CURSO OTOÑAL de CRACKEO

Por

Mr. Nobody

Documento creado por

Este documento, así como sus próximas revisiones lo puedes bajar desde **http://come.to/ATSASOFT**

Este documento ha sido elaborado con finalidades educacionales, no para fomentar el tráfico de códigos válidos para registrar programas. Para esta última finalidad ya existen sites especializados.

Cualquier tipo de duda, sugerencia,...puede ser planteada abiertamente enviando el mensaje a es.comp.crackers. No hay necesidad alguna de comunicarse individualmente por email, puesto que este es un "cursillo" público abierto a todo el que tenga interés en aprender....y hay mucho que aprender de las dudas de los demás :)

INDICE

Este documento ha sido elaborado con finalidades educacionales, no para fomentar el tráfico de códigos válidos para registrar programas. Para esta última finalidad ya existen sites especializados.

CURSILLO OTOÑAL DE CRACKEO

Un mensaje de Alfil y Kronos agradeciendo el envío de un tutorial sobre Winzip ha sido el detonante que ha despertado en Mr. Nobody el deseo de compartir los Conocimientos adquiridos a lo largo de estos últimos meses. Lejos de considerarse un "maestro" en el noble arte del crackeo, Mr. Nobody desea ayudar a la gente que tenga la curiosidad de conocer cómo funcionan por dentro estos programillas shareware que pululan por los CDS que mensualmente adquirimos en los quioscos o que nos bajamos desde la red.

Ante todo debería primar en nuestro "cursillo" el carácter puramente educativo y la voluntad de "aprender". Sus finalidades son puramente educacionales, así que no vamos a traficar de forma explícita con códigos para registrar programas.

No hay ninguna necesidad de hacerlo. Para ello ya existen sites especializadas como astalavista.box.sk o programas específicos como el mítico oscar2000. Os aseguro que en cinco minutos de visita a astalavista conseguirías los códigos de todas las utilidades que aquí podremos analizar.

Este pequeño proyecto se dirige exclusivamente a gente con ganas de aprender, compartir conocimientos y ansiosa de experimentar la indescriptible sensación de superar un reto planteado con aspiraciones de conocimiento. Considerémoslo pues como un juego de esos en los que hay que descubrir algo, que no sabemos dónde se encuentra. No entra dentro de nuestro objetivo perjudicar a nadie,... más bien podemos ayudar a evidenciar la falta de protección de algunos programas o sus fallos más habituales.

Si os parece bien, cada semana nos podemos plantear algún reto y analizaremos los errores que se han cometido y nos han permitido acceder a lo que estábamos buscando. Empezamos prácticamente de "cero", así que no va a perderse nadie que ponga un mínimo de atención. Dentro del temario del cursillo entrarán los siguientes temas:

- Localización de "harcoded" seria
- Uso del editor hexadecimal para la eliminación de nags molestos
- la técnica de la inversión del salto condicional (75/74)
- Uso de Softice
- Uso de regmon y filemon
- ... ????

Estaría bien que la gente interesada en el proyecto enviará algún mensaje a este grupo manifestando su deseo de que esto sigue adelante. Un mensaje con el título C.O.C. - DALE, DALE o bien C.O.C. - ADELANTE sería suficiente. Junto a este mensaje que estás leyendo vais a encontrar la primera lección.

ATENCIÓN!! El único condicionante para que una segunda lección sea enviada será que al menos dos mensajes de apoyo sean posteados al grupo :) Es importante para Mr. Nobody saber que no está solo :)

Lección 1 – Cursillo Otoñal de Crackeo (28/9/99)

HARDCODED SERIALS(*)

Requisitos:

- w32dasm
- ojos (2 / aunque con 1 bastaría)

Conocemos con el nombre de "hardcoded" serial a los códigos que van "escondidos" en el mismo interior del ejecutable del programa. "Escondidos" en realidad es un decir. Únicamente desensamblando el programa y observando el listado se puede localizar el código que estamos buscando.

Obviamente los programadores no suelen caer mucho en la tentación de obrar de una forma tan poco prevenida, pero creedme que aún es posible encontrar utilidades "protegidas" de una forma tan poco seria.

En este sentido, la única herramienta que necesitamos es un desensamblador. W32DASM es una herramienta excelente para esta finalidad. Una vez el fichero .exe ha sido desensamblado, en el Menú REFS, string data references.. Podemos encontrar todo el listado de cadenas de texto que el programa utiliza.

En este listado uno puede encontrar con frecuencia lo que está buscando :) DRIVE ALARM 2.1 (drvalm45.zip) es una utilidad que ha sido programada con la técnica citada anteriormente. Al intentar registrarlo se nos aparece un cuadro de diálogo dividido en dos secciones XXX XXXXXXX. Parece ser que nos piden tres caracteres + siete caracteres. Es un pequeño detalle para despistarnos.

No os dejéis engañar: 3+7=10!!! Desensamblemos, echemos un vistazo,....y a ver qué pasa.!! No hay ninguna necesidad de postear la solución. Los "deberes" son faena de uno mismo...y en este caso se supone que todo el mundo puede leer un listado de términos raros buscando algo parecido a un código y que tenga 10 caracteres.

PS: Encontraréis el programa en HOTSHAREWARE#30 o en cualquier site que distribuya productos shareware. Si cuando leáis esto hay alguna nueva versión del programa, puede ser que disponga ya de una protección nueva, cosa altamente recomendable, por cierto. Nuestra lección versa sólo sobre la versión 2.1

Eso es todo amigos. Mr. Nobody desea ver en el grupo 2 mensajes de aprobación. Hasta dentro de siete días (programación sujeta a cambios, por supuesto) ... si es que hay alguien que se suma al cursillo. Recordad, algo del tipo C.O.C.- Dale, dale o C.O.C. - lo que sea ¿Hay alguien?...Alguien?...alguien? .....Eco, eco... eco, eco. :)

PS: Cualquier tipo de duda, sugerencia, puede ser planteada abiertamente enviando el mensaje a es.comp.crackers. No hay necesidad alguna de comunicarse individualmente por e-mail, puesto que este es un "cursillo" público abierto a todo el que tenga interés en aprender... y hay mucho que aprender de las dudas de los demás:)

> Qué fácil (xDDD). Con esa versión sí que se puede, solo que el key es de 3+8, en vez que de 3+7. Esperamos tu próxima lección. Gracias.
>
> Para Bajarse el programita de marras:
> http://www-users.dragon.net.au/chilli/products/drvalm49.zip

ADDENDUM/Lección 1 – Cursillo Otoñal de Crackeo (01/10/99)

Ante todo, un saludo y mil agradecimientos a los que se han "inscrito" en el otoñal cursillo. Sinceramente, habéis emocionado a Mr. Nobody. :) Bien,...una vez probado que el interés por aprender "está ahí fuera", preparémonos para completar nuestra primera lección (HARDCODED códigos, ¿recordáis?) con una especie de "más difícil todavía":)

Requisitos:

- w32dasm
- Logosaver http://www.galttech.com/logosave.zip (224 kb)
- Visión frontal (en dirección hacia la pantalla del ordenador)

Bien. Nuestro material objeto de estudio es un pequeño salva pantallas que hace tiempo que está abandonado por la compañía que lo creó. Parece como si lo hubieran hecho como un simple entretenimiento y lo hubieran arrinconado en su site. Como podéis imaginar, su protección está basada en un hardcoded serial, es decir, el código está en el propio fichero principal. En este caso, no es un fichero exe, sino un *.scr ya que se trata de un screensaver que se alojará en c:\windows Para entrar el código deberéis ir a Configuración del protector de pantalla.

Como supondréis, habrá que descompilar con WDASM el fichero logosave.scr Una vez completado el proceso os daréis cuenta de que no hay STRING DATA REFERENCES (que viene a ser "lo que el programa considera como cadenas de texto"), no obstante ello no quiere decir que no haya "texto" en el listado desensamblado.

Alguno de vosotros dirá...joder!! Pero buscar en medio de todo este meollo algo parecido a un código, puede ser un auténtico palazo!! Es cierto, por esto hablamos de "más difícil todavía" Tenemos varias opciones:

1. Empezar a volvernos miopes leyendo cada línea de texto que encontremos en el "listado muerto" que nos ha proporcionado WDASM.
2. Habiendo observado que muchas cadenas de texto aparecen después de una secuencia de tipo DB " / podríamos buscar con el comando Find.... todo lo que empezara con estas letras (DB ").

3. Ir directos a buscar en los alrededores de la zona "caliente", es decir, cerca de la zona donde nos sueltan aquello de "gracias", "tu código no mola..".

Si supiéramos como se dice eso de "gracias" en inglés y le diéramos al Find... con la palabra en cuestión, aterrizaríamos de lleno en la zona caliente. Sólo haría falta, pues, husmear por los alrededores:)

Ya tenéis suficientes datos para conseguir lo que buscáis. Recordad que no es necesario postear ningún dato en es.comp.crackers. Sí queréis podéis comentar su dificultad, si habéis tenido algún problema, el rato que habéis tardado.. Pero no hace falta ser explicito en lo concerniente al código real, puesto que es lo de menos.

PD: Todos superaréis la prueba, lo sé. Los datos se guardarán en galtlogo.ini en c:\windows. Echadle un vistazo una vez hayáis acabado el ejercicio.

> Eso es todo amigos. Mr. Nobody se complacería en leer en el grupo nuevos mensajes de aprobación y ánimos. Su lectura resulta muy gratificante y complaciente. Hasta dentro de siete días (programación sujeta a cambios, por supuesto)

Cualquier tipo de duda, sugerencia,...puede ser planteada abiertamente enviando el mensaje a es.comp.crackers. No hay necesidad alguna de comunicarse individualmente por e-mail, puesto que este es un "cursillo" público abierto a todo el que tenga interés en aprender....y hay mucho que aprender de las dudas de los demás :)

APÉNDICE FINAL/Lección 1 – Cursillo Otoñal de Crackeo (03/10/99)

PEQUEÑAS VARIANTES DE HARDCODED

Bueno, vamos a dejar ya el apartado de "hardcoded" serial pero antes haremos un pequeño análisis en relación a algunos programas que los contienen para observar que, a veces, están escondidos en sitios inesperados o están allí aunque hayan intentado hacernos creer que no.

Mr. Nobody no ha probado todos los programas que aparecen en el listado siguiente. Si alguien quiere molestarse en comprobarlos uno a uno, puede hacerlo. Quizás las nuevas versiones hayan buscado métodos más serios de protección. No es imprescindible verificar encontrar cada uno de estos programas. Lo interesante es leer la descripción que ha sido añadida para aprender sobre "hardcoded" serial de una modalidad algo diferente a los de Logosaver o Drive Alarm. Allá van:

***Swapper 3.0** .....Con WDasm y un vistazo a listado se halla el "tesoro"

***Webscripting Editor 1.03** ..... Nos piden nombre, compañía y código, aunque sólo esto último es lo que importa. Quieren hacernos creer que se va a generar un

código diferente para cada usuario pero en realidad sólo importa entrar el código que se halla fácilmente con WDasm. Curioso, ¿no?

***Intellibots 1.2** ..... Nombre y código, aunque sólo importa el código!!!! También se halla en el listado desensamblado por WDasm.

***CCZip 3.12.01 Pro** ..... El "tesoro" no se halla en el ejecutable, sino en una DLL que se halla en el directorio del programa. ¿Sorprendidos? También hay que entrar un nombre de usuario pero no importa. El código siempre es el mismo

***Poster 6.1** ..... En el ejecutable desensamblado se halla el "tesoro". No-problema.

***Hyperpas** ((una utilidad para programadores en Delphi)) Es freeware, aunque sus creadores quieren que visites su página donde encontraras el password para utilizarlo. Podemos ahorrarnos este paso desensamblando el ejecutable y echando un vistazo a la "zona caliente"

ADVERTENCIAS FINALES... MUY SERIAS

Es lógico pensar que los programadores tienen la curiosidad por averiguar si pululan por la red los códigos válidos para sus creaciones. Algunos de ellos suelen estar muy "al día" y en cuanto se enteran de que el "tesoro" ha sido descubierto, se apresuran a sacar nuevas versiones en que el anterior código ya no funcionará.

En este sentido, nos podemos encontrar con que, después de desensamblar un ejecutable, nos encontremos con uno o varios códigos a la vista. Hay que ir con mucho cuidado, porque ¿qué pasa si el programador está cabreado? Puede haber preparado una bomba para el desprevenido que introduzca un código válido antiguo de los que aparece en el listado de WDasm !!!!!

Ejemplos:

*Getright: Los creadores de esta excelente utilidad acostumbran a "blacklistear" (considerar "indeseables") a toda una serie de códigos que valían para antiguas versiones. Cuando alguien desensambla el ejecutable y se los encuentra por allí, puede pensar: !Qué fácil!!Un hardcoded!! Al entrar alguno de aquellos códigos, se encuentra con un mensaje diciendo que está utilizando un código pirata y el programa bloquea el cuadro de diálogo de registro y ya no es posible registrarlo (En realidad ha introducido unos datos en el registro de Windows para impedir que Getright sea registrado, ya que ahora ya no se fían de nosotros). En otras palabras, hay "hardcoded" códigos que no son buenos, sino que están ahí porque están "blacklisteados". Tenedlo en cuenta!!

*Soluciones muy bestias: Ashell :El autor de este programa de gestión de ficheros comprimidos está muy cabreado. Si desensambláis su ejecutable encontraréis cantidad de nombres de personas (en su mayoría, crackers conocidos) y códigos que habían sido distribuidos para versiones anteriores. Si se os ocurre introducir alguno de estos datos, despedios de todos los ficheros que hay en Windows y Windows\System !!!! Su tamaño pasa a ser "0" !!!!!

CONCLUSIÓN

Los "harcoded" son algo excepcional. Sólo son usados por programadores con pocas ganas de proteger su producto y no son nada habituales. Desconfiad siempre de lo que halléis, porque hay mucha mala leche ahí fuera. En todo caso, no estará mal disponer de un ordenador de prueba, por si las moscas.

PD: La próxima lección será la 2, tened preparado el WDasm y un editor hexadecimal (Ultraedit, hiew, hexedit, hexworkshop,...) porqué empezaremos con la "ingeniería invertida" de lleno!!!! :)

Lección 2 – Cursillo Otoñal de Crackeo (06/10/99)

¡Saludos a todos! Entramos de lleno en el arte de la ingeniería invertida que en realidad no es más que el deseo de que los programas instalados en nuestro ordenador se comporten tal como nosotros queramos.

Las herramientas que requeriremos son WDasm y un editor hexadecimal (el que más os guste). La primera actividad que llevaremos a cabo constituye una de los más típicos y conocidos ejercicios de crackeo. Podéis llamarle como queráis, aunque cualquiera de los siguientes títulos podría valer:

- LA INVERSIÓN DEL SALTO CONDICIONAL
- LA TÉCNICA DEL 74-75

El objetivo de este mínimo acto de ingeniería invertida consiste en **conseguir que un programa acepte cualquier código falso como correcto** (y que, por otra parte, considere al código correcto como erróneo).

Más de alguna vez habréis pillado algún crack en cuyas instrucciones sólo os decían: copia el crack en el mismo directorio del programa, ejecútalo, entra después en el programa y puedes entrar el código que quieras. Pues bien, este detalle que nos dejó a todos maravillados en su momento :), En realidad se puede conseguir con muy pocos cambios en el programa (generalmente un único cambio). **El ejecutable del crack no hizo más que cambiar un 74 por un 75 o quizás al revés !!!** (Bueno, en realidad hay cracks que permiten entrar cualquier código y no se basan en el cambio 74/75, pero dejémoslo para más adelante, ¿no?)

Antes que nada, hay que decir que la técnica que aprenderemos en esta Lección 2 vale para un buen número de programas, pese a que ya no son muchos los que toman tan pocas precauciones. Más adelante expondremos un listado de algunos programas elaborados por personas con muy pocos deseos de proteger su producto.

- ***¿POR DÓNDE EMPEZAMOS?***

Estamos de acuerdo en que un programa no deja de ser un listado de instrucciones que se ejecutan de arriba hacia abajo y que el ordenador debe entender. Al parecer las máquinas sólo hablan / entienden "lenguaje máquina". Así que los ficheros ejecutables de nuestros programas contienen unos numeritos (0-9) y letras (A-F) que la máquina interpreta como instrucciones en lenguaje ensamblador:

80 fc 40 en un ejecutable es leído por la máquina como **CMP ah,40** (compara AH con 40)

75 14 en un ejecutable es leído por la máquina como **jne XXXXXX** (sino es igual a lo que se esperaba, salta 14 numeritos más abajo)

No os preocupéis por ahora sobre qué significan estas cosas. Sólo tenéis que creeros que los numeritos en hexadecimal que aparecen cuando abrimos un ejecutable con un editor hexadecimal son instrucciones que la máquina sabe entender.

¿Cómo deben funcionar las instrucciones que tienen que ver con la comprobación de un código? Evidentemente los esquemas pueden ser muy diversos, pero muchos programas recurren a un listado muy similar a lo que tenéis expuesto a continuación: (recordad que la máquina lee de arriba a abajo)

1) bla, bla, bla
2) Inicio de zona de comprobación de código
3) Cojamos el código que ha introducido el usuario y pongámoslo en un cajoncito llamado EAX
4) Cojamos el código auténtico y pongámoslo en un cajoncito llamado ECX
5) Comparemos EAX y ECX
6) Si no son iguales, saltemos a (9)
7) Emitir el mensaje "Gracias por registrarse"
8) Permitir acceso completo, desbloquear lo que sea,...bla,
9) Saldría un mensaje “Código Incorrecto, ...” o algo parecido.
10) Bla, bla...

Leamos de arriba a abajo las instrucciones. Al llegar a la comparación nos encontramos con que el programa debe tomar una decisión que dependerá del resultado de dicha comparación. En este caso el programa está preparado para saltar hacia (9) en el caso de que EAX(nuestro código) y ECX(código válido) no sean iguales. ¿Qué habría pasado si fueran iguales? Lógicamente, no habría saltado a (9), sino que habría continuado con la línea siguiente (7), donde nos dan las gracias por ser tan buenos.

Si habéis desensamblado algún ejecutable con WDasm, a lo mejor os habéis encontrado con que, cerca del mensaje de error, algo más arriba se encuentran los mensajes de felicitación!! Probablemente algo más arriba aún se encontrará el salto condicional (je o jne)en el que se decide si saltar a error (Chico malo) o no saltar, dejar que las instrucciones sigan hasta llegar a "Gracias" (Chico Bueno).

- ***¿HAY ALGUNA MANERA DE EVITAR QUE SE PRODUZCA EL SALTO A ERROR?***

Efectivamente! Si llegamos a identificar y localizar "el salto", tendremos la posibilidad de cambiar dicha instrucción (con un editor hexadecimal) por otra que nos

asegure que llegaremos a la zona de los agradecimientos. Llegados a este punto, pueden ser varias las soluciones que podríamos adoptar:

1.- Podríamos decirle por ejemplo que saltara, pero no hacia (9), sino hacia (7). En otras palabras, el programa comprobaría que los dos números no son iguales y por consiguiente saltaría, pero hacia la zona que más nos gusta (Buen chico)

2.- Otra posibilidad sería decirle que no hiciera nada, con lo cual, nuestro ordenador pasaría a leer la siguiente instrucción, que es la de los agradecimientos. Al arte de ordenar "no hacer nada", se le denomina **NOPEAR (NO OPERATION)**

3.- La tercera posibilidad para conseguir llegar a la zona de "Gracias" consiste en INVERTIR EL SALTO, es decir, DAR UNA INSTRUCCIÓN CONTRARIA A LA INICIAL.

Imaginaos que somos capaces de meter en "**6) Si no son iguales, saltemos a (9**)" una orden invertida "**6) Si SON iguales, saltemos a (9)**". ¿Qué pasaría entonces? Muy simple, el programa saltaría a error si fueran iguales, es decir, si entráramos el código correcto. Si no introdujéramos el código correcto, como no sería igual al esperado, no saltaríamos, sino que pasaríamos a la siguiente instrucción, llegando pues a la zona del Buen chico!!!!

En definitiva, el salto condicional original evitaba que llegásemos a la zona de Gracias, puesto que el salto se producía. Nuestra intención es la de evitar que el salto tenga lugar.. y la mejor forma que se nos ocurre es despistar al programa cambiando las condiciones del salto (salta si son iguales! Si entramos el código auténtico recibimos mensaje de error; mientras que cualquier otro código es enviado a la zona-del- Buen Chico)

Leed con detenimiento el siguiente punto porque es la clave de lo que vamos a trabajar a continuación.

- ***¿CÓMO LOCALIZAR EL "SALTO"?***

Para nosotros es vital que el programa nos envíe un mensaje de error, puesto que la cadena de texto contenida en ese mensaje de error será nuestra referencia clave para conseguir localizar el salto.

Para probar una vez más nuestra voluntad de no perjudicar a nadie con nuestro cursillo, nada mejor que hacer nuestro primer experimento con un programa que todo el mundo debe tener seguramente y que no vamos a craquear por completo.

Cualquier versión de Winrar nos sirve para lo que pretendemos practicar. Espero que el objetivo educacional del C.O.C quede demostrado por el hecho de que sólo vamos a forzar al programa a decirnos "Gracias" sin que, por ello, este totalmente craqueado.

Mr. Nobody ha utilizado Winrar 2.5 beta2, pero cualquier versión vale si os quedáis con el detalle de cómo funciona el tinglado. Instalémoslo, busquemos la manera de registrarnos. Nos piden nombre y código.
Introduzcamos lo que nos venga en gana. Ahí está el mensaje de error: **"Registration failed"**

Hagamos una copia del fichero ejecutable y desensamblémoslo con WDasm. (el motivo de lo de la copia es que no podremos realizar cambios con un editor hexadecimal mientras el fichero esté desensamblado con WDasm. De esta forma podremos realizar cambios en un exe mientras el otro esté siendo desensamblado por WDasm).

Busquemos el texto "Registration failed" en String Data References. (fijaos que podréis ver también cadenas de texto como el "Evaluation Version" que aparece en la parte superior del programa o el mismo mensaje de felicitación!!)

String Resource ID=00870: "Registration failed"

Una vez localizado, le hacemos doble clic y aparecemos aquí >>>>

* Possible Reference to String Resource ID=00870: "Registration failed"

```
:00408EE4 6866030000          push 00000366
:00408EE9 E856D6FFFF          call 00406544
```

Si nos fijamos en las cadenas que hay por el entorno, apreciaremos que los mensajes de felicitación vienen más abajo. Bueno, no pasa nada. No nos sorprende la existencia de diferentes variantes que puede haber de la misma estructura. La cosa pinta así:

Bla bla
"Registration failed"
"Correct registration "

Necesariamente, antes del "registration failed" debe haber un salto condicional que decide si debe llegar a "failed" o salta a la zona que viene después, la zona del buen Chico.

¿Cómo identificamos un salto condicional? Estas son las instrucciones en ensamblador que valen para los dos saltos que nos interesan:

je (en hex. = 74) salta si es igual (a lo que la máquina esperaba)
jne (en hex. = 75) salta si no es igual (a lo que se esperaba)

Subamos para arriba para ver qué se cuece en el código. Uy!!! qué es esto!!

```
************************************
:00408ECD E836C30100       call 00425208
:00408ED2 83C408           add esp, 00000008
:00408ED5 85C0             test eax, eax
:00408ED7 752C             jne 00408F05 >>> si no es igual a lo que esperaba, salta a  408f05
```

(2c hex. = 44 decimal indica saltar 44 bytes. Comprobadlo con la calculadora de Windows: 2c = 44)

```
:00408ED9 6A30             push 00000030
```

```
* Possible Ref. to Menu: MAIN_MENU, Item: "Change drive    Ctrl-D"
* Possible Reference to Dialog: ABOUTRARDLG, CONTROL_ID:0065, ""
* Possible Reference to String Resource ID=00101: "Warning"

:00408EDB 6A65                  push 00000065
:00408EDD E862D6FFFF            call 00406544
:00408EE2 59                    pop ecx
:00408EE3 50                    push eax

* Possible Reference to String Resource ID=00870: "Registration failed"

:00408EE4 6866030000            push 00000366
:00408EE9 E856D6FFFF            call 00406544
:00408EEE 59                    pop ecx
:00408EEF 50                    push eax
:00408EF0 8B4D08                mov ecx, dword ptr [ebp+08]
:00408EF3 51                    push ecx

* Reference To: USER32.MessageBoxA, Ord:0000h

:00408EF4 E89A220400            Call 0044B193
:00408EF9 33C0                  xor eax, eax
:00408EFB A37CF04400            mov dword ptr [0044F07C], eax
:00408F00 E9F9000000            jmp 00408FFE

* Referenced by a (U)nconditional or (C)onditional Jump at Address: :00408ED7(C)

:00408F05 6A40                  push 00000040 >>>>>el salto lleva hasta aquí!!!!!LA
                                                zona del Buen chico!!!

* Possible Reference to String Resource ID=00872: "Correct registration"

:00408F07 6868030000            push 00000368
:00408F0C E833D6FFFF            call 00406544
:00408F11 59                    pop ecx
:00408F12 50                    push eax

* Possible Reference to String Resource ID=00871: "Thank you for support"

**************************************************
```

El jne lleva hasta 00408F05 . Si bajáis la vista hasta esta dirección, veréis que es la zona en que se establece "Correct registration".

Evidentemente este salto no tiene lugar cuando entramos un código erróneo, sino que las instrucciones se ejecutan hasta llegar a "Registration failed". ¿Os gustaría saltar hasta la zona de felicitaciones? Sólo nos falta invertir el salto en cuestión: si se trata de jne, pondremos je (o a la inversa). El programa acabará haciendo lo contrario de lo que se supone que debe hacer.

¿CÓMO REALIZAR EL CAMBIO?

Si nos situamos con WDasm encima de la línea:
00408ED7 752C jne 00408F05
(Dirección) (hex.) ==== (instrucción ensamblador)

Podremos ver como se pone de color verde (si le dais a "Execute jump" en WDasm, saltaréis hacia la zona a la que apunta el salto). En la parte inferior de WDasm aparece (en mi caso):Offset@000084d7h.

Lo que nos interesa exactamente es 84d7. Esta es la "dirección" que buscaremos con el editor hexadecimal. Iniciamos nuestro editor hex., abrimos el ejecutable y buscamos el Ofsett 84d7. Esto debería llevarnos directamente a una zona con numeritos hex. y justo encima de un precioso 75 2C. Debemos cambiarlo por 74 2C, guardar los cambios y ya está!!! (Los cambios los llevamos a cabo, estando Winrar sin ejecutarse!!)

Al reiniciar, vamos a Options, Register... y podremos entrar lo que queramos. EL programa nos dirá "Thanks for...".

Hemos forzado Winrar para que nos dé las gracias. Si introducís un código real válido os dirá "Registration failed" Ahora llega la decepción. Al reiniciar el programa continúa estando sin registrar. **¿Porqué?** Pues porque el programador se ha molestado en incluir otro chequeo aparte del que nosotros acabamos de reventar. A lo largo del programa hay más de una comprobación con lo que sólo hemos "craqueado" parcialmente el producto. Ya hablaremos de la solución definitiva...

¿Todos los programas tienen más de una comprobación o chequeo? En absoluto. A continuación tenéis un pequeño listado (que iremos ampliando) de ejercicios que se solucionan por el mismo método (o similar) y no tienen ningún chequeo adicional, con lo cual, el método explicado será suficiente para conseguir lo que pretendemos. Si queréis una pequeña dosis de 74/75 fáciles:

SUPER TEXT SEARCH
FONTLOOK
DIRECTORY PRINTER
DIRECTORY COMPARE
SIMPLY3D (!!!)
COPYTO
FULLDISK 3.3
ACEEXPERT 3.0
CD-R DIAGNOSTIC 1.44
MORPHER 2.0

(Mr. Nobody no ha verificado cada uno de los programas citados, pero sí ha leído tutoriales al respecto que confirman su protección de 1 solo cambio)

(Ya sabéis que un simple NOMBREDELPROGRAMA + DOWNLOAD en Altavista será suficiente para localizarlos)

En la mayoría de estos programas, al encontrar el mensaje de error, vemos unas líneas más arriba algo parecido a:

* Referenced by a (U)nconditional or (C)onditional Jump at Address: 0040114C(C)

Con ello nos dicen DESDE DONDE se ha saltado a error. Con Goto / Goto code location... 40114c WDasm nos llevará directos al je / jne que nos interesa!!!

A trabajar!!! Recordad 1.mensaje error/ 2.de donde viene/ 3.invertir je /jne)

PS: A veces podéis encontraros 0F 84 (EN LUGAR DE 74) y 0F 85 (en lugar de 75). La inversión funcionaría de la misma forma.

PSS: En algunos manuales encontraréis que "je" es lo mismo que "jz" (y jne = jnz)

PSSS: ¿Demasiada información? Que cada uno se tome su tiempo :)

ADDENDUM/Lección 2 – Cursillo Otoñal de Crackeo (09/10/99)

Debido a que parece claro que es un problema que practiquemos con programas diferentes, versiones diferentes, etc. vamos a realizar una pequeña práctica con un mismo material.

Completamos nuestra lección 2 con un pequeño añadido que corrobora nuestro deseo de no perjudicar a nadie con nuestro cursillo. En esta ocasión trabajaremos con un programa diseñado adhoc para ser crackeado. Los interesados en conseguirlo podéis hurgar en es.binarios.misc con el título C.O.C- ejercicio lección 2

Cruhead's Crackme1.0
(el autor recomienda no ser tramposo y usar WDasm, pero para nosotros será un ejemplo muy ilustrativo)

Entramos cualquier dato y nos encontramos con el mensaje de error.

Desensamblamos con WDasm y buscamos el mensaje de error en StringData References. Doble clic y nos trasladamos a:

* Referenced by a CALL at Address: |:00401245

:00401362 6A00 push 00000000

* Reference To: USER32.MessageBeep, Ord:0000h

:00401364 E8AD000000 Call 00401416
:00401369 6A30 push 00000030

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

:0040136B 6860214000 push 00402160

* Possible StringData Ref. from Data Obj ->"No luck there, mate!"

:00401370 6869214000 push 00402169

Algo más arriba tenemos la pista de como hemos llegado hasta aquí: Reference by a CALL (podría ser un JUMP) at Adress 401245
Es el momento de darle a Goto Codee Location 401245. Esto nos hará llegar a:

```
dirección hex.          instrucciones ensamblador
:00401241 3BC3          cm ex, ex
:00401243 7407          je 0040124C
:00401245 E818010000    cal 00401362 >>>>>>>>CALL que nos ha llevado a
                                                 "error"
:0040124A EB9A          jmp 004011E6
```

¿Cómo podríamos haber evitado entrar en CALL 401362(ERROR)? Fijémonos en una línea más arriba: Hay un salto condicional justo después un CMP (comparación). Bien, así pues, el programa compara y decide si saltar a algún sitio o seguir hasta llegar al CALL-error.

Nuestra deducción es la siguiente: el salto no se ha producido y por tanto el programa ha llegado una línea más abajo al CALL-error. Si hacéis la prueba con WDasm, podéis darle al Execute Jump (cuando estéis sobre esta línea) y llegaréis a un CALL, entonces le dais al Execute CALL y entraréis de pleno en la zona del buen chico, así pues, estamos interesados en que el salto se produzca. La solución: Invertir el salto je 40124c por jne 40124c.

Para ello cambiaremos el 74 07 por 75 07. Nos colocamos encima de la línea y WDasm nos informa en la parte inferior que se trata de la dirección-offset @offset000843h, o sea, 843.

Los que tengáis Ultraedit como editor hexadecimal deberéis darle al Ctrl+G y entrar 0x843 y os situaréis justo en 74 07, cambiáis por 75 07 y guardar!!!

Hay otro método para buscar la cadena de bytes que nos interesa. Podemos buscar los bytes en hex. 3B C3 74 07 E8 18 01 00 00 EB 9A y cambiar el 74. Esta técnica es algo más delicada ya que deberemos asegurarnos de que esta secuencia sólo se encuentra una sola vez en todo el programa. Si hubiera más de una cadena de bytes similar, no sabríamos cuál es la que nos interesa!!

Una vez guardados los cambios, el programa admitirá cualquier chorrada que le introduzcamos.

Un detalle. Cuando practiquemos con Softice, tendremos acceso "en vivo" a lo

que se está comparando en "cmp eax, ebx". Veremos qué contiene EAX y qué contiene EBX. Se intuye pues, la diferencia de técnicas entre el LISTADO MUERTO y el DEBUGGEO EN VIVO?

PS: Como habéis visto, en este caso, teníamos claro que queríamos que se produjera el salto. Había otra forma de lograrlo, aparte de invertir el 74 por un 75. Podíamos haber cambiado el 74 por EB (JMP), que significa "saltar siempre", es decir, se trata de un salto incondicional, el cual nos traslada inevitablemente a la zona deseada, sea cual sea el resultado de la comparación.

• CONSIDERACIONES FINALES.

¿ES "NORMAL" LA ESTRUCTURA DE ESTE PROGRAMA EN LO CONCERNIENTE A LA ZONA DEL MENSAJE DE ERROR?

En realidad, cuando buscamos un mensaje de error con WDasm, lo más frecuente es que, algunas líneas más arriba, encontremos un: *Referenced by an (U)nconditional JUMP . Es decir, que al mensaje de error se ha llegado desde un jump (je, jne).

Fijaos que en nuestra práctica se llegaba a error por un *Referenced by a CALL. En otras palabras, no se llegaba a error por un salto, sino por una LLAMADA. No deja de ser una pequeña variación de lo habitual, pero tengamos en cuenta que lo "normal" es lo de la primera posibilidad.

Esto nos lleva a la última consideración... ¿QUÉ ES UN CALL? Cuando la máquina lee las instrucciones de un ejecutable, al llegar a un CALL XXXXXX, se desplaza hasta la dirección XXXXXXX y empieza a leer lo que allí se encuentra y sigue, sigue,.. hasta encontrarse la instrucción RET (return) que le devuelve al sitio donde había el CALL XXXXXX. En cierto modo es como una especie de salto momentáneo para ejecutar/comprobar/comparar hacer lo que sea,...y regresar al sitio inicial. Dentro de los CALLS suelen generarse los códigos reales, comparaciones entre el código chungo y auténtico,...y al regresar, la máquina lleva una serie de indicaciones que le harán tomar una u otra dirección al encontrarse, por ejemplo un je o un jne,... Muchas veces os encontraréis con secuencias del tipo:

call xxxxxxx (el programa salta a xxxxxx, hace una serie de cálculos y regresa)
test eax, eax
jne xxxxx

En nuestro Crackme1, el call 00401362 iba a una zona en que se establecía que se debía mostrar el mensaje de error, por tanto era un CALL a evitar. Cosa que hemos logrado invirtiendo el salto condicional que le precedía. Mr. Nobody os aconseja jugar un poco con los CALL. Poneos encima de uno, dadle a EXECUTE CALL XXXXXX y veréis como os traslada a XXXXXXX. Allí habrá una serie de instrucciones y tarde o temprano llegaréis (si vais descendiendo) hasta un RET , que se supone que nos devolverá a la dirección donde empezó el CALL XXXXXXX. Por cierto, un CALL puede contener en su interior otros CALLs (con sus propias direcciones y RETs). En

cierta manera tenemos que verlo como una especie de cuadro jerárquico. Habría pues un CALL-madre que en su interior contendría muchas instrucciones, nuevos calls,...pero que al final todo acabaría devolviéndonos al sitio en que se había iniciado todo.

Ejemplo:
push....
call XXXXXX (contiene instrucciones, calls,.....) y acabamos volviendo a...
test eax, eax
jne.....

Es frecuente ver que a lo largo de las instrucciones de un programa haya un CALL XXXXXXX que se repita con cierta frecuencia. Imaginaos que en XXXXXX hay unas instrucciones que verifican la autenticidad de un código y devuelven una respuesta positiva o no. EL programador puede hacer que esta sección del programa sea llamada con un CALL XXXXXXXX al iniciarse el programa, más tarde también en la sección del cuadro de diálogo que permite introducir un código para registrarse, o también, por ejemplo al intentar realizar una acción sólo disponible en la versión registrada,...

Bfff... basta por hoy, fale?

****

Bueno, habréis observado que el fichero que Mr. Nobody ha dejado en es.binarios.misc contiene un fichero llamado crackme2. Bueno, se trata de otro trabajito de Cruehead. ¿Creéis que la técnica para "arreglarlo" va a ser muy diferente de la de su Crackme 1.0, tratándose del mismo autor? Es hora de comprobarlo. Ya tenemos deberes para el fin de semana.

PSS: ¿No es una gozada desensamblar un programa tan pequeñito?

PSSS: ¿Se empieza a ver algo de luz al final del túnel?? :)

PSSSS: Muy pronto retomaremos el Winrar desde donde lo habíamos dejado.

EXAMEN LECCIÓN 2

Buenas!!!!! Aquí estamos con un bonito examen sorpresa. A ver qué tal nos sale. Trabajamos con el código de LVIEW PRO 1c5/32 (no creo que esté ya disponible en la red, puesto que es algo viejito ya)

Estaría muy bien que alguien (tan solo 1) se molestara a responder las preguntas y enviarlas. ¿Quién se atreve? Si hemos asumido los contenidos de la lección 2, nos resultará fácil. Sólo hay que prestar atención a las direcciones, ¿no?

Atención: Echemos un vistazo a esta zona de mal-chico que hemos sacado del programa. A continuación podemos responder al pequeño cuestionario final.

Direcciones hex. instrucciones en ensamblador
:0041EDC1 E86AFEFFFF call 0041EC30

```
:0041EDC6 83C408              add esp, 00000008
:0041EDC9 85C0                test eax, eax
:0041EDCB 751A                jne 0041EDE7
```

* Possible StringData Ref. from Data Obj ->"User name and ID numbers do not" ->"match, please verify if name and "
->"ID# were typed correctly."

```
:0041EDCD 68388F4600          push 00468F38
:0041EDD2 57                  push edi
:0041EDD3 E8B856FEFF          call 00404490
:0041EDD8 83C408              add esp, 00000008
:0041EDDB 33C0                xor eax, eax
:0041EDDD 5F                  pop edi
:0041EDDE 5E                  pop esi
:0041EDDF 5B                  pop ebx
:0041EDE0 81C478020000        add esp, 00000278
:0041EDE6 C3                  ret
```

* Referenced by a (U)nconditional or (C)onditional Jump at Address: |:0041EDCB(C)

```
:0041EDE7 8D44240E            lea eax, dword ptr [esp+0E]
:0041EDEB 8D8C2454010000      lea ecx, dword ptr [esp+00000154]
:0041EDF2 8D542424            lea edx, dword ptr [esp+24]
:0041EDF6 50                  push eax
:0041EDF7 51                  push ecx
:0041EDF8 52                  push edx
:0041EDF9 E842C5FFFF          call 0041B340
:0041EDFE 83C40C              add esp, 0000000C
:0041EE01 E8DAF8FFFF          call 0041E6E0
:0041EE06 8BF0                mov esi, eax
:0041EE08 85F6                test esi, esi
:0041EE0A 745E                je 0041EE6A
:0041EE0C 8B44240E            mov eax, dword ptr [esp+0E]
:0041EE10 8D8C2454010000      lea ecx, dword ptr [esp+00000154]
:0041EE17 25FFFF0000          and eax, 0000FFFF
```

* Possible StringData Ref. from Data Obj ->"RegInfo"

```
:0041EE1C 8B15D08D4600        mov edx, dword ptr [00468DD0]
:0041EE22 50                  push eax
:0041EE23 51                  push ecx
```

* Possible Reference to Menu: MenuID_0003

```
:0041EE24 6A03          push 00000003
:0041EE26 6A00          push 00000000
:0041EE28 52          push edx
```

(disculpad si el pedazo de código es demasiado largo, seguro que no hacía falta todo)

****
CUESTIONARIO
****

PREGUNTA 1: Nos encontramos en plena zona del mal-chico (mensaje de error). Habréis observado que hay un salto condicional justo antes del mensaje de error. ¿Crees que este salto se produce cuando entramos un código falso? Razonemos nuestra respuesta.

PREGUNTA 2: Hay (como mínimo) dos maneras para asegurarse que el programa llegue a la zona del buen-chico (en este caso, creedme, reginfo...). Coméntalos.

PREGUNTA 3: Supongamos que no queremos utilizar WDasm para conseguir la dirección hex. en la que hay que aplicar los cambios. ¿Qué método alternativo podríamos utilizar para hallar los bytes hex. que nos interesan y cambiarlos con el editor hexadecimal?

REPUESTAS EXAMEN

PREGUNTA 1: Nos encontramos en plena zona del mal-chico (mensaje de error). Habréis observado que hay un salto condicional justo antes del mensaje de error. ¿Crees que este salto se produce cuando entramos un código falso? Razonemos nuestra respuesta.

NO, si entramos un código falso, eax y eax serán iguales, con lo que el programa no efectuará el salto y entraremos en la zona de mal-chico

```
:0041EDC9 85C0          test eax, eax
:0041EDCB 751A          jne 0041EDE7
```

* Possible StringData Ref from Data Obj ->"User name and ID numbers do not "->"match, please verify if name and "->"ID# were typed correctly."

PREGUNTA 2: Hay (como mínimo) dos maneras para asegurarse que el programa llegue a la zona del buen-chico (en este caso, creedme, reginfo...). Coméntalos.

En este caso NOPEAR no nos serviría de nada, ya que nos interesa el salto
1.- Cambiamos jne por je con lo que saltaría si son iguales (75 por 74).
2.- Cambiamos jne por jmp con lo que saltaría siempre (75 por EB).

PREGUNTA 3: Supongamos que no queremos utilizar WDasm para conseguir la

dirección hex. en la que hay que aplicar los cambios. ¿Qué método alternativo podríamos utilizar para hallar los bytes hex. que nos interesan y cambiarlos con el editor hexadecimal?

Pues buscamos la cadena E8 6A FE FF FF 83 C4 08 85 C0 75 1A

```
:0041EDC1   E86AFEFFFF          call 0041EC30
:0041EDC6   83C408              add esp, 00000008
:0041EDC9   85C0                test eax, eax
:0041EDCB   751A                jne 0041EDE7
```

Bueno... eso creo

Agradecería a Mr. Nobody que me contestara si he aprobado o por el contrario no he acertado ni una, ya que sin tener el programa... aun funciono un poco con la teoría del ensayo y error.

DEBERES PARA FIN SEMANA

ftp://ftp.rediris.es/mirror/simtelnet/win95/graphics/asimg10.zip
ftp://ftp.tas.gov.au/pub/simtelnet/win95/graphics/asimg10.zip
ftp://sunsite.anu.edu.au/pub/pc/simtelnet/win95/graphics/asimg10.zip
ftp://ftp.univie.ac.at/mirror/simtelnet/win95/graphics/asimg10.zip

Elegir el que queráis. Asmallimage es un pequeño programa (635k) cuyo autor no ha leído nada sobre los peligros de hacer aparecer un mensaje de error y no incluir chequeos iniciales en el código. Ello implica que el señor 74-75 es suficiente para resolver todo el meollo con cualquier dato que le sea introducido.... Una lástima, una vergüenza

Si alguien no resuelve el ejercicio, que lo manifieste y Mr. Nobody lo explicitará en forma de capturas de pantalla (que, en caso de ser necesario, se postearían a es.binarios.misc

Ya queda menos para acabar lo del Winrar y llegar a la lección 3

APÉNDICE Final / lección 2 – Cursillo Otoñal de Crackeo (22/10/99)

Bienvenidos nuevamente a la parte final de nuestra lección 2, donde acabaremos de completar nuestros estudios sobre Winrar. Antes de empezar, no obstante, sería bueno recordar qué hemos estado viendo recientemente:

*La técnica del salto invertido 74/75 nos valió para forzar a Winrar a mostrarnos un mensaje de "Thanks for registering", aunque posteriormente, al reinicializar el programa, comprobábamos que había algún chequeo que devolvía el programa a su estado original.

*Mr. Nobody sugirió el estudio de otros programas como Copyto, Fontlook o Techfacts que serán analizados en breve y que, de no haber cambiado su perfil de protección, responderán también a la llamada del 74/75

*Hicimos un pequeño examen sobre el código de LView Pro 1c5/32 que fue magistralmente resuelto por uno de los "inscritos". La técnica del 74/75 también resolvía el "problema".

*Sugerimos el programa Asmallerimage 1.0 como ejercicio interesante en que el 74/75 nos llevaba al éxito. No perdamos de vista este programa, porque será objeto de estudio al llegar al uso del debugger Softice, muy, muy pronto.

*Vamos allá pues...a resolver lo pendiente que teníamos con Winrar.

La primera idea que debería habernos quedado clara con el uso del "listado muerto" de WDasm es que las cadenas de texto contenidas en Strng Data Ref.. nos son de gran utilidad. Al localizarlas, les hacemos doble clic y nos vemos trasladados a la zona en que aparecen. Nuestra misión consiste, pues, en detectar qué salto les ha llevado hasta allí y conseguir invertirlo con la ayuda de un editor hexadecimal. No sólo son susceptibles de ser invertidos los saltos hacia mensajes como "Invalid code", sino que cualquier otra cadena de texto puede ser evitada o forzada a aparecer. Al reiniciar Winrar nos reaparece el mensaje Evaluation Copy en la barra superior, pues bien, nuestra primera tarea consistirá en localizarlo entre las StringDataReferences, hacer doble clic y reflexionar sobre cómo podemos evitar tal cadena de texto. Esto es lo que tenemos:

****
Winrar 2.50 Beta 2
****

```
:004134F1 833D7CF0440000        cmp dword ptr [0044F07C], 00000000
:004134F8 752F                  jne 00413529

* Possible Reference to String Resource ID=00873: "evaluation copy"
                                                                    |
:004134FA 6869030000            push 00000369
:004134FF E84030FFFF            call 00406544
```

En este caso nos lo han puesto muy fácil. El salto condicional jne previo al mensaje desagradable apunta hacia 413529. Está claro que si se produce, nos enviará lejos de 4134fa (evaluation copy). ¿Qué haremos pues para conseguir nuestro propósito? Invertir el salto con un 742F o obligar a saltar EB27.

Si nos colocamos sobre la línea con WDasm, en la parte inferior nos aparecerá la dirección hex. que deberemos buscar con nuestro editor hexadecimal. Recordemos que en 0012AF8h (en mi caso), la -h sólo indica que se trata de "hexadecimal" y los 0000.. no tienen importancia.

Si aplicamos los cambios y los guardamos, al volver a arrancar Winrar ya no

tendremos que ver más mensajes de "evaluation copy", que resultan muy desagradables para la vista humana ya que parece como si quisieran recordarnos en todo momento que somos pobres :)

Muchos tutoriales sobre el crackeo de Winrar lo dejaban aquí. Menos mal que ya más de uno se ha percatado de que a pesar de la inversión de lo de "invalid code" y lo de "evaluation copy", el programa mantiene una serie de limitaciones.

En realidad no son ningún obstáculo para quien quiera usar Winrar y..seguramente, la compañía creadora de esta utilidad sólo mantienen estas limitaciones a nivel simbólico. Les encanta que la gente utilice su programa, con lo cual ponen unas limitaciones a determinadas opciones que son vitales para disfrutar del producto en sí.

Son las siguientes: Nos aparece un mensaje de "Available only in registered version" en las siguientes situaciones

menu-options >settings >general> log errors to file
menu-options >settings >compression> put authenticity verification
menu-commands > add files to archive > Erase destination disk before...
menu-commands > add files to archive > put authenticity verification

Una vez más se comete el error de darnos un mensaje de error (válgame la redundancia), puesto que podremos localizarlo sin problemas entre las StringData: Available in registered version only.

No obstante, antes de ser demasiado optimistas, Mr. Nobody se ha permitido el lujo de utilizar un programa fantástico denominado Search&Replace y ha buscado cuántas veces aparece este mensaje a lo largo del código desensamblado de winrar. Tiemblen!!

```
******************************
Processing file: D:\W32DASM\W32DASM\WPJFILES\WinRAR.alf

Line 6257 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 6424 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 6443 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 6621 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 7013 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 7164 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 7183 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 7321 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
```

Line 8063 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 8079 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 8651 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 8778 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 9009 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 9868 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 10790 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 21565 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 22173 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 22646 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 22706 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 24036 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 28973 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 29241 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 29815 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 29976 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 33632 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 38648 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 38841 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"
Line 59644 - * Possible Reference to String Resource ID=00106: "<Available> in registered version only"

Found 28 occurrences.
Searched 1 file(s), found 28 occurrences in 1 file(s)

****

El mensaje aparece 28 veces a lo largo del listado de WDasm. De hecho, si hacemos doble clic en StringData, nos traslada a la primera localización, un nuevo doble clic a la siguiente, etc, etc.....

Nosotros sólo estamos interesados en invertir el salto que lleva a cuatro de estos mensajes, pero por cuál empezamos, ¿lo invertimos todo? En realidad si echamos un vistazo a las diferentes zonas en que aparece "Available in reg." observaremos que no todas ellas están precedidas por un salto condicional. En algunos casos ni tan sólo aparece por ahí cerca algo que poder invertir. Fijémonos en dos ejemplos:

```
************************************
          Este parece fácil
************************************

:00403F03 85C0          test eax, eax
:00403F05 740A          je 00403F11
:00403F07 6A01          push 00000001

* Possible Ref to Menu: MAIN_MENU, Item: "Unselect group    Gray -"

* Possible Reference to Dialog: CMDWNDADD, CONTROL_ID:006A, ""

* Possible Reference to String Resource ID=00106: "Available in registered version
only"
                |
:00403F09 6A6A          push 0000006A
:00403F0B 53            push ebx


************************************************
Pero aquí vienen un par realmente poco claros para nosotros
************************************************

:00403FAD 833D7CF0440000    cmp dword ptr [0044F07C], 00000000
:00403FB4 7524              jne 00403FDA
:00403FB6 6A30              push 00000030

* Possible Ref to Menu: MAIN_MENU, Item: "Change drive   Ctrl-D"

* Possible Reference to Dialog: ABOUTRARDLG, CONTROL_ID:0065, ""

* Possible Reference to String Resource ID=00101: "Warning"

:00403FB8 6A65          push 00000065
:00403FBA E885250000        call 00406544
:00403FBF 59            pop ecx
:00403FC0 50            push eax


* Possible Ref to Menu: MAIN_MENU, Item: "Unselect group    Gray -"

* Possible Reference to Dialog: CMDWNDADD, CONTROL_ID:006A, ""

* Possible Reference to String Resource ID=00106: "Available in registered version
```

only"

```
:00403FC1 6A6A                push 0000006A
:00403FC3 E87C250000          call 00406544
:00403FC8 59                  pop ecx
:00403FC9 50                  push eax
:00403FCA 53                  push ebx

* Reference To: USER32.MessageBoxA, Ord:0000h

:00403FCB E8C3710400          Call 0044B193
:00403FD0 6A00                push 00000000


* Possible Ref to Menu: MAIN_MENU, Item: "Unselect group    Gray -"

* Possible Reference to Dialog: CMDWNDADD, CONTROL_ID:006A, ""

* Possible Reference to String Resource ID=00106: "Available in registered version
only"
                                                                              |
:00403FD2 6A6A                push 0000006A
:00403FD4 53                  push ebx

**************************************
```

¿Qué hacemos? Podemos intentar ir a lo loco a por los que parezcan fáciles como el primer ejemplo, invertirlos y luego arrancar Winrar para ver cuál de los 4 mensajes de "Available..." ha desaparecido. Evidentemente, cuando desparezca el mensaje "Available..", la opción estará habilitada para su uso.

La opción de "ir a lo loco" es válida, requiere algo de paciencia y siempre se puede dar con lo que buscamos aunque resulte un poco fatigante. En este caso, Mr. Nobody, cansado de invertir cosas y comprobar después que ninguno de los cuatro mensajes desaparecía, ha optado por hacer algo de trampa, al utilizar una herramienta en la cual nos introduciremos muy pronto en la lección 3, se trata de Softice.

Con este debuggeador uno puede dar la orden de que se detenga la ejecución del programa en determinadas situaciones. En su caso, Mr. Nobody ordenó a Softice que detuviera la ejecución de Winrar en el momento en que apareciera un messagebox (la típica cajita de error con el texto "Available...").

Al detener la ejecución de Winrar en el preciso momento de aparecer el mensaje de error, se tiene acceso a la información sobre en qué parte del programa nos encontramos. Entonces, no es precisó más que anotar la dirección en que nos encontramos e ir al listado de WDasm a buscar qué se cuece por ahí, cuál ha sido el último salto condicional que se ha producido.

Estos fueron los resultados:

menu-options >settings >general> log errors to file = zona 403FCB
menu-options >settings >compression> put authenticity verification = zona 40437F
menu-commands > add files to archive > Erase destination disk before...= zona 40a970
menu-commands > add files to archive > put authenticity verification = zona 40a93c

Ahora ya sabemos dónde se encuentra el meollo real y no la paja inútil. Veamos el primer ejemplo:

```
**************************+
:00403FAD 833D7CF0440000        cmp dword ptr [0044F07C], 00000000
:00403FB4 7524                  jne 00403FDA << salta más allá de error
:00403FB6 6A30                  push 00000030

* Possible Ref to Menu: MAIN_MENU, Item: "Change drive    Ctrl-D"

* Possible Reference to Dialog: ABOUTRARDLG, CONTROL_ID:0065, ""

*  Possible Reference to String Resource ID=00101: "Warning"

:00403FB8 6A65                  push 00000065
:00403FBA E885250000            call 00406544
:00403FBF 59                    pop ecx
:00403FC0 50                    push eax

* Possible Ref to Menu: MAIN_MENU, Item: "Unselect group     Gray -"

* Possible Reference to Dialog: CMDWNDADD, CONTROL_ID:006A, ""

* Possible Reference to String Resource ID=00106: "Available in  registered version only"

:00403FC1 6A6A                  push 0000006A
:00403FC3 E87C250000            call 00406544
:00403FC8 59                    pop ecx
:00403FC9 50                    push eax
:00403FCA 53                    push ebx


* Reference To: USER32.MessageBoxA, Ord:0000h

:00403FCB E8C3710400            Call 0044B193
:00403FD0 6A00                  push 00000000

* Possible Ref to Menu: MAIN_MENU, Item: "Unselect group     Gray
                                                                  |
* Possible Reference to Dialog: CMDWNDADD, CONTROL_ID:006A, ""
```

* Possible Reference to String Resource ID=00106: "Available in registered version only"

:00403FD2 6A6A push 0000006ª
:00403FD4 53 push ebx

* Reference To: USER32.CheckDlgButton, Ord:0000h

:00403FD5 E887700400 Call 0044B061

* Referenced by a (U)nconditional or (C)onditional Jump at Address: 00403FB4(C)

:00403FDA B801000000 mov eax, 00000001 >>>el salto viene hasta aquí!!!lejos de error
:00403FDF E969010000 jmp 0040414D

****

Fijémonos en que aparecen dos referencias a "Available..." y que más arriba aparece un salto condicional que, de producirse, nos enviaría un par de líneas más abajo de "Available.." con lo que evitaríamos el mensaje de error.

Quedémonos con el detalle de que el salto de dirige a una zona cuya primera instrucción es mov eax, 00000001.

Aquí sólo tenemos el ejemplo del primero de los cuatro mensajes, pero es igualmente válido para los tres restantes, cuya estructura es prácticamente idéntica. Sólo tenemos que buscar un lugar con dos referencias a "Available.." cercanas a una anotación del tipo * Reference To: USER32.MessageBoxA y que tenga algo más arriba un JNE que se dirige hacia una línea después de todo el meollo.

Los interesados pueden divertirse buscando los tres mensajes restantes a invertir (recordad, o 74 o EB puesto que lo que queremos es saltar). Por cierto, si queremos hacer una reflexión más profunda, podemos quedarnos con el detalle de que el salto deseado no se ha producido porque en la comparación

cmp dword ptr [0044F07C], 00000000 el valor que se esconde en la dirección de memoria 44f07c es 0.

Reflexionemos: el programa compara lo que hay en 44f07c con 0. y a continuación decide salta si no son iguales. ¿Ha saltado? No. Ello indica que 44f70c contiene un 0. Los 4 mensajes "Available..." aparecen después de una comparación idéntica. ¿Qué pasaría si en 44f07c hubiera un 1? Pues que sin necesidad de retocar el salto condicional, este saltaría (lejos del mensaje "Available..") puesto que 44f07c i 0 no serían iguales.(recordad = jne = salta si no son iguales)

¿Hay posibilidades de meter un 1 en la dirección 44f07c? Sí. pero para ello debemos esperar a que llegue el turno de Softice. Creed a Mr. Nobody. Con sólo meter un 1 en esa zona, todas las comparaciones acaban devolviendo resultados positivos para nosotros. Con mucha frecuencia el programa se interesa por saber si "el

cajoncito EAX" contiene un 0 o un 1 después de un CALL o si en determinada dirección de memoria hay un 0 o un 1. ¿Os imagináis que vienen a representar estos numeritos?
Habitualmente 0 = no registrado / 1= registrado (estamos generalizando, por supuesto!) A esta doble posibilidad se la denomina FLAG, o sea, la banderita que nos anuncia si SI o NO. Precioso, ¿verdad?

Eso es todo por el momento. Ya queda menos para la lección 3. Probablemente haremos un pequeño estudio de algún otro programa que se deje manosear por el señor 74 o el señor 75.

La reflexión que nos debe ocupar ahora es: ¿cómo es posible que conociendo sólo un par de instrucciones en ensamblador se puedan hacer estas cosas? Lo cierto es que hay un montón de programadores que procuran evitar que sus programas cedan ante un 74/75 y eso es, como mínimo, respetable. Cuando una utilidad cae con un simple 74/75 sólo podemos pensar que no tienen interés en proteger el producto ni un mínimo. Sería algo similar a cuando te muestran una imagen pornográfica súper-fuerte e incorporan una diminuta estrellita en un punto minúsculo para simular que "así ya no se verá nada" :) Son las comparaciones de Mr. Nobody.

Lección 3 – Cursillo Otoñal de Crackeo (03/11/99)

SOFTICE, "LA" HERRAMIENTA

Bueno, por fin hemos llegado al punto en que nos atrevemos con un peso pesado. Si hemos seguido con detenimiento las instrucciones de la lección 2, en estos momentos, estaremos más o menos en disposición de entender (aunque sea más o menos) que con WDasm podemos conseguir localizar cadenas de texto (como mensajes de error, por ejemplo), hacer doble clic en ellas, llegar al pedazo de código en que aparecen estas cadenas, localizar el salto que ha permitido llegar hasta esta zona e invertirlo para evitar esta fatalidad.

Por ejemplo al intentar registrar un programa obtenemos el mensaje "error". Desensamblamos el ejecutable con WDasm y buscamos en StringDataReferences “error" (doble clic) zona de "error" en el listado de código con WDasm (busquemos algo más arriba)

*Referenced by an (un)conditional jump at XXXXXX

Goto Code Location XXXXXXXX

Llegamos a una zona en que aparece: XXXXXX jne xxxxxx anotamos la dirección hex. e invertimos la instrucción con un editor hexadecimal.

Pues bien, las acciones que pueden llevarse a cabo con Softice van mucho más allá. Podríamos decir que WDasm nos permitía acceder al "listado muerto" del código que se ejecuta, mientras que con Softice se accede al código "en vivo", en el mismo momento

en que las cosas suceden, y esto nos confiere muchas posibilidades, como la de alterar las cosas en memoria (sin realizar cambios físicos con un editor hexadecimal) o la de ver qué datos se están moviendo de un sitio para otro, qué se guarda, qué se compara, qué se quiere esconder, ...:)

INSTALAR SOFTICE: Tradicionalmente se ha considerado que la instalación de Softice es algo complicadilla, aunque si nos dejamos asesorar por algún tutorial al respecto veremos que no reviste tantos problemas como se pudiera pensar inicialmente. Son sitios de visita muy recomendada las páginas del buen Craaaaack el Destripador, así como los que ofrece WKT en su site: http://welcome.to/craaaack http://free.prohosting.com/~ecd/ecd/IndexGenericos.htm

Un resumen de aquello **a tener en cuenta** lo tenemos a continuación:

1. Al iniciar la instalación se nos pregunta por el directorio en el que queremos instalar la bestia: c:\ice (por ejemplo).
2. Seleccionar el driver de video: Habilitar la opción Universal Video Driver puede evitarnos muchos problemas
3. Seleccionar Mouse (com2, ps/2,..)
4. Es aconsejable seleccionar "Do no make any changes" para que no se realicen cambios en nuestro autoexec.bat, ya que es mucho más cómodo retocarlo nosotros mismos, preparándonos una opción que nos permita arrancar Windows con o sin Softice (no siempre estaremos craqueando, ¿verdad?)

El grupo WKT sugiere que añadamos al final de nuestro autoexec.bat las siguientes líneas, que nos darán 3 segundos para escoger si queremos cargar Softice antes de Windows (pulsando S) o si dejaremos que Windows se cargue sin Softice.

Editar el AutoExec.Bat, y añadir al final :

```
CHOICE /T:N,3 "Cargar Debugger Soft-Ice "
IF ERRORLEVEL 2 GOTO SEGUIR
C:\ice\WINICE.EXE
:SEGUIR
```

(de lo que acabamos de decir, se deduce que Softice debe ser cargado antes del propio Windows)

Lo único que nos hace falta ahora es hacer unos pequeños retoques al fichero winice.dat que encontraremos en el directorio en que esté Softice: Editemos dicho fichero (si queréis desde MS-dos con "edit winice.dat") y...

1- Eliminemos los ; que están delante de las siguientes líneas:

```
EXP=c:\windows\system\kernel32.dll
EXP=c:\windows\system\user32.dll
EXP=c:\windows\system\gdi32.dll
EXP=c:\windows\system\advapi32.dll
```

2- Retoquemos la cifra que corresponde a la memoria ram de que disponemos.
3- Dejemos preparada la configuración del programa para cuando entremos en él. La línea siguiente informa al programa de las ventanas que queremos visualizar:

INIT="CODE ON;LINES 60;WR;WL;WD 13;WC 25;WATCH eax;WATCH *eax;X;"

Evidentemente podemos escoger otro tipo de visualización, con más o menos ventanas. En este caso, Mr. Nobody ha escogido la sugerida por WKT, que es la que comentaremos dentro de poco. En ella se especifican las líneas que queremos ver y de qué ventanas.

****

PRIMER ENCUENTRO CON SOFTICE: Al reiniciar nuestro ordenador, si hemos optado por la configuración sugerida más arriba,nos encontramos con que al ser leída en el autoexec.bat la línea CHOICE /T:N,3 "Cargar Debugger Soft-Ice " , dispondremos de 3 segundos para pulsar S. A continuación, se cargará Softice y después de él, Windows.

Una vez en Windows, si queremos podemos echar un primer vistazo a nuestro programa dándole al Ctrl+D. Un pantallazo nos enviará a un sitio extraño lleno de numeritos y cosas sin sentido. ¿Qué es todo esto? Allá van los nombres....

**********+
VENTANA DE REGISTROS

EAX EBX ECX EDX .....

VENTANA VALOR EAX (no es imprescindible, aunque sí práctica)

EAX=0

VENTANA DE DATOS

INCLUYE
DIRECCIONES : BYTES HEXA.....:DATOS MEMORIA

VENTANA DE CÓDIGO

INCLUYE
DIRECCIONES: INSTRUCCIONES EN HEXA :INSTRUCCIONES EN ENSAMBLADOR

(TIPO LISTADO WDASM)

VENTANA DE COMANDOS

Aquí entramos las instrucciones

****

Con Ctrl+D regresaremos a Windows dispuestos a realizar nuestra primera práctica.

Supongo que hay maneras mucho más científicas para explicar lo que viene a continuación, así que disculpad la terminología de Mr. Nobody. Creámonos que Softice es un debugger, un programilla que nos permite ejecutar una a una las instrucciones de un programa. Evidentemente no es una herramienta diseñada para craquear, sino para que los programadores depuren los códigos de sus creaciones, arreglen errores,...

Nosotros nos aprovecharemos de sus facultades debuggeadoras para ver el funcionamiento de los programas cuyas protecciones estudiemos. Para ello nos beneficiamos del hecho de que los programas que conocemos suelen realizar toda una serie de actividades utilizando una serie de funciones (API) propias de Windows.

Digamos que dichas "funciones", "acciones" o como queráis llamarlas son llevadas a cabo por todas estas DLL a las cuales les hemos quitado el;

Todos hemos visto alguna ventana con un mensaje de error al intentar registrar un programa. En realidad para que este hecho tuviera lugar, el programador ha dado instrucciones para que se llamara a la API Messageboxa, que es la encargada de traernos la ventana con el texto correspondiente.

Para que un programa compruebe si el número que se le ha introducido al registrarse es correcto o no previamente debe recibir la instrucción de recoger el número. Este hecho tiene lugar con el uso de una API llamada Getwindowtexta o Getdlgitemtexta (y otras..)

Reflexionemos: Sabemos que los programas usan APIs para realizar determinadas funciones (mostrar ventanas, preparar cuadros de diálogo, recoger texto escrito,...), y sabemos que estas APIs están , por decirlo así, contenidas en determinadas DLLs. Lo que nosotros hacemos es configurar Softice para que esté habilitado para controlar estas DLL. Nuestro amigo Softice ve pasar por debajo de él muchas cosas y nosotros tenemos la oportunidad de decirle: Detén el proceso en cuanto ocurra tal cosa. ¿No es maravilloso?

Veamos un ejemplo: Vamos a decirle a Softice que detenga la ejecución de un programa en el momento en que se haga una llamada a la API Messageboxa (mostrar una ventana con algún texto):

Ejecutamos nuestro conejillo de indias crackme1 de Cruehead (el original, sin cambios hechos con el editor hexadecimal!!! :), rellenamos la ventana con nombre y código y...antes de darle a OK, hacemos una visita a Softice y le damos la orden.

Ctrl+D > aparecemos en Softice. Anotamos: bpx messageboxa (+intro) (sólo es posible anotarlo en la ventana de comandos, está claro)

NOTA: bpx significa "breakpoint on execution". Por lo tanto, bpx messageboxa significa "detener el proceso al ejecutarse la API Messageboxa"

Con Ctrl+D (o F5) regresamos a Windows, justo ante el cuadro de diálogo le damos a OK y.....PLLOMMM, de nuevo en Softice!!!

¿Qué ha pasado? La API Messageboxa ha sido llamada por nuestro crackme1.exe y Softice, muy obediente, ha detenido el proceso. En este momento nos encontramos dentro de user32 (lo sabemos al mirar la fina línea verde con el texto:___user32 .+ code!_____)

Para ver desde donde se ha llamado esta API, debemos darle a F11 o a F12. Con ello regresamos a Windows y nos encontramos con la Messagebox de error. Si le damos a Aceptar, regresamos a Softice y ahora sí estamos en pleno código de crackme1.exe. Lo podemos comprobar echando un vistazo a la fina línea de color verde que así nos lo indica. Estas instrucciones que podemos ver en la ventana de código son las mismas que obtenemos con el listado de WDasm, pero la diferencia es que aquí están "vivas", podemos desplazarnos por ellas pulsando F8 o F10. Fijémonos que hemos aterrizado justo aquí (<<<<<), después que la Messageboxa fuera llamada

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

:0040136B 6860214000            push 00402160

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

:00401370 6869214000            push 00402169
:00401375 FF7508                push [ebp+08]

* Reference To: USER32.MessageBoxA, Ord:0000h

:00401378 E8BD000000            Call 0040143A (llamada a messageboxa)
:0040137D C3                    ret <<<<<<<<<<<<<<<<<<<
```

(Este código está extraído de WDasm. Fijaos en que las direcciones coinciden con las que vemos en Softice, sólo cambia esa porción que precede a las direcciones XXXX:00401378. Estas XXXX corresponden al CodeSegment (CS) que no es permanente y dependen de la aplicación y la memoria usada en Windows. En mi caso yo leo 0137:0040137D, pero lo que nos importa es la dirección que figura a continuación 401378)

¿Para que sirve todo esto? Como mínimo nos ha servido para "entrar" en el código del ejecutable que queremos estudiar. La idea de "entrar" es muy importante, porque una vez dentro, podemos analizar qué determina que se produzca o no un salto, hacia donde se han movido nuestros datos, con qué se están comparando....

Vamos a realizar una pequeña práctica "en vivo" con nuestro mismo crackme1 de Cruehead. ¿Os acordáis que fue objeto de estudio en su momento? Os hago un poco de memoria: En el listado muerto de WDasm vimos que el mensaje de error estaba precedido por * Referenced by a CALL at Address: :00401245

Si íbamos con GotoCodeLocation a 401245 observábamos que había un CALL que en realidad era el responsable del mensaje de error. ¿Cómo podíamos evitarlo?

```
:00401241 3BC3              cmp eax, ebx
:00401243 7407              je 0040124C >>>>>>>!!!!!!
:00401245 E818010000        call 00401362 >>CALL MENSAJE DE ERROR
```

En realidad todo dependía de un salto condicional previo, que debíamos invertir con el editor hexadecimal para conseguir saltar a "victoria". Os recuerdo que nos interesaba "saltar", ya que si no se producía el salto, llegábamos al CALL, que era lo mismo que llegar a ERROR.

Nuestra práctica va a consistir en lo siguiente. Vamos a repetir lo del bpx messageboxa (no hace falta volverlo a anotar). Esperaremos que Softice rompa en user32, le daremos a F11 o F12, volveremos a Windows, le daremos a "Aceptar" y regresaremos a Softice. Como ya estaremos en el código de nuestro amigo crackme1.exe, aprovecharemos para poner un nuevo breakpoint. Esta vez no ordenaremos paralizar ninguna API, sino que, como ya estamos DENTRO, ordenaremos al programa que se detenga en 401243, justo donde está el salto condicional crucial!!

Para ello anotaremos bpx 401243 (+intro) (estando dentro del código de crackme1.exe - fijémonos en la línea verde).

Para saber cuántos breakpoints tenemos metidos, basta con darle a bl (brekpoint list(?): Observaremos que nuestro primer bpx ha sido numerado como 0 y el nuevo breakpoint como 1.

Puesto que nuestro amigo bpx messageboxa ya no nos interesa vamos a deshabilitarlo con un bd 0 (deshabilitar breakpoint 0). Ello significa que este breakpoint ya no estará activo. (para habilitarlo nuevamente utilizaríamos un be 0 +intro) Si le damos nuevamente a bl observaremos que aparece con un * que le precede, indicando que no está activado. Mientras que el nuevo bpx 401243 está activado completamente.

Regresamos a Windows con Ctrl+D, rellenamos nuevamente la ventana para registrarse, le damos a OK y...bingo...aparecemos en Softice, justo en la línea que nos interesa:

```
(ventana de código)
:00401243 7407              je 0040124C  (no jump)
```

(bajo Softice los je aparecen como jz (jump-if-zero) y los jne aparecen como jnz (jump-if-not-zero), aunque su valor sigue siendo el de 74 (o 0f 84) y 75 (of 85) respectivamente)

Fijémonos que nos informan de que este salto no va a producirse. Con lo que acabaremos llegando al CALL maligno. Si le diéramos a F10 para avanzar una línea de instrucciones más, acabaríamos recibiendo el mensaje de error.

EMPIEZA LA MAGIA.....

Llegados a este punto, estamos en disposición de alterar lo inevitable. Vamos a hacer que este salto no se produzca y vamos a ver cómo se puede conseguir de diferentes maneras (nótese que todos los cambios que hagamos se producirán en memoria, no físicamente. No es como cambiar bytes con el editor hexadecimal, sino que los cambios son sólo momentáneos)

ALTERNATIVA 1 (la más cómoda)

El programa sabe que no tiene que producirse el cambio porque ha realizado una serie de comprobaciones y un FLAG, una banderita, le indica que no está autorizado a saltar. Bien. Nosotros tenemos la posibilidad de decirle al programa que haga lo contrario, invirtiendo este FLAG, este indicador de "si" o "no". Anotemos en la línea de comandos: R FL Z (+Intro) (invertir el zeroflag)

Inmediatamente vemos lo siguiente:
:00401243 7407 je 0040124C (jump)<< Ahora sí va a saltar

Si dejamos que el resto de instrucciones de ejecuten, todo irá bien. Démosle a F5 y...Premio. Estamos de vuelta en Windows con el mensaje de éxito!!!!!

ALTERNATIVA 2 (cambiar los bytes como con el editor hexadecimal)

Para este segundo método vamos a probar un nuevo sistema para entrar en el código. Destruyamos previamente todos nuestros breakpoints con un bc * (+intro). Cerramos el crackme y vayamos a buscar el icono de Symbol Loader..dentro del grupo de programas de Numega Softice.

File > Open module : buscar el fichero crackme.exe
Module > Load > Nos dirá "error..." : clic en "Sí"

Estamos en Softice, que ha cargado nuestro programa crackme1.exe desde el principio. Démosle una vez al F10 y comprobaremos que estamos en el código que nos interesa plenamente. Es el momento de anotar bpx 401243 (+intro). Con F5 regresamos a Windows y dejamos que el programa se ejecute. Introducimos nuestros datos, le damos a OK y....De nuevo en Softice justo en nuestra línea preferida.

Esta vez vamos a cambiar (en memoria) los bytes correspondientes al salto condicional.
Anotamos en la línea de comandos: e 401243 (+intro)

Con ello editamos los bytes correspondientes a esta dirección de memoria. Fijémonos como en la parte de la VENTANA DE DATOS, los bytes hex. se han habilitado para que realicemos un cambio. Estamos justo encima del 74 07 ...

Sólo tenemos que cambiar 74 por 75 (como cuando lo hacíamos con nuestro editor hex.)y darle a Intro. Los cambios se hacen efectivos. En la ventana de datos je se ha convertido en jne. Es el momento de darle al F5 y volver a degustar el mensaje de victoria

ALTERNATIVA 3 (escribiendo nuestra primera instrucción en ensamblador)

Para repetir todo el proceso netamente, volvemos a eliminar (bc 0) el breakpoint, salimos del crackme, volvemos a cargar el programa con el Symbol Loader y recolocamos el bpx 401243. F5 para volver a Windows, rellenar la ventana de registro, OK y.....Softice de nuevo.

En esta ocasión introduciremos nuestro propio código en ensamblador. Siempre sobre la línea 00401243 7407 je 0040124C
Vamos a la ventana de comandos y anotamos a 401243 (+Intro) (entrar instrucciones ASM en esta dirección)

Atención porque ahora vamos a entrar los datos en ensamblador en la misma línea de comandos y cuando términos, se harán efectivos en la ventana de código. Anotemos:
jne 0040124C (+Intro)

Softice espera que le introduzcamos más instrucciones, pero como ya estamos le damos un par de veces a Escape y ya está. La nueva instrucción está ahí. Un nuevo F5 nos traslada a Windows donde re-saborearemos la victoria.

Por ahora no hemos hablado demasiado de instrucciones en ensamblador, pero para cualquiera que haya leído la lección 2 no le será extraño lo que acabamos de anotar: sinoesigual-salta a 40124c.

Por cierto, también podríamos haber entrado un jmp 0040124c ya que lo que queremos realmente es saltar. Os lo dejo a vosotros para que practiquéis.

****
¿Decepcionados? Espero que no. En nuestra próxima lección pasaremos a la acción de lleno con Asmallerimage 1.0 que espero todos tengáis. Entonces será el momento de aclarar un poco más la diferencia entre F8 y F10, la utilidad de F4,...Por ahora Mr. Nobody considera que la información proporcionada aquí ya es suficiente para alguien que no se haya estrenado aún con Softice. Si algún concepto ha sido explicado con demasiada puerilidad o de forma imprecisa sólo es debido a la voluntad de "hacerlo algo más fácil" y a la incompetencia inherente a una persona que crackea sin conocimientos sólidos de ensamblador, programación e informática en general, sólo por intuición y mimetismo con respecto a lo que ha leído por ahí fuera :)

Lección 4 – Cursillo Otoñal de Crackeo (16/11/99)

"ACARICIANDO A LA BESTIA" (SOFTICE: Invirtiendo Saltos y Aprendiendo Comandos)

Si hemos conseguido aprender alguna cosa de la lección 3 estaremos perfectamente capacitados para poner en práctica nuestros progresos. Recordaremos que en la anterior lección aprendíamos a familiarizarnos con la idea de "entrar en el código de un programa" al aprovecharnos del uso que estos hacen de las APIs propias de Windows. Además pudimos comprobar cómo Softice era capaz de invertir saltos condicionales, aplicando cambios en las instrucciones del código de un programa (**aunque sólo en memoria**).

Vayamos a ver un ejemplo práctico con el programa Nico's Commander 4.01 (nc401.zip).-programa cuyo código real de registro está al alcance de todo aquel que visite sites de serialz, pero que a nosotros no nos incumbe.

Para nuestro ejercicio contaremos con la inestimable ayuda de WDasm. Iniciamos el programa, nos piden un código para registrarnos, introducimos cualquier chorrada y obtenemos el mensaje de error "Invalid Registration Code" (primer error grave del programador). Desensamblamos el ejecutable con WDasm, buscamos nuestro mensaje en las String Data References, lo encontramos, le hacemos doble clic y aparecemos en una zona similar a:

***Reference by a (U)ncoditional or (c)onditional**
Jump at Address :0041ce3d

***Reference to string "Invalid Message...."**
PUSH por aquí
MOV por allá
PUSH por más allá.....
YMUCHASCOSASMÁSQUENOENTENDEMOS

Está claro que se ha llegado a "error" desde la dirección 41ce3d. Vayamos al menú Goto Code Location y anotemos 41ce3d. Esto nos lleva directamente a la línea:

0041ce3d 75 19 jne 0041ce58 (salta a error)

Ya que estamos encima de esta línea, WDasm nos dice de qué offset se trata. La dirección hex. (ver parte inferior de la pantalla) parece ser **0001c23dh** (sólo nos interesa 1c23d)

Si abrimos el ejecutable con un editor hexadecimal, nos dirigimos a la dirección 1c23d y cambiamos el byte 75 por 74 (invirtiendo pues el salto condicional), creeremos que ya está todo terminado, pero....

Le damos al ejecutable y....no arranca!! Premio para el programador. Consciente de la existencia del arte crackeatorio, ha optado por incorporar una especie de chequeo para comprobar que no hay cambios en el fichero original. Como hemos cambiado un byte, el tío no arranca ya que se ha dado cuenta. Procedemos a devolver al ejecutable su estado original (75), guardamos y observaremos que se puede arrancar nuevamente .El señor Nico se merece ya nuestro respeto por el hecho de haber adoptado esta medida de precaución, pero la ha cagado enormemente al desconsiderar la posibilidad de que el programa puede ser abordado desde Softice.
Fijémonos en que ya sabemos la dirección en la que deseamos pararnos (41ce3d, no confundir con la dirección hex.). Procedamos a cargar el programa con Symbol Loader. Después de los típicos mensajes de error, acabamos entrando en la primera línea de código del programa con Softice. Bien. Ya estamos dentro!! Sólo nos queda decir donde queremos detenernos anotemos "bpx 41ce3d" (+intro). La trampa ya ha sido preparada. Le damos a F5 para regresar a Windows, introducimos cualquier tontería, le damos a OK y !!fgggoooommm!!! de nuevo en Softice, justo en la línea que nos interesaba. Ahora no ha habido problemas de chequeo porque el programa está en su estado original, no hemos cambiado nada,...aún,...y los cambios que realicemos tendrán lugar en memoria, no físicamente (como en el caso del editor hexadecimal).

Estamos en el lugar preciso, en pleno código del ejecutable que nos interesa (ver fina línea color verde). ¿Cómo cambiamos el salto? Recordad las tres modalidades:

1.- Anotando **R FL Z (+intro**) Es decir, cambiando el sentido del flag cero (¿se dice así?)
2.- **e 41ce3d** y cambiando el 75 por 74 en la ventana de datos. (posteriormente le damos dos veces a Escape)
3.- **a 41c3ed**: y anotamos **JZ 0041CE58** (+Intro). Nosotros escribiendo en ensamblador, ¿quién lo hubiera dicho? Le damos dos veces a Escape y listos

Una vez realizada una de estas tres operaciones, podemos eliminar ya nuestro breakpoint (con un simple bc *) o lo inhabilitamos con un bd * y le damos a F5 para volver a Windows. Taachhhaaa!!! Mensaje de éxito.

¿Será perdurable? Salimos del programa, lo volvemos a ejecutar y todo está en orden. Victoria!!! ¿Cómo ha sido posible? En realidad, Nico ha cometido un error grave al dejar que Softice se pudiera encargar del asunto, además nos lo ha puesto muy fácil ya que hemos podido desensamblarlo con WDasm, sin ningún problema. No ha procurado encriptar o comprimir el ejecutable. Por si fuera poco, ha programado su utilidad de forma que si se llega a visualizar el mensaje de éxito, una clave es introducida en el registro y queda allí como indicador de que nos hemos registrado. Para desregistrarnos tendremos que eliminar la clave siguiente del registro
de Windows (a mano con el regedit.exe)

\HKEY_CURRENT_USER\Software\NicoCuppen\NC\WinPositie\wp.flag wp.flags"=dword:00000001
>>> el 1 indica que estamos registrados!!!

Creo que la versión 4.02 del producto tiene la misma "protección" y se supone que será mejorada en el futuro. Recordad que esto no es más que un ejercicio que nos ha proporcionado un programa shareware. Una bonita actividad de ingeniería invertida. No estaría bien apropiarnos de este producto sin conseguir una licencia por parte del autor. Seamos honestos, continuemos usando el Explorer de Windows o solicitemos una licencia al autor, que se lo ha currado.

****
Aprovechando que estamos en plena faena, vamos a rememorar cómo podemos movernos por Softice. (Mr. Nobody advierte que la terminología utilizada en este apartado, así como determinadas descripciones futuras pueden ser técnicamente erróneas. Su desconocimiento de los vocablos exactos puede llevar a los auténticos maestros que puedan leer esto a descojonarse a costa de su ignorancia. Disculpadle. Recordad que en COC se crackea por instinto e intuición, no por ciencia (¡!ojalá pudiéramos!!). A pesar de ello, sería de agradecer cualquier tipo de puntualización sobre detalles mal explicados. Cualquier rectificación recomendada será gratamente bien recibida.)

Eliminemos la clave del registro wp.flags y démosle nuevamente a Nico's Commander. Vuelve a pedirnos el número de registro. Esta vez aprenderemos una nueva forma de entrar. Sabemos que los programas que piden códigos utilizan determinadas APi's de Windows **Getwindowtext** (para 16 bits) **Getwindowtexta** (para 32 bits) o bien **Getdlgitemtext** (para 16 bits) **Getdlgitemtexta** (para 32 bits)

Vamos a poner unos cuantos breakpoints para ver cual funciona en este caso concreto. Ante el cuadro de diálogo de entrar código, introducimos cualquier chorrada y antes de darle a OK le damos a Ctrl+D y entramos en Softice. Anotamos:

bpx getwindowtexta (+intro)
bpx getdlgitemtexta (+intro)
(trabajamos con 32 bits)

Con toda probabilidad, uno de los dos nos funcionará (si no fuese así recurriríamos a un casi infalible bpx hmemcpy que veremos más adelante)

Con Ctrl+D regresamos a Windows. Le damos a OK y !!plim!! de nuevo en Softice!! No estamos aún dentro del código de nc.exe, sino en user32! (observemos la línea verde). El programa ha llamado a user32 para que hiciera uso de su API getwindowtexta para recoger los datos del cuadro de diálogo. Aquí se ha detenido el proceso. Nosotros queremos volver al sitio desde donde se ha llamado esta API. Para ello le damos a F11 (o a F12) y !!bingo!! ahora sí estamos en pleno código de nc!.exe ¿Qué hacemos?

(f10) Desplacémonos por el código del programa dándole a F10, F10, F10... Estamos haciendo que las instrucciones se ejecuten una a una. Fijémonos en la parte superior cómo los registros EAX, EDX, ECX,.... van cambiando sus valores. Bonito, ¿verdad? No nos preocupemos por el hecho de no entender lo que está pasando, sólo disfrutemos con el suave deslizamiento de las instrucciones. Los saltos condicionales o incondicionales saltan de un sitio a otro, aparecen instrucciones raras como PUSH, CMP, ADD, LEA,.....

(f4) Si queremos echar una miradita a como está Windows, podemos darle a F4, aunque sólo para ver, de hecho continuamos en Softice. Con otro F4 regresamos al mundo oscuro de Softice.

(f8) Vayamos traceando (F10) hasta llegar a algún CALL y quedémonos encima de esta instrucción. Una vez llegados aquí, tenemos dos opciones: darle a F10, con lo cual todas las instrucciones que contiene este CALL se ejecutarán de golpe y pasaremos a la siguiente línea o bien darle a F8, con lo que nos introduciremos en la dirección a la que apuntaba el CALL. Ahora estaremos en otro sitio y podemos seguir traceando con f8 o con f10. ¿Queda clara su diferencia? Cada uno de ellos sirve para desplazarse por el código, aunque F10 ejecuta de golpe todo lo que contiene un CALL, mientras que F8 nos sirve para entrar en el CALL y desplazarnos también por las instrucciones línea a línea. Hasta que lleguemos a un RET que nos devolverá a la línea posterior al CALL en el que nos hemos introducido y podremos continuar allí donde lo dejamos.

(comando G :) Podemos desplazarnos a la dirección que deseemos con G dirección_deseada. No se trata de un breakpoint exactamente, sino una especie de ir_hasta, sin que por ello haga falta luego deshabilitar este comando. Pongamos un ejemplo. Mr. Nobody ha apreciado una instrucción interesante en la dirección 421fc8. Podríamos poner un bpx 421fc8 (+intro), darle al F5 y seguro que nos detendríamos allí. Luego eliminaríamos el breakpoint con bc x (siendo x el número de breakpoint que sea: 0, 1, 2,...). No obstante lo más rápido, si ya estamos dentro del código, es darle a G 421FC8 (+Intro). Directamente aparecemos en: 013F:00421fc8 push ecx >> Se está pusheando ecx. Podríamos decir que se está preparando-cargando lo que contenga ecx para lo que sea.

(comando D :) En este momento podemos saber qué contiene ecx con un simple D ECX. La ventana de datos nos mostrará qué esconde el registro ECX. En este caso, nuestro código chungo!!! Como se podrá suponer este comando nos será muy útil cuando se compare un código correcto y uno chungo con una instrucción del tipo CMP EAX, ECX . Con un D EAX y D ECX veremos qué esconde cada uno de ellos. Por cierto, si continuamos traceando con F10 hacia la instrucción siguiente: LEA ECX, [EBP-10] y la sobrepasamos, veremos que D ECX ya nos da unos datos distintos. Esta

instrucción ha cargado el contenido de la dirección EBP-10 en el registro ECX, con lo que los datos previos que contenía han desaparecido.

Unas líneas más abajo, hallaremos un PUSH 00466544. ¿Qué se debe estar pusheando ahora? Un nuevo D 466544 nos permitirá ver que en esta dirección de memoria se halla el texto Nico's Commander, precisamente el que preside la ventana de nuestro mensaje de error. Un par de F10 más nos llevan directos al mensaje de Invalid Registration... ya en Windows.

¿BUSQUEMOS NUESTRO CÓDIGO?

Reiniciemos nuestra operación. Si queremos deshabilitamos los breakpoints con un bc*, regresamos a Windows con Ctrl+D, rellenamos el cuadro de diálogo con un código chungo, regresamos a Softice y habilitamos de nuevo las trampas con un be* y volvemos a Windows para darle al Ok. Un F12 nos permite entrar por fin en el código de nc!.exe, movámonos con f10.... Si no nos hemos desplazado demasiado (hasta llegar al Mensaje de Error), puede que sea un buen momento para realizar la curiosidad de buscar nuestro código falso. Tendremos la oportunidad de ver en qué sitio de la memoria se ha guardado. Anotemos **S 0 L FFFFFFFF** "codigo_chungo" (+intro)-[respetemos lo de mayúsculas-minúsculas]- y pronto nos aparecerá en la ventana de datos. Allí es donde se aloja nuestro amigo, dispuesto a ser recogido, reconvertido, matematiqueado hasta ser comparado con lo que se esperaba que pusiéramos.

Con frecuencia, nuestro código chungo residirá en más de un lugar. Para comprobarlo anotamos S (+intro) y Softice realizará una nueva búsqueda. Si no me equivoco, nos lo volveremos a encontrar en otra dirección de memoria. Si realizamos una nueva búsqueda con S (+intro) nos encontraremos con algo parecido a codigo_chungo""""""..... que parece ser algo así como un eco de nuestra propia búsqueda, nada interesante.

(movernos por la ventana de datos y la de código) Con ALT + arriba, abajo nos desplazaremos por la ventana de datos, accediendo así a la información que hay por la zona colindante. Si deseamos ver las instrucciones en ensamblador previas o posteriores a la línea en la que nos encontramos, no tenemos más que darle a CTRL. + arriba, abajo para desplazarnos por dicho entorno.

Eso es todo por hoy. Nada del otro mundo, aunque muy pronto veremos que algo tan simple como el comando de búsqueda S nos puede ser suficiente para hallar el código real que andamos buscando. ¿Se puede ser tan poco prevenido? La respuesta es sí. ¿Se pueden obtener códigos de registro sin saber prácticamente nada de ensamblador? La respuesta vergonzosa sigue siendo , sí. Obviamente no todo el mundo es tan poco precabido, pero es evidente que hay más de algún programador que debería tener una copia de Softice para ver lo que jamás debería hacerse. Personalmente no creo que debamos reírnos de ellos . El hecho de atreverse a programar merece todas nuestras alabanzas, pero las cosas hay que hacerlas con más ganas. A nadie se le ocurriría diseñar una caja fuerte en la que al introducir un código se nos dijera "usted no ha introducido el código 5556784532 por tanto no es correcto" :) Son las comparaciones de Mr. Nobody.

Lección 5 – Cursillo Otoñal de Crackeo (23/11/99)

" DE CAZA CON LA BESTIA (SOFTICE) "

Ya estamos aquí de nuevo. Vamos a continuar con el análisis de protecciones defectuosas con nuestro buen amigo el Sr. Softice. Intentaremos aprender nuevas formas de entrar en el código de un programa y qué es lo que nunca deberíamos hacer si nos dedicáramos a programar.

El programa ejemplo del que nos valdremos para ilustrar una protección desacertada es A SMALLER IMAGE 1.0 . En realidad se trata de una utilidad cuya función es tan ínfima (cambiar el tamaño de imágenes) que es comprensible que la compañía responsable no se haya molestado en protegerla debidamente. De hecho, este programa forma parte una utilidad de más envergadura que (se supone) debe estar más protegida. Es una perfecta pieza fácil para los amantes del desensamblado con WDasm y la inversión del salto que lleva a error, pero en esta ocasión aprovecharemos para estudiarlo desde otra perspectiva.

El cuadro de diálogo de registro nos invita a introducir un nombre de usuario y una registration key. Fijémonos en el error inicial de advertirnos que no valen letras. Cuando intentamos introducirlas en el apartado de registration key, nos damos cuenta de que sólo se admiten números. Vamos allá...

NAME : CURSILLETE
REGKEY: 123123123

Le damos a OK y nos aparece un mensaje de error. En menos de un segundo el programa se ha dado cuenta de que nuestro código era incorrecto. Para empezar vamos a poner un **BPX MESSAGEBOXA** para ver lo que pasa.

Fijémonos en el hecho de que si entramos en Softice, será que se habrá procedido a llamar al mensaje de error, lo cual significará que todo ya habrá acabado, ya se habrán hecho todas las comprobaciones y habremos llegado a la zona del mal chico (error).

Ctrl+D (estamos en Softice) Anotamos **BPX MESSAGEBOXA** (+intro)
Ctrl+D (hemos vuelto a Windows). Reintroducimos los datos, damos a OK y....

!!De nuevo en Softice!! No estamos aún dentro del código del programa, sino en **----user32!!-----**(ver línea verde). Un simple F12 nos devolverá al sitio desde donde se ha llamado la messageboxa. Aparecemos en Windows, le damos a aceptar y...ahora sí estamos en pleno **----asmallerimage.exe!-----.**

¿Qué hacemos? Se supone que todo ha concluido. Ya nos han enviado a freír espárragos. ¿Seguro? Eso es lo que cabría suponer, no obstante vamos a ver si en la memoria aún se conserva algún resto de los datos que se han manejado. Procedamos a hacer una búsqueda de nuestro nombre:

s 30 L FFFFFFFF "CURSILLETE" (+intro)

(una curiosidad: si introducimos como nombre alguna palabra que también figure en documentos que tengamos en el escritorio o en datos del registro de Windows, puede ser que Softice los encuentre, en cuyo caso, no estaríamos encontrando lo que

pretendíamos. Tenemos las opción de darle a S para que vuelva a buscar o podemos optar por utilizar un nombre diferente. Al parecer buscar cadenas como 1234567 no suele ser muy recomendable porque podemos acabar topando con datos de Winzip. Se ve que por ahí en la memoria hay demasiadas cosas, por lo cual, en caso de tener que buscar algo, se aconseja que se trate de una secuencia única y original. Muy probablemente, si optamos por el nombre ABCDEFG, encontraremos demasiadas referencias a ello en la memoria.

j o d e r ! ! ! Nuestra búsqueda ha tenido éxito. Acabamos de topar con nuestro nombre, pero qué **cojones es este código de nueve dígitos** que aparece algo más abajo??? ¿Es lo que parece?? La respuesta es SÍIII!! Nos están dando el código válido para el usuario CURSILLO, ..muy mal!!

Centrémonos. Démonos cuenta de lo grave de la situación. Nos han enviado ya el mensaje de perdedores, nosotros buscamos restos de nuestro nombre en la memoria y resulta que nos aparece con su código correspondiente casi al lado, en plena ventana de datos!! Está claro que lo intolerable en este caso radica en el hecho de no haberse molestado a borrar nada. Imperdonable!!! Si es necesario que se genere un código real para hacer las comparaciones, al menos procuremos borrarlo todo una vez realizadas, ¿no?

Obviamente no suelen darse casos de descuidos tan flagrantes en el mundo de la programación, aunque existen otros ejemplos. El programa FONT LOOK es otro buen ejemplo de descuido. En este caso, no obstante, aprenderemos otra nueva forma de entrar en el código. Veamos cómo va la cosa....

Deshabilitamos nuestro anterior breakpoint en Softice con un **bd 0** (en caso de que sea el breakpoint 0).

Vamos a preparar una trampilla al bueno de FONTLOOK 3.5(otro programa cuyo autor ha demostrado estar desinteresado en proteger su producto en lo que parece más bien un intento calculado de marketing de hacerse popular en el mundillo informático. Supongo que el autor prefiere que la gente use su programa y a costa de esta popularidad irse ganando adeptos legales en oficinas, bufetes, despachos, escuelas, empresas....Bien hecho!)

BPX GETWINDOWTEXTA (+intro)
BPX GETDLGITEMTEXTA (+intro)

(vamos a ver si alguno de estos breakpoints funciona después de introducir los datos y darle a OK)

:(Fiasco!! Parece ser que ninguno de los dos funciona. Si le damos a OK, no entramos en Softice. Recurriremos pues a un nuevo breakpoint: **BPX HMEMCPY** (+intro)

Previamente habremos rellenado el cuadro de registro con cualquier chorrada (KURSILLO). Entraremos en Softice, eliminaremos los dos anteriores (BC 1 2) y anotamos **BPX HMEMCPY** (+intro) . Volvemos a Windows, le damos a OK y !!bingo!! Ahora sólo hace falta darle a F12 hasta que lleguemos al código de **---fontlook!exe--.** Pasamos por kernel, user(0a), user(1c)..hasta que nos aparece una fina línea verde sin ningún nombre (al menos en mi caso) que nos indica que estamos en el código del ejecutable deseado.

Traceemos con F10 (pasaremos por mogollón de RETs y mov, lea,...) hasta que nos aparezca el mensaje de error. Estamos en Windows. Le damos a Aceptar y regresamos a Softice. Busquemos nuestro código

S 30 L FFFFFFFF "KURSILLO" (+intro)

(el primero que encuentra no tiene nada interesante por los alrededores. Volvamos a buscar..) **S** (+intro)

!!!EEEEEPPPP!!!! La segunda aparición de nuestro código en la ventana de datos viene acompañada de un precioso código de siete dígitos que no es más que el serial correcto!!!! ¿Será posible?

(de hecho, unas líneas antes de saltar a error, el código ya estaba accesible:

En **013f:450278 mov eax, [ebp-08]** Después de esta instrucción un **D EAX** nos mostrará nuestro código (si lo anotamos en minúsculas, ahora estará pasado a mayúsculas). Aquí se ha movido el contenido de [ebp-08] hacia el registro EAX. Unas líneas más abajo
En **013f:450294 mov edx, [ebp-08].** Después de esta instrucción podremos darle a **D EDX** y tendremos el código real. Se ha movido el contenido de [ebp-08] en el registro EDX)

Nos queda como ejercicio comprobar si otro programa del mismo señor, DIRECTORY PRINTER está "protegido" de la misma forma. En caso de no ser tan sencillo, podemos armarnos de paciencia y después de cada MOV EAX,xxx O LEA EDX,XXX ir dándole al D EAX, D EDX,..para ver qué es lo que se va moviendo por ahí y seguro que tenemos éxito.

No olvidemos que para nosotros no es necesario trapichear con códigos auténticos de programas. Al menos esta no ha sido nunca la intención del CURSILLO OTOÑAL DE CRACKEO.

Cualquier persona con intenciones de programar que haya leído algún tutorial de crackeo se habrá dado cuenta de varios detalles que no debería pasar por alto: Debería pues olvidarse de dejar un hardcoded suelto por entre el código visible con un simple desensamblazo con WDasm, procurar empaquetar el ejecutable de forma que no fuera posible desensamblarlo con WDasm, intentar evitar la típica estructura de instrucciones en las que si no saltas a error, saltas a victoria, etc.....

Bien. Después de la lectura de la presente lección 4 será evidente que no es nada recomendable que un programador dé instrucciones a su programa para que calcule en función del nombre de usuario un código correcto y que luego lo compare con el código que haya sido introducido. Desde luego hay muchas formas de llevar a cabo este proceso y ciertamente, algunas no son nada fáciles de seguir, pero lo que sí resulta abominable es que se calcule un código correcto y se deje allí mismo a la vista de todo aquel que quiera echar una ojeada.

****
SECCIÓN OPCIONAL, PARA AMANTES DE LAS EMOCIONES FUERTES
****

Ya que hemos estado trabajando con A smaller image 1.0 quizás sea el momento para practicar un bellísimo ejercicio de ingeniería inversa: Convertir al programa en nuestro propio generador de claves. ¿Es eso posible para una persona que no ha programado nada de nada en su vida? La respuesta empieza por "s".

La clave de este ejercicio consiste en que localicemos el momento en que se prepara el mensaje de error dentro del programa para ser expuesto dentro del messagebox. Si entramos en el código del programa después de que ha pillado nuestros datos, localizaremos la instrucción en 40145C.

2 CAMPOS, 2 BREAKS

Para entrar en el programa utilizamos un **BPX GETWINDOWTEXTA**. Regresamos a Windows, le damos a OK y....Cuando saltemos a Softice, debemos tener en cuenta que nuestro nombre habrá sido recogido, pero aún faltará por recoger nuestro código, así que volvemos a darle a F5 para que rompa nuevamente. Efectivamente se han producido dos breaks sobre la **API getwindowtexta** (2 campos, 2 breaks). Ahora ya podemos darle a F12 y ya estaremos en pleno código.

Traceamos con F10 hasta llegar a:

0137:0040145C (bytes:68 40 41 43 00) push 00434140

Si le damos a D 434140 veremos en la ventana de datos el mensaje de error "**This product is still unregistered...**". Es evidente que ya se ha decidido que íbamos mal y se está pusheando/cargando el mensaje de error. Quedémonos quietecitos aquí mismo.

Recordaremos que nuestro código real aún debe estar por ahí en algún sitio. Busquemos nuestro nombre y estará por ahí cerca ¿no?

S 30 L FFFFFFFF "CURSILLO"

Efectivamente, ha aparecido!! nuestra búsqueda nos lleva a un sitio en que figura nuestro nombre y más abajo el código real (si no lo vemos, sería conveniente que le diéramos a ALT + flechaarriba-abajo para movernos por la ventana de datos). Supongamos que nuestro código real es 444555444. Hagamos una nueva búsqueda

S 30 L FFFFFFFF "444555444"

Lógicamente la búsqueda ha tenido éxito y es más, ahora disponemos de forma más clara de su dirección de memoria. Sólo tenemos que observar la parte más a la izquierda de la ventana de datos para ver que estamos en 00434960.

Reflexión: ¿Qué pasaría si le dijéramos al programa que, en lugar de **PUSH 00434140** (= mensaje de error) ejecutara un **PUSH 00434960** (=código válido)? Teóricamente en lugar de mostrar el mensaje de error, nos mostrará el código adecuado para nuestro nombre de usuario!!!! puesto que pushearía la dirección en que se halla el código válido en lugar de la dirección que contiene el mensaje de error!!! :)

****
HACIENDO QUE EL PROGRAMA SE COMPORTE COMO NOSOTROS QUEREMOS
****

Es el momento de hacer uso del comando "A". Ya que estamos en la línea adecuada, anotemos:

a 40145c (+INTRO)(queremos escribir algo en ensamblador en esta dirección, que es el sitio en que nos encontramos precisamente)

ahora podemos introducir **PUSH 00434960** Y DARLE AL INTRO. Atención!!! En este preciso momento los bytes hex. que corresponden a dicha instrucción han cambiado (ver la ventana de código). En lugar del previo **68 40 41 43 00** tenemos un **68 60 49 43 00** (que son las instrucciones que corresponden al nuevo **push 00434960** = código válido)

Le damos un par de veces a ESC. Deshabilitamos todos los breakpoints con un bd* y si le damos a F5, estaremos en Windows contemplando un messagebox con nuestro código válido, ¿precioso, no?

Para fijar estos cambios hechos en memoria, deberemos tener la precaución de anotar un montón de bytes hex. previos, de tal manera que al realizar la búsqueda con un editor hexadecimal tengamos la certeza de que hallamos el lugar que nos interesa. Una búsqueda con sólo 68 40 41 43 00 creo que sería insuficiente porque podríamos hallar más de un sitio con esta misma secuencia. Hará falta anotar una secuencia mayor, para lo cual sólo nos hace falta anotar los bytes de las instrucciones siguientes o anteriores, realizar la búsqueda con el editor hexadecimal y cambiar sólo la secuencia **del 68 40 41 43 00** por **68 60 49 43 00** . Si guardamos los cambios, A smaller image nos irá escupiendo todos los códigos que queramos.

En caso de introducir el código real para registrarnos, los datos se guardan en : [**HKEY_CURRENT_USER\Software\TriVista\3DImageScene\Settings**]. Sólo hace falta cargarse todo el directorio de TriVista para que A smaller image vuelva a arrancar como unregistered, cosa que espero que todos hagáis ya que no somos manguis, sino curiosos, ¿verdad?

Mr. Nobody desea que la lección haya servido de algo, al menos para futuros programadores, para que sean conscientes de lo que puede llegar a "ver" una persona ajena por completo a los conocimientos mínimos del mundo de la programación, ignorante supino de lenguaje ensamblador,..pero poseedor de una copia de Softice, una de WDasm y un editor hexadecimal.

Lección 6 – Cursillo Otoñal de Crackeo (02/12/99)

"LAS COMPARACIONES NO SIEMPRE SON TAN ODIOSAS" (DE CAZA CON LA BESTIA)

Está claro que cuando un programa nos pide que introduzcamos un código para registrarlo, más tarde o más pronto va a tener que hacer una comparación de lo que se haya introducido con otra cosa. Hay muchas formas de llevar a cabo este proceso de una forma más o menos discreta, o dicho de otra manera, se puede hacer bien o mal. En este detalle radica la debilidad de muchos productos shareware que han sido ideados sin tener en consideración la existencia de desensambladores o debuggers. ¿Pereza? ("Agh! Qué palo tener que escribir más código!)

¿Soberbia? (Buaff, si la gente no tiene ni idea, ¿cómo van a descubrir mi hardcoded serial?) ¿Falta de pericia? ¿o quizás simplemente desconocimiento de la existencia del arte del crackeo? El siguiente estudio va a poner en evidencia algunas de las cosas que no deberían hacerse, al menos para evitar que personas sin conocimientos como nosotros tengan acceso a aquello que supuestamente debería estar oculto. Después de ya algunos años de tradición shareware algunos programadores no han querido enterarse de que SON altamente sospechosas instrucciones como :

```
CALL xxxxxxX ===== Aquí dentro está el tomate!!
TEST EAX,EAX
JNZ (SALTA A MENSAJE ERROR)
```

o bien:

```
CMP EAX,EDX ====== El propio "tomate" sin envasar!!!!
JNZ (SALTA A MENSAJE ERROR)
```

Tomemos una muestra de producto shareware y analicemos sus errores:
COMMAND LINE 95 V.1.0 **http://mirror.direct.ca/pub/simtelnet/win95/util/cline95.zip**

(Da toda la impresión de que se trata de un buen ejercicio para la técnica del salto invertido con WDasm, pero vamos a optar por hacer uso de "la bestia" :)

Nada más darle al ejecutable una ventana nos invita a registrarnos. Entramos en el área de registro donde apreciaremos que se nos pide nombre y código. Lo Rellenamos con cualquier chorrada:

```
NAME:   COCFRIENDS
code:   11991199
```

Antes de darle a OK, pasamos a Softice con Ctrl+D y preparamos un par de breakpoints para detener el proceso en cuanto los datos sean recogidos:

```
bpx getwindowtexta (+intro)
bpx getdlgitemtexta (+intro)
```

Con Ctrl+D regresamos a Windows. Le damos a OK y !!bien!! Estamos en Softice. El break bpx getdlgitemtexta ha sido el triunfador. En circunstancias normales, le daríamos otra vez a F5 para regresar a Windows para que fuera recogido el segundo campo (un break por cada campo : nombre, code,..), pero si lo hacemos recibiremos el mensaje de error. Todo habrá terminado. Ignoro el motivo, pero aquí sólo hay un sólo break. ¿Acaso sólo pretende leer los datos de un único campo? Da lo mismo. Continuemos....

El break que nos ha dejado en Softice pero aún no estamos en el código del programa que nos interesa. Estamos en USER32!. Un toquecito suave al F12 nos devolverá al sitio desde donde fue llamado. Bien. Ahora ya estamos en zona interesante:

```
++++++++++++++++++
* Reference To: USER32.GetDlgItemTextA, Ord:00EDh
                    |
:0040246C FF1584B24000        Call dword ptr [0040B284] >>Aparecemos aquí
:00402472 6A00                 push 00000000
:00402474 6A00                 push 00000000
:00402476 6805040000           push 00000405
:0040247B BFE0934000           mov edi, 004093E0
:00402480 56                    push esi

* Reference To: USER32.GetDlgItemInt, Ord:00ECh
                    |
:00402481 FF1580B24000         Call dword ptr [0040B280]
:00402487 A3CC924000           mov dword ptr [004092CC], eax
:0040248C 8BD0                  mov edx, eax
:0040248E B9FFFFFFFF           mov ecx, FFFFFFFF
:00402493 2BC0                 sub eax, eax
:00402495 F2                    repnz
:00402496 AE                   scasb
:00402497 F7D1                 not ecx
:00402499 49                    dec ecx
:0040249A 0FBE05E0934000      movsx eax, byte ptr [004093E0]
:004024A1 0FAFC8                imul ecx, eax
:004024A4 C1E10A               shl ecx, 0A
:004024A7 81C1CCF80200         add ecx, 0002F8CC
:004024AD 890DD4934000         mov dword ptr [004093D4], ecx
:004024B3 3BCA                  cmp ecx, edx <<<<<<<<<interesante<<<<<
:004024B5 742F                  je 004024E6
:004024B7 81FACADE6103         cmp edx, 0361DECA <<<<interesante<<<<<
:004024BD 7427                  je 004024E6
:004024BF 6A10                 push 00000010

* Possible StringData Ref from Data Obj ->"REGISTRATION CODE ERROR"
                    |
:004024C1 68C4824000           push 004082C4
++++++++++++++++++
```

Podemos seguir traceando con F10 hasta llegar a alguna zona en que se decida alguna cosa. Está claro que no muy abajo se encuentran dos saltos condicionales después de un CMP (comparación). Si lo que se está comparando es idéntico, podremos saltar hacia 4024e6. Con el listado de WDasm podremos comprobar que este salto conduce hacia el **THANX FOR REGISTERING**.... Parece claro lo que se debe estar comparando, ¿no? Fijémonos en que si no se produce el salto, llegaremos a 4024c1, el mensaje de error. Está claro que nos interesa saltar. Si optáramos por parchear, podríamos convertir alguno de los dos saltos en un JMP que nos enviaría directos a la victoria. No obstante lo dejaremos tal como está.

Desplacémonos hasta la zona de la primera comparación:

```
004024B3 3BCA                  cmp ecx, edx
```

Llegados hasta aquí podemos optar por ver qué se está comparando. Siempre está bien observar qué hay en la memoria con un D ECX o D EDX, pero en este caso no obtenemos ningún dato. En este caso el comando "?" nos revelará el contenido de estos registros. Quedémonos con el detalle que nos aporta la ventana de registros (parte superior, en caso de tener el winice.dat configurado tal como sugerimos):

eax=xxxxxxxx ebx=xxxxxxx ecx=000D70CC edx=00B6F89F esi=xxxxxxx

La respuesta está ahí arriba, pero en hexadecimal. Podríamos pillar la calculadora de Windows (opciones avanzadas) e introducirle estos datos en hex. para después ver su equivalente en decimal. No nos hará falta ya que Softice nos ofrece la misma información con el siguiente comando:

? edx (+Intro):

En la ventana de comandos nos aparecen tres informaciones:

DATOS EN HEXA	DATOS EN DECIMAL	EQUIVALENTE EN ASCII
XXXXXXXXXXXX	XXXXXXXXXXXXX	XXXXXXXXXXXXXX
00b6f89f	0011991199	"XXXX"

¿Acaso no nos resulta familiar este 11991199? Es nuestro código chungo. inevitablemente, ecx contendrá el código real. La comparación es tan evidente que no ha habido ni el deseo de esconderla dentro de un CALL, sino que se ha hecho ahí a la vista de todos!! Mal! Muy mal!

Si sondeamos ? ecx apreciaremos que en decimal equivale a **0000xxxxxx** (las xxxxxx son el código real que deberíamos haber introducido para nuestro nombre de usuario)

Démonos cuenta de que la instrucción CMP funciona "de facto" como una especie de resta. Al comparar una cosa con otra, se espera resultado cero. Si hubiesen sido iguales, la resta de dos cosas iguales sería cero. La siguiente instrucción nos lo demuestra:

004024B5 742F je 004024E6
(Salta a "victoria" si el resultado es cero, id est, si son iguales)

Parece que todo ha terminado, pero no nos demos aún por satisfechos. ¿Qué tenemos a continuación?:

004024B7 81FACADE6103 cmp edx, 0361DECA
004024BD 7427 je 004024E6

Parece ser que tenemos otra oportunidad! Volveremos a saltar a victoria en caso de que nuestro número (ahora residente en formato hex. en EDX) sea igual a la cifra hex. 0361deca!!! Para saber cuál es el número mágico sólo tenemos que darle a ? 0361DECA y obtendremos el equivalente en decimal que constituye el segundo código válido para registrar el programa.

El autor de COMMAND LINE95 no ha puesto demasiado interés en esconder la inevitable comprobación. Lógicamente, no todas las CMP que nos encontremos a lo largo de un programa van a ser comparaciones de códigos chungos y válidos. No obstante, se aprecia una falta de interés en esconder lo que está llevando a cabo. De todas formas se trata de una utilidad algo desfasada y ello podría justificar el poco esmero empleado en su elaboración. A pesar de ello, Mr. Nobody aconseja no traficar con el código real de registro del programa porque esta no es la finalidad del COC. Si se

ha conseguido, pues perfecto,.. es el momento de eliminar el programa de nuestro disco duro y pasar a buscar nuevas diversiones. No olvidemos que es la industria shareware la que nos ofrece este grato entretenimiento y debemos considerarla como una especie de recurso educativo utilísimo.

Un último apunte: En el caso estudiado hemos visto cómo el programa se molestaba en llevar a cabo varias comprobaciones de posibles códigos válidos. Hay que ser siempre un poco desconfiado, puesto que hay programadores cabreados por ahí fuera que, habiendo comprobado que sus códigos reales circulan libremente, han optado por implementar comprobaciones de códigos ex-válidos y "castigar" a quien los use. No es una técnica muy extendida pero hay casos. Los castigos van desde la imposibilidad de volver a registrar el programa (creando unas claves en el registro de Windows que lo impiden), borrado automático del programa (autodestrucción ;) o borrado de ficheros imprescindibles de C:\windows\system (una crueldad innecesaria). No todo lo que encontremos puede ser "bueno", con lo que hay que verificar las consecuencias de las comparaciones, saltos, etc.... En este caso, los dos saltos conducían hacia un lugar en que se mostraba el mensaje "THANX FOR...", (podíamos comprobarlo con el listado de WDasm), con lo cual se trataba de una ruta segura :)

****
EJERCICIO OPCIONAL
****

Si nos molestamos en intentar registrar el programa observaremos que nuestros datos se almacenan en el fichero cline.ini, creado en c:\windows. Antiguamente este era el medio de almacenamiento típico, aunque en la actualidad es mucho más habitual lo de almacenar los datos en el registro de Windows. Para des-registrar un programa nos será muy útil el programa REGMON (REGISTRY MONITOR), que nos puede dar una idea de lo que se está leyendo o creando en el registro de Windows. En otros casos, los programadores suelen recurrir a ficheros específicos para el almacenamiento de datos e información conforme estamos registrados. Ficheros *.dat, *.reg o con otras extensiones. La utilidad FILEMON nos ayuda a saber qué ficheros está buscando nuestro programa al ejecutarse.

Volviendo a lo que íbamos, vamos a intentar hacer que **COMMAND LINE95** escriba en el registro el código correcto correspondiente a un nombre de usuario que introduzcamos. Si hemos hecho una prueba inicial, veremos que en cline.ini se almacena:

```
[Command Line95]
WinXPos=0
WinYPos=0
CurDir=C:\
WinTop=0
RegName=COCFRIENDS
RegNo=XXXXXX
```

Procedamos a eliminar el tal fichero cline.ini, reiniciemos COMMANDLINE95 y reintroduzcamos los datos. El bpx getdlgitemtexta nos devuelve a la zona que ya conocíamos (después de un F12). Trataremos hasta llegar a la zona en que se decide si saltamos a error o a victoria. Vamos a asegurarnos que saltaremos hacia victoria

convirtiendo el primer jz en un JMP (también podemos asegurarnos el salto con un R FL Z):

```
++++++++++++++++++
:004024B3 3BCA                cmp ecx, edx
:004024B5 742F                je 004024E6 >> cambiar a JMP 004024E6
:004024B7 81FACADE6103          cmp edx, 0361DECA
:004024BD 7427                je 004024E6

++++++++++++++++++
```

Llegaremos hasta aquí (la zona de la victoria)

```
:004024E6 6A00                push 00000000

* Possible StringData Ref from Data Obj ->"REGISTRATION CODE ACCEPTED"

:004024E8 687C824000            push 0040827C

* Possible StringData Ref from Data Obj ->"Thank you for registering
Command "->"Line95!"
   :004024ED 6850824000            push 00408250
:004024F2 56                  push esi
* Reference To: USER32.MessageBoxA, Ord:0188h
  :004024F3 FF15C4B24000          Call dword ptr [0040B2C4]
* Possible StringData Ref from Data Obj ->"CLine95.ini"
 :004024F9 68DC804000            push 004080DC
* Reference To: KERNEL32.WritePrivateProfileStringA, Ord:0253h
```

<<<<BIEN. ESTAMOS EN LA ZONA EN LA QUE SE PROCEDE A ESCRIBIR ALGO EN CLINE.INI!!!!!

```
:004024FE 8B3D98B14000          mov edi, dword ptr [0040B198]
:00402504 68E0934000            push 004093E0

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

:00402509 68A0804000            push 004080A0

* Possible StringData Ref from Data Obj ->"Command Line95"

:0040250E 6868804000            push 00408068
:00402513 FFD7                call edi
:00402515 6A0A                push 0000000A
:00402517 A1CC924000            mov eax, dword ptr [004092CC]
:0040251C 6820954000            push 00409520
:00402521 50                  push eax  >>>>>>>>>>>>>>>>>>>>aquí!!!!!!!
:00402522 E879070000            call 00402CA0
:00402527 83C40C              add esp, 0000000C

* Possible StringData Ref from Data Obj ->"CLine95.ini"

++++++++++
```

No nos asustemos, a Mr. Nobody tampoco le dice gran cosa este listado incomprensible, pero si nos molestamos en darle al comando "D", podremos ver qué se está preparando. Traceemos con f10 y curioseemos :)

Parece claro (por las referencias) que esta debe ser la zona en la que se escriben los datos del triunfador en el fichero INI. En nuestro caso, nos interesa ver cuando se está cargando nuestro código chungo para ser colocado en el registro. Le damos a D 408068, D 4093E0. D409520,... Y podremos ir suponiendo qué va pasando.

EEEEEPPPP!!!! Al llegar a "00402521 50 push eax " podremos comprobar con un ? eax (+intro) que el programa se dispone a cargar el contenido de EAX, que resulta ser nuestro código chungo (Pensemos que hemos llegado hasta aquí con trampas, por tanto el programa cree que este es el código real que nos ha permitido registrarnos). ¿Qué hacemos? Sabiendo que en ECX aún reside el código auténtico (recordemos la famosa CMP), podríamos sustituir el PUSH EAX por un PUSH ECX y listos!!!! ¿Como se hace? Con el comando "A" de Softice. Sólo hace falta anotar:

A 402521 (+Intro) : ahora escribiremos PUSH ECX

Le damos a Esc dos veces y dejemos que todo fluya con un F5. Volvemos a Windows, ya ha tenido lugar todo el proceso, incluso ya habremos visualizado el mensaje de éxito. Si buscamos el nuevo fichero cline.ini, observaremos que contendrá el código válido correspondiente a nuestro nombre!!

Si queremos obtener más códigos válidos y hacer que el programa sea nuestro generador de claves, sólo tenemos que fijar los cambios en el ejecutable con un editor hexadecimal (2 cambios :jz (74 hex.) por JMP(EB hex.) y PUSH EAX (50 hex.) por PUSH ECX (51 hex.)), borrar el clini.ini y rellenar el cuadro de registro para que se vuelva a crear otro con nuestros nuevos datos. Cuánta belleza en cuan pocos bytes! :)

****
UN PEQUEÑO VACILEO
****

Para el interesado en parchear el programa está el simple recurso de forzar el salto hacia victoria. (jz por JMP). No obstante, vamos a marcarnos una pequeña vacilada con un retoque que acabará consiguiendo lo mismo. Veamos nuevamente el sitio clave:

```
:004024B3 3BCA          cmp ecx, edx >>>>>!!
:004024B5 742F          je 004024E6
```

Para conseguir un resultado cero que permita que el salto se realice, podemos cambiar la CMP ECX,EDX (COMPARAR CÓDIGO CHUNGO CON CÓDIGO AUTÉNTICO) por una instrucción infalible en este caso: Si anotamos cmp ecx,ecx (3b c9 en hex.), a la fuerza el resultado de la comparación será excelente y el salto se producirá, ¿no? Esta será la nueva secuencia:

```
:004024B3 3BC9          cmp ecx, ecx (compara código auténtico consigo mismo)
:004024B5 742F          je 004024E6 (saltará seguro)
```

Hay muchas más formas elegantes que pueden conducirnos a la victoria. Como decía Dante, ancho y espacioso es el camino que lleva a la perdición (sí, a la perdición de algunos programas shareware mal protegidos) :) Eso es todo por hoy amigos, hasta dentro de siete días.

Lección 7 - Cursillo Otoñal de Crackeo (14/12/99)

"COMPARACIONES Y COMPARA-VB-CIONES"

En algún sitio Mr. Nobody recuerda haber leído que para despistar a un cracker lo más importante es que el programador procure que lo que está haciendo el código del programa no sea evidente. Hay que saber camuflar las piezas importantes para que no resulten obvias para todo aquel que se moleste en echar un vistazo a las entrañas de un programa. Todo programa que nos muestra un mensaje de error al intentar ser registrado, que puede ser descompilado sin ningún problema (encriptación, compresión,.) y que incluye pedazos de código del tipo "CALL xxxx, test eax,eax , jnz salto a error" tiene muchas posibilidades de ser noqueado por cualquier newbie.

Vamos a continuar con nuestro análisis de "la comparación". A continuación analizaremos un par de programillas cuyos autores no han tenido interés en leer tutoriales de crackeo y abordaremos el tema de los programas elaborados con Visual Basic. Allá vamos.....

C O M P A R A C I O N E S

Empezamos con TOGGLE QUICK SCROLL (www.toggle.com): Iniciamos el programa, vemos aparecer un mensaje con los días de evaluación que nos quedan y a continuación accedemos a un nuevo cuadro de diálogo a través del cual accedemos a la posibilidad de registrarnos. Rellenamos los datos con cualquier chorrada:

NAME: cursillo
COMPANY: C.O.C
CODE: 121212

Le damos a OK y vemos aparecer un mensaje de error : "The registration information...is not valid". (1er error)

El programa es perfectamente descompilable con WDasm. Buscamos entre las String Data References y al encontrar el mensaje anteriormente citado le doble-clickeamos y aparecemos en:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:  004038A8(C)

:004038C1 6A10                  push 00000010

* Possible StringData Ref from Data Obj ->"The registration information you
```

Parece ser que llegamos aquí desde un salto condicional en 4038a8. Vayamos a esta dirección (Goto Code Location 4038a8):

```
**********************************
:0040389E E8CDFAFFFF            call 00403370 >>>> interesante!!!!
:004038A3 83F801                cmp eax, 00000001
:004038A6 6A00                  push 00000000
:004038A8 7517                  jne 004038C1 <<<< nuestro salto!!!
:004038AA 6A40                  push 00000040

* Possible StringData Ref from Data Obj ->"Registration is Complete!"

:004038AC 68B4944200            push 004294B4
```

```
:004038B1 E80D7D0100          call 0041B5C3
:004038B6 6A00               push 00000000
:004038B8 8BCE                mov ecx, esi
:004038BA E802130100          call 00414BC1
:004038BF EB0C                jmp 004038CD

* Referenced by a (U)nconditional or (C)onditional Jump at Address:  004038A8(C)

:004038C1 6A10               push 00000010

* Possible StringData Ref from Data Obj ->"The registration information you "
                                        ->"have entered is not valid."
**********************************
```

Si analizamos esta porción de código veremos que nuestro salto evita que lleguemos a **"Registration is Complete"**. Aparentemente con un simple nopeo en 4038a8, convirtiendo el 7517 en 9090(nop,nop), podríamos asegurarnos de que no saltaríamos a error, sino que llegaríamos a la zona de la victoria. Mr. Nobody no se ha molestado en comprobar si funcionaría, ya que en esta ocasión abordaremos el tema desde otra perspectiva:

El salto hacia el error inevitablemente tiene lugar, ¿porqué? Bueno, en la línea anterior se compara el valor de eax con 1. Si no es igual a 1 , el programa decide saltar a error. Fijémonos en que jnz significa "si no es 0" o "si no es igual". Es importante que este detalle no nos confunda. No quiere decir que el programa espere que eax sea igual a 0. Recordemos que la CMP funciona como una especie de resta:

Si los dos operandos tienen idéntico valor, el resultado es cero (esto es lo importante). Por consiguiente, si EAX equivale a 0, la comparación-resta CMP eax,1 no dará como resultado 0 (y si-no-es-cero : jnz, saltamos a error). Expresémoslo gráficamente:

CMP eax,01 >>> Si eax=0, entonces 0(eax) - 01 = -1. Resultado -1 (no-cero) jnz a error >>> saltará a error, ya que el resultado es diferente-de-cero

CMP eax,01 >>> Si eax=1, entonces 1(eax) - 01 = 0 . Resultado 0 jnz a error>>> No saltará a error, puesto que sólo lo hará si el resultado es 0, es decir, si no son iguales. Por tanto, llegamos a victoria.

¿De dónde sale el valor de EAX 1 o 0? ¿De dónde se obtiene? La respuesta está unas líneas más arriba, en:

:0040389E E8CDFAFFFF call 00403370 >> Aquí se realizan determinadas operaciones y acaban con dos resultados posibles: eax puede salir con valor 1 (indicando: todo correcto) o con valor 0 (lo cual nos llevará al mensaje de error).

Está claro que el call 00403370 es responsable de todo lo que está ocurriendo. Podemos entrar en él para ver lo que pasa. Ya que por ahora nos encontramos en WDasm, situémonos encima de esta línea y démosle al menú "Execute text" > "Execute CALL", que nos dejará en la primera instrucción de dicha función: WDasm nos deja directamente en:

```
* Referenced by a CALL at Address:
|:0040389E
|
:00403370 6AFF               push FFFFFFFF <<< la 1a instrucc.del call
:00403372 6840F14100          push 0041F140 .....
.....bla,bla....
```

Recordemos que estamos en el interior del CALL. El programa ejecuta cada una de las instrucciones que aquí aparecen (incluso otros CALLs) hasta llegar a un RET, que le devuelve al sitio de donde venimos (concretamente, a la instrucción siguiente, el famoso :004038A3 CMP
eax, 00000001).

Deslicémonos por WDasm viendo estas instrucciones en ensamblador que no somos capaces de entender. Mucho mov, lea, más calls, push,...y algo interesante:

```
....
:004033CF 83F906              CMP ecx, 00000006 >> compara ecx a 6
:004033D2 0F8C21010000        jl 004034F9>>salta si es menor (jmp if less)
```

Es cierto que no todas las CMP son comparaciones de código correcto y chungo. Esta instrucción se utiliza en infinidad de ocasiones a lo largo de un programa, pero siempre es interesante ver qué se cuece en ellas. Si estuviéramos dentro de Softice justo en esta línea tendríamos acceso al contenido de ecx y veríamos que equivale a 8 (? ecx). La palabra cursillo tiene 8 dígitos, por tanto, están comprobando que nuestro nombre tenga como mínimo 5 dígitos. La instrucción jl (jump if less) se encarga de saltar hacia 4034f9 en caso de que sea menor de 6 nuestro nombre. ¿Qué hay en esta dirección? Con el propio WDasm sobre la línea 004033D2 jl 004034F9, podemos darle al menú Execute Text >> Execute Jump y veremos que hemos saltado hacia la parte casi final de la rutina. Hay unas cuantas instrucciones y una temible instrucción XOR EAX,EAX (convierte eax a 0) seguido de un RET. Jooodderrr. Qué prisa por poner el valor de EAX a 0 y hacernos salir del CALL. Está claro que esta zona es peligrosa. Recordemos que EAX=0 saliendo del CALL equivale a salto a error. Podemos regresar a la zona en que nos encontrábamos antes utilizando el menú Execute Text >> Return from last jump

Sigamos desplazándonos por las incomprensibles instrucciones en ensamblador para ver si aparece algo interesante:

```
.....
:004033F4 0FBE0C06            movsx ecx, byte ptr [esi+eax]
:004033F8 03CE                add ecx, esi <<(suma esi a ecx)
:004033FA 0FAFF9              imul edi, ecx <<(multiplica ecx por edi)
```

Al parecer aquí están cogiendo byte a byte nuestro nombre y aplicándole operaciones matemáticas,...fale! Algo más abajo, algo interesante:

```
......
:00403401 3BF1                CMP esi, ecx
:00403403 7CDA                jl 004033DF
```

Una comparación y un salto hacia atrás!!!! (4033df está más arriba) si el resultado el menor al esperado. ¿Qué pasa aquí? Uno de los registros funciona como contador, de forma que estamos ante un LOOP que nos devuelve a la zona en que se coge un nuevo carácter del nombre para reaplicarle las operaciones matemáticas ya citadas. En el momento en que todos los dígitos hayan pasado "por el tubo", el salto jl no tendrá lugar y continuaremos en la línea siguiente (ya no habrá salto hacia atrás). Atención!! Porque esto indica que las operaciones habrán concluido y lo inevitable estará a punto de acontecer:

.....
Unas cuantas matemáticas más ....y....

```
:00403426 3BC6                CMP eax, esi (2º y grave error)
```

Esta instrucción es demasiado sospechosa para pasarnos desapercibida y mucho más después de un montón de operaciones matemáticas. Qué bonito sería poder acceder al contenido de eax y esi en dicho emplazamiento!!! ¿Cómo podríamos averiguarlo? La "bestia" tiene las respuestas.

Arranquemos Softice, carguemos el programa con Symbol Loader y en cuanto lleguemos a la primera instrucción, anotemos directamente el breakpoint que nos interesa: bpx 403426 (+intro). F5 para volver a Windows Una vez llegados al cuadro de diálogo de registro, rellenamos los datos, le damos a OK y...Bingo...!! Estamos en la línea que nos interesa. Sondeemos los datos:

? eax : 0001d97c (hex.) 000121212 (decimal) ** (ASCII)**
? esi : xxxxxxxxxxxxx xxxxxxxxxxxxxxxxxx xxxxxxxxxxx

Si eax contiene nuestro código chungo, esi contendrá nuestro código real para el nombre y compañía que le hemos indicado. Efectivamente en decimal podremos ver una código de 7 dígitos (decimal) válido.

¿Dónde está el error? Evidentemente, no puede inserirse en un código una instrucción tan explicita. ¿Para qué han servido todas las matemáticas que calculaban el código real? Han hecho toda la faena por nosotros. No ha hecho falta ni entender qué pasaba. El mismo WDasm nos ha dicho la instrucción que nos interesaba. Por supuesto, la comprobación tenía que efectuarse en algún momento u otro, pero podría haberse camuflado en el interior de otro CALL, o podría haberse recurrido a la técnica de hacer previamente una comprobación parcial, pero sin ofrecérnoslo todo en bandeja. Hay que ocultar lo que se pretende estar haciendo.

Puesto que no somos mangurrinoides, sino gente curiosa y con ganas de aprender, podremos desregistrar nuestro buen Toggle QuickScroll eliminando la siguiente clave del registro:

[HKEY_CURRENT_USER\Software\Toggle Software\ToggleQUICKSCROLL\Registration]

MÁS COMPARACIONES

Nuestro segundo ejercicio nos va a mostrar un programa que ha omitido la debida ocultación de operaciones importantes. Se trata de un producto de Software by Design. (todas sus creaciones se encuentran en `http://www.execpc.com/~sbd`).

Para los 23 programas que dicha compañía mantiene, se ha tenido muy poco interés en ocultar los detalles importantes, lo cual confiere una evidente fragilidad a tantas horas de faena. En este sentido, vamos a estudiar una de sus creaciones, con la dignidad que nos caracteriza. Sin códigos explícitos circulando por ahí, con la neutralidad que siempre hemos procurado mantener y con el compromiso sincero de que, si realmente estamos interesados en utilizar alguno de estos programas (no es mi caso), procuraremos obtener una licencia como es debido. Nuestro objeto de estudio se denomina ICON EXTRACTOR 3.3.

Iniciamos el programa, nos vamos al menú Help y le damos a Register. Rellenamos el cuadro de diálogo:

NAME: COC-FRIENDS
COMPANY: COC
CODE: 343434

El damos a Aceptar y..."An invalid software registration number was detected". Esta vez trabajaremos con la bestia :
Ctrl+D > En Softice anotamos "bpx gltdlgitemtexta" > Ctrl+D (+intro)

De nuevo en Windows, le damos a OK y regresamos a Softice. Se ha producido el primer break. Como hay tres campos, le damos a F5 un par de veces para que Softice vuelva a romper. Un F12 nos dejará en pleno código del ejecutable que nos interesa:

```
**********
:00407D27 50                push eax <<< Aparecemos aquí
:00407D28 E833790000        call 0040F660
:00407D2D 83C404            add esp, 00000004
:00407D30 8BE8              mov ebp, eax
:00407D32 56                push esi
:00407D33 E858620000        call 0040DF90
:00407D38 83C404            add esp, 00000004
:00407D3B 3D92A71901        cmp eax, 0119A792 <<<<< ??????
:00407D40 7518              jne 00407D5A

* Possible StringData Ref from Data Obj ->"Gregory Braun"

:00407D42 68BCA74100        push 0041A7BC
**********
```

Si traceamos un poco con F10, llegamos a una primera CMP altamente sospechosa, más que nada porque no parece nada usual. Se compara el valor de EAX con una extraña cifra. Sondeemos:

```
? EAX    : 22F9181 (hex.)  0036671873 (dec) **** (ASCII)
? 119A72 : 00119a72 (hex.) 36671873 (dec) **** (ASCII)
```

Eax no parece contener el código que le hemos introducido. ¿Qué ha pasado? A primera vista EAX debe contener alguna modificación que se pueda haber ejercido a nuestro código, o el valor de nuestro nombre,...y se está comparando a un valor fijo (119a792hexa). ¿Con qué finalidad? Si nos fijamos hay un salto condicional en la línea siguiente. JNE, quiere decir que si no son iguales el salto se producirá y en unos momentos veremos que nos conducirá directos a una nueva comparación similar a la primera. Qué raro.

En todo caso, ya que estamos aquí, comentemos qué sucede si invertimos el salto o lo nopeamos. No saltamos, seguimos en la línea siguiente y si le damos a F5 para que todo acabe, nos damos cuenta de que tenemos el programa registrado a nombre de Gregory Braun. Si lo reinicializamos, nos percatamos de que el proceso es permanente: estamos registrados. Grave, muy grave. Con invertir un único salto, conseguimos el registro del programa a nombre de Gregory Braun (el autor? un amigo?) y no hay ningún chequeo adicional. Nuestros datos son almacenados en el registro:

```
[HKEY_CURRENT_USER\Software\Software by Design\Icon Extractor for Windows 95/NT\Registration]
```

Para más INRI, en el registro acabará figurando también el código válido del señor Braun que nosotros no hemos introducido. Demasiadas cosas dejadas al azar. Una vez más la máquina se lo traga todo, sin chequeos y además resolviendo las cosas por sí sola.

Lo que viene es aún más grave: Dejemos que el salto nos envíe a una nueva zona. Aterrizamos en otro lugar extraño:

```
:00407D5A 3D3CCE5F0D        cmp eax, 0D5FCE3C >>>> ?????
```

```
:00407D5F 750C                jne 00407D6D
```

No hace falta decir que EAX y el nuevo numerito tampoco coinciden. Si no son iguales se producirá un nuevo salto que nos enviará a...una nueva comparación (enseguida la veremos). Ya que estamos aquí, vamos a preguntarnos qué ocurriría si invirtiéramos el salto. (no hace falta
decir que la cosa tiene sus riesgos, ¿podría ser un código blacklisteadomaldito y que fuéramos castigados por usarlo?). Si invertimos el salto (jne>je) o lo nopeamos (750c>9090) o anotamos R FL Z (+intro), podremos continuar traceando justo después de 407d5f. Todo ello nos conducirá a que acabaremos siendo usuarios registrados con el nombre y compañía que hayamos puesto, sin más comprobaciones!!!!

Para más INRI, el registro de Windows contendrá el código correcto que deberíamos haber introducido (pero que el programa ha introducido por nosotros). Sin comentarios. Dejemos las cosas como están y dejemos que se ejecute el programa normalmente. El salto jne nos ha dejado en un nuevo sitio,...con regalo incluido:

```
:00407D6D 53                  push ebx < llegamos aquí
:00407D6E 56                  push esi
:00407D6F E88C5D0000              call 0040DB00
:00407D74 83C408              add esp, 00000008
:00407D77 3BC5                cmp eax, ebp <<<<<<<<<??????
```

En el nuevo emplazamiento somos recibido como por una manada de hawaianas. Un sugerente CMP eax,ebp se contornea ante nosotros. Acariciemos estos registros con las delicadas zarpas de Softice:

```
? ebp : xxxxxx(hex.) "343434" (decimal) **** (ASCII)
```

Este primer sondeo nos indica que ebp contiene el código chungo que le hemos introducido inicialmente. Inevitablemente ? eax nos aportará (en decimal) el código real de registro (10 dígitos).

En este nuevo ejemplo, los errores radican en la innecesaria reiteración en lo de las comparaciones. Demasiadas, con pocas trabas y dejando que la máquina lo haga todo. Fijémonos que en los dos primeros casos, ha sido el propio programa que ha introducido los datos correctos en el registro en lugar de los códigos chungos. Un simple chequeo inicial posterior al reiniciar el programa habría valido para invalidar la operación. Estas precauciones hay que tomarlas !! A pesar de ello, hay que decir que no son muchos los autores que pueden permitirse crear una veintena de aplicaciones distintas. Por ello cuentan con el respeto de Mr. Nobody (aunque en el tema de la protección tendrían que haber dedicado alguna horilla más :)

COMPARAVBCIONES

Nuestro último apartado va a estar dedicado al tema de los programas elaborados con Visual Basic. Veamos....
Hay opiniones que sugieren que las aplicaciones realizadas con Visual Basic no son en realidad programas, sino un listado de llamadas continuas a unas librerías (vbrun300.dll,....) que se encargan de realizar todas las operaciones. Este hecho dificulta el traceo con WDasm ya que no se obtiene nada claro (a pesar de que ya hay una versión de WDasm adaptada para trabajar con VB).

No obstante, algunos maestros del mundo de la ingeniería inversa se han ido percatando de la posibilidad de enfrentarse con retos de este tipo.

*Para los programas realizados con Visual Basic 3, existen descompiladores de código que permiten recomponer el código fuente y analizarlo.

*Para los programas hechos con VB 5 o 6, se recurre a Smartcheck, otro gran producto de Numega (los creadores de "la bestia") que permite acceder a todo lo que tiene lugar durante la operación de entrada de datos para registrarse.

*Para los programas realizados con VB 4, un personaje llamado RAZZIA encontró entro de la librería vb pertinente la sección encargada de realizar las comparaciones. Nuestro ejemplo versará sobre esta última técnica:

La aplicación se llama REMINDER 1.1 y está disponible en http://members.aol.com/ron2222

Iniciamos el programa, se nos pide que nos registremos. Bien. Ante todo deberíamos tener el Softice previamente bien configurado. Ya que nos moveremos por la librería correspondiente a las aplicaciones hechas con vb4 debemos añadir al winice.dat una nueva línea que nos permita entrar de lleno en dicha dll: EXP=c:\windows\system\vb40032.dll

Reiniciamos nuestro ordenador para que los cambios se hagan efectivos y nos disponemos a registrar el programa. Introducimos los datos:

NAME: COC-FRIENDS
CODE: 445566

Ahora nos interesa una forma de entrar en el programa. Un recurso muy exitoso suele ser el bpx hmemcpy". Ctrl+D > Softice : bpx hmemcpy (+intro) Ctrl+D > Estamos en Windows. Le damos a Aceptar y...Bingo..Estamos en _____kernel_____. Le damos a F12 siete veces (pasamos varias veces por ____user____) hasta aparecer en pleno _____vb40030_____. Ya estamos donde queríamos. Ahora nos interesa localizar la zona en que tiene lugar la comparación. Para ello debemos anotar:

S 0 L FFFFFFFF 56,57,8B,7C,24,10,8B,74,24,0C,8B,4C,24,14,33,C0,F3,66,A7 (+intro)

Teóricamente deberíamos encontrar esta cadena de bytes en hex. en una sola dirección de memoria, pero si aparece en más de un sitio puede ser debido a que también la tengamos en el escritorio (quizás en un documento abierto que la contenga). Lo importante es que vamos a poner un bpx (BREAK POINT ON EXECUTION) en las direcciones en que haya aparecido. En mi caso Softice encuentra la cadena en dos direcciones, pues habrá que bpxear en las mismas:

bpx 013f:003e0748
bpx 013f:0f97b348

(podemos eliminar el previo bpx hmemcpy, si queremos con un bc #numero correspondiente)

Una vez establecidos los nuevos breaks, le damos a F5 y...regresamos a Softice. La segunda dirección ha sido la responsable del break.

Estamos justo en el punto del vb40032.dll en que se establece la comparación entre código chungo y real. Sólo tenemos que mirar qué contienen ESI y EDI. En la ventana de datos nos aparecerán los códigos correspondientes en formato ancho (es decir

445566 aparecerá como 4.4.5.5.6.6). Sólo habrá que quitar estos puntos intermedios para acceder al código real de registro, que resulta ser el mismo para todo el mundo :(, no hay operaciones a partir del nombre ni nada de eso.

Podremos eliminar debidamente nuestros datos del registro de Windows en.
[HKEY_CURRENT_USER\Software\VB and VBA Program Settings\Reminder\Registration]

(por cierto, la utilidad REGISTRY MONITOR (regmon) nos resulta muy útil para saber en qué parte del registro se almacenan estos datos :)

Otra forma de aproximarse a los programas realizados en VB4 consisten en beneficiarse de la función multibytetowidechar. Si nos interesa realizar la prueba, rellenamos las casillas de los datos, creamos un "bpx multibytetowidechar" , regresamos a Windows y le damos a OK. Aparecemos en ___kernel___. Un F12 , otra f5 y otro f12 nos depositan en pleno ___vb40032____. Un simple "d ebp" nos mostrará cómo nuestro código ha pasado a formato ancho "4.4.5.5.6.6". Algo más abajo, en la misma ventana de datos, ya se nos apunta cual será el código con el que será comparado. Mal, muy mal.

El mismo autor dispone de otros productos elaborados con vb4. Siempre podemos entretenernos probando con algún otro programilla, eso sí, evitando siempre el tema de tráfiqueo insulso de códigos (no es nuestro estilo) y decidiéndonos por adquirir una licencia auténtica en caso de que consideremos que el producto vale la pena y queramos hacer uso de él en adelante.

¿Acaso el arte de crackear en privado, humildemente, no resulta tan apasionante como el más adictivo de los juegos de estrategia, acción y aventuras juntos?

"Nulla dies sine cracko" (ningún día sin crackear)

Lección 8 - Cursillo Otoñal de Crackeo (21/12/99)

"REMEDIOS CONTRA LA NAG-OFOBIA"

Todos nos hemos topado alguna vez con personas que tienden a repetir numerosas veces una mismo mensaje (una recomendación, una advertencia, una queja...). Resulta difícil detener esta especie de "loop" verbal sin parecer descorteses, de forma que debemos ingeniarnos alguna manera sutil para que el proceso no continúe indefinidamente. En el mundo de los programas informáticos nos encontramos con una actitud similar a la anteriormente citada: los nags, estos mensajes recordatorios que nos advierten, recuerdan, avisan sobre nuestra condición de "personas pobres" :)

Vamos a dedicar esta lección al tema de la eliminación de dichos mensajes. Nuestras habituales herramientas se pondrán al servicio de la ejecución más rápida de los programas, sin necesidad de visualizar en cada momento la misma ventanilla de siempre. Para las personas que padecen nagofobia, aquí llega un pequeño remedio para sus males.

En principio, la filosofía de nuestro trabajo se basará en la idea de que la mayoría de nags aparecen al inicializarse un programa. En este sentido, nuestro modus operandi se centrará en el hecho de cargar el programa desde el principio, tracear como unos locos y advertir el momento en que aparecerá el nag (algún CALL maligno será el responsable). Cuando localicemos el responsable, nos dispondremos a "actuar"

debidamente. También veremos que el "problema" se puede abordar desde breakpoints a la API dialogboxparama o a messageboxa.

"E J E M P L O # 1 - SOFTICE Y WDASM, LA MEJOR ALIANZA"

Para demostrar una vez más que nuestras intenciones son totalmente puras, sin deseo de perjudicar a nadie, optaremos por practicar con un programa Freeware llamado HEXVIEW (ftp://ftp.funduc.com/ftp/hexview.zip), un visualizador de ficheros en formato hex.

Al inicializar el programa, viene a recibirnos directamente un hermoso nag que nos indica que se trata de un programa freeware, cuyo código fuente incluso está disponible, bla, bla...

Procedamos a darle a nuestro icono **Numega > Symbol Loader > File >Open Module...**
Seleccionamos hexview.exe y procedemos a darle al **Module > Load**...

Nos aparece un mensaje de error, pero le damos a Aceptar y aparecemos en Softice. Estamos cargando el programa desde el principio!! Procedamos con el traceo a base de F10, estando especialmente atentos al momento en que nos aparezca el nag (lo cual nos hará regresar a Windows).

(por el momento, traceamos con F10 porque sería una locura introducirnos en cada uno de los CALLs que aparecen -nada de f8,pues-)

F10,F10,F10,F10,F10,F10,F10,F10,... Pasamos por un pesadísimo loop, no importa, F10,F10,F10,F10,.... de golpe, PLASHNAG!!! Aparecemos en Windows y ya tenemos en pantalla el nag junto al programa en cuestión.

Todo esto ha sucedido al pasar por:
00404B1D E845CA0000 call 00411567. >>> En su interior se han dado las instrucciones para iniciar el programa y mostrar el nag!!

Llegados a este punto, podríamos meter un **bpx 404b1d**, reiniciar el proceso, detenernos aquí y nopear la instrucción, cambiando **E845CA0000** por **9090909090** (no operation), pero no obtendríamos resultados.

Nuestro buen amigo CALL contiene numerosos subCALLS en su interior, así que deberemos entrar en el (con f8) y continuar con f10 hasta dar con el CALL responsable de nuestro nag.

Se supone que hemos reinicializado el proceso, cargamos de nuevo el ejecutable y para ahorrarnos traceo (y destrucción de la tecla f10 :), anotamos **G 404b1d** (+intro) (una especie de breakpoint momentáneo) y llegamos al punto que nos interesa. Le damos a F8 y entramos en el CALL. Ahora nos hallamos en la dirección de memoria **411567**.

Continuemos con F10 hasta ver cosas sospechosas....

F10,F10,F10..:00411577 E8328E0000 call 0041A3AE >>Nag nuevamente!!

¿Será este nuestro call responsable? No. Realizado el oportuno nopeo, vemos que habrá que buscar más adentro!!! Por lo cual, le damos a F8 y nos introducimos en dicho CALL. Ahora estamos en la dirección 41a3ae:

F10,F10,F10,F10,F10,F20(es broma;),F10,F10,....

:0041A3E5 FF5658 call [esi+58] >>Atención.
Parece pasar algo. Se ha cargado el programa, pero no el nag. Podemos verificarlo dándole a F4 para echar un vistazo a Windows. No ha aparecido aún nuestro nag, por lo que vamos a continuar con F10,F10,....

:0041A401 FF565C call [esi+5C] >> Aquí vuelve a aparecernos el Nag. ¿Podemos nopear ya? No! Si lo hacemos vemos que se produce un error, luego debemos entrar en el interior de este nuevo call con un f8 y esperar a ser más precisos....¿Estamos dentro ya? Aparecemos en la dirección 41776c y le damos al F10,f10,f10,....

:00417787 E84C020000 call 004179D8 >> Nuevamente el NAG!! ¿Podemos nopear ya? Tampoco. Si convertimos el call en 9090909090, la cosa no funciona. Parece ser que habrá que ir mucho más pa-dentro. Pues vayamos con f8 y f10,f10,f10,...

Maldición....entramos en una especie de loop raro en el que parece que se están cargando cosas. De golpe, aparece el nag y no hay señales claras de cómo ha emergido. Primeros síntomas de cansancio ¿Ha sido en vano todo nuestro esfuerzo? ¿Nos rendimos?

Hemos utilizado el sistema habitual de "traceo-desde-el-inicio" para localizar el CALL responsable pero aquí parece haber una jerarquía interminable de CALLs. Llegados a este punto, uno sólo puede recurrir al ZenCrack: la magia de la intuición, la reflexión y el análisis de los hechos.

Z E N C R A C K E A N D O ....

Pensemos después de un buen trago de IceTea (no os voy a recomendar Vodka del bueno como hacía +ORC, el maestro:) ¿Acaso en la parte superior de nag no pone "About hexview"? Si le damos al menú About.. accedemos a la misma ventanita molesta. ¿Porqué no interrogamos a nuestro amigo Wdasm acerca de Strings con el término ABOUT?

Desensamblamos el fichero ejecutable y localizamos la cadena About. En las primeras líneas del listado muerto vemos:

Name: DialogID_0064, # of Controls=004, Caption:"About hexview"

¿DialogID_0064? Busquemos esta referencia a lo largo del listado: Esto es lo primero que encontramos:

*** Referenced by a CALL at Address:|:004016CC**

:00401650 56 push esi
:00401651 6A00 push 00000000
:00401653 8BF1 mov esi, ecx

*** Possible Reference to Dialog: DialogID_0064 >>>Nuestro señor About!!!**

Parece ser que Mr. About ha sido llamado por un CALL en 4016CC, pues volvamos a Softice y metamos un bpx 4016cc:

:004016CC E87FFFFFFF call 00401650 >>Aquí estamos. Traceamos con F10 sobre el CALL y nada ha aparecido en pantalla. Bueno. Continuemos con F10 ya que por ahora no ha aparecido el nag...

:004016DD E8F71C0100 call 004133D9 >> Al pasar sobre esta instrucción vuelve a aparecer el nag!!! Procuremos entrar en ella y estar atentos a nuevas llamadas.¿Podemos nopearlo? Quizás sí, pero como estamos hartos de dejar colgado el aparato y tener que resetearlo, seamos pacientes,..el fin está cerca :)

Reiniciemos el proceso de cargar el ejecutable con SymbolLoader, démosle a **G 4016dd** para llegar rápido, entremos en él con f8. Démosle nuevamente a F10 para seguir traceando hasta encontrar el subCALL responsable..f10,f10,f10,....

:004134BA E88E300000 call 0041654D >> Al sobrepasarlo con un F10, el nag aparece! ¿Podemos nopearlo ya? Probemos!!! Cambiemos el E88E300000 por 9090909090 para anular la llamada. Dentro de Softice podemos llevarlo a cabo en memoria (no físicamente, es decir, tal como lo haríamos con un editor hexadecimal) anotando -si estamos en la línea pertinente- A 4134ba (+intro) : y a continuación : nop (+intro),nop (+intro),nop (+intro),nop (+intro) y nop (+intro). Un par de Escapes nos permiten dejar de entrar código ensamblador. Ahora es el momento de darle a F5 para que lea las nuevas instrucciones y continúe con el resto de programa. ¿Resultado? Prueba superada!!! Estamos en el programa y el nag no ha aparecido!!!! Ha sido durillo, pero ha valido la pena. Cuando los métodos tradicionales no nos conducen a ningún sitio, hay que optar por alternativas con imaginación...y a veces la cosa se consigue!!!

(para fijar los cambios de manera permanente habrá que recurrir a un editor hexadecimal, localizar el sitio y anotar los nuevos bytes!!!!)

E J E M P L O # 2 - SALTOS QUE IMPIDEN NAGS

Para practicar con otro ejemplo de nag molesto, vamos a hacer un pequeño experimento con el conocido por todos Winzip. Mr. Nobody utiliza la versión 6.1 porque no es muy propenso a actualizar el software cuando no se notan demasiadas mejoras y lo único que se ganan son más megas desperdiciados en su paupérrimo disco duro. En todo caso, el ejercicio será válido para cualquier versión de Winzip. Seguramente el proceso será similar. Lo único que variarán inevitablemente van a serlas direcciones de memoria.

Para empezar con el ejercicio conviene trabajar con una copia no registrada. Podemos desregistrarla :) eliminando la información de win.ini (en la versión 6.1) y del registro de Windows HKEY/current_user/software/Nico Mak Computing/Winzip/Winini (versiones 6.3 / 7.0).

Una vez desregistrados (¿por qué motivo presupondrá Mr. Nobody que ya estamos registrados?) veremos aparecer un nag inicial al ejecutar el programa. Bien. Nuestro modus abordajeandi en esta ocasión va a ser la de poner un Breakpoint con Softice en la API dialogboxparama, que suele regir un buen número de cuadros de diálogo del tipo que Winzip ostenta:

Softice > bpx dialogboxparama > Windows

Iniciamos Winzip y.....!Slash!!! De nuevo en Softice. No estamos aún en el código de Winzip, así que le damos a F12 para que nos devuelva al sitio desde donde se ha llamado esta función.

Bien! Aterrizamos en * Reference To: USER32.DialogBoxParamA, Ord:008Ah

```
:00429C53 FF1548B94C00 Call dword ptr [004CB948]
:00429C59 8945F8 mov dword ptr [ebp-08], eax
```

ahora traceamos hasta el siguiente RET (f10,f10,..)

```
:00429C98 C3 ret
```

Aterrizamos en la siguiente instrucción del call responsable. No parece haber ninguna instrucción interesante previa, así que seguimos hasta el siguiente RET:

```
:004035C8 C9 leave
:004035C9 C3 ret
```

Bueno, no hay nada interesante previo a la nueva zona en la que nos encontramos, así que seguimos hasta el siguiente RET:

```
:00403517 C9 leave
:00403518 C3 ret
```

Atención!! Porque después de 403518, llegamos a una zona en la que sí hay saltos condicionales previos, así que habrá que examinarlos con detalle para ver si podemos evitar el nag de forma limpia:

```
:004033D3 833D4C894C0000 cmp dword ptr [004C894C], 00000000
:004033DA 0F8419000000 je 004033F9 >>>>>>>>****eureka!******
:004033E0 C70504844C0001000000 mov dword ptr [004C8404], 00000001
:004033EA A104844C00 mov eax, dword ptr [004C8404]
:004033EF A300844C00 mov dword ptr [004C8400], eax
:004033F4 E930000000 jmp 00403429
:004033F9 E88B730000 call 0040A789 <<< call responsable de todo
:004033FE 85C0 test eax, eax<<<aparecemos aquí
:00403400 0F840F000000 je 00403415
```

Después de haber traceado por 3 RETS por fin hemos hallado el CALLmadre desde el que se ideó el dialogboxparama (en realidad todo sucedía en el interior de algún subCALL).

¿Qué tenemos aquí? Varios saltos condicionales. No estaría mal intentar analizarlos para ver cómo podemos evitar el call maligno. Fijémonos en que algo más arriba hay un je que salta directamente hacia el CALLmaligno. Si este salto no se produjera, llegaríamos hasta un jmp que esquivaría todo el meollo, ya que se dirige hacia una zona muy posterior. ¿Qué hacemos? Está claro. No queremos que el salto je tenga lugar, con lo que habrá que evitar el salto. Hay muchas opciones posibles.

Optaremos por cambiar los bytes 0F8419000000 por 909090909090 y nos aseguramos de que el salto no se produce, así pues, llegamos a la instrucción siguiente y hasta el JMP que nos lleva lejos de todo peligro.

Si fijamos los cambios con un editor hex., no hay más nags molestos en nuestro Winzip no-registrado :)

E J E M P L O # 3 - REAJUSTANDO PILAS

Aziconed v.1.8 (16 bits).antiguo editor de iconos. Nuestro siguiente ejemplo será llevado a término con un paciente de 16 bits. Vamos a utilizar un bpx dialogboxparam (fijaos que no aparece la A final, típica de 32 bits) que detendrá el programa en cuanto se pretenda mostrarnos un mensaje al arrancar el programa. Softice rompe, con un f12 y OK en la ventanita aterrizamos en pleno código del ejecutable:

```
:0001.1D55 9AFFFF0000 call USER.DIALOGBOX >>>>>
:0001.1D5A FF76FE push word ptr [bp-02]
:0001.1D5D FF76FC push word ptr [bp-04]
:0001.1D60 9AFFFF0000 call KERNEL.FREEPROCINSTANCE
:0001.1D65 8BE5 mov sp, bp
:0001.1D67 5D pop bp
:0001.1D68 C3 ret
```

Traceamos con F10 hasta el ret, que nos devolverá al CALLmadre. Allí observaremos:

```
:0001.04E8 3D0100 cmp ax, 0001
:0001.04EB 7416 je 0503 >>>> *******eureka!!*******
:0001.04ED E83218 call 1D22 >>> call madre
:0001.04F0 E8E516 call 1BD8 >>>aparecemos aquí
```

Lo tenemos fácil. Si forzamos la realización del salto je iremos a parar a un sitio muy lejano sin pasar por el callmadre responsable del nag. Cambiaremos pues 7416 por EB16 (jmp...) y el nag ya no tendrá lugar!!!!

A pesar de ello, en el menú About, aparecerá un feísimo Unregistered User que no tiene arreglo. No aparece la cadena por ningún sitio (por si pensábamos cambiarla con un editor hexadecimal a mano) y de hecho, el mismo autor en el apartado de ayuda informa de que al registrarse se recibirá una copia registrada (que no es la misma que la nuestra), con lo que se deduce que hay poco por hacer. No obstante, vamos a hacer un lindo ejercicio consistente en eliminar la ventanilla del About....

Ponemos un bpx dialogboxparam (sin la A final!!) -si no lo habíamos borrado, lo rehabilitamos con be* - y le damos a About... !!Bimm!! Estamos en Softice. F12 y OK y aterrizamos en el código...

```
:0001.0EF0 9AFFFF0000 call USER.DIALOGBOX >>>>*
:0001.0EF5 FF76FE push word ptr [bp-02]
:0001.0EF8 FF76FC push word ptr [bp-04]
:0001.0EFB 9AFFFF0000 call KERNEL.FREEPROCINSTANCE
:0001.0F00 EB00 jmp 0F02
:0001.0F02 5F pop di
:0001.0F03 5E pop si
:0001.0F04 8BE5 mov sp, bp
:0001.0F06 5D pop bp
:0001.0F07 C3 ret
```

Traceamos con F10 hasta el ret y ello nos lleva al call madre:

```
:0001.0BAC 56 push si
:0001.0BAD E89B01 call 0D4B >> call madre
:0001.0BB0 EB09 jmp 0BBB >> aparecemos aquí
```

Llegados a este punto, sabemos que el call 0d48 es el responsable de la aparición de la ventanita de About... ¿Podemos nopearlo directamente? Si lo hacemos podemos encontrarnos con un ligero problema: al eliminar el call, al ordenador no le acaban de salir las cuentas y la cosa puede colgarse. ¿porqué? Respuesta: reajuste de pila!! (registro ESP(win32) / SP (win16)).

Si nos colocamos en la instrucción anterior al CALL, observaremos en Softice, en la ventana de registros que el valor de ESP (en realidad SP) 5c10. Al salir del call (después de un F10): el valor de ESP es 5c12. Si nosotros nos cargamos el call habrá problemas con los datos de la pila, ya que se esperan unos valores determinados. ¿Qué hacemos? Nos cargaremos el CALL, pero reajustaremos los valores de la pila. Está claro que ESP ha salido con un incremento de 2, ¿no? ANTES 5C10 / DESPUÉS 5C12. La instrucción pertinente sería ADD SP,02 y nos va de perlas puesto que sus bytes son : 83 c4 02. Nos va perfecto para nuestros planes ya que los bytes correspondientes al CALL también son tres: e89b01. Sustituyámoslos por los correspondientes al ajuste de pila (83c402) y cuando le demos al menú About... ya no aparecerá nada que nos resulte molesto.

E J E M P L O # 4 - NOPEANDO CALLS IMPOSIBLES

En este caso escogemos como conejillo de indias a un fichero ejecutable que no es más que un magnífico tutorial de crackeo editado por tKC, uno de los grandes de la escena. En su tutorial#5 añadió un precioso nag con algunos segundos de retraso que anteceden a una utilísima colección de varios tutoriales. El mismo autor invitó a los lectores a crackear la ventana, así que vamos allá.

Cargamos el ejecutable con Symbol Loader, siguiendo el proceso ya explicado anteriormente, vamos dándole a F10 hasta indentificar algún call que provoque la emisión de la ventana.

Después de unas cuantas pulsaciones F10, al tracear sobre un CALL acaba apareciendo la ventanilla inicial.

¿Probemos a nopearlo? ¿Porqué no? Reiniciamos el proceso (si queremos descansar un poco, anotemos en la línea de comandos, por ejemplo T 40 (+intro) y él solito traceará 40 instrucciones por nosotros :) Llegamos al call responsable y sustituimos los bytes responsables por tantos 90 como haga falta:

013f:0042db0c: e8 b3 f3 ff ff call xxxxxxx

Dentro de Softice, cambiamos e8b3feffff por 90909090, le damos a F5 y !!Bien!! Entramos en el tutorial sin ventanita de retraso. Procedamos a realizar los cambios con un editor hexadecimal... ¿qué? No encontramos ni uno de los bytes ¿Qué pasa?

El ejecutable resulta que está comprimido. Lógicamente al ser inicializado, se descomprime y nosotros podemos acceder plenamente a sus instrucciones con Softice, pero cuando queremos realizar cambios físicos, no encontramos aquellos bytes con los que nos hemos peleado.

Ante esta situación, un cracker hiperexperimentado localizaría el momento en que el programa se ha descomprimido, forzaría un salto hacia una zona libre del ejecutable, allí establecería los cambios y forzaría un salto de retorno a la inicialización del mismo. Toda una operación de microcirugía muy lejana de las posibilidades de Mr. Nobody. No

obstante, existe una utilidad que permite realizar cambios en memoria a un ejecutable antes de que éste haya arrancado por completo. Risc's Process Patcher (http://personal2.redestb.es/karpoff/herramie.htm) permite crear

Loaders que cargan el programa y le meten en memoria los cambios que hayamos decidido. Lógicamente se trata de un parcheo en memoria, no "físico" y para arrancar tal aplicación habrá que ejecutar el Loader, no la aplicación en sí. En un sencillo fichero de extensión ppc podremos anotar a mano los cambios que queramos que se realicen. Para este caso, el fichero ppc de Mr. Nobody tiene la siguiente pinta:

```
#Process Patcher Configuration File v1.2
// Friendly Name tutorial de tkC5
// Program Filename
tutor5.exe
 // Number of Bytes to be patched
5
// Memory Addresses for patch
0x42db0c:0xe8:0x90
0x42db0d:0xb3:0x90
0x42db0e:0xf3:0x90
0x42db0f:0xff:0x90
0x42db10:0xff:0x90
#End of Configuration File
```

fácil, ¿verdad?

(no olvidemos que esta utilidad nos permitirá también alterar ficheros que tienen chequeo CRC (no permiten cambios "físicos" en el ejecutable), así como ejecutables comprimidos (en caso de no saber cómo descomprimirlos)

ps: Por cierto, para leer los tutoriales, si no nos aparece nada para clickear, hay que desplazar la barra lateral hacia arriba. Ello hará aparecer los diferentes menús clickeables en la parte superior. Un pequeño error de programación, supongo.

E J E M P L O # 5 - ZEN A CIEGAS (HEXVIEW, nuevamente)

Regresamos al tema de la eliminación del nag de Hexview porque Mr. Nobody se complacería en comunicar una de las pocas cosas que ha podido averiguar solito. ¿Fue la fortuna? ¿La casualidad? ¿Porque no?

Estos elementos suelen ser factores habituales en nuestras vidas. La explicación que viene a continuación no es más que el fruto de la observación y el paso de horas insípidas "jugando" con editores hexadecimales. En todo caso, puede considerarse la única contribución de cosecha propia por parte de Mr. Nobody al mundo del crackeo: Habiendo visto que el nag inicial del programa viene bajo el título de About hexview, procedemos a buscar dicho mensaje dentro del ejecutable utilizando un editor hex.. No aparece en una primera búsqueda sobre ASCII y es que las Strings están en formato ancho, propio de 32 bytes. Así que no buscamos About hexview, sino A b o u t h e x v i e w.

Este será nuestro proceder: Aquí tenemos una referencia para nuestra búsqueda, los valor hex. De la cadena:

41 62 6f 75 74 (no hace falta nada más)

A b o u t hexview
Sabiendo que A b o u t en formato ancho hex. será 41 00 62 00 6f 00 75 00 74, podemos proceder a la búsqueda de esta cadena hex. (somos vagos y no nos vamos a molestar en anotar todos los bytes faltantes): Nuestra búsqueda tiene éxito. Nos encontramos varias cadenas con About en formato ancho, pero no nos ponemos contentos hasta llegar a la que coincide en todo lo que nos interesa (About hexview):

87 00 75 00 51 E1 00 00 C0 00 80 00 00 00 00 *^`````````++
04 00 00 00 00 00 29 01 5E 00 00 00 00 41 00 ...).....A
62 00 6F 00 75 00 74 00 20 00 68 00 65 00.. b.o.u.t .h.e.x

Esta es más o menos la pinta de nuestro editor hexadecimal. Podemos ver a la derecha los valores ASCII y a la izquierda sus equivalentes hex.. Sin duda estamos ante nuestra nag. Si cambiamos el mensaje por E b a u t h e x v i e w , lo guardamos y ejecutamos el programa, observaremos que el nag inicial aparece bajo el título Ebaut hexview.

Bien. Mr. Nobody ha pasado alguno de sus mejores ratos trasteando con los bytes hex. y en su momento descubrió que hay cosas interesantes en los hex. que preceden al A b o u t ..... ¿Qué serán esos C0, 80, 04,...? Ni idea. Con aquello de "ensayo y error", uno se da cuenta de que cambiando alguno de estos bytes por 00, sucede que al intentar arrancar el programa, se produce un error. A veces cambiamos otro byte por 00 y resulta que no pasa nada, ...fale!! Lo mejorcito de este meollo es que la casualidad quiso que Mr. Nobody se diera cuenta de que el señor C0, cambiado por 00 da como resultado que ya no aparezca el nag inicial!!! Bingo!!! Pura suerte!! Bienvenida sea!!

¿Nos sirve de algo este detalle? Bueno, como mínimo nos sirve para librarnos del nag de cualquier producto de www.funduc.com. Además también se puede aplicar a otras aplicaciones. Buscamos la cadena y convertimos a 00 cualquier byte previo sospechoso. A Mr. Nobody le ha servido para una docena de aplicaciones aunque hay que ser consciente de que se trata de crackeo absolutamente a ciegas. Se supone que tiene que ver con el lenguaje con que ha sido programada la aplicación. A veces no cambiamos C0 por 00, sino un 01 sospechoso por 00. El caso es que si funciona, pues mejor. No se trata de crackeo científico, sino crackeo casual-fruto-de-la-curiosidad

Por ahora eso ha sido todo. buuuuuffffff!!!!!! :)

Lección 9 – Cursillo Otoñal de Cragueo (29/12/99)

Procedamos a introducirnos en un nueva y densa lección que nos aporte alguna información complementaria acerca del comportamiento habitual de los programas con los que solemos enfrentarnos. Se espera que el conjunto de técnicas y puntos de ataque comentados aquí sean de igual utilidad para los crackers principiantes (entre los que se cuenta Mr. Nobody), así como para los programadores que se decidan a leer estas líneas para mejorar y perfeccionar la protección de sus productos.

A lo largo de la lección nos toparemos con temáticas distintas. Nuestro análisis empezará con una revisión del tema de los límites temporales y los errores más habitualmente cometidos en su codificación; continuaremos con una pequeña reflexión sobre la forma en que un programa "se entera" de las cosas: ¿cuando debo saltar o no? ¿estoy registrado o no? ¿Este código es correcto o no?..y terminaremos con un breve vistazo a un simple método de encriptación basado en la operación XOR. Vamos allá....

L Í M I T A C I O N E S V A R I A D A S

LIMITES TEMPORALES

Hay diversas formas de abordar la cuestión de la evaluación limitada en el tiempo de un producto. Algunos crackers experimentados recurren a la interrupción del programa (breakpoint) en cuanto éste hace uso de APIs como Getlocaltime, Getsystemtime,.. Su nombre es suficientemente transparente como para deducir que pueden tener mucho que ver con el cálculo de los días transcurridos desde la instalación de un producto, etc.. A pesar de ello, hay que decir que no todas las llamadas que se producen a estas APIs son debidas al deseo de comprobar el límite de tiempo, así que podemos encontrarnos con múltiples llamadas a Getlocaltime o Getsystemtime al iniciarse un programa. ¿Cuál de ellas será la que nos interesa desviar? Bfffff.... Uno puede llegar a perder la paciencia cuando hay tantas líneas que no entendemos,...pero todo tiene remedio.

Probablemente el error más grande cometido por los programadores consiste en informarnos de que "la evaluación ha expirado". Nuestra pregunta entonces debe ser: ¿De dónde ha salido este mensaje?

En este sentido puede sernos de gran ayuda el listado muerto de WDasm. Localizar el mensaje de error y ver qué ha provocado el salto hacia aquella zona son los primeros pasos a seguir. Esta especie de back-tracing o búsqueda hacia atrás nos puede llevar hasta la zona a parchear (o hasta la dirección concreta de la API que nos interesaba). Vayamos a por los ejemplos:

S E A R C H & R E P L A C E 2.88 (www.funduc.com - ya van por la 3.0!) (ftp://ftp.funduc.com/ftp/)

Cuando arrancamos el programa después de 30 días nos encontramos con un mensaje diciendo "The evaluation period expired". Craso error!!!y aún más cuando el código se puede desensamblar libremente con WDasm. Sólo nos queda ir a las StringDataReferences, doble-clickear y aparecemos en....

```
:0041E78E E8DD270000          call 00420F70
:0041E793 85C0                test eax, eax
:0041E795 742A                je 0041E7C1
>>>>>>>>********sí*********
:0041E797 6AFF                push FFFFFFFF
:0041E799 6A04                push 00000004

* Possible Reference to String Resource ID=02170: "The evaluation period
expired.you like to see Orderin"

:0041E79B 687A080000          push 0000087A >>>>>>aquí!
```

Fijémonos en las líneas anteriores al mensaje de "expiración". Hay un salto que, de haberse producido, nos habría ahorrado la molestia de la ventanita con el mensaje puesto que habríamos ido a parar muy lejos (41e7c1). ¿Cómo podríamos asegurarnos de que el salto se produjera? Sencillo: cambiar 742A por EB2A (jmp forzoso :). Efectivamente este cambio sería suficiente para asegurarnos una victoria.

Otro cambio posible radicaría en la sutileza de alterar aquello que causa o no el salto. Veamos, ¿porque no se produce el salto je (salta si es igual, salta si es cero)? Está claro: previamente hay el testeo de eax. Si eax = 0 se salta, si eax es diferente de 0 no se salta

y se llega a "expired". ¿Quién se encarga de gestionar este valor de EAX? El call que aparece más arriba. Podemos intentar entrar el call y forzar su salida con un valor EAX=0. Nos colocamos sobre la línea del CALL, le damos a ExecuteCALL (menú Execute text de WDasm) y llegamos a...420f70. Una vez allí podemos forzar la siguiente instrucción:

```
420f70  33 C0 :  xor eax,eax   (convertir eax en 0)
420f72  C3    :  ret (volver al sitio desde donde se llamó el call)
```

(el resto de instrucciones en el interior del call no se llegarán a ejecutar nunca, debido al ret que hemos impuesto, que fuerza el retorno inmediato). Con estas dos simples instrucciones nos aseguramos que el call devuelva un eax = 0 . Con ello, no hará falta modificar el salto condicional (que sí tendrá lugar)

Reflexión: ¿Siempre se puede forzar de esta forma el valor de determinado registro al regresar de un CALL? No siempre. En este caso ha sido posible porque previamente habríamos comprobado con Softice que no había diferencias en el registro ESP, la pila. Si al regresar del call hubiera algún cambio, deberíamos asegurarnos de ajustar debidamente estos valores porque ello podría generar un error ya que al programa no le saldrían las cuentas. En este sentido, hemos tenido la suerte de toparnos con un CALL muy agradable puesto que no ha hecho falta hacerle más ajustes que el de forzar el valor de EAX.

Es de vital importancia entender el sentido de una secuencia como:

```
CALL XXXXX
TEST EAX,EAX
JE XXXXX (SI EAX ES 0, SALTA)

CALL XXXXX
TEST EAX,EAX
JNE XXXX (SI EAX NO ES 0,SALTA)
```

En el interior del CALL (que puede corresponder por ejemplo a una rutina que compara un código chungo y uno auténtico) se acaba emitiendo un veredicto: EAX 1 o EAX 0, que será decisivo para la ejecución (o no) del salto siguiente. Conocer este hecho puede ahorrarnos mucha faena. Algunos ejemplos:

***Getright:** Hubo quien intentó parchear alguna versión de este programa. Pues bien, para llevarla a cabo, era imprescindible al menos invertir unos 10 o 12 saltos condicionales. Todos ellos estaban precedidos por un mismo CALL y estaban localizados en sitios diferentes. Lógicamente, resulta mucho más práctico enterarse de qué valor nos interesa al salir del CALL (eax =1 o eax =0) y forzar alguno de esos valores. De esta forma, con un único cambio en el interior del CALL nos aseguramos de que los saltos que se van a producir en diferentes partes del código, serán buenos para nosotros. A eso se le llama patchear elegantemente. Cada vez que desde algún lugar se llame esta rutina, el resultado será el deseable!! La diferencia está en parchear una vez la causa (call,..) o doce veces la consecuencia (je/jne)

***Winzip:** Los interesados en parchearlo se darán cuenta de que la clave para que acepte cualquier código chungo radica en el valor que tiene eax al salir de determinado CALL. Eax=1 querrá decir "vale, registrado" / Eax=0 significará "vayamos a incorrect-code". De alguna forma u otra, deberemos asegurarnos de que EAX sea 1 al salir del call (por cierto, a veces el programa puede esperar un 0 y no un 1. Sólo hace falta estar atento

hacia donde nos hará saltar un valor u otro. Dentro de Softice podremos cambiar el valor de EAX anotando cuando nos interese R EAX=1 (+intro) o R EAX=0 (+intro). Sólo hace falta experimentar un poco.

***Acdsee 2.40 / 2.41:** Después de entrar los datos y tracear con f8/F10, uno acaba viendo que al salir de un CALL con EAX=0 nos enviarán al mensaje del perdedor. Si entramos en el CALL en cuestión, vemos que hay mucha prisa por enviarnos hacia una instrucción XOR EAX,EAX que convierte EAX a 0 y seguidamente un RET que nos hace salir de la rutina. Únicamente nopeando esta instrucción, salimos del CALL con EAX diferente de 0 y el programa admite in aeternum el código y los datos que hayamos introducido (además este resulta ser el mismo call que verifica los
datos del registro de Windows al iniciar la aplicación, o sea, que el resultado es óptimo para nosotros).

ps: Este truquillo no es ya válido para la versión Acdsee 2.42

U L T R A E D I T 3 2 V. 7.00a

Otro ejemplo de programa de evaluación limitada temporalmente. En este caso disponemos de 45 días para decidirnos. Cuando se supera este período de tiempo, una ventanilla nos avisa del hecho ("UltraEdit 45 Day Evaluation time expired!") y el programa se cierra. Puesto que el programador se toma la molestia de avisarnos, pongamos un bpx messageboxa bajo Softice para que se nos informe de la zona en que nos movemos. Al re-iniciar Ultraedit, justo después de ignorar (con la tecla Escape) el cuadro de diálogo que nos sugiere que entremos nombre y código, Softice rompe. Un F12 nos deja en pleno código y nos enteramos de la dirección por la que nos movemos:

```
* Reference To: USER32.MessageBoxA, Ord:01BEh

:00469242 FF1514B74B00          Call dword ptr [004BB714]
:00469248 399E44010000          cmp dword ptr [esi+00000144], ebx
>>>>Aquí (Mess.box)
```

Es el momento de desensamblar el ejecutable con WDasm y buscar qué es lo que nos ha hecho aterrizar hasta aquí. Unas líneas más arriba podremos apreciar una instrucción altamente sospechosa:

```
* Possible StringData Ref from Data Obj ->"Days to expire"

:004691D4 68C01A4E00            push 004E1AC0

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

:004691D9 68141A4E00            push 004E1A14
:004691DE E8B50F0300            call 0049A198
:004691E3 A16C844E00            mov eax, dword ptr [004E846C]
:004691E8 2B0574844E00          sub eax, dword ptr [004E8474]
:004691EE 50                  push eax
:004691EF E896440100            call 0047D68A
:004691F4 83F82D              cmp eax, 0000002D >>>compara EAX con
2Dhexa(45 decim!!!)
:004691F7 59                  pop ecx
:004691F8 7F0C                jg 00469206 >>>>(jump if great: salta
a "expired" si es mayor)
:004691FA 399E44010000          cmp dword ptr [esi+00000144], ebx
```

```
:00469200 0F8576FFFFFF          jne 0046917C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004691F8(C)

bla,bla

* Possible Reference to String Resource ID=00068: "UltraEdit 45 Day
Evaluation time expired!!!!"

bla,bla

* Possible Reference to String Resource ID=00069: "To continue to use
UltraEdit you must send the registration "

bla,bla

* Reference To: USER32.MessageBoxA, Ord:01BEh (Mensaje de Expiración!!)

:00469242 FF1514B74B00          Call dword ptr [004BB714]
:00469248 399E44010000          cmp dword ptr [esi+00000144], ebx
>>>>Aquí (Mess.box)

Bueno, bueno, bueno...veamos más de cerca el meollo:
:004691F4 83F82D                cmp eax, 0000002D
.................               ............
:004691F8 7F0C                  jg 00469206
```

¿No es realmente muy sospechoso? Comparar EAX con 45 (¿días?). Si es mayor, saltar y mostrar mensaje de Expiración. El método para hacer la comprobación temporal ha sido demasiado visible. Aquí lo que nos interesa es no-saltar hacia Expired, con lo que sólo tendremos que nopear el salto : 7F0C cambiado por 9090(nop, nop). A partir de este momento dispondremos del tiempo necesario para evaluar sin presiones cronológicas este fenomenal programa y decidirnos por su compra.

Hubo un tiempo en que los programas limitados a 30 días recurrían mayoritariamente a unas secuencias muy fácilmente identificables:

```
cmp eax, 0000001E  (compara eax con 1Ehexa(30 decimal: 30 días)
jng Continuar (Si no es mayor, salta a Continuar)
```

(Esta secuencia se podía "arreglar" forzando un JMP (saltar siempre) en lugar del JNG (saltar si no es mayor de 30 días)

```
cmp eax,0000001E (compara eax con 30 "días")
jg Expired (Si es mayor, salta a Expired)
```

(En este caso, el "arreglo" consiste en evitar el salto a Expired, con un nopeo (tantos 90 como hagan falta) de la instrucción JG xxx

La misma estructura puede aparecer con otro salto de similares características:

```
cmp eax,1E
jle Continuar (saltar si es menor o igual) Solución: forzar un JMP
```

Toda esta reflexión nos conduce inevitablemente a la necesidad de dar un listado general de "saltos" en ensamblador. Lógicamente, cuando un programa analiza si han pasado determinados días, si se ha ejecutado el programa determinadas veces, etc..ya no entran en juego los típicos JE / JNE, sino que entramos a jugar con saltos del tipo: salta-si-es-mayor , salta-si-está-por-encima-de ,... El siguiente listado está extraído de un tutorial de ManiacPc (KUt99):

Hexadecimal	Assembler	Significa
75 o 0F85	jne	Salta si no es equivalente
74 o 0F84	je	Salta si es equivalente
EB	jmp	Salta directamente a
90	nop	(No OPeration) Sin operación
77 o 0F87	ja	Salta si esta sobre
OF86	jna	Salta si no esta sobre
0F83	jae	Salta si esta sobre o igual
0F82	jnae	Salta si no esta sobre o igual
0F82	jb	Salta si es inferior
0F83	jnb	Salta si no es inferior
0F86	jbe	Salta si esta debajo o igual
0F87	jnbe	Salta si no esta debajo o igual
0F8F	jg	Salta si es mayor
0F8E	jng	Salta si no es mayor
0F8D	jge	Salta si es mayor o igual
0F8C	jnge	Salta si no es mayor o igual
0F8C	jl	Salta si es menor
0F8D	jnl	Salta si no es menor
0F8E	jle	Salta si es menor o igual
0F8F	jnle	Salta si no es menor o igual

LIMITACIONES OPERATIVAS

Dentro del mundillo shareware también es frecuente toparse con utilidades que están limitadas en algunas de sus funciones. En algunos casos este "recorte" es tan exagerado que hace imposible una evaluación mínima. Para solventar este tipo de problemas, siempre estaremos de suerte cuando se nos avise de forma explícita de la no-posibilidad de realizar determinada función. Una vez más la exposición de demasiada información por parte del programador acaba siendo un recurso que se
gira en su contra:

W I N Z I P K E Y 2.1.0 (www.lostpassword.com)

Este programa nos permite recuperar el password de un fichero zip, siempre que este no supere los 3 caracteres. Cuando estamos intentando conseguir un password que sea superior, acaba saliendo en pantalla un mensaje que informa de las limitaciones de esta versión demo se han sobrepasado y no se admiten passwords superiores a 3 caracteres. Un bpx messageboxa nos permitirá detener el programa en este momento. Después de algunos F12 nos percataremos de que el responsable de ello es:
4037e9: call 0041445a (mostrar mensaje)

Una vez situados en esta zona, sólo nos queda hacer un traceo hacia atrás para ver si hay alguna manera de evitar el meollo. Algo más arriba aparece algo interesante:

```
40374d: cmp ebx,02
403750: jle 00403804 (salta-si-es-menor-o-igual hacia 403804,lejos del  call-maligno)
```

Cuando se ha superado el límite programado, no se salta y se ejecutan las instrucciones siguientes hasta llegar al call-maldito. ¿Cual es la solución? Lógicamente queremos que el salto se produzca siempre, con lo que cambiaremos el señor JLE por un hermoso JMP. ps: El único problema que tenemos es que el fichero está comprimido. Podemos descomprimirlo con Procdump o bien utilizar un parcheador en memoria, como R!sc's Process Patcher y rellenar debidamente el fichero de configuración.

W I N R A R 2.50 BETA2

Este conocidísimo compresor contiene algunas limitaciones en algunas de sus funciones. Cuando le damos a Add files to archive..., nos aparece un cuadro de diálogo en el que aparecen varias opciones. Algunas de ellas (Erase destination disk.../ Put authenticity verification /..) no pueden ser pulsadas. Al hacerlo nos aparece un mensaje en el que leemos: **Available only in registered version.** Grave error!!

Desensamblemos el ejecutable y guardemos el listado muerto. Pongamos un bpx messageboxa bajo Softice y clickeemos sobre una de estas opciones imposibles (por ejemplo: **Put authenticity verification**). Aterrizamos en Softice y con un F12 llegaremos a....

```
* Possible Reference to String Resource ID=00106: "Available in
registered version only"

:0040A932 6A6A                push 0000006A
:0040A934 E80BBCFFFF          call 00406544
:0040A939 59                  pop ecx
:0040A93A 50                  push eax
:0040A93B 53                  push ebx

* Reference To: USER32.MessageBoxA, Ord:0000h

:0040A93C E852080400          Call 0044B193 >>>>Aquí!!!!!
```

Si nos fijamos, algo más arriba (ctr + flechaarriba) aparece un salto condicional, que de haberse producido, nos habría evitado el molesto mensaje.

```
:0040A91E 833D7CF0440000      cmp dword ptr [0044F07C], 00000000
:0040A925 7524                jne 0040A94B
>>>>>*salta-lejos-del-call*
:0040A927 6A30                push 00000030
```

Podemos remediar el problema con un simple JMP forzoso, aunque de esta forma habrá que parchear también cada uno de los saltos correspondientes a las otras opciones limitadas. Habrá que buscar la messageboxa correspondiente a "Erase destination disk" y hacer lo mismo y operar similarmente con el resto de limitaciones que el programa tenga. Pero... ¿Hay una manera más elegante de hacerlo? Sí!! Analizando "la causa". Curiosamente todas estas limitaciones tienen una misma comprobación previa:

```
cmp dword ptr [0044F07C], 00000000 >>Compara lo que haya en 44f07c con 0
jne 0040A94B >>Si no son iguales, salta a todo-correcto
```

Esta claro que el programa ha encontrado un 0 en [0044F07C], lo cual hace que el salto no tenga lugar y lleguemos al mensaje de "Available only...". ¿Quién pone el valor en [0044F07C]? Ahora lo veremos. Está claro que el valor que se encuentre en [0044F07C] es el flag que determina si hay que mostrar el mensaje o dejar que se utilice tal opción.

Pensemos: mmmmmmm....En algún momento el programa debe decir qué hay que meter en [0044F07C]. Por lógica el programa debe iniciarse con un valoren 0 en [0044F07C] y luego, después de alguna comprobación (ver si está registrado), pondrá de alguna manera un 1 (o algo diferente a 0) en [0044F07C] que permitirá al programa operar sin límites.

Resumiendo :
[0044F07C] con un 0 (no registrados, limitaciones a go-go)
[0044F07C] con un 1 (registrados, no limitaciones)

Ya que tenemos el listado muerto de WDasm, vamos a utilizar el programa Search & Replace con finalidades crackeantes. Busquemos en el listado, cualquier referencia a "[0044F07C]". Search&Replace nos dará un listado de cualquier línea que encuentre con este dato. Aquí van los resultados:

```
********
Processing file : C:\WINDOWS\Escritorio\WinRAR.alf
Line 6401 - :00403FAD 833D7CF0440000        cmp dword ptr  <[0044F07C]>, 00000000
Line 7141 - :0040435C 833D7CF0440000        cmp dword ptr  <[0044F07C]>, 00000000
Line 13903 - :00407028 833D7CF0440000       cmp dword ptr  <[0044F07C]>, 00000000
Line 18707 - :00408EFB A37CF04400           mov dword ptr   <[0044F07C]>, eax
**********************sí*********
Line 22623 - :0040A91E 833D7CF0440000       cmp dword ptr  <[0044F07C]>, 00000000
Line 22683 - :0040A952 833D7CF0440000       cmp dword ptr

  <[0044F07C]>, 00000000
Line 41554 - :004134F1 833D7CF0440000       cmp dword ptr  <[0044F07C]>, 00000000
Line 64430 - :0041D5CF 833D7CF0440000       cmp dword ptr <[0044F07C]>, 00000000
Line 64769 - :0041D931 833D7CF0440000       cmp dword ptr  <[0044F07C]>, 00000000
Line 68125 - :0041F80E 833D7CF0440000       cmp dword ptr  <[0044F07C]>, 00000000
Line 73158 - :00422305 833D7CF0440000       cmp dword ptr  <[0044F07C]>, 00000000
Line 73562 - :004226E1 833D7CF0440000       cmp dword ptr  <[0044F07C]>, 00000000
Line 77458 - :00424746 833D7CF0440000       cmp dword ptr  <[0044F07C]>, 00000000
Line 77616 - :00424886 833D7CF0440000       cmp dword ptr  <[0044F07C]>, 00000000
Line 78962 - :004254A4 A37CF04400           mov dword ptr   <[0044F07C]>, eax
**********************sí**********************
Line 79002 - :00425509 A17CF04400           mov eax, dword ptr  <[0044F07C]>
Line 81116 - :0042675F 833D7CF0440000       cmp dword ptr  <[0044F07C]>, 00000000
Line 81847 - :00426DA8 833D7CF0440000       cmp dword ptr  <[0044F07C]>, 00000000
Found 18 occurrences.
Searched 1 file(s), found 18 occurrences in 1 file(s)
************************
```

Mierda! En anteriores versiones de Winrar era fácil encontrar un mov dword ptr <[0044F07C]>, 00 altamente sospechoso. Sólo con cambiarlo por mov dword ptr <[0044F07C]>, 01 teníamos resulto el tema de todas las opciones no disponible. Habrá pues que buscar otro truco...

Hay muchas comparaciones con 0 y un par de "mov dword ptr <[0044F07C]>, eax". ¿No sería bonito que el valor de EAX en esta instrucción fuese 1? (aunque seguro que debe ser 0 = limitaciones).

SOLUCIÓN: Podemos forzar la maquina para que en este par de "mov dword ptr <[0044F07C]>, eax" el valor de EAX sea el correcto 1. ¿Por qué no lo cambiamos por un mov dword ptr <[0044F07C]>, 1 ?

RESPUESTA: Nada!! porque el programa no parece pasar por la zona en la que hemos realizado los cambios. Una vez más busquemos alternativas...

Si la dirección [0044F07C] es especialmente importante, podemos optar por localizar la primera vez que entra en juego al arrancar el programa y forzarle un valor 1. Para ello, cargamos el programa con Symbol Loader y una vez dentro anotamos bpm 0044F07C. Un breakpoint en esta zona de memoria. Un cuanto se acceda a ella, Softice detendrá el proceso. Le damos a F5 e inmediatamente volvemos a....

```
:004134F1 833D7CF0440000        cmp dword ptr [0044F07C], 00000000
:004134F8 752F                  jne 00413529 <<<<<Aquí!!!!!!!!

* Possible Reference to String Resource ID=00873: "evaluation copy"
|
:004134FA 6869030000            push 00000369
:004134FF E84030FFFF            call 00406544
:00413504 59                    pop ecx
:00413505 50                    push eax
```

Valla vaya!! La primera comparación con el valor de [0044F07C] tiene lugar para saber si hay que meter "evaluation copy" o no!! No se ha puesto ningún valor en [0044F07C], con lo que se deduce que hay un 0 que hará que no se produzca el salto. Vamos a encargarnos de meter un 1 en [0044F07C] para que ello remedie todas las futuras comparaciones. Preparados para escribir algo en ensamblador:

a 4314f8 (+intro) > Ahora Softice nos deja escribir:

MOV DWORD PTR [0044F07C],01 (meter 1 en la dirección de memoria mágica)
JMP 00413592 (saltar lejos a la fuerza)

Softice nos dirá inmediatamente cuales son los bytes que corresponden a estas instrucciones:
c7057CF0440001000000 (fijémonos en el [0044F07C], anotado al revés! Típico!) EB25

(por cierto, los cambios han sido aplicados después de la primera comparación, con lo que hay que cambiar 752F6869030000E84030fffff por los nuevos bytes. Da igual cómo quede el código siguiente ya que el salto forzoso hará que nunca se llegue hasta él).

A partir de este momento, todas las opciones limitadas estarán activadas, ya que al hacer la cmp dword ptr [0044F07C], 00000000 nuestro amigo siempre se encontrará con un 1 que hará que nos saltemos el mensaje molesto!!!!

Bueno. Ya hemos visto una forma más de eliminar mensajes y limitaciones. El error de siempre es el exceso de información. La utilización demasiado recurrente de la

dirección de memoria siempre es algo sospechoso puesto que salta a la vista que actúa como flag de chequeo!!!!

UNA FORMA DE ENCRIPTACIÓN: XOR

(atención!! leer con mucha calma)

Cuando asistimos desolados a la visión de un hardcoded serial válido, algunos nos preguntamos por qué motivo el autor no ha recurrido a un mínimo giro que haga menos visible el código válido. XOR (una operación matemática) es una instrucción que permite hacer que una cadena de texto, por ejemplo, tenga un aspecto diferente después de aplicársele una simple operación. El crackme2 de Cruehead, estudiado en lecciones anteriores, puede ilustrarnos algo sobre esta instrucción sobre la que debería decirse algo en cualquier cursillo de iniciación al crackeo (aunque es algo que corresponde a una lección algo avanzadilla)

Intentaremos crackear el crackme de Cruehead (eso es una bonita aliteración, cr,cr,cr...:) de dos puntos de vista. Primero trabajaremos con el método bruto y luego intentaremos analizar cual es el código real.

Al Atake. **METODO BRUTO**.
Desensamblamos el fichero con WDasm y nos encontramos con bonitos mensaje diciéndonos que no hay necesidad de utilizar el código desensamblado. A lo mejor es cierto, pero vamos a saltarnos las reglas.

Localizamos el mensaje de error aquí:

```
* Referenced by a CALL at Address:
|:00401243 >>> vayamos a ver desde donde hemos venido hasta aquí!!!!!

:00401349 6A00 push 00000000
* Reference To: USER32.MessageBeep, Ord:0000h

:0040134B E892000000 Call 004013E2
:00401350 6A30 push 00000030
:00401352 6860214000 push 00402160
* Possible StringData Ref from Data Obj ->"No luck there, mate!"

GotoCodeLocation "401243" y ahora tenemos lo siguiente:
:00401237 E87C010000 call 004013B8
:0040123C 83C404 add esp, 00000004
:0040123F 84C9 test cl, cl
:00401241 7407 je 0040124A >>>> YOOOHOOOO!!
:00401243 E801010000 call 00401349 >>>mensaje de error
:00401248 EB9C jmp 004011E6
```

Si invertimos este salto condicional (jne por je)saltaremos a la zona de mensaje de felicitaciones con cualquier código incorrecto que utilicemos!!!! Prueba superada!! Tratándose de un Crackme, vale la pena intentar aprender algo más...Vamos a buscar el código correcto!!!

PASO A PASO....
Softice : **bpx getdlgitemtexta** / ctrl+d / entramos cualquier cosa "axbxcxdxexfxg"

Al darle al OK, regresamos a Softice, le damos a F12 para aparecer en el ejecutable que nos interesa. Ahora vamos a buscar donde está el código estúpido que hemos introducido:

S 0 l ffffffff "axbxcxdxexfxg"
Softice lo encuentra en xxx:40217E. Vamos a poner un "bpm 40217E" para que el proceso se detenga cuando se acceda a esta zona de memoria para manejar nuestros datos. Le damos a F5 para continuar y !bboooom!! De vuelta a Softice,..alguien está jugando con nuestro código....veamos qué pasa: Aparecemos en 401373

```
00401371 8A06 mov al, byte ptr [esi]<< al = nuestro código,
uno a uno
:00401373 84C0 test al, al
:00401375 7419 je 00401390
:00401377 FE0518214000 inc byte ptr [00402118]
:0040137D 3C41 cmp al, 41
:0040137F 7204 jb 00401385
:00401381 3C5A cmp al, 5A
:00401383 7303 jnb 00401388 (*)
```

Nuestro número pasa carácter por carácter por este pedazo de código, donde es comparado con 41 (A) y con 5A (Z). Fijaos que estaban esperando letras mayúsculas. Si uno escribe una a (valor 61), el programa salta a una rutina(*) que le restará 20,

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401383(C)

:00401388 E825000000 call 004013B2
* Referenced by a CALL at Address:
|:00401388

:004013B2 2C20 sub al, 20 >> réstale 20, convirtiéndolo en mayúscula!!!
:004013B4 8806 mov byte ptr [esi], al
:004013B6 C3 ret
```

convirtiéndose pues en 41, es decir , A. ¿Lo habéis visto?, está pasando a mayúsculas lo que no lo esté!!! Eso quiere decir que el código correcto funcionará en mayúsculas o minúsculas!! Una vez todo el código ha sido verificado, llegamos a la parte en que nuestras letras (bueno, su valor hex.) es transformado:

```
:0040139D 8A8FA3214000 mov cl, byte ptr [edi+004021A3]
:004013A3 8A1E mov bl, byte ptr [esi] >> BL=nuestra letra
:004013A5 84DB test bl, bl
:004013A7 7408 je 004013B1
:004013A9 32D9 xor bl, cl >> CL xor BL. Ojo!!!!
:004013AB 881E mov byte ptr [esi], bl >>guarda el resultado
:004013AD 46 inc esi
:004013AE 47 inc edi
:004013AF EBEC jmp 0040139D
```

Veamos qué tenemos aquí. El valor hex. de nuestro código va pasando abl y una vez allí, se produce la operación clave,...CL XOR BL. Se produce una operación de xoreo. ¿Qué contiene "CL"? Pues bien, la primera vez contiene 4D (el hex. de M). La segunda vez que pasamos por aquí, nuestro segundo carácter es xoreado por un CL que ahora

equivale a 65 (el hex. de "e"),...más tarde nuestro tercer carácter será xoreado por un CL equivalente a 73 (hex. de "s")...

Esta operación se va repitiendo sucesivamente...Pronto nos daremos cuenta de que el valor fijo de CL se va sacando de la siguiente secuencia: "Messing-in-bytes". 16 caracteres!! Nuestro código deberá contener 16 caracteres para que puedan ser xoreados cada uno de ellos con cada uno de estos caracteres.
Finalmente, nos daremos cuenta que, con el xoreo, NUESTRO código inicial ha cambiado, se ha convertido en algo diferente (CODIGO NUESTRO XOREADO). Pues bien, llegará un momento en el que el programa compara CODIGO NUESTRO XOREADO con un CODIGO FIJO INTERNO, que por cierto, tiene esta pinta ".,76;=(.=&.1-;7>".

Esta comparación CODIGONUESTROXOREADO / CODIGOFIJOINTERNO tiene lugar aquí:

```
:004013B8 33FF xor edi, edi
:004013BA 33C9 xor ecx, ecx
:004013BC B110 mov cl, 10
:004013BE 8B742404 mov esi, dword ptr [esp+04] > ESI=CODIGONUESTROXOREADO
:004013C2 BF50214000 mov edi, 00402150 >>edi=CODIGOFIJOINTERNO
:004013C7 F3 repz
:004013C8 A6 cmpsb
:004013C9 C3 ret
```

Si la comparación establece que son iguales, el programa volverá a

```
:00401237 E87C010000 call 004013B8
:0040123C 83C404 add esp, 00000004 >> aquí
:0040123F 84C9 test cl, cl
:00401241 7407 je 0040124A
```

y todo se resolverá satisfactoriamente.

Pensemos pues!! Debemos entrar un CÓDIGO que, una vez xoreado por una serie de valores (Messiing-in-bytes) debe ser igual a .,76;=(.=&.1-;7>. ¿Cómo se soluciona esto?

Ejemplo estúpido. Supongamos que tenemos un programa que requiere un solo número para ser registrado. No sabe cual es, pero sabemos que el programa lo coge, lo multiplica por 8 y espera que el resultado sea 24. ¿Qué numero deberíamos haber entrado? Solución: 3. ¿Cómo lo sabemos? 24:8= 3 . Hemos realizado la operación a la inversa. Con nuestro crackme2.0 haremos algo similar. El resultado esperado(.,76;=(.=&.1-;7>) lo xorearemos por el valor fijo (Messing-in-bytes) y sabremos Qué deberíamos haber entrado!!!! Ante todo cabe decir que el programa calcula las cosas según su valor en hex..

El valor hex. de .,76;=(.=&.1-;7> es 1F 2C 37 36 3B 3D 28 19 3D 26 1A 31 2D 3B 37 3E
El valor hex. de Messing-in-bytes es 4D 65 73 73 69 6E 67 5F 69 6E 5F 62 79 74 65 73

Softice puede ayudarnos para realizar el xoreo invertido. Para calcular el primer carácter que deberíamos haber entrado escribimos lo siguiente en la ventana de comando de Softice:
? 1F^4D = 52 (R) (^equivale a XORear).
Para el segundo carácter que deberíamos haber entrado:
? 2C^65 = 49 (I)
Para el tercer carácter:
? 37^73 = 44 (D) .....

Podría realizar las trece operaciones restantes pero disculparéis su omisión. Hacedlo vosotros mismos. El resultado es

52 49 44 45 52 53 4f 46 54 48 45 53 54 4f 52 4d
R I D E R S O F T H E S T O R M

VOILÀ!!! El código correcto!! :)
Si queréis ahora seguir los pasos con Softice, veréis que nuestro código es xoreador por Messing-in-bytes y que el resultado es el esperado .,76;=(.=&.1-;7>

Un buen programa para ver cómo se calcula un serial real a partir de una operación de XOReo es UNINSTALL MANAGER 2.50/2.60. El programa recoge el valor ASCII de nuestro nombre y lo XORea por determinado valor (y en la versión 6 le suma 1). El resultado de la operación acaba siendo el código válido!!!!

Para un programador cauteloso, resulta mucho más inteligente utilizar un hardcoded serial-post-XOR-eado que un hardocded serial real. En el segundo caso, el serial corresponde a aquello que el usuario debe introducir, mientras que en el primer caso, el usuario debe introducir un código que, después de pasar por una serie de operaciones XOR, deberá ser igual a aquello hardcodeado. Sin duda habrá alguna forma de esquivarlo, pero al menos se habrán tomado más precauciones.

Sin duda,..la lección 9 ha sido ddeemmaassiaaddooo deeenssaaaaaa. Hasta la 10!!!!

Lección 10 – Cursillo Otoñal de Craqueo (29/12/99)

LAS OTRAS HERRAMIENTAS

No sólo Softice y WDasm van a acompañarnos en nuestro viaje crackeante. Hay muchas otras herramientas que nos serán de utilidad en otras situaciones y que nos aportarán informaciones valiosas a pesar de que no hayan sido diseñadas como herramientas para el crackeo. Empieza el viaje....
(http://protools.cjb.net - http://protools.hpage.net - http://w3.to/protools)

P R O C D U M P 1. 5 y G E T T Y P (para Acdsee 2.42)

Últimamente no es nada raro comprobar que un buen número de programas shareware tienen comprimido o encriptado su ejecutable. Ello implica, por ejemplo, que si localizamos algo que parchear con Softice, no podremos encontrar la cadena de bytes que nos interesa con el editor hexadecimal puesto que el fichero ejecutable está comprimido. Tampoco nos servirá de nada un desensamblador como WDasm. Está claro que podemos optar por parchear en memoria con la ayuda de un loader (R!sc's Process Patcher), pero para los más puristas llega esta utilidad excelente: Prodump!!, un producto capaz de "sorprender" a un ejecutable comprimido en el momento en que se está descomprimiendo y volcar su contenido a disco. Sería injusto decir que es únicamente un "desempaquetador", puesto que sus habilidades son muchas más. Vamos a estudiar su uso para resolver el tema del crackeo de ACDSEE 2.42:

A C D S E E 2 . 4 2 (Esta fue la primera versión que acdsystems sacó con ejecutable comprimido. Hasta su versión 2.41 era posible desensamblar, softicear y parchear sin problemas)

El ejecutable del programa no puede ser desensamblado con WDasm ni cargado con Symbol Loader de Softice. Aunque sí es posible entrar en el código utilizando breakpoints (bpx getversion -típico al iniciarse un programa- o bpx getdlgitemtexta, ...), no nos será posible cambiar bytes con un editor hexadecimal ya que lo que veamos en Softice no aparecerá por ningún sitio cuando lo busquemos dentro del editor hex.

En primer lugar procedamos a identificar el tipo de packer (empaquetador) que se ha utilizado. Para ello nos valdremos de la utilidad :

G e t T y p. Existen versiones del producto para MS-dos y Windows. Nos vamos al directorio donde está el ejecutable acdsee32.exe, copiamos allí el fichero gtw.exe y desde ms-dos anotamos: gtw acdsee32.exe

Muy rápidamente seremos informados de que estamos ante un ejecutable comprimido con ASPACK 1.0.8.3. Bueno, ya sabemos alguna cosa! Arrancamos Procdump. Le damos al botón UNPACK y escogemos el tipo de empaquetador que nos interese. Estamos de suerte, ASPACK 1.0.8.3 aparece entre la lista, así que lo seleccionamos. A continuación se nos pedirá el nombre del ejecutable que deseamos desempaquetar: Acdsee32.exe.

A continuación se nos dirá que esperemos a que todo el proceso esté cargado. En realidad hay que esperar a que todo el programa Acdsee haya sido arrancado. Cuando veamos su icono en la barra de tareas inferior, estaremos seguros de que podemos darle a Aceptar. Esperamos... y nos piden un nombre para guardar el nuevo ejecutable en el disco duro:

acd-dump.exe (por ejemplo)

El ejecutable original era de 775 kb., mientras que el nuevo (desempaquetado) ocupa 1558 kb. Comprobemos que el nuevo también funciona. Doble click! Sí, la cosa funciona: el programa Acdsee también ha arrancado desde el nuevo ejecutable. Perfecto! Todo en orden.

Llegados a este punto, si ya hubiéramos localizado los bytes hex. a cambiar, podríamos hacerlo con toda tranquilidad, puesto que serían localizables con un editor hex. (tal como los habríamos visto en Softice).

Aunque para nuestro abordaje no lo necesitaremos, ¿Probemos a desensamblar el ejecutable nuevo? Arrancamos WDasm y.....mierda! No se puede desensamblar. La poca información que nos da nos indica que hay una pequeña medida de seguridad antiDesensamblador. Bueno. Menos mal que nos gusta leer todo lo que encontramos por ahí y que el bueno de Karpoff recomendó en su momento lo siguiente:

1-Volver a Procdump
2-Seleccionar PE-edit
3-Seleccionar el ejecutable desempaquetado acd-dump.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 seleccionar al primero de ellos)
6-En el apartado Section Characteristics cambiar el C0000040 por E0000020.

Ho hace falta entender nada de cabeceras PE (portables ejecutables) ni nada de nada. Es cuestión de creerse que estos datos impedían el desensamblamiento y elogiar la labor de

las personas que han llegado a percatarse de este detalle y que son capaces de entender algo del tema de los PE.

Una vez realizados los cambios, como por arte de magia, ya podemos desensamblar el ejecutable dumpeado (acd-dump.exe) sin problemas (bueno, con algún problema, no aparecen los nombres de las APIs por el listado muerto,...en fin,...da igual)

PREPARADOS-LISTOS-YA....

¿Por donde empezamos nuestro ataque? Pensemos....mmmmmmm. ¿Cual es la limitación de Acdsee? Cuando estamos visualizando imágenes a pantalla completa nos aparece cada X imágenes un cuadro que nos recuerda que estamos ante una versión no registrada, bla, bla...Por cierto, hay un detalle que no debe pasarnos por alto. Se oye un "beep" antes de mostrarnos el cuadro. Bueno. Ya tenemos por donde atacar.

Softice > bpx messagebeep (breakpoint en cuanto se llama a la API encargada del pitido) > Volvemos a Windows:

Empezamos a visualizar imágenes a pantalla completa y de golpe !!beep!! Estamos en Softice!! Con un f12, aterrizamos en pleno código de acdsee, justo después de que fuera llamado el pitido:

```
* Reference To: USER32.USER32.DLL [MESSAGEBEEP]

:00458A03 FF154CF44D00          Call dword ptr [004DF44C]
:00458A09 8B0D10045000          mov ecx, dword ptr [00500410] >>>aquí
```

Si nos fijamos,(ctr + flechaarriba) hay un salto condicional anterior a estas instrucciones que -de haberse producido- podría haber evitado el pitido, pero no evitaría lo que vendrá a continuación, que es el super-nag recordatorio. Por tanto, limitémonos a tracear con F10 hasta que acabe el código referente al nag y salgamos de esta rutina.

Hay que tracear un buen rato con F10, incluso darle al F12 cuando nos colemos en kernel, user,..f10,f10,f10...hasta llegar a un precioso RET, que nos indicará que salimos de la rutina encargada del pitido-y-el-nag. En cuanto cruzamos este delicioso RET aparecemos en:

```
:00459ADC 7536              jne 00459B14 <<<< *********!!!!*****
:00459ADE 8B54240C             mov edx, dword ptr [esp+0C]
:00459AE2 6A01              push 00000001
:00459AE4 52                push edx
:00459AE5 8BCE                mov ecx, esi
:00459AE7 E894EBFFFF          call 00458680 <<<luego este es el call maligno que contenía el
pitido-nag

:00459AEC 85C0              test eax, eax>>>>>aparecemos aquí
:00459AEE 7524              jne 00459B14
:00459AF0 8B8ED4000000          mov ecx, dword ptr [esi+000000D4]
```

Ya hemos identificado el CALL maligno. ¿Qué tenemos aquí? Justo encima aparece un salto condicional que se dirige a 459B14, es decir, muy lejos de la zona maligna. Pongamos un bpx 459b14 para ver qué pasa. Cuando el programa pase por este salto, Softice hará detener el proceso:

Una vez puesto el nuevo breakpoint, le damos a F5 , estamos en Windows y procedemos para visualizar una nueva foto...!!plash!!! Regresamos a Softice, justo

encima de :00459ADC jne 00459B14 (jump) y el bueno de Softice nos dice que el salto se va a producir. Ello indica que no veremos el nag :) Démosle a F5 para volver a visualizar otra imagen...!!plash!! Otra vez en Softice, en el mismo salto y dispuesto a producirse de nuevo. Mmmmmmmm...

El programa pasa siempre por este salto cuando hemos dado la orden de visualizar una foto y aparentemente siempre tiene instrucciones de que el salto de produzca,....¿siempre? No!! Si continuamos dándole a F5 y visualizamos una nueva imagen con Acdsee, acabaremos encontrándonos con que en un momento dado aparecemos en :00459ADC jne 00459B14 (nojump) pero el salto no va a producirse!!! Esto quiere decir que llegaremos hasta el cALLmaligno y ...pitido y nag a la vista. Ya lo tenemos!!!!

El programa recorre siempre la instrucción :00459ADC jne 00459B14 antes de visualizar una nueva foto. Casi siempre viene con instrucciones de saltar (con lo que saltamos lejos de :00459AE7 call 00458680 (call que en su interior contiene el pitido-nag), pero cuando pasa por esta misma instrucción y no lleva instrucciones de saltar, acabamos llegando a la zona no deseada. ¿Qué determina que se salto o no? Ni idea. A lo mejor es algo totalmente aleatorio, pero da igual. ¿Como podemos hacer que salte siempre? Bien fácil, cambiando

00459ADC 7536 jne 00459B14 por
00459ADC EB36 jmp 00459B14
(saltarse siempre el call-maligno!!!)

Buscamos los bytes implicados en el editor hexadecimal, los encontramos, guardamos los cambios y...tenemos una versión no registrada de Acdsee pero sin molestias adicionales!!!!

Para los detallistas que prefieran eliminar el mensaje recordatorio que aparece en la barra superior siempre queda el recurso de buscar la cadena ACII con el editor hexadecimal y retocarla o poner otra cosa. Por cierto, hay que saber que el mensaje Acdsee242- [Unregistered] que se ve cuando estamos en la pantalla general, no es el mismo que se ve cuando estamos visualizando imágenes a pantalla completa. El primero de ellos cuesta un poco de encontrar puesto que está en formato ancho: [. U . n . r . e . g.....]

(aparece en la dirección F1EC0), mientras que el segundo aparece en formato normalillo, así que una simple búsqueda de "[Unregistered]" nos llevará hasta la dirección 15D613, donde podremos cambiarla por lo que nos plazca.

F I L E M O N

Esta utilidad nos permite saber qué ficheros consulta, abre o necesita la aplicación que estemos ejecutando. En su apartado Filter...se nos permite especificar los procesos a incluir o excluir, así como la "lectura" y/o la "escritura" en determinados ficheros. Mr. Nobody recuerda haber leído en algún tutorial sobre crackeo de una aplicación para splittear ficheros de gran tamaño que File Monitor avisaba de que al iniciar la aplicación se buscaba determinado fichero (que no aparecía por ningún sitio en el disco duro). Pues bien, si se creaba un fichero con el nombre que allí aparecía, inmediatamente se pasaba a tener el programa registrado. La existencia de aquel fichero misterioso era el que determinaba el status de registrado/no registrado. Curioso, ¿no?

Aunque no sea un caso frecuente, al menos queda ilustrado el valor de FileMon. Veámoslo en acción.... Esm Software's Amazing JPEG Screen Saver (es algo antiguo!) **http://ourworld.compuserve.com/homepages/esmsoftware**

Al ejecutar el protector de pantallas vemos que aparece bajo un mensaje en el que leemos "This is a demonstration....". Si intentamos localizar con un editor hex. dicha cadena de texto en el fichero amazingimages.scr, nos daremos cuenta de que no aparece en su interior. ¿De dónde sale pues el mensaje? A lo mejor está encriptado,...o a lo mejor proviene de otro sitio!! arranquemos filemon y anotemos debidamente configurado. Una vez inicializado el protector de pantalla, podemos volver a FileMon y observar en el enorme listado generado curiosidades que saltan a la vista como...

.....

4 Ajpeg32 Seek C:\WINDOWS\ESMJPG32.DLL SUCCESS Beginning Offset: 54272

5 Ajpeg32 Read C:\WINDOWS\ESMJPG32.DLL SUCCESS Offset: 54272 Length: 4096

.....

Las referencias a esmjpg32.dll son demasiadas como para pasar desapercibidas. Dicha librería parece tener mucho que ver con la ejecución del protector de pantalla. Si abrimos la DLL con un editor hexadecimal, no tardaremos en encontrar en su interior el molesto mensaje recordatorio "This is a demonstration SHAREWARE version...". Sólo nos queda ponernos sobre la primera letra (T = 54 hex.) de dicha cadena y anotar 00 (hex.). Con ello el programa pensará que ya se ha terminado todo lo que había por anotar !!!!

Aparte de otras sutilezas, FileMon también resulta un complemento ideal para monitorizar la posible búsqueda de ficheros sospechosos o ficheros raros que puedan indicar a un programa que ha expirado el tiempo de evaluación, etc.... Nunca se sabe!!

REGMON

Registry Monitor permite, como su nombre indica, monitorizar las actividades que tienen que ver con el registro de Windows. Indudablemente, es la herramienta ideal para saber dónde se alojan los datos que indican a la mayoría de programas que están registrados.

Cuando tenemos activada esta utilidad, podemos saber qué claves se están leyendo o creando, con lo que conseguimos información acerca de las "ramas" que puedan haberse creados, claves que hayan sido consultadas,etc..

Usos varios:

*Cualquier persona que haya intentado registrar Getright 4.x con un código antiguo se habrá encontrado con la sorpresa de serle imposible volver a intentar entrar un código. Si hubiese tenido activo Regmon, se habría dado cuenta de la creación de un par de claves en sitios extraños que son los que avisan al programilla de que el usuario ha recurrido a un código vetado.

*Imaginemos que algún programador ha ideado la creación de una clave en el registro para el momento en que un programa de evaluación limitada haya caducado. Cuando uno hace retroceder el reloj de Windows, se encuentra con que ya no es posible re-

arrancarlo. RegMon podría haberse dado cuenta!!!! Sólo hará falta darle al Inicio > Ejecutar >Regedit..
para editar el registro de Windows, buscar la clave maldita y erradicarla!

*Enotepad 2.13: Si lo arrancamos y monitorizamos las claves que busca en el registro (Name: KeyCode:,...), no tendremos más que crearlas nosotros mismos, con los datos que queramos y el programa considerará que está registrado ya que estas claves ahora sí existen!!! Muy fuerte,¿no?

*Un ejemplo:Nico's Commander 5.30:
Con el Regmon previamente puesto a punto, arrancamos el programa y nos indica, por ejemplo, que nos faltan 4 días para que caduque nuestra evaluación del programa. Volvemos a RegMon, que nos da un listado de todo lo que se ha consultado en el registro. Hay algunas claves con nombres evidentes, pero algunas pueden resultar sospechosas y no está mal sondearlas:

72 Nc QueryValueEx HKCU\Software\NicoCuppen\NC\WinPositie\wp.flags SUCCESS

Flags? Qué bien suena esta palabra. Si le damos al Regedit y buscamos en HotKeysCurrentUsers (HKCU) acabaremos viendo que el valor de dicha clave es: 2aad.

Siempre nos han dicho que no juguemos con el Registro, pero ¿Probemos a cambiar su valor? No parece posible que suceda nada malo, ¿no? Probemos con 0? Si hacemos doble click en la clave y cambiamos el valor por 0, al re-arrancar el programa observaremos que volvemos a tener 30 días evaluativos. Eso está mejor!!

¿Os imagináis qué podríamos hacer si arrancáramos algún programa de la compañía Toggle en que se busca una clave llamada "Registered"? ¿Qué valores puede estar buscando? La verdad es que hay pocas posibilidades: 0 o 1, True o False, Yes o No...

EDITORES HEXADECIMALES

No descartemos el poder crackeante de los editores hexadecimales. Su capacidad para cambiar cadenas de texto plenamente visibles o para visualizar hardcoded serials no debe pasarnos por alto.

Hay un buen número de protectores de pantalla antiguos que son plenamente crackeables con la localización y eliminación del mensaje molesto (convirtiendo la primera letra en 00 o sustituyendo todo el texto por espacios en blanco)
Avi Constructor Trial Version >

Un cómodo programa para convertir conjuntos de imágenes a ficheros AVI que a partir del quinto frame se decide por molestar al usuario con un mensaje recordatorio en sus creaciones. No hay más que buscar la cadena de texto molestón "Avi Constructor Trial.." y sustituirlo por espacios en blanco.

Un truquillo: Ya que encontramos más de una cadena de texto con las mismas palabras, resulta muy útil realizar los siguientes cambios:
Av1 Constructor Trial
Av2 Constructor Trial
Av3 Constructor Trial
Av4 Constructor Trial

(dejamos un pequeño indicador en cada texto que nos encontremos, guardamos los cambios y al crear un nuevo AVI, sabremos cuál de ellos es el responsable del naggeo "Av3 Constructor Trial", cuál es el responsable de la barra superior "Av2 Constructor Trial", cuál corresponde al mensaje que aparece en la ventana About...). Ahora ya sabremos qué debe ser eliminado.

OTRAS HERRAMIENTAS

Siempre que visitemos el site de algún grupo cracker, no olvidemos dar un vistazo a la típica sección TOOLS, en la que seguro que nos encontraremos con otras utilidades que completarán nuestros estudios crackeatorios:

S M A R T C H E C K: Una suculenta herramienta que monitoriza todo movimiento producido en el seno de aplicaciones realizadas con Visual Basic. Cuando se introducen códigos y se recibe el mensaje de error, el Sr.SmartCheck ha tomado nota de todo lo que se ha producido y lo que se debería haber producido :)

P A T C H E R S V A R I A D O S: Típicas utilidades que comparan un fichero original con un fichero crackeado, toma notan de los cambios que han tenido lugar y crean un ejecutable cuya función es parchear cualquier otro fichero original con los cambios crackeantes pertinentes. Se supone que sólo recurren a este tipo de programas los crackers que no se atreven a programar cracks por su cuenta pero desean distribuir de alguna forma los resultados de sus "investigaciones". Hay patchmakers para MS-dos (suelen crear ejecutables muy pequeñitos) y para Windows (con ventanillas preciosas, iconos,..y algo más tamañosos).

A D V A N C E D C O M P A R E: Cualquier programa que sea capaz de mostrar las diferencias entre dos ficheros binarios es una herramienta de aprendizaje excelente. Pensemos en cuánta información hay en un "crack" de los que pululan por la red. Si nos guardamos una copia del ejecutable original y aplicamos luego el crack, podremos hacer luego una comparación del original y el crackeado y ver qué cambios se han aplicado. Lo interesante empieza aquí: por qué habrán cambiado estos bytes por 9090909090? ¿por qué este cambio de un byte 74 por un 75? De hecho, potencialmente, cada crack que circula se puede convertir en un tutorial en potencia para nosotros, si somos capaces de entender el sentido de los cambios introducidos. Observar los cambios y aprender,...siempre dentro del limitado e íntimo mundo entre nosotros-y-nuestro-ordenador :) Entonces, ¿sería posible observar los cambios y crear nuestro propio nuevo ejecutable-patcher fingiendo ser unos grandes crackers? Sí, claro,...pero ¿en qué tipo de insectos nos convertiríamos? Supongo que esto nos valdría para ser considerados como rey de los "Lamers" (deducir el significado), con diferencia. No olvidemos jamás nuestros objetivos: curiosear, aprender, buscar explicaciones, "jugar" con los bytes,... No hay necesidad alguna de ser impostores, fingir, dañar o perjudicar a otros.

O T R O S: Está claro que ahí fuera quedan muchas más cosas por probar:

Editores de Recursos (con los que se pueden eliminar ventanas y hacer muchísimas cosas más),

Programas que realicen búsquedas en ficheros binarios: WindowsCommander, por ejemplo, permite hacer un listado de cadenas de texto que encuentre en un ejecutable. Hay gente que ha conseguido hardcoded serials únicamente echando un vistazo a la lista de cadenas encontradas, sin necesidad de desensamblar!

Espiadores de API's para saber donde hay que breakpointear o cualquier herramienta que monitorice cualquier aspecto es susceptible de convertirse en una buena herramienta para el arte del crackeo

Utilidades complementarias para Softice (algunas only para "profesionales", otras sencillas, como aplicaciones que indican la dirección hex. con sólo indicarles la dirección de memoria que hemos localizado con Softice,...)

Está claro que los tiempos y los métodos cambian. Se ha entrado en una carrera muy estimulante en lo que a protecciones se refiere. Se supone que los maestros del cracking han tenido algo...o mucho que ver con ello. En cierta manera se está forzando a la comunidad de programadores a salir de la comodidad y a esforzarse en la creación de protecciones serias, es decir, como mínimo no-superables por novatos con serias carencias de conocimientos generales en informática. A veces da la impresión de que algunos programadores actúen como "gurús" que piensan que "la plebe" no se entera de nada. Hoy en día, incluir hardcoded serials en el código es casi un insulto. Dejar que un programa pase a estar registrado con un simple cambio de un salto condicional , sin ningún otro chequeo, es una prueba del desinterés por parte de algunos programadores en proteger adecuadamente sus productos. El noble arte del crackeo puede interpretarse de múltiples maneras.

Mr. Nobody se queda con la modalidad de crackeo consistente en divertirse, a modo de juego emocionante, a veces, una especie de "arte de engañar al código" y otras veces, una simple actividad voyeurística. Los productos shareware son la materia prima de la que se sirve esta lúdica y agradable actividad que no se basa necesariamente en el deseo de adquirir un programa de forma gratuita. Cuando alguien se engancha al arte crackeatorio, no persigue necesariamente la utilización fraudulenta de un programa. Lo realmente bello es el "viaje" que le ha conducido a conseguir lo que se proponía. Acto seguido, en la mayoría de las ocasiones sólo queda deshacerse de aquel producto que, probablemente, no interesaba para nada. En último caso, sólo me queda recordar las palabras de un gran maestro: "Si crees que vale la pena utilizarlo, vale la pena pagar por él"

VOLANDO LIBRES!!!!

Después de 10 lecciones juntos, llega el momento de saltar del nido, sin temores. La idea con la que fue concebido el COC ha sido la de sentar unas mínimas bases para poder mejorar como principiantes,..y este caminos sólo se consigue a base de practicar (NULLA DIES SINE CRACKO) y leer todo lo que se nos eche por delante.

Si hemos sido capaces de "seguir" mínimamente las lecciones impartidas, estaremos en disposición de poder leer con seguridad la mayoría de tutoriales y cursillos que pululan por la red. La mayoría de ellos mucho mejores que el propio COC y todos ellos igualmente válidos.

Id preparando un par de paquetes de 500 folios y algunos recambios de tinta para la impresora, porque estos son algunos de los sitios que podremos visitar en nuestro viaje (está claro que los tutoriales hay que leerlos sobre papel, ¿no?)

EN CASTELLANO:

Crack el Destripador. Un gran maestro, amable y bondadoso:
http://welcome.to/craaaack

http://www.strega.org/TNT/tntschool.htm

http://www.wco.com/~micuan/WKT/Intro.htm

UN GRAN NEWBIE CON MUCHO TALENTO Y GANAS DE ENSEÑAR:
http://personal2.redestb.es/karpoff/Manuales.htm

http://kut.tsx.org

material ungerground en general
http://members.tripod.com/~WonderBoyz

EN INGLÉS

http://astalavista.box.sk
(Una búsqueda de "cracking tutorials" os llevará a diversas páginas con links a algunos de los tutoriales clásicos del tema. Desde crackeo por DOS, hasta iniciaciones variadas, ensamblador para crackers novatos,...)

MÁS DE 10 MEGAS EN FICHEROS HTM SOBRE CRACKEO (TODO TIPO DE NIVELES) http://129.105.116.5/fravia

55 TUTORIALES ZIPEADOS CON UNA MEDIA DE 5 EJEMPLOS EN CADA UNO. http://msjessca.da.ru/

CRACKZ: una joya con estudios de mochilas, trials de tiempo,..todo analizado al detalle y explicado magistralmente http://www.wco.com/~micuan/

75 ESTUDIOS SUPER ANALÍTICOS Y BIEN EXPLICADOS. INLCUYE EL CONJUNTO DE TUTORIALES DE +ORC, LA PERSONA QUE SENTÓ LAS BASES DEL CRACKEO ACTUAL. http://www.idca.com/~thesandman/

TUTZ VARIADOS, KEYGENERATORS,.. http://202.103.111.109/hambo/

UN BUEN MONTÓN DE TUTORIALES Y MATERIAL RELACIONADO CON CRACKMES
http://adenozin.tsx.org/
http://www.chesspro.net/TbC/adenozin/tuts.htm
http://203.147.214.201/iced-network/adenozin/tuts.htm

EN FRANCÉS

www.multimania.com (Introducid en el buscador algo como "cracking", "Softice",..y os aparecerán un montón de sites relacionados)

No olvidéis jamás visitar la parte de "links" de cualquier sitio en el que estéis. Inevitablemente os llevará a nuevos lugares y siempre descubriréis a alguien nuevo en la escena.

No hace falta añadir que no es necesario disponer del programa original para disfrutar de un tutorial. La lectura debería ser un placer en sí mismo. Además, siempre podemos aprovecharnos del método para actuar miméticamente cuando analicemos una versión diferente del programa sobre el que hemos leído, etc..

El COC llega pues a su fin, aunque no estaría nada mal que el "viaje" continuara en forma de periódicas publicaciones de los que han aprendido algo y sus aventuras con pequeños amigos shareware. Al parecer hubo un tiempo en que las newsgroups de crackeo eran algo parecido a esto, un forum en que se explicaban cómo se había conseguido esto o aquello, bla,bla,bla... Mr. Nobody se compromete por su parte a intentar enviar algún estudio complementario en cuanto le sea posible.

El cursillo de crackeo otoñal (casi lo conseguimos, sólo le hemos robado algunos días al invierno) llega al final de su propósito. Cuando leamos cualquier tutorial en que vemos que fuerzan un salto hacia un mov eax,1 o un nopeo de un CALL, o nos hablen de determinado breakpoint,..sabremos más o menos por dónde se supone que van los tiros. No hay ningún problema en leer estudios de los que entendamos poca cosa. Mr. Nobody os asegura que uno puede sorprenderse de llegar con el tiempo a entender algo que en su momento le pareciera imposible.

Sin ningún ánimo cínico, Mr. Nobody quisiera añadir que desea que el COC haya sido de utilidad tanto a aspirantes crackeantes como a futuros programadores que se decidan a proteger debidamente sus programas. El C.O.C. no hubiera sido posible sin el estímulo, la colaboración, los ánimos, la ayuda, las preguntas, las respuestas,...de las personas siguientes, a quienes Mr. Nobody considera parte de su alma: (no hay orden preferencial. Cada uno sabe lo mucho o poco que ha aportado)

Fido Alfil, Elkronos,Crack el Destripador,666burn,Javi'x,"Quillo", "Avalanche", Jose B.Moreno,Ignacio, "Pit",Ka_lim,"Elena", User12, "Bubba","Gideon", Gulliver,JMCF, Jorro, rafa 10, FCP, Sun Devil, RAGE, Davi Mese, 2073r, Carlos Q, Clash,Pecador, D.C.R., Fer, Devil, Ed_Marquez(Ed_M),"MT", Fernando, Frank ;David Mese ;Lisardo Quintanilla, Hommer, Cypher, Net64, Microled, andra93 , Javier Moratinos, ATSA, klonone, jorgr lópez, Mebupa, José Monzón ,Miguel Barrio Orsikowsky, Augusto Izquierdo , Robur, Skay, Frutis, JFH, DjTracker, Perico, Mad Moon, Dani, Patata, dfb, Jesuso, Juan Carlos Moreno, DANIGCBF/Iñaki ,d@vi , Lucio Hdez, Jesulillo ,Juan , puck ,Willyfax, LEFRIC, BPR ,jose-jmcentau, Manuel Reixach ,"infeliz" Drakonia, Okupa2, Patxi, Javivi, PROFUND LTDA, Seldon, Ramón Calderó, MOLGAR, Hummer, Santiago José López Borrazás, Snake Shamer, Pablo Maudes, Pepito ,Grillo, Rotulikos, Angelux, Maria, ACL & Bros ,Miguel Gonzalez, JKD ,SCR y a cualquiera que haya seguido las lecciones "en silencio" o a cualquiera quien Mr. Nobody haya podido olvidar (disculpadme!)

N U L L A D I E S S I N E C R A C K O

Lecciones del COC

http://come.to/ATSASOFT
http://leo.worldonline.es/albsolar/index.html
http://fly.to/TrAcKeR/index.html
http://www.gratisweb.com/cracking