



Victima	ChmEditor 1.3
Protección	Asprotect (15 días)
Herramientas	Olly,
Objetivo	Quitar la protección de tiempo de Asprotect

Introducción

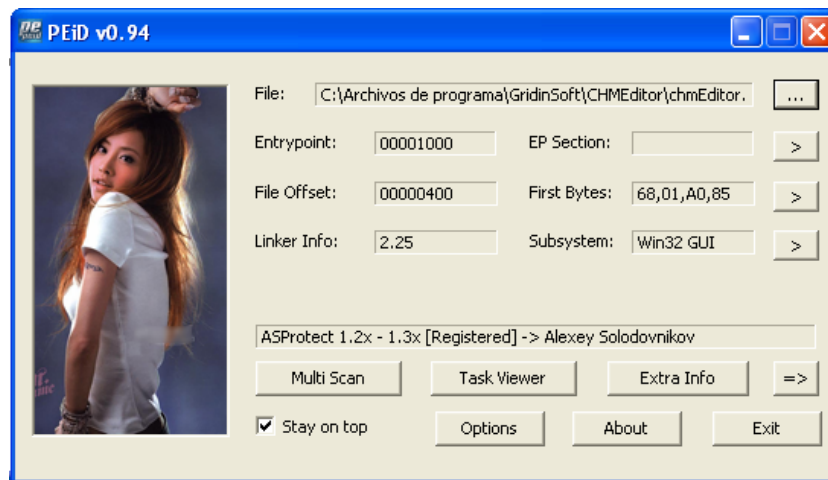
Hola a todos, después de un tiempo volvemos a la carga. En esta entrega vamos a ver como saltar la protección de tiempo de un Asprotect de los últimos.

Al parecer este packer sigue evolucionando, en este programa en concreto he visto complejo obtener un desempacado limpio, pero saltar el control de tiempo veremos que es muy sencillo.

Lo mejor de todo es que la protección de tiempo corre a cuenta del packer y muy posiblemente sea idéntica para todos los programas comprimidos con este bicho.

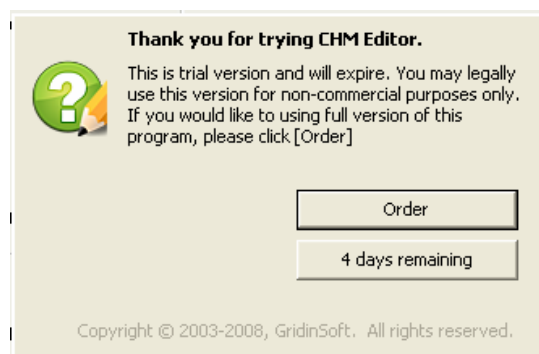
Conociendo al enemigo

Si pasamos el programa por el PEID vemos esto:



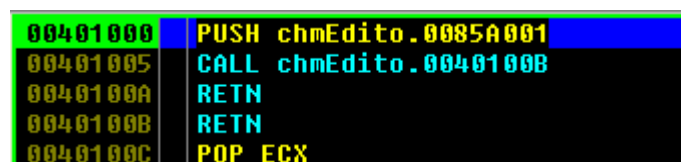
Como este PEID es viejo y no tengo mantenido las firmas pues lo único que podemos sacar en claro es la protección pero no su versión.

Si ejecutamos el programa tenemos esto:



En este caso el programa es totalmente funcional dentro del período de pruebas

Veamos lo en el Olly



Bueno esto nos confirma que es Asprotect, tiene la estructura típica estructura del PE.

Este Asprotect es de los últimos y no responde al truco de las excepciones, lo más parecido es el tuto de Solid *Asprotect 2.1 - 2.2 SKE - peleando contra el mutante_por Solid teoría 1111* en él Solid toma el truco de poner un BP en una API que se llame cerca del OEP. Hay muchos métodos pero el de poner un BP en el ret de la API GetModuleHandleA en este caso es muy rápido, hagámoslo.

Y vamos pasando con F9 hasta ver algo como esto en la pila

0012FF7C	00407949	RETURN to chmEdito.00407949 from chmEdito.00407874
0012FF80	00000000	
0012FF84	00400000	ASCII "MZP"
0012FF88	006BBC20	RETURN to chmEdito.006BBC20 from chmEdito.00407938
0012FF8C	0088D000	chmEdito.0088D000

Como vemos esta llamada se realizo desde la sección de código vallamos a ver que tenemos.

00407938	PUSH EBX
00407939	MOV EBX,EAX
0040793B	XOR EAX,EAX
0040793D	MOV DWORD PTR DS:[6BC798],EAX
00407942	PUSH 0
00407944	CALL chmEdito.00407874
00407949	MOV DWORD PTR DS:[6E57F8],EAX
0040794E	MOV EAX,DWORD PTR DS:[6E57F8]
00407953	MOV DWORD PTR DS:[6BC7A4],EAX
00407958	XOR EAX,EAX
0040795A	MOV DWORD PTR DS:[6BC7A8],EAX
0040795F	XOR EAX,EAX
00407961	MOV DWORD PTR DS:[6BC7AC],EAX
00407966	CALL chmEdito.0040792C
0040796B	MOV EDX,chmEdito.006BC7A0
00407970	MOV EAX,EBX
00407972	CALL chmEdito.00404D44
00407977	POP EBX
00407978	RETN

Aparentemente tiene poca pinta de OEP, veamos la otra dirección la 6BBC20

006BBC10	PUSH EBP
006BBC11	MOV EBP,ESP
006BBC13	ADD ESP,-10
006BBC16	MOV EAX,chmEdito.006B9320
006BBC1B	CALL chmEdito.00407938
006BBC20	XOR EAX,EAX
006BBC22	PUSH EBP
006BBC23	PUSH chmEdito.006BBC6B

Esto es otra cosa y parece un Delphi, vemos como se encuentra la sección code.

Address	Hex dump	ASCII
00401000	04 10 40 00 03 07 42 6F	■@.■Bo
00401008	6F 6C 65 61 6E 01 00 00	olean■..
00401010	00 00 01 00 00 00 00 10	..■....■
00401018	40 00 05 46 61 6C 73 65	@.■False
00401020	04 54 72 75 65 8D 40 00	■True■@.

Pues si esto confirma que estamos ante un Delphi y que lo que parece el OEP es la dirección **6BBC10**.

Bueno hasta aquí parece todo correcto pero si nos da por comprobar en que sección se encuentra nuestro OEP tenemos una sorpresa.

00400000	00001000	chmEdito		PE header	Imag	R	RWE
00401000	002B9000	chmEdito		code	Imag	R	RWE
006B0000	00002000	chmEdito		data	Imag	R	RWE
006B0000	000027000	chmEdito			Imag	R	RWE
006E3000	00028000	chmEdito			Imag	R	RWE
0070B000	00005000	chmEdito			Imag	R	RWE
00710000	00001000	chmEdito			Imag	R	RWE
00711000	00001000	chmEdito			Imag	R	RWE
00712000	00001000	chmEdito			Imag	R	RWE
00713000	000031000	chmEdito			Imag	R	RWE
00744000	00116000	chmEdito	.rsrc	resources	Imag	R	RWE
0085A000	0006B000	chmEdito	.data	imports,rel	Imag	R	RWE
008C5000	00001000	chmEdito	.adata		Imag	R	RWE

Parece que esta en la sección data. Si os fijáis este packer ha dejado muchísimas secciones he inicialmente estaban todas con Lectura/Escritura/Ejecución, vamos a verlo con el LordPE

[PE Editor] - c:\archivos de programa\gridinsoft\chmeditor\chmeditor.exe [READ...

Basic PE Header Information

[Section Table]

Name	VOffset	VSize	ROffset	RSize	Flags
	00001000	002B9000	00000400	00101000	E0000040
	002BA000	00002000	00101400	00001000	E0000040
	002BC000	00027000	00102400	00011400	E0000040
	002E3000	00028000	00113800	00000000	E0000040
	0030B000	00005000	00113800	00004200	E0000040
	00310000	00001000	00117A00	00000200	E0000040
	00311000	00001000	00117C00	00000000	E0000040

Magic: 010B NumDirvaAndSizes: 00000010

Podemos ver que muchas de las secciones no están en el disco físicamente, ya que tiene un RSize de 0, es decir solo existen en memoria.

Personalmente pienso que es un truco sucio del Asprotect para despistar, que haya subdivido la sección .code real en 2, pero bueno son simples suposiciones. Solid me lleo a comentar otro punto que el pensaba que podía ser más OEP ya que pensaba que no estaba totalmente desempacado la sección de código donde yo lo tome.

Como esto es un Delphi según nuestro OEP tenemos que tener una instrucción del tipo:

mov eax, DIRECCION DE LA INIT TABLE

En este caso es mov eax, 006B9320. Vamos a revisar como anda

006B9320	DB 7F
006B9321	DB 01
006B9322	DB 00
006B9323	DB 00
006B9324	DD chmEdito.006B9328
006B9328	DB 00
006B9329	DB 00
006B932A	DB 00
006B932B	DB 00
006B932C	DB 00
006B932D	DB 00
006B932E	DB 00
006B932F	DB 00
006B9330	DD chmEdito.006BA000
006B9334	DD chmEdito.00407814
006B9338	DB 00
006B9339	DB 00

Como podemos ver esta totalmente destrozada.

¿Y la IAT como estará?

00407871	LEA EAX,DWORD PTR DS:[EAX]	
00407874	CALL 00FE0004	
00407879	DB DB	
0040787A	MOV EAX,EAX	
0040787C	JMP DWORD PTR DS:[70BDF0]	kernel32.LocalAlloc
00407882	MOV EAX,EAX	
00407884	JMP DWORD PTR DS:[70BDEC]	kernel32.TlsGetValue
0040788A	MOV EAX,EAX	
0040788C	JMP DWORD PTR DS:[70BDE8]	kernel32.TlsSetValue
00407892	MOV EAX,EAX	

Podemos ir viendo los famosos CALL del Asprotect de los cuales Solid (entre otros) ha hecho unos cuantos script para Olly para su reparación.

Address	Value	Comment
0070BDB8	7C813559	kernel32.FindFirstFileA
0070BDBC	AAB64240	
0070BDC0	7C81CAA2	kernel32.ExitProcess
0070BDC4	C94E5CBE	
0070BDC8	7C81082F	kernel32.CreateThread
0070BDCC	7C80D293	kernel32.CompareStringA
0070BDD0	D556BCA2	
0070BDD4	7C862B8A	kernel32.UnhandledExceptionFilter
0070BDD8	7C947A40	ntdll.RtlUnwind
0070BDDC	7C81EAE1	kernel32.RaiseException
0070BDE0	EE1F0540	
0070BDE4	00000000	
0070BDE8	7C809BF5	kernel32.TlsSetValue
0070BDEC	7C809750	kernel32.TlsGetValue
0070BDF0	7C8099BD	kernel32.LocalAlloc
0070BDF4	59509FE6	
0070BDF8	00000000	
0070BDFC	77D21AD5	USER32.CreateWindowExW
0070BE00	77D2190B	USER32.CreateWindowExA
0070BE04	77D1C57E	USER32.WindowFromPoint

Por otra parte vemos que la IAT también está completamente destruida.

En este punto intenté con algunos script la reparación de la IAT pero sin éxito y teniendo en cuenta la cantidad de secciones que tiene me rondaba la cabeza que pudiera tener una VM, así que opte por olvidar el desempacado y centrarme en la comparación de tiempo.

Buscando la zona mágica

En este punto estuve un poco dando palos de ciego. Si vemos como arranca el programa aparece su Splash screen, luego la ventana principal del programa y un startup con los proyectos abiertos recientemente donde tenemos el cartel de UNREGISTER VERSION. Luego en un breve espacio de tiempo aparece nuestra NAG.

Con este proceso la verdad es que pensaba que la NAG pertenecía al programa y que por lo tanto la comprobación del tiempo aparecería después de nuestro OEP.

Con este razonamiento una vez parado en el OEP puse un parte de HE (hardware breakpoint) en GetLocalTime y GetSystemTime, pero sin resultado alguno. Siendo un Delphi llege a pensar que usaran algún método de la VCL que terminara tomando el tiempo de alguna otra forma.

Nada más lego de la realidad en unos de los reinicios al partir del EP me para aquí.

0012FEE0	00894829	CALL to GetLocalTime from chmEdito.00894824
0012FEE4	0012FEF0	pLocaltime = 0012FEF0

Es decir llamado desde el código de Asprotect. Vamos a seguirlo. Para ello visualizamos la dirección donde guardará los datos. En el commandline hacemos d 0012FEF0 y saltamos la API hasta llegar aquí

00894824	E8 F7E0FFFF	CALL chmEdito.00892920	JMP to
00894829	66:804C24 0E	MOV CX,WORD PTR SS:[ESP+E]	Día
0089482E	66:8B5424 0A	MOV DX,WORD PTR SS:[ESP+A]	mes
00894833	66:8B4424 08	MOV AX,WORD PTR SS:[ESP+8]	año
00894838	E8 07FEFFFF	CALL chmEdito.00894644	

La idea es seguirle la pista.

Tras muchísimas vueltas y operaciones llegamos a la siguiente dirección

0089961C	PUSH EBX
0089961D	PUSH ESI
0089961E	PUSH EDI
0089961F	ADD ESP,-40
00899622	MOV ESI,EDX
00899624	LEA EDI,DWORD PTR SS:[ESP+20]
00899628	PUSH ECX
00899629	MOV ECX,7
0089962E	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00899630	POP ECX
00899631	MOVS WORD PTR ES:[EDI],WORD PTR DS:[ESI]
00899633	MOV ESI,EAX
00899635	LEA EDI,DWORD PTR SS:[ESP]
00899638	PUSH ECX
00899639	MOV ECX,8
0089963E	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00899640	POP ECX
00899641	MOV EBX,ECX
00899643	MOV EAX,EBX
00899645	MOV EDX,35
0089964A	CALL chmEdito.00892AA4
0089964F	MOV EAX,DWORD PTR SS:[ESP+28]

Este es el principio de la rutina que se encarga del control del tiempo del Asprotect.

008996D1	POP EAX		
008996D2	POP EDX		
008996D3	MOV ESI,EAX		
008996D5	INC ESI		
008996D6	MOV WORD PTR DS:[EBX+A],SI		
008996DA	MOV AX,WORD PTR SS:[ESP+8]		
008996DF	CMP SI,AX		
008996E2	JBE SHORT chmEdito.008996EC		
008996E4	MOV WORD PTR DS:[EBX+C],0		
008996EA	JMP SHORT chmEdito.008996F4		
008996EC	SUB AX,SI		

Registers (FPU)	
EAX	0000000F
ECX	0000027F
EDX	00000000
EBX	008BA710
ESP	0012FE88
EBP	0012FF78
ESI	0000001C
EDI	0012FEA8
EIP	008996E2

En este punto se comprueban los valores AX con SI.

En AX tenemos el valor F (15 en decimal) y en SI 1C (28 dec). En este punto modifique el reloj para que aparezca como terminado el tiempo de prueba.

En ese caso se ejecuta la instrucción 8996E4 poniendo una variable a 0.

Si provocamos el salto seguimos traceando y llegamos a esta zona

008997D9	PUSH DWORD PTR SS:[ESP+18]
008997DD	PUSH DWORD PTR SS:[ESP+18]
008997E1	PUSH DWORD PTR SS:[ESP+3C]
008997E5	PUSH DWORD PTR SS:[ESP+3C]
008997E9	CALL chmEdito.008995A4
008997EE	TEST AL,AL
008997F0	JNZ SHORT chmEdito.00899818
008997F2	PUSH DWORD PTR SS:[ESP+18]
008997F6	PUSH DWORD PTR SS:[ESP+18]
008997FA	PUSH DWORD PTR SS:[ESP+34]
008997FE	PUSH DWORD PTR SS:[ESP+34]
00899802	CALL chmEdito.008995A4
00899807	TEST AL,AL
00899809	JNZ SHORT chmEdito.00899818
0089980B	CALL chmEdito.00897A38
00899810	TEST AL,AL
00899812	JE SHORT chmEdito.00899818
00899814	XOR EAX,EAX
00899816	JMP SHORT chmEdito.0089981A
00899818	MOV AL,1
0089981A	MOV BYTE PTR DS:[EBX+31],AL
0089981D	ADD ESP,40
00899820	POP EDI
00899821	POP ESI
00899822	POP EBX
00899823	RETN

En esta zona se comprueba que el reloj no se haya atrasado entre otras cosas, si todo va bien debe de llegar a la zona 899814 XOR EAX,EAX.

En definitiva tenemos 2 cambios a realizar para que se crea que siempre le quedan los 15 días de evaluación.

Modificamos la dirección **8996D3 por XOR SI,SI**

Y el salto de **8997F0 lo cambiamos por un JMP 899814**

Con esos simples cambios ya saltamos el control de tiempo.

El control de CRC

Ahora como siempre viene un pero. Si hacemos los cambios en caliente con el Olly y damos a RUN el programa termina en una excepción.

En antiguos Asprotect el CRC se comprobaba abriendo el fichero original y comprobando con lo que hay en memoria, en este caso solo trabaja con la memoria.

Vamos a poner un HW on Access sobre los bytes modificado de la dirección 8996D3 y veamos donde los lee y nos para aquí

00896B6B	8A5C0A FF	MOV BL, BYTE PTR DS:[EDX+ECX-1]
00896B6F	C1E3 18	SHL EBX, 18
00896B72	31D8	XOR EAX, EBX
00896B74	B3 08	MOV BL, 8
00896B76	D1E0	SHL EAX, 1
00896B78	73 05	JNB SHORT chmEdito.00896B7F

En BL se tiene el primer byte (66)

Sigamos la ejecución traceando con cuidado.

008998CB	310424	XOR DWORD PTR SS:[ESP], EAX	Registers (FPU) EAX 0088D000 chmEdito.0088D000 ECX 05C4972D EDX 00000194 EBX 008A03C4 chmEdito.008A03C4 ESP 0012FF7C EBP 0012FF98 ESI 00E60000
008998CE	8B05 D4948B00	MOV EAX, DWORD PTR DS:[8B94D4]	
008998D4	010424	ADD DWORD PTR SS:[ESP], EAX	
008998D7	C3	RETN	
008998D8	C3	RETN	
008998D9	8D40 00	LEA EAX, DWORD PTR DS:[EAX]	
EAX=0088D000 (chmEdito.0088D000)			
Stack SS:[0012FF7C]-59EC5D00			
Address	Hex dump	ASCII	
008996D3	AA 33 FA 66 89 73 0A 66	F20F5.F	0012FF7C 59EC5D00 0012FF80 0012FFE0 Pointer to next SEH record

Tras varias operaciones con los bytes de la zona llega a esta dirección en donde si nos fijamos en el ESP tenemos el numeraco de la operación, en EAX se da el valor de una dirección de Asprotect (a saber si es fijo o calculado). Al sumar los 2 debe de dar la dirección siguiente para seguir con la ejecución, como los datos no solo los correctos porque hicimos el parche se provoca la excepción.

Bueno esto tampoco parece que sea un gran problema, podemos intentar que los cambios vengan ya puestos con un inline parching, y luego al finalizar la ejecución intentar reponer los valores originales antes de que pase al CRC y listo.

Así que lo siguiente que queda sería seguir el tuto de Aboslom1 *Inline Patch Asprotect como si fuese un UPX*.

La técnica es sencilla pero muy laboriosa, pones en la sección un HW on WRITE y vamos tirando del hilo hasta dar con la zona dentro del programa donde se encuentra los bytes que seguramente estén encriptados.

Personalmente buscando me perdí y tras 1 hora, y viendo los cambios a realizar opté por probar el hacer un Loader con mi componente DebuggerCTRL en MASM ya que este Asprotect al menos no tiene chequeo de debugger jejeje.

El Loader

Como loader la cosa es aun más sencilla, lo que haremos es poner 2 HW on execute, una en la dirección 8996D3h en el mov esi,eax y cambiaremos el valor de EAX a 0 (NO TOCAMOS EL CODIGO SOLO EL VALOR DEL REGISTRO).

Por último otro HW on execute en la dirección 8997F0h el salto y cambiamos el EIP a la dirección 899814h, es decir al XOR EAX,EAX

Para empezar el código debe de tener los siguientes incluye

```
.386
.model flat, stdcall ;32 bit memory model
option casemap :none ;case sensitive

include Loader.inc

include ..\Comunes\Funciones.inc ; <= Requerido por el componente
include ..\Comunes\ListaDoble.inc ; <= Requerido por el componente
include ..\Comunes\DebuggerCTRL.inc ; <= EL componente
.code

start:

    invoke GetModuleHandle,NULL
    mov     hInstance,eax

    invoke InitCommonControls
    invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,addr DlgProc,NULL
    invoke ExitProcess,0

;#####

DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
    LOCAL MyD:DWORD
    mov     eax,uMsg
    .if eax==WM_INITDIALOG
        invoke ShowWindowAsync,hWin,SW_HIDE
        invoke CrearDepurador,addr szVictima,NULL,NULL,hWin,NULL,NULL
        .if eax != NULL
            mov hDBG,eax ; <= Se crea el Depurador bien
        .else
            invoke MessageBox,NULL,NULL,NULL,MB_OK ; <= Error creando el depurador
        .endif
    .elseif eax == WM_CREATEPROCESS_DBG ; 1º Mensaje enviado por el depurador
        invoke GetProcessEntryPoint,hDBG ; Obtenemos la dirección del EP
        invoke SetHWBreakPointDBG,hDBG,eax,HW_EJECUCION,HW_BYTE ; Ponemos un HW on Execute en el EP
    .elseif eax == WM_HW_BREAKPOINT_DBG ; Mensaje que se genera cuando se alcanza un HW
        invoke GetProcessEntryPoint,hDBG
        mov ebx,wParam
        .if ebx == eax
            ; Estamos en el EP
            invoke BorrarHWBP_DBG,hDBG,eax ; borramos el HW del EP
            invoke SetHWBreakPointDBG,hDBG,8996D3h,HW_EJECUCION,HW_BYTE ; Cambiar EAX a 0
            invoke SetHWBreakPointDBG,hDBG,8997F0h,HW_EJECUCION,HW_BYTE ; Cambiar el salto
        .elseif ebx == 8996D3h
            invoke BorrarHWBP_DBG,hDBG,8996D3h
            invoke Set_EAX,hDBG,0
        .elseif ebx == 8997F0h
            invoke BorrarHWBP_DBG,hDBG,8997F0h
            invoke Set_EIP,hDBG,899814h
        .endif
    .elseif eax == WM_EXITPROCESS_DBG ; Se cerró el programa víctima
        invoke PostMessage,hWin,WM_CLOSE,0,0 ; Importante que sea PostMessage para dar tiempo a cerrar el debugger
    .elseif eax==WM_CLOSE ; se ha cerrado el Loader
        .if hDBG != NULL
            invoke CerrarDepurador,hDBG
        .endif
        invoke EndDialog,hWin,0
    .else
        mov     eax,FALSE
        ret
    .endif
    mov     eax,TRUE
    ret
DlgProc endp
end start
```

Fichero Loader.inc

```
include advapi32.inc ; <= requerido por el control para escalar privilegios
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc

includelib ADVAPI32.LIB
includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib

DlgProc                                PROTO    :HWND,;UINT,;WPARAM,;LPARAM

.const

IDD_DIALOG1                            equ 101

IDC_BTN1 equ                            1001

;#####

.data

szVictima db "victima.exe",0 ;<=Nombre del proceso víctima

hDBG      dd 0
.data?

hInstance dd ?

;#####
```

Sencillo verdad, pues esto fue todo.

Seguramente podamos sacar una firma a la zona del control de tiempo y casi seguro que los Asprotect de esta versión caen igual sin requerir desempacado jejeje.

Bueno hasta la próxima.

