TRUQUITOS VARIOS

(LECCION CUARENTA Y UNO)

El programa de hoy se llama PC DATA FINDER 5.5, y sirve para buscar datos y muchas otras cosas dentro de una PC. Se baja de

http://www.silverlaketech.com/PCDFmain.asp

y es un programita que tiene algunas cositas para despistar aunque como veremos se puede crackear, las trabas no son muy grandes pero si lo suficientes para complicar un poco la vida.
Abrimos el ejecutable con el Wdasm y carga bien, vemos entre las cadenas de texto (STRINGS REFERENCES) que dice por ahí BORLAND, DEPLHI, etc, lo que significa que esta escrito en DELPHI.
Abrimos el programa y vamos a ABOUT y nos aparece la opción para registrarnos ponemos una clave falsa y cualquier nombre y cuando le damos a REGISTER, nos sale **INVALID REGISTRATION CODE**, bueno probamos con BPX MESSAGEBOX, BPX HMEMCPY (increíble que no pare en el SOFTICE con este), y varios mas y no entra en el SOFTICE y nos sale el funesto cartelito.
Dicho cartelito no aparece entre las STRINGS REFERENCES.
Probamos un BPX que suele servir bastante en DELPHI que es **BPX DRAWTEXTA** y ahí si por lo menos para en el SOFTICE y con F12 volvemos al ejecutable, aunque en cualquier lugar, borramos con BC*, y observamos algo raro.
En el SOFTICE las posiciones de memoria cuando estamos en el ejecutable son valores tipo 46eca7, en cambio en el WDASM son valores tipo 7ec18 muy diferentes, hmmm.
Que pasa aquí.
Vayamos al WDASM y veamos el ENTRY POINT, vamos a GOTO - GOTO PROGRAM ENTRY POINT y eso nos lleva aquí

```
//******************** Program Entry Point ********
:000 9A018 55                  push ebp
:0009A019 8BEC                 mov ebp, esp
:0009A01B 83C4EC               add esp, FFFFFFEC
```

a **9A018** seria en ENTRY POINT según el WDASM.

Ahora tratemos de cargar el programa desde el SYMBOL LOADER del Softice para que pare cuando arranque en el mismo ENTRY POINT, y vemos que allí nos dice **48a018** en realidad marca 48a017 pero ese valor es 00 y el siguiente es el 55 que corresponde a **push ebp.**
O sea que **9a018** del WDASM corresponde a **48a018** del SOFTICE y porque ocurre esto, debe ser si miramos el ejecutable con el PEEDITOR vemos que la **image base es 10000**, y ya que no esta comprimido ya que cargo directamente en el WDASM con sus STRINGS REFERENCES y todo, podemos suponer que este numero esta cambiado para despistar, o sea que si restamos 9a018 -10000 obtendríamos el offset que seria 8a018 y si allí le sumamos la image base que generalmente se utiliza que es 400000 seria

400000 + 8a018 = 48a018 que es el valor que sale en el softice.

O sea que hallamos una formulita para pasar los valores del WDASM al Softice en este caso, seria la siguiente al valor del WDASM restarle 10000 y sumarle luego 400000, para hacerlo en un solo paso le podemos sumar al valor de WDASM 3f0000 y listo ya que en hexa eso equivale a hacer toda esa operación en una sola.

9a018 + 3f0000 = 48a018

WDASM + 3f0000= Softice

Con esa formulita nos ayudaremos un poco. Bueno dado que no tenemos ganas de dar muchas vueltas y como esta hecho en DELPHI usaremos el DEDE, para eso tenemos la versión 2.50 que es la ultima y completa, tratamos de cargarlo con el dede y puum, error de no se que y se cierra el dede, ufff.
Otro truquito, pero para algo este cracker guarda todo, probaremos con una versión vieja del dede la 2.34 que le faltan muchas utilidades pero nos servirá.
Cargamos desde este DEDE y carga perfectamente.
Vamos a procedures y vemos REGISTRATION y si hacemos click allí a la derecha en EVENTS nos aparece OKBTNCLICK, y dado que cuando ingresas la clave apretas un botón que dice OK, probemos con esto.
Si hacemos click derecho encima de OKBTNCLICK, y vamos a Disassemble nos aparece

```
0007F0F4   55                  push   ebp
0007F0F5   8BEC                mov    ebp, esp
0007F0F7   81C4FCFEFFFF        add    esp, $FFFFFEFC
0007F0FD   53                  push   ebx
0007F0FE   33C9                xor    ecx, ecx
0007F100   894DFC              mov    [ebp-$04], ecx
0007F103   8BD8                mov    ebx, eax
0007F105   33C0                xor    eax, eax
0007F107   55                  push   ebp
```

O sea que haciendo la conversión para el SOFTICE seria
7f0f4 + 3f0000 = **46f0f4**

Pondremos un BPX allí en 46f0f4
podemos poner primero un BPX DRAWTEXTA y cuando vuelve al ejecutable poner el BPX 46f0f4 allí.
Ahora vemos que cuando ingresamos un nombre y una clave, para el softice en
46f0f4 push ebp
bueno a partir de allí podemos hacer F10 hasta que aparezca el cartelito
INVALID REGISTRATION CODE

Esto ocurre en **46f1ff CALL 45f3ff**

Un poquito antes hay un CALL, un **test al, al** y un salto que puede evitar este cartelito.

```
46f1d3      call 46ebd0
            test al, al
            jnz 46f21f
```

si a este salto lo invertimos y cuando llegamos allí con f10 ponemos

r eip=46f21f

vemos que nos dice que estamos registrados, pero al volver a arrancar el programa, vuelve a arrancar desregistrado.
se ve que realiza la comprobación en otra parte cuando comienza, esto lo vemos en el WDASM cuando entramos en ese CALL que dice que también es llamado desde otra parte del programa

esto se ve en el WDASM

```
* Referenced by a CALL at Addresses:
|:0007F070   , :0007F1D3
|
:0007EBD0 55                push ebp
:0007EBD1 8BEC              mov ebp, esp
:0007EBD3 83C4F8            add esp, FFFFFFF8
:0007EBD6 53                push ebx
:0007EBD7 33C9              xor ecx, ecx
```

allí dice que viene de 7fd13 que era donde estábamos y de otro lugar que es 7f070, todo esto en valores WDASM sumar 3f0000 para obtener el valor en el SOFTICE.

7f070 + 3f0000 = **460f70**

allí también hay una llamada al CALL un **test al, al** y un salto condicional pero si ponemos un BPX allí y una vez que arranca el programa invertimos el salto, arranca desregistrado, hmmmm.

Entonces el problema debe estar dentro del CALL miremos dentro
ESTO MUESTRA EL WDASM SUMAR 3f0000 para obtener los valores para el SOFTICE.

Esto encontramos un poco mas abajo dentro del CALL

```
:0007EC15 83F80B            cmp eax, 0000000B
:0007EC18 0F858B000000      jne 0007ECA9
```

parece ser una comparación de la cantidad de cifras ya que 0b en hexa es 11, y podría ser 11 cifras, probemos poniendo un BPX allí y cuando pare veremos que en EAX esta la

cantidad de cifras que pusimos, o sea que la clave debería tener 11 cifras, pero también sabemos que si no tiene 11 cifras va a 7eca9 que es que te manda al cesto de basura ya que.

```
:0007ECA9 33C0              xor eax, eax
:0007ECAB 5A                pop edx
:0007ECAC 59                pop ecx
:0007ECAD 59                pop ecx
:0007ECAE 648910            mov dword ptr fs:[eax], edx
:0007ECB1 68CBEC0700        push 0007ECCB

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0007ECC9(U)
|
:0007ECB6 8D45F8            lea eax, dword ptr [ebp-08]
:0007ECB9 BA02000000        mov edx, 00000002
:0007ECBE E86D4EF9FF        call 00013B30
:0007ECC3 C3                ret
```

alli hace EAX = 0 en el **XOR EAX, EAX** y llega al RET siendo EAX=0 y luego en **TEST al, al** como al es igual a cero no te registra.

Un par de sentencias mas arriba de **7eca9** vemos que sale de muchas comparaciones y hace BL = 1

deberíamos probar que pasa si salteamos desde donde comprueba las cifras a donde hace BL=1 para ver si registra.

```
:0007EC15 83F80B            cmp eax, 0000000B
:0007EC18 0F858B000000      jne 0007ECA9
```

Reemplazamos **jne 7eca9** por **jmp 7eca7** y ponemos un BPX en ese salto 7ec18 para que pare también allí cuando arranca el programa, y vemos que poniendo un JMP 7eca7 nos registra y cuando arranca el programa para allí dos veces y si hacemos también allí JMP 7eca7 arranca registrado.

O sea que lo único que hay que hacer para crackearlo es reemplazar el **jne 0007ECA9** por **jmp 7ECA7** Y LISTO **HACE BL=1** Y LUEGO PASA EL 1 A EAX, Y TE REGISTRA PERFECTAMENTE.

Ricardo Narvaja

OTRO PROGRAMA DE 16 BITS

(LECCION CUARNTA Y DOS

SE llama MEGAPAK 8.1.2 Multiempresa y esta hecho en 16 bits, se baja de:

http://148.233.25.146/sisteval.html#megapaqw

Aparentemente sirve para Facturación, compras ventas y esas cositas que hacen las empresas, jua.
Como ya saben por las lecciones anteriores, le huyo bastante a estos programas, pero bueno, era para un amigo, y bueno lo hicimos.
Tratamos primero de encontrar la clave, la cual esta bastante encriptada, para el que quiere practicar con claves encriptadas y la quiere hallar, le doy una ayuda transcripta de los mails que nos mandamos en la lista crackslatinos sobre el tema, por supuesto como yo encontre la forma de crackearlo sin tener la clave este camino quedo inconcluso, para el que lo quiera investigar le transcribo los mails, para el que quiere saber como crackearlo, saltee esto y siga mas abajo en donde esta el titulo **COMO CRACKEARLO**.

MAILS SOBRE LA CLAVE ENCRIPTADA

pones la clave falsa
BPX HMEMCPY

vas con F10 hasta REPZ MOVSB
allí haces
d es:di
me sale la clave falsa a mi (EN MI MAQUINA EN LA TUYA PUEDE VARIAR)
en
3677:354c
hago
PAGE 3677:354c
y me sale el resultado

LINEAR
811130cc

si hago d 0030:811130cc

me aparece la clave falsa donde la guardo
pongo un BPR allí en ese rango

BPR 0030:811130cc 0030:811130dc rw

borro el BPX HMEMCPY con BC00
y le doy a X para que corra el programa

para la primera vez que toca la clave en
015f:a9f5 Repz scasb
que se usa para ver cuantas cifras tiene
seguís con F10 hasta el RET y allí en EAX las dos últimas cifras son la
cantidad de cifras de tu clave falsa
hay que ver si la compara con algo
si compara AX con cero así que aquí no pasa nada le doy con X de nuevo
para en 0eff:5400
compara la primera cifra con cero
sigo con X
compara en 0eff:53e8
compara con 20 (espacio)
este ciclo se repite cifra por cifra
una vez que sale de allí
para en
015f:a9f5 REPNZ SCASB
cuenta las cifras igual que antes al salir del RET
en 0eff:0dd1

Compara la cantidad de cifras con 1a o sea 26 y parece que si no es 26 te tira al tacho por lo que hay que poner 26 cifras de la clave para seguir a partir de aquí. Luego en 0eff:5370 un poco más adelante empiezan comparaciones con valores fijos.
Bueno este es un programa que encripta la clave, por lo cual hay que seguirlo bastante, y tener en cuenta cada lugar que guarda y cada operación que hace, y lleva su tiempo. Como ya te dije tiene que tener la clave 26 cifras según lo que se ve en la comparación que te mencione en el mail anterior.
Otra cosa que hay que ver cuando pones un BPR en tu clave falsa es que cuando el programa toma una cifra y la coloca en EAX , en este caso AX o cualquier otro de esos, además de verificar las comparaciones y adonde lo guarda y allí volver a poner otro BPR, también hay que fijarse que casi siempre estos programas que comparan cifra por cifra, guardan con alguna sentencia tipo PUSH EAX o PUSH AX, (SIEMPRE QUE EN AX este el valor de alguna de tus cifras de la clave falsa)
De esta forma lo guarda en el STACK y mas adelante lo recupera de allí, por lo que si en este caso mi clave falsa la primera cifra es 9 , o sea que en hexa es 39, y lo guarda en AX, si el programa hace PUSH AX, lo esta guardando en el STACK y lo puede recuperar mas adelante, en este caso lo que hay que hacer, es poner un BPR en el lugar que lo guarda cosa de que cuando lo quiera volver a leer pare el SOFTICE allí en 10df:51b2 PUSH AX (puede variar la primera parte o sea 10df según la maquina) en AX esta 39 que es el 9 de mi primera cifra y donde lo guarda cuando hace PUSH EAX, hay que fijarse en registro ESP que es el puntero del STACK ese indica donde se guarda.
En mi caso es 10df:5370 pongo un BPR allí, conviene hacer el mismo procedimiento que hicimos cuando paramos en el HMEMCPY que te explique en el mail anterior o sea
PAGE 10df:5370
LINEAR
xxxxxxxxxx

D 0030::xxxxxxxxx

y si allí aparece el 39
hacer

BPR 0030:xxxxxxxxx 0030:xxxxxxxy rw
para que abarque el valor y pare el SOFTICE cuando lo lea de nuevo.
En este programa es este caso, lo guarda allí a la cifra
y cuando para el SOFTICE que la vuelve a leer sigue trabajando con ella,
esta es una forma de esconder la búsqueda de la clave y despista a la
mayoría de los novatos, es muy importante saber esto y si ves que EAX es 39
o EBX o cualquiera de esos, tracear con F10 y ver lo que hace y donde guarda
esos valores y si hace PUSH, por lo menos hasta que cambie el valor de EAX a
otra cosa.
Bueno
luego anote varios lugares donde para el SOFTICE
10df:51dd add es: [bx+si], al

allí guarda los valores si lo seguís vas a ves que a cada cifra la
transforma y si mi clave falsa es 9898989898 ..... (26 cifras)

toma el 9 , lo transforma en 90 y lo guarda en
550f:2e56
luego toma el 8 y se lo suma en la sentencia que te mostré antes
10df:51dd add es: [bx+si], al

o sea que si miro allí guardo 98 como es mi clave, y así sucesivamente
guarda la clave 98 98 98 98 98 98 98 .... cifra por cifra.
Ahora empieza lo bueno la encriptación

lógico que hay que poner un bpr allí donde esta mi clave así guardada 98 98.....

sigo corriendo el programa y para en

10df:56de mov al , [bp+si-66]

que es tomar la primera cifra de la clave, luego la guarda en 550f:2dd6,
poner otro BPR aquí
luego hay que ir anotando las operaciones que le realiza a cada cifra.
en
10df:56e5 XOR al, [bp+ff0e]

hace un XOR (muy usado en encriptaciones) con el valor E4, hay que anotar
todo ya que mas adelante hay que realizar todas las operaciones que realizo
sobre tu clave falsa, hacia atrás sobre el número que compara con el tuyo
encriptado para encontrar la clave verdadera.
Al es 7c luego de realizarle el XOR y lo guarda en 550f:2e00, poner otro BPR
aquí.

Bueno hasta aquí llegaron mis ganas de hallar la clave quizás a alguien le sirva y le interesa practicar y seguir desde aquí, bueno yo me canse.

COMO CRACKEARLO

Cuando ingresamos la clave ponemos un BPX MESSAGEBOX sin la A al final ya que es 16 bits y cuando para allí, puedo volver con RET, pero este camino no me lleva (esto ya que probé y probé si alguno de esos saltos que había antes del MESSAGEBOX me registraba. Como no pude hacer nada opte por ver las STRINGS REFERENCES y alli estaba.
Su... no está registrado quedan... veces para entrar al sistema que es un cartel que aparece cuando no estás registrado, si podemos lograr saltearlo puede funcionar como registrado, veremos. Aquí viene una larga parte copiada del WDASM que termina justo en la referencia a esa STRING.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.13E3(U)
|
:0001.1471 3D0000              cmp ax, 0000
:0001.1474 7503                jne 1479
:0001.1476 E96DFF              jmp 13E6


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.1474(C)
|
:0001.1479 3D0100              cmp ax, 0001
:0001.147C 7503                jne 1481
:0001.147E E9CBFF              jmp 144C


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.147C(C)
|
:0001.1481 3D0400              cmp ax, 0004
:0001.1484 7503                jne 1489
:0001.1486 E97FFF              jmp 1408


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.1484(C)
|
:0001.1489 3D0500              cmp ax, 0005
:0001.148C 7503                jne 1491
:0001.148E E999FF              jmp 142A
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.148C(C)
|
:0001.1491 E90000              jmp 1494

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0001.1405(U), :0001.1427(U), :0001.1449(U), :0001.146B(U), :0001.146E(U),
|:0001.1491(U)
|
:0001.1494 FFB650FB            push word ptr [bp+FB50]
:0001.1498 FFB654FB            push word ptr [bp+FB54]
:0001.149C 9AFFFF0000          call LCRYPKYD.CKCHALLENGE
:0001.14A1 898652FB            mov [bp+FB52], ax
:0001.14A5 8B8652FB            mov ax, [bp+FB52]
:0001.14A9 39865AFB            cmp [bp+FB5A], ax
:0001.14AD 7503                jne 14B2
:0001.14AF E91B00              jmp 14CD


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.14AD(C)
|
:0001.14B2 6A00                push 0000

* Possible StringData Ref from Data Seg 002 ->"Fall"
                        |
:0001.14B4 B80E03              mov ax, 030E
:0001.14B7 8CDA                mov dx, ds
:0001.14B9 52                  push dx
:0001.14BA 50                  push ax

* Possible StringData Ref from Data Seg 002 ->"Error"
                        |
:0001.14BB B80803              mov ax, 0308
:0001.14BE 8CDA                mov dx, ds
:0001.14C0 52                  push dx
:0001.14C1 50                  push ax
:0001.14C2 683020                  push 2030
:0001.14C5 9AB90A0000          call USER.MESSAGEBOX
:0001.14CA E9BA05              jmp 1A87


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.14AF(U)
|
:0001.14CD 6A01                push 0001
:0001.14CF 9AFFFF0000          call LCRYPKYD.GET1RESTINFO
```

```
:0001.14D4 8986ECFC             mov [bp+FCEC], ax
:0001.14D8 83BEECFC02           cmp word ptr [bp+FCEC], 0002
:0001.14DD 7503                 jne 14E2
:0001.14DF E90A00               jmp 14EC


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.14DD(C)
|
:0001.14E2 83BEECFC01           cmp word ptr [bp+FCEC], 0001
:0001.14E7 7403                 je 14EC
:0001.14E9 E96801               jmp 1654


* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0001.14DF(U), :0001.14E7(C)
|
:0001.14EC 8B46F8               mov ax, [bp-08]
:0001.14EF 8B56FA               mov dx, [bp-06]
:0001.14F2 398660FB             cmp [bp+FB60], ax
:0001.14F6 7403                 je 14FB
:0001.14F8 E95601               jmp 1651


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.14F6(C)
|
:0001.14FB 399662FB             cmp [bp+FB62], dx
:0001.14FF 7403                 je 1504
:0001.1501 E94D01               jmp 1651


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.14FF(C)
|
:0001.1504 6A02                 push 0002
:0001.1506 9AD0140000           call LCRYPKYD.GET1RESTINFO
:0001.150B 8946FC               mov [bp-04], ax
:0001.150E 6A03                 push 0003
:0001.1510 9A07150000           call LCRYPKYD.GET1RESTINFO
:0001.1515 8946F4               mov [bp-0C], ax
:0001.1518 8B46FC               mov ax, [bp-04]
:0001.151B 2B46F4               sub ax, [bp-0C]
:0001.151E 89865CFB             mov [bp+FB5C], ax
:0001.1522 837E1A00             cmp word ptr [bp+1A], 0000
:0001.1526 7503                 jne 152B
:0001.1528 E91801               jmp 1643
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.1526(C)
|
:0001.152B FF761C               push word ptr [bp+1C]
:0001.152E 6A00                 push 0000

* Possible Reference to Dialog: DialogID_03F3
                    |
:0001.1530 68F303               push 03F3
:0001.1533 6A00                 push 0000
:0001.1535 6A00                 push 0000
:0001.1537 6A00                 push 0000
:0001.1539 9AFFFF0000           call USER.CREATEDIALOG
:0001.153E 898656FB             mov [bp+FB56], ax
:0001.1542 8B5E0A               mov bx, [bp+0A]
:0001.1545 D1E3                 shl bx, 01
:0001.1547 D1E3                 shl bx, 01
:0001.1549 FFB75200             push word ptr [bx+0052]
:0001.154D FFB75000             push word ptr [bx+0050]

* Possible StringData Ref from Data Seg 002 ->"Registro de %s"
                    |
:0001.1551 B85B03               mov ax, 035B
:0001.1554 8CDA                 mov dx, ds
:0001.1556 52                   push dx
:0001.1557 50                   push ax
:0001.1558 8D86EEFE             lea ax, [bp+FEEE]
:0001.155C 8CD2                 mov dx, ss
:0001.155E 52                   push dx
:0001.155F 50                       push ax
:0001.1560 9ACA0E0000               call USER._WSPRINTF
:0001.1565 83C40C                   add sp, 000C
:0001.1568 FFB656FB                 push word ptr [bp+FB56]
:0001.156C 8D86EEFE                 lea ax, [bp+FEEE]
:0001.1570 8CD2                     mov dx, ss
:0001.1572 52                       push dx
:0001.1573 50                       push ax
:0001.1574 9ADC0E0000               call USER.SETWINDOWTEXT
:0001.1579 83BEECFC02               cmp word ptr [bp+FCEC], 0002
:0001.157E 7403                     je 1583
:0001.1580 E92D00                   jmp 15B0


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.157E(C)
|
:0001.1583 FFB65CFB             push word ptr [bp+FB5C]
```

```
:0001.1587 8B5E0A            mov bx, [bp+0A]
:0001.158A D1E3              shl bx, 01
:0001.158C D1E3              shl bx, 01
:0001.158E FFB75200          push word ptr [bx+0052]
:0001.1592 FFB75000          push word ptr [bx+0050]

* Possible StringData Ref from Data Seg 002 ->"Su %s no ha sido registrado, quedan "
                                           ->"%d veces para entrar al sistema."
```

En negrita están los diversos saltos que probando con el SOFTICE no sirven ya que arrancando el programa y poniendo BPXs en todos esos y invirtiéndolos el programa arranca desregistrado, aquí influye mucho esto de probar, pero si vemos el listado que acabo de transcribir, yo sospeche primero que nada de la REFERENCIA que esta al comienzo del listado.

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0001.13E3(U)

Yo sospeche que de aquí podia provenir ya que todas las otras referencias eran dentro de este mismo listado en cambio la referencia sospechosa provenia de otro lugar, como si el programa esta registrado no salta hacia este sector y sigue por otro lado, y era asi, el programa registrado no pasa por todo el listado que puse antes, entonces si viene de 13e3, vemos que hay por alli, para tratar de evitar que venga hacia el cartel maldito.

Ahora vemos el listado que termina en 13e3 donde está el salto que hay que evitar.

```
:0001.13A5 837EF200          cmp word ptr [bp-0E], 0000
:0001.13A9 7E03              jle 13AE
:0001.13AB E9DC06            jmp 1A8A

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.13A9(C)
|
:0001.13AE 8B46F2            mov ax, [bp-0E]
:0001.13B1 E9C006            jmp 1A74


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.1A81(U)
|
:0001.13B4 8D8656FB          lea ax, [bp+FB56]
:0001.13B8 8CD2              mov dx, ss
:0001.13BA 52                push dx
:0001.13BB 50                push ax
:0001.13BC 9ABE355E13        call 0001.35BE
:0001.13C1 83C404            add sp, 0004
```

```
:0001.13C4 898650FB          mov [bp+FB50], ax
:0001.13C8 8D8656FB          lea ax, [bp+FB56]
:0001.13CC 8CD2              mov dx, ss
:0001.13CE 52                push dx
:0001.13CF 50                push ax
:0001.13D0 9ABE35BF13        call 0001.35BE
:0001.13D5 83C404            add sp, 0004
:0001.13D8 F7AE50FB          imul word ptr [bp+FB50]
:0001.13DC 898654FB          mov [bp+FB54], ax
:0001.13E0 8B460A            mov ax, [bp+0A]
:0001.13E3 E98B00            jmp 1471
```

Vemos que las sentencias en negrita son las que importan, arriba en 13a9 esta la clave de todo, alli ponemos un **BPX 13a9** para probar arrancamos el programa y vemos qoe para que no vaya al salto en 13e3 no tiene que saltar ya que si salta va a 13ae y cae irremediablemente al salto de 13e3 que lleva al cartel maldito.

Bueno una vez que para allí por el BPX lo tenemos que nopear o sea hacemos
R eip= 13ab
que es la sentencia subsiguiente y luego hacemos X y arranca registrado, sin problemas.

O sea que si vamos al ULTRA EDIT y buscamos la cadena de es salto que es
7e 03 e9 dc 06 8b 46 f2 e9 c0 06 8d 86 56 fb

y reemplazamos en 7e 03 por 90 90 quedara el programa registrado perfectamente.

Ricardo Narvaja

PROGRAMAS BASTANTE MOLESTOS
(POR LO EXTENSO DE LA PROTECCIÓN)

(LECCION CUARENTA Y TRES)

Este es un programa llamado BANK EASY 2.1 que sirve para llevar cuentas bancarias y ese tipo de transacciones.
Se baja de:
http://www.ghostsolutions.com/Main/BankEasy/BankEasy.htm

Y la verdad que tiene tantos lugares que parchear que ya ni recuerdo bien todos, trataremos de ir de a poco.
Lo que si es cierto es que no esta comprimido ni nada, las STRINGS, aparecen bien aunque no todas salen en la lista de STRINGS REFERENCES del WDASM, a pesar que en el listado desensamblado del WDASM si aparecen.
No se cual es la verdadera razón de esto pero creo que puede ser esta:

Si en el Wdasm voy a Search - FIND y tecleo por ejemplo para que busque PC DATE que es parte de un cartelito molesto que aparece cuando adelantas la fecha de tu compu y ejecutas el programa y después volvés atrás la fecha y lo ejecutas te sale un cartel de que hubo un problema con la fecha de tu pc y es el siguiente:

* Possible StringData Ref from Data Obj ->"Messages**There seems to be an error "**
->"with your PC date. The last time "
->"the program was run was the "

Eso es lo que aparece en el maldito cartel exactamente **THERE SEEMS TO BE AN ERROR WITH YOUR PC DATE. THE LAST TIME.....**

Sin embargo esa STRING no aparece entre las STRINGS REFERENCES dela lista, y hay muchas así, no se la razón pero creo que debe ser por el tipo de cartel que utiliza la función **DialogBoxIndirectParama**, que si ponemos un Bpx allí para en esos cartelitos maléficos.
Que hay muchos y surtidos y cuando aparecen el programa arranca pero como vencido, los iconos no aparecen activos y no se puede usar, así que no solo hay que tratar de que el programa piense que estamos registrados sino también, que saltee todas estas protecciones de que si adelantaste el reloj, otro cartelito dice que no tenés licencia valida, otro que tu licencia es fraguada, otro que falta un driver (mentira) , y así hay varios, la cuestión es que cuando aparece uno de estos carteles, no arranca bien el programa, de idéntica forma como si esta vencido.
En cualquier caso que queramos buscar una STRING de esos cartelitos conviene mas buscarlas por SEARCH-FIND y poner una o dos palabras del cartelito y si no sale nada probar con otras dos palabras que buscarlas en la lista de STRING REFERENCES.

Bueno al ataque mis valientes con paciencia vamos a ir al revés de la solución a la explicación:
Aquí va primero la lista de bites que hay que modificar para crackear esta protección pelmazo. (por lo larga)

La primera columna tiene el offset la segunda los valores originales y la tercera los que hay que colocar en lugar de los originales.

Es bueno ver que para hallar la posición de memoria en el WDASM no podemos sumarle 400000 que es la Image base, porque si vemos con el PEDDITOR las secciones no están arregladas, o sea que por eso no da la suma pero podemos calcular fácilmente viendo el ENTRY POINT o cualquier punto, la diferencia entre el offset y el valor en el WDASM.
aquí hay que sumarle 400C00, podríamos arreglarlo con el PEEDITOR y luego sumar directamente 400000, pero por si acaso chequee las secciones o algo así, mas vale tomémonos la molestia y sumemos no es difícil calculadora en mano.

<offset>	<File 1 Byte>	<File 2 Byte>
45277h	28h	50h
452C8h	80h	C6h
452C9h	3Dh	5h
452CEh	0h	1h
452CFh	Fh	90h
452D0h	84h	90h
452D1h	B7h	90h
452D2h	0h	90h
452D3h	0h	90h
452D4h	0h	90h
45335h	83h	B0h
45336h	F8h	0h
45337h	0h	90h
4538Ch	89h	B0h
4538Dh	F0h	0h
9B21Dh	Fh	E9h
9B21Eh	85h	9Ch
9B21Fh	9Bh	0h
9B222h	0h	90h
B194Fh	7Eh	EBh
B1986h	7Eh	EBh
B19C6h	Fh	E9h
B19C7h	8Dh	C3h
B19C8h	C2h	1h
B19C9h	1h	0h
B19CBh	0h	90h
B1BA0h	Fh	E9h
B1BA1h	8Eh	C3h
B1BA2h	C2h	1h
B1BA3h	1h	0h
B1BA5h	0h	90h
B1D6Fh	75h	EBh
B1DAEh	74h	EBh
B1DD8h	75h	EBh

los primeros valores de la lista 45277 que corresponden al WDASM 445e77 son de la rutina de registro, se llega a ella por cualquier lugar pero podemos ver que es el
CALL 445e1c

en este listado se ve claro

```
:004B4572 E8A518F9FF        call 00445E1C
:004B4577 3C00              cmp al, 00
:004B4579 7524              jne 004B459F
:004B457B 8B442427          mov eax, dword ptr [esp+27]
:004B457F E848BDF8FF        call 004402CC
:004B4584 89E8              mov eax, ebp
:004B4586 BBFF000000        mov ebx, 000000FF
```

* Possible StringData Ref from Data Obj ->"**This feature is not avaliable "**
->"in your Level of the program. "
->" Consult your application supplier "
->"for more information"
|

Ahí esta el llamado al **CALL 445e1c** y al volver de la rutina si AL es 0 es como si estuvieras registrado (mas o menos ya veremos) y saltea el cartelito de que THIS FEATURE IS NOT AVAILABLE .... o sea que generalmente saltea las restricciones del programa si AL=0

Vemos también que las restricciones son bastantes observando las REFERENCES de donde es llamado ese **CALL 445e1c**

Sin aburrirnos miremos todos los lugares de donde proviene

* Referenced by a CALL at Addresses:
|:00402D32 , :00402D8A , :00402FB6 , :0040309B , :00404F90
|:00404FBD , :00406466 , :004065EA , :00409E6D , :0040A913
|:0040B475 , :0040B502 , :0040B579 , :0040B597 , :0040B5C1
|:0040B5DF , :0040B605 , :0040B623 , :0040D8BB , :0040F464
|:0040F6D0 , :0040FE00 , :00411E2E , :00415E80 , :00415EBF
|:00415EFB , :00415F8F , :00415FE2 , :00416035 , :00416085
|:00419E6B , :0041A82A , :0041EA0D , :0041EA59 , :0041EAC9
|:0041EAE3 , :0041EE23 , :0041EE8B , :0041EEF1 , :00421644
|:004216B6 , :0042172C , :004217A3 , :004217C5 , :0042182D
|:00421895 , :004218D6 , :004218FB , :00421920 , :00421945
|:0042196A , :0042198F , :004219B4 , :00421A84 , :00421AA9
|:00421B19 , :00421B3B , :00421BAF , :00421C31 , :00421C56
|:00421CCB , :00421D31 , :00421D53 , :00421DB9 , :00422C60
|:0042337C , :00423404 , :00425700 , :00425771 , :004257AA
|:0042581B , :00425890 , :00425945 , :00425967 , :004259CE
|:00425A35 , :00425A9E , :00425AC3 , :00425AE8 , :00425B0D

|:00425B2F , :00425B51 , :00425B73 , :00425BF0 , :00425C77
|:00425CE6 , :00425D08 , :00425D7B , :00425E01 , :00425EB4
|:00425F19 , :00425F3B , :00425FA0 , :00425FD5 , :004260AF
|:00426122 , :0042CD5B , :0042CDD3 , :0042D01E , :0042EE84
|:0042EEB7 , :0042EEEA , :0042EF1E , :0042EF67 , :0042EFB0
|:0042EFF6 , :0042F086 , :0042F20C , :0042F26D , :0042F31E
|:0042F7AD , :0042F7EB , :0042FA20 , :004307A3 , :00430913
|:00430961 , :00430D08 , :004316BA , :004316FB , :00432924
|:00433FB7 , :004340AC , :0044DCDD , :0044DD1D , :0044DE3C
|:0044DEA5 , :00488910 , :004889DC , :00489DE9 , :0048A1B5
|:0048A683 , :0048A6BD , :0048A6F7 , :0048A72E , :0048A765
|:0048A7A6 , :0048A7E7 , :0048A828 , :0048A85F , :0048A896
|:0048A8D7 , :0048A91E , :0048ABCC , :0048B3C7 , :0048B5B8
|:0048B660 , :0048B73F , :0048B827 , :0048B94A , :0048BA6D
|:0048D3AD , :0048D4B5 , :0048D612 , :0048D813 , :0048D938
|:0048DB3A , :0048DCA6 , :0048DEA8 , :0048DFB2 , :0048E175
|:0048E2CD , :0048E4CF , :0048E5A8 , :0048E5D2 , :0048E682
|:0048E6AC , :0048E75C , :0048EA12 , :0048EA5F , :0048F2BA
|:0048F33D , :0048F3C0 , :0048F54C , :0048F660 , :0048F98F
|:0048F9E8 , :0048FED0 , :0048FF3E , :0048FF5D , :0048FF7C
|:0049028E , :00490332 , :00490425 , :00490617 , :00491144
|:004927BA , :004927FE , :00492892 , :004928BC , :004928E6
|:00492944 , :0049299F , :00492A16 , :00492F36 , :0049315B
|:0049328E , :004932AC , :004932CF , :00493488 , :004936F6
|:00494517 , :00494D11 , :00494D2F , :00495847 , :00496832
|:00496A57 , :00496A7C , :00496AA1 , :00496D8A , :00496EA0
|:00496EBF , :00496F96 , :00496FB8 , :00497196 , :004976A4
|:00498C38 , :00498DCE , :00499F31 , :0049A2AA , :0049B31B
|:0049B5FA , :0049B62E , :0049EEFC , :0049F50B , :004A1EF8
|:004A2519 , :004A2535 , :004A25B8 , :004A26C0 , :004A4929
|:004B1189 , :004B1817 , :004B1BFC , :004B1C6A , :004B1C88
|:004B1CAA , :004B1CF7 , :004B1DFD , :004B3253 , :004B414C
|:004B42D2 , :004B4419 , :004B445E , :004B44A3 , :004B44E8
|:004B452D , :004B4572 , :004B45B7 , :004B45FC , :004B4641
|:004B4686 , :004B46CB , :004B4710 , :004B4755 , :004B479A
|:004B47DF , :004B4824 , :004B4869 , :004B48AB , :004B497F
|:004B49F0 , :004B5235 , :004B59B1 , :004B5A3E , :004B5DAD
|:004B5F48 , :004B6090 , :004B673A , :004B6797 , :004B735B
|:004B73E3 , :004B7CA0 , :004B7D3C , :004B858C , :004B871C
|:004B886E , :004B92D8 , :004BA99D , :004BA9FD , :004BAA39
|:004BB25E , :004BB426 , :004BB4FB , :004BB640 , :004BB6A0
|:004BD67C , :004BDDC3 , :004BDECA , :004BDF3A , :004BDF7C
|:004BDFEE , :004BED31 , :004BEF8E , :004BEFD4 , :004BF0DF
|:004BF3C3 , :004BFE67 , :004BFEBE , :004BFEF7 , :004C08A4
|:004C090C

Bastantes no?

O sea que si hacemos que AL sea cero ya superaremos este primer escollo, que son las limitaciones de tiempo del programa y muchas cosas que no te deja hacer si no estas registrado.

Se puede parchear como lo hice yo que es mas complicado o directamente en el RET del CALL poner XOR AL, AL que hace cero a AL y RET

El programa originalmente es así

```
:00445F94 83C420              add esp, 00000020
:00445F97 C3                  ret

* Referenced by a CALL at Address:
|:004C9359
|
:00445F98 60                  pushad
```

Quedaria asi

```
:00445F94 83C420              add esp, 00000020
:00445F97 32 C0               XOR AL , AL

* Referenced by a CALL at Address:
|:004C9359
|
:00445F98 c3                  RET
```

Yo no lo hice así originalmente porque como no hay lugar para escribir las sentencias, me sobrescribe el CALL siguiente pero probé de esta forma y funciona igual, no afecta aparentemente al otro CALL siguiente.
Sino pueden probar la forma que yo lo parchee en la lista de cambios realizados, es igual, sale siendo AL igual a cero.

Una vez que hacemos esto el programa al arrancar comienza a molestar con carteles distintos, y no arranca bien, los siguientes saltos son para evitar esto:

Nos sale un cartelito de un driver que no esta cargado es esta parte:

```
:0049BE1A 83F801              cmp eax, 00000001
:0049BE1D 0F859B000000        jne 0049BEBE
:0049BE23 B8F0515600          mov eax, 005651F0
:0049BE28 BBFF000000          mov ebx, 000000FF
:0049BE2D E86626FBFF          call 0044E498

* Possible StringData Ref from Data Obj ->"FM2: Warning - Driver not located "
                                        ->"- GPF likely"
                                |
```

Ese cartel aparece y si lo ignoras no arranca bien el programa, parcheando el salto que esta en 49be1d que esta en negrita, superamos este escollo.

Este otro salto

```
:004B254F 7E28          jle 004B2579
:004B2551 83FE00        cmp esi, 00000000
:004B2554 7514          jne 004B256A
:004B2556 B89C125900    mov eax, 0059129C
:004B255B BB13000000    mov ebx, 00000013
:004B2560 E833BFF9FF    call 0044E498
:004B2565 E8261E0000    call 004B4390
```

Era para saltar el CALL 4B4390 que si entra allí si o si caes en un cartelito maldito, dentro del CALL hay para todos los gustos, este es mas difícil de explicar solo que en determinado momento me aparecía un cartelito y para saltearlo, la única forma era no ingresando al CALL y salteándolo desde 4b254f que esta en negrita.

Ahora hay otros saltos como este

```
:004B2579 E826E3F8FF        call 004408A4
:004B257E 8B1DAD195700      mov ebx, dword ptr [005719AD]
:004B2584 39D8              cmp eax, ebx
:004B2586 7E28              jle 004B25B0
:004B2588 83FE00            cmp esi, 00000000
:004B258B 7514              jne 004B25A1
```

- Possible StringData Ref from Data Obj **->"ProductExpired"**

Aqui se ve claro que al volver del CALL 4408a4 si no salta caemos en el cartelito PRODUCT EXPIRED, por lo tanto para saltearlo hay que parchear el salto en negrita.

Estos saltos tiene en común que siempre ocurren como el anterior al retornar del **CALL 4408a4**, ese CALL es de seguridad así que si lo visitamos y vemos las referencias de donde proviene veremos los saltos que nos quedan por parchear.

```
:004B25B0 E8EFE2F8FF        call 004408A4
:004B25B5 8B1DB1195700      mov ebx, dword ptr [005719B1]
:004B25BB 39D8              cmp eax, ebx
:004B25BD 9C                pushfd
:004B25BE 8B54240C          mov edx, dword ptr [esp+0C]
:004B25C2 83E204            and edx, 00000004
:004B25C5 9D                popfd
:004B25C6 0F8DC2010000      jnl 004B278E
:004B25CC B801000000        mov eax, 00000001
:004B25D1 BB01000000        mov ebx, 00000001
```

```
:004B25D6 B92F080000          mov ecx, 0000082F
:004B25DB E894DEF8FF          call 00440474
:004B25E0 3905AD195700        cmp dword ptr [005719AD], eax
:004B25E6 0F8DA2010000        jnl 004B278E
:004B25EC 83FA00              cmp edx, 00000000
:004B25EF 0F8599010000        jne 004B278E
:004B25F5 83FE00              cmp esi, 00000000
:004B25F8 0F8586010000        jne 004B2784
```

*** Possible StringData Ref from Data Obj ->"MessagesThere seems to be an error "**
->"with your PC date. The last time "
->"the program was run was the "
|

Bueno aquí esta el otro cartelito cuando adelantas el reloj de la computadora y usas el programa, cuando lo volvés a la normalidad y usas el programa te salta este cartelito, parcheando el salto en negrita que esta después del CALL 4408a4 lo salteamos.

Este es similar al que parcheamos para saltear el CALL 4b4390, igual que el que parcheamos anteriormente.

```
:004B2968 833D5C1E570000      cmp dword ptr [00571E5C], 00000000
:004B296F 752C                jne 004B299D
:004B2971 8B442408            mov eax, dword ptr [esp+08]
:004B2975 83E002              and eax, 00000002
:004B2978 83F800              cmp eax, 00000000
:004B297B 7520                jne 004B299D
:004B297D 83FE00              cmp esi, 00000000
:004B2980 7514                jne 004B2996
:004B2982 B860145900          mov eax, 00591460
:004B2987 BB0C000000          mov ebx, 0000000C
:004B298C E807BBF9FF          call 0044E498
:004B2991 E8FA190000          call 004B4390
```

Este es idéntico para saltear CALL 4b4390

```
:004B29AE 7420                je 004B29D0
:004B29B0 83FE00              cmp esi, 00000000
:004B29B3 7514                jne 004B29C9
:004B29B5 B86C145900          mov eax, 0059146C
:004B29BA BB0B000000          mov ebx, 0000000B
:004B29BF E8D4BAF9FF          call 0044E498
:004B29C4 E8C7190000          call 004B4390
```

Y este tambien

```
:004B29D8 7531                jne 004B2A0B
```

```
:004B29DA 8B442408          mov eax, dword ptr [esp+08]
:004B29DE 83E008            and eax, 00000008
:004B29E1 83F800            cmp eax, 00000000
:004B29E4 7525              jne 004B2A0B
:004B29E6 83FE00            cmp esi, 00000000
:004B29E9 7514              jne 004B29FF

* Possible StringData Ref from Data Obj ->"FileCorrupted"
                            |
:004B29EB B878145900        mov eax, 00591478
:004B29F0 BB0D000000        mov ebx, 0000000D
:004B29F5 E89EBAF9FF        call 0044E498
:004B29FA E891190000        call 004B4390
```

Este último es de los que viene luego de CALL 4408a4 y saltea un cartelito de que cambio la fecha de la PC, está un poco más abajo no lo pongo aquí por no hacerlo tan largo.

```
:004B278E E811E1F8FF        call 004408A4
:004B2793 2DDA020000        sub eax, 000002DA
:004B2798 8B1DB1195700      mov ebx, dword ptr [005719B1]
:004B279E 39D8              cmp eax, ebx
:004B27A0 0F8EC2010000      jle 004B2968
```

Luego de parchear todos estos saltos el programa arranca perfectamente, como registrado, no cuenta mas el tiempo, no esta limitado, y no expira, por suerte, porque ya de que me aparezcan cartelitos diversos estoy cansado uffff.

Lo que queda son algunas llamadas al mismo CALL 4408A4 que no me saltaron hasta ahora, pero son pocas por lo que parece que ya quedo bien.

Esperemos que la lección 44 sea más breve no?

Ricardo Narvaja

UN MUNDO NUEVO EL P-CODE

(LECCION CUARENTA Y CUATRO)

Hagamos una pequeña introducción al P-CODE, como en todos estos casos les aclaro que yo estoy aprendiendo con ustedes, por lo que disculpen algún error que pueda tener al explicar ya que el P-CODE es un tema difícil, diferente a lo anteriormente visto, y nuevo para mi aunque ya tuve varios fracasos y medios aciertos con el crack del programa exponente máximo del P-CODE y una cuenta pendiente que me queda el CUENTAPASOS, esperemos que podamos ir avanzando con el P-CODE para que podamos también realizar un crack del CUENTAPASOS mas adelante.
Este tutorial como yo estoy aprendiendo esta basado en un tutorial en ingles sobre un crackme que se encuentra en la misma pagina de bajada de la herramienta que utilizaremos para crackear estas pesadillas y que a pesar de todo simplifica bastante el problema del P-CODE a punto de poder analizarlo con tranquilidad, igual yo he hecho algunos cambios sobre el tute en ingles que creo que lo hacen mas fácil para entender.
Bueno primero debemos conseguir el programa a crackear
http://vacarescu.addr.com/WkT/vbdebug/tutos/vbcrackme10.zip

Se baja de allí y por supuesto para los que no lo saben aun todos los programas que se craquearon en el curso así mismo, como este también están alojados en mi Kturn, el que los quiere bajar solo debe ir a **www.kturn.com** y allí registrarse gratis para sacar un Kturn propio te da 125 megas de almacenamiento y es gratis, una vez que ya tiene su Kturn entran y allí van a la opción OTHER USERS y allí teclean **ricnar458** y con eso ya entraron en la carpeta PUBLIC están todos los programas que se crackearon en el curso, obviamente sin crackear para el que quiera practicar, por orden y numerados según la Lección que corresponde.
En
http://vacarescu.addr.com/WkT/vbdebug/downloads.htm
esta El VBDEBUGGER.
Que diferencia hay entre P-CODE y Visual Basic sin P-CODE, bueno eso es algo muy técnico para explicarlo aquí, pero es bueno que sepan que en el P-CODE el ejecutable no se ejecuta nunca, paradójicamente.
El ejecutable en realidad son las librerías de Visual Basic.
Y entonces como sabe la librería de Visual Basic que es lo que tiene que hacer?
Buena Pregunta
Va ejecutando sentencias según los valores de memoria que encuentra en el ejecutable del programa, o sea para aclarar en un ejemplo burdo, el ejecutable del programa es una serie de números sueltos que si los desensamblamos no significan nada ya que podemos probar con el WDASM y no tienen ninguna coherencia, son solo como semáforos para que las librerías de Visual Basic sepan que hacer.
Es complicado verdaderamente.
O sea que si uno quiere crackear esto en la forma tradicional apenas arranca el programa si lo seguimos con el SOFTICE vemos que en la línea verde siempre se esta ejecutando **MSVBVM60.dll** si es en Visual Basic 6.0
y allí esa librería hace todo, y toma decisiones según los valores que toma del ejecutable, pero como valores sueltos de la memoria no como sentencias.

El hecho es que si hallamos por ejemplo en la librería de Visual Basic un salto que deberíamos parchear y hace registrar un programa, no podemos parchear ese dll ni cambiarlo porque Todos los programas de Visual Basic usan esa misma librería y dejarían de funcionar o los afectaría.
La única forma de parchear seria ver que valores del ejecutable hacen a la librería de Visual tomar ciertas decisiones, y cambiarlo allí, pero esto es una verdadera pesadilla y un dolor de cabeza total.
Por eso se hizo este debugger especial para P-CODE, para tratar de facilitar el trabajo a los pobres crackers vagos como yo, jua jua.
Es también de notar que el Smart Check no sirve para programas en P-CODE y sale un cartel diciendo que el programa es P-COMPILADO y que Smart Cheack no puede funcionar correctamente con el o algo así.

Bueno vamos a bajar la herramienta
Se baja de

http://vacarescu.addr.com/WkT/vbdebug/downloads.htm

De allí hay que bajar varias cosas no solo el programa.

De la pagina de bajada:

> This **new** version fixes some loader bugs, and adds a more detailed error message if something goes bad, also fixes some invalid opcodes, and fixes a minor visual dumping error ;-).
> Thanx to oskie for being so nice and helping in finding many bugs.
> **WKTVBDebugger v.1.3 (with English Help)**
> **WKTVBDebugger v.1.3 (with Spanish Help)**
>
> Luego estan los tutoriales de ejemplo
> Also check out this tutorials :
> **Cuentapasos v3.85,Se abrió La lápida (only spanish)**
> **Eternal's Bliss VB CrackMe 10 (only english)**
> Sobre el segundo esta basado este tutorial aunque con algunos cambios.
> Luego para el que no tiene instalado los runtimes de Visual Basic están aquí para bajarlos

Visual Basic Virtual machine files (latest version we have):

> **MSVBVM60.DLL v.6.0.89.64**
> **MSVBVM50.DLL v.5.0.82.44**
> **Estos habría que copiarlos en el c:/WINDOWS/SYSTEM aunque yo prefiero siempre estas cosas bajarlas directamente de Microsoft y completas con instalador y todo.**
> http://download.microsoft.com/download/vb50pro/utility/1/win98/EN-US/Msvbvm50.exe
> http://download.microsoft.com/download/vb60pro/install/6/Win98Me/EN-US/VBRun60.exe

de aquí se bajan completos desde la pagina de Microsoft luego de bajarlos instalarlos y listo.
Luego viene estas filas que se bajan y hay que copiarlas dentro de la carpeta de instalación del programa VBDEBUGGER ,
C:\Archivos de programa\WKTVBDE en la carpeta DBG
Symbolic debug files (DBG) files for use with the debugger:

MSVBVM60.DBG v.6.0
MSVBVM50.DBG v.5.0

Puff ya terminamos lo ultimo es la fila Bdasmdll.dll
la pueden buscar en su maquina, si la encuentran deben dejarla donde esta y copiarla además a la carpeta de instalación del programa también junto con el otro dll que trae el programa WKTVBDE.dll y por supuesto bdasmdll.dll.
El que no la encuentre en su maquina a la fila esta la puede bajar de Internet o bajarla de mi Kturn que allí va a estar.
Bueno esto es todo, cuesta un poco instalar todo esto pero el ahorro de tiempo y la facilidad de uso vale la pena.
Hay en la pagina de donde se baja el programa un FAQ aquí
http://vacarescu.addr.com/WkT/vbdebug/faq.html
por cualquier problema que tengan.
Bueno arrancamos el VBDEBUGGER, y allí vamos a OPEN y buscamos el archivo para crackear, el VBcrackme 1.0.
Este podemos ejecutarlo antes para ver como es, aparece una grilla donde hay que poner una combinación exacta de tildes, entonces vamos a CHECK y si es la combinación correcta nos aparece un cartel de felicitación y si no, no aparece nada.
Lo arrancamos desde el debugger y una vez que cargo vamos a EXECUTE y comienza a funcionar el programa aparecen algunos carteles que debemos aceptar, y si todo esta bien llega el programa a la ventana con la grilla y el botón CHECK.
Cuando hacemos click en Check luego de poner cualquier combinación de tildes, caemos dentro del debugger, el cual puede ser activado igual con la combinación de teclas CTRL+P como si fuera el SOFTICE.
Caemos aquí
004043DC: 28 LitVarI2 0063F360h 0h , 0

Como vemos los OPCODES de P-CODE son diferentes a los que conocemos ya los iremos aprendiendo pero aquí hay uno que importa y es el salto condicional BRANCHF y BRANCHT si presionamos OPCODES nos muestra la lista de comandos existentes aquí y estos dos corresponden a los valores 1C y 1D respectivamente.
Estos saltos equivalen a que salta si es Falso BranchF (false)
y salta si es verdadero BranchT (true)
Estos son las claves de las decisiones que toma el programa.

Hay un botón llamado BRANCH x que nos lista todos los saltos que hay en esta parte que esta listada aquí, que también puede ayudar.

00404404h : BranchF 00404410h
0040442Fh : BranchF 00404442h
00404461h : BranchF 00404474h
00404493h : BranchF 004044A6h
004044C5h : BranchF 004044D8h
004044F7h : BranchF 0040440Ah
00404529h : BranchF 0040443Ch
0040455Bh : BranchF 0040446Eh
0040458Dh : BranchF 004044A0h
004045BFh : BranchF 004044D2h
004045F1h : BranchF 00404404h
00404623h : BranchF 00404436h
00404655h : BranchF 00404468h
00404687h : BranchF 0040449Ah
004046B9h : BranchF 004044CCh
004046EBh : BranchF 00404401h
00404720h : BranchF 00404436h
00404755h : BranchF 0040446Bh
0040478Ah : BranchF 004044A0h
004047BFh : BranchF 004044D5h
004047F4h : BranchF 0040440Ah
00404829h : BranchF 0040443Fh
0040485Eh : BranchF 00404474h
00404893h : BranchF 004044A9h
004048B7h : BranchF 004043E3h

Estos son los saltos y vemos que todos caen adelante del siguiente así que podríamos probar si el último es el que toma la decisión final de si lo resolviste o no.
Pongamos un BPX en 4048b7
Vamos al BOTON breakpoints para poner un Breakpoint allí
por supuesto será BPX o sea al botón ON EXECUTION
escribimos 4048b7 y tecleamos ADD para que lo agregue y luego lo marcamos y vamos a ENABLE para que lo conecte.
Luego vamos al BOTON GO que es para que el programa siga corriendo, vemos que vuelve a parar en ese salto y que algunas sentencias mas abajo esta el cartel YOU HAVE SOLVED, esta listo para saltar como indica el debugger pero si lo dejamos que salte no saldrá el cartel que queremos así que habrá que invertirlo, podemos hacerlo aquí mismo cambiando el 1C por 1D que es BranchT con lo que estaría invertido el salto, aquí pulsamos el botón EDIT y doble cliqueamos encima del 1C que esta arriba de todo y queremos cambiar, nos aparece una ventanita donde cambiamos por 1D y luego vamos a PATCH NOW con lo que habremos cambiado en la memoria este valor. (copiamos la cadena de números que siguen al 1c para luego modificarlos con el Ultra Edit)

Cerramos esa ventana quitamos el BPX con DISABLE y luego vamos a GO y cada vez que ante cualquier combinación ponemos CHECK nos sale el cartel YOU HAVE SOLVED IT.
Por supuesto para cambiarlo definitivamente abrimos el ULTRA EDIT cerramos el debugger y como ya habíamos copiado en la ventana de EDIT la cadena que sigue al 1c que es
1c 07 05 27 e4 fe 27 04 ff 27 24 ff f5 00 00 00
la buscamos aquí cambiamos el 1c por 1d y programa crackeado.

No saben lo que es crackear esta pavada sin este debugger una pesadilla, así que mil gracias a los autores de esta linda herramienta que facilita el trabajo una barbaridad, seguiremos aprendiendo a usarla de a poco en lecciones siguientes.

Ricardo Narvaja

SIGAMOS CON EL P-CODE
(TIEMBLEN CRACKERS "EL CUENTAPASOS")

(LECCION CUARENTA Y CINCO)

La verdad que este programa siempre fue mi clavo, una espina clavada y creo que la de muchos crackers, pude hacer un crack parcial pero tenia algunos problemas, con el nuevo VBdebugger para solucionar el P-CODE puede facilitarse (UN POCO) .
Además contamos con un tutorial sobre este tema en la pagina en la cual se baja el debugger que esta en castellano y esta muy bien, aunque yo no me limito a copiar, lo voy haciendo y según como me va resultando hago lo mismo o no, y trato de facilitar las cosas, si hay algo complicado, en la medida que puedo.
Bueno este tutorial es para el CUENTAPASOS 3.80 REVISION 394 que es el ultimo que salió, se baja de
http://www.cuentapasos.com

Lo instalamos y tomamos el ejecutable cpasos32.exe y lo analizamos con algún analizador de filas, como el un-pack , nos dice que esta comprimido con NEOLITE 2.0 el cual esta incluido en los descompresores que trabaja el PROCDUMP con lo que se puede descomprimir fácilmente.
Sabemos que el debugger no trabaja con programas comprimidos así que ahora que esta descomprimido debería funcionar.
Una vez descomprimido lo copiamos a la carpeta del programa, lo ejecutamos y nos sale el cartel:
"No se ha encontrado el conjunto de opciones de INICIO C\Windows\....."
Bueno dado que como yo tengo el ejecutable descomprimido con el nombre UNPACKED.exe en este ultimo cartel dice C\WINDOWS\Unpacked.ini por lo que quizás este cartel aparezca por tener un nombre distinto el ejecutable cambiémoslo a Cpasos32.exe a ver que pasa.
BUMMM
No sale el cartel anterior pero sale otro que dice "No existe ninguna tarifa , se va a crear una tarifa de ejemplo" , aquí dado que este cartel aparece cuando ejecutamos el programa descomprimido y no cuando ejecutamos el original, puede ser que el programa testee el largo de la fila Cpasos32.exe o el checksum, ya veremos que es la primera opción la que hace aparecer este cartelito.

Bueno, comencemos, carguemos el cuentapasos desde el debugger , hay allí una ventana para poner breakpoints en APIS por lo que abrimos allí y nos aparecen una lista de las funciones utilizadas, veremos si hay alguna función que informe sobre el largo de una fila, que es lo que seguro el programa checkeará, miramos con paciencia ya que son muchas funciones y ponemos BPX en las que tengan algo que ver con largo de filas, nos guiamos por el nombre de las que hay allí hay **rtcFILELen** y **rtcFILELenght** que aparecen tener que ver con el largo de la fila, pongo BPXs en ambas
Voy a GO para que arranque y para en

00**4C57BF**: 5E ImpAdCallI4 **rtcFileLen** on address 0F036192h

Si le doy GO el programa sigue funcionando hasta que sale el cartel de la tarifa de ejemplo y no para mas en ese BPX así que, el bodrio debe estar allí, antes de volver a arrancar el debugger vayamos al ejecutable descomprimido del cuentapasos y fijémonos cual es el largo del mismo haciendo click derecho encima del mismo y viendo las PROPIEDADES, en el mío el largo es 1033728 bytes.
Esto puede variar un poco pero el valor estará cerca de ese.

Luego volvemos al debugger ejecutamos el CALL con F10 y la próxima sentencia es

00**4C57C4**: 99 FMemStI4 004C8590 -> 000FC600h , 1033728

Aquí vemos el numero 1033728 que es el largo del ejecutable y la función FMEMStI4 lo que hace es volcar el contenido del STACK que si vamos al cuadrito de la derecha y cambiamos la opción a DWORD podremos ver que allí esta el valor FC600 que es el hexadecimal de 1033728 y lo guarda en la posición de memoria 4c8590, si vamos a MEMORY DUMP y buscamos que muestre el contenido de la memoria 4c8590 vemos que hay solo ceros ahora al apretar F8 y ejecutar la sentencia se llenara con los valores 00 C6 0f que es fc600 al revés.
Bueno ya vimos que guarda el valor del largo en esa posición de memoria ahora ya que FMEMStI4 la palabra St debe significar STORAGE o sea GUARDAR debe haber entre los opcodes una función opuesta que cargue el valor ese de la memoria al STACK para operar o compararlo.
Si miramos entre los OPCODES

0F0FE145h	93h	FMemLdI2
0F0FE16Ch	**94h**	**FMemLdR4**
0F1056F8h	95h	FMemLdCy
0F10571Ch	96h	FMemLdFPR4
0F10573Dh	97h	FMemLdFPR8
0F0FE1B5h	98h	FMemStI2
0F0FE1DCh	**99h**	**FMemStI4**
0F105789h	9Ah	FMemStR8
0F1057ADh	9Bh	FMemStFPR4
0F1057D8h	9Ch	FMemStFPR8

Allí vemos el OPCODE 99 que es el que corresponde a FMemStI4 la función opuesta parecería ser la 94 FMemLdR4 ya que es FMEM LD parece ser LOAD o sea cargar y R4 en vez de I4 intentemos haciendo doble click sobre esta funciona a ver que pasa.

Parara varias veces allí pero son los valores que carga de distintos lugares de la memoria al STACK por ejemplo para aquí

004C57CE: 94 FMemLdR4 004C8514 -> 00000000h , 0

Que no sirve ya que no es esa posición de memoria ni el valor buscado seguimos haciendo click en GO varias veces hasta que para en

004C5FF4: 94 FMemLdR4 004C8590 -> 000FC600h , 1033728

aquí vemos de nuevo que cuando hagamos F8 para ejecutar esta sentencia va a tomar el valor que había guardado en 4c8590 , el valor fc600 que es el largo de la fila descomprimida y lo va a cargar al STACK, hacemos F8 y en el cuadro superior derecho siempre que este activado DWORD veremos el Fc600 allí,
En 4c5fff carga de nuevo el mismo valor en el STACK, cuando hacemos F8.
Luego lo compara con AAE60 que debe ser el valor que debería tener el ejecutable una vez descomprimido de NEOLITE.
Luego viene el salto condicional que esta después que invertiremos a ver que pasa.

004C600B: 1C BranchF 004C6018 (No Jump)

Cambiando 1C por 1E es cambiar por BRANCH que es equivalente a JMP o sea que salte siempre ya que el programa nos avisaba que no iba a saltar.
Vamos a EDIT reemplazamos el 1C por 1E y volvemos le damos a GO y no sale el CARTELITO mas hemos acertado, podemos volver al mismo lugar para copiar la cadena que hay que reemplazar con el ULTRAEDIT para que el cambio no sea solo en la memoria y se guarde.

1C F0 0B 00 0A 1B 42 00
54 08 00 74 01 00 02 00

Y HAY QUE CAMBIAR EL 1C POR 1E

con eso ya tenemos el primer salto parcheado.

Ahora vamos por la nag o sea ese cartel que dice los días que te quedan y que tiene los botones aceptar, cancelar, etc.
Aquí hay que aplicar el método que usamos a veces en el SOFTICE de ir traceando con F10 hasta que aparezca la NAG, allí anotamos la sentencia que provoco la aparición y volvemos a arrancar el debugger, y ponemos un BPX en esa dirección que hizo aparecer la NAG, entramos ahora con F8 y seguimos con F10 hasta que vuelva a aparecer la NAG y repetimos el proceso anotando cuando apareció poniendo un bpx allí, y luego cuando volvemos a arrancar el programa y para allí otra vez dentro con F8, así llegaremos a uno de los dos lugares en donde aparece la NAG, ya que este Tobarra (el autor) puso dos rutinas gemelas, y a veces a unos le salta la NAG en una y a otros en otra.

ambas son

4BB3B7: 0d VCallHresult meth__imethSHOW

4BB5ED: 0d VCallHresult meth__imethSHOW

ya sabiendo que ambas hacen aparecer la NAG esa buscamos un salto anterior que pueda saltearlas sin que aparezca el maldito cartel.

Esto se puede hacer con el programa EXDEC que es como un WDASM para P-CODE y muestra un listado muerto de todas las sentencias, o se puede poner un BPX un poco anterior a estas sentencias y buscar un salto que las esquive.

los saltos están en

4BB28A: 1c BranchF: 4bb3e4

4BB4E6: 1c BranchF: 4bb61a

Cada uno de estos saltos si cambiamos 1c por 1e se convertirá en un BRANCH o sea como si fuera un JMP y salta sobre los CALL respectivos, haciendo que el programa arranque y no aparezca la NAG y encima no venza, si vamos a ABOUT vemos que siempre esta en el día cero. Bueno una espina menos pongamos las cadenas que hay que parchear en ambos saltos:

1C 2C 04 00 15 04 10 FF
0A 26 00 04 00 04 10 FF

y el otro salto

1C 62 06 00 09 F4 FF 98
08 00 00 00 00 0C F4 FF

En ambos cambiamos 1C por 1E.

La verdad quien pensaría que cambiando solo tres valores estaría vencido este coloso.

Ricardo Narvaja

HAGÁMOSLO CON TRW2000
(TIRO AL PICHON)

(LECCION CUARENTA Y SEIS)

Al final de este tute pongo las claves para ingresar a los nuevos drives donde están los programas de TODAS las lecciones así como las mismas lecciones y las herramientas mas usuales para crackear.
En esta lección crackearemos con TRW2000, utilizaremos este programa en vez del SOFTICE dado que no solamente hemos hecho muchos tutoriales con Softice, sino que además no hay muchos tutoriales con TRW2000 y es bueno saber usarlo, dado que hay veces que el Softice no se puede utilizar y entonces hay que trabajar con TRW2000, así que cada tanto realizaremos un tute para TRW2000 así practicamos.
La victima es un programa llamado SCIENTIFIC NOTEBOOK y el link para bajarlo es
ftp://www.mackichan.com/download/version351/scinoteb351.exe
o de la pagina del programa
http://www.mackichan.com/
donde hay otros dos programas llamados SCIENTIFIC WORKPLACE y SCIENTIFIC WORD que tienen una protección similar lo cual hace que si descubrimos esta habremos crackeado tres en vez de uno.
Yo decidí bajar el SCIENTIFIC NOTEBOOK porque es el que menos pesa (20 megas aprox) los otros son mucho mas pesados (50 megas aprox)
Bueno instalemos el programa y veamos, ejecutémoslo, tomemos como norma siempre hacer una copia del ejecutable original y guardarlo en el escritorio o en algún lugar por si acaso corrompamos sin querer el ejecutable original podamos rescatar el funcionamiento sin tener que instalar todo de nuevo.
Bueno en el programa yendo a HELP-REGISTER y poniendo el punto en "I have and unlock code and want to enter it ", nos aparece la pantalla para ingresar la clave.
Aquí la sorpresa es que además de la clave para registrar el programa y que no expire también hay otros números de serie para registrar el programa para SPELLING DICTIONARYS y COMPUTACIÓN ENGINE o sea que para habilitarlo todo necesitaríamos como 20 claves para los distintos idiomas y para habilitar el resto de las opciones que tiene el programa.
Bueno, ya que avisa allí que la clave tiene que tener doce caracteres pongamos una cualquiera llenemos el cuadro con una clave falsa para cada opción (las tres principales nada mas) a ver que pasa cuando nos queramos registrar.
Luego vamos a SAVE YOUR LICENSE FILE cliqueamos allí y nos dice que para habilitar todas las nuevas opciones tenemos que cerrar y volver a abrir el programa.
Bueno hacemos eso y cuando lo volvemos a abrir es lógico que verifica si el serial es correcto.
Nos aparece un cartelito que parece ser del tipo messageboxa que nos dice LICENSE VERIFICATION ERROR, y después si queremos ver los detalles, luego arranca el programa y si vamos a HELP - SYSTEM FEATURES que es donde muestra las licencias habilitadas vemos que seguimos igual sin licencia.
VIEWER available
ENGLISH installed not licensed

no han cambiado mucho las cosas por aquí.
Si cerramos y volvemos a abrir el programa sin ir a la ventana de registro sigue apareciendo el cartel LICENSE VERIFICATION ERROR.
Bueno abramos el TRW2000 vayamos a BROWSE para buscar el ejecutable que esta en C:\Archivos de programa\Scientific Notebook y se llama scinoteb.exe , luego vamos a LOAD para que comience a ejecutarse.
Rompe en el inicio del programa en el trw2000 y hacemos f10 para ejecutar una sola sentencia y ya estar en el punto de entrada del programa.
Aquí en el TRW2000 ya que los números hexadecimales que corresponden a las sentencias no se muestran tenemos que activarlos a mano tecleando CODE ON con eso ya estaremos mas cerca de lo que estamos acostumbrados a ver en el SOFTICE.
Bueno ya que esa ventanita que dice LICENSE VERIFICATION ERROR parece una messageboxa, pongamos pues un BPX messageboxa para ver si para allí el TRW2000 .
Hagamos X para que corra y para dentro del MESSAGEBOXA.
En la línea donde marca que archivo se esta ejecutando vemos USER32 así que para que termine de ejecutarse esta función de windows y volver al ejecutable, a la sentencia posterior al punto donde entro, tecleamos F12 sale el cartelito maldito de LICENSE ERROR y luego de aceptar en el cartel, ponemos que no queremos ver los detalles y entonces llegamos hasta el RET donde termina esta función MESSAGEBOXA.
Aprieto F10 y salgo a la sentencia posterior a la llamada al messageboxa en 6e8302.
Aquí vemos una diferencia entre el SOFTICE y el TRW2000 si hubiéramos tecleado F12 en el mismo punto en el SOFTICE este nos hubiera depositado directamente aquí en 6e8302, con el trw2000con f12 llega hasta el RET y luego hay que teclear F10 para llegar a la misma posición.
Bueno una vez que llegamos allí a la sentencia posterior anotamos el CALL que corresponde al MESSAGEBOXA que esta en la sentencia anterior o sea 6e82fc.
Supuestamente debería encontrar un salto anterior que permita esquivar esta sentencia y que no se ejecute así que busco con CTRL. + FLECHA DEL CURSOR PARA ARRIBA las sentencias anteriores a 6e82fc algún salto condicional que esquive ese call o sea que salte después de 6e82fc para que no aparezca el cartelito de LICENSE ERROR.
Hay algunos saltos anteriores JZ, JA, JNA, JNZ pero todos saltan antes del call maldito, ninguno lo salta por encima a posiciones de memoria posteriores a 6e82fc, por lo que no nos sirven.
Volvemos a hacer F12 para terminar hasta el RET y luego F10 para volver a la sentencia posterior a la entrada del CALL donde estábamos.
Vemos que caemos en 5b0510 y que la sentencia anterior ahora que hay que esquivar para que no salga el cartelito es la anterior 5b0510 Call 6e824a.
El que quiere probar vuelva a arrancar el programa cuando para en el inicio ponga BPX 5b0510 y va a parar allí y al ejecutar el CALL con F10 aparecerá el cartelito maldito.
Bueno repetimos el proceso y buscamos con CTRL. + FLECHA ARRIBA para buscar algún salto que pase por encima de este CALL.
hay uno solo en 5b048a jz 5b0493que no salta a posiciones de memoria posteriores a 5b0510, por lo que no sirve.
Otro no se ve así que repitamos el proceso y volvamos a hacer F12 y F10 para llegar a la sentencia posterior al Call donde estábamos antes y esto nos hace caer en 6e8353 y el Call anterior que hay que esquivar es 6e834d call near [edx+8c]

Buscamos para arriba algún salto anterior y hay uno en 6e833e jz 6e8355 que saltearía justo el CALL maldito del cartel LICENSE ERROR.
Arranquemos de nuevo el programa y desde el trw2000 y cuando se inicia pongamos un BPX en el salto ese o sea BPX 6e833e.
Cuando para allí vemos que a la derecha del salto dice NO JUMP así que para invertirlo y que salte por encima del CALL maldito hay que hacer que JUMP o sea que salte a 6e8355.
Como EIP marca la próxima sentencia que se va a ejecutar, cambiando el valor de EIP a 6e8355 haremos que salte allí .
En el SOFTICE con poner R eip = 6e8355 ya estaba listo aquí hay que teclear R y enter, con eso se resalta EAX arriba entre los registros, luego con las flechas de cursor se mueve hasta que quede resaltado EIP y ahí tecleamos 6e8355 y listo se invirtió en salto.
El salto al ejecutar F10 pasa por encima del CALL maldito 6e834d pero si sigo tecleando F10 vemos que luego de ya haber esquivado el CALL que queríamos al pasar por encima del otro CALL posterior que esta en 6e8360 aparece también el cartel de LICENSE ERROR, eso quiere decir que el salto ese no sirve y para esquivar los dos carteles juntos debemos seguir hasta el RET con F12 y luego F10 y caer el la sentencia posterior al CALL que contiene los dos carteles, si saltamos ese CALL esquivaremos ambos juntos.
Ahora caímos en 63da13 vemos que el CALL maldito ahora es 63da0e , el que contiene los dos carteles de error dentro y que hay que esquivar.
Busquemos un salto anterior mirando hacia arriba con CTRL. + FLECHA para arriba.
Hay un JMP 63da13 que no es condicional y que saltearía por encima el CALL pero que pasa hay un salto condicional anterior AL JMP que lo pasa por encima y nos tira directo dentro del cartel maldito en 63da0e.
El salto condicional esta en 63d9f3 jz 63da07 ponemos un BPX allí y arrancamos de nuevo el programa, veo cuando para que dice JUMP y que va a pasar por encima del salto que esquiva el cartel, así que tengo que invertir este salto para que en vez de JUMP no salte y siga en la sentencia siguiente, o sea debo cambiar EIP a 63d9f5 así sigue ejecutándose allí.
sigo con F10 y al pasar por 63da00 sale otro cartel de LICENSE ERROR distinto al de 63da0e, por lo tanto el salto no sirve y debemos salir con F12 y F10 hasta la sentencia posterior a este CALL que contiene estos cartelitos para saltearlos a los dos juntos.
Caigo en 63d40a y el CALL ahora a esquivar es la sentencia anterior o sea 63d405 .
Miramos hacia arriba y hay unos cuantos saltos prueben si quieren practicar pero no sirve ninguno ya que no arranca bien el programa (prueben ustedes que pasa)
Vuelvo a apretar F12 y F10 y aparezco ahora en 63dac8 y el CALL a saltear es 63dac3.
Los saltos anteriores tampoco sirven si los invierto.
Vuelvo a hacer F12 y F10 y caigo en 51c41c y el call a esquivar es la sentencia anterior 51c417.Los saltos no hacen nada bueno.
Vuelta F12 y F10 aparezco en 51c1f0 y el call a saltear es 51c1eb.
Hay un salto anterior en 51c1e7 , arranco de nuevo el programa y cuando se inicia pongo un BPX allí.
Cuando para dice que no va a saltar (NO JUMP), si lo invierto de la forma que hicimos las veces anteriores no me sirve porque para como veinte veces en el mismo lugar y cada vez que para debo hace R y cambiar EIP por lo que mejor es cambiar la sentencia directamente una sola vez.
TECLEO A y ENTER cuando estoy encima del salto.
Si te dice que ese comando se puede usar solo en versiones registradas buscate otro TRW2000 bien crackeado.

luego de teclear A y ENTER escribo la nueva sentencia JMP 51c1f8 borro el BPX con BC* y arranco el programa con X.
Bueno por lo menos arranca el programa pero si voy a HELP-SYSTEM FEATURES vemos que las licencias no se han habilitado.
Hay allí varios saltos anteriores para probar pero el correcto es el que esta un poco mas arriba en:
51c195 Jz 51c214
pongo un BPX allí arranco el programa de nuevo.
Copio la cadena de números hexadecimales (ACORDARSE DE CODE ON SI NO NO SE VAN A VER) correspondiente a ese salto y algunos mas que le sigan.

74 7D 8A 55 2D 84 D2 0F 85 16 01 00 00

Sigamos hacemos A y enter, cambiamos por JMP 51c214, la única cifra que cambió en la cadena fue la primera en 74 por EB.

Borramos los BPX y le damos X para ver que pasa.
Sale un cartelito que dice que hay un error en la licencia que tenemos el numero de licencia de otro programa de la serie, aceptamos este, luego aparece otro, aceptamos y arranca el programa,
Si vamos a HELP -SYSTEM FEATURES vemos que el programa habilito TODAS las funciones diccionarios, y engines sale una lista de 30 todas habilitadas con su licencia.
Por lo tanto vemos que esos cartelitos anteriores de que teníamos la licencia de otros programas los pusieron para despistar.
Bueno podemos buscar la cadena con el ULTRA EDIT y cambiar el 74 por EB y hemos hecho la primera parte.
La tarea para el hogar (QUE YO YA HICE) y no es tan difícil como lo que ya se hizo es que ustedes por el mismo método encuentren el salto que complementa al que ya parcheamos y que hace que no salgan estos otros dos cartelitos falsos.
No lloren no hay que hacer tantos F12 como en la lección, unos poquitos no mas, ya saben BPX messageboxa y luego F12 y F10 así repetir el proceso buscando un salto que esquive estos dos carteles .
AYUDA

El salto esta en

_ _ _ _ _ 8 : JZ _ _ _ _ _ E

Completar aquí para saltear los dos carteles.
FIN

Para ingresar a los drives donde están los programas no hay que registrase ni sacar un drive propio ni nada, solo ir a

http://www.oodrive.com/

y entrar en la pestaña VISITEUR y allí poner las siguientes claves:

para el primer drive con los programas que se crackearon en el curso
user: ricnar456
pass: ricnar456

para el segundo drive con mas programas de las lecciones, mas las mismas lecciones y algún tutorial:
user: ricnar457
pass: ricnar457

Y para entrar al tercer drive donde hay herramientas para crackear
user: ricnar001
pass: ricnar458

Ricardo Narvaja

DURO PERO UTIL

(LECCION CUARENTA Y SIETE)

Este programita es como dice el titulo bastante duro para crackear pero es útil para todos aquellos que se la pasan convirtiendo archivos MP3 a WAV y MP3PRO entre otras cositas, además tendremos también el reproductor de MP3PRO la verdad nunca había escuchado nada en este formato nuevo y quede sorprendido al ver que un archivo Mp3 de 2.6 megas al transformarlo en MP3PRO quedo reducido solo a 1 mega y se escuchaba igual o mejor.
Bueno este programita añade una opción cuando hacemos click derecho sobre un MP3 la opción CONVERT y allí ya nos sale si queremos convertirlo a WAV a MP3 o a MP3PRO es muy útil y rápido, muchas veces al intentar grabar un CD el programa nos dice que el archivo esta muy comprimido y tenemos que hacerlo WAV de nuevo y volverlo a hacer Mp3 lo cual significa abrir dos programas cargar el archivo y blablabla, aquí con hacer click con el botón derecho encima del tema ya casi lo estamos convirtiendo.
Bueno adelante el programa se llama dbPoweramp y es gratis una parte, luego se baja un agregado que se llama POWERPACK que vence a los 30 días y es totalmente operativo que además de agregarle muchas opciones permite agregar efectos de sonido y muchas cositas mas.
El programa hay que bajarlo de
http://admin.dbpoweramp.com/

y el link directo para bajarlo es
http://admin.dbpoweramp.com/bin/dMC-r7.exe
el POWERPACK se baja de
http://admin.dbpoweramp.com/bin/dMC-PowerPack.exe
Los codecs para MP3pro se bajan de

http://codecs.dbpoweramp.com/store-alpha/codec-central.htm

user: codecs
pass: dream23

Allí en la pagina hay varios plugins inclusive se bajan aparte los codecs para MP3PRO y el player para MP3PRO se baja de

http://www.codingtechnologies.com/mp3PROzone/assets/mp3PROAudioPlayer_v1_0_2.exe

También hay un plugin para entradas auxiliares para ingresar de cassete o discos y que se yo cuantas cosas mas.

El que no quiere matarse bajando todo por separado en el oodrive de programas esta todo esto dentro de un solo zip, así uno lo baja en una sola vez ya que nos son largos los archivos.

Bueno una vez que bajamos todo y lo instalamos cuidadosamente vemos que hay un archivo que llama POWERPACK STATUS si hacemos click en el nos dirá que no estamos registrados, que nos quedan treinta días y que aun este programa no expiro.

No conviene parchear este programita porque solo nos dice en estado en que esta funcionando, por mas que lo parcheeemos y le hagamos decir que estamos registrados el POWERPACK vencerá a los 30 días así que, este powerpack status nos ayudara para saber el estado de las cosas, no lo parchearemos.

Para ver como trabaja nos posicionaremos encima de un archivo MP3 y haremos click derecho encima y nos posicionaremos encima de la opción CONVERT sin hacer click, entraremos en el SOFTICE y pondremos un BPX GETVERSION a ver que pasa saldremos del SOFTICE y haremos ahora si click en CONVERT, caemos en el SOFTICE y después de cómo 20 F12 mas o menos vemos en la línea verde que llegamos al ejecutable MUSICONVERT ejecutamos un par de sentencias con F10 hasta llegar al RET y salir de allí y caemos en
413b50, donde podremos un BPX para que cada vez que queramos ingresar al ejecutable no tengamos que repetir lo del BPX getversion.

Caemos por aquí:

```
:00413B50   8B15CC624200        mov edx, dword ptr [004262CC]
:00413B56   8B82E0000000        mov eax, dword ptr [edx+000000E0]
:00413B5C   50                  push eax
```

Bueno ya tenemos la forma de ingresar y sabemos que el ejecutable es el archivo MusicConverter.exe así que lo abrimos con el WDASM para mirar un poco.

La siguiente STRING REFERENCE podría ser útil

*** Possible StringData Ref from Data Obj ->"The 'Power Pack' part of dMC has "**
->"now expired, all advanced"

Sin embargo al poner un BPX en los saltos que están arriba de esa STRING

```
:00410623 0F87E2010000          ja 0041080B
:00410629 0F84CB010000          je 004107FA
:0041062F 8BC1                  mov eax, ecx
:00410631 48                    dec eax
:00410632 0F8463010000          je 0041079B
:00410638 83E80E                sub eax, 0000000E
:0041063B 740C                  je 00410649
:0041063D 48                    dec eax
:0041063E 0F85E5010000          jne 00410829
:00410644 E9CF020000            jmp 00410918
```

En ninguno para cuando el programa esta funcionando normalmente sin estar vencido por lo que aquí no es un punto donde decide si esta vencido o no, es probable que eso sea antes pero los autores del programa se cuidaron bastante bien de sugerirnos de donde viene esto, ya que si seguimos hacia arriba para ver si hay alguna REFERENCE anterior para saber desde donde llega hasta aquí vemos que no hay el comienzo de la rutina es:

```
:004105F9 90                nop
:004105FA 90                nop
:004105FB 90                nop
:004105FC 90                nop
:004105FD 90                nop
:004105FE 90                nop
:004105FF 90                nop
:00410600 64A100000000      mov eax, dword ptr fs:[00000000]
:00410606 8B4C2408          mov ecx, dword ptr [esp+08]
```

Y no hay ninguna REFERENCE de donde puede provenir esto.
Aquí los programadores han leído muchos tutes de crackers y saben que a veces uno se guía por las referencias que obtiene en un desensamblador como el WDASM por lo que para ingresar a esta parte llaman a la rutina desde una sentencia como:

CALL ESI o CALL [EAX+05]

o algo así que el WDASM no puede saber que valor tiene ESI o EAX+05 por lo que no puede determinar de donde proviene la llamada a la rutina.
Distinto seria si fuera una llamada numérica como CALL 4105f9 que llamar a la primera sentencia de esta rutina entonces el WDASM nos diría todos los lugares donde se encuentra una llamada así y sabríamos de donde proviene.
Podríamos adelantar el reloj para que venza y llegue hasta esta zona, y después hacer F12 para al salir de esta rutina volver a la sentencia posterior a donde fue llamada.
Yo seguiré por otro camino.
Les comento que si adelantan el reloj para que venza , no aparecerá mas el POWERPACK aunque luego vuelvan a atrasar el reloj de nuevo.
Para desbloquear el POWERPACK y que vuelva a funcionar, (esto se puede ver con el REGMON una vez bloqueado) el programa pone una marca en el registro para que no se pueda utilizar mas.
La clave es :
[HKEY_CURRENT_USER\Software\SCC]

Y hay que ingresar allí el valor 0 para que vuelva a funcionar y por supuesto volver a retrasar el reloj sino volverá a bloquearse.

```
[HKEY_CURRENT_USER\Software\SCC]
"DSTT"=dword:00000000
```

Entre las SRTINGS hay también una REGISTERED pero al igual que la anterior el comienzo de esa larga rutina no tiene ninguna REFERENCE para ver de donde proviene y

al estar desregistrado ningún BPX en esa zona para allí por lo que es una parte que no podremos acceder.

El que quiera registrarlo tiene que estudiar esta parte mas exactamente el CALL 410000 que decide si luego de aparecer la palabra REGISTRADO si vas a YES O NO, miren.

*** Possible StringData Ref from Data Obj ->"Registered: "**

```
                    |
:00409981 BF0C394200            mov edi, 0042390C
..
..
..
..

:004099AB E850660000            call 00410000
:004099B0 83C404                add esp, 00000004
:004099B3 8D942478010000        lea edx, dword ptr [esp+00000178]
:004099BA 84C0                  test al, al
:004099BC 740A                  je 004099C8
```

*** Possible StringData Ref from Data Obj ->"Yes"**

```
                    |
:004099BE BF08394200            mov edi, 00423908
:004099C3 E91E010000            jmp 00409AE6

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004099BC(C)
|
```

*** Possible StringData Ref from Data Obj ->"No   "**

Bueno era posible que modificando dentro del CALL para que salga siendo al=1 nos registráramos ya que eso hace que vayamos a la palabra YES y salteemos el NO, pero alguna comprobación posterior hace que no arranque el POWERPACK así que como no tenemos mas tiempo dejaremos abierta esta vía de crackeo para el que la quiera investigar.
Trataremos de que el programa no venza en vez de registrarnos igual funciona muy bien, así que miremos un poco esto.
Utilicemos el REGMON para ver si encontramos donde guarda la fecha de inicio de los 30 días de plazo que te da, cualquier clave rara que encontremos puede ser, abramos el REGMON y en filtro pongamos MUSIC* y arranquemos haciendo click derecho sobre un archivo MP3 hasta que arranque.
Nos salen varios resultados pero mi ojo avizor sospecha de algunos en particular, veamos:
Esto me resulta muy sospechoso:

```
CloseKey        HKCU\Software\SCC\
OpenKey         HKCU\Software\Illustrate\dBpowerAMP
QueryValueEx            HKCU\Software\Illustrate\dBpowerAMP\dMCPPi
```

CloseKey HKCU\Software\Illustrate\dBpowerAMP
OpenKey HKCU\Software\Illustrate\dBpowerAMP
CloseKey HKCU\Software\Illustrate\dBpowerAMP
OpenKey HKCU\Software\Illustrate\dBpowerAMP
QueryValueEx HKCU\Software\Illustrate\dBpowerAMP\dMCPPi 0 3B 76 26 1A 4B C1 1
CloseKey HKCU\Software\Illustrate\dBpowerAMP

Uno cuando practica bastante por lo menos ya sabe separar la paja del trigo en el caso del REGMON, entre tanta información que da hay que saber descartar las que son claves utilizadas por el sistema de las del programa en particular, bueno esta clave dice dbPOWERAMP, podría ser, veamos si esta en el WDASM por algún lado.

AQUIIIIIII

*** Reference To: KERNEL32.GetSystemTime, Ord:015Dh**

```
                    |
:00410B2E FF152CF14100      Call dword ptr [0041F12C]
:00410B34 8D54240C          lea edx, dword ptr [esp+0C]
:00410B38 8D44241C          lea eax, dword ptr [esp+1C]
:00410B3C 52                push edx
:00410B3D 50                push eax
```

*** Reference To: KERNEL32.SystemTimeToFileTime, Ord:029Bh**

```
                    |
:00410B3E FF1530F14100      Call dword ptr [0041F130]
```

*** Possible StringData Ref from Data Obj ->"Software\Illustrate\dBpowerAMP"**

```
                    |
:00410B44 68242A4200        push 00422A24
:00410B49 6801000080        push 80000001
```

- **Possible StringData Ref from Data Obj ->"dMCPPi"**

Mas sospechoso imposible justo es abierta esa clave del registro después de hacer GETSYSTEMTIME o sea después de que el programa se entera de la fecha que es, quizás esta cifra que esta en dmcppi sea la fecha inicial de prueba del programa pero encriptada.
Hagamos una prueba, adelantemos el reloj un año y antes de ejecutar el programa, vayamos a HKEY_CURRENT_USER\Software\Illustrate\dBpowerAMP

y marquemos DMCPPi y exportemos la clave al escritorio o algún lado seguro o sea luego de marcarla, REGISTRO - EXPORTAR ARCHIVO DEL REGISTRO, para poder jugar con esta clave a ver que pasa y si nos equivocamos poder restaurarla como estaba antes.
Ahora luego de haberla exportado borrémosla.
Ejecutemos el programita que nos dice el STATUS de cuantos días nos faltan para que venza POWERPACK STATUS.

DICE QUE NOS QUEDAN 30 días para probar jua jua, y si vamos a ver y cerramos y volvemos a abrir la misma clave vemos que creo otra diferente con la que debe ser la fecha actual encriptada a partir de la cual nos da de nuevo 30 días.
Si volvemos de nuevo el reloj hacia atrás tenemos que borrar esta clave de nuevo para que nos cree de nuevo la clave actual porque si lo ejecutamos fuera de fecha vencerá y se bloqueara para lo cual habrá que buscar para desbloquearlo la otra clave que hallamos al principio del tute.
Bueno la cuestión es que el programa busca la clave esta y si no esta, crea una nueva, podemos borrar la clave vieja y poner un BPX en REGCREATEKEYEXA para ver donde crea la nueva clave.

Para en

*** Reference To: ADVAPI32.RegCreateKeyExA, Ord:015Fh**

```
                    |
:00418078 FF151CF04100          Call dword ptr [0041F01C]
:0041807E 85C0                  test eax, eax
:00418080 7529                  jne 004180AB
:00418082 8B44240C              mov eax, dword ptr [esp+0C]
:00418086 8B4C2408              mov ecx, dword ptr [esp+08]
:0041808A 8B542404              mov edx, dword ptr [esp+04]
:0041808E 50                    push eax
:0041808F 8B442418              mov eax, dword ptr [esp+18]
:00418093 51                    push ecx
:00418094 6A03                  push 00000003
:00418096 6A00                  push 00000000
:00418098 52                    push edx
:00418099 50                    push eax
```

- **Reference To: ADVAPI32.RegSetValueExA, Ord:0186h**

Allí crea la nueva clave y con SET VALUE le da un valor nuevo.

Hagamos F12 para ver de donde viene esta llamada a este CALL ya que en el WDASM vemos que puede provenir de diferentes lugares.

Caemos en el mismo lugar que habíamos visto antes

*** Possible StringData Ref from Data Obj ->"dMCPPi"**

```
                    |
:00410B70 68C8384200            push 004238C8
:00410B75 E8C6740000            call 00418040
```

Ahora veamos un poco mas arriba hay un salto condicional que puede evitar que llegue a crear una clave nueva o sea que eso debe ocurrir normalmente cuando el programa encuentra la clave, y si no la encuentra crea una nueva, lo que podemos hacer nosotros es

parchear ese salto para que siempre este o no este la clave SIEMPRE caiga donde crea una nueva y cada vez que se ejecute el programa nos de 30 días mas, jeje.
Aquí vemos el salto a parchear

:00410B5D 751E jne 00410B7D

***** Possible StringData Ref from Data Obj ->"Software\Illustrate\dBpowerAMP"
|
:00410B5F 68242A4200 push 00422A24
:00410B64 6801000080 push 80000001
:00410B69 8D4C2414 lea ecx, dword ptr [esp+14]
:00410B6D 6A08 push 00000008
:00410B6F 51 push ecx

* Possible StringData Ref from Data Obj ->"dMCPPi"
|
:00410B70 68C8384200 push 004238C8
:00410B75 E8C6740000 call 00418040

Podemos probar poner un BPX 410b5d y veremos que cuando para allí siempre quiere saltar a no sea que hayamos borrado la clave donde guarda la fecha de inicio.
Por lo tanto cambiemos esos dos números del salto por 90 90 y no saltara nunca y siempre CAERA EN EL CALL donde esta createkey tendremos 30 días mas de uso cada vez que lo ejecutemos y nunca vencerá. La cadena a modificar en el ULTRA EDIT es:

75 1E 68 24 2A 42 00 68 01 00 00 80

Y debemos cambiar **75 1E POR 90 90**

Ricardo Narvaja

BUSCANDO CLAVES UN POCO MAS DIFÍCILES

(LEECION CUARENTA Y OCHO)

En esta oportunidad buscaremos la clave de un programa que es un poco bastante mas difícil que lo normal, quizás no llegue al nivel del encriptamiento furioso, ni nada que ver ya que la rutina de comparación es simple.
El problema es que pone un montón de partes para despistar como si estuviera encriptando, y yo llene como seis hojas de paginas con un encriptamiento de la clave que era falso, que era solo para despistar, por lo tanto en casos de claves que hay mucho que perseguir la táctica es desde que ponemos la clave falsa y caemos dentro del HMEMCPY hasta que llega al cartel de que la clave es incorrecta, conviene hacer una primera pasada superficial luego de poner los BPRs donde esta en la memoria la clave falsa, y anotar todos los lugares donde va parando y mirando un poco, sin seguir por primera vez por caminos donde empieza a hacer operaciones con los números de la clave falsa.
En mis hojas de anotaciones hay un montón de lugares donde paraba y tomaba uno o varios números de la clave y los empezaba a sumar y a operar con otros inclusive hay un ciclo terrible de que va tomando cifras le hace mil cuentas las suma divide multiplica etc etc, las guarda y sigue, y al final es una farsa completamente inútil, así que yo voy a hacer un breve repaso de los lugares mas importantes donde para sin operar demasiado hasta que llega a la rutinita de comparación, y suprimiré muchas de las partes puestas para despistar, que si uno anota donde comienzan y sigue adelante sin prestar atención, llega a la rutina de comparación perfectamente, y si uno no la encuentra entonces si vuelve atrás y comienza a revisar con mas cuidado lo que paso de largo.
El programa a analizar es el
http://ftp.idg.se/programbanken/pc/grafik/snag3243.exe

y no es la ultima versión que hay aunque no ha cambiado mucho en cuanto a la forma de hallar la clave, creo que las mismas rutinas esta pero en distintos lugares en la versión nueva, hice una anterior porque un amigazo me pidió que había tratado con esta versión y que lo ayude y entonces me puse a hacer esta.
Ya saben que los Oodrives que había puesto para guardar los programas que se crackean parece que van a comenzar a cobrar, hasta ahora están allí no se por cuanto tiempo mas , pero también están los kturns en los cuales cualquiera yendo a http://www.kturn.com/ y sacándose un kturn propio que es gratuito, te da 125 megas y puede desde el suyo acceder a los míos.
Entra al propio kturn va a donde dice OTHER USERS y allí pone cualquiera de estas tres claves :
ricnar458
ricnar457
ricnar456

que son los tres diferentes kturns que hay del curso con los programas, herramientas, y todo lo necesario.
Bueno sigamos adelante.
Lo primero que hay que hacer luego de instalarlo es localizar donde guarda en la memoria la clave falsa que yo introduzco, para eso veamos primero como se hace en SOFTICE y

después como se podría hacer en TRW2000. Pongo la clave falsa 9898989898.
Una vez que para dentro del HMEMCPY hay que tracear hasta que lleguemos a una sentencia Repz movsd en 015f:9f20, allí hago d es:di
que es donde va a guardar la clave falsa.,sigo con F10 hasta unas sentencias mas abajo hasta repz movsb y al ejecutarlo se termina de copiar la clave falsa a la memoria si no se ve hay que subir una línea en la ventana de datos allí veo en la parte del contenido
39 38 39 38 ....
que son los números hexa que corresponden a 989898... y a la derecha
989898...
Ahora tecleo PAGE mas la dirección de memoria donde esta la primera cifra de la clave, esto puede variar en la mía es **1197:0000**
por lo tanto hago
PAGE 1197:0000

y me sale como resultado

LINEAR
6ff450
que es la posición de memoria de 32 bits donde esta la clave

hago
d 0030:6ff450
y allí esta la clave falsa ya en la memoria.

Esto localiza donde guardo la clave en el SOFTICE ahora en el TRW2000 la función PAGE existe pero no funciona igual, no da el mismo resultado y con el método anterior lo único que logre es que la ventana de datos no muestre mas nada y quede enganchada en modo 16 bits (PROT 16) y tenga que cerrar el TRW2000 y volver a abrirlo para que muestre de vuelta en 32 bits (PROT 32)

Quizás la mejor manera en TRw2000 cuando caes en HMEMCOPY seria seguir traceando con F10 hasta volver al ejecutable o sea que ya estamos en 32 bits y allí hacer una búsqueda. (esto sirve también en el SOFTICE igual)

S 0 L FFFFFFFF '9898989898'

'Las comillas deben ser estas **'** no estas **"** y después de la S va un cero.

Allí aparecerá donde esta la clave falsa, puedo teclear nuevamente S y enter para que busque a continuación a ver si aparece en algún otro lado pero es poco probable

Bueno en el SOFTICE ahora habría que poner un breakpoint que abarque todas las cifras de la clave falsa cosa de que cuando el programa lea o haga algo con alguna cifra pare allí la sentencia para SOFTICE seria.

BPR INICIO FINAL rw

aquí seria

BPR 6ff450 6ff460 rw (SOFTICE)

Ya que la clave abarca desde 6ff450 hasta 6ff460 entonces con este breakpoint cada vez que lea o escriba allí parara.
Lamentablemente (Y MUCHO) en el TRW2000 no existen breakpoints de rango solo existen BPMs individuales o sea poner un **BPM 6ff450 rw** haría que parara allí cuando lea o escriba algo solo en la primera cifra de la clave.
Encima por la arquitectura del BPM diferente al BPR no para justo cuando lee o escribe de 6ff450 sino parara justo la sentencia posterior así que cuando pongamos un BPM y pare siempre hay que mirar la sentencia anterior a la que paro. Primero borramos con BC* el BPX HMEMCPY.
Bueno que vamos a hacer paciencia si usamos TRW2000 paciencia pongamos un BPM 6ff450 rw BPM 6ff451 rw y
así sucesivamente en todas las cifras.
Luego de hacer eso hay que hacer X y ver cuando el programa trata de hacer algo con la clave.
Aquí entonces habría que ir analizando cada vez que el programa trata de hacer algo con la clave, y si la trata de copiar a otra dirección volver a poner BPRs en los lugares donde la va copiando o guardando para que siempre el SOFTICE pare cuando trate de hacer algo con las cifras de la clave.
Bueno aquí vamos a empezar a bailar de a poquito, yo me olvide pero tenemos que cambiar la clave falsa por otra mas practica en vez de 989898... que es la que a mi me gusta aquí vamos a usar otra 123456789ABCDE que nos da la ventaja de saber fácilmente por el valor que cifra esta trabajando ya que en la otra si trabaja con un nueve no sabemos si es el primero el tercero, el quinto o cualquiera de los otros así que repitamos todo de nuevo con la clave **123456789ABCDE**.
Bueno ya tenemos todo listo para empezar y traceamos un poco antes de empezar a darle a X y en la comparación de 418850 vemos que si la clave que poníamos era menor que 14 cifras nos tiraba al tacho ya que ebp tiene el numero de cifras y 0E es 14 en decimal. Como la que pusimos es de 14 cifras no salta y sigue dentro de la comparación.

:**00418850** 83FD0E **cmp ebp, 0000000E**
:00418853 0F8229010000 jb 00418982

le damos a que siga con X y caemos en esta parte

:00449898 41 inc ecx
:00449899 8A06 mov al, byte ptr [esi]
:0044989B 0AC0 or al, al
:0044989D 7407 je 004498A6
:0044989F 46 inc esi
:004498A0 0FA30424 bt dword ptr [esp], eax
:004498A4 72F2 jb 00449898

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044989D(C)
|
:004498A6 8BC1                mov eax, ecx
```

Bueno parece no servir para mucho toma en 449899 la primera cifra le aplica la función OR queda igual que antes luego incrementa el contador del bucle y aplica BT que la verdad es la primera vez que veo BT, en 4498a0 pero no parece hacer nada quizás sea una comprobación inútil ya que BT parece no cambiar ningún registro ni el STACK, ni posición de memoria, por ahora lo dejamos ya que no parece útil, cualquier cosa volveremos por aquí.

Llegamos una vez que termina con todas las cifras a 4498a6.

Bueno adelante seguimos y caemos en 41887a.

```
:0041887A 80382D              cmp byte ptr [eax], 2D
:0041887D 7503                jne 00418882
:0041887F 40                  inc eax
:00418880 EBF8                jmp 0041887A

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041887D(C)
|
:00418882 8A10                mov dl, byte ptr [eax]
:00418884 8811                mov byte ptr [ecx], dl
:00418886 41                  inc ecx
:00418887 40                  inc eax
:00418888 3BCD                cmp ecx, ebp
:0041888A 72EE                jb 0041887A

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00418878(C)
|
:0041888C 802100              and byte ptr [ecx], 00
```

Aquí en 41887a compara la primera cifra con 2d que es el guión - , si no es guión salta el subsiguiente salto que llega a 4188d2, allí carga esa cifra a DL y la copia en el contenido de ECX o sea en el mismo lugar que estaba. Hace eso cifra por cifra, bueno si en 41887a encuentra un guión directamente no lo va a guardar y pasa a la cifra siguiente con lo que transforma una clave con guión en una completamente limpia sin guión. Vaya a saber porque hace esto, bueno lo tendremos en cuenta.

Bueno adelante
Caemos en bff7117e REPNZ SCASB y que demonios es esto.

la experiencia del que crackeo mucho sabe que esta sentencia se encuentra dentro de la función lsrtlen que es para sacar la cantidad de cifras nuevamente.
Una vez que traceando con F10 llegamos al ejecutable en EAX quedo la cantidad de cifras de la clave de la memoria.
Eax tiene 14 en hexa es 0e.
Estamos en 435cA3
:00435CA3 50 push eax

Y dado que va a hacer PUSH EAX justo que EAX tiene el valor de la cantidad de cifras y lo va a guardar veamos como es el mecanismo y donde lo guarda con la sentencia PUSH.

La sentencia PUSH guarda en el STACK que es como un archivo que no deja de ser parte de la memoria.
El registro ESP que quiere decir STACK POINTER es el que apunta a donde esta el stack y cual es el valor que dado que el stack es una pila de valores, apunta al valor que esta encima de la pila.
al hacer PUSH EAX agregamos un valor al stack y si vemos con D ESP allí en 6feec0 esta como primer numerito el 0e que guardo luego de ejecutar la sentencia PUSH EAX.
Bueno ya que esta allí guardado por algo (???) pongamos un BPR allí en el SOFTICE o un BPM en el TRw2000 por si alguna vez lo busca de allí para realizar alguna comparación.
BPR 6feec0 6feec0 rw (softice)
BPM 6feeco rw (trw2000)

Ahora para en 448a48 y trabaja con el valor 0e ese que estaba guardado en el stack pero no se ve nada importante por aquí
Luego de terminar allí para en 435d72
donde hace un PUSH y borra el valor de la cantidad de cifras que había guardado en 6feec0 por lo que ya puedo borrar ese BPR o BPM de allí.
Para ahora en 448b5c y empieza la joda.

```
:00448B5C 8B448EF4          mov eax, dword ptr [esi+4*ecx-0C]
:00448B60 89448FF4          mov dword ptr [edi+4*ecx-0C], eax
:00448B64 8B448EF8          mov eax, dword ptr [esi+4*ecx-08]
:00448B68 89448FF8          mov dword ptr [edi+4*ecx-08], eax
:00448B6C 8B448EFC          mov eax, dword ptr [esi+4*ecx-04]
:00448B70 89448FFC          mov dword ptr [edi+4*ecx-04], eax
:00448B74 8D048D00000000    lea eax, dword ptr [4*ecx+00000000]
:00448B7B 03F0               add esi, eax
:00448B7D 03F8               add edi, eax
:00448B7F FF2495888B4400    jmp dword ptr [4*edx+00448B88]
```

Aquí en la primera línea copia a EAX las cuatro primeras cifras al revés 34333231 o sea 4321 y en la siguiente las guarda en 6fef04, y va armando

allí la clave.
Luego toma las siguientes y las copia a continuación y las próximas también
o sea que en 6fef04 queda

123456789ABC

Copia hasta allí y llega al salto en 448b7f cada vez que la clave es copiada
a otro lugar hay que poner un BPR o BPMs que abarquen el nuevo lugar donde
esta la clave sin borrar los bPRs o BPMs anteriores.

Entonces pongo
bpr 6fef04 6fef13 rw (softice)

Bpm 6fef04 rw ..... hasta bpm 6fef13 rw (trw2000)

y sigo con X

Ahora para en 448bac para terminar de copiar toda la clave al nuevo lugar

```
:00448BAC 8A06          mov al, byte ptr [esi]
:00448BAE 8807          mov byte ptr [edi], al
:00448BB0 8A4601        mov al, byte ptr [esi+01]
:00448BB3 884701        mov byte ptr [edi+01], al
:00448BB6 8B4508        mov eax, dword ptr [ebp+08]
:00448BB9 5E            pop esi
:00448BBA 5F            pop edi
:00448BBB C9            leave
:00448BBC C3            ret
```

Mueve la primera cifra a AL y lego la copia a [EDI] que es a continuación de
donde esta guardando la clave
, sigue con la ultima cifra y llega al RET con la clave toda copiada en
6fef04.
No olvidar poner los BPR o BPMs allí seguimos.

para en
```
:0042BD8A 8A0408        mov al, byte ptr [eax+ecx]
:0042BD8D 50            push eax
```

All toma el valor de la letra D o sea 44 y lo mueve a AL luego lo hace PUSH
y lo manda al STACK así que luego de ejecutar el PUSH hago
D ESP para ver donde lo guardo

Esp es 6fee14 , y allí guardo el 44

Pongo BPM 6fee14 rw o BPR 6fee14 6fee14 rw (se entiende SOFTICE o TRW2000

ya no lo voy a repetir mas).

Este proceso va a ser repetitivo para cada cifra.

Para en 42bdd6

```
:0042BDD6 8A442404          mov al, byte ptr [esp+04]
:0042BDDA 3C61              cmp al, 61
:0042BDDC 720B              jb 0042BDE9
:0042BDDE 3C7A              cmp al, 7A
:0042BDE0 7707              ja 0042BDE9
:0042BDE2 0FB6C0            movzx eax, al
:0042BDE5 83E820            sub eax, 00000020
:0042BDE8 C3                ret
```

Este bucle es para pasar si hay minúsculas a mayúsculas brevemente compara con 61 (letra **a** minúscula) y si es mayor por ejemplo 64 que es d minúscula le resta 20 para hacerlo 44 que es D mayúscula, eso es todo.

Luego viene una parte importante

```
:0042BD97 663D3000          cmp ax, 0030
:0042BD9B 59                pop ecx
:0042BD9C 720D              jb 0042BDAB
:0042BD9E 663D3900          cmp ax, 0039
:0042BDA2 7707              ja 0042BDAB
:0042BDA4 05D0FF0000        add eax, 0000FFD0
:0042BDA9 EB11              jmp 0042BDBC

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042BD9C(C), :0042BDA2(C)
|
:0042BDAB 663D4100          cmp ax, 0041
:0042BDAF 7220              jb 0042BDD1
:0042BDB1 663D4600          cmp ax, 0046
:0042BDB5 771A              ja 0042BDD1
:0042BDB7 05C9FF0000        add eax, 0000FFC9

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042BDA9(U)
|
:0042BDBC 0FB7C0            movzx eax, ax
:0042BDBF C1E704            shl edi, 04
:0042BDC2 0BF8              or edi, eax
:0042BDC4 46                inc esi
:0042BDC5 663B742410        cmp si, word ptr [esp+10]
:0042BDCA 72B7              jb 0042BD83
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042BD81(C)
|
:0042BDCC 8BC7                  mov eax, edi
```

El que quiera saltear esta partecita sepa que esta hecha solo para despistar.
Bueno aquí compara el valor si esta entre 0 y 9 primero si esta entre 0 y 9 esta bien sino pasa a la parte de letras y compara si esta entre A (41)y F (46) esta bien sino parece que te tira al tacho.
O sea que solo acepta letras pero que signifiquen números en HEXA porque?
Veamos
Luego veo que en eax me quedo la misma letra o sea EAX=D y luego de hacer todo el mismo proceso con la letra E en 42bdcc eax es DE o sea en letras las dos ultimas cifras de la clave.
EAX=DE
Entonces por eso no acepta letras mayores que F ya que va a operar directamente con las letras y no puede hacer eso si la letra es G ya que no es un valor hexadecimal que pueda almacenar en algun lado.

Luego en 42be30

```
:0042BE30 8945FC                mov dword ptr [ebp-04], eax
```

Mueve a [Ebp-04] el valor DE que estaba en EAX.

Ebp-04 alli es 6feeb0 asi que pongamos un BPM o BPR alli
Bpr 6feeb0 6feec0 rw o BPM 6feec0 rw hasta 6feec5 por lo menos.

Para ahora en

```
:00448BAC 8A06                  mov al, byte ptr [esi]
:00448BAE 8807                  mov byte ptr [edi], al
:00448BB0 8A4601                mov al, byte ptr [esi+01]
:00448BB3 884701                mov byte ptr [edi+01], al
:00448BB6 8B4508                mov eax, dword ptr [ebp+08]
```

Donde toma el DE y lo mueve a AL y luego lo copia a [EDI], o sea que ahora DE también está en 6feed6

BPR o BPM allí

Ufff sigamos

Ahora repite el proceso con todas las cifras pero lo voy a obviar porque la verdad no sirve para nada, esta solo para despistar.

UN MEDICO PARA EL CRACKER POR FAVOR jua jua

pongo resaltado lo que sigue ya que es la rutina de comparación:

Uff la verdad es que es una rutinita resimple la que compara las cifras de
la clave del SNAGIT pero da tantas vueltas y desorienta tanto que a veces uno
se pierde. si vieran la cantidad de hojas que llene para llegar al final a
una simple comparación jua jua.
Lo que pasa es que con esto de perseguir claves yo cometí un error, hay que
hacer una mirada hasta el final de todos los lugares donde va parando tipo
superficial y luego si, si uno no encuentra alguna parte como esta que vamos
a ver se pone a fondo a buscar la clave.
La rutina de comparación es esta:

:0042BE99 8A0438 mov al, byte ptr [eax+edi]
:0042BE9C 50 push eax
:0042BE9D E834FFFFFF call 0042BDD6
:0042BEA2 59 pop ecx
:0042BEA3 8A4C35DC mov cl, byte ptr [ebp+esi-24]
:0042BEA7 83E10F and ecx, 0000000F
:0042BEAA 38440DEC cmp byte ptr [ebp+ecx-14], al
:0042BEAE 7510 jne 0042BEC0

En 42be99 para ya que allí había un BPR porque en EAX+EDI esta la clave
falsa.
Pasa a AL la primera cifra.
en 42beaa la compara con la primera cifra verdadera, en el SOFTICE aparece
arriba a la derecha en el TRW2000 hay que hacer
d ebp+ecx-14 , en mi caso en el primer
lugar esta el 34 que corresponde al 4, al no ser igual ya que mi primera
cifra de la clave falsa es uno ya me va a tirar fuera por lo que borro todos los BPs con BC* y
pongo un BPX allí en la comparación bpx 42beaa y salgo hasta el cartelito de
mala clave.
Cambio la primera cifra por la que encontré verdadera y la primera vez no me tira
afuera (ya que la cambie, jua jua) , la segunda vez que para vuelvo a hacer d ebp+ecx-14 y me fijo la
segunda cifra, la anoto que en mi caso es 31 o sea 1 y salgo de vuelta la
cambio y vuelvo a entrar así hasta que descubro que mi clave es 415387b4
esas son las cifras que testea a partir de allí las siguientes son de
relleno hasta tener las 14 cifras por ejemplo.
415387b4ffffffff
Esa seria mi clave o podría cambiar las f por otra cosa igual me registra ya
que solo testea las 8 primeras.

Ahí ven mi nombre de usuario Chancho y mi clave 415387B4..... LAS SIGUIENTES CIFRAS SOLO ESTAN PARA RELLENAR HASTA EL NUMERO 14 LA CANTIDAD DE CIFRAS.
Ah me olvidaba de decir que para que se habilite la opción de ingresar la clave que esta desactivada primero hay que ir a donde se ingresan los datos de nombre de usuario y compañía y recién allí se habilita el botón ENTER KEY para ingresar la clave.

Ricardo Narvaja

RELOJ NO MARQUES LAS HORASSSSSS

(LECCION CUARENTA Y NUEVE)

Este título se refiere a que hay programas que funcionan full durante el periodo de prueba y que uno cuando mira un poquito ve que buscar registrarlos es complicado entonces busca otra alternativa que funciona para mi, y como yo soy cracker pero a veces un poquito perezoso, siempre trato de buscar la mas fácil que haga que el programa funcione.
El programa en cuestión es el Microangelo 5 es para editar iconos y todas esas cosas, aquí dice lo que hace.

What do you want to do?

Improve my computer's icon display How?

Browse, view, and search for icons. How?

Change icons and cursors on my computer. How?

Organize icons in libraries. How?

Edit icons and cursors. How?

Bueno eso hace es un editor de iconos, y mucho mas.
La versión que estoy crackeando es la 5.02 y se baja de
http://www.impactsoft.com/

Pero me parece que ya salió allí la versión 5.5 bueno en el Kturn subiré esta la 5.02 aunque no debe haber mucha diferencia en la forma de crackearlo.
En principio cuando realizamos algún cambio en cualquier archivo nos sale un cartelito que dice que hay un problema con el muapp.dll y que debemos instalar de nuevo el programa.
En esta protección que es un cartelito tipo messageboxa , no podemos poner un **Bpx messageboxa** y una vez que para, fijarnos nada pues esta hecho para que una vez que sale el messageboxa no vuelve nunca mas al programa y lo cierra, por lo que no podemos volver al ejecutable.

Para ir de a poco dejemos todo como esta original y cambiemos algún valor que no haga que deje de funcionar obviamente en ese muapp.dll, puede ser las características de las secciones o cualquier valor que no importe mucho solo es para probar.
Reemplazamos el muapp.dll por el modificado, corremos el programa y zas, otra vez ese cartel maléfico.
Bueno, habrá que tracear hasta ver cuando aparece, antes de ejecutar el programa de nuevo uso un BPX getversion y ejecuto, cuando para vuelvo con F12 al ejecutable y comienzo a tracear y tracear y tracear.

La protección para cuando realizamos cambios solo en el muapp.dll esta aquí.

```
:6400137E FF1578800064          Call dword ptr [64008078]
:64001384 A140A00064            mov eax, dword ptr [6400A040]
:64001389 8B542414              mov edx, dword ptr [esp+14]
:6400138D 3BD0                  cmp edx, eax
:6400138F 740F                  je 640013A0
:64001391 5F                    pop edi
:64001392 5E                    pop esi
:64001393 5D                    pop ebp
:64001394 33C0                  xor eax, eax
:64001396 5B                    pop ebx
:64001397 81C4B8020000          add esp, 000002B8
:6400139D C20C00                ret 000C
```

Yo compare además con el muapp.dll sin modificar cuando no aparece el cartelito y allí el programa llega a 6400138d y salta, en cambio el modificado, llega a ese salto y no salta sigue hasta el RET donde una vez que llega allí va a funciones de windows sale el cartelito y se cierra el programa y nunca mas vuelve.
Por lo tanto lo que hay que hacer primero es cambiar ese salto condicional a JMP.

6400138f 74 0f hay que cambiar el 74 por EB y listo ya arranca igual aunque tenga cambios el dll.

La siguiente parte de la protección es cuando realizamos un cambio en alguna de las aplicaciones como Librarian o Muapanel, nos aparece un cartelito similar al que parcheamos recién pero en otro lugar mas adelante.
Aquí hay dos RET y dos saltos.

```
:640015A7 7523                  jne 640015CC

* Possible Reference to String Resource ID=00001: "I &Agree"
                        |
:640015A9 B801000000            mov eax, 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:64001569(C), :64001580(U), :64001597(U)
|
:640015AE 8B542414              mov edx, dword ptr [esp+14]
:640015B2 8B0C8530A00064        mov ecx, dword ptr [4*eax+6400A030]
:640015B9 3BD1                  cmp edx, ecx
:640015BB 743D                  je 640015FA
:640015BD 5F                    pop edi
:640015BE 5E                    pop esi
:640015BF 5D                    pop ebp
```

```
:640015C0 33C0                  xor eax, eax
:640015C2 5B                    pop ebx
:640015C3 81C4B8020000          add esp, 000002B8
:640015C9 C20C00                ret 000C
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:640015A7(C)
|
:640015CC 8D442430              lea eax, dword ptr [esp+30]

* Possible StringData Ref from Data Obj ->"engineer.exe"
                        |
:640015D0 6814A10064            push 6400A114
:640015D5 50                    push eax
:640015D6 FFD6                  call esi
:640015D8 F7D8                  neg eax
:640015DA 5F                    pop edi
:640015DB 5E                    pop esi
:640015DC 1BC0                  sbb eax, eax
:640015DE 5D                    pop ebp
:640015DF 40                    inc eax
:640015E0 5B                    pop ebx
:640015E1 81C4B8020000          add esp, 000002B8
:640015E7 C20C00                ret 000C
```

Bueno aquí es igual en cualquiera de los dos RET que cae aparece el cartelito, así que entonces en el primer salto no debe saltar o sea hay que nopearlo entonces cae en el segundo que si salta esquiva los dos RET.
Entonces nopeamos el primero y hacemos JMP el segundo y adiós protección podemos modificar lo que querramos que funciona igual.

640015a7 75 23 cambiar por 90 90 y

640015bb 74 3d cambiar por eb 3d .

Ahora como ya dijimos vamos al funcionamiento, miremos un poco el LIBRARIAN que es una de las utilidades que trae.
Sabemos que las funciones para que el programa se entere de la fecha son GETLOCALTIME o GETSYSTEMTIME.
Si cargamos el Librarian en el Wdasm y vamos a **IMP FN** el botón que muestra las funciones importadas, vemos que GETLOCALTIME y GETSYSTEMTIME solo una vez dentro del programa son llamadas en.

```
* Reference To: KERNEL32.GetLocalTime, Ord:012Fh
                        |
```

```
:004157AA FF15ACB14100          Call dword ptr [0041B1AC]
:004157B0 8D45E0                lea eax, dword ptr [ebp-20]
:004157B3 50                    push eax
```

*** Reference To: KERNEL32.GetSystemTime, Ord:0174h**

O sea que si recorremos un poco poniendo un BPX allí y entramos dentro de la función GETLOCALTIME veremos que guarda la fecha en este caso en 65f960 y GETSYSTEMTIME justo arriba en 65f950.

Por ejemplo veremos algo así luego de pasar las dos funciones:

65f950 D1 07 0A 00 04 00 .. .. ..
65f960 D1 07 0A 00 04 00 .. .. ..

D107 es al reves 07d1 el año 2001
000A es el mes 10
0004 es el dia 4

La fecha puede variar según en que día estemos jua jua.

Puedo probar que pasa si allí mismo edito la fecha para cambiarla haciendo
E 65f950
y cambiando los valores
, si hago pruebas veo que la que varia los días de evaluación es la que halla BPX GETLOCALTIME o sea la de abajo la de 65f960.
Si cambio d1 por d2 o sea al año 2002 me dice que expiro y que va como un año de evaluación lo mismo con los días y el mes.
Pongamos un bpr allí a ver cuando el programa lee esa fecha que esta en la memoria pero aun no la leyó.

Bpr 65f960 65f966 r

Hago X y ENTER

Para en

```
00415857 0FB745F6               movzx eax, word ptr [ebp-0A]
:0041585B 50                    push eax
:0041585C 0FB745F2              movzx eax, word ptr [ebp-0E]
:00415860 50                    push eax
:00415861 0FB745F0              movzx eax, word ptr [ebp-10]
:00415865 50                    push eax
```

aquí es donde el programa guarda la fecha actual o sea en la primera sentencia pasa el día actual a eax en la segunda el mes y en la tercera el año.

Si uno cambia aquí que a EAX pase una fecha con el cual el programa funciona en forma fija siempre va a quedar en ese día para siempre.

Quedo así para mi

```
:00415857 B00E          mov al, 0E
:00415859 90            nop
:0041585A 90            nop
:0041585B 50            push eax
:0041585C B00A          mov al, 0A
:0041585E 90            nop
:0041585F 90            nop
:00415860 50            push eax
:00415861 B0D1          mov al, D1
:00415863 B407          mov ah, 07
:00415865 50            push eax
```

En la primera muevo a AL el valor 0e o sea el día 14
nopeo dos veces ya que la sentencia es mas corta en 41585c muevo a AL el valor 0a del mes 10 y en la tercera tengo que hacerlo en dos partes muevo a AL D1 y a AH 07 como resultado en EAX queda 07d1 que es el Año 2001 o sea que con esto congelo la fecha en el primer día de evaluación mío 14/10/2001. Lo mismo existe en las otras aplicaciones.
Ahora en Muapanel es similar

Bpx getlocaltime es la que maneja todo en muapanel también

Getlocaltime esta en 40a082
y guarda la fecha en 5bfaec.

Al salir si ponemos un BPR 5bfaec 5bfaef r

para en 40a12f que la parte similar a la que parcheamos en librarian

Aquí ya esta modificada

```
:0040A12F B00E          mov al, 0E
:0040A131 90            nop
:0040A132 90            nop
:0040A133 50            push eax
:0040A134 B00A          mov al, 0A
:0040A136 90            nop
:0040A137 90            nop
:0040A138 50            push eax
:0040A139 B0D1          mov al, D1
:0040A13B B407          mov ah, 07
:0040A13D 50            push eax
```

Recuerden de poner una fecha para la cual el programa para ustedes funcione, no copien la misma que la mía ya que así no funcionara, fíjense la fecha en que lo instalan y pongan esa fecha y siempre el programa estará en el primer día de evaluación.
Las otras aplicaciones que tiene son similares y se las dejo para que practiquen.

Ricardo Narvaja

CRACKEANDO CIEGO

(LECCION CINCUENTA)

El programa que vamos a crackear hoy es el **Xguest Inviter Bot 1.72** , la versión no es la ultima creo que va por la 1.82 o 83 no se, pero sirve para practicar y es un programa para conectarse a IRC .
Aquí la novedad es que a pesar de conseguir el serial valido que nos registra, cuando nos conectamos a INTERNET para usar el programa, verifica que no es un usuario que pago el REGISTRO y sale un cartel diciendo que infringiste la licencia y te cierra el programa.
Les diré que tiene sus bemoles crackear este programa ya que esta muy protegido inclusive amigos que han instalado versiones posteriores a esta y después han querido usar esta mas vieja, no te deja, dice que tenés que usar la ultima versión o algo así, como a mi eso no me ocurrió ya que no instale la ultima solo esta, no puedo saber como saltear eso, pero no debe ser difícil ya que son simples messageboxa.
Ya saben que el programa va a estar en el Kturn y OOdrive para bajarse, para crackear y practicar.
Otro tema son las consultas que generalmente me hacen a mi mail **ricnar22@millic.com.ar,** como yo me voy a mudar ese mail no se si va a seguir funcionando ya que no se que servicio de internet tendré en el otro departamento, así que hasta que me mude, pueden escribirme a **ricnar456@data54.com** que es WEBMAIL y va a seguir funcionando vaya a donde vaya, disculpen si demoro en las respuestas estos días, pues estaré de mudanza y sin internet, aunque en mi trabajo si tengo y leeré allí mis mails.
Sigamos con este tema.
El ejecutable del Xguest esta compactado con PECOMPACT me dice el Unpack y dice que lo puede descomprimir, lo hago y tengo el ejecutable descomprimido, que oh sorpresa salen las Funciones Importadas pero no las STRINGS REFERENCES así que crackearemos ciegos.
Además esta hecho en DELPHI, como información, aunque el DEDE tampoco dice mucho.
Como primer paso hay que hallar una clave para registrarnos valida, utilizando el método, de HMEMCPY y poner BPRs en donde aparece la clave falsa siguiendo los lugares donde el programa lee la clave, veo que después de un SCASB que para el SOFTICE, como esa sentencia es generalmente parte de lsrtlen que mide el largo de mi clave falsa 98989898 cuando vuelve al ejecutable sale siendo EAX=8 o la cantidad de cifras que tiene la que ustedes colocaron.
Siguiendo ese EAX=8 , si lo guarda con PUSH EAX viendo con D ESP donde lo guardo siguiendo con los BPRs vemos que para en

```
:00404057 8B46FC        mov eax, dword ptr [esi-04]
:0040405A 8B57FC        mov edx, dword ptr [edi-04]
:0040405D 29D0          sub eax, edx
:0040405F 7702          ja 00404063
```

Allí carga en EAX y en EDX la cantidad de cifras de la clave falsa y de la verdadera, y encima esos valores están delante de ambas claves así que si cuando estamos en 404057 hacemos D ESI-04 y D EDI-04 veremos la clave falsa y la verdadera fácilmente.
En mi caso la clave para registrase fue:

User: narvaja
Password: 4B899FE

Así con minúsculas el nombre y la clave toda en mayúsculas, allí cualquiera puede obtener su propia clave, o usar la mía si quieren.

Bueno pongo esa clave y user y me registra, ahora cuando intento usar el programa y se conecta a INTERNET me aparece un messageboxa bastante feo que dice que rompí la licencia.

Pongo un BPX messageboxa allí para que pare el Softice cuando aparece el cartel ese maléfico, cuando vuelvo al ejecutable veo que ese messageboxa es llamado desde:

:0044CFFD 50 push eax

*** Reference To: user32.MessageBoxA, Ord:0000h**
|
:0044CFFE E8DDA7FBFF Call 004077E0
En el Wdasm puedo mirar hacia arriba para ver si hay algunos saltos que esquiven este messageboxa.
Los saltos que están un poco mas arriba no sirven ya que no esquivan el messageboxa, sigo hacia arriba hasta que llego al REFERENCE este.

*** Referenced by a CALL at Addresses:**
|:0044D10D , :0048F77F , :0049BC63 , :004C9886 , :004CA09B
|:004CA7AE , :004CADAB , :004CDEE4 , :004CE1F6 , :004D09C4

Esos son los lugares desde donde entra al CALL donde sale el messageboxa fastidioso, poniendo BPX en todos estos lugares para ir viendo de donde es llamado, vemos que la primera vez para en 4CADAB, y un poco mas arriba esta el salto para parchear en

:004CAD45 757A jne 004CADC1

Este es el primer salto a parchear, cambiándolo por JMP 4CADC1

Con eso ya no nos sale el cartel maldito apenas nos conectamos, pero aparece en otros lugares que yo constate, que son otros lugares que están en ese REFERENCE de mas arriba.

004CDEE4 , :004CE1F6

En estos lugares paro en SOFTICE pues había puesto BPXs en todos esos valores, cada uno tiene un salto un poco mas arriba que hay que cambiar por JMP.

Cuando para en 4cdee4 un poco mas arriba en 4cde8d también hay un salto para parchear.

:**004CDE8D** 757C jne 004CDF0B

aquí también hay que cambiar por jmp 4CDF0B

cuando para en **4CE1F6** HAY UN SALTO MAS ARRIBA EN

:004CE19F 747F je 004CE220

también hay que cambiar aquí por JMP para que no vaya al CALL del cartel maldito.

Con parchear estos saltos ya me pude conectar sin problemas y ingresar al IRC perfectamente y recibir y enviar bien, de cualquier manera viendo que todavía quedan los otros CALLs aunque a mi no me paro allí, el método si llega a caer dentro es el mismo poner un BPX allí y parchear el salto que esquiva ese CALL, o también preventivamente se puede mirar en el WDASM y parchear el salto que se encuentre antes de todos esos CALLS y que eviten que caiga en ellos.

Ricardo Narvaja