



CURS0

Cracker

CURSO DE CRACKEO PARA EMPEZAR DESDE CERO

(LECCION UNO)

A pedido de muchos listeros de CUANDOQUIERAS y CONECTADOS, voy a tratar de explicar en un curso sencillo las bases para crackear programas o mejor expresado las bases de la INGENIERIA REVERSIBLE que es el arte de revertir las protecciones que poseen ciertos programas para hacerlos accesibles a todos y no haya que pagar por usarlos. Mucha gente dice que esto no es ético y que esta mal pero yo no estoy de acuerdo con esa opinión, yo creo que hay que crackear y distribuir gratuitamente los cracks, sin hacer negocio, para favorecer a la gente con menos recursos a que puedan acceder a programas que de otra manera no podrían usar.

ES DIFICIL CRACKEAR? HAY QUE TENER GRANDES CONOCIMIENTOS?

En realidad no se necesitan grandes conocimientos para empezar a crackear, lo que si es necesario es dedicarse, practicar, preguntar lo que no se entienda (NO QUEDARSE CON DUDAS), y una vez que ya uno comprenda las cinco o seis primeras lecciones, intentar crackear todo lo que caiga en la mano de uno para practicar, siempre si uno se traba puede preguntar y así seguir aprendiendo.

Eso si, el programa victima es el rival de uno como en un ring de box, y si uno recién esta aprendiendo tiene que ir gradualmente buscando rivales que estén a la altura de lo que uno va aprendiendo, si después de leer esta primera lección sola alguien quiere animársele al CUENTAPASOS por ejemplo, que es un rival de fuste hasta para los grandes crackers, seguro va a caer derribado en el primer round, por eso es importante seguir el curso e ir practicando.

Generalmente los mejores rivales para las personas que recién se inician son los programas SHAREWARE de hace unos años porque así como creció el arte crackear también fueron creciendo las protecciones y no es lo mismo la versión 1.0 de un programa de hace 4 o 5 años que la versión 2000, que salió el mes pasado e incorpora lo ultimo en protecciones.

Igual no todo es tan lineal, he crackeado programas editados en estos últimos meses cuya protección es prácticamente inexistente y de cualquier manera gradualmente vamos a ir incorporando los conocimientos para ir subiendo nuestra categoría en el arte crack y siendo cada vez mejores rivales para los programas.

Lo que si quería dejar constancia es que yo me considero un aprendiz, aunque me considero capaz de crackear casi el 80 % de los programas que están disponibles, a principio de año comencé a estudiar este arte y he mejorado muchísimo, sobretodo por la practica y leyendo tutoriales y consultando en foros con los grandes maestros de esto, mas adelante cuando tengan ciertos conocimientos les voy a dar las direcciones de los foros, una vez que ya estén en el tema, para no preguntar cosas obvias y quedar mal.

Si mi léxico no es muy técnico y no respeta 100% los tecnicismos, discúlpenme yo estoy tratando de hacerme entender en la forma mas fácil posible aunque las palabras no sean exactas.

EMPECEMOS

Como dije voy a tratar de ir despacio y no atosigar a nadie, los conocimientos que vamos a aprender van a ir muy de a poco para no saturar y de una lección a la otra se puedan ir asentando.

NUESTRA PRIMERA VICTIMA (ARCHIVO CRACKME)

El archivo crackme.exe que subí a FREESERVERS va a ser nuestra primera víctima. Es un programita de 12k que no sirve para nada, solamente para comenzar a enseñar como se crackea y que fue hecho con ese fin.

COMO SE ENCARA EL CRACKEO

Antes que nada les recomiendo imprimir las lecciones, porque no es lo mismo hacer las cosas y leer a la vez que estar cerrando y abriendo programas para poder seguir la explicación.

El primer paso es estudiar a la víctima, ejecutarlo y ver que tipo de programa es (CON TIPO DE PROGRAMA ME REFIERO A QUE SI SE REGISTRA CON UN NUMERO DE SERIE, SI VENDE DENTRO DE UN PLAZO, ETC), en nuestro primer caso es un programa que se registra con un numero de serie.

Vamos a probar, lo ejecutamos y vamos a HELP y después a REGISTER, donde nos aparece una ventana con lugar para escribir el nombre (NAME) y el NUMERO DE SERIE (SERIAL).

Vamos a poner datos inventados en los dos casilleros para ver que pasa, para que todos sigamos el mismo hilo pongamos el mismo nombre y numero de serie que uso yo (IGUAL PODRIA SER CUALQUIERA SIMPLEMENTE USEMOS TODOS EL MISMO PARA PODER ESTAR TODOS IGUAL EN EL MISMO EJEMPLO)

NAME: PEPE

SERIAL:989898

PEPE VA CON MAYUSCULAS Y EL NUMERO 989898 lo elegí porque el 999999 ya esta muy quemado (BAH CUALQUIER NUMERO ES IGUAL), hacemos clic en OK y obtenemos como resultado una triste ventana que en ingles nos dice que no tuvimos suerte (**NO LUCK THERE MATE**), podría decir también NUMERO INVALIDO o cualquier otro mensaje de ese deprimente estilo.

Este mensaje además de decirnos que el numero de serie no es correcto, nos esta gozando (se burla) por lo que lo vamos a reventar de lo lindo al Gil que hizo esto.

Siempre que comencemos a craquear un programa y como método de organización les aconsejo, usar una hojita en blanco donde escribimos el nombre del programa y TODOS los datos que vamos obteniendo de el, en este caso, es muy importante escribir el mensaje exacto que nos dice cuando pusimos el numero equivocado, anotamos **No luck, there mate!** en el papel y al lado le ponemos MENSAJE DE CHICO MALO.

Ahora vamos a explicar que es CHICO MALO y CHICO BUENO para los crackers.

Cualquier programa que verifica un numero de serie, llega un punto en que INEVITABLEMENTE va a comparar el numero que pusimos equivocado (989898 en el ejemplo), con el numero verdadero de serie.

Tiene que comparar si o si, el numero nuestro con el verdadero y tomar una decisión, si decirnos que es incorrecto (CHICO MALO) o correcto y registrarnos (CHICO BUENO)

Se dice CHICO MALO a la parte del programa que después de comparar ambos números, nos tira para ponernos el cartel lúgubre de INCORRECTO o como en este caso **NO LUCK THERE MATE!**

CHICO BUENO es supuestamente si la verificación es correcta y yo puse el número de serie correcto (PORQUE LO PAGUE O LO CRACKEE) entonces soy un CHICO BUENO y la parte del programa donde sos derivado después de la comparación y que termina en un cartelito que dice NUMERO CORRECTO REGISTRADO o algo así.

Entonces tenemos varias alternativas, o ponernos a jugar insistiendo con números de serie distintos hasta que en cinco o seis años la peguemos y nos registremos o seguir este curso y crackear esa bazofia de una vez por todas.

Vamos a comenzar a usar la primer herramienta que es el WDASM, este es un desensamblador ideal para novatos como yo, ya que los desensambladores avanzados para mi complican mucho las cosas en cambio el WDASM es claro como el agua y fácil, fácil, fácil.

WDASM o el LISTADO MUERTO

Empecemos con la primera herramienta que aprende a usar todo crackear el WDASM. El Wdasm es un programa que toma un archivo y si puede lo desensambla, ahora bien que es desensamblar?

Un programa como este crackme por ejemplo es una seguidilla de números en FORMATO HEXADECIMAL o sea podría ser como ejemplo

FF A5 B9 32 44 76 99 , etc etc etc...

Evidentemente una cadena de números no nos dice mucho acerca de lo que el programa quiere hacer y crackear eso seria mas difícil que hacer que no se cuelgue WINDOWS. (CHISTE)

Me olvidaba decirles que es esto del formato hexadecimal, en comparación con la numeración decimal que usa números del 0 al 9, la numeración hexadecimal después del 9 usa la A que seria el 10 decimal, la B que seria el 11, la C que seria el 12 y así hasta la F que es el 16.

Para los que no quieren problemas, con la calculadora de WINDOWS, la cambian a VER-CIENTÍFICA y pueden convertir números DECIMALES EN HEXADECIMALES y viceversa, si en la pantalla pongo DF y después hago clic en decimal me aparece el resultado 223, y viceversa si pongo por ej. 87 y hago clic en HEXADECIMAL me sale el resultado que es 57.

Bueno ahora que ya tragamos ese primer sapo sigamos.

Como les dije antes una cadena de números no sirve para nada, si contamos con un programa como el WDASM, entre otras cosas nos puede agrupar esos números y traducirlos a sentencias en IDIOMA ENSAMBLADOR.

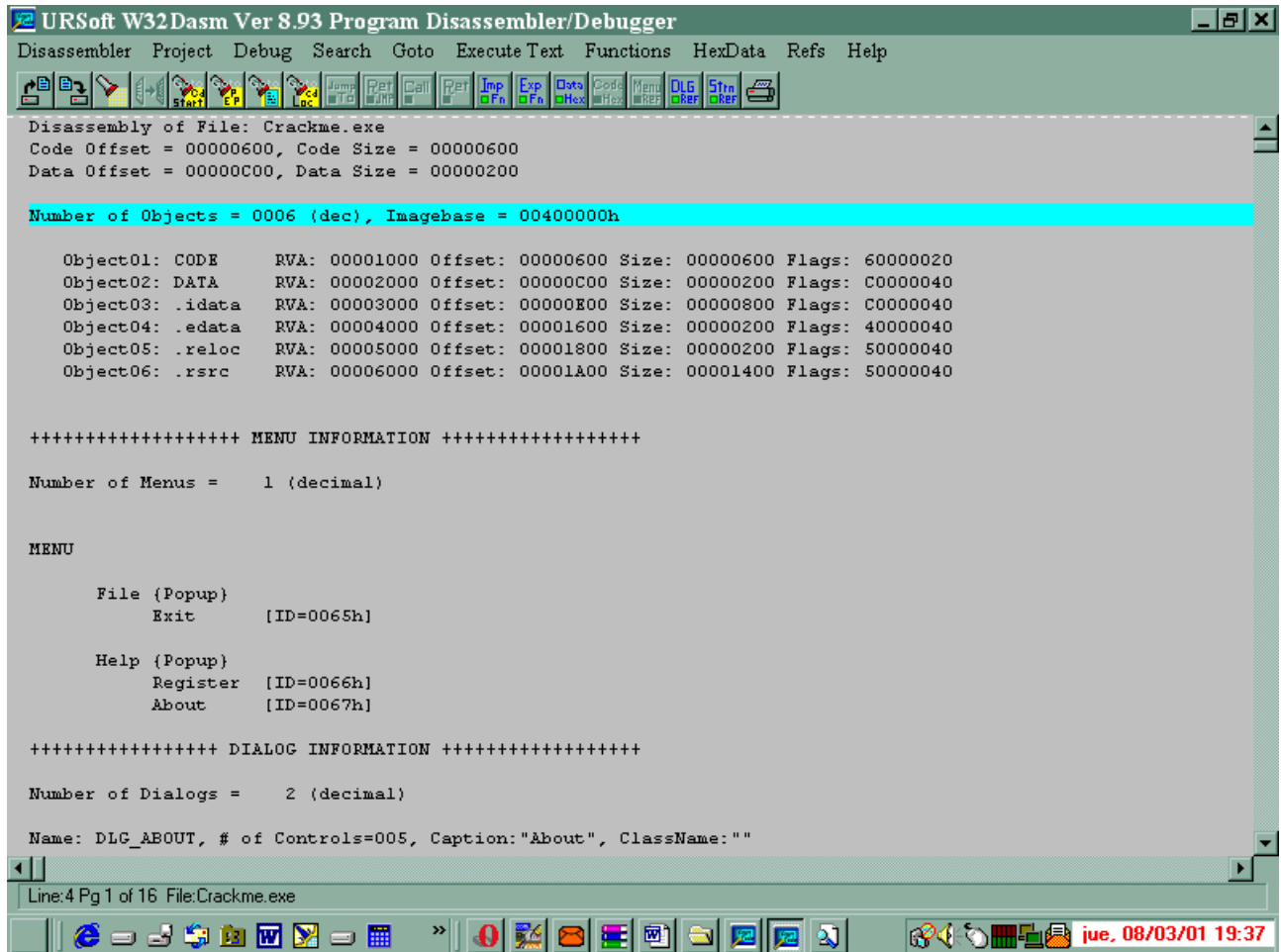
No se asusten que no es tan fiero el LEON como lo pintan, abramos el WDASM y en donde dice DISASSEMBLER hagamos clic y elijamos OPEN FILE TO DISSASSEMBLE para cargar la fila a destripar.

Buscamos por los directorios la fila crackme.exe y la cargamos. Bueno como es una fila chiquita se desensambla rápido, bueno ahí tenemos vamos despacio para no marearnos.

Lo primero que hay son datos sobre el archivo que por ahora no vamos a ahondar (YA LLEGARA SU MOMENTO), la primera pregunta que surge es bueno donde comienza el programa, cual es la primera sentencia que se ejecuta?

DIBUJO 1

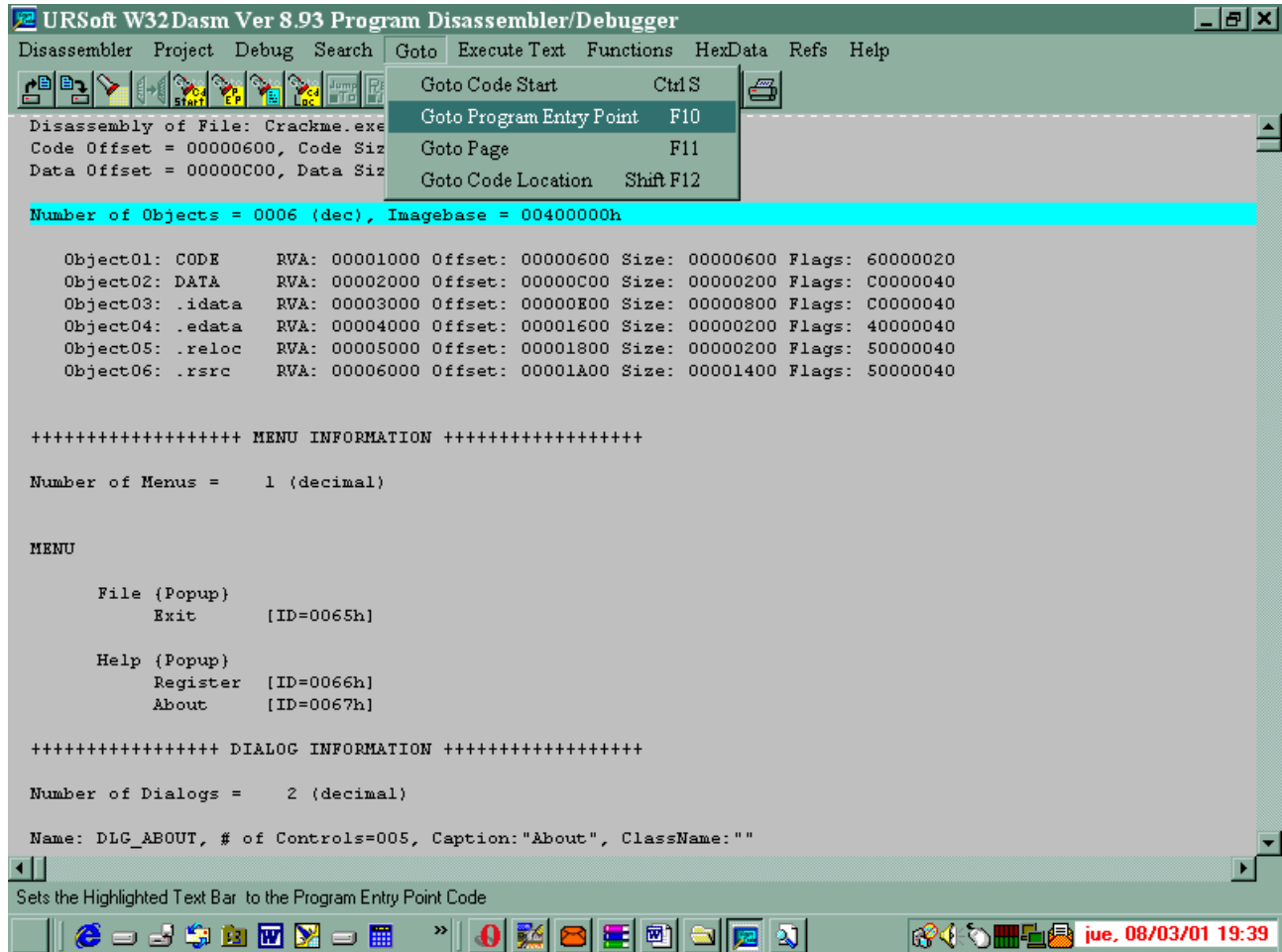
ESTO ES LO QUE SE VE CUANDO APENAS CARGO EL CRACKME



Bueno eso es bastante fácil, abrimos el menú GOTO y elegimos GOTO PROGRAM ENTRY POINT que quiere decir IR A PUNTO DE ENTRADA DEL PROGRAMA, si hacemos clic ahí vamos a la primera sentencia que es:

DIBUJO 2

AQUÍ VEMOS CUANDO BUSCAMOS EL ENTRY POINT (TODAVÍA NO APARECE AQUÍ)



PROGRAM ENTRY POINT

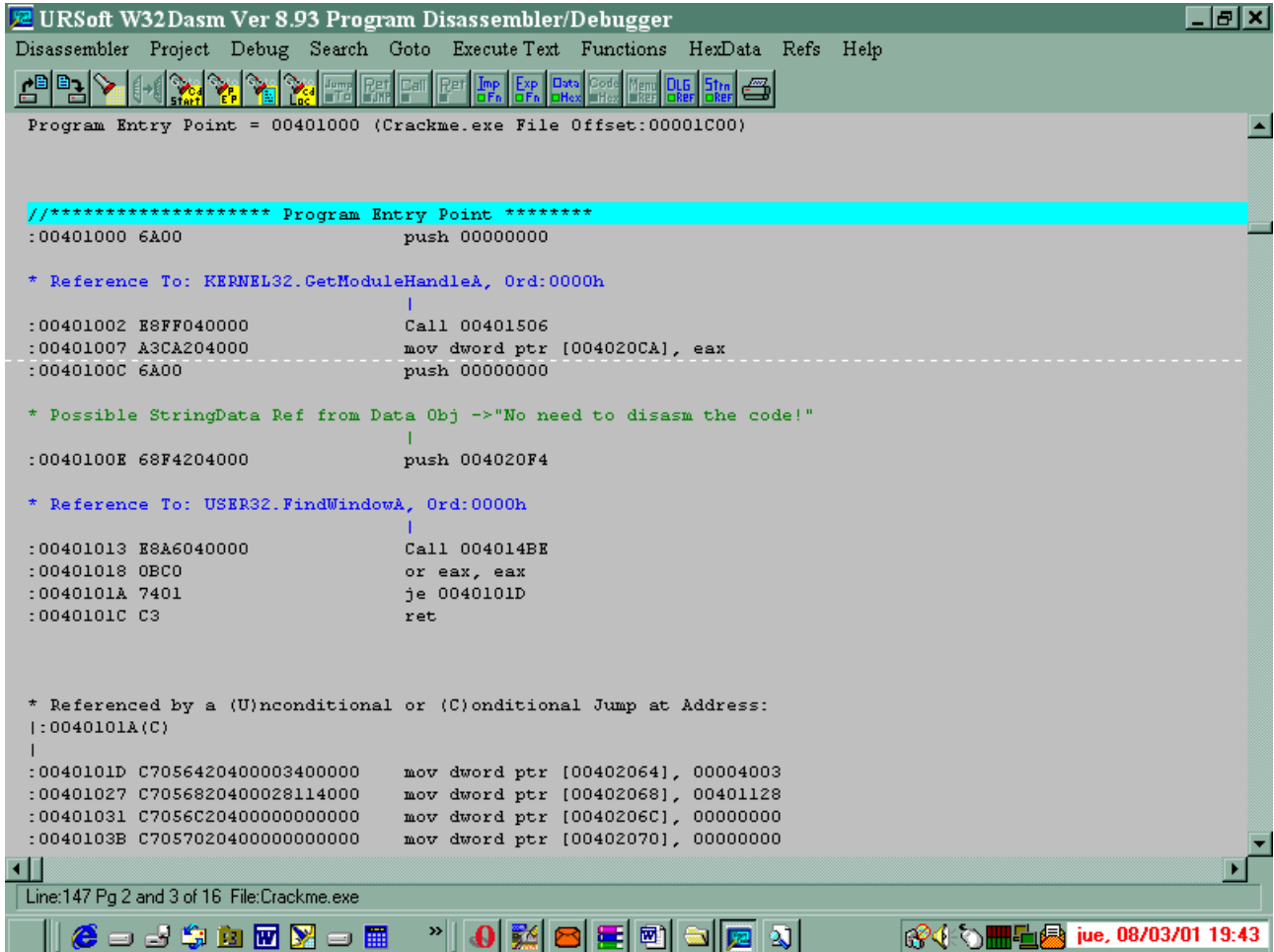
0040100 6A 00 PUSH 00000000

Bueno no es necesario enloquecerse con lo que significa la sentencia PUSH pues no es de gran utilidad para un cracker, lo que hay que saber es que 0040100 es la posición de memoria donde el programa va a comenzar a ejecutarse una vez que este en la memoria después de hacer clic sobre el archivo, y los números 6A 00 son los dos primeros números hexadecimales (EL QUE ESTA EL LA POSICION 40100 y 40101) que juntos forman la sentencia PUSH 00000000 en idioma ensamblador.

Bueno ya basta de pavadas y vamos a lo fácil después que tragamos ese pequeño sapo, podemos eso si anotar en nuestro papel la dirección del PUNTO DE ENTRADA que aunque no es necesario en este caso para crackear, en otros programas puede serlo y es bueno saber como encontrarlo.

DIBUJO 3

AQUÍ APARECE EL ENTRY POINT



```
URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger
Disassembler Project Debug Search Goto ExecuteText Functions HexData Refs Help

Program Entry Point = 00401000 (Crackme.exe File Offset:00001C00)

//***** Program Entry Point *****
:00401000 6A00          push 00000000

* Reference To: KERNEL32.GetModuleHandleA, Ord:0000h
|
:00401002 E8FF040000   call 00401506
:00401007 A3CA204000   mov dword ptr [004020CA], eax
:0040100C 6A00          push 00000000

* Possible StringData Ref from Data Obj ->"No need to disasm the code!"
|
:0040100E 68F4204000   push 004020F4

* Reference To: USER32.FindWindowA, Ord:0000h
|
:00401013 E8A6040000   call 004014BE
:00401018 0B00        or eax, eax
:0040101A 7401        je 0040101D
:0040101C C3          ret

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040101A(C)
|
:0040101D C7056420400003400000 mov dword ptr [00402064], 00004003
:00401027 C705682040000281140000 mov dword ptr [00402068], 00401128
:00401031 C7056C204000000000000000 mov dword ptr [0040206C], 00000000
:0040103B C7057020400000000000000000000000 mov dword ptr [00402070], 00000000

Line:147 Pg 2 and 3 of 16 File:Crackme.exe
jue, 08/03/01 19:43
```

ACCION, ACCION

La hinchada clama acción y vamos a empezar a crackear esto.

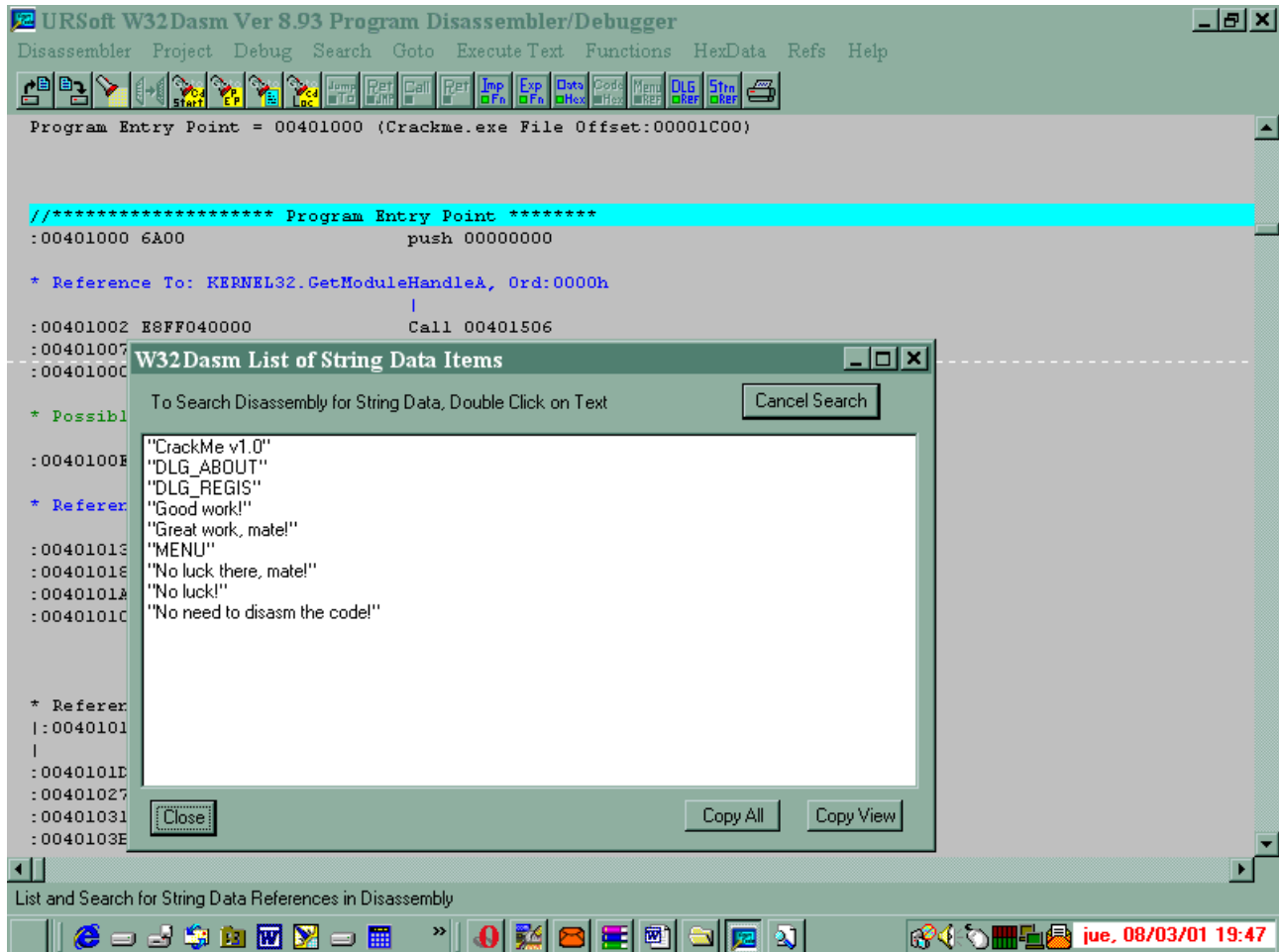
Entre los iconitos del WDASM hay uno que es el penúltimo, al lado de la impresora, que es el de STRING DATA REFERENCES, que son las referencias a las cadenas de texto que tiene el programa, nos aparece una ventanita con distintas cadenas de texto, entre ellas nuestra conocida NO LUCK THERE MATE! Que nos mostró el programa cuando pusimos como numero de serie el 989898, JA JA estamos entrando.

Hacemos doble clic exactamente encima de esa cadena de texto y el WDASM cambia de mostrarnos el ENTRY POINT a mostrarnos donde usa el crackme la cadena de texto.

ENTRAMOS ENTRAMOS.

DIBUJO 4

ESTO NOS MUESTRA CUANDO HACEMOS CLIC EN EL ICONO STRN REF (STRING REFERENCES) LA VENTANA DONDE APARECEN LAS CADENAS DE TEXTO QUE USA EL PROGRAMA.



REFERENCED BY A CONDICIONAL JUMP or UNCONDICIONAL JUMP AT
ADDRESS
40138B

```
004013AC      pop     esi
004013AD      push   00000030
004013AF      push   402160
```

; "No luck!"

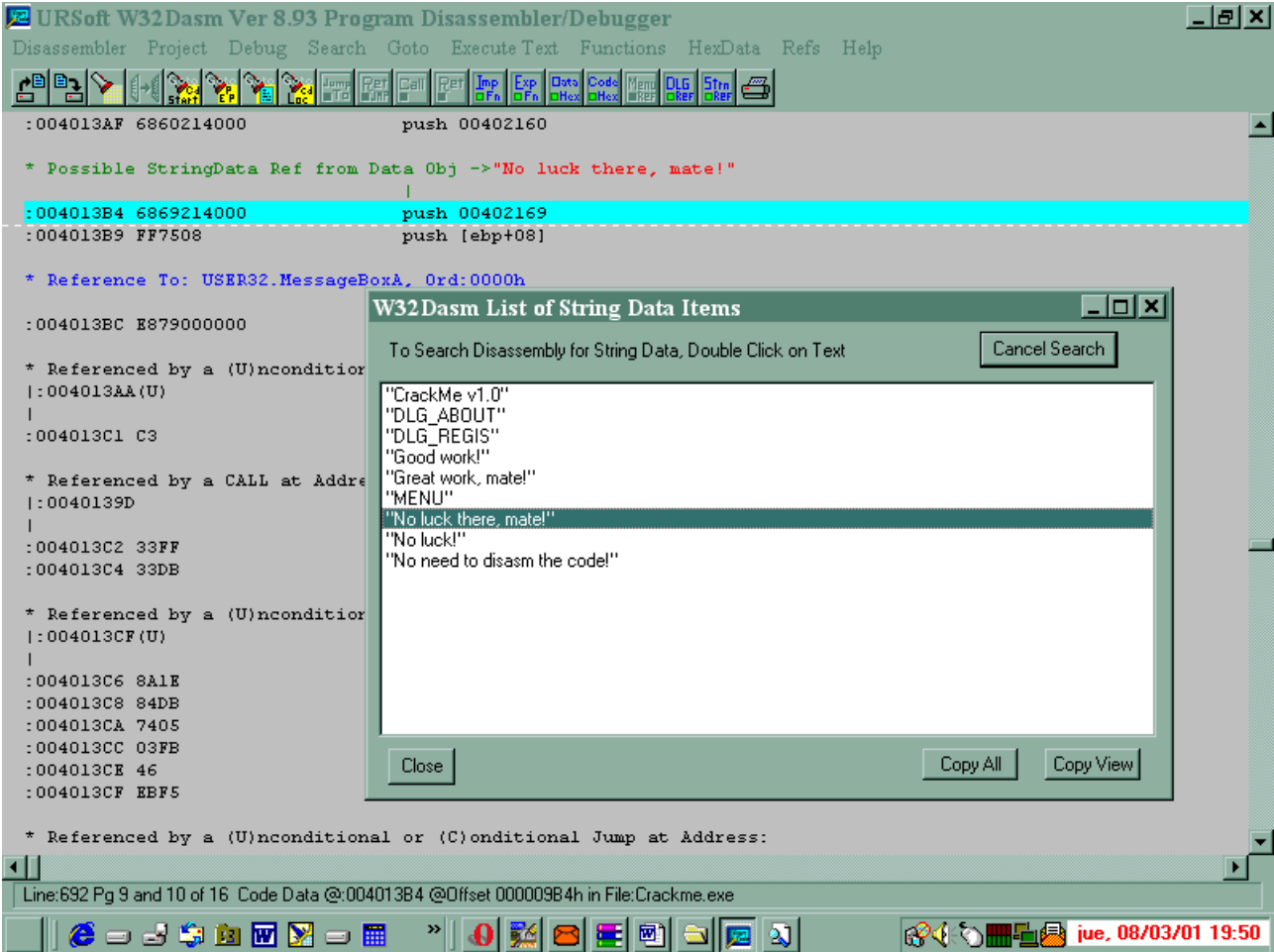
```
004013B4      push   402169
```

; "No luck there, mate!"

```
004013B9      push   [ebp+8]
004013BC      call  40143A
```

MessageBoxA

AQUI SE VE UNO DE LOS DOS LUGARES QUE CAIGO CUANDO HAGO DOBLE CLICK ENCIMA DE NO LUCK THERE MATE LA LINEA AZUL ESTA JUSTO SOBRE 4013b4. ALLI EN ROJO SE VE LA CADENA NO LUCK THERE MATE Y UN POCO MAS ARRIBA SE VERA DE DONDE VIENE EL PROGRAMA SALTANDO AQUÍ)



Esto es casi exactamente donde van a caer, le quite los números hexadecimales para no confundir, pero es esto, la parte del programa donde usa el texto NO LUCK THERE MATE!

O sea esta es la parte del programa del CHICO MALO, acá el programa nos tira después de haber comparado y ver que el numero que pusimos 989898 no es el correcto y decide tirarnos acá y ponernos el cartel triste de que nuestro numero no es valido.

Miremos aun sin saber nada un poco esto, cuando el WDASM nos dice

REFERENCED BY A CONDITIONAL JUMP or UNCONDITIONAL JUMP AT ADDRESS 40138B

Quiere decir que desde esa dirección (40138B) el programa nos tiro acá en esta zona de CHICO MALO,

SUBIENDO UN POQUITO VEMOS LA LINEA AZUL AHORA JUSTO ENCIMA DE REFERENCED BY A CONDICIONAL JUMP 40138B QUE ES EL SALTO QUE HACE LLEGAR EL PROGRAMA AQUÍ A LA ZONA EN QUE ESCRIBE NO LUCK THERE MATE.

```
URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger
Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

:004013A8 8BC7          mov eax, edi
:004013AA EB15          jmp 004013C1

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040138B(C)
|
:004013AC 5E           pop esi
:004013AD 6A30        push 00000030

* Possible StringData Ref from Data Obj ->"No luck!"
|
:004013AF 6860214000  push 00402160

* Possible StringData Ref from Data Obj ->"No luck there, mate!"
|
:004013B4 6869214000  push 00402169
:004013B9 FF7508      push [ebp+08]

* Reference To: USER32.MessageBoxA, Ord:0000h
|
:004013BC E879000000  call 0040143A

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004013AA(U)
|
:004013C1 C3          ret

* Referenced by a CALL at Address:
|:0040139D
|
:004013C2 33FF        xor edi, edi
:004013C4 33DB        xor ebx, ebx
```

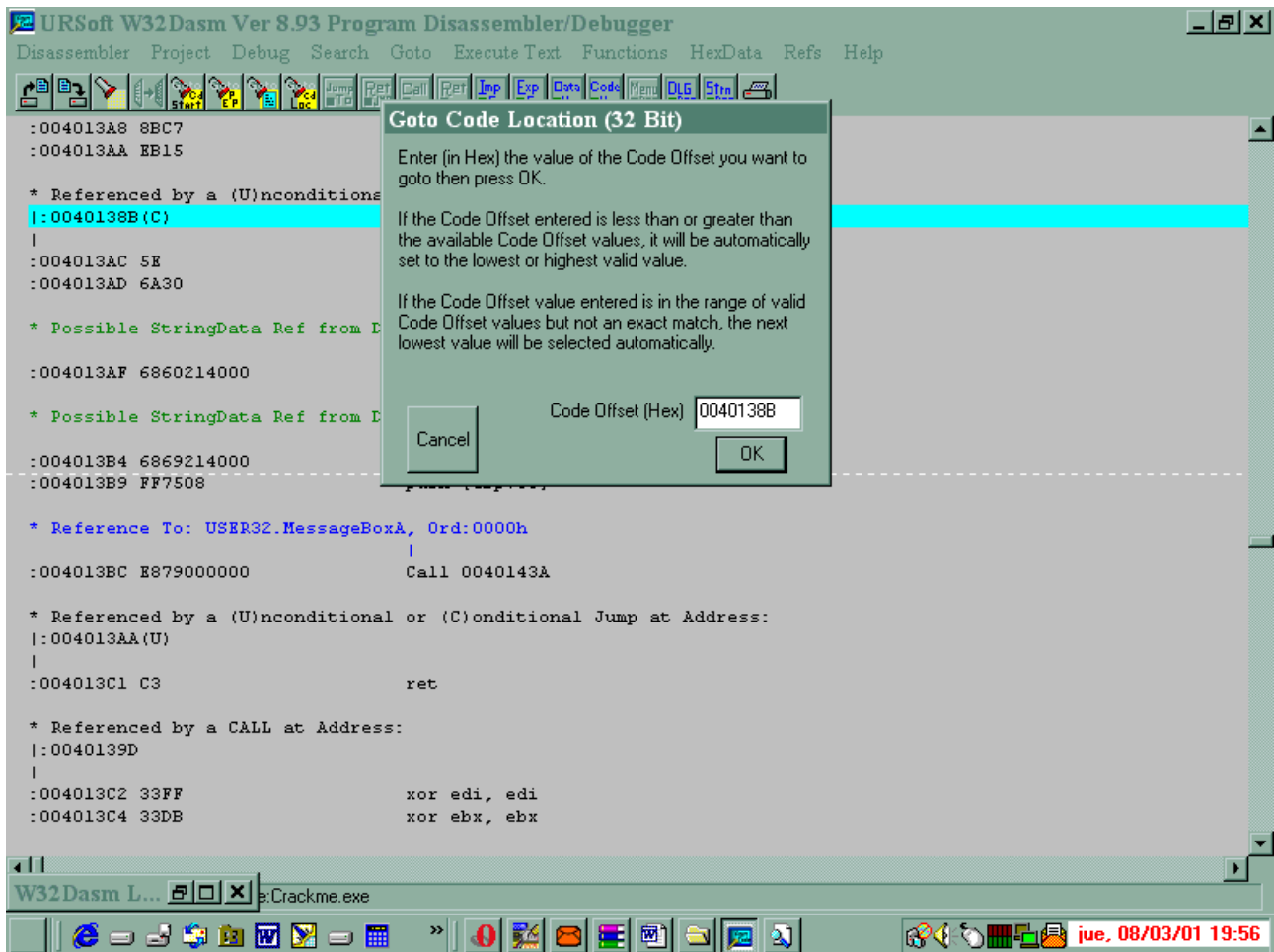
Por lo tanto si desde allí nos tiro a la zona de chico malo, por allí debe estar la comparación de los números de serie el 989898 con el verdadero y después el salto hacia la zona de chico malo o sino siguen en la zona de chico bueno.

Veamos

Vamos de vuelta a GOTO y elegimos GOTO CODE LOCATION, que sirve para ir a una dirección que nosotros queremos, en la ventanita que nos aparece tipeamos 40138b y aparecemos en el lugar donde hay comparaciones CMP y según el resultado de esas comparaciones un salto (CUALQUIER SENTENCIA QUE EMPIZA CON J o sea JE,JB,JZ,JNZ, etc)

Esa son las dos sentencias que mas nos interesan a los crackers CMP ebx,eax por ejemplo compara el contenido de eax con el contenido de ebx, y si en eax esta nuestra clave falsa, seguro que en ebx esta la clave verdadera, después siempre de la comparación esta el salto condicional que es que salta a la zona de CHICO MALO y a poner el cartel de NO LUCK THERE MATE si son distintos y sigue ejecutándose sin saltar si son iguales.

AQUÍ VEMOS LA VENTANITA DE GOTO CODE LOCATION Y ALLI PONGO EL VALOR DONDE QUIERO IR A VER QUE HAY O SEA 40138B QUE ERA MI REFERENCED....



Veamos que encontramos ahí.

```

401385    test al,al
401387    je 40139c
401389    cmp al,41
40138b    jb 4013ac  aquí salta a la zona de chico malo (NO LUCK THERE MATE!)
40138d    cmp al,5a
40138f    jnb 401354

```

O sea que el paso final aquí dado que el WDASM no puede ver los valores que se están comparando para saber la clave seria usar el SOFTICE pero el problema es que el uso del softice se va a ver en la lección 2, pero el softice permite ejecutar el programa y parar justo en la sentencia 401389 para ver que se compara ahí, igual sin haberlo hecho les digo que ahí no esta la comparación de la clave porque ahí se ve que compara el valor AL (DONDE PODRIA ESTAR 989898) por ejemplo, con un valor hexadecimal fijo.

AQUÍ SE VE DONDE COMPARA AL CON 41 Y CON 5A

```
URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger
Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

:00401383 8A06      mov al, byte ptr [esi]
:00401385 84C0      test al, al
:00401387 7413      je 0040139C
:00401389 3C41      cmp al, 41
:0040138B 721F      jb 004013AC
:0040138D 3C5A      cmp al, 5A
:0040138F 7303      jnb 00401394
:00401391 46        inc esi
:00401392 EBEB      jmp 00401383

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040138F(C)
|
:00401394 E839000000 call 004013D2
:00401399 46        inc esi
:0040139A EBEB      jmp 00401383

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401387(C)
|
:0040139C 5E        pop esi
:0040139D E820000000 call 004013C2
:004013A2 81F778560000 xor edi, 00005678
:004013A8 8BC7      mov eax, edi
:004013AA EB15      jmp 004013C1

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040138B(C)
|
:004013AC 5E        pop esi
:004013AD 6A30      push 00000030

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

W32Dasm L... @:0040138B @Offset 00000988h in File:Crackme.exe
jue. 08/03/01 19:58
```

Con el numero 41, así que aquí me parece que no es, el softice o el trw2000 tendrían la palabra.(EN REALIDAD EL PROFESOR ES UN TRAMPOSO Y SE FIJO CON EL SOFTICE QUE ESTA COMPARACION NO ES LA QUE BUSCAMOS)

Volvamos al WDASM a STRING DATA REFERENCES y hagamos click de nuevo en el texto NO LUCK THERE MATE, si hacemos doble click aparecemos en la sección que ya estudiamos, y si volvemos a hacer doble click aparecemos en otro lado o sea que el cartel NO LUCK THERE MATE! esta en otra parte del programa también (NOS QUISIERON CONFUNDIR) vemos que no esta en ninguna parte mas solo en esas dos.

Veamos donde aparecemos

REFERENCED BY A CALL at address 401245

Y después siguen varias sentencias donde usa el NO LUCK THERE MATE, entonces vamos a ver desde donde viene a 401245

CAEMOS AQUÍ SI VOLVEMOS A HACER CLIC EN NO LUCK THERE MATE DE NUEVO

```
URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger
Disassembler Project Debug Search Goto ExecuteText Functions HexData Refs Help

:0040136B 6860214000      push 00402160
* Possible StringData Ref from Data Obj -> "No luck there, mate!"
:00401370 6869214000      push 00402169
:00401375 FF7508          push [ebp+08]
* Reference To: USER32.MessageBoxA, Ord:0000h
:00401378 E8BD000000      call 0040143A
:0040137D C3              ret

* Referenced by a CALL at Address:
|:0040122D
|
:0040137E 8B742404        mov esi, dword ptr [esp+04]
:00401382 56              push esi

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00401392(U), :0040139A(U)
|
:00401383 8A06           mov al, byte ptr [esi]
:00401385 84C0           test al, al
:00401387 7413           je 0040139C
:00401389 3C41           cmp al, 41
:0040138B 721F           jb 004013AC
:0040138D 3C5A           cmp al, 5A
:0040138F 7303           jnb 00401394
:00401391 46             inc esi
:00401392 EBFF           jmp 00401383

W32Dasm L... @:00401370 @Offset 00000970h in File:Crackme.exe
jue, 08/03/01 20:00
```

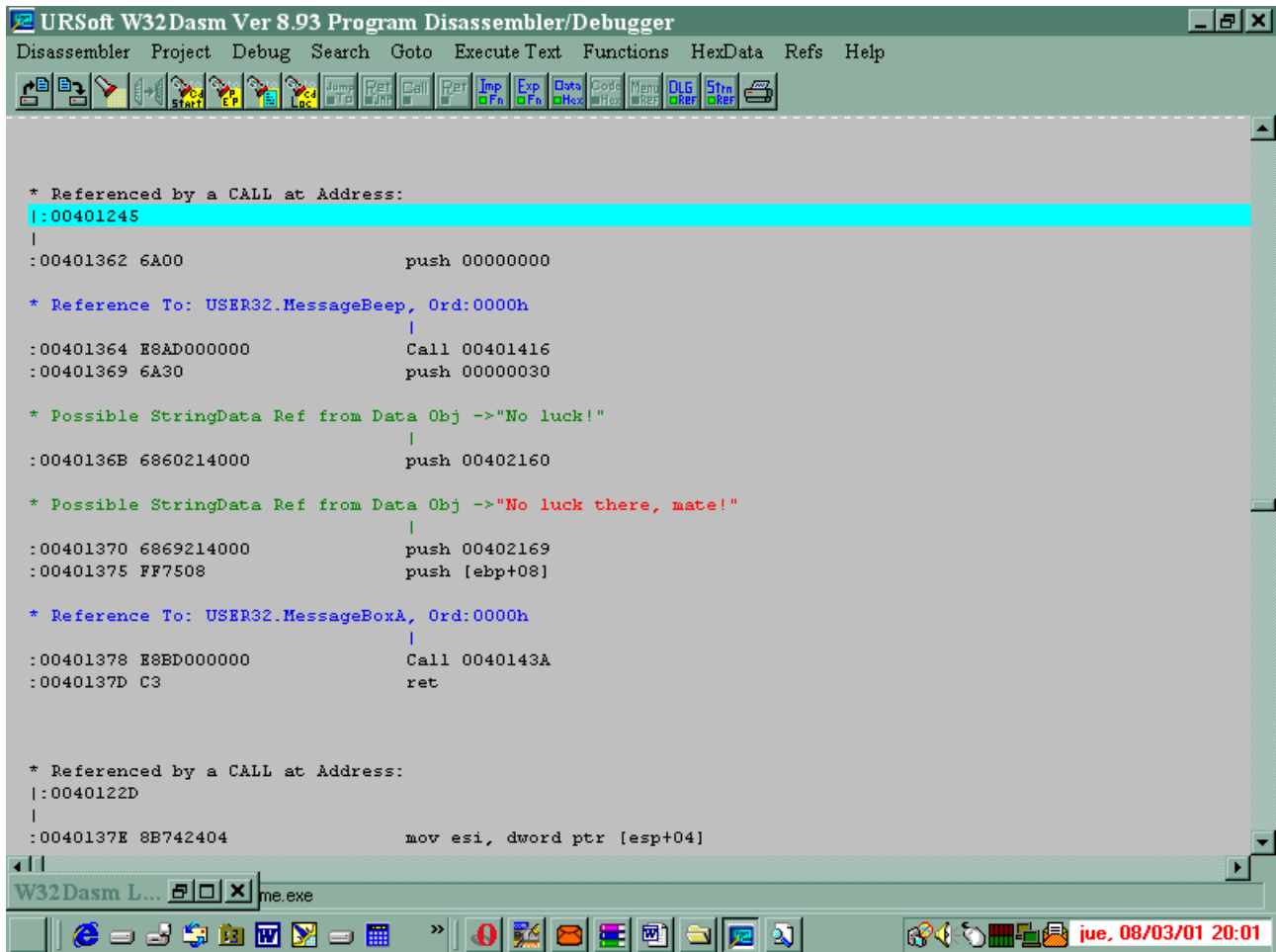
Vamos a GOTO - GOTO CODE LOCATION y tipeamos 401245

Lo que aparece es mucho mas agradable a la vista que lo que habíamos encontrado antes, aquí hay una comparación de eax y ebx que es mas posible que sea lo que buscamos (EL SOFTICE nos sacaría de dudas (COMO YO SOY TRAMPOSO A MI YA ME LAS SACO))

```
401241      cmp eax, ebx
401243      je 40124c      si son iguales salta a 40124c y evita la zona de CHICO
MALO
401245      call 401362
40124a      jmp 4011e6

40124c      call 40134d (ZONA DE CHICO BUENO)
```

AQUÍ VEMOS SI SUBIMOS UN POCO LA LINEA AZUL EN EL REFERENCED QUE ESTA JUSTO ARRIBA ME DICE QUE VIENE DE 401245.



```
URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger
Disassembler Project Debug Search Goto ExecuteText Functions HexData Refs Help

* Referenced by a CALL at Address:
|:00401245
|
:00401362 6A00          push 00000000

* Reference To: USER32.MessageBeep, Ord:0000h
|
:00401364 E8AD000000      Call 00401416
:00401369 6A30          push 00000030

* Possible StringData Ref from Data Obj ->"No luck!"
|
:0040136B 6860214000     push 00402160

* Possible StringData Ref from Data Obj ->"No luck there, mate!"
|
:00401370 6869214000     push 00402169
:00401375 FF7508         push [ebp+08]

* Reference To: USER32.MessageBoxA, Ord:0000h
|
:00401378 E8BD000000      Call 0040143A
:0040137D C3            ret

* Referenced by a CALL at Address:
|:0040122D
|
:0040137E 8B742404       mov esi, dword ptr [esp+04]
```

O sea que compara dos valores, si son iguales evita el call 401362 que nos llevaba a la zona de CHICO MALO y entonces sigue en 40124c donde seguro somos CHICOS BUENOS y nos va a registrar.

Ya terminamos ya que no podemos usar el softice todavía lo vamos a crackear sin conocer el numero de serie.

COMO SE HACE ESO

Muy fácil miren de vuelta el salto que esta en 401243 je 40124c

Ese salto es un salto condicional que si salta nos registramos y si no salta no.

Lo que hace que decida registrarnos o no es la comparación que esta en la sentencia anterior.

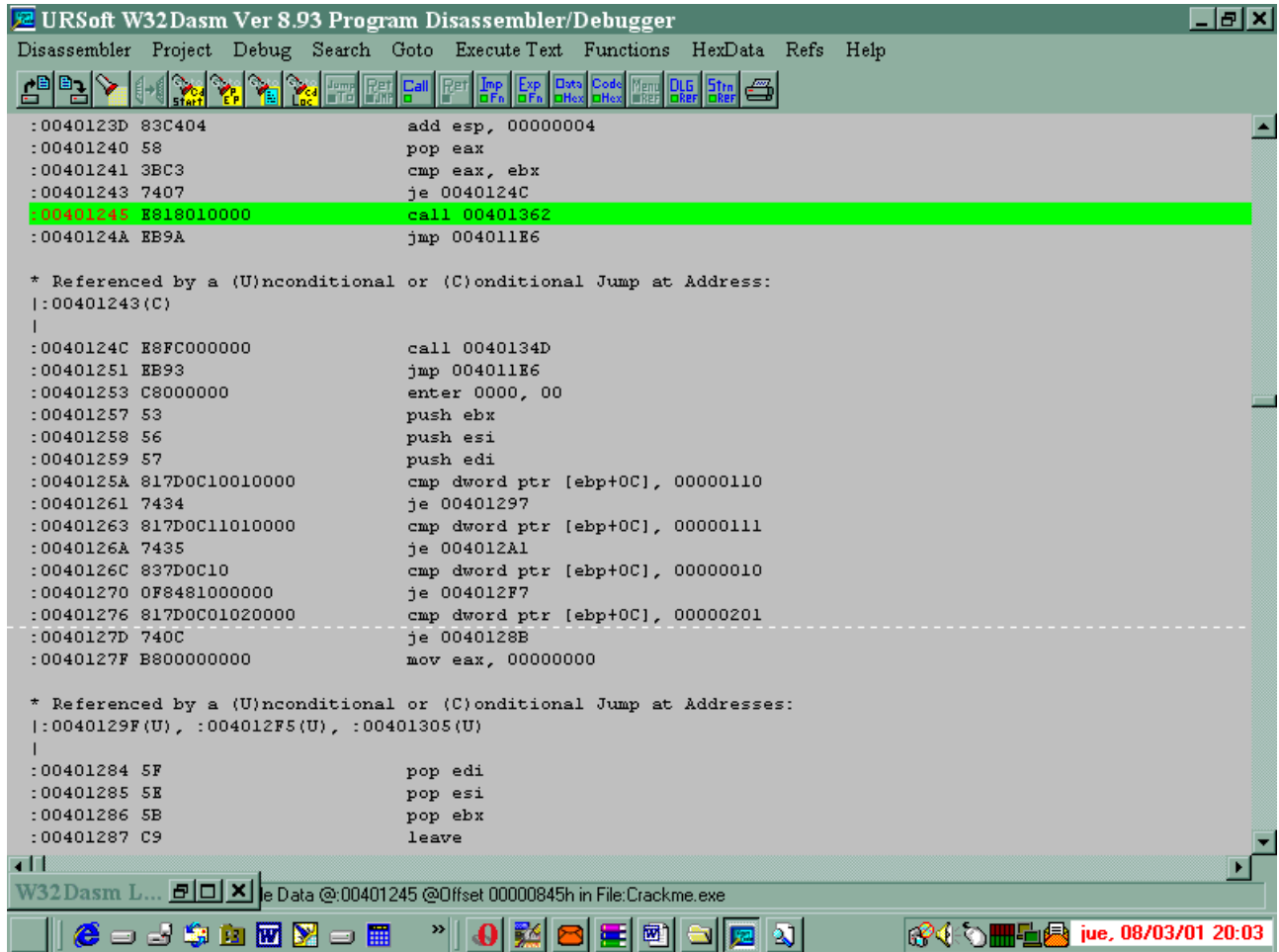
Lo que nosotros necesitamos es que siempre salte, independientemente de lo que compare y sea cual sea el número que pusimos como NUMERO DE SERIE.

Como se hace eso

FACIL

En vez de un salto condicional je usamos una sentencia que salte siempre que es la sentencia jmp

AQUÍ SE VE 401245 QUE ALLI HAY UN CALL Y ARRIBA UN SALTO QUE PUEDE SALTEAR ESE CALL PUES PUEDE IR A 40124C SI SALTA



O sea que tenemos que reemplazar je por jmp y saltaría siempre por encima del call que lleva a NOLUCK THERE MATE y nos registraría siempre sea cual fuere el numero que hayamos puesto.

Esto no debe asustarlos la mayoría de los cracks reemplazan saltos condicionales por JMP no es nada del otro mundo

COMO SE HACE

Vamos al WDASM y nos ponemos encima de la sentencia 401243 (CON GOTO CODE LOCATION)

Copiamos una cadena de números hexadecimales que comience en el mismo 401243 y seguimos copiando los números siguientes hasta hacer una cadena de 10 o 12 cifras. (copiamos los números que están en negrita)

```
:  
:00401243 7407          je 0040124C  
:00401245 E818010000     call 00401362  
:0040124A EB9A          jmp 004011E6
```

Por ejemplo **74 07 E8 18 01 00 00 EB 9A**

Vemos en la sentencia 40124a jmp 4011e6 que el primer numero hexadecimal que corresponde a la sentencia JMP es EB o sea que tenemos que reemplazar el 74 que es je por EB que es JMP.

Cerramos el WDASM, (PASO NECESARIO PORQUE VAMOS A ABRIR EL ULTRA EDIT Y SI EL ARCHIVO ESTA EN USO NO NOS VA A DEJAR GRABAR, LO MISMO QUE SI EL CRACKME ESTA FUNCIONANDO CERREMOSLO ANTES)

Abrimos el ULTRA EDIT y ponemos open file y abrimos el crackme, como el ULTRA EDIT ES UN EDITOR HEXADECIMAL nos va a mostrar todos los números hexadecimales del programa uno a continuación del otro (SON LOS MISMOS NUMEROS QUE ESTABAN EN EL WDASM SOLO QUE AQUÍ NO APARECEN LAS SENTENCIAS DESENSAMBLADAS)

Vamos a SEARCH - FIND y copiamos la cadena de números que deseamos buscar pero todos los números seguidos sin dejar espacio o sea
7407E818010000EB9A

Nos va a aparecer marcado donde esta la cadena a cambiar y hacemos click en el 74 y escribimos EB, después vamos a FILE y ponemos SAVE para que guarde los cambios.(SI NO LES DEJA GRABAR ES PORQUE ESTA ABIERTO EL WDASM O EL ARCHIVO CRACKME esta ejecutándose)

Ahora ejecutamos de nuevo el archivo crackme y vamos a donde dice REGISTER y ponemos PEPE y 989898, si quieren pongan otro numero, pero el nombre debe ser de letras nada mas no números, ponemos ACEPTAR y CHACHAN

GREAT WORK MATE, NOW TRY DE NEXT CRACKME

PROGRAMA CRACKEADO Y YA NO SE RIE MAS

Ricardo Narvaja

MAESTRO CIRUELA SIN TITULO

DESCARGADO GRATUITAMENTE DE
<http://visualinformatica.blogspot.com>

COMO COMENZAR A USAR EL SOFTICE

(LECCION DOS)

¿Qué es el SOFTICE exactamente?

El softice es exactamente un debugger o sea es una herramienta que usan los programadores para DEPURAR de errores sus programas, (DEBERIAN A VECES USARLO MAS Y DEPURARLOS BIEN), pero bueno, a nosotros los crackers nos sirve para DEPURAR los programas de las protecciones.

CUALQUIER PROGRAMA SE PUEDE CRACKEAR CON SOFTICE?

Cualquier programa puede ser destripado con softice, aunque hay ciertos programas que vienen con protecciones antisoftice, de cualquier manera esas protecciones generalmente pueden ser fácilmente evitadas utilizando el programita FROGSICE que les hice bajar entre las herramientas, al haber ejecutado el FROGSICE y estar este en la BARRA DE TAREAS activado, el softice es casi indetectable.

Después el softice nos da la posibilidad de ver el programa funcionando y hacer pruebas y muchas cosas más, queda en la habilidad del cracker y también en la dificultad de la protección la posibilidad de crackearlo o no.

HAY PROGRAMAS INCRACKEABLES?

Hay ciertos programas que son demostraciones que no traen todas las opciones, o sea que funcionan parcialmente, si en la demostración están deshabilitadas y se pueden habilitar no hay problema, el tema es cuando hay partes que directamente no vienen en la demostración o sea que el programa está incompleto, ahí la cosa se complica porque una cosa es crackear y otra completar un programa con cosas inexistentes.

Hay crackers especialistas que escriben partes faltantes de programas pero creo que eso más que INGENIERIA REVERSIBLE es ya otra ciencia.

BUENO COMENCEMOS Y BASTA DE CHACHARA

Hay dos formas de entrar a un programa con el softice, vamos a aprender primero la más fácil, (AUNQUE QUIZAS LA MENOS USADA).

Si en el escritorio el programa cuando se instalo les puso un ACCESO DIRECTO llamado solamente SYMBOL LOADER hagan click allí, si no, pueden ir a INICIO-PROGRAMAS-NUMEGA SOFTICE- SYMBOL LOADER (LES RECOMIENDO SI NO LO TIENEN HACER UN ACCESO DIRECTO).

Antes de seguir por favor impriman esta lección porque si no, van a tener que estar entrando y saliendo de SOFTICE a cada rato y se van a marear.

Una vez que ejecutaron SYMBOL LOADER les va a aparecer la ventana del SOFTICE que se usa solo para arrancar desde aquí las víctimas, por eso vamos a OPEN y buscamos el CRACKME.exe que no este crackeado (SI NO LO TIENEN BAJENLO DE NUEVO) y lo abrimos entonces nos va a aparecer el mensaje:

C:\WINDOWS\Escritorio\Crackme.exe opened successfully

O en su caso les aparecerá la carpeta donde está y que está abierto perfectamente.

Entonces comienza el acto de magia, vamos a MODULE y clickeamos LOAD y nos va a aparecer un cartelito de error (NO SE PREOCUPEN ES UN ERROR PROVOCADO A PROPOSITO POR EL SOFTICE PARA ENTRAR) pongan que SI y... ADENTRO MI ALMA.

ESTAMOS DENTRO DEL SOFTICE (PRIMER PANTALLAZO)

Que vemos aquí ahora que estamos dentro.

Un cuadradito titilando abajo en la zona de comandos (PARA ESCRIBIR)

Un poco más arriba una línea horizontal verde con en el medio un nombre (CRACKME +....) eso significa que estamos en la ejecución del archivo crackme.

O sea que allí aparece el nombre del archivo al cual pertenece la parte que estamos ejecutando.

Después un poco más arriba aparece la parte donde se va a desensamblar el programa, similar en forma a la que estamos acostumbrados a ver en el WDASM, la primera columna son las posiciones de memoria, la segunda los números hexadecimales y la tercera las sentencias en ensamblador.

Si recuerdan la primera lección yo les dije que anotaran el punto de entrada del programa o ENTRY POINT del crackme aquí vemos que la línea blanca que indica la sentencia que se va a ejecutar esta en

401000 que era el punto de entrada del programa crackme.

Al lado se ven por ahora en vez de las sentencias de ensamblador todas INVALID INVALID y eso es porque el programa SOFTICE genero un error para que el programa pare de golpe justo antes de empezar, pero eso no es nada.

La tecla F10 es la que hace que ejecutemos una sola sentencia (LA 401000 que es donde está la raya blanca), la tecleamos una sola vez.

Ahora la línea blanca paso a 401002 al haberse ejecutado la primera sentencia y ya aparece el programa como corresponde en la segunda columna tenemos los números HEXADECIMALES idénticos que en el WDASM (SI TIENEN DUDAS ENTREN AL WDASM Y VAYAN A GOTO ENTRY POINT Y COMPRUEBEN) y al lado las sentencias en idioma ensamblador, si alguno se fija el idioma que usa el SOFTICE no es exactamente igual que el del WDASM pero las sentencias son las mismas y hacen lo mismo, es una pequeña diferencia de notación pero las sentencias importantes son iguales. (HAY CASOS EN QUE NO SON IGUALES LOS NUMEROS HEXADECIMALES DEL WDASM Y EL SOFTICE Y ESO OCURRE CUANDO EL EJECUTABLE ESTA COMPRIMIDO, CON LO CUAL EL SOFTICE ES EL QUE NOS MARCA LA VERDAD DE LO QUE HAY EN MEMORIA Y PARA VER LO MISMO EN EL WDASM DEBEMOS DESCOMPRIMIR EL EJECUTABLE LO QUE SE ENSEÑARA EN FUTURAS LECCIONES)

¿Qué más podemos ver aquí?

Apretamos F2 y aparecen los valores de todos los registros en este instante.

Algo muy útil para cuando estemos en una sentencia CMP eax, ebx por ejemplo y queramos saber que está comparando exactamente.

También apretando F4 aparece congelada la pantalla de WINDOWS en ese preciso instante, ojo que esta tecla no es para volver a WINDOWS sino para mirar si apareció algún cartel a algo en el escritorio cuando ejecutamos alguna sentencia, volvemos al softice con F4 de nuevo. Más arriba aparece una banda horizontal que es el editor hexadecimal tiene posiciones de memoria y los bits en hexadecimal, con lo cual uno puede probar cambios en la memoria que después puede hacer en el ejecutable con el ULTRA EDIT.

Todos los cambios que hacemos con el softice son en la memoria y nos sirven para probar si al cambiar una cadena por otra en programa se comporta como nosotros queremos, pero si volvemos a cargar el programa de nuevo, nada cambio en el ejecutable, entonces hay que hacer los cambios que ya probamos que sirven en el SOFTICE con el ULTRAEDIT.

El editor hexadecimal no indica la misma posición de memoria que la que se está ejecutando y eso es porque yo puedo querer cambiar algún numero que este mas adelante en el programa y que no se esté ejecutando en este momento exacto.

Además en la misma línea horizontal a la derecha están los caracteres en modo texto (ASCII) por si queremos ver a qué carácter corresponde el numero hexadecimal.

Si alguno no puede ver alguna de las partes que acabo de mencionar, es probablemente porque en el WINICE.dat no está usando la misma línea INIT que uso yo, así que mejor cambie la que tiene por la que yo recomendé. (O TAMBIEN PUEDE ACTIVAR O DESACTIVAR PARTES CON LOS COMANDOS WC,WD,WF,WL,WR,WW,WS,WX al ejecutar estos comandos se activan las distintas partes del SOFTICE y si volvemos a ejecutar el mismo comando se desactivan nuevamente)

Por otro lado si alguien prefiere ver en vez de en este tipo de visualización, el SOFTICE funcionando en una ventana de WINDOWS, tiene que ir a INICIO-PROGRAMAS-NUMEGASOFTICE-DISPLAY ADAPTER SETUP y ahí poner la tilde en UNIVERSAL VIDEO ADAPTER y después clicar en test y salir de allí y reiniciar la maquina, por supuesto el funcionamiento es idéntico, hay que apretar la S exactamente igual al arrancar WINDOWS y para entrar y salir hay que teclear CTRL+D igual, a mí personalmente me cansa la vista un poco esta forma de visualizar pues la letra es mas chica, pero gustos son gustos. Además si queremos convertir un número de hexadecimal a decimal y ver también el carácter en modo texto ASCII el comando utilizado es:

? (NUMERO HEXA)

Por ejemplo si hacemos

? 47

0000047 000000 71 "G"

O sea el 47 HEXA corresponde al 71 decimal y al carácter G en ASCII

EMPECEMOS A HACER DESASTRES

Estábamos dentro del archivo crackme y paraditos en la sentencia 401002 si hacemos F10 nuevamente vemos que ejecuta el CALL que esta allí y sigue con la próxima sentencia, en realidad el comando F10 ejecuta una sentencia pero no entra a ver dentro de los CALL cual es el contenido de la subrutina y ejecutarla paso por paso si quisiéramos entrar a ejecutar paso por paso alguna rutina deberíamos teclear F8 en vez de F10, pero como nosotros no queremos investigar el CALL no entramos , simplemente hago mención por si alguna vez lo necesitamos, sepamos como hacer.

Podríamos ir si quisiéramos haciendo F10, miles de veces hasta que el programa arranque pero como chino no me quiero volver, vamos a dejar que siga ejecutándose el programa y vamos a poner un punto de interrupción para que cuando el programa llegue a ese punto se detenga allí y vuelva al SOFTICE.

Como ya sabemos por haber estudiado el WDASM cual puede ser el salto sospechoso, vamos a poner un punto de interrupción allí.

De cualquier manera yo puedo llegar a ese mismo salto sin haber utilizado para nada el WDASM pero como un cracker se ayuda de todo lo que tiene a mano, vamos a probar el salto ese, mas adelante les voy a enseñar cómo llegar a encontrar el salto directamente con el SOFTICE.

Para poner un punto de interrupción se utiliza el comando BPX que quiere decir breakpoint o sea tecleamos en la línea de comandos.

BPX 401243

Que era el salto sospechoso y ENTER (DESDE AHORA EN MAS CUANDO DIGO TECLEAR UN COMANDO SE SOBREENTIENDE QUE HAY QUE TECLEAR ENTER AL FIANALIZAR DE ESCRIBIR EL MISMO)

Y para que el programa continúe ejecutándose tecleo X y entonces el programa sigue funcionando y nos aparece la ventana del CRACKME, donde entramos en HELP y luego REGISTER y ponemos un nombre compuesto de todas letras por ahora y un número de serie cualquiera, y le damos a Ok y entonces el SOFTICE interrumpe el programa gracias al breakpoint que hemos puesto, entonces ahora la línea blanca esta en

401243 74 07 jz 0040124c NO JUMP!

Aquí el salto que en el WDASM era je, es llamado jz que es lo mismo en realidad, porque JE quiere decir que saltan si son iguales y JZ que salta si la diferencia entre los dos es cero lo que ocurre cuando son iguales, es diferencia de notación pero los números hexadecimales son los mismos 74 07 y al lado aparece la decisión que tomo el programa con respecto a la comparación y al salto o sea NO JUMP (NO VA A SALTAR)

Vemos que la secuencia de números hexadecimales que habíamos encontrado en el WDASM

74 07 E8 18 01 00 00 EB 9A

Aquí es la misma también ya que si seguimos copiando los números que están al lado de las sentencias siguientes vemos que después de 74 07 viene el CALL 401362 cuyos números hexadecimales son

E8 18 01 00 00

CON LO QUE UNIENDO LOS DOS EL **74 07** DEL SALTO Y EL **E8 18 01 00 00** DEL CALL TENEMOS LA CADENA CASI COMPLETA (podemos seguir unos números más por si acaso mirando la próxima sentencia)

Si queremos que en el editor hexadecimal aparezca la cadena a modificar usamos el comando E seguido de la dirección de memoria a partir de la cual queremos que se visualicen los números HEXA, tecleamos entonces:

E 401243

Y el cursor quedara titilando justo encima del 74 que hay que cambiar, si ahí tecleamos EB y después ENTER vemos que automáticamente la sentencia JZ cambio por JUMP y ahora cambio la decisión del programa NO JUMP! por JUMP! o sea que va a saltar y va a hacer en la memoria la prueba de lo que sucedería si nosotros cambiáramos los bits con el ULTRAEDIT en el ejecutable. Tecleamos X y entonces

GREAT WORK MATE!

Quiere decir que la prueba fue exitosa y nos registra, entonces podemos ir al ultraedit buscar la cadena y cambiarla en el EJECUTABLE para que quede definitivo, igual por ahora no lo hagan.

Cierren el CRACKME y cierren la ventana del SYMBOL LOADER.

Ejecuten el crackme normalmente sin abrirlo desde el SYMBOL LOADER y cuando aparezca la pantalla de registro pongan un nombre de letras y la clave de números.

Pongan Ok y listo, otra vez el softice detiene el programa en la misma sentencia, ya que como el BPX quedo en la memoria sigue funcionando y va a parar el programa ahí siempre, a menos que lo borremos.

Por ahora dejémoslo, igual el comando que borra los BREAKPOINTS es BC con eso se borran todos los breakpoints que pusimos, mas adelante lo haremos.

Hay otra forma también simple de probar si el salto al cambiarlo funciona y se usa generalmente cuando hay programas que tienen varios saltos dudosos y para no perder tiempo editando nada.

En la ventana superior de los registros se encuentra el registro EIP, este registro es muy importante porque es el que muestra cual es la próxima sentencia que va a ejecutar el programa que en nuestro caso es la de la línea blanca o sea 401243.

¿Se puede editar el registro EIP para que el programa salte adonde yo quiero?

¿Qué no se puede hacer con SOFTICE?

El comando para editar un registro es R seguido del registro que quiero cambiar y el signo igual y el nuevo valor que le quiero dar.

O sea R EIP=40124c que es el lugar adonde quiero que salte, de esa forma salto sin cambiar la sentencia a ver qué pasa y QUE PASA?

GRAT WORK MATE!

Otra forma de cambiar el salto es editar directamente la sentencia ensamblador eso se hace con el comando A

A 401243

Y cuando aparece el cursor titilando escribo la sentencia que reemplace al salto JZ escribo

JMP 40124c

Y cuando me aparece para escribir la sentencia siguiente tecleo ESC para salir del editor de sentencias.

Con esto editamos directamente la sentencia (MUY UTIL CUANDO UNO NO CONOCE EL CODIGO HEXADECIMAL DE UNA SENTENCIA)

Si no se qué numero corresponde a JMP 30124c al usar el comando A lo cambio directamente aunque no sepa que numero debe reemplazarse y después me fijo que numero hexadecimal quedo en el lugar de JMP y es EB como sabemos.

Bueno ya sabemos los métodos para reemplazar un salto en la memoria y probar si es correcto antes de hacer los cambios con el ULTRAEDIT.

El otro salto es igual, una vez que para en 401243, pongo otro BPX en el otro salto; pero para estudiarlo un poco mas pongo el BPX un poco antes o sea en 401382

BPX 401382

Una vez que pongo el nombre (USEMOS PEPE) y el número de serie (989898) el programa para en

401382

Vemos que está usando algo en el registro ESI, para ver si hay algún texto allí tecleamos

D ESI

Y en la ventana de texto ASCII nos aparece la palabra PEPE y los números hexadecimales que corresponden a ese texto son 50 45 50 45, es evidente que va a testear como dijimos si todos los caracteres que tecleamos son letras, ¿como hace?

Tecleo F10 para ejecutar la sentencia.

La próxima sentencia MOV AL, [ESI] mueve el primer carácter del contenido de ESI hacia AL o sea que una vez que tecleo F10 puedo fijarme el valor que toma AL.

? AL y toma el valor 50 que corresponde a la letra P.

Luego testea TEST AL, AL si es cero y como no es cero en el salto subsiguiente pone NO JUMP.

Luego compara AL con 41

Para ver que letras son ambos tecleamos

? AL

Me dice que es "P"

? 41

Me dice que es "A"

O sea que esa comparando si la letra es mayor que A (SI ES MENOR TE TIRA A NO LUCK THERE MATE) y si es mayor sigue compara si es menor que Z y si es mayor la transforma en MAYUSCULA y SIGUE.

Aquí también podemos probar el salto vemos que sin cambiar nada cuando pusimos una letra la llegar al salto critico nos aparece NO JUMP en cambio cuando ponemos un numero (EN EL NOMBRE) nos aparece JUMP o sea que nos va a tirar a NO LUCK THREE MATE.

Parcheando el salto con el editor hexadecimal, cambiándolo por 90 90 ó poniendo R EIP=40138d que es la sentencia siguiente o poniendo A 40138b NOP (ENTER) y nuevamente NOP (ENTER) parchearemos este salto y evitaremos este NO LUCK THERE MATE por poner números en vez de letras.

OJO QUE DESPUES LES VA A APARECER EL OTRO Y TIENEN QUE CAMBIARLO TAMBIEN PARA EVITAR EL OTRO CARTELITO.

ESTE ES LA PRIMERA PARTE DE LA LECCION DE SOFTICE (UN PRIMER PANTALLAZO PARA QUE LO VAYAN USANDO), OBVIAMENTE ES UN PROGRAMA QUE TIENE TANTAS POSIBILIDADES QUE ES MUY DIFICIL EN UNA SOLA LECCION EXPLICARLAS POR LO QUE SEGUIREMOS EN UNA SEGUNDA PARTE DONDE LES EXPLICARE COMO CRAKEAR SOLO CON SOFTICE Y EL RESTO DE LAS POSIBILIDADES DE INTERRUPCION QUE TIENE ESTE MARAVILLOSO PROGRAMA Y LA FORMA DE HALLAR NUMEROS DE SERIE CON SOFTICE.

PRACTIQUEN INVERTIR LOS SALTOS EN EL MIXIIIxa, bájelo de nuevo, la versión sin parchear instálenlo de nuevo y prueben invertir los saltos con los tres métodos conocidos.

AU REVOIR

Ricardo Narvaja

DESCARGADO GRATUITAMENTE DE
<http://visualinformatica.blogspot.com>

COMO UTILIZAR EL SOFTICE CON LAS FUNCIONES DE WINDOWS (api)

(LECCIÓN TRES)

El softice acepta además de los BPX numéricos otros que son BPX a funciones que utiliza WINDOWS para realizar ciertos trabajos que los programas aprovechan para no escribir tantas funciones cuando ya se puede realizar ese trabajo con funciones de WINDOWS.

Esto es muy útil para los crackers para que el programa se interrumpa cuando llama a estas funciones y con ello ayuda cuando el WDASM no nos dice mucho o cuando no aparecen las STRINGS REFERENCES que buscamos.

Por ejemplo en el crackme nosotros buscamos la STRING REFERENCE -NO LUCK THERE MATE- y allí no hubo problema porque aparecía claramente la STRING REFERENCE y entonces la buscamos así, pero con el softice podríamos también arrancar el programa ir a HELP - REGISTER y poner el nombre pepe y la clave 989898 (O CUALQUIER OTRA) y antes de poner OK hacemos CTRL. + D y en el SOFTICE tipeamos

BPX MESSAGEBOXA y después volvemos al programa con X y después si ponemos OK y el programa se interrumpe cuando se comienza ejecutar la función MESSAGEBOX que dibuja esas ventanitas con mensajes y entonces tecleamos F12 para que se ejecute toda esa función vemos que aparece en 40137d justo en la zona de chico malo que nos decía el WDASM y si ejecutamos F10 para que se ejecute el RET vemos que vuelve a la zona donde esta la comparación y el salto a la zona de chico malo que habíamos llegado con el WDASM.

Además si volvemos a repetir el proceso y antes de hacer el ret borramos todos los BPX con BC* y hacemos doble click encima de las sentencias 40136b y 401370 hasta que se pongan celeste (ESTO EQUIVALE A PONER UN BPX EN ESA POSICIÓN) y ejecutamos de nuevo el crackme y HELP-REGISTER ETC ETC va a parar en esas sentencias ahí podemos ver haciendo

```
D 401136b      VEMOS NO LUCK
D 401370      VEMOS NO LUCK THERE MATE
```

O sea vemos como está preparando los mensajes para cuando se ejecute la messagebox aparezcan los odiosos NO LUCK y NO LUCK THERE MATE y si editamos esos mensajes con E y ponemos otros valores hexadecimales que correspondan a otras letras van a ver que al seguir el programa van a aparecer en el cartelito el texto que ustedes pongan en vez de NO LUCK THERE MATE.

En el WDASM las funciones importadas que utiliza el programa se pueden ver en el iconito IMP FN que está a la izquierda del de STRING REFERENCES cinco o seis iconos a la izquierda del de STRING, allí está la lista de funciones utilizadas y haciendo doble clic en alguna de ellas se puede ver cuando las utiliza.

Si hacemos allí clic en USER32.MESSAGEBOXA va a tirarnos donde se usa esa función dentro del programa.

Existen muchas funciones y varias son las más utilizadas por los crackers, además de MESSAGEBOXA, esta también MESSAGEBOX sin la A que es la misma función equivalente para programas de 16 bits .

También hay funciones bastante parecidas

bpx MessageBox
bpx MessageBoxExA
bpx MessageBeep
bpx SendMessage

bpx DialogBoxParamA
bpx CreateWindow
bpx CreateWindowEx
bpx ShowWindow
bpx UpdateWindow

bpx GetDlgItemText
bpx GetDlgItemInt
bpx GetWindowText
bpx GetWindowTextWord
bpx GetWindowTextInt

Para programas con vencimiento de tiempo se usan estas funciones para que el programa tome la hora o la fecha del sistema

bpx GetLocalTime
bpx GetFileTime
bpx GetSystemtime

Entonces se pone un bpx en alguna de estas funciones y después cuando compara la fecha con la fecha en que el programa tiene que vencer se parchea el salto. (VEREMOS MAS ADELANTE EJEMPLOS DE ESTO)

Después esta para propósitos generales

BPX HMEMCPY

Para crackear números de serie

BPX LSTRCMPA
BPX MULTIBYTETOWIDECHAR
BPX WIDECHARTOMULTIBYTE

Generalmente dentro de estas funciones en la primera se comparan cadenas de texto y en las otras dos se transforman cadenas de texto de tipo normal a ancho o sea que si la cadena normal es

787878 si le aplicas la segunda función se transforma en 7.8.7.8.7.8 que se llama formato ancho y es muy utilizada en las comparaciones, después generalmente con D en las posiciones de memoria dentro de la función aparecerá la cadena de texto o la clave (MAS ADELANTE VEREMOS EJEMPLOS)

Iremos viendo en sucesivos crackeos que iremos estudiando cómo se utilizan estas funciones y como aprovechar sus utilidades para nuestro provecho.

Vemos también que pueden practicar con el MIXIIxa que también utiliza BPX MESSAGEBOX lo único que hay que tener en cuenta es que cuando ponemos el BPX MESSAGEBOX y cuando rompe en el softice apretamos F12 y después ponemos OMITIR y vuelve a romper en el SOFTICE esta vez la línea verde que nos muestra que programa se esta ejecutando dice MFC42 por lo cual tenemos que hacer F12 tantas veces como sea necesario para llegar al ejecutable, en este caso con una sola vez que tecleemos F12 vuelve a aparecer el MIXxaIII en la parte que habíamos estudiado con el WDASM y un poco mas arriba están los saltos a parchear.

Un ejemplo de cómo utilizar la función BPX HMEMCPY es el programa IDESK versión 2,60 que se puede bajar del FREEDRIVE

<http://www.freedrive.com/ASP/PostFolderShortcut.asp?fsc=8737470>

esta es la otra carpeta del FREEDRIVE donde hay algunos programas para crackear.

Este programa cuando lo instalamos se instala en C:/INTERNET DESK y allí esta el ejecutable IDESK.exe, si lo desensamblamos con el WDASM vamos a ver que no aparecen STRING REFERENCES ni funciones importadas (POSIBLEMENTE EL EJECUTABLE ESTE COMPRIMIDO LO CUAL APRENDEREMOS A DESCOMPRIMIR MAS ADELANTE PERO IGUAL LO VAMOS A CRACKEAR)

Ejecutamos el programa y cuando arranca vamos a donde está el SIGNO DE INTERROGACIÓN - ACERCA DE- REGISTRAR y ahí nos aparece una ventanita con un lugar para poner el número de serie y el nombre de usuario.

Si probamos con el softice vemos que BPX MESSAGEBOX NI MESSAGEBOXA son ya que ahí no para el programa. Entonces hacemos lo siguiente:

Escribimos un numero de validación cualquiera por ejemplo 989898 (POR AHORA USEN ESTE NUMERO) y un nombre de usuario por ejemplo: pepe y antes de apretar REGISTRAR hacemos CTRL.+D y tecleamos BPX HMEMCPY y luego X para volver al programa y ahí si ponemos REGISTRAR y BUUUUUUUUMP adentro.

Tecleamos varias veces F12 (SIETE EXACTAMENTE HASTA VOLVER AL IDESK FIJARSE LA LINEA VERDE DONDE DICE EL NOMBRE DEL PROGRAMA EJECUTADO), y ahí seguimos ejecutando con F10 sentencia por sentencia hasta que ejecuta todos esos RET y POP y vuelve a la parte buena del programa 4c9d7b.

Ahí podemos borrar todos los BPX con BC* y poner un BPX 4c9d7b para que si volvemos a ejecutar el programa no tengamos que hacer todo ese proceso y paremos directamente aquí.

Si seguimos tecleando F10 vamos a ver varios saltos que si los invertimos no pasa nada hasta que llegamos a la comparación y el salto de 4c9dc0 y 4c9dc6 allí esta el salto que al invertirlo nos registra pero no lo hagan ya que ahí hay más sorpresas podemos sacar el número de serie para el nombre pepe (O PARA EL QUE HAYAN PUESTO USTEDES)

Cuando estamos encima de la comparación CMP EAX,[52b504]

Esta sentencia compara EAX con el CONTENIDO de 52b504 como estamos justo encima de la sentencia el SOFTICE y allí hay una operación que es hallar el CONTENIDO DE 52b504 siempre que una sentencia tiene una operación y estamos justo encima de ella el

SOFTICE nos muestra en CELESTE el resultado de la operación, si vemos en la esquina superior derecha vemos

DS:52B504 =F1ACA

Veamos que es F1ACA

? F1ACA = 989898

O SEA QUE ESTA COMPARANDO ALGO CON 989898 que es el numero trucho de serie que yo puse y que puede comparar con el numero trucho 989898 EL NUMERO DE SERIE VERDADERO o sea que si hacemos ? EAX tenemos el número de serie que corresponde a nuestra maquina y a el nombre que hayamos puesto.

En general el método de HMEMCPY es bastante usado para hallar números de serie o saltos cuando el WDASM no ayuda mucho o en programas VISUAL BASIC o DELPHI también.

Siempre hay que volver al programa con F12 varias veces y después seguir ejecutando F10 hasta llegar a la parte de comparaciones y saltos.

En realidad las funciones de WINDOWS nos permiten interrumpir la ejecución de un programa casi cuando queramos, a medida que vayamos avanzando veremos otras funciones y como nos pueden ayudar a crackear programas distintos.

Ricardo Narvaja

DESCARGADO GRATUITAMENTE DE
<http://visualinformatica.blogspot.com>

COMO USAR EL RISC PROCESS PATCHER PARA HACER UN CRACK **DISTRIBUIBLE**

(LECCION CUATRO)

El titulo nos dice algo muy importante, como puedo distribuir cracks si no utilizo un programa que pueda hacer los cambios que yo hago con el ULTRAEDIT o en el SOFTICE igual en otras maquinas sin tener que enseñarles a usar el SOFTICE ni el ULTRA EDIT, simplemente con un programita que genere un cambio que pueda ser usado en todas las maquinas.

Para eso vamos a usar el programita RISC PROCESS PATCHER que bajamos junto con las herramientas, es un programa fácil de usar y entender cómo se usa y sirve para TODOS los casos sabiéndolo usar.

¿A QUÉ ME REFIERO CON TODOS LOS CASOS?

En el caso de ejecutables o archivos que pueden ser desensamblados con WDASM y se pueden encontrar los valores a cambiar en el ULTRAEDIT no hay problema para ningún parcheador que hay en el mercado, todos funcionan.

Pero el problema se da en programas como el IDESK o muchísimos que existen actualmente que vienen empaquetados o comprimidos y los parcheadores tradicionales no sirven porque lo que encontramos en el SOFTICE y cambiamos y funciona , no podemos encontrarlo en el WDASM ni en el ULTRAEDIT con lo cual solo podemos hacer los cambios en memoria con el SOFTICE y no podríamos realizarlos ni encontrarlos en el ejecutable ya que este está comprimido y para encontrar los números a cambiar habría que descomprimirlo lo cual es una lección que aprenderemos más adelante ya que es toda una ciencia (OJO NO ES DESCOMPRIMIR CON WINZIP, LOS PROGRAMADORES USAN PROGRAMAS ESPECIALES DE COMPRESIÓN QUE NO SE DESEMPAQUETAN ASI NOMAS Y APRENDER A DESCOMPRIMIRLOS LLEVARA UNA O DOS LECCIONES MAS ADELANTE)

Igualmente con el RISC PROCESS PATCHER podemos parchear programas comprimidos, descomprimidos y de cualquier tipo.

¿COMO TRABAJA ESTE PROGRAMITA?

Muy sencillo. Los parcheadores tradicionales localizan el ejecutable y hacen lo que nosotros hacemos con el ULTRAEDIT solo que automáticamente, localizan los números a cambiar y los reemplazan y listo.

Pero en los archivos comprimidos esto no sirve ya que las cadenas halladas con el SOFTICE no las vamos a encontrar. Aquí viene a tallar éste maravilloso programita.

El RISC PROCESS PATCHER es un cargador que no cambia ni un solo numero en el ejecutable, NADA NADA, con lo que nos evitamos ciertas protecciones de programas que chequean si la sumatoria de todos los números que lo componen fue modificada y dejan de funcionar.

¿Y SI NO MODIFICA EL EJECUTABLE COMO FUNCIONA?

Carga el ejecutable y espera que se descomprima o arranque solamente según el caso y entonces cambia los valores que nosotros queremos en la memoria directamente y sigue funcionando el programa normalmente.

O sea que es un cargador que hay que copiar en la carpeta de instalación del programa y arrancarlo siempre desde allí, ya que si arrancamos el programa desde su acceso directo no se producirá ningún cambio y funcionara como si no lo hubiésemos crackeado.

¿Y CÓMO HACEMOS PARA LOGRAR EL CRACK?

Una vez que descomprimos el RISC PROCESS PATCHER, descomprimos también el archivo ZIP SCRIPTS que se encuentra dentro del mismo y que va a crear una carpeta que se llama SCRIPTS.

Les RECOMIENDO que no esté funcionando el SOFTICE cuando usen el RPP ya que no son compatibles y a veces se cuelga la maquina.

Allí dentro de la carpeta SCRIPTS hay varios archivos con extensión rpp que son los que uno tiene que editar con el BLOC DE NOTAS para crear el crack.

Si no se abren con el BLOC DE NOTAS hacer clic derecho encima de alguno de ellos teniendo la tecla SHIFT apretada y cuando aparece el menú elegir ABRIR CON y allí seleccionar el BLOC DE NOTAS (NOTEPAD) y poner la tilde el ABRIR SIEMPRE CON ESTE PROGRAMA y listo, siempre se abrirán con el NOTEPAD.

Entonces una vez que encontramos los valores que debemos cambiar en el SOFTICE anotamos la posición de memoria en que esta el valor en un papel y los valores HEXADECIMALES correspondientes que debemos cambiar probados en el SOFTICE que funcionan antes de ir al RISC.

Veamos unos ejemplos de los SCRIPTS que tengo yo del RISC PROCESS PATCHER.

ESTO ES LO QUE DICE EL SCRIPT PARA CRACKEAR EL WINBABEL

; winbabel:

```
F=Winbabel.exe: ; PROCESS TO PATCH  
O=crack_the_winbabel.EXE : ; LOADER TO CREATE  
P=786d/74/eb: ;  
P=5fa4/74/EB: ;  
P=5faa/75/EB: ;  
$ ;end of script
```

La primera sentencia uno pone el nombre del programa a crackear para no olvidarse (COMO HAY UN PUNTO Y COMA AL PRINCIPIO ESTO NO TIENE NINGUN EFECTO)

Después de la letra F= tengo que poner el nombre del ejecutable a parchear.

Después de la letra O= tengo que poner con que nombre quiero que se grabe el archívito crack

Después de la P= pongo la dirección de memoria donde empiezo cambiar valores y si es un solo valor a cambiar pongo la barra y después el valor que existe en la memoria y debo reemplazar y después la barra y el valor que yo quiero que vaya allí reemplazando al anterior. En el ejemplo en la posición de memoria 786d en la cual el SOFTICE me dice que hay un valor de 74 yo lo quiero reemplazar por EB.

Las tres P siguientes son el mismo caso que el anterior ya que en este programa hay que reemplazar tres valores para crackearlo.

Y al final de todo \$; end of the script para terminar.

Hay que acordarse siempre de los dos puntos que lleva cada sentencia al final y que las aclaraciones deben ir después de un punto y coma (COMO POR EJEMPLO PROCESS TO PATCH)

Una vez que hice el archivito con este sistema y lo reviso que esta correcto, ejecuto el programa RPP y me pide que script quiero usar y le pongo el nombre del que cree y entonces acepto y me genera el crack con el nombre CRACK THE WINBABEL.exe dentro de la carpeta de los scripts y lo copio a la carpeta del programa y lo ejecuto, si todo está bien hecho debe funcionar perfectamente.

Me olvide de decirles que si hay que reemplazar varios números consecutivos en la sentencia P se debe escribir como en el siguiente ejemplo.

CRACK PARA EL TURBOGO

; TURBOGO crack

```
F=Turbogo.exe: ; PROCESS TO PATCH  
O=crack_the_turbogo1.exe: ; LOADER TO CREATE  
P=473f4b/0F,84,2B,01,00,00/90,90,90,90,90,90: ;  
P=48b925/75,46/90,90: ;  
$ ;end of script
```

Después de la P pongo la primera posición de memoria desde donde comienza la secuencia a cambiar en este caso 473f4b y después pongo la barra y los valores existentes y consecutivos que encontré en el SOFTICE separados por comas, una vez que termine la cadena a reemplazar pongo la barra de Nuevo y después la secuencia de valores que reemplazan a los anteriores también separados por comas y al final los dos puntos.

OJO LOS VALORES QUE DEBEN SER REEMPLAZADOS DEBEN TENER LA MISMA CANTIDAD DE NUMEROS QUE LOS QUE LOS REEMPLAZAN EN ESTE CASO SON 0f 84 2b 01 00 00 que son seis números y los reemplazan 90 90 90 90 90 90 que son seis números también.

¿CUÁLES PUEDEN SER LOS ERRORES AL HACER EL SCRIPT?

A VECES NOS OLVIDAMOS LOS DOS PUNTOS o EL PUNTO Y COMA Y CUANDO EJECUTAMOS EL RISC NOS DA ERROR PORQUE NO ENCUENTRA EL SCRIPT BIEN HECHO.

SI CUANDO LO COPIAMOS EN LA CARPETA DEL PROGRAMA A CRACKEAR Y LO EJECUTAMOS DICE QUE NO ENCUENTRA EL PROCESO SE DEBE A QUE:
O LO COPIAMOS EN CUALQUIR LADO EN VEZ DE EN LA CARPETA DONDE ESTA EL EJECUTABLE A PARCHEAR, O LOS VALORES QUE ESTAMOS REEMPLAZANDO NO LOS ENCUENTRA YA QUE LOS COPIAMOS MAL O NO PROBAMOS BIEN QUE ESTEN EN EL SOFTICE Y QUE AL REEMPLAZARLOS FUNCIONEN.

AQUÍ ESTA EL SCRIPT DEL CUENTAPASOS 3,78

; CUENTAPASOS 3,78

```
F=cposos32.exe: ; PROCESS TO PATCH  
O=crack_the_cuentapasos2.exe: ; LOADER TO CREATE  
P=4b1dcd/AE/bf: ;  
P=4b1a7b/AE/bf:  
$ ;end of script
```

CON ESTO EL PROGRAMA NO VENCE PERO EXISTE EL PROBLEMA QUE EL PROGRAMA SE CIERRA SOLO DESPUÉS DE ALGUNOS MINUTOS DE FUNCIONAR AL DETECTAR EL SOFTICE (AUNQUE NO LO CARGUEMOS) O CUALQUIER HERRAMIENTA CRACK.

AQUÍ ESTA EL SCRIPT DEL REVERSO EXPRESS

; REVERSO EXPRESS

```
F=register.exe: ; PROCESS TO PATCH  
O=crack_the_REVERSO.exe: ; LOADER TO CREATE  
P=402c5b/82,e0,02,00,00/45,f0,90,90,90: ;  
$ ;end of script
```

OJO QUE SIRVE PARA LA VERSIÓN EN LA CUAL PODES PONER UN NUMERO PARA REGISTRARTE, EN LA QUE LA VENTANA DE REGISTRACION APARECE GRIS NO FUNCIONA.

Ustedes pueden probar en hacer un crack para el crackme que acepte cualquier numero cuando lo cargamos desde el crack con los valores que hallaron cuando lo crackearon. Lo que hay que fijarse es que en la sentencia P hay que poner los valores que se deben reemplazar exactos y no la cadena completa que habíamos encontrado ya que la habíamos alargado un poco para buscarla con el ULTRAEDIT, aquí tiene que tener de largo exactamente los números a reemplazar y la cadena que reemplaza debe tener el mismo largo exacto.

Ricardo Narvaja

DESCARGADO GRATUITAMENTE DE
<http://visualinformatica.blogspot.com>

COMO DESEMPAQUETAR CON PROCDUMP AUTOMÁTICAMENTE

(LECCION CINCO)

Como hemos visto en lecciones anteriores y por si no se acuerdan hay programas que no se pueden visualizar con WDASM lo que hace mas dificil su crackeo, inclusive si encontramos algun valor en el SOFTICE con el cual crackear el programa, si esta comprimido no podremos encontrar los mismos valores en el ULTRAEDIT ya que el programa se descomprime y luego salta al inicio real del programa con lo cual si queremos verdaderamente analizarlo deberemos descomprimirlo o desempaquetarlo.

Atención de que aquí cuando hablamos de compresión hablamos de ciertas programas que utilizan los programadores para comprimir un ejecutable manteniendo la misma extension exe no de compresiones con WINZIP ni WINRAR ni nada de eso.

Este tipo de compresiones se realiza mas que para ahorrar espacio para proteger el ejecutable de que alguien lo pueda alterar aunque según hemos visto en la lección anterior aunque un ejecutable este comprimido si encuentro los valores a cambiar en el SOFTICE puedo hacer un cargador con el RISC PROCESS PATCHER y crackearlo de cualquier manera, igual se recomienda siempre para facilitar el crackeo lograr descomprimir el ejecutable con lo que podremos verlo en el WDASM así como sus STRING REFERENCES y nos orientaremos mejor.

También si logramos descomprimir bien el ejecutable y además de que nos sirva como referencia podemos reemplazar el ejecutable comprimido por el descomprimido y si funciona el programa, también podremos realizar los cambios con el ULTRAEDIT en la forma convencional ya estudiada.

Existen varias formas de descomprimir un ejecutable en esta lección veremos la forma automática de descompresión con el PROCDUMP, la cual sirve en ciertos casos y cuando no sirve hay que apelar a lo que estudiaremos la semana que viene la DESCOMPRESION MANUAL CON PROCDUMP.

Primero que nada y antes de utilizar el PROCDUMP y de realizar ningún cambio en el ejecutable les aconsejo fuertemente hacer una copia del ejecutable original y guardarlo en algún lado seguro, porque podemos arruinar el original y que no funcione y tendremos que bajar de nuevo el programa.

¿CUANDO PODEMOS SOSPECHAR QUE UN ARCHIVO ESTA COMPRIMIDO o EMPAQUETADO?

Cuando al tratar de verlo con el WDASM no muestra nada coherente no muestra STRING REFERENCES ni nada, ni IMPORTED FUNCTIONS la verdad una basura completa.

Otra forma de darnos cuenta es arrancar en programa y poner algún BPX MESSAGEBOX o BPX SHOWWINDOW o BPX HMEMCPY o cualquiera que funcione y volver al ejecutable apretando varias veces F12, cuando estamos en el ejecutable copiamos una cadena de 10 o 12 números y después la buscamos en el ULTRAEDIT y si no la encuentra síntoma claro de que estamos ante algo comprimido.

EL ALIADO DEL PROCDUMP EL GETTYPE

El gettype es un pequeño utilitario que nos ayuda en varias cosas para poder utilizar el PROCDUMP perfectamente, existen dos versiones una para DOS y otra para WINDOWS, una vez que lo descomprimimos en una carpeta, copiamos el ejecutable a estudiar a la misma carpeta, en este caso vamos a estudiar el WINOMEGA. ESTE PROGRAMA SE PUEDE BAJAR DEL FRREDRIVE DEL CURSO.

Lo instalo y al tratar de desensamblar el ejecutable con el WDASM no sale absolutamente nada ni STRING REFERENCES NADA DE NADA ni siquiera sentencias NADA todo basura.

Lo copio a la carpeta donde está el GETTYPE para analizarlo y me fijo como se llama el ejecutable del GETTYPE, hago esto porque el nombre del ejecutable varía según la versión existen versiones en las que el ejecutable se llama GTW.EXE en otros GT014.EXE en otros GT015.EXE etcétera.

En mi caso es GT014.EXE entonces abro una ventana DOS y me sitúo con CD hasta que llego al directorio donde esta el gettype o sea si estoy en C:/WINDOWS en mi caso tendría que hacer CD ESCRITORIO y después CD GT014 con lo que ya estoy situado en mi caso en el directorio del GETYPE (EN OTROS CASOS HABRA QUE TIPEAR DISTINTOS NOMBRES DE DIRECTORIOS HASTA LLEGAR AL GETYPE) Cuando llego allí tecleo:

```
GT014 WINOMEGA.EXE
```

Y nos dice que esta comprimido con BORLAND DELPHI 3/4 esto no es un compresor por lo tanto esta información no nos sirve, a veces el GETTYPE cuando no encuentra el compresor dice cualquier cosa, la idea era que el GETYPE nos dijera con que compresor estaba empaquetado para después sabiendo eso descomprimirlo automáticamente con PROCDUMP. Es probable que el compresor utilizado sea más nuevo que el gettype y este no lo conozca.

En éste caso vamos a ver si podemos hacer algo con el PROCDUMP solo.

Abrimos el PROCDUMP y allí podemos ver una lista a la izquierda de los procesos que están ejecutándose en este momento, por ahora no utilizaremos esto, si supiéramos que compresor fue utilizado iríamos a UNPACK y elegiríamos el compresor de la lista que nos sale y después nos pide buscar el ejecutable para cargarlo (OJO TIENE QUE SER EL ORIGINAL EN LA POSICIÓN ORIGINAL YA QUE CARGA EL PROGRAMA Y SI ESTA SUELTO NO VA A FUNCIONAR) como no sabemos podríamos probar con UNKNOW o sea desconocido pero no es seguro que funcione, hagamos otro intento.

Vamos a PE EDITOR, y allí cargamos el ejecutable original, luego vemos que aparecen distintos datos vamos a SECTIONS, allí vemos en la columna NAMES que estamos de suerte ya que el ultimo nombre es WWP32 que es el nombre del compresor utilizado, esto no siempre aparece por eso cuando no sabemos qué compresor es el utilizado porque el gettype no nos lo dice ni aparece en el PROCDUMP tenemos que hacerlo manualmente como explicaremos en la lección 6.

Entonces sabemos que el compresor es el WWP32, abrimos UNPACK buscamos el ultimo de la lista que es el WWP32 y cargamos el ejecutable de C:/ARCHIVOS DE PROGRAMAS/WINOMEGA/WINOMEGA.EXE y lo cargamos nos va a aparecer un cartel diciendo que hagamos clic cuando el programa arranque totalmente, entonces esperamos a que el programa arranque y cuando lo hace hacemos clic en esa ventana y va a

comenzar a tratar de descomprimirlo, en unos minutos el ejecutable tiene que terminar de descomprimirse y nos va a aparecer una ventana que nos pregunta donde queremos guardar el ejecutable descomprimido y con qué nombre (NO MODIFICA EL ORIGINAL SINO QUE CREA UN NUEVO ARCHIVO), lo guardamos y listo, descomprimido.

Si tarda mucho en descomprimir y sigue más de 5 o 6 minutos y no termina nunca significa que no lo pudo descomprimir, yo tengo dos versiones de PROCDUMP una desactualizada que no lo descomprime automáticamente y sigue siempre trabajando, la mas nueva tiene dos versiones del WWP32 la 1 y la 2 con la 2 se descomprime perfectamente. Si tienen la versión desactualizada en las paginas donde se bajan las herramientas suele haber un archivo de actualización del PROCDUMP.(OJO QUE ATUALIZA LOS COMPRESORES NO CAMBIA EL NUMERO DE LA VERSIÓN ESTA SIGUE SIENDO 1.6.2)

La versión vieja del PROCDUMP tiene 30 descompresores y si la actualizamos tiene 33 pero reemplaza antiguas versiones de algunos descompresores por las nuevas con lo que es mas seguro que funcione con programas nuevos.

Abrimos ahora el WDASM y cargamos el ejecutable descomprimido y se puede descomprimir perfectamente salen las STRING REFERENCES perfectamente, en algunos casos existe una pequeña protección adicional para que a pesar de haber descomprimido bien , igual no se pueda desensamblar.

En ese caso hacer exactamente esto.

- 1.- Volver a Procdump
- 2.- Seleccionar PE-edit
- 3.- Seleccionar el ejecutable desempquetado winomega.exe
- 4.- Darle a Sections
- 5.- Ante el nuevo apartado text, .rdata, .data, .rsrc,...Clicar en el primero de ellos con el botón derecho y darle a Edit Section (en nuestro caso: .text) (no siempre .text será el primero de la lista. Da igual. Hay que seleccionar al primero de ellos)
- 6.- En el apartado Section Characteristics cambiar el C0000040 por E0000020.

Esta es la protección antidesensamblado, en algunos programas viejos que no están comprimidos, solamente vienen equipados con esta simple protección antidesensamblado, no es necesario, saber porque ocurre esto, es así, si dice C0000040 no se puede desensamblar y si dice E0000020 si podremos desensamblar con el WDASM.

En este caso a mi no me fue necesario salió perfecto como chorizo.

Ahora la prueba de fuego reemplazar el ejecutable original por el desempquetado, esto funciona a veces ya que si vemos el ejecutable original tiene algo más de 1 mega y el desempaquetado mas de 3 megas por lo tanto el programa puede llegar a testear el largo del ejecutable y al ver que es más largo no funcionar, probemos.

Tenemos que guardar el original en otro lado por si acaso, y copiamos el desempquetado en la carpeta del programa nos fijamos que tengan el mismo nombre original y lo ejecutamos. FUNCIONA.

De cualquier manera si no hubiera funcionado nos hubiera servido para ver en el WDASM las STRING REFERENCES las IMPORTED FUNTIONS y la posibilidad de ojear el listado muerto del programa siempre ayuda.

En el caso de que no podamos descomprimir el ejecutable en forma automática con el PROCDUMP existe la descompresión manual con PROCDUMP para cuando no sabemos qué compresor se utilizó en el enredo.

LA PROXIMA LECCIÓN DESCOMPRESION MANUAL CON PROCDUMP, POR SUPUESTO AHORA QUE ESTA DESCOMPRIMIDO PUEDEN CRACKEAR FÁCILMENTE EL PROGRAMA Y CAMBIAR LOS BITES CON EL ULTRAEDIT.

PRACTÍQUENLO.

Ricardo Narvaja

DESCARGADO GRATUITAMENTE DE
<http://visualinformatica.blogspot.com>

DESCOMPRESION MANUAL CON PROCDUMP

(LECCION SEIS)

Llegamos a la lección 6 y el problema que a veces ocurre cuando el PROCDUMP no descomprime automáticamente algún ejecutable con ninguna de las opciones, en este caso si necesitamos descomprimir para poder ayudarnos a crackear o para por lo menos desensamblar, deberemos hacerlo a mano, lo cual es una tarea que a veces requiere un poquito de trabajo y a veces cuesta hacerlo si uno no tiene mucha practica.

¿COMO FUNCIONAN ESTOS PROGRAMITAS COMPRESORES QUE NOS PONEN LOS PELOS DE PUNTA A VECES?

Funcionan en general de la siguiente manera, cuando nosotros cargamos un ejecutable comprimido con el SYMBOL LOADER del SOFTICE y nos detenemos en el ENTRY POINT o PUNTO DE ENTRADA del programa en realidad lo que comienza a ejecutarse primero no es el programa en sí, si no una rutina descompresora que a veces es bastante larga con muchos CALL y a veces también rutinas de protección ANTISOFTICE que si uno tuviera la paciencia suficiente de ir con F10 traceando estas instrucciones (ESTO SERIA PEDIR INFINITA PACIENCIA AUNQUE MAS DE UNA VEZ YO HE DESCOMPRIMIDO DE ESTA FORMA CUANDO NO LO PUDE HACER DE OTRA MANERA AUNQUE NO LO RECOMIENDO PORQUE ES BASTANTE CANSADOR), sin entrar en los CALL, llegaríamos a una sentencia donde el descompresor terminaría su tarea y saltaría al punto de entrada del programa el cual se encuentra descomprimido en memoria.

Para los que son ya cancheros en encontrar a mano y peleando esta sentencia se puede llegar a identificar bastante bien pues el programa hace un JMP casi siempre a una dirección lejana a donde se está ejecutando la descompresión es ese momento es como si el descompresor estuviera en un lugar y el programa descomprimido en otro lugar y entonces el descompresor salta bastante lejos mientras que todos los JMPs anteriores en la rutina descompresora eran ahí nomás cortitos dentro de la misma rutina.

Creo explicarme bien y después lo vamos a ver con el ejemplo pero la rutina descompresora esta ACA y el programa descomprimido esta ALLA, entonces cuando la rutina termino de descomprimirlo tiene que saltar desde acá donde está trabajando hasta allá donde está el programa ya descomprimidito y listo para ejecutarse.

La mayoría de los descompresores usa una rutina JMP común numérica que en los mas berretas se encuentra fácilmente, y en los más avanzados se esconde un poco mas con JMP EAX o JMP [EAX] con lo que la sentencia para darnos cuenta que es la correcta tendremos que comprobar el valor de EAX para ver si salta lejos o cerca.

Bueno esto es una introducción para el suicida que lo quiera hacer el trabajo a mano pero como el softice tiene muchos recursos que todavía no enseñamos les voy a mostrar un recurso del mismo que nos va a ayudar a que el programa se detenga en la sentencia exacta donde el descompresor termina su trabajo y comienza a ejecutarse el programa descomprimido, una vez que estamos situados allí es muy sencillo poder lograr grabar el ejecutable descomprimido como veremos en el ejemplo que daremos a continuación.

Descomprimiremos nuevamente el WINOMEGA pero esta vez a mano sin usar la opción AUTOMATICA del PROCDUMP aunque este nos ayudara en partes vitales del trabajo aun no sabiendo que descompresor es el utilizado.

Primero antes que nada estudiaremos con el PROCDUMP algunos valores que nos pueden ser necesarios en el trabajo. Abrimos el PROCDUMP y vamos a PE EDITOR y después a SECTIONS con lo que nos aparece la tablita con las características del ejecutable.

Los ejecutables siempre están divididos en varias partes por lo tanto en **NAME** podemos ver los nombres de las distintas partes del código del ejecutable comprimido.

La segunda columna **VIRTUAL SIZE** es EL TAMAÑO QUE TENDRA LA SECCION DEL EJECUTABLE UNA VEZ DESCOMPRIMIDA.(EN HEXA LÓGICO)

La tercera columna **VIRTUAL OFFSET** seria lo que nos indicaría donde comienza el ejecutable descomprimido en la memoria (NO CONFUNDIR con el ENTRY POINT) para hallar la dirección en la memoria donde comenzara el ejecutable descomprimido le sumo al valor de la columna el de IMAGE BASE y tengo la dirección de memoria donde comienza el ejecutable descomprimido.

Después tengo tres columnas más que no son muy utilizadas en este trabajo que son RAW SIZE (TAMAÑO DEL EJECUTABLE EN EL DISCO O SEA ANTES DE DESCOMPRIMIRSE), RAW OFFSET (IGUAL QUE VIRTUAL OFFSET PERO DEL EJECUTABLE COMPRIMIDO) y CHARACTERISTICS que nos muestra si es posible desensamblar con el WDASM o no (VER LECCIÓN ANTERIOR)

Para lo que nos sirven estas columnas también es que si yo veo que el VIRTUAL SIZE ES mayor que el RAW SIZE se que el ejecutable esta comprimido y si son iguales no está comprimido.

Aquí lo interesante es obtener de esta tabla donde va a comenzar en la memoria la sección CODE que es la primera, y donde terminara en la memoria.

COMO YA DIJIMOS LA DIRECCIÓN DE INICIO ES:

INICIO= IMAGE BASE + VIRTUAL OFFSET

FINAL= INICIO + VIRTUAL SIZE -1

Si sabemos la dirección de inicio y sabemos el tamaño obviamente sabemos que sumándole a la dirección de INICIO el TAMAÑO que tendrá el ejecutable descomprimido tendremos la dirección FINAL a esto hay que restarle uno (PORQUE NO SE PERO ES ASI)

En nuestro caso Lo primero es hallar la IMAGE BASE si salimos de la tablita tendremos los datos en PE ESTRUCTURE EDITOR copiamos la IMAGE BASE que es 400000.

Como sabemos que VIRTUAL OFFSET es 00001000 por lo tanto podremos tener el INICIO del ejecutable descomprimido en

INICIO= IMAGE BASE + VIRTUAL OFFSET

INICIO= 400000 + 00001000

INICIO= 401000

Aquí comenzara el ejecutable descomprimido, atención que comenzara en la memoria no será el ENTRY POINT del ejecutable descomprimido que es lo que hay que hallar.

La dirección FINAL de la sección CODE será

FINAL = INICIO + VIRTUAL SIZE - 1

FINAL= 401000 + 243A10 -1

FINAL= 644A0F

Tenemos la dirección de memoria de inicio y la dirección de memoria final de la sección CODE por lo tanto (Y EN EL 90% DE LOS CASOS) el punto de entrada o ENTRY POINT del ejecutable descomprimido estará entre estos dos valores.

Si entonces alguien es tan valiente como para ir ejecutando a mano con F10 la rutina descompresora se encontraría que en un momento en el que concluye esa rutina habría un JMP a una dirección entre 401000 y 644A0F.

Alguien podría preguntar si el ENTRY POINT del ejecutable descomprimido podría estar no en CODE si no en alguna otra sección y yo le contesto que dicen que si pero en la práctica no encontré ningún caso que fuera así.

De cualquier manera este método puede aplicarse en la primera sección y si no funciona seguir y hacer exactamente lo mismo en la segunda sección y así sucesivamente hasta hallar el ENTRY POINT.

Carguemos el ejecutable con el SYMBOL LOADER del SOFTICE.

Una vez cargado hagamos un F10 y estaremos en el ENTRY POINT del ejecutable comprimido vemos que estamos en la posición de memoria 7D2001 que no está entre la posición de INICIO y FINAL que tendrá el ejecutable descomprimido.

Eso es, ejecutara una rutina por 7D2001 que es donde comienza la rutina descompresora y cuando termine de descomprimirse saltara a una dirección entre 401000 y 644A0F que será el nuevo ENTRY POINT del programa.

Ahora lo que deberíamos hacer (SI ESTAMOS LOCOS) es poner BPX a mano en todas las direcciones de memoria entre 401000 y 644A0F de modo que apenas ejecute algo entre esas dos direcciones pare el SOFTICE.

Evidentemente esto no se puede hacer a mano porque es un desastre pero el SOFTICE tiene un comando que lo hace automáticamente.

Utilizamos una sentencia del SOFTICE que coloca BREAKPOINTS en todo un rango de memoria esta sentencia es BPR (BREAKPOINT EN RANGO)

BPR INICIO FINAL

O sea es como si colocáramos breakpoints en todos los valores entre INICIO Y FINAL. En nuestro caso

BPR 401000 644A0F

Todavía le falta algo más a esto para funcionar bien pues si mientras esta descomprimiendo lee algún valor entre esos va a parar y nosotros no queremos que pare mientras descomprime entonces la modificamos así:

```
BPR 401000 644A0F R if (eip>=401000) && (eip<=644A0F)
```

Esto quiere decir que va a parar cuando lea algún valor entre INICIO Y FINAL pero SOLAMENTE si EIP está ENTRE INICIO Y FINAL también.

Esa condición solamente se va a dar en el nuevo entry point porque antes el EIP (QUE MARCA CUAL ES LA SENTENCIA A EJECUTARSE) siempre va a estar fuera de estos valores.

A veces para asegurarme por si acaso coloco las direcciones de INICIO Y FINAL completas para que no lea en otro lado.

```
BPR 017f:401000 017f:644A0F R if (eip>=401000) && (eip<=644A0F)
```

Bueno cuando pongo estos breakpoints y después pongo a ejecutar el programa con X veo que la ejecución del mismo sigue pero es muchísimo más lento y eso se debe a que el SOFTICE tiene que comparar cada condición en cada sentencia y eso enlentece todo.

Igual nos damos cuenta que la maquina no está colgada porque el cursor del mouse se mueve. (A VECES SE CUELGA TAMBIEN Y HAY QUE HACERLO DE NUEVO)

Esperamos unos minutos (BASTANTES EN MI CASO FUERON COMO CINCO) y ahí para el SOFTICE en el nuevo ENTRY POINT del programa descomprimido.

```
644840 push ebp
```

YA CASI LO TENEMOS

Ahora debemos copiar la cadena de la sentencia 644840 porque debemos modificarla para acordarnos como era originalmente.

```
55 8B EC 83 C4 F4 53 B8 18 40 64 00
```

Estando arriba de la sentencia 644840

```
Hacemos a 644840
```

```
Y escribimos JMP 644840
```

```
Esc
```

HACEMOS ESTO PARA QUE EL PROGRAMA QUEDE HACIENDO UN BUCLE INFINITO EN ESA SENTENCIA Y NOS PERMITA CON EL PROCDUMP BAJARLO DE LA MEMORIA.

Una vez que modificamos la sentencia 644840 entonces abrimos el PROCDUMP y en la lista de procesos que aparecen apenas lo abrimos buscamos el correspondiente a WINOMEGA entonces hacemos clic derecho y clickeamos en DUMP (FULL) y nos pedirá un directorio donde guardarlo y el nombre y listo lo guardamos con el nombre que queramos.

Ahora abrimos el PROCDUMP y vamos a PE EDITOR y vemos que el ENTRY POINT del ejecutable nuevo en vez de ser 644840 sigue siendo 7d2001 como antes de descomprimirlo, debemos modificarlo.

Restamos con la calculadora 644840 el nuevo ENTRY POINT la IMAGE BASE 400000 y así obtenemos el valor para modificar en donde dice ENTRY POINT ponemos 244840 que es el nuevo ENTRY POINT (AQUÍ HAY QUE COLOCARLO ASÍ SIN SUMARLE LA IMAGE BASE)

También podemos ver si quedó habilitado para desensamblar o sea como explique en la lección anterior hay que ir a SECTIONS y en la primera línea hacer clic derecho y poner EDIT SECTION y ahí en SECCION CHARACTERISTIC ver si hay un E000020 con lo cual se puede desensamblar perfectamente con el WDASM.

El último paso es modificar con el ULTRAEDIT la sentencia de entrada que habíamos cambiado.

644840 PUSH EBP

Como copiamos la cadena 55 8b ec 83 c4 f4 53 b8 18 40 64 00

Que era la original tenemos que volver a colocar estos números hexa en lugar de los que modificamos, abrimos el ULTRAEDIT y buscamos

EB FE EC 83 C4 F4 53 B8 18 40 64 00

Y reemplazamos

EB FE POR 55 8B

COPIAMOS EL NUEVO EJECUTABLE MAS LARGO Y LO ARRANCAMOS Y FUNCIONAAAAAAA.

Este método a pesar de ser difícil es casi 100 % eficaz para descompactar cualquier programa, lo único que a veces puede molestar es que algunos descompresores tienen protección antisoftice con lo que deberemos ocultarlo con el FROGSICE (NO ES ESTE CASO) y no podremos usar el SYMBOL LOADER, tendremos que poner un BPX numérico en el ENTRY POINT para que pare allí en vez de cargarlo desde el SYMBOL LOADER para usar el FROGSICE.

PRACTIQUEN Y PREGUNTEN QUE VIENEN LECCIONES MAS FACILES A PARTIR DE AQUÍ LO DURO YA ESTA PASANDO.

Ricardo Narvaja

DESCARGADO GRATUITAMENTE DE
<http://visualinformatica.blogspot.com>

**ALGUNAS COSITAS SOBRE LOS BREAKPOINTS EN PROGRAMAS CON
VENCIMIENTO DE TIEMPO Y UN POCO DE AMPLIACIÓN DE
CONOCIMIENTO**

(LECCIÓN SIETE)

Existen muchos BREAKPOINTS para cada caso y el hecho de acertar con el breakpoint justo puede ahorrarnos dar vueltas y vueltas por el código, por lo tanto es bueno conocer los breakpoints más usuales para utilizar en el SOFTICE.

Por ejemplo sabemos ya que los programas que tienen vencimiento de tiempo (30 días o cualquier periodo de tiempo), utilizan generalmente la función BPX GETSYSTEMTIME o también a veces BPX GETLOCALTIME, el problema generalmente con estos BREAKPOINTS es que son utilizados por muchos programas y muy frecuentemente por lo que cuando los colocamos a cada rato el softice rompe en esta función por cualquiera de los programas que la utiliza (HASTA EL EXPLORER LA USA FRECUENTEMENTE).

Lo que hay que hacer en casos en que la función es tan utilizada es lo siguiente, como primera opción entrar con CTRL+ALT+DEL y FINALIZAR todos los programas que están en funcionamiento menos el EXPLORER, cuando este quede solo en la ventana de procesos, entonces ahí entrar al softice y poner el BPX GETSYSTEMTIME o GETLOCALTIME y entrar en el programa.

La segunda forma es con el WDASM si el archivo no está comprimido, entonces vamos al icono de IMP FN o sea funciones importadas, y allí buscamos la función que queremos encontrar en este caso BPX GETSYSTEMTIME o BPX GETLOCALTIME y hacemos clic encima del nombre de la función y vamos anotando las posiciones de memoria en que el WDASM nos dice que es utilizada esa función (COMO HACIAMOS CON LAS STRING REFERENCES) y nos aseguramos de hacer clic varias veces encima del nombre de la función para asegurarnos si aparece más de una vez, y después en el softice ponemos BPX numéricos en esos valores que anotamos y entonces el softice va a parar en las funciones buscadas sin que nos molesten las llamadas a la misma función de otros programas.

Generalmente después de que la función concluye y vuelve al ejecutable hay una comparación donde testea la fecha con la fecha de vencimiento del programa, también puede guardarse en alguna posición de memoria para comprobarse más adelante.

Dentro de la función GETSYSTEMTIME podemos ver lo siguiente si la vamos recorriendo con F10

En BFFA0F1B **MOV ECX, [ESP+0C]**

SI ME SITUO ALLI EN ECX ESTA EL AÑO ACTUAL

? **ECX**

= 07D0 EN HEXA=2000 (el año EN DECIMAL)

EN BFF9E8E9 **MOV AX,[ESI]**

SI HAGO

? **AX**

=07D0=2000 (el año)

LUEGO EN **MOV CX,[ESI+02]**
SI HAGO
? CX
=0A EN HEXA=10 (o sea el mes actual)

Y EN **MOV DX,[ESI=06]**
SI HAGO
? DX
=10 EN HEXA=16 (o sea el DIA actual)

Después que obtiene estos valores los guarda más adelante en

```
MOV [EBP-10],AX  
MOV [EBP-0E],CX  
MOV [EBP-0C],DX
```

Y DESPUES LOS GUARDA DE NUEVO A CONTINUACION

Cuando salimos de la función y volvemos al ejecutable el registro **ESP+04** apunta a donde está el valor de la fecha si vamos al salir exactamente a

```
LEA ECX, [ESP+04]
```

O alguna sentencia así que tenga el ejecutable vemos que pasa el valor que está en **ESP+04** que es la fecha a ecx aquí es donde hay que actuar y reemplazar esta sentencia por una que mueva a ecx una fecha fija que habilite siempre el programa, o también podemos seguir haciendo f10 a ver si encontramos la comparación de la fecha con la fecha de vencimiento del programa, lo cual puede estar ahí nomas o bastante más adelante. Si hacemos allí

```
D ESP+04  
D0 07 0A 00 10 00
```

Los dos primeros son 07d0 el año en HEXA los dos segundos son 00 0a el mes en HEXA y los últimos dos son 0010 o sea el día en HEXA (siempre se escriben al revés)

Si queremos realizar el mismo trabajo en un programa de dos deberemos cambiar el BPX.

Nos damos cuenta que un programa es de DOS o 16 bits porque en el SOFTICE la dirección de memoria aparece en este formato:

POR EJEMPLO:

0076:**8989**

O sea que es de cuatro cifras en vez de ocho cifras que es lo que vemos siempre en los programas de 32 bits (PEJ: 0175:**00107652**)

En programas de 16 bits el Breakpoint para los programas con vencimiento de tiempo es

BPINT21 if ah==2a (HAY QUE ESCRIBIRLO ASI SIN BPX)

Escribiéndolo así el softice parara cuando el programa quiera leer la fecha y como aquí no hay función sino que parara directamente en el ejecutable, en

CX estará el año
DH estará el mes
DL estará el día

Si hacemos

?CX o ?DX ?DL

Podremos verlos, luego el ejecutable guardara estos valores con alguna sentencia como

mov [ebp-05], cx

etc con lo cual podrá ser modificado en el lugar que lo guarda.

Estos son generalidades sobre programas con vencimiento de tiempo que veremos más adelante con ejemplos concretos, cuando tengamos un programa de este tipo, aunque si es un programa con vencimiento de tiempo y tiene un NUMERO DE SERIE para registrar habría que tratar primero de crackear el número de serie y si esto está muy difícil entonces sí, crackear el tiempo.

Aquí hay una lista de los breakpoints más usuales que explicaremos brevemente a continuación:

bpx MessageBox
bpx MessageBoxExA
bpx MessageBeep
bpx SendMessage

Estos son breakpoints para los famosos cartelitos que aparecen y son fácilmente identificables por ser muy simples y tienen un texto que generalmente se encuentra en las STRING REFERENCES.

Messagebox solo sin la A al final es para 16 bits y con la A o MESSAGEBOXEXA es para 32 bits

bpx GetDlgItemText
bpx GetDlgItemInt
bpx GetWindowText
bpx GetWindowTextWord
bpx GetWindowTextInt

Son funciones para que el ejecutable lea texto, palabras o números que uno tipea, o sea que si ponemos un BPX de estos vamos a caer APENAS el ejecutable se entera de que es lo que tipeamos. También se usa mucho para esto directamente el **BPX HMEMCPY**.

bpx DialogBoxParamA
bpx CreateWindow
bpx CreateWindowEx
bpx ShowWindow
bpx UpdateWindow

Estos también son funciones para dibujar ventanas pero ya más complejas.

DE TIEMPO

bpint 21 if ah==2A (DOS)
bpx GetLocalTime
bpx GetFileTime
bpx GetSystemtime

Como vimos en esta lección

CD-ROM

bpint 13 if ah==2 (DOS)
bpint 13 if ah==3 (DOS)
bpint 13 if ah==4 (DOS)
bpx GetFileAttributesA
bpx GetFileSize
bpx GetDriveType
bpx GetLastError
bpx ReadFile
bpio -h (Your CD-ROM Port Address) R

Estos son para el que el ejecutable pare cuando accede al CDROM los tres primeros son para DOS los otros para 32 bits

ENTRADA DE TECLADO

bpint 16 if ah==0 (DOS)
bpint 21 if ah==0xA (DOS)

Estos son para DOS y para cuando tecleamos algo

DE FILAS

bpint 21 if ah==3dh (DOS)
bpint 31 if ah==3fh (DOS)
bpint 21 if ah==3dh (DOS)
bpx ReadFile
bpx WriteFile
bpx CreateFile
bpx SetFilePointer
bpx GetSystemDirectory

Estos son para cuando el programa accede a una fila.

Filas INI

bpx GetPrivateProfileString
bpx GetPrivateProfileInt
bpx WritePrivateProfileString
bpx WritePrivateProfileInt

Estos son para cuando accede a FILAS INI

Registro

bpx RegCreateKey
bpx RegDeleteKey
bpx RegQueryValue
bpx RegCloseKey
bpx RegOpenKey

Estos bpx son para cuando el programa accede al registro. Si abre una llave usa el OPEN KEY, si la cierra CLOSEKEY, cuando lee un valor QUERY VALUE y cuando crea o borra una llave CREATEKEY O DELETEKEY

Hay todavía más como los BPIO que son los BREAKPOINTS para los puertos, los BPM que son breakpoints que paran cuando se lee algún valor de la memoria aunque no se ejecute, etc etc, en las sucesivas lecciones veremos casos y ejemplos de los mismos.

Bueno con la práctica los iremos utilizando y explicando más en profundidad así como también los breakpoints condicionales (COMO EL QUE VIMOS EN LA LECCIÓN ANTERIOR)

Ricardo Narvaja

DESCARGADO GRATUITAMENTE DE
<http://visualinformatica.blogspot.com>

EMPEZANDO A LIDIAR DE A POQUITO CON CLAVES ENCRIPADAS- PRIMERA APROXIMACIÓN

(LECCIÓN OCHO)

Muchas veces en los programas actuales es difícil ya encontrar la clave a la vista comparándose fácilmente con el número falso que nosotros pusimos, aunque he encontrado algunas todavía cada vez son menos.

Hoy practicaremos con el archivo CRACKME2, el que sigue a continuación del que crackeamos en la lección uno y estos archivos que cada tanto iremos crackeando nos proporcionan una buena forma de ir practicando y avanzando entre las distintas protecciones que existen entre los programas actuales, obviamente este no es el último crackme que salió, y los últimos son casi una fortaleza protegida por todos lados pero bueno para caminar hay que gatear primero y saltar pasos no es bueno vamos de a poco.

Bueno una vez que nos hicimos del archivo CRACKME2.exe y lo ejecutamos vemos que se desbloquea con un número de serie y que al poner un número falso como 989898 que siempre ponemos nos dice la famosa frase **NO LUCK THERE MATE** que nos manda a bañarnos rápidamente.

Aquí podemos buscar la STRING REFERENCE NO LUCK THERE MATE en el WDASM y aparece fácilmente y a diferencia del primer crackme hay un solo lugar donde figura, nos fijamos arriba en REFERENCED BY A CALL at ADDRESS 401243 y allí entonces vamos poniendo GOTO CODE LOCATION 401243.

```
0040123C      add  esp, 4
0040123F      test cl, cl
00401241      jz   40124A
00401243      call 401349
00401248      jmp  4011E6
```

Bueno aquí hay un CALL en 401243 que nos envía a la sección de CHICO MALO y nos pone NO LUCK THERE MATE o sea que si el salto que esta en 401241 lo hacemos permanente o sea JMP 40124A ya estaría crackeado pero aquí el tema es encontrar el número de serie no solamente parchearlo lo cual es un poquito más difícil (NO MUCHO).

Bueno vamos adentro con el SOFTICE, abrimos el CRACKME y tecleamos 989898 y antes de poner OK entramos al SOFTICE con CTRL. +D y tecleamos BPX MESSAGEBOXA y volvemos al archivo con CTRL. +D y hacemos clic en OK y caemos dentro, luego tecleamos F12 y aparece el CARTELITO NEFASTO y luego al hacer clic en ACEPTAR volvemos al softice y aparece el 401364 en un RET, lo ejecuto con F10 y ya estoy en el lugar crítico.

Si borro el Breakpoint con BC* y pongo BPX 401232 y repito el proceso va a parar aquí:

```
00401232      push 40217E
00401237      call 4013B8
0040123C      add  esp, 4
0040123F      test cl, cl
00401241      jz   40124A
00401243      call 401349
00401248      jmp  4011E6
```

Obviamente si yo voy ejecutando con F10 hasta llegar al salto y allí pongo R EIP=40124A, lo hago saltar a la zona de CHICO BUENO, y estaría registrado, pero aquí hay que revisar un poco la comparación, la sentencia TEST CL,CL o sea testear contra si mismo sirve para ver si lo que se compara es cero o no, si es cero saltara en la próxima sentencia, con cualquier otro valor no.

Obviamente si paramos allí y hacemos ? CL vemos que allí puede haber un cero o un uno, o sea que de clave nada, es lo que se llama un FLAG o BANDERA que donde está la verdadera comparación cambia el programa el valor de CL a uno o cero si es correcto el chequeo depende del caso.

Bueno volvamos a empezar y paremos en el BPX que teníamos.

Allí en la sentencia PUSH vemos que si hacemos D 40217E en la ventana de datos nos aparece nuestro numero falso 989898 el cual se prepara a trabajar.

En este programita ya que no es muy largo podríamos entrar al CALL 4013b8 que es donde se cocina todo y hacer T y ejecutar F10 hasta que encontremos algo sospechoso, pero el método que se usa generalmente es el siguiente para seguir lo que hace un programa con la clave.

Lo que yo quiero es que el SOFTICE pare cuando el ejecutable lea o escriba de mi clave o sea apenas la toque que pare allí, ya que seguro va a trabajar sobre ella o también al comparar o cualquier cosa que haga con ella que pare allí inmediatamente.

En el SOFTICE pongo el siguiente Breakpoint.

BPR 40217E 40218E RW

Que es un Breakpoint de rango o sea que es como poner breakpoints en todos los valores que se encuentran entre 40217E Y 40218E , como verán me paso un poquito y abarco unos numeritos mas pero no es nada.

Ahora hago X o tecleo F5 y el SOFTICE para apenas el programa quiere hacer algo con la clave, veamos:

Para en:

401371 MOV AL, [ESI]

Como en ESI está la clave falsa al querer mover el primer numerito a AL el SOFTICE para ya que está trabajando con la clave nuestra trucheli. (FALSA O TRUCHA)

D ESI es obviamente 989898 y en AL luego de ejecutar esta sentencia esta la primera cifra ? AL =9

Luego pasa por una comprobación que es seguramente para pasar a mayúsculas si son letras ya que compara con 41 y con 5A o sea con A y con Z, seguro que si pongo minúsculas me las transforma en mayúsculas.

Después en

40139d MOV CL, [EDI+4021A3]

Mueve a CL la primera letra de la palabra que está en 4021A3.

D 4021A3 está la palabra **Messing_in_bytes-**

Después hace XOR entre BL y CL esta es la clave de todo, como en BL esta la primera cifra de mi clave y en CL la primera cifra de la palabra MESSING..... el resultado se guarda en BL y luego lo escribe en donde estaba mi clave trucha, o sea que el 9 que era la primera cifra ahora es t (T MINÚSCULA).

Repite este proceso hasta que transforma mi clave en t]JKPV.

Luego existe una sentencia que compara mi clave encriptada con algo esa sentencia es 4013CB REPZ CMPSB que compara número a número lo que está en ESI que es mi clave ENCRIPADA con lo que está en EDI que es la VERDADERA CLAVE ENCRIPADA. Si hago D edi veo unos caracteres que en HEXA son:

1F 2C 37 36 3B 3D 28 19 3D 26 1A 31 2D 3B 37 3E

o SEA QUE ES LA CLAVE ENCRIPADA.

Como cuando vimos nuestra propia clave, 989898 estudiamos que para encriptarla el programa hizo XOR con la palabra MESSING IN BYTES, entonces para encriptar la clave verdadera hizo lo mismo.

Los números HEXA de la palabra MESSING IN BYTES son:

4D 65 73 73 69 6E 67 5F 69 6E 5F 62 79 74 65 73

o SEA QUE SI HACEMOS XOR NUMERO POR NUMERO ENTRE LA CLAVE encriptada VERDADERA CON messing in bytes OBTENDREMOS LA CLAVE VERDADERA SIN ENCRIPAR O SEA HACEMOS EL CAMINO HACIA ATRÁS, COMO A NUESTRA CLAVE LA TRANSFORMO DE ESA FORMA ENTONCES HACEMOS LO MISMO HACIA ATRÁS CON LA CLAVE ENCRIPADA Y OBTENEMOS LA CLAVE FINAL SIN ENCRIPAR.

1F 2C 37 36 3B 3D 28 19 3D 26 1A 31 2D 3B 37 3E (CLAVE ENCRIPADA)

4D 65 73 73 69 6E 67 5F 69 6E 5F 62 79 74 65 73 (MESSING IN BYTES)

Con la calculadora de WINDOWS puesta en HEXADECIMAL pongo el primer valor que es **1f** y aprieto **XOR** y después **4d** igual a **52**, hago lo mismo cifra por cifra hasta obtener los números hexa de la clave

52 49 44 45 52 53 4f 46 54 48 45 53 54 4f 52 4d

Si voy al SOFTICE y los paso a ASCII con

? 52=R

?49=I

? 44=D

? 45=E

? 52=R

? 53=S

? 4F=O

?46= F

? 54=T

? 48=H
? 45=E
? 53=S
? 54=T
? 4F=0
? 52=R
? 4D=M

O SEA LA CLAVE PARA EL CRACKME2 ES:

RIDERSOFTHETORM

PRUEBEN EN LA VENTANA DE REGISTRO Y GREAT WORK MATE!!!!!!

Ricardo Narvaja

DESCARGADO GRATUITAMENTE DE
<http://visualinformatica.blogspot.com>

CRACKEANDO EN DELPHI

(LECCIÓN NUEVE)

Dado que todavía no terminamos con el crack del FONT TWISTER que está hecho en DELPHI y es muy difícil, practicaremos mientras con otro programa hecho en DELPHI, para trabajar con fuentes, llamado FONT CREATOR 3.0.

ACLAREMOS QUE EL CRACK DE ESTE PROGRAMA SE PUEDE HALLAR FÁCILMENTE CON LOS METODOS TRADICIONALES, PONIENDO BPX EN MESSAGEBOX y así encontrar el salto y un poquito más arriba la clave y la comparación. Es un programa fácil de crackear por métodos tradicionales, pero aquí vamos a aprender otros métodos a utilizar en DELPHI, para los cuales nos basamos en el tutorial de LEIRUS sobre cracking en DELPHI, que nos pueden servir para otros programas más difíciles que este en el mismo lenguaje.

Primer paso: Saber si esta hecho en DELPHI.

Para eso podemos usar el GETTYPE y copiamos el ejecutable fcp3.exe del programa dentro de la carpeta donde está el GETTYPE, una vez que está allí, abrimos una ventana de DOS y nos posicionamos con CD... en el directorio donde está el GETTYPE y allí tecleamos

```
GTW fcp3.exe /p
```

GTW en mi caso porque en mi gettype el ejecutable se llama así, si no poner el nombre correspondiente (PUEDE SER GT o GT019 DEPENDE DE LA VERSIÓN) y la P al final la colocamos para que pare y muestre una parte de la información y después tecleando enter siga, como cuando vemos con DIR un directorio muy largo sino ponemos aquí /p pasara de largo y no veremos nada.

Bueno el resultado nos muestra:

COMPILER: BORLAND DELPHI 3/4 y generalmente no se equivoca.

Igual ya veremos cómo asegurarnos de esto. Ahora utilizaremos una herramienta que no pedí al comienzo del curso porque son para lenguajes específicos, para DELPHI es la herramienta llamada **DEDE** que va por la versión 2.4 creo, aunque yo tengo la 2.34, también más adelante bajaremos herramientas para VISUAL BASIC en las lecciones sobre ese lenguaje. (SMART CHECK)

También hay una herramienta muy buena que se llama WINDOWSE y APISPY que complementan lo que se necesita para crackear en ambos idiomas, aunque hoy como es el primer ejemplo no las vamos a usar.

USAREMOS EL DEDE

Toda vez que es la primera vez que utilicemos este programa estaremos algo confundidos, ya que a pesar de ser una excelente ayuda para crackear en DELPHI, manejarse bien con el CUESTA UN POCO dado la terminología propia del idioma DELPHI, que todavía no conocemos bien, pero aquí vamos.

Cuando abrimos el DEDE vemos una barra con varias OPCIONES

CLASSES INFO- FORMS-PROCEDURES-PROJECTS y EXPORTS

CLASSES INFO: el programa nos suministra información sobre las clases del ejecutable

FORMS: nos muestra las características de los FORMULARIOS del programa

PROCEDURES: Permite ver los distintos procedimientos del programa (MUY IMPORTANTE PARA CRACKEAR)

Por ahora nos quedamos con esto más adelante habrá tiempo de profundizar.

Bueno abrimos justo a la izquierda de PROCESS está el menú para cargar la fila, buscamos allí el ejecutable y una vez que lo encontramos ponemos PROCESS y empieza a trabajar.

Aquí también veremos si lo que nos dijo el GETTYPE es correcto ya que si la fila no está programada en DELPHI aparece un mensaje de error avisándonos.

Una vez que termino vamos a PROCEDURES y allí en la columna UNIT NAME buscamos algo que tenga relación con la palabra REGISTER o REG o algo así.

Encontramos REGISTER FRM pulsamos en el y en el comando de la derecha sobre EVENTS, allí hay tres eventos pero lo que nos interesa es que pasa cuando hago clic en la ventana de REGISTRO en el BOTON que dice REGISTER .

Hago clic derecho para asegurarme que es el BOTON correcto y no me estoy confundiendo con otra cosa, allí elijo SHOW ADDITIONAL DATA y me aparece una aclaración diciéndome CAPTION: REGISTER eso quiere decir que lo que está escrito en el botón es exactamente la palabra REGISTER y no otra cosa (VAMOS BIEN).

Ahora vuelvo a hacer clic con el botón derecho y elijo DISASSEMBLE, esto nos va a desensamblar exactamente las sentencias que se ejecutan al clicar en el BOTON REGISTER.

COMO PUEDEN VER ESTO PUEDE SERVIR PARA VENTANAS MOLESTAS HECHAS EN DELPHI Y QUE QUEREMOS VER QUE OCURRE CUANDO CLICKEAMOS UN BOTON QUE ELLAS TENGAN.

Nos aparece el listado desensamblado en el cual vemos la sección de chico bueno y chico malo:

```
004F5634 E853F7F3FF      call 00434D8C
004F5639 8B45D4               mov  eax, [ebp-$2C]
004F563C 8B4DFC               mov  ecx, [ebp-$04]
004F563F 5A                   pop  edx
004F5640 E833FDFFFF          call 004F5378
004F5645 6A00                 push $00
004F5647 668B0DE0564F00     mov  cx, word ptr [$4F56E0]
004F564E B202                 mov  dl, $02
```

*** Possible String Reference to: "Thank you for registering Font Creator Program."**

```
|
004F5650 B8EC564F00          mov  eax, $004F56EC
004F5655 E8E650F6FF          call 0045A740
```

```

004F565A C7833402000001000000 mov dword ptr [ebx+$0234], $00000001
004F5664 EB1F jmp 004F5685
004F5666 6A00 push $00
004F5668 668B0DE0564F00 mov cx, word ptr [$4F56E0]
004F566F B201 mov dl, $01

```

*** Possible String Reference to: "Registration failed: Invalid Password"**

```

|
004F5671 B824574F00 mov eax, $004F5724
004F5676 E8C550F6FF call 0045A740
004F567B C7833402000002000000 mov dword ptr [ebx+$0234], $00000002
004F5685 33C0 xor eax, eax
004F5687 5A pop edx
004F5688 59 pop ecx
004F5689 59 pop ecx
004F568A 648910 mov fs:[eax], edx

```

Más arriba en el desensamblado vemos el salto hacia una u otra de esas partes:

```

004F560E 8B55DC mov edx, [ebp-$24]
004F5611 58 pop eax

```

*** Reference to: System..LStrCmp()**

```

|
004F5612 E84DEBF0FF call 00404164
004F5617 754D jnz 004F5666
004F5619 8D55D8 lea edx, [ebp-$28]

```

aquí si salta a 4f5666 va a CHICO MALO y te aparece el cartel de REGISTRATION FAILED y si no salta va a THANK YOU FOR REGISTERING y después pasa por encima la zona de CHICO MALO gracias a
4f5664 JMP 4f5685

Bueno llegamos hasta aquí sin usar el SOFTICE, ahora para ver la clave lo necesitamos, cerramos el DEDE y cargamos el programa del WINICE LOADER o ponemos un BPX cualquiera (COMO BPX GETVERSION) para que pare apenas se inicie el programa, una vez que estamos en el ejecutable (VER LA LINEA VERDE SINO TECLEAR F12 HASTA QUE ESTEMOS EN EL) ponemos BPX 4f560e

```

004F560E 8B55DC mov edx, [ebp-$24]
004F5611 58 pop eax

```

ALLI ESTA LA CLAVE PARA NUESTRO NOMBRE Y COMPANIA cuando estamos en la primera sentencia hacemos D EDX y tenemos nuestra clave falsa y ejecutamos la segunda sentencia y después de ejecutarla en D EAX esta nuestra clave buena.
En mi caso aparece para

NOMBRE: narvaja
COMPANIA: bamers
CLAVE: F4VK371D9FDS

Ahora si quiero verificar exactamente dentro del CALL que viene en:
004F5612 E84DEBF0FF call 00404164

Exactamente donde realiza la comparación (PARA EVITAR TRAMPAS Y CLAVES FALSAS QUE A VECES APARECEN EN LOS PROGRAMAS)

Cuando hice D EDX me apareció mi clave falsa 999999999999 en

Dc5704

Por lo tanto pongo un Breakpoint DE RANGO en las posiciones de memoria donde esta la clave para que cualquier cosa que haga el programa con mi clave pare allí.

BPR DC5704 DC5714 RW

Con lo que abarcamos toda la clave y hago X enter.
El programa para cuando comienza la comparación en

```
40418d mov ecx , [ESI]
404191 mov ebx , [EDI]
404193 cmp ecx, ebx
```

D ESI y D EDI son las claves trucha y buena y pasa las cuatro primeras cifras a ecx y a ebx y las compara luego hace lo mismo con las subsiguientes cuatro cifras, hasta que compara todo.

PROGRAMA CRACKEADO.

Esto fue solamente una primera aproximación a crackear en DELPHI, más adelante veremos programas más difíciles que este (EL FONT TWISTER. ¿PODREMOS CRAKEARLO?).

Ricardo Narvaja

DESCARGADO GRATUITAMENTE DE
<http://visualinformatica.blogspot.com>

COMO CRACKEAR EN VISUAL BASIC

(LECCIÓN DIEZ)

Bueno llegamos a la lección 10, quien lo hubiera dicho, vamos a ver un tema que es el crackeo en VISUAL BASIC, para esto le recomiendo copiar el WDASM en otra carpeta y a ese WDASM aplicarle el parche que se baja de internet para que aparezcan las STRING REFERENCES de VISUAL BASIC.

Por qué les digo que hagan un WDASM especial para VISUAL BASIC ?

Porque cuando apliquen el parche y quieran ver en ese WDASM modificado las STRING REFERENCES en un programa hecho en DELPHI, como el de la lección anterior, van a ver que no salen todas las STRING REFERENCES, solo algunas, por lo tanto es mejor tener dos el original para programas en general y para DELPHI y uno especial para VISUAL BASIC.

Recordemos que en el WINICE.DAT del SOFTICE debemos quitarle el punto y coma delante de

exp=c:\windows\system\msvbvm50.dll

exp=c:\windows\system\msvbvm60.dll

Para programas hechos en VISUAL BASIC 5 y 6. Hay que saber también que las funciones en VISUAL BASIC no son las mismas que conocemos:

<u>VB breakpoints</u>		<u>Breakpoints "normales"</u>
rtcmsgbox		messagebox/a/exa
rtcinputbox		getwindowtext/a, getdlgitemtext/a
__vbanew, __vbanew2	~	dialogbox, dialogboxparam/a
__vbastrcomp	~	lstrcmp

O sea que deberemos usar aquí las funciones de la primera columna mas alguna interesante que encontremos en la tabla de IMP FN o funciones importadas del WDASM.

Nos damos cuenta que es un programa hecho en VISUAL BASIC ya que en las IMP FN aparecen casi todas funciones pertenecientes a MSVBV50 que es la función que utiliza el VISUAL BASIC 5.

La víctima en este caso es un programa llamado READY CODE 98, que es un programa medio viejito pero que nos servirá para practicar, un poco como crackear en este idioma, además debemos bajarnos la excepcional herramienta de NUMEGA (los autores del SOFTICE) llamada SMART CHECK 6.03 (QUE CREO QUE ES LA ULTIMA VERSION), que es una excelente herramienta para crackear en VISUAL BASIC.

Yo la baje de

<http://www.crackstore.com/toolz/>

O si no directamente los links son:

<http://www.crackstore.com/toolz/f-nsc031.zip>
<http://www.crackstore.com/toolz/f-nsc032.zip>
<http://www.crackstore.com/toolz/f-nsc033.zip>
<http://www.crackstore.com/toolz/f-nsc034.zip>

Usaremos aquí algunos gráficos extraídos del tutorial de EISEL, COMO CRACKEAR EN VISUAL BASIC, sobre todo los que se refieren a la configuración del SMART CHECK.

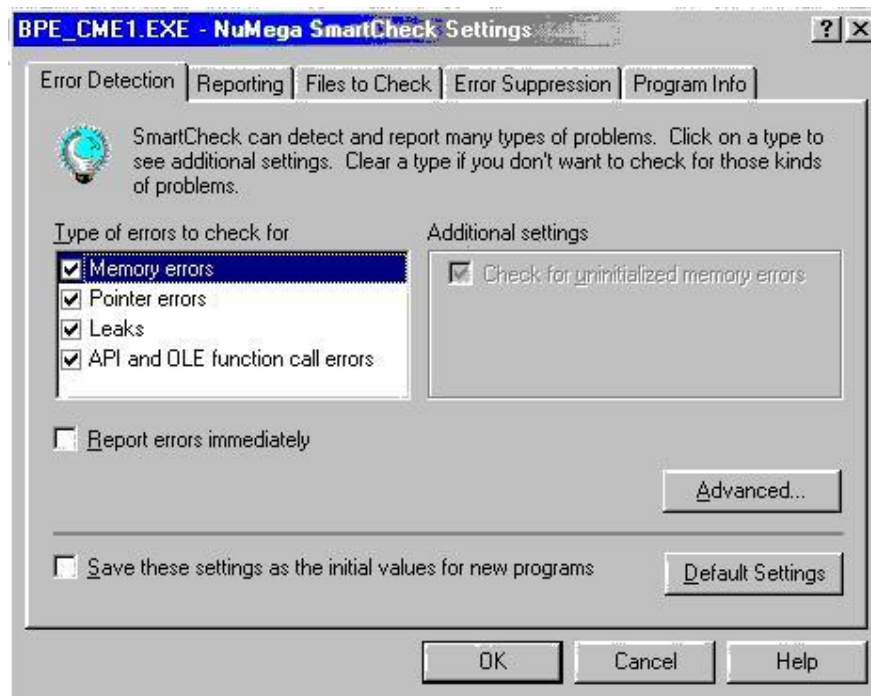
Instalamos el READY CODE y por allí aparece un cartel rojo enorme UNREGISTERED COPY y en HELP-REGISTER está la ventana para registrarse.

Hay que mencionar también que VISUAL BASIC trabaja las cadenas de texto en formato ancho, por lo tanto es bastante utilizada la función MULTIBYTETOWIDECHAR y la inversa WIDECHARTOMULTIBYTE que pasan una cadena de texto común a formato ancho y viceversa.

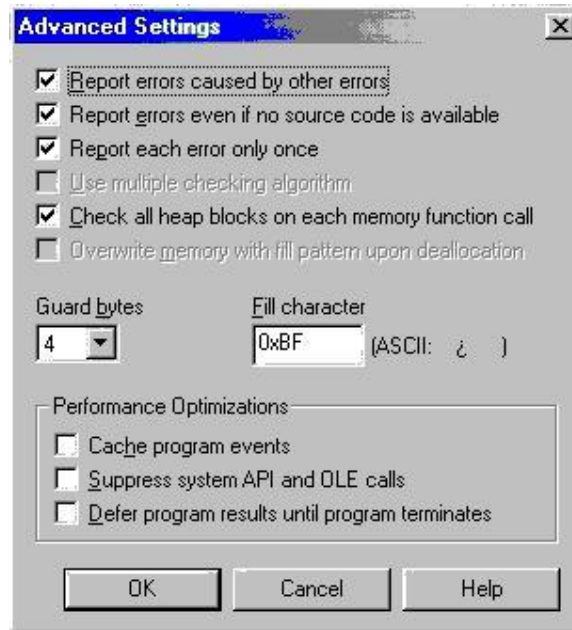
O sea que si tenemos la cadena de texto: 989898 y le aplicamos la primera función quedara 9.8.9.8.9.8 o sea en formato ancho y si le vuelvo a aplicar la segunda volverá a ser 989898.

Hay mil formas de crackear este programita ya que su protección es bastante tonta pero nos va a servir para aprender a usar el SMART CHECK. Instalémoslo y veamos como configurarlo:

Vamos al menú SETTINGS y en PROGRAM vemos esta ventana:

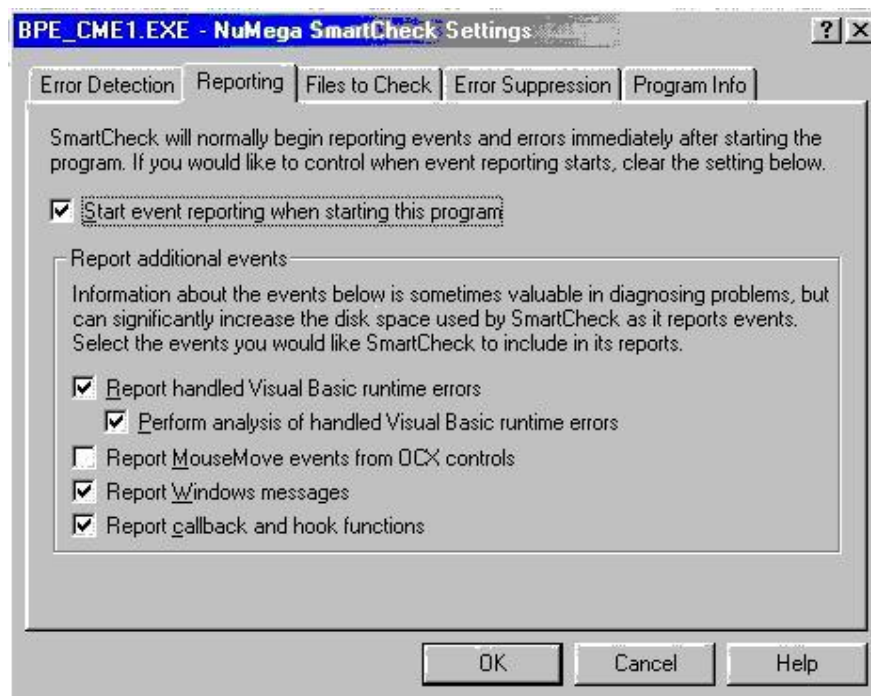


Y la configuramos así como se ve en la imagen. Es decir ponemos la tilde en donde dice TYPE ERRORS TO CHECKFOR: en las cuatro opciones que tiene y REPORT ERRORS IMMEDIATELY que no tenga tilde. Luego vamos a ADVANCED SETTINGS y ponemos lo siguiente:



Ponemos la tilde en las 6 opciones que hay (EN LAS QUE NO ESTEN GRISADAS OBVIO). Y no poner tampoco en MULTIPLE CHECKING ALGORITHM.

Y por ultimo



En la pestaña REPORTING ponemos la tilde en todas las opciones menos REPORT MOUSEMOVE EVENTS FROM OCX CONTROLS. Y listo:

Parece un programa difícil de usar pero no lo es, ahora cerramos el READY CODE y desde el SMART CHECK vamos a OPEN y abrimos el ejecutable del READY CODE, una vez que lo abrimos sale un mensaje diciéndonos que está abierto, ahora hay que ejecutarlo, vamos a PROGRAM -START y arranca el programa desde el SMART CHECK hasta que lo abre por completo.

Ahora vamos al menú del SMART CHECK y en VIEW ponemos SHOW ALL EVENTS para que nos muestre todos los eventos que realiza el programa. Volvemos a la ventana del READY CODE y entramos a la VENTANA DE REGISTRO escribimos:

NOMBRE: narvaja

REGISTRATION KEY: 989898

Por ahora pongan la misma que nosotros así hacemos lo mismo, apreten REGISTRO y sale el cartelito INVALID KEY, mientras tanto el SMART CHECK esta tomando nota de lo que el programa está haciendo. Una vez que ya nos apareció el cartel INVALID KEY, cerramos el READY CODE y volvemos al SMART CHECK.

Ponemos la barra azul que marca en el primer lugar y voy a EDIT- FIND: y pongo 989898 y que busque; La tercera vez que hago clic en FIND para cuando el programa utiliza la FUNCION de comparación de cadenas VbaStrCmp y cuando estoy parado allí en la ventana de la derecha que es la que muestra los argumentos de las funciones vemos que figura la clave 989898 y con otra cadena que el programa compara, O SEA LA CLAVE VERDADERAAAAA.

En mi caso es: **RC2A-42503-CC0214E1**

Lo escribo en la ventana de REGISTRO DEL READY CODE Y ME REGISTRA. Al cerrar y abrir el READY CODE NUEVAMENTE DESAPARECE EL CARTEL UNREGISTERED COPY.

El programa ya esta crackeado.

Todos los programas en VISUAL BASIC obviamente no serán tan fácil de crackear como éste, que ni siquiera usamos el SOFTICE, pero es bueno aprender a usar el SMART CHECK ya que ayuda bastante.

Este programa se podía haber crackeado también buscando la STRING REFERENCE en el WDASM y allí encontrando en salto o poniendo un BPX VbaStrCmp para que pare donde compara ambas cadenas, es bueno conocer todos los métodos.

Ricardo Narvaja

DESCARGADO GRATUITAMENTE DE
<http://visualinformatica.blogspot.com>