

CANU



Un simple unpackme, que hizo aprender bastante y deja misterios aun, publicado en el 2005, analizado por varios años, según se va aprendiendo, se va acotando.

fecha concurso : 15/10/2005 nivel 55/ opción 4 nombre: Can U

Escrito con Fines Educacionales

saludos Apuromafo



Introduccion

Bienvenidos una vez más a un escrito de otro execryptor, en este caso tenemos un crackme, o más bien un unpackme,

El tema especial a ver es que está protegido, con un protector comercial pero además le agregaron los famosos SDK o, componentes adicionales, Esto es para hacerlo más difícil.

Este unpackme se llama CANU, que significa en modo abreviado CAN YOU, traducción: "¿Puedes tú?", como ya lleva tiempo esta aplicación, me hizo reconocer muchos detalles en Execryptor, sobre todo saber que puede desempacar pero el tema es el post-unpacked, Como revisar un programa con virtualización o como buscar algún detalle importante para detener una bomba, que claramente nos presenta el autor, aquí comenzamos una hazaña más,

Información del concurso

Fecha concurso: 15/10/2005

Nivel 55/op 4

Nombre: Can U

Esperando se animen a practicar algún día con Canu, yo creo que esta vencido, pero a ver si me ayudan a encontrar los detalles que no encontré, les comparto este escrito, con fines educativos.

Herramientas:

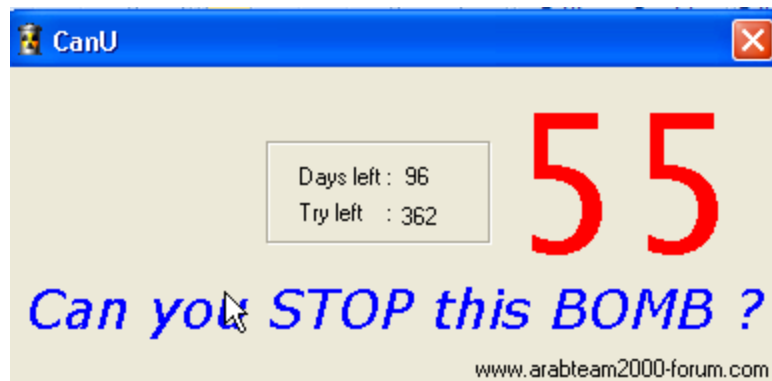
- 0) Pescan: Exeinfope, RDG Detector, PEid/sabueso/Quickunpack, ProtectionID, etc.
- 2) depurador para análisis: Ollydbg, plugins varios entre ellos phantom, Codedoctor, StrondOD.
- 3) Tools para chequeos y apoyo al tiempo: Trial Reset, Run AS date, programas hechos en Delphi,
- 4) Editor /Rebuilding :Pexplorer, LordPE, ImportREc 1.6/1.7,
- 5) Internet (lecturas),
- 6) Canu, el Objetivo de hoy



Contenido

Caracteristicas Generales.....	4
Evitar la Bomba / No Bomb:.....	5
Comenzando: Analisis packer	6
Encontrando los mensajes del tiempo.....	7
Trial Reset:	8
Lectura:	9
Lecturas en chino	9
Lecturas en español/Ingles	9
Comenzando en Olly, las primeras detecciones.	10
Mas mensajes	11
Configurando Olly 1.1	12
Parametro Rol r32,const	55
TLS.....	16
Descompresion:	27
Parametros OR, ROL, ROR,.....	38
ror ->encripta rol->decripta	39
Apis emuladas	42
Realizando el Inline	45
Restaurando el OEP.	50
Parametro Call offset, push eax, ror r32,const =call api	53
<i>Script para la iat</i>	56
Venciendo a la bomba	58
Trial Days.....	59
Inline patch para trial days.....	61
Clock Manipulation	64
Patron Push /pop/{random}Rol=mov r32,api.....	39
Proof Concept Trial Limit	67
Anexo:	77
Variable ebp-c en eax.....	77

Características Generales



Vemos

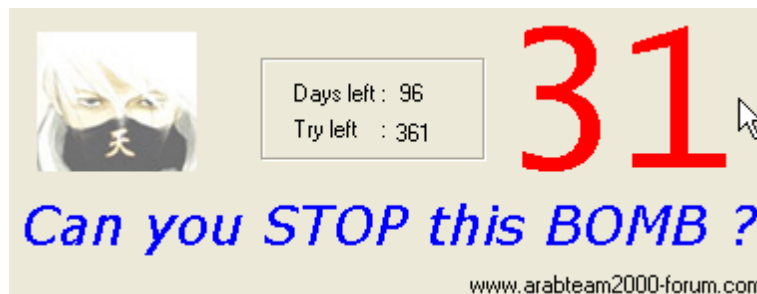
Título: CanU

Indicadores: 1->Days left=días restantes // 2->Try Left->intentos de ejecución restantes //

Tiempo: 3->reloj de 60 a menor, ahora está en 55

Mensajes en el Tiempo:

Luego a ciertos segundos aparece un ninja, quizás de la historia de naruto.



En los últimos segundos, muestra un mensaje en la parte inferior, cerrándose la aplicación al llegar al cero.

It's possible but i bet you can't-> traducción: "esto es posible, pero yo te reto a que tú no puedes."



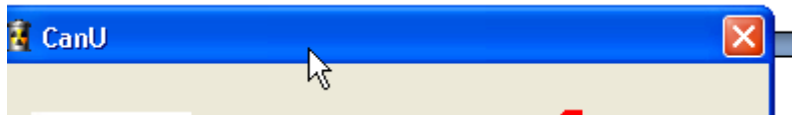
Evitar la Bomba / No Bomb:

¿Cómo podemos Vencer sin uso de herramientas? Una forma de ejecutarlo y que no rompa la bomba

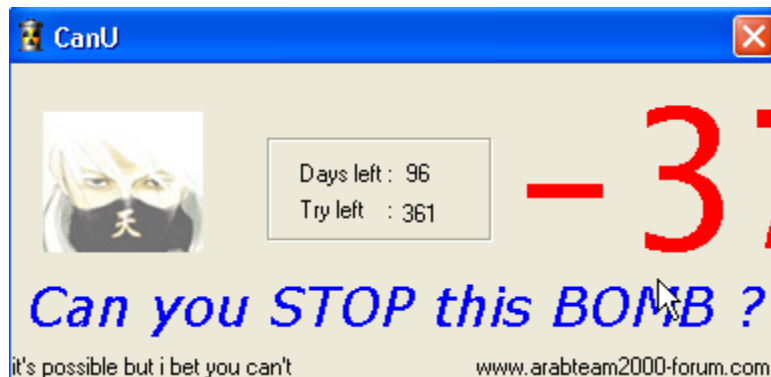
(Se cierre en el cero)

Si cada segundo que pasa mueve un handle, o si cambio de posición algo debe influir, exccryptor tiene muchos thread, veamos si hay algo nuevo, una vez moví, y me di cuenta de esto.

Es pulsar la ventana, moverla como si estuviera sujetado. Cuando este en los últimos segundos,



Y vemos claramente como del 1 pasa a números negativos y por mucho esperar y el contador claramente tiene bug y es un timer.



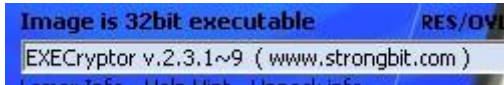
Como no explota, debemos cerrarlo, **Solucionado.**

Por lo cual tenemos que aunque no depuremos, tenemos una opción.

Comenzando: Análisis packer

Con el uso de Pe/Scanner Tenemos:

ExeinfoPE



```
Scanning -> C:\Documents and Settings\Mafo.CASA-0603EA19BE\Escritorio\escritorio mxe\Nueva
carpeta (3)\CanU\CanU.exe
File Type : 32-Bit Exe (Subsystem : Win GUI / 2), Size : 583987 (08E933h) Byte(s)
[x] Warning - FileAlignment seems wrong.. is 0x00000200, calculated 0x00000400
[File Heuristics] -> Flag : 00000000000000001100001000100011 (0x0000C223)
[!] EXE Cryptor v2.2.0 - v2.2.6 detected !
- Scan Took : 0.110 Second(s)
```

Protection id 0.6.4.0 July

Quick unpack 2.2 (PEid integrado) -> PEid scanning... **EXECryptor 2.2.4 -> Strongbit/SoftComplete Development (h1)**

«Multiples Protecciones»



EXECryptor Detección Heurística
EXECryptor (Compress Code & Data)
EXECryptor v2.1.15
EXECryptor v2.2.x - v2.3.x

Rdg Detector: EXECryptor v2.2.x - v2.3.x

Encontrando los mensajes del tiempo

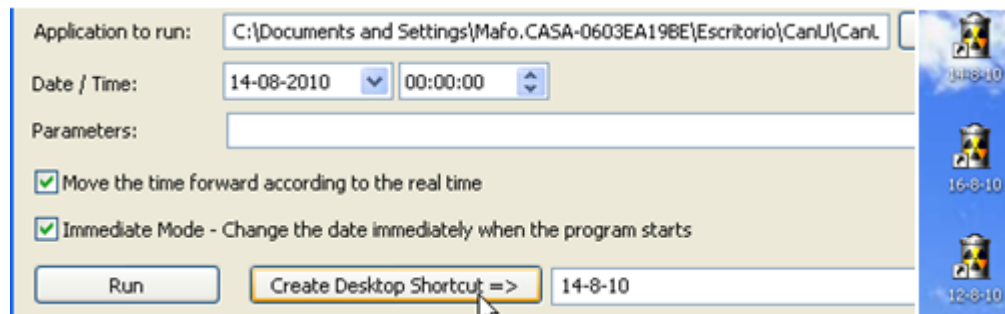
Run as Date: esta herramienta tiene la capacidad de cambiar la fecha, sin alterar el programa, por ende puedo saber, si me detecta o si es vulnerable a inyección en fecha por `GetSystemTime`, `GetLocalTime`, `GetSystemTimeAsFileTime`.

```
How does it work ?
=====

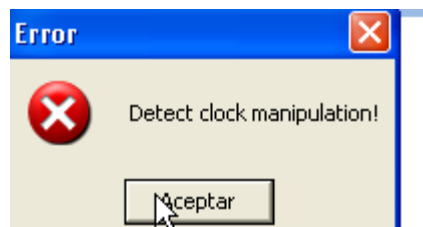
RunAsDate v1.10
Copyright (C) 2007 - 2009 Nir Sofer
Web site: http://www.nirsoft.net

RunAsDate intercepts the kernel API calls that returns the current date
and time (GetSystemTime, GetLocalTime, GetSystemTimeAsFileTime), and
replaces the current date/time with the date/time that you specify. |
```

Comienzo con el "Run as date" Configuro la ruta:

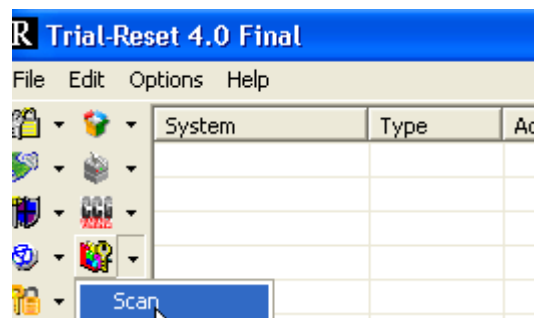


Otra configuro para otras 2 fechas con accesos directos



Para el día antes y después muestra

Como tenemos Una detección necesitaremos una buena herramienta para escanear lo trial y volverlo funcional, esto es Trial Reset, es una herramienta que escanea ramas de diversos Packer, a modo de seguir usándolo más de una vez, sirve en muchos packers, si llegara a expirar, podría usar esta herramienta y puedo volver a usar la aplicación, escaneando y borrando las ramas indicadas.



Trial Reset:

Está Confirmado que es Execryptor y Trial Reset me ayudara en caso de expirar:

System	Type	Address	Date/Time
EXECryptor	Reg32Key	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved\{1B0B1686-9575-E64B-57EA-74D4B8210EB7}\	08/14/2010
EXECryptor	Reg32Key	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved\	08/14/2010

Exportado tenemos:

REGEDIT4

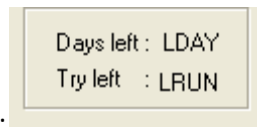
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved\]

[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved\ \{1B0B1686-9575-E64B-57EA-74D4B8210EB7\}]

**"iaokhlhpfdeibfgjbf"=hex:6a,61,64,6d,70,69,67,6a,67,67,67,66,64,67,6d,63,66,6d, **
61,62,00

[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved\ \{1B0B1686-9575-E64B-57EA-74D4B8210EB7\}]

**"iaokhlhpfdeibfgjbf"=hex:6a,61,64,6d,70,69,67,6a,67,67,67,66,64,67,6d,63,66,6d, **
61,62,00



Si No existiera esta comprobación, el crackme diría;

Uno está representado por Lday (“first day” y” lastday”) y otro por Lrun -> “Run Count”

Lectura:

Es bueno prepararse, y sobre todo leer

Lecturas en chino

Busco información en caso de ser necesario en CHINO, usar el traductor de google , aquí una referencia breve:

Antidebug 2005-10-23,03:11:10 : <http://bbs.pediy.com/showthread.php?t=17826>

onetouch 2007-11-22,19:53:25 <http://bbs.pediy.com/showthread.php?t=55333>

vm_1 <http://bbs.pediy.com/showthread.php?t=108117>

Lecturas en español/Ingles

457-Tuto setool2G execryptor 2 por GUAN.rar

462-EXECryptor v2.1.15_by_+NCR.rar

468-EXECryptor v2.15_CrackmeNoOficial.rar

474-AkirAExecryptor.rar

478-EXECryptor v2.15_CrackmeNoOficial_I_by_+NCR.rar

479-EXECryptor v2.15_CrackmeNoOficial_II_by_+NCR.rar

521-ExeCryptor v2.2.5.b4_by_+NCR.rar

540-Execryptor_a_look_on.rar

548-Detalles sobre EXECryptor_JuanJose_1Parte.rar

552-Detalles sobre EXECryptor_JuanJose_2Parte.rar

597-Unpacking_And_Dumping_ExeCryptor_and_Coding_Loader_by_deroko.rar

598-ExeCryptor v2.2.x_by_+NCR_doc.rar

724-Desempacando ExeCryptor 2.2.x - 2.3.x por RAIN.rar

915-ExeCryptor_2.2.x_- 2.3.x_Unpacking_tutorial_- _By_EvOIUtIoN.rar

950-tute_flvtoavi_execryptor_para_newbies_tincopasan.rar

1040-EXECryptor 2.2.50_todos_Solid+Tena.zip

1048-Unpack EXECryptor 2.2.x En Familiar Keylogger 2.80 por ^Munit[0].rar

1109-EXECRYPTOR_INTERNALS_MEGA_PROJECT

1148-ExeCryptor 2.2.6 (Unpacking).rar

1309-EC quick 3d execryptor +painter by Apuromafo.rar

En algunos sitios también han comentado del tema , entre los revisados son ingleses, vietnamitas, rusos,chinos, arabes-

En espera de nuevas técnicas o implementaciones, chequeos, scripts y hasta el famoso unpacker de RSI.

Unpacker ExeCryptor RC2 by RSI, ExeCryptor (Disabling CRC Checks), ExeCryptor Inline Patching, HWID patch,

execryptor oficial crackme+ sol-execryptor-kao , EXECryptor_2.xx__DecryptKey_Algo_by_sunbeam , ec_fix_unpack by

lcf, EXECryptor.V2.4.1.Cracked.By.[CUG], unpack method by hagggar, Unpacking EXECryptor 2.3x by wy no bar,

EXECryptor_tutorial_1_joker_italy.rar, EXECryptor_tutorial_2_joker_italy.rar, EXECryptor v2.4.1 All Protection at4re,

execryptor_unpackme_2009+sol by lcf, MUP Execrptor 2.2.5, sunbeam detecting

breackpoint,EXECryptor.v2.4.x.Tips.and.Tricks2 (tut1,2,3,4) by sunbeam, Execryptor 2.x IAT rebuilder by KaGra v1.1,

Execryptor_2.xx_IAT_Fixer_v1.02SC.By.Volx.osc, ExeCryptor 2.xx OEP Finder.txt (volx, orthodox)

,execryptor_2.x_2.41_find_VM_EPTrickyboy.osc

ANTI-UNPACKER TRICKS by peter ferrie y lecturas necesarias de sitios webs:

<http://www.woodmann.com/forum/showthread.php?t=12688>

<http://www.exetools.com/forum/showthread.php?t=3563&page=1>

<http://www.reversing.be/article.php?story=20061129191447423>

<http://www.reversing.be/article.php?story=20061206203632615&mode=print>

<http://bbs.pediy.com/showthread.php?p=467675>

<http://kioresk.wordpress.com/> -> http://kioresk.wordpress.com/2008/05/17/disabling_clock_manipulation_check/

<http://sunbeam.bubble.ro/> // <http://tuts4you.com/download.php?view.2228>

Comenzando en Olly, las primeras detecciones.

Comenzamos con Un Olly normal, Agrego el **plugin OllyAdvanced** y tildo la pestaña de iniciar en TLS, al ejecutar cae aqui:

0015A10	IF0:F1	lock int1
0015A10	I81CF E7235DA	or edi, A95D23E7
0015A11	I57	push edi

Esta excepcion es parte de una pequeña rutina donde Execryptor crea un HW breakpoint, a modo de detectar el depurador. si cae en este lugar otra vez puede crashear, colocamos pausa, y al volver ejecutara vuelve al lock. pues execryptor tambien resume el programa en algun instante (si coloco pausa, caeremos aquí, excepción y fuera).

Haggar en BiW Reversing ExeCryptor 2_3_9 – Unpacking, lo explica el origen, puede ser pasado Como agregar excepcion Custom Invalid Lock Sequence, pero ademas comenta de otros detalles importantes que crea un hw bp temporal, por eso es invalid lock sequence.

- Hardware exception trick -

This is new trick. It seems that EC sets temporary hardware breakpoint somehow. I do not know yet how this is performed, but within Olly this will cause file crushing/detection.

Under exceptions options in Olly, CHECK ALL boxes. But delete all custom exceptions! Be sure that you didn't place any memory, software or hardware breakpoint. Now hit Shift+F9 to run target under Olly. You will stop on first exception:

007637D0 F0:F1 LOCK INT1

Hit Shift+F9 three more times and then check hardware breakpoints (Debug menu -> Hardware breakpoints). You will see that one hardware breakpoint is placed:

Hardware breakpoints

Base Size Stop on

1 0076098F Temporary

This is new trick in EC. If you continue to running with this hwbp, target will crash eventually. Therefore, delete it at this point. Also, go to exception options and add last exception (C000001E INVALID LOCK SEQUENCE) to custom ones.

Luego de mirar un momento, Ahora agrego el Plugin Codedoctor: y encuentro código auto parchante, a modo que termina la rutina, parcha y no vuelve a ejecutar ese lugar, .

mov byte ptr ds [r32],parche

0005C544	. 8901	MOV DWORD PTR DS:[ECX],EAX
0005C546	. 59	POP ECX
0005C547	. 8D05 DFEA0A00	LEA EAX,DWORD PTR DS:[.AEADF]
0005C54D	. C600 C3	MOV BYTE PTR DS:[EAX],0C3
0005C550	. ^E9 8A250500	JMP 31_e.000AEADF

Comparaciones de variables constante, que varían en 1 número, y rutinas que intentaran detectar al depurador como si fuera FINDWINDOW

Haggar explico de forma que uno lo encuentra como el uso de

You will find similar checks if you search all EC sections. Search for `CMP DWORD[EAX],CONST` and you'll get results:

00099837	. 8B02	MOV EAX,EAX
00099838	. 5A	POP EDX
0009983C	. 8138 54444247	CMP DWORD PTR DS:[EAX],47424454
00099842	. ^0F84 73800300	JE unpack3.000018B8
00099848	. ^E9 8B20FEFF	JMP unpack3.0007B8D8
0009984D	. 81	DB 81
0009984E	. CA	DB CA
0009984F	. CC	INT3
00099850	. 80	DB 80
00099851	. 80	DB 80

Pero gracias a codedoctor lo vemos antes:

```
US 31_e.00005E2E
CMP DWORD PTR SS:[EBP-4],1
JNZ 31_e.0006C824
JE 31_e.0008D064
SUB EDX,EBX
ADD ESI,ECX
SBB ECX,2D46A1A6
SUB EDX,EBX
SBB EBP,EAX
LEA EAX,DWORD PTR SS:[EBP-2AA]
CMP DWORD PTR DS:[EAX],4742444F
JE 31_e.000018BB
JNZ 31_e.0006C824
JNZ 31_e.00092312
POPF
JE 31_e.00093BC7
CMP DWORD PTR SS:[EBP-4],2
JNZ 31_e.000A3DF1
JE 31_e.0007D451
SBB ECX,EDI
XCHG DWORD PTR DS:[ESI],EBP
CMP EAX,D1EF3D6B
JA 31_e.0008421A
MOV EBX,9847FB7F
XOR ECX,ESI
LEA EAX,DWORD PTR SS:[EBP-26F]
CMP DWORD PTR DS:[EAX],4742444F
JE 31_e.000018BB
JNZ 31_e.000A3DF1
NOP
INT3
```

Otras cosas que vemos son Uso de esp, sumas en eax, movimiento de trozos entre 2 variables (copiado de bloques),

Xoring(metodo mas comun para encriptar/desencrptar)

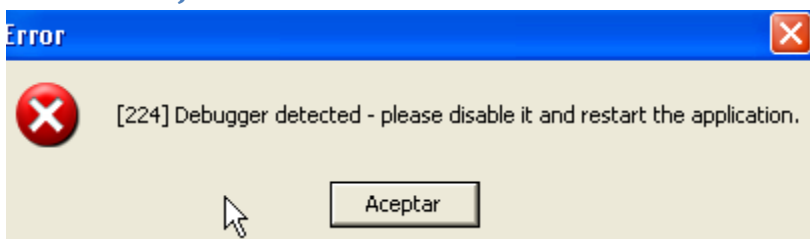
```
MOV ESP,2939E857
ADD DWORD PTR DS:[EAX],EAX
MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
XOR AL,0E8
PUSH EDX
```

Salto de estilo Push+ret Push+jmp+ret

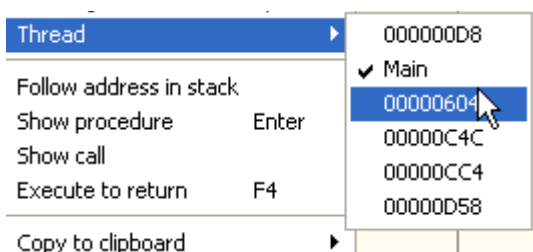
Por lo tanto quien quiera analizar un execryptor y no sepa que existen muchas variables asociadas, le sugiero primero leer ,y estudiar un poco Execryptor y luego atacar el Unpacked.El packed, es solo para tutoriales largos como este.

Si intento tener más herramientas, en un momento dado también puede ocurrir alguna Deteccion de depurador

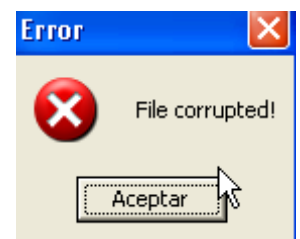
Mas mensajes



Todo esto porque hay Multiples thread



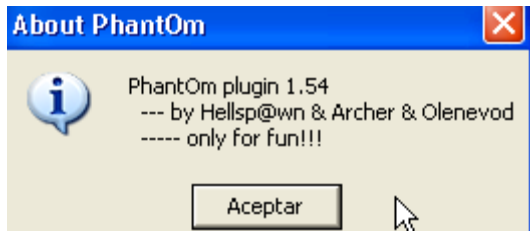
Tambien podemos encontrar Deteccion de CRC



Configurando Olly 1.1

La forma mas simple de pasar todo esto, es evitando los handles invalidos, y pasar los RTDSC,es con plugins

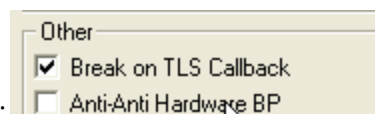
Vamos a configurar un olly 1.1 sin parches, agregando PHANTOM



Tildando todo, y accediendo en ollydbg.ini

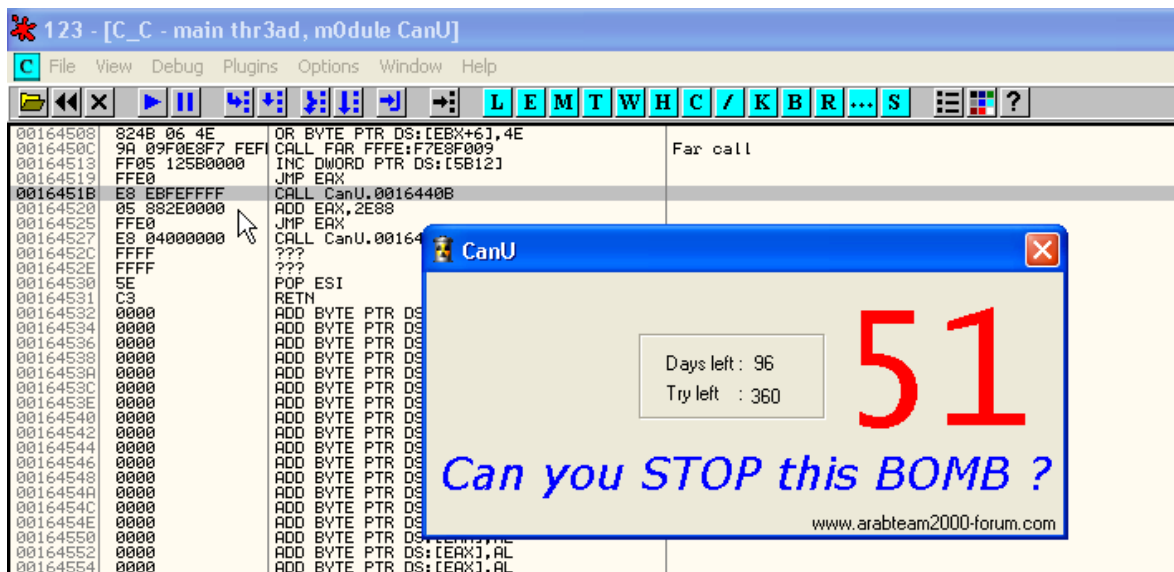
Cambiamos los valores de **deltardtsc, hidename, rdtscname , captext y pretext**

```
;;=====
[Plugin Phantom]
PEB=1
GETCOUNT=1
DRX=1
SETCONTEXT=1
DEBSTRING=1
WINVER=0
GETTIMES=0
REMOVEEP=1
HANDLE=1
WINDOWS=1
DRIVER=1
CAPTION=1
RDTSC=1
VERSION=154
DELTARDTSC=34815
HIDENAME=789
RDTSCNAME=456
CAPTEXT=123
PRETEXT=C_C|
SINGLE=1
BLOCK=1
[Plugin Bookmarks]
```

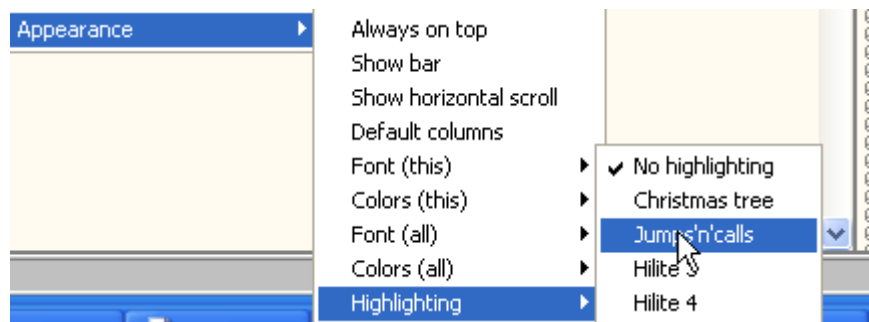


Y OllyAdvanced solo tildando esta opción:

Le doy run desde aqui y el resultado es esto:



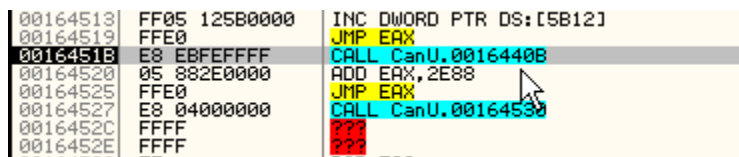
Como es nuevo este olly, agrego una mejor apariencia:



Comienzo denuevo coloco un bp en ejecucion en el jmp eax superior:

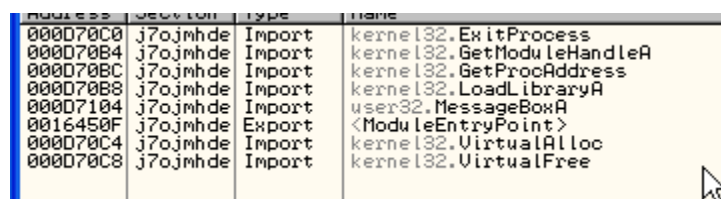
Normalmente el primer jmp eax, va a desenscriptar segun su libreria,

El superior , salta a la parte final donde accede a las bases de las dll



Y ahora de colocar f9, cae en este acceso

Y ahora busco las referencias de las apis que deberian estar desenscriptadas



Si vemos donde son llamadas vemos esta estructura mas o menos tipica en execryptor, donde guarda las imagebase de las dll, donde luego calculara un valor, y luego no restaurar otra vez, por ende todo execryptor que dejemos 1 variable mal puesta , no correra en otra maquina

001642A3	^E9 42A8FFFF	JMP CanU.0015EAEA	
001642A8	55	PUSH EBP	
001642A9	8BEC	MOV EBP,ESP	
001642AB	83C4 F4	ADD ESP,-0C	
001642AE	56	PUSH ESI	
001642AF	57	PUSH EDI	
001642B0	53	PUSH EBX	
001642B1	BE 00800500	MOV ESI,CanU.00058000	
001642B6	B8 00000100	MOV EAX,CanU.00010000	ASCII "MZP"
001642BB	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
001642BE	89C2	MOV EDX,EAX	
001642C0	8B46 0C	MOV EAX,DWORD PTR DS:[ESI+C]	
001642C3	09C0	OR EAX,EAX	
001642C5	^0F84 8D000000	JE CanU.00164358	
001642CB	01D0	ADD EAX,EDX	
001642CD	89C3	MOV EBX,EAX	
001642CF	50	PUSH EAX	
001642D0	FF15 B4700D00	CALL DWORD PTR DS:[<&kernel32.GetModuleHandleA	kernel32.GetModuleHandleA
001642D6	09C0	OR EAX,EAX	
001642D8	^0F85 0F000000	JNZ CanU.001642E0	
001642DE	53	PUSH EBX	
001642DF	FF15 B8700D00	CALL DWORD PTR DS:[<&kernel32.LoadLibraryA	kernel32.LoadLibraryA
001642E5	09C0	OR EAX,EAX	
001642E7	^0F84 63000000	JE CanU.00164350	
001642ED	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX	
001642F0	6A 00	PUSH 0	
001642F2	8F45 F4	POP DWORD PTR SS:[EBP-C]	
001642F5	8B06	MOV EAX,DWORD PTR DS:[ESI]	
001642F7	09C0	OR EAX,EAX	
001642F9	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	
001642FC	^0F85 03000000	JNZ CanU.00164305	
00164302	8B46 10	MOV EAX,DWORD PTR DS:[ESI+10]	
00164305	01D0	ADD EAX,EDX	
00164307	8945 F4	ADD EAX,DWORD PTR SS:[EBP-C]	
0016430A	8B18	MOV EBX,DWORD PTR DS:[EAX]	
0016430C	8B7E 10	MOV EDI,DWORD PTR DS:[ESI+10]	
0016430F	01D7	ADD EDI,EDX	
00164311	037D F4	ADD EDI,DWORD PTR SS:[EBP-C]	
00164314	09D8	OR EBX,EBX	
00164316	^0F84 E1000000	JE CanU.001643F0	
0016431C	F7C3 00000080	TEST EBX,80000080	
00164322	^0F85 04000000	JNZ CanU.0016432C	
00164328	8D5C13 02	LEA EBX,DWORD PTR DS:[EBX+EDX+2]	
0016432C	81E3 FFFFFFFF	AND EBX,7FFFFFFF	
00164332	53	PUSH EBX	
00164333	FF75 F8	PUSH DWORD PTR SS:[EBP-8]	
00164336	FF15 BC700D00	CALL DWORD PTR DS:[<&kernel32.GetProcAddress	kernel32.GetProcAddress
0016433C	09C0	OR EAX,EAX	
0016433E	^0F84 0C000000	JE CanU.00164350	
00164344	8907	MOV DWORD PTR DS:[EDI],EAX	
00164346	8345 F4 04	ADD DWORD PTR SS:[EBP-C],4	
0016434D	^E9 0455FFFF	JMP CanU.00164355	

Si vemos las intermodular call, vemos los llamados a exitProcess,

00154D63	CALL DF0354A9	
0015504F	CALL 6C373AC6	
00156214	CALL DWORD PTR DS:[<&kernel32.ExitProcess	kernel32.ExitProcess
001563BB	CALL DWORD PTR DS:[<&kernel32.ExitProcess	kernel32.ExitProcess
00157E8E	CALL 3376679A	
001589F5	CALL 003242E3	
00158F17	CALL DWORD PTR DS:[<&kernel32.ExitProcess	kernel32.ExitProcess
0015A00D	CALL 9F823380	
0015B59A	JNS CanU.0016103D	(Initial CPU selection)
0015B740	CALL B5EA9F45	
0015BCDD	CALL 575B90F9	
0015BFF1	CALL 5839C47D	
0015D0C7	CALL B7E49467	
0015D3D8	CALL DWORD PTR DS:[C7104]	DS:[000C7104]=E8240487
0015E4FF	CALL DWORD PTR DS:[<&kernel32.ExitProcess	kernel32.ExitProcess
00160BE3	CALL DWORD PTR DS:[<&kernel32.ExitProcess	kernel32.ExitProcess
00162156	CALL DWORD PTR DS:[<&kernel32.ExitProcess	kernel32.ExitProcess
00162D8E	CALL DWORD PTR DS:[<&kernel32.ExitProcess	kernel32.ExitProcess
00162D8D	CALL EA162D00	
00163FC3	CALL FF38F6B1	
001642D0	CALL DWORD PTR DS:[<&kernel32.GetModuleHandleA	kernel32.GetModuleHandleA
001642DF	CALL DWORD PTR DS:[<&kernel32.LoadLibraryA	kernel32.LoadLibraryA
00164336	CALL DWORD PTR DS:[<&kernel32.GetProcAddress	kernel32.GetProcAddress
001643C3	CALL DWORD PTR DS:[C7104]	DS:[000C7104]=E8240487
001643CB	CALL DWORD PTR DS:[<&kernel32.ExitProcess	kernel32.ExitProcess

Y luego algunos call directo a las apis

Si ejecuto mas de una vez vemos algunas variables, que estan encryptadas y en un momento dado se desencripta:

Vemos derrepente variables en eax:  

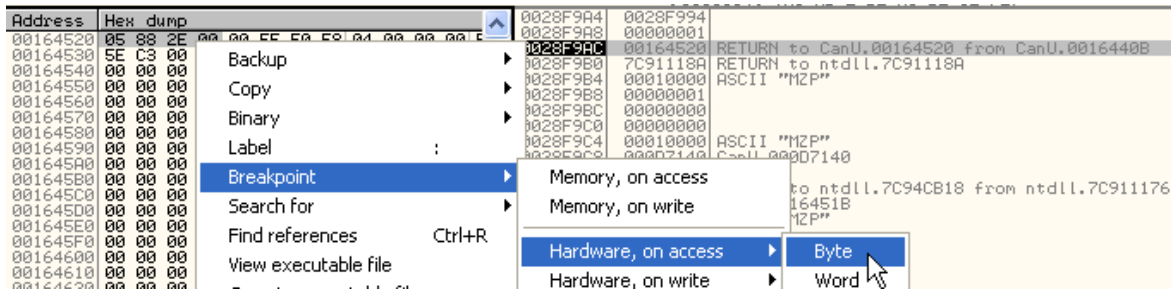
Pero mas que estas variables que pasan por stack y todos se ven en eax

TLS

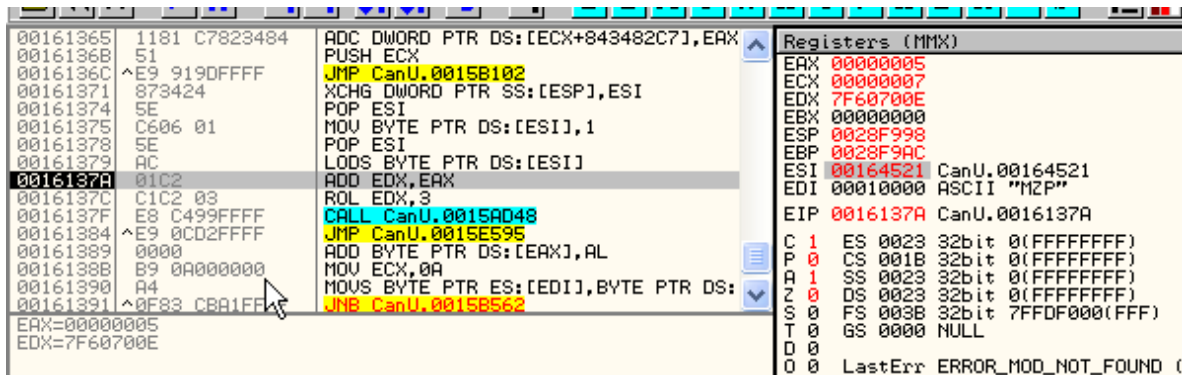
Comenzare a investigar parte de este codigo ofuscado, asi conocer como usa el TLS

, decido en el primer call, osea en TLS, usa el call y comienzo:

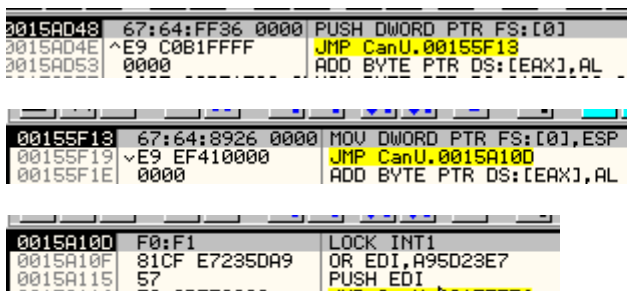
El primer call coloco bp en access en esp, sobre el valor que esta en stack sobre el dword



Se detiene aqui:



Luego va a a un seh



Que tal?, el primer int esta relacionada con el primer call ejecutado en el exe

Pero si hay un bp, estoy mas que seguro, que eax, tendria el valor de breackpoint(vease bien el "lods byte"

Sigamos Observando antes de comenzar a ver hazañas.

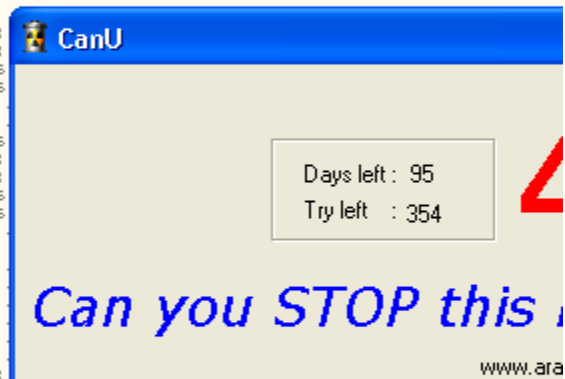
Si ejecuto y veo en LOG vemos:

Breckpoint, singlestep, invalid lock, +threads

```

Thread 00000B7C terminated, exit code 0
Exception 80000003 (Breakpoint) at address: 0015ABF9
Exception 80000003 (Breakpoint) at address: 00163E42
Exception 80000004 (Single Step) at address: 0015EF6E
Exception 80000004 (Single Step) at address: 00158DA7
Exception C000001E (Invalid Lock Sequence) at address: 0015FCFA
Thread 00000F34 terminated, exit code 0
Exception C000001E (Invalid Lock Sequence) at address: 000CBAF1
Exception 80000004 (Single Step) at address: 000CA429
Exception 80000003 (Breakpoint) at address: 000A3F17
Exception 80000004 (Single Step) at address: 000CD38A
Exception 80000004 (Single Step) at address: 000A5C90
Exception 80000004 (Single Step) at address: 00073724
Exception 80000004 (Single Step) at address: 00092351
Exception 80000004 (Single Step) at address: 0005BA64
Exception C000001E (Invalid Lock Sequence) at address: 0007167C
Exception C000001E (Invalid Lock Sequence) at address: 00079701
Exception 80000003 (Breakpoint) at address: 0015ABF9
Exception 80000003 (Breakpoint) at address: 00163E42
Exception 80000004 (Single Step) at address: 0015EF6E
Exception 80000004 (Single Step) at address: 00158DA7
Exception C000001E (Invalid Lock Sequence) at address: 0015FCFA
Thread 00000F2C terminated, exit code 0
3669 New thread with ID 00000B04 created
Exception 80000003 (Breakpoint) at address:
Exception 80000003 (Breakpoint) at address:
Exception 80000004 (Single Step) at address:
Exception 80000004 (Single Step) at address:
Exception C000001E (Invalid Lock Sequence)
Exception C000001E (Invalid Lock Sequence)
Exception 80000004 (Single Step) at address:
Exception 80000003 (Breakpoint) at address:
Exception 80000003 (Breakpoint) at address:
Exception 80000004 (Single Step) at address:
Exception 80000004 (Single Step) at address:
Exception C000001E (Invalid Lock Sequence)
Thread 00000B04 terminated, exit code 0
Exception C000001E (Invalid Lock Sequence)
Exception C000001E (Invalid Lock Sequence)
Exception C000001E (Invalid Lock Sequence)
Exception C000001E (Invalid Lock Sequence)
Exception C000001E (Invalid Lock Sequence)
Exception 80000003 (Breakpoint) at address:
Exception 80000003 (Breakpoint) at address: 00163E42
Exception 80000004 (Single Step) at address: 0015EF6E
Exception 80000004 (Single Step) at address: 00158DA7
Exception C000001E (Invalid Lock Sequence) at address: 0015FCFA
Thread 00000FE4 terminated, exit code 0
Exception C000001E (Invalid Lock Sequence) at address: 00077DEC
Exception 80000003 (Breakpoint) at address: 0015ABF9
Exception 80000003 (Breakpoint) at address: 00163E42
Exception 80000004 (Single Step) at address: 0015EF6E
Exception 80000004 (Single Step) at address: 00158DA7
Exception C000001E (Invalid Lock Sequence) at address: 0015FCFA
Thread 00000F64 terminated, exit code 0
Exception C000001E (Invalid Lock Sequence) at address: 00077DEC
Exception 80000003 (Breakpoint) at address: 0015ABF9
Exception 80000003 (Breakpoint) at address: 00163E42
Exception 80000004 (Single Step) at address: 0015EF6E
Exception 80000004 (Single Step) at address: 00158DA7
Exception C000001E (Invalid Lock Sequence) at address: 0015FCFA
Thread 00000F24 terminated, exit code 0
Exception C000001E (Invalid Lock Sequence) at address: 00077DEC
Exception C000001E (Invalid Lock Sequence) at address: 00077DEC
Exception 80000003 (Breakpoint) at address: 0015ABF9
Exception 80000003 (Breakpoint) at address: 00163E42

```



Son bastantes, pero recurrentes.

Si vemos el bp en access en el jmp eax

00164513	FF05 125B0000	INC DWORD PTR
00164519	FF05	JMP EAX
0016451B	E8 EBFEFFFF	CALL CanU.0016451B
0016451D	9C 007E0000	INC DWORD PTR DS:[ESI]

Caeremos en la linea anterior de lo que acabamos de ver

```

00161379 AC LODS BYTE PTR DS:[ESI]
0016137A 01C2 ADD EDX,EAX
0016137C 01C2 03 ROL EAX,3

0015E023 67:64:FF36 0000 PUSH DWORD PTR FS:[0]
0015E029 67:64:8926 0000 MOV DWORD PTR FS:[0],ESP
0015E02F ^E9 BF85FFFF JMP CanU.001585F3
0015E034 0000 0000 ADD BYTE PTR DS:[ESI],0

001585F3 F2: PREFIX REPNE:
001585F4 F1 INT1
001585F5 ^E9 27460000 JMP CanU.0015CC21
001585FA 8BCC MOV ECX,ESP
001585FC 81C1 10000000 AND ECX,10

```

00161B4E	8BD1	MOV EDI,ECX
00161B50	E8 CEC4FFFF	CALL CanU.0015E023
00161B55	E9 A098FFFF	JMP CanU.0015B3FA

Por ende esi, intenta acceder los bytes de tls,o de la seccion antes de saltar a la primera excepcion

Detenidos veremos valores Unicode de ciertas apis y tambien Ascii

28F9C0	00000000	
28F9C4	7FFDFC18	UNICODE "01000000000290"
28F9C8	7C98EEC8	UNICODE "advapi32.dll"
28F9CC	0000E538	
28F9D0	0028F9C0	
28F9D4	7C91E920	ntdll.7C91E920
28F9D8	0028FA4C	UNICODE "82476501-484763869-839522"
28F9DC	7C91E920	ntdll.7C91E920
28F9E0	7C922D80	ntdll.7C922D80
28F9E4	FFFFFFFF	
28F9E8	7C922D78	RETURN to ntdll.7C922D78 from ntdll
28F9EC	0028FA04	
28F9F0	7C91FCB7	RETURN to ntdll.7C91FCB7 from ntdll
28F9F4	00000008	
28F9F8	00000001	
28F9FC	00000000	
28FA00	00000008	
28FA04	0028FA74	UNICODE "39522115-1004\SOFTWARE\Mi
28FA08	7C91FCB7	RETURN to ntdll.7C91FCB7 from ntdll

Comenzamos otra vez.entrando desde TLs y vemos :

0016442D	6A 00	PUSH 0
0016442F	B8 C4700D00	MOV EAX,<%kernel32.VirtualAlloc>
00164434	8B0418	MOV EAX,DWORD PTR DS:[EAX+EBX]
00164437	FFD0	CALL EAX
00164439	59	POP ECX
0016443A	BA 885A1500	MOV EDI,CanU.00155A88
0016443F	01DA	ADD EDI,EBX
00164441	52	PUSH EDI
00164442	53	PUSH EBX
00164443	50	PUSH EAX
00164444	89C7	MOV EDI,EAX
00164446	89D6	MOV ESI,EDI
00164448	FC	CLD
00164449	F3:A4	REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
0016444B	B9 04591500	MOV ECX,CanU.00155904
00164450	01D9	ADD ECX,EBX
00164452	FFD1	CALL ECX
00164454	58	POP EAX
00164455	8B1C24	MOV EBX,DWORD PTR SS:[ESP]
00164458	68 00800000	PUSH 8000
0016445D	6A 00	PUSH 0
0016445F	50	PUSH EAX
00164460	B8 C8700D00	MOV EAX,<%kernel32.VirtualFree>
00164465	8B0418	MOV EAX,DWORD PTR DS:[EAX+EBX]
00164468	FFD0	CALL EAX
0016446A	59	POP ECX
0016446B	58	POP EAX
0016446C	5B	POP EBX
0016446D	83EB 05	SUB EBX,5
00164470	C603 B8	MOV BYTE PTR DS:[EBX],0B8
00164473	43	INC EBX
00164474	8903	MOV DWORD PTR DS:[EBX],EAX
00164476	83C3 04	ADD EBX,4
00164479	C603 C3	MOV BYTE PTR DS:[EBX],0C3
0016447C	89C9	OR ECX,ECX

El primer reserva memoria, y realiza el primer parcheo

Que va enfocado al call superior de donde entramos

00164519	FFE0	JMP EAX
0016451B	E8 EBFEFFFF	CALL CanU.0016440B
00164520	05 882E0000	ADD EAX,2E88
00164525	FFE0	JMP EAX

Ahora si quiero dejarlo mas interesante, y evitar detecciones Además colocare **Strong OD**, pero sin tildar nada, pues el se encargara de parchar parte de mis dll en runtime, y otras cosas para evitar crasheos en caso de colocar bp en Access

Ahora coloco en el En el PEB un bp en access

Llegando fácilmente a donde llama por isdebugger present de una forma solo para verificar script que acceden, y luego cambian a cero

00058377	E9 F950400	JMP CanU.00058377	EAX 7C8130A3	kernel32.IsDebuggerPresent
0005839E	64:FF35 0000000	PUSH DWORD PTR FS:[0]	ECX 0028F934	
000583A5	64:8925 0000000	MOV DWORD PTR FS:[0],ESP	EDX 7C802644	kernel32.7C802644
000583AC	8B00	MOV EAX,DWORD PTR DS:[EAX]	EBX 00000000	
000583AE	B8 01000000	MOV EAX,1	ESP 0028F7F0	
000583B3	E9 D6FFFFFF	JMP CanU.0005838E	EBP 0028F804	
000583B8	53	PUSH EBX	ESI 0028F9C0	ASCII "MZP"
000583B9	E9 20530600	JMP CanU.000B06DE	EDI 00010000	CanU.000583AC
000583BE	8905 24610500	MOV DWORD PTR DS:[56124],EAX	EIP 000583AC	CanU.000583AC
000583C4	8D05 682C0C00	LEA EAX,DWORD PTR DS:[C2C68]	C 1 ES 0023 32bit 0(FFFFFFFF)	
000583CA	E8 458C0600	CALL CanU.000C1014	P 1 CS 001B 32bit 0(FFFFFFFF)	
000583CF	C1C3 1F	ROL EBX,1F		
000583D3	E9 07000300	JMP CanU.000303FF		

En forma simple la rutina mueve a eax 1 , y asi va comprobando las apis

0005838E	64:8F05 0000000	POP DWORD PTR FS:[0]
00058395	83C4 04	ADD ESP,4
00058398	C3	RETN
00058399	E9 F9650400	JMP CanU.0009E997
0005839E	64:FF35 0000000	PUSH DWORD PTR FS:[0]
000583A5	64:8925 0000000	MOV DWORD PTR FS:[0],ESP
000583AC	8B00	MOV EAX,DWORD PTR DS:[EAX]
000583AE	B8 01000000	MOV EAX,1
000583B3	E9 D6FFFFFF	JMP CanU.0005838E

Para cualquiera pensaria que quiere llamar a la api Isdebuggerpresent, pero en este caso esta tomando las primeros bytes de cada api para verificar la presencia de saltos, bp, y call entre otros.

El push de abajo y luego va alos pop de arriba..asi se va moviendo y distrayendo

Como mueve el valor 1:

Para que luego haga el primer text

0009C98A	84C0	TEST AL,AL
0009C98C	^0F85 73B7FEFF	JNZ CanU.00088105
0009C992	^E9 DE1B0100	JMP CanU.000AE575
0009C997	81F0 F9E663F7	XOR EAX,F763E6F9
0009C99D	81E0 CE39CE68	AND EAX,68CE39CE
0009C9A3	81E0 7E5763C3	XOR EAX,63C37E57

Ahora viene lo que comentaba sunbeam y lo que explicaba haggar y ya todos los que ven execryptor deben tener presente:presencia de bp

0006D5CA	5E	POP ESI	
0006D5CB	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	kernel32.IsDebuggerPresent
0006D5CE	8A00	MOV AL,BYTE PTR DS:[EAX]	
0006D5D0	2C 99	SUB AL,99	
0006D5D2	^E9 F1FE0200	JMP CanU.0009D4C8	
0006D5D7	87FB	XCHG EBX,EDI	
0006D5D9	^E9 D7860400	JMP CanU.000B5CB5	
0006D5DF	C3	RETN	

Luego hace operaciones de wrapper para detectar bp

00093325	8B12	MOV EDX,DWORD PTR DS:[EDX]	kernel32.IsDebuggerPresent
00093327	F62A	IMUL BYTE PTR DS:[EDX]	
00093329	3C A4	CMP AL,0A4	
0009332B	^0F85 A26BFEFF	JNZ CanU.00079ED3	
00093331	^E9 C2300300	JMP CanU.000C63F8	

Tambien existen comparaciones mas explicitas que deben ser cautelosas, si verificamos la documentacion desde el gran escrito de at4re

```

SHR  EAX, 01h          ;| checks for bp (0xCC) ; This the only real bad boy
CMP  EAX, 066h          ;|
JNZ/JE badboy
MOV  EAX, LOCAL_01
CMP  WORD PTR DS:[EAX], 02ECDh      ; checks for int 2E
JNZ/JE badboy

```

Luego incrementa y sigue buscando

000934CF	8BEC	MOV EBP,ESP	
000934D1	83C4 F8	ADD ESP,-8	
000934D4	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	kernel32.IsDebuggerPresent
000934D7	^0F88 C80FFFFF	JS CanU.000944A5	
000934DD	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
000934E0	0FB600	MOVZX EAX,BYTE PTR DS:[EAX]	
000934E3	833C85 904F0700	CMP DWORD PTR DS:[EAX*4+74F901],0	
000934E5	AE9 E071EEEE	JMP CanU.00070600	
0007178E	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
00071791	8038 E8	CMP BYTE PTR DS:[EAX],0E8	
00071794	^0F85 C20B0300	JNZ CanU.000A235C	
0007179A	^E9 4D6D0300	JMP CanU.000A84EC	
0007179F	81F3 2F621A3E	XOR EBX,3E1A622F	
000717A0	83D0	AND EBX,EBX	

Si saco el procedimiento este es el mas delicado:

```

CMP  BYTE PTR DS:[EAX], 00E8h      ; chequea si hay call call
JNZ  @@_4
MOV  EAX, LOCAL_01
INC  EAX
MOV  EAX, DWORD PTR DS:[EAX]        ; si es call, calcula el destino
ADD  EAX, LOCAL_01
ADD  EAX, 05h
CMP  EAX, DWORD PTR DS:[DD_EXITPROCESS] ; compara con ExitProcess
JE   @@_1
MOV  EAX, LOCAL_01
INC  EAX
MOV  EAX, DWORD PTR DS:[EAX]
ADD  EAX, LOCAL_01
ADD  EAX, 05h
CMP  EAX, DWORD PTR DS:[DD_RTLEXITUSERPROCESS] ; Y recomprar con RtlExitUserProcess
JE   @@_1

```

Tambien es capaz de comparar saltos

Tambien existen compraciones

```

CMP  BYTE PTR DS:[EAX], 00E9h      ; short jump
CMP  BYTE PTR DS:[EAX], 00EBh      ; near jump

```

Por el momento usa la variable en ebp-4

000A2362	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
000A2365	E8 5A75FFFF	CALL CanU.000998C4	
000A236A	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX	
000A236D	837D F8 00	CMP DWORD PTR SS:[EBP-8],0	
000A2371	^0F8E 14040200	JLE CanU.000C278B	
000A2377	AE9 0A30EEEE	JMP CanU.0009EBCF	

Otras comparaciones pueden encontrarse dentro del mismo

```
0005EC98 0F85 1A000000 JNZ CanU.0005ECB8
0005EC9E 837D E4 66 CMP DWORD PTR SS:[EBP-1C],66
0005ECA2 0F85 93850600 JNZ CanU.000C723B
0005ECA8 B8 06000000 MOV EAX,6
0005ECAD 2B45 F4 SUB EAX,DWORD PTR SS:[EBP-C],EAX
0005ECB0 8945 F4 MOV DWORD PTR SS:[EBP-C],EAX
0005ECB3 E9 1A960600 JMP CanU.000C82D2
```

```
000C723B 837D E4 67 CMP DWORD PTR SS:[EBP-1C],67
000C723F 0F85 A1A3FCFF JNZ CanU.000915E6
000C7245 B8 06000000 MOV EAX,6
```

Y ojo ahora hace esto

```
000915E6 8B45 E0 MOV EAX,DWORD PTR SS:[EBP-20]
000915E9 0FB600 MOVZX EAX,BYTE PTR DS:[EAX]
000915EC 8945 E4 MOV DWORD PTR SS:[EBP-1C],EAX
000915EF E9 021E0100 JMP CanU.000A33F6
000915F4 81C1 C890DDA9 ADD ECX,A9DD90C8
000915F9 81F1 A9D77CFD AND ECX,ED7CD7A9
DS:[7C8130A4]=A1
EAX=7C8130A4 (kernel32.7C8130A4)
```

Multiplica..y otra comparacion

```
0006F89E 8B45 FC MOV EAX,DWORD PTR SS:[EBP-4]
0006F8A1 8038 CF CMP BYTE PTR DS:[EAX],0CF
0006F8A4 0F84 915A0200 JE CanU.0009533B
0006F8AA E9 AA220500 JMP CanU.000C1B59
0006F8AF 81EA B8CF4CCE SUB EDX,CE4CCFB8
0006F8B5 81FA B8CF4CCE SUB EDI,CE4CCFB8
->chequea IRETD
```

Para quien este interesado de cómo se hizo, esto es mas entendible gracias al proyecto en At4re, que quiere explicar el procedimiento Interno de Execryptor:

EC_MEGA_PROJECT\PROJECTS\EC_LIB_API_PROCS\EC_REDIR_BP_CHECKER\PROCEDURES.inc

Sigamos luego viene una ofuscación, usando CODEDOCTOR, vemos que hara algo como esto:

```
000880B0 68 580E5DCF PUSH CF5D0E58
000880C2 5A POP EDX
000880C3 C1C2 1B ROL EDX,1B
000880C6 81E2 69B49DCD AND EDX,CD9DB469
000880CC 81C2 49150241 ADD EDX,41021549
000880D2 81EA 60ED5163 SUB EDX,6351ED60
000880D8 81F2 2641CE00 XOR EDX,0CE4126
000880DE E9 484BFEFF JMP CanU.0006CC2B
000880E3 0F85 82B02000 JNZ CanU.000A8C6B
000880E9 E9 19710400 JMP CanU.000CF207
```

Sigamos vemos esto:

Pero el uso de codedoctor es un mov edx, a106896f

```
NOP
NOP
NOP
MOV EDX,A106896F
NOP
CALL CanU.0009F9D6
MOV EAX,DWORD PTR SS:[ESP]
CALL CanU.00094B2A
RETN
NOP
NOP
```

Por ende tarde o temprano ira a comparar con el test al,

Para ver las variables

```
0009C985 E8 C089FFFF CALL CanU.0009534A
0009C98A 84C0 TEST AL,AL
0009C98C 0F85 73B7FEFF JNZ CanU.00088105
0009C992 E9 DE1B0100 JMP CanU.000AE575
0009C997 81F0 F9E663F7 XOR EAX,F763B\F9
0009C99D 81F0 F9E663F7 XOR EAX,F763B\F9
```

Y vuelve a la misma historia

Chequeo para *ret and ret X (C2 y C3)*

0009EAC9	24 F6	AND AL,0F6
0009EACB	3C C2	CMP AL,0C2
0009EACD	^0F84 6868FFFF	JE CanU.0009533B
0009EAD3	^E9 422E0300	JMP CanU.0000191A
0009EAD8	^0F84 B4220300	JE CanU.00000092

Chequeo para *jmp [XXXXXXXX]*

0006AAEC	66:25 FF38	AND AX,38FF
0006AAFB	66:3D FF20	CMP AX,20FF
0006AAFD	^E9 00140600	JMP CanU.000CBEF9
0006AAFF	^0F87 702F0600	JE CanU.00000070

Todo esto para guardar en la variable de ebp-5 el valor de al

00069F8C	^0F84 A9B30200	JE CanU.0009533B
00069F92	33C0	XOR EAX,EAX
00069F94	8845 FB	MOV BYTE PTR SS:[EBP-5],AL
00069F97	8A45 FB	MOV AL,BYTE PTR SS:[EBP-5]
00069F9A	59	POP ECX
00069F9B	59	POP ECX
00069F9C	5D	POP EBP
00069F9D	C3	RETN
00069F9E	873424	XCHG DWORD PTR SS:[ESP],ESI
00069FA1	5E	POP ESI

Luego sigue

00098D40	837D E4 0F	CMP DWORD PTR SS:[EBP-1C],0F
00098D44	^0F85 C0680100	JNZ CanU.000AF60A
00098D4A	8B45 E4	MOV EAX,DWORD PTR SS:[EBP-1C]
00098D4D	C1E0 08	SHL EAX,8

0006AE64	8178 E4 F6000000	CMP DWORD PTR DS:[EAX-1C],0F6
0006AE6B	^E9 694D0300	JMP CanU.0009FB09
0006AE70	8B05 48380900	MOV EAX,DWORD PTR DS:[93848]
0006AE76	09C0	OR EAX,EAX
0006AE78	^0F85 9D610400	JNZ CanU.000B101B
0006AE7E	^E9 E9120500	JMP CanU.000BC16C

0009FB09	^0F84 10000000	JE CanU.0009FBEF
0009FBDF	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
0009FBE2	8178 E4 F7000000	CMP DWORD PTR DS:[EAX-1C],0F7
0009FBE9	^0F85 AF5F0200	JNZ CanU.000C5B9E
0009FBEF	^0F82 B6B4FCFF	JB CanU.0006B0AB
0009FBF5	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
0009FBF8	F640 F0 38	TEST BYTE PTR DS:[EAX-10],38
0009FBFC	^0F85 9C5F0200	JNZ CanU.000C5B9E
0009FC02	^E9 3EC8FFFF	JMP CanU.0009C445
0009FC07	13C8	ADC ECX,EBX
0009FC09	81ED 578A48AB	SUB EBP,AB488A57
0009FC0F	81E5 7444ED83	AND EBP,83ED4474
0009FC15	81ED C136BC89	SUB EBP,89BC36C1
0009FC1B	^0F89 D4B9FDFF	JNS CanU.0007B5F5
0009FC21	^0F80 8AB7FDFF	JO CanU.0007B3B1
0009FC27	^E9 C4B9FDFF	JMP CanU.0007B5F0
0009FC2C	8901	MOV DWORD PTR DS:[ECX],EAX
0009FC2E	59	POP ECX
0009FC2F	8D05 DF250800	LEA EAX,DWORD PTR DS:[825DF]
0009FC35	C600 C3	MOV BYTE PTR DS:[EAX],0C3
0009FC38	^E9 A229FEFF	JMP CanU.00082SDF

Para preparar guardar en

000823B7	^E9 0A4C0200	JMP CanU.000A6FC6
000823BC	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX
000823BF	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]
000823C2	8BE5	MOV ESP,EBP
000823C4	5D	POP EBP
000823C5	C3	RETN
000823C6	^E9 5DAEFFFF	JMP CanU.0007D228
000823CB	837D F0 01	CMP DWORD PTR SS:[EBP-10],1
000823CF	^E9 96FFFFF	JMP CanU.00082360

Luego sigue con otras apis

```
CALL CanU.00000000
0 B2F49100 51 XOR DWORD PTR DS:[EAX+91F4B21,50]
8F8001=7C859CAE (kernel32.CheckRemoteDebuggerPresent)
```

Luego va por

```
CALL CanU.00000000
72008059 XOR EAX,59800072
201=7C810647 (kernel32.CreateThread)
```

Pero luego comienza lo interesante:

Address	Disassembly	Comment
0006C2B0	E8 FA130000	CALL CanU.0006D6AF
0006C2B5	✓E9 463D0000	JMP CanU.00070000
0006C2BA	✓E9 E48D0400	JMP CanU.000B50A3
0006C2BF	✓E9 80020300	JMP CanU.0009C544
0006C2C4	68 65E70600	PUSH CanU.0006E765
0006C2C9	✓E9 86350300	JMP CanU.0009F854
0006C2CE	5D	POP EBP
0006C2CF	8B05 34380900	MOV EAX,DWORD PTR DS:[93834:09C0]
0006C2D5	09C0	OR EAX,EAX
0006C2D7	✓0F85 06850500	JNZ CanU.000C47E3
0006C2DD	✓E9 BB750000	JMP CanU.0007389D
0006C2E2	C3	RETN

Registers (3DNow!)

EAX	7C810647	kernel32.CreateThread
ECX	0028F944	
EDX	0028F984	
EBX	00000000	
ESP	0028F968	
EBP	0028F994	
ESI	0028F9C0	
EDI	00010000	ASCII "MZP"
EIP	000C47E6	CanU.000C47E6
CS	0028	32bit 0(FFFFFFFF)

Ahora comienza a crear las Apis con los rol y salta ahi.

Y ahora llega a un poco mas arriba,

Address	Disassembly	Comment
00164514	05 125B0000	ADD EAX,5B12
00164519	FFEB	JMP EAX
0016451B	E8 EBF0FFFF	CALL CanU.0016440B
00164520	05 882E0000	ADD EAX,2E88
00164525	FFEB	JMP EAX
00164527	E8 04000000	CALL CanU.00164530
0016452C	FFFF	???
0016452E	FFFF	???
00164530	5E	POP ESI
00164531	C3	RETN
00164532	0000	ADD BYTE PTR DS:[EAX],AL
00164534	0000	ADD BYTE PTR DS:[EAX],AL
00164536	0000	ADD BYTE PTR DS:[EAX],AL
00164538	0000	ADD BYTE PTR DS:[EAX],AL
0016453A	0000	ADD BYTE PTR DS:[EAX],AL
0016453C	0000	ADD BYTE PTR DS:[EAX],AL
0016453E	0000	ADD BYTE PTR DS:[EAX],AL
00164540	0000	ADD BYTE PTR DS:[EAX],AL
00164542	0000	ADD BYTE PTR DS:[EAX],AL
00164544	0000	ADD BYTE PTR DS:[EAX],AL
00164546	0000	ADD BYTE PTR DS:[EAX],AL
00164548	0000	ADD BYTE PTR DS:[EAX],AL
0016454A	0000	ADD BYTE PTR DS:[EAX],AL
0016454C	0000	ADD BYTE PTR DS:[EAX],AL
0016454E	0000	ADD BYTE PTR DS:[EAX],AL
00164550	0000	ADD BYTE PTR DS:[EAX],AL
00164552	0000	ADD BYTE PTR DS:[EAX],AL
00164554	0000	ADD BYTE PTR DS:[EAX],AL
00164556	0000	ADD BYTE PTR DS:[EAX],AL
00164558	0000	ADD BYTE PTR DS:[EAX],AL
0016455A	0000	ADD BYTE PTR DS:[EAX],AL
0016455C	0000	ADD BYTE PTR DS:[EAX],AL
0016455E	0000	ADD BYTE PTR DS:[EAX],AL
00164560	0000	ADD BYTE PTR DS:[EAX],AL
00164562	0000	ADD BYTE PTR DS:[EAX],AL
00164564	0000	ADD BYTE PTR DS:[EAX],AL
00164566	0000	ADD BYTE PTR DS:[EAX],AL
00164568	0000	ADD BYTE PTR DS:[EAX],AL

EAX=0015A888 (CanU.0015A888)

Y luego de algún procedimiento Va Haciendo un código con seh.

01B80000	870424	XCHG DWORD PTR SS:[ESP],EAX
01B80003	58	POP EAX
01B80004	8B3424	MOV ESI,DWORD PTR SS:[ESP]
01B80007	C70424 109D1500	MOV DWORD PTR SS:[ESP],159D10
01B8000E	90	NOP
01B8000F	90	NOP
01B80010	90	NOP
01B80011	90	NOP
01B80012	90	NOP
01B80013	90	NOP
01B80014	90	NOP
01B80015	90	NOP
01B80016	90	NOP
01B80017	90	NOP
01B80018	90	NOP
01B80019	64:FF35 00000000	PUSH DWORD PTR FS:[0]
01B80020	0000	ADD BYTE PTR DS:[EAX],AL
01B80022	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
01B80029	0000	ADD BYTE PTR DS:[EAX],AL
01B8002D	CC	INT3

Asi que ejecuto a colocar bp en ejecucion o bp normal.(no olvidando retirarlo al ejecutado)

00159D10	51	PUSH ECX
00159D11	✓E9 CF250000	JMP CanU.0015C2E5
00159D16	0081 C950E3F3	ADD BYTE PTR DS:[ECX+F3E350C9]
00159D1C	53	PUSH EBX
00159D1D	81C1 01331A0C	ADD ECX,0C1A3301
00159D23	✓E9 619E0000	JMP CanU.00163B89
00159D28	C1FA 0F	SHI FAX,0F

Y se ve como quiere sacar los Debugging register desde el context.

001586AC	8BCC	MOV ECX,ESP
001586AE	81C1 10000000	ADD ECX,10
001586B4	8B09	MOV ECX,DWORD PTR DS:[ECX]
001586B6	C701 13000100	MOV DWORD PTR DS:[ECX],CanU.00010013
001586BC	✓E9 53280000	JMP CanU.0015AF14
001586C1	0068 C7	ADD BYTE PTR DS:[ECX-39],CH

Cambiando el context flag, y anular los registros

00		XOR EAX,EAX
01 04 00000000		MOV DWORD PTR DS:[ECX+4],0
01 08 00000000		MOV DWORD PTR DS:[ECX+8],0
01 0C 00000000		MOV DWORD PTR DS:[ECX+C],0
01 10 00000000		MOV DWORD PTR DS:[ECX+10],0
		PUSH ESI
04000000		MOV ESI,4
01 14		ADD ECX,14
0E605CFE		CALL CanU.0015603A
30		ADD BYTE PTR DS:[EAX],AL
		RETN
		NOP

El cual me lleva al segundo seh_exit

0015B488	81C1 A0000000	ADD ECX,0A0
0015B491	C701 4F8F1500	MOV DWORD PTR DS:[ECX],CanU.00158F4F
0015B497	33C0	XOR EAX,EAX
0015B499	59	POP ECX
0015B49A	C3	RETN
0015B49B	00E9	ADD CL,CH
0015B49D	F4	HLT
0015B49E	45	INC EBP
0015B49F	0000	ADD BYTE PTR DS:[EAX],AL
0015B4A1	✓F9 98040000	JMP CanU.0015B93F

Que deberia terminar en algun POP DWORD PTR FS:[00h]

Luego se ve otra forma de parchar

mov edi, direccion,

mov tamaño edi,parche

00157A22	57	PUSH EDI
00157A23	BF 22881500	MOV EDI,CanU.00158822
00157A28	✓E9 03210000	JMP CanU.00159C00
00157A2D	3BD5	CMP EDX,EBP

00159C00	C607 8D	MOV BYTE PTR DS:[EDI],8D
00159C03	5F	POP EDI
00159C04	E8 2EA40000	CALL CanU.00164037
00159C06	8B05	MOV EAX,DWORD PTR DS:[EBP+5]

Y mirando en general

00159BF9	51	PUSH ECX	
00159BFA	✓E9 806E0000	JMP CanU.00160A7F	
00159BFF	00C6	ADD DH,AL	
00159C01	07	POP ES	
00159C02	8D5F E8	LEA EBX,DWORD PTR DS:[EDI-18]	Modific
00159C05	2E:A4	MOVS BYTE PTR ES:[EDI],BYTE PTR CS:[ESI]	
00159C07	0000	ADD BYTE PTR DS:[EAX],AL	
00159C09	0000	ADD BYTE PTR DS:[EAX],AL	
00159C0B	✓E9 02270000	JMP CanU.0015C312	
00159C10	0000	ADD BYTE PTR DS:[EAX],AL	
00159C13	51	PUSH ECX	

159c0b con codedoctor vemos como seguirá:

Ahora se prepara para la primera excepción

001578D8	67:64:FF36 0000	PUSH DWORD PTR FS:[0]
001578DE	67:64:8926 0000	MOV DWORD PTR FS:[0],ESP
001578E4	✓E9 69190000	JMP CanU.00159252
001578E9	67:64:FF36 0000	PUSH DWORD PTR FS:[0]
001578EF	67:64:8926 0000	MOV DWORD PTR FS:[0],ESP

0015C117	F0:CC	LOCK INT3
0015C119	81D0 249ABDAE	ADC EAX,AE8D9A24
0015C11F	81F9 EDAD3175	CMP ECX,7531ADE0
0015C125	^E9 5EC6FFFF	JMP CanU.00158788
0015C12A	F7C5 A9A81787	TEST EBP,8717A8A9
0015C12D	^E9 20150000	JMP CanU.00158000

Como si o si ira a excepción:

028F994	0028FA08	Pointer to next SEH record
028F998	00161E2C	SE handler
028F99C	00000000	
028F9A0	00000000	

vamos al se handler

00161E2C	51	PUSH ECX
00161E2D	^E9 E696FFFF	JMP CanU.0015B518
00161E32	81C6 FACCE05D	ADD ESI,5DE0CCFA
00161E38	873C24	XCHG DWORD PTR SS:[ESP]
00161E3B	5F	POP EDI
00161E3D	^E9 82 8F9CFFFF	JMP CanU.0015B000

Donde denuevo borrar los hwbp

009F0000	5E	POP ESI
009F0001	81E6 57D17FBF	AND ESI, 57D17FBF
009F0007	81EE 14EF3A7B	SUB ESI, 14EF3A7B
009F000D	81E6 B5E891F6	AND ESI, B5E891F6
009F0013	81C6 10607E79	ADD ESI, 10607E79
009F0019	03CE	ADD ECX, ESI
009F001B	5E	POP ESI
009F001C	8B09	MOV ECX, DWORD PTR DS:[ECX]
009F001E	C701 13000100	MOV DWORD PTR DS:[ECX], 10013
009F0024	33C0	XOR EAX, EAX
009F0026	C741 04 00000000	MOV DWORD PTR DS:[ECX+4], 0
009F002D	C741 08 00000000	MOV DWORD PTR DS:[ECX+8], 0
009F0034	C741 0C 00000000	MOV DWORD PTR DS:[ECX+C], 0
009F003B	90	NOP
009F003C	90	NOP
009F003D	90	NOP
009F003E	90	NOP
009F003F	90	NOP
009F0040	90	NOP
009F0041	90	NOP
009F0042	90	NOP
009F0043	90	NOP
009F0044	90	NOP
009F0045	90	NOP
009F0046	90	NOP
009F0047	90	NOP
009F0048	90	NOP
009F0049	90	NOP
009F004A	90	NOP
009F004B	90	NOP
009F004C	90	NOP
009F004D	90	NOP
009F004E	83C1 10	ADD ECX, 10
009F0051	E8 3B0077FF	CALL CanU.00160091
009F0052	64 5F05 00000000	PUSH DWORD PTR FS:[0]

160091 Para ir a 158794

009F0000	873C24	XCHG DWORD PTR SS:[ESP], EDI
009F0003	5F	POP EDI
009F0004	8901	MOV DWORD PTR DS:[ECX], EAX
009F0006	8941 04	MOV DWORD PTR DS:[ECX+4], EAX
009F0009	8941 08	MOV DWORD PTR DS:[ECX+8], EAX
009F000C	C781 A8000000 9	MOV DWORD PTR DS:[ECX+A8], 158794
009F0016	90	NOP
009F0017	90	NOP
009F0018	90	NOP
009F0019	90	NOP
009F001A	90	NOP
009F001B	90	NOP
009F001C	90	NOP
009F001D	90	NOP
009F001E	90	NOP
009F001F	90	NOP
009F0020	90	NOP
009F0021	90	NOP
009F0022	33C0	XOR EAX, EAX
009F0024	59	POP ECX
009F0025	C3	RETN

158794

00158794	67:64:8F05 0000	POP DWORD PTR FS:[0]
0015879A	870424	XCHG DWORD PTR SS:[ESP], EAX
0015879D	58	POP EAX
0015879E	68 B8682911	PUSH 112968B8
001587A3	✓E9 530F0000	JMP CanU.001596FB

Luego parchara y luego un call a 1574aa

64:8F05 00000000	POP DWORD PTR FS:[0]
0000	ADD BYTE PTR DS:[EAX], AL
870424	XCHG DWORD PTR SS:[ESP], EAX
58	POP EAX
52	PUSH EDX
8BD6	MOV EDX, ESI
871424	XCHG DWORD PTR SS:[ESP], EDX
C605 80911500 8	MOV BYTE PTR DS:[159100], 8B
5E	POP ESI
BB 04000000	MOV EBX, 4
83C0 04	ADD EAX, 4
E8 827475FF	CALL CanU.001574AA
C3	RETN
90	NOP
90	NOP

Muestra una rutina que descomprimira

00155904	55	PUSH EBP	
00155905	8BEC	MOV EBP,ESP	
00155907	83EC 14	SUB ESP,14	
0015590A	90	NOP	
0015590B	53	PUSH EBX	
0015590C	57	PUSH EDI	
0015590D	56	PUSH ESI	
0015590E	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
00155911	8955 F8	MOV DWORD PTR SS:[EBP-8],EDX	
00155914	89C6	MOV ESI,EAX	
00155916	89D7	MOV EDI,EDX	
00155918	66:8138 4A43	CMP WORD PTR DS:[EAX],434A	compara con JC

Descompresion:

00155946	0245 EF	ADD AL,BYTE PTR SS:[EBP-11]	
00155949	AA	STOS BYTE PTR ES:[EDI]	
0015594A	^EB E9	JMP SHORT CanU.00155935	
0015594C	E8 FC000000	CALL CanU.00155A40	
00155951	^0F82 97000000	JB CanU.001559EE	
00155957	E8 F1000000	CALL CanU.00155A40	
0015595C	^73 5B	JNB SHORT CanU.001559B9	
0015595E	B9 04000000	MOV ECX,4	
00155963	E8 FD000000	CALL CanU.00155A65	
00155968	48	DEC EAX	
00155969	^74 DE	JE SHORT CanU.00155949	
0015596B	^0F89 C7000000	JNS CanU.00155A38	
00155971	E8 D7000000	CALL CanU.00155A40	
00155976	^73 1B	JNB SHORT CanU.00155993	
00155978	55	PUSH EBP	
00155979	BD 00010000	MOV EBP,100	
0015597E	E8 D7000000	CALL CanU.00155A5A	
00155983	8807	MOV BYTE PTR DS:[EDI],AL	
00155985	47	INC EDI	
00155986	4D	DEC EBP	
00155987	^75 F5	JNZ SHORT CanU.0015597E	
00155989	E8 BF000000	CALL CanU.00155A40	
0015598E	^72 E9	JB SHORT CanU.00155979	
00155990	5D	POP EBP	
00155991	^EB A2	JMP SHORT CanU.00155935	

Address	Hex dump	ASCII
00011000	04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00011010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00011020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00011030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

La cual comienza a escribir en 11000

Pasando por un loop que baja un valor de ebp

00155A40	01D2	ADD EDX,EDX	
00155A4F	^75 08	JNZ SHORT CanU.00155A59	
00155A51	8B16	MOV EDX,DWORD PTR DS:[ESI]	
00155A53	83C6 04	ADD ESI,4	
00155A56	F9	STC	
00155A57	11D2	ADC EDX,EDX	
00155A59	C3	RETN	
00155A5A	B9 08000000	MOV ECX,8	
00155A5F	E8 01000000	CALL CanU.00155A65	
00155A64	C3	RETN	
00155A65	31C0	XOR EAX,EAX	
00155A67	E8 E1FFFFFF	CALL CanU.00155A40	
00155A6C	11C0	ADC EAX,EAX	
00155A6E	^E2 F7	LOOPD SHORT CanU.00155A67	
00155A70	C3	RETN	
00155A71	31C9	XOR ECX,ECX	

Cuento corto:

00155935	31C0	XOR EAX,EAX	
00155937	E8 11010000	CALL CanU.00155A40	
0015593C	^73 0E	JNB SHORT CanU.0015594C	
0015593E	8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]	
00155941	E8 1F010000	CALL CanU.00155A65	
00155946	0245 EF	ADD AL,BYTE PTR SS:[EBP-11]	
00155949	AA	STOS BYTE PTR ES:[EDI]	
0015594A	^EB E9	JMP SHORT CanU.00155935	
0015594C	E8 FC000000	CALL CanU.00155A40	
00155951	^0F82 97000000	JB CanU.001559EE	
00155957	E8 F1000000	CALL CanU.00155A40	

El call marcado será cuando termine de descriptar ese bloque

El jb saltara y veremos lo que tiene oculto:

Posiblemente sea un delphi!! Descripto algo "Boolean"

Y de a poco lo va desenscriptando

La rutina que escribe

00155A13	5D	POP EBP	
00155A14	89C3	MOV EBX,EAX	
00155A16	E8 56000000	CALL CanU.00155A71	
00155A1B	3D 00000100	CMP EAX,CanU.00010000	ASCII "MZP"
00155A20	73 14	JNB SHORT CanU.00155A36	
00155A22	3D FF370000	CMP EAX,37FF	
00155A27	73 0E	JNB SHORT CanU.00155A37	
00155A29	3D 7F020000	CMP EAX,27F	
00155A2E	73 08	JNB SHORT CanU.00155A38	
00155A30	83F8 7F	CMP EAX,7F	
00155A33	77 04	JA SHORT CanU.00155A39	
00155A35	41	INC ECX	
00155A36	41	INC ECX	
00155A37	41	INC ECX	
00155A38	41	INC ECX	
00155A39	56	PUSH ESI	
00155A3A	89FE	MOV ESI,EDI	
00155A3C	29C6	SUB ESI,EAX	
00155A3E	F3:A4	REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:	
00155A40	5E	POP ESI	CanU.000D731A
00155A41	^E9 EFFEFFFF	JMP CanU.00155935	
00155A46	89F0	MOV EAX,ESI	
00155A48	5E	POP ESI	

Stack [0028F974]=000D731A (CanU.000D731A)
ESI=0001137A (CanU.0001137A)

Address	Hex dump	ASCII
0001133F	61 05 00 8B C0 FF 25 28 61 05 00 8B C0 53 83 C4	a*.i.l%(a*.i.l'sa-
0001134F	BC 8B 0A 00 00 00 54 E8 FF FB E0 9B F6 44 24 2C	"n....Tp '0e+D\$,
0001135F	01 74 05 0F B7 5C 24 30 8B C3 83 C4 44 5B C3 8B	0t**A\s0itâ-D[i
0001136F	C0 FF 25 24 61 05 00 8B C0 FF 25 20 61 05 00 8B	l%\$a*.i.l% a*.i
0001137F	C0 FF 25 00 00 00 00 00 00 00 00 00 00 00 00	l%.....
0001138F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0001139F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000113AF	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Al terminar todo

00155A82	C3	RETN	
00155A83	8BE5	MOV ESP,EBP	
00155A85	5D	POP EBP	
00155A86	C3	RETN	
00155A87	90	NOP	
00155A88	87AC24	XCHG DIWARD PTR SS:[ESP],EC	

00159180	8B10	MOV EDX,DWORD PTR DS:[EAX]	CanU.00042200
00159182	E8 F51D0000	CALL CanU.0015AF7C	
00159187	^E9 AADEFFFF	JMP CanU.00157036	
0015918C	51	PUSH ECX	
0015918D	^E9 95D8FFFF	JMP CanU.00156A27	
00159192	0000	ADD BYTE PTR DS:[EAX],AL	
00159194	C3	RETN	

Va por un int 3

0015C9ED	3E:CC	INT3	Superfluous prefix
0015C9EF	^E9 CA660000	JMP CanU.001630BE	
0015C9F4	FF4D F0	DEC DWORD PTR SS:[EBP-10]	
0015C9F7	68 FAE21500	PUSH CanU.0015E2FA	
0015C9FC	^E9 6AB6FFFF	JMP CanU.0015806B	

No creo que haga excepción, asi que debere ayudarme un poco con el int anterior con shift+f9

0028F998	0028FA08	Pointer to next SEH record
0028F99C	00159187	SE handler
0028F9A0	00011000	CanU.00011000
0028F9A4	0015E75A	CanU.0015E75A

159187

Luego de los 2 saltos desde 15987 veo que quiere hacer

8B4C24 10	MOV ECX,DWORD PTR SS:[ESP+10]
C701 13000100	MOV DWORD PTR DS:[ECX],10013
8A41 18	MOV AL,BYTE PTR DS:[ECX+18]
0081 B4000000	ADD BYTE PTR DS:[ECX+B4],AL
C781 B8000000	MOV DWORD PTR DS:[ECX+B8],1621AC
33C0	XOR EAX,EAX
59	POP ECX
C3	RET
64:8F05 00000000	POP DWORD PTR FS:[0]
0000	ADD BYTE PTR DS:[EAX],AL
870424	XCHG DWORD PTR SS:[ESP],EAX
58	POP EAX
52	PUSH EDX
8B06	MOV EDX,ESI
871424	XCHG DWORD PTR SS:[ESP],EDX
83C0 04	ADD EAX,4
870C24	XCHG DWORD PTR SS:[ESP],ECX
8BF1	MOV ESI,ECX
59	POP ECX
E8 DCD773FF	CALL CanU.0015D81E
02CD	ADD CL,CH
B3 B9	MOV BL,0B9
0000	ADD BYTE PTR DS:[EAX],AL
64:8925 00000000	MOV DWORD PTR FS:[0],ESP
0000	ADD BYTE PTR DS:[EAX],AL

Y denueov intenta ir a 1621ac

Pauso un poco, Porque hace tanta vuelta?,

Borra los registros

001581C7	81C1 04000000	ADD ECX,4
001581CD	8901	MOV DWORD PTR DS:[ECX],EAX
001581CF	81C1 A0000000	ADD ECX,0A0
001581D5	C701 94871500	MOV DWORD PTR DS:[ECX],CanU.00158794
001581DB	33C0	XOR EAX,EAX
001581DD	59	POP ECX
001581DE	E9 C8F6FFFF	JMP CanU.001578AB

Luego ira

001596F8	5B	POP EBX
001596FC	52	PUSH EDX
001596FD	8B06	MOV EDX,ESI
001596FF	871424	XCHG DWORD PTR SS:[ESP],EDX
00159702	BE 80911500	MOV ESI,CanU.00159180
00159707	E9 B1500000	JMP CanU.0015E7B0
		MOV BYTE PTR DS:[ESI],8B

Con eso transforma un call cosa que cuando termino el proceso

0015917A	5B	POP EBX
0015917B	E8 84C7FFFF	CALL CanU.00155904
00159180	8B10	MOV EDX,DWORD PTR DS:[EAX]
00159182	E8 F51D0000	CALL CanU.0015AF7C
00159187	E9 AADEFFFF	JMP CanU.00157036
0015918C	51	PUSH ECX

Y ahora puede continuar con otra excepción

001620C6	51	PUSH ECX
001620C7	E9 4A8DFFFF	JMP CanU.0015AE16

001605A1	F0:CC	LOCK INT3
001605A3	E9 839AFFFF	JMP CanU.0015A02B
001605A8	0000	ADD BYTE PTR DS:[EAX],AL

En forma simple, accede a la api, y si es diferente el valor va a la excepción

Para terminar en una comparación con edx

00163D7A	FF00	INC DWORD PTR DS:[EAX]	
00163D7C	68 84B61500	PUSH CanU.0015B6B4	
00163D81	^E9 8279FFFF	JMP CanU.0015B739	
00163D86	0051 B9	ADD BYTE PTR DS:[ECX-47],DL	
00163D89	9A 8B1500E9 B85	CALL FAR 99B8:E90015B8	Far call
00163D90	FFFF	???	Unknown command
00163D92	870424	XCHG DWORD PTR SS:[ESP],EAX	
00163D95	09D2	OR EDX,EDX	
00163D97	~0F84 05000000	JE CanU.00163D92	
00163D9D	E8 A239F7FF	CALL CanU.000D7144	
00163DA2	E8 687CFFFF	CALL CanU.0015B80F	

EDX	00042200	C4
EBX	00000000	
ESP	0028F99C	
EBP	0028F9AC	
ESI	0028F9C0	
EDI	00010000	AC
EIP	000D7144	C4
C 0	ES 0023 32	
P 1	CS 001B 32	
D 2	?? 0023 32	

000D7144	56	PUSH ESI	
000D7145	51	PUSH ECX	
000D7146	89C6	MOV ESI,EAX	
000D7148	89D1	MOV ECX,EDX	
000D714A	83E9 04	SUB ECX,4	
000D714D	FC	CLD	
000D714E	AC	LODS BYTE PTR DS:[ESI]	
000D714F	D0E8	SHR AL,1	
000D7151	80F8 74	CMP AL,74	
000D7154	~75 0E	JNZ SHORT CanU.000D7164	
000D7156	8B06	MOV EAX,DWORD PTR DS:[ESI]	
000D7158	0FC8	BSWAP EAX	
000D715A	01C8	ADD EAX,ECX	
000D715C	8906	MOV DWORD PTR DS:[ESI],EAX	
000D715E	83C6 04	ADD ESI,4	
000D7161	83E9 04	SUB ECX,4	
000D7164	49	DEC ECX	
000D7165	^7F E7	JG SHORT CanU.000D714E	
000D7167	59	POP ECX	
000D7168	5E	POP ESI	
000D7169	C3	RETN	
000D716A	8BC0	MOV EAX,EAX	

La cual arregla otra vez del trozo escrito

Ahora va a otra excepción

00157AE0	67:64:8926 0000	MOV DWORD PTR FS:[0],ESP	
00157AE6	F3:	PREFIX REP:	Su
00157AE7	F1	INT1	
00157AE8	0F8A C9880000	JPE CanU.001603B7	
00157AEE	~E9 48230000	JMP CanU.00159E3B	
00157AF3	0000	ADD BYTE PTR DS:[EAX],AL	

Seh->00163DA7

Para luego

F0000	C701 13000100	MOV DWORD PTR DS:[ECX],10013	
F0006	33C0	XOR EAX,EAX	
F0008	C741 04 00000000	MOV DWORD PTR DS:[ECX+4],0	
F000F	C741 08 00000000	MOV DWORD PTR DS:[ECX+8],0	
F0016	C741 0C 00000000	MOV DWORD PTR DS:[ECX+C],0	
F001D	C741 10 00000000	MOV DWORD PTR DS:[ECX+10],0	
F0024	C741 14 00000000	MOV DWORD PTR DS:[ECX+14],0	
F002B	C741 18 00000000	MOV DWORD PTR DS:[ECX+18],0	
F0032	C781 B8000000 4	MOV DWORD PTR DS:[ECX+B8],159E46	
F003C	90	NOP	
F003D	90	NOP	
F003E	90	NOP	
F003F	90	NOP	

159e46

00159E46	67:64:8F06 0000	POP DWORD PTR FS:[0]	
00159E4C	870424	XCHG DWORD PTR SS:[ESP],EAX	
00159E4F	58	POP EAX	
00159E50	870C24	XCHG DWORD PTR SS:[ESP],ECX	
00159E53	68 70A31500	PUSH CanU.0015A370	
00159E58	~E9 9E170000	JMP CanU.0015E5E7	

Luego otra excepción

0015E32F	67:F1	INT1	
0015E331	~E9 FD340000	JMP CanU.00161833	
0015E336	871424	XCHG DWORD PTR SS:[ESP],EDX	
0015E339	En	END ENY	

Seh->0015DC19

Para ir a

0015ADDE	C701 32E61500	MOV DWORD PTR DS:[ECX],CanU.0015E632	
0015ADDE4	✓E9 BB440000	JMP CanU.0015F2A4	
0015ADDE8	E8 E1E60000	CALL CanU.0015E632	

Después viene un int 3

0015BD2F	67:CC	INT3	8F99C 0028FA08 Pointer to next SEH record
0015BD31	C1EB 05	SHR EBX,5	8F9A0 0015E641 SE handler
0015BD34	99	CDQ	8F9A4 0015E75A CanU.0015E75A

El cual ira a 0015E641

El cual seguirá con

0016399A	871424	XCHG DWORD PTR SS:[ESP],EDX	
0016399D	5A	POP EDX	
0016399E	81C1 04000000	ADD ECX,4	
001639A4	C701 249E1500	MOV DWORD PTR DS:[ECX],CanU.00159E24	
001639AA	33C0	XOR EAX,EAX	
001639AC	59	POP ECX	
001639AD	C3	RETN	->159e24

Y llegamos al segundo or edx que descripto

0015BB81	0000	ADD BYTE PTR DS:[EAX],AL	
0015BB83	09D2	OR EDX,EDX	CanU.0005A000
0015BB85	✓0F84 45310000	JE CanU.0015ECD0	
0015BB88	52	PUSH EDX	
0015BB8C	✓E9 1C010000	JMP CanU.0015BCAD	
0015BB91	0000	ADD BYTE PTR DS:[EAX],AL	
0015BB93	✓0F85 7AA8FFFF	JNZ CanU.00156413	
0015BB99	8B10	MOV EDX,DWORD PTR DS:[EAX]	
0015BB9B	✓E9 18010000	JMP CanU.0015BCB8	
0015BBA0	00C6	ADD DH,AL	
0015BBA2	0187 5981E306	ADD DWORD PTR DS:[EDI+6E38159],EAX	
0015BBA8	5A	POP EDX	
0015BBA9	A2 4381C3B4	MOV BYTE PTR DS:[B4C38143],AL	
0015BBAB	A3 75BF871C	MOV DWORD PTR DS:[1C87BF75],EAX	
0015BBB3	24 E9	AND AL,0E9	
0015BBB5	6E	OUTS DX, BYTE PTR ES:[EDI]	I/O command
0015BBB6	CD FF	INT OFF	
0015BBB8	FF00	INC DWORD PTR DS:[EAX]	
0015BBBA	DF05 9AB51500	FILD WORD PTR DS:[15B59A]	
0015BBBC	✓E9 C7210000	JMP CanU.0015D08C	

EDX=0005A000 (CanU.0005A000)

Address	Disasm	Comment
054000	32 13 8B C0 02 00 8B C0 00 8D 40 00 00 8D 40 00	2!i!0.i!i!0..i!0.
054010	00 8D 40 00 00 00 00 00 00 00 00 10 21 01 00	.i!0.....!i!0.
054020	98 22 01 00 0C 26 01 00 52 75 6E 74 69 6D 65 20	y"0..&0.Runtime
054030	65 72 72 6F 72 20 20 20 20 20 61 74 20 30 30 30	error at 000
054040	30 30 30 30 30 00 8B C0 45 72 72 6F 72 00 8B C0	00000.i!Error.i!0.
054050	30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46	0123456789ABCDEF
054060	20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20	
054070	20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20	
054080	20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20	
054090	20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20	
0540A0	0D 0A 8B C0 00 00 00 00 00 00 00 00 00 00 00	..i!0.....
0540B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0540C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0540D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0540E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0540F0	00 00 00 00 8C 77 01 00 00 00 00 00 00 00 00	...i!w0.....
054100	00 00 00 80 00 00 00 40 00 00 00 C0 00 00 00	...C...0...L....
054110	00 00 00 00 01 00 00 00 02 00 00 00 03 00 00	A A A

El cual vuelve a la rutina de jc, hacemos el estilo upx, pushad (mov ebp,esp)

Y bp en hw en los bytes

Llegando al final

00155A82	C3	RETN
00155A83	8BE5	MOV ESP,EBP
00155A85	5D	POP EBP
00155A86	C3	RETN
00155A87	9A	NOP

En este bloque se descripto-

00	ff	...	<p..
FF
00
6E
65
00
00
73
73
6C
65
7C
00
00
64
78
00
00

	MOV EDX,DWORD PTR DS:[EAX]
	ADD EAX,4
	CALL CanU.00158CED
	ADD BYTE PTR DS:[EAX],AL
00	ADD ECX,0A0
00	MOV DWORD PTR DS:[ECX],162E00
	XOR EAX,EAX
	POP ECX
	NOP

Y en forma simple seguirá:

Luego el bp en Access seguirá mostrando los bloques para los edx

424
2
4	061A0000	...
		...
F40C21B3		...
		...
F7160000		...
7	1C245B51	...
FD9AFFFF		...

Y luego de unos varios instantes vemos esto, por fin termino de desenscriptar esa zona. 6 hojas para llegar aqui.

Ahora la rutina que dara los problemas de compatibilidad en los Sistemas operativos, ahora guardara las dll y guardara lo esencial.

001642A8	55	PUSH EBP	
001642A9	8BEC	MOV EBP,ESP	
001642AB	83C4 F4	ADD ESP,-0C	
001642AE	56	PUSH ESI	
001642AF	57	PUSH EDI	
001642B0	53	PUSH EBX	
001642B1	BE 00000500	MOV ESI,CanU.00058000	
001642B6	B8 00000100	MOV EAX,CanU.00010000	
001642BB	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
001642BE	89C2	MOV EDX,EAX	
001642C0	8B46 0C	MOV EAX,DWORD PTR DS:[ESI+C]	
001642C3	09C0	OR EAX,EAX	
001642C5	74F84 8D000000	JE CanU.00164358	
001642CB	01D0	ADD EAX,EDX	
001642CD	89C3	MOV EBX,EAX	
001642CF	50	PUSH EAX	
001642D0	FF15 B4700D00	CALL DWORD PTR DS:[<&kernel32.GetModuleHandleA]	kernel32.GetModuleHandleA
001642D6	09C0	OR EAX,EAX	
001642D8	74F85 0F000000	JNZ CanU.001642ED	
001642DE	53	PUSH EBX	
001642DF	FF15 B8700D00	CALL DWORD PTR DS:[<&kernel32.LoadLibraryA]	kernel32.LoadLibraryA
001642E5	09C0	OR EAX,EAX	
001642E7	74F84 63000000	JE CanU.00164350	
001642ED	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX	
001642F0	6A 00	PUSH 0	
001642F2	8F45 F4	POP DWORD PTR SS:[EBP-C]	
001642F5	8B06	MOV EAX,DWORD PTR DS:[ESI]	
001642F7	09C0	OR EAX,EAX	
001642F9	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	
001642FC	74F85 03000000	JNZ CanU.00164305	
00164302	8B46 10	MOV EAX,DWORD PTR DS:[ESI+10]	

Al terminar pasa por

0015AD53	0000	ADD BYTE PTR DS:[EAX],AL	
0015AD55	C605 9AB51500 0	MOV BYTE PTR DS:[15B59A],0F	
0015AD5C	C605 10891500 0	MOV BYTE PTR DS:[15B910],0F	
0015AD63	C3	RETN	
0015AD64	0000	ADD BYTE PTR DS:[EAX],AL	
0015AD66	52	PUSH EDX	
0015AD67	BA B58B1500	MOV EDX,CanU.00158B85	
0015AD6C	✓E9 D9180000	JMP CanU.0015C64A	

Luego otras excepciones

00158DA5	65:F1	INT1	
00158DA7	✓E9 BD860000	JMP CanU.00161469	
00158DAC	67:64:8F06 0000	POP DWORD PTR FS:[0]	
00158DB2	870424	XCHG DWORD PTR SS:[ESP],I	
00158DB5	58	POP EAX	
00158DB7	FF06	CALL EBX	

Llegando aquí:

00159726	8B3F	MOV EDI,DWORD PTR DS:[EDI]	
00159728	873C24	XCHG DWORD PTR SS:[ESP],EDI	
0015972B	E8 0E570000	CALL CanU.0015EE3E	
00159730	00E9	ADD CL,CH	
00159732	33F5	XOR ESI,EBP	
00159734	FFFF	23	Un
00159736	✓E9 6A620000	JMP CanU.0015F9A5	
0015973B	✓E9 98670000	JMP CanU.0015FED8	
00159740	0000	ADD BYTE PTR DS:[EAX],AL	
00159742	✓E9 09250000	JMP CanU.00158C50	
00159747	E8 257C0000	CALL CanU.00161371	
0015974C	00E9	ADD CL,CH	
0015974E	5D	POP EBP	
0015974F	53	PUSH EBX	
00159750	0000	ADD BYTE PTR DS:[EAX],AL	
00159752	0000	ADD BYTE PTR DS:[EAX],AL	
00159754	✓E9 1D490000	JMP CanU.0015E076	
00159759	✓E9 391C0000	JMP CanU.0015B397	
0015975E	000F	ADD BYTE PTR DS:[EDI],CL	
00159760	84E5	TEST CH,AH	
00159762	5A	POP EDX	
00159763	0000	ADD BYTE PTR DS:[EAX],AL	
00159765	00E9	ADD CL,CH	
00159767	3BAA 00000000	CMP EBP,DWORD PTR DS:[EDX]	
0015976D	68 A136E9AF	PUSH AFE936A1	
00159772	F7D5	NOT EBP	
00159774	✓E9 C85A0000	JMP CanU.0015F241	
00159779	0000	ADD BYTE PTR DS:[EAX],AL	
0015977B	C3	RETN	
0015977C	00E9	ADD CL,CH	
0015977E	54	PUSH ESP	
0015977F	✓77 00	JAE SHORT CanU.00159781	
00159781	00E9	ADD CL,CH	
00159783	C9	LEAVE	
00159784	14 00	ADC AL,0	

Stack DS:[0028F9BC]=00000000
EDI=0028F9BC

159726

000	CanU		PE header	IP
000	CanU	CODE		IP
000	CanU	DATA	data	IP
000	CanU	BSS		IP
000	CanU	uninitialized		IP

Como ya he visto gran parte de la memoria, voy a colocarlo en las 3 secciones

Si coloco un bp a golpe en las secciones, llego a pasarme de largo, si no hubiera sido mas largo que la pagina 26 a la 30.

00015F16	C3	RETN	
00015F17	90	NOP	
00015F18	50	PUSH EAX	
00015F19	6A 00	PUSH 0	
00015F1B	E8 F8FEFFFF	CALL CanU.00015E18	JMP to kernel32.GetModuleHandleA
00015F20	BA A4400500	MOV EDX,CanU.000540A4	
00015F25	52	PUSH EDX	
00015F26	8905 08540500	MOV DWORD PTR DS:[554D8],EAX	CanU.00010000
00015F2C	8942 04	MOV DWORD PTR DS:[EDX+4],EAX	
00015F2F	C742 08 00000000	MOV DWORD PTR DS:[EDX+8],0	
00015F36	C742 0C 00000000	MOV DWORD PTR DS:[EDX+C],0	
00015F3D	E8 8AFFFAFF	CALL CanU.00015ECC	
00015F42	5A	POP EDX	
00015F43	58	POP EAX	
00015F44	E8 73D7FFFF	CALL CanU.000136BC	
00015F49	C3	RETN	
00015F4A	8BC0	MOV EAX,EAX	
00015F4C	55	PUSH EBP	

Ahora bien como resumen, para mostrar cuanto cuesta solo llegamos a concluir que se des-encrpto un packer

si sola comienza ahora cuando ya se escribió, por completo, pasado los dos mov 0f, vemos que llega arriba de donde teníamos virtualalloc

Por ende vamos denuevo entro al call y subo ahora y coloco un bp en ejecucion en la parte superior y luego donde ya esta descriptado, y esos call, son partes de procesos cuando compara con exitprocess cuando son call xxx

001643B8	8085 00FFFFFF	LEA EAX,DWORD PTR SS:[EBP-100]	
001643C0	50	PUSH EAX	
001643C1	6A 00	PUSH 0	
001643C3	FF15 04710C00	CALL DWORD PTR DS:[C7104]	
001643C9	6A FF	PUSH -1	
001643CB	FF15 C0700D00	CALL DWORD PTR DS:[&kernel32.ExitProcess]	kernel32.ExitProcess
001643D1	8BE5	MOV ESP,EBP	
001643D3	5D	POP EBP	
001643D4	C3	RETN	
001643D5	8B85 F8FEFFFF	MOV EAX,DWORD PTR SS:[EBP-100]	
001643DB	8B95 FCFEFFFF	MOV EDX,DWORD PTR SS:[EBP-104]	
001643E1	8A12	MOV DL,BYTE PTR DS:[EDX]	
001643E3	8B9405 00FFFFFF	MOV BYTE PTR SS:[EBP+EAX-100],DL	
001643EA	FF85 F8FEFFFF	INC DWORD PTR SS:[EBP-108]	
001643F0	FF85 FCFEFFFF	INC DWORD PTR SS:[EBP-104]	
001643F6	E9 9BFFFFFF	JMP CanU.00164396	
001643FB	0000	ADD BYTE PTR DS:[EAX],AL	
001643FD	83C6 14	ADD ESI,14	
00164400	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	
00164403	E9 B8FEFFFF	JMP CanU.001642C0	
00164408	00C3	ADD BL,AL	
0016440A	00B8 885A1500	ADD BYTE PTR DS:[EAX+155A88],BH	
00164410	C3	RETN	
00164411	53	PUSH EBX	
00164412	51	PUSH ECX	
00164413	56	PUSH ESI	
00164414	57	PUSH EDI	
00164415	50	PUSH EAX	
00164416	8B1C24	MOV EBX,DWORD PTR SS:[ESP]	
00164419	81EB 10441600	SUB EBX,CanU.00164410	
0016441F	B8 569D0000	MOV EAX,9D56	
00164424	50	PUSH EAX	
00164425	6A 04	PUSH 4	
00164427	68 00100000	PUSH 1000	
0016442C	50	PUSH EAX	
0016442D	6A 00	PUSH 0	
0016442F	B8 C4700D00	MOV EAX,&kernel32.VirtualAlloc	
00164434	8B0418	MOV EAX,DWORD PTR DS:[EAX+EBX]	
00164437	FFD0	CALL EAX	
00164439	59	POP ECX	
0016443A	BA 885A1500	MOV EDX,CanU.00155A88	
0016443F	01DA	ADD EDX,EBX	
00164441	52	PUSH EDX	
00164442	53	PUSH EBX	
00164443	50	PUSH EAX	
00164444	89C7	MOV EDI,EAX	
00164446	89D6	MOV ESI,EDX	
00164448	FC	CLD	
00164449	F3A4	REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]	
0016444B	B9 04591500	MOV ECX,CanU.00155904	
00164450	01D9	ADD ECX,EBX	
00164452	FFD0	CALL EAX	

Y a partir de aqui si miramos las referencias

00154919	CALL 94C237F4	
00154C8F	CALL FDE7E01A	
00154D63	CALL DF0354A9	
0015504F	CALL 6C373AC6	
00156214	CALL DWORD PTR DS:[&kernel32.ExitProcess]	kernel32.ExitProcess
001563B8	CALL DWORD PTR DS:[&kernel32.ExitProcess]	kernel32.ExitProcess
00157E8E	CALL 3376679A	
001589F5	CALL 003242E3	
00158F17	CALL DWORD PTR DS:[&kernel32.ExitProcess]	kernel32.ExitProcess
0015A000	CALL 9F823380	
0015B740	CALL B5EA9F45	
0015BCDD	CALL 575B90F9	
0015BFF1	CALL 5839C47D	
0015D0C7	CALL B7E49467	
0015D30B	CALL DWORD PTR DS:[C7104]	DS:[000C7104]=E8240487
0015E4FF	CALL DWORD PTR DS:[&kernel32.ExitProcess]	kernel32.ExitProcess
00160BE3	CALL DWORD PTR DS:[&kernel32.ExitProcess]	kernel32.ExitProcess
00162156	CALL DWORD PTR DS:[&kernel32.ExitProcess]	kernel32.ExitProcess
00162D8E	CALL DWORD PTR DS:[&kernel32.ExitProcess]	kernel32.ExitProcess
00162DB0	CALL EA162D00	
00163FC3	CALL FF38F6B1	
001642D0	CALL DWORD PTR DS:[&kernel32.GetModuleHandleA]	kernel32.GetModuleHandleA
001642DF	CALL DWORD PTR DS:[&kernel32.LoadLibraryA]	kernel32.LoadLibraryA
00164336	CALL DWORD PTR DS:[&kernel32.GetProcAddress]	kernel32.GetProcAddress
001643C3	CALL DWORD PTR DS:[C7104]	DS:[000C7104]=E8240487
001643CB	CALL DWORD PTR DS:[&kernel32.ExitProcess]	kernel32.ExitProcess

Vemos mas claro donde debemos evitar los exit si hay algun call,

pero la segunda vez, llega otra vez aqui

00164410	C3	RETN
00164411	53	PUSH EBX
00164412	51	PUSH ECX
00164413	56	PUSH ESI
00164414	57	PUSH EDI
00164415	50	PUSH EAX
00164416	8B1C24	MOV EBX,WORD PTR SS:[ESP]
00164419	81EB 10441600	SUB EBX,CanU.00164410
0016441F	B8 569D0000	MOV EAX,9D56
00164424	50	PUSH EAX
00164425	6A 04	PUSH 4
00164427	68 00100000	PUSH 1000
0016442C	50	PUSH EAX
0016442D	6A 00	PUSH 0
0016442F	B8 C4700D00	MOV EAX,<kernel32.VirtualAlloc>
00164434	8B0418	MOV EAX,DWORD PTR DS:[EAX+EBX]
00164437	FFD0	CALL EAX

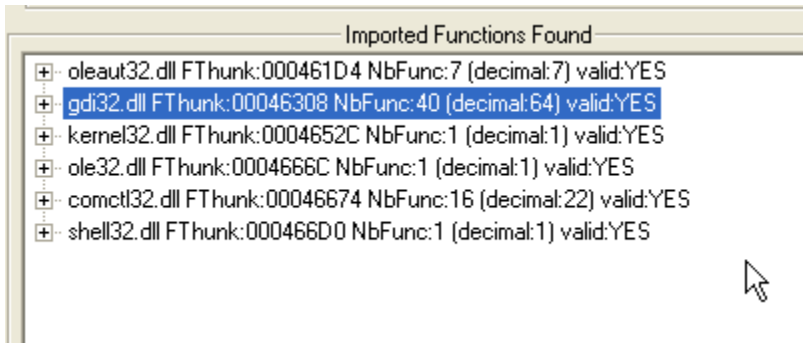
Y ahora recién llama al jmp eax que esta arriba

00164514	05 125B0000	ADD EAX,5B12
00164519	FFE0	JMP EAX
0016451B	E8 EBFEFFFF	CALL CanU.0016440B
00164520	05 882E0000	ADD EAX,2E88
00164525	FFE0	JMP EAX
00164527	E8 04000000	CALL CanU.00164530

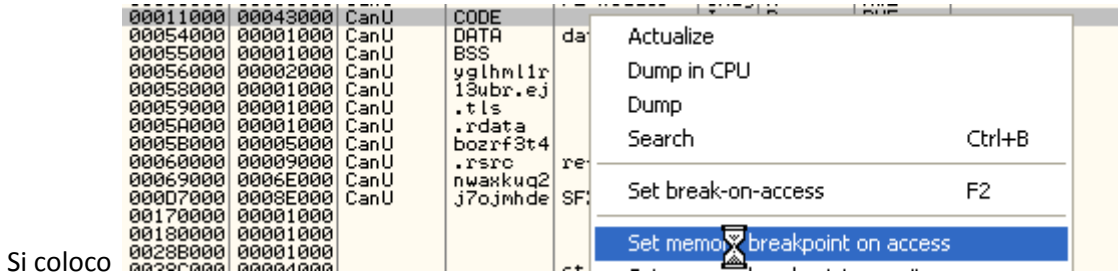
Que esta vez comienza con la estructura

0015B59A	0F89 9D5A0000	JNS CanU.0016103D
0015B5A0	E8 3F6A0000	CALL CanU.00161FE4
0015B5A5	00E9	ADD CL,CH
0015B5A7	3E:8D FFFF0000	MOV EBP,0FFFF
0015B5AD	00E9 98130000	JMP CanU.0015C94A
0015B5B2	0087 042458E8	ADD BYTE PTR DS:[EDI+E8582404],AL
0015B5B8	9E	SAHF
0015B5B9	1A00	SBB AL,BYTE PTR DS:[EAX]
0015B5BB	0000	ADD BYTE PTR DS:[EAX],AL
0015B5BD	68 374FA4DC	PUSH DC444F37
0015B5C2	58	POP EAX
0015B5C3	81C0 CCBAC2D5	ADD EAX,D5C2BACC
0015B5C9	E8 82520000	CALL CanU.00160850
0015B5CE	0000	ADD BYTE PTR DS:[EAX],AL
0015B5D0	00E9 A6450000	JMP CanU.0015FB7B
0015B5D5	0089 0181C104	ADD BYTE PTR DS:[ECX+4C18101],CL
0015B5DB	0000	ADD BYTE PTR DS:[EAX],AL
0015B5DD	0089 0181C104	ADD BYTE PTR DS:[ECX+4C18101],CL
0015B5E3	0000	ADD BYTE PTR DS:[EAX],AL
0015B5E5	0068 F5	ADD BYTE PTR DS:[EAX-B],CH
0015B5E8	2216	AND DL,BYTE PTR DS:[ESI]
0015B5EA	00E9	ADD CL,CH
0015B5EC	65:4F	DEC EDI
0015B5EE	0000	ADD BYTE PTR DS:[EAX],AL
0015B5F0	0F83 55840000	JNB CanU.00163A4B
0015B5F6	00E9 74CEFFFF	JMP CanU.0015846F
0015B5FB	C3	RETN
0015B5FC	00E9 A0D6FFFF	JMP CanU.00158CA1
0015B601	00E9	ADD CL,CH
0015B603	2B11	SUB EDX,DWORD PTR DS:[ECX]
0015B605	0000	ADD BYTE PTR DS:[EAX],AL
0015B607	0067 64	ADD BYTE PTR DS:[EDI+64],AH
0015B60A	8F06	POP DWORD PTR DS:[ESI]
0015B60C	0000	ADD BYTE PTR DS:[EAX],AL
0015B60E	870424	XCHG DWORD PTR SS:[ESP],EAX
0015B611	58	POP EAX
0015B612	871424	XCHG DWORD PTR SS:[ESP],EDX
0015B615	871C24	XCHG DWORD PTR SS:[ESP],EBX
0015B618	00E9 98180000	JMP CanU.0015CEB5
0015B61D	0087 042481C8	ADD BYTE PTR DS:[EDI+C8812404],AL
0015B623	0001	ADD BYTE PTR DS:[ECX],AL
0015B625	0000	ADD BYTE PTR DS:[EAX],AL
0015B627	870424	XCHG DWORD PTR SS:[ESP],EAX
0015B62A	9D	POPAD
0015B62B	90	NOP
0015B62C	00E9 F9EAFFFF	JMP CanU.0015A12A
0015B631	0000	ADD BYTE PTR DS:[EAX],AL
0015B633	E8 9BA8FFFF	CALL CanU.00155ED3
0015B638	0000	ADD BYTE PTR DS:[EAX],AL
0015B63A	00E9 28050000	JMP CanU.0015BB67
0015B63F	00E9	ADD CL,CH
0015B641	97	ADD

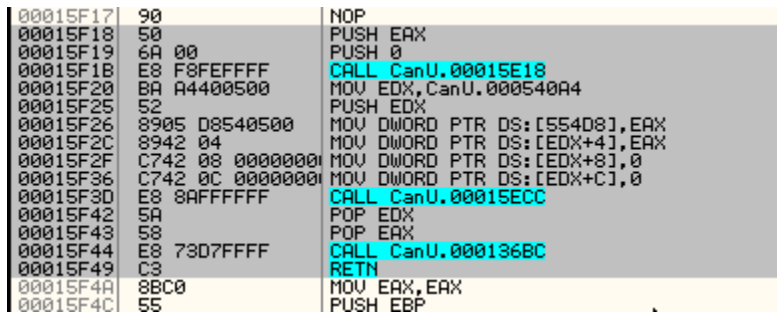
Si veo a este punto cuantas apis hay tengo:



¿que tal, todas estas dll, fueron apareciendo de la nada, pero ahi estan



Si coloco

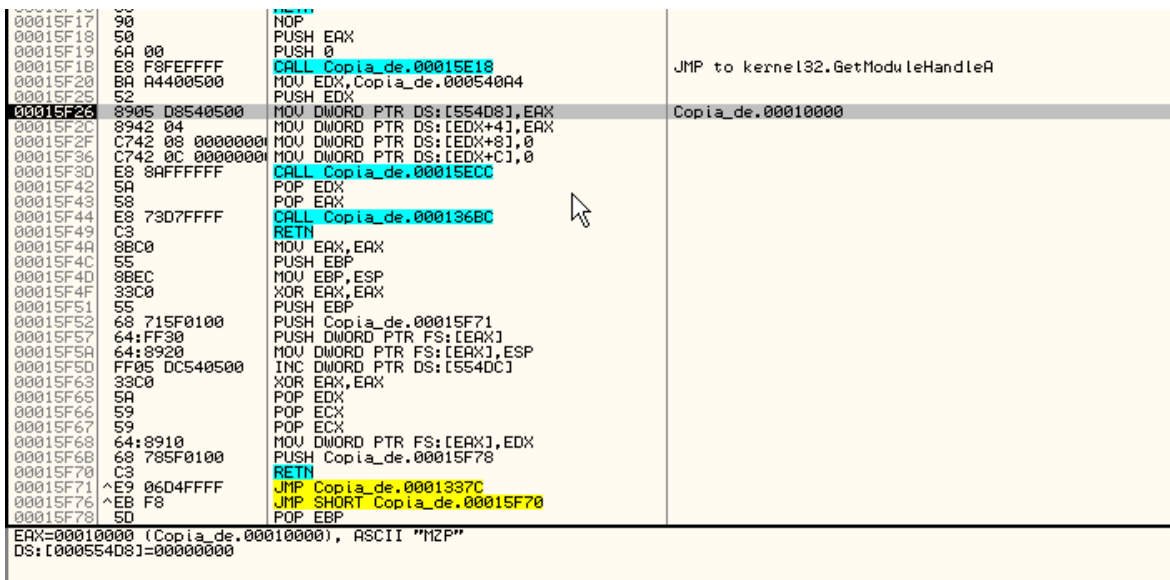


Me clavo en

Hago una pausa:

Pero si hacemos el dump, y colocamos esa iat, y el comienzo le coloco el lugar que mostre

Si ejecuto tengo esto:



Chachan, el lugar donde termina realmente las llamadas, y me demuestra el lugar expuesto para avanzar. Esto ya esta desempacado, pero esta mal alineado, obviamente debemos reconstruir el oep /fin de la pausa

En cualquier delphi, mueve antes del push eax, un valor, que corresponde a un valor importante de inicializacion, en este caso con un bp

Parametros OR, ROL, ROR,

000BA253	^E9 2032FFFF	JMP CanU.000AD478
000BA258	0F80 FE1D0100	J0 CanU.000CC05C
000BA25E	8B15 08270900	MOV EDX,DWORD PTR DS:[92708]
000BA264	09D2	OR EDX,EDX
000BA266	^E9 08E40000	JMP CanU.000C8673
000BA26B	C3	RET
000BA26C	68 88EF0500	PUSH CanU.0005EF88
000BA271	^E9 3C380100	JMP CanU.000CD0B2
DS:[00092708]=00000000		
EDX=00015F20 (CanU.00015F20)		

Encontrando denuevo un or edx, en resolucion de ciertos variables

Ahora comienzo a mirar un poco los ROL

En una posicion de

0009D97D	^70 5B	J0 SHORT CanU.0009D9D4
0009D97F	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
0009D982	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX
0009D985	52	PUSH EDX
0009D986	68 7FF82D05	PUSH 52DF87F
0009D98B	5A	POP EDX
0009D98C	C1C2 0D	ROL EDX,0D
0009D98F	^E9 C1790200	JMP CanU.0009C535
0009D994	8905 A0610500	MOV DWORD PTR DS:[561A0],EAX

EAX	0009DEE4	ASCII "user32.dll"
ECX	7C80E63B	kernel32.7C80E63B
EDX	BF0FE0A5	
EBX	00000000	
ESP	0028FF30	
EBP	0028FF40	
ESI	00000027	
EDI	00052E9C	CanU.00052E9C

Registers (FPU)	
AX	01F5FF48 ASCII "advapi32.dll"
CX	7C80E63B kernel32.7C80E63B
DX	BF0FE0A5
BX	7FFD8000
SP	01F5FF20
BP	01F5FF20

Registers (FPU)	
EAX	000AC150 ASCII "Software\Microsoft\Protected Storage System Provider"
ECX	7C92005D ntdll.7C92005D
EDX	BF0FE0A5
EBX	7FFD8000
ESP	0028F7A0
EBP	0028F7B0
ESI	FFFFFFFF

Registers (FPU)	
EAX	0028FD20 ASCII "\"Registry\User\""
ECX	0028FA36
EDX	BF0FE0A5
EBX	7FFD8000
ESP	0028F7B8
EBP	0028F7C8
ESI	FFFFFFFF
EDI	7C920228 ntdll.7C920228

Y luego de un rato, debemos estudiar lo que dicen String Encrypt/decrypt, en este caso el uso de xor , puede guiar.

00D10057	90	NOP
00D10058	90	NOP
00D10059	90	NOP
00D1005A	90	NOP
00D1005B	81F2 3CF7191A	XOR EDX,1A19F73C
00D10061	66:33C2	XOR AX,DX
00D10064	E8 19773AFF	CALL CanU.000B7782
00D10069	55	PUSH EBP
00D1006A	8BEC	MOV EBP,ESP
00D1006C	51	PUSH ECX
00D1006D	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
00D10070	E8 41AD3AFF	CALL CanU.000BADB6
00D10075	C1C1 19	ROL ECX,19
00D10078	81F1 C71AB3D7	XOR ECX,07B31AC7
00D1007E	F7C1 00000008	TEST ECX,80000000
00D10084	-0F84 CEAF38FF	JE CanU.0009B058
00D1008A	-0F85 70FA3CFF	JNZ CanU.000A58AA

Voy reconociendo donde es importante, solo con los rol, cuando es necesario revisar

Si tuviera mas tiempo quizas intentaria revisar si realmente

ror ->encripta rol->decripta

Patron Push /pop/{random}Rol=mov r32,api

Tambien hay ofuscaciones que son dificiles de entender hasta que se ve:

1) Push/pop/rol/xor /or /add= example->GetDesktopWindow

000782CC	..E9 2DDE0300	JMP 15.000B60FE
000782D1	> 68 5C51027B	PUSH 7B02515C
000782D6	. 5B	POP EBX
000782D7	. C1C3 17	ROL EBX,17
000782DA	. 81F3 EED5C736	XOR EBX,36C7D5EE
000782E0	. 81CB AAFF78CF	OR EBX,CF78EFAA
000782E6	. 81C3 BA650A20	ADD EBX,200A65BA
000782EC	..E9 3CB70200	JMP 15.000A3A2D
000782F1	\$ 870C24	XCHG DWORD PTR SS:[ESP],ECX
000782F4	..E9 3CB70200	JMP 15.000A3A2D

Desofuscado el mismo trozo:

000782CC	..E9 2DDE0300	JMP 15.000B60FE
000782D1	? 90	NOP
000782D2	? 90	NOP
000782D3	? 90	NOP
000782D4	? 90	NOP
000782D5	? 90	NOP
000782D6	. 90	NOP
000782D7	. 90	NOP
000782D8	? 90	NOP
000782D9	? 90	NOP
000782DA	. 90	NOP
000782DB	? 90	NOP
000782DC	? 90	NOP
000782DD	? 90	NOP
000782DE	? 90	NOP
000782DF	? 90	NOP
000782E0	. 90	NOP
000782E1	? 90	NOP
000782E2	? 90	NOP
000782E3	? 90	NOP
000782E4	? 90	NOP
000782E5	? 90	NOP
000782E6	. BB A8650500	MOV EBX,<&00000000>R32.GetDesktopWindow>
000782E7	? 90	NOP
000782E8	? 90	NOP
000782E9	..E9 3CB70200	JMP 15.000A3A2D

Mov ebx,valor donde decia jmp dword api

Otro ofuscado

2) Push/push /pop/sub/or/add ejemplo=5d110 ReadFile //CRC

0005D0F0	. 53	PUSH EBX
0005D0FE	. 68 70D790D8	PUSH D890D770
0005D103	. 5B	POP EBX
0005D104	. 81EB D8B812FE	SUB EBX,FE12B8D8
0005D10A	. 81CB CEB6D5CB	OR EBX,CBD5B6CE
0005D110	. 81C3 64EC0624	ADD EBX,2406EC64
0005D116	..E9 D5190700	JMP 18.000CEAF0
0005D11D	..E9 D5190700	JMP 18.000CEAF0

Desofuscado

000500F8	EB 05820300	JMP 18.00095302
000500FD	53	PUSH EBX
000500FE	90	NOP
000500FF	90	NOP
00050100	90	NOP
00050101	90	NOP
00050102	90	NOP
00050103	90	NOP
00050104	90	NOP
00050105	90	NOP
00050106	90	NOP
00050107	90	NOP
00050108	90	NOP
00050109	90	NOP
0005010A	90	NOP
0005010B	90	NOP
0005010C	90	NOP
0005010D	90	NOP
0005010E	90	NOP
0005010F	90	NOP
00050110	BB 42AC0600	MOV EBX,<JMP.&KERNEL32.ReadFile>
00050115	90	NOP

3) Push/pop/and/sub/add =mov ebp,1

Ofuscado:

000A7871	68 D410CFCF	PUSH CFCF10D4
000A7876	5D	POP EBP
000A7877	81E5 28DB2BDF	AND EBP,DF28DB28
000A787D	81ED A0D800E5	SUB EBP,E50D8A0
000A7883	81C5 A1C8F515	ADD EBP,15F5C8A1
000A7889	E9 BA2AFFFF	JMP 15.0009A348
000A788E	5F	POP EDI

Desofuscado:

000A7871	90	NOP
000A7872	90	NOP
000A7873	90	NOP
000A7874	90	NOP
000A7875	90	NOP
000A7876	90	NOP
000A7877	90	NOP
000A7878	90	NOP
000A7879	90	NOP
000A787A	90	NOP
000A787B	90	NOP
000A787C	90	NOP
000A787D	90	NOP
000A787E	90	NOP
000A787F	90	NOP
000A7880	90	NOP
000A7881	90	NOP
000A7882	90	NOP
000A7883	BD 01000000	MOV EBP,1
000A7888	90	NOP
000A7889	E9 BA2AFFFF	JMP 15.0009A348

4) Antes optimizar vease push+ret +jmp+jmp

00051844	55	PUSH EBP
00051845	8BEC	MOV EBP,ESP
00051847	33C0	XOR EAX,EAX
00051849	55	PUSH EBP
0005184A	68 69180500	PUSH 17.00051869
0005184F	64:FF30	PUSH DWORD PTR FS:[EAX]
00051852	64:8920	MOV DWORD PTR FS:[EAX],ESP
00051855	FF05 00580500	INC DWORD PTR DS:[55800]
00051858	33C0	XOR EAX,EAX
0005185D	5A	POP EDX
0005185E	59	POP ECX
0005185F	59	POP ECX
00051860	64:8910	MOV DWORD PTR FS:[EAX],EDX
00051863	68 70180500	PUSH 17.00051870
00051868	C3	RETN
00051869	E9 0E1BFCFF	JMP 17.0001337C
0005186E	EB F8	JMP SHORT 17.00051868
00051870	5D	POP EBP
00051871	C3	RETN
00051872	8BC0	MOV EAX,EAX
00051874	832D 00580500 0	SUB DWORD PTR DS:[55800],1
0005187B	C3	RETN

Desofuscando: push +ret

00051863	EB 0B	JMP SHORT 17.00051870
00051865	90	NOP
00051866	90	NOP
00051867	90	NOP
00051868	90	NOP
00051869	E9 0E1BFCFF	JMP 17.0001337C
0005186E	EB F8	JMP SHORT 17.00051868
00051870	5D	POP EBP
00051871	C3	RETN

Optimizado (los 2 saltos no los usara (claramente ganamos espacio):

00051853	FF05 00500500	INC DWORD PTR DS:[500500]
0005185B	33C0	XOR EAX,EAX
0005185D	5A	POP EDX
0005185E	59	POP ECX
0005185F	59	POP ECX
00051860	64:8910	MOV DWORD PTR FS:[EAX],EDX
00051863	5D	POP EBP
00051864	C3	RETN
00051865	90	NOP
00051866	90	NOP
00051867	90	NOP
00051868	90	NOP
00051869	90	NOP
0005186A	90	NOP
0005186B	90	NOP
0005186C	90	NOP
0005186D	90	NOP
0005186E	90	NOP
0005186F	90	NOP
00051870	90	NOP
00051871	90	NOP
00051872	8BC0	MOV EAX,EAX
00051874	832D 00580500 0	SUB DWORD PTR DS:[55800],1
0005187B	C3	RETN

5)Con analisis:

0009EB5A	✓E9 605E0200	JMP 18.000C49BF	
0009EB5F	87	DB 87	
0009EB60	04	DB 04	
0009EB61	24	DB 24	CHAR '\$'
0009EB62	E8	DB E8	
0009EB63	2B3C0100	DD 18.00013C2B	
0009EB67	B8	DB B8	
0009EB68	> CE	INTO	
0009EB69	0398 76E8BC3B	ADD EBX,DWORD PTR DS:[EAX+3BBCE876]	
0009EB6F	FF	DB FF	
0009EB70	FF	DB FF	
0009EB71	53	PUSH EBX	
0009EB72	✓E9 FCB9FCFF	JMP 18.0006A573	
0009EB77	> E8 99330200	CALL 18.000C1F15	
0009EB7C	48	DEC EAX	

Sin analisis

0009EB5A	✓E9 605E0200	JMP 18.000C49BF	
0009EB5F	870424	XCHG DWORD PTR SS:[ESP],EAX	
0009EB62	E8 2B3C0100	CALL 18.000B2792	
0009EB67	B8 CE039876	MOV EAX,769803CE	
0009EB6C	E8 BC3BFFFF	CALL 18.00092720	
0009EB71	53	PUSH EBX	
0009EB72	✓E9 FCB9FCFF	JMP 18.0006A573	
0009EB77	E8 99330200	CALL 18.000C1F15	

Ahora bien , un dato curioso EBP-8, tambien es importante y puedo ver:

Mov edx, dword ebp-valor

000A7FC6	5E	POP ESI	
000A7FC7	8B55 F8	MOV EDX,DWORD PTR SS:[EBP-8]	
000A7FCA	E8 03D30200	CALL CanU.u.000D52D2	
000A7FCF	E8 5039FDFF	CALL CanU.u.0007B924	
000A7FD4	1920	SBB DWORD PTR DS:[EAX],ESP	
Stack SS:[001FFF78]=00229B00, (ASCII "LastRun")			
EDX=00229B06			

Creo que con este comienzo en Canu , otros tambien pueden animarse.

Sigamos con otro trozo:Pero luego vemos una rutina interesante

000BADB6	870424	XCHG DWORD PTR SS:[ESP],EAX	
000BADB9	58	POP EAX	
000BADBA	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
000BADBD	8A00	MOV AL,BYTE PTR DS:[EAX]	
000BADBF	^0F89 D57EFCFF	JNS CanU.00082C9A	
000BADC5	2C 99	SUB AL,99	
000BADC7	✓E9 01790100	JMP CanU.000D2790	
000BADCC	870424	XCHG DWORD PTR SS:[ESP],EAX	
000BADCF	8B00	MOV EAX,EAX	

Ese sub al,99 es antidebug

Luego panoramas

000121DE	8BC6	MOV EAX,ESI
000121E0	8B52 08	MOV EDX,DWORD PTR DS:[EDX+8]
000121E3	83CA 02	OR EDX,2
000121E6	8910	MOV DWORD PTR DS:[EAX],EDX
000121E8	83C0 04	ADD EAX,4
000121EB	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
000121EE	FF05 20540500	INC DWORD PTR DS:[55420]
000121F4	83EB 04	SUB EBX,4
000121F7	011D 24540500	ADD DWORD PTR DS:[55424],EBX
000121FD	E8 52120000	CALL CanU.00013454
00012202	✓E9 84000000	JMP CanU.0001228B
00012207	3B1D 84540500	CMP EBX,DWORD PTR DS:[55484]
0001220D	✓7F 4A	JG SHORT CanU.00012259
0001220F	291D 84540500	SUB DWORD PTR DS:[55484],EBX
00012215	833D 84540500	CMP DWORD PTR DS:[55484],0C
0001221C	✓7D 0D	JGE SHORT CanU.0001222B
0001221E	031D 84540500	ADD EBX,DWORD PTR DS:[55484]
00012224	33C0	XOR EAX,EAX
00012226	A3 84540500	MOV DWORD PTR DS:[55484],EAX
0001222B	A1 88540500	MOV EAX,DWORD PTR DS:[55488]
00012230	011D 88540500	ADD DWORD PTR DS:[55488],EBX
00012236	8B03	MOV EDX,EBX
00012238	83CA 02	OR EDX,2
0001223B	8910	MOV DWORD PTR DS:[EAX],EDX
0001223D	83C0 04	ADD EAX,4
00012240	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
00012243	FF05 20540500	INC DWORD PTR DS:[55420]
00012249	83EB 04	SUB EBX,4
0001224C	011D 24540500	ADD DWORD PTR DS:[55424],EBX
00012252	E8 FD110000	CALL CanU.00013454
00012257	✓EB 32	JMP SHORT CanU.0001228B
00012259	8BC3	MOV EAX,EBX
0001225B	E8 BCFDFFFF	CALL CanU.0001201C
00012260	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
00012263	33C0	XOR EAX,EAX
00012265	5A	POP EDX
00012266	59	POP ECX
00012267	59	POP ECX
00012268	64:8910	MOV DWORD PTR FS:[EAX],EDX
0001226B	68 8B220100	PUSH CanU.0001228B
00012270	803D 45500500	CMP BYTE PTR DS:[55045],0
00012277	✓74 0A	JE SHORT CanU.00012283
00012279	68 34540500	PUSH CanU.00055434

Terminan solo en ret

00050000	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
00050003	FF05 20540500	INC DWORD PTR DS:[55420]
00050009	83EB 04	SUB EBX,4
0005000C	011D 24540500	ADD DWORD PTR DS:[55424],EBX
00050012	E8 3D342CFF	CALL CanU.00013454
00050017	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
0005001A	5F	POP EDI
0005001B	5E	POP ESI
0005001C	5B	POP EBX
0005001D	59	POP ECX
0005001E	59	POP ECX
0005001F	5D	POP EBP
00050020	C3	RETN
00050021	90	NOP

Creo que el uso de OFFSET , lo hace para no usar eax, ebp, ebx, por ende asi trabaja la maquina virtual, no tanto en robar bytes, sino emular todo, en un orden que parezca ilegible

Apis emuladas

Si busco donde tambien accede a lsidebugger vemos ripeado la api

0008751A	64:FF35 00000000	PUSH DWORD PTR FS:[0]
00087521	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
00087528	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008752F	✓E9 44760000	JMP unpack3.0008EB78
00087534	68 71C8739D	PUSH 9D73C871
00087539	58	POP EAX
0008753A	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008753B	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008753C	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008753D	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008753E	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008753F	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087540	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087541	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087542	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087543	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087544	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087545	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087546	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087547	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087548	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087549	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008754A	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008754B	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008754C	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008754D	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008754E	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008754F	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087550	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087551	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087552	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087553	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087554	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087555	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087556	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087557	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087558	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087559	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008755A	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008755B	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008755C	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008755D	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008755E	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008755F	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087560	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087561	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087562	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087563	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087564	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087565	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087566	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087567	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087568	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087569	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008756A	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008756B	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008756C	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008756D	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008756E	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008756F	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087570	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087571	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087572	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087573	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087574	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087575	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087576	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087577	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087578	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087579	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008757A	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008757B	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008757C	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008757D	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008757E	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008757F	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087580	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087581	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087582	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087583	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087584	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087585	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087586	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087587	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087588	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087589	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008758A	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008758B	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008758C	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008758D	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008758E	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008758F	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087590	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087591	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087592	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087593	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087594	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087595	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087596	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087597	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087598	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
00087599	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008759A	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008759B	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008759C	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008759D	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008759E	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
0008759F	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875A0	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875A1	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875A2	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875A3	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875A4	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875A5	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875A6	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875A7	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875A8	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875A9	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875AA	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875AB	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875AC	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875AD	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875AE	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875AF	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875B0	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875B1	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875B2	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875B3	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875B4	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875B5	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875B6	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875B7	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875B8	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875B9	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875BA	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875BB	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875BC	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875BD	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875BE	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875BF	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875C0	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875C1	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875C2	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875C3	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875C4	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875C5	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875C6	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875C7	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875C8	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875C9	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875CA	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875CB	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875CC	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875CD	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875CE	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875CF	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875D0	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875D1	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875D2	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875D3	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875D4	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875D5	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875D6	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875D7	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875D8	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875D9	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875DA	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875DB	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875DC	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875DD	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875DE	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875DF	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875E0	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875E1	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875E2	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875E3	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875E4	64:8B05 30000000	MOV EAX,DWORD PTR FS:[30]
000875E5	64:8B05 30000000	

Normalmente una estructura comun en findwindow es

```
8945 F8      MOV DWORD PTR SS:[EBP-8],EAX
837D F8 00   CMP DWORD PTR SS:[EBP-8],0
0F84 CAA534FE JE CanU.0006A5E2
0F85 8E3E35FE JNZ CanU.00073EAC
90          NOP
```

->el JE es una deteccion , si no salta, no estaremos detectado

Pero no todas las veces cae en estructuras como estas

Pero siguiendo con la accion comienzo de cero otra vez

```
0016451B E8 EBFEEFFF CALL CanU.0016440B
00164520 05 882E0000 ADD EAX,2E88
00164525 FFE0      JMP EAX
00164527 E8 04000000 CALL CanU.00164530
0016452C FFFF      ???
0016452E FFFF      ???
00164530 5E        POP ESI
00164531 C3        RETN
00164532 0000      ADD BYTE PTR DS:[EAX],AL
00164534 0000      ADD BYTE PTR DS:[EAX],AL
```

Estoy en tls:

Entro al call

```
0016440B E8 00000000 CALL CanU.00164410
00164410 58        POP EAX
00164411 53        PUSH EBX
00164412 51        PUSH ECX
00164413 56        PUSH ESI
00164414 57        PUSH EDI
00164415 50        PUSH EAX
00164416 8B1C24    MOV EBX,DWORD PTR SS:[ESP]
00164419 81EB 10441600 SUB EBX,CanU.00164410
0016441F B8 569D0000 MOV EAX,9D56
00164424 50        PUSH EAX
00164425 6A 04     PUSH 4
00164427 68 00100000 PUSH 1000
0016442C 50        PUSH EAX
0016442D 6A 00     PUSH 0
0016442F B8 C4700D00 MOV EAX,<kernel32.VirtualAlloc>
00164434 8B0418    MOV EAX,DWORD PTR DS:[EAX+EBX]
00164437 FFD0      CALL EAX
00164439 59        POP ECX
0016443A BA 885A1500 MOV EDI,CanU.00155A88
```

```
16440B E8 00000000 CALL CanU.00164410
164410 58        POP EAX
164411 53        PUSH EBX
164412 51        PUSH ECX
164413 56        PUSH ESI
164414 57        PUSH EDI
164415 50        PUSH EAX
164416 8B1C24    MOV EBX,
164419 81EB 10441600 SUB EBX,
16441F B8 569D0000 MOV EAX,
164424 50        PUSH EAX
164425 6A 04     PUSH 4
164427 68 00100000 PUSH 1000
16442C 50        PUSH EAX
16442D 6A 00     PUSH 0
16442F B8 C4700D00 MOV EAX,
164434 8B0418    MOV EAX,
164437 FFD0      CALL EAX
164439 59        POP ECX
16443A BA 885A1500 MOV EDI,
16443F 01DA     ADD EDI,
164441 52        PUSH EDI
164442 53        PUSH EBX
164443 50        PUSH EAX
164444 89C7     MOV EDI,
164446 89D6     MOV ESI,
164448 FC       CLD
164449 F3:A4    REP MOVSB
16444B B9 04591500 MOV ECX,
16444D 01D9     ADD ECX,
```

Backup
Copy
Binary
Assemble Space
Label :
Comment ;
Breakpoint
Toggle F2
Conditional Shift+F2
Conditional log Shift+F4
Run to selection F4
Follow Enter
New origin here Ctrl+Gray *
Go to
Follow in Dump
Memory, on access
Memory, on write
Hardware, on execution
Remove hardware breakpoint

Run y luego, run y descriptado

```
0016440B B8 885A1500 MOV EAX,CanU.00155A88
00164410 C3        RETN
00164411 53        PUSH EBX
00164412 51        PUSH ECX
00164413 56        PUSH ESI
00164414 57        PUSH EDI
00164415 50        PUSH EAX
00164416 8B1C24    MOV EBX,DWORD PTR SS:[ESP]
```

Al ret

00164514	05 125B0000	ADD EAX,5B12
00164519	FFE0	JMP EAX
0016451B	E8 EBFEFFFF	CALL CanU.0016440B
00164520	05 003E0000	ADD EAX,3E00

Llego a

0015B59A	0F89 9D5A0000	JNS CanU.0016103D
0015B5A0	E8 3F6A0000	CALL CanU.00161FE4
0015B5A5	00E9	ADD CL,CH
0015B5A7	3E:BD FFFF0000	MOV EBP,0FFFF
0015B5AD	E9 98130000	JMP CanU.0015C94A
0015B5B2	0087 042458E8	ADD BYTE PTR DS:[EDI+E8582404],AL
0015B5B8	9E	SAHF
0015B5B9	1A00	SBB AL,BYTE PTR DS:[EAX]
0015B5BB	0000	ADD BYTE PTR DS:[EAX],AL
0015B5BD	68 374FA4DC	PUSH DCA44F37

Bp memory en data y llego a la rutina antes mencionada

Luego de usar los mov, veo que las comparaciones son importantes, en este caso tengo anotado: "#89015983#"

Encuentro esto:

00086DB2	81C1 590551C3	ADD ECX,C3510559
00086DB4	59	MOV DWORD PTR DS:[ECX],EAX
00086DB5	837D F8 00	POP ECX
00086DB9	E9 47C02000	CMP DWORD PTR SS:[EBP-8],0
00086DBE	0F88 0098FEFF	JMP CanU.000AEA02
00086DBF	E8 E77EFFFF	JS CanU.00070644
00086DC4		JMP CanU.00075C00

Luego como no es cero salta a un CALL dword de ebp-8, que ira directo a ISDEBUGGERPRESENT B5F96

000B5F96	FF55 F8	CALL DWORD PTR SS:[EBP-8]	kernel32.IsDebuggerPresent
000B5F99	F7D8	NEG EAX	
000B5F9B	1BC0	SBB EAX,EAX	
000B5F9D	F7D8	NEG EAX	
000B5F9F	51	PUSH ECX	
000B5FA0	E9 BF2F0000	JMP CanU.000B8F64	
000B5FA5	0F85 09EFFFFF	JNZ CanU.000B4EB4	
000B5FA6	EB 00000000	JMP CanU.000B4EB4	

Vemos la primera llamada a isdebugger present y coloco un bp en access donde accede a este lugar:

00155A34	89FE	MOV ESI,E01	
00155A3C	29C6	SUB ESI,EAX	
00155A3E	F3:A4	REP MOVSB BYTE PTR ES:[EDI],BYTE PTR DS:	
00155A40	5E	POP ESI	
00155A41	0F85 FFFFFFFF	JMP CanU.00155935	

Veo que en el comienzo de todo esto fue descriptada

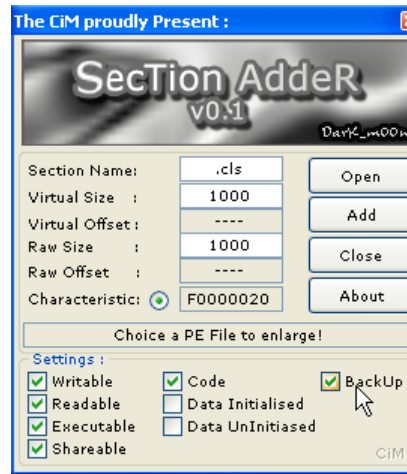
0000714D	FC	CLD	
0000714E	AC	LODS BYTE PTR DS:[ESI]	
0000714F	D0E8	SHR AL,1	
00007151	80F8 74	CMP AL,74	
00007154	75 0E	JNZ SHORT CanU.00007164	
00007156	8B06	MOV EAX,DWORD PTR DS:[ESI]	
00007158	0FC8	BSWAP EAX	
0000715A	01C8	ADD EAX,ECX	
0000715C	8906	MOV DWORD PTR DS:[ESI],EAX	
0000715E	83C6 04	ADD ESI,4	
00007161	83E9 04	SUB ECX,4	
00007164	49	DEC ECX	
00007165	7F E7	JG SHORT CanU.0000714E	
00007167	59	POP ECX	
00007168	5E	POP ESI	
00007169	C3	RETN	

este codigo es la parte mas importante de execryptor que hara los diversos saltos a las diversas secciones, conforme va descriptando, y se puede **inlinear aqui**, su region de descompresion

Realizando el Inline

Como bien se que a todos no resultara en unpacked, comencemos primero en inline, pues hay que hacer nuevas ideas:

Inserto una nueva seccion en el CANU.exe, no con topo, porque me lo borro el antivirus, esta vez usare una herramienta que hizo darkmoon en CIM. Pulso add y luego close



En mi caso estoy posicionado en 165000

00010000	00001000	CanU		PE header	Imag R	RWE	
00011000	00043000	CanU	CODE		Imag R	RWE	
00054000	00001000	CanU	DATA	data	Imag R	RWE	
00055000	00001000	CanU	BSS		Imag R	RWE	
00056000	00002000	CanU	yglhml1r		Imag R	RWE	
00058000	00001000	CanU	13ubr.ej		Imag R	RWE	
00059000	00001000	CanU	.tls		Imag R	RWE	
0005A000	00001000	CanU	.rdata		Imag R	RWE	
0005B000	00005000	CanU	bozrf3t4		Imag R	RWE	
00060000	00009000	CanU	.rsrc	resources	Imag R	RWE	
00069000	0006E000	CanU	nwakkug2		Imag R	RWE	
000D7000	0008E000	CanU	j7ojmhde	SFX, code, im	Imag R	RWE	
00165000	00001000	CanU	.cls		Imag R	RWE	
00170000	00001000				Priv RW	RW	
00180000	00001000				Priv RW	RW	

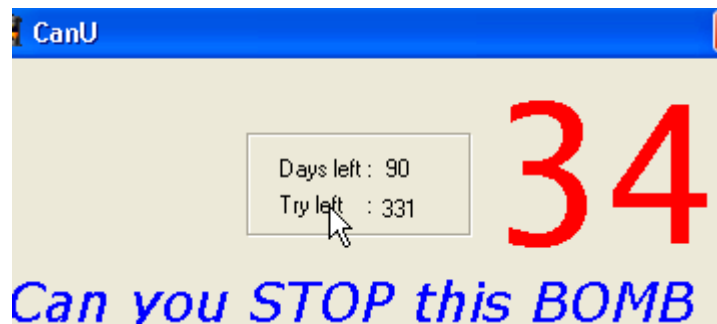
Como bien comentaba los thread a veces se resumen y a veces estan activo, asi que execryptor aun tiene algo mas.

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000008F4	7C810669	7FFD0000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s
00000910	7C810669	7FFD4000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0156 s
00000940	7C810669	7FFAB000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s
00000944	7C810669	7FFDB000	ERROR_INVALID_ID_WIN	Active	32 + 0	0.0000 s	0.0000 s
00000960	7C810669	7FFAD000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
00000CA0	7C810669	7FFDE000	ERROR_SUCCESS (000)	Active	32 - 15	0.2343 s	0.0000 s
00000CA4	7C810669	7FFDD000	ERROR_SUCCESS (000)	Active	32 - 15	0.0156 s	0.0000 s
00000F74	0016450F	7FFDF000	ERROR_SUCCESS (000)	Active	32 + 0	0.2968 s	0.0625 s



En forma resumida Pueden originar el [File corrupted!], esto es por 3 razones, compararon el handle con -1 (si es invalido) . el uso de createthread con antidebug, si encuentra que el tamaño no es igual con uso de readfile y otras apis como _lopen, si encuentra algo mas leyendo la aplicacion como es delphi en forma monitorizada, pues mensaje de aviso-

Pero con el cartel o no, El CanU esta corriendo con la seccion nueva



En la gran mayoria de los Execryptor, (en su seccion de codigo vemos esta rutina, esto es parte de un sector privilegiado para inline, por ende, si usan el unpacker de RSI, no funcionara para desempacar , pues esta zona fue presentada por rsi /Sunbeam el estilo es colocando un loop eterno, yo lo usare con el "codecave "expuesto por sunbeam.

Detectada la zona

000D714E	AC	LODS BYTE PTR DS:[ESI]
000D714F	D0E8	SHR AL,1
000D7151	80F8 74	CMP AL,74
000D7154	75 0E	JNZ SHORT canu_ret.000D7164
000D7156	8B06	MOV EAX,DWORD PTR DS:[ESI]
000D7158	0FC8	BSWAP EAX
000D715A	01C8	ADD EAX,ECX
000D715C	8906	MOV DWORD PTR DS:[ESI],EAX
000D715E	83C6 04	ADD ESI,4
000D7161	83E9 04	SUB ECX,4
000D7164	49	DEC ECX
000D7165	7F E7	JG SHORT canu_ret.000D714E

Y esta seccion puede ser usada en un codigo remoto usamos el jg para realizar el inline

000D7164	49	DEC ECX
000D7165	7F E7	JG SHORT CanU_inl.000D714E
000D7167	59	POP ECX
000D7168	5E	POP ESI
000D7169	C3	RET

,yo antes usaba la parte de comparacion con JC, pero veo que es mas facil aqui,

es casi la mejor instancia para el Inline en execryptor, pudiendo verificar con ESI, cuando la rutina esta desenscriptada, en versiones nuevas, suelen haber 2, pero en Canu solo hay una,

En este trozo usare lo siguiente:

000D7165	7F E7	JG SHORT canu_ret.000D714E
000D7167	E9 94DE0800	JMP canu_ret.00165000
000D716C	0010	ADD BYTE PTR DS:[EAX],DL
000D716E	0100	ADD DWORD PTR DS:[EAX],EAX

JMP codecave

Guardo los cambios y sigo en mi codigo:

00165000	9C	PUSHFD	
00165001	81FE FE650D00	CMP ESI,canu_ret.000D65FE	guardo los flags por el salto que usare
00165007	74 04	JE SHORT canu_ret.0016500D	cmp esi=ultimo a desencryptar //desencryptado
00165009	9D	POPFD	si es igual al ultimo, parchea la direccion
0016500A	59	POP ECX	restauro los flags
0016500B	5E	POP ESI	vuelve aqui:
0016500C	C3	RETN	
0016500D	9D	POPFD	
0016500E	C705 31520B00 C	MOV DWORD PTR DS:[B5231],900004C2	
00165018	90	NOP	
00165019	90	NOP	
0016501A	90	NOP	
0016501B	90	NOP	
0016501C	90	NOP	
0016501D	90	NOP	
0016501E	90	NOP	
0016501F	90	NOP	
00165020	90	NOP	
00165021	90	NOP	
00165022	90	NOP	
00165023	90	NOP	
00165024	90	NOP	
00165025	90	NOP	
00165026	90	NOP	
00165027	90	NOP	
00165028	90	NOP	
00165029	90	NOP	
0016502A	90	NOP	
0016502B	90	NOP	
0016502C	90	NOP	
0016502D	90	NOP	
0016502E	90	NOP	
0016502F	90	NOP	
00165030	90	NOP	
00165031	90	NOP	
00165032	90	NOP	
00165033	90	NOP	
00165034	90	NOP	
00165035	90	NOP	
00165036	EB D2	JMP SHORT canu_ret.0016500A	vuelve a pop ecx+esi+ret
00165038	0000	ADD BYTE PTR DS:[EAX],AL	
0016503A	0000	ADD BYTE PTR DS:[EAX],AL	

Coloco bp en execute en CreateThread *similar a cuando Trex explico como bypasear el crc o como indica

Full_Kill_Anti_debug_in_EXECryptor_V2.4X_by_Ahmadmansoor-eXeTools

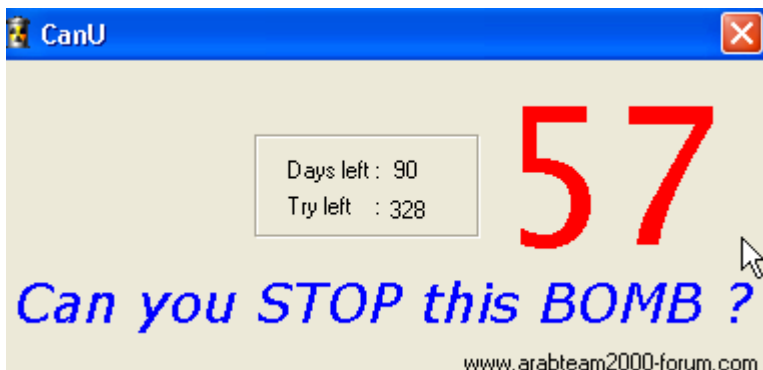
0028F92C	000BE00B	CALL to CreateThread
0028F930	00000000	pSecurity = NULL
0028F934	00000000	StackSize = 0
0028F938	000B5231	ThreadFunction = canu_ret.000B5231
0028F93C	00000000	pThreadParam = NULL
0028F940	00000000	CreationFlags = 0
0028F944	0028F948	pThreadId = 0028F948
0028F948	7C800000	kernel32.7C800000

Y me da que ira a la funcion de thread b5231, pues lo parcheo con RET 4, para que regrese como si hubiese usado un call. Otra opcion mejor, es matar el call de origen, pero por motivos de inline, solo haremos aquello para ahorrar codigo

Ahora el threadFuntion, uso los bytes, para parcharlo

Normalmente los delphis o algun packer, suelen usar SDK, pero normalmente el primer SDK es EC_antidebug, y luego sigue con algunos que desencryptan, encriptan, haran otras cosas. Ya habra tiempo de leer los SDK como lo enseñaron kioresk o sunbeam , solo hoy debemos intentar ejecutar unpacked esto.

El resultado de este inline esta a la vista:



Espero a que llegue a cero..y no hay file corrupt.

Asi que no fue necesario buscar patrones de crc, porque la version que tiene es menor a la 2.4.1

Vamos a la segunda hazaña, me gusta colocar bp condicionales en las apis, asi que usare bp en access en CreateThread

```
00058305 64:8925 00000000 MOV DWORD PTR FS:[0],ESP
0005830C 8B00 MOV EAX,DWORD PTR DS:[EAX]
0005830E B8 01000000 MOV EAX,1
00058313 ^E9 D6FFFFFF JMP CanU.inl.0005838E
00058318 53 PUSH EBX
00058319 -E9 20530600 JMP CanU.inl.000BD0DE
0005831E 8905 24610500 MOV DWORD PTR DS:[56124],EAX
```

Llega por aqui:

```
006D5C7 873424 XCHG DWORD PTR SS:[ESP],ESI
006D5CA 5E POP ESI
006D5CB 8B45 FC MOV EAX,DWORD PTR SS:[EBP-4]
006D5CE 8A00 MOV AL,BYTE PTR DS:[EAX]
006D5D0 2C 99 SUB AL,99
006D5D2 74 05 JMP CanU.inl.000D5D7
```

Y vemos

```
00093325 8B12 MOV EDX,DWORD PTR DS:[EDX] kernel32.CreateThread
00093327 F62A IMUL BYTE PTR DS:[EDX]
00093329 3C A4 CMP AL,0A4
0009332B ^0F85 A26BFEFF JNZ CanU.inl.00079ED3
0009332E 74 05 JMP CanU.inl.00093333
```

Y se me ocurre que siempre debe saltar y no dara error

```
00093325 8B12 MOV EDX,DWORD PTR DS:[EDX]
00093327 F62A IMUL BYTE PTR DS:[EDX]
00093329 3C A4 CMP AL,0A4
0009332B ^0F85 A26BFEFF JNZ CanU.inl.00079ED3 siempre debe saltar.
0009332E 74 05 JMP CanU.inl.00093333
00093330 1A0F SBB AL,0F
```

Tambien puedo encontrar cosas como que realmente es importante ebp-8

```
0008953F 837D F8 00 CMP DWORD PTR SS:[EBP-8],0
00089543 ^0F84 9910FEFF JE CanU.0006A5E2
00089549 ^E9 5B720300 JMP CanU.000C07A9
0008954E ^0F81 FA8FEFF JNO CanU.0007254E
00089554 8D05 E80D0700 LEA EAX,DWORD PTR DS:[70DE8]
0008955A 8945 FC MOV DWORD PTR SS:[EBP-4],EAX
0008955D 8D45 F8 LEA EAX,DWORD PTR SS:[EBP-8]
00089560 50 PUSH EAX
00089561 6A 00 PUSH 0
00089563 ^E9 DF4D0000 JMP CanU.0008E347
00089568 03D5 ADD EDI,EBP
0008956A C1C0 0A ROL EAX,0A
0008956D 81F0 A9AB90CB XOR EAX,CB90ABA9
00089573 52 PUSH EDI
00089574 68 693EDBCB PUSH CBDB3E69
00089579 870C24 XCHG DWORD PTR SS:[ESP],ECX
0008957C ^E9 ED130200 JMP CanU.000AA96E
00089581 0BC1 OR EAX,ECX
00089583 81CD 0D27C296 OR EBP,96C2270D
00089589 68 DBB7C300 PUSH 307CBB0B
```

Registers (FPU)

EAX	00000000
ECX	7C92005D ntdll.7C92005D
EDX	008D0003
EBX	0007874A CanU.0007874A
ESP	010EFF84 ASCII "OLLYDBG"
EBP	010EFFA4
ESI	00000000
EDI	0028F998
EIP	0008953F CanU.0008953F
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 1	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FFD7000(FFF)
T 0	GS 0000 NULL
D 0	
O 0	LastErr ERROR_SUCCESS (00000000)

Pero el uso

0006A5E2	0F82 E1840600	JB CanU.000D2AC9
0006A5E8	8D45 E0	LEA EAX,DWORD PTR SS:[EBP-20]
0006A5EB	C700 57697370	MOV DWORD PTR DS:[EAX],70736957
0006A5F1	68 50C8B7E9	PUSH E9B7C850
0006A5F6	E9 C7E00100	JMP CanU.000886C2
0006A5FB	81C6 B32044ED	ADD ESI,ED4420B3
0006A601	8916	MOV DWORD PTR DS:[ESI],EDX
0006A603	5E	POP ESI
Stack address=012EFF84, (ASCII "OLLYDBG")		
EAX=00000000		

han pensado que pasa si el contexto trabaja con handles invalidas y luego le dice, que envíe un nag?, provocando una excepcion, no sabia a que ventana volver sii, acertaron esta es la forma que usa para detectar

mediante thread miren lo que hace con el nombre de la handle:1

00AF0002	C745 E4 57696E6	MOV DWORD PTR SS:[EBP-1C],646E6957
00AF0009	8BC5	MOV EAX,EBP
00AF000B	C745 E8 6F77436	MOV DWORD PTR SS:[EBP-18],6C43776F
00AF0012	83E8 18	SUB EAX,18
00AF0015	90	NOP

Pero sin dar lios como nos detecta y como crashea si pilla un bp,

Sigo en el Inline

Como sabemos que el inline trabajara sobre el desempacado no habra lio, ya lo probe con otros y este no es la excepcion.

00165018	C705 29330900 3	MOV DWORD PTR DS:[93329],A3E9A43C
00165022	C705 2D330900 6	MOV DWORD PTR DS:[9332D],90FFFE6B
0016502C	90	NOP

Quedando asi , en el resultado de stack se ve como parcha correctamente

00165000	90	NOP
0016500E	C705 31520B00 C	MOV DWORD PTR DS:[B5231],900004C2
00165018	C705 29330900 3	MOV DWORD PTR DS:[93329],A3E9A43C
00165022	C705 2D330900 6	MOV DWORD PTR DS:[9332D],90FFFE6B
0016502C	90	NOP
0016502D	90	NOP
0016502E	90	NOP
0016502F	90	NOP
00165030	90	NOP
00165031	90	NOP
00165032	90	NOP
00165033	90	NOP
00165034	90	NOP
00165035	90	NOP
00165036	^EB D2	JMP SHORT CanU_inl.0016500A
00165038	0000	ADD BYTE PTR DS:[EAX],AL
0016503A	0000	ADD BYTE PTR DS:[EAX],AL
0016503C	0000	ADD BYTE PTR DS:[EAX],AL
0016503E	0000	ADD BYTE PTR DS:[EAX],AL
00165040	0000	ADD BYTE PTR DS:[EAX],AL
00165042	0000	ADD BYTE PTR DS:[EAX],AL
00165044	0000	ADD BYTE PTR DS:[EAX],AL
00165046	0000	ADD BYTE PTR DS:[EAX],AL
00165048	0000	ADD BYTE PTR DS:[EAX],AL
0016504A	0000	ADD BYTE PTR DS:[EAX],AL
0016504C	0000	ADD BYTE PTR DS:[EAX],AL
0016504E	0000	ADD BYTE PTR DS:[EAX],AL
00165050	0000	ADD BYTE PTR DS:[EAX],AL
00165052	0000	ADD BYTE PTR DS:[EAX],AL
00165054	0000	ADD BYTE PTR DS:[EAX],AL
00165056	0000	ADD BYTE PTR DS:[EAX],AL
00165058	0000	ADD BYTE PTR DS:[EAX],AL
0016505A	0000	ADD BYTE PTR DS:[EAX],AL
0016505C	0000	ADD BYTE PTR DS:[EAX],AL
0016505E	0000	ADD BYTE PTR DS:[EAX],AL
00165060	0000	ADD BYTE PTR DS:[EAX],AL
00165062	0000	ADD BYTE PTR DS:[EAX],AL

Address	Hex dump	Disassembly	Comm
00093329	3C A4	CMP AL,0A4	
0009332B	^E9 A36BFEFF	JMP CanU_inl.00079ED3	siem
00093330	90	NOP	
00093331	^E9 C2300300	JMP CanU_inl.000C63F8	Anulado el bp en las apis.

Restaurando el OEP.

Teniamos como idea que podiamos pasarlos de largo, pero el bp en access en code, luego del escrito llego aqui:

00015F18	50	PUSH EAX	
00015F19	6A 00	PUSH 0	
00015F1B	E8 F8FEFFFF	CALL <JMP.&KERNEL32.GetModuleHandleA>	pModule = NULL
00015F20	BA A4400500	MOV EDX,Copia_de.000540A4	GetModuleHandleA
00015F25	52	PUSH EDX	
00015F26	8905 08540500	MOV DWORD PTR DS:[55408],EAX	
00015F2C	8942 04	MOV DWORD PTR DS:[EDX+4],EAX	
00015F2F	C742 08 000000	MOV DWORD PTR DS:[EDX+8],0	
00015F36	C742 0C 000000	MOV DWORD PTR DS:[EDX+C],0	
00015F3D	E8 8AFFFFFFFF	CALL Copia_de.00015EC0	
00015F42	5A	POP EDX	
00015F43	58	POP EAX	
00015F44	E8 73D7FFFF	CALL Copia_de.00013680	
00015F49	C3	RET	
00015F4D	8B00	MOV EAX,FOV	

Y esta apuntando en un bp en ejecucion llegamos near oep.

00015F17	90	NOP	
00015F18	50	PUSH EAX	CanU_inl.00052E94
00015F19	6A 00	PUSH 0	
00015F1B	E8 F8FEFFFF	CALL CanU_in	
00015F20	BA A4400500	MOV EDX,Copia_de.000540A4	

Por lo tanto a reconstruir en el oep debe ser esto: o muy parecido

		PUSH EBP	
		MOV EBP,ESP	
		ADD ESP,-0C	
		MOV EAX,Copia_de.00052E94	
		CALL Copia_de.00015F18	

Push ebp

Mov ebp,esp

Add esp,-0c //tamaño stack para los dword

Mov eax, { valor en eax}

Call { lugar del push eax}

Luego de la rutina, debe volver a algun lugar, esto es la imagen, cuando pasa el ret

0009B6E7	-E9 F878FBFF	JMP CanU_inl.00052FE4	
0009B6EC	-E9 8A110100	JMP CanU_inl.000AC87B	
0009B6F1	F8 12FFD0FF	CALL CanU_inl.0007B608	

Pues ni modo.. todo esto esta con virtualizacion , normalmente hay varios dwords (normalmente 64) que debemos resolver, y asi va descriptando via flag y mediante registro uno en eax otro para ebp y así para cada registro, habiendo también condiciones similares en sitios iguales, condicionados en saltos.

Gracias a algun chino llamado softworm , compartio bien el codigo de vm EP, y otros analizan aquello, , y esto es lo que deberiamos buscar, pero dar con una rutina sin entender del todo chino, creo que no tiene sentido, mientras no estudie bien los patrones no seguire con eso.. Sigo con la otra idea

Reconstruccion del pseudo oep

Encontramos algun push eax y mas o menos una estructura similar a todo delphi , con la tool de deroko o bien al metodo que uno quiera.

Encontramos esto: (si usaramos IDA, pues tambien apunta que por aqui es el comienzo)

00015F18	. 50	PUSH EAX	
00015F19	. 6A 00	PUSH 0	
00015F1B	. E8 F8FEFFFF	CALL 00015E18	pModule = NULL
00015F20	. BA A4400500	MOV EDX,540A4	GetModuleHandleA

Bp y cuando iniciamos:

EAX	00052E94	c
ECX	001FFFB0	
EDX	7C91EB94	r
EBX	7FFD6000	
ESP	001FFFB0	
EBP	001FFFC0	
ESI	7C920738	r
EDI	7C91EE18	r
EIP	00015F18	c eax vale 52e94

stalk

001FFFB0	0009B6E7	canu_adi.0009B6E7
001FFFB4	0007B85F	canu_adi.0007B85F
001FFFB8	7C91EB94	ntdll.KiFastSystemCallRet
001FFBFC	000C0BC1	canu_adi.000C0BC1
001FFFC0	001FFFC0	
001FFFC4	7C816D4F	RETURN to kernel32.7C816D4F

Esto es claro, cuando llega al retn

00015F17	90	NOP
00015F18	50	PUSH EAX
00015F19	6A 00	PUSH 0
00015F1B	E8 F8FEFFFF	CALL 00015E18
00015F20	BA A4400500	MOV EDX,540A4
00015F25	52	PUSH EDX
00015F26	8905 08540500	MOV DWORD PTR [55408],EAX
00015F2C	8942 04	MOV DWORD PTR [EDX+4],EAX
00015F2F	C742 08 000000	MOV DWORD PTR [EDX+8],0
00015F36	C742 0C 000000	MOV DWORD PTR [EDX+C],0
00015F3D	E8 8AFFFFFF	CALL 00015ECC
00015F42	5A	POP EDX
00015F43	58	POP EAX
00015F44	E8 73D7FFFF	CALL 000136BC
00015F49	C3	RETN
00015F4A	8BC0	MOV EAX,EAX
00015F4C	55	PUSH EBP
00015F4D	8BEC	MOV EBP,ESP
00015F4E	33C0	XOR EAX,EAX

Return to 0009B6E7 (canu_adi.0009B6E7)

Ira al valor que estaba en stalk

2) lugar ideal para el parche:

Este salta a 0009B6E7 ^\E9 0090FDFF JMP 000746EC

Si vamos a aquel lugar

000746EC E8 24D80400 CALL 000C1F15 ; 63.000C1F15

Si veo las referencias de este call,tengo suficientes como para pensar que esto es un cripter

References in 63:nwaxkuq2 to 000C1F15 , son 63 referencias , si alteramos el call, pues tendremos la posibilidad de cambiar en vivo desde este lugar e ideal para quitar las limitaciones.

Anexo ademas algun form:

00052B80: TForm1.FormCreate

00052E0C: TForm1.FormMouseMove

00052D68: TForm1.Label4DbClick

00052DE0: TForm1.Label4MouseMove

00052C0C: TForm1.Timer1Timer

00052C2C: TForm1.Timer2Timer

00052D50: TForm1.Timer3Timer

A desempacar y de oep tenemos esto, es un delphi:

00052FC0	74280500	DD CanU_u.00052B74	
00052FC4	642E0500	DD CanU_u.00052E64	
00052FC8	342E0500	DD CanU_u.00052E34	
00052FCC	00	DB 00	
00052FCD	00	DB 00	
00052FCE	00	DB 00	
00052FCF	00	DB 00	
00052FD0	6C2E0500	DD CanU_u.00052E6C	
00052FD4	:- E9 746E0500	JMP 000A9E4D	CanU_u.000A9E4D
00052FD9	52	DB 52	CHAR 'R'
00052FDA	E1	DB E1	
00052FDB	8C	DB 8C	
00052FDC	7D	DB 7D	CHAR 'J'
00052FDD	22	DB 22	CHAR 'M'
00052FDE	BE	DB BE	
00052FDF	9A	DB 9A	
00052FE0	44	DB 44	CHAR 'D'
00052FE1	8A	DB 8A	
00052FE2	17	DB 17	
00052FE3	D1	DB D1	
00052FE4	:- E9 03170200	JMP 000746EC	CanU_u.000746EC
00052FE9	E8	DB E8	
00052FEA	60BB0400	DD CanU_u.0004BB60	
00052FEE	CA	DB CA	
00052FEF	68	DB 68	CHAR 'h'
00052FF0	B9	DB B9	
00052FF1	18	DB 18	
00052FF2	6D	DB 6D	CHAR 'm'
00052FF3	00	DB 00	
00052FF4	30	DB 30	CHAR '0'

Coloco nop+ret debajo del salto y espero ver que valor utilizara (metodo para ver donde caera)

00052FC0	C3	RETN	
00052FC8	C3	RETN	
00052FDC	C3	RETN	
00052FDD	C3	RETN	
00052FDE	C3	RETN	
00052FDF	C3	RETN	
00052FE0	C3	RETN	
00052FE1	C3	RETN	
00052FE2	C3	RETN	
00052FE3	C3	RETN	
00052FE4	C3	RETN	
00052FE5	C3	RETN	
00052FE6	C3	RETN	
00052FE7	C3	RETN	
00052FE8	C3	RETN	
00052FE9	C3	RETN	
00052FEA	C3	RETN	
00052FEB	C3	RETN	
00052FEC	C3	RETN	
00052FED	C3	RETN	
00052FEE	C3	RETN	

Return to 001FFFE0

Ahora tengo mas que claro que restauro los bytes y luego sigo de donde proviene aquel salto. Y el objetivo es restaurar el oep, y quitar las referencias al oep, y apuntarla a otra dirección, esto es debido a que no quiero ver que crashee asi evitaremos que algo vaya a nuestro real oep, en caso x, también se pueden redirigir los saltos a los ret a un lugar común y ayudar a revisar algun poco de codigo `jmp ret1 jmp ret2` nuevo desde las otras direcciones algunos ret Reconstituyo el oep, y evito las primeras instancias de antidebug por estos intentos leves, Quedando el OEP, reestablecido.

00052FD4	55	PUSH EBP	
00052FD5	8BEC	MOV EBP,ESP	
00052FD7	83C4 F4	ADD ESP,-0C	
00052FDA	B8 942E0500	MOV EAX,52E94	valor de eax*
00052FDF	E8 342FFCFF	CALL 00015F18	sys init
00052FE4	:- E9 FE860400	JMP 0009B6E7	valor de stalk

Parametro Call offset, push eax, ror r32,const =call api

Hay que seguir estudiando mas, las bases de las apis son usadas para calcular el valor, y luego es accedida directamente, esto es desde el OEP, hacia la ejecucion

Intentare buscar estructuras con ROR, OR, y otras a modo de reconstruir las apis:

El ejemplo mas facil de buscar es la estructura

Encontramos en push eax, el valor de la api en esp

0009795A	81F2 3DBA4AF2	XOR EDX,F24ABA3D	
00097960	E8 64A80200	CALL CanU_u_r.000C21C9	call imp, esp, y eax ->apis
00097965	50	PUSH EAX	
00097966	C1C8 0C	ROR EAX,0C	
00097969	8905 508C0B00	MOV DWORD PTR DS:[B8C50],EAX	
0009796F	E9 137EFCFF	JMP CanU_u_r.0005F787	

Ahora bien la primera llamada es de

00074E3A	BA 04C2F5AF	MOV EDX,0FF5C204	
00074E3F	E8 85D30400	CALL CanU_u_r.000C21C9	eax =user 32 base
00074E44	50	PUSH EAX	user32.SetTimer
00074E45	C1C8 06	ROR EAX,6	
00074E48	E9 E1990400	JMP CanU_u_r.000BE82E	
00074E4D	0BF2	OR ESI,EDX	
00074E4F	E8 84690300	CALL CanU_u_r.000AB7D8	
00074E54	E9 07C90100	JMP CanU_u_r.00091760	
00074E59	E8 AA670000	CALL CanU_u_r.0007B608	
EAX=7E398C2E (user32.SetTimer)			

Set Timer

000BB7FB	8905 24B50800	MOV DWORD PTR DS:[8B524],EAX	
000BB801	C3	RETN	
000BB802	E9 9C11FAFF	JMP CanU_u_r.0005C9A3	

Guarda la variable con uso de ror y luego

Ret->salta a la api

Address	Hex dump	Disassembly	
000C6A87	55	PUSH EBP	
000C6A88	E8 57AFF0FF	CALL CanU_u_r.000A19E4	
000C6A8D	E9 56ACFEFF	JMP CanU_u_r.000B16E8	
000C6A92	E9 45F2FEFF	JMP CanU_u_r.000B5CDC	
000C6A97	E9 2362FDFF	JMP CanU_u_r.0009CCBF	
000C6A9C	E9 0EBAFDFF	JMP CanU_u_r.000A24AF	
000C6AA1	FF25 70650500	JMP DWORD PTR DS:[<&USER32.GetMenuState u	
000C6AA7	E9 B548FFFF	JMP CanU_u_r.000BB361	
001FFF50	00075873	CALL to SetTimer from CanU_u_r.0007586E	
001FFF54	00000000	hwnd = NULL	
001FFF58	00000000	TimerID = 0	
001FFF5C	000075D8	Timeout = 30168. ms	
001FFF60	000C6A87	Timerproc = CanU_u_r.000C6A87	
001FFF64	00000001		
001FFF68	000075D8		
001FFF6C	000C6A87	CanU_u_r.000C6A87	
001FFF70	001FFF84		
001FFF74	000C6A87	RETURN to CanU_u_r.000C6A87 from CanU_u_r.0005	

La segunda con el mismo parametro

000B5F3F	BA 0F3B4120	MOV EDX,20413B0F	
000B5F44	E8 80C20000	CALL CanU_u_r.000C21C9	
000B5F49	50	PUSH EAX	
000B5F4A	C1C8 0E	ROR EAX,0E	
000B5F4D	E9 3A17FFFF	JMP CanU_u_r.000A768C	

000B5F3F	BA 0F3B4120	MOV EDX,20413B0F	
000B5F44	E8 80C20000	CALL CanU_u_r.000C21C9	
000B5F49	50	PUSH EAX	
000B5F4A	C1C8 0E	ROR EAX,0E	
000B5F4D	E9 3A17FFFF	JMP CanU_u_r.000A768C	
000B5F52	C3	RETN	
000B5F53	E9 C7A0FBFF	JMP CanU_u_r.0007001F	
EAX=7C80DE05 (kernel32.GetCurrentProcess)			

000B3298	BE B5000007	MOV ESI,878000B5	
000B329D	8906	MOV DWORD PTR DS:[ESI],EAX	
000B329F	5E	POP ESI	
000B32A0	C3	RETN	
000B32A1	E9 AA43FDFF	JMP CanU_u_r.00087650	
EAX=7815F203 DS:[000AF838]=00000000			

Y ahora si hago un seguimiento ira a

00002337	81C2 6B95DEAC	ADD EDX,ACDE956B	
0000233D	8B12	MOV EDX,DWORD PTR DS:[EDX]	kernel32.CheckRemoteDebuggerPresent
0000233F	871424	XCHG DWORD PTR SS:[ESP],EDX	
00002342	C3	RETN	
00002343	✓E9 33380000	JMP CanU_u_r.00005B7B	
00002348	^0F89 3292FCFF	JNS CanU_u_r.0009B580	
0000234E	F7D3	NOT EBX	
00002350	^E9 7D8BFAFF	JMP CanU_u_r.0007AED2	
00002355	^0F8D B130FEFF	JGE CanU_u_r.000B540C	
0000235B	^E9 ECF1FCFF	JMP CanU_u_r.000A154C	
00002360	^0F85 2D92FDFF	JNZ CanU_u_r.000AB593	
00002366	23CD	AND ECX,EBP	
00002368	2BC6	SUB EAX,ESI	
0000236A	^0F84 1245FFFF	JE CanU_u_r.000C6882	
Stack DS:[001FFF74]=7C859CAE (kernel32.CheckRemoteDebuggerPresent) EDX=001FFF74			

Guardando en EBP-1

000AF96F	33C0	XOR EAX,EAX	
000AF971	8845 FF	MOV BYTE PTR SS:[EBP-1],AL	
000AF974	8A45 FF	MOV AL,BYTE PTR SS:[EBP-1]	
000AF977	✓E9 6A500100	JMP CanU_u_r.000C49E6	
AL=00 Stack SS:[001FFF7F]=00			

Y retornar en

000C49E1	✓E9 78500000	JMP CanU_u_r.000C9A5E	
000C49E6	8BE5	MOV ESP,EBP	
000C49E8	5D	POP EBP	
000C49E9	C3	RETN	
000C49EA	^E9 56B4FBFF	JMP CanU_u_r.0007FE45	
000C49EF	B0 01	MOV AL,1	
000C49F1	^E9 376DF9FF	JMP CanU_u_r.0005B72D	
000C49F6	^0F85 C70E0100	JNZ CanU_u_r.000058C3	
000C49FC	^E9 BCECFDFF	JMP CanU_u_r.000A36BD	
000C4A01	C1E7 02	SHL EDI,2	
000C4A04	3BF2	CMP ESI,EDX	
000C4A06	✓E9 C40E0000	JMP CanU_u_r.000C58CF	
000C4A0B	5F	POP EDI	
Return to 0009F389 (CanU_u_r.0009F389)			

Y luego de ahí ira a el check de las apis, ahora bien siguiendo con la mini estructura

000B0FF2	^E9 4B3BFFFF	JMP CanU_u_r.000A4B42	
000B0FF7	E8 CD110100	CALL CanU_u_r.000C21C9	
000B0FFC	50	PUSH EAX	
000B0FFD	C1C8 18	ROR EAX,18	
000B1000	8905 48380900	MOV DWORD PTR DS:[93848],EAX	
000B1006	C3	RETN	
000B1007	^E9 2833FCFF	JMP CanU_u_r.00074334	
000B100C	✓E9 5F610000	JMP CanU_u_r.000B7170	
000B1011	^E9 FDF0FBFF	JMP CanU_u_r.00070113	
000B1016	^E9 F6C9FDFF	JMP CanU_u_r.0008DA11	
000B101B	C1C0 18	ROL EAX,18	
000B101E	52	PUSH EDX	
000B101F	^E9 883BFEFF	JMP CanU_u_r.00094BAC	
000B1024	^0F8C 6A7DFAFF	JL CanU_u_r.00058D94	
000B102A	68 29D58AC3	PUSH C38AD529	
EAX=7C80C118 (kernel32.SetThreadPriority)			

```

00C3FF90  000C8B00  /CALL to SetThreadPriority from CanU_u_r.000C8AFB
00C3FF94  FFFFFFFE  |hThread = FFFFFFFE
00C3FF98  FFFFFFF1  \Priority = THREAD_PRIORITY_IDLE

```

Luego otra mas

0006B28E	E8 366F0500	CALL CanU_u_r.000C21C9	
0006B293	50	PUSH EAX	
0006B294	C1C8 04	ROR EAX,4	
0006B297	^E9 550AFFFF	JMP CanU_u_r.0005BCF	
0006B29C	68 41040C00	PUSH CanU_u_r.000C04	
0006B2A1	✓E9 E3870400	JMP CanU_u_r.000B3A8	
0006B2A6	B8 D7B080B9	MOV EAX,B980BDD7	
0006B2AB	E8 A8EF0400	CALL CanU_u_r.000BA2	
EAX=7E39CD97 (user32.EnumWindows)			

000B8ECD	66:FD	STD
000B8ECF	FB	STI
000B8ED0	E8 F4920000	CALL CanU_u_r.000C21C9
000B8ED5	50	PUSH EAX
000B8ED6	C1C8 06	ROR EAX,6
000B8ED9	68 7C2D0C00	PUSH CanU_u_r.000C2D7C
000B8EDE	E8 1C970000	CALL CanU_u_r.000C25FF
000B8EE3	3825 802FA200	CMP BYTE PTR DS:[A22FA80],AH
EAX=7C91D92E (ntdll.ZwQuerySystemInformation)		

Y Si me detecta accede a estas:->

00098D53	810424	ACMG DWORD PTR DS:[E8F3],EAX
00098D58	58	POP EAX
00098D59	E8 6B940200	CALL CanU_u_r.000C21C9
00098D5E	50	PUSH EAX
00098D5F	C1C8 05	ROR EAX,5
00098D62	8905 40B50800	MOV DWORD PTR DS:[8B540],EAX
00098D68	^E9 377CFFFF	JMP CanU_u_r.000909A4
00098D6D	^0F8D 5F96FFFF	JGE CanU_u_r.000923D2
00098D73	03FB	ADD EDI,EBX
00098D75	^E9 BD7A0200	JMP CanU_u_r.000C0837
EAX=7C83293F (kernel32.ResumeThread)		

0006CB3D	81F2 2CE08C12	XOR EDX,128CE02C
0006CB43	E8 81560500	CALL CanU_u_r.000C21C9
0006CB48	50	PUSH EAX
0006CB49	C1C8 1C	ROR EAX,1C
0006CB4C	57	PUSH EDI
0006CB4D	8BFB	MOV EDI,EBX
0006CB4F	873C24	XCHG DWORD PTR SS:[ESP],EDI
0006CB52	^E9 18300000	JMP CanU_u_r.0006FB6F
0006CB57	872C24	XCHG DWORD PTR SS:[ESP],EBP
EAX=77DCC110 (advapi32.LookupPrivilegeValueA)		

00099D51	BA DA42B28C	MOV EDX,8CB242DA
00099D56	E8 69840200	CALL CanU_u_r.000C21C9
00099D5B	50	PUSH EAX
00099D5C	C1C8 07	ROR EAX,7
00099D5F	^E9 41960200	JMP CanU_u_r.000C33AA
00099D64	0F8B DE2A0300	JPD CanU_u_r.000CC84D
00099D6F	^0F8D 0E6E0200	JGE CanU_u_r.000C0883
EAX=77DA7CA9 (advapi32.AllocateAndInitializeSid)		

Parametro Rol r32,const

Mostrare una costumbre de execryptor que no suelen comentar La gran mayoria de **rol r32, const**

dan origen a apis, las cuales busca si es valor cero, la calcula, si no es calcula esta parte, dando origen a la api, previamente calculada, gracias a esto si es cero, se puede generar, si tenemos tiempo podemos anular todo el trozo y colocar las apis correctas, a modo que pueda correr en cualquier s.o.

Todas las direcciones iran directamente a la api, veamos el listado que encontraremos con este patron.

00087003	C1C8 0B	ROL EAX,0B
0008700C	^E9 100FFFFF	JMP CanU.00077F21
00087011	E8 0E49FFFF	CALL CanU.0007B924
00087016	7A 23	JPE SHORT CanU.0008703B

Registers (FPU)

EAX	7C802520	kernel32.WaitForSingleObject
ECX	7C810613	kernel32.7C810613
EDX	FFFFFFFF	

Usando aquello el uso de rol o ror , muestro la Donde se ven algunas como

Registers (FPU)	Registers (FPU)
7C911000 ntdll.RtlEnterCriticalSection	7C9110E0 ntdll.RtlLeaveCriticalSection
00158910 CanU.00158910	00000012
00158910 CanU.00158910	00004010

000A9E08	C1C8 06	ROL EAX,6
000A9E0E	53	PUSH EBX
000A9E0F	8BD8	MOV EBX,EAX
000A9E11	871C24	XCHG DWORD PTR SS:[ESP],EBX
000A9E14	E8 11ADFEFF	CALL CanU.00094B2A
000A9E19	^E9 F361FCFF	JMP CanU.00070011

Registers (FPU)

7C809B57	kernel32.CloseHandle
7C8025F0	kernel32.7C8025F0
7C91E514	ntdll.KiFastSystemCallRet
2EE65426	
00000000	

Registers (MMX)		Registers (MMX)	
EAX	7C810647 kernel32.CreateThread	EAX	7C80FD3D kernel32.GlobalAlloc
ECX	0028F5D4	ECX	01E7FF6C
EDX	000C56F3 ASCII "h#Eo"	EDX	7C80FE35 kernel32.7C80FE35
EBX	00000000	EBX	001C0001

Luego push eax+retn->es el salto directo a la api.

Creo que esta mini reseña, o parte de como se resuelve las apis, tambien se puede encontrar en parte en el escrito de Evolution , luego de Unpacked, cuando se daba cuenta que habian apis que no estaban reparadas,

Script para la iat

Incursionado cuando descripta, creo que debo usar el arsenal y dejarlo unpacked, pues el tiempo que tengo es muy escazo. yo creo que en este caso usare el script de trickboy para ayudarme, cerca del oep ya estaba.

Los 2 mensajes me indican el VM oep, y el OEP

```

-----
MSG ODbgScript
-----
VM EP: A9E4D RVA: 99E4D
-----
Aceptar  Cancelar
-----
MSG ODbgScript
-----
OEP: 52FD4 RVA: 42FD4
-----
Aceptar  Cancelar
-----

```

52fd4

Y para la iat, usare temporalmente solo para saber donde comienza y donde termina:

```

\ExeCryptor 2.xx OEP Finder + IAT Repair v1.0.txt
-----

```

El resultado me muestra el comienzo y fin de la iat

```

30023 Breakpoint at 01730023
      iat_start: 00056104
      iat_end: 000566D0

```

Configuro en el de kagra o pe/kill

```

//=====
// Èièòèàèèçèòòâi IAT
//=====
mov iat_start,0056104
mov iat_end,00566D0

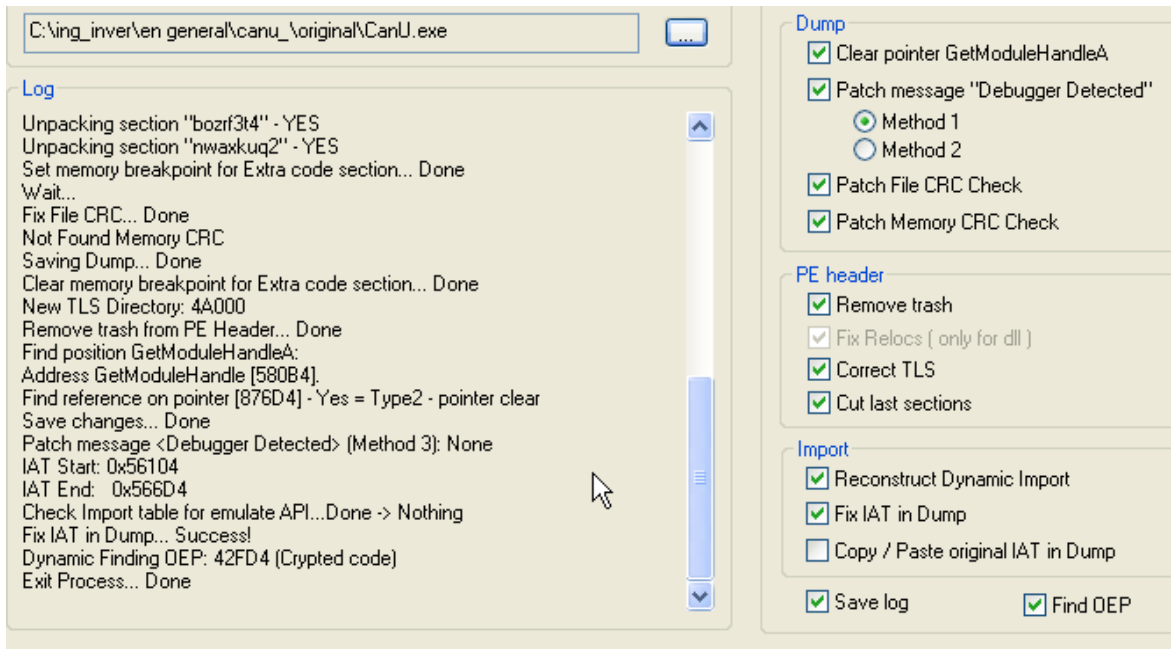
//=====
// Îiéo÷ââi èiôïôïàòèþ î îðîòâññâ

```

podria seguir con script, pero creo que por comodidad y por la cantidad de versiones, solo usare usar solo el unpacker de RSI sobre el original, (porque el inline, sera el objetivo)

Unpacker RSI

El cual resume la hazaña en 1 movimiento



Unpacked.exe

O bien sin tildar cada crc check ni eliminar el pointer, pues el ret de createthread, hace el trabajo de eliminar el antidebug, pero no elimina los otros antidebug como closehandle, createthread+waitfordebug..

O bien en el oep, arreglar la imagebase con LORD PE, Dumpear con import rec y restaurar la iat con el plugin de deroko y/o complementar con las herramientas como UIF 1.2, dejo a eleccion del lector la forma que quiera desempacar a este CANU

Venciendo a la bomba en el Unpacked

Desempacado, hay que ver la bomba, solo hay 1 minuto, y varios intentos, Vamos a Atacar al tiempo:

Como hablamos de tiempo en aumento y disminucion, deberia estar relacionado a settimer lo vi antes, si hablamos de delphi debo terminar o revisar algun ID o control,

, pero si hablamos del execryptor en si, no usa el valor del pc, sino el valor establecido en regedit, bajo una premisa de sdk, gettrialdays, y similares

Ahora bien, Como tengo el unpacked a mano, comienzo a atacar

Si coloco un ret en el comienzo de esta rutina que dice killtimer: el reloj se detiene

00052534	C3	RETN	
00052535	8BEC	MOV EBP,ESP	
00052537	6A 00	PUSH 0	
00052539	53	PUSH EBX	
0005253A	56	PUSH ESI	
0005253B	8B08	MOV EBX,EAX	
0005253D	33C0	XOR EAX,EAX	
0005253F	55	PUSH EBP	
00052540	68 B3250500	PUSH CanU_u2.000525B3	
00052545	64:FF30	PUSH DWORD PTR FS:[EAX]	
00052548	64:8920	MOV DWORD PTR FS:[EAX],ESP	
0005254B	6A 01	PUSH 1	
0005254D	8B43 28	MOV EAX,DWORD PTR DS:[EBX+28]	TimerID = 1
00052550	50	PUSH EAX	hWnd
00052551	E8 1641FCFF	CALL <JMP.&USER32.KillTimer>	KillTimer
00052556	8B73 24	MOV ESI,DWORD PTR DS:[EBX+24]	
00052559	85F6	TEST ESI,ESI	
0005255B	74 40	JE SHORT CanU_u2.00052590	
0005255D	807B 38 00	CMP BYTE PTR DS:[EBX+38],0	
00052561	74 3A	JE SHORT CanU_u2.00052590	
00052563	66:837B 32 00	CMP WORD PTR DS:[EBX+32],0	
00052568	74 33	JE SHORT CanU_u2.00052590	
0005256A	6A 00	PUSH 0	Timerproc = NULL
0005256C	56	PUSH ESI	Timeout
0005256D	6A 01	PUSH 1	TimerID = 1
0005256F	8B43 28	MOV EAX,DWORD PTR DS:[EBX+28]	hWnd
00052572	50	PUSH EAX	SetTimer
00052573	E8 2C42FCFF	CALL <JMP.&USER32.SetTimer>	
00052578	85C0	TEST EAX,EAX	
0005257A	75 21	JNZ SHORT CanU_u2.00052590	
0005257C	8D55 FC	LEA EDI,DWORD PTR SS:[EBP-4]	
0005257F	A1 F44C0500	MOV EAX,DWORD PTR DS:[54CF4]	
00052584	E8 F32BFCFF	CALL CanU_u2.00015170	
00052589	8B4D FC	MOV ECX,DWORD PTR SS:[EBP-4]	
0005258C	B2 01	MOV DL,1	
0005258E	A1 78BB0100	MOV EAX,DWORD PTR DS:[1BB78]	
00052593	E8 2468FCFF	CALL CanU_u2.000180B0	
00052598	E8 170EFCFF	CALL CanU_u2.000133B4	
0005259D	33C0	XOR EAX,EAX	
0005259F	5D	POP EAX	



Luego de dejar el pc, como **20 minutos**, esperando que se cerrara por algo..no ocurrio nada, asi que estamos ok, matamos el timer.Cualquiera se sentiria orgulloso

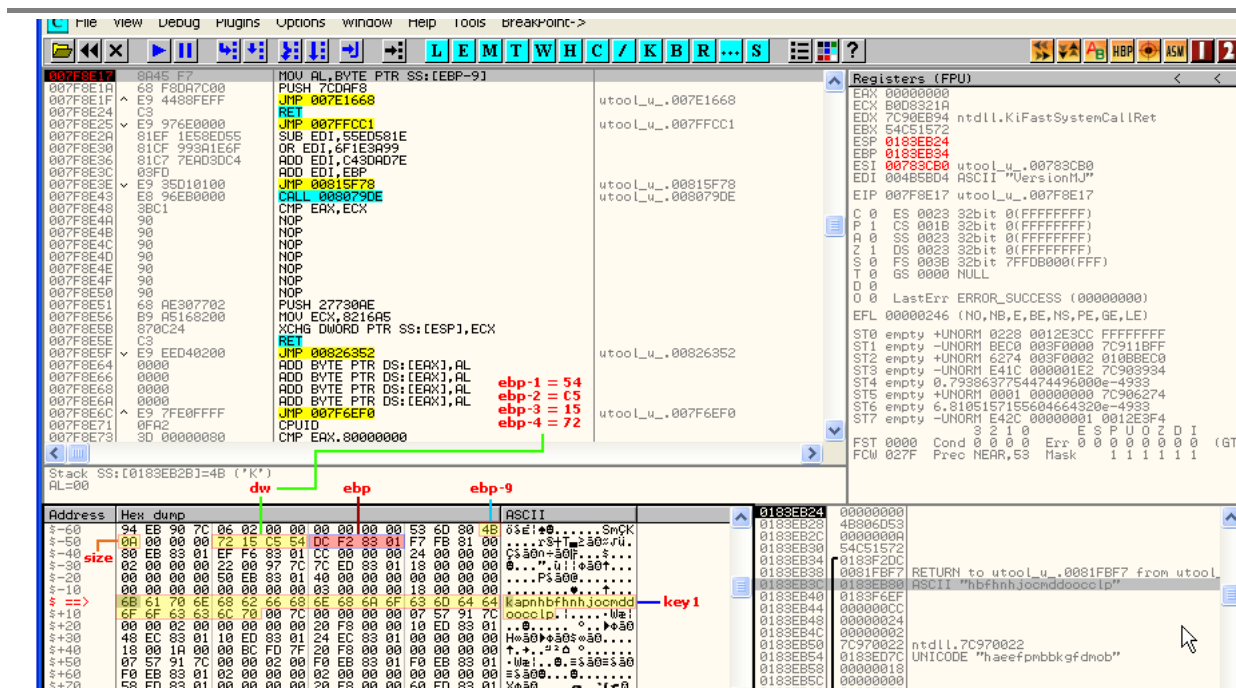
SDK Trial Days

trial days o bien GetLocalTime, y el resultado de aquello fue que exectryptor determina mediante sdk,

todo esto por diversos patrones

vemos un ejemplo Sunbeam comentaba el tema de la clave como la guardaba y como era determinado por un dword

esto fue en un foro: **Subject:** EXECryptor v2.x - DecryptKey Algo **Author:** SunBeam **Time:** 2008-8-10 12:15



Vemos que esta aplicación también usa esta forma de guardar y guiarse:

Esto es moviendo la letra, restando y seguir letra por letra **Sub eax,61**

```
MOVZX EAX, BYTE PTR DS:[EAX]
SUB EAX, 61
PUSH ECX
PUSH FBFB3B85E
JMP CanU_u3.000B3B3E
```

Pero en forma simple descripta de palabra a palabra: **001FF66C** ASCII "jaaloffjjngmdjahlonj"
00229400 ASCII "14.8.2010"
00000000

En forma simple de sdk en forma simple debería ser esto:

EC_GetTrialDaysLeft() llama al sdk EC_GetDate() y forma 2 valores para FirstRun y LastRun , estos valores guardados con EC_SecureRead() y descriptado por EC_DecryptString(), todo esto para decir cuantos dias tenemos o bien algun mensaje.

Si veo stack o eax , en algun momento veremos esto:

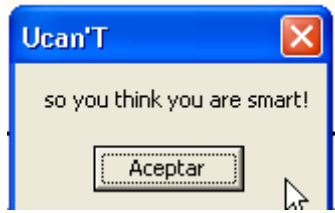
Y usa 2 variables

ASCII "FirstRun"	ASCII "LastRun"
------------------	-----------------

Ahora la parte interesante LOS CASOS:

1)

Si first es >(mayor) al last el resultado sera:



Osea si es valido para 100 ejecuciones y dice ciento diez, me mostrara el cartel, si es valido desde el 1julio hasta el 1 agosto, y ahora es 2 agosto, muestra el mensaje.

2) Si first esta escrito pero last no, pues corra dependiendo si verificara la fecha denuevo, porque el primero, no encuentra nada , pero el segundo necesita saber el periodo.

Por lo tanto CORRE, si no hay valor, Muestra los labels, si hay valor, calcula de forma que maneja junto a ESI y otras variables.

3) Si fist dice una fecha , y last la de hoy, sobre escribira la fecha y puede ejecutar, pero la comprobacion sera de nuevo en el inicio y no lo tolerara 2 veces, en otras palabras, si el programa es valido para 3 dias, y le coloco que sea valido para 4 dias, corra la primera vez para 4 y la segunda cero.

Inline patch para trial days

Metodo nuevo: Ahora el ataque para estos casos:

La gran mayoría de acceso a regedit, siempre se llama usando un push 80000 o similar en las zonas ofuscadas , veamos

```
01000080 PUSH 80000001
E3020000 CALL <JMP.&advapi32.RegCreateKeyExA>
C0 OR EAX,EAX
02 IN2 SHORT word_00401002
```

```
hKey = HKEY_CURRENT_USER
RegCreateKeyExA
```

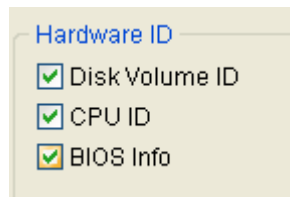
Y otra api suele ser usada &advapi32.RegQueryInfoKeyA> Y veo

```
001FF794 00003CEC CALL to RegQueryInfoKeyA from CanU_u3.00003CE7
001FF798 000000FC hKey = FC
001FF79C 00000000 Class = NULL
001FF7A0 00000000 pClassCount = NULL
001FF7A4 00000000 Reserved = NULL
001FF7A8 001FF7D0 pnSubkeys = 001FF7D0
001FF7AC 001FF7CC pMaxSubkeyLength = 001FF7CC
001FF7B0 00000000 pMaxClassLength = NULL
001FF7B4 00000000 pnValues = NULL
001FF7B8 00000000 pMaxValueNameLength = NULL
001FF7BC 00000000 pMaxValueLength = NULL
001FF7C0 00000000 pSecurity = NULL
001FF7C4 00000000 pLastWrite = NULL
001FF7C8 9F3A21BA
```

Y tambien accede a

```
03639 CALL to RegQueryValueExA
000C8 hKey = C8
29C98 ValueName = "SystemBiosVersion"
00000 Reserved = NULL
7FF90 pValueType = 00D7FF90
2A008 Buffer = 0022A008
7FF94 pBufSize = 00D7FF94
038D5 RETURN to CanU_u3.000038D5
7FFB4
```

Todo esto es parte de lo que ofrece el packer



para identificar la maquina:

Las primeras pruebas fueron buscando lugares

Ahora sigamos con parametros para apis

```
00052BC4 . 8300 CALL CanU_u_r.00086E6E
00052BC6 . E8 A3420300 CALL CanU_u_r.00086E6E
00052BC8 . 8905 8C650500 MOV DWORD PTR DS:[<%USER32.GetKeyboardLayoutList>]
00052BD1 . 8D05 E37D0A00 LEA EAX,DWORD PTR DS:[A7DE3]
00052BD7 . C600 C3 MOV BYTE PTR DS:[EAX],0C3
00052BDA . vE9 32000000 JMP CanU_u_r.00052C11
```

Vemos el uso de xchg para acceder a eax, osea el valor de abajo

00086E5D	C1C0 08	RUL EAX,8	
00086E60	81F8 83420F80	CMP EAX,800F4283	
00086E66	✓E9 E71A0100	JMP CanU_u_r.00098952	
00086E6B	C1C5 05	ROL EBP,5	
00086E6E	870424	XCHG DWORD PTR SS:[ESP],EAX	
00086E71	58	POP EAX	001FFF34
00086E72	81C3 5BBAFEDF	ADD EBX,DFFEBA5B	
00086E78	✓E9 D6690400	JMP CanU_u_r.000CD853	
00086E7D	✓0F84 8B140000	JE CanU_u_r.0008830E	
00086E83	✓E9 E06B0000	JMP CanU_u_r.0008DA68	
00086E88	870C24	XCHG DWORD PTR SS:[ESP],ECX	
00086E8B	59	POP ECX	
00086E8C	F62A	IMUL BYTE PTR DS:[EDX]	
00086E8E	3C A4	CMP AL,0A4	
00086E90	✓0F84 E81CFDFF	JE CanU_u_r.00058B7E	
00086E96	✓E9 3249FEFF	JMP CanU_u_r.0006B7CD	
00086E9B	F7C3 08000000	TEST EBX,8	
00086EA1	✓E9 C425FEFF	JMP CanU_u_r.0006946A	
00086EA6	3B03	CMP EDX,EBX	
00086EA8	✓E9 DF2DFEFF	JMP CanU_u_r.00069C8C	
00086EAD	56	PUSH ESI	
00086EAE	✓E9 EECC0100	JMP CanU_u_r.000A3BA1	
00086EB3	E8 5DB00300	CALL CanU_u_r.000C1F15	
00086EB8	97	XCHG EAX,EDI	
00086EB9	09DE	OR ESI,EBX	
00086EBB	✓77 5E	JA SHORT CanU_u_r.00086F1B	
00086EBD	81F0 76FE6C7F	XOR EAX,7F6CFE76	
00086EC3	03C5	ADD EAX,EBP	

Stack [001FFF1C]=001FFF34 (001FFF34), ASCII "user32.dll"
EAX=00052BCB (CanU_u_r.00052BCB)

Apunta a la direccion de la api, pero parece distractor, porque usa el comparador de apis(comparando los comienzos de las apis).

Luego de recorrer un poco, creo un Injerto, para mi maquina , lo que hace es simple, sacar de un trozo mas recurrente, guardando el flag y no ser detectado , y retornando, para cambiar de la siguiente forma

Accede a rama 1 direccion(aproved\valores) , y lee la rama 2,(aproved\), por ende siempre sera el dia 1

, probe con un trial days , y el resultado fue operativo , pudiendo manejar los dias y las ejecuciones en caso de opciones mas elaboradas y al leer el contenido tenia una estructura nueva asi

@trialdays=1fecha

Trialdays=2fecha

como todas las veces stack es comparado en un punto especial lo anule, y es importante, porque no todas las maquinas son iguales, en mi pc ya lo tengo descubierto , pero

este es el primer avance:

300530B4	> 9C	PUSHFD	Unicode "0003EA19BE\Escritorio\unpack5.exe"
300530B5	. 817C24 10 288	CMP DWORD PTR SS:[ESP+10],228528	importante para el day!!
300530B6	. 817C24 10 308	CMP DWORD PTR SS:[ESP+10],228630	ASCII "SOFTWARE\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved\c"
300530C5	. C605 60FD1F00	MOV BYTE PTR DS:[1FFD60],0	
300530CC	. 90	NOP	ASCII "\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved\c"
300530CD	. 817C24 10 60F	CMP DWORD PTR SS:[ESP+10],1FFE60	
300530DE	. 817C24 10 18F	CMP DWORD PTR SS:[ESP+10],1FF818	ASCII "jaalnfeijocpchmdobg"
300530DD	. 817C24 10 308	CMP DWORD PTR SS:[ESP+10],228630	Unicode "0003EA19BE\Escritorio\unpack5.exe"
300530E5	. 817C24 10 68F	CMP DWORD PTR SS:[ESP+10],1FFE68	
300530ED	. 817C24 10 388	CMP DWORD PTR SS:[ESP+10],228738	
300530F5	. 817C24 10 20F	CMP DWORD PTR SS:[ESP+10],1FFC20	
300530FD	. 817C24 10 288	CMP DWORD PTR SS:[ESP+10],228528	
30053105	. 817C24 10 BCF	CMP DWORD PTR SS:[ESP+10],1FFC8C	
3005310D	. 817C24 10 F88	CMP DWORD PTR SS:[ESP+10],2286F8	
30053115	. 817C24 10 908	CMP DWORD PTR SS:[ESP+10],228790	
3005311D	. 803D A4FE1F00	CMP BYTE PTR DS:[1FFFA4],7B	
30053124	. 0F84 13000000	JE unpack5.00053130	
3005312A	. 803D ACFE1F00	CMP BYTE PTR DS:[1FFFA4],7B	
30053131	. 74 16	JE SHORT unpack5.00053149	
30053133	. D900	FNOP	
30053135	. D900	FNOP	
30053137	. 90	NOP	
30053138	. ^E9 6AFFFFF	JMP unpack5.000530A7	
3005313D	. C605 A5FE1F00	MOV BYTE PTR DS:[1FFFA5],0	
30053144	. ^E9 5EFFFFF	JMP unpack5.000530A7	
30053149	. C605 ADFE1F00	MOV BYTE PTR DS:[1FFFA0],0	
30053150	. ^E9 52FFFFF	JMP unpack5.000530A7	
30053155	. 817C24 10 908	CMP DWORD PTR SS:[ESP+10],228790	
3005315D	. 817C24 10 888	CMP DWORD PTR SS:[ESP+10],228688	Unicode "000226-1229)"
30053165	. 817C24 10 808	CMP DWORD PTR SS:[ESP+10],228580	Unicode "nyName"
3005316D	. 817C24 10 908	CMP DWORD PTR SS:[ESP+10],228790	
30053175	. 817C24 10 A88	CMP DWORD PTR SS:[ESP+10],2288A8	
3005317D	. 817C24 10 E88	CMP DWORD PTR SS:[ESP+10],2284E8	
30053185	. 817C24 10 A88	CMP DWORD PTR SS:[ESP+10],2288A8	Unicode "FO\system32\kernel32.dll"
3005318D	. 9C	PUSHFD	
3005318E	. 803D A4FE1F00	CMP BYTE PTR DS:[1FFFA4],7B	
30053195	. 74 24	JE SHORT unpack5.000531B8	
30053197	. 803D ACFE1F00	CMP BYTE PTR DS:[1FFFA4],7B	
3005319E	. 74 24	JE SHORT unpack5.000531C4	
300531A0	. 803D F8FC1F00	CMP BYTE PTR DS:[1FFCF8],7B	
300531A7	. 74 24	JE SHORT unpack5.000531C0	
300531A9	. D900	FNOP	
300531AB	. D900	FNOP	
300531AD	. D900	FNOP	
300531AF	. 90	POPFD	
300531B0	. 90	NOP	
300531B1	. C8 01000000	PUSH 00000001	
300531B8	. ^E9 4CC0700	JMP unpack5.000CFE07	
300531BB	. C605 A5FE1F00	MOV BYTE PTR DS:[1FFFA5],0	
300531C2	. ^EB DC	JMP SHORT unpack5.000531A0	
300531C4	. C605 ADFE1F00	MOV BYTE PTR DS:[1FFFA0],0	
300531C8	. ^EB D3	JMP SHORT unpack5.000531A0	
300531CD	. C605 F8FC1F00	MOV BYTE PTR DS:[1FFCF8],0	
300531D4	. ^EB CA	JMP SHORT unpack5.000531A0	

Con eso tenemos una transformacion en forma Inline de Regedit, donde accede y donde escribe,

Cambia " { -> 0" en el aproved

Lo que hace el injerto es en forma simple En forma simple accede a rama 1 y escribe en rama 2 , por ende leera solo cierta información ojo, Una cosa es escribir y otra leer, aun no expondre todas las apis, mientras no desofusque todo esos trozos, pero el tema es que jamas expirara.

Y con respecto al contador, el 530c5, es el lugar para colocar esto a cero, pues no escribira el valor y jamás dira que acaso tu no puedes(caso 1).

Busco la estructura en esp+4, **también suele ser útil:Y se injerta con salto, pushfd y comparaciones**

Además agrego otro injerto c1f15 para verificar fuera de mi injerto:

000C1F08	F0:0FC12D F09A0	LOCK XADD DWORD PTR DS:[89AF0],E
000C1F10	. ^E9 52CF0000	JMP CanU_u6.000CEE67
000C1F15	. ^E9 8211F9FF	JMP CanU_u6.0005309C
000C1F1A	. 90	NOP
000C1F1B	. 9C	PUSHFD
000C1F1C	. 51	PUSH ECX
000C1F1D	. 8BC8	MOV ECX,EAX
000C1F1F	. ^E9 2190FAFF	JMP CanU_u6.0006BA45
000C1F24	. 57	PUSH EDI

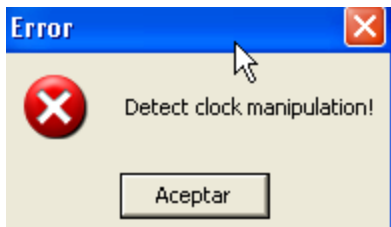
Donde me servirá para inspeccionar en caso x, que quiera ver alguna otra variable que se me escape.

0005309C	. 9C	PUSHFD
0005309D	. 817C24 04 266	CMP DWORD PTR SS:[ESP+4],CanU_u7.00086E26
000530A5	. 74 0A	JE SHORT CanU_u7.000530B1
000530A7	. 90	POPFD
000530A8	. 90	NOP
000530A9	. 90	NOP
000530AA	. ^E9 FB8B0600	JMP CanU_u7.000BB0CA
000530AF	. 90	NOP

Programa inlineado.

Clock Manipulation

Pero Me apareció un detalle mas:



bajo la enseñanza de Kioresk este mensaje es por una estructura conocida por el, sugiere parchar , busco la estructura y lo cambio usando un patron , segun el *escrito de busqueda multiple*, agrego los detalles:

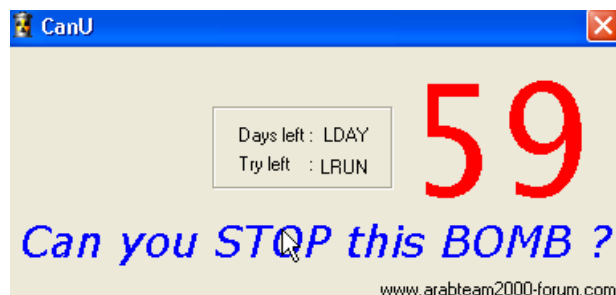
```
mov comentario , "clock_manipulation "  
mov observ, "parchar con C6 45 FF 00"  
findmem #0F9545FF#, code_start  
jmp vuelvedeproc
```

000B194A	C6 45 FF 00	MOV BYTE PTR SS:[EBP-1],0	clock_manipulation
000B194E	E8 5D300000	CALL CanU_u7.000B49B0	
000B1953	^E9 BC65FEFF	JMP CanU_u7.00097F14	
000B1958	9C	PUSHFD	
000B1959	^E9 2882FBFF	JMP CanU_u7.00098B86	
000B195F	81F2 38F22F6D	AND EDI, 00000000	

El resultado osea no tengo dias, y tengo full intentos:



Si actualmente borro la rama de regedit con trial reset Muestra aquel mensaje de clock+(si es que no anulo se ve la nag, si la anulo, se ve solo esto:



Claramente el injerto fue mas alla que lo que pensaba, anulamos la completa lectura de los valores en execryptor.

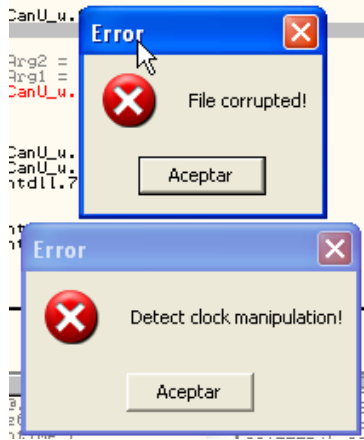
Por ende no lee los dias.y correcto, no hay verificacion de dias. Y el timer ya no funciona.

Verificamos con Trial Reset: Y el programa solo escribe en 2 ramas separadas.

System	Type	Path	Expiry time	Status
EXECryptor	Reg32Key	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved\{}	08/26/2010 18:38:38	Found
EXECryptor	Reg32Key	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved\	08/26/2010 18:38:38	Found

Aha, ese injerto fue improvisado, pero resulta , pues regedit puede usar Unicode o Ascii

O bien usamos los 3 tipos de busqueda para las 2 detecciones



Que implementado realiza una busqueda en el unpacked para encontrar algo y loguearlo

1:

```
mov comentario , "crc1: " //
mov observ, "fix con C6 45 F7 00"
findmem #0F9545FB#, code_start
jmp vuelvedeproc
```

2:

```
mov comentario , "crc2: "
mov observ, "parchar con C6 45 F7 00"
findmem #0F9545F7#, code_start
jmp vuelvedeproc
```

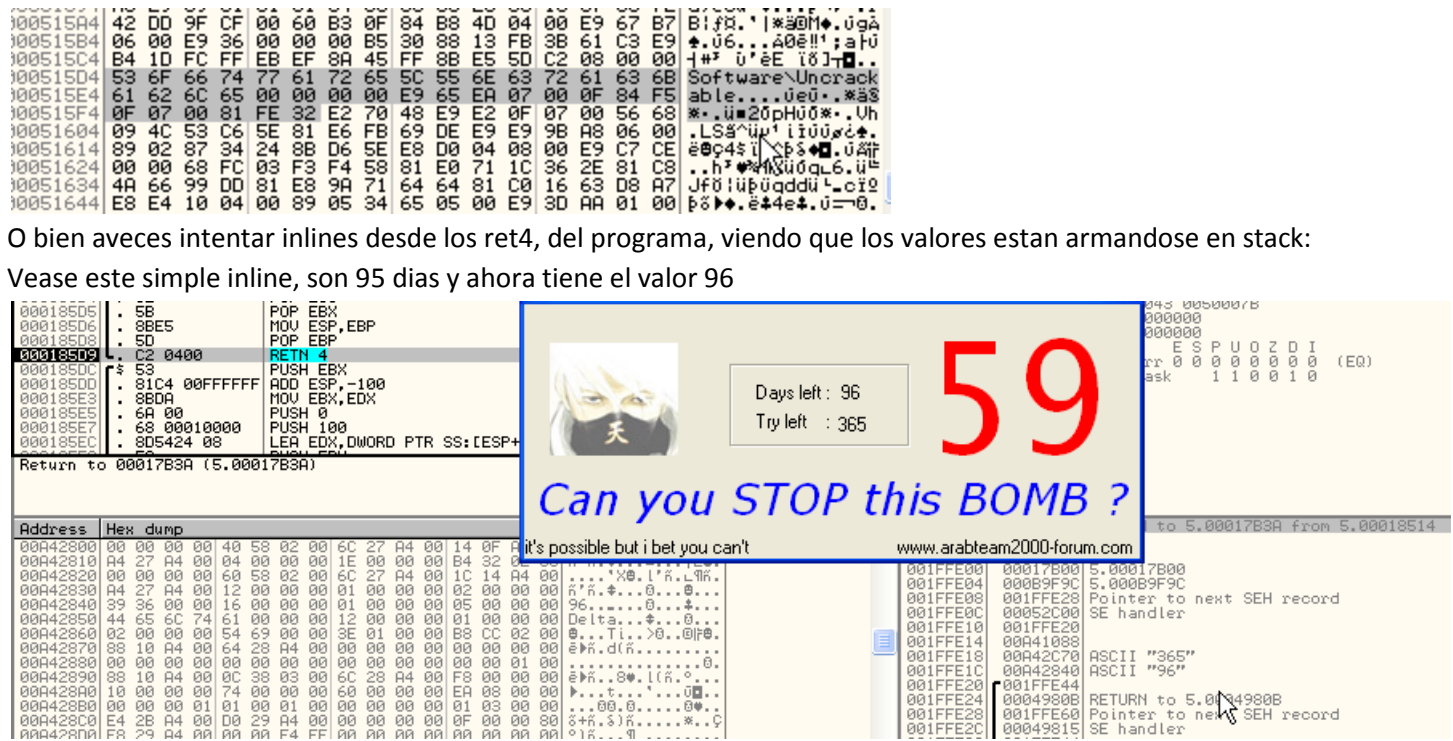
3:

```
mov comentario , "clock_manipulation "
mov observ, "parchar con C6 45 FF 00"
findmem #0F9545FF#, code_start
jmp vuelvedeproc
veremos algo asi
```

```
7eacc 0F9545 FB SETNE BYTE PTR [EBP-5]
encontre crc1: 7EACC fix con C6 45 F7 00
883d2 0F9545 FB SETNE BYTE PTR [EBP-5]
encontre crc1: 883D2 fix con C6 45 F7 00
c300a 0F9545 FB SETNE BYTE PTR [EBP-5]
encontre crc1: C300A fix con C6 45 F7 00
268ed 0F9545 FF SETNE BYTE PTR [EBP-1]
encontre clock_manipulation 268ED parchar con C6 45 FF 00
3a5a3 0F9545 FF SETNE BYTE PTR [EBP-1]
encontre clock_manipulation 3A5A3 parchar con C6 45 FF 00
79ab7 0F9545 FF SETNE BYTE PTR [EBP-1]
encontre clock_manipulation 79AB7 parchar con C6 45 FF 00
b194a 0F9545 FF SETNE BYTE PTR [EBP-1]
encontre clock_manipulation B194A parchar con C6 45 FF 00
```

Otros SDK

Buscando donde comienzan los sdk suelen ser software\execryptor, en canu colocaron:



Igual se puede anular los try left, cuando se anula un seh, creo que es antidebug, pero creo que es un buen comienzo

Proof Concept Trial Limit

Ahora vamos a con el tema de los dias,

Proof Concept, Trial limit

->start->

Proof of concept de trial days

Creo un minimum.exe

```
; #####
.386
.model flat, stdcall
option casemap :none ; case sensitive
; #####

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

; #####
.data
.code
start:

    jmp @F
    szDlgTitle    db "EC for CLS",0
    szMsg         db "Proof Concept trial days",0
@@:

push MB_OK
push offset szDlgTitle
push offset szMsg
push 0
call MessageBox
push 0
call ExitProcess

; -----
; The following are the same function calls using MASM
; "invoke" syntax. It is clearer code, it is type checked
; against a function prototype and it is less error prone.
; -----

; invoke MessageBox,0,ADDR szMsg,ADDR szDlgTitle,MB_OK
; invoke ExitProcess,0

end start
```

Archivo Edicion Formato ver Ayuda

Recho off

```
if exist minimum.obj del minimum.obj
if exist minimum.exe del minimum.exe

\masm32\bin\ml /c /coff /nologo minimum.asm
\masm32\bin\Link /SUBSYSTEM:WINDOWS /MERGE:.rdata=.text minimum.obj > nul

dir minimum.*

pause
```

Ahora lo empaco

Virtualization level: 0%

☐ Anti-debug

- ☐ Aggressive mode
- ☐ Kill Registry/File monitors
- ☐ Debug messages
- ☐ Active watch

☐ Patch protection

☐ Anti-trace

☒ Protect entry point

Compression

- ☐ Compress resources
- ☐ Compress code & data
- ☐ Use maximum compression
- ☐ Strip relocation info
- ☐ Keep overlay data

☐ Dynamic Import

☒ Delay DLL loading

☒ VMWare/VirtualPC/wine compatible

Configuration: One-touch trial

Product name: prueba_2.2.5 %D-day, %R-runs

Order URL: http://www.mysite.com/order.html

Limitation Messages

☒ Trial period 2 days

☒ Executions 5 runs

☐ Allow lock serial to hardware

General settings Protection options Serial number support **Advanced**

Hardware ID

- ☐ Disk Volume ID
- ☐ CPU ID
- ☒ BIOS Info

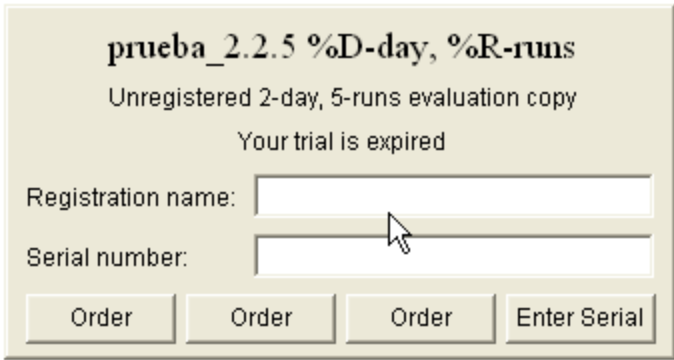
On detect clock manipulation

- ☒ show message and exit
- ☐ return big date
- ☐ return date: 16-01-2005
- ☐ nothing

Reset trial on change

- ☐ Major version
- ☐ Minor version
- ☐ Release
- ☐ Custom version 0

EXECryptor 2.4.1
Copyright (c) 2002-2006 SoftComplete Development
Registered to: me not (Professional Single license)
Source: Copia (2) de minimum1.exe
Found 1 marks.
Relocation info not found.
Randomize: 4280078436, 1868234802, 1322036807, 1431687558
Entry point at 0x0000102E: Size 32 bytes; 8 instructions; 0 jcc.
Detect EXECryptor API usage:
Process import ...
Compile EXECryptor API library 124963 bytes; 35468 instructions; 2564 jcc.
Local transform: 74468 bytes; 14 instructions; 0 jcc.
Global transform: 74474 bytes; 16 instructions; 0 jcc.
Compile VM core 10149 bytes; 3328 instructions; 267 jcc.
Total: 209586 bytes; 38812 instructions; 2831 jcc.
Virtualization: 30943 instructions
Total: 282115 bytes; 50125 instructions; 1298 jcc.
Create extra code section at 0x00002000
Assemble
Create TLS directory
Extra code section size: 413642 --> 282419 bytes
Flush relocation info.
Save
Done.
Pruempo 1, 2, 3 , 4 , 5 y expira



Borro gracias a que execryptor puede tambien anular su propia proteccion, lo borro

Busco todas las variantes eax+4

Cmp dword ptr ds: "

Address	Disassembly
0040C9E1	CMP DWORD PTR DS:[EAX+4],0
00436C8E	CMP DWORD PTR DS:[EAX+4],0
0043E481	CMP DWORD PTR DS:[EAX+4],0
00446595	MOV WORD PTR DS:[EDX],AX

Cuando no esta con los dias

0043E47E	8B45 E8	MOV EAX,DWORD PTR SS:[EBP-18]
0043E481	8378 04 00	CMP DWORD PTR DS:[EAX+4],0
0043E485	E8 5629FFFF	CALL EC_2dias.00430DE0
0043E48A	FF80 391A0704	INC DWORD PTR DS:[EAX+4071A39]
0043E490	3016	XOR BYTE PTR DS:[ESI],DL
0043E492	E8 2E08FFFF	CALL EC_2dias.0042ECC5
0043E497	FA	BUSH EBX

0043E47E 8B45 E8 MOV EAX,DWORD PTR SS:[EBP-18]
 0043E481 8378 04 00 CMP DWORD PTR DS:[EAX+4],0
 0043E485 E8 5629FFFF CALL EC_2dias.00430DE0
 Stack DS:[0012F6C0]=00000000

Segunda vez

0043E47E 8B45 E8 MOV EAX,DWORD PTR SS:[EBP-18]
 0043E481 8378 04 00 CMP DWORD PTR DS:[EAX+4],0
 0043E485 E8 5629FFFF CALL EC_2dias.00430DE0
 Stack DS:[0012F6C0]=00000000

Tercera

0043E47E 8B45 E8 MOV EAX,DWORD PTR SS:[EBP-18]
 0043E481 8378 04 00 CMP DWORD PTR DS:[EAX+4],0
 0043E485 E8 5629FFFF CALL EC_2dias.00430DE0
 0043E48A FF80 391A0704 INC DWORD PTR DS:[EAX+4071A39]
 0043E490 3016 XOR BYTE PTR DS:[ESI],DL
 0043E492 E8 2E08FFFF CALL EC_2dias.0042ECC5
 0043E497 53 PUSH EBX
 0043E498 2245 01 AND AL,BYTE PTR SS:[EBP+1]
 0043E49B 07 POP ES
 0043E49C 04 50 ADD AL,50
 0043E49E EB E9 JMP SHORT EC_2dias.0043E489
 0043E4A0 25 54FDF87 AND EAX,87FFFD54
 0043E4A5 04 24 ADD AL,24
 0043E4A7 58 POP EAX
 0043E4A8 68 A686A644 PUSH 44A686A6
 0043E4AD 5A POP EDX
 0043E4B5 81C0 F323DC22 OR EDI,23DC2322
 Stack DS:[0012F694]=00000000

Address	Hex dump	ASCII
00402000	68 C0 6C 41 00 E9 A6 03 04 00 E8 F1 1C 04 00 22	h^lA

Cuarta

0043E47E 8B45 E8 MOV EAX,DWORD PTR SS:[EBP-18]
 0043E481 8378 04 00 CMP DWORD PTR DS:[EAX+4],0
 0043E485 E8 5629FFFF CALL EC_2dias.00430DE0
 0043E48A FF80 391A0704 INC DWORD PTR DS:[EAX+4071A39]
 0043E490 3016 XOR BYTE PTR DS:[ESI],DL
 0043E492 E8 2E08FFFF CALL EC_2dias.0042ECC5
 0043E497 53 PUSH EBX
 0043E498 2245 01 AND AL,BYTE PTR SS:[EBP+1]
 0043E49B 07 POP ES
 0043E49C 04 50 ADD AL,50
 0043E49E EB E9 JMP SHORT EC_2dias.0043E489
 0043E4A0 25 54FDF87 AND EAX,87FFFD54
 0043E4A5 04 24 ADD AL,24
 0043E4A7 58 POP EAX
 0043E4A8 68 A686A644 PUSH 44A686A6
 0043E4AD 5A POP EDX
 0043E4B5 81C0 F323DC22 OR EDI,23DC2322
 Stack DS:[0012F694]=00000000

Quinta

0043E47E 8B45 E8 MOV EAX,DWORD PTR SS:[EBP-18]
 0043E481 8378 04 00 CMP DWORD PTR DS:[EAX+4],0
 0043E485 E8 5629FFFF CALL EC_2dias.00430DE0
 0043E48A FF80 391A0704 INC DWORD PTR DS:[EAX+4071A39]
 0043E490 3016 XOR BYTE PTR DS:[ESI],DL
 0043E492 E8 2E08FFFF CALL EC_2dias.0042ECC5
 0043E497 53 PUSH EBX
 Stack DS:[0012F6A4]=00000000

Address	Hex dump	ASCII
00402000	20 C0 2F 41 00 E9 A6 03 04 00 E8 F1 1C 04 00 22	h^lA

Y corre

prueba_2.2.5 %D-day, %R-runs
 Unregistered 2-day, 5-runs evaluation copy
 You have 2 days and 5 runs left

Registration name:

Serial number:

02:06
Sábado
23-08-2003

Vemos el registry hoy lo tengo con

Veamos que construyo:

Name	Value
FirstRun....0	23.8.2003
LastRun....0	23.8.2003
RunCount....0	4

Ahora ejecuto el mismo, pero vemos una gran diferencia, en el tercero muestra un 22

0043E476	871424	XCHG DWORD PTR SS:[ESP],EDX
0043E479	E9 AA9DFFFF	JMP EC_2dias.00438228
0043E47E	8B45 E8	MOV EAX,DWORD PTR SS:[EBP-18]
0043E481	8378 04 00	CMP DWORD PTR DS:[EAX+4],0
0043E485	E8 5629FFFF	CALL EC_2dias.00430DE0
0043E48A	FF80 391A0704	INC DWORD PTR DS:[EAX+4071A39]
0043E493	2B1C	MOV BYTE PTR DS:[ESI],DL
Stack DS:[0012F694]=00000022		

El cuarto un 22

0043E47E	8B45 E8	MOV EAX,DWORD PTR SS:[EBP-18]
0043E481	8378 04 00	CMP DWORD PTR DS:[EAX+4],0
0043E485	E8 5629FFFF	CALL EC_2dias.00430DE0
0043E48A	FF80 391A0704	INC DWORD PTR DS:[EAX+4071A39]
0043E493	2B1C	MOV BYTE PTR DS:[ESI],DL
Stack DS:[0012F694]=00000022		

El quinto un 12

0043E47E	8B45 E8	MOV EAX,DWORD PTR SS:[EBP-18]
0043E481	8378 04 00	CMP DWORD PTR DS:[EAX+4],0
0043E485	E8 5629FFFF	CALL EC_2dias.00430DE0
0043E48A	FF80 391A0704	INC DWORD PTR DS:[EAX+4071A39]
0043E493	2B1C	MOV BYTE PTR DS:[ESI],DL
Stack DS:[0012F6A4]=00000012		

prueba_2.2.5 %D-day, %R-runs

Unregistered 2-day, 5-runs evaluation copy

You have 2 days and 4 runs left

Registration name:

Serial number:

Evaluate Order Order Enter Serial

Y realize la prueba, de cambiar el byte siempre a 2

Ahora la comparacion la cambio a 2 por ejemplo

0043E481	C640 04 02	MOV BYTE PTR DS:[EAX+4],2
0043E485	E8 5629FFFF	CALL EC_2dias.00430DE0

Me explico, si el largo que lee es de 12 o 20 , si lee 2, no creo que tome en cuenta los dias..del todo cambio solo el dword, a byte y inserto como mov.

0043E481	C640 04 02	MOV BYTE PTR DS:[EAX+4],2
0043E485	E8 5629FFFF	CALL EC_2dias.00430DE0

El resultado es que esto no expira:

Name	Value
FirstRun....0	23.8.2003
LastRun....0	23.8.2003
RunCount...0	4
RegistrationName	
SerialNumber	

original packed->Cambio la fecha a la de hoy y muestra que esta expirado ,
el que cambie a mov 2-> Es que ejecuta y dice que tenemos la misma cantidad de dias

Y lo que lee execryptor gracias a ese mini parche es la nueva fecha y actualizo

Name	Value
FirstRun....0	7.9.2010
LastRun....0	7.9.2010
RunCount...0	4
RegistrationName	
SerialNumber	

Y edito a que sea cero RUNCOUNT, a modo que este expirado
y este injertado lo vuelve a 4 denuevo..

Name	Value
FirstRun....0	7.9.2010
LastRun....0	7.9.2010
RunCount...0	4
RegistrationName	
SerialNumber	

Por ende cambiando la variable runcount, a mayor a cero, podemos evitar que CANU, mueste ese mensaje,
Estas variables las busco con `eax+4` desde `ebp-18` resulta

Ahora Experimento numero 2, empero otro mas, con otras opciones de dias

Antes del injerto

00402633	50	PUSH EAX
00402634	E8 E33F0400	CALL execry_.0044661C
00402639	8B45 E8	MOV EAX,DWORD PTR SS:[EBP-18]
0040263C	8378 04 00	CMP DWORD PTR DS:[EAX+4],0
00402640	E8 017D0400	CALL execry_.0044A346
00402645	42	INC EDX

, como ya se injertar

00402633	50	PUSH EAX	
00402634	E8 E33F0400	CALL execry_.0044661C	
00402639	E9 49130500	JMP execry_.00453987	salto al injerto
0040263E	90	NOP	
0040263F	90	NOP	
00402640	E8 017D0400	CALL execry_.0044A346	
00402645	42	INC EDX	

00453987 8B45 E8 MOV EAX,DWORD PTR [EBP-18] ->instruccion original

0045398A 8378 04 00 CMP DWORD PTR [EAX+4],0 ->instruccion original

0045398E 74 07 JE SHORT 00453997 ; execry_.00453997

00453990 C740 04 02000000>MOV DWORD PTR [EAX+4],2 ->injerto ;)

00453997 ^ E9 A4ECFAFF JMP 00402640 ->salta despues de las instrucciones iniciales

Y cuando no es cero, que mueva 2,

00453987	8B45 E8	MOV EAX,DWORD PTR SS:[EBP-18]
0045398A	8378 04 00	CMP DWORD PTR DS:[EAX+4],0
0045398E	74 07	JE SHORT execry_.00453997
00453990	C740 04 02000000	MOV DWORD PTR DS:[EAX+4],2
00453997	E9 A4EFAFF	JMP execry_.00402640

Y tambien pasa lo mismo, la aplicacion corre como si tuviera todos los dias.

Por ende de esa forma IDEALMENTE debemos parchar execryptor con Limitacion de Dias cambiando la lectura de los bytes, que en forma hardcoded, lee encripta y desencripta , a que lea solo 1 letra y asi no expira, siempre y cuando nos hayan dado dias triales

Tenemos x dias y x ejecuciones

Donde pueden verse eso?

Veamos otro ejemplo

Ejecuto y busco los cmp dword prt ebp+8 con cero y veo

2 dias:

837d0800

0043CDB3	52	PUSH EDX	
0043CDB4	891C24	MOV DWORD PTR SS:[ESP],EBX	
0043CDB7	33C0	XOR EAX,EAX	
0043CDB9	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
0043CDBC	837D 08 00	CMP DWORD PTR SS:[EBP+8],0	
0043CDC0	E9 5C330000	JMP EC_2dias.00440121	
0043CDC5	81C0 C7454197	ADD EAX,974145C7	
0043CDC8	E8 CD35FDFF	CALL EC_2dias.00410390	
0043CDD0	124E 41	ADC CL,BYTE PTR DS:[ESI+41]	
0043CDD3	D7	XLAT BYTE PTR DS:[EBX+AL]	
0043CDD4	07	POP ES	Modifica
0043CDD5	04 B0	ADD AL,0B0	
0043CDD7	3233	XOR DH,BYTE PTR DS:[EBX]	
0043CDD9	D2E8	SHR AL,CL	
0043CDDB	36:6BFC FF	IMUL EDI,ESP,-1	Superfluo
0043CDDF	68 D0E54200	PUSH EC_2dias.0042E5D0	
0043CDE4	E9 3EC2FCFF	JMP EC_2dias.00409027	
0043CDE9	C3	RETN	
0043CDEA	E9 FAB2FDFF	JMP EC_2dias.004180E9	
0043CDEF	C3	RETN	
0043CDF0	E9 FBFBFEFF	JMP EC_2dias.0042C9F0	
0043CDF5	E9 FD78FDFF	JMP EC_2dias.004146F7	
0043CDFA	81C8 FCBA8730	OR EAX,3087BAFC	
0043CE00	0F89 83D2FFFF	JNS EC_2dias.0043A089	
0043CE06	E9 5D8AFDFF	JMP EC_2dias.00415868	
0043CE0B	E9 3B000000	JMP EC_2dias.0043CE4B	
0043CE10	0F8B 6AE5FCFF	JPO EC_2dias.0040B380	
0043CE16	E9 AB74FDFF	JMP EC_2dias.004142C6	

Stack SS:[0012FE60]=00000002

004436D5	837D 08 00	CMP DWORD PTR SS:[EBP+8],0	
004436D9	E8 BFCCFCFF	CALL EC_2dias.00410390	
004436DE	66:BB 31AE	MOV BX,0AE31	
004436E2	05 042085E8	ADD EAX,E8852004	
004436E7	F1	INT1	
00443733	E9 1C27FCFF	JMP EC_2	

Stack SS:[0012FE60]=00000002

5 ejecuciones

00443340	E9 5B5DFCFF	JMP EC_2dias.004090A0	
00443352	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
00443355	837D 08 00	CMP DWORD PTR SS:[EBP+8],0	
00443359	E8 A8EAFFFF	CALL EC_2dias.00441E06	

Stack SS:[0012FE60]=00000005

00438D5F	^E9 7DF2FFFF	JMP EC_2dias.00437FE1
00438D64	^E9 B597DFFF	JMP EC_2dias.0041251E
00438D69	837D 08 00	CMP DWORD PTR SS:[EBP+8],0
00438D6D	E8 FED3FFFF	CALL EC_2dias.00436170
00438D72	EF	OUT DX,EAX
00438D73	8DC0	LEA EAX,EAX
00438D75	F3:	PREFIX REP:
00438D76	0204F0	ADD AL,BYTE PTR DS:[EAX+ESI*8]
00438D79	^74 E8	JE SHORT EC_2dias.00438D63
00438D7B	46	INC ESI

Stack SS:[0012FE60]=00000005

You have 2 days and 5 runs left

on name:

Ahora colocare denuevo

0043CDEF	C3	RETN
0043CDF0	^E9 FBFBEFF	JMP EC_2dias.0042C9F0
0043CDF5	^E9 FD78DFFF	JMP EC_2dias.004146F7
0043CDFA	81C8 FCBA8730	OR EAX,3087BAFC
0043CE00	^0F89 83D2FFFF	JNS EC_2dias.0043A089
0043CE06	^E9 5D8AFDFF	JMP EC_2dias.00415868
0043CE0B	^E9 3B000000	JMP EC_2dias.0043CE4B
0043CE10	0F8B 6AE5FCFF	JPD EC_2dias.0040B380
0043CE16	^E9 AB74DFFF	JMP EC_2dias.004142C6
0043CE1B	53	PUSH EBX
0043CE1C	68 65D5E64E	PUSH 4EE6D565
0043CE21	5B	POP EBX
0043CE22	81C3 4BAD8C4E	ADD EBX,4E8CAD4B
0043CE28	03D3	ADD EDX,EBX
0043CE2A	5B	POP EBX

Hexadecimal 00000045
Signed
Unsigned

OK Cancel

Stack SS:[0012FE60]=00000045

Address	Hex dump	ASCII
---------	----------	-------

Y la otra a quinientos

00443376	F700	MOV EAX
00443378	81D8 C9678359	SBB EAX,598367C9
0044337E	^E9 D146DFFF	JMP EC_2dias.00417A54
00443383	0F8A ABF5FBFF	JPE EC_2dias.00402934
00443389	890424	MOV DWORD PTR SS:[ESP],EAX
0044338C	58	POP EAX
0044338D	68 8F1390F5	PUSH F590138F
00443392	58	POP EAX
00443393	2B05 094E4200	SUB EAX,DWORD PTR DS:[424FA9]
00443399	81C8 E2A23928	OR EAX,2839A2E2
0044339F	^E9 6C59DFFF	JMP EC_2dias.00418D10
004433A4	F7C7 03785A0A	TEST EDI,0A5A7803
004433AA	^E9 D239FCFF	JMP EC_2dias.00406D81
004433AF	0F85 B1A7FCFF	JNZ EC_2dias.00400B66
004433B5	0F81 8ADCFFFF	JNO EC_2dias.00441045
004433BB	^E9 76BAFFFF	JMP EC_2dias.0043EE36
004433C0	873C24	XCHG DWORD PTR SS:[ESP],EDI

Hexadecimal 000001F4
Signed
Unsigned

OK Cancel

Stack SS:[0012FE60]=00000005

Veamos

prueba_2.2.5 %D-day, %R-runs

Unregistered 2-day, 5-runs evaluation copy

You have 69 days and 500 runs left


stration name:

Si verifico el exe esta expirado y solo se desbloquea con clave, pues ni modo, era un proof->end

Aplicacion del proof Concept

Days left: **CMP DWORD PTR [EBP+8],0**

Days left : 1610612
Try left : 365


STOP this B
an't www.arable

II "161061273"
URN to 64_close.000980B

000B87EE	: 93	XCHG EAX,EBX
000B87EF	: 8945 FC	MOV DWORD PTR [EBP-4],EAX
000B87F2	: 837D 08 00	CMP DWORD PTR [EBP+8],0
000B87F6	: ^ E9 5B2DFFFF	JMP 0009B556

Ejemplo por 9898

000B87F2	: 837D 08 00	CMP DWORD PTR [EBP+8],0	
000B87F6	: ^ E9 5B2DFFFF	JMP 0009B556	64_close.0009B556
000B87FB	: > B8 4CB96C0D	MOV EAX,0D6CB94C	
000B8800	: 56	PUSH ESI	
000B8801	: 68 A790D11B	PUSH 1B0190A7	
000B8806	: 5E	POP ESI	009A1088
000B8807	: ^ E9 3DDFFBFF	JMP 00076749	00076749
000B880C	: > 0F85 DFB5FEF	JNZ 000A8DF1	000A8DF1
000B8812	: ^ E9 80FEFDFF	JMP 00098697	00098697
000B8817	: > 68 8098E4B1	PUSH B1E49880	
000B881C	: 5B	POP EBX	
000B881D	: 81E3 C1E8419	AND EBX,9141E8C1	
000B8823	: 81C3 14C8CB6	ADD EBX,6ECBC814	
000B8829	: 871C24	XCHG DWORD PTR [ESP],EBX	
000B882C	: ^ E9 FF950000	JMP 000C1E30	
000B8831	: E8	DB E8	
000B8832	: 12	DB 12	
000B8833	: 00	DB 00	
000B8834	: 00	DB 00	
000B8835	: 00	DB 00	
000B8836	: 90	NOP	
000B8837	: 25	DB 25	
000B8838	: 7F	DB 7F	
000B8839	: AB	DB AB	
000B883A	: 00	DB 00	

Stack SS:[001FFE04]=00000060

Modify dword at 001FF...

Hexadecimal 000026AA

Signed 9898

Unsigned 9898

OK Cancel

Days left : 9898
Try left : 365

Y tenemos

Try left:

0007BD56 8902 MOV DWORD PTR [EDX],EAX

0007BD58 5A POP EDX

0007BD59 837D 08 00 **CMP DWORD PTR [EBP+8],0**

Ahora bien la otra variable se ve con

0007BD59	: 837D 08 00	CMP DWORD PTR [EBP+8],0	
0007BD5D	: ^ E9 A7B00300	JMP 000B6E09	64
0007BD62	: 81F2 9EAF7E6	XOR EDX,EDX	64
0007BD68	: ^ 0F83 04C40300	JNB 000B6E09	64
0007BD6E	: 9C	PUSHFD	64
0007BD6F	: C1C7 10	ROL EDI,5	64
0007BD72	: 81CF 58A9765F	OR EDI,5	64
0007BD78	: 81C7 27BE0800	ADD EDI,27BE0800	64
0007BD7E	: 873C24	XCHG DWORD PTR [EDX],EDI	64
0007BD81	: ^ E9 33760300	JMP 000B6E09	64
0007BD86	: 68 8F330894	PUSH 9408338F	64
0007BD8B	: ^ E9 42C9FFFF	JMP 00076749	64
0007BD90	: ^ 0F88 40E60300	JS 000B6E09	64
0007BD96	: 81D9 94D97868	SBB ECX,EDX	64
0007BD9C	: ^ E9 2AE60300	JMP 000B6E09	64
0007BDA1	: 8945 FC	MOV DWORD PTR [EDX],ECX	64
0007BDA4	: E8 B1C5FFFF	CALL 00076749	64

Modify dword at 001FF...

Hexadecimal 00000160

Signed 365

Unsigned 365

OK Cancel

Luego de terminar el injerto veo que el valor que lee mi maquina es siempre igual, luego de verificar algunos seh_exit

Ahora la estructura extraña

MOV AX,CS

XOR AL,AL

OR EAX,EAX

00099077	09DA	OR EDX,EBX	
00099079	0158 81	RCR DWORD PTR DS:[EAX-7F],1	
0009907C	F0:CC	LOCK INT3	
0009907E	DB6B D5	FLD TBYTE PTR DS:[EBX-2B]	
00099081	81C0 114494F8	ADD EAX,F8944411	
00099087	81E0 7EDB0E51	AND EAX,510EDB7E	
0009908D	C1C0 00	ROL EAX,00	
00099090	^E9 B020FEFF	JMP unpack5.0007B145	
00099095	5F	POP EDI	
00099096	2BEF	SUB EBP,EDI	
00099098	^E9 719AFFFF	JMP unpack5.00092B0E	
0009909D	C3	RETN	
0009909E	8CC8	MOV AX,CS	
000990A0	30C0	XOR AL,AL	
000990A2	09C0	OR EAX,EAX	
000990A4	^0F85 65010200	JNZ unpack5.000B920F	
000990AA	^E9 E9C00100	JMP unpack5.000B5198	
000990AF	C1CA 04	ROR EDX,4	

Registers (FPU)
001FFF34 ASCII "kernel32.dll"
00087694 unpack5.00087694
001FFF34 ASCII "kernel32.dll"
7FFD7000
001FFF20
001FFF28
FFFFFFFF
7C920228 ntdll.7C920228
0009909E unpack5.0009909E
ES 0023 32bit 0(FFFFFFFF)
CS 001B 32bit 0(FFFFFFFF)
SS 0023 32bit 0(FFFFFFFF)
DS 0023 32bit 0(FFFFFFFF)
FS 003B 32bit 7FFDF000(FFF)
GS 0000 NULL

PopfD

0009C18E	C1E8 02	SHR EAX,2	
0009C191	C680 FC9A0800 0	MOV BYTE PTR DS:[EAX+89AFC],0	
0009C198	870C24	XCHG DWORD PTR SS:[ESP],ECX	
0009C19B	8BC1	MOV EAX,ECX	
0009C19D	59	POP ECX	
0009C19E	9D	POPF0	
0009C19F	C3	RETN	
0009C1A0	^E9 44ADF0FF	JMP unpack5.00076EE9	
0009C1A5	^E9 8365FFFF	JMP unpack5.0009272D	
0009C1AA	^E9 811F0100	JMP unpack5.000AE130	
0009C1AF	81E8 3DA2D541	SUB EAX,41D5A23D	
0009C1B5	81C8 96D9B03C	OR EAX,3CB0D996	
0009C1BB	81F0 C698D82D	XOR EAX,2D0898C6	
0009C1C1	33D0	XOR EDX,EAX	
0009C1C3	5A	POP EAX	

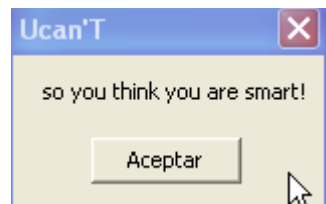
Luego veo un valor que veia antes de llamar a reg... y anule el valor de value name:

En mi pc siempre cae en 1ffd60, ese valor cambiandolo que lea cero..estamos como el injerto completo y no hay expired

001FF7B4	77DA7BA8	advapi32.77DA7BA8	
001FF7B8	8ABA800F		
001FF7BC	000B8C3E	unpack5.000B8C3E	
001FF7C0	77DA7A9B	advapi32.RegQueryValueExA	
001FF7C4	0008BD2C	CALL to RegQueryValueExA	
001FF7C8	000000F0	hKey = F0	
001FF7CC	001FFD60	ValueName = ""	
001FF7D0	00000000	Reserved = NULL	
001FF7D4	001FFF64	pValueType = 001FFF64	
001FF7D8	00000000	Buffer = NULL	
001FF7DC	001FFF68	pBufSize = 001FFF68	
001FF7E0	001FF7FC		
001FF7E4	001FF7FC		

O bien

Aplicacion ante fin de ejecucion;



Accedo a RegQueryValueExA

Cuando tengo:

4	0008BD2C	CALL to RegQueryValueExA	
8	000000C4	hKey = C4	
C	001FFD60	ValueName = "iaeeidjjeofndblkac"	
0	00000000	Reserved = NULL	
4	001FFF64	pValueType = 001FFF64	
8	00000000	Buffer = NULL	
C	001FFF68	pBufSize = 001FFF68	
A	001FF7FC		

Retrocede aqui:

0008B02C	E8 78370400	CALL 000CF4A9
0008B031	65:E8 0962E977	CALL 77F21F40
0008B037	6A01 00	TMU F0V DWORD PTR F0

8bd31

Y se me ocurre anular el hkey de stalk y de eax

000AE2A5	BD 01000000	MOV EBP,1	
000AE2AA	9C	PUSHFD	
000AE2AB	3B4424 0C	CMP EAX,DWORD PTR [ESP+C]	veng_esl.0001
000AE2AF	75 0A	JNZ SHORT 000AE2B8	veng_esl.000F
000AE2B1	33C0	XOR EAX,EAX	
000AE2B3	C74424 0C 0000	MOV DWORD PTR [ESP+C],0	
000AE2BB	9D	POPFD	
000AE2BC	90	NOF	
000AE2BD	BD 01000000	MOV EBP,1	
000AE2C2	90	NOF	
000AE2C3	E9 795BFEFF	JMP 00093E41	veng_esl.0005
000AE2C8	0000	MOV EAX,DWORD PTR [EBP+1	

Y luego evito la nag con un injerto mas largo , pero como bien comentaba que fue una hazaña, creo que es mas que suficiente comenzar a comentar que se puede inlinear en accesos a regedit, siempre y cuando se use trial reset de apoyo.

Otras curiosidades

Creo que ha sido suficiente para comenzar con canu y conforme al nivel desarrollado encontraremos mas detalles

Aveses se ven apis peligrosas, pero no le daremos importancia

0028FDE0	00000000	
0028FDF0	00010001	ASCII "ZP"
0028FDF4	00000000	
0028FDF8	0028F5B0	
0028FDFC	000000A0	
0028FE00	000006B4	
0028FE04	000006A8	
0028FE08	7E3AF383	user32.SendMessageA
0028FE0C	7C8021CC	kernel32.ReadProcessMemory
0028FE10	7C859F08	kernel32.OutputDebugStringA
0028FE14	FFFF4FF0	

Pero buscando mas variables encontré la ultima Estaba buscando a createThread y encuentre que hay una variable ebp-c en eax, mas o menos comun

Variable ebp-c en eax

veamos

000AF16E	5A	POP EDX	
000AF16F	5A	POP EDX	
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	kernel32.CreateThread
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
000AF176	57	PUSH EDI	
000AF177	68 F3FF410E	PUSH 0E41FFF3	
000AF17C	E9 004BFDF	JMP unpack5.00083C81	
000AF181	81CB C758A0F4	OR EBX,F4A058C7	
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0	
000AF18D	C1C6 1D	ROL ESI,1D	

000AF149	C1C8 05	ROR EAX,5	
000AF14C	E9 58AEFEFF	JMP unpack5.00099FAC	
000AF151	68 260C2418	PUSH 18240C26	
000AF156	58	POP EAX	
000AF157	81C0 E8E0FA2	ADD EAX,A2AFE0E8	
000AF15D	81E0 A1C5A94D	AND EAX,4DA9C5A1	
000AF163	C1C0 12	ROL EAX,12	
000AF166	E9 F0AEFBFF	JMP unpack5.0006A068	
000AF16B	871424	XCHG DWORD PTR SS:[ESP],EDX	
000AF16E	5A	POP EDX	
000AF16F	5A	POP EDX	
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	kernel32.
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
000AF176	57	PUSH EDI	
000AF177	68 F3FF410E	PUSH 0E41FFF3	
000AF17C	E9 004BFDF	JMP unpack5.00083C81	
000AF181	81CB C758A0F4	OR EBX,F4A058C7	
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0	
000AF18D	C1C6 1D	ROL ESI,1D	

EAX	7C802520	kernel32.WaitForSingleObject
ECX	7C810613	kernel32.7C810613
EDX	7C802644	kernel32.7C802644
EBX	7FFD7000	
ESP	001FFE30	
EBP	001FF533	
ESI	FFFFFFFF	
EDI	7C920228	ntdll.7C920228
EIP	000AF170	unpack5.000AF170
C 0	ES 0023	32bit 0(FFFFFFFF)
P 0	CS 001B	32bit 0(FFFFFFFF)
A 0	SS 0023	32bit 0(FFFFFFFF)
Z 0	DS 0023	32bit 0(FFFFFFFF)
S 0	FS 003B	32bit 7FFD7000(FFF)
T 0	GS 0000	NULL
D 0		
O 0	LastErr	ERROR_INVALID_HANDLE (00000006)

000AF147	8BCF	MOV ECX,EDI		Registers (MMX)
000AF149	C1C8 05	ROR EAX,5		EAX 7C80FD3D kernel32.GlobalAlloc
000AF14C	^E9 5BAFEFEF	JMP unpack5.00099FAC		ECX 001FFFA4
000AF151	68 260C2418	PUSH 18240C26		EDX 7C802644 kernel32.7C802644
000AF156	58	POP EAX		EBX 7FFD7000
000AF157	81C0 E8E0AFA2	ADD EAX,A2AFE0E8		ESP 00C7FE68
000AF15D	81E0 A1C5A94D	AND EAX,4DA9C5A1		EBP 00C7FF90
000AF163	C1C0 12	ROL EAX,12		ESI FFFFFFFF
000AF166	^E9 FDAEFBFF	JMP unpack5.0006A068		EDI 7C920228 ntdll.7C920228
000AF168	871424	XCHG DWORD PTR SS:[ESP],EDX		EIP 000AF170 unpack5.000AF170
000AF16E	5A	POP EDX		C 0 ES 0023 32bit 0(FFFFFFFF)
000AF16F	5A	POP EDX		P 0 CS 001B 32bit 0(FFFFFFFF)
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	kernel32.	A 0 SS 0023 32bit 0(FFFFFFFF)
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]		Z 0 DS 0023 32bit 0(FFFFFFFF)
000AF176	57	PUSH EDI		S 0 FS 003B 32bit 7FFDE000(FFF)
000AF177	68 F3FF410E	PUSH 0E41FFF3		

000AF147	8BCF	MOV ECX,EDI		Registers (MMX)
000AF149	C1C8 05	ROR EAX,5		EAX 7C80FF29 kernel32.GlobalLock
000AF14C	^E9 5BAFEFEF	JMP unpack5.00099FAC		ECX 7C80FE35 kernel32.7C80FE35
000AF151	68 260C2418	PUSH 18240C26		EDX 7C802644 kernel32.7C802644
000AF156	58	POP EAX		EBX 7FFD7000
000AF157	81C0 E8E0AFA2	ADD EAX,A2AFE0E8		ESP 00C7FE6C
000AF15D	81E0 A1C5A94D	AND EAX,4DA9C5A1		EBP 00C7FF94
000AF163	C1C0 12	ROL EAX,12		ESI FFFFFFFF
000AF166	^E9 FDAEFBFF	JMP unpack5.0006A068		EDI 7C920228 ntdll.7C920228
000AF168	871424	XCHG DWORD PTR SS:[ESP],EDX		EIP 000AF170 unpack5.000AF170
000AF16E	5A	POP EDX		C 0 ES 0023 32bit 0(FFFFFFFF)
000AF16F	5A	POP EDX		P 0 CS 001B 32bit 0(FFFFFFFF)
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	kernel32.	A 0 SS 0023 32bit 0(FFFFFFFF)
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]		Z 0 DS 0023 32bit 0(FFFFFFFF)
000AF176	57	PUSH EDI		S 0 FS 003B 32bit 7FFDE000(FFF)
000AF177	68 F3FF410E	PUSH 0E41FFF3		

000AF130	81F2 241FB46D	XOR EDX,6DB41F24		Registers (MMX)
000AF136	81EA 09311FB8	SUB EDX,B91F3109		EAX 7C809B57 kernel32.CloseHandle
000AF13C	81C2 055DC282	ADD EDX,32C25D05		ECX 7C8025F0 kernel32.7C8025F0
000AF142	^E9 8E16FCFF	JMP unpack5.000707D5		EDX 7C802644 kernel32.7C802644
000AF147	8BCF	MOV ECX,EDI		EBX 7FFD7000
000AF149	C1C8 05	ROR EAX,5		ESP 001FFE38
000AF14C	^E9 5BAFEFEF	JMP unpack5.00099FAC		EBP 001FFF60
000AF151	68 260C2418	PUSH 18240C26		ESI FFFFFFFF
000AF156	58	POP EAX		EDI 7C920228 ntdll.7C920228
000AF157	81C0 E8E0AFA2	ADD EAX,A2AFE0E8		EIP 000AF170 unpack5.000AF170
000AF15D	81E0 A1C5A94D	AND EAX,4DA9C5A1		C 0 ES 0023 32bit 0(FFFFFFFF)
000AF163	C1C0 12	ROL EAX,12		P 0 CS 001B 32bit 0(FFFFFFFF)
000AF166	^E9 FDAEFBFF	JMP unpack5.0006A068		A 0 SS 0023 32bit 0(FFFFFFFF)
000AF168	871424	XCHG DWORD PTR SS:[ESP],EDX		Z 0 DS 0023 32bit 0(FFFFFFFF)
000AF16E	5A	POP EDX		S 0 FS 003B 32bit 7FFD0000(FFF)
000AF16F	5A	POP EDX		T 0 GS 0000 NULL
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	kernel32.	D 0
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]		O 0 LastErr ERROR_INVALID_HANDLE (0000
000AF176	57	PUSH EDI		FFI 00000020 (NO.NR.NF.A.NS.PO.AF.G)
000AF177	68 F3FF410E	PUSH 0E41FFF3		
000AF17C	^E9 004BFDFE	JMP unpack5.00083C81		

000AF168	871424	XCHG DWORD PTR SS:[ESP],EDX	
000AF16E	5A	POP EDX	
000AF16F	5A	POP EDX	
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
000AF176	57	PUSH EDI	
000AF177	68 F3FF410E	PUSH 0E41FFF3	
000AF17C	^E9 004BFDFE	JMP unpack5.00083C81	
000AF181	81CB C758A0F4	OR EBX,F4A058C7	
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0	
000AF18D	C1C6 1D	ROL ESI,1D	
000AF190	81E1 5DF08B6A	AND ECX,6A8BF05D	
000AF196	^0F85 6619FCFF	JNZ unpack5.00070B02	
000AF19C	^0F85 1CEEFFFF	JNZ unpack5.000ADFBE	
000AF1A2	^E9 D8D1FDFF	JMP unpack5.0008C37F	
000AF1A7	E8 B90FFFFF	CALL unpack5.000A0165	
000AF1AC	0E	PUSH CS	
EAX=7C8017E5 (kernel32.GetSystemTimeAsFileTime)			
Stack SS:[00C7FF8C]=00000000			

000AF15D	81E0 A1C5A94D	AND EAX,4DA9C5A1	
000AF163	C1C0 12	ROL EAX,12	
000AF166	^E9 FDAEFBFF	JMP unpack5.0006A068	
000AF168	871424	XCHG DWORD PTR SS:[ESP],EDX	
000AF16E	5A	POP EDX	
000AF16F	5A	POP EDX	
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	k
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
000AF176	57	PUSH EDI	
000AF177	68 F3FF410E	PUSH 0E41FFF3	
000AF17C	^E9 004BFDFE	JMP unpack5.00083C81	
000AF181	81CB C758A0F4	OR EBX,F4A058C7	
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0	
000AF18D	C1C6 1D	ROL ESI,1D	
000AF190	81E1 5DF08B6A	AND ECX,6A8BF05D	
000AF196	^0F85 6619FCFF	JNZ unpack5.00070B02	
000AF19C	^0F85 1CEEFFFF	JNZ unpack5.000ADFBE	
000AF1A2	^E9 D8D1FDFF	JMP unpack5.0008C37F	
000AF1A7	E8 B90FFFFF	CALL unpack5.000A0165	
000AF1AC	0E	PUSH CS	
EAX=7C80E876 (kernel32.FileTimeToLocalFileTime)			
Stack SS:[00C7FF88]=00000000			
Address Bay dump			
ASCII			

000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
000AF176	57	PUSH EDI
000AF177	68 F3FF410E	PUSH 0E41FFF3
000AF17C	^E9 004BF0FF	JMP unpack5.00083C81
000AF181	81CB C758A0F4	OR EBX,F4A058C7
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0
000AF18D	C1C6 1D	ROL ESI,1D
000AF190	81E1 5DF08B6A	AND ECX,6A8BF05D
000AF196	^0F85 6619FCFF	JNZ unpack5.00070B02

EAX=77DA7832 (advapi32.RegOpenKeyExA)
Stack SS:[00D7FF50]=00000000

000AF16E	5A	POP EDX
000AF16F	5A	POP EDX
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
000AF176	57	PUSH EDI
000AF177	68 F3FF410E	PUSH 0E41FFF3
000AF17C	^E9 004BF0FF	JMP unpack5.00083C81
000AF181	81CB C758A0F4	OR EBX,F4A058C7
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0
000AF18D	C1C6 1D	ROL ESI,1D
000AF190	81E1 5DF08B6A	AND ECX,6A8BF05D
000AF196	^0F85 6619FCFF	JNZ unpack5.00070B02

EAX=77DA7A9B (advapi32.RegQueryValueExA)
Stack SS:[00E7FF78]=00000000

000AF16E	5A	POP EDX
000AF16F	5A	POP EDX
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
000AF176	57	PUSH EDI
000AF177	68 F3FF410E	PUSH 0E41FFF3
000AF17C	^E9 004BF0FF	JMP unpack5.00083C81
000AF181	81CB C758A0F4	OR EBX,F4A058C7
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0
000AF18D	C1C6 1D	ROL ESI,1D
000AF190	81E1 5DF08B6A	AND ECX,6A8BF05D
000AF196	^0F85 6619FCFF	JNZ unpack5.00070B02

EAX=77DA6C07 (advapi32.RegCloseKey)
Stack SS:[00D7FF60]=00000000

000AF16E	5A	POP EDX
000AF16F	5A	POP EDX
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
000AF176	57	PUSH EDI
000AF177	68 F3FF410E	PUSH 0E41FFF3
000AF17C	^E9 004BF0FF	JMP unpack5.00083C81
000AF181	81CB C758A0F4	OR EBX,F4A058C7

EAX=7C809F01 (kernel32.InitializeCriticalSection)
Stack SS:[001FF794]=00000000

000AF16B	8/1424	XCHG DWORD PTR SS:[ESP],EDX
000AF16E	5A	POP EDX
000AF16F	5A	POP EDX
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
000AF176	57	PUSH EDI
000AF177	68 F3FF410E	PUSH 0E41FFF3
000AF17C	^E9 004BF0FF	JMP unpack5.00083C81
000AF181	81CB C758A0F4	OR EBX,F4A058C7
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0
000AF18D	C1C6 1D	ROL ESI,1D
000AF190	81E1 5DF08B6A	AND ECX,6A8BF05D
000AF196	^0F85 6619FCFF	JNZ unpack5.00070B02

EAX=7C80902B (kernel32.EnterCriticalSection), ASCII "NTDLL.RtlEnterCriticalSection"
Stack SS:[001FF798]=00000000

000AF16E	5A	POP EDX
000AF16F	5A	POP EDX
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
000AF176	57	PUSH EDI
000AF177	68 F3FF410E	PUSH 0E41FFF3
000AF17C	^E9 004BF0FF	JMP unpack5.00083C81
000AF181	81CB C758A0F4	OR EBX,F4A058C7
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0
000AF18D	C1C6 1D	ROL ESI,1D
000AF190	81E1 5DF08B6A	AND ECX,6A8BF05D
000AF196	^0F85 6619FCFF	JNZ unpack5.00070B02

EAX=7C911000 (ntdll.RtlEnterCriticalSection)
Stack SS:[001FF798]=00000000

Address: 001FF67C Hex dump: 4E000000

000CE355	81F0 CFF29B6F	XOR EAX,6F9BF2CF	
000CE358	^E9 7629FEFF	JMP unpack5.000B0CD6	
000CE360	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	
000CE363	837D F0 00	CMP DWORD PTR SS:[EBP-10],0	
000CE367	^0F85 3610F0FF	JNZ unpack5.0009F3A3	
000CE36D	837D F4 05	CMP DWORD PTR SS:[EBP-C],5	
000CE371	^E9 2010F0FF	JMP unpack5.0009F396	
000CE376	51	PUSH ECX	

EAX=00000004
Stack SS:[001FF624]=7FFDFC00, (UNICODE "NTDLL.dll")

000AF168	871424	XCHG DWORD PTR SS:[ESP],EDX	
000AF16E	5A	POP EDX	
000AF16F	5A	POP EDX	
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	kernel32.LeaveCriticalSection
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
000AF176	57	PUSH EDI	
000AF177	68 F3FF410E	PUSH 0E41FFF3	
000AF17C	^E9 004BF0FF	JMP unpack5.00083C81	
000AF181	81CB C758A0F4	OR EBX,F4A058C7	
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0	
000AF18D	C1C6 1D	ROL ESI,1D	

EAX=7C809137 (kernel32.LeaveCriticalSection), ASCII "NTDLL.RtlLeaveCriticalSection"
Stack SS:[001FF798]=00000000

000AF16E	5A	POP EDX	
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	ntdll.RtlLeaveCriticalSection
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
000AF176	57	PUSH EDI	
000AF177	68 F3FF410E	PUSH 0E41FFF3	
000AF17C	^E9 004BF0FF	JMP unpack5.00083C81	
000AF181	81CB C758A0F4	OR EBX,F4A058C7	
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0	
000AF18D	C1C6 1D	ROL ESI,1D	

EAX=7C9110E0 (ntdll.RtlLeaveCriticalSection)
Stack SS:[001FF798]=00000000

000AF16E	5A	POP EDX	
000AF16F	5A	POP EDX	
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	advapi32.RegCreateKeyExA
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
000AF176	57	PUSH EDI	
000AF177	68 F3FF410E	PUSH 0E41FFF3	
000AF17C	^E9 004BF0FF	JMP unpack5.00083C81	
000AF181	81CB C758A0F4	OR EBX,F4A058C7	
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0	
000AF18D	C1C6 1D	ROL ESI,1D	

EAX=77DAE834 (advapi32.RegCreateKeyExA)
Stack SS:[001FF7B0]=00000000

000AF16E	5A	POP EDX	
000AF170	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	advapi32.RegSetValueExA
000AF173	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
000AF176	57	PUSH EDI	
000AF177	68 F3FF410E	PUSH 0E41FFF3	
000AF17C	^E9 004BF0FF	JMP unpack5.00083C81	
000AF181	81CB C758A0F4	OR EBX,F4A058C7	
000AF187	81E5 B0CF1A09	AND EBP,91ACFB0	
000AF18D	C1C6 1D	ROL ESI,1D	

EAX=77DAE927 (advapi32.RegSetValueExA)
Stack SS:[001FF7BC]=00000000