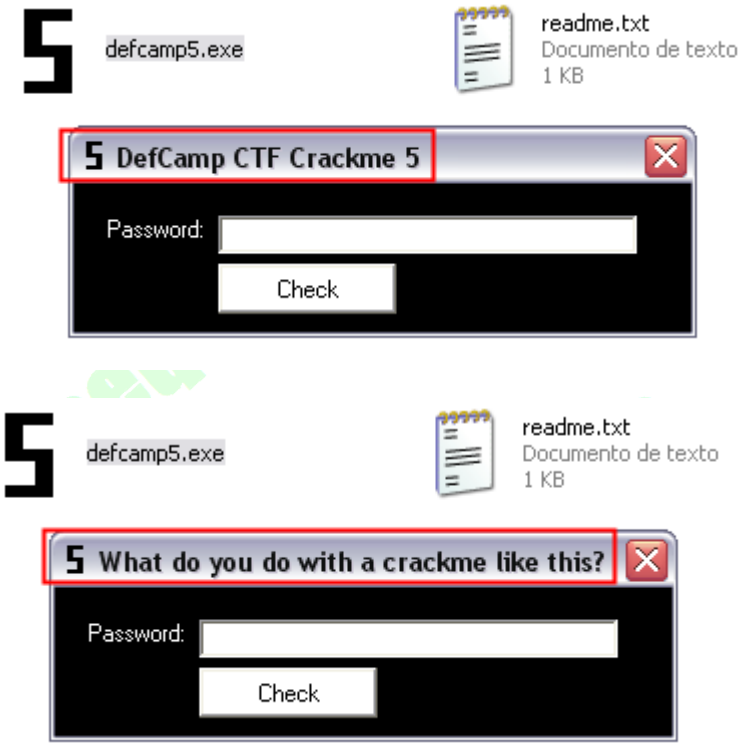
VB-DCTF5 Crackme

Saludos gente. Después de tanto tiempo,aquí estoy de nuevo para compartirles este tutorial sobre un crackme que ni sé desde cuándo lo tenía en mi disco duro. Es un crackme que envió Ricardo para un concurso. Cuando tenía algo de tiempo, me ponía a estudiarlo y a mirarle las tripas, mientras esperaba que a alguien se le encendiera la bombilla y consiguiera resolverlo y leer su correspondiente tutorial: pero ese dia no llegaba. Un dia casi borro el crackme para olvidarme de él y así, de paso, hacer limpieza en mi ordenador. Pero en el último momento me arrepentí y no lo borré y me decidí a retomar su estudio esperando resolverlo de una dichosa vez.
Bien, el crackme está escrito en Visual Basic y tiene dos principales protecciones: se cierra si utilizas el PEID, OllyDbg e IDA y si utilizas SmartCheck para capturar sus eventos, practicamente te lo satura utilizando un Timer. El Timer basicamente sirve para cambiar el titulo de la ventana del crackme y lo hace cada tres segundos:



Cuando se dispara el evento del Timer, se entran en una serie de rutinas que hacen innumerables operaciones. Todo esto lo va recogiendo SmartCheck (que se toma su tiempo en resolver). Cuando se vuelve a disparar el Timer despues de tres segundos, SmartCheck puede que ni haya terminado e resolver el tick anterior del Timer y así sucesivamente con lo que os podeis imaginar cómo se complica la cosa para el SmartCheck. Pero eso no es todo: según qué maquina tengas, igual te la bloquea así que lo que hay que hacer es cambiar el intervalo del Timer para que SmartCheck trabaje desahogado. Bien, ¿cómo cambio el intervalo del Timer?. Bien, creé un ejecutable básico en Visual Basic con solamente un Timer en el Form y le asigné un intervalo de 51966 milisegundos (52 segundos aprox.). 51966 en hexadecimal es CAFE así lo vemos facil en un editor hexadecimal jeje. Abro éste ejecutable básico en el VBDE y veo dónde está el Timer:

```
VBDE 0.85 by iorior - C:\Archivos de programa\Microsoft Vis
Log   Project   Forms   Procedure
                                 00001288
Form1
'offset: 000011D5
VERSION 5.00
Begin VB.Form Form1
  Caption      =  "Form1"
  LinkTopic    =  "Form1"
  ClientLeft   =  60
  ClientTop    =  450
  ClientWidth  =  4680
  ClientHeight =  3090
  StartUpsituation =  3  'Windows Default
'offset: 00001218
  Begin VB.Timer Timer1
    Interval     =  51966
    Left         =  1920
    Top          =  1080
  End
End
```

Bien, como vemos, a partir del offset 1218 está el Timer así que con el editor hexadecimal nos vamos a esa direccion y buscamos:

```
00001200  3C 00 00 00 C2 01 00 00  48 12 00 00 12 0C 00 00  <   Å   H
00001210  46 03 FF 01 1F 00 00 00  01 06 00 54 69 6D 65 72  F ÿ       Timer
00001220  31 00 0B 03 FE CA 00 00  07 80 07 00 00 08 38 04  1   þÊ   ▌    8
00001230  00 00 FF 02 04 00 00 00  50 00 00 00 45 2A 77 64    ÿ      P   E*wd
00001240  30 90 18 4A 9A 75 81 C0  28 FD 3D 0B 00 00 00 00  0▌ J▌u▌À(ý=
```

Ahí lo tenemos. Señalado en verde el Timer y el intervalo en hexadecimal pero al reves FECA. Si hacemos el mismo proceso con el crackme buscamos el Timer y su intervalo. Puesto que el crackme tiene un intervalo de 3 segundos (3000 milisegundos) deberíamos buscar su representacion en hexadecimal. 3000=BB8:

```
00001700  00 35 2D 00 00 00 77 01  00 00 0C 12 00 00 56 04   5-    w        V
00001710  00 00 46 02 FF 01 1F 00  00 00 01 06 00 54 69 6D    F ÿ         Tim
00001720  65 72 31 00 0B 03 B8 0B  00 00 07 C8 0A 00 00 08  er1    ¸    È
00001730  D0 02 00 00 FF 03 30 00  00 00 02 08 00 43 6F 6D  Ð   ÿ 0       Com
00001740  6D 61 6E 64 31 00 04 01  05 00 43 68 65 63 6B 00  mand1      Check
```

Es la misma estructura: 4 bytes acontinuacion del Timer tenemos su intervalo. En este caso como en el anterior, está al reves (B80B). Un buen dato para los que van comenzando y quieren aprender como funcionan las cosas. Aquí en el editor hexadecimal, si cambiamos a un valor muy alto,obtendríamos un intervalo de tiempo del tick muy alto. Digamos por ejemplo que lo cambiamos a FFFF que serían 65535 milisegundos o lo que es lo mismo, unos 65 segundos aproximadamente. Con esto lo que hacemos es dejar en paz al SmartCheck y que no monitoree nada relacionado con el Timer porque simplemente el evento del Timer no se va a lanzar hasta que no pase 1 minuto. Tiempo de sobra para meter el serial en el crackme y pulsar el boton de comprobacion del serial. Dudo mucho que alguien pueda tardar tanto desde que escribe el serial hasta que pulsa el boton.A no ser que se haya pasado con la hierba y no atine ni siquiera con el raton jajaja!!!. Bueno, cambiamos el intervalo:

```
00001710  00 00 46 02 FF 01 1F 00  00 00 01 06 00 54 69 6D    F ÿ         Tim
00001720  65 72 31 00 0B 03 FF FF  00 00 07 C8 0A 00 00 08  er1    ÿÿ   È
00001730  D0 02 00 00 FF 03 30 00  00 00 02 08 00 43 6F 6D  Ð   ÿ 0       Com
00001740  6D 61 6E 64 31 00 04 01  05 00 43 68 65 63 6B 00  mand1      Check
```

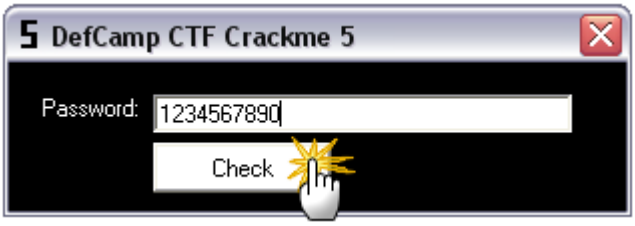
Entonces, ésta modificacion la guardamos con otro nombre por ejemplo “defcamp5_60.exe” para saber que es el que tiene modificado el Timer. Lo pruebo y efectivamente, el título en el crackme cambia cuando pasan 60 segundos aproximadamente. Si abro con el VBDE el crackme original se vería lo siguiente:

```
'offset: 0000171A
  Begin VB.Timer Timer1
    Interval      =  3000
    Left          =  2760
    Top           =  720
  End
'offset: 0000173A
  Begin VB.CommandButton Command1
    Caption       =  "Check"
```

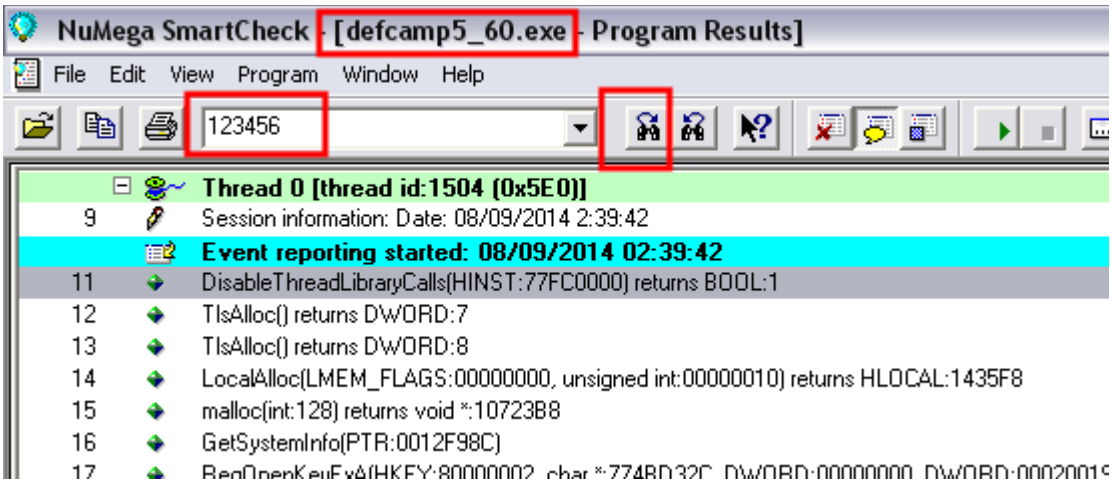
Y el que acabamos de modificar se vería así:

```
  ClientHeight    =  1110
  StartUpSituation =  2  'CenterScreen
'offset: 0000171A
  Begin VB.Timer Timer1
    Interval      =  65535
    Left          =  2760
    Top           =  720
  End
'offset: 0000173A
  Begin VB.CommandButton Command1
```

Vale, ahora queda lo mas complicadillo: estudiar cómo se comprueba el serial. Hay que sacar toda la informacion que podamos del crackme y una buena ayuda sería el VBDecompiler pero yo prefiero utilizar el SmartCheck (SC a partir de ahora) para ver qué hace el crackme cuando pulsamos el boton Check que verifica el serial. Cargo en SC el crackme que acabamos de crear con el Timer modificado y le meto para probar el serial 1234567890 y pulso Check:



Y SC empieza a capturar todos los eventos que lanza el crackme. Esperamos un pelin y termina de capturar y ya podemos empezar a mirar:



Lo mas sensato es intentar buscar el serial que hemos metido en el crackme así que en el campo de busqueda escribimos parte de nuestro serial. Y ojo: digo parte por que si le metemos en el campo de la busqueda el serial completo (1234567890), SC no lo va a encontrar puesto que SC solo maneja los primeros ocho caracteres de las cadenas en la ventana de eventos:

```
__vbaObjSet(LPINTERFACE *:0012F434, LPINTERFACE:014A0D44) ret
Text1.Text
__vbaStrCmp(String:"", String:"12345678...") returns DWORD:1
__vbaFreeStr(LPBSTR:0012F438) returns DWORD:20
__vbaFreeObj(LPINTERFACE *:0012F434)
Chr(Integer:84)
```

En la busqueda, la primera en la que encuentra nuestro serial es la de la imagen anterior. Se ve que solo se ven los primeros ocho caracteres de nuestro serial pero nos vale.Tambien se ve claramente que se compara nuestro serial con una cadena vacía representada con las dobles comillas (“”). Es la típica comparacion para saber si al menos escribimos algo. Es decir, si pulsamos el boton Check del crackme sin haber escrito nada en la caja de textos. Vale, le damos otra vez al icono de busqueda (el prismatico) y la segunda vez que encuentra nuestro serial:

```
Text1.Text
__vbaStrCmp(String:"ThisIsOn...", String:"12345678...") returns DWORD:FFFFFFFF
__vbaFreeStr(LPBSTR:0012F438) returns DWORD:20
__vbaFreeObj(LPINTERFACE *:0012F434)
_TextBox::AddRef(LPINTERFACE:014A0D44) returns DWORD:1
__vbaObjSet(LPINTERFACE *:0012F434, LPINTERFACE:014A0D44) returns LPVOID:14A0
```

Es para compararlo con una cadena (o parte de ella) que por lo que vemos es (ThisIsOn...). Observad lo de los puntos suspensivos: es para indicar que la cadena es más larga de lo que se representa ahí. Y es verdad. SC se divide en dos ventanas: la de la izquierda es la de los eventos y la de la derecha lo que sería para entendernos el stack. No es esactamente eso pero es para pillar el concepto. Seleccionando esa misma linea en la ventana de eventos de SC, en la ventana de la derecha vemos esto:

```
DEFCAMP5_60.EXE!00003942 (no debug info)
unsigned short * string1 = 0014ABB4
    = "ThisIsOneOfTheEasiestInTheWorld"
unsigned short * string2 = 0015665C
    = "1234567890"
```

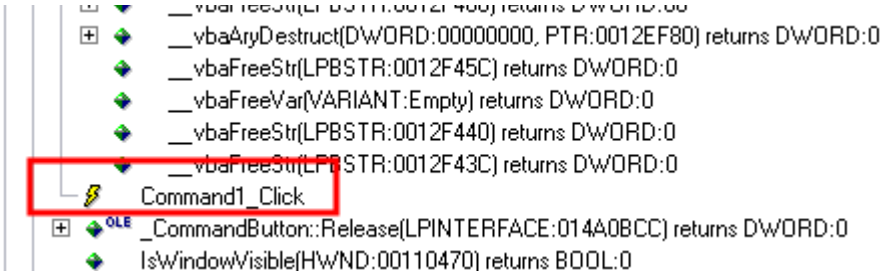
Son las dos cadenas que intervienen en la comparacion: el serial que le escribimos y otra. Ésta comparacion simplemente es para despistar pues si vemos ésto, rapidamente nos entra el nervio pensando que es el serial bueno y no lo es por que si probamos, al pulsar el boton Check, sale un mensaje pitorreandose de nosotros. Así que aquí simplemente el programador juega al despiste con nosotros. Vamos a seguir con la busqueda:

```
Text1.Text
Len(String:"12345678...") returns LONG:10
Long (10) --> Integer (10)
__vbaFreeStr(LPBSTR:0012F438) returns DWORD:20
__vbaFreeObj(LPINTERFACE *:0012F434)
__vbaObjSetAddref(LPINTERFACE *:0012F434, LPINTERFACE:00143238) returns LPVOID:143238
```

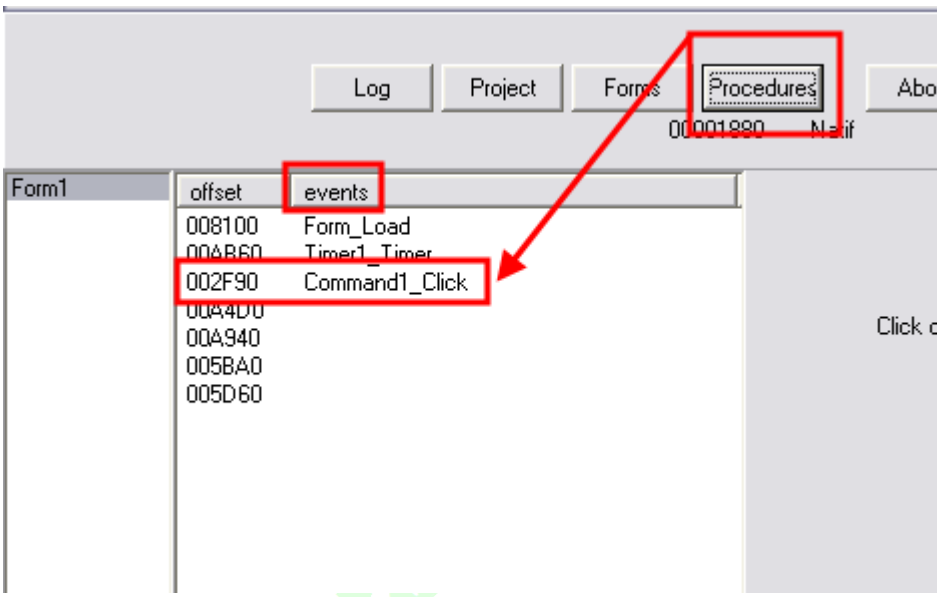
Aquí, en la tercera busqueda se ve claramentenque se toma el largo del serial que escribimos con la funcion Len y el resultado es, obviamente, 10 caracteres. Miro alrededor de esta busqueda y no encuentro nada más. Si vuelvo a hacer una busqueda, vuelve al principio (donde se comparaba con cadena vacía). Así que solo en estas tres busquedas se encuentra nuestro serial. Miré y remiré algo más que me diera pistas. Busqué algún sitio dónde por ejemplo se trabaje con el largo del serial (10 caracteres) pero no encuentro nada de nada. Si se mira algo más abajo solo se encuentran llamadas a DestroyWindow:

```
GetFocus() returns HWND:0
DestroyWindow(HWND:0012046A) returns BOOL:1
HeapFree(HANDLE:01490000, FLAGS:00000000, PTR:014A0CF0) returns BOOL:1
```

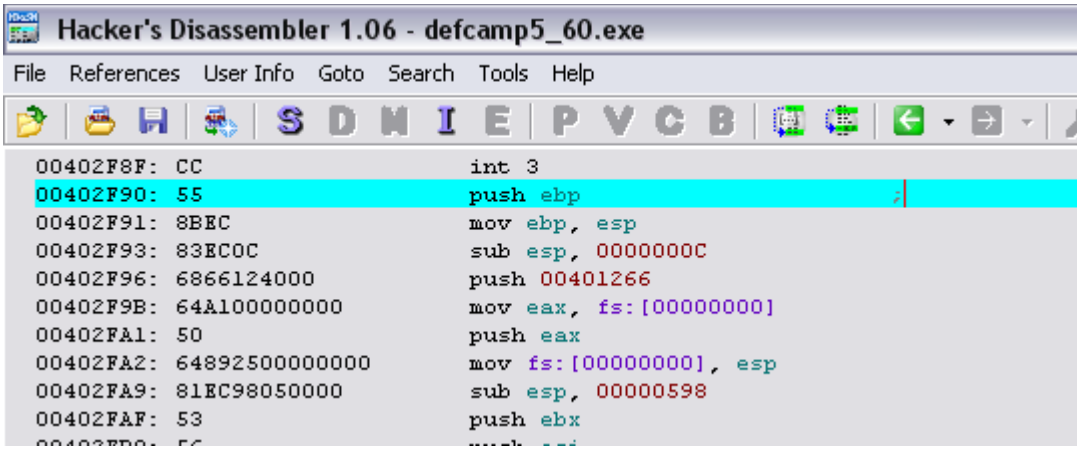
Y la finalizacion del evento Click del boton ó lo que sería para los que trasteamos con la programacion en Visual Basic, el “End Sub”:



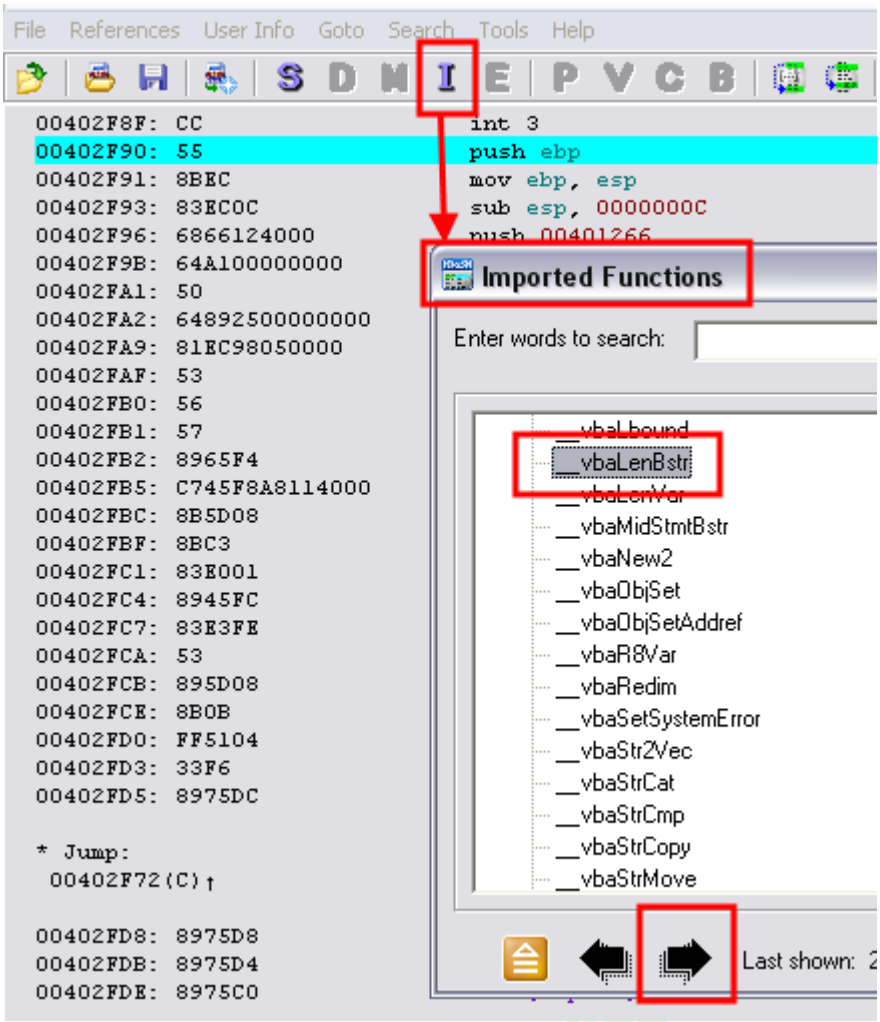
Esto no puede ser. Aquí nos falta informacion pues de repente se acaba el evento click y no vemos nada de nada. Ninguna pista. Algo con lo que podamos trabajar. Yo e mirado 20 veces la configuracion de SC pensando si tengo que modificar dicha configuracion o algo pero o no encuentro nada. Entonces lo que pienso es que SC está pasando por alto informacion. Pero no se por qué así que tendremos que buscar alguna pista más por otro lado. Lo primero que se le puede venir a uno a la cabeza es poder averiguar dónde o en qué direccion comienza el evento Click del boton y ya que tenemos el VBDE, lo utilizamos para averiguarlo:



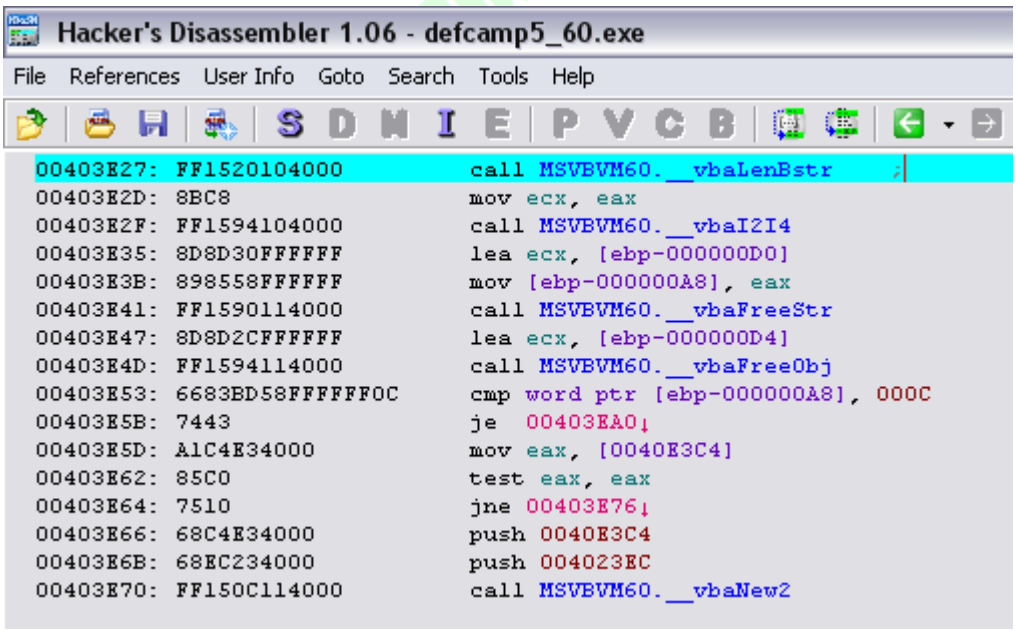
Bueno, abrimos el crackme en el VBDE y pulsamos el boton de los procedimientos y buscamos el evento Click del Boton y esa direccion está en en offset 002F90 así que utilizo por ejemplo el Hacker's Disassembler por que me sirvió perfectamente para lo que quería. Abro el crackme con dicha herramienta y me voy al offset 002F90:



Ahí está marcada en azul la linea donde comienza el evento Click del boton. Como lo que queremos es buscar pistas donde se toma el largo de la cadena que escribimos en la caja de textos, buscamos con ésta misma herramienta la funcion que se encarga de extraer el largo de una cadena. Es la __vbaLenBstr:

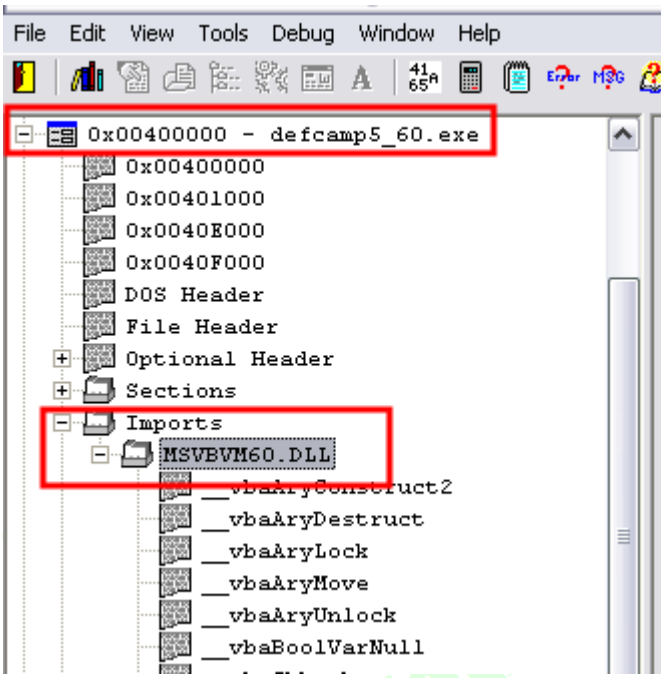


En la toolbar de la herramienta, pulsamos sobre la I para que nos saque las funciones importadas por el crackme y pulsamos sobre la __vbaLenBstr y acontinuacion pulsamos el botoncito de la flecha negra para localizar dicha funcion en el codigo desde la linea que está en azul hacia abajo. Pulsamos sobre la flecha y ahí la encontramos:

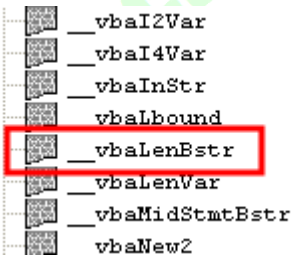


Bien, encontramos esa funcion en el offset 003E27 (403E27). Cuando se llama a esa funcion, el retorno queda en EAX y se mueve a ECX en la siguiente linea. Vale, en la tercera linea (403E2F) se llama a la funcion __vbaI2I4 que lo que hace es convertir a entero el largo de la cadena y depositarla en EAX. Bien, en la linea 403E3B se mueve a [EBP-000000A8] el resultado de dicha conversion y en la linea 403E53 se compara el contenido de [EBP-000000A8] con C que es la reperesentacion hexadecimal de 12.

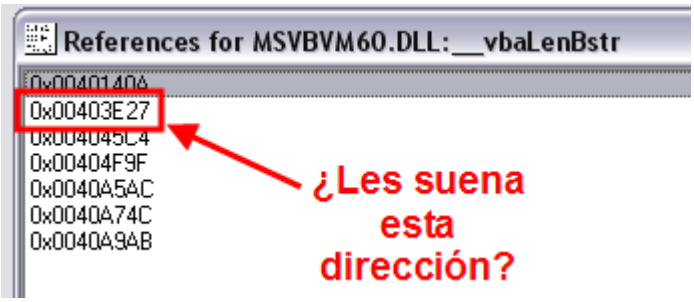
Si resulta que son iguales dichos valores, en la linea 403E5B se salta a otra parte del código. Viendo ésto, a cualquiera ya se le debe haber encendido la bombilla y deducir al momento que ésta era la pista que buscabamos y que SmartCheck no nos la enseñaba. SC precisamente se salta dicha comparacion por eso nos resultaba tan rarísimo que todo terminara así sin mas. Con esto lo que quiero decir es que si una herramienta no te da las respuestas que estás buscando o te desconcierta lo que esa herramienta te ofrece, utiliza otra u otras hasta que comprendas lo que está pasando. Ésto me puede valer para intentar averiguar porqué SC no recoje ésa comparacion que para nosotros en este caso es crucial e importantísima. Se podría haber buscado esta misma pista depurando el crackme y les voy a explicar cómo se puede hacer. Yo voy a utilizar el PeBrowseDbg que tambien nos sirve para depurar NET's. Yo lo vengo utilizando desde hace años y como el crackme tiene proteccion contra Olly e IDA pues me viene de maravilla. Cargo entonces el crackme en el PeBrowseDbg:



Bien, cargado el crackme, expando el arbol y me voy a las imports y vemos las que toma de la librería de los Visual Basic. La famosísima dll MSVBVM60.dll y busco la __vbaLenBstr:



Bien, ahí está. Hago doble click sobre dicha funcion para saber desde qué partes del código del crackme se llama a dicha función:



¿Qué les parece?. La misma dirección que encontrabamos antes así que doble click sobre esa referencia a la función para que PeBrowseDbg nos lleve a esa zona del código:

```
Disassembly of 0x00403E27 in defcamp5_60.exe

ARG/VAR/S  Character  Value
EBP+0x10
EBP+0xC
EBP+0x8
EBP-0x4

;*****************************************************************************
; Section: .text
0x403E27: FF1520104000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaLenBstr]; (0x401020)
0x403E2D: 8BC8                MOV     ECX,EAX
0x403E2F: FF1594104000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaI2I4]; (0x401094)
0x403E35: 8D8D30FFFFFF        LEA     ECX,[EBP-0xD0]
0x403E3B: 898558FFFFFF        MOV     DWORD PTR [EBP-0xA8],EAX
0x403E41: FF1590114000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaFreeStr]; (0x401190)
0x403E47: 8D8D2CFFFFFF        LEA     ECX,[EBP-0xD4]
0x403E4D: FF1594114000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaFreeObj]; (0x401194)
0x403E53: 6683BD58FFFFFF0C    CMP     WORD PTR [EBP-0xA8],0xC
0x403E5B: 7443                JE      0x403EA0          ; (*+0x45)
0x403E5D: A1C4E34000          MOV     EAX,DWORD PTR [0x40E3C4]
```

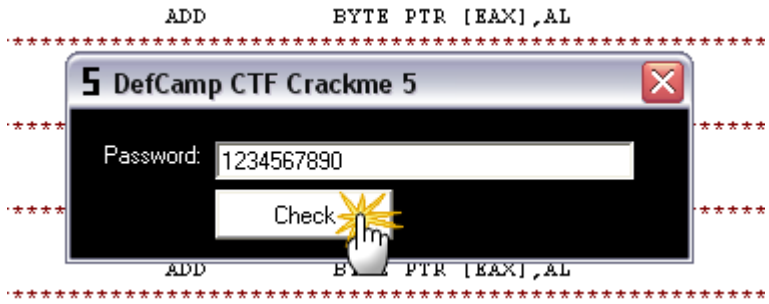
Ahí en grisacea está la linea 403E27 así que le voy a poner un BP y le doy a run para que empiezar a tracear y demostrarles lo que les decía antes:

```
Disassembly of 0x00403E27 in defcamp5_60.exe

ARG/VAR/S  Character  Value
EBP+0x10
EBP+0xC
EBP+0x8
EBP-0x4

;*****************************************************************************
; Section: .text
0x403E27: FF1520104000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaLenBstr];
0x403E2D: 8BC8                MOV     ECX,EAX
0x403E2F: FF1594104000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaI2I4]; (0
0x403E35: 8D8D30FFFFFF        LEA     ECX,[EBP-0xD0]
0x403E3B: 898558FFFFFF        MOV     DWORD PTR [EBP-0xA8],EAX
0x403E41: FF1590114000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaFreeStr];
0x403E47: 8D8D2CFFFFFF        LEA     ECX,[EBP-0xD4]
0x403E4D: FF1594114000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaFreeObj];
0x403E53: 6683BD58FFFFFF0C    CMP     WORD PTR [EBP-0xA8],0xC
0x403E5B: 7443                JE      0x403EA0          ; (*+0x45)
```

Al colocar en dicha linea el BP, se pone en rojo. Doy run, aparece el crackme y pongo el serial "1234567890":



Cuando pulso en el boton Check, PeBrowseDbg se clava en el BP que colocamos y empezamos a tracear:

```
Disassembly of 0x00403E27 in defcamp5_60.exe

ARG/VAR/S  Character  Value
EBP+0x10               +0x00401EB8
EBP+0xC                +0x0012F524
EBP+0x8                +0x0014E380
EBP-0x4                0x00000001

        ; Section: .text
  1     0x403E27: FF1520104000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaLenBstr];
  1 >   0x403E2D: 8BC8                MOV     ECX,EAX
        0x403E2F: FF1594104000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaI2I4]; (0
        0x403E35: 8D8D30FFFFFF        LEA     ECX,[EBP-0xD0]
        0x403E3B: 898558FFFFFF        MOV     DWORD PTR [EBP-0xA8],EAX
        0x403E41: FF1590114000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaFreeStr];
        0x403E47: 8D8D2CFFFFFF        LEA     ECX,[EBP-0xD4]
        0x403E4D: FF1594114000        CALL    DWORD PTR [MSVBVM60.DLL!__vbaFreeObj];
        0x403E53: 6683BD58FFFFFF0C    CMP     WORD PTR [EBP-0xA8],0xC

Registers for EIP: 0x00403E2D
EAX:  0x0000000A (ERROR_BAD_ENVIRONMENT)
EBX:  +0x0014E380 (Default Process Heap + 0xE380)
ECX:  +0x7C92005D
```

Una vez pulsado el boton Check, PeBrowseDbg se para en la linea donde está el BP y pulso F10 para avanzar una linea en el depurador y como ven, el retorno de la funcion __vbaLenBstr queda en EAX y se va a mover a ECX. Sigo traceando y se mueve a [EBP-A8] lo que contiene EAX:

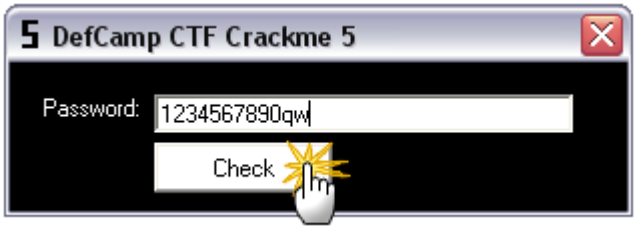
```
ARG/VAR/S  Character  Value
EBP-0xA0               0x01920001
EBP-0xA4               0x00000008
EBP-0xA8               0x0000000A
EBP-0xAC               0x00000000

        ; Section: .text
  1     0x403E27: FF1520104000        CALL    DWORD PTR [MSVBVM60.DLL!
  1     0x403E2D: 8BC8                MOV     ECX,EAX
  1     0x403E2F: FF1594104000        CALL    DWORD PTR [MSVBVM60.DLL!__
  1     0x403E35: 8D8D30FFFFFF        LEA     ECX,[EBP-0xD0]
  1     0x403E3B: 898558FFFFFF        MOV     DWORD PTR [EBP-0xA8],EAX
  1 >   0x403E41: FF1590114000        CALL    DWORD PTR [MSVBVM60.DLL!__
        0x403E47: 8D8D2CFFFFFF        LEA     ECX,[EBP-0xD4]
        0x403E4D: FF1594114000        CALL    DWORD PTR [MSVBVM60.DLL!__
```

Y finalmente sigo traceando y vemos qué se compara:

```
  1     0x403E4D: FF1594114000        CALL    DWORD PTR [MSVBVM60.DLL!__v
  1 >   0x403E53: 6683BD58FFFFFF0C    CMP     WORD PTR [EBP-0xA8],0xC
        0x403E5B: 7443                JE      0x403EA0          ; (*+0x4
```

Y lo vemos ahí: se compara el contenido de [EBP-A8] que contiene A (10) con C (12) así que la intuicion nos dice que al menos, probemos un serial de 12 caracteres de largo, no perdemos nada por probar. Cierro el PebrowseDbg y vuelvo a cargar el crackme en el SC y le meto un serial de 12 caracteres:



Ahora le metí el nuevo serial “1234567890qw”. Pulso el boton Check y espero a que SC termine de capturar. Ya solo por lo que tarda SC en capturar eventos ya se nota que está capturando más operaciones que la vez anterior. Eso ya es algo. Cuando SC termina, me sitúo donde el evento Click termina y tiro un poco hacia arriba y veo lo siguiente:

```
__vbaStrCmp(String:"4108761A", String:"337CE663") returns DWORD:FFFFFFFF
__vbaVarCmpEq(VARIANT:Const String:"", VARIANT:String:"A70086D2") returns DWORD:12F424
__vbaVarCmpEq(VARIANT:Const String:"", VARIANT:String:"14F1163F") returns DWORD:12F414
__vbaVarAnd(VARIANT:Boolean:False, VARIANT:Boolean:False) returns DWORD:12F404
__vbaVarAnd(VARIANT:Boolean:False, VARIANT:Boolean:False) returns DWORD:12F3F4
__vbaBoolVarNull() returns DWORD:0
```

Vemos que hay tres comparaciones distintas. La primera comparacion es con las cadenas “4108761A” y “337CE663”. La segunda compara una cadena vacia con la cadena “A70086D2” y en la ultima comparación se compara una cadena vacía con la cadena “14F1163F”. Quiero saber si esto es arbitrario u obedece a algun patron así que pruebo unos cuantos seriales (todos ellos de 12 caracteres de largo) y observo que hay un patron. Les pongo otra captura donde se ve que el patron que se sigue en las comparaciones es el mismo y siempre están las mismas cadenas excepto una (en azul) que es el resultado de las operaciones que se hace con el serial:

```
SysFreeString(BSTR:0015684C)
__vbaStrCmp(String:"4108761A", String:"A9BB1374") returns DWORD:1
__vbaVarCmpEq(VARIANT:Const String:"", VARIANT:String:"A70086D2") returns DWORD:12F424
__vbaVarCmpEq(VARIANT:Const String:"", VARIANT:String:"14F1163F") returns DWORD:12F414
__vbaVarAnd(VARIANT:Boolean:False, VARIANT:Boolean:False) returns DWORD:12F404
__vbaVarAnd(VARIANT:Boolean:False, VARIANT:Boolean:False) returns DWORD:12F3F4
__vbaBoolVarNull() returns DWORD:0
```

Así que la logica nos dice que se toma el serial, se hacen las operaciones con los caracteres de dicho serial y al final, se compara el resultado de dichas operaciones con ésas cadenas que están encuadradas en rojo.Bien, sabiendo esto, toca revisar todo lo que a capturado SC y localizamos dónde se empieza a tomar los caracteres del último serial que escribí “qwertyuiopas”:

```
__vbaStrCmp(String:"N", String:"N") returns DWORD:0
Integer (1) --> Long (1)
Mid(VARIANT:ByRef String:"qwertyui...", long:1, VARIANT:Integer:1)
__vbaStrVarVal(VARIANT:String:"q") returns DWORD:151DE4
Asc(String:"q") returns Integer:113
__vbaVarMove(VARIANT:Integer:113, VARIANT:Integer:0) returns DWORD:12F488
__vbaFreeStr(LPBSTR:0012F438) returns DWORD:0
SysFreeString(BSTR:00151DE4)
```

Aquí vemos cómo se toma el primer carácter del serial que escribí la ultima vez que estuve probando varios seriales para comprobar el patron de comparaciones. Repito, puse el serial “qwertyuiopas” y con la funcion Mid se toma carácter a carácter. En la imagen se toma el primero que es el carácter “q” que corresponde al ASCII 113. Sigo revisando en SC y más abajo veo que se le resta uno al 113 (113-1=112):

```
__vbaStrCmp(String:"F", String:"F") returns DWORD:0
__vbaVarSub(VARIANT:Integer:113, VARIANT:Integer:1) returns DWORD:12F424
__vbaVarMove(VARIANT:Integer:112, VARIANT:Integer:113) returns DWORD:12F488
Len(String:"MANAAAFA...") returns LONG:461
```

Sigo mirando hacia abajo y veo que se le vuelve a restar uno al resultado anterior (112). Sigo revisando y se repite la operación de resta hasta que al final, con varias restas, se le resta en total 8 al valor inicial es decir: de 113 pasa a 105. Éste 105 se multiplica con otra cifra:

```
__vbaVarMul(VARIANT:Integer:107, VARIANT:Integer:105) returns DWORD:12F424
__vbaVarMove(VARIANT:Integer:11235, VARIANT:Integer:107) returns DWORD:12F4B8
```

Ok. ¿Y éste 107 de donde sale?. Ahi que volver hacia arriba otra vez revisando con mucho cuidado hasta descubrir de dónde sale el 107:

```
Integer (1) --> Long (1)
Mid(VARIANT:ByRef String:"abcdefgh...", long:1, VARIANT:Integer:1)
__vbaStrVarVal(VARIANT:String:"a") returns DWORD:1566CC
Asc(String:"a") returns Integer:97
__vbaVarMove(VARIANT:Integer:97, VARIANT:Integer:0) returns DWORD:12F4B8
```

Y es de aquí. Se toma el primer carácter de una cadena fija con la funcion Mid:

```
DEFCAMP5_60.EXE!00004A13 (no debug info)
string (variant)
  unsigned short * * .pbstrVal = 0012F43C
    String  = 00156C24
      = "abcdefghijklmnopqrstuvwxyz0123456789"
Long  length = 1 0x00000001
start (variant)
  Integer  .iVal = 1 0x0001
```

Se toma la “a” que corresponde al ASCII 97 y se le va sumando progresivamente de uno en uno hasta conseguir 107 así que ya sabemos de dónde sale éste 107. Retomando: 107x105=11235. Sigo revisando cuidadosamente en SC y se toma el segundo carácter del serial que escribimos “qwertyuiopas”:

```
Integer (2) --> Long (2)
Mid(VARIANT:ByRef String:"qwertyui...", long:2, VARIANT:Integer:1)
__vbaStrVarVal(VARIANT:String:"w") returns DWORD:151454
Asc(String:"w") returns Integer:119
__vbaVarMove(VARIANT:Integer:119, VARIANT:Integer:105) returns DWORD:12F488
__vbaFreeStr(LPBSTR:0012F438) returns DWORD:0
```

La letra “w” cuyo ASCII es 119 y la operación que se hace ahora es sumar el resultado anterior con 119:

```
__vbaStrCmp(String:"K", String:"K") returns DWORD:0
__vbaVarAdd(VARIANT:Integer:11235, VARIANT:Integer:119) returns DWORD:12F424
__vbaVarMove(VARIANT:Integer:11354, VARIANT:Integer:11235) returns DWORD:12F4B8
```

Tenemos por ahora 11354. Se toma de nuevo 119 y se hace un xor con -1 y al resultado se le hace un and con -1:

```
__vbaVarXor(VARIANT:Integer:119, VARIANT:Integer:-1) returns DWORD:12F424
__vbaVarAnd(VARIANT:Integer:-120, VARIANT:Integer:-1) returns DWORD:12F414
__vbaVarMove(VARIANT:Integer:-120, VARIANT:Integer:119) returns DWORD:12F488
```

Bien, obtenemos -120 y lo siguiente es multipllicar 11354 x -120=-1362480

```
__vbaVarMul(VARIANT:Integer:11354, VARIANT:Integer:-120) returns DWORD:12F424
__vbaVarMove(VARIANT:Long:-1362480, VARIANT:Integer:11354) returns DWORD:12F4B8
Len(String:"MANAAAFA...") returns LONG:461
```

Quisiera hacer un inciso: según estoy obserbando en SC las operaciones que hace el crackme, veo esto que lo utiliza muchísimo el crackme:

```
__vbaVarMove(VARIANT:Integer:-120, VARIANT:Integer:11
Len(String:"MANAAAFA...") returns LONG:461
Mid(VARIANT:ByRef String:"MANAAAFA...", long:29, VARIA
```

El crackme accede infinidad de veces a esta cadena, que según la funcion Len, tiene un largo de 461 caracteres. ¡¡¡Toma yá!!!. SC en la ventana de eventos sólo nos enseña los primeros 8 caracteres de esa cadena. La ventana de los argumentos, nos enseña algunos caracteres más:

```
DEFCAMP5_60.EXE!000045C5 (no debug info)
⊟ String string1 = 001550AC
    = "MANAAAFAAFAAFAweFAFFFFQGINKOQareIIIIIIITOKGNUQGNQKJsorryJJTKVWGI"
```

Tampoco nos enseña todos en realidad, pues según la funcion Len, son 461. Si vamos a un editor hexadecimal y buscamos esta cadena la exportamos al portapapeles obtenemos lo siguiente:

```
MANAAAFAAFAAFAweFAFFFFQGINKOQareIIIIIIITOKGNUQGNQKJsorryJJ
TKVWGIINRTQtoGGQJJJJIIIIIIIinterruptIITOKENHNQKAAAAyourAAA
AAAAAAAAAAAAworkAAAAAAAAAAbutAAAAAAweIIIIIwouldTUGGNOQVlik
eWGNIMEEtoEEEEEEEEEtellEEEEEEEyouEEQGNFFFFthatFFFFFFFFFFth
eFOFFFFFFFFinputFFFFFyouFFKKKKKareKKKKKKKQSORTsearchingINT
UGNFFFFforFEEEEEEEEEEEconsistsEEEEEEEEEEEEofEEEEEEEEEEonly
EEEEEElowercaseEQIIIIIhexIIIIIITOAAcharactersAAAAfromAAAKGJ
aJIIItoIIJNOfKTFFandFFFFFFnumbersFFFFF0FFFFtoFFFFF9QNQ
```

Vemos una sucesion de caracteres pero si nos fijamos muy bien, no todos son caracteres en mayusculas. Hay otros tanto en minúscula y si nos fijamos más aún, los caracteres en minúsculas forman una frase con sentido y que nos da una pista fundamental para resolver el crackme:

```
MANAAAFAAFAAFAweFAFFFFQGINKOQareIIIIIIITOKGNUQGNQKJsorryJJ
TKVWGIINRTQtoGGQJJJJIIIIIIIinterruptIITOKENHNQKAAAAyourAAA
AAAAAAAAAAAAworkAAAAAAAAAAbutAAAAAAweIIIIIwouldTUGGNOQVlik
eWGNIMEEtoEEEEEEEEEtellEEEEEEEyouEEQGNFFFFthatFFFFFFFFFFth
eFOFFFFFFFFinputFFFFFyouFFKKKKKareKKKKKKKQSORTsearchingINT
UGNFFFFforFEEEEEEEEEEEconsistsEEEEEEEEEEEEofEEEEEEEEEEonly
EEEEEElowercaseEQIIIIIhexIIIIIITOAAcharactersAAAAfromAAAKGJ
aJIIItoIIJNOfKTFFandFFFFFFnumbersFFFFF0FFFFtoFFFFF9QNQ
```

No los señalé todos en verde. Solo para que se fijen el mensaje oculto que hay en esa cadena tan larga de 461 caracteres. La extraigo completa:

```
we are sorry to interrupt your work but we would like to tell you that the input you
are searching for consists of only lowercase hex characters from a to f and
numbers 0 to 9
```

Pues está más que claro: cristalino. Hablando en cristiano el autor del crackme viene a decir que siente interrumpir nuestro trabajo, pero que la cadena que se está buscando consiste en solo caracteres hexadecimales en minúsculas desde la a hasta la f y numeros desde el 0 hasta el 9. Así que esta info nos viene fenomenal para luego programar un bruteforce. Ya sabiendo qué caracteres intervienen, la fuerza bruta no se va a demorar demasiado y va a encontrar un serial bastante rapidito.
Bien, aclarando ésto seguimos por donde se quedó.
Teniamos que las operaciones se quedó cuando obteníamos -1362480. Ahora, se toma el noveno carácter de la cadena fija “abcdefghijklmnopqrstuvwxyz0123456789”:

```
  Integer (9) --> Long (9)
⊞ Mid(VARIANT:ByRef String:"abcdefgh...", long:9, VARIANT:Integer:1)
  __vbaStrVarVal(VARIANT:String:"i") returns DWORD:156AFC
⊞ Asc(String:"i") returns Integer:105
  __vbaVarMove(VARIANT:Integer:105, VARIANT:Integer:-120) returns DW
```

Se toma el carácter “i” que tiene el ASCII 105 y se le hace un xor con -1 y el resultado se le hace un and con -1:

```
__vbaStrCmp(String:"U", String:"U") returns DWORD:0
__vbaVarXor(VARIANT:Integer:105, VARIANT:Integer:-1) returns DWORD:12F424
__vbaVarAnd(VARIANT:Integer:-106, VARIANT:Integer:-1) returns DWORD:12F414
__vbaVarMove(VARIANT:Integer:-106, VARIANT:Integer:105) returns DWORD:12F488
Len(String:"MANAAAFA  ") returns LONG:461
```

105 xor -1=-106. -106 and -1=-106 y acontinuacion, se suma -1362480 + -106= -1362586:

```
__vbaVarAdd(VARIANT:Long:-1362480, VARIANT:Integer:-106) returns DWORD:12F424
__vbaVarMove(VARIANT:Long:-1362586, VARIANT:Long:-1362480) returns DWORD:12F4E
```

Ok. Ahora, se toma el tercer carácter del serial que escribimos:

```
Integer (3) --> Long (3)
Mid(VARIANT:ByRef String:"qwertyui...", long:3, VARIANT:Integer:1)
__vbaStrVarVal(VARIANT:String:"e") returns DWORD:156504
Asc(String:"e") returns Integer:101
__vbaVarMove(VARIANT:Integer:101, VARIANT:Integer:-106) returns DWORD:12F488
```

El tercer carácter del serial que escribimos en el crackme es la “e” que tiene el codigo ASCII 101. Seguidamente se hace un xor entre -1362586 y 101. -1362586 xor 101= -1362685:

```
__vbaVarXor(VARIANT:Long:-1362586, VARIANT:Integer:101) returns DWORD:12F424
__vbaVarMove(VARIANT:Long:-1362685, VARIANT:Long:-1362586) returns DWORD:12F
```

Lo siguiente es multiplicar -1362685 x 101= -137631185:

```
__vbaVarMul(VARIANT:Long:-1362685, VARIANT:Integer:101) returns DWORD:12F424
__vbaVarMove(VARIANT:Long:-137631185, VARIANT:Long:-1362685) returns DWORD:1
```

Ahora se toma el cuarto carácter del serial que escribimos:

```
Integer (4) --> Long (4)
Mid(VARIANT:ByRef String:"qwertyui...", long:4, VARIANT:Integer:1)
__vbaStrVarVal(VARIANT:String:"r") returns DWORD:15664C
Asc(String:"r") returns Integer:114
```

El cuarto carácter es la “r” ASCII 114 y se multiplica -137631185 x 114= -15689955090

```
__vbaVarMul(VARIANT:Long:-137631185, VARIANT:Integer:114) returns DWORD:12F42
__vbaVarMove(VARIANT:Double:-1,569e+010, VARIANT:Long:-137631185) returns DWC
```

Ahora, se suma -15689955090 + 114= -15689954976. Para aliviar un poco el tuto, voy a exponer las operaciones que se hacen a continuacion sin necesidad de ir poniendo más y más imagenes. Creo que se hace innecesario. Con las imagenes se demuestra cómo funciona todo pero son imagenes que practicamente se repiten con diferentes valores. Podemos prescindir de ellas. Seguimos.
Ahora se toma el sexto carácter de la cadena fija “abcdefghijklmnopqrstuvwxyz0123456789”. El carácter es la “f” con el ASCII 102.
Se suma -15689954976 + 102= -15689954874.
Se niega -15689954874. __vbaVarNeg(-15689954874)=15689954874.
Se divide 15689954874 / 16= 980622179,625.
Se toma la parte entera de 980622179,625. __vbaVarInt(980622179,625)= 980622179.
Se resta 980622179,625 – 980622179= 0,625
Se multiplica 0,625 x 16= 10
Se resta 15689954874 – 10= 15689954864
Se divide 15689954864 / 16= 980622179
Ahora, se toma el resultado obtenido tres lineas antes. El 10. El 10 en formato cadena hexadecimal es la “A”. Ya tenemos el primer carácter de una cadena de ocho caracteres que posteriormente se tendrían que compararar con los patrones encuadrados en rojo de la pagina 10.

Seguimos.
Se divide 980622179 / 16= 61288886,1875.
Se toma la parte entera de 61288886,1875= 61288886
Se resta 61288886,1875 – 61288886= 0,1875
Se multiplica 0,1875 x 16= 3
Se resta 980622179 – 3= 980622176
Se divide 980622176 / 16= 61288886
Ahora, se toma el resultado obtenido tres lineas antes. El 3. El 3 en formato cadena hexadecimal es obviamente “3” con lo que tenemos un segundo carácter que se le añade al obtenido en la pagina anterior. Tenemos hasta ahora “A3”.
Seguimos.
Se divide 61288886 / 16= 3830555,375
Se toma la parte entera de 3830555,375= 3830555
Se resta 3830555,375 – 3830555= 0,375
Se multiplica 0,375 x 16= 6
Se resta 61288886 – 6= 61288880
Se divide 61288880 / 16= 3830555
Ahora, se toma el resultado obtenido tres lineas antes. El 6. El 6 en formato cadena hexadecimal es “6”. Éste carácter “6” se añade a la cadena “A3” de antes y vamos obteniendo “A36”.
Si se fijan, éstas operaciones son un bucle. Agrupé por colores los valores que se utilizan para el siguiente bucle.Espero que lo entiendan. Pongo todo mi empeño en explicarles de la mejor forma posible para que no se pierdan. Puedo entender que es algo dificil asumirlo para algunos.
Como es un bucle, dicho bucle continúa mientras el resultado de la division sombreada en verde sea mayor que cero. Cuando diho resultado es cero, se termina el bucle y se toman los caracteres recolectados hasta ese momento. Para éste caso particular, la cadena que se obtiene es “A36B137A3”. Ésta cadena se invierte con lo que pasamos a tener “3A731B63A”. De ésta última cadena se extraen los ocho primeros con la funcion Mid:

```
Len(String:"3A731B63...") returns LONG:9
Mid(VARIANT:ByRef String:"3A731B63...", long:1, VARIANT:Integer:8)
__vbaStrVarMove(VARIANT:String:"3A731B63") returns DWORD:156AD4
__vbaStrMove(String:"3A731B63", LPBSTR:0012F484) returns DWORD:156AD4
```

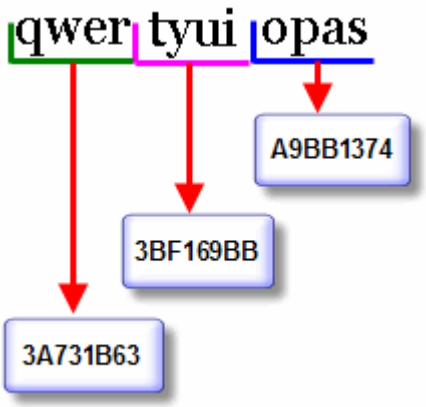
Y tenemos al final “3A731B63”

```
DEFCAMP5_60.EXE!00004FF9 (no debug info)
string (variant)
unsigned short * ,bstrVal = 00156AD4
= "3A731B63"
```

Recopilando datos:

Para llegar aquí, del serial que escribimos en el crackme se toman los indices 1,2,3 y 4 y de la cadena fija “abcdefghijklmnopqrstuvwxyz0123456789” se toman los indices 1,9 y 6. Al ASCII del primer carácter del serial se le resta 8 y al ASCII del carácter indice 1 de la cadena fija, se le suma 10. Y se empieza a hacer todas las operaciones que hemos visto hasta ahora.
Os voy a ahorrar el tener que leer practicamente lo mismo sobre el resto de las operaciones pero para resumir un poco,os digo lo que se hace para buscar el segundo patron:
De la cadena fija se toman los caracteres con indices 8,13 y 18 y del serial que escribimos se toman los caracteres con indices 5,7,6 y 8. Como ya sabemos cómo es el bucle, se realizan las operaciones pertinentes con los ASCII de ellos y obtenemos como resultado la cadena “3BF169BB”. Ya tenemos la segunda cadena. Vamos a por la tercera.
Se toman del serial que escribimos los caracteres con indices 9,10,10,11 y 12 y de la cadena fija se toman los caracteres con indices 19,19,20,30 y 32. Se opera con los valores ASCII de todos ellos y obtenemos la cadena “A9BB1374”. Ya tenemos las tres cadenas que se van a comparar con los patrones y ésta última cadena obtenida es la que se ve en la imagen de la pagina 10 encuadrada en azul. Teniendo ya todos estos datos podemos programar un bruteforce de la siguiente manera:

Teniendo en cuenta que el serial debe ser de 12 caracteres y que gracias al programador con la pista que nos dió, dichos caracteres pueden ser desde la “a” hasta la “f” y desde el “0” hasta el “9”, un ejemplo de serial sería así: “a1b2c3d4e5f6”. Cualquier numero de “0” a “9” y cualquier letra minuscula de “a” a “f”. Bien, por lo que hemos visto hasta ahora, podriamos hacer un pequeño grafico de lo que se a conseguido:



Éstas son las cadenas que se obtienen con el serial que escribimos en el crackme. Segun vemos en la pagina 10, se compara la cadena “A9BB1374” que hemos obtenido con los cuatro últimos caracteres del serial, con el patron “4108761A” así que deberíamos encontrar qué caracteres tendrían que ir en los cuatro ultimos del serial que escribamos para que la comparacion sea entre iguales. Lo mismo para los otros dos grupos restantes del serial. Para programar el bruteforce utilizé el antiguo IDE de Visual Basic 6 (qué tiempos aquellos) que imagino que al generar ejecutables tan livianos y que se ejecutan bien rapido en el ordenador, para un bruteforce seguro que va mas rapido que si lo programo en NET. Igual algun dia hago la prueba. No se. Bien, yo pensé en cuatro bucles For-Next y que cada uno tome los caracteres de la cadena "abcdef0123456789" pues ésa es la pista que nos dió el programador del crackme:

```
Private Sub Form_Load()
DoEvents
Patron = "abcdef0123456789"
For Contador1 = 1 To Len(Patron)
CaracterCont1 = Mid(Patron, Contador1, 1)
For Contador2 = 1 To Len(Patron)
CaracterCont2 = Mid(Patron, Contador2, 1)
For Contador3 = 1 To Len(Patron)
CaracterCont3 = Mid(Patron, Contador3, 1)
For Contador4 = 1 To Len(Patron)
CaracterCont4 = Mid(Patron, Contador4, 1)
Resultado1 = Algoritm(CaracterCont1 & CaracterCont2 & CaracterCont3 & CaracterCont4, 1, 2,
    If Resultado1 = "A70086D2" Then
    Text1.Text = CaracterCont1 & CaracterCont2 & CaracterCont3 & CaracterCont4
    Exit For
    End If
Resultado2 = Algoritm(CaracterCont1 & CaracterCont2 & CaracterCont3 & CaracterCont4, 1, 2,
    If Resultado2 = "14F1163F" Then
    Text2.Text = CaracterCont1 & CaracterCont2 & CaracterCont3 & CaracterCont4
    Exit For
    End If
Resultado3 = Algoritm(CaracterCont1 & CaracterCont2 & CaracterCont3 & CaracterCont4, 1, 2,
    If Resultado3 = "4108761A" Then
    Text3.Text = CaracterCont1 & CaracterCont2 & CaracterCont3 & CaracterCont4
    Exit For
    End If
Next
Next
Next
Next
```

Ok, los cuatro bucles For-Next y el ultimo bucle, lo que hace es enviar a una funcion (a la que llamé Algoritm), los cuatro caracteres concadenados. A dicha funcion se entra con los caracteres encadenados, los indices de posicion tanto de la cadena fija “abcdefghijklmnopqrstuvwxyz0123456789” como los indices

de posicion de los caracteres encadenados con los que tiene que trabajar el algoritmo. Suena un poco lio pero si siguieron atentamente el tuto, saben a qué me refiero. Una partede la funcion es esta que por tamaño no está entera:

Proyecto1 - Module1 (Código)

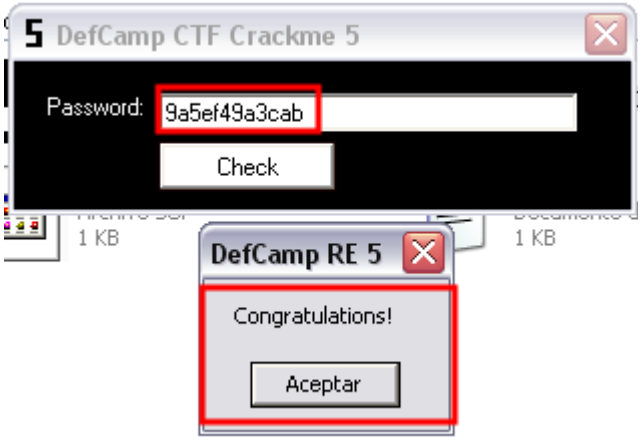
(General) | Algoritm

```
Public Function ExtraerCaracteresDeFija(ByVal numero As Integer) As String
CadenaFija = "abcdefghijklmnopqrstuvwxyz0123456789"
CaracterCadFija = Mid(CadenaFija, numero, 1)
ExtraerCaracteresDeFija = CaracterCadFija
End Function

Public Function Algoritm(ByVal Caracteres As String, ByVal Indice1 As Integer, ByVal Indice2
DoEvents
CaracterSerial = Mid(Caracteres, Indice1, 1)
AsciiCaracterSerial = Asc(CaracterSerial)
If Ext2 = 32 Then
AsciiCaracterSerial = AsciiCaracterSerial + Sumador
Else
AsciiCaracterSerial = AsciiCaracterSerial - Restador
End If
CaracterCadFija = ExtraerCaracteresDeFija(IndiceCadFija1)
AsciiCaracterCadFija = Asc(CaracterCadFija)
If CaracterCadFija = "s" Then
GoTo Salto0
CaracterSerial = Mid(Caracteres, Indice2, 1)
AsciiCaracterSerial = Asc(CaracterSerial)
AsciiCaracterSerial = AsciiCaracterSerial - Restador
AsciiCaracterSerial = XorAnd(AsciiCaracterSerial)
AsciiCaracterSerial = AsciiCaracterSerial - Restador
Else
AsciiCaracterCadFija = AsciiCaracterCadFija + Sumador
End If
Salto0:
```

En rosita,la funcion Algoritm y parte de sus argumentos. Incluí en la funcion, argumentos para las sumas y restas de valores. Si ven en la pagina 11, que se suman 10 al ASCII de “a”, se utiliza otro valor diferente para la resta de otro carácter así que a la funcion se le aporta el valor necesario para dicha resta.
Ésta funcion retorna una cadena con ocho caracteres alfanumericos fruto de las operaciones de la funcion.
En verde, una funcion para cuando se tenga que extraer los caracteres de la cadena fija.
En azul, algo que siempre recomendaban no utilizar. Los famosos GoTo. Que no se porqué siempre le dieron mala fama. A veces yo los utilizo y me llevo bien con ellos. Si me sirven para algo, los utilizo y punto.
Como sabran, en la primera vuelta de los cuatro bucles, se le aporta a la funcion los caracteres “aaaaaaaaaaaa”. En los bucles anidados, el último es el que se sigue ejecutando y hasta que no termine no se pasa al anterior y así hasta llegar al primero. La siguiente cadena que entraría en la funcion Algoritm sería la “aaaaaaaaaaab”, la siguiente sería “aaaaaaaaaaac” y así sucesivamente hasta que se utilizen todos los caracteres necesarios para buscar un serial valido. Recuerden que eran caracteres de la “a” hasta la “f” y numeros del “0” hasta el “9”.
El ultimo bucle recoge el retorno de la funcion y se compara dicho retorno con los tres patrones a conseguir y si uno de ellos coincide, hago plasmar en un TextBox los caracteres con los que se entró en la funcion para que se vean con qué cuatro caracteres se encuentra un patron. Bien, pongo en marcha el bruteforce y en menos de 15 segundos (pensaba que iba a tardar más) me encuentra un serial:

Bueno, pues la mejor forma de probar si todo lo que hicimos mereció la pena es meter este serial “9a5ef49a3cab” en el crackme:



A la vista está que mereció la pena. Y me dió satisfaccion doble porque éste crackme me hizo sudar sangre y casi perder la vista pues practicamente me tuvo analizando linea por linea en SC y apuntando en un papel.A veces, crackme como este hace que perdamos la paciencia y la ilusion pero esto es así. Perseverar hasta encontrar la solucion aunque nos cueste.
Bueno amigos de CracksLatinos, un saludos y gracias por la atencion prestada.
¡¡Hasta la proxima!!

sequeyo
crackslatinos