CURSO DE CRACKING DESDE CERO

POR RICARDO NARVAJA

GRUPO CRACKS LATINOS

Recopilado en PDF y DOC por Ice Cube (Marzo 2003)

Contenido

LECCIÓN 01: CURSO DE CRACKEO PARA EMPEZAR DESDE CERO

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 DIFÍCIL CRACKEAR? HAY QUE TENER GRANDES CONOCIMENTOS?

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 lecciónes, 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 cracker 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 último 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, más 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 más 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 victima. 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 lecciónes, 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 victima, ejecutarlo y ver que tipo de programa es (CON TIPO DE PROGRAMA ME REFIERO A QUE SI SE REGISTRA CON UN NÚMERO DE SERIE, SI VENCE DENTRO DE UN PLAZO, ETC), en nuestro primer caso es un programa que se registra con un número 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 NÚMERO 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 número 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 NÚMERO 989898 lo elegí porque el 999999 ya esta muy quemado (BAH CUALQUIER NÚMERO 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 NÚMERO INVALIDO o cualquier otro mensaje de ese deprimente estilo.

Este mensaje además de decirnos que el número 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 número 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 número de serie, llega un punto en que INEVITABLEMENTE va a comparar el número que pusimos equivocado (989898 en el ejemplo), con el número verdadero de serie.
Tiene que comparar si o si, el número 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 NÚMERO CORRECTO REGISTRADO o algo asi.

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 cracker 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 más 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 asi hasta la F que es el 15.

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.

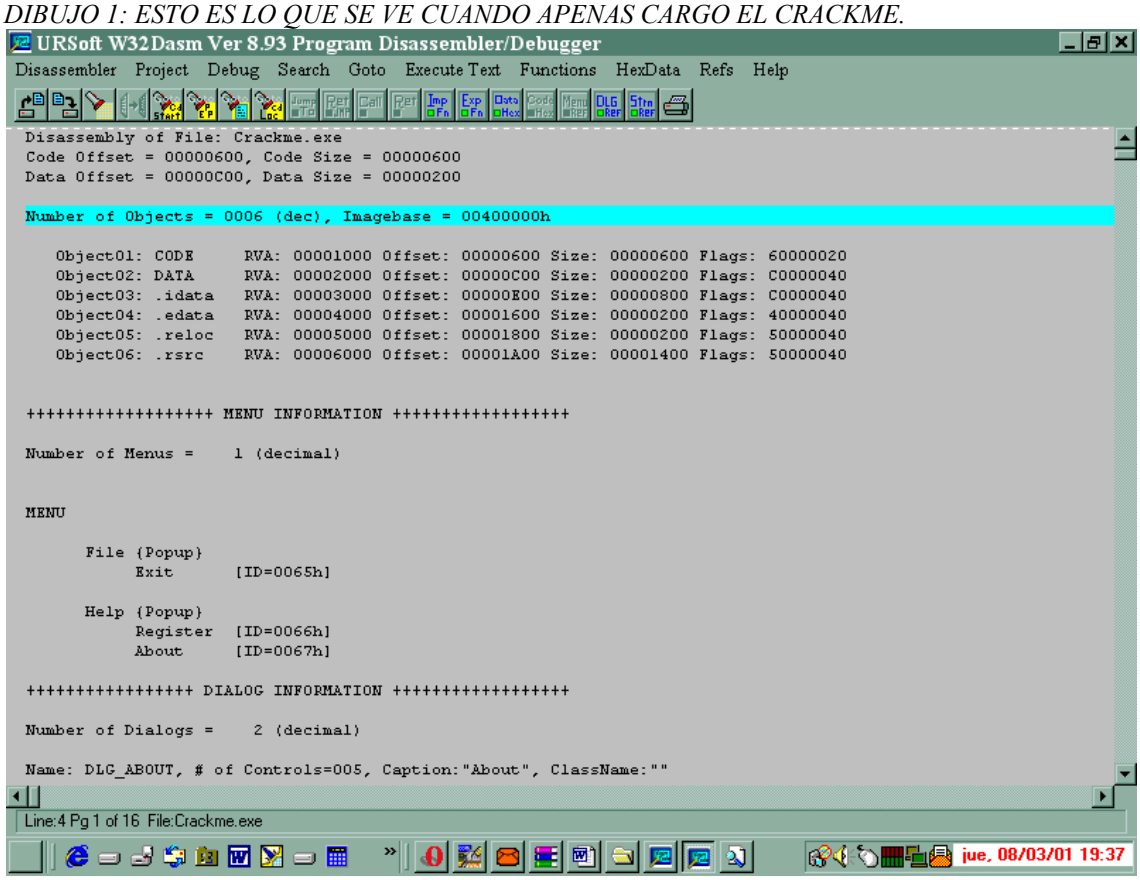
No se asusten que no es tan fiero el LEON como lo pintan, abramos el WDASM y en donde dice DISASSEMBLER hagamos clic y elijamos OPEN FILE TO DISSASEMBLE para cargar la fila a destripar.

Buscamos por los directorios la fila crackme.exe y la cargamos.

Bueno como es una fila chiquita se desensambla rapido, bueno ahí tenemos vamos despacio para no marearnos.

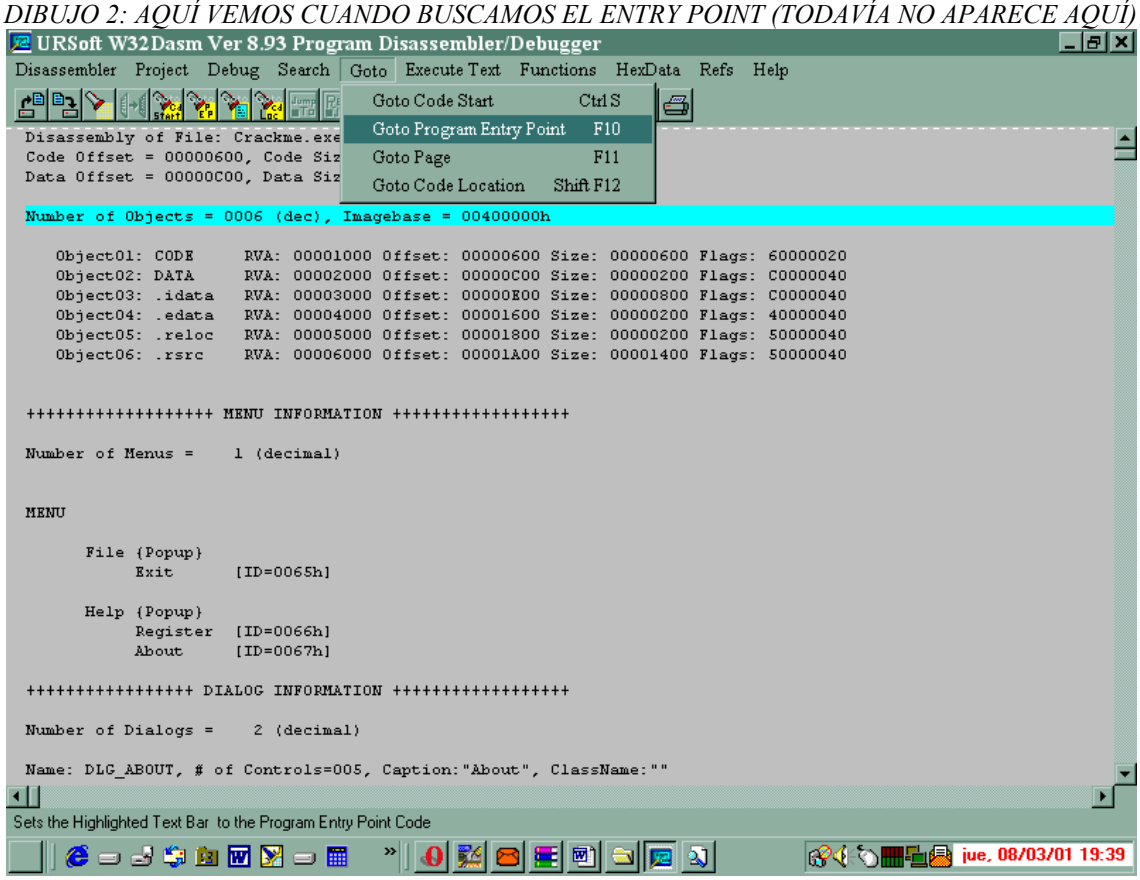
Lo primero que hay son datos sobre el archivo que por ahora no vamos a ahondar (YA LLEGARA SU MOMENTO), la primera pregunta que surge es bueno donde comienza el programa, cual es la primera sentencia que se ejecuta?

DIBUJO 1: ESTO ES LO QUE SE VE CUANDO APENAS CARGO EL CRACKME.



Bueno eso es bastante fácil, abrimos el menu 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)



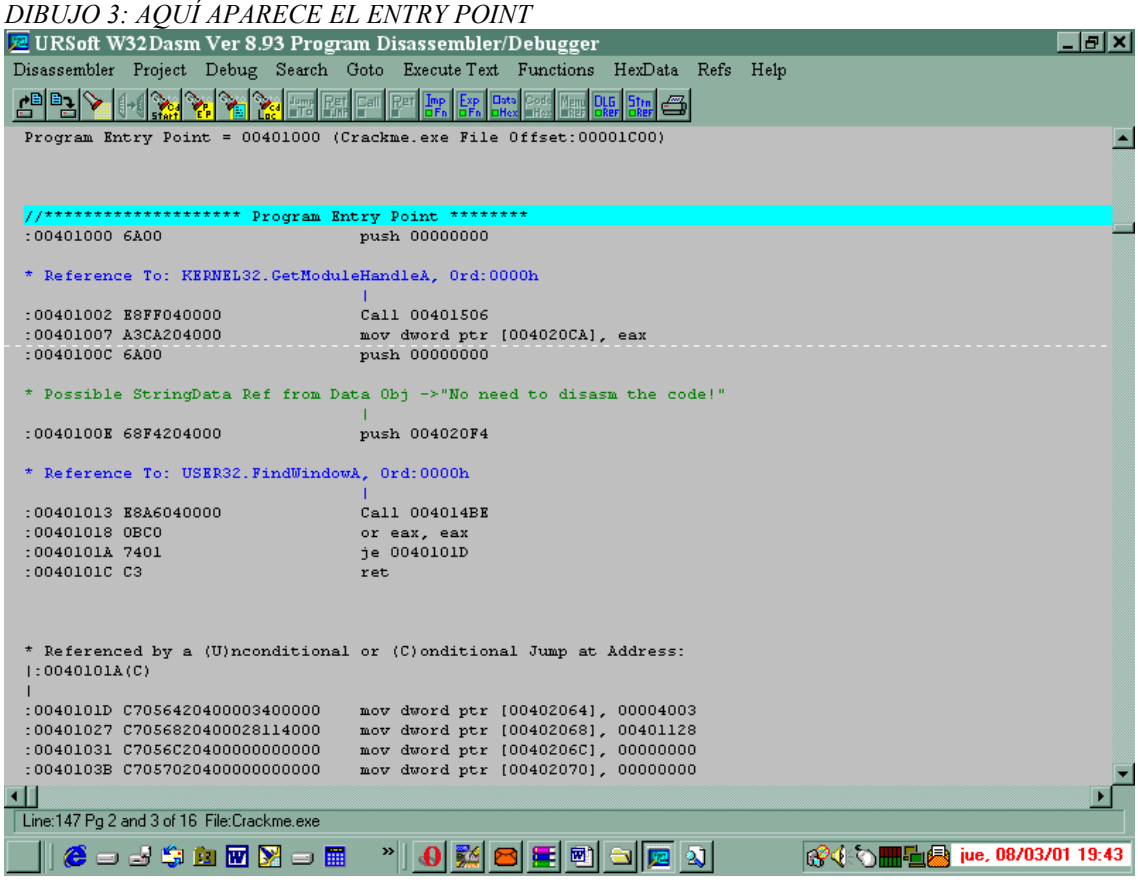
PROGRAM ENTRY POINT

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



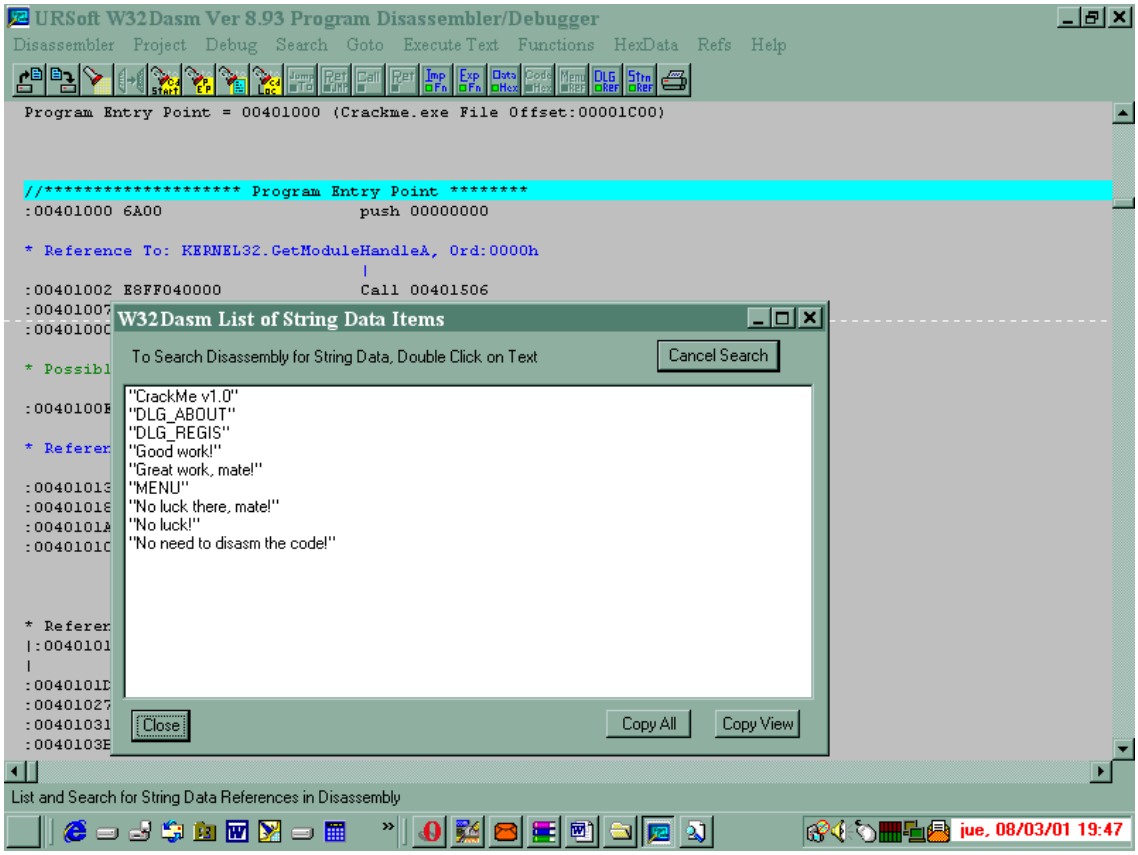
ACCION, ACCION

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 número 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.

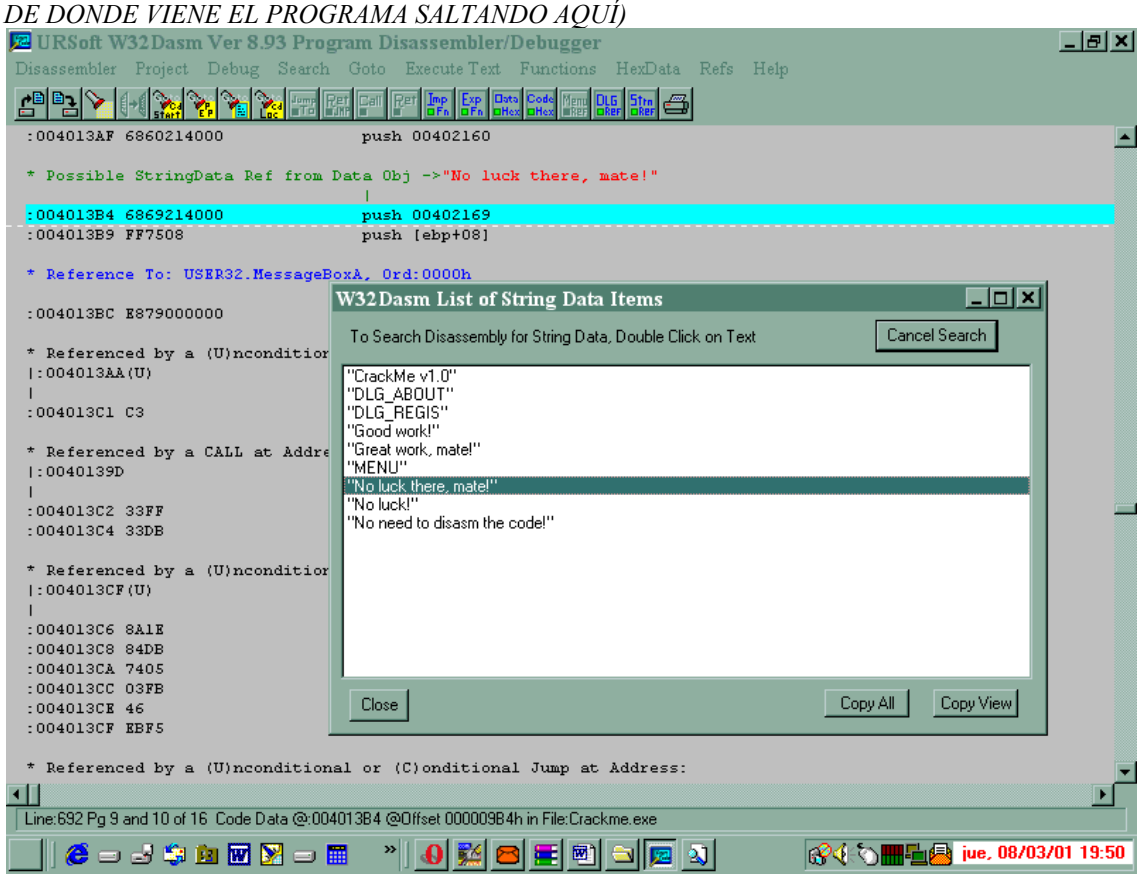
DIBUJO 4: ESTO NOS MUESTRA CUANDO HACEMOS CLIC EN EL ICONO STRN REF (STRING REFERENCES) LA VENTANA DONDE APARECEN LAS CADENAS DE TEXTO QUE USA EL PROGRAMA.




```
REFERENCED BY A CONDICIONAL JUMP or UNCONDICIONAL JUMP AT ADRESS
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.**

AQUI EN ROJO SE VE LA CADENA NO LUCK THERE MATE Y UN POCO MÁS ARRIBA SE VERÁ 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!

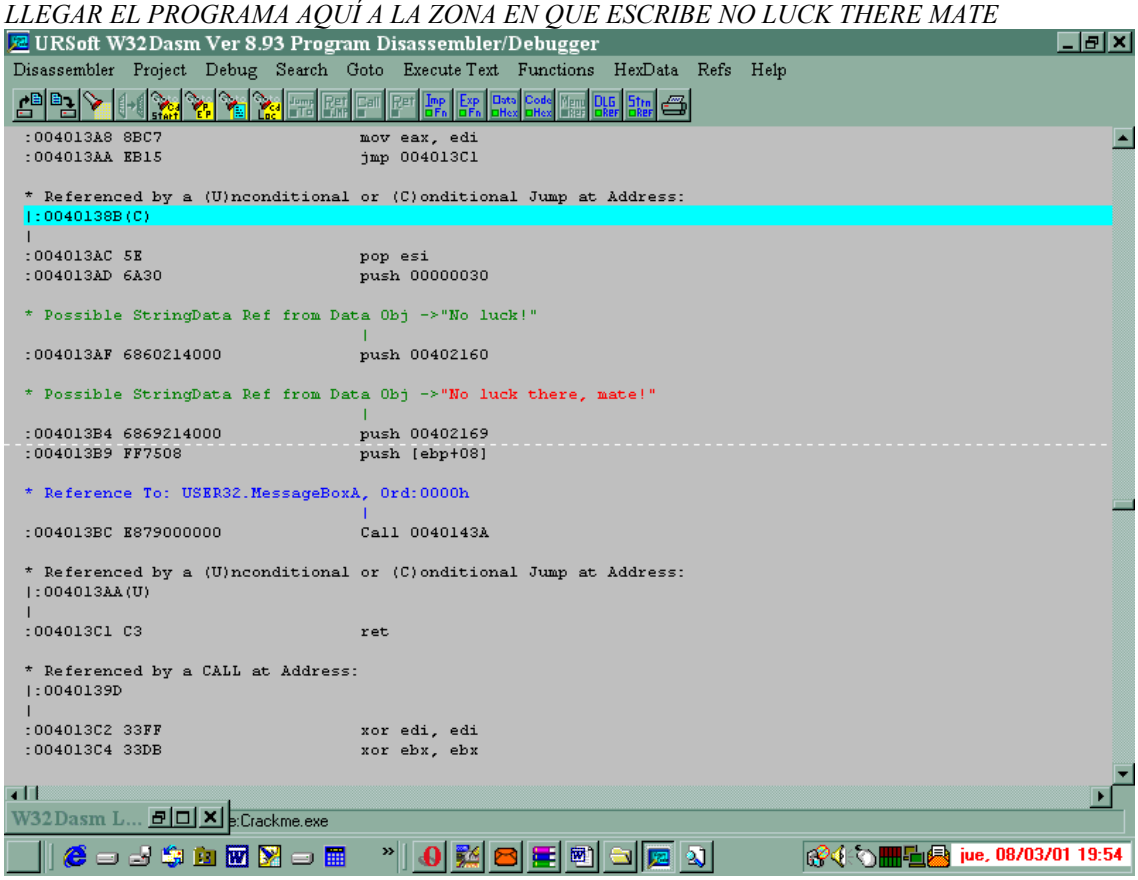
O sea, esta es la parte del programa del CHICO MALO, aca el programa nos tira después de haber comparado y ver que el número que pusimos 989898 no es el correcto y decide tirarnos aca y ponernos el cartel triste de que nuestro número no es valido.

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

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

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

SUBIENDO UN POQUITO (VER DIBUJO SIGUIENTE) 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

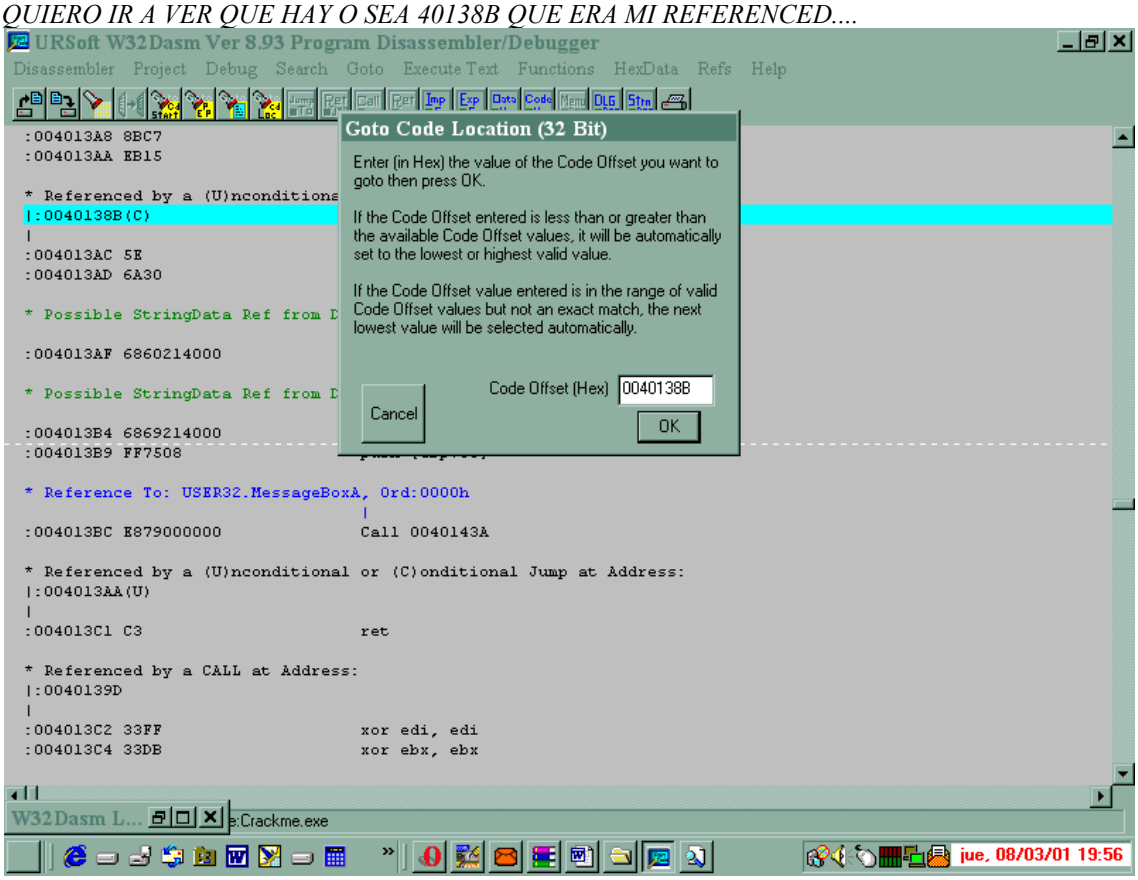


Por lo tanto si desde allí nos tiro a la zona de chico malo, por allí debe estar la comparación de los números de serie el 989898 con el verdadero y después el salto hacia la zona de chico malo o sino siguen en la zona de chico bueno.

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).

Esas son las dos sentencias que más 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....



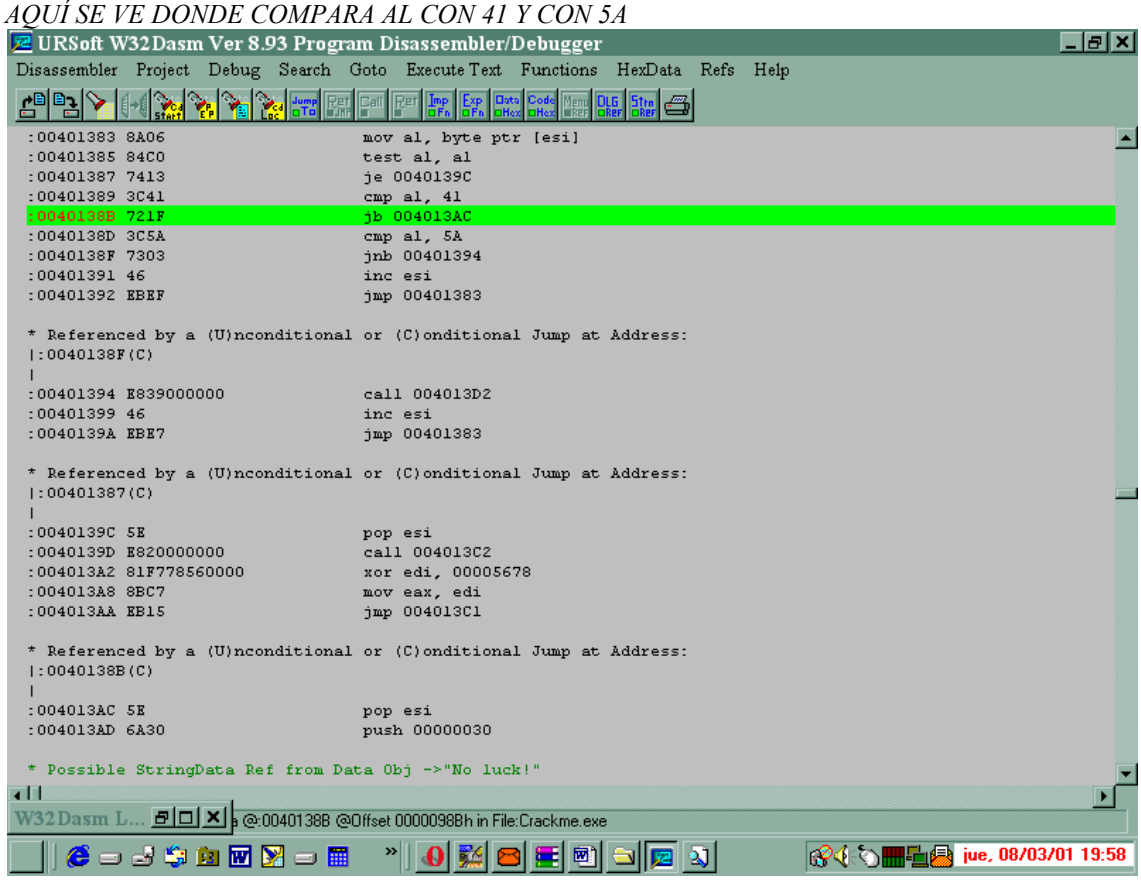
Veamos que encontramos ahí.

```
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 comparacion de la clave porque ahí se ve que compara el valor AL (DONDE PODRIA ESTAR 989898) por ejemplo, con un valor hexadecimal fijo, con el número 41, asi 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)

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



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 más solo en esas dos.

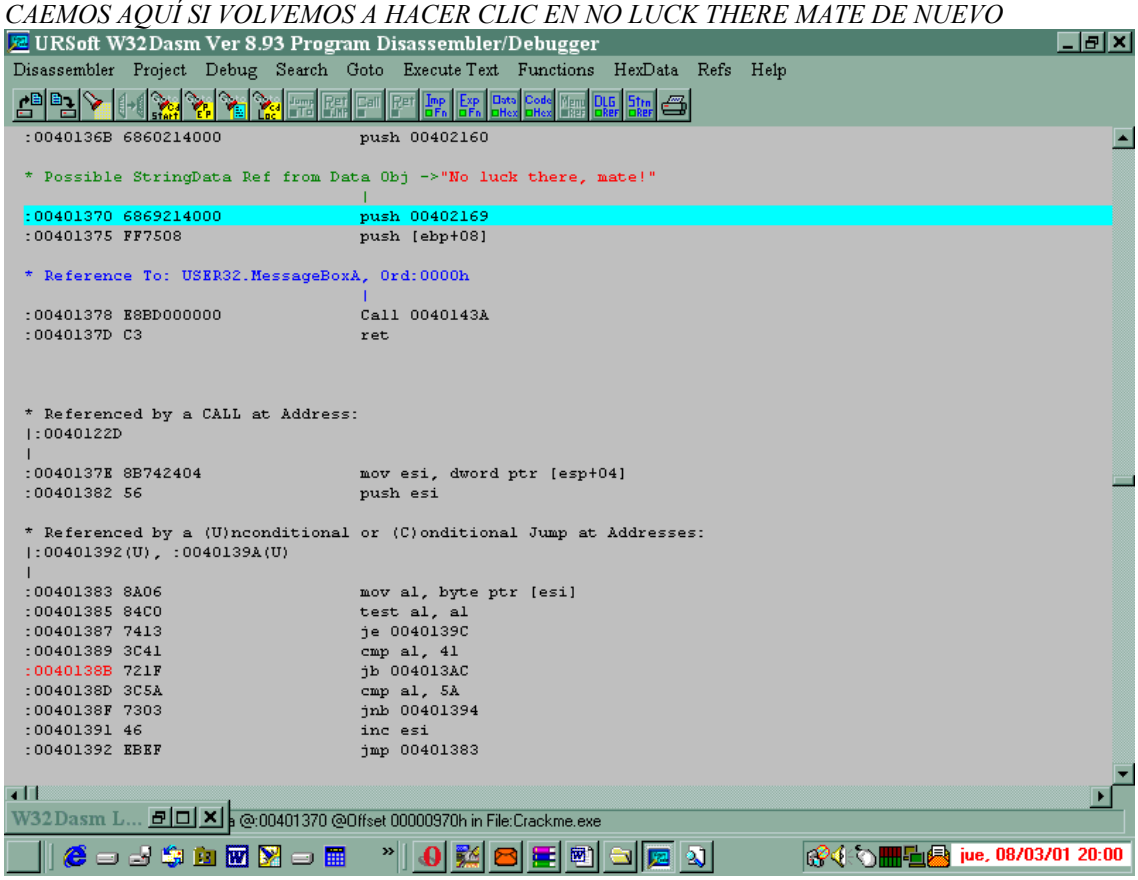
Veamos donde aparecemos

REFERENCED BY A CALL at adress 401245

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

Vamos a GOTO - GOTO CODE LOCATION y tipeamos 401245

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



Lo que aparece es mucho más agradable a la vista que lo que habíamos encontrado antes, aquí hay una comparación de eax y ebx que es más 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)
```

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.

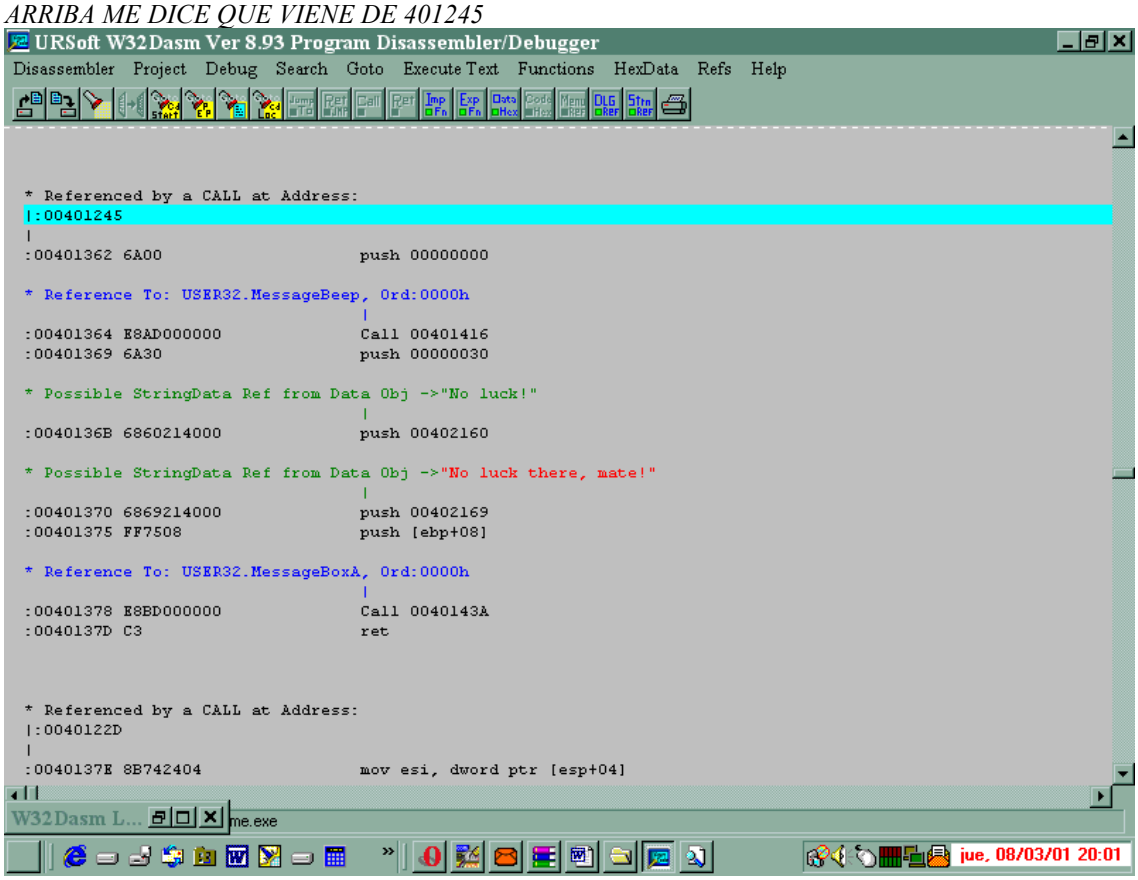
Ya terminamos ya que no podemos usar el softice todavía lo vamos a crackear sin conocer el número de serie.

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.

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



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 NÚMERO DE SERIE.

COMO SE HACE ESO?

FÁCIL.

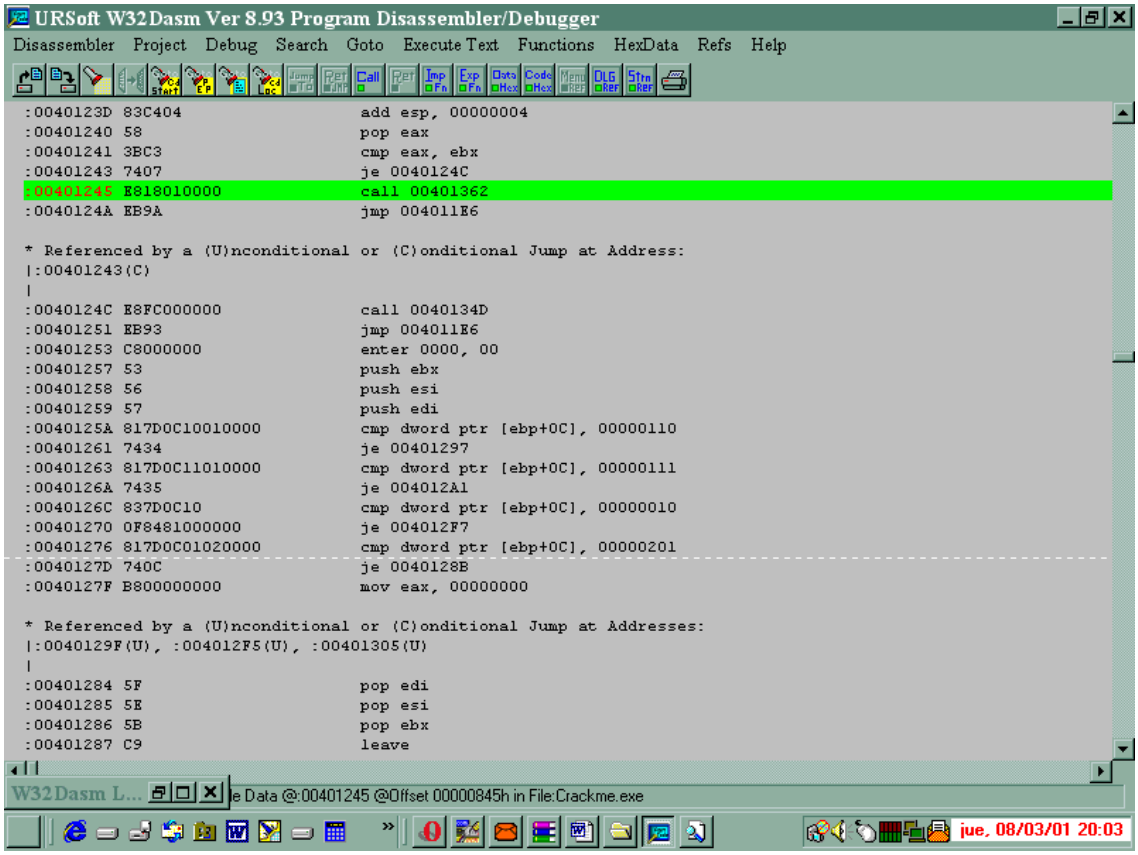
En vez de un salto condicional je usamos una sentencia que salte siempre que es la sentencia jmp.

O sea que tenemos que reemplazar je por jmp y saltaría siempre por encima del call que lleva a NO LUCK THERE MATE y nos registraría siempre sea cual fuere el número que hayamos puesto.

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

COMO SE HACE?

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



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 número 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 NÚMEROS 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 número, pero el nombre debe ser de letras nada más no números, ponemos ACEPTAR y CHACHAN

GREAT WORK MATE, NOW TRY DE NEXT CRACKME

PROGRAMA CRACKEADO Y YA NO SE RIE MÁS

La próxima lección LA BESTIA el SOFTICE.

Si no entendieron algo por favor pregunten en las listas. Si pudieron entender lo que hice no van a tener problemas para ser crackers, porque hay que aplicar esto un poco más un poco menos para crackear.

EL SOFTICE LA PROXIMA SEMANA

POR EL MISMO BATICANAL

Ricardo Narvaja

MAESTRO CIRUELA SIN TITULO

LECCIÓN 02: COMO COMENZAR A USAR EL SOFTICE

Que 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 MÁS 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 esta 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 victimas, 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 esta y que esta 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 esta 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 SOFICE 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 NÚMEROS 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 LECCIÓNES)

Que 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 esta 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 esta ejecutando y eso es porque yo puedo querer cambiar algún número que este más adelante en el programa y que no se este 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 que carácter corresponde el número hexadecimal.

Si alguno no puede ver alguna de las partes que acabo de mencionar, es probablemente por que en el WINICE.dat no esta usando la misma línea INIT que uso yo, asi 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 clickear 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 mi personalmente me cansa la vista un poco esta forma de visualizar pues la letra es más chica, pero gustos son gustos.

Además si queremos convertir un número de hexadecimal a decimal y ver tambien el carácter en modo texto ASCII el comando utilizado es:

? (NÚMERO HEXA)

por ejemplo si hacemos

? 47

El resultado es

00000047 0000000071 "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, más adelante les voy a enseñar como 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 MÁS CUANDO DIGO TECLEAR UN COMANDO SE SOBREENTIENDE QUE HAY QUE TECLEAR ENTER AL FIANALIZAR DE ESCRIBIR EL MISMO), y para que el programa continué 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 heaxdecimales 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, más 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?

Que 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 que pasa y QUE PASA?.

GREAT 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 que número corresponde a JMP 30124c al usar el comando A lo cambio directamente aunque no sepa que número debe reemplazarse y después me fijo que número 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 más 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 la dirección 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 lo 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 está 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 al llegar al salto crítico nos aparece NO JUMP en cambio cuando ponemos un número (EN EL NOMBRE) nos aparece JUMP o sea que nos va a tirar a NO LUCK THRE MATE.

Parcheando el salto con el editor hexadecimal, cambiándolo por 90 90 , o 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 LECCIÓN DE SOFTICE (UN PRIMER PANTALLAZO PARA QUE LO VAYAN USANDO), OBVIAMENTE ES UN PROGRAMA QUE TIENE TANTAS POSIBILIDADES QUE ES MUY DIFÍCIL EN UNA SOLA LECCIÓN 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 NÚMEROS DE SERIE CON SOFTICE.

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

LA PROXIMA PARTE DE LA LECCIÓN DE SOFTICE EL VIERNES QUE VIENE.

AU REVOIR

RICARDO NARVAJA

LECCIÓN 03: COMO UTILIZAR EL SOFTICE CON LAS FUNCIONES DE WINDOWS (API'S)

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 asi, 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 a 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 dobleclick 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 allí podemos ver haciendo...

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

O sea vemos como esta 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 esta a la izquierda del de STRING REFERENCES cinco o seis iconos a la izquierda del de STRING, allí esta 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 GetWindowWord
bpx GetWindowInt
```

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 MÁS ADELANTE EJEMPLOS DE ESTO)

Después está 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 (MÁS ADELANTE VEREMOS EJEMPLOS)

Iremos viendo en sucesivos crackeos que iremos estudiando como se utilizan estas funciones y como aprovechar sus utilizaciones 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 más 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 MÁS ADELANTE PERO IGUAL LO VAMOS A CRACKEAR)

Ejecutamos el programa y cuando arranca vamos a donde esta el SIGNO DE INTERROGACIÓN - ACERCA DE - REGISTRAR y allí 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 allí no para el programa. Entonces hacemos lo siguiente:

Escribimos un número de validación cualquiera por ejemplo 989898 (POR AHORA USEN ESTE NÚMERO) 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 BUUUUUUUUUMP adentro.

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

Allí 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 allí 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 miramos 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 número trucho de serie que yo puse y ¿que puede comparar con el número trucho 989898?... EL NÚMERO DE SERIE VERDADERO o sea que si hacemos ? EAX tenemos el número de serie que corresponde a nuestra maquina y al 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.

ESCRIBAN NO SEAN VAGOS que eso da fuerza para seguir.

Ricardo

LECCIÓN 04: COMO USAR EL RISC PROCESS PATCHER PARA HACER UN CRACK DISTRIBUIBLE

El título 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 como se usa y sirve para TODOS los casos sabiéndolo utilizar.

A que 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 esta 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 LECCIÓNES MÁS 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 este maravilloso programita.

El RISC PROCESS PATCHER es un cargador que no cambia ni un solo número 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 como hacemos para lograr el crack?

Una vez que descomprimimos el RISC PROCESS PATCHER , descomprimimos 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 este 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 selecciónar 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 archivito crack.

Después de la P= pongo la dirección de memoria donde empiezo a 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 alli 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 lo 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 esta 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 NÚMEROS 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.

Cuales 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=cpasos32.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 NÚMERO 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 número 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.

PRUEBEN Y DESPUÉS ME CUENTAN.

RICARDO NARVAJA.

LECCIÓN 05: COMO DESEMPAQUETAR CON PROCDUMP AUTOMÁTICAMENTE

Como hemos visto en lecciónes anteriores y por si no se acuerdan hay programas que no se pueden visualizar con WDASM lo que hace más difícil su crackeo, inclusive si encontramos algún 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 que aquí cuando hablamos de compresión hablamos de ciertos 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 más 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.

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 MESSAGEBOXA 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 cosillas 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 esta el GETTYPE para analizarlo y me fijo como se llama el ejecutable del GETTYPE, hago esto porque el nombre del ejecutable varia 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 este 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 último nombre es WWP32 que es el nombre del compresor utilizado, esto no siempre aparece por eso cuando no sabemos que 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 último de la lista que es el WWP32 y cargamos el ejecutable de C:/ARCHIVOS DE PROGRAMAS/WINOMEGA/WINOMEGA.EXE

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 que 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 más 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 PERO NO CAMBIA EL NÚMERO 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 más 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. **Selecciónar PE-edit.**
3. **Selecciónar el ejecutable desempaquetado winomega.exe.**
4. **Darle a Sections.**
5. **Ante el nuevo apartado text,.rdata,.data,.rsrc,...Cliquear 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 selecciónar 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 estan 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 desempaquetado, esto funciona a veces ya que si vemos el ejecutable original tiene algo más de 1 mega y el desenpaquetado más 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 desempaquetado 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 que 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

LECCIÓN 06: DESCOMPRESION MANUAL CON PROCDUMP (LA LECCIÓN MÁS DURA)

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 si, 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 MÁS 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 esta ejecutando la descompresión. En 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 esta trabajando hasta allá donde esta el programa ya descomprimidito y listo para ejecutarse.

La mayoría de los descompresores usa una rutina JMP común numérica que en los más berretas se encuentra fácilmente, y en los más avanzados se esconde un poco más 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 alli 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 trabajito.

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 HEXADECIMAL 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 esta 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 sera 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 practica 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 esta 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 por que 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 esta 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 lenta 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 esta 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 ASI 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 E0000020 con lo cual se pude 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 MÁS 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 LECCIÓNES MÁS FACILES A PARTIR DE AQUÍ LO DURO YA ESTA PASANDO.

RICARDO NARVAJA

LECCIÓN 07: ALGUNAS COSITAS SOBRE LOS BREAKPOINTS EN PROGRAMAS CON VENCIMIENTO DE TIEMPO Y UN POCO DE AMPLIACIÓN DE CONOCIMIENTO (LECCIÓN SUAVE DESPUES DE LA ANTERIOR QUE FUE DURÍSIMA)

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 esta 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 FUNCION Y VOLVEMOS AL EJECUTABLE EL REGISTRO ESP+04 APUNTA A DONDE ESTA EL VALOR DE LA FECHA.

SI VAMOS AL SALIR EXACTAMENTE A

LEA ECX, [ESP+04]

O ALGUNA SENTENCIA ASI QUE TENGA EL EJECUTABLE VEMOS QUE PASA EL VALOR QUE ESTA 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 TAMBIEN 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 ALLI

D ESP+04 veremos **D0 07 0A 00 10 00**

LOS DOS PRIMEROS SON O7D0 EL AÑO EN HEXA LOS DOS SEGUNDOS SON 00 0A EL MES EN HEXA Y LOS ÚLTIMOS DOS SON 0010 O SEA EL DIA EN HEXA. (siempre se escriben al revés)

SI QUEREMOS REALIZAR EL MISMO TRABAJO EN UN PROGRAMA DE MS-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** o **? 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 NÚMERO DE SERIE para registrar habría que tratar primero de crackear el número de serie y si esto esta muy difícil entonces si, 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 GetWindowWord
bpx GetWindowInt

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 tambien 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.

Ficheros INI

bpx GetPrivateProfileString
bpx GetPrivateProfileInt
bpx WritePrivateProfileString
bpx WritePrivateProfileInt

Estos son para cuando accede a ficheros 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 lecciónes veremos casos y ejemplos de los mismos.

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

Espero que esta lección haya sido un oasis con respecto a la anterior ya volveremos con todas las pilas a partir de la próxima.

RICARDO NARVAJA

LECCIÓN 08: EMPEZANDO A LIDIAR DE A POQUITO CON CLAVES ENCRIPTADAS (PRIMERA APROXIMACIÓN)

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 saltear 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 ADRESS 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 aparezco el 401364 en un RET, lo ejecuto con F10 y ya estoy en el lugar critico.

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 esta 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 número 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

Este 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 más 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 esta la clave falsa al querer mover el primer numerito a AL el SOFTICE para ya que esta 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 esta 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 esta en ESI que es mi clave ENCRIPTADA con lo que esta en EDI que es la VERDADERA CLAVE ENCRIPTADA.

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 ESTA ES LA CLAVE ENCRIPTADA.

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 NÚMERO POR NÚMERO ENTRE LA CLAVE encriptada VERDADERA CON "messing in bytes" OBTENDREMOS LA CLAVE VERDADERA SIN ENCRIPTAR. 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 ENCRIPTADA Y OBTENEMOS LA CLAVE FINAL SIN ENCRIPTAR.

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

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 (CLAVE BUENA)

Si voy al SOFTICE y los paso a ASCII tecleando...

```
? 52=R
? 49=I
? 44=D
? 45=E
? 52=R
```

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

OBTENGO QUE LA CLAVE PARA EL CRACKME2 ES:

RIDERSOFTHESTORM

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

CHAU HASTA LA SEMANA QUE VIENE.

RICARDO NARVAJA

LECCIÓN 09: CRACKEANDO EN DELPHI (ALGO FÁCIL)

Dado que todavía no terminamos con el crack del FONT TWISTER que esta 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 esta el GETTYPE, una vez que esta allí, abrimos una ventana de DOS y nos posicionamos con CD... en el directorio donde esta 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 como 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 lecciónes 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 esta 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 esta 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 esta 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 DISASEMBLE, esto nos va a desensamblar exactamente las sentencias que se ejecutan al clickear 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 las 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 Crea
|                                tor 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 Passwo
|                               rd"
|
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

Y aparecemos aquí

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

ALLI ESTA LA CLAVE PARA NUESTRO NOMBRE Y COMPAÑIA 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 esto abarcamos toda la clave y hago X enter. El programa para cuando comienza la comparación en

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

D ESI y **D EDI** mostrarán 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?... ya veremos.

HASTA LA PRÓXIMA LECCIÓN

Ricardo Narvaja

LECCIÓN 10: COMO CRACKEAR EN VISUAL BASIC

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 que 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 respectivamente.

Hay que saber también que las funciones en VISUAL BASIC no son las mismas que conocemos:

VB breakpoints	**Breakpoints "normales"**
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 más 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 librería que utiliza el VISUAL BASIC 5.

La victima 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 de estos otros links:

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 de el tutorial de EISEL, COMO CRACKEAR EN VISUAL BASIC, sobre todo los que se refieren a la configuración del SMART CHECK.

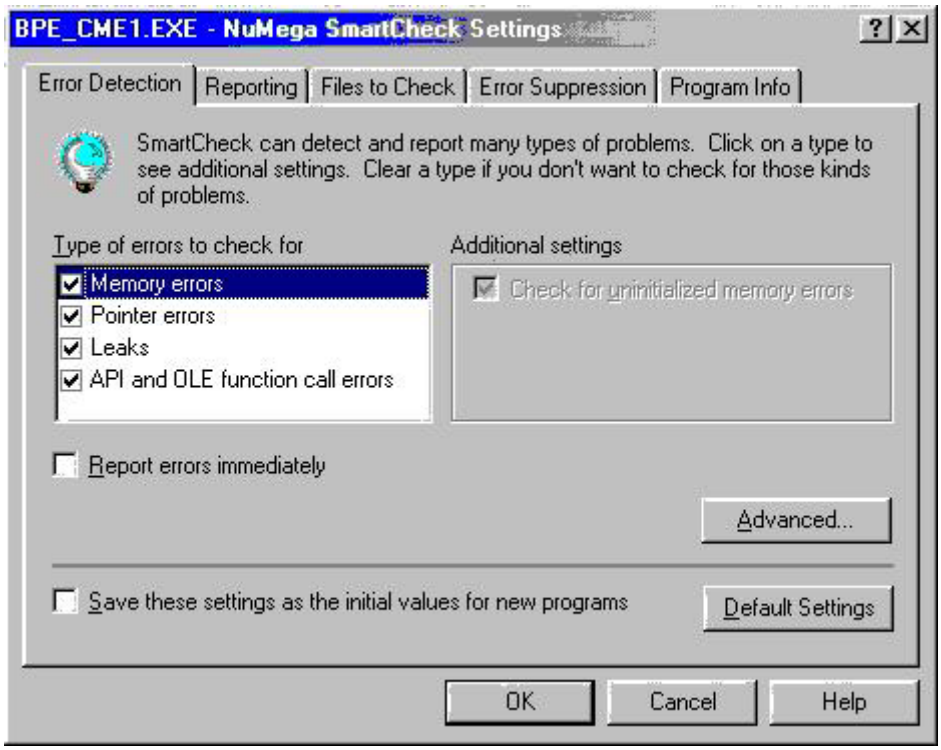
Instalamos el READY CODE y por allí aparece un cartel rojo enorme UNREGISTERED COPY y en HELP-REGISTER esta la ventana para registrarse.

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.**

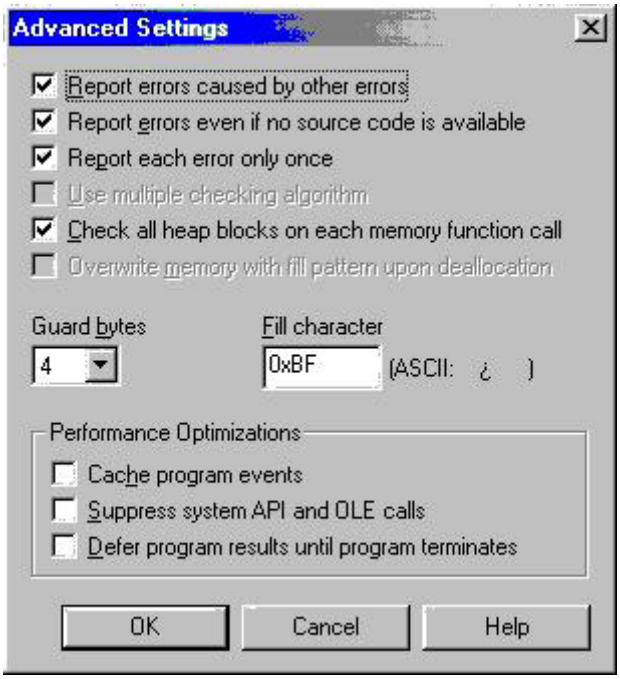
Hay mil formas de crackear este programita ya que su protección es bastante tonta pero nos va a servir para aprender a usar el SMART CHECK.

Instalémoslo y veamos como configurarlo. Vamos al menú SETTINGS y en PROGRAM vemos esta ventana:



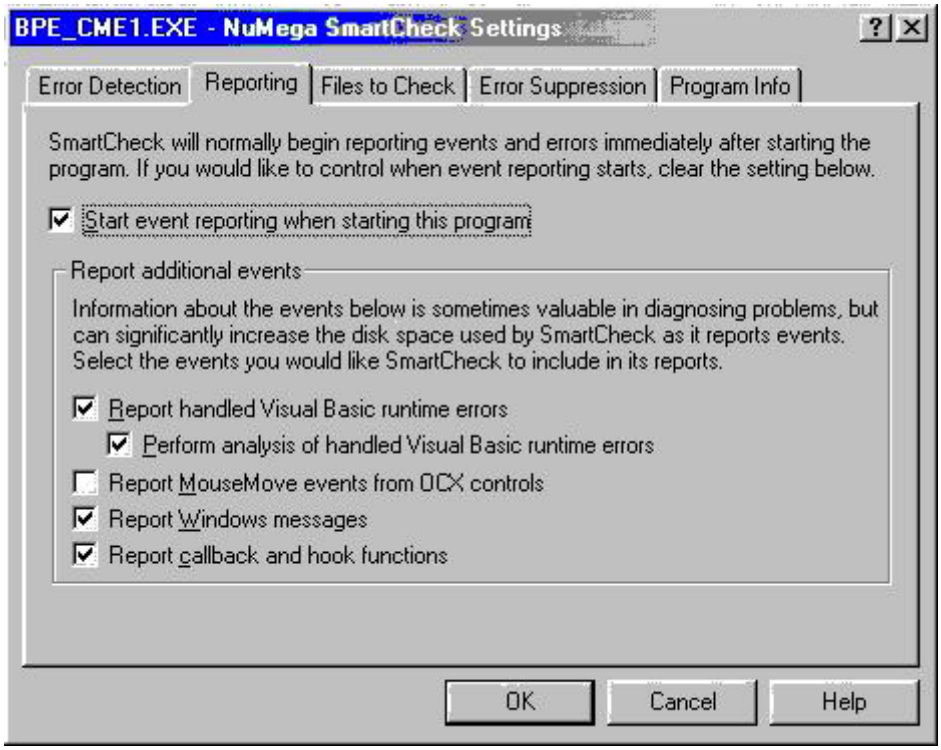
Y la configuramos así como se ve en la imagen, o sea, ponemos la tilde en donde dice TYPE OF ERRORS TO CHECKFOR en las cuatro opciones que tiene y REPORT ERRORS INMEDIATELY que no tenga tilde.

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 último



En la pestaña REPORTING ponemos la tilde en todas las opciones menos en 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 esta 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 y ponemos

NOMBRE: narvaja
REGISTRATION KEY: 989898

Por ahora pongan la misma que nosotros así hacemos lo mismo, aprieten REGISTRO y sale el cartelito INVALID KEY, mientras tanto el SMART CHECK esta tomando nota de lo que el programa esta 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 de unregistered copy. El programa ya esta crackeado.

Todos los programas en VISUAL BASIC, obviamente, no serán tan fácil de crackear como este, 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

LECCIÓN 11: COMO CRACKEAR UN PROGRAMA PROTEGIDO CON ARMADILLO

Que cuernos es ARMADILLO?

Es una protección comercial que ya lleva varias versiones, la del programa que nos ocupa es la versión 1.83.

COMO NOS DAMOS CUENTA QUE ENTRE TANTAS PROTECCIONES QUE HAY ES ARMADILLO LA QUE USA EL PROGRAMA VICTIMA?

Bueno como primer paso, para ver algo con el SOFTICE tendremos que sortear las protecciones ANTI-SOFTICE que tiene el armadillo, estas son tres tipos de protección, se pueden desbloquear a mano, pero es muy laborioso y si tenemos la ultima versión del FROGS-ICE pasara todas las protecciones perfectamente y sin que aparezcan pantallas azules ni nada, como si no estuviera el SOFTICE.

Les recomiendo abiertamente bajar la ultima versión del FROGSICE pues ha mejorado muchísimo e inclusive se puede utilizar el SYMBOL LOADER que en versiones anteriores no se podía.

Pues entonces cargo el FROGSICE 1.085b que hasta hoy creo que es la ultima versión que hay y como por default aparece desactivado entro al menú y pongo ENABLE SOFTICE y listo.

Ahora ejecuto el programa RELAX 1.1 que esta protegido con ARMADILLO 1.83 y arranca perfectamente sin enterarse de que el SOFTICE esta allí. BUENA POR EL FROGSICE.

Ahora aparece una ventana de que el programa va a ser utilizado por siete días y después vencerá y una opción REGISTER donde entro y veo que me pide un número de serie y una clave.

Pongo narvaja y 989898 como siempre y me aparece una messageboxa que dice que no que no que no. Bah que esta mal la clave!!

Voy al SOFTICE y allí pongo un BPX MESSAGEBOXA y vuelvo a la ventana y pongo OK y PUFFF dentro del SOFTICE.

Ahora, si nos habíamos tomado la molestia de mirar el archivo RELAX con el Wdasm vemos que aparecen STRING REFERENCES y ninguna como la que nos dice que la clave no es buena, lo mismo que no hay STRINGS que digan nada de VENCIDO o algo de eso.

Hmmm, pensemos... Volvamos al softice después de haber caído allí por el messageboxa y aprieto F12 para que aparezca el mensaje de error y hago clic en ACEPTAR y de nuevo en el SOFICE, lo primero que vemos es que el nombre el archivo es ARM... lo cual nos da la idea de que aquí hay algo raro ya que el archivo RELAX es el único que se puede ejecutar en el directorio del programa y no hay otro por allí, y sin embargo aquí se realiza la verificación de la clave en un archivo ARM... que encima si busco un poco más arriba puedo ver un CALL, un TEST AL,AL y un salto que pasa por encima del messageboxa, y si invierto el salto me dice que La clave es correcta que la va a GUARDAR, hmmm, y cuando arranca de nuevo el programa esta de nuevo desregistrado.

Bueno, donde esta el BENDITO archivo ese ARM...? Voy a INICIO - BUSCAR y allí pongo arm* y entre otras cosas me aparece un ARM seguido de un número .TMP por ejemplo **ARM1025.TMP**.

Bueno entonces el protector crea un archivo temporal donde ocurre la registración, entonces si quitamos la protección del ARMADILLO, que pasara con el programa RELAX? CHA CHAN CHA CHAN...

Les digo que hay una forma de quitar el armadillo MANUAMENTE pero al existir un programa que lo hace perfectamente y funciona bien, para que nos vamos a matar.

Ese programa es el ARMADILLO KILLER 1, y se consigue en las mejores casa del ramo, hasta la versión 1.83 del armadillo la hace PURE.

Lo abro y me pide que ponga el ejecutable protegido con ARMADILLO, lo busco al RELAX y lo abro, espero un poquito y me dice donde lo quiero guardar, lo guarda.

Ahora lo busco y lo ejecuto, SORPRESA, no aparece más la ventana que dice que es una versión de prueba ni la pantalla para registrarse ni nada, ni tampoco la protección antisoftice , ni la de que el programa te retaba cuando adelantabas el reloj, nada quedo el programa pelado y funcionando y parece que bien.

Adelanto un mes el reloj y sigue funcionando, todo parece estar bien, y el programa crackeado, por ahora cumplimos con nuestro objetivo, si al pasar el tiempo surge alguna novedad , (LO QUE NO CREO PUES TODA LA PROTECCIÓN ESTABA EN EL ARMADILLO) seguiremos con el tema, por ahora.

PROGRAMA CRACKEADO.

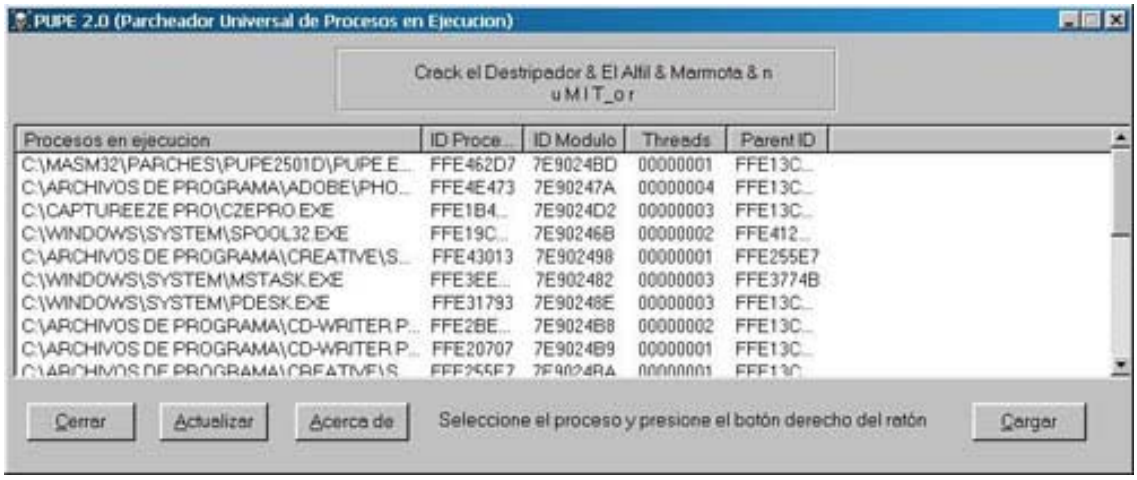
Ricardo Narvaja

LECCIÓN 12: UNA NUEVA HERRAMIENTA QUE NOS PUEDE AHORRAR TIEMPO Y ESFUERZO EN MUCHAS TAREAS AUXILIARES DEL CRACKER: EL PUPE

Como ya avisé en mails anteriores vamos a comenzar a estudiar esta excelente herramienta auxiliar llamada PUPE que fue escrita por crackers a diferencia de muchas de las otras herramientas que podemos encontrar, además esta totalmente en CASTELLANO ya que fue hecha por crackers Hispanos, se puede bajar de la pagina de uno de sus autores CRACK EL DESTRIPADOR, en http://teleline.terra.es/personal/guillet/ sección Herramientas.

Esta herramienta en sus primeras versiones fue un parcheador de la memoria pero le fueron agregando más utilidades y prometen agregarle más todavía con lo que esta herramienta puede llegar a convertirse en una gran ayuda para nosotros, voy a tratar de describir un poco algunas utilidades que pueden llegar a ser muy importantes y que nos pueden ahorrar tiempo y esfuerzo en nuestras tareas de crackear.

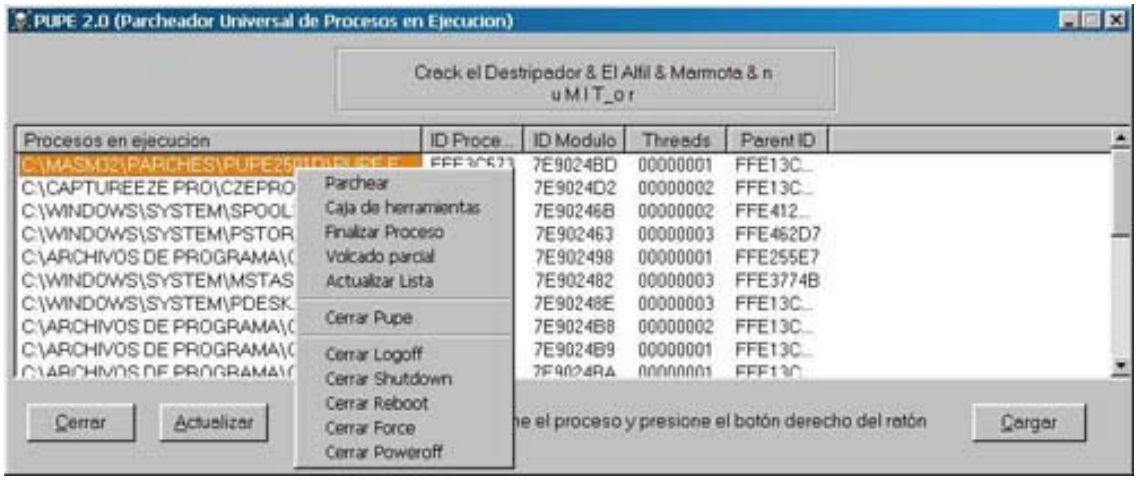
La pantalla que nos aparece apenas abrimos en PUPE es la siguiente:



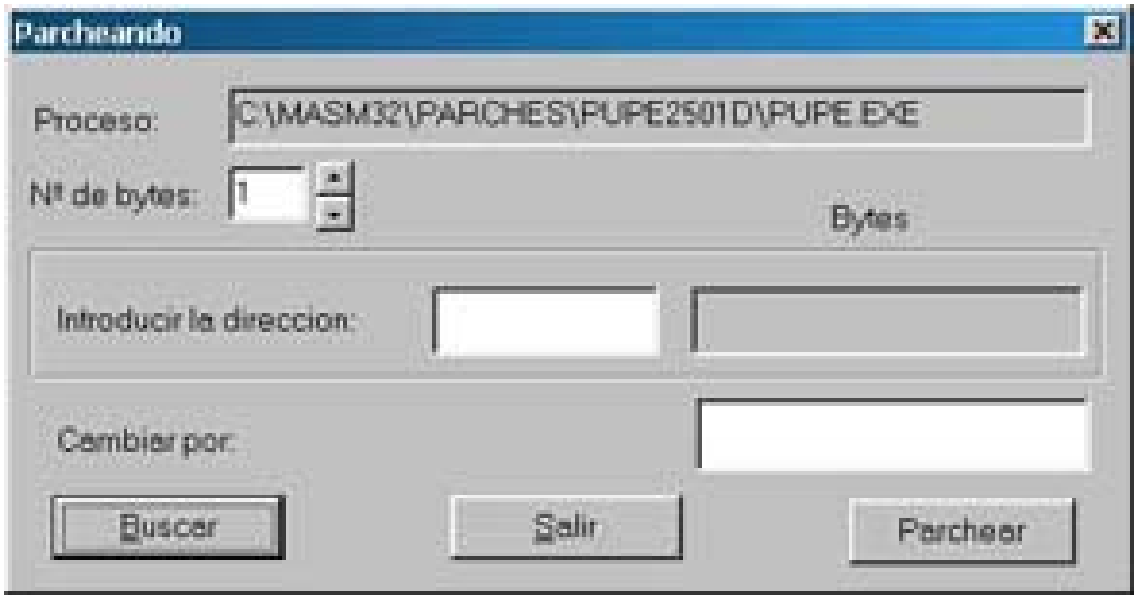
Allí podemos ver los procesos que están ejecutándose en este momento en la computadora, una gran ventaja que tiene PUPE es que no es reconocido por la mayoría de los descompresores (YO DIRIA QUE POR NINGUNO) por lo que pasa completamente desapercibido, inclusive, uno puede arrancar el programa a crackear primero y una vez que ya arranco y no realiza más verificaciones, arrancar el PUPE después con lo que su identificación es prácticamente imposible.

Esto de la identificación del programa es muy importante sobre todo con las últimas versiones de compresores como el ASPACK 2000 y 2001, que cuando arranca el programa tienen triple protección ANTISOFTICE (AUNQUE CON EL ÚLTIMO FROGSICE SON EVITADAS) pero también desactivan el REGMON, FILEMON y cualquier parcheador que existe hasta hoy como nuestro RISC PROCESS PATCHER que no funciona con esos compresores, además como la descompresión y que salga el archivo funcionando es complicadísima, para esos casos puede utilizarse el PUPE para parchear en memoria siendo que es el único que no es detectado por esos compresores.

Una vez que arranca nuestro programa victima, va a aparecer en la lista de procesos que vimos anteriormente, allí seleccionamos el nombre del proceso sobre el cual vamos a trabajar y hacemos clic derecho y allí nos aparecen varias opciones.

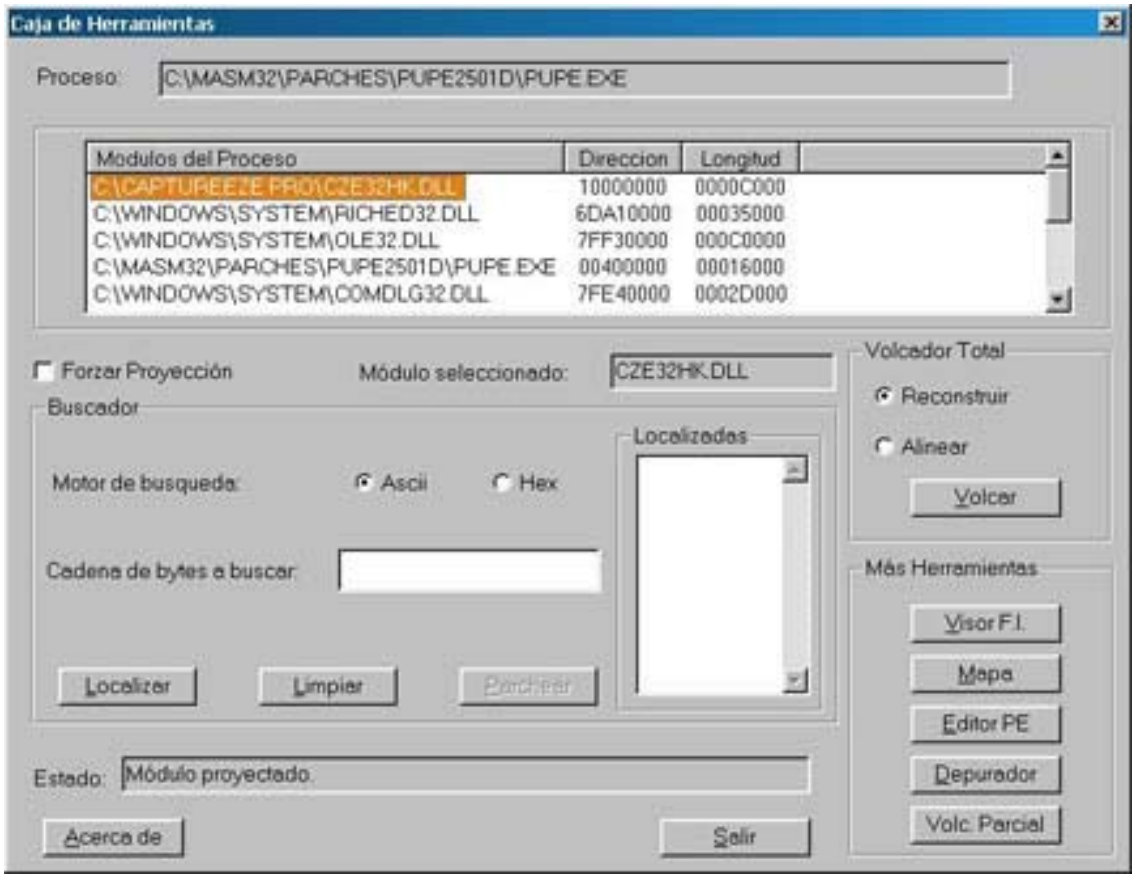


Si queremos utilizar el PUPE como parcheador solamente, elegimos entre estas opciones que aparecen, PARCHEAR y en una ventana de dialogo como la que vemos a continuación, colocamos la dirección que averiguamos con el SOFTICE o el WDASM que necesitamos parchear , cuantos Bits deseamos parchear y el valor de estos bits y listo, serán reemplazados en memoria inmediatamente.



Esta función de parcheador en memoria directamente, puede evitarnos mucho tiempo ya que si vimos algún salto sospechoso en el WDASM y antes de intentar con el SOFTICE poniendo BPX y todo eso aquí podemos probar directamente lo que queremos cambiar y ver si funciona, sin tanta historia, muchas veces necesitamos verificar varios saltos y esto nos puede ahorrar muchísimo tiempo, además que como dijimos en algunos compresores como ASPACK 2000/ 2001 hasta hoy es la única herramienta capaz de hacerlo.

La segunda opción es la opción CAJA DE HERRAMIENTAS. Si elegimos esta opción vemos lo siguiente:



Bueno aquí ya entramos a trabajar con nuestro programa directamente, aparece una lista de todos los módulos que utiliza el programa, y aunque casi siempre elegiremos el ejecutable, el programa nos da la opción de trabajar con cualquiera de los módulos que utiliza el mismo (MUY BUENA OPCION EN CIERTOS CASOS)

Elegimos el nombre del ejecutable o modulo sobre el cual vamos a trabajar y lo ingresamos haciendo clic derecho hasta que aparece el nombre en donde dice MODULO SELECCIÓNADO.

Ahora tenemos aquí varias herramientas que pueden ser de mucho interés.

Hasta ahora para descomprimir un ejecutable era muchas veces bastante complicado según el compresor que sea, aquí podemos obtener un ejecutable descomprimido que aunque puede no funcionar, nos permite rápidamente poder visualizar con el WDASM las STRING REFERENCES sin tener que perder horas descomprimiendo a mano.

Vemos a la derecha la opción VOLCADOR TOTAL y tenemos dos alternativas, podemos probar cual de las dos es la que funcionara pero si es un compresor moderno, usaremos la segunda opción, podemos probar pues tarda un segundo en realizar la tarea.

Una vez que tenemos el ejecutable nos queda solamente verificar con el PROCDUMP (O CON EL MISMO PUPE SEGÚN VEREMOS MÁS ADELANTE) si hay que cambiar la protección ANTIDESENSAMBLADO de C000040 u otro valor a E0000020 como vimos en las lecciónes de descompresión y listo, un ejecutable para poder ver con el WDASM en menos que canta un gallo, y sin rompernos mucho la cabeza.

Esta opción de VOLCADOR TOTAL funciona con CUALQUIER COMPRESOR QUE EXISTE y el resultado es un archivo para ser analizado con el WDASM, muchas veces el ejecutable que descomprimimos así funciona perfectamente, otras hay que trabajar un poco para que funcione, pero, es muy útil listar en el WDASM las STRING REFERENCES de un programa comprimido en segundos, y además poder parchearlo aquí mismo.

Si cuando selecciónamos el módulo a utilizar nos dice que es muy largo entonces hay que colocar la tilde en FORZAR PROYECCIÓN para poder trabajar sobre el mismo.

Otra opción muy importante en la caja de herramientas es la de MOTOR DE BÚSQUEDA, la cual permite buscar en la memoria cadenas de texto, valores hexa o cadenas de valores hexa, esto nos puede ayudar en el crackeo en ciertas ocasiones.

Pongámosle un programa que tiene quince ejecuciones o que vence en 30 días y que cuando comienza nos dice cuantos días nos quedan o cuantas veces nos quedan para utilizar, también podemos usarlo en juegos en los cuales tenemos cierta cantidad de vidas o municiones o avioncitos que se yo, pongámosle cualquier caso de esos es lo mismo.

Pongamos el caso de que nos queden quince usos de un programa, buscamos el valor hexa de quince o sea F, colocamos que busque F dentro del programa, por supuesto nos saldrán muchísimos resultados, ahora copiamos los valores de memoria que nos salen y utilizamos una vez más el programa para que nos diga que quedan 14 usos, ahora copiamos los resultados para catorce (E) y vemos que ya las posiciones de memoria que salen en ambas búsquedas son menores y seguimos el mismo proceso hasta que descartamos y nos queda solo la posición de memoria donde el programa guarda los días, o las balas que utiliza el jueguito, o las veces que nos quedan utilizar, etc.

Una vez que ya sabemos la posición de memoria donde se guarda ese dato, podemos interrumpir el jueguito y cuando se nos están acabando las balas o nos queda una sola vida, utilizamos la opción parchear y colocamos el valor de la cantidad de balas que queremos o de vidas, o de días, etc, y lo modificamos desde el PUPE también.

Podemos también encontrar cadenas de texto y reemplazar desde aquí directamente, con el parcheador, las posibilidades son ilimitadas.

Después otra utilidad muy importante de la caja de herramientas es el VISOR DE FUNCIONES IMPORTADAS, lo que en el WDASM se conoce como IMP FN, ya que muchos programas al descomprimirse muestran las SRTINGS REFERENCES pero no salen la lista de Funciones Importadas, aquí podemos ver cuales son las que utiliza el mismo sin hacernos muchos problemas.

La sección MAPA nos permite realizar un volcado a un archivo de la sección de memoria que queremos para analizarlo con tranquilidad.

La sección EDITOR PE nos permite hacer el mismo trabajo que realizamos con el Editor Pe del PROCDUMP (O SEA CAMBIAR LA PROTECCIÓN ANTIDESENSAMBLADO DESDE AQUÍ MISMO SIN ABRIR OTRO PROGRAMA) pero agregándole la posibilidad de volcar las secciones que queremos a un archivo. (ESTO PUEDE SER MUY UTIL YA LO VAMOS A VER)

En la sección DEPURADOR podemos hacer un desensamblado del programa directamente desde la memoria, MUUUY UTIL, en HEXA o en lenguaje ensamblador como el WDASM y verlo aquí o volcarlo a un archivo, además que aquí también tenemos botones para acceder al parcheador y al buscador en la memoria, además tenemos la opción de realizar un volcado parcial de alguna parte que queramos, como vemos es una linda herramienta que comenzamos a ver hoy y que en las sucesivas lecciónes nos va a ayudar muchas veces en el trabajo de crackeo, y vamos a ver ejemplos prácticos de cómo utilizarla y aprovecharla en nuestro beneficio ya que para eso fue hecha por crackers.

EN LAS PROXIMAS LECCIÓNES LA UTILIZAREMOS EN NUESTRO PROVECHO CON EJEMPLOS PRACTICOS DE SU UTILIDAD, BYE BYE, HASTA LA LECCIÓN 13.

LECCIÓN 13: COMO CRACKEAR EL MORPHINK 99 TRIAL, UN PROGRAMA PROTEGIDO CON VBOX

Un amigo y listero me pidió que lo ayudara con el MORPHINK 99 Trial (GRANDE VESPERS) que es un programa que vence a los 15 días de utilizarlo, y me dijo que estaba muy protegido, lo cual es totalmente cierto, es CASI una fortaleza, salvo un pequeño detalle que abrió la puerta a la desprotección.

Ya cuando lo abrí con el WDASM empezaron las sorpresas, no hay STRING REFERENCES, no esta hecho en VISUAL BASIC ni en DELPHI, si le aplico un identificador para ver si esta comprimido, como el LANGUAGE 2000 o el FILE ANALIZER no identifica nada.

Sin embargo el programa se desensambla con el WDASM no es como esos programas comprimidos en los que no aparece nada y termina rápidamente no aquí trabaja y desensambla bien.

Bueno aquí comienza la desesperación pero a no asustarse hasta un novato como nosotros puede si insiste encontrarle la vuelta al programa y una vuelta fácil de entender, ya que para mi los tutoriales sobre el tema son bastante complejos, entonces busco la solución fácil (SI LA HAY), y aquí la hay.

Bueno las STRING REFERENCES no nos dicen nada y las IMPORTED FUNTIONS me dicen VBOXP401, que no me dice mucho salvo que creo haber visto por ahí algún tutorial sobre VBOX pero no me acuerdo donde.

Ah ya lo encontré, busco en los tutoriales que tengo y veo que es un sistema de protección comercial (COMO EL ARMADILLO) pero este se basa en unos DLL que copia en la carpeta WINDOWS/SYSTEM que hacen el trabajo sucio de contar los días, de poner esa pantalla molesta que dice los días que faltan para que venza, (SON QUINCE EN TOTAL) y la protección ANTISOFTICE.

Allí hay tutoriales con sesudas explicaciones de cómo parchear en memoria esta porquería, y como descomprimirlo, pero a decir verdad a mi no me sirvió ninguna, no se si será porque no es igual para todos los archivos la protección Vbox pero después de descomprimir el archivo según esos tutoriales no me funciona a pesar de haber arreglado el ENTRY POINT y todo eso, por ahí me equivoque yo, casi seguro, pero ya que halle otra forma más fácil por lo menos para este programa, bueno utilizare esta, y dejare la otra para los bochos de la descompresión.

La protección ANTISOFTICE tenemos que esquivarla y en esto agradezco nuevamente a VESPERS que me paso el datito.

Existe una utilidad para limpiar esta protección comercial que se llama VBOX Unwrapper [uCF] by MAK & EinZtein in 2000.

Pero para mi hay malas noticias This Unwrapper will only run on Windows 2000!!!!.

O sea que solo funciona en WINDOWS 2000, ay, a quien se le ocurre hacer una herramienta solo para WIN2000 y que no funciona en WIN98?, y es así la probé y se cuelga en WIN98.

Bueno lo que si tenemos es una gran herramienta como es el nuevo PUPE que fue programado por crackers y es evidente que es de gran utilidad, por lo menos para mi, grande MARMOTA, CRACK EL DESTRIPADOR Y COMPANIA sin PUPE no se que habría hecho.

Arranco el programa y espero que aparezca la ventana que dice cuantos días me quedan, (QUE HORROR), antes de poner TRY o cualquier otra cosa arranco el PUPE, y allí elijo el proceso del MORPHINK y allí entro a la CAJA DE HERRAMIENTAS, y elijo el modulo que según los tutoriales es autor de todo el desastre, **VBOXT401.DLL**. Elijo VOLCADO TOTAL y alinear y lo guardo.

Lo abro con el WDASM y me aparece perfecto con FUNCIONES IMPORTADAS y STRING REFERENCES, lo que no seria posible si lo buscara en el rígido y lo abriera con el WDASM ya que esta comprimido.

Para saltear la protección ANTI SOFTICE veo la parte siguiente...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:07006F44(C)

:07006F4B   6A00            push 00000000

* Possible Reference to Dialog: DialogID_00D1, CONTROL_ID:00FF, ""

:07006F4D   6AFF            push FFFFFFFF
:07006F4F   683B9D0000      push 00009D3B
:07006F54   8D8D34FEFFFF    lea ecx, dword ptr [ebp+FFFFFE34]
:07006F5A   E8E1220100      call 07019240
:07006F5F   8945BC          mov dword ptr [ebp-44], eax
:07006F62   8D458C          lea eax, dword ptr [ebp-74]
```

Ese CALL me devuelve un cero si no esta el SOFTICE y me devuelve un uno cuando esta presente, así que hay que poner un **BPMB 7006F5F** justo cuando después de pasar el CALL y allí poner **r eax=0**. Entonces pasa la protección.

Miramos las STRING REFERENCES del dll maldito. Allí en las STRING REFERENCES veo algo que me llama la atención; dos cadenas **UNLIMITED TRIAL** y **TRIAL EXPIRED**, hmmm sospechoso, y hay un salto en **70117A2** que me tira a **TRIAL EXPIRED** y si no salta va a **UNLIMITED TRIAL**, mmm que lindo esta esto.

```
:0701179f   test  al,10
:070117a1   push  ebx
:070117A2   je    70117be     (SI SALTA VA A TRIAL EXPIRED)

Posible Reference to String Resource "UNLIMITED TRIAL"

:070117a4   push  00001478
```

O sea que testea AL con 10 y allí decide si seguir el camino 'TRIAL EXPIRED' o 'UNLIMITED TRIAL'.

Abro el SYMBOL LOADER y allí una vez que cargo el ejecutable y para cuando se inicia le pongo un **BPMB 701179f x**, un poquito antes del salto.

Pongo BREAKPOINTS DE EJECUCIÓN en memoria en vez de BPX porque el programa detecta si pones BPX y te dice que tienes un debugger activo y no funciona.

Una vez que para allí, ya que compara AL con 10 y según eso salta o no y ya que AL no es igual a 10 lo que va a hacer que vaya por la zona de TRIAL EXPIRED CHICO MALO, entonces modificamos AL con **R AL=10**, y ahora en vez de saltar es como si hubiéramos invertido el salto, va a UNLIMITED TRIAL, saco todos los BREAKS con **BC*** y corro el programa con X o F5 o CTRL+D.

La primera vez la pantalla que dice los días que faltan aparece pero ya no dice los días, esa zona aparece negra. Paso la protección ANTI DEBUGGER como vimos antes, cierro el programa y lo vuelvo a abrir y ya no aparece más la NAG asquerosa esa y arranca directamente el MORPHINK 99 sin ningún problema. Adelanto el reloj una año y sigue arrancando perfectamente.

LECCIÓN 13: SEGUNDA PARTE (COMO DESPROTEGER COMPLETAMENTE AL MORPHINK)

Una vez que reseteé la máquina el programa se volvió a desregistrar.

La verdad, que el programa me haya engañado ya me había subido la presión, así que decidí no parar hasta no limpiarlo completamente y dejarlo como un archivo suelto sin necesidad de dlls Vbox ni nada.

Lo primero que me llamo la atención en todo este proceso es lo siguiente, porque a mi el MORPHINK no me detecta el SOFTICE y a mi amigo VESPERS si, dado que los dos tenemos la misma versión del programa bajada del mismo lugar y el mismo softice 4.05, porque a el si esta el SOFTICE cargado le aparece el mensaje de error y no le funcionaba el MORPHINK y a mi no me aparece?

La verdad que yo me confundí con respecto a esto, como hice tantas cosas no sabia si exactamente era algo que había hecho yo sin querer o porque sucedía.

Cuando el programa se desregistro, tuve una segunda oportunidad de ver bien que pasaba y la situación en mi maquina después de muchísimas pruebas es la siguiente:

Si arranco el programa con el SOFTICE cargado , me arranca SIEMPRE el MORPHINK y no lo detecta.

Si pongo BPXs ahí si recién me sale el cartelito de error.

Mientras me mantenga poniendo BPMs y BPR el programa arranca normalmente.

Al revés si utilizo el FROGSICE me sale el mensaje de error. Lo mismo que si en el CALL que puse en la primera parte de la lección para saltear el SOFTICE pongo r eax=0 me sale el mensaje de error y si dejo el valor de EAX=22 que me aparece solo, arranca normalmente el programa.

Ahora cual es la diferencia entre mi softice y el de VESPERS?

Después caí en la cuenta que yo había modificado el ejecutable del SOFTICE con un ocultador llamado BACKDOOR KEEPER que MODIFICA y OCULTA según el autor, algunos puntos débiles que el FROGSICE no puede tapar.

Por eso el programa a mi no me detecta el Softice directamente y a VESPERS si, de cualquier manera a la persona que le sirva la forma de ocultar el SOFTICE que puse en la primera parte que la use, funciona para el que no modifico el SOFTICE con BACKDOOR KEEPER.

Ahora el BACKDOOR KEEPER es la primera vez que me tapa la detección del SOFTICE ya que siempre lo había podido hacer con el FROGSICE, y bueno ahí esta la diferencia ya que el BACKDOOR KEEPER MODIFICA el ejecutable del SOFTICE para SIEMPRE y como yo no me acordaba que lo había hecho, no sabia que pasaba, igual solo sirve para algunos pocos casos (COMO ESTE POR EJEMPLO)

Bueno al grano. Lo primero que hice fue arrancar el programa MORPHINK hasta que pase la pantalla de PRUEBA y llegue a arrancar completamente.

Ahora probé que si el programa esta arrancado y vuelvo a arrancarlo sin haberlo cerrado antes, esta segunda vez arranca sin poner la pantalla de TRY o sea de probar, arranca directamente.

Fijándome en las funciones importadas del archivo VBOXP410.dll encuentro **BPX GETPROCESSADDRESS.** Entro en el SOFTICE y pongo **BPX GETPROCESSADDRESS,** siempre con el programa abierto y trabajando sobre una segunda apertura simultanea del mismo, o sea siempre dejo minimizado el programa ya abierto y trabajo ahora como si no estuviera abierto, ejecutándolo por segunda vez.

Una vez que empieza a arrancar después de un rato para, me fijo abajo a la derecha que esta parada corresponda al proceso MORPHINK y una vez que aparece allí MORPHINK (OJO NO EN LA LINEA VERDE SINO BIEN ABAJO A LA DERECHA), entonces borro todo con **BC*** y vuelvo con F12 hasta que aparezco en VBOXP410.

Allí veo que el modulo VBOXP410 se ejecuta en direcciones superiores a 5.000.000, lo que puedo ver también con el WDASM, y el programa MORPHINK se ejecuta en direcciones apenas superiores a 400.000, esto lo comprobé también cuando al tener arrancado del todo el programa veo que al poner un BPX hmemcpy y tocar cualquier cosa y entrar en el SOFTICE el programa tiene direcciones apenas superiores a cuatrocientos mil, mientras que el VBOX se ejecuta por encima de los cinco millones.

Entonces volviendo al punto en que estoy en el softice dentro del VBOXP410 supongo que en algún momento el archivo este va a dejar de trabajar y pasar el control al programa MORPHINK.

Por eso de aquí hago como en la lección de descompresión manual y tecleo a ojo sin calcular valores

BPR 400000 500000 r if (eip>=400000) && (eip<=500000)

para que el programa pare en la primera sentencia del MORPHINK que debe estar a ojo entre esos valores 400000 y 500000.

Pongo a correr el programa y para en **43E1C0** que es nuestro nuevo ENTRY POINT.

Ahora borro todo con **BC*** y hago un loop infinito allí para que el programa quede trabajando siempre en la misma sentencia

A 43E1C0 JMP 43E1C0

y vuelvo con X a WINDOWS.

Ahora la única diferencia con el método que usamos en la lección de descompresión manual es que aquí no funciona el PROCDUMP, así que yo abrí el PUPE y elegí el proceso MORPHINK y el ejecutable Morphink.exe e hice un volcado total con la opción ALINEAR activada. Lo guarde en el escritorio con el nombre MOCO.exe.

Ahora con el ULTRAEDIT modifico los valores que tuve que cambiar para lograr el LOOP INFINITO o sea busco

EB FE EC 6A FF 68 D0

Y VUELVO A COLOCAR COMO ERA ORIGINALMENTE EN VEZ DE **EB FE**, LOS VALORES **55 8B**, QUE COPIE ANTES DE MODIFICAR PARA HACER EL LOOP INFINITO.

UNA VEZ QUE HICIMOS ESTO SOLO NOS QUEDA MODIFICAR EL ENTRY POINT AL NUEVO VALOR, Y ESO LO PODEMOS HACER COM PUPE O CON PROCDUMP, O SEA SI USO PROCDUMP POR EJEMPLO EL NUEVO ENTRY POINT DEBE SER **003E1C0** YA QUE LE RESTE LOS 400000 DE LA image base.

Ahora tengo el archivo moco sobre el escritorio, cierro todas las instancias de MORPHINK, reseteo la maquina para que no me vuelva a engañar, lo ejecuto allí mismo en el escritorio y ABRE PERFECTAMENTE Y FUNCIONA.

Otra protección que pudimos vencer no sin bastante trabajo.

Gracias a VESPERS que cuando yo me equivoque me ayudo, la verdad que ya es un cracker a todo trapo y estoy contento de que aunque sea yo con mis lecciónes haya colaborado un poquito con algo de todo lo que aprendió.

Otra cosa que quiero destacar es la utilidad del PUPE, la verdad una muy linda herramienta de inapreciable valor.

Ricardo Narvaja

LECCIÓN 14: UNA LECCIÓN PARA RELAJARSE Y CRACKEAR ALGO FÁCIL (BIKERS LOG 3.5)

Ya que las ultimas lecciónes fueron bastante difíciles y dado que llegan las fiestas vamos a crackear algo sencillo, a pesar de que es un programa nuevo, en su ultima versión la 3.5 , para que recuperemos la fe en que todos los programas no son armatostes complicados en DELPHI, con protecciones VBOX, ARMADILLO, ASPACK y todas esas porquerías que cuestan un PERU crackear.

Este es un programa refácil de crackear que nos hará recordar los buenos viejos tiempos de cracks fáciles, tiene protección ANTI-SOFTICE pero con el Frogsice se pasa, es refácil, sencillón, para las fiestas, con un pan dulce y una sidra en la mano, y la cabeza tranquila sin rompérsela por nada.

Voy a subir el BIKERS LOG 3.5 al FREEDRIVE que es un programa para los que les gusta andar en bicicleta y se pueden anotar todos los datos que necesitan los ciclistas obviamente, como yo no soy ciclista, no lo uso, pero es un placer crackear algo así.

Además si alguien quiere experimentar con algo sencillo de crackear les recomiendo que no lean el tutorial y se larguen sin miedo, es fácil.

Lo único un poquito más complicado es que esta comprimido y no se ve nada con el WDASM pero si lo agarran con el PUPE y cuando ya arrancó el programa le hacen un VOLCADO TOTAL, con la opción ALINEAR van a poder desensamblarlo con el WDASM y ver las STRING REFERENCES.

Bueno vamos a UNLOCK y ahí ponemos mi clave preferida 989898 y una vez que la escribimos, entramos al SOFTICE con CTRL+D y tecleamos **BPX HMEMCPY** y con CTRL+D volvemos a la ventana para escribir la clave.

Esperamos unos segundos para ver si no vuelve a entrar solo en el SOFTICE antes de apretar el BOTON OK de la ventana de REGISTRO (YA SABEMOS QUE SI ENTRA SOLO AL SOFTICE ANTES DE APRETAR OK TENEMOS QUE SALIR Y CERRAR LOS PROGRAMAS QUE ESTAN FUNCIONANDO CON CTRL+ALT+DEL y FINALIZAR TAREA MENOS EL EXPLORER Y EL BIKERS LOG OBVIO)

Una vez que apretamos OK entra en el SOFTICE por la acción del BPX HMEMCPY, aprieto F12 hasta volver al ejecutable BIKER (son siete veces que hay que apretar F12) y una vez que veo BIKER en al línea verde del SOFTICE empiezo con F10 a recorrer el ejecutable saliendo de esos RET y cuando vemos alguna sentencia que transfiere algún valor a algún registro, EAX, EDX, ECX o alguno de esos, miramos con **D EAX** o el registro que sea y seguimos traceando y mirando hasta que llegamos a **6de41e** y **6de421** donde se transfiere a EDX y EAX mi clave falsa y la clave buena.

Si hacemos justo después de ejecutar esas dos instrucciones **D EAX** y **D EDX**, vamos a ver la clave verdadera que es **977-0956.....**, se las dejo para que la averigüen ustedes, ja ja.

FELICES FIESTAS PARA TODOS
RICARDO NARVAJA

LECCIÓN 15: UN NUEVO WINOMEGA (5.15.13)

A veces cuando sale una nueva versión de un programa que ya crackee es lindo ver los cambios que se han realizado en la protección que uno ya conoce y comparar si ha mejorado en ese sentido.

Este es el caso del nuevo WINOMEGA 5.15 (el anterior era 5.12) que se puede bajar de www.winomega.com

Una vez desinstalada la versión anterior para que no tome registros de antes, procedemos a mirar la nueva, parece bastante similar aunque tiene una decoración NAVIDEÑA por las fiestas y un poquito mejorada la interfase, ahora vamos a lo nuestro.

En la versión anterior el ejecutable lo descomprimimos con PROCDUMP utilizando WWPACK32 II y ahora vemos con el WDASM que esta comprimido también, probemos con el mismo descompresor a ver que pasa y VOILLAAA, funcionaaaaa.

Hasta aquí todo igual, pero no es todo igual, reemplazo el ejecutable descomprimido y arranca el programa, protección ANTISOFTICE no tiene, esperaremos la próxima versión para esto.

Vamos a AYUDA - REGISTRARSE y ponemos cualquier número por ejemplo 99999999999 y nos sale PIN INCORRECTO, que parecido que es hasta aquí..

Bueno arranco el SOFTICE y pongo BOX MESSAGEBOXA y pongo un número falso y no para allí o sea que no usa MESSAGEBOXA, busco en el WDASM la STRING PIN INCORRECTO y no aparece, hmmmm, esto se comienza a complicar.

Pongo BPX HMEMCPY y trato de volver al ejecutable a ver que pasa, da mil vueltas y no se llega a nada, es un despiole, parece que en algo mejoro, han cerrado algunas puertas un poco mejor.

Seria muy piola decir fui aquí y hice esto, pero la verdad es que tuve que intentar bastante para abrir esta puerta esta vez, prueba que te prueba, a veces los crackers ganamos por cansancio o nos ganan a veces por cansancio a nosotros.

Bueno después de muchos intentos infructuosos, me fui a abrir con el DEDE para ver si el ejecutable descomprimido estaba hecho en DELPHI, dada la cantidad de vueltas que da el programa, y si, esta escrito en DELPHI y el DEDE lo abrió perfectamente.

Bueno, sabemos que con el DEDE solo podemos trabajar con ejecutables descomprimidos si no, no funciona, así que puesto el ejecutable descomprimido en el lugar del original, lo abrió perfectamente.

Bueno una vez allí vamos a PROCEDURES y allí buscamos algo y allí esta REGISTERFRM o sea algo que tenga que ver con REGISTRO.

Aclaremos que creo que aquí sin el DEDE nos volveríamos japoneses o chinos buscando entre vueltas y vueltas de DELPHI, así que su ayuda es invaluable.

Bueno haciendo clic en REGISTERFRM - EVENTS aparecen los EVENTOS que hay que mirar, hay varios botones y un poco por intuición y otro poco por mirar un poquito los otros y no encontrar nada interesante, me incline por el primero que es el OK BUTTON. Descarte el HELP button ya que no creo que un botón de help te registre y en el FORM BUTTON no hay comparaciones y saltos interesantes, además el REGBUTTON parece empezar donde termina el OK BUTTON o sea que primero parece ejecutarse el OK BUTTON.

Bueno la cosa estaba entre el OKBUTTON Y EL REGBUTTON ya que el OKBUTTON esta primero lo vamos a estudiar primero.

Después de probar salto por salto con el SOFTICE a ver que pasaba (NO SON TANTOS SON CINCO O SEIS), di en el clavo con el salto de **5E136A.** ESE CUANDO CAMBIE EL VALOR DE **BL** PARA INVERTIR EL SALTO ME DIJO QUE ESTABA REGISTRADO, al reiniciar el programa me desregistro de nuevo así que debe haber al igual que en la versión anterior un CALL antes del salto que es llamado a comparar la clave y lo hace antes del salto **5e136a** y lo llama a ese CALL cuando comienza el programa, ese CALL es **5e1326 CALL 662020.**

Y si lo vemos con el WDASM podemos ver que viene de otro lugar en

REFERENCED BY A CALL AT...

64f43e CALL 662020 (CUANDO ARRANCA EL PROGRAMA)

Así que ahora solo queda poner un BPX en 64f43e para que pare cuando arranque invertir ese salto, y queda registrado para siempre, se pueden realizar los cambios con el ULTRA EDIT y NOPEAR el salto que hay después de 64f43e así siempre te acepta tu registro y no te desregistra.

Parece fácil una vez ya hecho pero el hecho de no tener referencias claras en las STRINGS y de no ser ventanas fácilmente localizables, hacen que sea un poco más difícil, encontrar los saltos, otra cosa que se puede ver que si seguimos el salto **5e136a** y vamos a la parte de CHICO MALO enseguida haciendo F10 varias veces encontramos cual es el CALL de la famosa ventana de PIN INCORRECTO, ya que en DELPHI las ventanas son casi siempre CALLS enteros y esta lo es.

Hasta la lección 16, el año que viene. (BAH, LA SEMANA QUE VIENE)

FELICIDADES
Ricardo Narvaja

LECCIÓN 16: COMPRESORES MALDITOS

Vamos a crackear el programa WEBCLICKER 1.0

La verdad, esta historia de los compresores a veces lo saca a uno de quicio, y este es el caso, vamos a abreviar, esto esta comprimido todavía no se con que, pero no se puede descomprimir, por lo menos con las herramientas clásicas.

El PROCDUMP cuando lo lee se cuelga, el WDASM también el PUPE lo vuelca pero el archivo resultante es una porquería que no se puede abrir con nada, una verdadera basura.

Este tema de los nuevos compresores que existen por ahí es bastante molesto, ya tuve experiencia con el ASPROTECT que es más difícil que este que vamos a ver hoy día y todavía no le encuentro la vuelta de cómo descomprimirlo, a este tampoco, pero eso no quiere decir que no lo vamos a poder crackear, lo haremos usando la única herramienta que nos permite este compresor que es el SOFTICE, aunque también tiene trampa anti-debugger, si modificamos el ejecutable del SOFTICE con el BACKDOOR KEEPER y usamos el nuevo FROGSICE , podemos hacer algunas cosas, otras no, pero igual lo haremos.

Esto es como crackear con una mano atada y sin ver pero lo más importante en este crack es ser creativos y usar un camino que inventemos aunque no sea el más recorrido en los tutoriales de crack, por suerte este compresor permite parchear en memoria con el RISC PROCESS PATCHER, cosa que el ASPROTECT no permite, así que tiene esa pequeña puertita abierta, donde nos vamos a tratar de meter para crackearlo, la verdad es que fue difícil, muy difícil, tarde más de una semana en hacerlo, aunque aquí parezca breve ya que muchos intentos frustrados yo no los transcribo y parece fácil entonces.

Bueno manos a la porquería esta, jaja, chiste. Probemos que nos deja hacer y que no con el SOFTICE.

Arrancar el programa:

SI SE PUEDE siempre que tengas el SOFTICE modificado con el BACKDOOR KEEPER y el nuevo FROGSICE, arranca perfectamente.

Con el SYMBOL LOADER:

Arranca el programa y no para cuando se inicia, o sea que no sirve para nada arrancarlo de aquí, no para porque en las características de las secciones que leí con el PE BUILDER (EL UNICO QUE SE DIGNO A ABRIRSE), me dice que todas las secciones son C0000040 lo que hace que no pueda ser desensamblado ni pare cuando arranca con el SYMBOL LOADER, si le cambio a E0000020, el programa para con el SYMBOL LOADER pero no arranca ya que se testea a si mismo y al verse modificado sale un mensaje de error.

Si le modifico a E0000020 también se desensambla con el WDASM pero no aparece nada interesante ya que esta comprimido y no aparecen STRIGS ni IMP FN ni nada.

Es un acorazado, por donde lo podemos atacar?

Si pongo un BPX antes de que arranque el programa vamos a ver que pasa.

Si es MESSAGEBOXA me sale DEBUGGER DETECTED y no arranca, paciencia amigo ya lo vamos a destripar, probemos con BPX GETVERSION es un BPX que usan la mayoría de los programas cuando se inician, ya que así saben la versión de WINDOWS que tenés en tu maquina y ajustan el programa a eso.

Entonces BPX GETVERSION, para en el SOTICE (ACLARACIÓN SI EN LA ESQUINA INFERIOR DERECHA QUE ES DONDE APARECE EL NOMBRE DEL PROCESO QUE LLAMA A LA FUNCION APARECE EXPLORER EN VEZ DE WEBCLIKER, HAGO X HASTA QUE APAREZCA ALLI WEBCLICKER no en la línea verde sino en el ángulo inferior derecho de la pantalla) pongo **BC*** y **X** a ver si arranca, ARRANCA, bueno ya tenemos una forma de entrar, vamos despacito.

Probemos buscar el punto de inicio del programa descomprimido para ver si pasa. Pero antes tenemos que volver al ejecutable, tenemos que borrar el BPX que habíamos puesto sino va a aparecer DEBUGGER DETECTED, y para volver si hacemos F12 no sirve, a mi se me cuelga así que tengo apretada la tecla F10 hasta que en la línea verde aparezca el nombre del EJECUTABLE WEBCLICKER y ahí suelto.

Según los datos que obtuve con el PE BUILDER y aplicando el método de descompresión manual que vimos en la LECCIÓN 6, escribo

BPR 401000 473000 R if (eip>=401000) && (eip<=473000)

y a ver que pasa X.

PASA Y PARA EN EL INICIO.

4720BC PUSH EBP

Les digo para que ahorren tiempo que si aplicamos aquí el método de cambiar a un loop infinito con

A (ENTER)

y escribo **JMP EIP** o **JMP 4720BC** para que quede en loop infinito y dumpearlo, el PROCDUMP se cuelga cuando lo querés grabar y el PUPE lo que sale no sirve para nada es un chorizo que ni se puede abrir con el PE BUILDER como el ejecutable original.

Todos los otros DUMPEADORES que probé lo mismo lo que sale es una bazofia.

Bueno por lo menos estamos en el inicio del programa, no podemos ver el listado muerto, no tenemos STRING REFERENCES, ni IMP FN, pero a no llorar, parece que llegamos aquí en condiciones lamentables pero hagamos un repaso de los pertrechos que tenemos:

1) ESTAMOS EN EL ARRANQUE DEL PROGRAMA, y podemos partir de aquí usar el SOFTICE a voluntad ya que no chequea más nada ni hay más protecciones ANTI DEBUGGER a partir de aquí.

2) Si encontramos aquí algún salto o algo importante que REGISTRE el programa lo podemos modificar con el RISC PROCESS PATCHER y a cobrar.

Bueno algo tenemos ahora veamos la línea de razonamiento que seguí para crackear esto.

Miré el programa cuando arranca y vi en la ventana del mismo por varios lugares la palabra SHAREWARE, y además sabemos que el programa se registra ONLINE así que debe haber algún lugar que testee si estas REGISTRADO o no y según eso haga aparecer la palabra SHAREWARE en pantalla si no estas REGISTRADO y si estas REGISTRADO no tiene que aparecer.

Eso vamos a buscar la decisión de que aparezca la palabra SHAREWARE en la ventana o que no aparezca, como eso tiene que ver con que si estas REGISTRADO o no puede ser algo que nos lleve a la solución (O NO).

Allí en el punto de INICIO del programa uso la sentencia que tiene el SOFTICE para buscar CADENAS DE TEXTO

S 0 L FFFFFFFF 'SHAREWARE'

ASI CON MAYUSCULAS como aparece en la ventana.

La sentencia de buscar texto es así y si quieren buscar otra vez ya que así sale solo la primera SHAREWARE que encuentra tiene que volver a escribir

S XXXXXX L FFFFFFFF 'SHAREWARE'

donde XXXXXX es la dirección de memoria justo después de donde apareció la primera vez.

Ya que aquí apareció en 470b8b (EN MI CASO ESTO PUEDE VARIAR EN CADA COMPU), si quisiera encontrar la segunda vez que aparece tendría que escribir.

S 470b9b L FFFFFFFF 'SHAREWARE'

y además tener en cuenta que el SOFTICE aquí acepta como comillas no las comillas dobles estas " , sino las comillas simples estas '.

Bueno apareció en 470b8b, entonces pongamos un BPR allí para ver cuando el programa la usa a esa palabra. Primero borro los BP anteriores con **BC*** y escribo

BPR 470b8b 470b9b RW

cosa que cuando el programa se decida a utilizar la palabra SHAREWARE pare allí el SOFTICE, y estaremos en la zona de CHICO MALO, y tendremos que buscar la decisión en una comparación y un salto anterior para llegar a la ZONA DE CHICO BUENO o REGISTRADOS y que evite que use la palabra SHAREWARE.

Bueno para en **402855 REPZ MOVSD** que es la sentencia que esta copiando la palabra SHAREWARE para escribirla en pantalla.

Veo que unas sentencias más abajo hay un RET lo cual quiere decir que estoy dentro de un CALL y ese RET es la FINALIZACION del CALL, entonces para salir del CALL y volver a la sentencia subsiguiente después del mismo, traceo con F10 hasta el RET y lo traceo también y salgo a **403d40.**

El CALL que esta justo antes en **403d3b CALL 402828** que era adentro de donde yo estaba y donde el programa trabaja la palabra SHAREWARE es para mi la ZONA de CHICO MALO, tengo que tratar de saltear ese CALL de alguna forma para que no caiga allí.

Pruebo poniendo BPX en los saltos que están arriba del CALL y reiniciando todo para que caiga allí pero no funcionan, entonces como veo otro RET me doy cuenta que estoy en otro pequeño CALL todavía y hago lo mismo para salir haciendo F10 hasta el RET y caigo en la sentencia posterior a

470960 CALL 403d0c

o sea que este CALL también es la ZONA DE CHICO MALO ya que dentro de el esta el otro CALL 402828 que salí primero, o sea que si caigo en 470960 voy a parar a la palabra SHAREWARE, miro arriba del CALL y encuentro algo interesante bastante arriba en **470815 jz 47094b.**

Es un salto que si no salta, al seguir traceando con f10 veo que sigue línea por línea hasta **470946 jmp 4709e4** que saltea el CALL maldito, y si el salto en 470815 salta, al esquivar el JMP ya que cae después del mismo, nos lleva directo al CALL SHAREWARE. Quiere decir que ese salto no debería saltar para que no caiga en SHAREWARE, probemos.

Pongo un BPX en 470815 y reinicio todo (TENGO QUE HACER TODOS LOS PASOS QUE HICE ANTES PARA QUE PARE SI NO PASA DE LARGO), bueno cuando paro allí, veo que el SOFTICE me avisa que va a saltar y mandarme a SHAREWARE, invierto el salto con

r eip=47081b

y luego hago X y PROGRAMA REGISTRADO, oh que maravilla no solo esquivo SHAREWARE sino que agarro por el camino de CHICO BUENO directo. Igual queda un detalle que es que a veces aparece el cartel ese blanco diciendo que nos tenemos que registrar a pesar que en el PROGRAMA nos dice que ya estamos registrados, eso se arregla fácil veamos que compara antes del salto

MOV eax, [474e08]

mueve a eax el contenido de 474e08 que es 47451c, o sea que eax vale ahora 47451c. Después compara el primer byte de el contenido de 47451 con cero y si es cero nos tira a los chanchos y si es uno estamos registrados. Aquí lo que ocurre es que este es un flag que le dice al programa si esta registrado o no, y más adelante lo debe testear para ver si sale el cartel blanco ese que aparece cada cuatro o cinco veces que arrancas el programa, entonces para evitar problemas, cambio la sentencia

cmp byte ptr [eax],00

a

mov byte ptr [eax],01

con lo que pasa el flag a ser uno y nunca más aparece el cartel blanco ese y para el programa estamos perfectamente registrados.

Luego hacemos el cargador con el RISC PROCESS PATCHER cuyo script seria el siguiente:

```
; WEBCLICKER
F=Webclicker.exe:                ; PROCESS TO PATCH
O=crack_the_webclicker2.exe:     ; LOADER TO CREATE
P=470812/80,38,00/c6,00,01:      ;
P=470815/0f,84/eb,04:            ;
$                                ;end of script
```

en la primera línea **P** cambio la sentencia comparar a MOVER como vimos antes y en la segunda para no nopear tantos bites directamente cambio el salto por un JMP 47081b, sino tendría que poner muchos NOPs, lo hago al parche lo copio en la carpeta del programa , lo ejecuto, y arraaaaancaaaa perfectamente registrado.

En resumen le hicimos morder el polvo a una difícilísima protección sin siquiera descomprimir el ejecutable, jaja, pero fue muuuy duro realmente.

El programa Webclicker estará en el freedrive, para bajarlo.

HASTA LA LECCIÓN 17, amigos.

Ricardo Narvaja

LECCIÓN 17: ALGO SUUUPER FÁCIL. EL CRACK DE LA ACTUALIZACION DE VIRUS DEL NORTON

Bueno como ustedes sabrán si no se les cuento, nuestro amigo NORTON nos dejaba actualizar un año por medio de su LIVE UPDATE las definiciones de virus, y cuando se te vencía ese año (QUE PARECE QUE DURA UNOS MESES SOLAMENTE), podías seguir bajando las actualizaciones de la pagina de SYMANTEC

http://www.symantec.com/avcenter/cgi-bin/navsarc.cgi

o

http://www.sarc.com/avcenter/cgi-bin/navsarc.cgi

y no había problema. Bajabas el archivito que tiene un nombre así **0825i32.exe** que quiere decir que es la actualización del 25 del 08 o sea de agosto, bueno todo era hermoso, hasta que se le ocurrió, limitar también el archivo que podes bajar, o sea que una vez que lo ejecutas, testea si te venció ese año y si ya paso la fecha no te deja actualizarlo y te pide que pagues una suma para poder actualizar los virus.

Bueno les comento que crackear ese archivito es muy sencillo, desde hace dos meses todos los archivitos de actualización tienen la misma protección y en el mismo lugar, o sea que al día de hoy 10 de enero del 2001, lo que explico en esta lección es valido, si llegara a cambiar de lugar la protección, el método es muy sencillo y seguro va a ser el mismo aunque en otro lugar, pero hasta hoy parece que va a seguir igual.

Si ejecuto el archivo de definiciones de virus cuando ya vencieron las actualizaciones me sale un cartel que dice:

YOUR VIRUS SUSCRPICIONS CANNOT BE UPDATED. YOUR SUSCRIPCION HAS EXPIRED... y un par de pavadas más.

Abrimos el archivo con el WDASM y vemos las STRINGS REFERENCES y allí aparece la cadena de texto YOUR VIRUS SUSCRIPCION... parece que NORTON no tenia ganas de molestarse mucho, hacemos doble clic en esa STRING REFERENCE y vemos que corresponde a un solo lugar **4042af** (hasta hoy), y que viene de un

REFERENCED BY A CALL AT 401dff

Vamos a **401dff** y vemos un poquito más arriba que hay dos saltos. Probamos ambos si queremos con el SOFTICE, pero es el primero de los dos el que funciona bien esquivando el CALL que nos lleva al cartelito maldito.

:401df4 751C jne 401e12

Aquí si salta pasa por encima del CALL así que reemplazamos **75** por **EB** para hacer que salte siempre, pero no lo vamos a hacer con el ULTRA EDIT porque si no cada archivo que bajemos vamos a tener que editarlo, vamos a hacer un cargador con el RISC PROCESS PATCHER para cargar el archivo de actualización.

Como el RISC busca el nombre del ARCHIVO lo único que vamos a tener que hacer cuando bajemos uno, es cambiarle el nombre a NORTON.exe y listo.

El SCRIPT es el siguiente:

```
; NORTON UPDATE
F=NORTON.exe:                ; PROCESS TO PATCH
O=PARCHEARNORTON.exe:        ; LOADER TO CREATE
P=401df4/75/eb:              ;
$                            ;end of script
```

Con esto el parche cargara cualquier archivo de actualización llamado NORTON.EXE (ACORDARSE DE RENOMBRARLO) y buscara el salto en **401df4** y allí reemplazara el **75** por un **EB** convirtiéndolo en un **JMP** para que siempre esquive el CALL. FACILISIMO.

Probamos el cargador y FUNCIONA, actualiza los virus perfectamente. Si más adelante el cargador alguna vez no funciona es:

1) Porque no renombraron el archivo de actualización a NORTON.EXE.

2) Porque no pusieron ambos el parche y el NORTON.EXE en la misma carpeta.

3) Porque la protección cambio de lugar y el salto esta en otro lado.

De cualquier manera se ve que es muy fácil hallar donde esta ya que el archivo no esta comprimido ni protegido, por lo tanto solamente mirándolo con el WDASM, sin siquiera ser necesario usar el SOFTICE pudimos hacer el PARCHE.

Esta LECCIÓN fue como un oasis de descanso después de la 16 que fue difícilísima, no?

Nos vemos en la 18

Ricardo Narvaja

LECCIÓN 18: SIGAMOS CON LA ONDA LIGHT. CRACK DEL ATAMA 1.8

Bueno vamos a seguir la onda light con cracks fáciles ya que el verano y el calor están haciendo estragos en nuestras neuronas como para complicarnos mucho.

Este programa es supuestamente sobre HOMEOPATIA de lo cual yo no se nada ni tampoco de medicina, si esta ciencia sirve o no, yo no puedo opinar, debería preguntar alguien más versado sobre el tema como el famoso Doctor Theumann que de esto sabe, jaja, pero bueno, igual nos va a servir para practicar un poco.

Supuestamente el programa dice según la dolencia cual es el remedio que hay que usar, es como un inventario sobre Homeopatía, y bueno quizás a alguien le sirva. Se baja de

http://www.abouthomeopathy.com/

Instalamos el programa y abrimos el ejecutable ATAMA.exe con el WDASM y vemos en FN IMP (FUNCIONES IMPORTADAS) la mención de **MSVBVM6.0** lo cual nos dice que esta hecho en VISUAL BASIC. Igual resultado podríamos haber obtenido abriendo el ejecutable con algún identificador.

Probamos abrir el ejecutable con el SMART CHECK y no funciona, parece estar protegido contra el SMART pero no nos hagamos problema.

Cerramos entonces el WDASM y abrimos el otro WDASM el que esta modificado para VISUAL BASIC según vimos en la lección sobre VISUAL y allí lo abrimos y lo desensambla perfectamente y aparecen las STRING REFERENCES lo que indica que no esta comprimido.

Si ejecutamos el programa vemos que se limita luego de algunos usos y al arrancar aparece una ventana donde haciendo clic en REGISTRATION aparece donde ingresar la clave.

Si ingreso un número cualquiera me sale el siguiente cartelito:

"INVALID REGISTRATION NUMBER"

Vamos al WDASM y nos fijamos en las STRING REFERENCES y ALLI ESTAAAA, hacemos click encima de INVALID REGISTRATION NUMBER y el WDASM nos marca un solo lugar **43D1CA** y allí vemos la mención al cartel podrido y más arriba vemos que viene de dos lugares:

REFERENCED BY A UNCONDITIONAL O CONDITIONAL JUMP AT ADDRESS

43D043 y 43D050

Estos parecen ser dos comparaciones una después de otra, y en cualquiera que salta a esta zona de chico malo nos aparece el cartelito maldito... habrá que evitarlo!!!

Vamos a ver en el WDASM que hay en **43d043.** En GOTO - GOTO CODE LOCATION, ponemos 43DO43 y nos muestra

43D043 0F8481010000 JE 0043D1CA

AQUÍ VEMOS EL SALTO CRITICO Y DOS RENGLONES MÁS ABAJO HAY OTRO QUE ES EL DE **43D050** QUE NOS TIRA A CHICO MALO TAMBIEN.

Si arranco el programa y voy a la ventana de registro y pongo cualquier número y antes de ACEPTAR entro en el SOFTICE y pongo BPX HMEMCPY y hago clic en REGISTER, rompe en el SOFTICE. Pero si quiero volver al ejecutable con F12 hago como treinta veces F12 y nunca aparezco en el ejecutable pero si me aparece el cartelito de INVALID.

Como entro en el ejecutable para poner los BPX en los puntos que ya hallamos (43D043 y 43D050)?.

Vuelvo a poner el BPX HMEMCPY y cuando rompe en el SOFTICE hago F12 hasta que llego al archivo de VISUAL BASIC MSVBVM60 y a partir de allí tengo apretada F10 hasta que me aparece en la línea verde ATAMA, que es cuando llegamos al ejecutable, ufff.

Una vez allí borro los BPX viejos con **BC*** y pongo **BPX 43D043** y **BPX 43D050** y hago X. Parara en **43D043,** si no para porque se habían pasado de esa sentencia vuelvan a poner cualquier número en la ventana de registro y ahora si parara allí en 43D043.

Vemos que el SOFTICE indica que en el salto 43D043 va a saltar a la ZONA DE CHICO MALO (JUMP) o sea que podemos hacer **r eip=43D056** y así salteamos los dos saltos que nos llevan a CHICO MALO, hacemos X y vemos que nos sale el cartelito **DEMO HAS BEEN CONVERTED TO SHAREWARE** o sea que el programa dejo de ser una versión de prueba, que bien.

Vemos que si arrancamos de nuevo el programa aparece la ventanita de REGISTRACIÓN pero ya no cuenta las veces que faltan para que caduque si salteamos la ventana de REGISTRACIÓN arranca el programa normalmente, o sea que funciona y no vence, aunque queda pendiente eliminar la ventana molesta del inicio, lo cual dejaré para que ustedes practiquen. El que pueda hacerlo, escríbame a mi email un pequeño informe de cómo lo hizo y pasara a engrosar la nueva sección de la pagina WEB del curso, llamado **LOS TUTORIALES DE LOS LISTEROS**, el primero que me lo mande y la solución sea correcta, su tutorial va a ser el que va a aparecer ya que no podemos poner a todos.

Envíen sus tutoriales a la lista crackslatinos o a mi email **ricnar456@data54.com** y el primero que llegue y sea correcta la solución se lo publicaremos en la pagina WEB del curso.

Página del curso: http://www.virtue.nu/latinos/curso/curso.htm

Subscribirse:
crackslatinos-subscribe@egroups.com

Borrarse de la lista:
crackslatinos-unsubscribe@egroups.com

Aquí esta el parche hecho en el RISC PROCESS PATCHER para que no caduque más, (LA ELIMINACIÓN DE LA VENTANA NO ESTA AQUÍ EN ESTE PARCHE)

```
;ATAMA
F=AtamA.exe:              ; PROCESS TO PATCf
```

```
O=crack_the_AtamA.exe:    ; LOADER TO CREATE
P=43d043/0f,84/EB,11:     ;
$                         ;end of script
```

Hasta la próxima

RICARDO NARVAJA

LECCIÓN 19: OTRO EN VISUAL BASIC PERO MAAAS DIFÍCIL

Este programa llamado Registry Compare 1.22 se baja de http://www.kmcsonline.com/rcompare.htm

Es un programa que permite tomar como una instantánea del registro y después de instalar un programa volver a tomar otra para ver cuales fueron las modificaciones en el mismo, yo para eso uso el programa ART ADVANCED REGISTRY TRACER que es más detallado y te dice que entradas fueron agregadas, cuales fueron borradas y en cuales fueron cambiados algún valor.

El ART se baja de http://www.elcomsoft.com/art.html y el crack de ASTALAVISTA y algunos se preguntaran porque no crackeamos el ART, en vez del Registry Compare, lo que pasa es que el ART ya lo crackee y no tiene algo interesante para enseñar pero el REGISTRY TRACER nos puede enseñar dos o tres cosas. O sea vamos a usar el ART para detener el conteo de 14 veces de prueba del REGISTRY COMPARE dándole un trago de su propia medicina, es decir, con el mismo método que utiliza el programa (14 veces para probar un programa, MISERABLE... ja,ja)

Primero bajo el ART y lo instalo, bajo el crack de ASTALAVISTA y una vez que hice todo eso, tomo la primera foto del registro antes de instalar el REGISTRY COMPARE o sea en el ART voy a SCAN REGISTRY y me toma una instantánea con fecha y hora del registro.

Una vez que termino eso, instalo el REGISTRY COMPARE y una vez instalado, lo ejecuto una vez y lo cierro, supuestamente ya se crearon las entradas en el registro donde guarda la información de las veces que lo usaste, así que vuelvo a abrir el ART y de nuevo SCAN REGISTRY y una vez que termina, marco la primera de las dos fotos que tome para que compare a partir de allí, y en el menú elijo COMPARE FROM HERE o sea COMPARE DESDE AQUÍ.

Como ahora queremos ver las nuevas entradas que creo el programa al instalarse, entonces vamos a la pestaña ADDED que son las entradas agregadas para ver que hay de nuevo en el REGISTRO después de haber instalado el REGISTRY COMPARE.

Vemos que en added hay tres nuevas entradas. Las tres nuevas entradas son:

1)HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\KMCS Registry Compare_is1

2)HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Explorer\MenuOrder\Start Menu\&Programs\KMCS Computer Software

3)HKEY_USERS\.DEFAULT\Software\VB and VBA Program Settings\KMCS\REGCOMPARE

Bueno ahora voy a INICIO - EJECUTAR -REGEDIT y allí busco cada una de las cadenas estas y cuando están completamente abiertas y muestran los datos en la parte derecha, entonces voy al menú REGISTRO y allí EXPORTAR ARCHIVOS DEL REGISTRO y exporto un archivo por cada entrada que agregó el programa y las copio en un lugar que tenga a mano.

Ejecuto un par de veces el REGISTRY COMPARE para que consuma un par de veces... ahora me dice que me quedan doce veces para utilizar.

Cierro el programa REGISTRY COMPARE y ejecuto uno por uno los archivos que creé del registro para inyectar los datos como estaban originalmente, al ejecutar me dice si quiero ingresar datos al registro y pongo que si, arranco el REGISTRY COMPARE y me dice que me quedan 14 veces para probar, juuuuy, ya descubrí como volverlo para atrás, jajaja.

Esto es muy importante porque me da tiempo para descubrir el crack y si quiero lo puedo seguir utilizando así, ya que cuando llegue a que me quedan cinco veces vuelvo a ejecutar los tres archivos del registro y me vuelve la cuenta atrás a 14 de nuevo, iuuupi.

Ahora podemos trabajar tranquilos pero no voy a abundar mucho en como crackear esto, ya que para mi lo importante era que vean como se hace en programas que vencen después de varias veces y guardan la información en el registro.

Si por esas casualidades, (QUE NO ES ESTE CASO), al realizar estos pasos con otro programa y restaurar el registro a los valores que tenia al momento de instalar el programa este no vuelve atrás es posible que guarde la información en un archivo, ejemplo de lo cual veremos más adelante, pero les adelanto que utilizando el FILEMON vemos que filas usa y las copiamos en otro lado y cuando usamos un par de veces el programa, volvemos a copiar las filas esas que guardamos en algún lugar encima de las del programa y podemos lograr que vuelva atrás y si utiliza una fila en C también la podemos detectar con el FILEMON y reemplazarla por una copia después de usar el programa un par de veces.

Bueno si arrancamos el ejecutable COMPARE.EXE con el SYMBOL LOADER y cuando empieza vamos traceando con F10, y no salteamos ningún CALL, si encontramos uno entramos con **T** y seguimos traceando, enseguida nos vamos a encontrar que estamos en el archivo de VISUAL BASIC **VB40032.**

Ahora viene el truco de magia ponemos **BPX 0f79b358** y hago correr el programa con X, cada vez que para allí hago **D EDI** o **D ESI** y me dará la clave.

Si llego a la ventana de registro y no me dio la clave sigo, pongo una clave falsa y acepto y una o dos veces después que paro allí me dio mi clave **RCEQ70NS** en formato ancho, o sea con puntitos entre las letras pero en la ventana de registro hay que escribirla normal.

COMO DESCUBRI ESTO?

Ja ja, es difícil explicar las mil vueltas que tuve que dar para llegar a esto pero ahí van varios datos para el que quiera marearse un poco.

El programa a pesar de ser de VISUAL BASIC al estar escrito en P-CODE no es mucho lo que se puede ver con el SMART CHECK ya que este avisa que con archivos en P-CODE no son muchos los datos que suministra para crackear.

Yo lo hice y me costo mucho con el método de HMEMCPY una vez que escribí la clave falsa puse aceptar y allí entré en el SOFTICE y seguí según el método que vimos en las lecciónes pasadas con la diferencia que aquí en vez de **d ds:esi** o **es:edi** hay que utilizar **ds:si** o **es:di** ya que según vemos estamos en HMEMCPY de 16 bits en vez de 32.

Pero es igual, después hacemos PAGE el valor que nos aparece la cadena 989898 (QUE ES LA QUE YO PUSE) y una vez que localizo el valor donde se encuentra empiezo a poner BPR siempre y cuando mueve la cadena a otro lugar también BPR y cuando pasa con MULTIBYTETOWIDECHAR el valor a formato ancho y descubro donde lo guarda , allí también un BPR y cada tanto hago un

S 0 L FFFFFFFF '989898'

Y

S 0 L FFFFFFFF '9.8.9.8.9.8'

y así llegue a que en un momento, DESPUES DE BASTANTE TRABAJO, paro en

0F79B358 REPZ CMPSW

QUE ES UNA SENTENCIA DE COMPARACIÓN QUE EN **D ESI** y en **D EDI** están las claves falsa y buena en formato ancho.

Así que copie mi clave que decía en el SOFTICE **R.C.E.Q.7.0.N.S** y la escribí en la ventana para REGISTRARME todo corrido **RCEQ70NS** y voila jitanjafora, estoy REGISTRADO bastante trabajo mediante, jaja.

SI A USTEDES NO LES SIRVE MI CLAVE ES PORQUE USA UNA DIFERENTE PARA CADA COMPUTADORA PERO LA PUEDEN ENCONTRAR FÁCILMENTE COMO LOS INDIQUE.

No importa aquí mucho como hallar el crack pero si el método de detener el conteo por medio de las fotos del registro. Es lo que vale ya que el crack es cansador por lo vueltero pero con paciencia se encuentra, por eso no insistí mucho con eso y si con el método de las fotos con el ART y ahora también lo podemos hacer con el REGISTRY COMPARE.

Ricardo Narvaja

LECCIÓN 20: COMO QUITAR EL CARTELITO MOLESTO Y REGISTRAR EL ATAMA

Sinceramente esta semana no pensaba realizar ninguna lección dado que tuve muchísimo trabajo y no tuve tiempo, pero dado que no hubo soluciones sobre como quitar el cartelito del ATAMA de la LECCIÓN 18 vamos a quitarlo y de paso REGISTRARNOS, en una lección breve, ya que no tuve mucho tiempo esta semana.

Dado que el ATAMA no tiene trucos ANTISOFTICE lo arranco desde el SYMBOL LOADER y realizo este procedimiento:

Apenas arranca encuentra un CALL que si lo ejecutamos arranca el programa y no para más, volvemos a arrancarlo y entramos dentro de ese CALL, y así seguimos traceando con F10 y tomamos la precaución de antes de pasar por encima de un CALL poner un BPX allí cosa de que si arranca el programa, podemos reiniciar y parar allí.

Realizamos este procedimiento y cuando encontramos un CALL que arranca el programa y nos hace aparecer el cartelito maldito, volvemos a arrancar y como habíamos puesto un BPX vuelve a parar allí y ahora en vez de pasar por encima del CALL entro con T y sigo traceando con F10.

Cada CALL que encuentro antes de pasarle por encima borro todos los BREAKS anteriores con **BC*** y pongo un BPX en el último CALL este, si paso por encima y no sale el cartelito maldito, sigo hasta el otro CALL, y allí repito el procedimiento, entrando con T en los CALLs que me hacen aparecer el cartelito y salteando los CALLS que cuando los paso por encima no paso nada.

Así me voy acercando cada vez más al lugar donde el cartel principal de ATAMA tenga un CALL y el cartelito maldito otro CALL separado esto ocurre en:

417FA7 CALL [EAX+2A0] (CARTEL ATAMA)
4180A9 CALL [EDI+2B0] (CARTEL MALDITO QUE HAY QUE ELIMINAR)

Es seguro que en el programa registrado el primer cartel que dice ATAMA aparece y el segundo cartel no, por lo tanto entre los dos CALLs debe haber una comparación y un salto que evita el segundo cartel y que puede registrarnos ya que seguimos el camino del CHICO BUENO en ese salto.

La comparación y el salto están a partir de **417FCF** ya que allí compara si el valor que hay en **459090** es cero y según esa comparación salta evitando el cartelito molesto.

Puedo probar a ver que pasa si coloco a mano en **459090** un **UNO** haciendo **D 459090** y después **E** con lo que puedo escribir el primer número y cambiarlo de **00** a **01**, quedándome ahora el valor de **459090** en UNO.

Ejecuto la comparación y el salto y arranca el programa sin aparecer el cartelito maldito y cuando voy a ABOUT me dice REGISTERED, jaja, pudimos con el.

Para asegurarme que en **459090** quede el valor uno que me asegura que este registrado voy a cambiar la sentencia

417FCF CMP WORD PTR [459090],00

O SEA, LA SENTENCIA QUE COMPARA SI ESTAS REGISTRADO POR

417FCF MOV WORD PTR[459090],01

LO QUE VA A COLOCAR EN **459090** EL VALOR **UNO** PARA QUE QUEDE REGISTRADO Y AHORA MODIFICO EL SALTO QUE VIENE DESPUES PARA QUE SALTE SIEMPRE, PONGO

417FD8 JMP 418117

Y LISTO.

Puedo ir al Ultra Edit, buscar la cadena y cambiarla sin miedo ya que este programa no esta protegido contra esos cambios.

66833d90904500000f853a010000

lo reemplazo por

66c705909045000100e93a010000

y listo, queda registrado para siempre y no sale más el cartelito molesto.

Hasta la próxima.

Ricardo Narvaja

LECCIÓN 21: AGENDA MSD 2.0

A pedido de una amiga VERÓNICA VILLAVERDE que actualizó su programa AGENDA MSD de una versión anterior a la nueva 2.0 y le borro el registro que tenia anteriormente, lo que hizo que buscara el crack en ASTALAVISTA pero ese crack que era un GENERADOR DE CLAVES no funcionaba, entonces me propuse hallar la clave de este programa.

Primero trate de ver el ejecutable AgendaMSD.exe en el WDASM y desensamblarlo pero no sale nada ya que esta comprimido.

Para averiguar que compresor se ha utilizado, uso el UNPACK 2000 que esta bastante actualizado ya que el GETTYPE esta un poco desfasado y casi no reconoce nada.

Coloco una copia del ejecutable en la carpeta del UNPACK y salgo a DOS voy hasta el directorio donde esta el unpack en mi caso

D:\Download\herramientas crcak\unpack20

y allí tecleo

un-pack agendamsd.exe

y después de un montón de información al final me sale

Packed by ASPack v.1.08.04 by Alexey Solodovnikov
Use UnAspack v.1.0.9.1 by BiWeiGuo for unpack

O sea que esta comprimido con ASPACK 1.08.04 y también me aconseja un descompresor el UNASPACK 1.0.9.1 pero como en el PROCDUMP vi que estaba esa versión del compresor lo haré con PROCDUMP.

Tengo varias versiones del PROCDUMP increíblemente la que me funciono bien es la original ya que otras actualizaciones me daban al descomprimir SCRIPT ERROR.

Bueno lo descomprimo como vimos en la LECCIÓN de descompresión automática con procdump eligiendo ASPACK108.4 y después de un rato de trabajar lo guarda.

Ojo que hay que quitarle la protección antidesensamblado C0000040 y cambiarla por E0000020 para poder verlo en el WDASM.

Una vez que hice todo eso lo abro con el WDASM, me salen las STRING REFERENCES y allí una muy importante que es el mensaje que aparece cuando pones una clave que no es correcta para registrarte como vemos en la imagen (EL PROGRAMA NO SE HA REGISTRADO CORRECTAMENTE....)

Haciendo doble clic en la STRING REFERENCE y después subiendo un poco la imagen vemos que viene de una

REFERENCE BY A CONDITIONAL....JUMP at address 629430

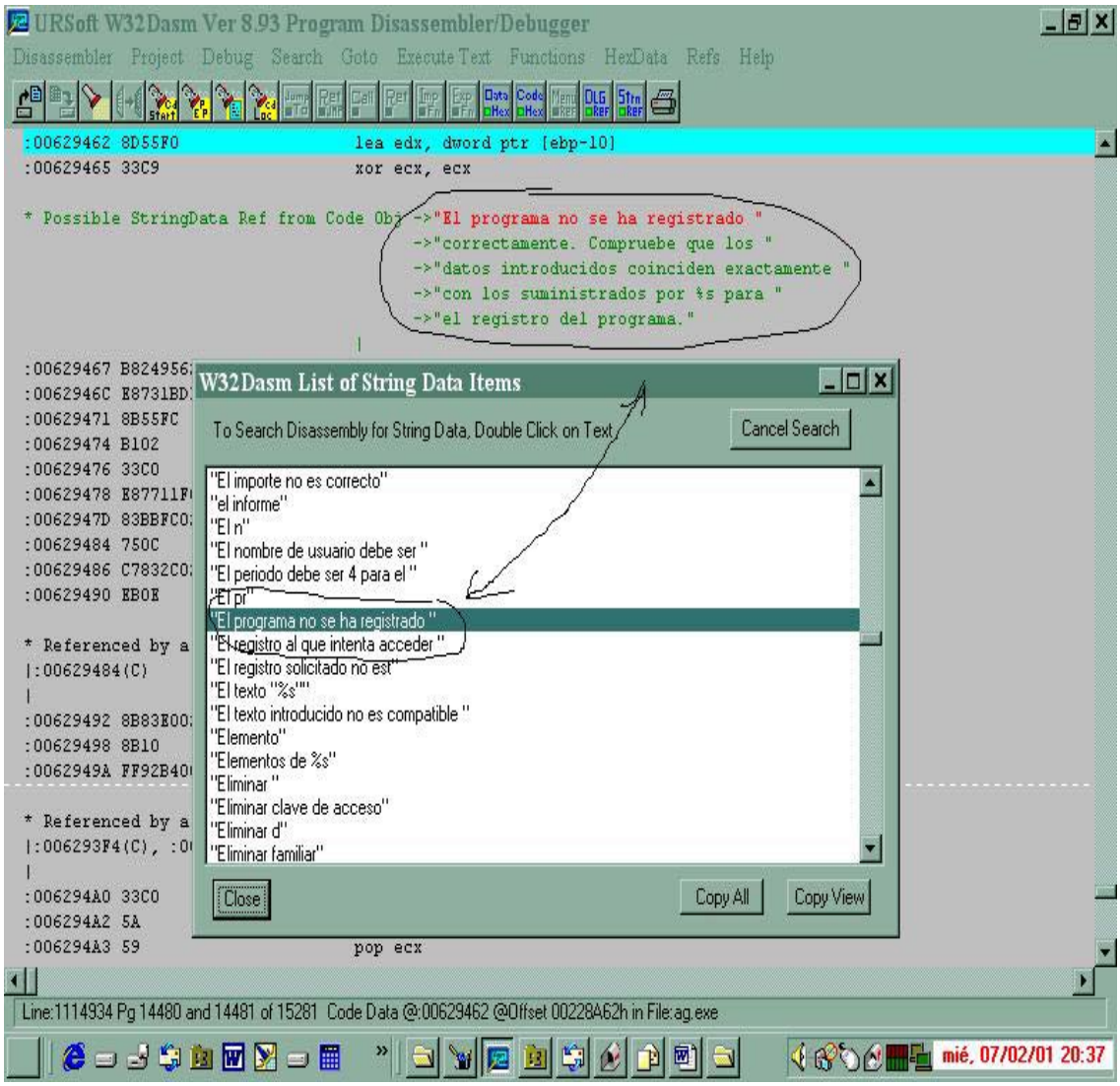
que es un poquito más arriba de donde estamos. Allí vemos un salto en **629430** que claramente decide si vamos a estar registrados ya que si salta nos lleva a la ZONA DE CHICO MALO que estábamos antes y si no

salta vemos que abajo aparece otro cartelito que dice **ENHORABUENA EL PROGRAMA HA SIDO REGISTRADO SATISFACTORIAMENTE.**

Bueno ya sabemos cual es el salto que puede registrarnos, pero podremos ver la clave?

Entonces arranquemos la maquina con el SOFTICE y arranquemos el programa, vayamos a MANTENIMIENTO - DESBLOQUEAR EL PROGRAMA y la ventana de registración te pide que pongas un nombre de más de ocho caracteres y separados por un espacio vacío, podría ser:

Nombre: **narvaja ricardo** o el que quieran y pongan una clave falsa que llene todos los espacios vacíos por ejemplo
Clave: **9898-9898-9898-9898-9898-9898.**



Antes de tratar de DESBLOQUEAR el programa entremos al SOFTICE y pongamos **BPX HMEMCPY** y volvamos a WINDOWS con CTRL+D. Esperamos un poquito para ver que no vuela a romper el softice solo y aceptamos para que ingrese la clave lo que rompe en el SOFTICE nuevamente.

Tecleamos F12 tantas veces como sea necesario hasta que en la línea verde del SOFTICE hayamos vuelto al ejecutable AGENDA.

Vamos a poner un **BPX** en **62940E** un poco antes de el salto para ver si aparece por ahí la clave verdadera. Una vez que para ahí voy traceando y haciendo **D ...** para ver si aparece la clave y me aparece así.

Traceo hasta **629423** allí hago **D EAX** y aparece mi nombre **ricardo narvaja**. Ejecuto con F10 hasta que llego al salto, allí bajo un poco la ventana de datos para ver si hay una secuencia de números o letras de 24 cifras, si no aparece pongo X y repito el proceso, a la tercera vez a mi me apareció siempre, en **629423** hice **D EAX** y apareció mi nombre y seguí traceando con F10 hasta llegar al salto y allí fui bajando la ventana de datos para ver debajo de mi nombre y 20 o 30 líneas más abajo apareció la clave de 24 letras y números, ojo que son mayúsculas y minúsculas y hay que copiarla bien, porque una **i** parece una **I** o un **1**, son parecidos fíjense bien.

Bueno ya hallamos la clave para el agenda MSD 2.0 un saludo hasta la próxima LECCIÓN.

Para mi nombre empieza con **1jN4-9P13-11x2-0Hp6-1w12-....** y los cuatro últimos descúbranlos ustedes. A ver quien manda primero la clave MIA completa.

Ricardo Narvaja

LECCIÓN 22: CIBER-BOSS 2.1 (PROGRAMA PARA CIBER CAFES)

Recibí un mail de un amigo que me decía lo siguiente:

> Ricardo:
>
> Estimado amigo... estoy intentando instalarme con un CyberCafé en la ciudad de .....
>
> Y para llevar un buen control, tanto del tiempo, como el de las impresoras y Scanner, encontré un excelente
>
> programa en Español...se llama **"Ciber Boss 2.1"**...está en la siguiente dirección
>
> http://www.jm-soft.com

Bueno ya que parece interesante fuimos a ver de que se trata, hicimos un crack y como yo no tengo CyberCafé no lo puedo probar pero como no hubo quejas supongo que funciono bien.

La primera vez que lo ejecuto puedo saltear la identificación del usuario con aceptar solamente pero es conveniente que una vez dentro, vayamos a MAQUINA SERVIDOR - MANTENIMIENTO y allí vayamos a MANTENIMIENTO DE EMPLEADOS y editemos el que aparece allí con grado MASTER y le pongamos una clave y un login que debemos recordar ya que después de unas veces de usarlo si no lo hicimos no nos dejara entrar más. También se pueden configurar aquí los empleados que van a usar el programa y ponerles una clave y login y nivel OPERADOR, y así sucesivamente.

Una vez bajado el programa veo que el ejecutable se llama servidor.exe y que se puede desensamblar con el WDASM y busco las STRING REFERENCES de la ventana que me aparece al empezar el programa después de colocar la clave.

Dicha ventana dice:

VERSIÓN SHAREWARE DE CIBER BOSS

Usted esta utilizando una versión shareware (PROBAR ANTES DE COMPRAR) de CIBER BOSS.....

Bueno, este cartelito aparece solo en esta versión sin registrar así que si encontramos la STRING REFERENCE quizás podamos ver donde el programa decide si tiene que aparecer esta ventanita o no tiene que aparecer (REGISTRADO).

Abrimos el WDASM y vamos a IMP FN (IMPORTED FUNCION) y allí vemos que las referencias son todas al archivo MSVBVM60 que es de VISUAL Basic, así que cerramos este WDASM y abrimos el que tenemos modificado para ver STRINGS REFERENCES de Visual Basic.

Allí en las STRING REFERENCES aparece esta:

"USTED ESTA UTILIZANDO UNA VERSI", obviamente la escondió un poco pero no lo suficiente, hago doble clic encima de esta STRING y aparecemos en **447b81** y arriba justo la STRING esta.

Vamos mirando hacia arriba de esa posición y el primer salto condicional que encontramos que puede evitar este cartelito ya que lo saltea por encima se encuentra en **447aa0.**

```
447a97  movsx edx, word ptr [4b2094]
447a9e  test  edx, edx
447aa0  je    447da5
```

O sea, que según el valor que hay en **4b2094** el cual es transferido a **EDX,** luego compara EDX si es cero y si es cero salta por encima del cartel a **447da5.**

Lo que tenemos que hacer aquí para registrarnos es modificar un poquito estas sentencias para mover a 4b2094 un valor de cero para que más adelante el programa si vuelve a evaluar si estamos registrados encuentre un cero allí y crea que lo estamos.

Podemos probar con el SOFTICE abrir el SYMBOL LOADER y arrancar el programa y poner un **BPX** en **447a97** y cuando para allí, hacemos **D 4b2094** y vemos que en las dos primeras cifras hay **FF FF.** Hacemos clic con el mouse en la primera y reemplazamos ambas por **00 00** ya que testea las dos.

Ejecutamos con F10 y llegamos hasta el salto y vemos que marca JUMP o sea que va a saltar, tecleamos X y el programa arranca normalmente como si estuviéramos registrados y no aparece la ventana maldita ni tampoco cuando cerramos el programa nos aparece la otra ventana que dice si lo queremos comprar, entonces ya esta listo.

Lo que tenemos que hacer es una vez que para en 447a97 teclear A y ENTER y escribir estas tres sentencias

```
447a97 mov word PTR [4b2094], 0000
447aa0 jmp 447da5
447aa5 nop
```

y si lo ejecutamos realiza el mismo trabajo de guardar el doble cero en 4b2094 y arrancar como REGISTRADO.

Entonces copiamos la cadena que comienza en 447a97

66C70594204B000000E90003000090

y LA ORIGINAL ERA

0FBF1594204B0085D20F84FF020000

ESTA ULTIMA LA BUSCAMOS EN EL ULTRA EDIT, LA REEMPLAZAMOS POR LA DE ARRIBA Y PROGRAMA CRACKEADO.

RICARDO NARVAJA.

LECCIÓN 23: UNA FÁCIL

Ya que esta semana trabaje mucho y estoy muuuy cansado, voy en esta lección a hacer un crack fácil dentro de todo.

Es el programa **TURBONOTE 4.5** se baja de http://turbonote.com/

Y una vez que lo abrimos y vamos a GENERAL SETTINGS esta la opción REGISTER TURBO NET donde se puede poner una clave y al ACEPTAR abajo si no es correcta desaparece la ventana sin salir ningún cartelito para buscar en el WDASM.

Abro el WDASM y trato de abrir el ejecutable tbnote.exe y se cuelga la maquina hmmm, que cosa, podría ponerme a descomprimirlo pero voy a usar el PUPE para rápidamente hacer un archivo que pueda desensamblar con el WDASM.

Ejecuto el Turbonote y cuando ya arrancó abro el PUPE, con él elijo en la lista de procesos el que corresponde a TURBONOTE y hago clic derecho y elijo CAJA DE HERRAMIENTAS. Allí vuelvo a elegir entre los módulos el ejecutable TBNOTE y hago clic derecho para cargarlo y después voy a VOLCADOR TOTAL - ALINEAR y VOLCAR y guardo el ejecutable descomprimido que aunque no funcione me va a servir para ver en el WDASM.

Por supuesto antes tengo que usar el PROCDUMP para cambiar las características de las secciones y eliminar la protección Antidesensamblado como vimos en las lecciónes anteriores. Aquí hay que cambiar 60000020 por E0000020 y ya se desensambla con el WDASM.

Una vez que se desensambla miramos las STRING REFERENCES y si miramos en el listado del programa bajando encontramos con un poco de paciencia en **412d08 "THANKS FOR REGISTERING TURBONOTE+"**, lo cual es una punta para empezar.

Vemos que eso viene de una REFERENCED BY... que esta un poco más arriba que dice **412cb4**. Vamos allí con GOTO CODE LOCATION y vemos el salto que nos llevaría supuestamente a REGISTRARNOS pero si probamos poner un BPX en el SOFTICE, y ponemos una clave de registro, no para allí. Quiere esto decir que aquí no llega por alguna razón, sigamos hacia atrás.

El REFERENCED que esta arriba de donde nos encontramos dice **412c85**. Vamos allí y vemos también un salto que si no salta nos aparecería un cartelito que dice THERE IS A PROBLEM SAVING THE REGISTRY KEY, sigamos más hacia arriba, bastante más arriba en **412bf3** hay una comparación de **EAX** con **14** justo después de una función que se ve en azul **LSTRLENA** que es una función que te dice el largo de una STRING o sea que esto puede ser que verifica la cantidad de cifras de nuestra clave y si no es mayor de 14 hexadecimal o sea 20 decimal nos tira fuera de esta parte de registración.

Vamos al programa y probemos a poner una clave de 20 o más cifras y ahora al ACEPTAR dice **I'M SORRY THATS NOT A VALID REGISTRATION CODE.**

Bueno, ya logramos algo, saber que la clave tiene 20 o más cifras. Ahora podemos poner un **BPX** en **412c85** para invertir el salto a ver que

pasa con el SOFTICE y cuando invierto el salto me aparece el CARTEL de THANKS FOR REGISTERING pero no nos registra ya que sigue diciendo UNREGISTERED por todos lados, allí mismo en la ventana de registro y en ABOUT también.

Generalmente cuando ocurre esto es porque hay que mirar el CALL que esta antes del salto en 412c7e , entramos dentro del CALL poniendo en el SOFTICE un BPX en **412c7e** y cuando para allí entro con T y traceo por allí hasta que encuentro un salto, lo invierto a ver que pasa y voy probando los distintos saltos que encuentro a ver si alguno me registra, y también mirando dentro de los CALLS que hay dentro e invirtiendo los saltos. Al final al llegar a

44BC29 754E jne 44bc79

este es salto clave. Si lo invierto me aparece el cartelito de registrado, pero si dejo el BPX allí en 44bc29 veo que accede al salto en varios lugares del programa por lo que haremos un cargador con el RISC PROCESS PATCHER para que cargue el programa y modifique ese salto en la memoria así siempre estaremos registrados.

Tengo que hacer que en 44bc29 no realice el salto o sea que tengo que reemplazar los bytes originales **75** y **4e** por **90 90** para que pase de largo.

El SCRIPT del RISC ES ESTE:

```
;INTERNET DESK
F=tbnote.exe:
O=loaderturbonote.exe:
P=44bc29/75,4e/90,90:
$ ;end of script
```

Y listo, cargamos el programa con el LOADER y vamos a registro y ponemos un número de más de 20 cifras y ya esta. Lo que si hay que cargar siempre el programa desde el LOADER así que hay que borrar en INICIO – PROGRAMAS - INICIO la referencia a TURBONOTE para que no arranque cuando inicia WINDOWS y poner un acceso directo al LOADER si queremos que arranque, y borrar los accesos directos del escritorio al ejecutable y hacer accesos directos al LOADER, inclusive se puede cambiar el icono del ACCESO DIRECTO para que tenga el mismo icono que el TBNOTE para que quede más lindo.

PROGRAMA REGISTRADO

Ricardo Narvaja

LECCIÓN 24: EL MALDITO ASPROTECT

La verdad ya me tenia bastante podrido el ASPROTECT este, hace un par de meses me habían alcanzado un programa protegido con este compresor maldito y a pesar de encontrar un par de saltos que invirtiéndolos se REGISTRARIA el programa, no pude hacer nada.

PORQUE? Porque este compresor diseñado por el mismo que hizo el ASPACK es una fortaleza llevada al máximo de su expresión.

EJEMPLOS:

Protección ANTISOFTICE

Protección Antidesensamblado

Compresor no reconocido por la mayoría de los programas identificadores de compresores ya que este loco hizo de la misma versión miles de variaciones y existe una colección de distintos ASPROTECTS que hacen difícil su reconocimiento y varias versiones distintas con muchísimas modificaciones, es más, este programa yo lo pude crackear y todavía no se verdaderamente que versión de ASPROTECT es la que utiliza.

Si lo descomprimís manualmente como hicimos en la LECCIÓN sobre ese tema, te sale un ejecutable que convenientemente modificado (E0000020) puede ser desensamblado pero no funciona ya que destruye la tabla de importaciones y da error.

Bueno dirá alguno si no se puede descomprimir le aplicamos un LOADER como el RISC PROCESS PATCHER, craso error si haces eso, te aparece un ERROR DE PROTECCIÓN 15 y no arranca si lo arrancaste desde un LOADER, probé con otros PARCHEADORES como el PELG que dice que funciona en ASPROTECT y nada no lo detecta pero tampoco lo parchea.

Probé muchísimas herramientas que había por ahí y ninguna funcionaba, no quería caer en esos tutoriales complicados que leí sobre el tema que dicen que tenés que rescribir medio ejecutable o inyectar a mano la tabla de importación que es un lío, igual lo intente hacer y no me funciono el ejecutable resultante.

Así que hoy un tiempo después me encontré con otro engendro diabólico de ese ASPROTECT el programa ADVANCED DIRECT REMAILER versión 2.0, desde ya los cracks que hay en Internet a mi no me funcionaron, hay un LOADER por ahí que dicen que funciona pero a mi me sigue diciendo UNREGISTERED y eso no me gusta, no esta muy bien hecho.

Bueno manos a la obra después de tan extenso preámbulo para que conozcan al rival, es un rival de fuste quizás demasiado para mi verdaderamente y es probable que me noquee en el primer round pero quizás como DAVID Y GOLIAT este humilde cracker lo pueda hacer caer y vencerlo de alguna forma quizás más por insistencia que por conocimiento.

Lo primero es detectarlo, ni el unpack 2000 lo detecta, a este lo detecta el LANGUAJE 2000, el único que dice algo que te puede orientar.

Lo cargo con el LANGUAJE 2000 pongo ANALIZAR y me sale como resultado:

ASProtect/ASPack
Autor: Alexey Solodovnikov

No es mucho, no dice la versión ni nada y no distingue si es ASPack o ASProtect pero dado que ya vimos que los tipo ASPack los detecta el Unpack fácilmente y a este no, concluimos que es ASProtect y no se que versión será.

Vamos primero a ver, no se puede desensamblar, así que arranco el programa, arranco el PUPE y como vimos en las lecciónes anteriores pongo ALINEAR y hago un VOLCADO TOTAL, quito la protección Antidesensamblado (RECORDAR E0000020 de lecciónes anteriores) y aunque no funciona al ejecutarlo, se puede desensamblar con el WDASM.

Vamos al programa, hay una parte para registrarlo que hay que poner una clave, pongo una clave falsa a ver que dice, sale un cartel que se mueve para todos lados (QUE RARO NO?) que me dice...

THE CODE YOU'VE ENTERED IS INVALID

Bueno pongo cualquier clave y pongo un BPX HMEMCPY (EL SOFTICE MODIFICADO CON EL BACKDOOR KEEPER NO ES DETECTADO), y cuando hago ACEPTAR, caigo en el SOFTICE hago F12 hasta que vuelvo al ejecutable **ADR**, eso es en **430601**.

Voy a probar si falta mucho para llegar al cartelito de que el código es invalido, tengo apretada F10 hasta que salte el cartelito, una vez que aparece el cartel, pongo ACEPTAR y vuelvo al SOFTICE, a ver donde salió el cartelito, o sea la sentencia anterior a donde estoy.

(ACLARACIÓN EN EL SOFTICE NO CONVIENE TENER APRETADO MUCHO TIEMPO SEGUIDO SINO UN RATITO Y SOLTAR Y ASI, YA QUE SI UNO TIENE MUCHO TIEMPO APRETADO CUANDO SUELTA SIGUE EJECUTANDO SOLO MUCHAS SENTENCIAS Y NO PARA)

Bueno la sentencia anterior adonde estoy es

430836 CALL [0043c3c4]

puedo borrar con **BC*** los BPX anteriores y poner un BPX allí en ese CALL y volver al programa a la parte de REGISTRO para ver si para antes de que salga justo la ventanita maldita.

Pongo **BPX 430836** y hago X enter y vuelvo al programa voy a REGISTRO pongo otra clave falsa y cuando acepto veo la ventana agitarse y justo cuando va a salir el mensaje cae en el SOFTICE jeje.

Bueno un poco más arriba hay un RET y un poco más arriba un CALL un TEST y un salto que saltea justo ese RET lo que puede hacer que si no salta no llegue al cartel maldito y se desvié el programa en el RET hacia otro lado, probemos.

4307fA jz 430827

Pongamos un BPX en 4307fa y volvamos a repetir el proceso. Cuando el softice para allí en el salto, me indica JUMP o sea que va a saltar y tirarme a la zona del cartel maldito o sea que si hago **R EIP=4307fC** para que siga ejecutándose en la siguiente sentencia y no salte, hago X y ENTER y...

THANKS YOU FOR REGISTERING ADR

Quiere decir que vamos por el camino correcto aunque el programa no nos registra solo aparece el cartel pero sabemos que aquí esta la clave del asunto.

Bueno ya vimos que hay un CALL en 4307f0 que es el CALL comparador que devuelve EAX=0 si es la clave incorrecta y UNO si es correcta. Asi, si cambio **R EAX=1** antes de la sentencia TEST también va al cartel de REGISTRADO.

No ingreso a buscar la clave porque esta bastante encriptada y como soy vago, jaja, hago lo más fácil supuestamente.

Voy al WDASM y voy a GOTO CODE LOACION 4307f0 y veo que si ingreso dentro del CALL hay referencias de tres lugares distintos que llaman a este CALL posiblemente para ver si estas bien REGISTRADO. Podríamos en vez de parchear todos los saltos que hay en esos sitios, parchear directamente en algún lado dentro de este CALL para que siempre me devuelva UNO sea lo que sea lo que compare, y que mejor que justo cuando finaliza el CALL justo antes del RET agregar algunas cosillas para que de aquí salga siendo siempre EAX=1 y en cualquier lado que llame a este CALL nos diga que estamos registrados.

RECORDAR QUE SI **EAX=1 al salir del CALL significa REGISTRADOS**, vemos que el RET donde termina este CALL esta en **4303EA RET** y que después hay varios NOPs (CINCO EXACTAMENTE) que bueno lugar vacío, jajaja.

Puedo entonces allí que hay lugar escribir...

4303EA ADD eax, 01
4303EF RET

LO QUE CABE JUSTO YA QUE OCUPA LOS SEIS LUGARES QUE HABIA Y LE SUMA UNO A EAX Y DESPUES HACE EL RET, O SEA QUE SI EAX AQUÍ LLEGO SIENDO CERO, LE SUMO UNO Y SIEMPRE VALDRA UNO AL SALIR DEL CALL, LO LEA DESDE DONDE LO LEA.

O SEA QUE SI EL PROGRAMA SE PUDIERA PARCHEAR CON EL ULTRAEDIT CON BUSCAR LA CADENA C39090909090 Y REEMPLAZARLA POR 0501000000C3 ESTARIA RESUELTO EL CRACK.

PERO COMO DEMONIOS PODEMOS MODIFICAR ESTE EJECUTABLE SI ESTA COMPRIMIDO, Y NO PODEMOS MODIFICARLO TAMPOCO DESDE UN LOADER COMO EL RISC PROCESS PATCHER O SIMILAR.

Estamos a 27 de febrero del 2001 cuando yo escribo este tutorial, ya mencione a ese RUSO que escribió el ASProtect se ve que es bastante bueno en esto, pero como el es bueno y a mi me derroto, hay otros programadores que también son buenos en hacer herramientas para batir al ASProtect, y esta que uso yo, salió solo hace 5 días en la versión que me funciono y descomprimió bastante bien el ASPROTECT. Es el **CASPR**

1.00 lo busque y lo encontré entre muchísimos que probé, es un utilitario muy simple.

Aquí transcribo lo que dice el README del CASPR.

```
ASProtect is a very powerful win32 protector.

It has compression, encryption, anti-debugging code, anti-
disassembler code, crc checking, anti-dumping, etc.

Features. It uses also strong cryptographic algorithms like
Blowfish, Twofish, TEA, etc, and also uses RSA1024 - very strong
public keys cryptographic algorithm - as registration keys
generation.

It communicates with the application through API hooks including
import hooks(GPA hook) and export hooks.

Furthermore, since v1.1 ASProtect uses the Win95 Marburg's
polymorphic engine. ASProtect v1.11c was added anti-Apihook
code(stolen from Win32.Crypto) and the BPE32's polymorphic
engine.

The big question: Is it a super virus or a commercial protector?

CASPR is a cool unpacker for ASProducts. It is designed for
experienced crackers, not for newbies. Sorry, its source code
can't be provided.

II. Features :
^^^^^^^^^^^^^^
  û  Support for ASProtect v0.95b - v1.2x and ASPack v1.00b -
v2.11x
  û  Own decryption and decompression code
  û  Import table rebuilding engine
  û  Export table rebuilding engine
  û  Resources recovering engine
  û  x86 CPU emulator
  û  Win32(Win9x/Me/NT/2k/XP) compatible

III. Usage :
^^^^^^^^^^^^
  CASPR has no GUI, you must run it under command line:

         CASPR [-|/L] <input> [[-|/P] output]
```

Bueno ya vemos allí todo lo que tiene el ASProtect y después las versiones que soporta, inclusive soporta varias versiones de ASPACK.

Es muy fácil de usar. Hay que copiar el ejecutable a descomprimir al directorio del CASPR entrar en MODO DOS y allí teclear

CASPR ADR.exe

y listo sale esto en la pantalla de DOS

CASPR v1.000 - A cool unpacker for ASProducts

Copyright (c) 2000,2001 Coded by SAC/UG2001! aLL rIGHTS rEVERSED!

Unpacking Adr.exe...OK!

Don't forget to see readme.txt. It's helpful to reverse ASProducts.

Y listo me hizo un archivito llamado ADR.ex que si lo renombro ya esta limpiado de toda la protección ASProtect y funciona solo hay un detalle que arreglar. El programa arranca y sale una messagebox que dice **CRYPT API NOT FOUND,** pero es una messagebox colocada allí por si alguien llega a descomprimir el ejecutable. Hay dos de estas, una cuando comienza el programa y si parcheamos esa el programa arranca, la otra esta cuando entro a GENERAL SETTINGS.

No hay problema con el SOFTICE ponemos **BPX MESSAGEBOXA** y vemos que en la primera después de aceptar y volver con F12 vemos que el CALL al MESSAGEBOXA esta en **42f660** y el salto para que pase por encima esta en

42f651 7516 jnz 42f669

Con el ULTRA EDIT cambiamos este salto por un JMP o sea el **75** por un **EB** ahora lo podemos hacer porque el programa esta descomprimido.

Y ARRANCA perfectamente y si le hacemos el PARCHE en el CALL que habíamos estudiado en **4303ea** arranca registrado perfectamente.

Queda solamente con el mismo mecanismo que con el primer cartelito entrar a GENERAL SETTINGS y cuando rompe en el MESSAGEBOXA parchear el salto que esquiva el CALL. Este esta en **425c2b,** y lo reemplazamos por un **JMP 425cb6,** lo editamos con el ULTRA EDIT y funciona perfecto.

Obviamente pudimos con él gracias a dos cosas **EL CASPR** y la persistencia en buscar una solución que siempre tenemos que tener los crackers, ser tozudos, buscar y buscar y buscar que de alguna forma se va a abrir y va a poder ser crackeado.

PUUFFF

CAYO GOLIAT.
Ricardo Narvaja

LECCIÓN 25: SORPRESA CON EL ADR 2.0

Bueno como vimos en la LECCIÓN 24 ya parecía que el ADR 2,0 estaba completamente crackeado, pero nos tenia deparada una sorpresa, a pesar de que decía REGISTRADO por todos lados y que parecía que estaba 100% funcional, no era así, ya que a los quince días de usarlo cuando mandabas un mail por el OUTLOOK EXPRESS o cualquier programa similar si tenias configurado el ADR para trabajar mandando los mails, aparecía un cartel de que estaba vencido que ya habían pasado los quince días de prueba y que tenias que comprarlo para seguir usándolo y se cerraba el programa sin enviar el mail.

Si lo abrías de nuevo, ahí recién enviaba el mail anterior pero si enviabas uno nuevo, de nuevo el cartelito molesto y el programa se cerraba de nuevo, tornándose molesto.

Bueno crackearlo no fue nada fácil y hubo que improvisar y salirse del método ortodoxo de crackeo (CUAL SERA NO?) pero bueno lo pudimos crackear y costo bastante, vamos a aprender otra forma de encontrar el salto para parchear, el método es un poco manual y de prueba y error pero funciono y eso es lo que importa. (TODO SE VALE EN LA GUERRA Y EN LOS CRACKS, jaja).

Veamos el cartel dichoso, no es una messageboxa, puesto que es más complejo que un simple cartelito de mensaje, tiene varios botones para conectarse para registrarse, poner la clave, etc, tiene como cuatro o cinco botones no recuerdo bien, así que messageboxa no es. Podría ser una dialogboxparama , puesto que este tipo de cartelitos pueden ser un poco más complejos, que mejor que utilizar el softice para comprobarlo.

Ponemos **BPX DIALOGBOXPARAMA** y cuando aparece el cartelito caemos en el softice, lo que quiere decir que si es DIALOGBOXPARAMA.

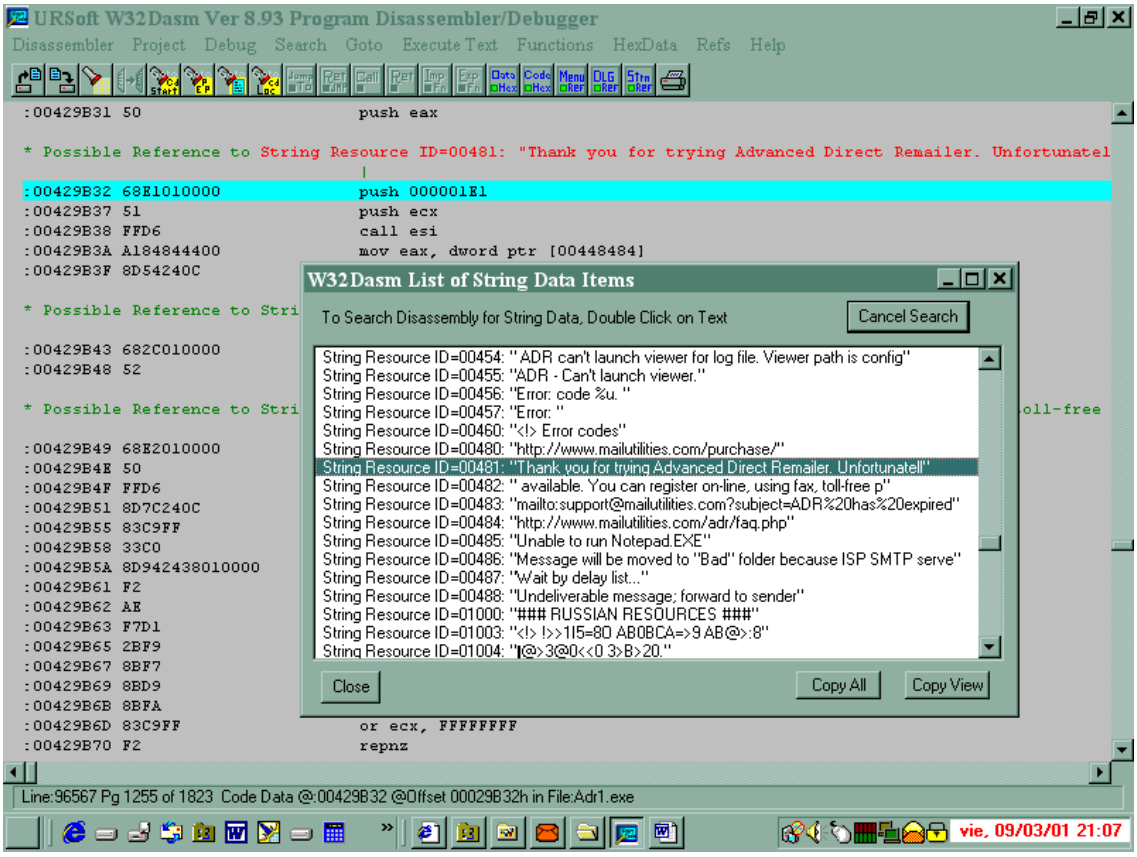
Igual vemos que cuando aparece el cartelito ya desapareció la flecha del ADR en la barra de tareas así que el programa va camino a cerrarse, y cuando hacemos F12 en la ventana se cierra el programa después de unas poquitas sentencias, lo que hace que no podamos llegar al RET y terminar el CALL y volver al programa principal. Lo que ocurre es que cuando detecta que esta vencido toma un camino el programa sin retorno, sale la ventana y después se cierra así que tenemos que ir hacia atrás en el programa hasta donde todavía este la flechita en la BARRA DE TAREAS y tome la decisión de que aparezca la ventana o que siga funcionando normalmente.

Esto es hacia atrás en el programa, y parece que bastante, veremos. Ya que el programa esta descomprimido podemos ver si la STRING que aparece en la ventana aparece en el WDASM dentro de las STRING REFERENCE, y allí está, la STRING **"THANK YOU FOR TRYING ADVANCED DIRECT REMAILER- UNFORTUNATE...** se te venció, jaja", y chau, se cerro el programa.

Si hago clic encima de esa STRING aparezco donde muestra la imagen, en rojo se ve exactamente donde el programa utiliza la STRING esa.

Si subimos un poco en el WDASM encontramos el REFERENCED de donde viene el programa, es la dirección siguiente:

```
* Referenced by a CALL at Address:
|:00429CD9
|
:00429B10   8B0D84844400     mov ecx, dword ptr [00448484]
:00429B16   81EC84030000     sub esp, 00000384
:00429B1C   8D84242C010000   lea eax, dword ptr [esp+0000012C]
:00429B23   53               push ebx
:00429B24   56               push esi
:00429B25   8B358CC34300     mov esi, dword ptr [0043C38C]
:00429B2B   57               push edi
:00429B2C   6858020000       push 00000258
:00429B31   50               push eax

Possible Reference to String Resource ID=00481: "Thank you for trying
Advanced Direct Remailer. Unfortunatell"
```



O sea, **viene de 429CD9.**

Vamos a GOTO CODE LOCATION 429CD9 y vemos las siguientes líneas:

```
:00429CD9   E832FEFFFF   call 00429B10
:00429CDE   83C40C       add esp, 0000000C
:00429CE1   C21000       ret 0010
```

Vemos que allí no hay ningún salto antes de ese CALL entonces seguimos hasta el REFERENCED BY A CALL que esta un poco más arriba

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

00429C9E

Vamos a GOTO CODE LOCATION 429C9E y hay un salto

:00429C9E 742A je 00429CCA

Si ponemos en el SOFTICE un **BPX 429C9E** Y CUANDO MANDAMOS UN EMAIL DESDE EL OUTLOOK EXPRESS NOS CAE DENTRO DEL SOFTICE Y PARA JUSTO ANTES DE QUE APAREZCA LA VENTANA, Y ENCIMA PARA VARIAS VECES EN EL MISMO SALTO ANTES DE QUE SALGA LA VENTANA MALDITA, PERO IGUAL NOS DAMOS CUENTA QUE EL SALTO ESTE NO NOS SIRVE YA QUE CUANDO PARA POR PRIMERA VEZ ALLI, SI MIRAMOS CON F4 COMO ESTA LA PANTALLA DE WINDOWS, VEMOS QUE YA DESAPARECIO LA FLECHITA EN LA BARRA DE TAREAS, Y NOSOSTROS TENEMOS QUE BUSCAR UN SALTO QUE CUANDO PARE ALLI TODAVÍA LA FLECHITA ESTE Y NO HAYA DESAPARECIDO, POR LO TANTO DEBEMOS IR TODAVÍA MÁS ATRÁS EN EL PROGRAMA, VOLVEMOS AL SOFTICE CON F4 NUEVAMENTE Y VOLVEMOS AL WDASM.

Miramos en el Wdasm un poco más arriba del salto anterior y vemos otro REFERENCED...

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:00429C0C(C)
|
:00429C71 C3 ret

Pero si el programa viniese de allí encontraría como siguiente sentencia un RET y no llegaría al salto, estamos perdidos, el programa desensamblado no nos dice de donde viene y no podemos seguir hacia atrás.

COMO PUEDE SER ESTO? Es muy fácil el programa tiene una sentencia que es la que lleva a esos NOP que están arriba del salto pero el WDASM no muestra una REFERENCED..... ya que puede ser una sentencia tipo **CALL eip**, o alguna sentencia en la cual no esta determinado el valor, por lo tanto cuando encuentra esta sentencia el WDASM no puede crear una REFERENCE ya que no sabe cuanto va a valer EIP, eso solo se puede saber ejecutando el programa o con el SOFTICE el WDASM no sabe los valores de los registros.

O sea que para crear una REFERENCE el WDASM debería encontrar una sentencia tipo **CALL 45890** o **JMP 34789**, en la cual esta determinado a donde va a ir el programa.

En cambio las sentencias **CALL EIP**, **CALL (ESI+EDI)**, **JMP EAX** o sentencias donde no sabemos cuanto vale EIP, ESI+EDI o EAX, no podemos saber donde va a ir salvo que pongamos un BPX en el softice y al parar en esas sentencias veamos el valor que no conocemos (estas sentencias indeterminadas son muy usadas para despistar crackers).

Por lo tanto, si con el WDASM no podemos ir más atrás... como podemos hacer?

Aquí usaremos métodos menos ortodoxos pero que pueden servir.

Tenemos que poner un **BPX 429C9E** y cuando para allí, utilizamos un comando del SOFTICE muy importante para vencer estos truquitos que no nos dejan ver lo que pasa hacia atrás en el programa.

BPR INICIO FINAL T

O sea INICIO Y FINAL deberían ser las direcciones de inicio y final del programa pero si uno sabe más o menos en que valores esta trabajando el programa, se puede acotar a un valor menor. En este caso yo puse **BPR 420000 430000 T.**

Lo que hace la letra T que le agregamos al BPR es que el SOFTICE memoriza las sentencias anteriores al BPX en donde pare o sea que si esta activo

BPR 420000 430000 T y **BPX 429C9E**

cuando pare en este último BPX tendremos en la memoria de la maquina las sentencias anteriores que vino ejecutando el programa una por una.

Una vez que para en el BPX tecleo **SHOW 1** y me va a mostrar la sentencia exacta donde estoy o sea **429c9e** y si presiono la flecha de subir voy a ver las sentencias que ejecuto el programa antes. Puedo subir y bajar con las flechas y recorrer estas sentencias.

Por supuesto al softice no le alcanza la memoria para guardar todas las sentencias desde que se inicia el programa pues no tiene tanta memoria pero guarda muchísimas y si las que guardo no alcanzan se pone un BPX en la primera de todas y se vuelve a repetir el proceso para ir más atrás en la lista de sentencias que ejecuto el programa.

Ahora aquí hay varias posibilidades; es obvio que hay que anotar en un papel estas sentencias ya que no se pueden imprimir, hay crackers que anotan todas las sentencias (MUYYY LARGO YA QUE SON MUCHAS), hay crackers que anotan solo los saltos condicionales por donde paso el programa (BUENA OPCION Y MÁS BREVE), hay quien anota los saltos y los CALLS y hay quien como yo anota una sentencia cada tanto, las que ve interesantes.

Estas son las anotaciones de puntos interesantes por donde pasa el programa antes de llegar al BPX ese, otra persona puede llegar a anotar muchas más o menos sentencias cada cual vera.

429C97, 429D15, 429D0C, 422460, 42C5C4, 42BEB7, 42BE80, 42BE7A, 42C553, **42F2AC ← AQUI ENCONTRE LO INTERESANTE**, 42C3F1, 42C7BC

Además se observa que el programa al ir hacia atrás las primeras sentencias son casi un ciclo o sea que se repiten varias veces la misma secuencia una vez que vamos más atrás y salimos de ese ciclo repetitivo , encontramos lo interesante.

Ahora el tema es poner un BPX antes de que desaparezca la flechita del ADR de la Barra de tareas y probando en los valores que anote veo que en **42f2AC** que si pongo un BPX para allí antes de que desaparezca la flecha y esta sentencia es un CALL

:0042F2AC FF1580C24300 call dword ptr [0043C280]

cuando ejecuto ese CALL desaparece justo la flechita del ADR o sea que por aquí esta la cosa, ahora la cuestión es buscar un salto que evite

llegar a este CALL ya que una vez que el programa ingresa allí, KAPUTTT.

Vemos que más arriba hay un REFERENCE que nos dice que puede venir el programa desde **414c33** o de **42f264.**

Ponemos un BPX en ambas direcciones de memoria para ver desde cual de las dos viene el programa, mandamos un email y para en 414c33, pum caímos en el softice.

Ya casi llegamos probamos a invertir el salto que esta un poquito más arriba en **414bf7** y no sirve, ya casi llegamos uff.

Hay más arriba un REFERENCED que viene de varios lugares

*** Referenced by a CALL at Addresses:**
|:00415B08 , :00415CE5 , :00415DC4 , :00415E21 , :00416158
|:004161CE

Ponemos un BPX en cada uno de ellos para ver de donde viene el programa y mando otro email (POBRE AL QUE SE LO ESTE MANDANDO YA LO LLENE DE MAILS)y para en **416158.**

Bueno vamos allí a ver que hay y vemos esta parte:

```
:00416147  7515      jne 0041615E  (SALTO A PARCHEAR)
:00416149  8B442404  move exam, word per [esp+04]

* Possible Reference to Menu: MenuID_0001
|
* Possible Reference to Dialog: DialogID_0001
|
* Possible Reference to String Resource ID=00001: "Advanced Direct
Retailer"
|
:0041614D  C705186A440001000000  move word per [00446A18], 00000001
:00416157  50                    push exam
:00416158  E893EAFFFF            call 00414BF0
```

Habíamos caído en **416158** y justo arriba hay un salto, así que adivinen lo invierto y SORPRESA el programa saltea al call maldito manda el mail perfectamente y no sale cartelito alguno ni falla ni se cierra, jeje.

ESPERO QUE NO TENGA ESTE PROGRAMA NINGUNA OTRA SORPRESITA MÁS ASI QUE POR AHORA PROGRAMA CRACKEADO, NOS VEMOS EN LA LECCIÓN 26.

Ricardo Narvaja

Lección 26: WINALUM 2.0.1

Bueno después del ADR (ufff) vamos a un crack en DELPHI, el Winalum 2.0.1 es un programa para gente que hace trabajos en aluminio, ventanas, marcos, etc.

El programa se baja de http://www.coproinf.com/winalum.htm y viene en tres archivos (HAY QUE BAJAR LOS TRES) Winalum.exe, Winalum.r00 y Winalum.r01 y al ejecutar el EXE se descomprimen.

Primero de todo lo desensamblamos con el WDASM ya que no esta comprimido, no hay problema, no tiene protección anti desensamblado ni anti softice, ni nada, que bueno.

Lo desensamblamos y nos damos cuenta que esta hecho en DELPHI, ya que entre las STRING REFERENCES aparece una referencia a SOFTWARE\BORLAND\DATABASE ENGINE a veces puede decir SOFWARE\BORLAND\LOCALES o BORLAND solo, la mejor prueba que podemos hacer ya que esta descomprimido, es cargarlo con el DEDE, el cual ya va por la versión 2.50 actualmente, para no dar tantas vueltas.

Abrimos el DEDE y cargamos el ejecutable del WINALUM y lo carga por lo tanto esta en DELPHI, además buscamos aquí ya que la STRING REFERENCE que aparece cuando pones una clave falsa no aparece por ningún lado en el WDASM.

Una vez que desde el DEDE se arranca el programa, cerramos el mismo haciendo clic en la ventanita que dice que hagamos clic cuando el programa esta completamente funcionando y vamos al DEDE, a PROCEDURES y allí buscamos algo que tenga que ver con REGISTRO y encontramos **UedtRegistrePrg** que parece tener que ver.

Marcamos y vamos a EVENTS y allí aparecen tres botones que seguro corresponden a los que tiene la ventana el primero que hay en la ventana es para ACEPTAR la clave (Dácord), el segundo es DEMOSTRACIÓN y el tercero es CANCELAR, todo eso lo podemos ver haciendo clic derecho encima de los botones y eligiendo SHOW ADITIONAL DATA lo que nos muestra en CAPTION el texto del botón.

Por supuesto, vamos al de ACEPTAR y hacemos clic derecho y elegimos DESENSAMBLAR (DISSASEMBLE) y allí nos aparece la parte del programa que se ejecuta cuando hacemos clic en ACEPTAR cuando ponemos una clave falsa.

Si ponemos un BPX en el SOFTICE donde comienza la rutina en **593700,** (podemos tener que poner un BPX HMEMCPY primero y cuando entra en el SOFTICE volver al ejecutable y allí borrar el BPX HMEMCPY con BC* y poner el BPX 593700), y entonces ahora si vuelvo a la ventana de registro pongo un número de varias cifras y hago clic en ACEPTAR y para en 593700

```
:00593758  A130246000  mov eax, dword ptr [00602430]
:0059375D  DC18        fcomp qword ptr [eax]
:0059375F  DFE0        fstsw ax
:00593761  9E          sahf
:00593762  741A        je 0059377E       (SALTO a invertir)
:00593764  8D55F4      lea edx, dword ptr [ebp-0C]
```

```
:00593767  A14C1C6000  mov eax, dword ptr [00601C4C]
:0059376C  E84B35E7FF  call 00406CBC
:00593771  8B45F4      mov eax, dword ptr [ebp-0C]
:00593774  E89378EAFF  call 0043B00C      (cartel de chico malo)
```

Si vamos ejecutando con F10 vemos que llegamos al CALL que esta en **593774** y al pasar por encima de ese CALL sale el cartelito de que la clave no es correcta (chico malo), y justo antes esta el salto que puede pasar por encima ese CALL y si probamos invirtiéndolo vemos que se registra, el problema es que cuando arranca el programa de nuevo se vuelve a desregistrar, así que debe haber algún chequeo al comienzo del programa, por lo pronto ya tenemos el primer salto a modificar, para registrarse

```
:00593762  741A  je 0059377E
```

cambiar a

```
:00593762  EB1A  jmp 0059377E
```

con lo que el programa acepta cualquier clave.

Ahora veamos la verificación al principio del programa, seguro que el CALL anterior al salto es el responsable del chequeo de la clave, asi que veamos de donde es llamado es CALL para ver si desde alguna otra parte del programa es utilizado.

Si quieren divertirse vean desde que otros lugares entra a ese CALL

```
* Referenced by a CALL at Addresses:
|:0042648B   , :004A5E2D   , :004BE17A   , :004E38A7   , :004E86B3
|:004E86E6   , :004E8D9C   , :004E8DCF   , :004F2098   , :004F20CB
|:004F2135   , :004F2168   , :004F2198   , :004F21CB   , :004F21FE
|:004F23C7   , :004F2452   , :004F24DD   , :004F2568   , :004F25F3
|:004F267E   , :004F2900   , :00516404   , :0051649F   , :005164C3
|:0052F513   , :0052F51F   , :005334F9   , :005336A2   , :005336D9
|:00533710   , :00533744   , :00533778   , :00536DE8   , :00536E74
|:00536EEE   , :00537007   , :00537059   , :005370B8   , :00537117
|:00537176   , :00537DE9   , :00537E0E   , :00537E3B   , :00537E60
|:0053804F   , :0053806C   , :0053B0CF   , :0053B0F0   , :0053B111
|:0053B45A   , :0053B47B   , :0053B49C   , :0053B4BD   , :0053B4DE
|:0053CAC6   , :0053CBBF   , :0053CC98   , :0053CE14   , :0053CE6C
|:0053CED7   , :0053CF42   , :0053CFAD   , :0053D47D   , :0053D4A2
|:0053D4C7   , :0053D4EC   , :0053D511   , :0053D649   , :0053D66E
|:0053D693   , :0053D8D0   , :0053D8F5   , :0053D922   , :0053D947
|:0053D9D0   , :0053D9F5   , :0053DA22   , :0053DA47   , :0053E076
|:0053E0A2   , :0053E0CE   , :0053E0FA   , :0053E126   , :0053E2BE
|:0053E31D   , :0053E340   , :0053E363   , :0053FEEC   , :0053FF0F
|:00540057   , :005400FB   , :005401AF   , :0054020A   , :0054024B
|:00540336   , :005403F6   , :0054129C   , :005412BF   , :00541407
|:005414AB   , :0054155F   , :005415BA   , :005415FB   , :005416E6
|:005417C2   , :005424B3   , :00542557   , :0054260B   , :00542666
|:005426A7   , :0054302B   , :005430CF   , :00543183   , :005431DE
|:0054321F   , :00543367   , :00543380   , :005433A3   , :005433BC
|:00543559   , :005435A9   , :00543621   , :00543647   , :005437A8
|:005437DA   , :0054381A   , :005438C4   , :005439A6   , :005439FF
|:00543A54   , :00543B25   , :00543B51   , :00543BA6   , :0054554E
|:00545580   , :005455D9   , :00545656   , :00545743   , :00546009
```

```
|:0054607A   , :005460EB   , :0054615C   , :005461CD   , :005462EB
|:00546349   , :0054637B   , :0054660A   , :0054666E   , :00546718
|:005467FC   , :0054682E   , :0054689E   , :005468D0   , :005471CB
|:005471FD   , :00547256   , :005472D3   , :005473C0   , :00547BC1
|:00547C25   , :00547C96   , :00547D07   , :00547D78   , :00547E22
|:00547F06   , :00547F38   , :00547FA8   , :00547FDA   , :00548325
|:00548389   , :00548433   , :00548517   , :00548549   , :005485B9
|:005485EB   , :0054949C   , :005494FC   , :0054956F   , :005495CA
|:0054960B   , :0055543A   , :0055549E   , :0055550F   , :00555580
|:005555F1   , :005556D4   , :00555718   , :00555783   , :005557EE
|:0055586E   , :00555968   , :0055599A   , :005559CC   , :005559FE
|:00555A30   , :00555E23   , :00555E87   , :00555F6A   , :00555FB0
|:0055601B   , :00556086   , :00556106   , :005610E9   , :0056114C
|:005611F5   , :0056121D   , :00561351   , :0056137C   , :005613A7
|:00561438   , :00561476   , :005614B4   , :0056155C   , :0056159F
|:0056177D   , :005617B3   , :00561C46   , :00561D3F   , :00561E18
|:00561F94   , :00561FEC   , :00562057   , :005620C2   , :0056212D
|:005778BB   , :00577913   , :0057904C   , :005790A4   , :00579109
|:0057916E   , :005791D3   , :00579373   , :0057939F   , :005793CB
|:005793F7   , :00579423   , :005797F6   , :0057985A   , :00579F22
|:00579F4F   , :00579F7C   , :00579FA9   , :00579FDC   , :0057A34D
|:0057A3E1   , :0057A54F   , :0057BDD7   , :0057BE04   , :0057BE31
|:0057BE64   , :0057BE97   , :0057C0DA   , :0057C425   , :0057C56F
|:0057C60B   , :0057C7AE   , :0057EEDD   , :0057EFD6   , :0057F0AF
|:0057F22B   , :0057F27D   , :0057F2E8   , :0057F353   , :0057F3BE
|:005807A7   , :00583A59   , :00583A7D   , :00583AA1   , :00583B46
|:005847FD   , :00584875   , :005848ED   , :00584965   , :00584AD5
|:00584B23   , :00584B64   , :00585309   , :00585381   , :005853F9
|:00585471   , :005854F9   , :00585547   , :00585588   , :00585D1F
|:00585D7A   , :00585DBB   , :005873CC   , :0058742C   , :0058748C
|:00588D45   , :00588D69   , :00588D8D   , :00588E32   , :00593753
|:005AAC1C   , :005AAC7C   , :005AACDC   , :005B789D   , :005B7915
|:005B798D   , :005B7A05   , :005B7A8D   , :005B7ADB   , :005B7B1C
|:005B82E5   , :005B8334   , :005B8A89   , :005B8B01   , :005B8B79
|:005B8BF1   , :005B907D   , :005B90CC   , :005D3CCC   , :005DF594
|:005DF904   , :005E1D4D   , :005E1DDB   , :005E1E69   , :005E1EF7
|:005E1F9A   , :005E2022   , :005E20AA   , :005E2132   , :005E277A
|:005E27E7   , :005E2A40   , :005E2AAD   , :005E613C   , :005E6150
|:005E65C1
```

Creo que si ponemos un BPX en cada uno nos morimos de aburrimiento por lo tanto hagamos otra cosa

Este es el CALL que estamos investigando entero desde que comienza hasta el RET

```
:0040B5F8  push ebx
:0040B5F9  add  esp, FFFFFFEC
:0040B5FC  mov  ebx, eax
:0040B5FE  mov  eax, ebx
:0040B600  call 00404414
:0040B605  mov  edx, esp
:0040B607  xor  ecx, ecx
:0040B609  call 0040FC90
:0040B60E  test al, al
:0040B610  jne  0040B62B
:0040B612  mov  dword ptr [esp+0C], ebx
:0040B616  mov  [esp+10], 0B
```

```
:0040B61B  lea  edx, dword ptr [esp+0C]
:0040B61F  mov  eax, dword ptr [00601AB8]
:0040B624  xor  ecx, ecx
:0040B626  call 00409CA8

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040B610(C)
|
:0040B62B  fld  tbyte ptr [esp]
:0040B62E  add  esp, 00000014
:0040B631  pop  ebx
:0040B632  ret
```

Bueno lo transcribimos todo pero lo único que vamos a hacer es poner un BPX en el RET para ver cuando el programa llama a este CALL y sale de aquí a donde va.

Pongo un **BPX en 40b632** y arranco el programa de nuevo, la primera vez que para allí, salgo del RET con F10 y no veo ningún salto condicional que puede decidir desregistrar o no el programa, así que hago X y vuelve a parar en el RET, repetimos esto hasta que al salir del RET aparecemos en **5E6155**

```
:005E6150  call  0040B5F8
:005E6155  mov   eax, dword ptr [00602430] (al salir del RET caemos aqui)
:005E615A  fcomp qword ptr [eax]
:005E615C  fstsw ax
:005E615E  sahf
:005E615F  jne   005E618D (salto importante)
```

que es una secuencia muy parecida a la que había en donde parcheamos el primer salto. Las sentencias son casi idénticas, y tenemos un salto condicional en **5E615f.** Llegamos hasta él haciendo F10 y vemos que aquí va a saltar (JUMP) o sea que tenemos que invertirlo para que no salte.

R eip= 5e6161

Si quitamos todos los BPX con BC* el programa arranca normalmente, y si no lo habías registrado pones cualquier clave la ACEPTAS te registra y al volver a arrancar no te desregistra más ya que parcheamos la comprobación al principio del programa.

:005E615F jne 005E618D (salto importante)

cambiar a

```
:005E615F  Nop
:005E6160  Nop
```

O bien

:005E615F 752C por **9090** que es lo mismo.

O sea parcheando los dos saltos el de **593762** y el de **5e615f** PROGRAMA CRACKEADO

Hasta la 27
Ricardo Narvaja

LECCIÓN 27: UN ESPIA EN TU COMPUTADORA

Se trata del programa STARR 007 versión 2.0 que se baja de http://www.iopus.com/starr.htm

Es un verdadero espía que lo podés configurar para que vea todo lo que se tipea, claves, programas, usos, y lo mejor de que cuando esta funcionando el monitor espía, si haces CTRL+ALT+DEL no aparece en la lista de procesos activos , ni allí ni con ningún programa que muestre la lista de procesos (PROCDUMP, PUPE, ETC), o sea que es casi imposible que la persona que usa la computadora sepa que el programa esta monitoreando todo lo que se hace, cuando uno accede a la computadora pide el REPORTE y te saca un reporte detallado de todo lo que se hizo en la misma.

ES IDEAL PARA COMPUTADORAS QUE COMPARTEN VARIAS PERSONAS Y UNO QUIERE SABER QUE HACEN LOS OTROS CON LA COMPUTADORA.

Bueno basta de promoción una vez instalado el programa vamos a la pestaña REGISTER y ponemos una clave falsa y al hacer clic en REGISTER sale allí mismo el cartel **WRONG REGISTRATION CODE ENTERED - PLEASE TRY AGAIN.**

Bueno abrimos el ejecutable con el WDASM y lo desensambla perfectamente, igual conviene trabajar con una copia del ejecutable que este en otro lugar, por si acaso.

Allí en STRINGS REFERENCES aparece **WRONG REGISTRATION CODE ENTERED.** Si vamos subiendo vemos diversos carteles que aparecen y algunas REFERENCED... pero que son referidos a saltos apenas más arriba. Seguimos más arriba y vemos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
:004195D0
```

y si vemos ese salto con GOTO CODE LOCATION **4195d0** nos damos cuenta que ese salto en 4195d0 es el que decide si te registra o no.

```
:004195D0 0F8EAB000000 jle 00419681
:004195D6 6A40         push 00000040

* Possible StringData Ref from Data Obj ->"Registration"
|
:004195D8 68680A4500   push 00450A68

* Possible StringData Ref from Data Obj ->"Your 007 STARR..."
                                        ->"successful - Thank you ! "
```

O sea que si no salta en **4195d0** lo que nos lleva a la zona de CHICO MALO, sigue aquí y nos lleva a **YOUR 007 STARR REGISTRATION WAS SUCESSFULL** o sea que estaríamos registrados, si invertimos este salto con el SOFTICE vemos que nos aparece el cartel de REGISTRADO y todo, pero cuando iniciamos el programa nos desregistra de nuevo.

Analicemos el CALL antes del salto donde se realiza la comparación generalmente este esta justo antes del salto

:004195B5 E8FC550000 call 0041EBB6

Según el resultado de este CALL salta o no, el tema es que vemos que según el resultado que tiene EAX al salir de este CALL salta o no salta, además también vemos que este CALL es llamado desde varios lugares así que en vez de parchear todos los lugares desde donde es llamado el CALL podemos modificar el final de esa rutina para que siempre nos de al salir de ella el valor de EAX que hace que estés registrado, lo llame de donde lo llame.

Si entramos al CALL en el WDASM, posicionándonos encima y haciendo clic en el icono CALL entramos dentro del CALL, bajamos bastante hasta el RET donde termina ese CALL y este esta en

:0041EF28 C3 ret
:0041EF29 CC int 03
:0041EF2A CC int 03
:0041EF2B CC int 03
:0041EF2C CC int 03
:0041EF2D CC int 03
:0041EF2E CC int 03
:0041EF2F CC int 03

Además después del RET vemos algunas posiciones de memoria que no se usan donde podemos escribir las modificaciones que queramos.

Si probamos que dándole a EAX el valor de UNO al salir del CALL, el programa se registra, entonces lo modificamos y quedaría de esta forma

:0041EF28 B801000000 mov eax, 00000001
:0041EF2D C3 ret

Movemos a EAX el valor uno y después ejecutamos el RET, esto funcionaria supuestamente mejor que parchear el salto solamente pues aquí al modificar la rutina de comparación en cualquier punto del programa que se le ocurra chequear si es correcta la registración al llamar a este CALL le va a devolver **EAX=1** o sea registrado SIEMPRE.

Realizamos esta modificación con el ULTRAEDIT, y arrancamos el programa nuevamente y ponemos una clave trucha y la acepta y si lo cerramos y volvemos a iniciar nos sigue diciendo que estamos REGISTRADOS, lo cual significa que lo vencimos.

Igual queda un detalle en otro archivo, en el archivo ESPIA llamado **wsys.exe** que esta en la misma carpeta, cuando arrancamos el MONITOR nos sale un cartelón diciendo que nos registremos, y recién después arranca el monitor espía, evidentemente esto fue puesto para que uno si no modifica esto, no le sirva para nada el programa, porque de que me sirve un espía que me avisa con un cartelón que esta comenzando el trabajo de espía... hay que quitarlo!!!

Es fácil ya que el cartel es un **dialogboxparama** así que con el SOFTICE ponemos un BPX DIALOGBOXPARAMA y cuando aparece el cartel para allí, y cuando vuelve nos muestra donde esta en el programa ese cartelón.

En el WDASM vemos esa parte

:00412C8A **Cmp dword ptr [0043228C], 01**

```
:00412C94  jz   00412CA8
:00412C96  mov  al, byte ptr [6A004128]
:00412C9B  add  byte ptr [edx+74], ch
:00412C9E  mov  edx, dword ptr [ebp+08]
:00412CA1  push edx

* Reference To: USER32.DialogBoxParamA, Ord:0093h
|
:00412CA2  Call dword ptr [004273A8]

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00412C88(C), :00412C94(U)
|
:00412CA8  push 00000000
```

Allí vemos la comparación y el salto justo antes a la llamada al cartelón que esta en 412ca2 y es un dialogboxparama , el salto anterior en **412c94**, saltea el cartelón y hace que no aparezca, podemos reemplazar el **JZ** por un **JMP** y listo, saltaría por encima del cartel, pero por si acaso realica el mismo test más adelante vamos a MOVER a **43228c** el valor de uno que es el que compara aquí y hace que salte.

Quedaría así

```
:00412C8A C7058C22430001000000  mov   dword ptr [0043228C], 00000001
:00412C94 EB12                  jmp   00412CA8
:00412C96 A02841006A            mov   al, byte ptr [6A004128]
:00412C9B 006A74                add   byte ptr [edx+74], ch
:00412C9E 8B5508                mov   edx, dword ptr [ebp+08]
:00412CA1 52                    push  edx

* Reference To: USER32.DialogBoxParamA, Ord:0093h
                                  |
:00412CA2 FF15A8734200          Call  dword ptr [004273A8]

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00412C88(C), :00412C94(U)
|
:00412CA8 6A00                  push  00000000
```

Hacemos con esto que además de que salte por encima del cartel y que no caiga en el DIALOGBOXPARAMA, nos guarde en 43228c el valor uno que es lo que tiene que tener para estar REGISTRADO.

CON ESTO PROGRAMA CRACKEADO.

HASTA LA LECCIÓN 28
Ricardo Narvaja

LECCIÓN 28: LABEL MATRIX 5.0

Siguiendo con el crackeo a pedido, este programa me lo pidieron y por eso lo crackee, sirve para hacer etiquetas con códigos de barras e imprimirlas, y varias cosas más.

Se baja de http://www.strandware.com/prod/labelmatrix.html

Para el que sabe ingles esto dice la pagina que hace el programa:

Label Matrix provides advanced design, printing and database features that give you the flexibility to meet your most demanding labeling needs. With hundreds of thermal and thermal transfer custom printer drivers, you can maximize your printer's performance and print large runs of labels in a very short time. Whether you need address labels for your small business or thousands of inventory labels for a multi-site warehousing plant, Label Matrix can do the job.

Bueno instalamos el programa y lo iniciamos y vemos ventanas que nos recuerdan que es una versión demo, vemos en una ventana exactamente la STRING entre paréntesis **(Demo Versión)**, así que si abrimos el ejecutable con el WDASM y vemos que allí no esta, pero si ponemos algún Breakpoint en el SOFTICE como **BPX Drawtexta** o **BPX Getversion**, al volver de la función vemos que el ejecutable enseguida le pasa el control a otro archivo y es este el que sigue con la protección. Es un dll que se llama Lmw500v.dll y es un archivo de cuatro megas y pico donde prácticamente se encuentra todo el programa.

Abrimos con el Wdasm este dll que se encuentra en el directorio donde se instalo el programa, generalmente **C:\Program Files\lmwdemo.**

Se desensambla perfectamente ya que no esta comprimido ni nada, y allí entre las STRINGS REFERENCES encontramos rápidamente **(Demo Version)**. Al hacer doble click nos lleva a esta sección del código

```
:1023F66F E82C7EE1FF  call 100574A0
:1023F674 2540010000  and eax, 00000140
:1023F679 85C0        test eax, eax
:1023F67B 7444        je 1023F6C1

* Possible Reference to String Resource ID=07453: "(Demo Version)"
```

Este salto nos haría registrarnos, si lo invertimos o por lo menos saltear la aparición de la palabra **DEMO VERSIÓN**, pero miremos dentro del CALL anterior que es el que decide si salta o no, porque según el valor de EAX que sale de ese CALL decide saltar, y puede ser que ese CALL sea el verificador de si estas registrado o no, y también desde otros lugares salte allí en distintas partes del programa para verificar si estas registrado. **Entremos dentro del CALL 100574a0.**

Las referencias a este CALL son muchísimas no quiero llenar la hoja con ellas pero este CALL es llamado desde muchísimas partes del programa probablemente a verificar si estas registrado a cada rato, así que como no podemos parchear cientos de lugares desde donde es llamado el CALL, parchearemos el contenido para que siempre que sea llamado al final salga siendo **EAX=0** que es lo que al salir del CALL

verifica, testea y hace invertir el salto, cualquier otro valor de EAX hace que vaya a **DEMO VERSIÓN.**

Veamos el CALL

```
:100574A0 55            push ebp
:100574A1 8BEC          mov ebp, esp
:100574A3 51            push ecx
:100574A4 894DFC        mov dword ptr [ebp-04], ecx
:100574A7 8B45FC        mov eax, dword ptr [ebp-04]
:100574AA 8B8020010000  mov eax, dword ptr [eax+00000120]
:100574B0 8BE5          mov esp, ebp
:100574B2 5D            pop ebp
:100574B3 C3            ret
:100574B8 CC            int 03
:100574B9 CC            int 03
:100574BA CC            int 03
:100574BB CC            int 03
:100574BC CC            int 03
```

En negrita esta marcado el RET donde termina el CALL. Hay que cambiar ese **RET** por **MOV eax,00** y después **RET** para que vuelva. O sea que independientemente de lo que haga antes, salga moviéndose a EAX el valor cero y recién después haga el RET para volver.

Quedaría así

```
:100574B2 5D            pop ebp
:100574B3 B800000000    mov eax, 00000000
:100574B8 C3            ret
:100574B9 CC            int 03
:100574BA CC            int 03
:100574BB CC            int 03
```

Cambiamos con el ULTRAEDIT estos valores buscamos la cadena, y la reemplazamos por estos valores de abajo y PROGRAMA REGISTRADO.

Aclaración: El programa tiene muchísimas limitaciones cuando esta en DEMO, con este crack ya no vence y las limitaciones se van, igual yo no pude probar todas las funciones del programa, para saber si queda alguna , si a alguien le ocurre que le surge alguna limitación , que avise y lo seguimos en la LECCIÓN 29.

Igual todas esas llamadas al CALL para ver si estas registrado o no, es obvio que las hace cuando el programa tiene que ver si te limita o no, y al salir **EAX=0** del CALL, siempre te habilita a usarlo, como si estuvieras registrado.

Nos vemos en la LECCIÓN 29

Ricardo Narvaja

PD: Luego de probar el programa los que lo utilizan se comprobó que tiene todavía algunas protecciones más que yo no me había dado cuenta, no referente al vencimiento, eso funciona perfecto, sino al funcionamiento, por lo tanto terminaremos esos detalles en la LECCIÓN 29.

LECCIÓN 29: COMO HACER BIEN EL CRACK DEL LABEL MATRIX 5.0

Luego de hacer el crack según la LECCIÓN anterior, Mi amigo NIKI se dio cuenta de varios errores que generaba en el funcionamiento del programa, por ejemplo cuando ibas a LABEL PROPERTIES no aparecían los contenidos de las pestañas, y a veces se colgaba dando error, y se cerraba el programa.

Bueno lo que podemos rescatar de la LECCIÓN anterior es que la rutina de comprobación de si estas registrado es esa pero estaba mal parcheada y generaba errores.

El error estaba en que yo considere esa CALL como que solamente servia para comprobar si estabas registrado o no, y si salía valiendo **EAX=0** siempre estarías registrado.

Allí estaba el error ya que ese CALL además de para eso se usaba para muchas otras cosas, y si ponías EAX=0 siempre, corrompías esas otras partes del programa y daba error por todos lados.

Ahora si funciona todo ya corregí el error del crack anterior y funciona 100%. Una vez que habíamos aplicado el primer crack había quedado modificado así

```
:100574A0 55            push ebp
:100574A1 8BEC          mov  ebp, esp
:100574A3 51            push ecx
:100574A4 894DFC        mov  dword ptr [ebp-04], ecx
:100574A7 8B45FC        mov  eax, dword ptr [ebp-04]
:100574AA 8B8020010000  mov  eax, dword ptr [eax+00000120]
:100574B0 8BE5          mov  esp, ebp
:100574B2 5D            pop  ebp
:100574B3 B800000000    mov  eax, 00000000
:100574B8 C3            ret
```

lo que esta en negrita es lo que habíamos agregado y debe de quedar para que funcione perfecto de esta forma

```
:100574A0 55             push ebp
:100574A1 8BEC           mov  ebp, esp
:100574A3 51             push ecx
:100574A4 894DFC         mov  dword ptr [ebp-04], ecx
:100574A7 8B45FC         mov  eax, dword ptr [ebp-04]
:100574AA C6056043130200 mov  byte ptr [02134360],00
:100574B0 8B8020010000   mov  eax,[eax+120]
:100574B2 8BE5           mov  esp,ebp
:100574B3 5D             pop  ebp
:100574B8 C3             ret
```

Lo que esta en negrita es lo nuevo que agregamos. En realidad al funcionamiento solo le agregamos **mov byte ptr [02134360],00**

Lo tenemos que intercalar justo allí, porque en la sentencia subsiguiente mueve el valor de EAX+120 a eax y por lo tanto, como cuando el programa verifica la registración EAX+120 es igual a 2134360, previmos que allí haya un cero para que EAX salga valiendo cero solo en ese caso, ahora cuando EAX+120 vale otra cosa, la rutina

funciona como cuando no estaba parcheado, o sea que ponga un cero en 2134360 no afecta ya que después mueve el contenido de EAX+120 a eax y este es el valor que usa el programa normalmente, lo que no lo afecta para nada y funciona bien 100%.

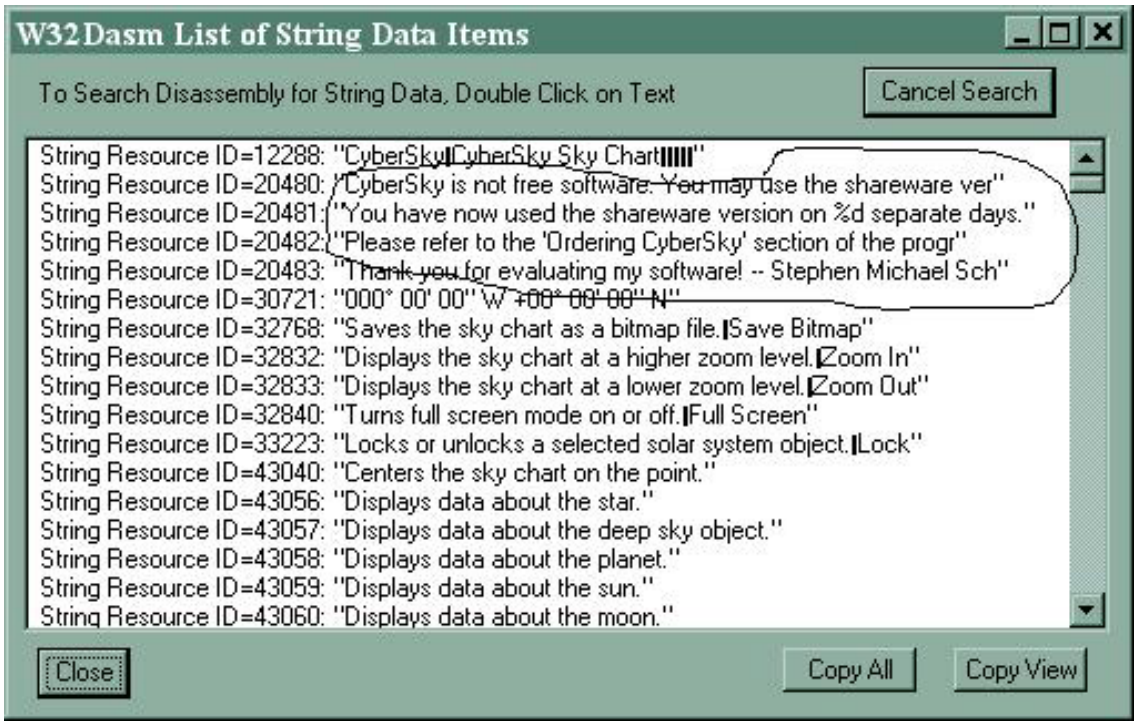
Ricardo Narvaja

LECCIÓN 30: EL MOVIMIENTO DE LAS ESTRELLAS: CIBERSKY

Esta lección se refiere al programa CIBERSKY 3.2.1 que se baja de http://www.cybersky.com/, que muestra las constelaciones y como se van moviendo las estrellas. Es muy lindo para la persona que esta interesada en astronomía.

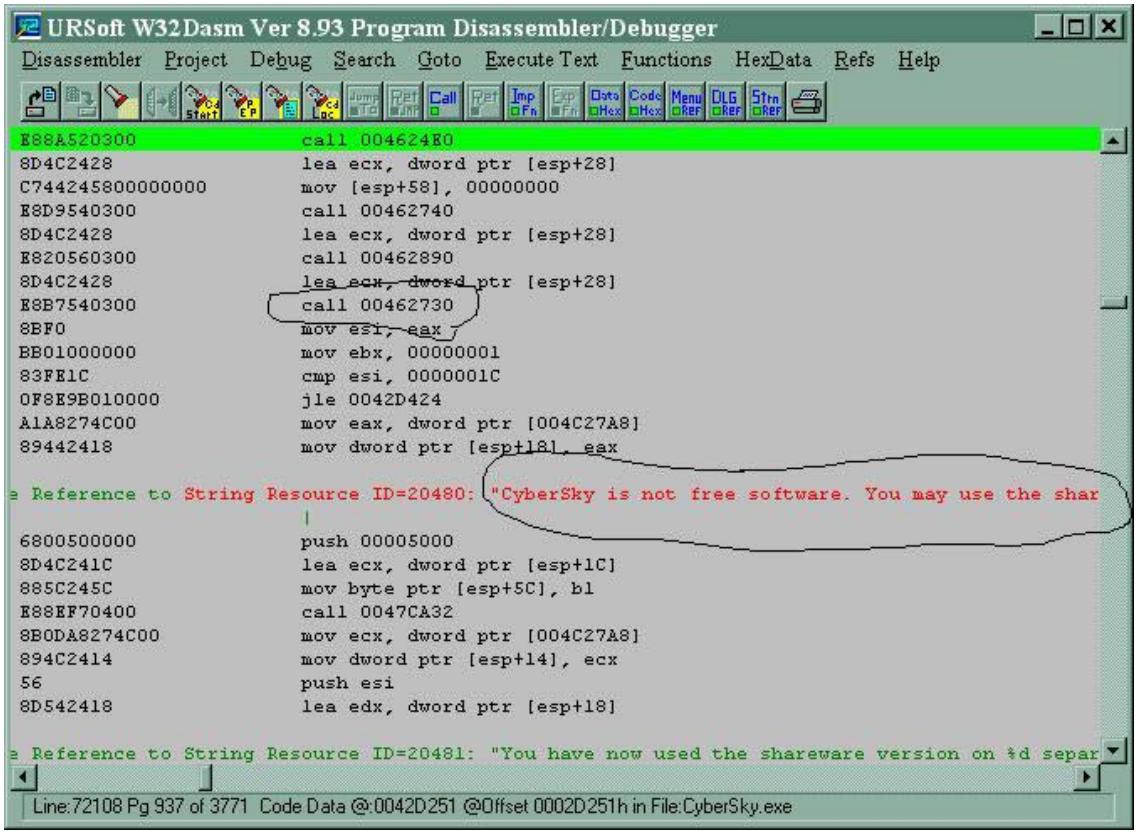
Es bastante fácil de crackear así que será bastante breve la LECCIÓN ya que no esta ni comprimido, ni tiene protección ANTISOFTICE, ni nada, lo único molesto que tiene es que sale un cartel diciéndonos que nos registremos a los 15 días de usarlo y a los 60 días ese cartel te tapa todo y no te deja ver nada.

Lo verdaderamente que cansa en este programa es que si quiero adelantar el reloj para ver ese cartel, no sale, o sea que el programa suma día por día que vas usando, y no le importa la fecha del reloj, solo es un día más, aunque diga AÑO 3005, así que vamos a desensamblar con el WDASM el ejecutable.



Aquí en el grafico vemos las STRINGS REFERENCES del cartel molesto, **TANKS FOR EVALUATING MY SOFTWARE** y antes **CIBERSKY IS NOT FREE SOFWARE**, etc, etc, etc.

Hacemos clic en la primera, y aparecemos en el WDASM en



Este es uno de los lugares donde cae, si hacemos clic de nuevo vamos a ver que aparecen estos mensajes en varios lugares distintos del programa, pero antes siempre esta el CALL que esta marcado en el dibujo **CALL 462730**, que es donde el programa cuenta los días, y sale **EAX valiendo los días que usaste el programa.**

Miremos dentro de ese CALL poniéndonos encima y haciendo clic en el ICONO CALL.

```
:00462730 8B4124  mov eax, dword ptr [ecx+24]
:00462733 8B5114  mov edx, dword ptr [ecx+14]
:00462736 2BC2    sub eax, edx
:00462738 C3      ret
```

Si modificamos este CALL y hacemos que eax salga valiendo un valor fijo es como si claváramos los días que usamos en un valor fijo probemos esto.

```
:00462730 8B4124      mov eax, dword ptr [ecx+24]
:00462733 8B5114      mov edx, dword ptr [ecx+14]
:00462736 2BC2        sub eax, edx
:00462738 B800000000  mov eax, 00000000
:0046273D C3          ret
```

Con lo que siempre sale EAX valiendo cero y la comparación que hay al salir este call hace que el salto condicional siguiente saltee el cartel maldito.

Como este CALL es siempre el que decide la aparición del CARTEL MALDITO en todo el programa, entonces ya no aparece nunca más en ningún caso.

PROGRAMA CRACKEADO

Hasta la 31
Ricardo Narvaja

LECCIÓN 31: MEDIO DIFÍCIL

Esta lección se refiere al programa **NBG clean registry 1.7.2** que sirve para limpiar el registro, de entradas no validas, de basura, de restos de entradas de programas desinstalados, etc. Se baja de www.tucows.com y allí esta para bajar esta versión.

Cuando abrimos el programa sale una horrible pantalla roja para registrarse, allí dice los días que te quedan de prueba, y si haces clic en el icono con la llavecita, te trata de conectar a internet vaya a saber para que pero no te podes registrar por allí, (SALVO PAGANDO Y ESO NO LO RECOMIENDO), jua jua, y cuando cerrás el programa vuelve a aparecer la ventan roja horrible esa.

Bueno tratamos de abrir el ejecutable con el WDASM y nada no se desensambla, miro con el Languaje 2000 y me dice que esta comprimido con **ASPACK O ASPROTECT**, así que con el CASPR 1.012 o la ultima que haya salido, lo descomprimo fácil, según vimos en lecciónes anteriores, y me entrega el ejecutable sin la protección ASPROTECT o ASPACK. Lo que queda es quitar la protección antidesensamblado con el PROCDUMP cambiando C0000040 por E0000020 y listo, queda perfecto para desensamblar con el WDASM para DELPHI (o sea el que no esta modificado para VISUAL BASIC).

Igual el programa mantiene una protección antisoftice pero con el FROGSICE pasa fácilmente.

Cuando quiero ejecutar el programa descomprimido pongo el ejecutable NBGcleanRE.exe en la carpeta de instalación del programa C:\NBGCleanRE y lo ejecuto haciendo doble clic y OOOOPS aparece un mensaje diciendo que o un virus o alguien modifico el CRC del programa y no funciona.

Entre las STRINGS REFERENCES encontramos fácilmente este cartelito que dice exactamente:

INCORRECT CRC CHECKSUM AND SIZE

Si hacemos doble click encima de esa STRING REFERENCE caemos en

:4e9f49 75 EF

Cambiamos el **75** por **EB** y listo, saltea ese cartel y arranca el programa perfectamente.

```
:004E9F45 807DF300  cmp byte ptr [ebp-0D], 00
:004E9F49 EB7F      jmp 004E9FCA
```

Con eso ya saltamos ese cartelito que es tipo protección por si alguien le limpia el ASPROTECT o ASPACK.

Bueno ya arranca el programa y ya esta limpio, ahora hay que crackearlo lo cual no es fácil tampoco.

Buscaremos alguna STRING REFERENCE que diga si estas registrado o no.

Me gusta una cuando arrancas el programa y una vez que ingresas, y entras al mismo, haces clic arriba a la izquierda donde dice en azul

NBG CLEAN REGISTRY y eso abre una ventana que dice **VERSIÓN 1.7.2 ... SHAREWARE UNREGISTERED...**

Vuelvo al WDASM y busco esa STRING y allí esta, si hago clic encima aparece dos veces, casualmente una arriba de la otra.

```
* Possible StringData Ref from Code Obj ->"Version 1.7.2 Release Skin ENG"
                                        ->"(build 2012) shareware"
|
:004FAB87 6874AD4F00     push 004FAD74
:004FAB8C FFB308050000   push dword ptr [ebx+00000508]
:004FAB92 68C4AD4F00     push 004FADC4
:004FAB97 FFB30C050000   push dword ptr [ebx+0000050C]
:004FAB9D 8B8308030000   mov  eax, dword ptr [ebx+00000308]
:004FABA3 05A0000000     add  eax, 000000A0
:004FABA8 BA04000000     mov  edx, 00000004
:004FABAD E82E94F0FF     call 00403FE0
:004FABB2 EB15           jmp  004FABC9
:004FABB4 8B8308030000   mov  eax, dword ptr [ebx+00000308]
:004FABBA 05A0000000     add  eax, 000000A0

* Possible StringData Ref from Code Obj ->"Version 1.7.2 Release Skin ENG"
                                        ->"(build 2012) shareware"
```

Así que veamos la de arriba de las dos.

```
:004FAB7E  cmp byte ptr [eax+00000504], 00
:004FAB85  jnz 4fabb4

* Possible StringData Ref from Code Obj ->"Version 1.7.2 Release Skin ENG"
                                        ->"(build 2012) shareware"
```

Bueno, parece que en el contenido de **[eax+504]** o sea en **1AE24A8**, ESTA EL FLAG DE SI ESTAS REGISTRADO O NO, PROBEMOS CON EL SOFTICE ANTES DE ENTRAR EN ESTA VENTANA, PONEMOS UN **BPX SHOWWINDOW** Y UNA VEZ QUE HACEMOS CLIC Y ENTRA EN EL softice , VOLVEMOS CON F12 AL EJECUTABLE, Y ALLI BORRO CON **BC*** Y PONGO UN **BPX 4FAB7E**, PARA QUE PARE ALLI. CIERRO LA VENTANA LA VUELVO A ABRIR Y PARA ALLÍ ANTES DE QUE APAREZCA.

CUANDO ESTOY ENCIMA DE 4FAB7E HAGO:

E 1AE24A8 Y ENTER

CON ESO PUEDO CAMBIAR EL VALOR QUE ESTA COMPARANDO, YA QUE HAY UN CERO, PONGO UN UNO A VER QUE PASA, LUEGO HAGO ENTER, Y HAGO X.

Me aparece la ventana diciendo REGISTRADO, hmmmm, que bueno.

Bueno aparentemente esa posición de memoria es la que siempre testea para ver si estas registrado o no, por lo tanto, podemos poner un **BPR 1AE24A8 1AE24A8 RW** para que pare allí cada vez que trate de leer el valor de ese flag.

Si quiero poner ese BPR antes de que arranque el programa, no para en ningún lado, alguna protección hay, entonces lo que hacemos es poner BPX GETVERSION y cuando para allí volvemos al ejecutable (DESPUES DE UNOS 40’S F12) y allí recién pongo el BPR.

Para en estos cuatro lugares:

4EAECA, 4EB142, 4FF39E, 4FAB7E

Donde hay siempre comparaciones de **EAX+504 con cero o con uno**

cmp byte ptr [eax+00000504], 00

También para en otros lugares donde mueve valores allí, pero esos no me interesan solo cuando toma decisiones según el valor que hay allí.

Y ESPERO QUE NINGUNO OTRO MÁS. IGUAL TIENE UNA CIERTA TRAMPITA EN ALGUNAS DE ESTAS COMPARACIONES. EL VALOR QUE HAY EN 1AE24A8 ES UN UNO Y ESPERA UN CERO Y OTROS ES CERO Y ESPERA UN UNO. CON EL SOFTICE DETERMINAMOS QUE VALOR TIENE EN CADA SALTO, PARA SABER COMO PARCHEAR CADA UNO.

EN RESUMIDAS CUENTAS TIENE QUE QUEDAR ASI:

LA COMPARACIÓN EN 4EAECA:

```
:004EAECA C6800405000001  mov byte ptr [eax+00000504], 01
:004EAED1 EB04            jmp 004EAED7 (O NOP NOP)
:004EAED3 90              nop
:004EAED4 90              nop
:004EAED5 90              nop
:004EAED6 90              nop
:004EAED7 8B45FC          mov eax, dword ptr [ebp-04]
```

LA REEMPLAZO POR MOVER A [EAX+504] EL VALOR 01 (YA QUE HABIA CERO) Y DESPUES NO SALTO, YA QUE VI EN EL SOFTICE QUE CUANDO LLEGO ALLI SALTA Y TENGO QUE HACER LO CONTRARIO POR ESO **SALTO A 4EAED7 QUE ES LO MISMO QUE NOPEAR.**

LUEGO QUEDA LA COMPARACIÓN EN 4EB142 DE ESTA FORMA:

```
:004EB142 C6800405000001  mov byte ptr [eax+00000504], 01
:004EB149 E9BC110000      jmp 004EC30A
```

LO MISMO, VEO QUE EN EL CONTENIDO DE EAX+504 HAY UN CERO POR ESO HAGO QUE LA SENTENCIA MUEVA ALLI UN UNO, Y COMO DESPUES NO SALTA LO HAGO SALTAR (que contrera, JUA).

LUEGO VIENE 4FF39E QUE QUEDA ASI:

```
:004FF39E C6800405000001  mov byte ptr [eax+00000504], 01
:004FF3A5 90              nop
:004FF3A6 90              nop
```

Y LA ULTIMA 4FAB7E:

```
:004FAB7E C6800405000001  mov byte ptr [eax+00000504], 01
:004FAB85 90              nop
:004FAB86 90              nop
```

Bueno una vez que hacemos todos estos cambios probamos a ver que pasa y la pantalla roja horrible ya no aparece (cuando cerramos el programa

tampoco) y el programa arranca directamente y dice que estamos registrados, igual este tipo de programas puede tener por ahí otra comparación media escondida por lo que habría que probarlo en profundidad pero parece funcionar bien.

Ya veremos, si aparece alguna sorpresa lo seguiremos en la lección 32, pero por ahora...

PROGRAMA REGISTRADO

Ricardo Narvaja
Hasta la LECCIÓN 32

LECCIÓN 32: SOFTCOPIER 3.4.2.4

Este programita es una linda utilidad para sacar fotocopias rápido, scannea y manda directo a la impresora, para fotocopiar, rápido y fácil uno o varios papeles. Se baja de www.buzzsoft.com

Bueno este programa va contando la cantidad de copias que uno hace y cuando llega a 25 se vence, teniendo que actualizarse por Internet para poder copiar 25 más, ufff.

Bueno para mi gusto, a pesar de que los cracks que hay por allí disponibles, trabajan sobre ese registro, lo que yo usé es tratar de que no llegue nunca la cuenta a 25 copias para que nunca venza.

Lo primero es ver con WDASM el ejecutable SC3 que esta en **C:\Archivos de programa\BuzzSoft**Sc

Se descomprime fácilmente y entre las STRINGS REFERENCES lo primero que llama la atención es:

"You have just completed a demonstration of SoftCopier. SoftCopier is a powerful yet"

Ojo que hay que usar el WDASM original o sea sin la modificación para VISUAL BASIC sino esta STRING no aparecerá ya que esta hecho en DELPHI.

Bueno dice que se acabo la demostración y que se va a modificar el programa para que si no lo actualizas no lo puedas usar más y bla bla bla.

Bueno tenemos que tratar de evitar que por cualquier motivo el programa llegue aquí a esta ZONA de CHICO MALO, yo diría muuuy malo, jua jua.

Si vamos un poco hacia arriba encontramos en **459d25** este salto que esquiva todo este cartelón y todo este sector de chico malo; justo evita lo que no queremos, que por cualquier cosa me venza.

```
:00459D1D A17C1D4600    mov eax, dword ptr [00461D7C]
:00459D22 803800        cmp byte ptr [eax], 00
:00459D25 0F8574060000  jne 0045A39F
```

Obviamente tengo que reemplazar ese salto condicional por un JMP para que nunca entre a esa zona que me puede modificar algo y que no funcione más. Esto hay que hacerlo aunque logremos que el programa no aumente de alguna forma la cuenta de las copias que hizo.

Esto después de modificado quedaría así:

```
:00459D1D A17C1D4600   mov eax, dword ptr [00461D7C]
:00459D22 803800       cmp byte ptr [eax], 00
:00459D25 E975060000   jmp 0045A39F
:00459D2A 90           nop
```

El NOP al final es porque el salto condicional ocupaba 6 valores hexa y el JMP ocupa 5 solamente, para que quede bien y reemplace los seis.

Bueno ahora que ya modificamos esto y estamos un poco más tranquilos tenemos que encontrar donde guarda este programejo las copias que va haciendo. Lo primero que pensé es usar el ART para ver si hace cambios en el REGISTRO al hacer una copia, pero ya que no estoy seguro de que sea en el registro, voy a usar otra herramienta poderosa llamada **TECHFACTS** que entre muchísimas utilidades que tiene permite hacer lo mismo pero en todo el sistema, y cualquier cambio que haga te lo saca en una lista.

Muy bien, como nosotros queremos estudiar los cambios que produce el SOFTCOPIER conviene cerrar cualquier otra aplicación que este corriendo ya que puede realizar algún cambio y confundirnos.

Esto lo podemos hacer con CTRL+ALT+DEL y cerrar todo uno por uno hasta que quede solo el explorer y ahí recién arrancar el TECHFACTS.

Como verán dentro de este programa hay muchísimas utilidades pero lo que nosotros vamos a utilizar esta en la pestaña TOOLS y haciendo click en **WATCH SYSTEM**, poniendo la tilde en **RUN THIS PROGRAM** no tocamos más nada y abrimos el buscador para encontrar el ejecutable de SOFTCOPIER que esta en **C:\Archivos de programa\BuzzSoft\Sc** y se llama SC3.

Hay otro ejecutable en otro directorio que es una barra de herramientas que después abre este ejecutable así que nos vamos a concentrar directo en este ya que es el que hace la copias. Lo abrimos y abajo hacemos click en **GO** para que empiece a trabajar y el TECHFACTS hará primero una instantánea del sistema completo, correrá el programa, donde nosotros haremos una copia para que aumente el valor de copias en uno más, y cerraremos el programa, luego de lo cual el TECHFACTS hará otra instantánea del sistema y me dirá donde realizo cambios el SC3.

El reporte a mi me dice lo siguiente:

TechFacts 98 System Watch Report
04/23/01 08:44:33 pm

The following directories and files were added:(0)

The following files were modified:(4)
c:\WINDOWS\SYSTEM.INI
c:\WINDOWS\TWAIN.LOG
c:\WINDOWS\Twain001.Mtx
c:\WINDOWS\WIN.INI

Changes made to: C:\WINDOWS\WIN.INI...
Keys changed in: C:\WINDOWS\WIN.INI: (1)
[SoftCopier3]Number of Copies=3 to 4

Changes made to: C:\WINDOWS\SYSTEM.INI...
Keys changed in: C:\WINDOWS\SYSTEM.INI: (1)
[NonMSApps]TWAIN_SC3REF=74 to 93

Registry key values changed: (2)
HKEY_USERS\.DEFAULT\Software\Borg
Value "Winmax": from "18973185" to "18813185"

HKEY_USERS\.DEFAULT\TWAIN\BUZZTW32
Value "REFNO": from "845" to "914"

Marque con negrita lo importante, el programa al hacer una copia realizo todos estos cambios.

Vemos que dice que modifico el **WIN.ini** y el **SYSTEM.ini** y en el WIN.ini dice el número de copias (esto se pone interesante). Hay un par de claves de registro que también cambiaron, como se ve.

Bueno yo me tome el trabajo de editar el WINI.ini a mano para ver si cambiando el número de copias que ahora esta en cuatro. Lo ponía a mano en cero, el programa cuando arrancaba me decía que había hecho cero copias, pero no, ese valor no influye, es solamente informativo.

También los valores del registro los probé volver al valor anterior a mano:

HKEY_USERS\.DEFAULT\Software\Borg
Value "Winmax": from "18973185" to "18813185"

Fui al regedit y cambié el valor que había en esa clave **18813185** por el que había antes **18973185** y volví a arrancar el programa, y seguía con las cuatro copias, lo mismo ocurrió con la otra clave del registro:

HKEY_USERS\.DEFAULT\TWAIN\BUZZTW32
Value "REFNO": from "845" to "914"

Volví a escribir **845** que era lo que decía allí cuando había hecho solo tres copias y arranque el programa de nuevo, y otra vez cuatro copias.

Lo único que me quedaba era el system.ini

Changes made to: C:\WINDOWS\SYSTEM.INI...
Keys changed in: C:\WINDOWS\SYSTEM.INI: (1)
[NonMSApps]TWAIN_SC3REF=74 to 93

Así que abrí el **SYSEDIT** y busque el system.ini y cambié el valor de **93** por el de **74**, como había antes y arranque el programa de nuevo BIIINGOOO, ahora dice tres copias, me fije que de 74 a 93 hay 19 y trate de ir más atrás reste a 74-19=55. Lo puse en el SYSTEM.ini y arranque de nuevo el programa y DOS COPIAS, jua jua.

Bueno si a 55 le vuelvo a restar 19 dos veces más puedo llegar hasta cero de nuevo, pero lo dejo así en 55 ya que es un número fácil de encontrar, sabiendo que en hexadecimal es 37 no? O sea que buscare por allí el 37 con el SOFTICE.

Voy al WDASM y me doy cuenta que esta la STRING REFERENCE "SYSTEM.INI" y cuando hago click encima me lleva a seis posiciones distintas a saber:

45cded, 45d673, 454261, 4542ee, 454466, 45bad8

Es seguro que cuando utilice esta STRING el programa estará leyendo o grabando el valor del SYSTEM.ini, así que en el softice arranco el

programa con el SYMBOL LOADER y cuando arranca pongo BPXs en todos estos puntos.

También puedo poner un **BPX GETVERSION** y una vez que rompe allí por segunda o tercera vez volver al ejecutable y poner los BPXs es lo mismo.

Para dos veces cuando arranca y una vez después que sacamos una copia y una vez cuando se cierra el programa.

Paro aquí:

```
* Possible StringData Ref from Code Obj ->"System.Ini"
|
:0045CDED B9D0CF4500  mov  ecx, 0045CFD0
:0045CDF2 B201        mov  dl, 01
:0045CDF4 A124EF4300  mov  eax, dword ptr [0043EF24]
:0045CDF9 E88221FEFF  call 0043EF80
:0045CDFE A3042B4600  mov  dword ptr [00462B04], eax
:0045CE03 6A13        push 00000013

* Possible StringData Ref from Code Obj ->"TWAIN_SC3REF"
|
:0045CE05 B9E4CF4500  mov  ecx, 0045CFE4

* Possible StringData Ref from Code Obj ->"NonMSApps"
|
:0045CE0A BAFCCF4500  mov  edx, 0045CFFC
:0045CE0F A1042B4600  mov  eax, dword ptr [00462B04]
:0045CE14 E89F22FEFF  call 0043F0B8
:0045CE19 8BF0        mov  esi, eax
:0045CE1B 8BC6        mov  eax, esi
```

Como ven desde donde esta la STRING (en negrita) de SYSTEM.ini hasta donde esta lo que hay que parchear (también en negrita) no hay mucho, unas cuantas sentencias no más.

Allí al salir del CALL que esta en rojo en EAX me aparece el valor 37 que va a pasar a ESI, entonces lo modifico para que no sea así, lo cambio por esto:

```
:0045CE19 2BF6  sub esi, esi
:0045CE1B 8BC6  mov eax, esi
```

O sea que SUB es restar ESI-ESI o sea que va a quedar siempre ESI=0, luego ese valor se pasa de nuevo a EAX, que queda valiendo cero, hago los cambios a ver que pasa y cuando arranca el programa me dice CERO COPIAS, jua jua, va mejorando.

El próximo lugar que vi es cuando se cierra el programa. Allí paró una vez que hice una copia y me aumento el valor de copias a uno, seguro para guardar ese valor en el SYSTEM.ini.

Aquí vemos de nuevo la STRING system.ini

```
* Possible StringData Ref from Code Obj ->"System.Ini"
|
:0045BAD8 B97CBB4500  mov  ecx, 0045BB7C
```

```
:0045BADD B201        mov  dl, 01
:0045BADF A124EF4300  mov  eax, dword ptr [0043EF24]
:0045BAE4 E89734FEFF  call 0043EF80
:0045BAE9 8906        mov  dword ptr [esi], eax
:0045BAEB 6BC313      imul eax, ebx, 00000013
:0045BAEE 83C011      add  eax, 00000011
:0045BAF1 8BD8        mov  ebx, eax
:0045BAF3 53          push ebx
```

y tiene que quedar asi

```
0045BAEB 6BC313       imul eax, ebx, 00000013
0045BAEE 2BC0         sub  eax, eax
0045BAF0 2BDB         sub  ebx, ebx
0045BAF2 90           nop
0045BAF3 53           push ebx
```

o sea, saliendo el valor de eax y ebx siendo cero. Esto también se ve claro en el SOFTICE que aquí tiene los valores que va a guardar en el SYSTEM.ini.

La última modificación es para que no aumente la cuenta mientras uno va sacando copias y no cierra el programa ya que a pesar de que siempre empezamos de cero, como sigue aumentando los valores hasta que uno cierra el programa y graba un cero, en el SYSTEM.ini, puede ocurrir que uno se pase de la 25 copias antes de cerrar el programa y KAPUTTT.

Eso está unas líneas más arriba justo donde hay una STRING que dice **NÚMERO DE COPIAS**

```
:0045BAAD 8B1D802B4600  mov  ebx, dword ptr [00462B80]
:0045BAB3 031D842B4600  add  ebx, dword ptr [00462B84]
:0045BAB9 53            push ebx
```

*** Possible StringData Ref from Code Obj ->"Number of Copies"**

Allí, en **462b80** guarda el número VIEJO de copias que uno tenia y en **462b84** el número NUEVO de copias que hizo, para actualizarlo, los tiene que sumar y es lo que hace en la sentencia **45bab3** que suma a EBX, que tiene el valor viejo de copias, el número de copias que hice para actualizarlo. Por lo tanto si anulamos esta suma en EBX me quedara el valor viejo de copias que tenia y no se actualizara aunque hayamos hecho varias copias, siempre quedara igual, por lo tanto a

0045BAB3 031D842B4600 add ebx, dword ptr [00462B84]

la reemplazamos por 6 NOPS; quedaría así:

```
:0045BAAD 8B1D802B4600  mov  ebx, dword ptr [00462B80]
:0045BAB3 90            nop
:0045BAB4 90            nop
:0045BAB5 90            nop
:0045BAB6 90            nop
:0045BAB7 90            nop
:0045BAB8 90            nop
:0045BAB9 53            push ebx
```

Modifico esto, arranco el programa me dice cero copias, sigo haciendo copias y me dice cero copias y cierro el programa y en el system.ini, dice cero copias.

Ustedes dirán, porque si logramos que no variara el número de copias pusimos el JMP en el vencimiento del programa ya que no es necesario?

Yo no estoy seguro pero ese valor en el registro que sigue aumentando

HKEY_USERS\.DEFAULT\TWAIN\BUZZTW32
Value "REFNO": from "845" to "914"

cada vez que hacemos una copia sigue aumentando a pesar de todo. Por si acaso si hay un limite máximo para este número que el programa vaya al vencimiento por aquí, es conveniente poner el JMP, por si acaso.

Quizás después de dos meses el programa compruebe este valor y me tire a vencido, al estar el JMP, me salvo.

También puedo buscar en el programa cuando lee de aquí y buscar este valor de 914, para modificar el programa para que no pase nada, pero creo que con lo que se hizo es suficiente.

Programa crackeado
Hasta la 33
Ricardo Narvaja

LECCIÓN 33: ALGO FÁCIL PARA UNA SEMANA DIFÍCIL (PARA MI)

El programa victima se llama **SMPServer** y la versión que yo tengo es la Standard R4 build 20010409. Eso dice en el programa si quieren saber para que sirve y saben ingles aquí esta lo que dice la pagina del mismo que es http://www.eastbow.com/eng/smp_download.htm

What can you do with SMP?

You can get the following data from your PCs: **(en red)**

IP address, Computer name, Login name, Department, Etc, CPU, RAM, OS, Disk, CD-ROM, Media card, Modem, Video card, Monitor, Printer, Mouse, Keyboard, Network card, SCSI, USB, Current process, Screen settings and installed software list.

With these data, you can make the report including,

H/W list/statistics,
S/W list/statistics

in Microsoft Excel (tab seperated) and HTML format.
In addition, SMP can catch the changes including, hardwares, IP movement.

Bueno la limitación es que es para 5 licencias o PCs solamente y trataremos de que no tenga esta limitación.

Desensamblamos el ejecutable con el WDASM y no hay problema lo hace perfectamente, no esta comprimido. Allí hay una STRING REFERENCE sospechosa

"You need more licenses to add more PC information."

O sea dice que necesitamos más licencias para agregar más información; aquí parece estar la limitación.

```
:0041998B E8E079FEFF  call 00401370
:00419990 85C0        test eax, eax
:00419992 7D19        jge 004199AD
:00419994 6A01        push 00000001

* Possible StringData Ref from Data Obj ->"You need more licenses to add "
                                        ->"more PC information."
```

Aquí parecería que con parchear este salto, ya estaría pero vamos a ver un poco ese **CALL 401370** que hay antes del salto de donde sale EAX tomando un valor que decide si salta o no.

```
* Referenced by a CALL at Addresses:
|:0040DFB5, :0041998B, :004199E2

:00401370 A1808F4500  mov eax, dword ptr [00458F80]
```

Bueno vemos que entra a este CALL llamado de varios lugares distintos del programa, o sea que en varios puntos existirá esta limitación pero si parcheamos el CALL para que cuando salga valga EAX=0 eso hace

saltar siempre los saltos condicionales que hay a la salida de este CALL en los distintos puntos del programa para que evite la limitación.

Uno de los puntos desde donde llega a ese CALL es

```
:0040DFB5 E8B633FFFF    call 00401370
:0040DFBA 8BD8          mov  ebx, eax
:0040DFBC 85DB          test ebx, ebx
:0040DFBE 0F8D24080000  jnl  0040E7E8

...

:0040DFE9 50            push eax

* Possible StringData Ref from Data Obj ->"A PC is tried to connect in the
                                        ->"condition of license full."
```

Aquí hay otro lugar donde hay una limitación vemos que sale del **CALL 401370** como en la limitación anterior, y aquí también el valor de EAX, se pasa a EBX y se testea EBX y si es cero salta por encima del cartel molesto que dice que una PC esta tratando de conectarse en la condición de licencia llena, y no te deja.

O sea que aquí también si sale EAX siendo cero vamos a sortear la limitación.

Por lo tanto dentro del CALL 401370 reemplacemos

```
:004013A1 83C8FF  or  eax, FFFFFFFF
:004013A4 5E      pop esi
:004013A5 C3      ret
```

OR EAX, FFFFFFFF por **XOR EAX, EAX** que es una función que hace EAX igual a cero siempre.

Y también aquí

```
:004013A6 8BC6  mov eax, esi
:004013A8 5F    pop edi
:004013A9 5E    pop esi
:004013AA C3    ret
```

Reemplazamos **MOV EAX,ESI** por **XOR EAX,EAX** por que a veces dentro del CALL sale a veces por el primer RET y a veces por este segundo RET ya que hay un salto condicional en

```
:00401392 7412  je 004013A6
```

que hace que salga a veces por un RET y a veces por otro. Ahora salga por donde salga siempre va a ser **EAX=0** y va a seguir funcionando sin caer en los cartelitos que dicen que no se pueden usar más de cinco licencias.

O sea en 4013a1, **83 c8 ff**, lo reemplazo por **33 c0 90** que es **XOR eax,eax** y un **NOP** ya que son tres cifras.

Y en 4013a6 reemplazo **8b c6** por **33 c0** aquí sin NOP ya que son solo dos cifras.

Hasta la 34.
Programa craqueado

Ricardo Narvaja

LECCIÓN 34: EMPEZANDO A USAR EL REVIRGIN (TRAGO AMARGO Y CON PACIENCIA)

El REVIRGIN es una herramienta que permite reconstruir los ficheros que pudimos dumpear (descomprimir) ya sea con el PROCDUMP o ICEDUMP al disco rígido y no funcionan, generalmente la mayoría de los compresores nuevos destruyen la tabla de funciones importadas, para que no puedan arrancarse y apenas arranquen den errores varios de WINDOWS.

Por lo tanto es un muy buen programa sobre todo en casos como compresiones con ASPROTECT o algunos otros, que no permiten ser parcheados en memoria con RISC PROCESS PATCHER o similares.

También les comento que en el uso del REVIRGIN yo también estoy haciendo mis primeros pinitos y aprendiendo a la par de ustedes, así que estamos aprendiendo juntos, en este caso utilicé como base un tutorial existente sobre REVIRGIN aunque esta en ingles y además es más complicado que el que voy a hacer yo pues como es de hace unos meses, y las nuevas versiones de REVIRGIN han avanzado bastante, muchas cosas que eran necesario hacer en esa época a mano, ahora las hace el programa automáticamente. Aunque desde ya aclaremos que el programa nos facilita las cosas, pero no lo hace todo, tendremos aquí que meter mano y buscar a mano ciertos datos necesarios.

Aquí vamos a utilizar otro editor de PE que no es el PROCDUMP, se llama PE Editor y va por la versión 1.7 y lo baje de

http://212.14.34.87/~fkiepski/down/uzytki/peditor.zip

Aunque está en las principales paginas de herramientas de cracks, lo mismo que el último Revirgin, está en SUDDENDISCHARGE, el link directo es

http://www.suddendischarge.com/files/Unprotection%20Tools/revirgin110b9b.zip

Hasta ahora es el último aunque salen versiones con mejoras bastante seguido y es bueno siempre tener la ultima.

La otra herramienta que voy a usar es el ICEDUMP

http://www.suddendischarge.com/files/File%20Dumping/id6023.zip

Ustedes se preguntaran porque no el PROCDUMP, yo también pero el mismo programador del REVIRGIN dice que para mejores resultados usemos el ICEDUMP para dumpear el programa de la memoria al rígido (el que quiera hacerlo con PROCDUMP creo que funcionara, lo que parece es que es más incompatible el ejecutable resultante como para llevarlo a otras maquinas).

Bueno lo último que nos falta bajar es la victima (APLAUSSSOSSS) que esta comprimida con ASPROTECT y se llama **Cool Mouse 3.4**. Aclaremos que aquí solo veremos la descompresión del programa y que quede funcionando descomprimido, no haremos el crack del ejecutable, aunque una vez descomprimido y funcionando ya queda servido en bandeja para hacerlo.

La pagina del programa es http://www.mycoolmouse.com/ aunque yo lo baje de http://www.shelltoys.com/files/cmset.exe

Bueno una vez que nos hicimos de todo esto comencemos, instalamos el COOL MOUSE y luego lo primero es instalar el ICEDUMP, el mismo trae distintos archivos en la carpeta W9X para las distintas versiones de SOFTICE que tengamos, allí vemos desde 3.22 hasta 4.21.53, por lo pronto yo tengo la versión del SOFTICE 4.05 así que pruebo copiando icedump.exe dentro de la carpeta del SOFTICE y lo ejecuto si esta todo bien y es el que corresponde debería decir

icedump v6.0.2.3 for winice v4.05 build 334 loader
icedump was not loaded yet
icedump loaded

Lógicamente si da error hay que probar con otro hasta que encontremos el correcto, una vez que lo copiamos a la carpeta del SOFTICE, en mi caso **C:\Archivos de programa\NuMega\SoftIce95**, hacemos un acceso directo al archivo icedump.exe para arrancarlo desde el escritorio o algún lugar cómodo ya que cada vez que utilicemos el ICEDUMP en distintas oportunidades lo deberemos ejecutar para que funcione.

Una vez ejecutado y que nos dice ICEDUMP LOADED, podemos comenzar con el DUMPEO.

Podemos ya usar también el PEDITOR para obtener los valores 400000 y 51000 que son La IMAGE BASE y el largo del ejecutable por lo tanto terminara en 400000 + 51000 = 451000.

Abriendo el PEEDITOR y yendo a BROWSE y allí buscando en **C:\Archivos de programa\Shelltoys\Cool Mouse** cargamos el ejecutable Cmouse.exe y allí vemos IMAGE BASE 400000 y SIZE OF IMAGE 51000 o sea donde comienza y el tamaño que tendrá en memoria.

Podemos usar el método del BPR que hemos visto en lecciónes anteriores para encontrar el entry point, pero aquí solo ponemos un **BPX GETVERSION** y la tercera vez que pare volvemos con F12 y estaremos justo un poquito abajo del ENTRY POINT. Para en 408797 y el entry point esta un poquito más arriba en **40876b** donde vemos la típica sentencia de comienzo de programa **PUSH EBP** y luego **MOV EBP, ESP.**

Esta forma de encontrar el ENTRY POINT no siempre funciona pero a veces si, y hace más rápido el dumpeo que usar el BPR, se basa en que casi todos los programas lo primero que hacen es un BPX GETVERSION entonces si seguimos los BPX GETVERSION y vemos cual cae dentro del ejecutable en la primera sección o en una de las primeras sabremos que mirando un poquito más arriba encontraremos el PUSH EBP que es el principio del programa o sea el ENTRY POINT.

Bueno volvemos a ejecutar el programa pero ahora después del primer o segundo GETVERSION volvemos al ejecutable y tecleamos

BPM 40876b X

Para que pare en el ENTRY POINT. Es como poner un BPX pero más disimulado para que ASPROTECT no lo detecte, y borramos el BPX

GETVERSION y hacemos X para que siga y pare en 40876b. Allí estamos en el ENTRY POINT ahora hay que dumpearlo.

Tecleamos A ENTER y allí hacemos un loop infinito con **JMP EIP** (por si acaso copiemos la cadena que había antes de modificarla ya que después tendremos que volver a los valores originales con el ULTRA EDIT).

O sea había **558bEC6AFF68B8D24000** Y REEMPLAZAMOS LOS DOS PRIMEROS POR **EBFE** O SEA QUE QUEDÓ AHORA **EBFEEC6AFF68B8D24000**

La anotamos pues después la utilizaremos y hacemos el DUMP con la sentencia que tecleamos en el SOFTICE directamente

/DUMP 400000 51000 c:/Temp./dump.exe

Si al teclearla les dice Syntax Error es porque no cargaron el ICEDUMP antes.

Con eso tenemos el archivo dumpeado en C:/TEMP. Sugiero dejarlo allí y trabajar sobre una copia del mismo ya que un error nos obligara a hacer todo el DUMPEADO DE NUEVO, así que copiemos al escritorio el DUMP.EXE y dejemos guardado el DUMP.exe también en c:/TEMP por si acaso.

Lo primero que hay que hacer es arreglar un poco este coso que acabamos de copiar al escritorio, el DUMP.exe, que ni siquiera tiene un icono ya que así como esta es una porquería que ni forma tiene de tan mal que esta.

Bueno abrimos el PE EDITOR y cargamos este DUMP.exe que tenemos en el escritorio, vamos a SECTIONS y como podemos apreciar el tamaño VIRTUAL y el tamaño físico son distintos esto hay que arreglarlo ya que como lo sacamos de la memoria tiene que ser el tamaño en el DISCO igual al tamaño que ocupaba en la memoria. Por suerte tenemos un comando que lo hace automáticamente, hacemos click derecho en cualquier sección y ponemos DUMPFIXER y automáticamente se arreglaran todas las secciones y como dice allí RS=VS y RO=VO o sea RAW SIZE IGUAL A VIRTUAL SIZE y RAW OFFSET IGUAL A VIRTUAL OFFSET que en cristiano quiere decir que hace igual el tamaño de las secciones en el disco como las teníamos en la memoria y punto.

Una vez que hicimos esto vemos que si copiamos el archivo ya casi reparado a otro lugar aparece el icono del COOL MOUSE quiere decir que esta mejorando.

Lo otro que ya que estamos podemos hacer es cambiar las características de C0000040 por E0000020 para que se pueda desensamblar por si necesitamos verlo con el WDASM. Hacemos click derecho en la primera sección que queremos editar y elegimos EDIT y allí en la ventana de la derecha en CARACTERÍSTICAS cambiamos C0000040 por E0000020 para quitar la protección Antidesensamblado.

Lo último seria cambiar el ENTRY POINT por el que conseguimos o sea donde dice ENTRY POINT cambio 38001 por 876b que es el offset del nuevo Entry Point.

AHORA REVIRGIN

Como ya dijimos usaremos el revirgin, si hay algún error sepan disculpar pues yo también estoy aprendiendo esto.

Bueno arrancamos el programa comprimido para que esté en la memoria y lo dejamos una vez que arranco allí abierto, luego arrancamos el REVIRGIN (no olvidar copiar lo dos dll que trae en en C:/WINDOWS/SYSTEMS).

En la lista de procesos que están ejecutándose que esta arriba a la izquierda, buscamos el del COOL MOUSE y allí aparece CMOUSE.exe. Si no esta pueden hacer click derecho y poner REFRESH para que aparezca, si el programa esta funcionando debe aparecer. Luego seleccionamos CMOUSE.exe de esa lista y le hacemos click arriba y nos aparece un cartelito que dice **IMAGE IMPORT CORRUPTED try RESOLVER**, o sea que deberíamos ir a **IAT RESOLVER** pero esto no servirá de nada pues falta lo más importante, llenar los dos datos que hay abajo **IAT START RVA** y **IAT LENGHT**. Esos son los que necesitamos, hay otros dos IT START y IT LENGHT pero eso los llena solo el REVIRGIN el tema es ahora aprender a encontrar ese IAT STAR RVA y IAT LENGHT para comenzar a trabajar.

Paremos un poco aquí y expliquemos que es eso.

Si uno toma un ejecutable normal y que funciona, tomaremos como ejemplo el archivo NOTEPAD.exe que viene dentro de la carpeta del REVIRGIN (y que no esta comprimido) si lo arrancamos y entramos en el SOFTICE y ponemos BPX DIALOGBOXPARAMA y vamos a la AYUDA que es un SIGNO DE INTERROGACIÓN (?) y allí elegimos A PROPÓSITO DEL BLOC DE NOTAS (esta en francés) cliqueamos allí y entra en el softice, hicimos esto para entrar fácil, se puede usar cualquier otro BPX que nos deje en el ejecutable principal, ahora volvemos con F12 aceptamos en el cartel y F12 de nuevo y ya estamos en el ejecutable ya que en la línea verde dice NOTEPAD, bueno ahora sin preguntar mucho ya que lo voy a explicar después tecleemos

S 400000 l ffffffff ff,25

que es la secuencia para buscar cadenas de bits en el SOFTICE así que lo que hago es buscar la cadena FF 25 dentro del ejecutable por eso empiezo en 400000 ya que esa es la dirección de inicio del mismo.

Allí en la primera vez que busque me aparecen una serie de números de la siguiente forma en **404e72** que es la dirección donde halla **FF 25.**

FF 25 14 65 40
FF 25 10 65 40 (la direccion de memoria esta al reves es 406510)
FF 25 0C 65 40

Esta es una tabla de saltos que podemos ver si hacemos **U 404e72** allí podemos ver en el SOFTICE una tabla de saltos uno abajo del otro que en el SOFTICE figura con el nombre de la función a la que va a saltar.

En realidad no aparece pero estos saltos son

JMP [406514] **(ya que FF25 es la funcion JMP [xxxxxx])**
JMP [406510]
JMP [40650c]

y así todos, como el contenido de por ejemplo 406514 es la dirección de una función directamente si hacemos **D 406514** vemos que las cuatro primeras cifras que es el lugar donde salta son (están al revés) **7fe1589a** que es la dirección de la función **ComDlgExtendedError** por eso el SOFTICE no nos muestra el salto como **JMP [406514]** sino como **JMP [CommDlgExtendedError]** porque ya sabe que va a saltar allí.

Bueno todo muy lindo pero como encontramos la **IAT** y **IAT lenght** aquí y como podemos encontrarlo en el CMOUSE.

Bueno ya vimos que **FF 25** corresponde a la sentencia **JMP [xxxxxx]** o sea **"saltar al contenido de"** y que a donde van a saltar definitivamente todos esos saltos esta determinado por el contenido de (en el caso anterior), valores guardados en **406514, 406510, 40450c, etc,** allí guarda a donde va a saltar, pues bien ese lugar donde se guarda a donde va a saltar es la famosa **IAT.**

Y como obtenemos donde empieza y el largo que tiene?

Bueno, en este caso son pocos saltos pero conviene mirar la lista de JMP [xxxxxx] y anotar el valor máximo y mínimo de todos esos para que cuando elijamos no nos quede ninguno afuera.

O sea que si encuentro con el SEARCH todos los **FF 25** que hay uno abajo del otro, tendré que fijarme en los numeritos que hay al lado para ver cual es el mayor y el menor para tener una idea de que todos están dentro.

En el caso del NOTEPAD

FF 25 **10 65 40** JMP [406510]
FF 25 **0C 65 40** JMP [40650c]

Y así miro, la verdad es un poco incomodo tener que abrir y mirar en el SOFTICE esto pero les quería dar una idea de que hay un salto JMP al contenido de una dirección de memoria y por lo tanto esa dirección de memoria es la que determina a donde va a saltar.

Si abrimos el NOTEPAD.exe con el ULTRAEDIT y en Buscar ponemos FF25 nos aparece allí los mismos **FF 25** que vimos con el SOFTICE seguidos de los números que nos interesan

FF 25 **10 65 40**
FF 25 **0C 65 40**

O sea que desde el ULTRAEDIT en un programa descomprimido se pueden ver los saltos y en que posición de memoria están guardados a donde va a saltar, en este caso **406510, 40650c.**

Bueno busquemos esa posición de memoria aquí en el ULTRAEDIT **406510** seria **6510** en offset. Busquémosla en la columna de la izquierda donde están los offsets, hasta que lleguemos allí.

Ahora la cuestión es:

1) Saber distinguir el Comienzo de la IAT

2) Verificar que todos los JMP [xxxxxx] o sea las posiciones xxxxxx de memoria están dentro de la zona que vamos a elegir o sea:

   A) Si subimos desde 6510 un poco más arriba vemos en 62E0 como comienza la IAT la zona cambia justo allí, y antes hay varios ceros, la finalización de la zona debería ser donde hay una seguidilla de nueve ceros o más. así que si bajamos buscando una cadena de ceros, vemos que en 6De0 termina la IAT, por lo tanto:

      IAT ADRESS: 4062e0

      Iat lenght o largo seria la resta de donde termina menos donde comienza o sea, 406de0 - 4062e0 = b00

      O sea que los datos que deberíamos colocar en el revirgin para el NOTEPAD serian:

      Iat address: 62e0 (ya que es en offset)
      Iat length : b00

   B) La segunda parte era ver si entre **4062e0** y **406de0** están todas las direcciones de los JMP [xxxxxx]

      JMP [406514]
      JMP [406510]
      JMP [40650c]

      Aquí vemos que los tres valores que copie 406514,406510 y 40650c están los tres dentro de esa región IAT.

Bueno esto seria como obtener los datos de **IAT ADRESS** y **IAT lenght** en el NOTEPAD, veamos como hacerlo ahora en el COOL MOUSE.

Aclaremos ahora antes que nada que los programas ASPROTECT y comprimidos que se dicen destruyen la IMPORT TABLE lo que hacen es cambiar esta zona IAT, para que en vez de guardar en esas posiciones de memoria que apuntan directamente a las funciones (por ejemplo posiciones de memoria en la zona 7fe15891), apunten a una parte del programa que se carga al descomprimirse el ASPROTECT generalmente por la zona de C00000, y al DUMPEAR esa zona no se dumpea, ya que esta muy lejos y quedaría un ejecutable larguísimo, por lo que cuando el programa va a **JMP [406514]** en ese caso vamos a la **IAT en 406514** y el contenido es una dirección cerca de C00000 y de allí salta a la función. Es como un paso intermedio que al ejecutar el archivo DUMPEADO cuando este busca una función va a **JMP [406514]** y allí en esa posición de memoria esta por ejemplo **C00000** y como no esta en la memoria cargada esa parte nos tira error.

Por lo tanto en revirgin lo que trata de hacer es volver a restaurar los saltos en la IAT directo a las funciones sin pasar por ese error. Volviendo a poner en 406514 el contenido que haga saltar directo a la función (7fe15891) por ejemplo.

Bueno basta de teoría y volvamos al Cool Mouse y a la acción.

Teniendo cargado el Cool Mouse en la memoria y abriendo el REVIRGIN hacemos click en el proceso CMOUSE.exe.

Cuando hay que encontrar la IAT no podemos cargar el que esta protegido con ASPROTECT con el ULTRAEDIT ya que esta comprimido, entonces cargamos con el ULTRA EDIT el EJECUTABLE DUMPEADO llamado DUMP.exe y ponemos **FIND FF 25** para encontrar los saltos a la IAT y vemos que aquí hay más que en el NOTEPAD.

La primera vez que para es en **8094** y allí esta el salto a **FF25** y a continuación están los números que indican la posición de memoria de la IAT **AC D2 40** y la dirección de memoria es **40d2ac** (esta al revés).

Luego hay varios FF 25 más con distintas posiciones de memoria al lado, podemos ver la mínima y la máxima de todas esas posiciones y luego subir hasta donde empieza la IAT o con una sola la buscamos en el ULTRAEDIT y subimos para ver donde empieza.

Tomemos la primera que hallamos **40d2ac** el offset es **D2Ac**. Lo buscamos en el ULTRAEDIT y si subimos un poquitos vemos que se ve claramente que la IAT comienza en **D000** ya que antes hay una zona de muchos ceros, y busquemos el final de la zona IAT generalmente donde hay 9 ceros seguidos o más. Puedo poner la raya amarilla cliqueando en el inicio o sea en D000 e ir a FIND y poner que busque la cadena de nueve ceros o sea 000000000000000000, y como la raya amarilla esta en D000 busca a partir de allí hacia abajo, y la encuentra en **D2AE** que seria el último número antes de que empiecen los ceros.

Por lo tanto ya tenemos el comienzo de la IAT y el final de la IAT. Si chequeamos vamos a ver que los FF 25 que aparecen en la tabla de saltos caen todos aquí dentro.

Restamos para saber el largo de la IAT, **D2AE - D000 = 2AE.**

O sea que debemos colocar en el REVIRGIN como datos **IAT address D000** y **IAT lenght 2AE.**

Con esos datos podemos ya trabajar con el revirgin.

Volvemos al revirgin y por supuesto tiene que estar activo el programa COOL MOUSE cuando hacemos click en el nos dice que la tabla esta corrupta. Ponemos **0000d000** y **000002ae** y hacemos click en IAT RESOLVER. Luego nos pedirá que carguemos el ejecutable dumpeado DUMP.exe que tenemos en el escritorio, y con eso nos dará un primer resultado sobre la IAT a la derecha con los nombres de las funciones y las direcciones de memoria. Miremos un poco eso.

Hay algunos casilleros que ya están resueltos y completos, y hay algunos que dicen redirected y hay uno (el último) que esta vacío. Los que dicen redirected ya están detectados, para que termine de llenar bien las casillas correspondientes le damos a **RESOLVE AGAIN.**

Bueno ya tenemos la tabla casi completa nos quedan dos lugares sin llenar, el que corresponde a **B0c218** y la ultima casilla que corresponde **B2246c.**

Como vemos en estos dos saltos que quedan el programa protegido con ASPROTECT salta a B0c218 y B2246c, mientras que las otras casillas ya

han sido reparadas y saltaran directamente al valor de la función o sea **BFc03e8e** por ejemplo, que es un valor mayor. Si nosotros no reparáramos estas dos entradas que quedan y dejáramos la tabla así cada vez que el programa llegara a una llamada a (por ejemplo en la ultima casilla)**JMP [40d2ac]** ya que el contenido de **40d2ac** es **b2236c** y como el programa dumpeado no llega hasta allí, nos daría error. Hay que buscar reemplazarla por la función directa correspondiente y el salto a la zona de las funciones.

Bueno tomemos la ultima y marquémosla para que resalte, hagamos click derecho encima y elijamos TRACE al tracear ya nos cambia el valor de **b2236c** por **7fe16112** que es la zona de las funciones y nos aparece traced. Ahora hagamos RESOLVE AGAIN y vemos que ya se completa esa casilla con el número de la función GET OPEN FILE NAME.

Tratamos de hacer lo mismo con el otro casillero vacío que tenemos por el medio y como dice redirected no acepta que traceemos. Aquí ya el programa no nos ayuda más y tendremos que ir a fijarnos con el SOFTICE que función es la que usa el programa cuando llega a B0C218.

Como el programa esta funcionando pongamos un **BPX DIALOGBOXPARAMA** o cualquier otro que nos haga entrar al ejecutable y hagamos click derecho en el iconito del mouse que aparece en la barra de tareas, y elijamos SETTINGS, con eso parara en el SOFTICE, luego hagamos F12 y cuando aparezca la ventana pongamos OK y caeremos de nuevo en el SOFTICE, justito dentro del ejecutable CMOUSE.

Ahora hagamos **U b0c218** para ver que función llama el programa allí, y vemos que unas líneas más abajo aparece **KERNEL 32 - GETPROCADDRESS.** Volvamos al REVIRGIN y hagamos click derecho encima de la entrada que no esta terminada y pongamos EDIT, y escribamos el nombre de la función GetProcAddress. Ponemos un BPX en B0c218, y arrancamos el programa de nuevo y cuando pare allí tracemos hasta ver la dirección de la función GETPROCADDRESS y es **Bff76da8.**

Reemplazamos en el REVIRGIN y entonces escribimos en lugar de B0c218, el nuevo valor y KERNEL32.dll y GetProcADDRESS. Hacemos RESOLVE AGAIN (hay veces que cuando pones el valor de la funcion solo y haces RESOLVE AGAIN te completa solas las otras dos casillas sin necesidad de buscarlas a mano).

A esta altura antes de seguir conviene ir a SAVE RESOLVED y guardar un archivo con la tabla como esta hasta ahora, por cualquier error la podemos cargar con LOAD RESOLVED de nuevo.

Una vez que ya completamos todas las casillas vamos a IAT generator y si hay algo incompleto aun, nos dirá que hay algo todavía sin resolver, sino no nos dirá nada. Hay que tener cuidado también que la entrada que escribimos a mano que no tenga errores ortográficos ya que a mi me paso que escribí “GetProcadres” con una sola “d” y una sola “s” y me aparecía que estaba incompleto siempre.

Bueno vamos a IAT GENERATOR. Aquí no me dice nada de que hay algo mal por lo tanto sigo, me pide que le ponga un nombre para archivar la tabla y le pongo por ejemplo ITdump.exe, pero a la vez que lo guarda en un archivo, también lo agrega al archivo dumpeado. Si ven los otros dos valores IT e IT LENGHT nos dice donde lo va a agregar, IT 51000, o

sea al final del archivo que terminaba en 51000 va a agregar el largo 108 en mi caso probablemente en una nueva sección.

Bueno supuestamente con eso ya debería funcionar el archivo DUMP.exe. Lo colocamos en el directorio donde esta el original Cmouse.exe, lo hacemos para probar pero nos habíamos olvidado de cambiar los bytes que modificamos para dumpear en el inicio cuando hicimos JMP EIP y debemos cargar el Dump.exe con el ULTRA EDIT para volverlos a su valor original.

En fin, pongo la cadena **EBFEEC6AFF68B8D2** Y CAMBIO **EBFE** POR **558B** que eran los que se encontraban originalmente en el programa. Una vez que guardamos los cambios, corremos el Dump.exe y chan chan ERROR DE WINDOWS... que pasa aquí?... veamos!!!

Pongamos SET FAULTS OFF y veamos donde fue en error. Según sale en el cartelito del error es en **40821c** el tema. Si ponemos un **BPX 40821c** y para allí vemos que no hay ningún llamado a una función por lo tanto no es un error de que este mal hecha la IAT. Lo más probable es que allí se este verificando si esta presente el ASPROTECT en memoria como una protección por si alguien lo dumpea y lo arregla. Por lo tanto, NOPEAREMOS esta instrucción a ver que pasa.

Vamos al ULTRAEDIT y allí en 40821c esta la cadena **8a014184c07440f7c1**. Reemplazamos **8a 01** por **90 90** y guardamos los cambios. Ejecutamos y VOILA arranca y funciona perfectamente sin ningún error.

De cualquier manera esta es recién una introducción al tema REVIRGIN en el cual aprenderemos juntos. Hay muchísimos compresores diferentes y aunque el método es siempre parecido lo iremos perfeccionando y buscando como aplicarlo en otros casos más difíciles. Ahora que el ejecutable esta descomprimido y se puede parchear directamente, les dejo la tarea de registrarlo para ustedes. Yo me despido hasta la LECCIÓN 35 y vamos a tratar de seguir con otro programa que haya que reparar con el revirgin, así le vamos tomando la mano a usar esta buenísima pero difícil herramienta.

Nos vemos en la 35
Ricardo Narvaja

LECCIÓN 35: IAT ADDRESS E IAT LENGTH AUTOMÁTICAMENTE

Esta no es una lección completa sino un pequeño apéndice de la LECCIÓN 34 donde aprendimos a utilizar el REVIRGIN y vimos también que quizás lo más difícil era localizar la dirección donde comienza la IAT y el largo.

Bueno, ya no es necesario buscarlo a mano, pues hay un programa que lo hace automáticamente, y te dice la IAT ADDRESS y la IAT LENGHT; se llama **Import REConstructor v1.2**

Yo lo baje de

http://www.suddendischarge.com/files/Unprotection%20Tools/imprec12.zip

o también esta en

http://www.suddendischarge.com/

Lo buscamos en el SEARCH por su nombre y allí esta.

Este programa se supone que hace lo mismo que el REVIRGIN, aunque a mi no me dio tan buen resultado para terminar de reconstruir bien la tabla de importaciones. Por lo menos automáticamente nos encuentra la IAT address y IAT lenght necesarias para trabajar en el REVIRGIN.

El que quiera probarlo para reconstruir completamente la tabla de importaciones, hágalo, quizás funcione bien, pero la costumbre de utilizar el REVIRGIN además de que este Import REConstructor v1.2 suele colgarse bastante, hace que lo use solo para que me diga los valores que necesito y llevarlos al Revirgin para continuar allí con el trabajo.

Cuando lo ejecutamos al IMPORT, vamos a ATTACH TO AN ACTIVE PROCESS ya que como en el Revirgin el proceso a reconstruir debe estar funcionando en la memoria, y allí elijo el que quiero estudiar.

Arranquemos el COOL MOUSE original que no esta desprotegido para ver que nos dice sobre la IAT LENGHT y ADDRESS.

Busco entre los procesos el CMOUSE.exe y lo cargo, una vez que se carga coloco el ENTRY POINT que halle al dumpear en la casilla OEP en este caso es **40876b**, o sea debemos llenar la casilla con el OFFSET por lo tanto será **876b** lo que coloquemos en la casilla OEP y luego iremos a IAT AUTOSEARCH. Cuando apretemos ese botón nos aparecerá en **RVA=d000** y **Size=2b4** que son los valores donde comienza la IAT y su largo como vimos en la LECCIÓN 34 averiguándolos a mano.

IAT ADDRESS=D000 y IAT LENGHT=2b4

En la LECCIÓN anterior habíamos visto "O sea que debemos colocar en el REVIRGIN como datos **IAT address = D000** y **IAT lenght = 2AE**"

Aquí vemos que el lenght lo calcula un poco más largo justo entre los ceros donde termina la tabla, pero funciona igual, y si ponemos estos valores en el REVIRGIN vemos que la tabla que calcula es exactamente la misma, siguiendo el método que vimos en la LECCIÓN ANTERIOR.

La idea era simplificar un poco la forma de encontrar la IAT ADDRESS e IAT LENGHT ya que como vimos en la LECCIÓN anterior explicar como obtener esos datos me llevo casi 2 paginas bastante difíciles. Aquí en el programa con solo ponerle el dato del ENTRY POINT, que ya lo tenemos, nos calcula directamente los valores necesarios sin rompernos la cabeza ni buscar como locos a mano.

Bueno, espero que esto le facilite las cosas al que quiera reconstruir una tabla de importaciones, no lo puse en la LECCIÓN ANTERIOR sencillamente porque no lo sabía en ese entonces y me enteré recién ahora. Igual no les va a venir mal saber hallar a mano esos valores.

Hasta la 36
Ricardo Narvaja

LECCIÓN 36: EN LA CARA DEL NUEVO ASPROTECT 1.2

Seguimos por varias lecciónes más con el tema Revirgin, y ahora haremos la reconstrucción de el ejecutable de un programa protegido con el último y nuevo e imbatible (hasta ahora) ASPROTECT 1.2 y que mejor para probar si podemos con el que mojarle la oreja al propio ASPROTECT, reconstruyendo el ejecutable del mismo ASPROTECT 1.2 que esta protegido con ASPROTECT 1.2, jua jua.

Se baja de http://www.aspack.com/files/asprotect12.zip

El primer paso es arrancar el ICEDUMP, una vez que esta cargado, tenemos que hallar el ENTRY POINT y aquí en esta versión esta un poco más complicado, pero se puede hacer:

Aquí puedo usar un BPX GETVERSION o BPX GETPROCADDRESS, cualquiera e los dos a la tercera o cuarta vez que para y volvemos al ejecutable con F12, nos damos cuenta que estamos en el momento que se esta descomprimiendo ya que en la línea verde que siempre aparece el nombre del programa que esta ejecutándose aquí no aparece nada, o sea sin nombre, y si vemos con el PEEDITOR las secciones del ejecutable vamos a ver que algunas no tienen nombre, es obvio que esta ejecutándose alguna de estas secciones.

Usamos el famoso **BPR INICIO FINAL r if (eip>=INICIO) && (eip<=FINAL)**

Dado que aquí esta en alguna de esas secciones sin nombre, podemos pensar que en ENTRY POINT estará en la primera sección y casi siempre es así, usaremos pues como INICIO = 400000 y FINAL el final de la primera SECCION o sea 459000, lo que podemos ver con el PEEDITOR, probaremos:

BPR 400000 459000 r if (eip>=400000) && (eip<=459000)

Bueno, largamos y vemos que la primera vez que para es en **401014 RET**, pero es obvio que este no es el ENTRY POINT ya que no hay programa que comience en un RET (creo, jua jua), así que le doy a X y ENTER de nuevo para que siga ejecutándose y para la segunda vez en

458f04 PUSH EBP

Este tiene un buen aspecto pues es un PUSH EBP, pero ASPROTECT tiene sus tretas así que ejecutemos cinco o seis sentencias para ver que pasa con F10, llega a un RET y vuelve al compresor sin nombre, jua, jua, es un ENTRY POINT falso, puesto para despistar.

Vuelvo a hacer X y ENTER y parará en **458f14** y **458ef4**. Todos **PUSH EBP** y todos falsos ya que enseguida llegan a un **RET** y vuelven al compresor lo que nos damos cuenta ejecutando cinco o seis líneas.

Al fin LLEGA a **458fD8 PUSH EBP**

Este es el verdadero ENTRY POINT ya que no vemos ningún RET y si ejecutamos vemos que no vuelve nunca más al COMPRESOR SIN NOMBRE.

Bueno, una vez que para aquí tenemos que DUMPEAR. Ya comprobé que no es necesario con el ICE DUMP hacer el LOOP infinito **JMP EIP**, así que no necesitamos cambiar nada, una vez que este en 458fd8 escribimos:

/DUMP 400000 49e000 c:/Temp/asp.exe

El **400000** y **49e000** lo sacamos del PEEDITOR son la **IMAGE BASE** y el largo total **SIZE OF IMAGE 9e000.** Por lo tanto si dumpeamos la memoria entre 400000 y 49e000 volcaremos todo lo que tenemos en la memoria.

Anotemos este valor de **ENTRY POINT = 458fd8** para colocarlo en el IMPORT RECONSTRUCTOR para que nos calcule el IAT ADDRESS y IAT LENGHT o sea la dirección donde comienza la IAT y el largo.

Usando el Import Reconstructor como vimos la LECCIÓN pasada, buscamos donde dice ATTACH AN ACTIVE PROCESS el proceso del Asprotect.exe y una vez que lo abrimos ponemos en OEP el ENTRY POINT hallado, **en offset 58fd8** y al cliquear en IAT AUTOSEARCH obtenemos los valores que necesitamos

RVA = 66114 y SIZE = 5CC

Estos son los valores que necesitamos en el REVIRGIN para usarlo

IAT ADDRESS= 66114 y LENGTH O SIZE= 5CC

Cierro el IMPORT RECONSTRUCTOR y abro el REVIRGIN, y busco allí el proceso Asprotect.exe que debe estar abierto.

Coloco esos datos en **IAT ADDRESS = 66114 Y LENGHT = 5CC** y le doy a IAT RESOLVER. Me pide el nombre del ejecutable dumpeado que ya arregle con el PEEDITOR como vimos en las lecciónes anteriores (o sea, abro las secciones y pongo DUMPFIXER para que repare el archivo DUMPEADO), y luego le doy nuevamente a RESOLVE AGAIN.

La verdad la tabla queda bastante completa, se arregla bastante bien, solo una entrada queda sin solucionar, es la de **C1c424** que ni traceando te la arregla.

Bueno arrancando de nuevo el programa original y haciendo que pare en el ENTRY POINT, allí pongo **bpx C1c424** para ver que pasa allí, y entonces veo que unas líneas más abajo está **GetProcAddress** y al entrar en ese CALL la dirección de la función es **BFF76DA8.**

Bueno llevo esos datos al REVIRGIN pongo la tilde en EDIT y lleno los casilleros vacíos, el primero con **KERNEL32.dll** luego **GetProcAddress** y donde dice **C1c424** pongo **Bff76da8.**

Hago Resolver y lo toma, uso la opción **AUTOFIX SECTIONS + IT PASTE** tildada como vimos las lecciónes anteriores, voy a IAT GENERATOR y me dice donde quiero guardar la tabla. La guardo en cualquier lado, pero a su vez me la agrega al archivo dumpeado que ya había hecho con el ICEDUMP y reparado con el PEEDITOR. Además, por si acaso, me hace una copia con extensión bak del archivo como era original por si algo sale mal.

Bueno con eso y luego de arreglar el entry point del archivo y colocarlo en **458fd8** o sea **58fd8 en offset** y copiar el archivo

arreglado a la carpeta donde esta el ejecutable original, lo arranco y salen errores en varios lugares.

Lo miro con el WDASM luego de cambiar las características de la sección a E0000020 con el PEEDITOR y veo que las Funciones Importadas aparecen perfectas e impecables.

Bueno desde el SYMBOL LOADER me dispongo a tracear para reparar lo que es seguramente alguna protección de ASPROTECT para que de error si alguien lo arregla y veo que en los lugares que da error es un **CALL [464fac]** o **CALL [464fb0]** y que si nopeo estos calls arranca perfectamente el programa y funciona bien.

Puedo buscar con el ULTRAEDIT la secuencia de estos calls ya que siempre son iguales para saber donde están y nopearlos.

Ya que esos CALL tiene los siguientes valores HEXA, en el ULTRAEDIT pongo FIND y busco en que lugares aparecen:

FF15ac4f4600
FF15b04f4600

Y veo que hay varios lugares donde esta, entre otros:

45495f, 454b3a, 453437, 45777d, 458cc6, 4579c4 y **453480**

Nopeando en estas direcciones de memoria las llamadas de estos CALL que te direccionan a **Cxxxxx** el ejecutable funciona perfectamente. Es obvio que estos CALL están puestos para que si alguien lo descomprime le de error, por que nopeandolos funciona perfecto el ejecutable y arranca con todas las funciones activas.

Bueno con esto ya esta limpito, limpito de protección Asprotect, les queda a ustedes la tarea de crackearlo.

Costo pero valió la pena, jua.

Gracias a METAMORFER que me enseño como hacerlo.

Ricardo Narvaja
Hasta la LECCIÓN 37

LECCIÓN 37: SEGUIMOS APRENDIENDO A USAR REVIRGIN

Esta vez es el turno de otro programa protegido con ASPROTECT, ya que más adelante veremos otras protecciones. Este se baja de

http://www.helexis.com/ic/icatc303.zip

Se llama ICON CATCHER y la versión 3.03 viene protegida con ASPROTECT. Aquí en este caso, aprovecharemos dado que la reconstrucción de la tabla no es difícil, sino la verdadera dificultad esta en hallar el ENTRY POINT.

Preparo el ICEDUMP para que este listo y poniendo un **BPX GETPROCADDRESS**, una vez que arranca el programa y para dentro del SOFTICE a la tercera vez que para ya nos aparece en la esquina inferior derecha IC que significa que paro dado el proceso ICON CATCHER.

Si antes miramos en el PEEEDITOR donde comienza y donde finaliza la primera sección para trasladar estos datos al SOFTICE escribiendo el famoso **BPR INICIO FINAL r if (eip>=INICIO) && (eip<=FINAL)**, en este caso **INICIO = 400000** y **FINAL = 49d000.**

Por lo tanto borramos el BP que había con BC* y luego escribimos **BPR 400000 49d000 r if (eip>=400000) && (eip<=49d000)** y le damos X para ver donde para.

Anoto estos lugares donde fue parando

401000 RET no es un entry point
401005 RET lo mismo
401014 RET ídem
48e38c PUSH EBP

Este parece ser un ENTRY POINT pero si mantenemos apretado F10 vemos que después de un tiempo vuelve al compresor por lo que no es el ENTRY POINT.

Vuelvo a ejecutar con X y para en **40687c PUSH EBP.** Este parece más, ya que si mantengo apretado F10 no vuelve al compresor y cuando sale del ejecutable es para encontrar alguna función.

Yo Dumpee aquí habiendo buscado el **INICIO = 400000** y **SIZE OF IMAGE = E2000** con el PEEDITOR.

/DUMP 400000 e2000 c:/Temp./icon.exe

Con el Import Reconstructor y el programa ICON CATCHER funcionando, le puse el ENTRY POINT hallado en **offset 687c** y me dio la **IAT ADDRESS = a1164 y SIZE=738** que es correcta ya que si vemos con el ULTRA EDIT y buscamos con el método de FF 25 para verificar, vemos que los saltos caen en esa posición de memoria (4a1164).

Abro el REVIRGIN y busco el proceso de ICON CATCHER y le pongo **IAT ADDRESS=a1164** y el tamaño **IAT LENGHT=738.**

Me sale una primera tabla, me pide el ejecutable dumpeado lo busco, y luego le doy a RESOLVE AGAIN para que termine, salen solo dos entradas sin resolver que ni traceando se resuelven.

Las dos apuntan a **C6c424** así que la función es la misma en ambas. Tengo que fijarme corriendo el programa original adonde va cuando llega a **C6c424** así que lo ejecuto y pongo por ejemplo algún BPX que pare o sino **Bpr 40687c 40687d r** que en unas dos o tres veces parara en **40687c el ENTRY POINT.**

Una vez allí me fijo con **U c6c424** y veo que la función es **GetProcAddress** lo único que me quedaría es poner un BPX alli para ver la dirección exacta cuando entra. Así que pongo un **BPX c6c424** y corro el programa. Cuando para allí traceo con F10 y cuando entra a la función copio la dirección de memoria, en mi caso es **Bff76da8** aunque puede variar según la maquina.

Vuelvo al Revirgin y pongo la tilde en EDIT y completo a mano los cuadros vacíos sin errores (sino no funcionara). En el primero pongo **Kernel32.dll** el número no lo se, así que pongo 0001 (el programa lo corregirá solo). Pongo el nombre de la función **GetProcAddress** y la dirección reemplazo **C6c424** por **Bff76da9** o lo que obtuvieron ustedes y le doy a RESOLVE AGAIN y lo toma. Es de notar que si a veces dejo 0000 en el número, no toma la función y vuelve a poner redirected, igual cambió solo el 1 por el valor correspondiente que es 1A3 en mi caso.

En la otra entrada copiamos los mismos datos ya que va al mismo lugar. Luego de RESOLVER AGAIN y que esta todo lleno, vamos a IAT GENERATOR y ya esta listo, cambiamos el ENTRY POINT con el PEEDITOR ponemos **687c**, supuestamente debería estar listo. Lo copio a la carpeta del programa y lo arranco y no funciona. Por que?

No es ninguna protección de ASPROTECT ni nada de eso, la tabla esta correcta y no hay errores allí, porque no funciona?

El ENTRY POINT no es el correcto, como lo supe?. Después de mucho bregar me di cuenta que apretando F10 el ejecutable se seguía descomprimiendo aunque sin volver al compresor ASPROTECT, como si estuviera comprimido dos veces, primero con Asprotect y ahora sin salir del ejecutable, lo que hace inservible el método del BPR para seguir a partir de aquí.

Vencidos, casi, pero alguien (GRANDE VIPER) me soplo una ayuda: El DEDE te dice el ENTRY POINT si el programa es en DELPHI, tendremos suerte?

Abro el DEDE voy a TOOLS - DUMP ACTIVE PROCESS y allí vemos que la mayoría de los programas que están funcionando cuando los cliqueamos no se activa el botón **GET RVA ENTRY POINT**. Si hacemos click encima del Icon Catcher si se activa, ya que solo se activa con los programas escritos en DELPHI y este lo es. Aprieto el botón y me dice el ENTRY POINT que es **49d22c.**

Tengo que realizar el dump nuevamente así que repito el proceso, con **BPX GETPROCESSADDRESS** y allí puedo hacer que pare en **49d22c** con **BPR 49d22c 49d22d R**

Así después de tres o cuatro veces parará en el **ENTRY POINT 49d22c.** Allí dumpeo igual que antes, arreglo con el PE EDITOR y la opción DUMPFIXER, cambio el ENTRY POINT al nuevo, realizo el mismo trabajo para con el Revirgin agregarle la IMPORT TABLE exactamente igual que antes.

Y una vez que voy a IAT Generator, y guardo la tabla nueva, el ejecutable sin ninguna reparación posterior copiándolo a la carpeta del programa funcionará perfectamente sin problemas y como siempre en estos últimos ejemplos les dejare el tema de registrarlo para que practiquen, ya que no es muy difícil una vez desprotegido de ASPROTECT.

Hasta la LECCIÓN 38
Ricardo Narvaja

LECCIÓN 38: EL ASPROTECT MÁS DIFÍCIL QUE VI.

CONTROL FREAK para usar el Revirgin, brrr.

Asprotect 1.2 tiene varias versiones parece y dado que ya hemos desprotegido algunos programas compactados con el, veremos este que parece ser una versión especial, más difícil que la común, quizás al pagar más haya versiones más difíciles, no lo se.

La cuestión de que arreglar la tabla de importaciones de este me costo bastante más que cualquiera de los otros ASPROTECTS y fue muy difícil, ni siquiera estoy seguro de que sea ASPROTECT ya que no es reconocido por ningún identificador al día de hoy, pero es casi seguro, igual lo descomprimiremos.

La pagina del programa es http://www.hace.us-inc.com/ aunque ya existe allí la versión 1.01 que no debe diferir mucho de esta, pero será distinta así que subiré la 1.0 al Freedrive para el que quiera practicar.

Bueno, con el PEDITOR busco la **IMAGE BASE = 400000** y la **IMAGE SIZE = 8b000**

Arranco el ICE DUMP, y con **BPX GETVERSION** o **BPX GETPROCADDRESS** caigo dentro del SOFTICE, y allí después de dos o tres veces al volver con F12, quedare en el descompresor ASPROTECT aunque aquí tampoco aparece ningún nombre en la línea verde donde dice el nombre de lo que se esta ejecutando.

Bueno la primera sección empieza en 400000 y termina en 466000, podemos ver con el PEEDITOR así que hacemos con el famoso BPR.

BPR 400000 466000 r if (eip>=400000) && (eip<= 466000)

Parará en **465654** que es el verdadero ENTRY POINT, allí dumpeo.

/DUMP 400000 8b000 c:/Temp./cfreak.exe

Reparo con el PEEDITOR y DUMPFIXER al archivo DUMPEADO, y cambio el ENTRY POINT a **65654 offset** así ya quedara listo cuando le arregle la tabla de importaciones.

Voy al IMPORT RECONSTRUCTOR, elijo el proceso del CONTROL FREAK y pongo el **ENTRY POINT = 65654** y me sale **IAT ADRESS = 69150** y **IAT LENGHT = 673**.

Traslado estos datos al Revirgin y abro el proceso del CONTROL FREAK, y hago RESOLVER. Cargo el dumpeado, y luego RESOLVE AGAIN.

Los resultados no son muy alentadores, la tabla tiene diez posiciones sin resolver, el máximo de lo que he visto hasta hoy con ASPROTECT.

Con el método de ir a fijarse al programa original uno por uno, cargando el programa y poniendo primero **BPR 465654 465655 R** para que pare allí, en el ENTRY POINT y una vez que pare, pongo BPX en las direcciones que nos dice el REVIRGIN que va el programa.

000691A4 00C0C468 0000 KERNEL32.dll GetProcAddress

Aquí debemos buscar en **COC468** y cuando entra en la función, en mi caso la dirección de memoria es **bff76da8.** Reemplazo C0c468 por esa dirección que encontré y cambio el número **0000** por **0001** y le doy a **RESOLVE AGAIN**, y ya esta solucionada esta entrada, así soluciono:

691a4 GetProcAddress bff76da8
691a8 GetModulehandleA bff77716
69250 GetModulehandleA bff77716
69314 GetProcAddress bff76da8
69318 GetModulehandleA bff77716

La ultima columna puede variar según cada caso, hay que hallarlo uno mismo.

Ahora quedan cinco que si pongo un BPX y voy a ver no hay ninguna función...

691b8 c0686c
692c8 c0c874
69300 c0c834
69334 c0c864
6933C c0c87c

Estas son las entradas que no se pueden arreglar. No importa, una vez que arreglo las cinco primeras voy a IAT GENERATOR y aunque me dice que no esta terminada de solucionar acepto igual.

Entrando de nuevo en el programa y poniendo BPX en las direcciones de la segunda columna para ver que pasa, copiaremos las sentencias que hallamos.

```
---------------------------------------
C0C86C  A17836C100    MOV EAX, [c13678]
        C3            RET
---------------------------------------
C0C834  A1D835C100    MOV EAX, [C135D8]
        C3            RET
---------------------------------------
C0C864  A1E035C100    MOV EAX, [C135E0]
        C3            RET
---------------------------------------
C0C874  55            PUSH EBP
        8BEC          MOV EBP, ESP
        5D            POP EBP
        C20400        RET 0004
---------------------------------------
C0C87C  55            PUSH EBP
        8BEC          MOV EBP, ESP
        5D            POP EBP
        C20400        RET 0004
---------------------------------------
```

Estas son las cinco entradas que no se pueden RESOLVER, y el programa las necesita ya que si el DUMPEADO busca en el primer caso por ejemplo ir a **C0c86c** cae a error ya que esa parte no esta en el DUMP, y tampoco

si ponemos un RET el programa arranca. Tenemos que tratar de que ejecute las mismas sentencias que el programa original.

Nos queda otra cosita, veamos el primer caso:

```
---------------------------------------
C0C86C   A17836C100   MOV EAX, [c13678]
         C3           RET
---------------------------------------
```

Tenemos que ver cual es el contenido de c13678, para tomar ese valor constante y hacer que EAX, tome ese valor directamente sin buscar en el contenido de c13678 lo cual también daría error.

Vamos de nuevo allí con el SOFTICE y vemos que **(en mi caso ya que este número varia en cada computadora)** EAX toma el valor 81819974.

O sea que si yo, en donde se encuentra el salto en el DUMP reemplazo el salto a **C0c86c** por directamente **MOV EAX, 81819974** y **RET**, el programa funcionaria de la misma forma que el original.

Busco en el ULTRA EDIT el FF 25 del primer caso que es FF 25 b8 91 46 **(las tres ultimas cifras son 4691b8 al revés que es el dato del Revirgin 691b8 de esa primera entrada).**

Casualmente **FF 25 b8 91 46 00** que es **jmp [4691b8]** tiene la misma cantidad de bytes, por lo tanto lo reemplazo por **b8 74 99 81 81 c3** que es **MOV EAX, 81819974** y **RET**, con lo que el programa funcionaria igual.

En la segunda entrada, **FF 25 00 93 46 00** que es **JMP [469300]** lo reemplazo por **b8 04 0a 00 c0 c3** que es **MOV EAX, c0000a04** y **RET**.

C0000a04 es el número que busque con el SOFTICE que mueve a EAX.

En la tercera entrada, **FF 25 34 93 46** lo reemplazo por **B8 0b cb 06 ff c3.** En mi caso **C3 ff 06 cb** es el número que pasa a EAX.

En los dos últimos casos YA QUE AQUÍ NO HAY TRANSFERENCIA DE NINGUNA POSICIÓN DE MEMORIA SINO SOLO DE REGISTROS ENTRE SI, BUSCO LA CADENA **55 8B EC 5D C2 04 00** EN EL DUMP Y APARECE EXACTAMENTE IGUAL EN **416040** O SEA **OFFSET 16040.** SI REDIRIGIMOS ESTAS DOS ENTRADAS ALLÍ FUNCIONARA BIEN.

YA QUE LOS DOS ÚLTIMOS SALTOS SON

JMP [4692c8] y **JMP [46933c]**

Cambiando los contenidos de **4692c8** y de **46933c** por **416040** saltara dentro del dump y ejecutara los mismos comandos.

Voy al ULTRA EDIT y busco el offset 692c8. Allí encuentro 74 c8 c0 que era donde saltaba originalmente (c0c874), lo cambio por 406041 y en el segundo caso igual y habremos logrado terminar.

Con eso en la propia computadora de uno funcionara, pero se me ocurrió instalar el programa en otra computadora y llevar el ejecutable reparado a la otra maquina y no funciono. Comparé los números que

lleva a EAX en las tres entradas que arreglamos y son diferentes. Si los cambio por los que genera en esa maquina arranca allí también.

Probé además que los dos números últimos pueden ser cualquiera, el tema estaba en la primera entrada.

MOV EAX, 81819974 y **RET**

Aquí era donde se determinaba si funcionaba en otra maquina o no, ya que más adelante tomaba el contenido de 81819974 y daba error ya que era una zona invalida. Probé a cambiar el número 81819974 por 00465654 que es el ENTRY POINT y siempre su contenido tendrá un número valido, y arranco en las dos maquinas, jua jua, ese era el truco. La verdad que no vi hasta ahora ningún caso de reconstrucción de tabla tan difícil como este, si alguien consigue un método más fácil, please, avíseme, jua jua.

Hasta la LECCIÓN 39

Ricardo Narvaja

LECCIÓN 39: CRACKEANDO EN 16 BITS. PARECE REFÁCIL Y ES REFÁCIL... PERO QUE MOLESTO QUE ES

Bueno ya que no pude terminar lo que había prometido para esta semana, lo dejamos para más adelante y seguimos con otro tema pendiente, crackear en 16 bits.

Ojo esto no es solo un tema pendiente en las lecciónes porque si, yo odio crackear en 16 bits, le esquivo como a la viruela, y tengo más dudas que certezas en este tema, y no creo ser el único, muchos a los que consulte sobre el tema me ignoraron olímpicamente, muy probablemente por que tampoco les es cómodo.

Ahora POR QUE ES DIFÍCIL?

ES VERDADERAMENTE DIFÍCIL? O ES LA FALTA DE COSTUMBRE. Ante casi el 95 por ciento de programas que son en 32 bits, el 5 por ciento restante solamente es en 16 bits. Como no estamos acostumbrados nos cuesta? No se, a mi me cuesta horrores y tengo que usar más trucos y cosas raras que nunca, quizás por desconocimiento, y si alguien sabe de alguna forma mejor de hacer esto que me lo diga así aprendo.

Este es en las casi 40 lecciónes el primer programa en 16 bits que crackeamos y tampoco vamos a crackear uno todos los días, pero por ser el primero busque uno muy fácil (para mi) ya que solamente consiste en eliminar un cartel.

Se baja de http://www.boennrep.de/dl0206/BRep450.exe y es un programa sobre medicina y síntomas que funciona 100% solo que cada tres segundos aparece un cartelito molesto que no te permite usarlo con tranquilidad. Lo tenés que estar cancelando y a los dos o tres segundos aparece de nuevo. El tema es eliminar ese cartel molesto.

Hay dos tipos de programas de 16 bits o modo DOS que yo conozco; los que funcionan dentro de WINDOWS y los que lo hacen si o si fuera de WINDOWS.

Este y todos los que veremos estarán en el primer caso o sea dentro de WINDOWS, los del segundo caso, tendrán que buscar otro profesor, jua jua.

Si ya para un programa de 16 bits que funciona en WINDOWS no funcionan la mayoría de las herramientas (PROCDUMP, ni PEEDITOR, ni PUPE, ni DEDE, ni REGMON, ni FILEMON, ni muchísimas más) para los que funcionan fuera no funciona ninguna de las conocidas. Solo hay una versión de SOFTICE vieja para DOS que es más mala que la peste y si el programa te la detecta no hay FROGSICE que te salve, hay que hacer todo a mano.

Por lo tanto reagrupemos las fuerzas y veamos con que contamos:

- Con el SOFTICE
- Con el WDASM
- Con el ULTRAEDIT

Creo que otra cosa no funciona para 16 bits, nos tendremos que arreglar.

COMO DIFERENCIAMOS UN PROGRAMA DE 32 bits de uno de 16 bits? (valor ya no quedan muchos y cada vez son menos). De esta forma:

Esto es sacado del WDASM de un programa de 16 bits

16 bits

```
:0010.13A7   668B460E   mov eax, [bp+0E]
:0010.13AB   050800     add ax, 0008
:0010.13AE   6650       push eax
```

32 bits

```
:00403F3A    57         push edi
:00403F3B    50         push eax
:00403F3C    8D4505     lea eax, dword ptr [ebp+05]
```

Aquí se ve la diferencia, en el de 32 bits la dirección de memoria es un número de 8 dígitos por ejemplo **00403f3a** y en el de 16 bits es un número de cuatro dígitos precedido de un número de bloque, por ejemplo **0010:13a7.**

Esto que parece tan simple tiene sus cuitas, ya que en 32 bits el número te indica ya directamente una posición. Si en el WDASM ves **403f3a** sabes que en el SOFTICE va a ser **403f3a** y la encontrarás fácil; en cambio en 16 bits la cosa cambia.

El WDASM empieza a contar los bloques del programa desde 0001, o sea, desde el primer bloque y el SOFTICE no. En la computadora ese programa no esta en el primer bloque sino en el **5375** (POR EJEMPLO). Por lo tanto, la cosa se pone peliaguda.

Si el Wdasm dice **0001:1345** solo se que será **1345** pero **vaya a saber de que bloque** (SI ALGUIEN SABE CALCULAR EXACTAMENTE COMO ENCONTRAR FÁCILMENTE UN BLOQUE DEL WDASM EN EL SOFTICE SIN ANDAR TANTEANDO Y SIN USAR EL SEARCH AYUDE A ESTE POBRE CRACKER EN DESGRACIA).

En resumidas cuentas si yo se que es **1345** en el WDASM y lo tengo que encontrar en el SOFTICE puedo hacer la siguiente trampa:

Correr el programa poner algún BPX que se que use el mismo y una vez adentro, buscar la secuencia de números hexadecimales (cadena) que en el WDASM corresponden a esa sentencia y en el SOFTICE buscarla

S 0 l ffffff 44,55,66,77, etc

O sea, **44 55 66 77 etc** es la cadena de valores hexadecimales de la sentencia que encontré en el WDASM, y si es al revés pasar del SOFTICE al WDASM busco, si no son muchos bloques, uno por uno.

Por ejemplo, si era **5324:1234**, busco en **0001:1234** veo si están las mismas sentencias, y así sigo con **0002:1234** hasta que las encuentro.

Esto quizás parecerá un poco precario, pero hasta hoy nadie me dio ni explico una forma mejor de hacerlo (ESCUCHO SUGERENCIAS, EH?)

Bueno empecemos con el programa. Lo desensamblo con el WDASM al ejecutable y, encima que tarda como media hora, una vez desensamblado no esta la STRING REFERENCE que buscamos, gr..

Encima dentro de la carpeta del programa hay varios dll que puede ser que en alguno de ellos este, desensamblo uno por uno, buscando la STRING pero nada de nada no esta.

Vamos al SOFTICE, como parece ser una MESSAGEBOX (SIN la A al final ya que A es 32 bits y sin la A es 16 bits) pongo un BPX MESSAGEBOX y corro el programa espero unos segundos y faaaa, cuando quiere aparecer el cartelito, salta el softice. Aprieto F12 ACEPTO en el cartelito y vuelvo al SOFTICE.

Allí caigo (en mi caso) en

1927:1b45 mov [ebp-02], ax

que es la sentencia siguiente al CALL Messagebox.

Veo que no esta en el ejecutable principal sino en una dll llamada SBUWIN30.dll que si la busco esta en la carpeta del programa. La copio a otro lado para trabajar tranquilo y la abro con el WDASM.

Para no buscar como tarados uno por uno todos los bloques abrimos las Funciones Importadas, buscamos USER.MESSAGEBOX y hacemos click encima de la función hasta que atrás aparezca la parte buscada. Tenemos suerte que al hacer click sobre messagebox solo aparecen dos lugares, uno de ellos es **1b40** (es 0013:1b40).

Aquí esta en el WDASM la misma parte que encontramos en el SOFTICE

```
:0013.1B40  9AFFFF0000  call USER.MESSAGEBOX
:0013.1B45  8946FE      mov [bp-02], ax
:0013.1B48  57          push di
```

Era el **0013:1b40**. Allí esta el call maldito y mirando arriba no hay ningun salto condicional que lo pueda evitar para forzarlo a saltar. Busco de donde proviene esto en la REFERENCE que hay un poco más arriba pero sigo y sigo para atrás sin encontrar ninguna forma de evitar este MESSAGEBOX y si lo NOPEO me da Error de Windows, grrr..

Como puedo hacer que no aparezca este cartel maldito?

Voy a ver si puedo insertar un JUMP un poco más arriba del cartel para saltearlo, veo esta parte:

```
:0013.1B11  51          push cx
:0013.1B12  56          push si
:0013.1B13  51          push cx
:0013.1B14  56          push si
:0013.1B15  8976D0      mov [bp-30], si
:0013.1B18  894ED2      mov [bp-2E], cx
:0013.1B1B  9A7E160000  call KEYBOARD.OEMTOANSI
:0013.1B20  FF7606      push word ptr [bp+06]
:0013.1B23  9AFFFF0000  call USER.MESSAGEBEEP
:0013.1B28  57          push di
:0013.1B29  8D46D4      lea ax, [bp-2C]
```

```
:0013.1B2C  16          push ss
:0013.1B2D  50          push ax
:0013.1B2E  6A00        push 0000
:0013.1B30  0E          push cs
:0013.1B31  E85200      call 1B86
:0013.1B34  57          push di
:0013.1B35  66FF76D0    push word ptr [bp-30]
:0013.1B39  66FF760C    push word ptr [bp+0C]
:0013.1B3D  FF7606      push word ptr [bp+06]
:0013.1B40  9AFFFF0000  call USER.MESSAGEBOX
:0013.1B45  8946FE      mov [bp-02], ax
```

Voy a tratar de insertar un JUMP en **1b11** que salte el MESSAGEBEEP Y EL MESSAGEBOX, pero si lo hago saltar a **1b45** que es la sentencia justo posterior al cartelito MESSAGEBOX me sale error de WINDOWS, veo que más abajo hay un salto condicional

```
:0013.1B58  7412        je 1B6C
```

Probare a hacer un JUMP a **1b6c** directo desde **1b11**. Puedo probarlo con el SOFTICE; una vez que para en el MESSAGEBOX, pongo un **BPX 1b11** y una vez que para allí hago **A [ENTER]** y escribo **JUMP** 1b6c para que en la memoria quede esa sentencia y borro con BC* y hago X para volver al programa.

HMMM... desapareció el cartelito y no sale más, jajaja, sin error de WINDOWS ni nada, había que saltar a ese punto, fue un buen intento.

Ahora abro el SBUWIN30.dll con el ULTRA EDIT y la cadena que copie antes de cambiar con el SOFTICE era **515651568976d0**. Reemplazo el **5156** por **EB59** que es el salto a **JUMP 1b6c**

Era originalmente así

```
:0013.1B11  51    push cx
:0013.1B12  56    push si
:0013.1B13  51    push cx
:0013.1B14  56    push si
```

Y quedo así luego de modificado

```
:0013.1B11  EB59  jmp 1B6C
:0013.1B13  51    push cx
:0013.1B14  56    push si
```

Por suerte para mí, programa crackeado. Uff, espero no torturarlos con programas de 16 bits por un tiempo.

Ricardo Narvaja

Hasta la 40

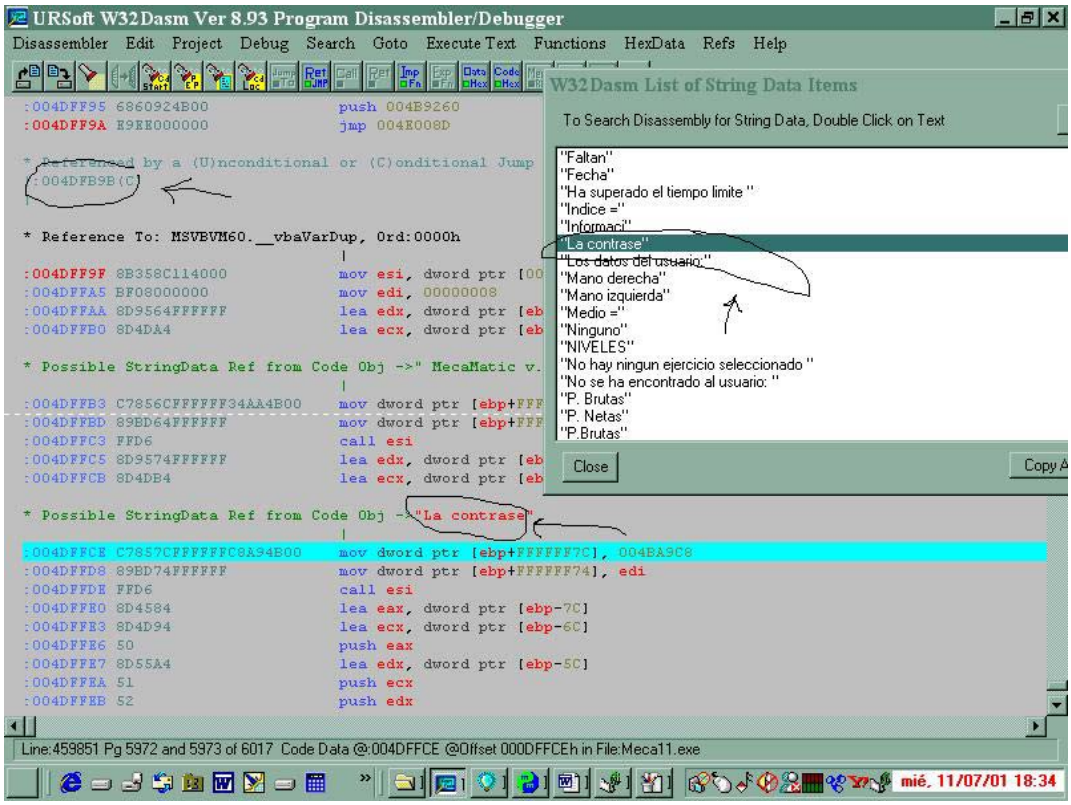
LECCIÓN 40: UNA DE VISUAL BASIC, RÁPIDA Y FÁCIL

El programa victima se llama MECAMATIC 1.1. Se baja de http://leo.worldonline.es/antgarco/MecaMatic%20v.1.1.zip y es un programa para aprender a escribir a maquina y esta en castellano y en VISUAL BASIC.

Te das cuenta pues al instalarlo, ya te dice que esta copiando las librerías de VISUAL BASIC 6, el Wdsam lo abre perfectamente , no esta comprimido ni nada. Hay que usar el WDASM modificado para VISUAL BASIC, no el de DELPHI.

Una vez que se abre el programa nos pide un nombre de usuario, para iniciar, y luego yendo al menú de ayuda, allí esta la ventana para registrarse. Nos pide nombre y número de serie.

Pongo de nombre **Ricardo** y como número de serie **989898** y al aceptar nos sale el cartel de que la contraseña es incorrecta. Buscando entre las STRINGS REFERENCES en el WDASM aparece la misma en **4dffcb** y proviene de un salto condicional en **4dfb9b**.



Allí están marcados en la imagen la STRING **"La contrase"** y el salto de donde proviene la **REFERENCE by a condicional JUMP de 4dfb9b.**

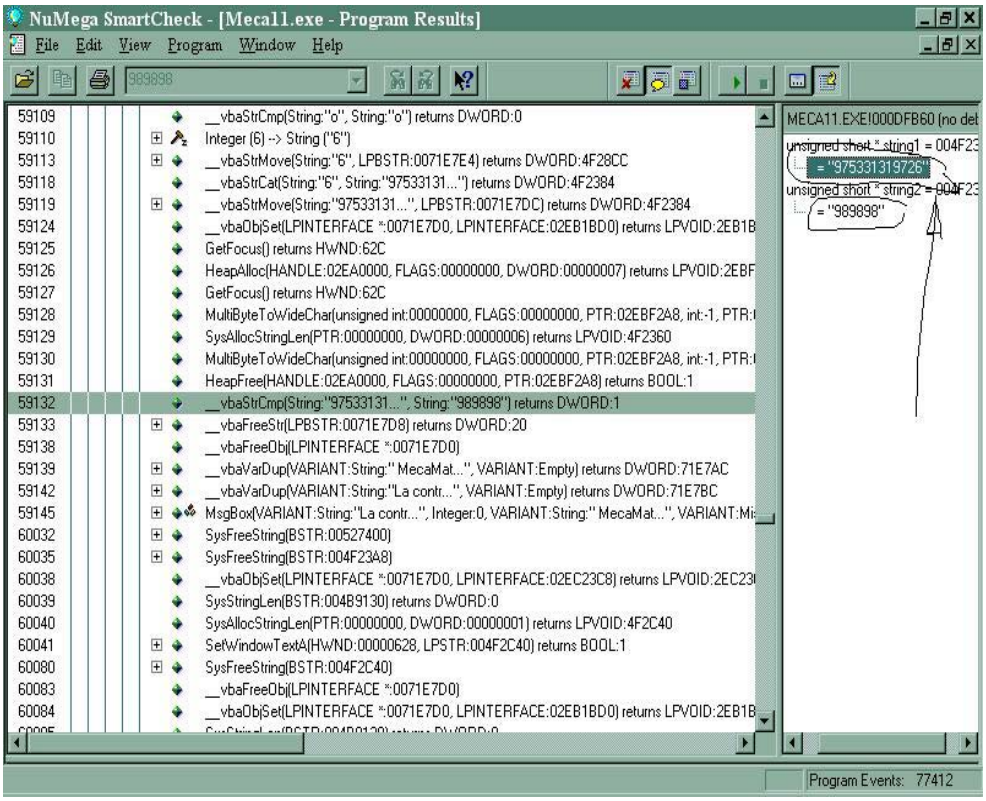
Aquí se pueden tomar varios caminos, analizarlo con el Softice hasta encontrar la comparación o utilizar el **Smart Check** para ver si compara las claves directamente.

Probaremos el SMART CHECK. Lo abrimos, cargamos el ejecutable MECA11.exe y ejecuta el programa. Hay que tener bien configurado el

Smart check como dice en la LECCIÓN sobre el mismo y si no se ve la ventana en la cual aparece la clave a la derecha, es porque esta cerrada sobre el margen derecho. Poniendo el cursor allí y arrastrando hacia la izquierda se abrirá la ventana donde aparecerán las claves.

Una vez que arranca el programa vamos a la ventana de ayuda y ponemos de nuevo name **Ricardo** serial **989898** y le damos a ACEPTAR. Nos dice que la contraseña es incorrecta, vamos al Smart Check y detenemos la ejecución.

Vamos a FIND y colocamos **989898** y ponemos la marca en la primera línea ya que busca desde allí hacia abajo y en la segunda vez que para ya me aparece el serial correcto.



Allí en la imagen anterior esta el momento de la comparación. Arriba está el número de serie verdadero y debajo el falso que puse yo. Copio la clave **975331319726** que corresponde a **Ricardo** y abro de nuevo el programa voy a ayuda y introduzco mi nombre, Ricardo, y el número de serie, 975331319726, y al aceptar me dice que la clave es correcta que estoy registrado. Eso si, puede ser que sirva solo para mi maquina ese número de serie, quizás en otra maquina sea diferente pero la forma de hallarlo es la misma.

Por si no se entienden las imágenes las mando adjuntas también.

Hasta la LECCIÓN 41

Ricardo Narvaja

LECCIÓN 41: TRUQUITOS VARIOS

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

Se baja de http://www.silverlaketech.com/PCDFmain.asp y es un programita que tiene algunas cositas para despistar aunque como veremos se puede crackear, las trabas no son muy grandes pero si lo suficientes para complicar un poco la vida.

Abrimos el ejecutable con el Wdasm y carga bien, vemos entre las cadenas de texto (STRINGS REFERENCES) que dice por ahí BORLAND, DEPLHI, etc, lo que significa que esta escrito en DELPHI.

Abrimos el programa y vamos a ABOUT y nos aparece la opción para registrarnos. Ponemos una clave falsa y cualquier nombre y cuando le damos a REGISTER, nos sale **INVALID REGISTRATION CODE**, bueno probamos con BPX MESSAGEBOX, BPX HMEMCPY (increíble que no pare en el SOFTICE con este), y varios más y no entra en el SOFTICE y nos sale el funesto cartelito.

Dicho cartelito no aparece entre las STRINGS REFERENCES. Probamos un BPX que suele servir bastante en DELPHI que es **BPX DRAWTEXTA** y, ahí si, por lo menos para en el SOFTICE y con F12 volvemos al ejecutable, aunque en cualquier lugar. Borramos con BC*, y observamos algo raro. En el SOFTICE las posiciones de memoria cuando estamos en el ejecutable son valores tipo 46eca7, en cambio en el WDASM son valores tipo 7ec18 muy diferentes, hmmm.

Que pasa aquí?. Vayamos al WDASM y veamos el ENTRY POINT, vamos a GOTO - GOTO PROGRAM ENTRY POINT y eso nos lleva aquí:

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

En **9A018** sería en ENTRY POINT según el WDASM.

Ahora tratemos de cargar el programa desde el SYMBOL LOADER del Softice para que pare cuando arranque en el mismo ENTRY POINT, y vemos que allí nos dice **48a018** en realidad marca **48a017** pero ese valor es 00 y el siguiente es el **55** que corresponde a **push ebp**.

O sea que **9a018** del WDASM corresponde a **48a018** del SOFTICE y... porque ocurre esto?. Si miramos el ejecutable con el PEEDITOR vemos que la **image base es 10000**, y ya que no esta comprimido ya que cargó directamente en el WDASM con sus STRINGS REFERENCES y todo, podemos suponer que este número esta cambiado para despistar, o sea que si restamos **9a018-10000** obtendríamos el offset que seria **8a018** y si allí le sumamos la image base que generalmente se utiliza que es **400000** sería **400000+8a018=48a018** que es el valor que sale en el softice.

O sea que hallamos una formulita para pasar los valores del WDASM al Softice, en este caso seria la siguiente:

Al valor del WDASM restarle 10000 y sumarle luego 400000, para hacerlo en un solo paso le podemos sumar al valor de WDASM 3f0000 y listo ya que en hexa eso equivale a hacer toda esa operación en una sola.

9a018 + 3f0000 = 48a018

WDASM + 3f0000 = Softice

Con esa formulita nos ayudaremos un poco.

Bueno, dado que no tenemos ganas de dar muchas vueltas y como esta hecho en DELPHI usaremos el DEDE, para eso tenemos la versión 2.50 que es la ultima y completa, tratamos de cargarlo con el dede y puum, error de no se que y se cierra el dede, ufff.

Otro truquito, pero para algo este cracker guarda todo, probaremos con una versión vieja del dede la 2.34 que le faltan muchas utilidades pero nos servirá.

Cargamos desde este DEDE y carga perfectamente. Vamos a procedures y vemos REGISTRATION y si hacemos click allí a la derecha en EVENTS nos aparece OKBTNCLICK, y dado que cuando ingresas la clave apretas un botón que dice OK, probemos con esto.

Si hacemos click derecho encima de OKBTNCLICK, y vamos a Disassemble nos aparece

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

O sea que haciendo la conversión para el SOFTICE seria **7f0f4 + 3f0000 =** 46f0f4 (Pondremos un BPX allí en 46f0f4)

Podemos poner primero un BPX DRAWTEXTA y cuando vuelve al ejecutable poner el BPX 46f0f4 allí.

Ahora vemos que cuando ingresamos un nombre y una clave, para el softice en **46f0f4 push ebp.**

Bueno a partir de allí podemos hacer F10 hasta que aparezca el cartelito **INVALID REGISTRATION CODE**. Esto ocurre en

46f1ff CALL 45f3ff

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

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

Si este salto lo invertimos y cuando llegamos allí con f10 ponemos

R eip=46f21f

vemos que nos dice que estamos registrados, pero al volver a arrancar el programa, vuelve a arrancar desregistrado.

Se ve que realiza la comprobación en otra parte cuando comienza. Esto lo vemos en el WDASM cuando entramos en ese CALL que dice que también es llamado desde otra parte del programa.

Esto se ve en el WDASM aquí:

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

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

7f070 + 3f0000 = 460f70

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

Entonces el problema debe estar dentro del CALL miremos dentro (ESTO MUESTRA EL WDASM SUMAR 3F0000 PARA OBTENER LOS VALORES PARA EL SOFTICE) y esto es lo que encontramos un poco más abajo dentro del CALL

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

Parece ser una comparación de la cantidad de cifras ya que 0B en hexa es 11, y podría ser 11 cifras. Probemos poniendo un BPX allí y cuando pare veremos que en EAX esta la cantidad de cifras que pusimos, o sea que la clave debería tener 11 cifras, pero también sabemos que si no tiene 11 cifras va a 7eca9 que es que te manda al cesto de basura ya que...

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

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0007ECC9(U)
|
:0007ECB6  8D45F8        lea  eax, dword ptr [ebp-08]
```

```
:0007ECB9  BA02000000  mov  edx, 00000002
:0007ECBE  E86D4EF9FF  call 00013B30
:0007ECC3  C3          ret
```

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

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

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

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

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

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

DISCULPEN QUE NO ESTE SACANDO UNA LECCIÓN POR SEMANA COMO ANTES PERO PROBLEMAS PERSONALES ME LO IMPIDEN PERO APENAS PUEDA SIEMPRE HARE UNA LECCIÓN.

HASTA LA 42
RICARDO NARVAJA

LECCIÓN 42: OTRO PROGRAMA DE 16 BITS

Se llama **MEGAPAK 8.1.2** Multiempresa y esta hecho en 16 bits se baja de http://148.233.25.146/sisteval.html#megapaqw y, aparentemente, sirve para facturación, compras ventas y esas cositas que hacen las empresas, jua.

Como ya saben por las lecciónes anteriores, le huyo bastante a estos programas, pero bueno, era para un amigo, y bueno lo hicimos.

Tratamos primero de encontrar la clave, la cual esta bastante encriptada, para el que quiera practicar con claves encriptadas y la quiera hallar, le doy una ayuda transcripta de los mails que nos mandamos en la lista crackslatinos sobre el tema. Por supuesto, como yo encontre la forma de crackearlo sin tener la clave este camino quedo inconcluso. Para el que lo quiera investigar le transcribo los mails; para el que quiere saber como crackearlo, saltee esto y siga más abajo en donde esta el titulo **COMO CRACKEARLO**.

MAILS SOBRE LA CLAVE ENCRIPTADA

Pones la clave falsa BPX HMEMCPY, vas con F10 hasta REPZ MOVSB y allí haces d es:di

Me sale la clave falsa a mi (EN MI MAQUINA EN LA TUYA PUEDE VARIAR) en 3677:354c. Hago PAGE 3677:354c y me sale el resultado LINEAR 811130cc

Si hago d 0030:811130cc me aparece la clave falsa donde la guardo pongo un BPR allí en ese rango BPR 0030:811130cc 0030:811130dc rw

Borro el BPX HMEMCPY con BC00 y le doy a X para que corra el programa para la primera vez que toca la clave en 015f:a9f5 Repz scasb que se usa para ver cuantas cifras tiene seguís con F10 hasta el RET y allí en EAX las dos ultimas cifras son la cantidad de cifras de tu clave falsa

Hay que ver si la compara con algo si compara AX con cero así que aquí no pasa nada le doy con X de nuevo para en 0eff:5400 compara la primera cifra con cero

Sigo con X compara en 0eff:53e8 compara con 20 (espacio) este ciclo se repite cifra por cifra

Una vez que sale de allí para en 015f:a9f5 REPNZ SCASB

Cuenta las cifras igual que antes al salir del RET en 0eff:0dd1 y compara la cantidad de cifras con 1a o sea 26 y parece que si no es 26 te tira al tacho por lo que hay que poner 26 cifras de la clave para seguir a partir de aquí.

Luego en 0eff:5370 un poco más adelante empiezan comparaciones con valores fijos.

Bueno este es un programa que encripta la clave, por lo cual hay que seguirlo bastante, y tener en cuenta cada lugar que guarda y cada operación que hace, y lleva su tiempo.

Como ya te dije tiene que tener la clave 26 cifras según lo que se ve en la comparación que te mencione en el mail anterior.

Otra cosa que hay que ver cuando pones un BPR en tu clave falsa es que cuando el programa toma una cifra y la coloca en EAX, en este caso AX o cualquier otro de esos, además de verificar las comparaciones y adonde lo guarda y allí volver a poner otro BPR, también hay que fijarse que casi siempre estos programas que comparan cifra por cifra, guardan con alguna sentencia tipo PUSH EAX o PUSH AX, (SIEMPRE QUE EN AX) este el valor de alguna de tus cifras de la clave falsa

De esta forma lo guarda en el STACK y más adelante lo recupera de allí, por lo que si en este caso mi clave falsa la primera cifra es 9, o sea, que en hexa es 39, y lo guarda en AX, si el programa hace PUSH AX, lo esta guardando en el STACK y lo puede recuperar más adelante, en este caso lo que hay que hacer, es poner un BPR en el lugar que lo guarda cosa de que cuando lo quiera volver a leer pare el SOFTICE allí.

En 10df:51b2 PUSH AX (puede variar la primera parte o sea 10df según la maquina) en AX esta 39 que es el 9 de mi primera cifra y donde lo guarda cuando hace PUSH EAX, hay que fijarse en registro ESP que es el puntero del STACK ese indica donde se guarda.

En mi caso es 10df:5370 pongo un BPR allí, conviene hacer el mismo procedimiento que hicimos cuando paramos en el HMEMCPY que te explique en el mail anterior o sea PAGE 10df:5370

```
LINEAR xxxxxxxxxx
D 0030::xxxxxxxxx
```

Y si allí aparece el 39 hacer

```
BPR 0030:xxxxxxxxx
0030:xxxxxxxy rw
```

para que abarque el valor y pare el SOFTICE cuando lo lea de nuevo.

En este programa es este caso, lo guarda allí a la cifra y cuando para el SOFTICE que la vuelve a leer sigue trabajando con ella, esta es una forma de esconder la búsqueda de la clave y despista a la mayoría de los novatos, es muy importante saber esto y si ves que EAX es 39 o EBX o cualquiera de esos, tracear con F10 y ver lo que hace y donde guarda esos valores y si hace PUSH, por lo menos hasta que cambie el valor de EAX a otra cosa.

Bueno, luego anote varios lugares donde para el SOFTICE

```
10df:51dd  add es: [bx+si], al
```

Allí guarda los valores, si lo seguís vas a ver que a cada cifra la transforma y si mi clave falsa es 9898989898 ..... (26 cifras) toma el 9 , lo transforma en 90 y lo guarda en 550f:2e56

Luego toma el 8 y se lo suma en la sentencia que te mostré antes
10df:51dd add es: [bx+si], al

O sea que si miro allí guardó 98 como es mi clave, y así sucesivamente

guarda la clave 98 98 98 98 98 98 98 .... cifra por cifra.
Ahora empieza lo bueno, la encriptación.

Lógico que hay que poner un bpr allí donde esta mi clave así guardada 98 98..... sigo corriendo el programa y para en

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

que es tomar la primera cifra de la clave, luego la guarda en

550f:2dd6 (poner otro BPR aquí)

Luego hay que ir anotando las operaciones que le realiza a cada cifra.

En 10df:56e5 XOR al, [bp+ff0e] hace un XOR (muy usado en encriptaciones) con el valor E4. Hay que anotar todo ya que más adelante hay que realizar todas las operaciones que realizo sobre tu clave falsa, hacia atrás sobre el número que compara con el tuyo encriptado para encontrar la clave verdadera.

Al es 7c luego de realizarle el XOR y lo guarda en 550f:2e00 (poner otro BPR aquí).

BUENO HASTA AQUÍ LLEGATON MIS GANAS DE HALLAR LA CLAVE QUIZAS A ALGUIEN LE SIRVA Y LE INTERESE PRACTICAR Y SEGUIR DESDE AQUÍ, BUENO YO ME CANSE.

COMO CRACKEARLO

Cuando ingresamos la clave ponemos un BPX MESSAGEBOX sin la A al final ya que es 16 bits y cuando para allí, puedo volver con RET, pero este camino no me lleva (esto es así ya que probe y probe si alguno de esos saltos que habia antes del MESSAGEBOX me registraba).

Como no pude hacer nada opte por ver las STRINGS REFERENCES y alli estaba

"SU ... NO ESTA REGISTRADO QUEDAN ... VECES PARA ENTRAR AL SISTEMA", que es un cartel que aparece cuando no estas registrado. Si podemos lograr saltearlo puede funcionar como registrado, veremos.

Aquí viene una larga parte copiada del WDASM que termina justo en la referencia a esa STRING.

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

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

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

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

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

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

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

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

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

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.14AF(U)
|
:0001.14CD 6A01        push 0001
```

```
:0001.14CF 9AFFFF0000  call LCRYPKYD.GET1RESTINFO
:0001.14D4 8986ECFC    mov [bp+FCEC], ax
:0001.14D8 83BEECFC02  cmp word ptr [bp+FCEC], 0002
:0001.14DD 7503        jne 14E2
:0001.14DF E90A00      jmp 14EC

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

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

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

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

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

* Possible Reference to Dialog: DialogID_03F3
|
:0001.1530 68F303      push 03F3
:0001.1533 6A00        push 0000
:0001.1535 6A00        push 0000
:0001.1537 6A00        push 0000
:0001.1539 9AFFFF0000  call USER.CREATEDIALOG
```

```
:0001.153E 898656FB    mov [bp+FB56], ax
:0001.1542 8B5E0A      mov bx, [bp+0A]
:0001.1545 D1E3        shl bx, 01
:0001.1547 D1E3        shl bx, 01
:0001.1549 FFB75200    push word ptr [bx+0052]
:0001.154D FFB75000    push word ptr [bx+0050]

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

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.157E(C)
|
:0001.1583 FFB65CFB    push word ptr [bp+FB5C]
:0001.1587 8B5E0A      mov bx, [bp+0A]
:0001.158A D1E3        shl bx, 01
:0001.158C D1E3        shl bx, 01
:0001.158E FFB75200    push word ptr [bx+0052]
:0001.1592 FFB75000    push word ptr [bx+0050]

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

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

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

Yo sospeche que de aquí podia provenir ya que todas las otras referencias eran dentro de este mismo listado. En cambio la referencia sospechosa provenia de otro lugar, como si el programa esta registrado no salta hacia este sector y sigue por otro lado, y era asi, el programa registrado no pasa por todo el listado que puse antes.

Entonces, si viene de **13e3**, vemos que hay por alli, para tratar de evitar que venga hacia el cartel maldito.

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

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

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

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.1A81(U)
|
:0001.13B4 8D8656FB    lea ax, [bp+FB56]
:0001.13B8 8CD2        mov dx, ss
:0001.13BA 52          push dx
:0001.13BB 50          push ax
:0001.13BC 9ABE355E13  call 0001.35BE
:0001.13C1 83C404      add sp, 0004
:0001.13C4 898650FB    mov [bp+FB50], ax
:0001.13C8 8D8656FB    lea ax, [bp+FB56]
:0001.13CC 8CD2        mov dx, ss
:0001.13CE 52          push dx
:0001.13CF 50          push ax
:0001.13D0 9ABE35BF13  call 0001.35BE
:0001.13D5 83C404      add sp, 0004
:0001.13D8 F7AE50FB    imul word ptr [bp+FB50]
:0001.13DC 898654FB    mov [bp+FB54], ax
:0001.13E0 8B460A      mov ax, [bp+0A]
:0001.13E3 E98B00      jmp 1471
```

Vemos que las sentencias en negrita son las que importan. Arriba en **13a9** esta la clave de todo, allí ponemos un **BPX 13a9** para probar.

Arrancamos el programa y vemos que para que no vaya al salto en **13e3** no tiene que saltar ya que si salta va a **13ae** y cae irremediablemente al salto de **13e3** que lleva al cartel maldito.

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

O sea que si vamos al ULTRA EDIT y buscamos la cadena de ese salto que es **7e 03 e9 dc 06 8b 46 f2 e9 c0 06 8d 86 56 fb** y reemplazamos en **7e 03** por **90 90** quedara el programa registrado perfectamente.

Hasta la LECCIÓN 43
Ricardo Narvaja

LECCIÓN 43: PROGRAMAS BASTANTE MOLESTOS (POR LO EXTENSO DE LA PROTECCIÓN)

Este es un programa llamado BANK EASY 2.1 que sirve para llevar cuentas bancarias y ese tipo de transacciones.

Se baja de http://www.ghostsolutions.com/Main/BankEasy/BankEasy.htm y la verdad que tiene tantos lugares que parchear que ya ni recuerdo bien de todos, trataremos de ir de a poco.

Lo que si es cierto es que no esta comprimido ni nada, las STRINGS aparecen bien aunque no todas salen en la lista de STRINGS REFERENCES del WDASM, a pesar que en el listado desensamblado del WDASM si aparecen.

No se cual es la verdadera razón de esto pero creo que puede ser esta:

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

```
*Possible StringData Ref ... ->"MessagesThere seems to be anerror"
                             ->"with your PC date.  The last time"
                             ->"the program was run was the"
```

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

Sin embargo esa STRING no aparece entre las STRINGS REFERENCES dela lista, y hay muchas así, no se la razón pero creo que debe ser por el tipo de cartel que utiliza la función **DialogBoxIndirectParama**, que si ponemos un Bpx allí para en esos cartelitos maléficos.

Que hay muchos y surtidos y cuando aparecen el programa arranca pero como vencido. Los iconos no aparecen activos y no se puede usar, así que no solo hay que tratar de que el programa piense que estamos registrados sino también, que saltee todas estas protecciones de que si adelantaste el reloj, otro cartelito dice que no tenés licencia valida, otro que tu licencia es fraguada, otro que falta un driver (mentira), y así hay varios. La cuestión es que cuando aparece uno de estos carteles, no arranca bien el programa, de idéntica forma como si esta vencido.

En cualquier caso que queramos buscar una STRING de esos cartelitos conviene más buscarlas por **SEARCH – FIND**, y poner una o dos palabras del cartelito y si no sale nada probar con otras dos palabras, que buscarlas en la lista de STRING REFERENCES.

Bueno al ataque mis valientes, con paciencia vamos a ir al revés de la solución a la explicación.

Aquí va primero la lista de bites que hay que modificar para crackear esta protección pelmazo (por lo larga).

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

Es bueno ver que para hallar la posición de memoria en el WDASM no podemos sumarle 400000 que es la Image base, porque si vemos con el PEDDITOR las secciones no están arregladas, o sea, que por eso no da la suma pero podemos calcular fácilmente, viendo el ENTRY POINT o cualquier punto, la diferencia entre el offset y el valor en el WDASM.

Aquí hay que sumarle 400C00, podríamos arreglarlo con el PEEDITOR y luego sumar directamente 400000, pero por si acaso chequea las secciones o algo así, más vale tomémonos la molestia y sumemos, no es difícil calculadora en mano.

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

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

En este listado se ve claro

:004B4572 E8A518F9FF call 00445E1C

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

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

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

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

Sin aburrirnos miremos todos los lugares de donde proviene...

```
* Referenced by a CALL at Addresses:
|:00402D32, :00402D8A, :00402FB6, :0040309B, :00404F90
|:00404FBD, :00406466, :004065EA, :00409E6D, :0040A913
|:0040B475, :0040B502, :0040B579, :0040B597, :0040B5C1
|:0040B5DF, :0040B605, :0040B623, :0040D8BB, :0040F464
|:0040F6D0, :0040FE00, :00411E2E, :00415E80, :00415EBF
|:00415EFB, :00415F8F, :00415FE2, :00416035, :00416085
|:00419E6B, :0041A82A, :0041EA0D, :0041EA59, :0041EAC9
|:0041EAE3, :0041EE23, :0041EE8B, :0041EEF1, :00421644
|:004216B6, :0042172C, :004217A3, :004217C5, :0042182D
|:00421895, :004218D6, :004218FB, :00421920, :00421945
|:0042196A, :0042198F, :004219B4, :00421A84, :00421AA9
|:00421B19, :00421B3B, :00421BAF, :00421C31, :00421C56
|:00421CCB, :00421D31, :00421D53, :00421DB9, :00422C60
|:0042337C, :00423404, :00425700, :00425771, :004257AA
|:0042581B, :00425890, :00425945, :00425967, :004259CE
|:00425A35, :00425A9E, :00425AC3, :00425AE8, :00425B0D
|:00425B2F, :00425B51, :00425B73, :00425BF0, :00425C77
|:00425CE6, :00425D08, :00425D7B, :00425E01, :00425EB4
|:00425F19, :00425F3B, :00425FA0, :00425FD5, :004260AF
|:00426122, :0042CD5B, :0042CDD3, :0042D01E, :0042EE84
|:0042EEB7, :0042EEEA, :0042EF1E, :0042EF67, :0042EFB0
|:0042EFF6, :0042F086, :0042F20C, :0042F26D, :0042F31E
|:0042F7AD, :0042F7EB, :0042FA20, :004307A3, :00430913
|:00430961, :00430D08, :004316BA, :004316FB, :00432924
|:00433FB7, :004340AC, :0044DCDD, :0044DD1D, :0044DE3C
|:0044DEA5, :00488910, :004889DC, :00489DE9, :0048A1B5
|:0048A683, :0048A6BD, :0048A6F7, :0048A72E, :0048A765
|:0048A7A6, :0048A7E7, :0048A828, :0048A85F, :0048A896
|:0048A8D7, :0048A91E, :0048ABCC, :0048B3C7, :0048B5B8
|:0048B660, :0048B73F, :0048B827, :0048B94A, :0048BA6D
|:0048D3AD, :0048D4B5, :0048D612, :0048D813, :0048D938
|:0048DB3A, :0048DCA6, :0048DEA8, :0048DFB2, :0048E175
|:0048E2CD, :0048E4CF, :0048E5A8, :0048E5D2, :0048E682
|:0048E6AC, :0048E75C, :0048EA12, :0048EA5F, :0048F2BA
|:0048F33D, :0048F3C0, :0048F54C, :0048F660, :0048F98F
```

```
|:0048F9E8, :0048FED0, :0048FF3E, :0048FF5D, :0048FF7C
|:0049028E, :00490332, :00490425, :00490617, :00491144
|:004927BA, :004927FE, :00492892, :004928BC, :004928E6
|:00492944, :0049299F, :00492A16, :00492F36, :0049315B
|:0049328E, :004932AC, :004932CF, :00493488, :004936F6
|:00494517, :00494D11, :00494D2F, :00495847, :00496832
|:00496A57, :00496A7C, :00496AA1, :00496D8A, :00496EA0
|:00496EBF, :00496F96, :00496FB8, :00497196, :004976A4
|:00498C38, :00498DCE, :00499F31, :0049A2AA, :0049B31B
|:0049B5FA, :0049B62E, :0049EEFC, :0049F50B, :004A1EF8
|:004A2519, :004A2535, :004A25B8, :004A26C0, :004A4929
|:004B1189, :004B1817, :004B1BFC, :004B1C6A, :004B1C88
|:004B1CAA, :004B1CF7, :004B1DFD, :004B3253, :004B414C
|:004B42D2, :004B4419, :004B445E, :004B44A3, :004B44E8
|:004B452D, :004B4572, :004B45B7, :004B45FC, :004B4641
|:004B4686, :004B46CB, :004B4710, :004B4755, :004B479A
|:004B47DF, :004B4824, :004B4869, :004B48AB, :004B497F
|:004B49F0, :004B5235, :004B59B1, :004B5A3E, :004B5DAD
|:004B5F48, :004B6090, :004B673A, :004B6797, :004B735B
|:004B73E3, :004B7CA0, :004B7D3C, :004B858C, :004B871C
|:004B886E, :004B92D8, :004BA99D, :004BA9FD, :004BAA39
|:004BB25E, :004BB426, :004BB4FB, :004BB640, :004BB6A0
|:004BD67C, :004BDDC3, :004BDECA, :004BDF3A, :004BDF7C
|:004BDFEE, :004BED31, :004BEF8E, :004BEFD4, :004BF0DF
|:004BF3C3, :004BFE67, :004BFEBE, :004BFEF7, :004C08A4
|:004C090C
```

Bastantes no?

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

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

El programa originalmente es asi...

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

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

Quedaria asi

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

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

Yo no lo hice así originalmente porque como no hay lugar para escribir las sentencias, me sobrescribe el CALL siguiente, pero probé de esta forma y funciona igual, no afecta (aparentemente) al otro CALL siguiente.

Sino pueden probar la forma que yo lo parchee en la lista de cambios realizados. Es igual, sale siendo AL igual a cero.

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

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

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

* Possible StringData Ref... ->"FM2: Warning - Driver not located "
                             ->"- GPF likely"
```

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

Este otro salto...

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

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

Ahora hay otros saltos como este

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

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

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

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

```
:004B25B0 E8EFE2F8FF    call 004408A4
:004B25B5 8B1DB1195700  mov ebx, dword ptr [005719B1]
:004B25BB 39D8          cmp eax, ebx
:004B25BD 9C            pushfd
:004B25BE 8B54240C      mov edx, dword ptr [esp+0C]
:004B25C2 83E204        and edx, 00000004
:004B25C5 9D            popfd
:004B25C6 0F8DC2010000  jnl 004B278E
:004B25CC B801000000    mov eax, 00000001
:004B25D1 BB01000000    mov ebx, 00000001
:004B25D6 B92F080000    mov ecx, 0000082F
:004B25DB E894DEF8FF    call 00440474
:004B25E0 3905AD195700  cmp dword ptr [005719AD], eax
:004B25E6 0F8DA2010000  jnl 004B278E
:004B25EC 83FA00        cmp edx, 00000000
:004B25EF 0F8599010000  jne 004B278E
:004B25F5 83FE00        cmp esi, 00000000
:004B25F8 0F8586010000  jne 004B2784

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

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

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

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

Este es idéntico para saltear CALL 4b4390

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

Y este también

```
:004B29D8 7531        jne 004B2A0B
:004B29DA 8B442408    mov eax, dword ptr [esp+08]
:004B29DE 83E008      and eax, 00000008
:004B29E1 83F800      cmp eax, 00000000
:004B29E4 7525        jne 004B2A0B
:004B29E6 83FE00      cmp esi, 00000000
:004B29E9 7514        jne 004B29FF

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

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

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

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

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

Esperemos que la lección 44 sea más breve no?

Hasta la 44
Ricardo Narvaja

LECCIÓN 44: UN MUNDO NUEVO EL P-CODE

Hagamos una pequeña introducción al P-CODE, como en todos estos casos les aclaro que yo estoy aprendiendo con ustedes, por lo que disculpen algún error que pueda tener al explicar ya que el P-CODE es un tema difícil, diferente a lo anteriormente visto, y nuevo para mi.

Aunque ya tuve varios fracasos y medios aciertos con el crack del programa exponente máximo del P-CODE y una cuenta pendiente que me queda el CUENTAPASOS, esperemos que podamos ir avanzando con el P-CODE para que podamos también realizar un crack del CUENTAPASOS más adelante.

Este tutorial como yo estoy aprendiendo esta basado en un tutorial en ingles sobre un crackme que se encuentra en la misma página de bajada de la herramienta que utilizaremos para crackear estas pesadillas y que, a pesar de todo, simplifica bastante el problema del P-CODE al punto de poder analizarlo con tranquilidad. Igual yo he hecho algunos cambios sobre el tute en ingles que creo que lo hacen más fácil para entender.

Bueno primero debemos conseguir el programa a crackear.

http://vacarescu.addr.com/WkT/vbdebug/tutos/vbcrackme10.zip

Se baja de allí y por supuesto, para los que no lo saben aun, todos los programas que se craquearon en el curso así mismo, como este también, están alojados en mi Kturn.

El que los quiera bajar solo debe ir a www.kturn.com y allí registrarse gratis para sacar un Kturn propio te da 125 megas de almacenamiento y es gratis. Una vez que ya tiene su Kturn entran y allí van a la opción OTHER USERS y allí teclean ricnar458 y con eso ya entraron.

En la carpeta PUBLIC están todos los programas que se crackearon en el curso, obviamente sin crackear para el que quiera practicar, por orden y numerados según la Lección que corresponde.

En http://vacarescu.addr.com/WkT/vbdebug/downloads.htm esta el VBDEBUGGER para P-CODE.

Que diferencia hay entre P-CODE y Visual Basic sin P-CODE?. Bueno, eso es algo muy técnico para explicarlo aquí, pero es bueno que sepan que en el P-CODE el ejecutable no se ejecuta nunca, paradójicamente.

El ejecutable en realidad son las librerías de Visual Basic.

Y entonces, como sabe la librería de Visual Basic que es lo qe tiene que hacer?

Buena Pregunta. Va ejecutando sentencias según los valores de memoria que encuentra en el ejecutable del programa, o sea para aclarar en un ejemplo burdo, el ejecutable del programa es una serie de números sueltos que si los desensamblamos no significan nada ya que podemos probar con el WDASM y no tienen ninguna coherencia. Son solo como semáforos para que las librerías de Visual Basic sepan que hacer.

Es complicado verdaderamente.

O sea que, si uno quiere crackear esto en la forma tradicional apenas arranca el programa. Si lo seguimos con el SOFTICE vemos que en la línea verde siempre se esta ejecutando **MSVBVM60.dll** (si es en Visual Basic 6.0)

Y allí esa librería hace todo, y toma decisiones según los valores que toma del ejecutable, pero como valores sueltos de la memoria no como sentencias.

El hecho es que si hallamos por ejemplo en la librería de Visual Basic un salto que deberíamos parchear y hace registrar un programa, no podemos parchear ese dll ni cambiarlo porque Todos los programas de Visual Basic usan esa misma librería y dejarían de funcionar o los afectaría.

La única forma de parchear seria ver que valores del ejecutable hacen a la librería de Visual tomar ciertas decisiones, y cambiarlo allí, pero esto es una verdadera pesadilla y un dolor de cabeza total.

Por eso se hizo este debugger especial para P-CODE, para tratar de facilitar el trabajo a los pobres crackers vagos como yo, jua jua.

Es también de notar que el Smart Check no sirve para programas en P-CODE y sale un cartel diciendo que el programa es P-COMPILADO y que Smart Cheack no puede funcionar correctamente con el o algo así.

Bueno vamos a bajar la herramienta.

Se baja de http://vacarescu.addr.com/WkT/vbdebug/downloads.htm

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

De la pagina de bajada...

This **new** version fixes some loader bugs, and adds a more detailed error message if something goes bad, also fixes some invalid opcodes, and fixes a minor visual dumping error ;-).

Thanx to oskie for being so nice and helping in finding many bugs.

WKTVBDebugger v.1.3 (with English Help)
WKTVBDebugger v.1.3 (with Spanish Help)

Luego estan los tutoriales de ejemplo...

Also check out this tutorials :

Cuentapasos v3.85, Se abrió la lápida (only spanish)
Eternal's Bliss VB CrackMe 10 (only english)

Sobre el segundo esta basado este tutorial aunque con algunos cambios.

Luego para el que no tiene instalado los runtimes de Visual Basic están aquí para bajarlos...

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

MSVBVM60.DLL v.6.0.89.64
MSVBVM50.DLL v.5.0.82.44

Estos habría que copiarlos en el C:/WINDOWS/SYSTEM aunque yo prefiero siempre estas cosas bajarlas directamente de Microsoft y completas con instalador y todo

http://download.microsoft.com/download/vb50pro/utility/1/win98/EN-US/Msvbvm50.exe

http://download.microsoft.com/download/vb60pro/install/6/Win98Me/EN-US/VBRun60.exe

De aquí se bajan completos desde la pagina de Microsoft. Luego de bajarlos, instalarlos y listo.

Luego vienen estas filas que se bajan y hay que copiarlas dentro de la carpeta de instalación del programa VBDEBUGGER, C:\Archivos de programa\WKTVBDE en la carpeta DBG

Symbolic debug files (DBG) files for use with the debugger:

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

Puff ya terminamos lo ultimo es la fila **Bdasmdll.dll**. La pueden buscar en su maquina, si la encuentran deben dejarla donde esta y copiarla además a la carpeta de instalación del programa también, junto con el otro dll que trae el programa **WKTVBDE.dll** y por supuesto **bdasmdll.dll**.

El que no la encuentre en su maquina, la fila esta la puede bajar de Internet o bajarla de mi Kturn que allí va a estar.

Bueno, esto es todo. Cuesta un poco instalar todo esto pero el ahorro de tiempo y la facilidad de uso vale la pena.

Hay en la pagina de donde se baja el programa un FAQ aquí http://vacarescu.addr.com/WkT/vbdebug/faq.html por cualquier problema que tengan.

Bueno arrancamos el VBDEBUGGER, y allí vamos a OPEN y buscamos el archivo para crackear, el VBcrackme 1.0.

Este podemos ejecutarlo antes para ver como es, aparece una grilla donde hay que poner una combinación exacta de tildes, entonces vamos a CHECK y si es la combinación correcta nos aparece un cartel de felicitación y si no, no aparece nada.

Lo arrancamos desde el debugger y una vez que cargó vamos a EXECUTE y comienza a funcionar el programa. Aparecen algunos carteles que debemos aceptar, y si todo esta bien llega el programa a la ventana con la grilla y el botón CHECK.

Cuando hacemos click en Check luego de poner cualquier combinación de tildes, caemos dentro del debugger, el cual puede ser activado igual con la combinación de teclas CTRL+P como si fuera el SOFTICE.

Caemos aquí, **004043DC: 28 LitVarI2 0063F360h 0h, 0**

Como vemos los OPCODES de P-CODE son diferentes a los que conocemos, ya los iremos aprendiendo, pero aquí hay uno que importa y es el salto condicional BRANCHF y BRANCHT. Si presionamos OPCODES nos muestra la lista de comandos existentes aquí y estos dos corresponden a los valores 1C y 1D respectivamente.

Estos saltos equivalen a que salta si es Falso BranchF (false) y salta si es verdadero BranchT (true).

Estas son las claves de las decisiones que toma el programa.

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

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

Estos son los saltos y vemos que todos caen adelante del siguiente así que podríamos probar si el último es el que toma la decisión final de si lo resolviste o no.

Pongamos un BPX en 4048b7. Vamos al BOTON breakpoints para poner un Breakpoint allí. Por supuesto será BPX o sea al botón ON EXECUTION, escribimos 4048b7 y tecleamos ADD para que lo agregue y luego lo marcamos y vamos a ENABLE para que lo conecte.

Luego vamos al BOTON GO que es para que el programa siga corriendo. Vemos que vuelve a parar en ese salto y que algunas sentencias más abajo esta el cartel **YOU HAVE SOLVED**. Esta listo para saltar, como indica el debugger, pero si lo dejamos que salte no saldrá el cartel que queremos, así que habrá que invertirlo. Podemos hacerlo aquí mismo

cambiando el **1C** por **1D** que es BranchT con lo que estaría invertido el salto.

Aquí pulsamos el botón EDIT y doble cliqueamos encima del **1C** que está arriba de todo y queremos cambiar. Nos aparece una ventanita donde cambiamos por **1D** y luego vamos a PATCH NOW con lo que habremos cambiado en la memoria este valor (copiamos la cadena de números que siguen al 1c para luego modificarlos con el Ultra Edit).

Cerramos esa ventana quitamos el BPX con DISABLE y luego vamos a GO y cada vez que ante cualquier combinación ponemos CHECK nos sale el cartel YOU HAVE SOLVED IT.

Por supuesto para cambiarlo definitivamente abrimos el ULTRA EDIT, cerramos el debugger y como ya habíamos copiado en la ventana de EDIT la cadena que sigue al **1c,** que es **1c 07 05 27 e4 fe 27 04 ff 27 24 ff f5 00 00 00,** la buscamos aquí y cambiamos el **1c** por **1d** y programa crackeado.

No saben lo que es crackear esta pavada sin este debugger, una pesadilla. Así que, mil gracias a los autores de esta linda herramienta que facilita el trabajo una barbaridad.

Seguiremos aprendiendo a usarla de a poco en lecciónes siguientes.

Hasta la 45
Ricardo Narvaja

LECCIÓN 45: SIGAMOS CON EL P-CODE (TIEMBLEN CRACKERS... "EL CUENTAPASOS")

La verdad que este programa siempre fue mi clavo, una espina clavada y creo que la de muchos crackers, pude hacer un crack parcial pero tenia algunos problemas. Con el nuevo VBdebugger para solucionar el P-CODE puede facilitarse (UN POCO).

Además contamos con un tutorial sobre este tema en la pagina en la cual se baja el debugger que esta en castellano y esta muy bien, aunque yo no me limito a copiar, lo voy haciendo y según como me va resultando hago lo mismo o no, y trato de facilitar las cosas si hay algo complicado, en la medida que puedo.

Bueno este tutorial es para el CUENTAPASOS 3.80 REVISION 394 que es el ultimo que salió. Se baja de http://www.cuentapasos.com

Lo instalamos y tomamos el ejecutable cpasos32.exe y lo analizamos con algún analizador de filas, como el un-pack. No dice que esta comprimido con NEOLITE 2.0 el cual esta incluido en los descompresores que trabaja el PROCDUMP con lo que se puede descomprimir fácilmente.

Sabemos que el debugger no trabaja con programas comprimidos así que ahora que esta descomprimido debería funcionar.

Una vez descomprimido lo copiamos a la carpeta del programa, lo ejecutamos y nos sale el cartel:

> "No se ha encontrado el conjunto de opciones de INICIO C\Windows\Unpacked.ini......."

Bueno dado que como yo tengo el ejecutable descomprimido con el nombre UNPACKED.exe en este ultimo cartel dice C\WINDOWS\Unpacked.ini por lo que quizás este cartel aparezca por tener un nombre distinto el ejecutable. Cambiémoslo a Cpasos32.exe a ver que pasa.

BUMMM, no sale el cartel anterior pero sale otro que dice "No existe ninguna tarifa, se va a crear una tarifa de ejemplo".

Aquí dado que este cartel aparece cuando ejecutamos el programa descomprimido y no cuando ejecutamos el original, puede ser que el programa testee el largo de la fila Cpasos32.exe o el checksum. Ya veremos que es la primera opción la que hace aparecer este cartelito.

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

Voy a GO para que arranque y para en

004C57BF: 5E ImpAdCallI4 rtcFileLen on address 0F036192h

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

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

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

Aquí vemos el número **1033728** que es el largo del ejecutable y la función **FMEMStI4** lo que hace es volcar el contenido del STACK (que si vamos al cuadrito de la derecha y cambiamos la opción a DWORD podremos ver que allí esta el valor FC600 que es el hexadecimal de 1033728) y lo guarda en la posición de memoria **4c8590.**

Si vamos a MEMORY DUMP y buscamos que muestre el contenido de la memoria 4c8590 vemos que hay solo ceros; ahora al apretar F8 y ejecutar la sentencia se llenara con los valores 00 C6 0f que es fc600 al revés.

Bueno ya vimos que guarda el valor del largo en esa posición de memoria. Ahora, para FMEMStI4 (la palabra St debe significar STORAGE o sea GUARDAR) debe haber entre los opcodes una función opuesta que cargue el valor ese de la memoria al STACK para operar o compararlo.

Si miramos entre los OPCODES

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

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

Parará varias veces allí pero son los valores que carga de distintos lugares de la memoria al STACK. Por ejemplo para aquí

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

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

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

Aquí vemos de nuevo que cuando hagamos F8 para ejecutar esta sentencia va a tomar el valor que había guardado en **4c8590**, el valor fc600 que es el largo de la fila descomprimida, y lo va a cargar al STACK.

Hacemos F8 y en el cuadro superior derecho, siempre que este activado DWORD, veremos el Fc600 allí. En **4c5fff** carga de nuevo el mismo valor en el STACK, cuando hacemos F8. Luego lo compara con **AAE60** que debe ser el valor que debería tener el ejecutable una vez descomprimido de NEOLITE.

Luego viene el salto condicional que esta después y que invertiremos a ver que pasa.

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

Cambiando **1C** por **1E** es cambiar por BRANCH que es equivalente a JMP, o sea que salte siempre ya que el programa nos avisaba que no iba a saltar.

Vamos a EDIT reemplazamos el 1C por 1E y volvemos. Le damos a GO y no sale el CARTELITO más. Hemos acertado!!!

Podemos volver al mismo lugar para copiar la cadena que hay que reemplazar con el ULTRAEDIT para que el cambio no sea solo en la memoria y se guarde

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

Y HAY QUE CAMBIAR EL 1C POR 1E

CON ESO YA TENEMOS EL PRIMER SALTO PARCHEADO.

Ahora vamos por la nag, o sea ese cartel que dice los días que te quedan y que tiene los botones aceptar, cancelar, etc.

Aquí hay que aplicar el método que usamos a veces en el SOFTICE de ir traceando con F10 hasta que aparezca la NAG. Allí anotamos la sentencia que provoco la aparición y volvemos a arrancar el debugger, y ponemos un BPX en esa dirección que hizo aparecer la NAG. Entramos ahora con F8 y seguimos con F10 hasta que vuelva a aparecer la NAG y repetimos el proceso anotando cuando apareció y poniendo un bpx allí. Y luego, cuando volvemos a arrancar el programa y para allí otra vez, dentro con F8 y así llegaremos a uno de los dos lugares en donde aparece la NAG, ya que este Tobarra (el autor) puso dos rutinas gemelas, y a veces a unos le salta la NAG en una y a otros en otra.

Ambas son

4BB3B7: 0d VCallHresult meth__imethSHOW

4BB5ED: 0d VCallHresult meth__imethSHOW

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

Esto se puede hacer con el programa **EXDEC** que es como un WDASM para P-CODE y muestra un listado muerto de todas las sentencias, o se puede

poner un BPX un poco anterior a estas sentencias y buscar un salto que las esquive.

Los saltos están en

4BB28A: 1c BranchF: 4bb3e4

4BB4E6: 1c BranchF: 4bb61a

Cada uno de estos saltos si cambiamos 1c por 1e se convertirá en un BRANCH o sea como si fuera un JMP y salta sobre los CALL respectivos, haciendo que el programa arranque y no aparezca la NAG y encima no venza. **Si vamos a ABOUT vemos que siempre esta en el día cero.**

Bueno, una espina menos. Pongamos las cadenas que hay que parchear en ambos saltos:

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

Y EL OTRO SALTO

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

En ambos cambiamos 1c por 1e.

LA VERDAD, QUIEN PENSARIA QUE CAMBIANDO SOLO TRES VALORES ESTARIA VENCIDO ESTE COLOSO.

HASTA LA 46
RICARDO NARVAJA

LECCIÓN 46: HAGÁMOSLO CON TRW2000 (TIRO AL PICHON)

Al final de este tute pongo las claves para ingresar a los nuevos drives donde están los programas de TODAS las lecciónes así como las mismas lecciónes y las herramientas más usuales para crackear.

En esta lección crackearemos con TRW2000. Utilizaremos este programa en vez del SOFTICE dado que no solamente hemos hecho muchos tutoriales con Softice, sino que además no hay muchos tutoriales con TRW2000 y es bueno saber usarlo dado que hay veces que el Softice no se puede utilizar y entonces hay que trabajar con TRW2000. Así que, cada tanto realizaremos un tute para TRW2000 así practicamos.

La victima es un programa llamado **SCIENTIFIC NOTEBOOK** y el link para bajarlo es ftp://www.mackichan.com/download/version351/scinoteb351.exe o de la pagina del programa http://www.mackichan.com/ donde hay otros dos programas llamados **SCIENTIFIC WORKPLACE** y **SCIENTIFIC WORD** que tienen una protección similar. Así que, si descubrimos esta habremos crackeado tres en vez de uno.

Yo decidí bajar el SCIENTIFIC NOTEBOOK porque es el que menos pesa (20 megas aprox). Los otros son mucho más pesados (50 megas aprox).

Bueno, instalemos el programa y veamos, ejecutémoslo, tomemos como norma siempre hacer una copia del ejecutable original y guardarlo en el escritorio o en algún lugar por si acaso corrompemos sin querer el ejecutable original podamos rescatar el funcionamiento sin tener que instalar todo de nuevo.

Bueno, en el programa yendo a HELP-REGISTER y poniendo el punto en **"I have and unlock code and want to enter it"**, nos aparece la pantalla para ingresar la clave.

Aquí la sorpresa es que, además de la clave para registrar el programa y que no expire, también hay otros números de serie para registrar el programa para SPELLING DICTIONARYS y COMPUTACIÓN ENGINE. O sea, que para habilitarlo todo necesitaríamos como 20 claves para los distintos idiomas y para habilitar el resto de las opciones que tiene el programa.

Bueno, ya que avisa allí que la clave tiene que tener doce caracteres pongamos una cualquiera. Llenemos el cuadro con una clave falsa para cada opción (las tres principales nada más) a ver que pasa cuando nos queramos registrar.

Luego vamos a SAVE YOUR LICENSE FILE cliqueamos allí y nos dice que para habilitar todas las nuevas opciones tenemos que cerrar y volver a abrir el programa.

Bueno hacemos eso y cuando lo volvemos a abrir es lógico que verifica si el serial es correcto.

Nos aparece un cartelito, que parece ser del tipo messageboxa, que nos dice **LICENSE VERIFICATION ERROR**, y después que si queremos ver los detalles. Luego arranca el programa y si vamos a HELP - SYSTEM FEATURES que es donde muestra las licencias habilitadas vemos que seguimos igual sin licencia.

VIEWER available
ENGLISH installed not licensed

No han cambiado mucho las cosas por aquí. Si cerramos y volvemos a abrir el programa sin ir a la ventana de registro sigue apareciendo el cartel LICENSE VERIFICATION ERROR.

Bueno, abramos el TRW2000 y vayamos a BROWSE para buscar el ejecutable que esta en **C:\Archivos de programa\Scientific Notebook** y se llama **scinoteb.exe;** luego vamos a LOAD para que comience a ejecutarse.

Rompe en el inicio del programa en el trw2000 y hacemos f10 para ejecutar una sola sentencia y ya estar en el punto de entrada del programa.

Aquí en el TRW2000 ya que los números hexadecimales que corresponden a las sentencias no se muestran tenemos que activarlos a mano tecleando CODE ON. Con eso ya estaremos más cerca de lo que estamos acostumbrados a ver en el SOFTICE.

Bueno, ya que esa ventanita que dice LICENSE VERIFICATION ERROR parece una messageboxa, pongamos pues un BPX messageboxa para ver si para allí el TRW2000. Hagamos X para que corra y para dentro del MESSAGEBOXA.

En la línea donde marca que archivo se esta ejecutando vemos USER32 así que para que termine de ejecutarse esta función de windows y volver al ejecutable, a la sentencia posterior al punto donde entro, tecleamos F12. Sale el cartelito maldito de LICENSE ERROR y luego de aceptar en el cartel, ponemos que no queremos ver los detalles y entonces llegamos hasta el RET donde termina esta función MESSAGEBOXA.

Aprieto F10 y salgo a la sentencia posterior a la llamada al messageboxa en **6e8302.**

Aquí vemos una diferencia entre el SOFTICE y el TRW2000. Si hubiéramos tecleado F12 en el mismo punto, en el SOFTICE este nos hubiera depositado directamente aquí, **en 6e8302.** En el trw2000 con f12 se llega hasta el RET y luego hay que teclear F10 para llegar a la misma posición.

Bueno, una vez que llegamos allí, a la sentencia posterior, anotamos el CALL que corresponde al MESSAGEBOXA que está en la sentencia anterior o sea **6e82fc.**

Supuestamente debería encontrar un salto anterior que permita esquivar esta sentencia y que no se ejecute. Así que busco con CTRL + FLECHA DEL CURSOR PARA ARRIBA en las sentencias anteriores a 6e82fc algún salto condicional que esquive ese call o sea que salte después de 6e82fc para que no aparezca el cartelito de LICENSE ERROR.

Hay algunos saltos anteriores JZ, JA, JNA, JNZ, pero todos saltan antes del call maldito, ninguno lo salta por encima a posiciones de memoria posteriores a 6e82fc, por lo que no nos sirven.

Volvemos a hacer F12 para terminar hasta el RET y luego F10 para volver a la sentencia posterior a la entrada del CALL donde estábamos.

Vemos que caemos en **5b0510** y que la sentencia anterior ahora que hay que esquivar para que no salga el cartelito es la anterior **5b0510 Call 6e824a.**

El que quiere probar que vuelva a arrancar el programa y cuando para en el inicio que ponga BPX 5b0510 y va a parar allí y al ejecutar el CALL con F10 aparecerá el cartelito maldito.

Bueno repetimos el proceso y buscamos con CTRL + FLECHA ARRIBA algún salto que pase por encima de este CALL. Hay uno solo en **5b048a jz 5b0493** que no salta a posiciones de memoria posteriores a **5b0510,** por lo que no sirve.

Otro no se ve, así que repitamos el proceso y volvamos a hacer F12 y F10 para llegar a la sentencia posterior al Call donde estábamos antes y esto nos hace caer en **6e8353** y el Call anterior que hay que esquivar es **6e834d call near [edx+8c].**

Buscamos para arriba algún salto anterior y hay uno en **6e833e jz 6e8355** que saltearía justo el CALL maldito del cartel LICENSE ERROR.

Arranquemos de nuevo el programa y desde el trw2000 y cuando se inicia pongamos un BPX en el salto ese o sea BPX 6e833e.

Cuando para allí vemos que a la derecha del salto dice NO JUMP así que para invertirlo y que salte por encima del CALL maldito hay que hacer que JUMP o sea que salte a 6e8355.

Como EIP marca la próxima sentencia que se va a ejecutar, cambiando el valor de EIP a 6e8355 haremos que salte allí .

En el SOFTICE con poner **R eip = 6e8355** ya estaba listo. Aquí hay que teclear **R** y **enter;** con eso se resalta EAX arriba entre los registros y luego, con las flechas de cursor, se mueve hasta que quede resaltado EIP y ahí tecleamos 6e8355 y listo se invirtió en salto.

El salto, al ejecutar F10, pasa por encima del CALL maldito 6e834d pero si sigo tecleando F10 vemos que luego de haber esquivado el CALL que queríamos, al pasar por encima de otro CALL posterior que esta en 6e8360, aparece también el cartel de LICENSE ERROR. Eso quiere decir que el salto ese no sirve y para esquivar los dos carteles juntos debemos seguir hasta el RET con F12 y luego F10 y caer en la sentencia posterior al CALL que contiene los dos carteles. Si saltamos ese CALL esquivaremos ambos juntos.

Ahora caímos en **63da13.** Vemos que el CALL maldito ahora es **63da0e,** el que contiene los dos carteles de error dentro y que hay que esquivar.

Busquemos un salto anterior mirando hacia arriba con CTRL + FLECHA PARA ARRIBA. Hay un **JMP 63da13** que no es condicional y que saltearía por encima el CALL pero también hay un salto condicional anterior al JMP que lo pasa por encima y nos tira directo dentro del cartel maldito en 63da0e.

El salto condicional esta en **63d9f3 jz 63da07.** Ponemos un BPX allí y arrancamos de nuevo el programa. Veo cuando para que dice JUMP y que va a pasar por encima del salto que esquiva el cartel, así que tengo

que invertir este salto para que en vez de JUMP no salte y siga en la sentencia siguiente, o sea, debo cambiar EIP a 63d9f5 y así sigue ejecutándose allí.

Sigo con F10 y al pasar por **63da00** sale otro cartel de LICENSE ERROR distinto al de **63da0e**, por lo tanto el salto no sirve y debemos salir con F12 y F10 hasta la sentencia posterior a este CALL que contiene estos cartelitos para saltearlos a los dos juntos.

Caigo en **63d40a** y el CALL ahora a esquivar es la sentencia anterior o sea **63d405**.

Miramos hacia arriba y hay unos cuantos saltos. Prueben si quieren practicar pero no sirve ninguno ya que no arranca bien el programa (prueben ustedes que pasa).

Vuelvo a apretar F12 y F10 y aparezco ahora en **63dac8** y el CALL a saltear es **63dac3**. Los saltos anteriores tampoco sirven si los invierto.

Vuelvo a hacer F12 y F10 y caigo en **51c41c** y el call a esquivar es la sentencia anterior, **51c417**.Los saltos no hacen nada bueno.

Vuelta F12 y F10 y aparezco en **51c1f0** y el call a saltear es **51c1eb**. Hay un salto anterior en **51c1e7**. Arranco de nuevo el programa y cuando se inicia pongo un BPX allí.

Cuando para dice que no va a saltar (NO JUMP), si lo invierto de la forma que hicimos las veces anteriores no me sirve porque para como veinte veces en el mismo lugar y cada vez que para debo hace R y cambiar EIP por lo que mejor es cambiar la sentencia directamente una sola vez.

Tecleo A y ENTER cuando estoy encima del salto (si te dice que ese comando se puede usar solo en versiones registradas buscate otro TRW2000 bien crackeado). Luego de teclear A y ENTER escribo la nueva sentencia **JMP 51c1f8**, borro el BPX con BC* y arranco el programa con X.

Bueno, por lo menos arranca el programa pero si voy a HELP-SYSTEM FEATURES vemos que las licencias no se han habilitado.

Hay allí varios saltos anteriores para probar pero el correcto es el que esta un poco más arriba en **51c195 Jz 51c214**. Pongo un BPX allí y arranco el programa de nuevo.

Copio la cadena de números hexadecimales (ACORDARSE DE CODE ON SI NO NO SE VAN A VER) correspondiente a ese salto y algunos más que le sigan.

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

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

Borramos los BPX y le damos X para ver que pasa.

Sale un cartelito que dice que hay un error en la licencia, que tenemos el número de licencia de otro programa de la serie, aceptamos este y luego aparece otro; aceptamos y arranca el programa.

Si vamos a HELP - SYSTEM FEATURES vemos que el programa habilito TODAS las funciones diccionarios y engines. Sale una lista de 30, todas habilitadas con su licencia.

Por lo tanto, vemos que esos cartelitos anteriores de que teníamos la licencia de otros programas los pusieron para despistar.

Bueno podemos buscar la cadena con el ULTRA EDIT y cambiar el 74 por EB y hemos hecho la primera parte.

La tarea para el hogar, (QUE YO YA HICE) y no es tan difícil como lo que ya se hizo, es que ustedes por el mismo método encuentren el salto que complementa al que ya parcheamos y que hace que no salgan estos otros dos cartelitos falsos.

No lloren no hay que hacer tantos F12 como en la lección, unos poquitos no más. Ya saben BPX messageboxa y luego F12 y F10; así repetir el proceso buscando un salto que esquive estos dos carteles.

AYUDA: El salto esta en **_ _ _ _ _ 8: JZ _ _ _ _ _ E**

Completar aquí para saltear los dos carteles.

FIN

Para ingresar a los drives donde están los programas no hay que registrase ni sacar un drive propio ni nada, solo ir a http://www.oodrive.com/ y entrar en la pestaña VISITEUR y allí poner las siguientes claves:

Para el primer drive con los programas que se crackearon en el curso

user: ricnar456
pass: ricnar456

Para el segundo drive con más programas de las lecciónes, más las mismas lecciónes y algún tutorial:

user: ricnar457
pass: ricnar457

Y para entrar al tercer drive donde hay herramientas para crackear

user: ricnar001
pass: ricnar458

Hasta la proxima
Ricardo Narvaja

LECCIÓN 47: DURO PERO UTIL

Este programita es, como dice el titulo, bastante duro para crackear pero es útil para todos aquellos que se la pasan convirtiendo archivos MP3 a WAV y MP3PRO entre otras cositas. Además, tendremos también el reproductor de MP3PRO. La verdad, nunca había escuchado nada en este formato nuevo y quede sorprendido al ver que un archivo Mp3 de 2.6 megas al transformarlo en MP3PRO quedo reducido solo a 1 mega y se escuchaba igual o mejor.

Bueno este programita añade una opción cuando hacemos click derecho sobre un MP3, la opción CONVERT, y allí ya nos sale si queremos convertirlo a WAV a MP3 o a MP3PRO. Es muy útil y rápido, muchas veces al intentar grabar un CD el programa nos dice que el archivo esta muy comprimido y tenemos que hacerlo WAV de nuevo y volverlo a hacer Mp3 lo cual significa abrir dos programas cargar el archivo y blablabla. Aquí, con hacer click con el botón derecho encima del tema ya casi lo estamos convirtiendo.

Bueno, adelante, el programa se llama **dbPoweramp** y es gratis una parte. Luego se baja un agregado que se llama **POWERPACK** que vence a los 30 días y es totalmente operativo. Además de agregarle muchas opciones permite agregar efectos de sonido y muchas cositas más.

El programa hay que bajarlo de http://admin.dbpoweramp.com/ y el link directo para bajarlo es http://admin.dbpoweramp.com/bin/dMC-r7.exe

El POWERPACK se baja de http://admin.dbpoweramp.com/bin/dMC-PowerPack.exe

Los codecs para MP3pro se bajan de http://codecs.dbpoweramp.com/store-alpha/codec-central.htm

user: codecs
pass: dream23

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

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

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

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

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

No conviene parchear este programita porque solo nos dice el estado en que está funcionando; por más que lo parcheemos y le hagamos decir que estamos registrados el POWERPACK vencerá a los 30 días así que, este

powerpack status nos ayudara para saber el estado de las cosas, no lo parchearemos.

Para ver como trabaja nos posicionaremos encima de un archivo MP3 y haremos click derecho encima y nos posicionaremos encima de la opción CONVERT sin hacer clic. Entraremos en el SOFTICE y pondremos un BPX GETVERSION a ver que pasa. Saldremos del SOFTICE y haremos, ahora si, click en CONVERT. Caemos en el SOFTICE y después de unos 20 F12 más o menos vemos en la línea verde que llegamos al ejecutable MUSICONVERT.

Ejecutamos un par de sentencias con F10 hasta llegar al RET y salir de allí y caemos en **413b50,** donde podremos un BPX para que cada vez que queramos ingresar al ejecutable no tengamos que repetir lo del BPX getversion.

Caemos por aquí:

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

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

La siguiente STRING REFERENCE podría ser útil

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

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

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

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

```
:004105F9 90            nop
:004105FA 90            nop
:004105FB 90            nop
:004105FC 90            nop
:004105FD 90            nop
:004105FE 90            nop
```

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

Y no hay ninguna REFERENCE de donde puede provenir esto.

Aquí los programadores han leído muchos tutes de crackers y saben que a veces uno se guía por las referencias que obtiene en un desensamblador como el WDASM por lo que para ingresar a esta parte llaman a la rutina desde una sentencia como:

CALL ESI o **CALL [EAX+05]**

... o algo así donde el WDASM no puede saber que valor tiene ESI o EAX+05 por lo que no puede determinar de donde proviene la llamada a la rutina.

Distinto seria si fuera una llamada numérica como **CALL 4105f9** la que llamara a la primera sentencia de esta rutina entonces el WDASM nos diría todos los lugares donde se encuentra una llamada así y sabríamos de donde proviene.

Podríamos adelantar el reloj para que venza y llegue hasta esta zona, y después hacer F12 para al salir de esta rutina volver a la sentencia posterior a donde fue llamada.

Yo seguiré por otro camino. Les comento que si adelantan el reloj para que venza, no aparecerá más el POWERPACK aunque luego vuelvan a atrasar el reloj de nuevo.

Para desbloquear el POWERPACK y que vuelva a funcionar, (esto se puede ver con el REGMON una vez bloqueado) el programa pone una marca en el registro para que no se pueda utilizar más.

La clave es **[HKEY_CURRENT_USER\Software\SCC]** y hay que ingresar allí el valor 0 para que vuelva a funcionar y, por supuesto, volver a retrasar el reloj sino volverá a bloquearse.

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

Entre las SRTINGS hay también una REGISTERED pero al igual que la anterior el comienzo de esa larga rutina no tiene ninguna REFERENCE para ver de donde proviene y al estar desregistrado ningún BPX en esa zona para allí por lo que es una parte que no podremos acceder.

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

```
* Possible StringData Ref from Data Obj ->"Registered: "
|
:00409981 BF0C394200      mov edi, 0042390C
..
..
:004099AB E850660000      call 00410000
:004099B0 83C404          add esp, 00000004
:004099B3 8D942478010000  lea edx, dword ptr [esp+00000178]
```

```
:004099BA 84C0            test al, al
:004099BC 740A            je 004099C8

* Possible StringData Ref from Data Obj ->"Yes"
|
:004099BE BF08394200      mov edi, 00423908
:004099C3 E91E010000      jmp 00409AE6

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004099BC(C)
|
* Possible StringData Ref from Data Obj ->"No"
```

Bueno, era posible que modificando dentro del CALL para que salga siendo AL=1 nos registráramos ya que eso hace que vayamos a la palabra YES y salteemos el NO, pero alguna comprobación posterior hace que no arranque el POWERPACK así que, como no tenemos más tiempo, dejaremos abierta esta vía de crackeo para el que la quiera investigar.

Trataremos de que el programa no venza en vez de registrarnos, igual funciona muy bien, así que miremos un poco esto.

Utilicemos el REGMON para ver si encontramos donde guarda la fecha de inicio de los 30 días de plazo que te da. Cualquier clave rara que encontremos puede ser, abramos el REGMON y en filtro pongamos **MUSIC*** y arranquemos haciendo click derecho sobre un archivo MP3 hasta que arranque.

Nos salen varios resultados pero mi ojo avizor sospecha de algunos en particular, veamos:

Esto me resulta muy sospechoso:

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

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

```
* Reference To: KERNEL32.GetSystemTime, Ord:015Dh
|
:00410B2E FF152CF14100  Call dword ptr [0041F12C]
:00410B34 8D54240C      lea edx, dword ptr [esp+0C]
:00410B38 8D44241C      lea eax, dword ptr [esp+1C]
:00410B3C 52            push edx
:00410B3D 50            push eax
```

```
* Reference To: KERNEL32.SystemTimeToFileTime, Ord:029Bh
|
:00410B3E FF1530F14100  Call dword ptr [0041F130]

* Possible StringData Ref... ->"Software\Illustrate\dBpowerAMP"
|
:00410B44 68242A4200    push 00422A24
:00410B49 6801000080    push 80000001

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

Más sospechoso imposible, justo es abierta esa clave del registro después de hacer GETSYSTEMTIME o sea, después de que el programa se entera de la fecha que es. Quizás esta cifra que esta en **dmcppi** sea la fecha inicial de prueba del programa pero encriptada.

Hagamos una prueba, adelantemos el reloj un año y antes de ejecutar el programa, vayamos a HKEY_CURRENT_USER\Software\Illustrate\dBpowerAMP y marquemos **DMCPPi** y exportemos la clave al escritorio o algún lado seguro. O sea, luego de marcarla, REGISTRO - EXPORTAR ARCHIVO DEL REGISTRO, para poder jugar con esta clave a ver que pasa y si nos equivocamos poder restaurarla como estaba antes.

Ahora, luego de haberla exportado borrémosla. Ejecutemos el programita que nos dice el STATUS de cuantos días nos faltan para que venza POWERPACK STATUS.

DICE QUE NOS QUEDAN 30 días para probar jua jua, y si vamos a ver y cerramos y volvemos a abrir la misma clave vemos que creo otra diferente con lo que debe ser la fecha actual encriptada a partir de la cual nos da de nuevo 30 días.

Si volvemos otra vez el reloj hacia atrás tenemos que borrar esta clave de nuevo para que nos cree la clave actual porque si lo ejecutamos fuera de fecha vencerá y se bloqueara para lo cual habrá que buscar para desbloquearlo la otra clave que hallamos al principio del tute.

Bueno, la cuestión es que el programa busca la clave esta y si no esta, crea una nueva. Podemos borrar la clave vieja y poner un BPX en REGCREATEKEYEXA para ver donde crea la nueva clave.

Para en

```
* Reference To: ADVAPI32.RegCreateKeyExA, Ord:015Fh
                                    |
:00418078 FF151CF04100  Call dword ptr [0041F01C]
:0041807E 85C0          test eax, eax
:00418080 7529          jne 004180AB
:00418082 8B44240C      mov eax, dword ptr [esp+0C]
:00418086 8B4C2408      mov ecx, dword ptr [esp+08]
:0041808A 8B542404      mov edx, dword ptr [esp+04]
:0041808E 50            push eax
:0041808F 8B442418      mov eax, dword ptr [esp+18]
:00418093 51            push ecx
:00418094 6A03          push 00000003
:00418096 6A00          push 00000000
:00418098 52            push edx
```

```
:00418099 50            push eax

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

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

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

Caemos en el mismo lugar que habíamos visto antes

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

Ahora veamos que un poco más arriba hay un salto condicional que puede evitar que llegue a crear una clave nueva o sea, que eso debe ocurrir normalmente cuando el programa encuentra la clave, y si no la encuentra crea una nueva. Lo que podemos hacer nosotros es parchear ese salto para que esté o no esté la clave SIEMPRE caiga donde crea una nueva y cada vez que se ejecute el programa nos de 30 días más, jeje.

Aquí vemos el salto a parchear

```
:00410B5D 751E          jne 00410B7D

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

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

Podemos probar a poner un **BPX 410b5d** y veremos que cuando para allí siempre quiere saltar a no ser que hayamos borrado la clave donde guarda la fecha de inicio.

Por lo tanto, cambiemos esos dos números del salto por **90 90** y no saltara nunca; siempre CAERA EN EL CALL donde esta createkey y tendremos 30 días más de uso cada vez que lo ejecutemos.

La cadena a modificar en el ULTRA EDIT es

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

Donde debemos cambiar **75 1E** por **90 90**

Hasta la 48
Ricardo Narvaja

LECCIÓN 48: BUSCANDO CLAVES UN POCO MÁS DIFÍCILES

En esta oportunidad buscaremos la clave de un programa que es un poco bastante más difícil que lo normal. Quizás no llegue al nivel del encriptamiento furioso, ni nada que ver ya que la rutina de comparación es simple.

El problema es que pone un montón de partes para despistar como si estuviera encriptando, y yo llene como seis hojas de paginas con un encriptamiento de la clave que era falso, que era solo para despistar.

Por lo tanto, en casos de claves que hay mucho que perseguir, la táctica es, desde que ponemos la clave falsa y caemos dentro del HMEMCPY hasta que llega al cartel de que la clave es incorrecta, hacer una primera pasada superficial luego de poner los BPRs donde esta en la memoria la clave falsa, y anotar todos los lugares donde va parando y mirando un poco, sin seguir por primera vez por caminos donde empieza a hacer operaciones con los números de la clave falsa.

En mis hojas de anotaciones hay un montón de lugares donde paraba y tomaba uno o varios números de la clave y los empezaba a sumar y a operar con otros. Inclusive hay un ciclo terrible donde va tomando cifras, les hace mil cuentas, las suma, divide, multiplica etc etc, las guarda y sigue, y al final es una farsa completamente inútil.

Así que, yo voy a hacer un breve repaso de los lugares más importantes donde para sin operar demasiado hasta que llega a la rutinita de comparación. Suprimiré muchas de las partes puestas para despistar, que si uno anota donde comienzan y sigue adelante sin prestar atención, llega a la rutina de comparación perfectamente, y si uno no la encuentra entonces si, vuelve atrás y comienza a revisar con más cuidado lo que pasó de largo.

El programa a analizar es el http://ftp.idg.se/programbanken/pc/grafik/snag3243.exe y no es la ultima versión que hay aunque no ha cambiado mucho en cuanto a la forma de hallar la clave. Creo que las mismas rutinas estan pero en distintos lugares en la versión nueva.

Hice una anterior porque un amigazo me pidió que había tratado con esta versión y que lo ayude y entonces me puse a hacer esta.

Ya saben que los oodrives que había puesto para guardar los programas que se crackean parece que van a comenzar a cobrar. Hasta ahora están allí, no se por cuanto tiempo más, pero también están los kturns en los cuales cualquiera yendo a http://www.kturn.com/ y sacándose un kturn propio, que es gratuito, te da 125 megas y puede desde el suyo acceder a los míos.

Entra al propio kturn, ve a donde dice OTHER USERS y allí pon cualquiera de estas tres claves:

ricnar458
ricnar457
ricnar456

que son los tres diferentes kturns que hay del curso con los programas, herramientas, y todo lo necesario.

Bueno, sigamos adelante. Lo primero que hay que hacer luego de instalarlo es localizar donde guarda en la memoria la clave falsa que yo introduzco.

Para eso veamos primero como se hace en SOFTICE y después como se podría hacer en TRW2000.

Pongo la clave falsa 9898989898. Una vez que para dentro del HMEMCPY hay que tracear hasta que lleguemos a una sentencia **Repz movsd** en **015f:9f20**. Allí hago **d es:di** que es donde va a guardar la clave falsa.

Sigo con F10 hasta unas sentencias más abajo, hasta **Repz movsb** y al ejecutarlo se termina de copiar la clave falsa a la memoria. Si no se ve hay que subir una línea en la ventana de datos; allí veo en la parte del contenido **39 38 39 38...** (que son los números hexa que corresponden a 989898...) y a la derecha 989898...

Ahora tecleo PAGE más la dirección de memoria donde esta la primera cifra de la clave. Esto puede variar, en la mía es **1197:0000**, por lo tanto hago **PAGE 1197:0000** y me sale como resultado

LINEAR
6ff450

que es la posición de memoria de 32 bits donde esta la clave.

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

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

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

S 0 L FFFFFFFF '9898989898'

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

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

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

En nuestro caso sería **BPR 6ff450 6ff460 rw** (SOFTICE). Ya que la clave abarca desde 6ff450 hasta 6ff460 entonces, con este breakpoint,cada vez que lea o escriba allí parará.

Lamentablemente (Y MUCHO) en el TRW2000 no existen breakpoints de rango; solo existen BPMs individuales o sea, poner un **BPM 6ff450 rw** haría que parara allí cuando lea o escriba algo solo en la primera cifra de la clave.

Encima, por la arquitectura del BPM diferente al BPR no para justo cuando lee o escribe de 6ff450 sino, parara justo la sentencia posterior así que cuando pongamos un BPM y pare siempre hay que mirar la sentencia anterior a la que paró.

Primero borramos con BC* el BPX HMEMCPY. Bueno, que vamos a hacer paciencia si usamos TRW2000. Paciencia, pongamos un BPM 6ff450 rw, BPM 6ff451 rw y así sucesivamente en todas las cifras.

Luego de hacer eso hay que hacer X y ver cuando el programa trata de hacer algo con la clave. Aquí entonces, habría que ir analizando cada vez que el programa trata de hacer algo con la clave, y si la trata de copiar a otra dirección volver a poner BPRs en los lugares donde la va copiando o guardando para que siempre el SOFTICE pare cuando trate de hacer algo con las cifras de la clave.

Bueno, aquí vamos a empezar a bailar de a poquito, yo me olvidé pero tenemos que cambiar la clave falsa por otra más practica en vez de 989898... que es la que a mi me gusta.

Aquí vamos a usar otra, 123456789ABCDE, que nos da la ventaja de saber fácilmente por el valor que cifra esta trabajando ya que en la otra si trabaja con un nueve no sabemos si es el primero el tercero, el quinto o cualquiera de los otros. Así que, repitamos todo de nuevo con la clave **123456789ABCDE**.

Bueno, ya tenemos todo listo para empezar. Traceamos un poco antes de empezar a darle a X y en la comparación de **418850** vemos que si la clave que poníamos era menor que 14 cifras nos tiraba al tacho ya que EBP tiene el número de cifras y 0E es 14 en decimal.

Como la que pusimos es de 14 cifras no salta y sigue dentro de la comparación.

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

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

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

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044989D(C)
```

```
|
:004498A6 8BC1          mov eax, ecx
```

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

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

Bueno, seguimos adelante y caemos en **41887a**.

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

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

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

Aquí, en **41887ª**, compara la primera cifra con **2d** que es el guión "-". Si no es guión salta el subsiguiente salto que llega a **4188d2**. Allí carga esa cifra a DL y la copia en el contenido de ECX o sea en el mismo lugar que estaba.

Hace eso cifra por cifra; bueno, si en **41887a** encuentra un guión directamente no lo va a guardar y pasa a la cifra siguiente con lo que transforma una clave con guión en una completamente limpia sin guión. Vaya a saber porque hace esto. Bueno, lo tendremos en cuenta.

Bueno, adelante. Caemos en **bff7117e REPNZ SCASB** y... que demonios es esto. La experiencia del que crackeó mucho sabe que esta sentencia se encuentra dentro de la función **lsrtlen** que es para sacar la cantidad de cifras nuevamente.

Una vez que traceando con F10 llegamos al ejecutable en EAX quedó la cantidad de cifras de la clave de la memoria. EAX tiene 14 en hexa es 0E.

Estamos en 435cA3 (**00435CA3 50 push eax**) y, dado que va a hacer PUSH EAX, justo que EAX tiene el valor de la cantidad de cifras y lo va a guardar veamos como es el mecanismo y donde lo guarda con la sentencia PUSH.

La sentencia PUSH guarda en el STACK que es como un archivo que no deja de ser parte de la memoria. El registro ESP, que quiere decir STACK POINTER, es el que apunta a donde esta el stack y cual es el valor que, dado que el stack es una pila de valores, apunta al valor que esta encima de la pila.

Al hacer PUSH EAX agregamos un valor al stack y si vemos con **D ESP** allí en **6feec0** está como primer numerito el **0e** que guardo luego de ejecutar la sentencia PUSH EAX.

Bueno, ya que está allí guardado por algo (???), pongamos un BPR allí en el SOFTICE o un BPM en el TRw2000 por si alguna vez lo busca de allí para realizar alguna comparación.

BPR 6feec0 6feec0 rw (softice)
BPM 6feeco rw (trw2000)

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

Para ahora en **448b5c** y empieza la joda.

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

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

Luego toma las siguientes y las copia a continuación y las próximas también o sea que en 6fef04 queda **123456789ABC**.

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

Entonces pongo

Bpr 6fef04 6fef13 rw (softice)

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

y sigo con X.

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

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

Mueve la primera cifra a AL y luego la copia a [EDI] que es a continuación de donde esta guardando la clave, sigue con la última cifra y llega al RET con la clave toda copiada en 6fef04.

No olvidar poner los BPR o BPMs allí y seguimos... para en

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

Allí toma el valor de la letra D o sea 44 y lo mueve a AL luego lo hace PUSH y lo manda al STACK así que luego de ejecutar el PUSH hago **D ESP** para ver donde lo guardó, **Esp** es **6fee14**, y allí guardo el 44.

Pongo **BPM 6fee14 rw** o **BPR 6fee14 6fee14 rw** (se entiende SOFTICE o TRW2000 ya no lo voy a repetir más). Este proceso va a ser repetitivo para cada cifra.

Para en **42bdd6**

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

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

Luego viene una parte importante

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

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
| :0042BD9C(C), :0042BDA2(C)
```

```
|
:0042BDAB 663D4100    cmp ax, 0041
:0042BDAF 7220        jb 0042BDD1
:0042BDB1 663D4600    cmp ax, 0046
:0042BDB5 771A        ja 0042BDD1
:0042BDB7 05C9FF0000  add eax, 0000FFC9

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

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

El que quiera saltear esta partecita sepa que esta hecha solo para despistar. Bueno, aquí compara el valor si esta entre 0 y 9 primero. Si está entre 0 y 9 está bien sino pasa a la parte de letras y compara. Si está entre A (41) y F (46) está bien sino parece que te tira al tacho.

O sea que solo acepta letras pero que signifiquen números en HEXA porque?. Veamos... luego veo que en eax me quedo la misma letra o sea EAX=D y luego de hacer todo el mismo proceso con la letra E en 42bdcc eax es DE o sea, en letras, las dos ultimas cifras de la clave, EAX=DE.

Entonces por eso no acepta letras mayores que F ya que va a operar directamente con las letras y no puede hacer eso si la letra es G ya que no es un valor hexadecimal que pueda almacenar en algun lado.

Luego en 42be30

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

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

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

Ahora para en

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

donde toma el DE y lo mueve a AL y luego lo copia a [EDI], o sea que ahora DE también está en 6feed6. **BPR** o **BPM** allí.

Ufff sigamos.

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

UN MEDICO PARA EL CRACKER POR FAVOR jua, jua.

LO QUE SIGUE A CONTINUACIÓN ES LA RUTINA DE COMPARACIÓN:

Uff, la verdad es que es una rutinita resimple la que compara las cifras de la clave del SNAGIT pero da tantas vueltas y desorienta tanto que a veces uno se pierde.

Si vieran la cantidad de hojas que llené para llegar al final a una simple comparación jua, jua. Lo que pasa es que con esto de perseguir claves yo cometí un error. Hay que hacer una mirada hasta el final de todos los lugares donde va parando, tipo superficial, y luego si, si uno no encuentra alguna parte como esta que vamos a ver se pone a fondo a buscar la clave.

La rutina de comparación es esta:

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

En **42be99** para ya que allí había un BPR porque en EAX+EDI esta la clave falsa. Allí pasa a AL la primera cifra.

En **42beaa** la compara con la primera cifra verdadera. En el SOFTICE aparece arriba a la derecha; en el TRW2000 hay que hacer

D ebp+ecx-14

En mi caso, en el primer lugar está el **34** que corresponde al **4**. Al no ser igual, ya que mi primera cifra de la clave falsa es uno, ya me va a tirar fuera por lo que borro todos los BP's con BC* y pongo un BPX allí, en la comparación. **BPX 42beaa** y salgo hasta el cartelito de mala clave.

Cambio la primera cifra por la que encontré verdadera y la primera vez no me tira afuera (ya que la cambie, jua jua). La segunda vez que para vuelvo a hacer **d ebp+ecx-14** y me fijo en la segunda cifra. La anoto que en mi caso es **31** o sea **1** y salgo de vuelta, la cambio y vuelvo a entrar. Así hasta que descubro que mi clave es **415387b4**. Esas son las cifras que testea. A partir de allí las siguientes son de relleno hasta tener las 14 cifras, por ejemplo **415387b4ffffffff**.

Esa seria mi clave o podría cambiar las **f** por otra cosa igual me registra ya que solo testea las 8 primeras.

Ahí ven mi nombre de usuario Chancho y mi clave 415387B4.....

LAS SIGUIENTES CIFRAS SOLO ESTAN PARA RELLENAR HASTA EL NÚMERO 14 LA CANTIDAD DE CIFRAS.

Ah! me olvidaba de decir que para que se habilite la opción de ingresar la clave que esta desactivada primero hay que ir a donde se ingresan los datos de nombre de usuario y compañía y recién allí se habilita el botón ENTER KEY para ingresar la clave.

Hasta la lección 49, bay bay.

Ricardo Narvaja

LECCIÓN 49: RELOJ NO MARQUES LAS HORASSSSSS

Este título se refiere a que hay programas que funcionan full durante el periodo de prueba y que uno cuando mira un poquito ve que buscar registrarlos es complicado entonces busca otra alternativa que funciona para mi, y como yo soy cracker pero a veces un poquito perezoso, siempre trato de buscar la más fácil que haga que el programa funcione.

El programa en cuestión es el Microangelo 5 es para editar iconos y todas esas cosas, aquí dice lo que hace.

What do you want to do?

Improve my computer's icon display.
Browse, view, and search for icons.
Change icons and cursors on my computer.
Organize icons in libraries.
Edit icons and cursors.

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

Pero me parece que ya salió allí la versión 5.5. Bueno, en el Kturn subiré esta, la 5.02, aunque no debe haber mucha diferencia en la forma de crackearlo.

En principio cuando realizamos algún cambio en cualquier archivo nos sale un cartelito que dice que hay un problema con el muapp.dll y que debemos instalar de nuevo el programa.

En esta protección, que es un cartelito tipo **messageboxa**, no podemos poner un Bpx messageboxa y una vez que para fijarnos nada pues está hecho para que una vez que sale el messageboxa no vuelve nunca más al programa y lo cierra; por ello no podemos volver al ejecutable.

Para ir de a poco dejemos todo como esta original y cambiemos algún valor que no haga que deje de funcionar, obviamente, en ese muapp.dll. Pueden ser las características de las secciones o cualquier valor que no importe mucho, solo es para probar.

Reemplazamos el muapp.dll por el modificado; corremos el programa y zas, otra vez ese cartel maléfico.

Bueno, habrá que tracear hasta ver cuando aparece. Antes de ejecutar el programa de nuevo uso un **BPX getversion** y ejecuto. Cuando para vuelvo con F12 al ejecutable y comienzo a tracear y tracear y tracear.

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

```
:6400137E FF1578800064  Call dword ptr [64008078]
:64001384 A140A00064    mov eax, dword ptr [6400A040]
:64001389 8B542414      mov edx, dword ptr [esp+14]
:6400138D 3BD0          cmp edx, eax
:6400138F 740F          je 640013A0
:64001391 5F            pop edi
```

```
:64001392 5E            pop esi
:64001393 5D            pop ebp
:64001394 33C0          xor eax, eax
:64001396 5B            pop ebx
:64001397 81C4B8020000  add esp, 000002B8
:6400139D C20C00        ret 000C
```

Yo compare además, con el muapp.dll sin modificar cuando no aparece el cartelito, y allí el programa llega a **6400138d** y salta. En cambio el modificado, llega a ese salto y no salta sigue hasta el RET donde una vez que llega allí va a funciones de windows sale el cartelito y se cierra el programa y nunca más vuelve.

Por lo tanto lo que hay que hacer primero es cambiar ese salto condicional a JMP.

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

La siguiente parte de la protección es cuando realizamos un cambio en alguna de las aplicaciones como Librarian o Muapanel. Nos aparece un cartelito similar al que parcheamos recién pero en otro lugar más adelante.

Aquí hay dos RET y dos saltos.

```
:640015A7 7523          jne 640015CC

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

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:64001569(C), :64001580(U), :64001597(U)
|
:640015AE 8B542414      mov edx, dword ptr [esp+14]
:640015B2 8B0C8530A00064  mov ecx, dword ptr [4*eax+6400A030]
:640015B9 3BD1          cmp edx, ecx
:640015BB 743D          je 640015FA
:640015BD 5F            pop edi
:640015BE 5E            pop esi
:640015BF 5D            pop ebp
:640015C0 33C0          xor eax, eax
:640015C2 5B            pop ebx
:640015C3 81C4B8020000  add esp, 000002B8
:640015C9 C20C00        ret 000C

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

* Possible StringData Ref from Data Obj ->"engineer.exe"
                                |
:640015D0 6814A10064    push 6400A114
:640015D5 50            push eax
:640015D6 FFD6          call esi
```

```
:640015D8 F7D8            neg eax
:640015DA 5F              pop edi
:640015DB 5E              pop esi
:640015DC 1BC0            sbb eax, eax
:640015DE 5D              pop ebp
:640015DF 40              inc eax
:640015E0 5B              pop ebx
:640015E1 81C4B8020000    add esp, 000002B8
:640015E7 C20C00          ret 000C
```

Bueno, aquí es igual, en cualquiera de los dos RET que cae aparece el cartelito. Así que, entonces, en el primer salto no debe saltar o sea hay que nopearlo entonces cae en el segundo que si salta y esquiva los dos RET.

Entonces nopeamos el primero y hacemos JMP el segundo y adiós protección, podemos modificar lo que querramos que funciona igual.

640015a7 75 23 cambiar por **90 90**

y en

640015bb 74 3d cambiar por **eb 3d**

Ahora, como ya dijimos, vamos al funcionamiento. Miremos un poco el LIBRARIAN que es una de las utilidades que trae.

Sabemos que las funciones para que el programa se entere de la fecha son GETLOCALTIME o GETSYSTEMTIME.

Si cargamos el Librarian en el Wdasm y vamos a **IMP FN**, el botón que muestra las funciones importadas, vemos que GETLOCALTIME y GETSYSTEMTIME solo una vez dentro del programa son llamadas en:

```
* Reference To: KERNEL32.GetLocalTime, Ord:012Fh
|
:004157AA FF15ACB14100  Call dword ptr [0041B1AC]
:004157B0 8D45E0        lea eax, dword ptr [ebp-20]
:004157B3 50            push eax

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

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

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

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

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

La fecha puede variar según en que día estemos jua jua. Puedo probar que pasa si allí mismo edito la fecha para cambiarla haciendo **E 65f950** y cambiando los valores.

Si hago pruebas veo que la que varía los días de evaluación es la que halla BPX GETLOCALTIME o sea la de abajo, la de **65f960.**

Si cambio **d1** por **d2,** o sea al año 2002, me dice que expiró y que va como un año de evaluación; lo mismo con los días y el mes.

Pongamos un bpr allí a ver cuando el programa lee esa fecha que esta en la memoria pero aun no la leyó.

Bpr 65f960 65f966 r

Hago X y ENTER y para en...

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

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

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

Quedó así para mi...

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

En la primera muevo a AL el valor 0E, o sea el día 14 y nopeo dos veces ya que la sentencia es más corta.

En **41585c** muevo a AL el valor 0A del mes 10 y en la tercera tengo que hacerlo en dos partes, muevo a AL D1 y a AH 07; como resultado en EAX queda 07d1 que es el Año 2001. Con esto congelo la fecha en el primer día de evaluación mío 14/10/2001.

Lo mismo existe en las otras aplicaciones.

Ahora, en Muapanel es similar. Bpx getlocaltime es la que maneja todo en muapanel también. Getlocaltime esta en **40a082** y guarda la fecha en **5bfaec**.

Al salir, si ponemos un **BPR 5bfaec 5bfaef r** para en **40a12f** que es la parte similar a la que parcheamos en librarian.

Aquí ya esta modificada...

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

Recuerden de poner una fecha para la cual el programa para ustedes funcione. No copien la misma que la mía ya que así no funcionara.

Fíjense en la fecha en que lo instalan y pongan esa fecha y siempre el programa estará en el primer día de evaluación.

Las otras aplicaciones que tiene son similares y se las dejo para que practiquen.

Hasta la lección 50
Ricardo Narvaja

LECCIÓN 50: CRACKEANDO CIEGO

El programa que vamos a crackear hoy es el **Xguest Inviter Bot 1.72.** la versión no es la última, creo que va por la 1.82 o 83 no se, pero sirve para practicar y es un programa para conectarse a IRC.

Aquí la novedad es que a pesar de conseguir el serial valido que nos registra, cuando nos conectamos a INTERNET para usar el programa, verifica que no es un usuario que pago el REGISTRO y sale un cartel diciendo que infringiste la licencia y te cierra el programa.

Les diré que tiene sus bemoles crackear este programa ya que está muy protegido. Inclusive amigos que han instalado versiones posteriores a esta y después han querido usar esta más vieja, no te deja, dice que tenés que usar la ultima versión o algo así.

Como a mi eso no me ocurrió ya que no instale la última, solo esta, no puedo saber como saltear eso, pero no debe ser difícil ya que son simples messageboxa.

Ya saben que el programa va a estar en el Kturn y OOdrive para bajarse, para crackear y practicar.

Otro tema son las consultas que generalmente me hacen a mi mail ricnar22@millic.com.ar. Como yo me voy a mudar ese mail no se si va a seguir funcionando ya que no se que servicio de internet tendré en el otro departamento, así que hasta que me mude, pueden escribirme a ricnar456@data54.com que es WEBMAIL y va a seguir funcionando vaya a donde vaya. Disculpen si demoro en las respuestas estos días, pues estaré de mudanza y sin internet, aunque en mi trabajo si tengo y leeré allí mis mails.

Sigamos con este tema. El ejecutable del Xguest esta compactado con PECOMPACT, me dice el Unpack, y dice que lo puede descomprimir; lo hago y tengo el ejecutable descomprimido que, oh sorpresa, salen las Funciones Importadas pero no las STRINGS REFERENCES así que crackearemos ciegos.

Además está hecho en DELPHI, como información, aunque el DEDE tampoco dice mucho.

Como primer paso hay que hallar una clave para registrarnos valida. Utilizando el método de HMEMCPY y poner BPRs en donde aparece la clave falsa. Siguiendo los lugares donde el programa lee la clave, veo que después de un **SCASB** que para el SOFTICE, **como esa sentencia es generalmente parte de lsrtlen que mide el largo de mi clave falsa 98989898**, cuando vuelve al ejecutable sale siendo EAX=8 o la cantidad de cifras que tiene la clave falsa que ustedes colocaron.

Siguiendo ese EAX=8, si lo guarda con PUSH EAX viendo con **D ESP** donde lo guardo y siguiendo con los BPRs vemos que para en...

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

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

En mi caso la clave para registrase fue:

User: narvaja
Password: 4B899FE

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

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

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

```
:0044CFFD 50          push eax

* Reference To: user32.MessageBoxA, Ord:0000h
|
:0044CFFE E8DDA7FBFF  Call 004077E0
```

En el Wdasm puedo mirar hacia arriba para ver si hay algunos saltos que esquiven este messageboxa.

Los saltos que están un poco más arriba no sirven ya que no esquivan el messageboxa. Sigo hacia arriba hasta que llego al REFERENCE este.

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

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

```
:004CAD45 757A  jne 004CADC1
```

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

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

004CDEE4, :004CE1F6

En estos lugares paró en SOFTICE pues había puesto BPXs en todos esos valores. Cada uno tiene un salto un poco más arriba que hay que cambiar por JMP.

Cuando para en **4cdee4** un poco más arriba, en **4cde8d**, también hay un salto para parchear.

:004CDE8D 757C jne 004CDF0B

Aquí también hay que cambiar por **jmp 4CDF0B.**

Cuando para en **4CE1F6** HAY UN SALTO MÁS ARRIBA EN

:004CE19F 747F je 004CE220

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

Con parchear estos saltos ya me pude conectar sin problemas e ingresar al IRC perfectamente y recibir y enviar bien.

De cualquier manera, viendo que todavía quedan los otros CALLs aunque a mi no me paro allí, el método si llega a caer dentro es el mismo; poner un BPX allí y parchear el salto que esquiva ese CALL, o también preventivamente se puede mirar en el WDASM y parchear el salto que se encuentre antes de todos esos CALLS y que eviten que caiga en ellos.

Hasta la LECCIÓN 51 que creo que ya va a ser en mi nuevo domicilio.

Ricardo Narvaja

LECCIÓN 51: DIFÍCIL PERO ARREGLÁNDONOS

De aquí se baja la victima de hoy se llama **ILLUMINATUS OPUS 2001.** La versión común, no la PRO, es una versión de evaluación. Tiene ciertas limitaciones ya que no se puede registrar, pero bueno, practicaremos con el a ver que podemos hacer. Esto dicen sobre el programa en la página web http://www.digitalworkshop.com/downloads/evaluation.shtml

Multimedia program that combines video, audio, animation, words and pictures into stunning interactive publications.

- ***EXPIRE after 30 days.ANY PUBLICATIONS CREATED WITH THE EVALUATION VERSION WILL ALSO CEASE TO FUNCTION WHEN YOUR EVALUATION PERIOD ENDS.***

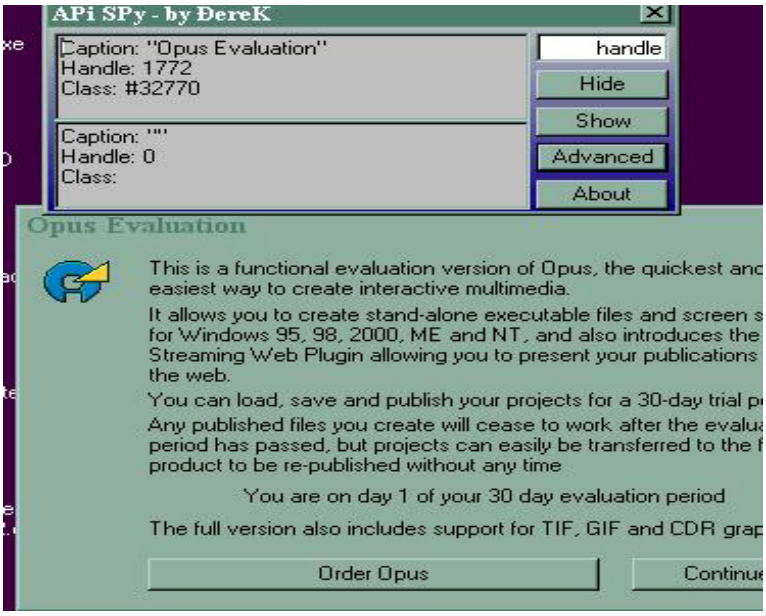
- ***These publications will then only be recoverable with a full version of the program. No support GIF, TIFF or CDR files.***

Bueno, aquí dice que el programa expira a los 30 días y las publicaciones que hagas también. Trataremos de que ambas sigan funcionando.

El programa tiene un ejecutable chiquito llamado OPUS.exe que está comprimido con aspack. Con el **Caspr** se puede descomprimir y reemplazar pero no afecta en nada ya que el programa no realiza nada importante desde el ejecutable.

El problema es que la ventana que dice los días que van de la evaluación no es ni messageboxa ni dialogboxparama ni para con drawtexta ni con nada conocido. Usaremos otra técnica para ver donde se cierra esta ventana desde el programa principal.

Arrancamos el programa y una vez que aparece la nag esa donde dice los días que nos quedan, arrancamos algún programa analizador de ventanas como el **API-SPY** o **WINDOWSE,** que lo utilizamos para que nos diga el HANDLE de la ventana.



Como se ve en la imagen, la ventana del API-SPY me da la información de la ventana sobre donde coloco el cursor, así que lo coloco sobre la ventana a estudiar.

La parte principal es la que dice el texto **OPUS EVALUATION** y allí vemos que el HANDLE en mi caso es 1172.

Es importante que el cursor este bien sobre el texto principal ya que si lo pongo en los botones salen otros resultados.

Bueno, lo que importa es ver el handle de la ventana, para mí 1172. Para cada uno variará el valor y cada vez que ejecute el programa será distinto este valor. El APY-SPY te da el handle en decimal el WINDOWSE en hexa, para verificar, con la calculadora pasamos a hexa el valor 1172 y es 06ec.

Si con la ventana todavía activa entramos al softice con CTRL+D y allí tecleamos **HWND** nos sale un listado de la información de todos los handles que están en ese momento.

Si con ENTER voy bajando hasta que en la columna OWNER comiencen a aparecer los del proceso OPUS vamos a encontrar allí lo mismo que nos dijo el API-SPY, el handle en hexa para mi de esta ventana es **06ec.**

Bueno, la ventana todavía esta activa. Existe un breakpoint para que caiga en el SOFTICE cuando se cierre la ventana...

BMSG handle Wm_Destroy

En mi caso

BMSG 06ec Wm_Destroy

Con CTRL+D vuelvo a WINDOWS y al hacer click en el BOTON CONTINUE y al desaparecer la ventana caemos en el SOFTICE.

Hay que volver con F12 varias veces hasta que llegamos en este caso al DLL **ILMEDITMAIN**, que aparece en la línea verde, ya que el ejecutable OPUS aquí casi no hace nada, todo lo hacen las DLL.

Caemos en **BDB932 Mov eax, [ebp-00dc]** y justo en la sentencia anterior hay un **Call EAX** que seguro es la ejecución del cartel entero. Para probar eso pongamos un **BPX bdb930** y arranquemos el programa de nuevo. Vemos que justo para en el BPX y no salió aun la ventana molesta (podemos mirar con F4). Si hacemos F10 y ejecutamos el CALL aparece la ventana y al continuar caemos en la sentencia posterior al CALL o sea en **BDB932.**

Si ponemos un BPX un poquito antes, en BDB8E8, cuando para allí vamos traceando con F10, y ya que en la ventana aparece la cantidad de días que lleva de evaluación, por aquí debe cargar ese valor y eso ocurre en **BdB8E8.** Justo a EAX va la cantidad de días de evaluación menos uno, ya que unas sentencias más adelante al hacer INC EAX queda justo en EAX la cantidad de días transcurridos.

Si modificamos en **BDB8E8** para que reemplacemos la sentencia original por **XOR EAX, EAX** (que pone EAX a cero), luego al incrementar EAX queda

siempre el valor de los días de evaluación en uno. A pesar de adelantar el reloj, siempre dice que va un solo día de evaluación.

Este es el primer paso. El segundo es hacer una publicación y ver que pasa.

Si la cargo con el programa en la fecha actual, la abre sin problemas, pero si adelantamos el reloj nos dice que la publicación ha expirado y la cierra.

Ya que este cartel es un BPX messageboxa ponemos allí un breakpoint y, cuando adelantamos el reloj, para en ese break. Hacemos F12, aceptamos la ventana y volvemos a caer en el SOFTICE. Allí nos damos cuenta, cuando volvemos al ILMEDITMAIN, que arriba del lugar de donde caemos hay un salto que esquiva el cartel, y esta en **C08AC9 JAE C08B53**.

Si reemplazamos ese JAE por un JMP no aparecerá el cartel que cierra las publicaciones cuando vencen.

Ahora el problema es que cuando copiamos las cadenas para modificar el ILMEDITMAIN.dll, estas no se encuentran con el ultraedit.

PARA EL REEMPLAZO POR **XOR EAX,EAX** EN **BDB8E8** LA CADENA ES:

8B8520FFFFFFBB8051010033D2

Y HAY QUE REEMPLAZAR LAS NEGRITAS POR

33C090909090

PARA EL SALTO QUE HAY QUE REEMPLAZAR EN **C08AC9** LA CADENA ES:

0F83840000008D4DC0

Y HAY QUE REEMPLAZAR LAS NEGRITAS POR

E98500000090

El problema es que cuando buscamos estas cadenas en el ULTRAEDIT no aparecen, por lo que el dll debe estar comprimido. Si lo vemos con el PEEDITOR vemos que en los nombres de las secciones una de ellas se llama ASPACK. Así que, con el CASPR, descompactamos el dll. Ahora si, buscamos las cadenas que aparecen, realizamos los cambios y funcionaaaaa.

Con eso el programa sigue funcionando sin vencer y las publicaciones también.

Hasta la 52
Ricardo Narvaja

LECCIÓN 52: OTRA VEZ P-CODE PERO SIN EL DEBUGGER

Hoy crackearemos un programa que tiene una parte en P-CODE y una parte normal de VISUAL BASIC.

El programa se llama **DIGITAL PEN 1.0** y se baja de

http://camtech2000.net/

Es un programita resencillo para hacer STATIONERYS o sea, animaciones para el OUTLOOK EXPRESS.

Lo primero que dice el tipo cuando funciona es que vence en 10 usos o 10 días, cualquiera de las dos cosas.

Bueno, podemos analizar con el TECHFACTS 98 a ver cada vez que lo usamos que cambios hace cada vez que se ejecuta.

Arrancamos el TECHTACTS 98 y vamos a la pestaña TOOLS y allí a WATCH SYSTEM y donde dice RUN THIS PROGRAM buscamos el ejecutable del DIGITAL PEN y lo arrancamos desde abajo, donde dice GO.

Luego de que hace un escaneo de todo el sistema arranca el programa, se ejecuta, lo abrimos para que consuma una vez y luego lo cerramos y el TECHFACTS hace otro escaneo para ver que cambios hizo el programa y aquí esta el resultado:

```
Registry key values changed: (12)
HKEY_CLASSES_ROOT\NSPlay32s
Value "GamesAvail": from "4" to "5"
HKEY_USERS\.DEFAULT\Software\Microsoft\HTMLUrlls
Value "Visited": from "4" to "5"
HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Explorer\StreamMRU
Value "MRUListEx": binary data changed
HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Internet
Settings\Connections
Value "SavedLegacySettings": binary data changed
HKEY_LOCAL_MACHINE\Software\CLASSES\NSPlay32s
Value "GamesAvail": from "4" to "5"
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Msversions
Value "WinKeys": from "4" to "5"
```

Bueno, estas claves son donde guarda la cantidad de veces que va usando, si ahora exportamos estas claves y cada vez que usamos el programa ejecutamos los archivitos del registro para que vuelva la cuenta de la cantidad de veces usadas para atrás, con eso lo mantenemos funcionando mientras lo crackeamos.

También hay una clave del registro, esta la vi con el REGMON, y es cuando el programa expira. Cambia este 1 y no funciona más.

```
[HKEY_CURRENT_USER\Software\VB and VBA Program Settings\Digital Pen\Logged]
"Expired"="1"
```

Aquí también conviene exportar esta clave por si nos vence volverla a inyectar en el registro para hacer que funcione nuevamente.

Bueno, si probamos tratar de verlo con el P-CODE debugger con que crackeamos el CUENTAPASOS, vemos que arranca el programa pero no para en el debugger y se cierra sin poder hacer nada, por lo que no lo podemos utilizar.

Si descomprimimos el programa con el WDASM de VISUAL BASIC vemos que salen las STRING REFERENCES bien.

El SOFTICE se me cuelga cuando quiero usarlo con este programa así que usando el TRW2000. Veremos que podemos hacer.

```
:00449CD2 8B7508      mov esi, dword ptr [ebp+08]

* Possible StringData Ref from...->"This program will stop functioning "
                                  ->"after 10 uses or 10 days."
|
:00449CD5 6824DA4000  push 0040DA24
```

Aquí vemos una parte del WDASM que aparece en el cartel molesto cuando arranca y dice que la versión registrada no muestra este cartelito así que trataremos de esquivarlo.

Un poco más arriba hay un salto pero, si paro en el TRW2000 y lo invierto, da error.

Más arriba en el WDASM esta este otro cartel

```
* Possible StringData Ref from Code Obj ->"Expired"
```

O sea, que por aquí no hay que pasar si estas registrado. Sigo más arriba en el WDASM y siguen los carteles que no deben aparecer en la versión registrada.

```
:00449AFD 8B3D50104000  mov edi, dword ptr [00401050]

* Possible StringData Ref from...->"This program has timed out and "
                                  ->"will not function anymore."
```

Más arriba

```
* Possible StringData Ref from Code Obj ->"Program Timed Out"
```

O sea, que por aquí no hay que andar. Siguiendo para arriba vemos que vuelven a salir mensajes de TIME OUT y EXPIRED y también los nombres de las claves del registro que guardan la cantidad de veces que se usaron, que en la versión registrada no debe utilizar. Sigamos más arriba.

```
* Possible StringData Ref from Code Obj ->"GamesAvail"
|
:004499FC BAB8D64000  mov edx, 0040D6B8
:00449A01 8D4DE0      lea ecx, dword ptr [ebp-20]
:00449A04 FFD6        call esi

* Possible StringData Ref from Code Obj ->"HKEY_CLASSES_ROOT\NSPlay32s"
```

NSPlay32s es una clave del registro donde guarda la cantidad de veces. Seguimos más arriba y siguen apareciendo claves del registro de la versión de prueba y más EXPIRED, etc.

La conclusión es que toda esta rutina debe ser para la versión NO REGISTRADA así que vamos a seguir bastante hacia arriba a ver donde comienza.

La rutina comienza en

```
:00448A50 55      push ebp
:00448A51 8BEC    mov ebp, esp
:00448A53 83EC0C  sub esp, 0000000C
```

Ya que hay una REFERENCE que dice que el programa viene de un salto en **4056a6** y más arriba de esto hay un RET que parece ser otra cosa veamos en 4056a6 que hay.

Aquí hay una zona con una tablita de saltos

```
:0040568C E90F310400              jmp 004487A0
:00405691 816C240463000000        sub dword ptr [esp+04], 00000063
:00405699 E952320400              jmp 004488F0
:0040569E 816C240433000000        sub dword ptr [esp+04], 00000033
:004056A6 E9A5330400              jmp 00448A50
:004056AB 816C240433000000        sub dword ptr [esp+04], 00000033
:004056B3 E958470400              jmp 00449E10
:004056B8 816C24045F000000        sub dword ptr [esp+04], 0000005F
:004056C0 E9AB480400              jmp 00449F70
:004056C5 816C240437000000        sub dword ptr [esp+04], 00000037
:004056CD E94E490400              jmp 0044A020
:004056D2 816C24045B000000        sub dword ptr [esp+04], 0000005B
:004056DA E9F1490400              jmp 0044A0D0
```

Si cargamos el programa con el TRW2000 vemos que traceando con T, enseguida entramos en el DLL de VISUALBASIC MSVBVM60.dll, que es la parte de P-CODE, y no podemos volver al ejecutable por más que hagamos F12. Quizás porque la parte de P-CODE termina en un JMP donde no paramos con F12.

Posiblemente esa tablita de JMPs sea donde el programa salta de partes de P-CODE a VISUAL COMUN. Podemos poner en todos esos JMPs unos breakpoints a ver que pasa.

Arrancamos el TRW2000 y ponemos BPX en todos los JMPs de esa listita. La primera vez que para lo hace en **4056a6** justo el salto que va a la parte de no registrado. Seguimos ejecutando el programa y sale el cartel maldito y luego vuelve a parar en **40568c** y luego arranca el programa.

Ahora, ¿que pasaría si en **4056a6**, que es cuando va a ir a la parte de no registrado, cambiamos el salto al de arranque del programa que esta en **40568c**?. Saltearía la parte de no registrado y arrancaría directo en teoría, probemos.

Pongo un **BPX 4056a6**, donde esta el **JMP 448a50** y cuando para allí hago A y ENTER y escribo el otro salto, **JMP 40568c**, para que arranque el

programa directo. Le doy a X y FUNCIONNNNAAA y no aparece la maldita ventana de desregistrado.

El cambio con el ULTRAEDIT es

E9A5330400 816C240433000000

CAMBIAR POR

E9F5300400 816C240433000000

Y listo, no vence más, ni lee las claves donde guarda la cantidad de veces, ni las modifica nunca más lo que se puede ver con el regmon o el techfacts.

HASTA LA LECCIÓN 53
RICARDO NARVAJA

LECCIÓN 53: UNA MOCHILA PESADA

Hoy vamos a tratar un programa bastante completo llamado **Cimagrafi 6.02** que se puede bajar de mi FTP (pedir dirección del mismo a ricnar22@millic.com.ar) o también lo voy a subir a mi kturn.

Allí hay una carpeta que hay que descomprimir y NO INSTALAR a partir del SETUP que hay allí. ¿Porque digo que no usen el SETUP?; bueno, el programa es de esos que funcionan con una mochila o aparatito que se conecta a un puerto de la computadora. Por supuesto yo no tuve el gusto de conocer dicho aparato, pero me llego el programa y la idea es tratar de hacerlo funcionar sin poseer ese chiche.

La mochila que usa este programa es la llamada mochila Hasp y hay varios tratados de gente que sabe de cómo emularla o como atacarla.

Nosotros no vamos a hacer eso, quizás porque este humilde cracker no esta a la altura de semejante desafío, solo trataremos de hacer funcionar las aplicaciones que tiene allí sin la dichosa mochila y punto.

Dentro de esa carpeta una vez descomprimida están los ejecutables de las aplicaciones que vamos a tratar de hacer funcionar. GRAFICAD.exe es una y la primera que vamos a estudiar; en las otras, P2P.exe, TRACE.exe y MIL.exe, la protección es similar e inclusive la cadena encontrada es la misma en todas estas aplicaciones y se puede reemplazar igual en todas.

Bueno, si alguno cometió el error de ejecutar SETUP.exe a pesar de que se lo advertí lo que va a hacer es instalar el HASP driver que si detecta la presencia del Softice no solo te impide que arranque sino que te renombra ficheros del mismo para que se cuelgue y funcione mal.

Bueno, puede eliminarlo usando el editor del registro y borrando la clave

[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\HASP95]

Con eso ya eliminamos el driver molesto ese.

Así que, sin instalar el programa, veremos que podemos hacer y si podemos hacer que arranque.

Tomamos el archivo GRAFICAD.exe que va a ser nuestro objetivo, lo ejecutamos y zas PLUG NOT FOUND, ya nos aparece un cartelito que nos cierra el programa.

Bueno, puedo poner un **BPX messageboxa** y para allí pero a pesar de que vuelvo al ejecutable este ya se cierra y no obtengo mucha información de esto.

Si voy traceando hasta que aparece la ventana de PLUG NOT FOUND y llego a un CALL que al ejecutarlo aparece el cartel, y voy anotando ese CALL y luego pongo un BPX allí y paro en el SOFTICE y entro con T y sigo traceando con F10 y así sucesivamente voy a llegar a un momento en que este en este sector:

```
:00490C04 E8332CF7FF  call 0040383C
:00490C09 A1D4485700  mov eax, dword ptr [005748D4]
```

Al ejecutar ese **CALL 40383C** sale el cartel maldito y se cierra el programa, no pasando ya por la sentencia subsiguiente **490c09** lo que puede comprobarse poniendo un BPX 490c09 y viendo que no para allí y se cierra el programa igual.

Bueno, un buen primer paso seria analizar dentro de ese **CALL 40383c.**

```
:0040383C 5A          pop edx
:0040383D 54          push esp
:0040383E 55          push ebp
:0040383F 57          push edi
:00403840 56          push esi
:00403841 53          push ebx
:00403842 50          push eax
:00403843 52          push edx
:00403844 54          push esp
:00403845 6A07        push 00000007
:00403847 6A01        push 00000001
:00403849 68DEFAED0E  push 0EEDFADE
:0040384E 52          push edx

* Reference To: kernel32.RaiseException, Ord:0000h
|
:0040384F E91CDAFFFF  Jmp 00401270
:00403854 C3          ret
```

Bueno no vemos mucho aquí solo que pone algunos valores en los registros y luego hace un JMP a Raiseexception que no se exactamente que hace pero no me gusta nada. Probemos que pasa si en 40383c ponemos directamente un RET y evitamos que siga por este camino que nos lleva al cartel maldito y a cerrarse el programa.

Hagamos **A 40383c ENTER** y escribamos **RET** a ver que pasa. Luego salgamos con Escape y sigamos ejecutando el programa con X.

ARRANCAAAA!!!... No se si en forma perfecta pero sale el cartel de Cimagrafi y la pantalla principal y las herramientas para dibujar y al parecer funcionan bien. Si hay algún error se arregla individualmente pero yo no encontré ninguno aun.

Bueno, la cadena que debemos buscar en el ULTRAEDIT para reemplazar es

5A 54 55 57 56 53 50 52 54 6A 07 6A 01 68 DE

Y ALLI SOLO REEMPLAZAR EL **54** POR **C3** QUE CORRESPONDE A LA SENTENCIA **RET.**

Lo más divertido es que la misma cadena la podemos encontrar en los otros ejecutables y reemplazar el 54 por el c3 y hacer funcionar todos.

Ojo, si la buscamos en el Wdasm la posición de memoria en uno y otro programa no es la misma, pero si buscamos en el ULTRAEDIT la cadena esta aparecerá y solo modificando el 54 por c3 arrancaran perfectamente sea el TRACE, el P2P, el MIL, el Graficad, etc.

Bueno, ahora ya que arranco podemos registrarlo si hacemos clic en ABOUT vemos que dice **DEMO** que esta entre las STRINGS REFERENCES del Wdasm para DELPHI ya que este programa esta hecho en DELPHI.

Esta parte es la del mensaje de DEMO

```
:00503DCC E85BD7F8FF    call 0049152C
:00503DD1 85C0          test eax, eax
:00503DD3 7C4B          jl 00503E20
:00503DD5 8D55D8        lea edx, dword ptr [ebp-28]
:00503DD8 8BB3D0020000  mov esi, dword ptr [ebx+000002D0]
:00503DDE 8BC6          mov eax, esi
:00503DE0 E8FBF5F2FF    call 004333E0
:00503DE5 FF75D8        push [ebp-28]

* Possible StringData Ref from Code Obj ->" DEMO (More "
```

Ahí vemos el cartel de DEMO y un poco antes un **CALL 49152c**, que según el valor de EAX que sale aparece el cartel de DEMO o no.

Si entramos dentro del CALL 49152c

```
:0049152C E873FEFFFF      call 004913A4
:00491531 0FBE05C3485700  movsx eax, byte ptr [005748C3]
:00491538 C3              ret
```

Vemos que entra dentro de otro CALL 4913a4; veamos allí dentro.

Bueno, allí empieza una larga parte que es donde decide si esta registrado o no.

```
:004913A4 53          push ebx
:004913A5 BBAC485700  mov ebx, 005748AC
:004913AA C64310 00   mov [ebx+10], 00
:004913AE C64311 00   mov [ebx+11], 00
:004913B2 C64312 00   mov [ebx+12], 00
:004913B6 C64313 00   mov [ebx+13], 00
:004913BA C64314 00   mov [ebx+14], 00
:004913BE C64315 00   mov [ebx+15], 00
:004913C2 C64316 00   mov [ebx+16], 00
:004913C6 C64317 00   mov [ebx+17], 00
:004913CA C64318 00   mov [ebx+18], 00
:004913CE C64319 00   mov [ebx+19], 00
:004913D2 C6431A 00   mov [ebx+1A], 00
:004913D6 C6431B 00   mov [ebx+1B], 00
:004913DA 8B5008      mov edx, dword ptr [eax+08]
```

Bueno, sigue y es mucho más larga pero esos son los valores que siempre testea más adelante para ver si está registrado o no, me refiero a EBX+10, EBX+11 y asi sucesivamente.

Probé a cambiar estos valores por 01 y no salio registrado; probé luego a cambiarlos por FF uno por uno y desapareció el CARTEL DEMO.

O sea, que deberíamos buscar esta cadena en el ULTRAEDIT

C6431000

C6431100
C6431200
C6431300
...
...
...
hasta C6431B00

y así sucesivamente y cambiar los 00 por FF y listo, queda registrado.

Quedaria

C64310FF
C64311FF
C64312FF
C64313FF

(seguir cambiando los 00 por FF hasta C6431BFF)

Y listo, el programa funciona bastante bien. Quizás halla que corregir algún error suelto que aparezca pero ya el hecho de que funcione sin la mochila y lo podamos registrar es un gran avance. Luego los retoques se pueden ir hacienda a medida que se van encontrando.

Hasta la 54
Ricardo Narvaja

LECCIÓN 54: ROBADA DE MI AMIGO SILVER STORM Y DEL MISMO MARMOTA (ESPERO QUE ME PERDONEN JAJA)

Este tute esta basado en uno de mi amigo SILVER STORM y otro de MARMOTA que lo hizo ya que quería demostrar las utilidades del nuevo gestor de ventanas de Marmota WINGESTOR que esta en mi FTP para bajar y que mejor que estos tutes.

El primero es el ya famoso ULTRAEDIT. Como hacerlo funcionar sin cambiar ni una coma (con el WINGESTOR). Ojo, esto hay programas donde se puede hacer y otros que no pues estamos aprovechando un fallo en la implementación del programa ULTRA EDIT que, obviamente, no todos los programas tienen. Pero es muy lindo experimentar un poco y este tute basado de los de ambos es para hacer conocida la herramienta y más adelante ya haré uno mío propio.

El ULTRA EDIT lo conocemos casi todos ya que lo usamos y lo tenemos registrado. Pero así como viene sin registrar adelantamos el reloj un par de meses para vencerlo y aparece una ventana para registrarnos y poner el nombre y el número de registro, y si no ponemos el correcto y cancelamos se cierra el programa y chau.

Bueno, arrancamos el WINGESTOR y el ULTRAEDIT y una vez que aparece la ventana para registrarnos no tocamos nada lo dejamos ahí.

Vamos al WINGESTOR y en la lista aparecen muchas ventanas activas. Enseguida por el texto ubicamos la ventana de registro del ULTRAEDIT, buscamos una ventana con la clase **#32770** y el texto (Autorización), y hacemos clic en la línea que encontramos en el WINGESTOR. Entre las opciones que aparecen elegimos OCULTAR y listo desapareció. Ahora nos queda la ventana principal que no recibe nada del teclado, esta como muerta; a esa hay que habilitarla.

Vamos otra vez al WINGESTOR y por el texto que tiene (UltraEdit - 32 - [Editar1]) encontramos la ventana principal y hacemos clic y elegimos habilitar y listo, funciona a pesar de estar vencido.

Con el WINGESTOR nos podemos divertir haciendo clic en muchas opciones y ocultando cosas y activando otras que están inactivas.

Este es el otro caso, el del tute de Silver Storm

Ftp://ftp.novastor.com/pub/demos/BACKUP/WIN95/nsw9end.exe

Aquí copio del mismo SILVER STORM (perdón amigo)

> "Hola. Este tute es facilito. Es convertir un programa en su propio verdugo.
>
> Uno más con esta herramienta de marmota :)
>
> Ftp://ftp.novastor.com/pub/demos/BACKUP/WIN95/nsw9end.exe
>
> Se trata de **Nova backup** la ultima versión :)
>
> Bajamos el programa, lo instalamos, entramos al programa... luego about... comprar... y entramos donde nos dice **convertir la**

demostración. Ahí nos pide un número de tarjeta... ponemos el número que queramos de 10 dígitos, por ejemplo 1234567890.

En donde nos pide código lo dejamos en blanco. Si observamos el botón de OK está desactivado. Abrimos el gestor de ventanas y nos vamos a la ventana que nos interesa (verificación de número de acceso) la abrimos y nos paramos sobre el OK. Damos dos click y le damos a **Habilitar**.

Jejeje, que genial, se activa el botón de Ok. Lo presionamos y nos salen unas pantallas de agradecimientos, etc. Le decimos que OK a todo y retorna a la ventana de entrada de datos. Entramos de nuevo donde dice "convertir la demostración" y jejeje ¿que les parece?, aparece el código ya puesto y, lo mejor del caso, nos lo muestra el programa original sin ninguna modificación. Se convirtió en un keygen, jeje, a mi me dio 579138HZKP.
Gracias.
Marmota and Jack. Good Job ;)”

Espero que por ser ladrón no me metan preso jaja.

Los invito con el WINGESTOR a probar y, aunque no todo se pueda registrar tan fácil como estos dos ejemplos, hay botones deshabilitados, barras que activar y mucha diversión por delante; prueben y verán.

Ahh, me olvidaba, si la ventana tiene asteriscos al hacer clic con el WINGESTOR aparece el significado de esos asteriscos y además podemos cambiar el texto de las ventanas en la memoria.

Bastante lindo para divertirse

Hasta la 55
Ricardo Narvaja (preso)

LECCIÓN 55: FLASH FXP POR MR GANDALF (GRACIAS)

INTRODUCCION POR RICARDO NARVAJA: Dado que me he tomado el mes de enero de vacaciones después de 54 lecciónes duras, y que a Mr. Gandalf le pedí si podía reemplazarme durante el mes de enero haciendo algún tute o los que pudiera, bueno, aquí esta el tute hecho por él, bien prolijo (eso si que no lo aprendió de mi jaja), además, que siempre cuando quiera colaborar con algún tute suyo, o de quien siga el curso y quiera colaborar, bienvenido sea.

Programa: FlashFXP Version 1.3 Build 770

Download: www.FlashFXP.com

Protección: Serial. Trial de 30 días. Nag screen.

Dificultad: Sí lo he hecho yo...

Herramientas: SoftICE 4.0 - Languaje 2000 - W32Dasm 8.93 - Dede 2.50 - Hiew 6.76 - Advanced Registry Tracer

Cracker: Mr Gandalf.

Fecha: 30/12/2001

Introducción: Se trata de un conocido programa para FTP`s, que en la versión traducida por Pedator, es el que yo uso para chupar del disco duro de Ricardo, entre otros. Inicialmente me gustó más que el CuteFTP y, aunque este tenía su crack en Astalavista, empecé a usarlo.

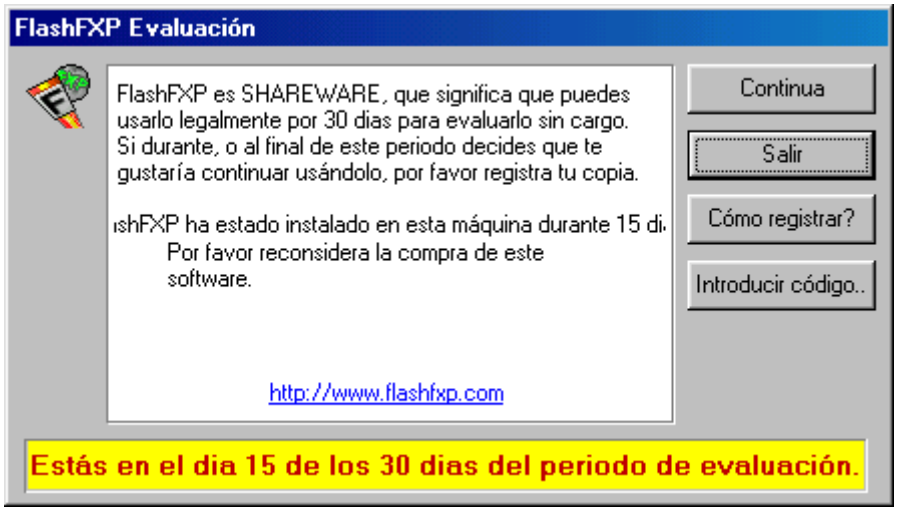
Conforme avanzaban los días se iba consumiendo el trial, así que me decidí a intentar el asalto.

Primero conseguí parar el conteo basándome en un tutorial de Karpoff sobre el MemTurbo 1.5. Así que lo estuve utilizando durante mucho tiempo de esta manera, aunque al inicio del programa salía una molesta ventana recordándome que estaba en el día 0 (bendita inocencia).

Finalmente deje el asunto aparcado en mi carpeta de cracks hasta que Ricardo Narvaja me instó a escribir un tutorial para llenar un mes de enero que se prometía silencioso. Volví sobre el tema de la nag y esta vez todo pareció mucho más fácil.

Desguaze: De lo primero que nos percatamos al iniciar el programa es de esa molesta nag que nos aparece diciéndonos los días que han pasado desde la instalación y que son 30 los que tenemos para evaluarlo. Esta nag tiene 4 botones y uno de ellos es para introducir un serial que parece será largo por que abre una gran caja de dialogo y por que ofrece la posibilidad de traerlo desde el portapapeles.

Si pulsamos continuar entramos en la versión demo que es completamente funcional y que también tiene un apartadito desde donde podemos intentar introducir el serial, solo que esta vez si nos equivocamos ya no nos dejará más oportunidades de intentarlo durante esa misma sesión.

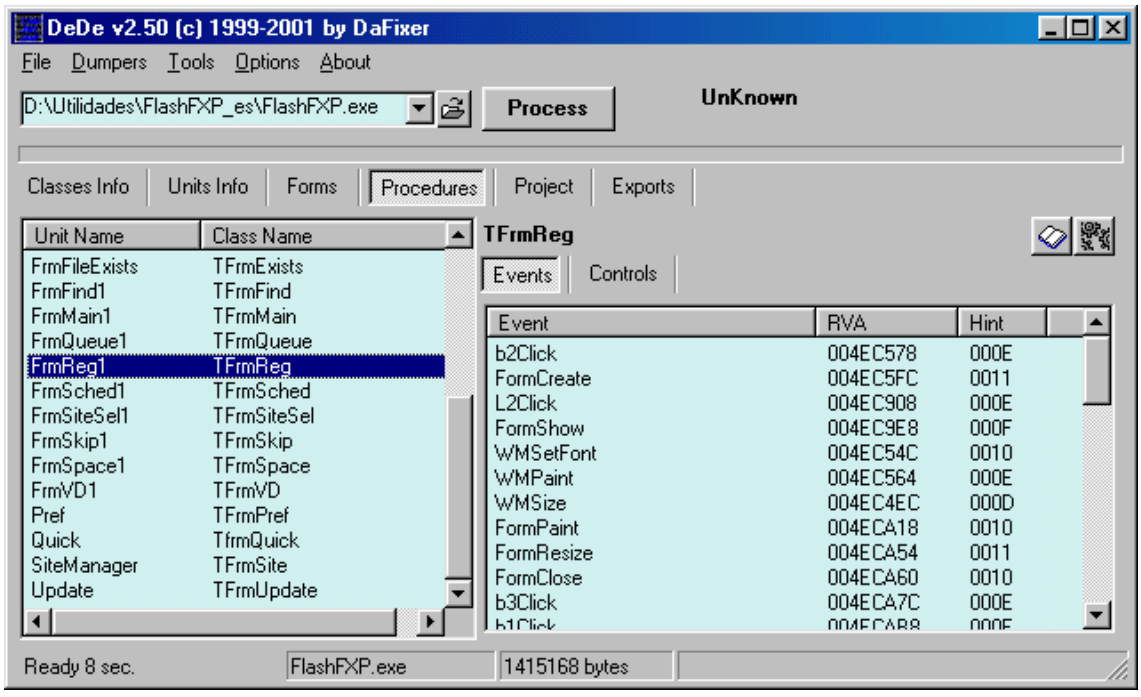


Lo primero es lo primero ...

Antes que nada tenemos que saber que nos traemos entre manos. Con el Languaje 2000 vemos que es un programa no empacado escrito en Delphi.

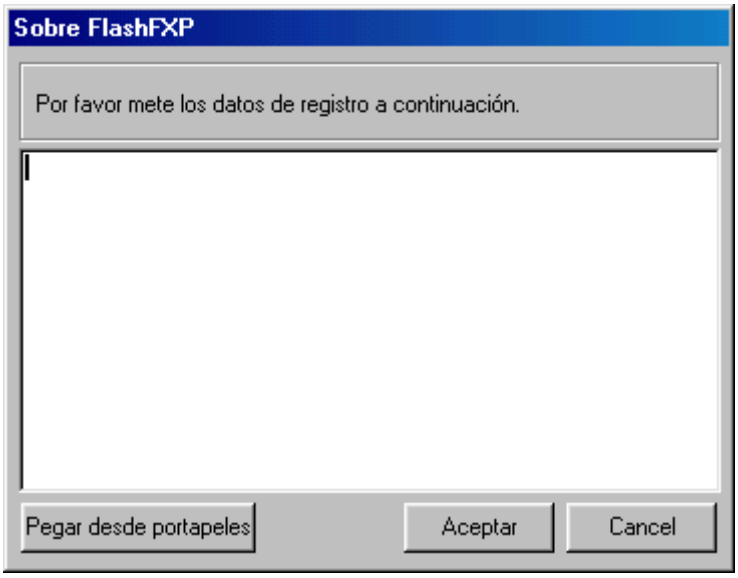
Desensambla muy bien con el Dede y vemos que hay un **FrmReg1** con un **b2click** que tiene como referencia el texto "Introducir código".

Si arrancamos con el Sice y ponemos un **Bpx en 4EC578** nos sorprenderá comprobar que aquí no para. Los otros events tampoco parece que contengan información determinante.



Encontrar el serial ...

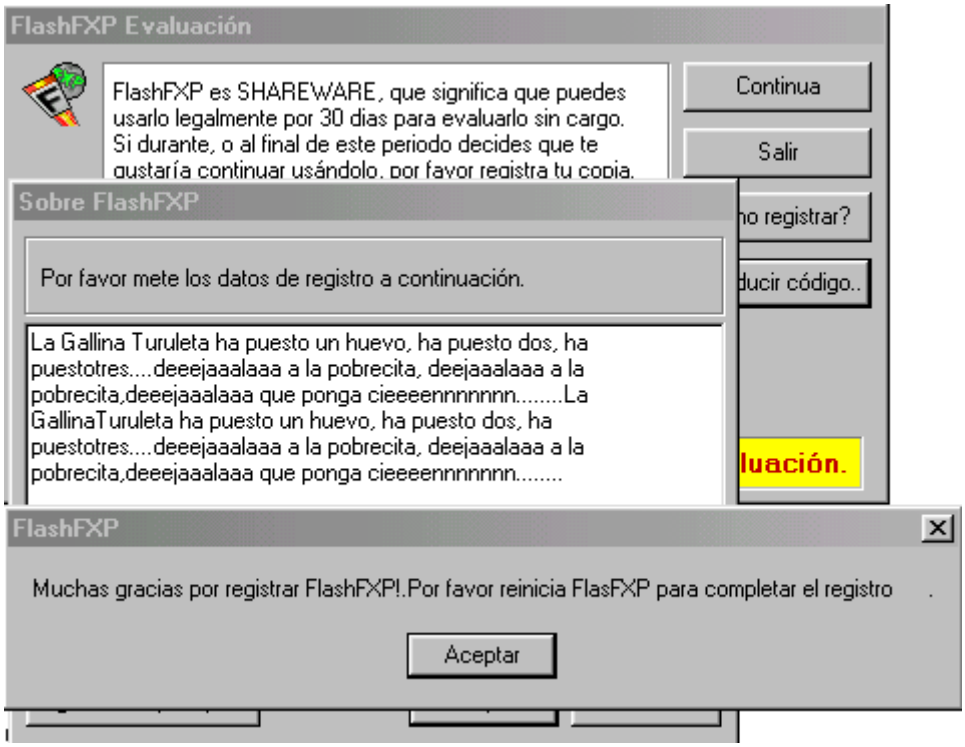
Ya no nos podemos aguantar más las ganas de entrar a saco con el Sice y metemos un **Bpx Hmemcpy** en la misma entrada del serial.



Después de tracear un poquito venimos aquí:

```
4BCC73 CMP EAX,8
4BCCBC CMP EAX,100
4BCCC7 JLE 4BCD9B    <-La longitud de la clave debe ser mayor de 100!
```

Sí forzamos este salto a ejecutarse, o simplemente metemos una cadena mayor de 100, obtendremos un mensaje de éxito instándonos a reiniciar el programa para completar el registro.



(He de reconocer que soy fan acérrimo de Fofo, Miliki y Fofito... son de mi época, sniff, sniff)

Al reiniciarlo veremos una ventanita que dice "Registrarse Fallido" y vuelta al cartelito con los días que nos faltan...

Detener el conteo ...

Ya que no podemos engañar al programa con que nos hemos registrado intentemos hacerle creer que el tiempo no pasa para nosotros.

Lo primero es averiguar que método utiliza el señor Charles de Wise (autor del programa) para llevar el conteo de los días. Las protecciones basadas en la evaluación de un programa por un tiempo durante el cual podemos utilizarlo plenamente y si nos gusta registrarnos, ofrecen una vulnerabilidad adicional que es precisamente el conteo del tiempo. Dicho conteo se realiza con una variedad de API`s como GetSystemTime, GetLocalTime, etc...

Una vez que el programa comprueba la fecha, hace sus cálculos para saber sin nos quedan más días de evaluación y si no es así nos saca un mensaje de "Evaluación Finalizada" o algo así y puede que realice cambios irreversibles para que ya no pueda utilizarse más una versión no registrada.

La cuestión no es saber como nos cuenta los días, bien guardando la fecha en que fue instalado, bien guardando los días que nos quedan o de otra forma aún más rebuscada, sino **DONDE** guarda esta información.

Básicamente hay tres posibilidades: En el registro, en un archivo que utiliza el programa con este propósito o en el propio ejecutable.

Para conocer cual de estos métodos es el que nos interesa en nuestro caso, avanzamos un día el reloj, corremos el TechFacts 98 y vemos que modificaciones se producen.

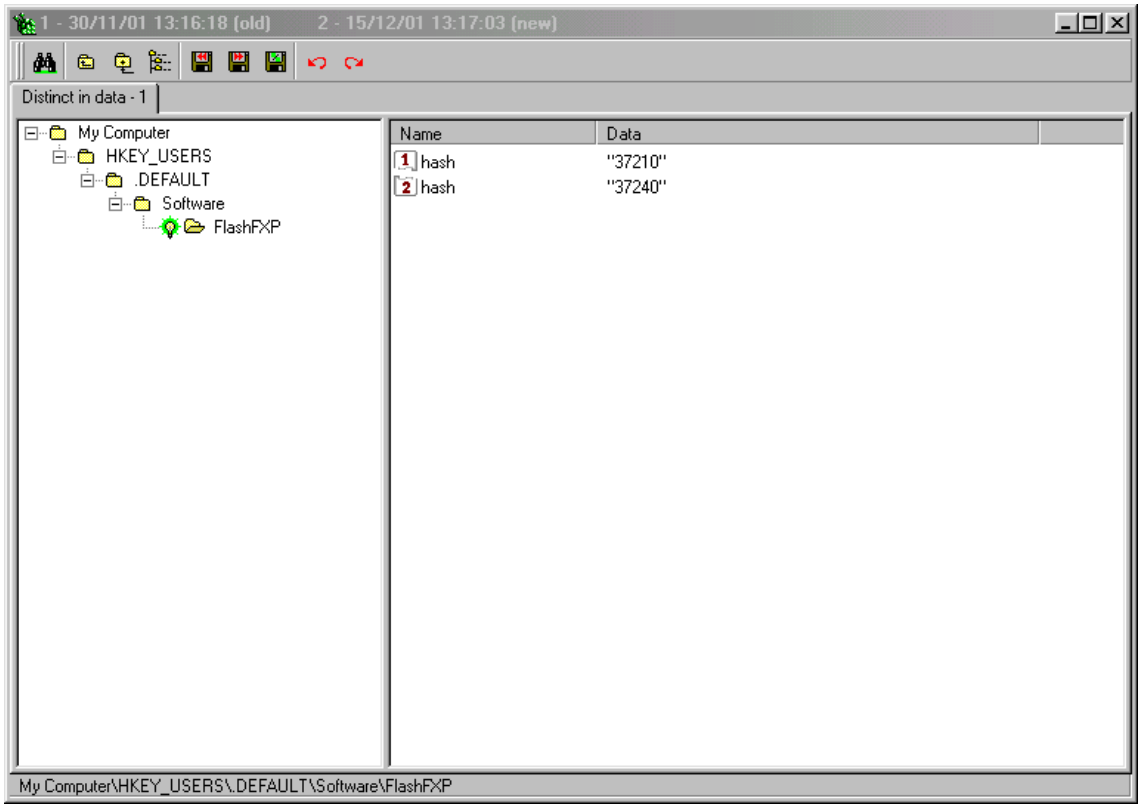
Hay 11 ramas del registro que se modifican:

```
[HKEY_LOCAL_MACHINE\Software\CLASSES\.FlashFXP_PI]
[HKEY_LOCAL_MACHINE\Software\CLASSES\.fqf]
[HKEY_LOCAL_MACHINE\Software\CLASSES\Applications\FlashFXP.exe]
[HKEY_LOCAL_MACHINE\Software\CLASSES\Applications\FlashFXP.exe\shell]
[HKEY_LOCAL_MACHINE\Software\CLASSES\Applications\FlashFXP.exe\shell\open\comm
and]
[HKEY_LOCAL_MACHINE\Software\CLASSES\FlashFXP.Document]
[HKEY_LOCAL_MACHINE\Software\CLASSES\FlashFXP.Document\shell]
[HKEY_LOCAL_MACHINE\Software\CLASSES\FlashFXP.Document\shell\open\command]
[HKEY_LOCAL_MACHINE\Software\CLASSES\FlashFXP_PI]
[HKEY_LOCAL_MACHINE\Software\CLASSES\FlashFXP_PI\DefaultIcon]
[HKEY_USERS\.DEFAULT\Software\FlashFXP]
```

Como puede verse no todo es entrar en el Regedit y pedirle que busque FlashFXP, pues la 2° rama de esta colección no la encontraría, aunque en nuestro caso eso sería irrelevante.

En algunos programas la clave de registro que interesa está escondida, encriptada y además no alude para nada al nombre de nuestro programa, por lo que es conveniente monitorizarlo. Yo tengo la costumbre de hacerlo con el TechFacts en los momentos claves; durante la instalación y cuando muevo el reloj o hay algún cambio que me parece interesante.

En alguna ocasión puede ser interesante volver al archivito de instalación y ver que hizo el programa. Una vez que sé que es en el registro donde se guarda esta información suelo correr el Advanced Registry Tracer que nos permite jugar un poco más con el registro.



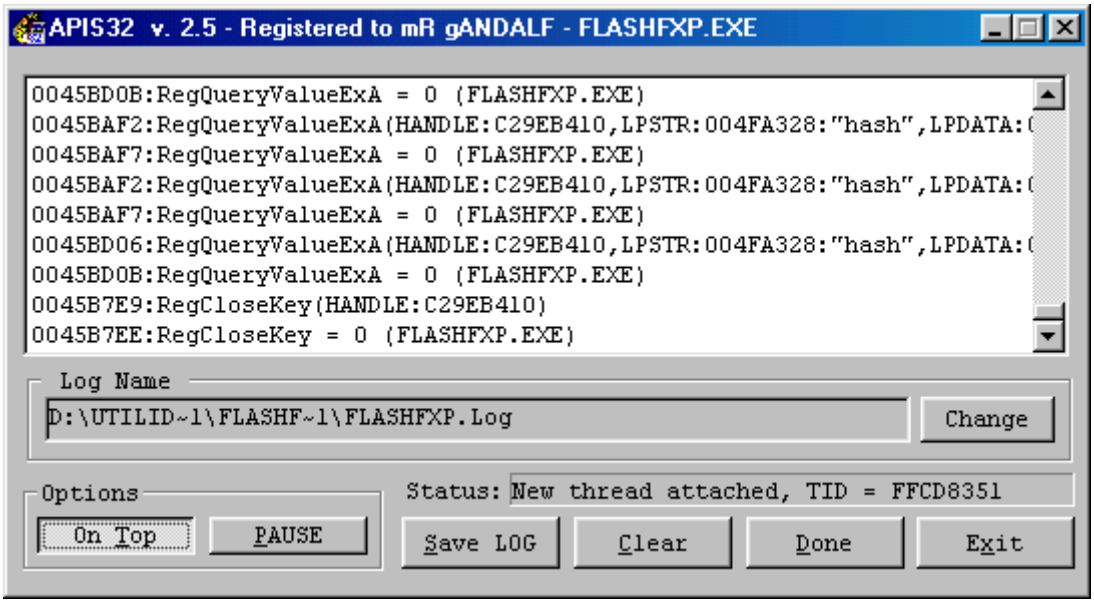
Vemos que adelantando el reloj del día 15 al día 30 el valor de **"hash"** cambia de **37210** a **37240**, e igual le sucede a **"MaxIcon"** (que no sale en la figura). De las dos clave es MaxIcon la que nos interesa.

Esto se averigua por el simple procedimiento de ensayo y error, tan util para los que como yo, no estamos tocados por la varita del Zen Cracking.

Posiblemente estas cifras representen formas más o menos crípticas de representar la fecha, pero simplemente poniendo MaxIcon a 0 nos basta para tener el contador siempre a 0.

Otro camino posible que se nos abre al conocer cual es la clave que gobierna el conteo es monitorizar los accesos del programa a dicha clave y llegar hasta el corazón mismo de la comparación de fechas, dejando que el complejo proceso de conteo avance pero sin que el programa sé de cuenta. Esta es una bonita maniobra que viene muy bien explicada en el tutorial de Karpoff sobre MemTurbo 1.5.

Tomamos el APISpy y seleccionamos todas las API`s que vienen con ADVAPI32, o mejor aún solo la mitad última de ellas que hacen referencia al registro, y tendremos un listado limpio:



El listado de FlashFXP.log, que es donde ApiSpy guarda toda esa información que hace scroll en esta ventanita, es un poco más largo que este, pero yo he selecciónado la parte interesante:

0045BAF2:**RegQueryValueExA**(HANDLE:CAA0B560,LPSTR:004FB218:**"MaxIcon"**,LPD
ATA:00000000,LPDATA:007AFBDC,LPDATA:00000000,LPDATA:007AFBF8)

0045BAF7:RegQueryValueExA = 0 (FLASHFXP.EXE)

0045BAF2:**RegQueryValueExA**(HANDLE:CAA0B560,LPSTR:004FB218:**"MaxIcon"**,LPD
ATA:00000000,LPDATA:007AFBBC,LPDATA:00000000,LPDATA:007AFBD8)

0045BAF7:RegQueryValueExA = 0 (FLASHFXP.EXE)

0045BD06:**RegQueryValueExA**(HANDLE:CAA0B560,LPSTR:004FB218:**"MaxIcon"**,LPD
ATA:00000000,LPDATA:007AFBD4,LPDATA:0196D740,LPDATA:007AFBE4)

0045BD0B:RegQueryValueExA = 0 (FLASHFXP.EXE)

0045B7E9:RegCloseKey(HANDLE:CAA0B560)

0045B7EE:RegCloseKey = 0 (FLASHFXP.EXE)

Aquí vemos desde que parte del código se "lee" la clave MaxIcon. Podríamos ir a mirar con el Wdasm y probar unos saltos con el Sice, pero creedme, no merece la pena tanto refinamiento para esta parte del crack.

Si adelantamos la fecha más allá del tope del trial veremos que lo único que pasa es que donde quedaba el botón continuar ahora sale uno de esos contadores de tiempo que te hace esperar 10 segundos antes de continuar. Luego, lo que verdaderamente nos limita es la ventana en sí y no el conteo. De todas formas, metiendo esta clave en el registro lo conseguimos parar:

[HKEY_LOCAL_MACHINE\Software\CLASSES\FlashFXP_PI\DefaultIcon]
@="D:\\UTILIDADES\\FLASHFXP_ES\\FLASHFXP.EXE,1"

"MaxIcon"="0"

Como podéis ver yo tengo instalado el FlashFXP en
D:\Utilidades\FlashFXP_es

Vamos a por esa molesta nag ...

Como yo soy un chico muy aplicado voy a seguir los pasos explicados en el tutorial de Dek-Oin sobre Nag`s que es, sobre todo un método, la mejor arma que podemos tener en esta guerra contra las protecciones.

Método de Anticipación: Consiste en anticiparnos a la sálida de la Nag. Simplemente ponemos un Bpx en todas las API`s que suelen utilizarse para hacer Nag`s y esperamos a que el Sice salte. Allí donde caigamos es chico malo y a partir de hay que tracear hacía atrás hasta encontrar un salto que invertir o un call que nopear.

Las API`s que nos interesan vienen en la lección 3 de Ricardo y que son:

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 GetWindowWord
bpx GetWindowInt

Lamentablemente el Sice no para en ninguna de estas, luego el señor Charles de Wise utiliza otro sistema para crear la Nag, vaya usted a saber cual.

Método de Salida: Consiste en identificar el punto donde la ventana se destruye para encontrar desde donde fue llamada. El comando de Sice es **Bmsg handle wm_destroy**, donde el handle, o manipulador, lo podemos encontrar con el wingestor de Crack el destripador y Marmota, o bien con el Sice haciendo task y buscando.

Con este wingestor es cosa de niños eliminar nags, pero no nos sirve para hacerlo definitivamente. Siguiendo con el método de salida encuentro que es especialmente difícil tracear hacía atrás la nag en las ventanas con varios botones, pero igual son manías.

Existe un tercer método descrito por Dek-Oin que se me antoja similar al del wingestor, que consiste en in visibilizar la ventana utilizando el Resource Hacker y el exescope.

Llegados aquí seguimos sin tener solución para nuestra nag, por lo que habrá que improvisar.

Afortunadamente y gracias a DaFixer (alabado sea) tenemos una herramienta llamada DeDe que incluso cuando más inútil parece nos termina sacando las castañas del fuego.

Sí volvemos a **FrmReg1** y ponemos Bpx al azar en alguno de estos eventos y probamos a tracear con F10 veremos desde donde se llama a la nag:

```
* Reference To: user32.PostMessageA, Ord:0000h
|
:004FA1DE E8B1D3F0FF    Call 00407594
:004FA1E3 A180E05200    mov eax, dword ptr [0052E080]
:004FA1E8 8B00          mov eax, dword ptr [eax]
:004FA1EA 8B10          mov edx, dword ptr [eax]
:004FA1EC FF92D0000000  call dword ptr [edx+000000D0] <- Aquí se llama
:004FA1F2 B201          mov dl, 01                      a la nag;
:004FA1F4 A130B64500    mov eax, dword ptr [0045B630]
:004FA1F9 E87215F6FF    call 0045B770
```

Y un poco más adelante se produce otra llamada:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004FA27B(U)
|
:004FA26D 8B45F8        mov eax, dword ptr [ebp-08]
:004FA270 E8AF8CF0FF    call 00402F24
:004FA275 C3            ret

:004FA276 E94993F0FF    jmp 004035C4
:004FA27B EBF0          jmp 004FA26D
:004FA27D A180E05200    mov eax, dword ptr [0052E080]
:004FA282 8B00          mov eax, dword ptr [eax]
:004FA284 E8E736F5FF    call 0044D970                <- Aquí se llama
                                                        de nuevo a la
                                                        nag;
```

Traceando un poquito hacia arriba encontramos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004FA006(C)
|
:004FA036 803DDCDE520000  cmp byte ptr [0052DEDC], 00
:004FA03D 0F8546020000    jne 004FA289    <- Bin Laden haciendo aeromodelismo?
:004FA043 8B0D80E05200    mov ecx, dword ptr [0052E080]
:004FA049 A13CE35200      mov eax, dword ptr [0052E33C]
:004FA04E 8B00            mov eax, dword ptr [eax]
```

Sí hacemos **jmp 004FA289** evitamos limpiamente la nag.

Hacia el corazón de la protección ...

Y como no podía ser menos en un programa hecho en Delphi termina saliendo todo el tomate. La cuestión obvia es:

Ese **cmp byte ptr [0052DEDC], 00** ¿será una Flag?

Abrimos nuestro Hiew y a buscar. Salen solamente 45 líneas en las que aparece un **cmp byte ptr [0052DEDC], 00** y siempre se sigue de un **JE** o un **JNE.**

Aquí hay material para andar probando que hace cada una y dejar el programa limpio como una patena, pero la protección ya está vencida y hay que empezar a prepararse para la fiestaaaaaaaaa!!!!!!!!!!

FELIZ 2002 A TODOS

Agradecimientos en especial para mi MAESTRO, Ricardo Narvaja, de cuyos tutoriales he aprendido todo lo que sé.

LECCIÓN 56: OTRA DE MI AMIGO GANDALF

INTRODUCCION DE RICARDO NARVAJA: Otro tute magnifico de Mr. GANDALF que mientras yo me repongo del STRESS y se me vuelven a conectar los cables, pelados durante mis vacaciones de enero del curso, ya Gandalf hizo dos magníficos tutes los cuales le agradezco mucho y espero que les gusten como a mi, y si les gustan, por favor, háganselo saber así le dan fuerzas para hacer más. No saben que lindo es que te digan que un esfuerzo que uno hizo es reconocido por los demás.

GRACIAS A Mr. GANDALF.

Programa: Internet Browser E-Mail Capturer 1.0.

Download: http://www.tecnomarketing.com/

Protección: Serial/Name. ASPACK. ShareWare Funciones limitadas.

Dificultad: Newbie.

Herramientas: SoftICE 4.0 - FrogsIce 1.09b4 - IceDump 6.0.2.5 - PEditor 1.5 - Import Reconstructor 1.2 - W32Dasm 8.93 - Ppatcher 4.0

Cracker: Mr Gandalf.

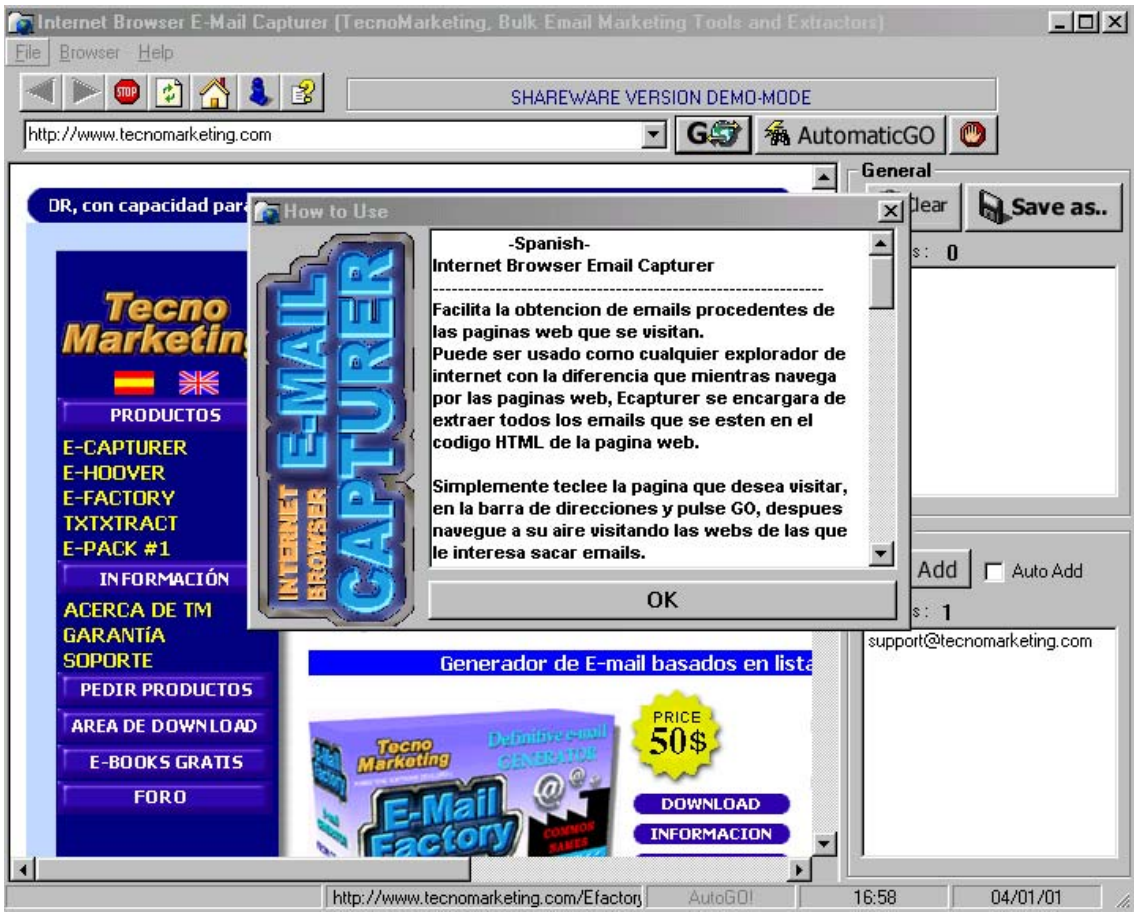
Fecha: 4/01/2002

Introducción: Casualmente encontré esta petición de Just_me en es.comp.cracks.


Google Grupos destripador Búsqueda en Google · Búsqueda avanzada de Grupos
Buscar sólo en es.comp.cracks Buscar en todos los grupos Buscar en la Web
Groups search result 2 for destripador
Autor: Just _Me (justme@flashmail.com) Search Result 2
Asunto: Para Crack el Destripador
Grupos de noticias: es.comp.cracks
Fecha: 2001-12-19 00:28:33 PST View: Complete Thread (42 articles) | Original Format
Tengo este problema que no puedo crackear, que opinas?
El programa es E-Capturer 1.0 http://www.tecnomarketing.com/Ecapturer.asp
Espero pueda ayudarme
Just Me
--
Posted from [200.37.176.234]
via Mailgate.ORG Server - http://www.mailgate.org/
Post a follow-up to this message

Francamente, el programa en sí no me interesaba mucho, pero quise probar y al comprobar que estaba hecho en Visual Basic pensé que sería tarea fácil.

La función del programa parece ser facilitar la obtención de emails procedentes de las paginas web que se visitan, o al menos así versa.



Desguaze: En realidad no es tan fácil como arrancar el SmartCheck y leer la clave. De entrada está comprimido y protegido contra los depuradores de Numega, incluido el Sice. El FileInspector de Vipper fue el analizador que mejor y más información aportó.

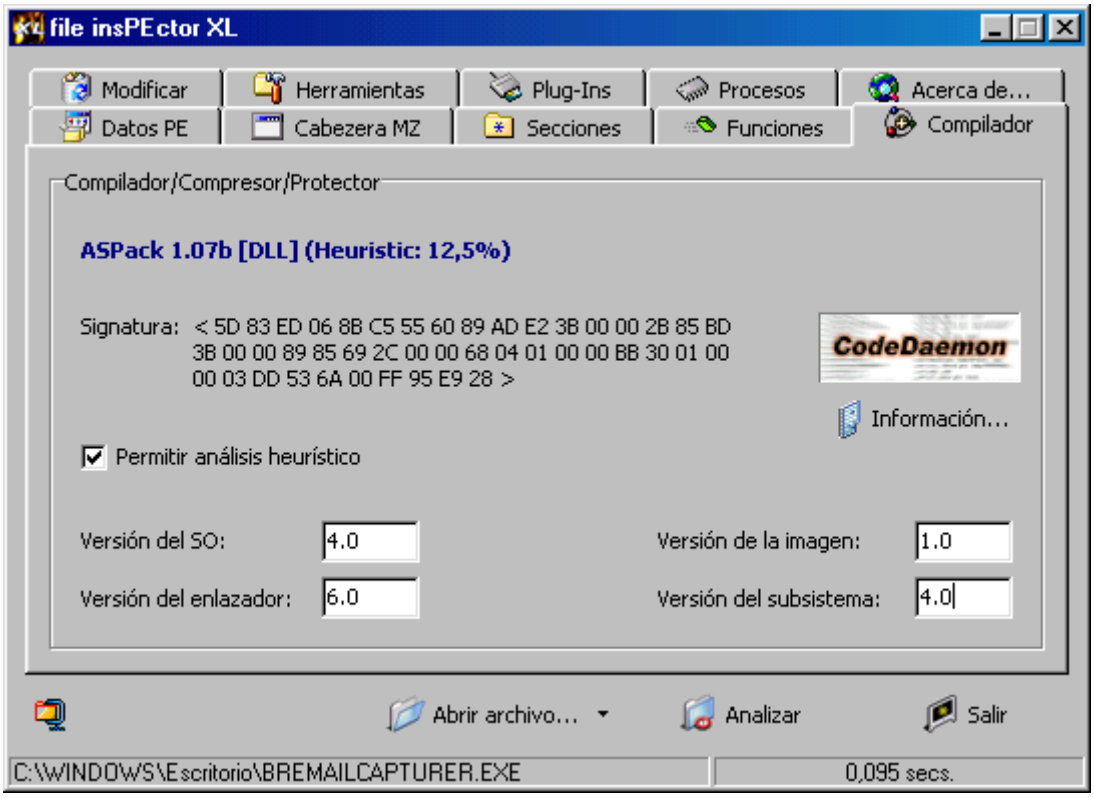
Por lo visto es ASPack 1.07b y misteriosamente ningún desempacador de los que probé pudo con él, por lo que tuve que desempacarlo manualmente.

Lo primero es repasarse las lecciónes 6, 34 y 35 del curso. Después todo va como la seda.

Resumiendo, un programa protegido con un compresor / encriptador comercial como ASPack comienza ejecutando una rutina que descomprime / desencripta el programa para que pueda ejecutarse justo a continuación.

Por tanto, si desensamblamos el programa tal cual no obtendremos nada, o lo que obtendremos será erróneo. ASPack además trae una protección antiSICE, es decir, además de las rutinas de descompresión se ejecuta un código que busca la presencia de Sice en memoria y sí lo encuentra no permite que se inicie el programa o bien provoca un cuelgue de windows, como es el caso.

Para obtener un ejecutable desprotegido de ASPack debemos copiarlo de la memoria una vez descomprimido / desencriptado, cambiar el entry point para que señale al comienzo del programa en sí y no al inicio de ASPack, y reconstruir la tabla de secciones. Siguiendo estos pasos ...



Dumpeado fácil...

Para bajarnos el programa de la memoria necesitamos detener la ejecución justo en el inicio de la primera instrucción del programa en sí y entonces copiar el trozo de memoria donde está.

La primera instrucción del programa en sí suele estar contenida en la primera sección de las que se muestran en la tabla de secciones.

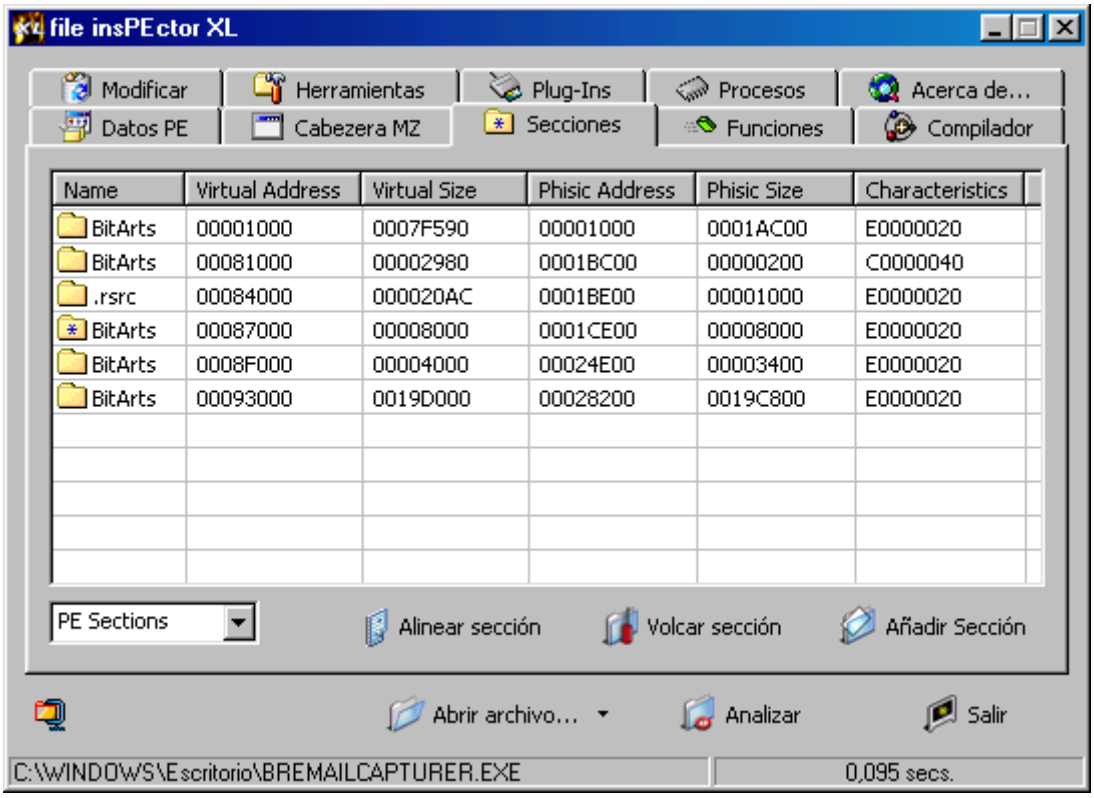
Calculamos el inicio y el final de esa primera sección y ponemos un **Bpr inicio final** con el Sice.

Inicio = Image Base + Virtual Offset
Final = Inicio + Virtual Size -1

	A	B	C	D	E
1	Image Base=	400000			
2	Virtual Offset=	1000			
3	Virtual Size=	7F590			
4	Raw Size=	1ac00			
5	Entry Point =	401900			
6	Size of Image=	230000			
7					
8	COMPRESIÓN	SI			
9	Inicio 1° sección=	401000			
10	Final 1° sección=	48058F			
11	Offset entry Point=	1900			
12	Incio del DUMP=	400000			
13	Fín del DUMP=	630000			

Hoja1 / Hoja2 / Hoja3

Yo utilizo esta pequeña utilidad en Excel para ahorrarme abrir la calculadora científica.



Entonces arrancamos el FrogsIce con las opciones **Hide FPLoader, Protect Sice Files** y **Autoscan on exit/start-up.** Cargamos IceDump y arrancamos el programa con el Symbol Loader (podría hacerse también desde el FrogsIce con la opción Hook int03h) y entramos al programa (es el principio de ASPack).

Ponemos **Bpr 018F:401000 018F:48058F r if (eip>=401000)&&(eip<=48058F),** pulsamos F5 y después de unos 15 segundos aparecemos en **401900 Push 40F610.**

Esta es la primera instrucción del programa que dumpearemos. Podría ser un falso entry point, p.ej podría volver a ASPack, incluso varias veces. Sabemos que es el verdadero por que si seguimos traceando con F10 un buen rato vemos que ya nunca vuelve a ASPack sino que carga las DLL's (en este caso MSVBVM60) y comienza a ejecutarse el programa en sí.

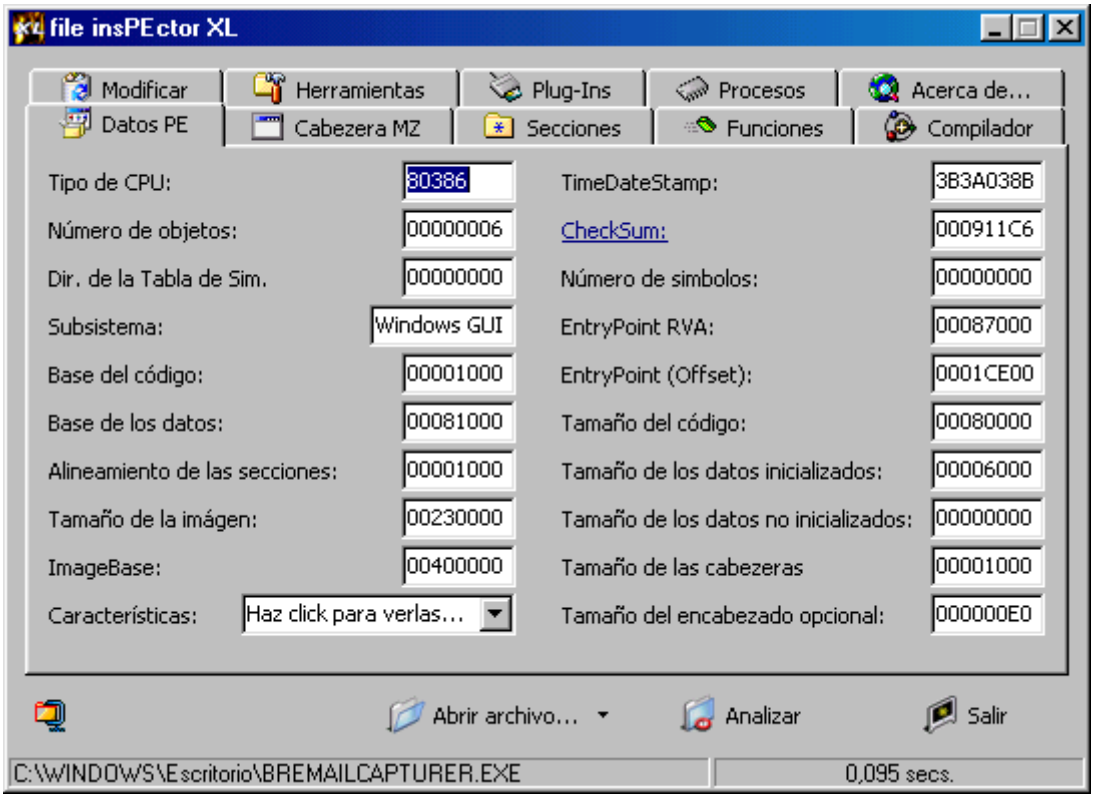
Sin salir de Sice ni movernos de **EIP 401900**, ponemos:

/DUMP inicio fin c:\Temp./dump.exe

Ahora inicio-fin son los del programa entero, no los de la primera sección. Estos nos los dice de nuevo el File inspector. El inicio es la image base y el tamaño es el size of image **(por lo tanto fin = image base + size of image).**

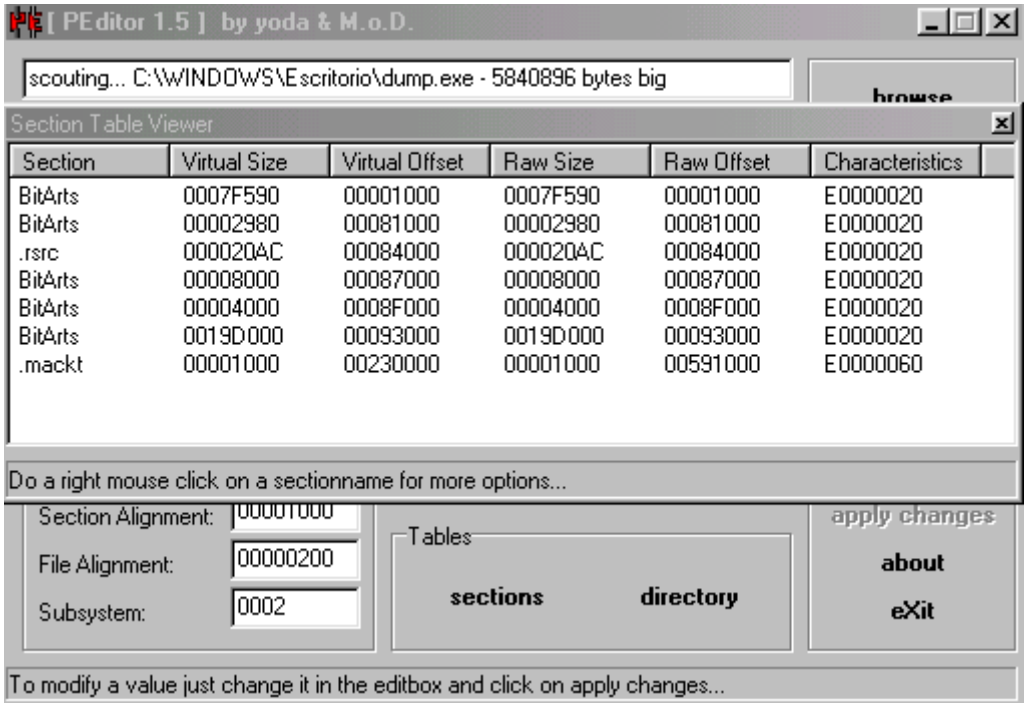
Todo este procedimiento viene muy bien explicado en las lecciónes 34 y 35 de RICARDO, pero como nunca viene mal un poco de practica, podéis coger el programa y seguir estos pasos.

Escribimos **/DUMP 400.000 630000 c:\Temp./dump.exe** y tenemos el programa dumpeado con un tamaño de 5,57 MB (por 1,76 del ejecutable comprimido!).



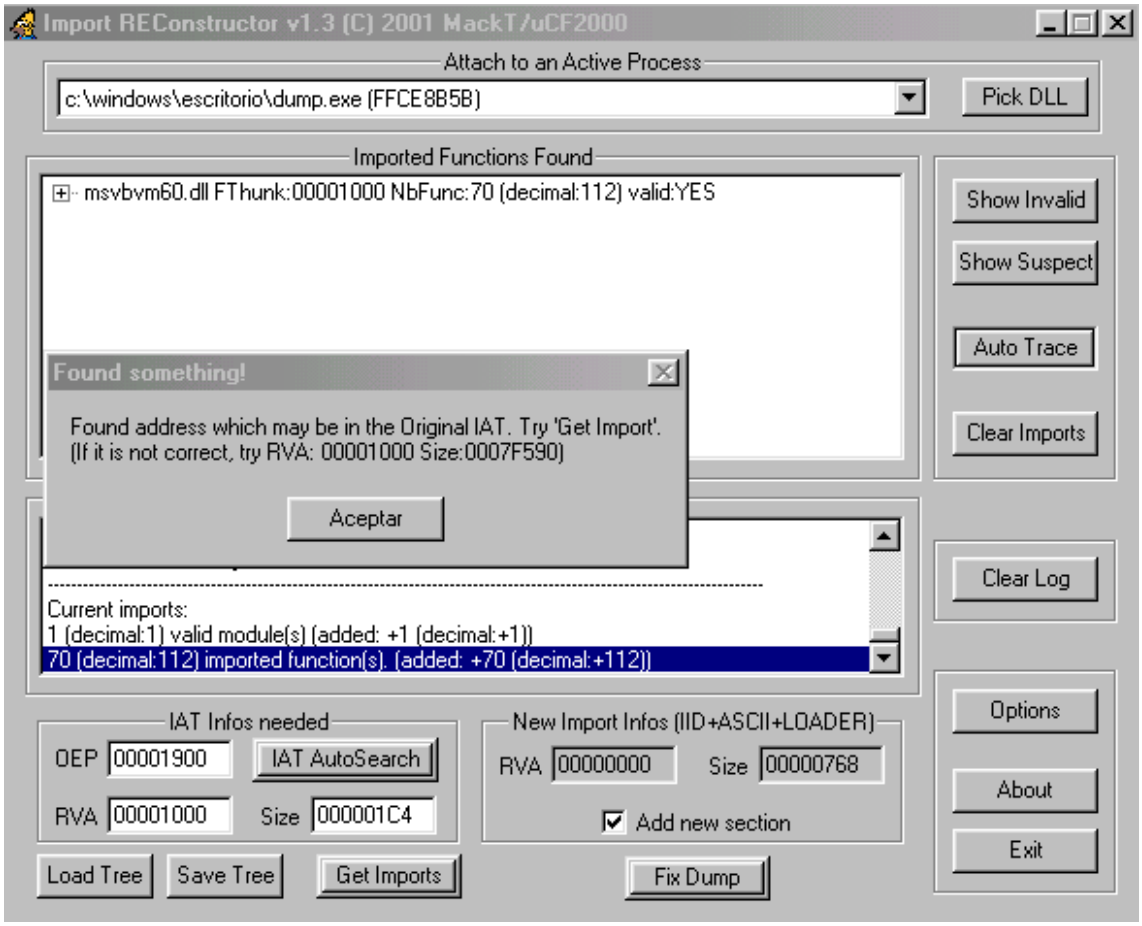
Arreglando el dump...

Ahora tenemos que "arreglar" el volcado, ya que así no nos serviría de nada. Abrimos el PEditor y empezamos los primeros cambios:



Cargamos el programa en browse, abrimos la tabla de secciones en sections y pulsando con el botón derecho del ratón se abre un menú en el que elegimos **dumpfixer**, con lo que se iguala el virtual size y raw size por un lado, y el virtual offset y raw offset por otro (queda como en la figura). También podemos aprovechar para habilitar poniendo **E0000020** en características de todas las secciones.

Después hay que calcular el IAT adress y el IAT length, para lo cual existe un método manual, pero yo probé el Import Reconstructor 1.2 y va de perlas. Hay que tener ejecutándose el programa original cuando arranquemos el Import Reconstructor y poner en OEP el offset del nuevo entry point, o sea, 1900 que ya teníamos calculado con nuestra hojita de Excel (**RVA - Image Base = 401900 - 400000 = 1900**)



Entonces pulsamos IAT reconstructor y se rellenan RVA y Size, avisándonos de que si estos valores no nos sirven podemos usar otros. Pulsamos **Get Imports** y en **imported functions** aparece **msvbsm60.dll (...) valid:YES**, que indica que vamos bien.

Ahora pulsamos **Fix Dump**, seleccionamos el archivo dumpeado cuando nos lo pida y Voila! Programa descomprimido y sin ASPack que podemos debbugear sin FrogsIce y desensamblar con sus String references y todo.

Sí con este Fix dump no nos hubiera quedado un volcado decente, o se colgara el programa, podriamos llevar los datos de RVA y Size de la izquierda al Revirgín, como en la lección 34.

Listo para el almuerzo...

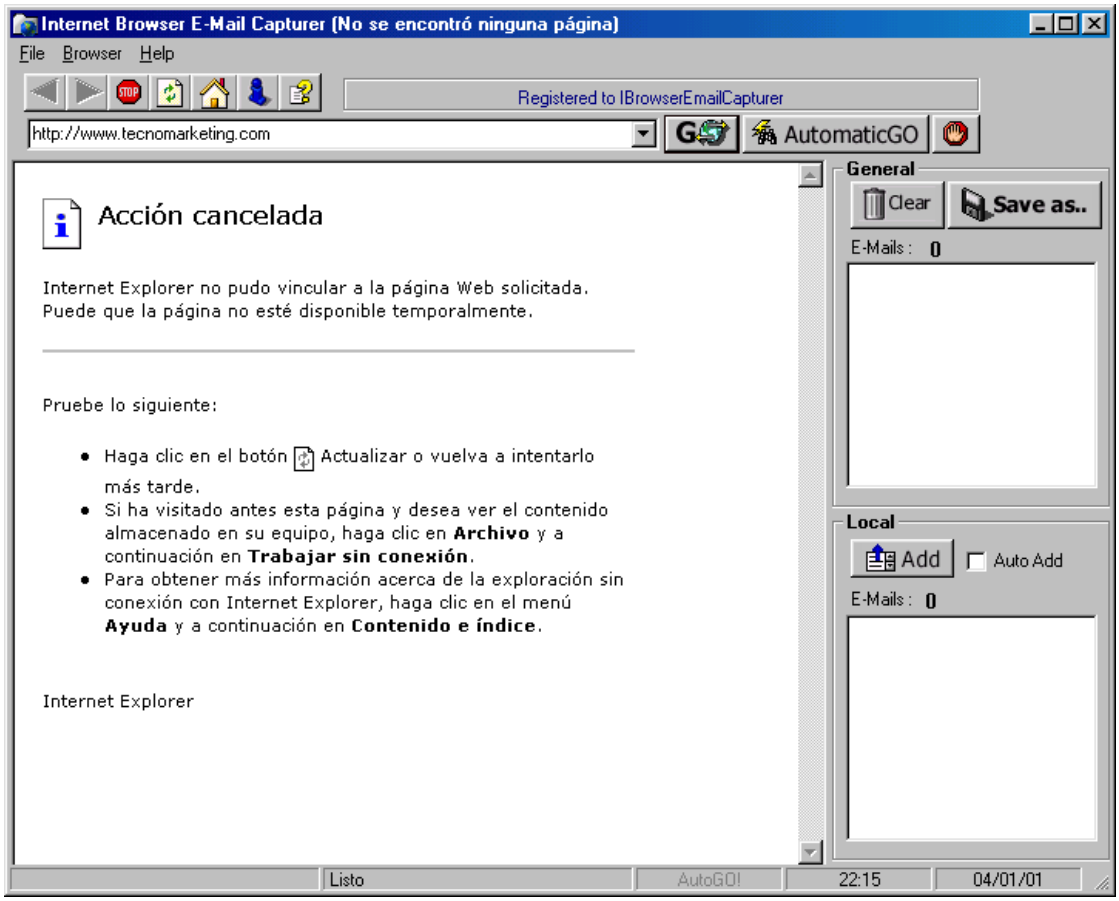
Así vencido y desnudito el pobre queda que da pena romperlo. Pero nosotros a lo nuestro, nos ha dado su buen trabajito y ahora hay que entrar a matar.

El SmartCheck sigue sin funcionar pero el Wdasm preparado para VB es más que suficiente.

Recordáis que lo primero que sale cuando arrancamos el programa es un cartelito que dice "Unregistered versión working in demo mode"?. Un poco más arriba del cartelito aparece **47A611 JE 47A6AA**.

Sí cambiáis este **JE** por **JNE** entrareis registrados como Ibrowseremailcapturer. Sin cartelito, ni límites ni nada. Para el ejecutable comprimido hice el loader con el Ppatcher, que me gusta por que solo actúa cuando la víctima está ya confiada y por su nombre castizo.

Que aproveche.



PD: Gracias a Ricardo, de cuyas lecciónes aprendí todo lo que sé y espero seguir aprendiendo (hay que pincharle para que haga más).

LECCIÓN 57: OTRA DE MI AMIGO GANDALF

INTRODUCCION DE RICARDO NARVAJA: Otro tutazo de Mr. GANDALF que cada dia escribe mejor. Muy bueno y completo con imágenes aclaratorias. Para las listas donde no salen los gráficos sepan que este tute los tiene, y seguimos descansando el mes de enero, en la playa (jaja ojala) tirado en la cama y trabajando muerto de calor, jua.

Muy bueno GANDALF gracias por ayudarme, tu nivel es de primera y estoy reagradecido.

Programa: Simstat 1.3

Download: http://www.simstat.com/

Protección: Trial de 30 días o 7 usos. ASPACK.

Dificultad: Newbie.

Herramientas: SoftICE 4.0 - FileINspector XL - CASPR v1.012 - ProcDump32 1.6.2 - TechFacts 98 - Advanced Registry Tracer 1.43 - DeDe 2.50 - W32Dasm 8.93 - Ppatcher 4.0

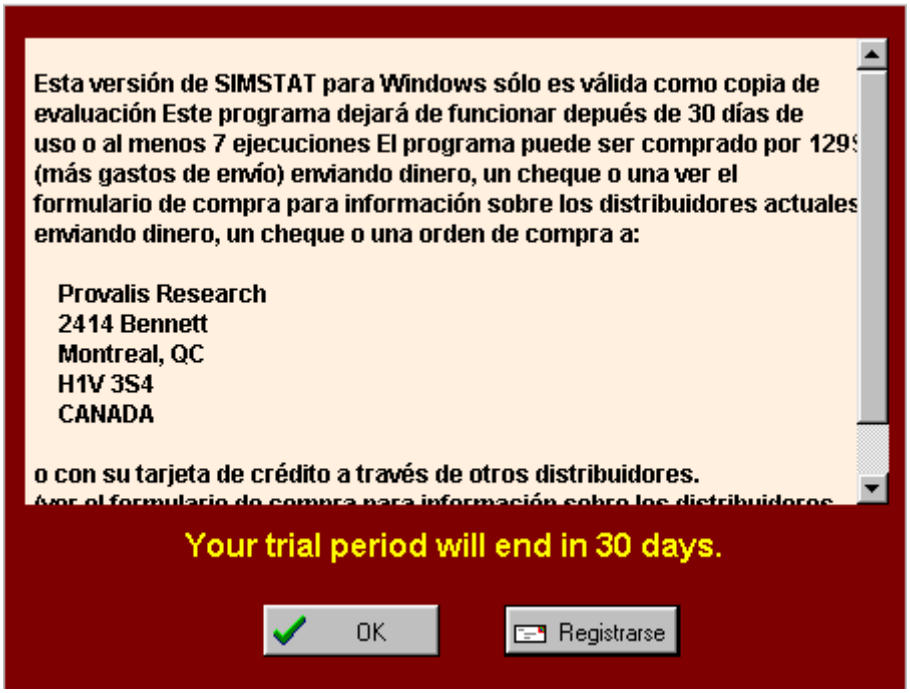
Cracker: Mr. Gandalf

Fecha: 10/01/2002

Introducción: Simstat 1.3 es un programa de la empresa canadiense Provalis Research que permite realizar detallados análisis estadísticos y, en opinión del Dr.Jorge R. Fernández, es uno de los mejores que hay.

Fue precisamente a petición suya en la lista de cracks latinos que me decidí a intentarlo.

Nada más arrancar el programa y mientras se carga vemos una ventanita que nos dice que no estamos registrados, desaparece y, acto seguido, se nos muestra otra "ventanita" informándonos de que en versión demo solo puede usarse durante 30 días o 7 usos, lo cual es verdaderamente un comienzo prometedor.



En el botón registrase entramos en una página de ayuda en la que nos explica las múltiples formas que tenemos de pagar para poder registrarnos. Según esta info lo recibiremos por correo cuando soltemos la pasta.

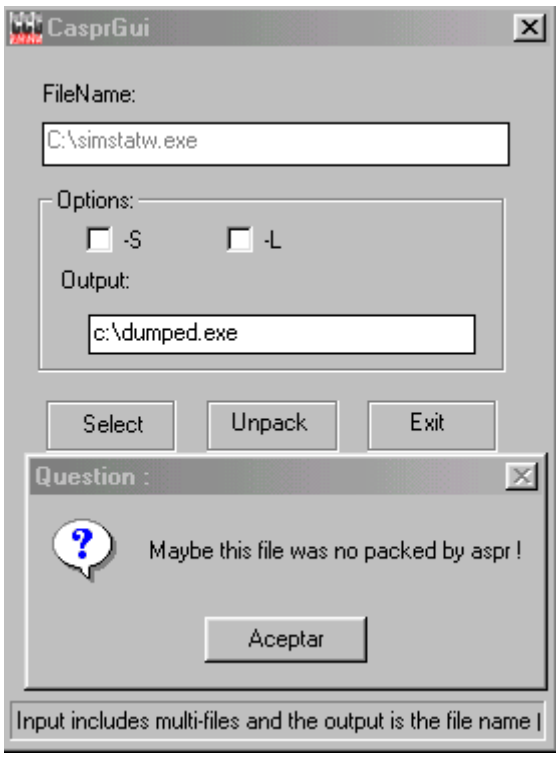
Después de pulsar OK entramos en el programa que ya no muestra ningún signo de que estemos ante una versión share ni ofrece la posibilidad de introducir un pass o serial para registrarnos.

Por otra parte, si pasan los 30 dias pero no lo hemos usado 7 veces, el programa nos permitira consumir estas oportunidades.

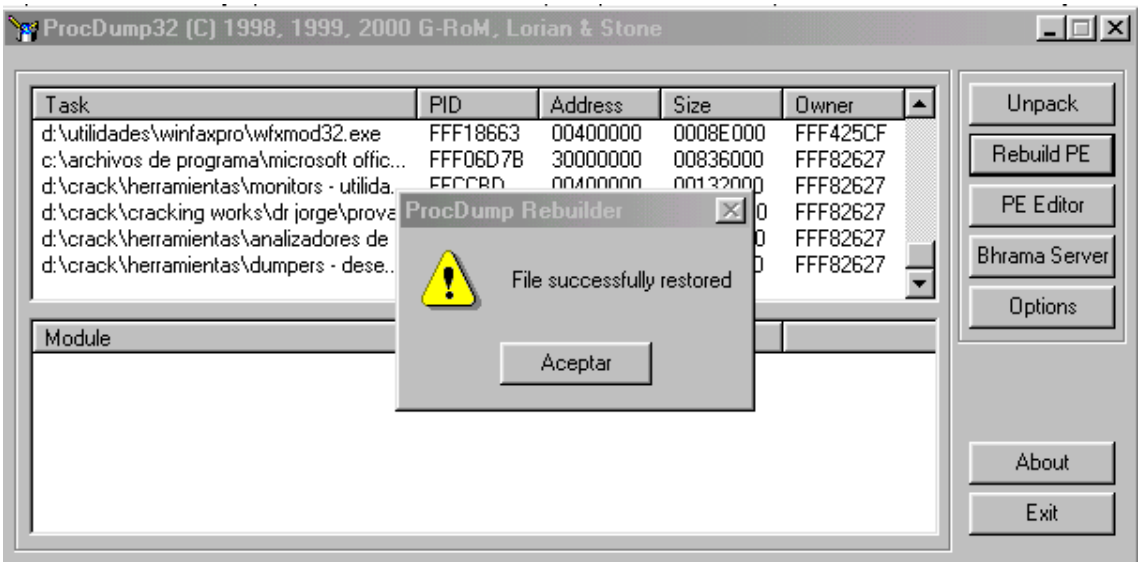
Siempre lo primero que hay que plantearse es el ...

Desguaze...

Sí abrimos el ejecutable simstatw.exe con el **FileInspector** de VIPER nos encontraremos con que se trata de **ASPack 1.08.04.** En esta ocasión Un-Pack 2000 tampoco funcionó, pero teníamos a Caspr que lo desempacó sin dificultad, aunque nos muestra un cartel que dice que tal vez este archivo no este comprimido por ASPr.



Antes de poder desensamblar tenemos que arreglar el encabezado del archivo con ProcDump. Para ello pulsamos rebuild PE y selecciónamos el volcado (c:\dumped.exe). Después de esto editamos la tabla de secciones en PE Editor y ponemos características como E0000020 para todas las secciones.



Si este archivo lo analizamos de nuevo con FileInspector nos llevaremos una sorpresa. Nos dice que esta compilado con Applok 95. La pregunta clave es: Mandeeeé??!??! Pero como somos crackers de recursos nos iremos al **Languaje 2000** que nos dice que está hecho en Delphi...ufff, que susto... Después de esta información es obligado pasarse por el DeDe. Aquí veremos algunas cosas interesantes.

Como siempre, "pescar" en el Dede depende del olfato de cada uno y aunque a todos nos mosquean palabras como Tregister, puede haber otras tanto o más interesantes.

Siempre es conveniente dedicar todo el tiempo necesario a revisar la estructura del programa en el DeDe y aunque a veces no encontremos nada de nuestro gusto y probemos a ver las string references en el Wdasm o a tracear las API`s en el Sice, es a menudo útil volver al DeDe con la información recopilada de otras herramientas y buscar aquellas direcciones que nos parecieron interesantes.

En este caso, la verdad es que no hay nada destacable salvo un Tregister y un Shware, pero en ninguno de los dos para el Sice.

Como decíamos en el tutorial sobre el Flash FXP, el programa debe llevar la cuenta de los días que faltan en algún sitio: El registro, un archivo auxiliar o en el propio ejecutable. También "sabe" sí movemos el reloj hacia atrás, pues si hacemos esto nos sale un cartelito diciendo que la fecha del sistema ha sido cambiada e instándonos a registrarnos. Después de esto el programa ya no funciona más.

Es de suponer que además de llevar la cuenta de los días que quedan también lleva la del día que instalamos el programa y si no concuerdan nos castiga. Sí nos damos un paseito por el Techfacts veremos que hay 4 archivitos y una rama de registro que se modifican cada vez que adelantamos un día el reloj:

```
c:\WINDOWS\APPLOG\APPLOG.ind
c:\WINDOWS\msds32.dll
c:\WINDOWS\simst32.ini
c:\WINDOWS\SYSTEM\mstx32.dll

HKEY_LOCAL_MACHINE\Software\Microsoft\Active Setup\Installed
Components\{44BBA840-CC51-11CF-AAFA-00AA00B6015C}.Restore
Value "DontAsk": from "15794178" to "15532034"
```

Sin embargo, si corremos el Advanced Registry Tracer y lo hacemos después de haber cerrado ese montón de aplicaciones que corren en secundario, veremos que ningún valor del registro cambia como consecuencia de adelantar el reloj.

De entre los 4 archivos son msds32.dll y mstx32.dll, ambos creados durante la instalación del programa, los que llevan el conteo de los días y los usos, al parecer de forma duplicada. Borrando esos dos archivitos volvemos a 0 usos y 30 días de Trial, salvo que el que nos haya vendido el ordenador sea el mismo que se lo vendió al Dr. Jorge R. Fernández, en cuyo caso es mejor ponerse una cataplasma fresquita en el occipucio, por lo que pueda pasar. Con esto podemos decir que lo tenemos cogido y va a ser muy difícil que se nos escape.

Si echamos un vistazo en el **APISs32** a las API`s que escriben en fichero, veremos que WriteFile es llamada en dos ocasiones durante el arranque del programa, una para escribir en msds32.dll y otra para escribir en mstx32.dll.

Además esta API no se utiliza en ninguna otra parte del programa, como podemos comprobar poniendo un Bpx en el Sice y corriendo todas las opciones. Esta API viene muy bien documentada en las Crackers Notes, traducidas gentilmente por TXELI.

WriteFile

La función WriteFile escribe datos a un archivo y está diseñada para la operación sincrónica y asíncrona. La función comienza escribiendo datos al archivo en la posición indicado por el indicador del archivo.

Después de que la operación de escritura se ha completado, el indicador del archivo es ajustado realmente por el número de bytes escritos, excepto cuando el archivo es abierto con FILE_FLAG_OVERLAPPED.

Si el manipulador de archivos se creara para la entrada y salida solapada (I/O), la aplicación debe ajustar la posición del indicador del archivo después de que la operación de escritura es terminada.

```
BOOL WriteFile(
 HANDLE hFile, // el manipulador de archivo para escribir
 LPCVOID lpBuffer,     // la dirección de datos para escribir al archivo
 DWORD nNumberOfBytesToWrite, // el número de bytes a escribir
 LPDWORD lpNumberOfBytesWritten, // la dirección del número de bytes escritos
 LPOVERLAPPED lpOverlapped // la direc. de estructura necesaria para I/O solapada
);
```

Returns

Si la función tiene éxito, el valor de retorno es TRUE.

Si la función falla, el valor del retorno es FALSE.

Para conseguir información extendida del error, llamar a GetLastError.

Lo que en nuestro listado de Wdasm corresponde a:

```
:00408B96 6A00        push 00000000
:00408B98 8D442404    lea eax, dword ptr [esp+04]
:00408B9C 50          push eax
:00408B9D 57          push edi
:00408B9E 56          push esi
:00408B9F 53          push ebx

* Reference To: KERNEL32.WriteFile, Ord:0000h
|
:00408BA0 E82BE4FFFF  Call 00406FD0
:00408BA5 85C0        test eax, eax
:00408BA7 7507        jne 00408BB0
```

Es en ESI donde se pasa **LPCVOID**, que es un puntero a la dirección de los datos que van a ser escritos, o sea, el número de días que faltan.

Ahora solo tenemos que cambiar ESI por otra cosa y habremos conseguido que el programa, después de haber hecho múltiples cálculos y

comprobaciones para ver si estamos registrados, termine guardando lo que nosotros queramos.

Si cambiamos **408B9E PUSH ESI** por **PUSH EDI**, o sea **0x408B9E/0x56/0x57** en el Process Patcher, habremos conseguido que el trial dure indefinidamente.Ahora solo nos queda quitarnos de en medio la nag.

Por curiosidad, si echamos una ojeada en **FileMon** (que tiempos aquellos de Mortadelo) veremos que el programa intenta acceder a un archivo **simstat.lic** (lic de licencia, se supone).

Si creamos un archivo simstat.lic entonces vemos que lo abre, lee algo, y luego sigue como si tal. Evidentemente no le convenció lo que le ofrecimos. Este era otro camino para romper la protección pero al ver que se complicaba la cosa lo dejé como segunda opción.

Para eliminar la nag ya hemos visto cuales son los procedimientos habituales en el tute del FlashFXP. Ninguno de estos funciona tampoco en el Simstat.

La nag se forma elemento a elemento, es decir fondo rojo (428171 CALL 0042FC0) y el resto dentro de un bucle, texto (4281A8 CALL 406FD0 la vez 43ª y 44ª que se ejecuta), botón registrar (el mismo CALL a la 45ª vez) y botón OK (46ª vez).

```
:004281A6 8B03                  mov eax, dword ptr [ebx] ----------------------
:004281A8 E89B210000            call 0042A348
:004281AD 8B03                  mov eax, dword ptr [ebx]
:004281AF 80787C00              cmp byte ptr [eax+7C], 00
:004281B3 740F                  je 004281C4
:004281B5 8B45FC                mov eax, dword ptr [ebp-04]
:004281B8 C7805001000002000000  mov dword ptr [ebx+00000150], 00000002
:004281C2 EB14                  jmp 004281D8

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004281B(C)
|
:004281C4 8B45FC                mov eax, dword ptr [ebp-04]
:004281C7 83B85001000000        cmp dword ptr [eax+00000150], 00000000
:004281CE 7408                  je 004281D8
:004281D0 8B45FC                mov eax, dword ptr [ebp-04]
:004281D3 E814FAFFFF            call 00427BEC

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004281C2(U), :004281CE(C)
|
:004281D8 8B45FC                mov eax, dword ptr [ebp-04]
:004281DB 8B8050010000          mov eax, dword ptr [eax+00000150]
:004281E1 85C0                  test eax, eax
:004281E3 74C1                  je 004281A6 ----------------------------------
```

La cosa no pinta facilona. Si rompemos el bucle, por ejemplo, haciendo **004281E3 jne 004281A6**, parecerá que hemos vencido. No sale la nag, el programa aparentemente funciona, pero... no sale la ventana de about (en Help) ni el Tools Setup (en Tools).

Parece que ese bucle hacía más cosas que dar la lata. También podemos probar el método de tracear con F10. Por este método veremos que el que da entrada a esta parte del código es un **5F0395 CALL 005F0000**.

Este CALL es responsable de un procedimiento, según vemos en DeDe, que al parecer es el responsable de toda la seguridad del programa.

Nopeando este CALL queda completamente crackeado el programa, no hacen falta más cambios. De todos los loader que probé, nuevamente solo el Process Patcher funcionó correctamente.

mR gANDALF

PD: Gracias a mi maestro, RICARDO, de cuyos tutoriales aprendí todo lo que sé y espero seguir aprendiendo.

LECCIÓN 58: OTRA DE MI AMIGO GANDALF

INTRODUCCION DE RICARDO NARVAJA: Quería agradecerle yo, ya que no ha habido muchos agradecimientos en la lista a Mr. GANDALF, un verdadero amigo y como han visto ya, todo un cracker, la ayuda que me ha dado durante este mes de enero, y decirle que siempre que quiera colaborar en cualquier momento con un tute de el, será bienvenido ya que a mi me gustan mucho sus tutes. La opinión de los demás no la sabemos ya que nadie dijo nada, pero bueno yo lo aliento por que yo sé lo que es estar horas detrás de una computadora tratando de crackear un programa para ayudar a alguien o para enseñar, con solo el rédito de la satisfacción por ayudar a los demás sin esperar nada.
POR ESO AMIGO GANDALF, MUCHISIMAS GRACIAS Y QUISIERA QUE ESTE NO FUERA TU ULTIMO TUTE COMO VOS ME ESCRIBISTE. SEA POR EL MOTIVO QUE SEA ME GUSTARIA QUE ALGUNA VEZ CUANDO PUEDAS Y TE DE EL TIEMPO, MANDES UNA COLABORACION PARA DESPUNTAR EL VICIO TUYO Y MIO, EL CRACKING.

UN ABRZO GANDALF SABES QUE ACA TENES UN AMIGO
RICARDO NARVAJA

Programa: SciTech GLDirect Versión 2 (28 Jun 2001)

Download: www.scitechsoft.com

Protección: Trial de 21 dias. Registración por name y serial.

Dificultad: Fácil.

Herramientas: SoftICE 4.0 - W32Dasm 8.93 - Hiew 6.76

Cracker: mR gANDALF.

Fecha: 30/01/2002

Introducción: Después de un tiempo al margen de la lista debido a problemas técnicos (estallido de un CD en la unidad de DVD en periodo de garantía del ordenador, con lo que hubo que llevarlo a la tienda con el consiguiente cashondeo y "vuelvaustedmañaneo") el humilde aprendiz de crackeador y amigo vuestro, mR gANDALF, resurge como el Ave Fenix para atender una petición de la lista.

Se trata de noseman, muy interesado en este programita que para funcionar necesita DirectX 7 o superior, ojo!, que podéis bajar de la red, pero no utilicéis para ello la utilidad que acompaña al programa, es mi consejo.

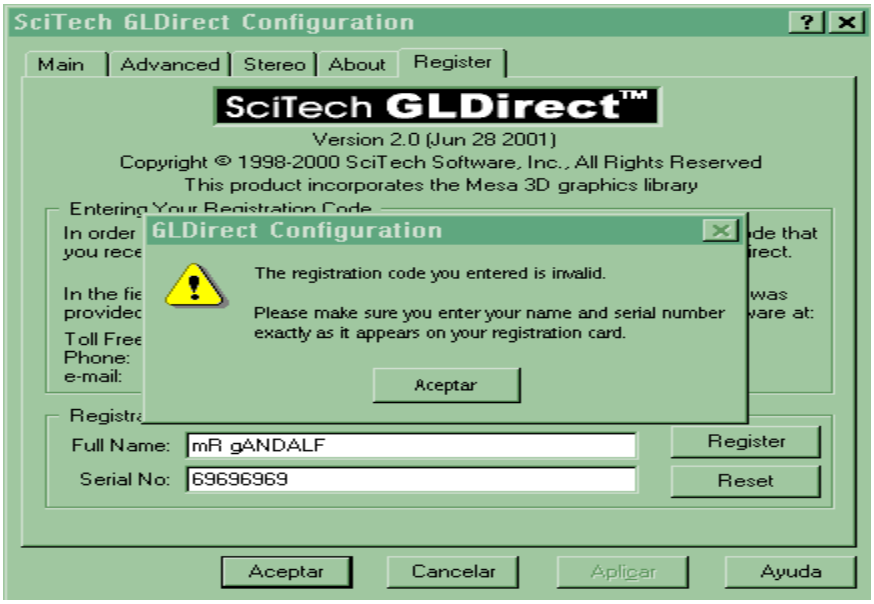
Según reza la ayuda del programa (in english, of course)...

> SciTech GLDirect is the utility package for Windows 95/98/Me that combines the power of the OpenGL API with the wide availability of Direct3D hardware drivers.
>
> It accomplishes this by enabling OpenGL based games and applications to access 3D hardware acceleration through the Direct3D 7.x drivers provided by your graphics hardware manufacturer. The OpenGL API is the cross-platform, high performance standard for 3D graphics applications.

Desguaze: Como siempre, empezando por el principio es como mejor se llega al final. Lo primero es instalarlo bajo estrecha monitorización de TechFacts guardando el informe en un archivo por si acaso luego hay que buscar algo interesante. El primer uso también lo monitorizo con TechFacts.

Después hay que echarle un vistazo con el FileInspector de VIPPER y ver con quien nos medimos: Parece no empacado y además es Visual C++ 6.0, buena señal.

Si echamos un vistazo en la pestaña que dice sections veremos que en características nos pone E0000020, que es lo que nos permite desensamblarlo con Wdasm. Por tanto cambiémoslo y veamos el "listado muerto" con Wdasm. Aquí aparecen cosas interesantes, como el texto de los mensajes de chico malo que nos salen al hacer un intento de registrarnos:



Francamente esto va tomando buena pinta. Si ahora ponemos un **Bpx Hmencpy** antes de pulsar el botón Register y traceamos hacia delante con F10 llegaremos al cartel de chico malo en **405AFC CALL dword ptr [041E054] "The registration code you entered is invalid"**.

Fijense que hay distintos carteles de chico malo. En todos pone arriba: "The registration code you entered, etc... ". Sin embargo, abajo el mensaje es distinto, según que no sea correcta la longitud u otros parámetros que el programa mira en el serial.

Ahora el procedimiento es sencillo, se trata simplemente de intentar buscar un salto que evite estos mensajes y que nos registre. Después de un par de intentos facilones sin éxito mirando en el listado de Wdasm, me fui al Sice que me gusta más.

Con un **Bpx** en **Hmencpy** y sabiendo cual es el salto de chico malo fui traceando hacia delante, mirando por aquí y por allá como el programa toma mi name y mi serial y se pone a hacer malabarismos binarios para encriptarlos y que no sea obvia la comparación.

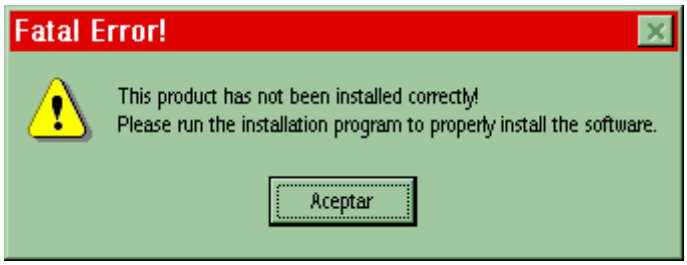
Lo más "limpio" aquí seria intentar desenmascarar la rutina encriptadora y hacer un keygen. Pero si te encuentras perezoso y piensas que con registrarlo es suficiente, entonces solo sigue traceando hacia delante. Verás que en sucesivos JE's precedidos de los respectivos Coles (de Bruselas) va realizando cada una de las comprobaciones pertinentes cuyos mensajes de error encontraras en las String References de Wdasm:

```
:00408E80 E88BF6FFFF  call 00408510    <- Aquí vuelve eax 0
:00408E85 83C41C      add esp, 0000001C
:00408E88 85C0        test eax, eax
:00408E8A 750B        jne 00408E97     <- Aquí evita el cartel "Please
:00408E8C 5F          pop edi             make sure you entered the
:00408E8D B802000000  mov eax, 00000002   name..."
:00408E92 5E          pop esi
:00408E93 83C408      add esp, 00000008
:00408E96 C3          ret

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408E8A(C)
|
:00408E97 837C240801  cmp dword ptr [esp+08], 00000001
:00408E9C 760B        jbe 00408EA9     <- Aquí evita el cartel "Please
:00408E9E 5F          pop edi             make sure you obtain the..."
:00408E9F B802000000  mov eax, 00000002
:00408EA4 5E          pop esi
:00408EA5 83C408      add esp, 00000008
:00408EA8 C3          ret

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408E9C(C)
```

Aquí sigue con la secuencia que muestra el cartel de registrado (...)

Y direis... ¡Pues que fácil, jope!... Pues NO!!, jope NO que es de pijo, lesheee...

Probad a reiniciarlo y ... ooohhhhh!!!!...

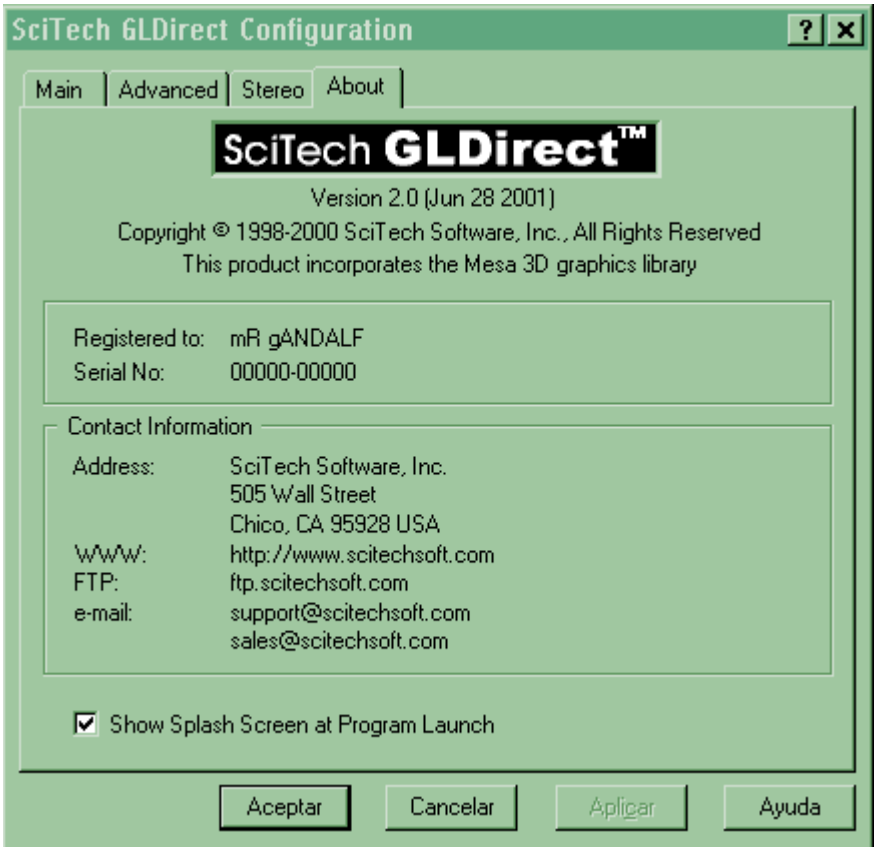


Que maric!!... Con esto aprenderemos algo importante. El método en esto del crackeo es fundamental, como en cualquier cosa. Sí hubiéramos seguido un buen método, considerando la posibilidad de que el programa realizara más comprobaciones o marcará nuestro ordenador de algún modo para que después de detectar un intento de pirateo no arrancara más, por ejemplo monitorizando con TechFacts (y mejor si guardamos también un "snapshoot" en ART), ahora todo seria coser y cantar.

Probablemente el programa guarde nuestra clave o un engendro encriptado y escondido de nuestra clave en el registro o en un archivito perdido en el disco duro, realice una comprobación al iniciarse y se de cuenta del pastel que le hemos preparado.

Como nos hemos confiado y no hemos hecho esto, lo cual nos facilitaría mucho las cosas, ahora solo nos queda el procedimiento de buscar de donde sale esta ventanita y quitarla, a ver que pasa. Traceando hacia delante vemos que sale de:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00407D2D(C)
|
:00407D6D E84E0D0000  call 00408AC0
:00407D72 85C0        test eax, eax
:00407D74             jz 407D8F      <- Aquí se evita el CALL
:00407D75 1968BC      sbb dword ptr [eax-44], ebp
:00407D78 124200      adc al, byte ptr [edx+00]
:00407D7B E84E0D0000  call 408AC0    <- Aquí sale el cartelito malvado
:00407D7C D01B        rcr byte ptr [ebx], 1
:00407D7E 0000        add byte ptr [eax], al
:00407D80 83C404      add esp, 00000004
:00407D83 33C0        xor eax, eax
:00407D85 5F          pop edi
:00407D86 5E          pop esi
:00407D87 5D          pop ebp
:00407D88 5B          pop ebx
:00407D89 83C438      add esp, 00000038
:00407D8C C21000      ret 0010
```



A pesar de todo el programa nos lo puso fácil. Este crack puede hacerse con cualquier patcher o loader pero adolecerá de un elemento que puede ser importante: La caducidad.

Si lo aplicamos a un programa que ya ha caducado posiblemente no funcione. Yo ya no puedo dar marcha atrás pues aunque lo desinstale no me da la posibilidad de volver a entrar en modo share y estudiar el modo de vencer el límite de tiempo, que no puede ser muy difícil, visto lo visto. Por eso lo dejo para otra mente más aguda o que siga más escrupulosamente el método, tan importante en esto del cracking como ya dijo el maestro Descartes...

mR gANDALF

PD: Y a mi amigo Ricardo Narvaja, maestro también, un fuerte abrazo y mucho ánimo para que siga con este curso estupendo que ayudará, sin duda, a muchos a iniciarse en esto del cracking.

LECCIÓN 59: OTRA DE MI AMIGO GANDALF

Programa: McAfee Firewall 3.0 (versión en inglés de 30 días)
McAfee Virus Scan 6.0 Pro (versión en ingés de 30 días)
McAfee Virus Scan 6.0 (versión en español de 30 días)
McAfee QuickClean 2.0 (versión en castellano de 30 días)
McAfee Internet Security 4.0 (version en ingles de 90 días)

Protección: Trial de 30 días.

Download: CD que acompaña a la revista PC Actual Diciembre del 2001.

Dificultad: Mediana.

Herramientas: SoftICE 4.0 - TechFacts 98 - Advanced Registry Tracer 1.43 - Hiew 6.76

Cracker: mR gANDALF.

Fecha: 03/02/2002

Introducción: Se trata del completo y conocido paquete de aplicaciones de McAfee sobre seguridad informática, a saber, Firewall, VirusScan, QuickClean e Internet Security (que incluye los dos últimos), que en su versión shareware, completamente operativa durante un periodo variable de tiempo, ha sido recientemente distribuida en un CD gratuito por algunas revistas informáticas.

En este caso se trata del número de Diciembre del 2001 de PC Actual, que generosamente aporta tan suculenta y, a juzgar por la búsqueda en Astalavista, pocas veces escogida víctima.

Por otra parte y por aquello del qué dirán, tengo que recordarles que este tutorial se publica en la lista de cracks latinos únicamente con fines educativos y que si les interesa el programa deben comprarlo.

Desguaze: Digamos que para crackear este programa es suficiente con emplear un poco de lógica y ser escrupulosos con cada paso que damos, haciendo uso exhaustivo del "método Cartesiano" del que hablábamos en la lección 58 (verdad que sí, noseman?).

Sepamos que se trata de un programa Visual C++ 6.0, no empacado ni encriptado, con múltiples librerías distribuidas por varias carpetas del directorio C:\Archivos de Programa\McAfee.

... lo primero SIEMPRE es el planteamiento...

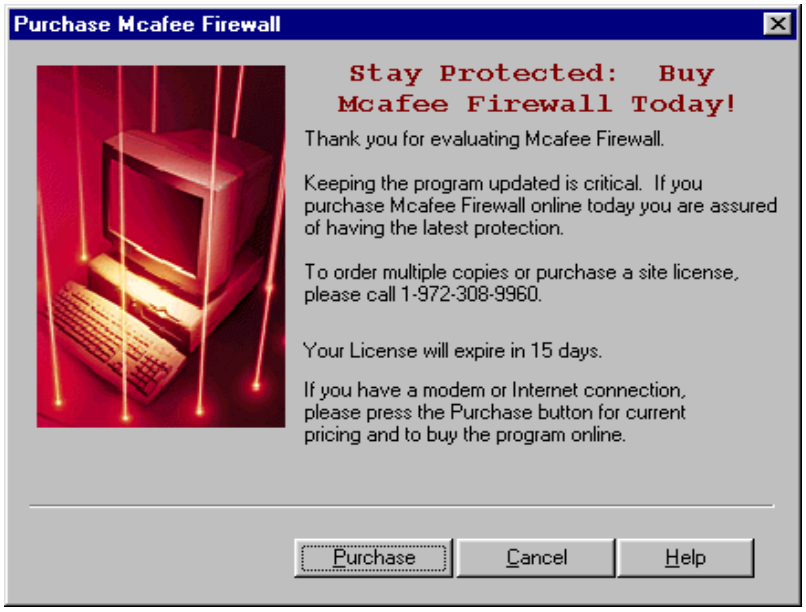
Nuestro objetivo es conocer como lleva el programa la cuenta del tiempo. De un modo general esto puede hacerse de varias maneras: Guardando la fecha en que fue instalado, la fecha en la que caduca, el número de días que se llevan gastados, el número de días que faltan o una combinación cualquiera de estos o de todos ellos.

Por otra parte, el sitio físico donde esta información se guarda puede ser una clave del registro (habitualmente; muy interesante el tutorial de Karpoff sobre el MemTurbo 1.5), uno o varios archivos escondidos (véase lección 57, Simstatw), en el propio ejecutable (hasta ahora nunca lo vi) o mediante una combinación cualquiera.

Para saber cual de estas es/son la/las que el programa emplea deberemos monitorizar que cambios se producen en el disco duro y en el registro de Windows mientras modificamos la fecha del sistema. Aquí es donde el TechFacts y el ART, cuyo manejo no voy a repetir por que ya lo hizo Ricardo (lecciónes 32 y 19 respectivamente), son fundamentales.

Nos centraremos en el McAfee Firewall 3.0, aunque adelanto que el sistema de protección es común a todas las aplicaciones y que crackeada una, crackeadas todas.

Este programa tiene una limitación de 30 días, aunque ya llegando al 15 empieza a avisarnos de que el tiempo corre que se las pela.



Pasado el día 30 nos muestra un cartelito similar avisándonos que el trial concluyó y el programa simplemente se cierra, eso sí, siempre nos da la opción "Purchase".

... el trabajo de campo ...

Utilizando el TechFacts y el ART llegamos a la conclusión de que es en el registro donde se guarda la fecha en la que el programa fue instalado y a partir de ahí hace sus cálculos.

Normalmente los programas suelen guardar también un conteo de días y cuando se percatan de que les toquetean el reloj hacen vencer el trial (lección 55, Flash FXP). Además, cuando desinstalamos un programa NO se borran las claves del registro que informan de la caducidad del mismo, por lo que reinstalarlo es inútil (verdad noseman?). Ni siquiera reinstalar Windows sirve, pues el programa de instalación recupera los archivos donde asienta el registro ,system.dat y user.dat, que guardan información tan molesta como que programas han caducado.

Un conocimiento algo más profundo del registro de Windows es necesario y para ello recomiendo el excelente tutorial de Karlitox que podréis encontrar en la pagina de Karpoff, como siempre.

Lógicamente, si atrasamos el reloj una vez el programa ha caducado, podrá volver a funcionar, cosa que no sucede en la mayoría de programas, como ya dijimos. Gracias al ART podemos borrar esas incomodas huellas del registro, para volver a instalar el programa con una fecha distinta.

También podemos crear archivos *.reg con los que engañarlo sobre la fecha de instalación. Este es un paso fundamental para llegar a saber cual es la clave del registro que guarda la información, que es lo que nos llevará hasta la "madriguera del oso".

Veamos, antes de instalar la 1ª vez tomamos una "foto" del registro, llamémosla F1. Hacemos la 1ª instalación con fecha de Febrero, p.ej, y

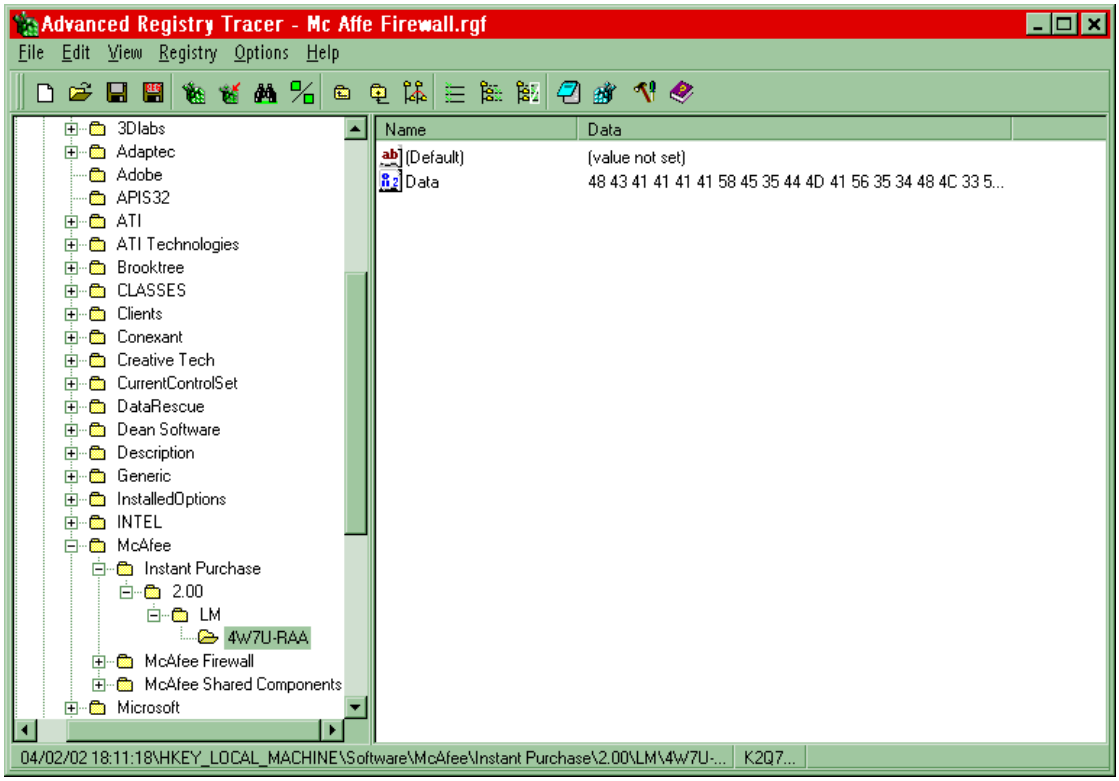
tomamos otra foto, F2. Si comparamos ambas y extraemos un archivo desinstalación.reg con el comando "Undo" podremos borrar toda huella de instalaciones previas. Además, si hacemos un archivo Reinstalar.reg con el comando "Redo", conseguiremos el efecto opuesto.

Veamos, sí hacemos una nueva instalación (tomamos foto, F3) una vez "limpio" el registro, con fecha de Abril, p.ej, y ejecutamos el archivo reinstalacion.reg que obtuvimos de comparar F1 y F2, veremos que el programa se caduca, pues le hemos introducido una clave de registro que le informa de que fue instalado en Febrero. Este efecto se invierte con una archivo "Redo" obtenido de comparar F2 y F3. El cerco se estrecha.

Ya tenemos todos los elementos necesarios para localizar la clave que buscamos. Si abrimos un archivo reinstalar.reg, p.ej, veremos que se compone de una sección delete que engloba una serie de ramas del registro y de claves que son borradas y de una sección add, donde son añadidas. A nosotros nos interesa, lógicamente, la sección add. En esta sección hay innumerables ramas y claves y una o varias de ellas son las que guardan la fecha. Habrá menos ramas cuanto más "pulcros" hayamos sido a la hora de usar el ART. Necesitaremos un mataprocesos como el de ProcDump para limpiar el sistema de aplicaciones que no vienen al caso y que hacen uso simultaneo del registro.

Tomando dos archivos reinstalar.reg que guarden información de dos fechas distintas y eliminando sucesivamente una clave tras otra para ver si el archivo reinstalar.reg conserva sus propiedades, llegamos a la conclusión de que es esta curiosa clave la que guarda la información que nos interesa:

[HKEY_LOCAL_MACHINE\Software\McAfee\Instant Purchase\2.00\LM\4W7U-RAA]
"Data"=hex:48,43,41,41,41,41,58,45,35,44,4D,41,56,35,34,48,4C,33,53,51
,46,57,\4C,4B,35,51,44,4B,47,57,38,55,4C,50,32,59,35,41,41,41,41,41,41
,41,41,41,41,41,\41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41
,41,41,41,41,59,50,41,41,\41,41,41,41,41,41,41,41,41,5A,4C,41,41,41,41
,41,58,35,48,41,41,41,41,41,41,41,\41,41,41,41,41,41,41,41,41,41,41,41
,41,41,41,41,41,41,41,41,41,56,58,4A,41,4C,\41,41,52,41,41,36,41,41,4B
,36,41,41,43,41,56,45,41,41,59,57,41,4B,38,50,43,41,\41,43,4C,41,56,41
,41,41,37,41,41,41,36,41,48,36,41,41,46,4B,56,4E,41,41,41,41,\41,35,50
,47,41,35,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,
\41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,
41,41,41,\41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,
41,41,41,41,41,41,\41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,
41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,\41,41,41,41,41,41,41,41,
41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,\41,41,41,41,41,41,
41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,\41,41,41,41,
41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,41,4
1,\41,41,41,41,41,41,41,41,41,41,4B,41,41,41,41,41,41



Esta clave tiene como fecha de instalación 2/Febrero/2002 y por sí sola se basta para hacer que el programa se comporte como si hubiera sido instalado en esa fecha aunque no sea así. El resto de claves que componen el archivo reinstalar.reg contienen otro tipo de información, no por ello accesoria.

Aquí hay que evitar la tentación de intentar descifrar la clave ya que, obviamente, esta encriptada en hex (429 cifras). Si fuera más obvia, habría una bonita manera de crakearlo basada en la introducción de código en el espacio libre que queda en los encabezados PE y que nos puede servir para modificar la información del registro cada vez que se carga el programa, adecuándolo a nuestros intereses de uso infinito (tutorial espléndido de un miembro del grupo WKT, perdónenme que ahora no recuerdo). Como no es el caso, deberemos acercarnos a la rutina de comprobación del trial basándonos en la información que tenemos.

... olfateando con LA BESTIA ...

Generalmente prefiere usarse el APISpy para localizar de que línea parte la llamada del registro que lee la clave que nosotros sabemos guarda la información necesaria para comprobar si aún estamos autorizados para utilizar la aplicación. ¿Cómo se hace esto?, pues muy fácil, monitorizando la aplicación con las APIS del registro que están en ADVAPI32.

En este caso, no sé muy bien por que, APYSpy no mostró que el registro leyera la clave **[HKEY_LOCAL_MACHINE\Software\McAfee\Instant Purchase\2.00\LM\4W7U-RAA]**, lo cual no es posible como se demuestra monitorizando simultáneamente con Regmonitor y viendo que, efectivamente, esta clave se lee después de unas 12, las últimas de

las cuales son **Software\McAfee\Instant Purchase\2.00\EconWr** y **Software\McAfee\McAfee shared components\Updater**. Pero tenemos a la BESTIA (alguien dijo algo de TRW2000?). Lo primero es saber que API vamos a tracear para saber como recibe los valores de la pila (aquí recomiendo el magnifico tute de Karpoff, "usos y abusos de la pila en las sesiones de depuración").

Una muy buena ayuda son las crackers notes de la lamentablemente desaparecida Inmortal Descendants:

RegOpenKeyExA / RegOpenKeyExW

La función RegOpenKeyEx abre la clave especificada.

```
LONG RegOpenKeyEx(
     HKEY hKey,          // el manipulador de clave abierta
     LPCTSTR lpszSubKey,// la dirección del nombre de la subclave a abrir
     DWORD dwReserved,  // reservado
     REGSAM samDesired, // máscara de acceso de seguridad
     PHKEY phkResult    // la dirección del manipulador de la clave abierta
     );
```

Returns

Si la función tiene éxito, el valor de retorno es ERROR_SUCCESS.

Si la función falla, el valor de retorno es un valor de error.

Vemos que es en el penúltimo elemento en el que se pasa el nombre de la clave que se va a abrir a la pila, HKEY hkey (se pasan de derecha a izquierda). Ponemos un BPX en esta API y observamos los siguientes breaks:

1° CPDCLNT

```
       push 401219      <- Software\McAfee\McAfee Firewall
       push (...)
401219 call RegOpenKeyExA
```

2° CPDCLNT

```
       push 407330      <- Software\McAfee\McAfee Shared components
       push (...)
401BA9 call RegOpenKeyExA
```

3° CPDCLNT

```
       push 407330      <- Software\McAfee\McAfee Shared components\central
       push (...)
401BCF call RegOpenKeyExA
```

4° central

```
       push  40034C6C   <-  Software\McAfee\McAfee Shared components\central settings
       push (...)
4016D9 call RegOpenKeyExA
```

5° Ole32!txt

```
         push  7FF4E38E   <- ClSID
         push (...)
```

```
7FF4E38E call RegOpenKeyExA
```

6° Ole32!txt

```
         push  7FF4E3DA  <- Software\Microsoft\Ole
         push (...)
7FF4E3DA call RegOpenKeyExA
```

7° Wininet

```
         push  7612FB20  <- Remote Access\Addresses
         push (...)
7613BE15 call RegOpenKeyExA
```

8° RNR20

```
        push  7828FAE0  <-  System\Current Controlset\Control\Service Provider\Service Types
        push (...)
7828BE0 call RegOpenKeyExA
```

9° RNR20

```
         push  78287996  <- System\Current Controlset\Control\Service
         push (...)
78287996 call RegOpenKeyExA
```

10° FWHOME

```
         push  ecx       <- Software\McAfee\Instant Purchase\2.00\EconWr
         push (...)
010E1D76 call RegOpenKeyExA
```

11° FWHOME

```
       push edi          <- Software\McAfee\McAfee shared components\Updater
       push (...)
010E1C89 call RegOpenKeyExA
```

12° Recapi

```
         push edx        <- ??? 00,00,11,08,00,00,71 (valores hex)
         push (...)
019C3A7B call RegOpenKeyExA
```

Llegados aquí nos aparece en la ventana de comandos del Sice el siguiente mensaje de depuracion:

```
-------- 02/05/2002   17:29:57 ---------------------------------
[Entering ECLicense Function, szProduct= MFW, szlanguaje = (null)
ECAPI-------------------------------------------------------------
```

Si volvemos a pulsar F5 (recuerden que hacemos un BPX RegOpenKeyExA) sale la ventana de trial acabado, lo cual, junto con el conocimiento por el regmonitor del orden en que se leen las claves, nos hace pensar que esta **"12° Recapi"** es la que contiene la que a nosotros nos interesa, **[HKEY_LOCAL_MACHINE\Software\McAfee\Instant Purchase\2.00\LM\4W7U-RAA].**

Conforme traceamos hacia delante nos vamos dando cuenta de que nos adentramos en las secuencias de lectura de clave. A continuación se ejecutan RegQueryValueExA y RegCloseKey. Más adelante vemos que se ha metido en EAX un puntero al inicio de la secuencia de valores hexadecimales que componen la clave. Posteriormente realiza un buen número de operaciones complejas en un bucle de 267 ciclos que contiene

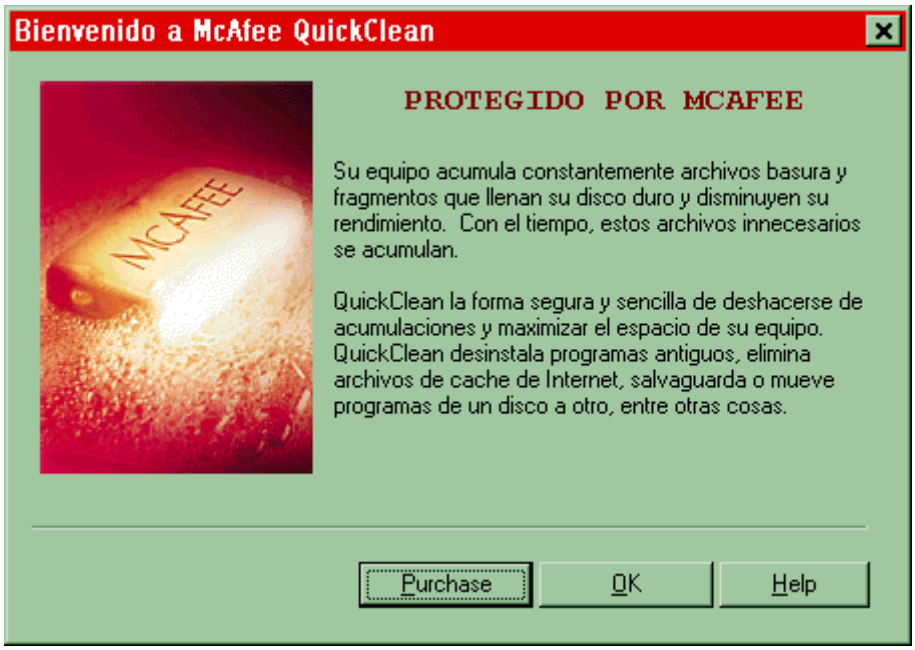
a su vez varios bucles más pequeños, que empieza en **19C32EE** y termina en **19C3379**, que es donde se "cuece el tomate" de la clave.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10003379(U)
|
:100032EE 8B7C2418      mov edi, dword ptr [esp+18]
:100032F2 85FF          test edi, edi
:100032F4 740C          je 10003302
:100032F6 8B44241C      mov eax, dword ptr [esp+1C]
:100032FA 3B08          cmp ecx, dword ptr [eax] <- ecx es el contador,
                                                    dword ptr [eax] = 10B
:100032FC 0F878D000000  ja 1000338F <- por aquí se sale del bucle cuando ecx = 10C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:100032F4(C)
|
:10003302 83FA08        cmp edx, 00000008
:10003305 7D51          jge 10003358

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000334F(C)
|
:10003307 85F6          test esi, esi
:10003309 7448          je 10003353
:1000330B 8A4D00        mov cl, byte ptr [ebp+00]
:1000330E 45            inc ebp
:1000330F 4E            dec esi
:10003310 80F920        cmp cl, 20    <- "marcas" especificas en la clave; marca 20
:10003313 7437          je 1000334C
:10003315 80F909        cmp cl, 09    <- "marca 09"
:10003318 7432          je 1000334C
:1000331A 80F92D        cmp cl, 2D    <- "marca 2D"
:1000331D 742D          je 1000334C
:1000331F 80F961        cmp cl, 61    <- "marca 61"
:10003322 7C08          jl 1000332C
:10003324 80F97A        cmp cl, 7A    <- "marca 7A"
:10003327 7F03          jg 1000332C
:10003329 80C1E0        add cl, E0

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:10003322(C), :10003327(C)
|
:1000332C 33C0          xor eax, eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000333C(U)
|
:1000332E 388810F90110  cmp byte ptr [eax+1001F910], cl
:10003334 7408          je 1000333E
:10003336 40            inc eax
:10003337 83F820        cmp eax, 00000020
:1000333A 7D42          jge 1000337E
:1000333C EBF0          jmp 1000332E

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

```
|:10003334(C)
|
:1000333E 83F820        cmp eax, 00000020
:10003341 7D3B          jge 1000337E
:10003343 8BCA          mov ecx, edx
:10003345 D3E0          shl eax, cl
:10003347 0BD8          or ebx, eax
:10003349 83C205        add edx, 00000005

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:10003313(C), :10003318(C), :1000331D(C)
|
:1000334C 83FA08        cmp edx, 00000008
:1000334F 7CB6          jl 10003307
:10003351 EB05          jmp 10003358

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10003309(C)
|
:10003353 83FA08        cmp edx, 00000008
:10003356 7C33          jl 1000338B

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:10003305(C), :10003351(U)
|
:10003358 85FF          test edi, edi
:1000335A 7407          je 10003363
:1000335C 881F          mov byte ptr [edi], bl
:1000335E 47            inc edi
:1000335F 897C2418      mov dword ptr [esp+18], edi

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000335A(C)
|
:10003363 8B4C2414      mov ecx, dword ptr [esp+14]
:10003367 33C0          xor eax, eax
:10003369 41            inc ecx
:1000336A 8AC7          mov al, bh
:1000336C 83EA08        sub edx, 00000008
:1000336F 894C2414      mov dword ptr [esp+14], ecx
:10003373 85F6          test esi, esi
:10003375 8BD8          mov ebx, eax
:10003377 7416          je 1000338F
:10003379 E970FFFFFF    jmp 100032EE <- Vuelve al inicio del bucle.

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:1000333A(C), :10003341(C)
```

Dentro del bucle va cargando cada elemento hexadecimal de la clave dentro de CL, y realiza varias comprobaciones de cada uno de los elementos con valores "marca", de tal forma que si estan presentes se fuerza la salida del bucle. Varias de estas "marcas" tienen el valor de permitir un uso indefinido del programa, aunque a menudo haga falta reiniciarlo o aparezcan molestas nags, como la de QuickClean de la imagen. Posiblemente estos saltos nos lleven por una región del programa que modifique el registro en el sentido de permitir un uso

indefinido, pero no investigué esta posibilidad por encontrarme ya muy cerca de la solución definitiva.



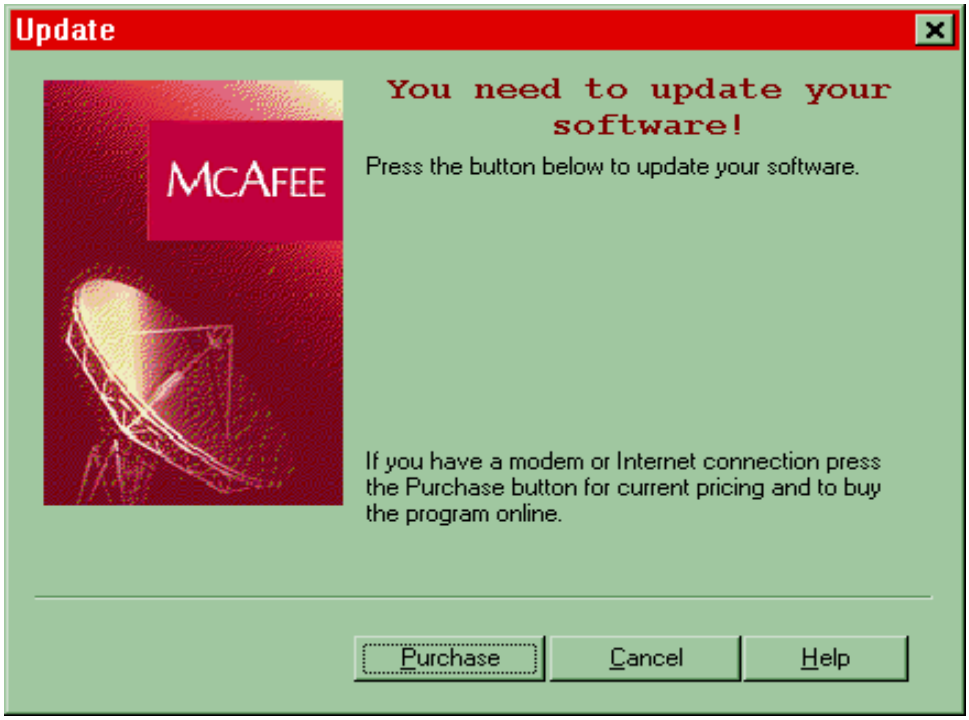
Hacia el inicio del bucle ECX tiene función de contador, y así sabemos cuantos ciclos realiza, conocimiento necesario para saber por donde sale y a donde se dirige.

Una vez fuera del bucle, siempre dentro de **Recapi**, que es una dll incluida dentro de la carpeta **C:\Archivos de programa\McAfee\McAfee Shared Components\Instant Updater**, llegamos a este interesante trozo de código:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000489A(U)
:1000489E 8BC8              mov ecx, eax
:100048A0 889C2420010000    mov byte ptr [esp+00000120], bl
:100048A7 8906              mov dword ptr [esi], eax
:100048A9 E802F1FFFF        call 100039B0
:100048AE 8B0E              mov ecx, dword ptr [esi]
:100048B0 8D442414          lea eax, dword ptr [esp+14]
:100048B4 50                push eax
:100048B5 E876F5FFFF        call 10003E30
:100048BA 85C0              test eax, eax <- ¿Permitir uso indefinido?
:100048BC 745B              je 10004919   <- No: salta fuera de la zona de "registro"
:100048BE 6824020000        push 00000224
:100048C3 E8D9F40000        call 10013DA1
:100048C8 83C404            add esp, 00000004
:100048CB 8944240C          mov dword ptr [esp+0C], eax
:100048CF 3BC3              cmp eax, ebx
:100048D1 C684242001000002 mov byte ptr [esp+00000120], 02
:100048D9 740E              je 100048E9
:100048DB 8D4C2414          lea ecx, dword ptr [esp+14]
:100048DF 51                push ecx
:100048E0 8BC8              mov ecx, eax
:100048E2 E859EEFFFF        call 10003740
```

```
:100048E7 EB02              jmp 100048EB

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:100048D9(C)
```

Nopear este salto vence el límite de una forma mucho más limpia que los anteriores, solo deja una ventana que nos insta a realizar un update y que tenemos que cancelar para continuar con el programa.

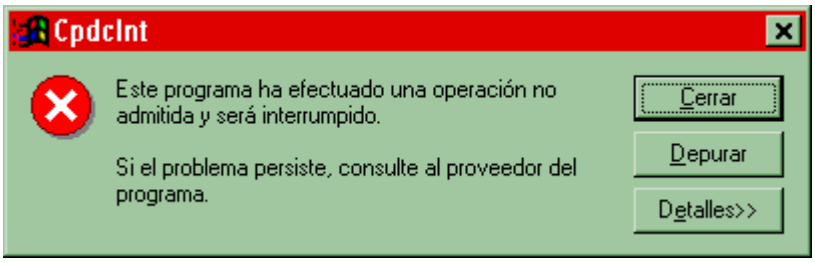


Traceando un poquito vemos que sale de:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000539D(C)
|
:10005366 33FF        xor edi, edi
:10005368 EB4E        jmp 100053B8
:1000536A 8B0E        mov ecx, dword ptr [esi]
:1000536C E8BFECFFFF  call 10004030
:10005371 85C0        test eax, eax
:10005373 8BCE        mov ecx, esi
:10005375 741F        je 10005396   <- Forzando este salto se evita la
:10005377 E8F4F6FFFF  call 10004A70 <- Nag
:1000537C 8BF8        mov edi, eax
:1000537E EB38        jmp 100053B8
:10005380 8B0E        mov ecx, dword ptr [esi]
:10005382 E8A9ECFFFF  call 10004030
:10005387 85C0        test eax, eax
:10005389 8BCE        mov ecx, esi
:1000538B 7409        je 10005396   <- Forzando este salto se evita
:1000538D E8DEF6FFFF  call 10004A70 <- de nuevo la nag
:10005392 8BF8        mov edi, eax
:10005394 EB22        jmp 100053B8
```

O sea, que cambiando estos **3 saltos en Recapi.dll** hemos conseguido una versión plenamente funcional de McAfee Firewall 3.0 de uso permanente y exenta de nags. Sin embargo aún queda un apartado que corregir.

Hemos evitado la comprobación del trial por que posiblemente entramos al programa por donde este evalúa si las librerías que posee están actualizadas o no, de ahí que nos apareciera esta nag. Posiblemente por esto algunos elementos necesarios para que el programa realice una actualización no se cargan o se cargan mal y como consecuencia al pulsar sobre "Check for a Firewall Update" aparece una molesta ventanita.



Esta ventanita puede localizarse fácilmente poniendo un Bpx Hmencpy y traceando un poquito. Vemos que sale de **Rupdate.dll** que está en la carpeta **C:\McAfee\McAfee Shared Components\Instant Updater**, igual que Recapi.dll.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:10001564(U), :100015A8(U), :100015C9(U), :100015E7(U), :10001605(U)
|:10001623(U)
|
:10001641 E8CA010000    call 10001810
:10001646 50            push eax

* Reference To: USER32.CreateDialogParamA, Ord:004Fh
|
:10001647 FF1538F30110  Call dword ptr [1001F338]
:1000164D 8BF0          mov esi, eax
```

Vemos que es un **CreateDialogParamA**. Viene referenciado de **1001564**, que a su vez puede evitarse cambiando un **JNZ**.

```
* Possible StringData Ref from Data Obj ->"Home"
|
:10001546 68F0340210  push 100234F0
:1000154B 57          push edi
:1000154C 33F6        xor esi, esi
:1000154E E85DDD0000  call 1000F2B0
:10001553 85C0        test eax, eax
:10001555 7512        jne 10001569  <- Cambiamos por jmp
:10001557 50          push eax
:10001558 8B442414    mov eax, dword ptr [esp+14]
:1000155C 68A06E0010  push 10006EA0
:10001561 50          push eax
:10001562 6A6F        push 0000006F
:10001564 E9D8000000  jmp 10001641  <- llama al DialogParamA
```

Con esto evitamos que el pulsar sobre update tenga efecto alguno. Hay que decir que en la versión "virgen" pulsar update abre una ventana en la que metemos nuestros datos para ser conectados al servidor de McAfee que nos registra como paso previo a bajar los ficheros de actualización.

Registrar un programa no tiene mucho sentido en una lista de cracks, así que esta pequeña amputación del programa no creo que sea echada en falta.

Tal vez pueda hacerse algo con el fichero de actualización bajado de internet por otros medios, como se hacia con la actualización del Norton Antivirus (lección 17), pero eso es ya otra guerra.

Muchos procesos se han obviado de este tute. Por ejemplo, si ponemos las características en la tabla de secciones a E0000020 y desensamblamos Recapi.dll, veremos interesantes String References como "Mode evaluation" o "Mode registered" y muchas otras, pero su valor es limitado ya que muchas son flags de depuración del programa (curiosamente incluye más elementos de depuración de lo habitual, como visteis cuando Sice hace el 12° BPX en RegOpenKeyExA y como veréis muchos más si traceais vosotros el programa por esa zona).

Con todo esto no es un programa muy difícil de crackear, los avances van cayendo como fruta madura hasta que termina cantando la gallina. Lo mejor de crackear el Mc Afee Firewall es que parcheando definitivamente Recapi.dll quedan también automáticamente crackeados el QuickClean y el VirusScan, aunque esta conclusión resulto un poquito ardua de demostrar.

Nuevas instalaciones de productos de McAfee "machacan" el archivo parcheado, por lo que debe guardarse en otra carpeta un Recapi.dll parcheado o fabricarse un parcheador.

mR gANDALF

PD: Mis sinceros agradecimientos a Ricardo por su ayuda y apoyo. Deseo que este tute le sirva para medir su labor de enseñanza del craking y sobre todo le deseo que obtenga de la vida todo lo bueno que se merece.

Un fuerte abrazo Ricardo, y espero que este tute sea el último de una serie, que lo será por estupendos motivos para mí.

LECCIÓN 60: OTRO COLABORADOR, "NOSEMAN"

INTRODUCCION: Aqui tenemos un nuevo colaborador, Noseman, que con sus 16 años ha hecho un buen tute sobre cómo quitar el vencimiento al programa **REAL PLAYER ONE** en su ultima versión y esta muy bien hecho, y funciona; así que agradecemos su colaboración que pasa a engrosar las lecciónes del curso.

MUCHAS GRACIAS NOSEMAN

Programa: Real Player One

Protección: No se, pero después de Agosto 1 de 2002 ya no se sigue ejecutando porque pide un update

Dificultad: Fácil

Herramientas: SoftICE 4.05.334, W32Dasm 8.93, HexWorkShop 3.11

Cracker: noseman

Fecha: 06/02/2002

Introducción: Después de mi primer día en la universidad y con las pupilas dilatadas y con la imagen en mi cabeza de que de ahora en adelante voy a tener que utilizar lentes, aquí estoy con mi primer "tutorial" y hasta sorprendido estoy con todo lo que he progresado en esto del cracking gracias a Ricardo, por supuesto. He seguido todas sus lecciónes menos la de los programas que sobrepasan las 15Mb (Que pereza bajarme eso).

El programa que aquí se trata es el famoso Real Player de Real Networks en su ultima versión que es **RealPlayerOne 6.0.10.446.**

Lo crackee por cuenta propia porque me disgustaba que el programita pidiera un update después del 1 de agosto de 2002 y no seguía funcionando más.

Desguaze: Empezando como siempre analizando el exe del Real Player con el File Inspector XL y el Language 2000, no muestran nada de ponerle atención puesto que no esta empaquetado ni nada por el estilo. Ahora miremos como funciona el programa antes del 1 de agosto.

Vemos que el programa no tiene ningún problema al arrancar y reproduce todo lo que uno le diga (los formatos que soporta). Ahora adelantemos el reloj al 1 de agosto.

Vemos que dice que esa versión del real player ha expirado, que te conectes a internet para bajar automáticamente la ultima versión, pero yo estoy muy contento con esta versión, me funciona perfectamente para lo cual esta diseñada y no quiero bajar otra (por pereza en mi caso, o por lo que quieran), que aparte de ser lo mismo pesa más.

Entonces, que hacemos???. Pues a una persona normal no le quedaría otra alternativa que updatear el programa, pero como nosotros sabemos algo de esto le encontraremos solución al problema. O no???

Abrimos el exe con el Peditor y vemos que **sections** en el **.data** esta en **C0000040**. Entonces ya sabemos que hacer, cambiarlo por **E0000020** para abrirlo con el w32dasm. Al abrirlo desensambla perfectamente, sin ningún error pero cuando vamos a las string data references solo vemos algo sospechoso que es un **"update"**; pero al darle ahí no nos lleva a ningún lugar interesante, entonces, como no estamos orientados habrá que utilizar al sice, no hay de otra.

Lo más lógico aquí seria un **getlocaltime**. Si traceamos con F12 vemos que para en demasiadas partes, unas en el kernel, otras en dlls del realplayer y muchas más.

Simplifiquemos las cosas con otro breakpoint, por ejemplo el hmemcpy, que al tracearlo con F12 no para en tantas partes. Traceamos hasta que salga el cartelito ese diciendo que actualicemos el programa y cuando le damos close vuelve al sice en esta dirección **60001E21** que, si vemos, no pertenece exactamente al exe del realplayer sino a un dll del mismo que es el **rpap3260.dll**.

Vemos que para en **test eax**, **eax** y encima hay un call (sospechoso no?). Volvamos al sice y ponemos un breakpoint en esta call haber si para.

Vemos que si para y si entramos al call, (F8) para tracearlo con calma, hasta que salga el aviso del RP y le damos close vemos que resulta en otro **test eax**, **eax** y encima un call (**60022AE1**).

Pongamos un bpx allí para ver si por casualidad para y, bingo!!!! Si paro, y si lo ejecutamos con F10 sale el aviso nefasto ese, o sea que por aquí debe estar la clave del asunto.

Entremos al call a ver que pasa. Nos lleva a un lugar que dice INVALID por todos lados, pero si volvemos a dar F10 nos lleva a nuestras hermosas sentencias. Pero si están perezosos (como yo) y no quieren analizar todo ese código tan largo volvemos al call del principio (**60022AE1**) y probamos a nopearlo. Al hacerlo sale error por todas partes y tenemos que reiniciar el ordenador. Entonces observamos que después de la call sigue un test eax, eax. ¿Que tal si la call la cambiamos por un xor eax, eax?, sí, si, funciona.

Procedemos a copiar la cadena para ir a buscarla al editor hexadecimal y tendríamos que buscar lo siguiente:

E8 9A 3C 00 00 85 C0 0F 85 B8 06 00 00

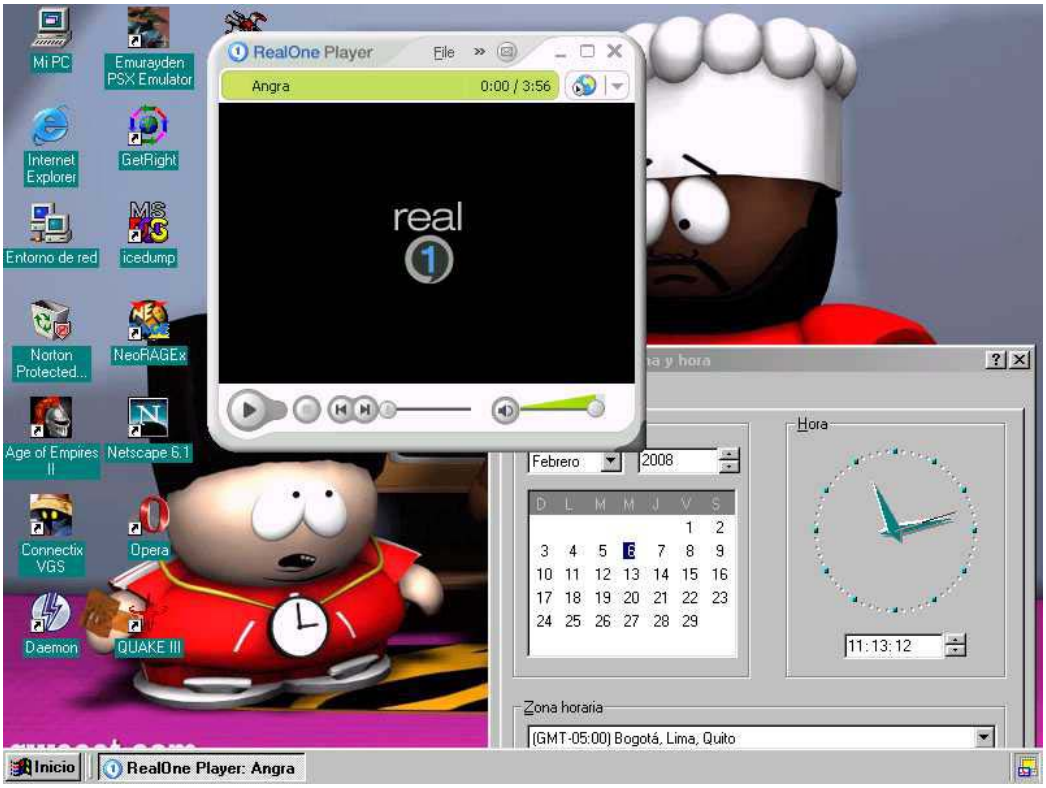
y reemplazar

E8 9A 3C 00 00 -->> que es el call

por esto

33 C0 90 90 90 -->> que es **xor eax**, **eax** y tres NOP’s “para rellenar”

Y listo, no hay que preocuparse más por el vencimiento del programa.



EL METODO ANTERIOR PODRA NO SER MUY ORTODOXO PERO SIRVIO PARA EL PROPÓSITO QUE ERA.

AGRADECIMIENTOS A RICARDO POR SU GRAN CURSO, Y DISCULPAS PARA MR. GANDALF POR HABER COPIADO EL ESQUEMA DE LAS LECCIÓNES DE ÉL PORQUE ES QUE, EN REALIDAD, NO SABIA COMO HACER EL MIO.

ATT:

NOSEMAN

P.D: ESTA MAL PARA MIS 16 AÑOS?

LECCIÓN 61: OTRA DE MI AMIGO GANDALF

INTRODUCCION DE RICARDO NARVAJA: mR gANDALF esta hecho una máquina de hacer tutes de calidad. Otra muestra de ello es este tute, que es nuestra lección 61, una vez más.

Le deseamos suerte en lo que esta esperando, el sabe.
GRACIAS GANDALF

Programa: Kitchendraw 4.0

Download: www.scitechsoft.com

Protección: Trial de tiempo. Carga de horas a través de un código

Dificultad: Fácil

Herramientas: SoftICE 4.0, W32Dasm 8.93, TechFacts 98, Advanced Registry Tracer 1.43, Hiew 6.76

Cracker: mR gANDALF

Fecha: 8/02/2002

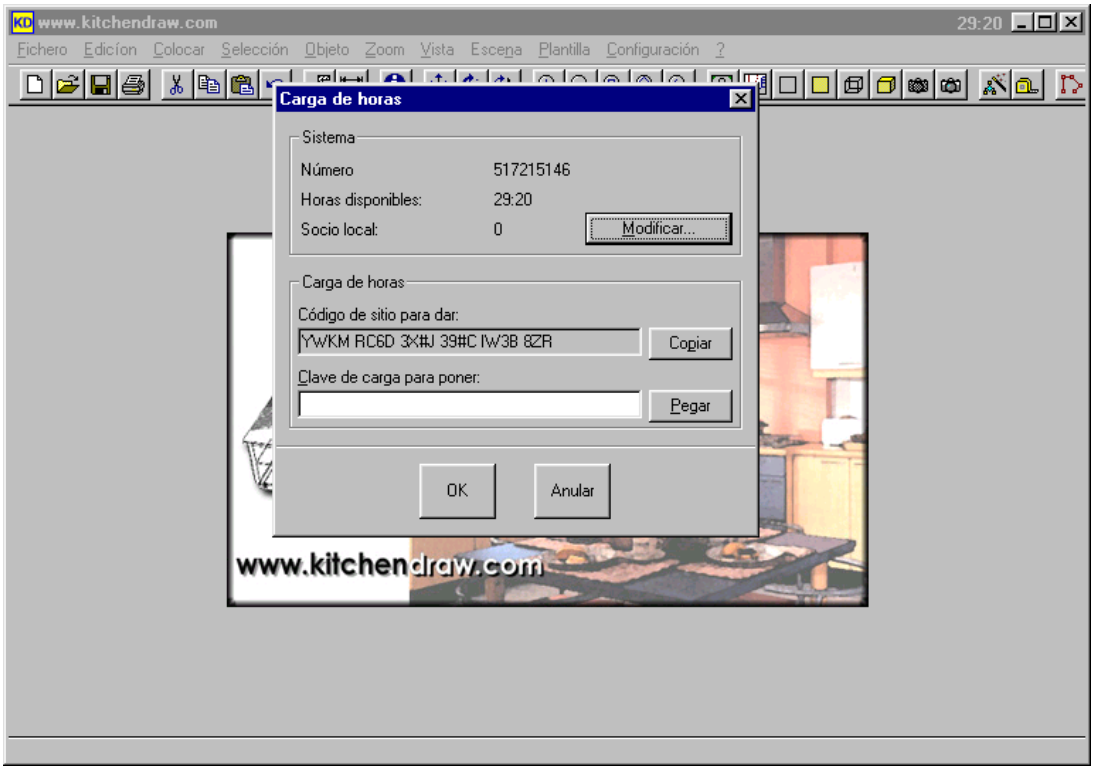
Introduccion: Mientras espero una llamada importante solo tengo dos opciones, volver a fumar o seguir "desguazando" bytes.

Nada calma mis dendritas como cepillarme una protección. Como no me llame hoy el Cifuentes me va a dar un "jamacuco".

En fín, que como anduvieron pidiendo el crack del kitchendraw este lo he instalado y la verdad que es facilón. El programa en sí se trata de un CAD de cocinas, no lo he probado mucho, todo sea dicho, pero skrapie sí y le gusta bastante.

Cuando lo instalamos y lo ejecutamos por primera vez nos aparece un contador de tiempo en la esquina sup-dcha que índica 29:55. Son 30 las horas que este trial nos dejará usarlo y cada vez que lo usemos nos restará al menos 5 minutos.

Tenemos una ventana que nos permite introducir un código para la recarga de tiempo que previamente habremos comprado por teléfono o por internet, algo parecido a las tarjetas de teléfono móvil.



Desguaze: El objetivo de este tutorial es averiguar como realiza el programa el conteo del tiempo y detenerlo. Este es el listado del TechFacts cuando el programa cuenta que nos quedan 29 horas y 20 minutos:

```
TechFacts 98 System Watch Report
02/08/02 04:17:00 pm

The following files were modified:(5)
c:\KD\Catalogs\catalogs.lst
c:\KD\Scenes\Key0202.log
c:\KD\Scenes\scenes.lst
c:\WINDOWS\APPLOG\APPLOG.ind
f:\kd\Space.ini

No changes made to INI file: C:\WINDOWS\WIN.INI

No changes made to INI file: C:\WINDOWS\SYSTEM.INI

Registry key values changed: (4)
HKEY_CLASSES_ROOT\G41512715
Value "517215146": binary data changed
HKEY_LOCAL_MACHINE\Software\CLASSES\G41512715
Value "517215146": binary data changed

Registry key values deleted: (1)
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\TechF
acts 98="F:\ARCHIVOS DE PROGRAMA\CRACK\HERRAMIENTAS\MONITORS -
UTILIDADES ESPIA\TEKFCT98\TEKFCT98.EXE"
```

Pues bien, siguiendo el "método cartesiano" de las lecciónes 58 y 59, es decir, haciendo uso intensivo de las aplicaciones ART (cuidado que cuelga el ordenador cuando corremos también el Sice, aunque usemos el FPLoader - aquí yo recomiendo la opción ASPAck/ASProtect más que la opción ART que incorpora el programa-) y TechFacts, llegamos a la conclusión de que es la clave **HKEY_LOCAL_MACHINE\Software\CLASSES\G41512715** y el archivo **catalogs.lst** los que guardan el tiempo que nos queda.

El archivo **Key0202.log** también debe ser importante, pues si lo borramos pasamos automáticamente a tiempo 0. Si guardamos una copia de la clave **G41512715** y del archivo **catalogs.lst** al contar el programa 29:30, los podremos utilizar para volver a esta cifra siempre que queramos.

En si, podemos considerar ya la protección vencida, pues nos basta un cargador *.bat para vencer la limitación. Pero nos sentimos curiosos y queremos llegar más lejos.

... el método es el método (Perogrullo S. XXI dc) ...

Un listado del APYSpy muestra desde donde se lee esta clave:

```
=============  Created by APIS32  v. 2.5  =============
70BE2CF8:RegOpenKeyExA()
70BE2CFE:RegOpenKeyExA = 2 (SHLWAPI.DLL)
70BE3003:RegOpenKeyExA()
70BE3009:RegOpenKeyExA = 2 (SHLWAPI.DLL)
7FCB1972:RegOpenKeyExA()
7FCB1978:RegOpenKeyExA = 0 (SHELL32.DLL)
7FCBD590:RegCreateKeyA(HANDLE:80000001,LPSTR:7FCBD618:"Software\Micros
oft\Windows\CurrentVersion\Explorer",LPDATA:7FD381B4)
7FCBD592:RegCreateKeyA = 0 (SHELL32.DLL)
7FCBD59D:RegCreateKeyA(HANDLE:80000002,LPSTR:7FCBD618:"Software\Micros
oft\Windows\CurrentVersion\Explorer",LPDATA:7FD381B0)
7FCBD59F:RegCreateKeyA = 0 (SHELL32.DLL)
7FE1F662:RegQueryValueExA()
7FE1F664:RegQueryValueExA = 2 (COMDLG32.DLL)
7FE1F67D:RegQueryValueExA()
7FE1F67F:RegQueryValueExA = 2 (COMDLG32.DLL)
7FE1F68C:RegCloseKey()
7FE1F692:RegCloseKey = 0 (COMDLG32.DLL)
00487210:RegCreateKeyA(HANDLE:80000002,LPSTR:0043DC6F:"Software\Laudri
n\InSitu",LPDATA:0085FB4C)
00487215:RegCreateKeyA = 0 (DV.DLL)
004871B5:RegSetValueExA(HANDLE:C49FC390,LPSTR:0043DC87:"Busy",DWORD:00
000000,DWORD:00000004,LPDATA:0085FB70,DWORD:00000004)
004871BA:RegSetValueExA = 0 (DV.DLL)
00487234:RegCloseKey()
00487239:RegCloseKey = 0 (DV.DLL)
004871CF:RegOpenKeyA(HANDLE:80000000,LPSTR:0085F37C:"G41512715",LPDATA
:0085F32C)
004871D4:RegOpenKeyA = 0 (DV.DLL)
00487198:RegQueryValueExA()
0048719D:RegQueryValueExA = 0 (DV.DLL)
004871F5:RegCloseKey()
004871FA:RegCloseKey = 0 (DV.DLL)
```

```
004871CF:RegOpenKeyA(HANDLE:80000000,LPSTR:0085FA94:"G41512715",LPDATA
:0085FA44)
004871D4:RegOpenKeyA = 0 (DV.DLL)
00487198:RegQueryValueExA()
0048719D:RegQueryValueExA = 0 (DV.DLL)
004871F5:RegCloseKey()
004871FA:RegCloseKey = 0 (DV.DLL)
004871CF:RegOpenKeyA(HANDLE:80000000,LPSTR:0085FA94:"G41512715",LPDATA
:0085FA44)
004871D4:RegOpenKeyA = 0 (DV.DLL)
00487198:RegQueryValueExA()
0048719D:RegQueryValueExA = 0 (DV.DLL)
004871F5:RegCloseKey()
004871FA:RegCloseKey = 0 (DV.DLL)
00487210:RegCreateKeyA(HANDLE:80000000,LPSTR:0085FABC:"G41512715",LPDA
TA:0085FA4C)
00487215:RegCreateKeyA = 0 (DV.DLL)
004871B5:RegSetValueExA(HANDLE:C49FC390,LPSTR:0085FA94:"517215146",DWO
RD:00000000,DWORD:00000003,LPDATA:0085FA74,DWORD:00000020)
004871BA:RegSetValueExA = 0 (DV.DLL)
00487234:RegCloseKey()
00487239:RegCloseKey = 0 (DV.DLL)
004871CF:RegOpenKeyA(HANDLE:80000000,LPSTR:0085F998:"G41512715",LPDATA
:0085F948)
004871D4:RegOpenKeyA = 0 (DV.DLL)
00487198:RegQueryValueExA()
0048719D:RegQueryValueExA = 0 (DV.DLL)
004871F5:RegCloseKey()
004871FA:RegCloseKey = 0 (DV.DLL)
004871CF:RegOpenKeyA(HANDLE:80000000,LPSTR:0085F820:"G41512715",LPDATA
:0085F7D0)
004871D4:RegOpenKeyA = 0 (DV.DLL)
00487198:RegQueryValueExA()
0048719D:RegQueryValueExA = 0 (DV.DLL)
004871F5:RegCloseKey()
004871FA:RegCloseKey = 0 (DV.DLL)
```

Parece que es en **004871CF** donde se abre la clave y en **0048719D** donde se lee. Sí ponemos un Bpx en 0048719D y traceamos un poquito (tratad bien a F10 que tiene que durarnos para otros tutes) llegamos a este trocito de código perteneciente a **DV.dll**:

```
:004E820D E8E4970000  call 004F19F6
:004E8212 83C408      add esp, 00000008
:004E8215 8B45E4      mov eax, dword ptr [ebp-1C]
:004E8218 8B55EC      mov edx, dword ptr [ebp-14]
:004E821B 03C2        add eax, edx
:004E821D 8B4DE8      mov ecx, dword ptr [ebp-18]
:004E8220 8B55F0      mov edx, dword ptr [ebp-10]
:004E8223 03CA        add ecx, edx          <- ecx=N° de usos gastados
:004E8225 2BC1        sub eax, ecx          <- eax=N° usos restantes
:004E8227 8BE5        mov esp, ebp
:004E8229 5D          pop ebp
:004E822A C3          ret
```

Imagínense las posibilidades que se nos ofrecen... por cierto, entren en **DV.dll** poniendo un **bpx en hmencpy** antes de pulsar OK en la ventana

de introducción de códigos de recarga y tengan en cuenta que en el Sice la dirección es **558225** en vez de **004E8225.**

El remate final lo dejo a su gusto, crackers.

mR gANDALF

PD: Por 2, que me llame el Cifu pronto...

LECCIÓN 61bis: REMATANDO LA FAENA

Programa: Kitchendraw 4.0

Download: http://www.KITCHENDRAW.COM/ESP

Objetivo: Habilitar la "producción de imágenes fotorealistas"

Dificultad: Si yo lo he hecho, facilísimo

Herramientas: TRW2K (o Softice), W32Dasm 8.93, Editor Hexa (p.ej.: Ultraedit), R!SC's Process Patcher v1.5.1

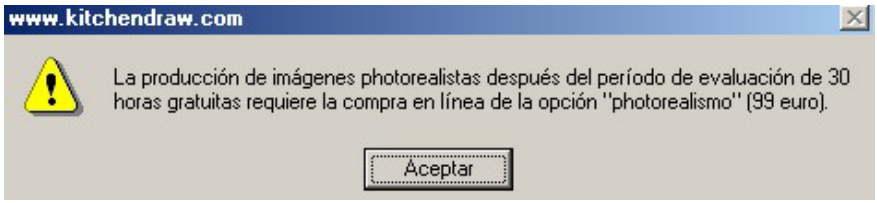
Cracker: Scrapie

Fecha: 15/02/2002

Introducción: Después de la Lección 61, hice el crack (gracias a mR gANDALF) para acabar con la limitación de tiempo de este programa, y como no lo uso para nada, creí que ya estaba totalmente crackeado. Pero, debido al aburrimiento, un día me puse a enredar con él y descubrí que la opción de "producción de imágenes fotorealistas" no estaba habilitada, y sirve para poder ver el diseño de la cocina en una especie de realidad virtual aunque un poco más triste (yo con mi tarjeta de video de 8Mb casi ni lo veo).

Pero ¿Cómo pueden cobrarte 3 Euros por hora y además exigir 99 Euros más?. ¡¡Increíble!! ¡¡¡Tenemos que reventarlo totalmente, se lo merece!!! (Aunque exclusivamente con fines educativos, evidentemente)

Re-Desguace: ¡Al ataqueee! Ejecutamos el programa y abrimos el proyecto de ejemplo que adjunta. Pulsamos sobre un icono que es como una cámara fotográfica y nos aparece este mensaje:



Desensamblamos con el W32Dasm y en String Referentes no veo nada que se le parezca (aquí hablo en 1ª persona a propósito, por motivos que veremos más abajo). Es hora de recurrir al TRW2K...

Antes de darle, otra vez, a  probamos con **bpx messagebox** y **messageboxexa**, pero en ninguno de los dos salta el "debugger" al aparecer la ventana. Como casi nunca falla lo intentaremos esta vez con **bpx hmemcpy** y... nada tampoco funciona. Pensemos un poco... el programa tiene que comprobar si hemos comprado el producto o no, y, o accede al registro (vimos en la gran primera parte de esta lección, que se cambian algunas "cosas") o lo comprueba de algún archivo escondido en las entrañas de nuestro PC.

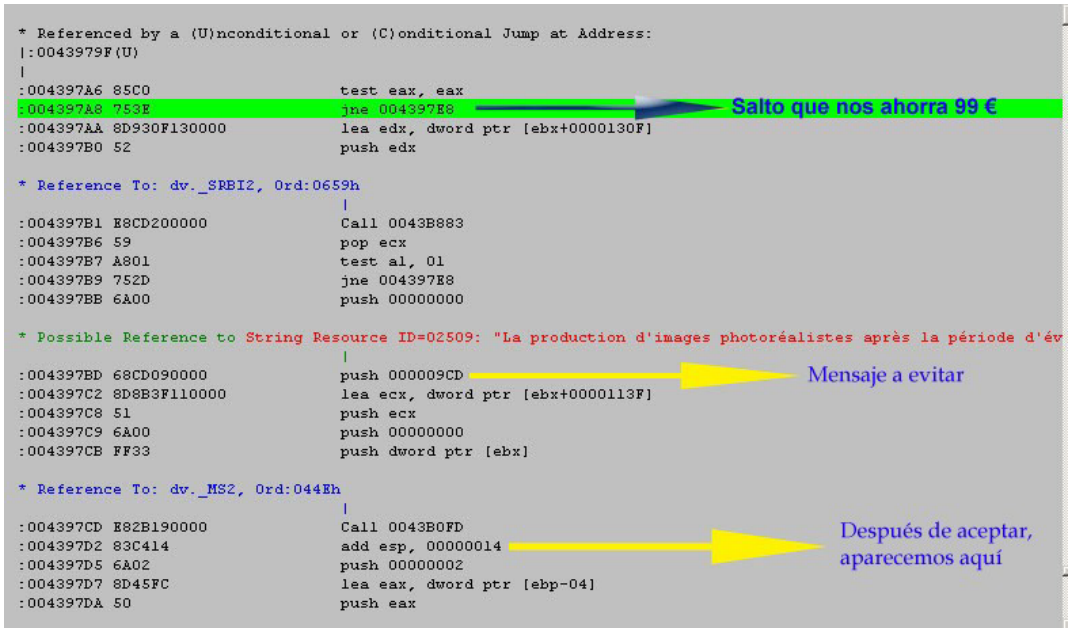
Lo intentaremos con el registro, poniendo **bpx RegQueryValue** (nunca lo había usado) y procedemos...

(Técnica de La Marcha Atrás)

¡Si! Esta vez ha saltado el TRW, aparece en el Kernell, quitamos el bp (bc *) y pulsamos F12 hasta que aparece el "molesto cartelito".

Aceptamos y vuelve a saltar el TRW, esta vez aparece en User; clavamos el dedo en F10 (cuidado los que usen Softice) hasta que volvamos al código del programa (KD).

Miramos donde hemos ido a parar y estamos en **004397D2.** Usando la técnica de mirar lo que hay encima, nos pueden parecer interesantes algunas Calls y algunos saltos cercanos, pero como la zona "conflictiva" ya la hemos localizado, para verlo todo más cómodos volvemos al W32Dasm y buscamos 004397D2:



Ohh!! El mismo mensaje de "chico malo" (pero en francés) está ahí y en String Referentes antes no había visto nada, lo compruebo y está de las primeras. ¡Dolorrrrrrrr! Pero como aquí lo que intentamos es aprender no importa, porque gracias a no verlo he usado un bpx que hasta ahora no había empleado, y puede sernos útil para crackear programas donde no nos lo ponen tan fácil. Seguimos.

La imagen lo aclara todo, los saltos que evitan el mensaje señalan hacia la misma dirección o sea que parcheando cualquiera de ellos funcionará igual (esto lo sé porque lo he comprobado con el debugger, jeje).

Con un editor hexadecimal cambiamos p. ej. el primer salto por **74** o **EB** (para que siempre salte), y ejecutamos el programa... Oh!!! Me estoy empezando a enfadar con los programadores de este "engendro".

El programa no funciona y da error, debe tener algún tipo de protección contra modificaciones en su código, pero ¿siendo tan fácil de crackear se molestan poniendo esto?...

Tengo que borrarlo e instalarlo otra vez porque aunque hice una copia del original, esta también ha dejado de funcionar.

La solución sería parchearlo en memoria: crearemos un loader con el R!SC's Process Patcher. Este es el script:

```
; KitchenDraw 4 (Produccion de imagen fotorealista)99 Euros menos ;-)
; Loader para Poder usar la Produccion de imagen fotorealista
; Esta opción sirve para poder ver el diseño en 3D con movimiento

F=kd.EXE:                          ; VICTIMA
O=Loader_KitchenDraw.exe:          ; ASESINO
P=4397a8/75/74:                    ; Salta siempre, por favor
$ ;end of script                   ; Fin
```

Lo probamos y funciona. ¡Lo hemos reventado, y además se lo merece! (Doble alegría)

Scrapie

Saludos a todos los miembros de la lista.

PD: Gracias a Ricardo y mR gANDALf por sus fabulosas lecciónes y perdón a mR gANDALf por copiar (también) la plantilla que usa para hacer sus magníficos tutes.

PD2: Este es mi primer tute, y no hace mucho que me incorporé a la lista por lo que pido perdón si hay algún error, ya que me doy cuenta cuando leo las lecciónes de Ricardo y los demás, de lo poco que sé y de lo mucho que me queda por aprender.

LECCIÓN 62: EASY PDF 1.6.1

Programa: EasyPDF 1.6.1

Download: www.visagesoft.com

Protección: VisualProtect 1.1. Trial de 20 días. La versión demo imprime un texto y un logotipo sobre el documento creado

Dificultad: Mediana

Herramientas: SoftICE 4.0, Dede 2.51, W32Dasm 8.93, Hiew 6.76, Princess Sandy Patcher 1.0

Cracker: mR gANDALF

Fecha: 16/02/2002

Introducción: El Cifuentes no trajo buenas noticias. Pero bueno, que se le va hacer. En este, mi octavo tutorial, quiero atraer su atención sobre un bonito programa llamado **EasyPDF**. Este programa sirve para convertir nuestros textos al formato PDF, pesa mucho menos que el Adobe Acrobat 4.0 y es muy sencillo de manejar.

Tiene una interesante protección que consta de dos elementos. Por un lado el Visual Protect, compresor de la misma casa que el EasyPDF y que ya fue diseccionado en un más que interesante tutorial de Kuato Thor que les recomiendo encarecidamente que lean http://karpoff.topcities.com/tutoriales/archivos/visualp k.htm. Fue Skrapie quién me llamo la atención sobre este tutorial.

Por otra parte, la versión demo realiza una especie de sobreimpresión del logotipo de visagesoft sobre nuestro documento y además añade en la cabecera unas palabritas en estruendoso color rojo animándonos a registrarnos. Todo un detallito para los que buscaban como animar sus textos.

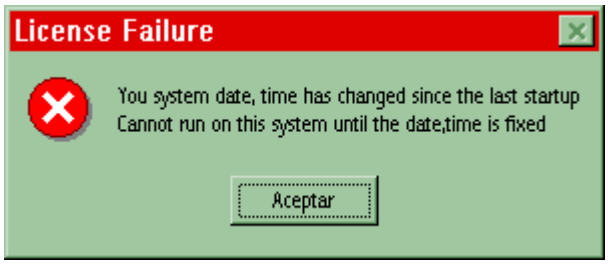
Para registrarnos debemos ponernos en contacto con la compañía que suponemos nos remite un key file con el que se solucionan estos inconvenientes. Pero nosotros vamos a ver una ruta alternativa a este key file que resulta mucho más ... "interesante".

Desguaze: Si echamos un vistazo con FileINspector veremos que es Visual Protect 1.1. ¿Y que demonio es esto?. Pues es un sistema de protección que consiste en una nag inicial que nos informa de que estamos en versión demo, nos dice los días que nos faltan y nos muestra un botón Buy para registrarnos a través de Internet, un botón OK para continuar y un botón QUIT para salir. Además, esta protección incluye la compresión del ejecutable y dll`s.

Todo este meollo se realiza desde **vp.dll** que tiene un call que llama a la nag y vuelve con un número en EAX que representa el botón pulsado. Si cambiamos este call por un mov eax,1 el programa no llama a la nag, se comporta siempre como si hubiésemos pulsado Ok, y además tiene el efecto de que no reconoce el vencimiento del trial. Este cambio puede realizarse con el Princess Sandy Loader de Eminence (protools).

... algunos restos del Visual Protect ...

Hasta aquí un breve resumen del tute de Kuato Thor. Pero si prueban a retrasar el reloj un mes, p.ej, verán que el programa reconoce que le han toqueteado el reloj, ya que seguramente guarde la fecha de instalación en el registro, y nos muestra un molesto cartelito.



Este cartelito nos aparece aunque hallamos cargado el programa con un loader evitando el call anterior. De hecho aparece antes de ese Call y es un messageboxexa. Traceando un poquito y cambiando alternativamente la fecha del sistema, vemos que:

```
:017C6581 8B151CB97C01  mov edx, dword ptr [017CB91C] <- Flag
:017C6587 3B02          cmp eax, dword ptr [edx]      <- EAX=Flag?
:017C6589 0F849C090000  je 017C6F2B                   <- Ir al cartel
:017C658F A164BC7C01    mov eax, dword ptr [017CBC64]
:017C6594 8B00          mov eax, dword ptr [eax]
:017C6596 8B1568BB7C01  mov edx, dword ptr [017CBB68]
:017C659C 3B02          cmp eax, dword ptr [edx]
:017C659E 0F84D8080000  je 017C6E7C
:017C65A4 A164BC7C01    mov eax, dword ptr [017CBC64]
:017C65A9 8B00          mov eax, dword ptr [eax]
:017C65AB 8B1504BC7C01  mov edx, dword ptr [017CBC04]
:017C65B1 3B02          cmp eax, dword ptr [edx]
:017C65B3 0F84CB090000  je 017C6F84
:017C65B9 A164BC7C01    mov eax, dword ptr [017CBC64]
:017C65BE 8B00          mov eax, dword ptr [eax]
:017C65C0 8B15BCBA7C01  mov edx, dword ptr [017CBABC] <- Flag
:017C65C6 3B02          cmp eax, dword ptr [edx]      <- EAX=Flag?
:017C65C8 7432          je 017C65FC                   <- Evita cartel
:017C65CA A164BC7C01    mov eax, dword ptr [017CBC64]
:017C65CF 8B00          mov eax, dword ptr [eax]
:017C65D1 8B151CBB7C01  mov edx, dword ptr [017CBB1C]
:017C65D7 3B02          cmp eax, dword ptr [edx]
```

Similares saltos presenta en **17C66A6** y **17C6793**. "Corrigiéndolos" evitamos que si tenemos que mover el reloj para atrás por alguna razón nos salga el cartelito.

... la segunda parte: ese molesto logotipo en nuestros PDF`s ...

Fíjense que feo cartel nos parece utilizando la versión demo.

Fíjense además que en la cabecera aparece una frase de 3 líneas sobre las ventajas de registrarse (eliminación de cartelito incluida, claro esta). Indudablemente, durante el proceso de creación de nuestro PDF se ha sobreimpreso "esto".

Antes de apretar Ctrl-D como posesos pensemos un poco. Efectivamente se trata de Delphi y en estos casos olvídense de hacer nada sin el Dede. Tracear a lo bruto solo nos lleva a desgastar F10 y volvernos locos. Pero como Dede no admite empacados lo mejor es dumpear con ProcDump selecciónando de la ventana de procesos activos easypdf.exe (clic derecho y selecciónen dump full).

El dumpeado original lo editan con PE Editor, hacen dumpfix (lección 34) arreglan la tabla de secciones para que permita usar el Wdasm (E0000020) y lo cargan en el Dede. Aquí vallan a Procedures y elijan Main. Échenle un vistazo, sin duda se trata del procedimiento que crea la pagina principal, la que contiene el botoncito que al apretarlo nos crea nuestro PDF.

Verán que hay uno que se llama DoPDFExecute, lo cual resulta más que sugerente. Si ponemos un Bpx al comienzo del procedure, o sea, en

68b144, veremos que nada más tocar el botón caemos en el Sice. Aquí el procedimiento lógico es ponerse a mirar saltos condicionales y probar a cambiarlos, a ver que pasa, igual que en el Winomega y en tantos otros. Ya destaca desde el principio la escasez de estos saltos, pero cuando nos ponemos a cambiarlos vemos que efectivamente así no vamos a ningún lado.

Probemos a buscar algún elemento de la cadena que sobreimprime, por ejemplo **"evaluation",** con el comando **S 0 l FFFFFFFF ´evaluátion´.** Ya el primer hallazgo es significativo y no hay que seguir buscando más, caemos en plena cadena. Ponemos un bpmd en un punto de la misma, ya que con los bpr se cuelga la maquina (creo que forma parte de la protección), y le damos a F5. Pulsamos F12 para volver a **vp.dll** y terminamos cayendo en un sitio como este:

```
:0068D04D E8EAD2FAFF    call 0063A33C  <- Escribe texto de evaluación
:0068D052 6800000040    push 40000000
:0068D057 6834D26800    push 0068D234
:0068D05C 8B45F4        mov eax, dword ptr [ebp-0C]
:0068D05F 50            push eax
:0068D060 E863D2FAFF    call 0063A2C8  <- Pone rojo el Logo
:0068D065 6800004842    push 42480000
:0068D06A 8B45F0        mov eax, dword ptr [ebp-10]
:0068D06D 50            push eax
:0068D06E 8B45F4        mov eax, dword ptr [ebp-0C]
:0068D071 50            push eax
:0068D072 E8B5D2FAFF    call 0063A32C  <- Da tamaño grande al logo
:0068D077 6A00          push 00000000
:0068D079 6A00          push 00000000
:0068D07B 6A00          push 00000000
:0068D07D 680000803F    push 3F800000
:0068D082 6830D16800    push 0068D130
:0068D087 6844D26800    push 0068D244
:0068D08C 8B45F4        mov eax, dword ptr [ebp-0C]
:0068D08F 50            push eax
:0068D090 E8EFD2FAFF    call 0063A384  <- etc…
:0068D095 6A00          push 00000000
:0068D097 68C9C8483F    push 3F48C8C9
:0068D09C 68C9C8483F    push 3F48C8C9
:0068D0A1 68C9C8483F    push 3F48C8C9
:0068D0A6 6830D16800    push 0068D130
:0068D0AB 6834D16800    push 0068D134
:0068D0B0 8B45F4        mov eax, dword ptr [ebp-0C]
:0068D0B3 50            push eax
:0068D0B4 E8CBD2FAFF    call 0063A384
:0068D0B9 683CD16800    push 0068D13C
:0068D0BE 6840D16800    push 0068D140
:0068D0C3 DD45E8        fld qword ptr [ebp-18]
:0068D0C6 D8354CD26800  fdiv dword ptr [0068D24C]
:0068D0CC 83C4FC        add esp, FFFFFFFC
:0068D0CF D91C24        fstp dword ptr [esp]
:0068D0D2 9B            wait
:0068D0D3 DD45E0        fld qword ptr [ebp-20]
:0068D0D6 83C4FC        add esp, FFFFFFFC
:0068D0D9 D91C24        fstp dword ptr [esp]
:0068D0DC 9B            wait
:0068D0DD DD45E8        fld qword ptr [ebp-18]
:0068D0E0 D83550D26800  fdiv dword ptr [0068D250]
```

```
:0068D0E6 83C4FC        add esp, FFFFFFFC
:0068D0E9 D91C24        fstp dword ptr [esp]
:0068D0EC 9B            wait
:0068D0ED 6A00          push 00000000
:0068D0EF 6854D26800    push 0068D254
:0068D0F4 8B45F4        mov eax, dword ptr [ebp-0C]
:0068D0F7 50            push eax
:0068D0F8 E83FD2FAFF    call 0063A33C
:0068D0FD 6A00          push 00000000
:0068D0FF 6834D26800    push 0068D234
:0068D104 8B45F4        mov eax, dword ptr [ebp-0C]
:0068D107 50            push eax
:0068D108 E8BBD1FAFF    call 0063A2C8
:0068D10D 8B45F4        mov eax, dword ptr [ebp-0C]
:0068D110 E87FD1FAFF    call 0063A294

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0068CF56(C) <- Aqui viene el salto que evita el "marcado" del texto.
|
:0068D115 8BE5          mov esp, ebp
:0068D117 5D            pop ebp
:0068D118 C21000        ret 0010
```

Con esto queda limpia la protección. Puede hacerse un bonito loader con el Princess Sandy patcher.

Hasta la proxima... o nó, quien sabe?

mR gANDALF

PD: Agradecimientos en especial para Ricardo, a quién estimo más cada día.

LECCIÓN 63: BUSTOUT! VERSIÓN 2.0

INTRODUCCION DE RICARDO NARVAJA: A pesar de que estamos de lleno metidos con el nuevo curso, nos siguen llegando colaboraciones de amigos, esta es de un amigo (JIKI) de 15 años, que hizo su primer tute y se lo agradecemos mucho, lo colocamos como LECCIÓN 63 del curso viejo. **GRANDE AMIGO JIKI.**

Programa: Bustout! Versión 2.0
Tipo: Juego
Protección: Solo te deja jugar 3 pistas
Herramientas: TRW; w32dasm; hex workshop; language 2000; ProcDump
Cracker (si se me puede llamar así): JIKI

Introducción...

Ya que todos están escribiendo algún tute es hora de que yo haga uno. Este juego es bastante viejo, pero a mí me encanta y seguro que a muchos de ustedes también les va a gustar. Es un juego que te ofrece la posibilidad de registrarte metiendo un serial, pero ese trabajo se lo dejaremos a un cracker con más experiencia.

Pos parto...

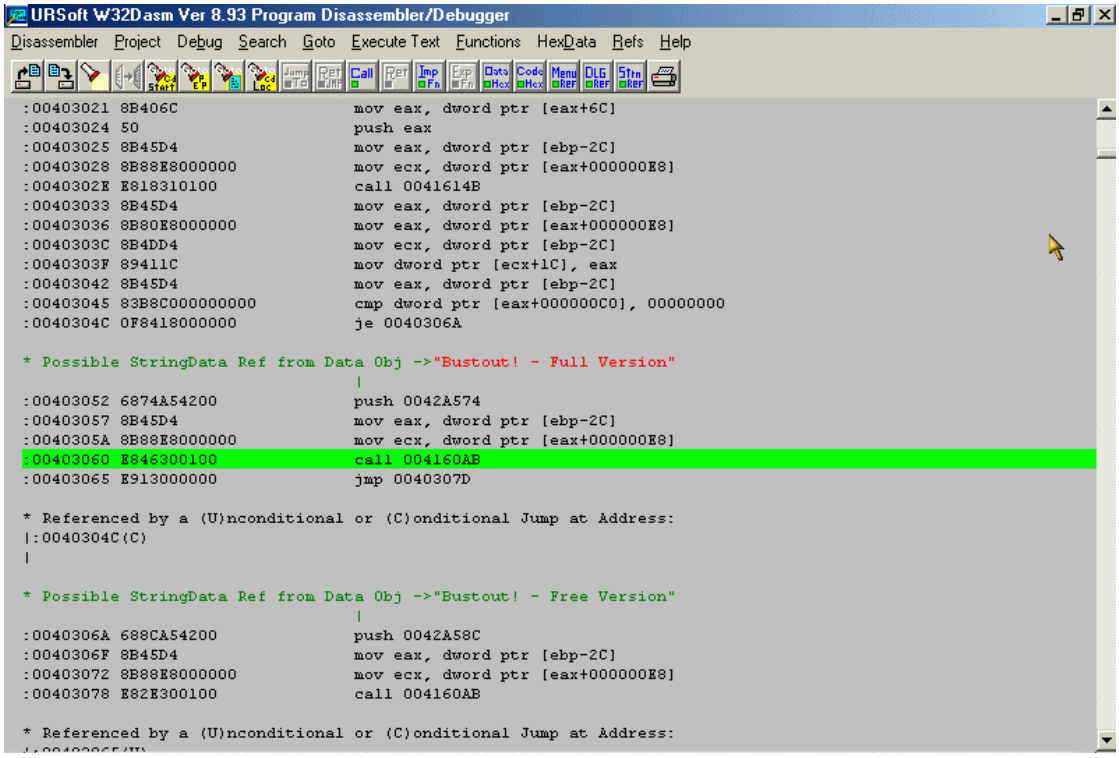
Abrimos el juego con el Language 2000 y vemos que no está comprimido y está hecho en Visual C++. Paso siguiente, abrimos el ProcDump, vamos a PE EDITOR, seleccionamos el nombre del juego y hacemos clic en abrir. Ni bien apretamos abrir nos saldrá una ventana en la cual hacemos un clic donde dice SECTIONS. Luego procedemos a hacer clic derecho en **.data** y ponemos edit section y en donde dice sections characteristics cambiamos de **C0000040** a **E0000020**. Hacemos lo mismo con el **.idata**.

Al ataque...

Abrimos el W32Dasm y desensamblamos el juego. Una vez abierto vamos a String Data Referentes y buscamos **Free versión**. Miren la sorpresa que debajo de Free versión está **Full versión**.

Bueno, ahí vamos (a full versión). Hacemos 2 clicks y vemos que para llegar ahí nos manda desde un salto condicional. Anotamos esa dirección en un papel. Volvemos a String data referentes y hacemos de nuevo 2 clicks en Full versión.

Ahora vemos que es desde otra dirección de donde salta; la anotamos y volvemos a hacer lo mismo solo que esta vez vemos otra cosa:



Ahí podemos ver que en **40304c** hay un salto condicional que si nos registramos nos manda a Full versión, y si no a free versión.

Anotamos eso en un papel y listo. Ahora vamos al TRW y ponemos bpx en todos los saltos que teníamos (40174f, 4018c5 y 40304c). Cargamos y vemos en cuál para.

Paró en el tercero (40304c), o sea que vamos al Hex Workshop y cambiamos ese **0f8418000000** por **909090909090**.

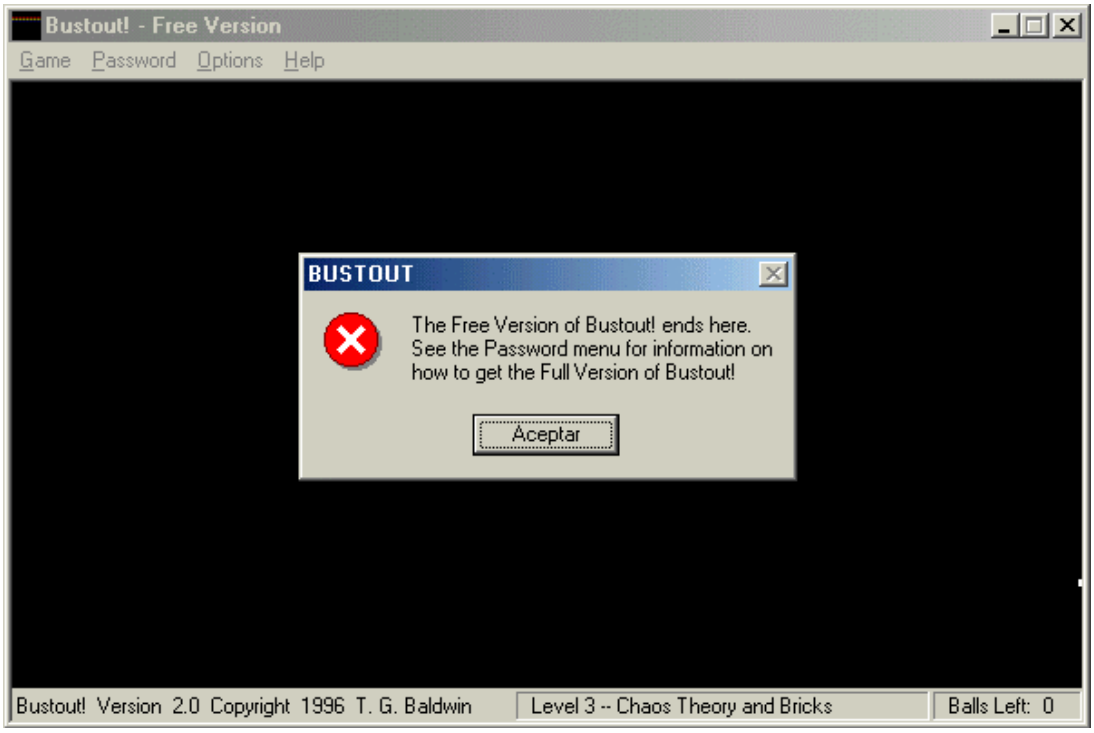
Entramos en el Bustout! Y bingo!!! Full Versión. Bueno, jugamos un rato, y justo nuestra madre nos llama para comer ponemos pausa (click derecho) y vemos que ese Full Versión pasó a **Free Versión (paused)** ¿qué pasa acá?. Bueno, que no cunda el pánico.

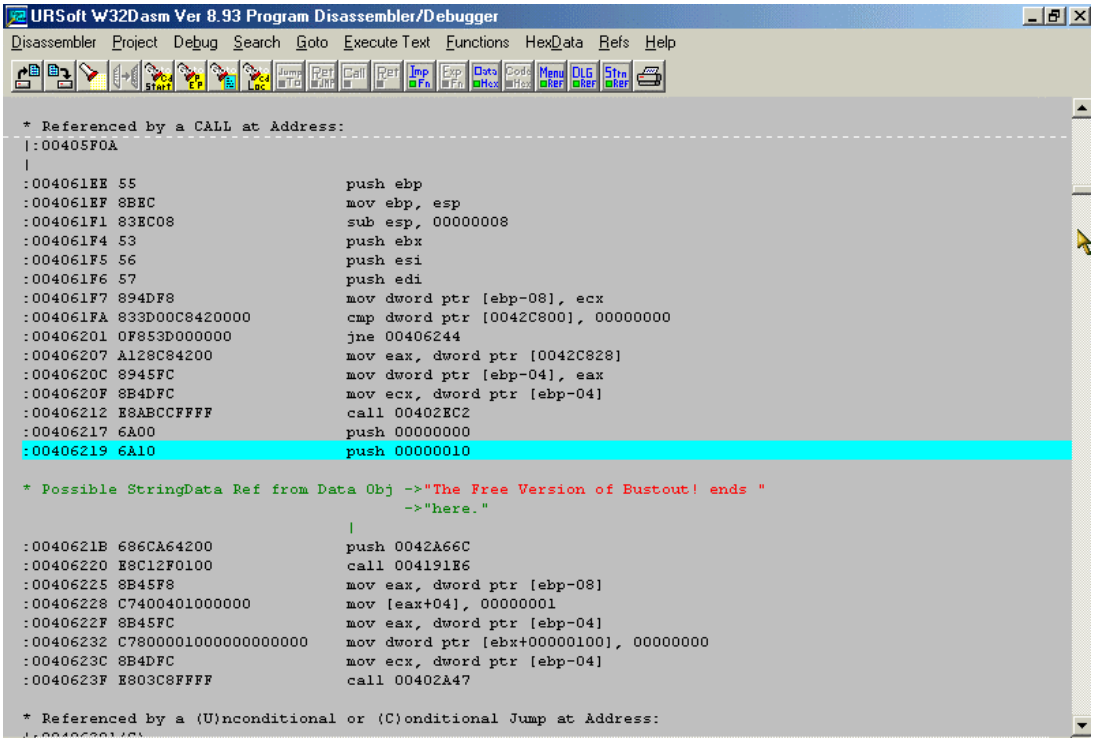
Cuando terminamos de comer volvemos y echamos un vistazo en las String Data Referentes a ver qué pasa con Free Versión (Paused). Vemos que es llamada desde **40176e**. Anotamos esa dirección, vamos al Hex Workshop y cambiamos ese **0f8412000000** por **909090909090**.

Abrimos el juego ponemos pausa y... NO PASA NADA!!! SIGUE EN FULL VERSION!!!, sacamos la pausa para seguir jugando y... #$%"& SE CAMBIO A FREE VERSION!!!.

Bueno, como yo soy muy haragán decidí buscarle otra salida. Empecé a jugar el juego y al pasar la tercera pista apareció un cartel que decía algo parecido a **"The Free versión of Bustout! Ends here"** y algunas cosas más.

Bueno, vamos al W32Dasm a buscar eso. Miramos un poco y casi al final lo encontramos. Hacemos 2 clicks y caemos aquí:



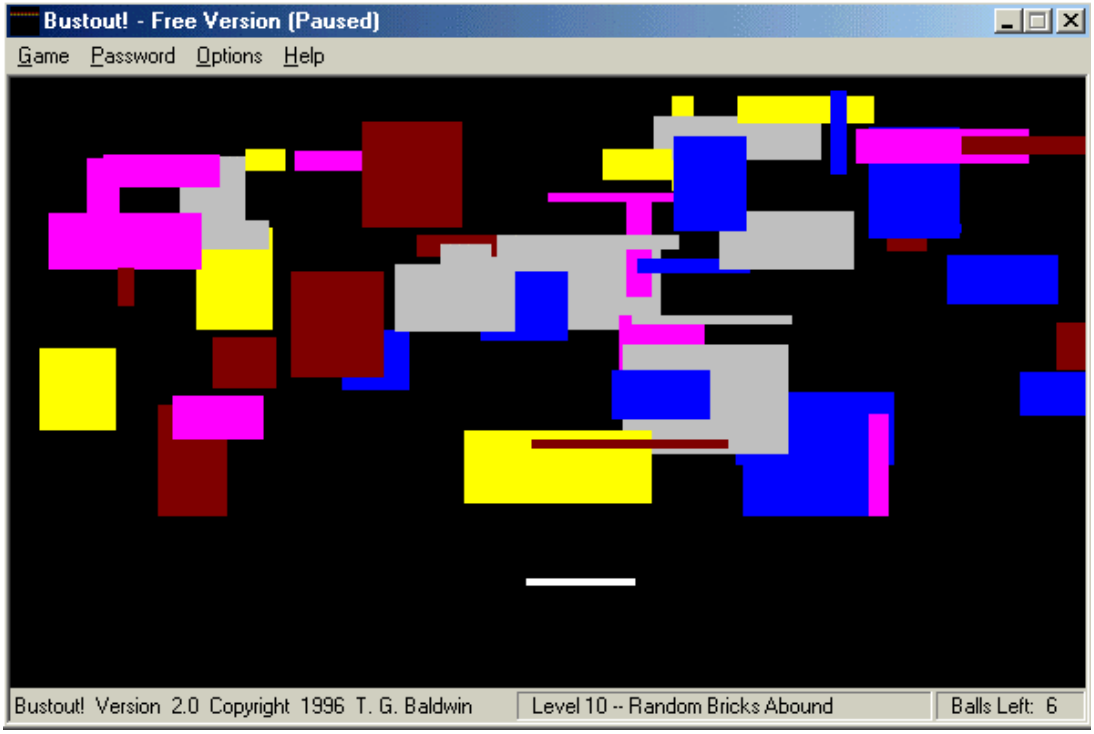


Vemos que ese cartel es llamado desde una Call en la dirección **405f0a**. Vamos a Goto/Goto code location y tipeamos la dirección. Así que, lo que tendríamos que hacer sería anular esa Call.

Tenemos que cambiar **E8df020000** por **9090909090** y listo, ya podemos jugarle a todas las pistas sin siquiera estar en "Full Versión".

¿Vieron que siempre hay otra salida?

(Esta es una foto del último level)



PD: Un enorme abrazo a Ricardo por confiar en mí aunque en un principio le hice muchas preguntas (¿Y QUE QUIEREN? SOLO TENGO 15 AÑOS).

¡¡¡AGUANTE ARGENTINA!!!

LECCIÓN 64: CALENDAR 200X 4.3

INTRODUCCION DE RICARDO NARVAJA: Sigue uno de nuestros jóvenes crackers con otro tute, despachandose a gusto con el y colaborando nuevamente.
Gracias JIKI

Programa: Calendar 200X 4.3
Tipo: Aplicación
Protección: Ni se le puede llamar protección
Herramientas: W32dasm; hex workshop; language 2000; ProcDump; Caspr
Cracker (si se me puede llamar así): JIKI

Introducción...

Este es mi segundo tute (espero que les haya gustado el primero) y les digo que estos tutes son para Newbies (de un Newbie para otro) ya que no son unas protecciones muy buenas y la verdad es que a veces la suerte ayuda.

A este programa yo lo quería crackear "a lo bruto" (sin buscar el serial), pero por esas casualidades de la vida caímos justo con el serial.

PD: Quiero tratar de hacer otro tute antes de que me empiezen a matar de deberes en la escuela.

Pos parto...

Abrimos la aplicación con el Language 2000 y vemos que está comprimido con Aspack y está hecho en Visual C++. Bueno, el Aspack no es problema, ya que con el Caspr queda limpito como un tomate.

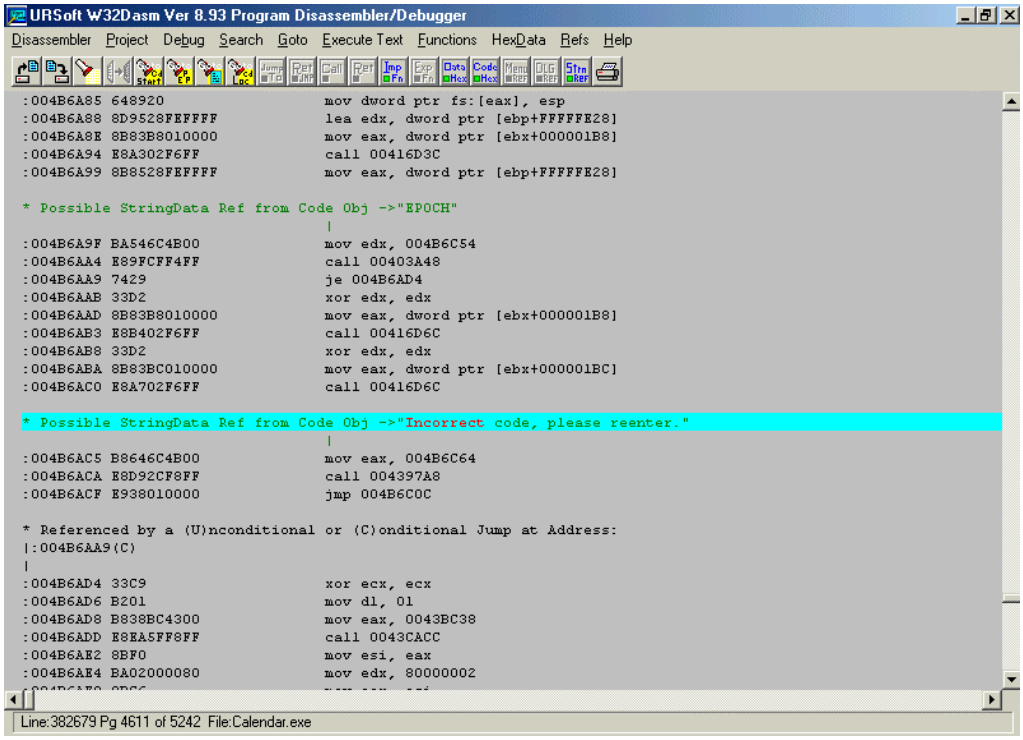
Paso siguiente, abrimos el ProcDump, vamos a PE EDITOR, selecciónamos el nombre del programa y hacemos clic en abrir. Ni bien apretamos abrir nos saldrá una ventana en la cual hacemos un clic donde dice SECTIONS. Luego procedemos a hacer clic derecho en CODE, ponemos edit section y en donde dice sections characteristics cambiamos de **C0000040** a **E0000020**. Hacemos lo mismo con todas las otras secciones (DATA; BSS; idata, etc.)

Al ataque...

Bueno, abrimos el Calendar 200X, vamos a **File/Register Shareware**, completamos con cualquier dato para ver qué pasa y... PIIIII, "**Incorrect code, plese reenter**".

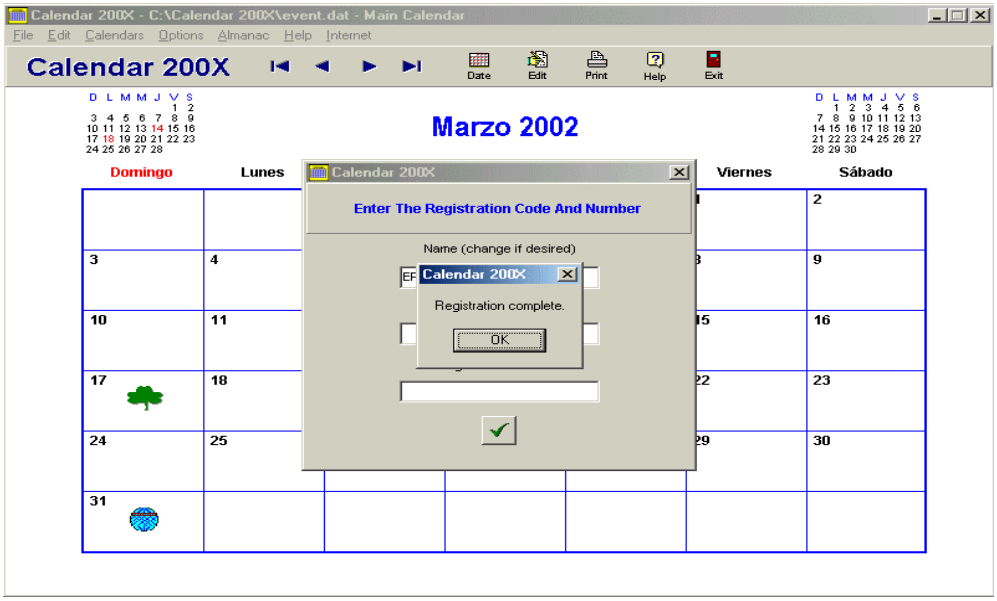
Abrimos el W32Dasm y desensamblamos la aplicación. Una vez abierto vamos a Search/Find Text y buscamos "Incorrect".

Caemos en:



Ahí vemos que no hay ningún salto condicional anteriormente. El único salto que hay es uno que está en la palabra **"EPOCH" (004B6AA9 7429 je 004b6ad4)**. Miren que raro que es desde esa palabra que se evita el comando **"Incorrect code, please reenter"**. Bueno, yo ni me preocupé. La joda es que lo crackeé a "lo bruto" (tenía un par de protecciones más) (**External Exception %x** y **Access violation at adress %x** y algo más que no va al punto).

Bueno, el chiste es que una vez que lo crackeé me quedó una duda con la palabra **EPOCH**, así que instalé de nuevo el Calendar 200X, fui a File/Register Shareware, completé todos los datos con la palabra **EPOCH** y...



Bueno, después me dí cuenta que en el único lugar en donde tiene que estar la palabra **EPOCH** es en **REGISTRATION CODE,** en donde dice **Name** y Registration Number pueden poner lo que quieran.

PD: Bueno, un abrazo grande a Ricardo y aprovecho para saludarlo ya que hace mucho que no le escribo.

PD: DISCULPEN!!! En el tute anterior seguramente no habrán podido cambiar ningún valor en el Hex Workshop y es porque el atributo del juego estaba en "Solo Lectura" para cambiarlo hagan clic derecho en BUSTOUT, van a propiedades, le sacan el tilde a "Solo Lectura" y se lo ponen a "Archivo".

PD: Traten de crackear el programa Calendar 200X con el TRW ¡¡¡ES MÁS FÁCIL QUE CON EL W32DASM!!!

¡¡¡PERDONENME POR LO DE LA LECCIÓN ANTERIOR!!!

LECCIÓN 65: TELEPORT PRO 1.29.1590

Programa: Teleport Pro 1.29.1590
Tipo: Para bajar Webs completas al disco rígido
Protección: Un dolor de huesos (si sos Newbie como yo)
Herramientas: TRW; w32dasm; language 2000
Cracker (si se me puede llamar así): JIKI

Introducción...

Y se va la tercera... Este es mi tercer tute y se lo voy a dedicar a Scrapie ya que casi tiene mi edad y espero que me escriba así tengo un amigo allá por España.

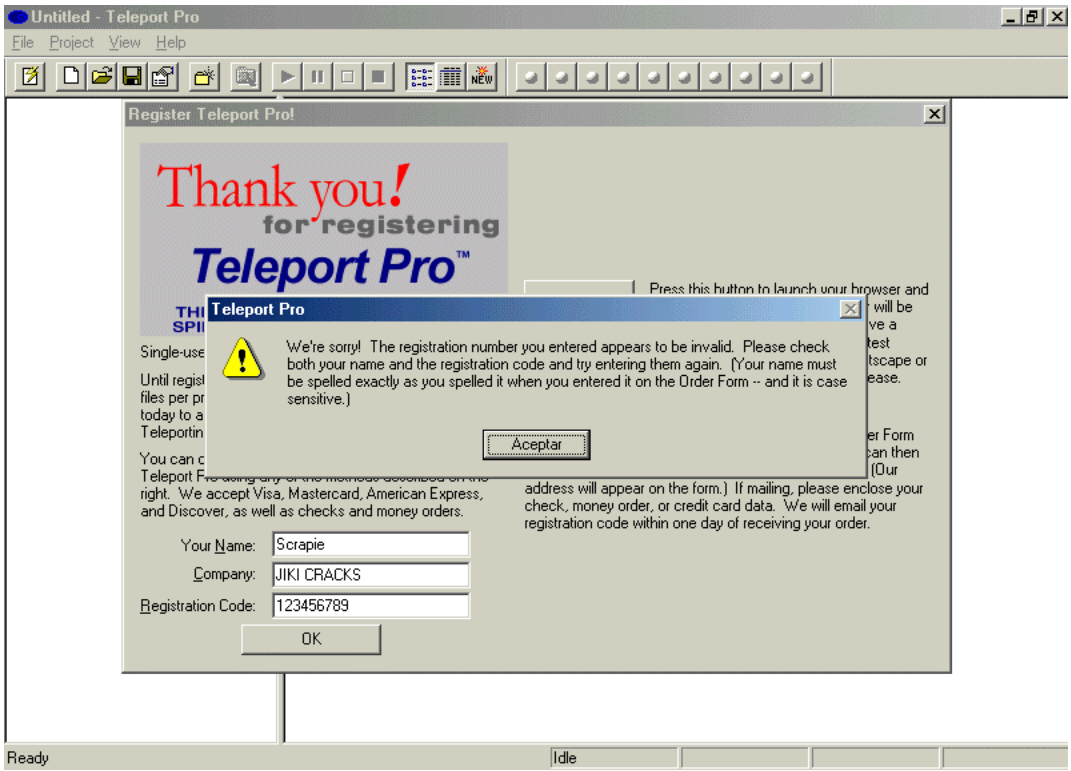
Este programa está bastante bueno, aunque no creo que sea muy útil para algunas personas pero estoy seguro de que otras lo van a necesitar algún día.

Pos parto...

Abrimos el programa con el Language 2000 y vemos que no está comprimido y está hecho en Visual C++ (Hasta ahora anda todo bien).

Al ataque...

Bueno, abrimos el Teleport Pro, vamos a Help/Register... completamos con nuestros datos y vemos qué pasa...

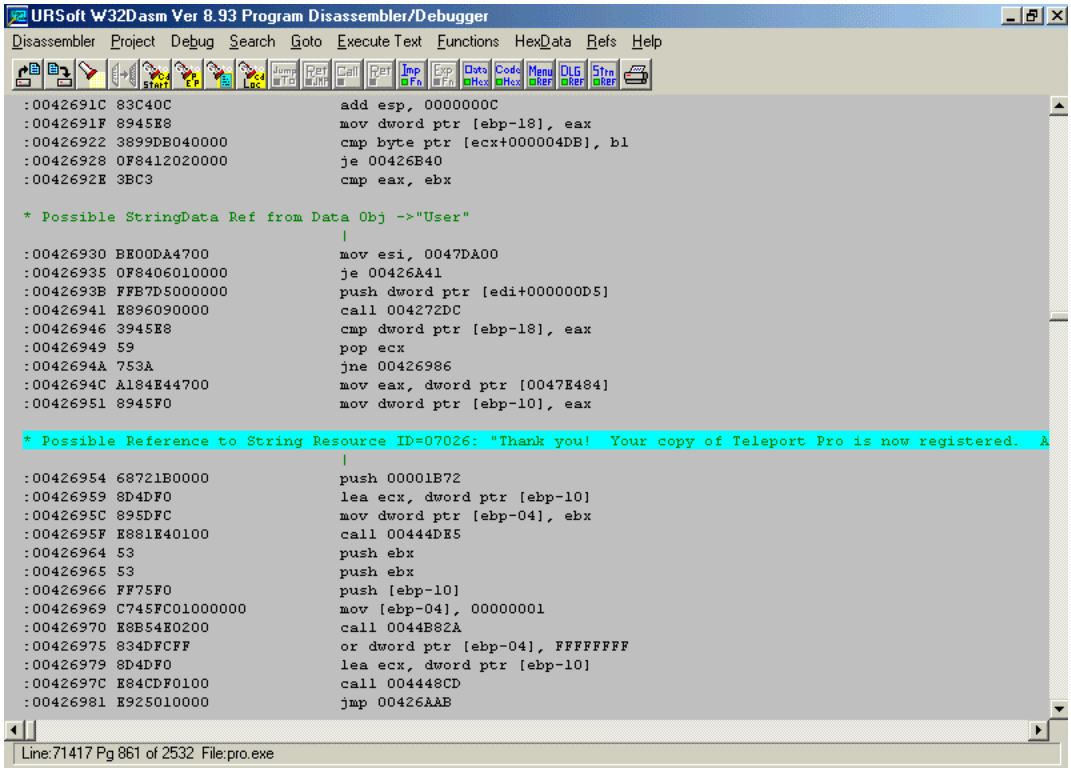


Bueno, parece que no nos dejan registrarnos pero no importa. Vallamos al W32Dasm y busquemos ese cartel de error (**We're sorry! Bla, bla, bla, bla, bla).**

Una vez que lo desensamblamos vamos a Search/Find Text, y ponemos We're sorry!, lo encuentra y por suerte es la única vez que aparece.

NOTA: Les cuento que lo traté de crackear a "a lo bruto", pero es más difícil que buscar el serial, después de media hora de estar intentando crackearlo a lo bruto sin resultados satisfactorios, me decidí a buscar el serial, estuve otra media hora pero apareció.

Lo que en realidad tenemos que buscar ahora para encontrar el serial es **"Thank you! Your copy of teleport pro is now registered"**. Bueno, lo buscamos y caemos aquí:



Recuerden que estamos buscando el serial, así que no se emocionen y cambien ningún valor en el Hex Workshop, pero si se animan a hacerlo solos ¡NO HAY PROBLEMA VIEJO!

Bueno, ya tenemos una idea de lo que está pasando:

```
:00426941 E896090000  call 004272DC                  <- Se genera el serial
:00426946 3945E8        cmp dword ptr [ebp-18], eax   <- Lo compara
:00426949 59          pop ecx
:0042694A 753A          jne 00426986                  <- Si está bien no salta
:0042694C A184E44700  mov eax, dword ptr [0047E484] <- y continua a
:00426951 8945F0      mov dword ptr [ebp-10], eax       Thank you!...

* Possible  Referente to String Resource ID= 07026: "Thank  you!
  Your  copy  of  Teleport Pro is  now registered"
|
:00426954 68721B0000  push 00001B72
:00426959 8D4DF0      lea ecx, dword ptr [ebp-10]
```

```
:0042695C 895DFC      mov dword ptr [ebp-04], ebx
:0042695F E881E40100  call 00444DE5
:00426964 53          push ebx
:00426965 53          push ebx
```

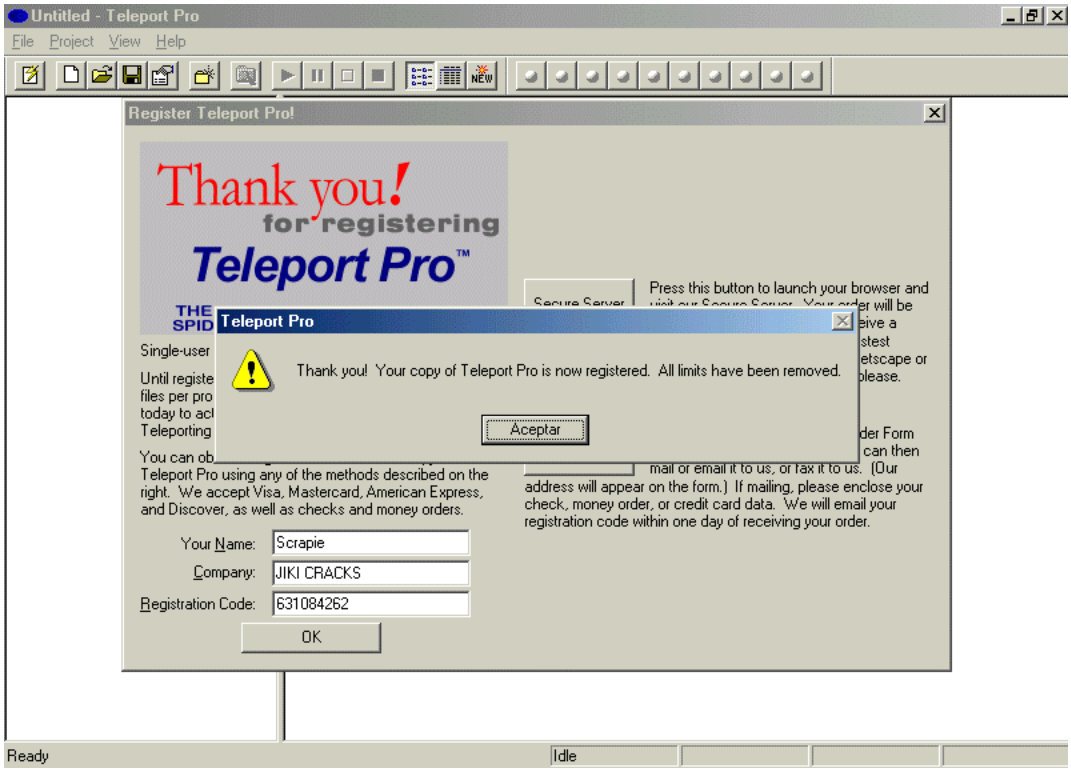
Bueno, vamos al TRW y ponemos un bpx en 426946. Vamos al Teleport Pro y en Help/Register... ponemos nuestros datos y... adentro mi alma (esa expresión me la copié de Ricardo, espero que no me demande, jaja)

Bueno, donde caemos hacemos:

? ebp-18 y copiamos la cadena en DEC **(7402600)**
? eax y copiamos la cadena en DEC **(631084262)**

Bueno, vamos de nuevo a Help/Register... completamos nuestros datos solo que esta vez en Registration Code ponemos **7402600** y... **"We're sorry! Bla, bla, bla, bla.**

Esperen!!!, no se desesperen, todavía nos queda probar con el otro número. Vamos de nuevo a Help/Register... completamos nuestros datos solo que esta vez en Registration Code ponemos **631084262** y...



Ahora vamos a Help/About Teleport Pro y...

NOTA: Cuando hagan **? ebp-18** y **? eax** los resultados van a ser distintos según el nombre que hayan puesto.

PD: UN GRAN SALUDO PARA RICARDO Y A TODOS LOS QUE ME MANDARON MAILS FELICITÁNDOME!!!

LECCIÓN 66: DIRECTORY PRINTER 1.8

Programa: Directory Printer 1.8
Tipo: Programa
Protección: No tiene
Herramientas: TRW; w32dasm; hex workshop; language 2000; ProcDump
Cracker (si se me puede llamar así): JIKI

Introducción...

Yo necesitaba un programa para imprimir las cosas que tenía en un CD y justo encontré este programa, pero es tan malo que voy a tener que buscar otro.

No tenía pensado hacer un tute sobre este programa ya que es tan fácil de crackear que no ocuparía ni media hoja, pero este va a ser un curso 2 en 1, ya que lo vamos a crackear "a lo bruto" y también vamos a buscar el serial.

Les recomiendo que si son Newbies y estaban buscando un programa fácil de crackear este es el de ustedes, anímense solos que pueden.

Pos parto...

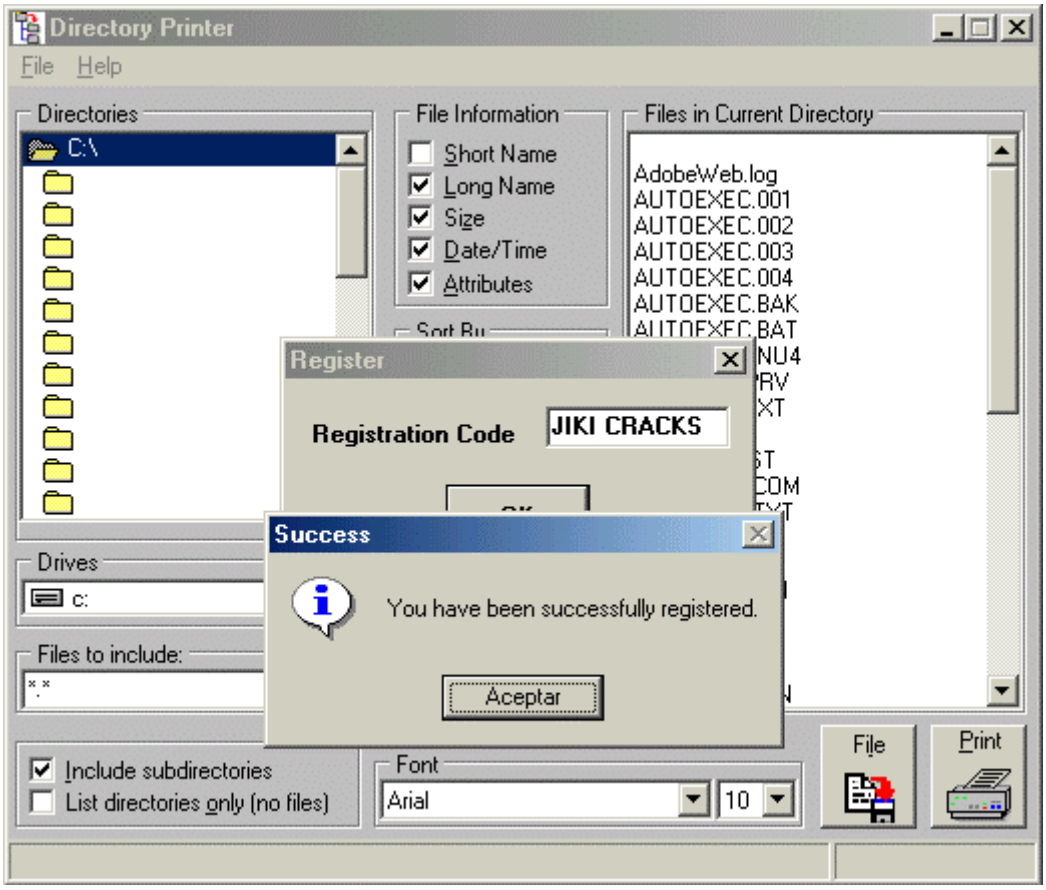
Abrimos el programa con el Language 2000 y vemos que no está comprimido y está hecho en Delphi. Paso siguiente, abrimos el ProcDump, vamos a PE EDITOR, seleccionamos el nombre del programa y hacemos clic en abrir. Ni bien apretamos abrir nos saldrá una ventana en la cual hacemos un clic donde dice SECTIONS. Luego en donde dice Charasteristics todos los **C0000040** a **E0000020**.

Al ataque (a lo bruto)...

Abrimos el W32Dasm y desensamblamos el programa. Una vez abierto vamos a String Data Referentes y buscamos algo como **"registration"**. Lo encontramos **(Incorrect registration code)**. Bueno, buscamos un poco más arriba y vemos que hay otra string que dice **"You have been successfully registered"**.

El salto condicional desde donde es llamado no sirve de nada, el salto que realmente tenemos que anular está un poco más arriba en **44644a 0f85f0000000 jne 446540**. Lo tenemos que nopear para que nunca nos mande al cartel de error.

Lo nopeamos y...



Al ataque (por el serial)...

Abrimos el TRW y cargamos el Directory Printer. Una vez dentro de él (del Directory Printer) vamos a Help/Enter Registration Code..., ponemos cualquier cosa, por ejemplo 1234567, vamos al TRW, ponemos un bpx hmemcpy, y pulsamos Ok.

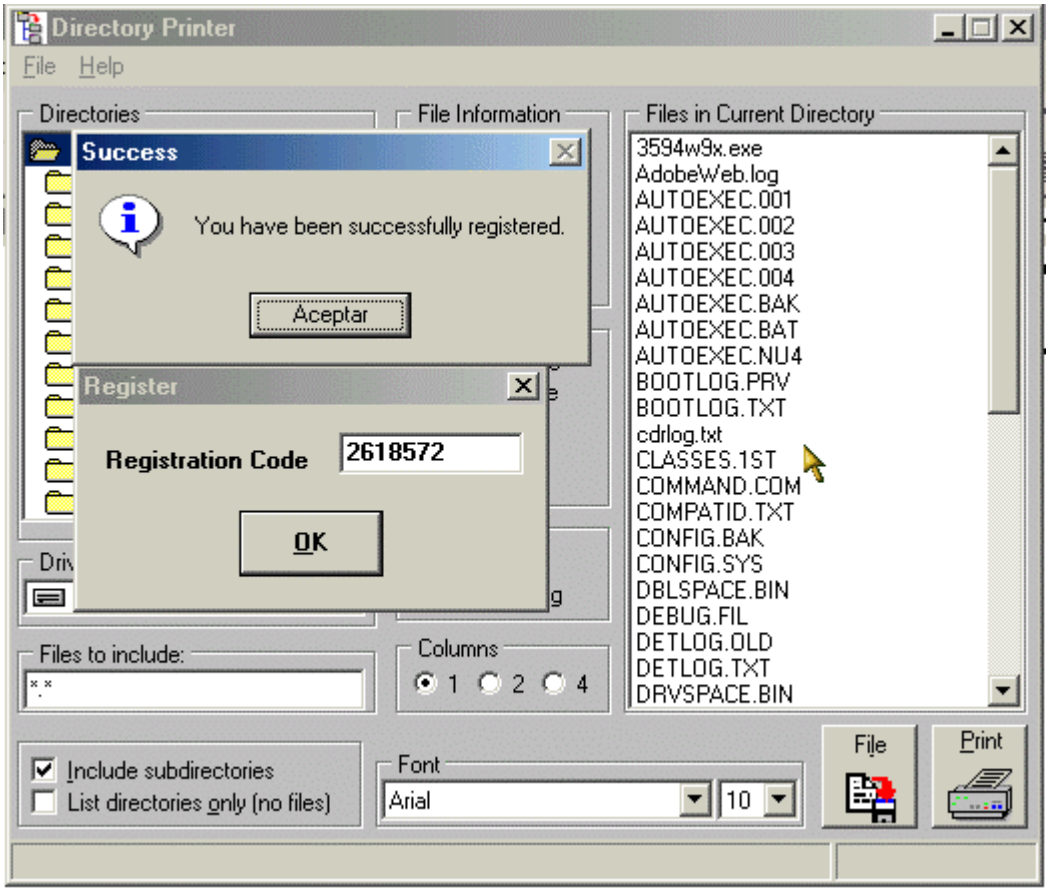
Apretamos f12 hasta caer en **DIRPRINT!CODE** (hay que apretar f12 exactamente 8 veces). Salimos de todos los RET (hay que apretar f10 exactamente 23 veces) y caemos en **446425**. Bueno, traceamos un poco y nos encontramos que en **446437** trae 2 valores:

446437 mov eax, 0027f4cc (Eso quiere decir que trae 2 valores, **eax** y **27f4cc**)

Ahí hacemos:

D eax y nos muestra nuestro código trucho (**1234567**)

? 27f4cc y nos muestra el serial válido (**2618572**)



PD: El logo anterior estaba mal ya que JIKI aparecía con "Y", pero ahora aparece bien.

PD: ¡¡¡¡UN SALUDO A TODOS!!!!

CIERRE:

Y esto es todo amigos; 66 lecciónes de conocimiento que el MAESTRO Ricardo nos ha dejado para la posteridad.

En el momento de finalizar esta recopilación el nuevo curso que Ricardo está desarrollando ya ha llegado a su "concurso" cuarenta y tantos. No dejéis de leer esa nueva obra maestra que entre todos los componentes del Grupo Cracks Latinos están llevando a cabo.

Desde aquí y para finalizar, quiero manifestar mi agradecimiento y reconocimiento al inestimable esfuerzo que personas como Ricardo dedican a ayudar de forma altruista, en este o en otros ámbitos de la vida, a personas que por unos u otros motivos necesitan esa ayuda. Desde mi punto de vista esa es la mayor grandeza que el ser humano puede alcanzar.

Ice Cube