Cracking By Softdownload

Indice

1.UN POCO DE TODO

¡Saludos Familia! He decidido crear esta serie de capítulos con un objetivo puramente educativo (sin ningún tipo de obscuro interés comercial). Mi objetivo es ayudar a los nuevos crackers en sus primeros pasos. Agradecimientos al TodoPoderoso +ORC, a su acólito +FRAVIA y a los miembros de WKT por su apoyo.

¿Qué es eso de crackear?
Crackear es el arte de reventar protecciones software/hardware con fines intelectuales, personales pero no lucrativos. Crackear también se llama ingeniería inversa (Reverse Engineering), ya que sin el programa fuente se es capaz de analizar el comportamiento del programa y modificarlo para tus intereses.

¿Por qué crackear?
Vivimos en un una sociedad que da asco. Es inconcebible que se destruyan o penalicen la producción de productos básicos (cereales, lácteos...) en aras de una estabilidad de precios. Vivimos en una sociedad donde el 95% de la gente se ve sometida al control de un 5 %. Toda persona debería tener un mínimo de recursos para ser feliz, pero esto no es así. El noble arte del crakeo es una herramienta para distribuir la riqueza entre la sociedad. Si necesitas un programa (que tienes en un CD-ROM) y no tienes una cuenta en un banco USA, ni 30 dólares, ¿por qué narices tienes que esperar y pagar si lo necesitas?, crackealo y publica el crack en Internet. Así ayudarás a la gente menos afortunada que está axifiada por esta sociedad desigualitaria.

¿A quién va dirigido este curso?
Este curso va dirigido a toda persona con interés en el crack y/o con la filosofía crack. Sin olvidar a los programadores.

¿Qué es lo que vamos a aprender?
Dejaremos de un lado el añorado DOS, para centrarnos en cracks para programas en W95.

¿No estaremos quitando el pan a los programadores de aplicaciones?
Los programadores viven muy bien a costa de los royaltis que pagan las grandes empresas y esos repugnantes yupis encorbatados con escasez de neuronas. Además, un programa crakeado es más conocido y utilizado que uno que no lo esté. Digamos que el crack es una forma de publicidad. Baste recordar el compilador de Pascal de la casa Borland. Originalmente fue copieteado y distribuido casi libremente hasta la saciedad. Borland conocía este hecho y por eso no introdujo ningún tipo de protección. Al cabo de poco tiempo, esos estudiantes se convirtieron en programadores que reclamaban a sus empresas la compra del compilador que sabían utilizar, el Pascal de Borland. Si las casas de soft ya son ricas sin nuestro dinero, ¿a qué estado de corrupción se llegaría si lo tuvieran?. Aunque parezca extraño este ensayo está dedicado a formar a los programadores , mostrándoles sus defectos y el camino para producir software de calidad.
En último caso depende de la conciencia de cada uno, si crees que un programador ha realizado una buena aplicación, que te es útil y que además está bien programada, entonces obra adecuadamente y recompénsalo registrándote. A decir verdad sólo he encontrado un programa de este estilo

¿Qué necesito pa esto de crakear?

- Interés y PazYCiencia.
- Algún conocimiento de ensamblador. Cuanto más conozcas mejor crackearás, pero para este curso mas bien poco, intentaré hacer las cosas fáciles.
- Ayuda de otro cracker más experto (por ejemplo Estado+Porcino)

2.IDEAS BÁSICAS SOBRE CRACKING

Un programa no es más que un montón de instrucciones (haz, esto, haz lo otro), una tras otra. Estas instrucciones, (en general) están en memoria, parte de ellas se encargan de impedir la ejecución del resto (el verdadero programa). Resumiendo, tienes un bonito programa tras una puerta (esquema de protección), nuestro objetivo es reventar la puerta, buscar la llave, pasar por las rendijas, quemarla..., como puedes ver una cantidad inagotable de ataques.

Un pequeño inciso, sólo puedes reactivar instrucciones que se encuentren en el programa. Por ejemplo, si se ha desactivado la opción de salvar, puede ser que el conjunto de sentencias para salvar el programa esté presente pero desactivado, o bien que simplemente no esté. Si están las sentencias, se pueden reanimar, sino están entramos en un mundo diferente, una evolución del cracking, la apasionante Reconstrucción Software, pero eso será tema de otro capítulo.

3.HERRAMIENTAS CRACK

Demos un breve repaso a las principales herramientas que utilizaremos a lo largo del curso. Existen otras muchas herramientas que las comentaré cuando nos sean necesarias.

Editor hexadecimal

Los programas no son más que un conjunto de instrucciones y cada instrucción no es más que un conjunto de bits, pero donde demonios se guardan esos bits?.

Los bits del programa se localizan en los ficheros, p.e. las instrucciones del programa de compresión arj se guardan en el fichero arj.exe. Hay algunos programas que no guardan todas sus instrucciones en único fichero, si no en varios, un ejemplo de esto son los programas que utilizan librerías dinámicas (o dll).

Un editor hexa, no es más que un programa, que permite "editar" los ficheros de instrucciones de otros programas, o sea, que permite ver, modificar, copiar, pegar... los bits de los programas. Para simplificar la cosa no se muestran los bits a pelo, sino que se muestran en hexadecimal, de ahí su nombre. Nosotros lo utilizaremos para alterar el comportamiento de los programas. Supongamos que conocemos la instrucción sentencia de la rutina de protección que debemos modificar, sea jz 23 y queremos modificarla por jnz 23, bien como toda instrucción no es más que un conjunto de bits, sea 0110 para jz 23 y 1001 para jnz 23, sólo nos queda buscar estos bits dentro del fichero ejecutable del programa

(que es, en general, el que contiene las sentencias del programa). Como usamos un editor hexa, debemos buscar la secuencia de un unos y ceros en hexa en el fichero del programa que queremos modificar. Si la secuencia que buscamos es muy común deberemos utilizar las instrucciones que se encuentran entorno a la instrucción a modificar.

Esto es muy importante, sólo debe existir una localización del patrón de búsqueda en el fichero, si existe más de una, debemos añadir a la búsqueda las sentencias de alrededor, sino se corre el riego de modificar la sentencia equivocada, lo que provoca casi siempre un "cuelgue".

Un crack, no es más que una modificación de las instrucciones de un fichero, así pues para hacer un crack debemos saber que instrucciones modificar y en qué fichero. Una vez crackeado el fichero, el programa se comportará siguiendo la nueva sentencia que le hemos modificado.

Hay un montón de editores hexa, yo os recomiendo UltraEdit-32 Professional, os lo podéis bajar de http://ftpsearch.ntnu.no/ (excelente herramienta de búsqueda en la Web, utilízala cuando no encuentres algún fichero) busca w32dasm. Yo utilizo la versión 4.31a. Cada cracker tiene su editor favorito, !encuentra el tuyo!. Este programa es shareware, así que tendrás que crackearlo, recuerda : lo primero que tienes que crakear son tus propias herramientas. Si no puedes crackearlo, busca otro editor hexa, los hay a montones.

Desensamblador

Un desensamblador toma un fichero de instrucciones en hexa y lo convierte a lenguaje ensamblador. El lenguaje ensamblador, es el conjunto de sentencias que entiende el microprocesador (tu Pentium o mi 486). El procesador es el corazón del ordenador, todas las sentencias son ejecutadas por él y sólo por él. Por ejemplo un 43 en hexa se transforma en inc eax. Se necesitan algunos conocimientos de ensamblador pa esto de crackear.

Nosotros usaremos el desensamblador para crakear con la técnica de la lista muerta que veremos más adelante.

Puedes pillar un magnífico desensamblador para W95 en http://www/expage.com/page/w32dasm , es shareware, así que deberás crackearlo. Próximamente le dedicaremos un capítulo al w32dasm y aprenderemos a crackearlo. Yo utilizo la versión 8.9, aunque una posterior es también útil.

Debugger

Un debugger permite ejecutar instrucción a instrucción (instrucciones en ensamblador, se entiende) un programa, por tanto también ejecutará instrucción a instrucción la rutina de protección, ya que es parte del programa. Esto nos permitirá analizar su comportamiento.

El mejor debugger para W95 es sin duda el Softice quo te lo puedes bajar de http://www.numega.com , es sin duda uno de los mejores programas que he visto en mi vida. Te recomiendo que te bajes también los manuales. El Softice es una de las mejores herramientas crack, cuanto más domines al Softice, mejor crakearás, y te aseguro que se pueden realizar verdaderas maravillas con él. Para trazar (esto es, pasar el debug) a

programas MS-DOS, puedes utilizar versiones anteriores del Softice o utilizar algunos de los excelentes debugger que hay para Dos como el SymbDebug.
Os indicaré un par de consejos y trucos para el Softice.

- **Instalación**
  Haz la prueba de pantalla, si te la saltas, lo más seguro es que cuando actives el Softice se te quede la pantalla a rayas. Busca el driver de la tarjeta de vídeo que tengas instalada en W95 dentro de la lista que se ofrece. Aunque la encuentres, haz la prueba de pantalla, es posible que no funcione correctamente. Si no encuentras tu tarjeta de vídeo o tienes problemas de pantalla, corta la instalación y selecciona "Adaptador de Vídeo Standard VGA", reinstala el Softice y utiliza ese mismo driver para la prueba de pantalla. Si sigues teniendo problemas te remito a la Doc del Softice (que deberías haberte bajado). Particularmente, con el adaptador Standard VGA no he tenido nunca problemas. Hay por ahí sueltos una ampliación de drivers de tarjetas para Softice, búscalos si no quieres pasa a la VGA Standard.

  Activa la opción de ratón, te evitará escribir bastantes secuencias hexa de instrucciones, te permitirá moverte libremente por las ventanas, seleccionar, copiar y pegar texto así como redimensionar las ventanas.

- **Configuración**
  El Loader es una pequeña utilidad que permite especificar el programa que queremos depurar y además nos permite configurar el Softice, En Edit/SoftIce Initialization Settings/Initialization string pon X;wl;wr;wd7; code on; y en Histoy Buffer Size pon 512 Abre el fichero winice.dat (dentro del directorio de instalación del Softice) , toda línea del tipo EXP=c:\windows\system\system.drv , quítale el punto y como inicial.

- **Ejecución**
  El Softice debe cargarse siempre antes que W95. Si estás en W95,
  reinicializa en modo MS-DOS y ejecuta WINICE.EXE, este cargará el Softice y W95. Te recomiendo que utilices un fichero bat.

Las ventanas que aparecen son:

- *Ventana de registros de la CPU (wr)*. A los típicos se ha añadido una E así se diferencia los registros normales que admiten 16 bits, de los registros que soportan 32 bits. Por ejemplo, ax se llama ahora eax. Siempre podremos referencias a ax, sabiendo que nos quedaremos con 16 bits últimos de eax.

- *Ventana de datos de la CPU (wd)*. Muestra la memoria en formato hexa, ahí se pueden apreciar los datos que maneja el programa, entre otras cosas.

- *Ventana de código (wc)*. Muestra, la dirección de memoria en la que se encuentra la instrucción, la codificación hexa de la instrucción y la instrucción en ensamblador.

- *Ventana de control*, en la que aparece arriba el nombre del programa que estamos

trazando. Aquí es donde introduciremos los comandos del SoftIce.

He aquí algunos comandos fundamentales para el Softice:

- *CTRL+D* → Conmuta de W95 al Softice y viceversa. No te asustes por el pantallazo ni por el aspecto cutre inicial, llegarás a quererlo, créeme. Si salen rayas, vuelve a leer el apartado -->Instalación

- *F4* → Estando en Softice, te permite echar un ojo al estado actual en W95 sin necesidad de conmutar.

- *F8* → Ejecuta una instrucción, las modificaciones en los registros del sistema aparecen de diferente color. Si la instrucción es una llamada a una rutina, se ejecutarán una a una todas las instrucciones de la rutina llamada.

- *F10* → Ejecuta una instrucción, las modificaciones en los registros del sistema aparecen de diferente color. Si la instrucción es una llamada a una rutina, se ejecutan de golpe todas las sentencias de la rutina llamada.

- *F11* → Ejecuta de golpe todas las instrucciones de la rutina actual y se para en la instrucción siguiente de la rutina padre que llamó a esta rutina. Esto nos permite trazar al padre, estando en una rutina hija. En cuanto la uses menudo verás que no es tan complicado como parece.

- *F12* → Ejecuta todas las sentencias hasta el primer ret (incluido)

- *bpx nombreRutina* → Salta al Softice cuando se ejecuta la rutina cuyo nombre es nombreRutina. Ejemplo bpx messageboxa saltará al Softice cuando el programa muestre una ventana de mensaje del tipo mensaggebox.

- *bpx dirInstrucción* → Salta al Softice cuando se ejecuta la instrucción que está es la dirección de memoria dirInstrucción.

- *bpr dirIni dirFin rw* → Salta al Softice cuando hay un acceso de lectura o escritura en las direcciones dirIni, dirFin ambas incluidas. Ejemplo bpr 100 109 rw que puesto de otra forma más fácil de expresar, nos queda algo como bpr 100 100+9 rw

- *s l dir tam'cad'* → Busca la cadena cad a partir de dir hasta dir+tam. Esta sentencia casi siempre tendrá este aspecto. Ejemplo s 30:00 l ffffffff 'cad' . Las comillas son importantes. 30:00 es la dirección de comienzo del segmento de datos, o sea la dirección de memoria donde están los datos del programa y ffffffff es el tamaño del segmento de datos, como veréis hay 4GB de espacio para almacenar datos.

- *d registro* → Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección guardada en registro. Ejemplo d eax muestra a lo que apunta eax.

- *d dirección* → Muestra en la ventada de datos el contenido de lo que hay a partir de la

dirección.

- *d nomRutina* → Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección donde comienza la rutina nomRutina.

- *Impr Pant* → Vuelca el contenido de la pantalla por la impresora, quizás tengas que darle varia veces hasta que el buffer de la impresora se llene.

4.ESQUEMAS DE PROTECCION

Citaré algunas de las técnicas utilizadas por los programadores para proteger su soft.

- Números de Serie.

- Cripple Software (software limitado), en diversas variantes:
  - Tiempo limitado a meses , días, minutos..
  - Número de ejecuciones o usuarios limitado.
  - Funciones deshabilitadas.

- Protecciones por discos o CD-ROM llave.

- Protecciones anti-herramientas crack
  - Antidebuggers.
  - Antidescompiladores.
  - Antidesensambladores.
  - Encriptación parcial o total.

- Ninguna de las anteriores.

5.ESQUEMAS DE PROTECCIÓN BASADOS EN NÚMEROS DE SERIE

Esta es una "antigua" técnica de protección utilizada por las toneladas de shareware que nos inundan, basta comprarse un CD por 4 perras y ver 650 MB de programas basura, en general, deseosos de exprimir nuestras carteras. Veamos como funcionan.

El programa puede ser total o parcialmente funcional, pero posee estúpidas ventanas que nos recuerdan que somos usuarios no registrados, o bien pitidos mal sonantes o mensajes perennes proclamando que les mandemos dinero. A veces, pasado cierto tiempo o pasado un número de ejecuciones, el programa deja de funcionar. Todo esto inconvenientes se resuelven llamando a la casa que construyó el software (imaginar lo que puede costar una llamadita o un Fax a un pueblo mal oliente del esto de Utah en USA), o bien mandando un e-mail. En cualquier caso hay que indicar el número de VISA donde ellos clavarán sus garras

para rellenar sus ya nutridas arcas. Una vez desplumado recibimos por e-mail o por teléfono una palabra mágica, un número de serie, una password, o lo que sea, yo lo llamaré "pwd". Esta pwd desbloquea el programa y/o elimina las estúpidas ventanas recordatorio.

Acerca de los programadores

Sólo dos cosas:

a) En general son perezosos y a veces estúpidos.
b) Nunca les creas.

La opción a) es clara, sus esquemas de protección son arcaicos, apenas han sufrido modificaciones, tan sólo incorporan trucos viejos sobre esquemas bien conocidos. Programas en lenguajes de alto nivel tipo C++, Visual Basic, estos compiladores se basan en librerías bien conocidas. El programador no tiene el amor propio de crear su propia rutina de protección en ensamblador, prefieren dejarla en manos de compiladores que crean código ensamblador ineficiente y fácilmente legible. ¿A qué se debe todo esto?, a su mentalidad comercial de la vida, no trabajan por placer, son esclavos de su trabajo, verdaderos zombies andantes. Lo importante es acabar y pronto no importa la calidad del soft o las quejas del estúpido usuario por la lentitud del programa o por los "cuelgues".

Una breve disgresión, no os habéis preguntado por que las versiones de los programas salen como churros cada 2 días. La respuesta es que se dejan a propósito las cosas sin hacer o mal hechas para que al cabo de dos días se pueda sacar una flamante nueva versión con una leve modificación la cual debes comprar para no quedarse obsoleto, Dios mío que abominación!

Respecto a b) baste decir que siempre tratan de encubrir sus errores con malolientes mentiras.

PROGRAMADORES, leer esto y aprender, pero que digo, no tenéis tiempo ni para joder, ni para ver por donde os joden....:-)

Análisis de esquemas con número de serie

En general existen dos formas en las que trabajan las rutinas de protección de número de serie:

a) Número de serie independiente del usuario.
b) Números de serie adaptados al usuario.

a) Si extraemos una pwd, ésta servirá a todos los usuarios. Una pwd válida se diferencia de una inválida (por las muletas, es un chiste) por la presencia de ciertos caracteres en posiciones fijas (p.e. el carácter 8 debe ser una 'C', el 10 un '-'). Toda pwd que cumpla las restricciones será una pwd válida. Por norma, casi todas las pass incorporan el carácter '-', 2b en hexa. A veces no se requieren caracteres fijos , sino que la suma ASCII cumpla cierta condición. Cada letra del alfabeto y cada carácter numérico tiene una cantidad asociada, su código ASCII

(p.e. par el acute0acutees el 30 en hexa o el 40 en decimal). La técnica consiste en sumar el código de cada carácter y comprobar la suma (que a veces se llama checksum) con una cierta cantidad. Una variante es modificar el valor ASCII a través de una tabla que asocia a cada carácter un número distinto.

Crackear esto es fácil de crackear:

cmp suma, sumacorrecta ---->cmp suma, sumacorrecta
jz ok ---------->jnz ok

Es posible mezclar las tres técnicas, caracteres fijos, cheksum y modificación del código ASCII. En un capítulo próximo veremos un ejemplo de esto.

b) En este caso se utiliza el nombre, los apellidos, o el nombre de la empresas o todo junto para generar un pwd. Aquí las técnicas son más imaginativas: coger cierto caracteres y repetirlos hasta llegar a un tamaño, usar el código ASCII de ciertas caracteres como índice de una tabla de encriptación ... En fin, depende de las paranoias del programador. Lo cierto es que se debe generar la pwd correcta para nuestros dato y ésta se debe comparar con la introducida. Aquí es donde podemos atacar ,en al comprobación. Realmente no hace falta crackear, basta con copiar la pwd correcta e introducirla como nuestra pwd, o bien crackear las sentencias de comprobación para que sirva cualquier pass. Lo primero es mejor ya que nos servirá para futuras versiones.

Se puede mezclar ambas técnicas, como veremos en un capítulo próximo, y generar un checksum para una cierta cadena extraída a partir de los datos del usuario y comprobar el cheksum de nuestra cadena.

¿Cómo atacar?

Antes de ir como unos desposeídos a reventar el programa, es totalmente necesario echar un vistazo, ver el posible esquema de protección y los posibles puntos de ataque. En general no hay que buscar mucho, los puntos débiles tiene un letrero rojo que dice " Hey estoy aquí".

En el caso de los esquemas basados en números de serie, la cosa está clara: esos estúpidos mensajes recordatorios son la puerta de entrada. Si somos usuarios no registrados aparecen, si nos registramos desaparecen. Por tanto debe haber algo que indique cuando deben o no aparecer, este algo es un flag, que no es más que un conmutador (como el de una bombilla). Cuando está en ON aparecen los mensajes, cuando en OFF desaparecen. En nuestro contexto, un flag no es más que una posición dentro de la memoria con un cierto valor. La memoria del ordenador es como un cartón de huevos (1 huevo = 1 bit), donde cada hueco tiene un número diferente al que se le llama dirección de memoria. En cada hueco puede haber un huevo (valor 1) o no haberlo (valor 0). Los agujeros se agrupan, 8 agujeros es un Byte, 1024 Bytes es un MegaByte o MB, 1024 MB es un GigaByte o GB, 1024 GB es un TeraByte o TB. Fácil, ¿verdad?. La memoria está compuesta de bits, estos bits se pueden interpretar de muchas formas, flags, datos, instrucciones. Por ejemplo 01010101 puede ser un flag de activación, la cadena "hola" o lo sentencia pinta la pantalla, depende de como lo consideremos. En el caso de tomarlo como

instrucciones, se habla de dirección de la instrucción en memoria, que no es más que la dirección del primer del primer bit que la compone.

El valor que podemos encontrar en un flag puede variar. Para ON podemos encontrar un 1 y para OFF un 0. Se puede usar la llamada lógica negada y tiene en ON un 0 y en OFF un 1. Todo lo que se pueda hacer con 0 y 1, se pude reconvertir cambiando los 0 por 1 y los 1 por 0. Una "mejora" de los programadores es utilizar flags distintos a 0,1, cuán inteligentes!. Recuerdo cierto esquema que utilizaba el flag DEAD en hexadecimal. Los sistemas de numeración (como el hexadecimal o hexa para abreviar) son formas diferentes de contar y de representar cantidades. En base 10, la de toa la vida, se empieza en 0 y se acaba en 9. en hexa se comienza en 0 y se acaba en F (10=a,11=b,12=c,13=d,14=e,15=f).

Veamos algo más práctico:

```
cmp ax,flag; Compara el valor de ax con el valor del flag
jz mensajes; Si son iguales muestra los mensajes.
sigue: inc dx; Continúa normalmente.
.....
mensajes: mov edx,45
.....

jmp sigue; Salta y continua normalmente.
```

Este puede ser un esquema típico. Dependiendo del valor del flag se muestran los mensajes o no.

Llegamos a la parte interesante, cada mensaje recordatorio debe tener una comprobación como la del ejemplo. Basta con analizar los mensajes recordatorio y descubrir la dirección de memoria del flag. Pero quién narices rellena el flag?. Obviamente debe haber como mínimo dos inicializaciones, una al comienzo de la ejecución del programa que pone al flag a OFF y la rutina de protección que lo debe poner a ON si la pwd es correcta. ¿Me sigues hasta ahora?.

Es fácil ahora saber donde atacar, un crack elegante sería poner la inicialización al comienzo del programa a ON en vez de OFF. Recuerda esto: "Un buen cracker debe ser ante todo elegante y sutil, nada intrusivo".

Otro punto de ataque

Hasta ahora hemos visto que analizando los estúpidos mensajes se puede conocer la dirección de memoria del flag y a partir de ahí su inicialización. Pero en los esquemas basados en números de series existe un punto de entrada más claro aún que los flags: la propia rutina de protección. Veamos un método sencillo para llegar a ella.

Si uno se va a la opción de registro e introduce un número de serie falso, aparecerá una estúpida ventana indicando que nos hemos equivocado: "Sorry your password is invalid" o algo parecido que traducido al cristiano es "Tío te ha equivocado, JAAARL". Esto no es una vía de entrada, esto es una autopista de 1GB de carriles. Basta con pensar un poco, quién

es la encargada de mandar este mensaje? ,evidentemente la propia rutina de protección, interesante verdad?. Ya sólo queda encerrar la rutina, ver como trabaja , cambiar un par de bytes (siempre de la forma más elegante posible) y listo, programa crackeado.

Comparación de ataques

¿Qué crack es mejor?, el de flags o de la rutina de protección?. Esto depende en gran medida de programa, de tus habilidades y del tipo de que dispongas. Con la rutina de protección se puede analizar en profundidad el esquema, ver como trabaja y hasta extraer tu propio número de serie, o sea el número de serie que la empresa te da si te registras, pero esto requiere tiempo y esfuerzo, obteniendo una satisfacción moral e intelectual. Además, en la próxima versión del programa est pwd posiblemente funcionará y no necesitarás crackear de nuevo. Mediante cracks al flag, se requiere un tiempo menor, pero la próxima versión habrá que crackearla de nuevo (no importa seguro que estos estúpidos programadores habrán seguido la mismo patrón de protección). Un crack a la rutina de exige un conocimiento profundo de la misma, lo que puede llevar a tu propio generador de claves (igualito o seguramente mejor que el tiene la empresa).

6.CÓMO CRACKEAR TECHFACTS 95

Objetivo: TechFacts 95.
Versión: 1.30 3/7/97
Nombre del ejecutable: Teckfct95.exe
Website: http://ourworld.compuserve.com/homepage/deansoft Tamaño del ejecutable: 1.251.840 bytes.
Tipo de protección: Por número de serie.
Dificultad: ameba.
Tiempo de crackeo: 2 minutos.
Herramientas: SoftIce 3.0 y Editor Hexadecimal.

Siguiendo las recomendaciones de +ORC empezaremos por crackear nuestras propias herramientas crack. El programa en cuestión es una pequeña joya que nos permitirá, entre otras muchas cosas, rastrear las acciones de un determinado programa, buscar cadenas de caracteres en ficheros, trabajar con dll.. Generalmente, lo utilizo para rastrear programas de instalación, obteniendo información de los ficheros creados, las entradas de registro añadidas o borradas, ...

Manos a la obra. El programa es un ejecutable que no necesita otro fichero para funcionar (cosa rara en estos días). Así pues, arranquemos el programa veamos lo que ocurre. Aparece una horrible ventana diciendo que utilicemos nuestra VISA o MASTERCARD y que soltemos los 19,99 dólares (unas 2500 pesetas) que tenemos para ir a tomar cervezas.

Echemos un vistazo al programa. Entre otras cosas, hay una opción en TOOLS/WATCH SYSTEM, que nos permite rastrear un programa. En HELP/HELP TOPICS/ORDER FORM aparece una hoja de registro en la que nos avisa de que además tenemos que paga 2 dólares para gastos de envío, ¡cómo si costará 250 pelas enviar un mail con el número de

serie!.

En HELP/ABOUT TECHFACTS 95 encontramos un botón USE REG KEY. Aquí es donde tenemos que introducir nuestro Nombre (First name), apellidos (Last name) y el número de serie correspondiente que lo recibiremos por mail si pagáramos 19,99 dólares más 2 dólares de gastos de envío. Empecemos por aquí.

Pongamos un nombre, un apellido y un número cualquiera y pulsemos el botón REGISTER. Entonces escuchamos un pitido y aparece una ventana de mensaje diciendo REGISTRATION KEY FAILED. Ahora ¡pensemos un poco!, apliquemos un poco de ZEN CRACKING.

Lo único anormal es el pitido. Si tu fueras un programador y quisieras que pitará tu "cacharro" tienes dos opciones construirte un bonito programa en ensamblador que lo haga, o bien utilizar una función de pitido presente en alguna de las vomitivas librerías de funciones, también llamadas API. ¿ Qué piensas que ha hecho nuestro "vago" programador ?. ¡Bingo! ha utilizado la función MessageBeep de la librería USER32.DLL. Este un punto de ataque muy claro, aunque existen muchos otros.

Apliquemos la técnica LIVE, es decir, utilizaremos el SoftIce. Reinicialicemos nuestro ordenador en modo Ms-Dos, lancemos el WinIce y volveremos a Windows.

Abramos el LOADER de SoftIce y en FILE/OPEN MODULE seleccionemos el fichero Tekfct95.exe. Pulsemos Load o el botón con las ruedecillas dentadas. Nos aparece una ventana de mensaje del SoftIce diciendo que no puede cargar la tabla de símbolos, pulsemos el botón SÍ y aparecemos directamente en el SoftIce con la pantalla en modo texto. En este momento nos encontramos en la primera sentencia de nuestro programa. Pulsemos bpx messagebeep con esto tomaremos el control antes de que aparezca el pitido. Con Ctrl-D volvemos a Windoce y el programa sigue ejecutándose normalmente pero con un cebo en messagebeep. Elegimos la opción de registro y escribimos cualquier cosa en nombre, apellidos y número de serie, pulsamos el botón y aparecemos de bruces en :

```
USER32!MessageBeep
014F:BFF623C1 B148 MOV CL,48 **** Aparecemos aquí.****
014F:BFF623C3 EB12 JMP BFF623D7
```

Si pulsamos en este momento F12(continuar hasta un RET) nos situaremos en:

```
014F:0047BA65 EB11 jmp 0047BA78
014F:0047BA67 6A30 push 00000030
014F:0047BA69 E822A7F8FF Call 00406190 **** Llamada a MessageBeep****
014F:0047BA6E B8BCBB4700 mov eax, 0047BBBC
014F:0047BA73 E824BEFBFF call 0043789C **** Pintamos la ventana de error ****
```

En tu ordenador las direcciones de memoria pueden ser diferentes.

¡Sintamos el código!. Estamos en mitad de las sentencias de error, lo que implica que debe

haber un salto condicional a este conjunto de sentencias de error. El salto debe ser condicional porque en caso de haber metido correctamente el número de serie habríamos obtenido algún tipo de mensaje de felicitación. Así pues, sólo debemos encontrar ese salto condicional y modificarlo.

Miremos por encima de la dirección 014F:0047BA69, nos encontramos en 014F:0047BA65 un salto incondicional jmp 0047BA78, en una ejecución normal nunca llegaríamos a 0047BA67 ya que siempre saltaríamos a 0047BA78. Por tanto, lo que debemos buscar es un salto condicional a la dirección 0047BA67. Si volvemos hacia atrás un poco con los cursores encontramos este bonito salto:

014F:0047B934 E89B73F8FF call 00402CD4
014F:0047B939 0F8528010000 jne 0047BA67 **** ¡BINGO! ****
014F:0047B93F 8D45B7 lea eax, dword ptr [ebp-49]

Hemos encontrado el salto, solamente hay que modificarlo. Fijaos que el salto se produce después de una llamada a la rutina call 00402CD4 apostaría el pellejo a que esta es una rutina para comprobar si tu número de serie es correcto. Si no es igual (jne) salta a las sentencias de error. Si es igual continua ejecutándose. Hay muchas formas de invertir el salto:

1.- Cambiar 0F8528010000 jne 0047BA67 por
0F8500000000 jne 0047B93F

2.- Cambiar 0F8528010000 jne 0047BA67 por
404840484048 inc eax,dec eax, inc eax, dec eax, inc eax, dec eax

La 1 es un salto neutro, sea igual o no siempre se ejecuta la sentencia siguiente. La 2 es la preferida por +ORC, cambia el salto por un conjunto de parejas incrementar - decrementar que dejan el registro eax sin cambios en su contenido.

Solamente hay que tener en cuenta dos cosas para modificar el código, sustituir siempre el mismo número de bytes (cambias 2 bytes por 2 bytes) y que tus modificaciones sean una sentencia en ensamblador correcta.

El SoftIce nos permite hacer cambios On-Fly, es decir, en ese mismo instante, pero el cambio no es permanente. Para ello, nos vemos obligados a utilizar algún editor hexadecimal, con el cual abriremos el fichero ejecutable, y buscaremos la cadena en hexadecimal E89B73F8FF0F8528010000 y la cambiaremos por E89B73F8FF0F8500000000. La cadena se encuentra en el offset 0X7AD34(los números en hexa llevan delante un 0X) que en decimal es 503092.

Así pues tenemos que irnos al byte 503092 de fichero ejecutable y comenzar a hacer cambios.

Ahora tendremos el ejecutable parcheado, si nos registramos nuestro número de serie siempre será aceptado.

Un crack no es más que un pequeño programa que abre un fichero y cambia un par de bytes por otros. ¡Nada más sencillo! Sólo hay que saber qué bytes hay que cambiar. Cuantos menos bytes se cambien más elegante será el crack.
Si habéis seguido todos los pasos habéis crackeado vuestro primer programa. Aun nos sois cracker pero estáis en la buena senda. Sólo hay que poner interés.

Para gentes más avezadas, comentaré que el flag de activación se iniciativa correctamente en :0047BA5E mov byte ptr [004CF31A],00 La rutina de protección es bastante patética, con gran cantidad de código inactivo. Empieza en :47B5C0. Obviamente se podría haber hecho algún otro tipo de crack, pero este es el más simple (se podría haber obtenido el número de serie real, o haber creado un generador de claves).El programador ha puesta a "pelo" la dirección de retorno en :47BA3F push 47BA54. Es un ridículo truco que nos hará perdernos si continuamos ejecutando normalmente, por ello es conveniente pulsar "F12" y mirar hacia por encima sin ejecutar sentencias.

DESCENSO A LOS INFIERNOS

Veamos de una vez por todas cómo se ejecuta una sentencia en el procesador, desde el inicio hasta el final.

Supongamos que estamos programando en un lenguaje de Alto Nivel (C, C++, Pascal, Delphi, Visual Basic). Se llaman de Alto Nivel para diferenciarlos de los lenguajes más próximos al procesador, como el Ensamblador, a los que se llama lenguajes de Bajo Nivel. Cuanto más "Alto" programemos, más control perderemos sobre nuestro programa, y esto es un grave problema.

Supongamos un programa, escrito en Alto Nivel, que pinta la frase "HOLA MUNDO" en pantalla. ¿Qué pasos se siguen hasta que realmente se pinta la frase?. Nuestro programa debe de residir en un fichero, al que se denomina fichero fuente, en el que aparece la sentencia para pintar la frase. Este fichero no es entendible por el procesador, sólo es un conjunto de caracteres, mu diferente del conjunto de 0 y 1 que necesita para trabajar. Es aquí donde entra el compilador, transforma el fichero fuente en un fichero intermedio, también llamado fichero objeto. En esta transformación se comprueba la sintaxis de las sentencias (falta el punto y coma) y la semántica (has pasado un entero cuando se esperaba un real). El compilador realiza entonces una fase de linkado para reunir los distintos ficheros objeto que conforman nuestro programa final (aunque tengamos un único fichero fuente). En esta fase se determinan el mapeo final del programa en memoria (que dirección de memoria va a tener cada instrucción del conjunto de ficheros objeto). Tras la fase de linkado, el programa final se encuentra en un lenguaje llamado pseudocódigo, mu sencillote. Aquí se pueden tomar 3 vías.

Primera: Dejar el programa como está, y que otros programas o librerías (como la vbrun500.dll de Visual Basic) lo traduzcan (lo interpreten) a sentencias entendibles por el procesador.

Segunda: Transformar el pseudocódigo a un lenguaje de Bajo Nivel como el ensamblador. En tal caso, se necesitará un compilador de ensamblador para que el programa pueda ser ejecutado. OJO, el ensamblador no es entendible por el tonto procesador que sólo ve unos y ceros, son dos cosas distintas.

Tercera: La más común, transformar directamente de pseudocódigo a ejecutable.

Un fichero ejecutable consta de unos y ceros (o de números en hexa, según se mire) ordenados de una forma especial; ordenados en instrucciones: Los 3 primeros números son el tipo de instrucción, los 4 siguientes el operando1, el siguiente el operador...Cada instrucción es despiezada dentro del procesador y dan a lugar a la ejecución de un conjunto de programas presenten dentro del procesador, son las microrutinas. Estas microrutinas son las encargadas de bloquear buses, activar multiplexores, dar tensión a un transistor o no, pa enterndernos. Accionando correctamente buses, multiplexires... se pintará realmete la frase en pantalla. Bien, esto es todo.

Mu bien y esto, ¿pa que me sirve?
Existe una correspondencia directa entre lenguaje ensamblador y programa ejecutable. Gracias a un desensamblador (W32DASM, IDA PRO), a partir de un ejecutable podemos obtener el programa el lenguaje ensamblador sin disponer del fichero fuente. Un programa en ensamblador puede ser fácilmente entendido por los humanos (o eso dicen algunos). Esto nos da un poder tremendo a los crackers. Podemos saber cómo funciona un programa sin necesitar del programa original. Y lo que es más aún, independientemente del lenguaje de Alto Nivel. Todos los lenguajes tienen que pasar a ejecutable de alguna u otra forma, y es aquí cuando usamos nuestro desensamblador y extraer su listado en ensamblador. Da igual que programa esté hecho en Pascal, O C++, lo entenderemos igualmente ya que leeremos ensamblador.

3.EL LISTADO MUERTO

La idea es sencilla. Cogemos nuestro desensamblador favorito y se lo pasamos al objetivo. Obtendremos un listado en ensamblador de nuestro programa a crackear. La técnica crack se llama Listado muerto porque entenderemos y manejaremos el programa con este listado, sin tener que ejecutarlo, con el programa muerto. A diferencia de cuando lanzamos el SoftIce y entendemos el programa cuando se está ejecutándose, cuando "vive".

Hay tres ventajas fundamentales para utilizar el Listado Muerto:

- Podemos seguir el programa fácilmente de atrás hacia adelante, basta con pasar de página, no hace falta volver a ejecutarlo.

- Es mucho más relajado imprimir y estudiar 4 páginas de código que rastrear con el SoftIce. Este es uno de los consejos de +ORC.

- Se descubren pequeños secretos, como rutinas inactivas.

La paciencia y la tranquilidad son dos requisitos fundamentales en un cracker. Es fácil perderte trazando con el SoftIce e imposible con el Listado Muerto.

Manos a la Obra
Una vez desensamblado el objetivo, la idea es buscar cadenas de texto interesantes, como "unregistered", "expired", "congratulations" y mirar alrededor de estas cadenas buscando un salto mágico. Las palabras en concreto dependen del programa y son las que aparecen para recordarte que aún no te has registrado.

4.CÓMO CRACKEAR UEDIT 5.0

Objetivo: Uedit 5.0.
Versión: 5.0 3/7/99
Nombre del ejecutable: uedit32.exe
Website: http://www.uedit.com
Tamaño del ejecutable: 812.514 bytes.
Tipo de protección: Por número de serie y temporal.
Dificultad: Medio.
Tiempo de crackeo: 5 minutos.
Herramientas: W32dasm8.X, SoftIce.

Siguiendo la recomendación del maestro +ORC, continuamos con el crack a nuestras herramientas de trabajo. En este caso nos encontramos ante un excelente editor hexadecimal, vital para nuestros negocios :-)

Instalemos el programa, ejecutémoslo y veamos lo que nos encontramos. ARRJJ!!, una horrible ventana nos dice que tenemos 45 días para registrarnos. Además tiene un bonito botón "Enter Authorization Code". Pulsemos y veamos. Un típico nombre de usuario y número de serie (al que le llamaré password o pass). Si pulsamos cualquier guarrada en ambos, sorpresa, ningún mensaje advirtiendo del error, ningún pitido (recordáis el capítulo I), nada excepto una ventana de mensaje que dice que hace falta cerrar el programa para validar el código. ¿Habrá leído Ian D. Mead las lecciones de Estado+Porcino?. Bien, ¿por dónde atacar?. No tenemos nada que nos indique que nos hemos equivocado. ¿Qué tal si usamos el Listado Muerto amiguitos?

Una vez desensamblado el programa y dentro del W32dasm pulsemos el botón de Strn Ref (el botón que está al lado del botón de impresora) para ver las cadenas de caracteres que aparecen en el nuestro objetivo. Qué vemos, qué casualidad , tenemos la frase "Thank you fot supporting Shareware" , hagamos doble click y veamos donde aparecemos:

* Referenced by a (U)nconditional or (C)onditional Jump at Address: |:00401B77(C) |
:00401B7D 83FB09 cmp ebx, 00000009
:00401B80 7504 jne 00401B86

:00401B82 C645EC20 mov [ebp-14], 20 * Referenced by a (U)nconditional or (C)onditional Jump at
:00401B86 8D45C8 lea eax, dword ptr [ebp-38]
* Possible Reference to String Resource ID=00010: " Thank you for supporting Shareware." |
:00401B89 6A0A push 0000000A

En :00401B89 tenemos una referencia a la cadena que nos interesa. Cada frase del programa tiene asociado un número, en este caso es el 0000000A y este número se les pasa a las rutinas que tienen que imprimir los mensajes. La forma tradicional de pasarle parámetros a una rutina es a través de la pila mediante push (como en :00401B89). Los parámetros se pasan empezando por el último, es lo que se llama paso de parámetros mediante el modelo de PASCAL, existen otros modelos, pero son poco utilizados.

¿Estamos en el camino adecuado?, nop, ya que el número de nuestra frase, el 0000000A (es el número 10 en decimal) es muy utilizado en cualquier programa y cada vez que aparezca, el desensamblador pensará que se está haciendo referencia a nuestra frase. Bien pensemos un poco.
Nuestro programa está limitado por 45 días de uso, pasado ese tiempo, lo normal es que nos aparezca una frase diciendo algo así como "Evaluation time expired". Busquemos (con el botón de la linterna) la palabra expired pasada a una rutina mediante push a ver que encontramos.

:043F7D3 E8375DFCFF call 0040550F
:0043F7D8 391DDC0C4A00 cmp dword ptr [004A0CDC], ebx
:0043F7DE 758A jne 0043F76A
:0043F7E0 8D4D10 lea ecx, dword ptr [ebp+10]
:0043F7E3 E829C00100 call 0045B811
:0043F7E8 8D4D14 lea ecx, dword ptr [ebp+14]
:0043F7EB C645FC01 mov [ebp-04], 01
:0043F7EF E81DC00100 call 0045B811
* Possible Reference to String Resource ID=00068: "UltraEdit 45 Day Evaluation time expired!!!!" |
:0043F7F4 6A44 push 00000044

BINGO. ¿No sentís el código?, ¿no percibís como estamos en el camino correcto?. Si, tenemos en :0043F7F4 un push con el número asociado a la frase que buscamos y justo en :0043F7DE un salto a 0043F76A que evita imprimir el mensaje, pero el punto clave es ver la comprobación del salto en :0043F7D8. Se comprueba si el contenido de EBX es igual a una dirección fija de memoria la [004A0CDC]. Las direcciones fijas son las típicas variables globales tan mal utilizadas en los programas. Tenemos una variable global que controla la aparición de un mensaje de error. En algún punto del programa debe de inicializarse [004A0CDC] con un valor; si localizamos este trozo de código, estaremos en plena rutina de comprobación. ¿Fácil, verdad?

Busquemos [004A0CDC] y veamos quién la inicializa, sólo nos interesan las sentencias que inicializen la variable, no las sentencias que comprueban su valor. Normalmente se inicializa por defecto a un valor de error (indicando que no estamos registrados) y se inicializa

correctamente cuando nos registremos. Conforme aparecen ocurrencias de nuestra variable global sabemos que estamos en el buen camino porque siempre está rodeada de mensajes de error o de felicitación.

Buscando [004A0CDC] encontramos las siguientes sentencias que modifican la variable (el resto de apariciones son comprobaciones del valor)

```
:00405541 893DDC0C4A00 mov dword ptr [004A0CDC], edi
:004056A3 891DDC0C4A00 mov dword ptr [004A0CDC], ebx
:004057D4 8325DC0C4A0000 and dword ptr [004A0CDC], 00000000
:00426924 891DDC0C4A00 mov dword ptr [004A0CDC], ebx
:0043F684 C705DC0C4A0001000000 mov dword ptr [004A0CDC], 00000001
```

¿Qué tenemos aquí?. Parece lógico pensar que en :004057D4 tenemos la inicialización por defecto, ya que un AND con ceros da cero. La sentencia contraria la tenemos en :0043F684 que mueve 1 a la variable, esto sin duda indica que nos hemos registrado. También podría ser al revés, cero registrado, uno no registrado, pero este no es el caso. Basta ejecutar el Softice y poner bpr 004A0CDC 004A0CDC rw , la primera modificación debe ser la asignación por defecto, en este caso la :004057D4. Por tanto solo debemos centrarnos en la asignación a uno :0043F684, olvidando el resto de asignaciones. Esto es un axioma fundamental crack, ante la duda, elige siempre la solución más sencilla. Bien, veamos que hay entorno a la :0043F684

```
:0043F65C E89A560300 call 00474CFB
:0043F661 8B7804 mov edi, dword ptr [eax+04]
:0043F664 8B4514 mov eax, dword ptr [ebp+14]
:0043F667 48 dec eax
:0043F668 7478 je 0043F6E2
:0043F66A 48 dec eax
:0043F66B 0F85F9000000 jne 0043F76A
:0043F671 391DDC0C4A00 cmp dword ptr [004A0CDC], ebx
:0043F677 0F85DA010000 jne 0043F857
:0043F67D 833DBC024A0000 cmp dword ptr [004A02BC], 00000000
:0043F684 C705DC0C4A0001000000 mov dword ptr [004A0CDC], 00000001
```

Tenemos diversos saltos que evitan nuestra asignación a uno. El primer salto, está en :0043F668 7478 je 0043F6E2 que tal si lo cambiamos por EB1A jmp 43F684. Osea, siempre saltamos a 43F684 evitando las comprobaciones de :0043F66B y :0043F677.El código EB es la instrucción de salto incondicional JMP, 1A es el número de bytes desde la sentencia condicional hasta la sentencia donde queremos saltar. Fácil, ¿verdad?.

Perfecto, rula. Basta con buscar en el ejecutable uedit32 la secuencia 8B451448747848 y cambiarla por 8B451448EB1A48. Pero hay un peque problema, el crack funciona pero no tenemos un número de serie correcto. En principio basta, pero pensando un poco podremos sacar nuestro propio número de Serie. ¿Qué se os ocurre?

Sip, exactamente, ¿qué tal si le seguimos la pista a nuestro número de serie basura y vemos con quién se comparará?. La pregunta es, donde coloco un bpx para pararme justo antes de que se compruebe mi número de serie. La respuesta es sencilla, en :43F618 (echarle un vistazo al listado muerto) comienza la rutina en la que se asigna a 1 nuestra variable global. Este puede ser un buen comienzo. Abrimos el Softice con el uedit, ponemos nuestro nombre Estado+Porcino y un número basura 1212121212121212. Cerramos el uedit y lanzamos de nuevo el SoftIce poniendo la sentencia bpx 43F618. Aparecemos en :43F618, ahora es el momento de buscan nuestro número de serie con s 30:00 l ffffffff ´12121212´ lo encontramos es :942F9C (esta dirección puede cambiar en tu ordenador). Borramos el punto de ruptura anterior con bc 0 y creamos uno nuevo con la dirección donde está nuestra password con bpr 942F9C 942F9C+f rw seguimos adelante con Crtl+D para ver quien caen en vuestra trampa. Aparecemos en :40B73A con varios movsd , nuestra password se está copiando en otro sitio.

La sentencia movsd, copia caracteres de ees:esi a ees:edi. Pongamos en el SoftIce d ees:edi para ver como realmente se va a copiar, además pongamos otro punto de ruptura en la nueva posición de nuestra password con bpr ees:edi ees:edi+f rw .Curiosamente, si nos movemos un poco con los cursores por ees:edi aparecen las passwords correctas, pero todavía no es el momento. Lancemos de nuevo el SoftIce con Crtl+D y aparecemos en :444FOE, aquí encontramos una pequeña comprobación, en ecx tenemos nuestra pass (para comprobarlo basta con poner d ecx) y en edx apuntamos a una zona de nuestro nombre, concretamente a "tado+Porcino". Esto no es exactamente lo que buscamos, así que sigamos adelante con Crtl+D y aparecemos en :444EBO con una comprobación entre edx y ecx a través de al. Curiosamente edx apunta a nuestra pass y ecx apunta a Y2+cHdcBd6=DBC/P este churro es la pass correcta, si seguimos con Crtl+D aparecemos en el mismo sitio con edx apuntando a nuestra pass y ecx apunta a JWKTUUTH02166710 otra pass correcta. A lo largo de la evolución del programa desde sus primeras versiones, se ha cambiado 2 veces de generador de pass por cuestiones de seguridad, ¡qué estúpidos!. Por eso la doble comprobación, ver si nuestra pass es del antiguo generador o del nuevo.

Eso es todo por ahora, no seáis unos descerebrados y utilicéis mi pass, buscad la vuestra, es mu sencillote.

Espero vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com

En breve analizaremos tipos de protecciones mucho más interesantes.

Recordar bebed de la fuente, buscad a +ORC en la red.