, nos daremos cuenta que las primera línea está el CALL a la función que sería la correcta para cada entrada.

Para proseguir, debemos hacer el DUMP.

0	OllyDump - Test6.exe 👂										
	Start <u>A</u> d	dress:	40000)	<u>S</u> ize	: 20A	000				D <u>u</u> mp
	Entry Po	oint:	205D3	4	-> <u>M</u> odify	: 11F	0	<u>G</u> et EIP	as OE	P	Ca <u>n</u> cel
	Base of <u>C</u> ode: 1000			Base of	<u>D</u> ata:	3000					
	💌 <u>F</u> ix Ra	w Size	& Offset	of D	ump Image	:					
	Section	Virtual	Size	Virt	ual Offset	Raw	Size	Raw Off	set	Cha	aractaristics
		0001F	7000	000)01000	0001	-000	000010	00	EOC	000040
	.rsrc	00000)8C4	000	20000	0000	J8C4	0002000	00	C00	000040
	.idata 👘	00001	000	000)21000	00001000		0002100	00	C00	000040
		00101	000	00022000		00101000		00022000 E00		E00	000040
	egems 000C7000 0		001	00123000		000C7000		00123000 E00		000040	
	lvwkkaji 00001000 0		001	EA000	0000	1000	001EA0	00	E00	000040	
	.lol0	00009	30E 0	001	EB000	0000	30E 0	001EB0	00	600)00060
	.lol1	00014	171F	001	F5000	0001	471F	001F50	00	E20	000020
	Rebui	ld Impor	i.								

Luego abrimos el ImportRec, ponemos el OEP, 0x11F0 y damos IAT Auto Search.

IAT Infos needed								
0EP 000011F0	IAT AutoSearch							
RVA 00001000	Size 00000088							
Load Tree Save 1	ree Get Imports							

Ahí tenemos los datos correctos (aunque el Size es 0xB4, pero bue... los otros 4 bytes son puros ceros :)

Luego clic a "Get Imports".



Bueno... así es la vida

Desplegamos, y sombreamos todos desde el inicio.

→ ? FThunk:00001000 NbFunc:2D (decimal:45) valid:N0
 → rva:00001000 ptr:02E10000
 → rva:00001008 ptr:02E10169
 → rva:00001000 ptr:02E10231
 → rva:00001000 ptr:02E1042C
 → rva:00001010 ptr:02E104C2
 → rva:00001014 ptr:02E10570
 → rva:00001018 ptr:02E106D0
 → rva:0000101C ptr:02E107E0

Hasta el final.



Y Luego miramos como quedó \rightarrow



Jeje, muchos mas ponito no? ;)

Miramos el Log.

Current imports: 0 (decimal:0) valid module(s) 2D (decimal:45) imported function(s). (2 (decimal:2) unresolved pointer(s)) (added: -2A (decimal:-42))

Uuu.... Dos entradas sin resolver.

Subiendo...



Hay 1 una entrada ahí, seguimos para arriba...



2 entradas por resolver.

La primera \rightarrow RVA = 1004 y la VA 401004

Address Hex dump			Disassembly				
0040113A	-	FF25	04104000	JMP	DWORD	PTR	DS: [401004]
00401140	-	FF25	90104000	JMP	DWORD	PTR	DS: [40109C]

FOLLOW.

Address	Hex dump	Disassembly
02E10169	FB	STI
02E1016A	60	PUSHAD

Se acuerdan? El byte 0xFB, si lo miro en el log que guardé... por que lo guardaron no?

\$RESULT_2: _adj_fptan

Val: 00000FB

Y lo corregimos todo en el ImportRec, hacemos doble clic a la entrada mala.

IMPORT EDITOR	×
Module	
msvbvm60.dll	-
Function	
ord:02D1 name:_adj_fdivr_m32 ord:02D2 name:_adj_fdivr_m32i ord:02D3 name:_adj_fdivr_m64 ord:02D4 name:_adj_fpatan ord:02D5 name:_adj_fprem ord:02D6 name:_adj_fprem1 ord:02D7 name:_adj_fptan ord:02D8 name:_adj_fptan ord:02D8 name:_adj_fptan ord:02D8 name:_adj_fptan ord:02D8 name:_ord:02D8 name: ord:02D8 name: ord:02D8 name: ord:02D8 name: ord:02D8 name: ord:02D8 name: ord:02D8 name:	
ord:U2DC name: ord:02DD name:	~
Name adi_fptan	
RVA 00001004 OK C	ancel

Y clic al botón "OK".

⇒ ? FThunk:00001000 NbFunc:2D (decimal:45) valid:N0
 → rva:00001000 mod:msvbvm60.dll ord:009D name:_Clcos
 → rva:00001004 mod:msvbvm60.dll ord:02D7 name:_adj_fptan

Listo! Queda la última \rightarrow RVA = 0x104C VA= 40104C

Address	Hex dump	Disassembly				
00401128	- FF25 4C104000	JMP DWORD PTR DS:[40104C]				
0040112E	- FF25 64104000	JMP DWORD PTR DS:[401064]				
00401134	- FF25 18104000	JMP DWORD PTR DS:[401018]				
00401134	- FF25 04104000	.MP DWORD PTR DS:[401004]				

FOLLOW.

Address	Hex dump	Disassembly
02E20916	FC 🛃	CLD
02E20917	60	PUSHAD
02E20918	E9 0F000000 2050 01	CMP 62E2092C

Y ese valor es de la API...

\$RESULT_2: _adj_fpatan

Val: 00000FC

Hacemos el mismo procedimiento al anterior, y lo agregamos en el ImportRec.



Listo entrada buena!

Current imports: 1 (decimal:1) valid module(s) (added: +1 (decimal:+1)) 2D (decimal:45) imported function(s). (0 (decimal:0) unresolved pointer(s)) (added: -1 (decimal:-1)) Congratulations! There is no more invalid pointer, now the guestion is: Will it work? :-)

Ahí el log nos dice que todo está de poca 😊

Yo voy a adjuntar el Log del ImportRec, por si las caiguas y ya ustedes sólo le queda modificar el path.

Fixiando

Fixiamos nuestro DUMP, lo saco fuera del directorio para que no tome su DLL modificada.

Doble Clic y... tan tan, ni que cosas ahí está el screenshot \rightarrow



Unpack resuelto, miremos el peso del Unpacked.

	Dumpexe
Tipo de archivo:	Aplicación
Descripción:	Dump_
Ubicación:	C:\Documents and Setting:
Tamaño:	2.04 MB (2,142,208 bytes)
Tamaño en disco:	2.04 MB (2,142,208 bytes)

Vaya! Muy pesado, ahora miremos el Packed.

	Test6.exe
Tipo de archivo:	Aplicación
Descripción:	Test6
Ubicación:	C:\Documents and Setting:
Tamaño:	1.03 MB (1,089,536 bytes)
Tamaño en disco:	1.03 MB (1,089,536 bytes)

Cómo que muy pesado no?, pues agradecimientos a mi amigo Eddy, por el tip de sacarles las secciones que no usa.

Para eso abrimos el SirPE.

🕅 PE	PE Editor - Dump_,exe - [PE-32]											
Option	Dptions Data Header											
[Section Name	RVA	VSize	ROffset	RSize	P. Reloc	N. Reloc	P. LineNumber	N. LineNum	Flags		
		00001000	0001F000	00001000	0001F000	00000000	0000	00000000	0000	E0000040		
	.rsrc	00020000	000008C4	00020000	000008C4	00000000	0000	00000000	0000	C0000040		
	.idata	00021000	00001000	00021000	00001000	00000000	0000	00000000	0000	C0000040		
		00022000	00101000	00022000	00101000	00000000	0000	00000000	0000	E0000040		
	egemswqn	00123000	000C7000	00123000	000C7000	00000000	0000	00000000	0000	E0000040		
	lvwkkgji	001EA000	00001000	001EA000	00001000	00000000	0000	00000000	0000	E0000040		
	.lol0	001EB000	000090E0	001EB000	000090E0	00000000	0000	00000000	0000	60000060		
	.lol1	001F5000	0001471F	001F5000	0001471F	00000000	0000	00000000	0000	E2000020		
	.mackt	0020A000	00001000	0020A000	00001000	00000000	0000	0000000	0000	E0000060		

Tenemos los siguientes, y debemos borrar las secciones dando clic al botón "*Remove Section*" y luego al botón "*Align All to Virtual*", para que quede así \rightarrow

Section Name	RVA	VSize	ROffset	RSize	P. Reloc	N. Reloc	P. LineNumber	N. LineNum	Flags
.Nox	00001000	0001F000	00001000	0001F000	00000000	0000	00000000	0000	E0000040
.ISIC	00020000	00001000	00020000	00001000	00000000	0000	00000000	0000	C0000040
.mackt	0020A000	00001000	0020A000	00001000	00000000	0000	00000000	0000	E0000060

Recordemos que este packer usa la TLS Table para comenzar a desempacar.

			Stablish	
TIsTable	0000000	00000000	Set to Zero	
			SectoZeio	

Clic a la Tls Table –el ítem- y luego nos dirigimos a la parte inferior del SirPE, cliqueamos el botón "Set to Zero", y luego al botón "Stablish", para finalmente guardar los cambios.

Hacemos clic derecho al proggie para ver si en algo a cambiado el peso.

	Dumpexe
Tipo de archivo:	Aplicación
Descripción:	Dump_
Ubicación:	C:\Documents and Setting:
Tamaño:	2.04 MB (2,142,208 bytes)
Tamaño en disco:	2.04 MB (2,142,208 bytes)

No nada, y claro falta hacerle un ReBuild, abrimos el LordPE, elegimos el proggie unpacked, y le damos con todo jeje.

[Rebuild Status]	
Starting to rebuild Dumpexe Filesize: 20B000h	OK
Dumpfixdone Realigningdone Current filesize: 1FB0Ah File minimized to: 6% Validate PE imagedone	
New filesize: 1FBQAh File minimized to: 6% Rebuilding finished.	٢

Ejecutamos!



Sin ningún problema.

Clic derecho para mirar el peso del Unpacked!

	Dump.exe
Tipo de archivo:	Aplicación
Descripción:	Dump
Ubicación:	C:\Documents and Setti
Tamaño:	127 KB (130,314 bytes)
Tamaño en disco:	128 KB (131,072 bytes)

Je! disminuyó inmensamente.

Adjunto en la descarga:

Modifica - Head of the Functions.txt (Script que modifica a la MSVBVM60.DLL).

Log - ImportRec.txt (Log de todas las APIS de este proggie sólo basta con cambiar el path y darle al botón Load Tree, y se evitan el laburo en el IR).

FunVal.txt (El Log que guardamos de la DLL MSVBVM60.DLL)

MSVBVM60_MODIFICADA.DLL (Bueno ya saben x).

Test6.exe (Proggie enpaquetado)

Unpacked + Fixed.exe (reparado, sin basura, y menos peso ©).

Y este Tute!.

Hay muchas cosas más que debería agregar y otras que estoy pasando por alto de seguro, pero bueno, creo que con lo explicado se cumple el objetivo, espero que le haya gustado el ortodoxo y humilde Unpacking!

Gracias por la lectura!

28 de Enero del 2012. Nox.