The Italian CRaCKiNG Encyclopedia vol. 1

CRaCKiNG With WDAsm

Scritto da

---==: ALoR :==---
---==> Proud member of NEURO ZONE 2 <==---

Aprile 1998

Target = **, ***

(*=Lamer, **=Novizio, ***=Apprendista, ****=Esperto, *****=CrackMaster)

==========

Table of context :

==========

1.0 DISCLAIMER

Every reference to facts, things or persons (virtual, real or esoteric) are purely casual and involuntary. Any trademark nominated here is registered or copyright of their respective owners. The author of this manual does not assume any responsability on the content or the use of informations retriven from here; he makes no guarantee of correctness, accuracy, reliability, safety or performance. This manual is provided "AS IS" without warranty of any kind. You alone are fully responsible for determining if this page is safe for use in your environment and for everything you are doing with it!

E' in inglese lo so... ma fa più figo... se l'hai capito, bene!
Altrimenti, aggiornati!! Siamo nel 2000 !!!
Ok. e dopo questa menata totalmente inutile, ma che serve a me per pararmi il culo, possiamo iniziare con il manuale vero e proprio.

2.0 I TOOLS NECESSARI

Dunque... Incominciamo col dire che la materia prima del cracking è senza dubbio il programma che devi crackare... Uno shareware con nag screen (poi vi spiego), una trial scaduta, ecc. insomma, tutto ciò che non funziona come la versione commerciale del programma è da considerarsi MATERIA PRIMA.
Ovviamente non basta avere la materia prima, occorrono anche altri programmi che aiutano l'aspirante cracker a compiere il suo losco "lavoretto".
Innanzi tutto è necessario un debugger (SoftIce o WDasm). Il debugger è un programma che analizza il programma in esecuzione e fornisce all'utente la possibilità di vedere, a livello assembly, cosa succede in ogni istante nei registri del computer. Inoltre il debugger permette di eseguire una singola istruzione assembly e fermarsi per annotare i cambiamenti. Oltre al debugger

può essere utile avere un disassemblatore. WDasm 8.9 funge sia da debugger che da disassembler, quindi io ritengo che sia il programma migliore per i nostri scopi.
Il debugger opera solo in memoria, quindi una volta trovato il punto da modificare dovrete renderlo effettivo nel file del programma. Per fare ciò vi occorre un buon editor esadecimale. Io vi consiglio UltraEdit 5.1.
Se non siete molto ferrati nella programmazione in assembly vi consiglio di avere sempre sotto mano un manuale di istruzioni assembly, e siccome l'ambiente di lavoro è Windows 98/NT un elenco completo delle API di Windows.

Ricapitolando vi servono:

* Un piccì (of course..)
* Un programma da crackare :-) (come sono simpatico.....)
* Un disassemblatore (Es.: WDasm 8.9 or higher)
* Un editor esadecimale (Es.: UltraEdit)
* Un manuale di Assembly
* Un manuale delle API.
* Una buona dose di culo :-)

3.0 IL TIPO DI PROTEZIONE

Esistono vari tipi di "scocciature" associate ai programmi shareware o trial. Può esserci una scadenza del software dopo un certo lasso di tempo, alcune funzioni sono disabilitate nella versione shareware, il software è completo ma, entrando o uscendo dal programma, si presente una fastidiosissima finestra di dialogo (nag screen) che ci informa che il programma è shareware e può essere usato ancora per un certo numero di giorni o di utilizzi.
Ovviamente il modo di procedere varia a seconda della situazione in cui ci si trova. Vediamo dunque come procedere nelle situazioni più comuni.

3.1 NAG SCREEN

Se il programma che avete deciso di crackare ha un semplice nag screen,bisogna innanzi tutto trovare il jump o la call che richiama la schermata del nag. Trovata la call o il jump, abbiamo il suo indirizzo di memoria. Basterà non fare eseguire questa istruzione (vedi 5.0 e 6.0),e il programma non presenterà il nag screen. Togliere un nag screen non è difficile, basta avere pazienza (trovare la call può richiedere anche 10 o più breakpoint) ed anche un pizzico di fortuna (la call può trovarsi dove meno te lo aspetti...).

3.1.1 NAG SCREEN CON NUMERO DI REGISTRAZIONE

Numerosi programmi danno la possibilità di annullare il Nag Screen inserendo un codice di registrazione del prodotto. In questo caso non sarà necessario disabilitare il nag screen (ci pensa già lui... :-)), basterà inserire un codice di registrazione. Non preoccupatevi non è necessario conoscere quello esatto... basterà inserirne uno a casaccio... :-) Vediamo come.
Quando noi inseriamo un codice nel programma, questo controlla se il codice è corretto, e in caso contrario non ci registra. Se riusciamo a trovare il punto in cui è svolto il controllo, sarà un gioco da ragazzi modificarlo a nostro piacimento (vedi 5.0 e 6.0). Una volta modificato, si inserirà un codice qualunque, e il programma ci registrerà. ATTENZIONE ! Se per caso vi capita di inserire il vero codice di registrazione, il programma lo rifiuterà. Poiché noi abbiamo modificato il programma in modo che si registri SOLO se il codice è errato.

3.2 FUNZIONI DISABILITATE

Per prima cosa, è necessario scoprire se le funzioni sono disabilitate, oppure se non sono nemmeno implementate nel programma. Nel secondo caso, c'è ben poco da fare, a meno che non ve le creiate dal nulla, rimarranno disabilitate.
Se invece, siamo nel primo caso, bisogna trovare la procedura che decide se abilitare o meno le funzioni. Di solito queste funzioni leggono dei flag scritti da qualche parte nel registro di configurazione o nei propri file, li confrontano e decidono se abilitare le funzioni. I flag, ovviamente, corrispondono al programma shareware o commerciale. Trovando la procedura di controllo, e modificando i flag letti (vedi 5.0), si avranno a disposizione tutte le funzioni desiderate.

3.3 TRIAL A TEMPO

I programmi denominati "Try&Buy" sono programmi completi che dopo un certo lasso di tempo smettono di funzionare. Alcuni dopo un certo numero di giorni dall'installazione, altri dopo una certa data. I primi, scrivono in qualche zona dell'Hard Disk un valore che aumenta di giorno in giorno, e che quando supera un certo valore, blocca il programma. Potete usare dei monitor sull' Interrupt 13h per scoprire in che file lo scrive, oppure dei registry tracker per vedere in che punto del registro di configurazione di Windows lo scrive, e poi modificarlo a mano. Ma secondo me è più conveniente e meno noioso, trovare la funzione di controllo e modificarla o jumparla. I secondi, sono ancora più semplici, mi direte voi, basta tirare indietro la data di sistema e il gioco è fatto. Sì, ma francamente a me scoccia avere l'orologio del piccì indietro di un anno o due. Basterà anche in questo caso trovare la funzione a modificarla in modo opportuno (vedi 5.0 e 6.0).

4.0 TROVARE IL PUNTO DA CRACKARE

La prima cosa da fare, in qualsiasi situazione ci si trovi, è disassemblare il programma (vedi appendice -A-) con il nostro buon WDASM.
Dopodiché bisogna individuare il punto da modificare. Di solito io procedo in questo ordine:

*** 1) Cerco nel [String Data Ref] se è possibile trovare una stringa che ci può interessare. Ad esempio se sul nag screen c'è scritto: "Unregistered version. 35 days left." noi cercheremo "Unregistered", "version", "days" e "left" o qualsiasi altra cosa che possa comparire nel nag screen. O se nel form di registrazione compare la scritta: "Immettere il codice" ovviamente cercheremo "immettere" e "codice". Se siete fortunati e il programmatore non è stato abbastanza scaltro, troverete una di queste stringhe. Posizionatevi nel punto di codice a cui si riferisce, e saprete dove operare. Scorrete in su e in giù il disassemblato del programma analizzando attentamente i jump che si trovano nelle vicinanze.

Es.

La stringa :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040E4AD(U)
|
:0040F20D ED                    in ax, dx
:0040F20E ED                    in ax, dx
```

ci suggerisce che all'indirizzo 0040E4AD ci sarà un jump verso 0040F20D. Ed in particolare sarà un "jmp 0040F20D" poiché è in jump (U)nconditional. Ed in questo caso non ci interessa più di tanto perché il programma salta in OGNI caso in questo punto. Comunque non tralasciare MAI informazioni che sembrano inutili, magari appena sopra il "jmp 0040F20D" si trova un jump (C)onditional.

Al contrario, la stringa:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040E461(C)
|
:0040E423 50                    push eax
:0040E424 50                    push eax
```

ci indica che all'indirizzo 0040E461 c'è un jump, questa volta condizionato, verso 0040E423. Di solito si troveranno dei jump di questo tipo "je", "jne", "jle", "jnle" e sono proprio questi i jump che dovremo seguire per arrivare alle righe di codice che svolgono il paragone tra il valore corretto e quello errato (vedremo nel 6.0 come modificarli).

*** 2) Se la ricerca delle stringhe non ha avuto successo, dobbiamo iniziare a debuggare il programma per trovare la call del nag screen o il jump della nostra funzione di controllo.
Carichiamo il processo in memoria. E iniziamo a debuggare.
Ovviamente ci sono due modi di procedere per due tipi differenti di protezioni

- A) NAG Screen.
[Auto Step Over] fino a che non compare il nostro NAG. A questo punto ci annotiamo l'indirizzo di memoria su cui si è fermato il debugger, e terminiamo

il processo. Mettiamo un breakpoint sull'indirizzo di memoria annotato, e ripetiamo l'[Auto Step Over]. Questa volta, prima di eseguire l'istruzione al nostro indirizzo di memoria (probabilmente una CALL) il debugger si fermerà, aspettando nostre istruzioni. [Step Into], ed entriamo nella CALL. [Auto Step Over], fino a che non compare il NAG. Ripetere la sequenza fino a che non si trova la call che richiama SOLO il NAG Screen.
Questo accade perché le call sono incapsulate una nell'altra, e quella del Nag può essere richiamata anche dopo che la finestra del programma è già stata tutta disegnata. Se modificate una call che contiene parti di codice non solo del Nag, potrete ritrovarvi con un programma disegnato a metà o,peggio ancora, un GPF (General Protection Fault).

- B) Codice di registrazione.
La strada da seguire è molto simile a quella per trovare il NAG, però, questa volta, lo scopo non è trovare una call, bensì un jump. Ed in particolare il jump che segue le righe di controllo del codice di registrazione.
Quindi, [Auto Step Over] fino a che il programma non ha finito di caricarsi, selezionate "immettere il codice" o qualcosa di simile, insomma fate finta di volervi registrare.... A questo punto [Auto Step Into] e fate attenzione a quello che succede e dove punta il debugger. A questo punto dovreste essere nei paraggi della funzione di controllo. Immettete un codice a caso, e seguite passo passo quello che succede, [Step Into] ripetutamente.
Quando compare il messaggio che vi avvisa di aver sbagliato il codice, fermatevi e risalite al jump che vi ha portati lì. Modificando questo jump, il programma registrerà tutti i codici errati, ma non quello esatto (vedi 6.0).

*** 3) Se anche il debug non è andato a buon fine, non vi demoralizzate. Date sfogo alla vostra fantasia. Andate a cercare tutto ciò che riguarda il prg. da crackare, e tentate di capire cosa c'è sotto. A volte un buon Registry Tracker (come quello della symantec) può aiutarvi a capire se il nostro prog. legge o scrive nel registro di configurazione di Windows, e soprattutto su quali chiavi agisce. Oppure un monitor dell'INT 13 aiuta a capire quali file sono modificati o controllati dal programma durante l'esecuzione.
Dopo aver raccolto più informazioni possibili, ricontrollate il disassemblato e cercate di scoprire il punto giusto. Magari ripetete la ricerca delle stringhe, questa volta con il nome della chiave del registro che viene modificata. Insomma ricercate e ricercate e se avrete una buona dose di culo (ingrediente fondamentale nel cracking) riuscirete senz'altro a trovare il punto giusto. Good Luck :-)

5.0 MODIFICARE IL CODICE

Una volta trovato il punto cruciale, bisogna modificarlo a dovere.
Prima di spiegare come modificare il codice farò una rapida carrellata delle istruzioni assembly che modificheremo, giusto per sapere che cosa stiamo facendo ed avere un minimo di basi per interpretare le righe che circondano la nostra.
All'inizio del manuale si parlava di call, jump e Compare....
Bene, eccole qua!

 ** - CMP - **

```
Utilizzo :  cmp registro1, registro2
            cmp memoria, registro
            cmp registro, memoria

Es.:        cmp eax, ebx
```

Confronta due registri e assegna il valore 1 al flag ZF (Zero Flag) se i registri sono uguali, altrimenti lascia 0.

 ** - Jump - **

```
Utilizzo:   Jxx indirizzo

Es.:        Jxx 004056DE
```

Esistono molti tipi di Jump e, a seconda del tipo le "xx" di Jxx cambieranno.

Opcode	Mnemonic	Meaning	Jump Condition
77	JA	Jump if Above	CF=0 and ZF=0
73	JAE	Jump if Above or Equal	CF=0

72	JB	Jump if Below	CF=1
76	JBE	Jump if Below or Equal	CF=1 or ZF=1
72	JC	Jump if Carry	CF=1
E3	JCXZ	Jump if CX Zero	CX=0
74	JE	Jump if Equal	ZF=1
7F	JG	Jump if Greater (signed)	ZF=0 and SF=OF
7D	JGE	Jump if Greater or Equal (signed)	SF=OF
7C	JL	Jump if Less (signed)	SF != OF
7E	JLE	Jump if Less or Equal (signed)	ZF=1 or SF!=OF
	JMP	Unconditional Jump	unconditional
76	JNA	Jump if Not Above	CF=1 or ZF=1
72	JNAE	Jump if Not Above or Equal	CF=1
73	JNB	Jump if Not Below	CF=0
77	JNBE	Jump if Not Below or Equal	CF=0 and ZF=0
73	JNC	Jump if Not Carry	CF=0
75	JNE	Jump if Not Equal	ZF=0
7E	JNG	Jump if Not Greater (signed)	ZF=1 or SF!=OF
7C	JNGE	Jump if Not Greater or Equal (signed)	SF != OF
7D	JNL	Jump if Not Less (signed)	SF=OF
7F	JNLE	Jump if Not Less or Equal (signed)	ZF=0 and SF=OF
71	JNO	Jump if Not Overflow (signed)	OF=0
7B	JNP	Jump if No Parity	PF=0
79	JNS	Jump if Not Signed (signed)	SF=0
75	JNZ	Jump if Not Zero	ZF=0
70	JO	Jump if Overflow (signed)	OF=1
7A	JP	Jump if Parity	PF=1
7A	JPE	Jump if Parity Even	PF=1
7B	JPO	Jump if Parity Odd	PF=0
78	JS	Jump if Signed (signed)	SF=1
74	JZ	Jump if Zero	ZF=1

Ai fini crackistici a noi interessano principalmente Je e Jne.

** - CALL - **

Utilizzo: CALL destinazione

Es.:   CALL 004DF856   ---> esegue la sub 004DF856

           subroutine:

           004DF856 ...
           004DF857 ...
             ....
           004DFA21 RET   ---> ritorna all'istruzione successiva a CALL

Esegue una subroutine alla destinazione "004DF856" e ritorna quando incontra
RET.

** - NOP - **

Utilizzo: NOP

Es.:   NOP

Non esegue alcuna istruzione. (e allora a che serve??? :-)
Ai fini crackistici serve per non fare eseguire le CALL (vedremo in seguito).

** - MOV - **

Utilizzo: MOV destinazione, sorgente

Es.:   MOV eax, ebx

Copia il valore di [sorgente] in [destinazione].

Per le altre istruzioni assembly e per i vari registri della CPU vi rimando
ad un buon manuale di assembly.
Vediamo ora come modificare queste istruzioni in modo che facciano quello che
desideriamo. Nei nostri due casi le CALL andranno NOPpate, e i Jump invertiti.

CALL
Una call richiamerà la subroutine che visualizza il nag screen. Quindi noi
dobbiamo fare in modo di non fargliela richiamare. In nostro aiuto viene la

NOP. Sostituendo la CALL con delle nop avremo ottenuto il nostro risultato.
ATTENZIONE!! una CALL di solito occupa 8-12 byte mentre una NOP solo 2.
Bisognerà fare in modo di coprire TUTTI i byte della call con delle NOP.
Quindi se la call è di 8 byte, metteremo 4 nop; se è di 12, 6 nop ecc..
Così facendo la call non sarà eseguita, la suo posto saranno eseguite n NOP.
(vedi appendice -A- per capire come patchare il codice).

Es.:

Le righe :

```
      :00401026 50                      push eax
      :00401027 E8541E0000              call 00402E80    -----------
      :0040102C 8B06                    mov eax, dword ptr [esi]    |
                                                                      |
per far saltare la call devono diventare :                            |
                                                                      |
      :00401026 50                      push eax                    |
      :00401027 90                      nop   -------\              |
      :00401028 90                      nop   --------|             |
      :00401029 90                      nop   --------|-------------
      :0040102A 90                      nop   --------|
      :0040102B 90                      nop   -------/
      :0040102C 8B06                    mov eax, dword ptr [esi]
```

OSSERVAZIONE PER I PIU' ESPERTI : Alcuni programmi "sniffano" le NOP multiple
e non funzionano correttamente se si accorgono della manomissione. Invece di
due NOP consecutive si può incrementare e decrementare di uno un registro a
caso.

Es.:

```
      :00401027 90                    nop   -------\
      :00401028 90                    nop   --------|
                                                    |
diventa :                                           |
                                                    |
      :00401027 40                    inc eax   ----|
      :00401028 48                    dec eax   ---/
```

Jxx
I jump che interessano a noi, di solito sono preceduti da una CMP o da un TEST
Queste due istruzioni effettuano un paragone tra registri e se sono uguali
settano il valore dello ZF (Zero Flag) a 1, altrimenti 0.
Quindi se il confronto è seguito da un Je (Jump if Equal ZF=1) basterà
sostituirlo con un Jne (Jump if Not Equal ZF=0) e il programma salterà quando
il confronto è errato. Quindi accetterà tutti i codici di registrazione tranne
quello corretto... :-)

Es.:

Le righe:

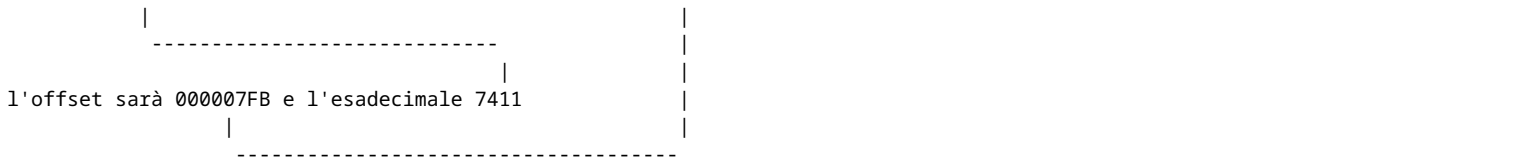
```
      :0040102E 83C404                  add esp, 00000004
      :00401031 85C0                    test eax, eax
      :00401033 740F                    je 00401044   --------
      :00401035 50                      push eax              |
                                                              |
devono diventare :                                            |
                                                              |
      :0040102E 83C404                  add esp, 00000004     |
      :00401031 85C0                    test eax, eax         |
      :00401033 750F                    jne 00401044  --------
      :00401035 50                      push eax
```

In alcuni casi, torna utile modificare le istruzioni MOV. Ammettiamo di aver
trovato il punto in cui il programma scrive nel registro di configurazione
quante volte è stato usato il programma, oppure quanti giorni sono passati,
basterà modificare a dovere la MOV ed il programma scriverà sempre lo stesso
numero, non incrementandolo mai.... :-)

6.0 RENDRERE EFFETTIVE LE MODIFICHE

Una volta trovato e patchato il punto giusto, bisogna modificare fisicamente
il file eseguibile o la Dll. Questa procedura è necessaria poiché WDASM patcha
il codice SOLO in memoria e lascia intatto il file d'origine.
Per rendere effettive le modifiche ci serve un editor esadecimale per editare
il file eseguibile. Prima di fare ciò bisogna prendere nota di due cose :
l'offset e il corrispondente esadecimale dell'istruzione assembly.

Es.:



Ora carichiamo il file eseguibile nell'editor e ci posizioniamo sull'offset
desiderato e modifichiamo ciò che c'è scritto con il nostro nuovo codice (in
questo caso 7411).
Se poi volete creare un "crack.exe" che svolga questo lavoro in modo che si
possa distribuire ai comuni mortali (lamer) che non sanno nemmeno cos'è un
offset piuttosto che un editor esadecimale, siete liberi di farlo. Bastano
poche righe in Pascal oppure in C ed il gioco è fatto. Nell'appendice -C- si
trova il sorgente di un patcher generico (il GPP by ALoR) scritto in C.

 Appendice -A- USARE WDASM 8.9

Le funzioni di Wdasm 8.9 sono innumerevoli. Se avete l'intenzione di
scoprirle tutte leggetevi il manuale in linea che è molto ben fatto e aiuta a
capire bene il funzionamento del programma. Ora io riporterò e spiegherò solo
le principali funzioni (quelle che più useremo).
Procederò in ordine cronologico, seguendo più o meno la procedura da seguire.
Per evitare di ripetere spesso alcune parole indicherò con [ ] i bottoni e
con ^ ^ i menu.
Quando diventerete pratici, non userete più i menu ma i tasti che la toolbar
ci offre, oppure i shortcut.

Disassemblare : ^Disassembler^ Open file to disassemble. A seconda del
                  computer che possedete e della lunghezza del file questa
                  operazione richiederà più o meno tempo (a volte anche
                  10-15 min. per un file da 3 Mb su un P133).

Salvare il listato ^Disassembler^ Save Disassembly Text File

Ricerca stringhe : ^Refs^ String Data Reference. Apparsa la finestra di
                     dialogo, doppio-click sulla stringa che ci interessa.
                     Una breve pausa... et voilà ecco la nostra stringa nel
                     punto di codice che la richiama.

Ricerca di testo : ^Search^ Find Text. potete ricercare qualsiasi parola che
                     compaia nel disassemblato. Anche un'istruzione.

Andare ad un indirizzo : ^Goto^ Goto Code Location(F12). Inserite
                           l'indirizzo in esadecimale.

Caricare il processo ^Debug^ Load Process (Ctrl+L). Carica in memoria il
                       processo, per poi poterlo debuggare.

Caricato il processo appariranno altre due finestre di dialogo con i comandi
del debugger. In oltre, adesso, altre funzioni sono ora disponibili.

Impostare i Breakpoit : ^Debug^ Breakpoit Toggle (F2).

Debuggare il programma : [Auto Step Over],[Auto Step Into],[Step Over],
                           [Step Into]. Il funzionamento dei vari tasti è già

stato illustrato in precedenza, comunque c'è sempre
il tasto F1 :-)

Andare ad un indirizzo : [Goto Address].

Modificare il codice : [Patch Code]. Questa è la parte più delicata del cracking. Bisogna avere un minimo di conoscenza di quello che si scrive. Altrimenti il programma eseguirà istruzioni non valide o, peggio, illegali. Nel caso della modifica di un jump, basterà riscrivere tutta la riga, ovviamente con il jump negato.

Es.: Se l'EIP contiene l'istruzione : jne 00456124, voi dovrete scrivere
je 00456124 [return].

Nel caso di una CALL o di qualsiasi altra istruzione da far saltare, bisognerà fare attenzione a coprire tutta l'istruzione.

Es.: Se l'EIP contiene l'istruzione : E8541E0000 call 00402E80, voi dovrete scrivere:

```
     E8          54          1E          00          00
nop [return] nop [return] nop [return] nop [return] nop [return].
```

Una volta premuto l'ultimo [return] la Code Patch List verrà modificata. Quando avete finito, [Apply Patch], e confermate la domanda. Infine [Close] per chiudere la finestra.

NOTA : il codice va modificato PRIMA di cominciare il debug. Altrimenti non potremo verificare se il codice che abbiamo modificato funziona o meno.

Annotare il codice modificato La patch list si presenterà più o meno in questo modo.

Es.:

```
:0040102E  740F                  je 00401044
:00401030  90                    nop
:00401031  90                    nop
:00401032  90                    nop
       |    |                     |
       |     ------               |
       |           |              |
  indirizzo   valore HEX       istruzione
               (opcode)
```

Dopo aver annotato il valore HEX e gli indirizzi corrispondenti, chiudiamo la finestra di patch. Dalla finestra del listato andiamo agli indirizzi annotati. Nella status bar comparirà l'offset dell'istruzione. Annotato anche questo, possiamo chiudere il WDASM e modificare il codice con un editor esadecimale.

Appendice -B- USARE ULTRAEDIT 5.1

Premetto che, per modificare un file binario, non è necessario avere UltraEdit, un semplice editor esadecimale va più che bene.
Le funzioni che useremo sono solo quattro: aprire il file, andare al giusto offset, modificare, salvare. Come vedete non è difficile... :-)

Aprire un file : ^File^ Open...(Ctrl+O)

Trovare l'offset : ^Search^ Goto Line/Page. Inserire un offset esadecimale.
Es.: 0x00003793

Modificare : Basta scrivere il valore HEX (opcode) dell'istruzione all'offset trovato.

Salvare : ^File^ Save (Ctrl+S)

```
-------------------------------------
 Appendice -C-  GENERIC PATCH PROGRAM
-------------------------------------


Riporto qui il sorgente, scritto in C, del GPP (Generic Patch program).
Modificando le righe opportune, lo si può adattare a qualsiasi programma da
patchare.

////////////////////// TAGLIARE QUI /////////////////////////

/*
 Seguire le istruzioni all'interno del file per personalizzare il tuo patch.
 Questo prg. e' TOTALMENTE free, comunque se lasciate la riga
 "Written by ALoR" vi saro' riconoscente.

 Compilare con Borland C++ 3.1 o superiore.                                 */


/*-------------------------------- LIBRERIE -------------------------------
------------------------------------------------------------------------------*/
#include
#include
#include
#include
#include

/*------------------------------ DICHIARAZIONI ----------------------------
------------------------------------------------------------------------------*/

// inserire il nome del file da patchare tra le "  "
        #define FC              "prg.exe"

// inserire il nome del programma tra le "  "
        #define NAME            "Proggy 2.0"

// inserire il tipo di programma tra le "  "
        #define TIPO            "Pgoggy"

// inserire il tipo di protezione tra le "  "
        #define PROTECTION      "Trial 5 days."

// inserire la lunghezza in byte del file da patchare
        #define DIMFILE         3364864l        // Non toccare la l in fondo

// inserire l'offset dell'istruzione da patchare.
// nel caso di piu' istruzioni definire altre variabili offset1, offset2 ecc.
        #define OFFSET          0x00003b13



/*-------------------------------- PROCEDURE ------------------------------
------------------------------------------------------------------------------*/

void Logo(void)
{
  textcolor(LIGHTCYAN);
  cprintf(" ============================================================\n\r");
  cprintf(" ----------===> Generic Patch Program v 1.0 <===---------\n\r");
  cprintf(" ============================================================\n\r");
  textcolor(LIGHTBLUE);
  cprintf("\n ---------------===> Written by  ALoR <===---------------\n\r");
  textcolor(BLUE);
  cprintf(" ----===> Proud member of  N E U R O  Z O N E  2 <===----\n\n\r");
  textcolor(DARKGRAY);
}

/*------------------------------------------------------------------------------
------------------------------ PROGRAMMA MAIN -------------------------------
------------------------------------------------------------------------------*/

int main(void)
{ FILE *FileCrack;

  unsigned char c;

  clrscr();
```

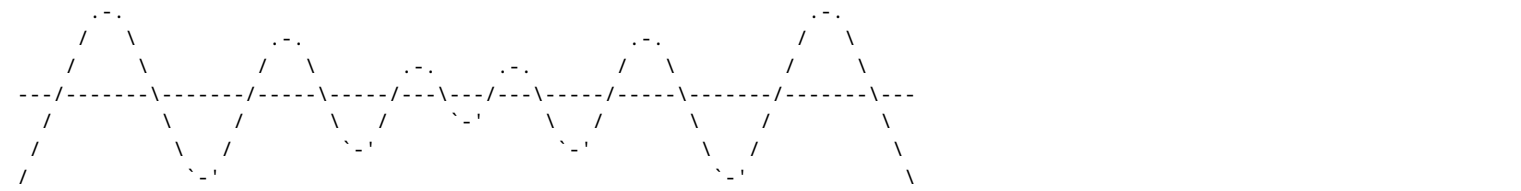
```
  Logo();
  cprintf("ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ\n\r");
  cprintf(" PRoGRaM    | %s\n\r",NAME);
  cprintf(" TyPe       | %s\n\r",TIPO);
  cprintf(" PRoTeCTioN | %s\n\r",PROTECTION);
  cprintf("ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ\n\r");

//---- Esiste ? -----}
  ffblk *dati;
  char *FileName;
  int result;

  strcpy(FileName,FC);

  result=findfirst(FileName,dati,FA_RDONLY | FA_HIDDEN | FA_ARCH | FA_SYSTEM);

  cprintf(" Searching for %s ....",FC);
  for (int i=0;i<12-strlen(FC);i++)
      cprintf(".");
  delay(500);

  if (result) {         // Findfirst ritorna 0 quando va a buon fine
    cprintf("%s not found !!\n\n\r",FC);
    return(1);
  }
  else
    cprintf("OK.\n\r");


//{---- Versione ----}

  cprintf(" Checking FiLE Version .........");
  delay(500);

  if (dati->ff_fsize==DIMFILE)
    cprintf("OK.\n\r");
  else {
    cprintf("Invalid Version !!\n\n\r");
    return(2);
  }

//{---- Cracking ---- }

  FileCrack=fopen(FileName,"rb+");
  cprintf("\n Cracking FiLE .");

  for (i=0;i<16;i++) {
    delay(80);
    cprintf(".");
  }

/* le righe che seguono dipendono da quanti byte devi cambiare.
   In questo caso 1 Byte. Se devi cambiare piu' byte, basta copiare queste
   righe per il numero di byte che ti servono. Fai attenzione a cambiare
   OFFSET con OFFSET1, OFFSET2, ecc.                                    */


  fseek(FileCrack,OFFSET,SEEK_SET);
  fread(&c,sizeof(unsigned char),1,FileCrack);

// "c" e' il byte da cambiare, ovviamente in esadecimale.

  if (c==0x74)                          // controlla se e' gia' patchato
    cprintf("Already Cracked !!\n\r");
  else {
    c=0x74;
    fseek(FileCrack,-1,SEEK_CUR);
    fwrite(&c,sizeof(unsigned char),1,FileCrack);


    cprintf("Done.\n\r");
  }
  fclose(FileCrack);
  getch();
  return(0);
}
 //      :ú---------==> THE END <==----------ú:
```

////////////////////// TAGLIARE QUI ////////////////////////

 7.0 CONCLUSIONI

Spero che il mio manuale vi sia stato d'aiuto per iniziare a capire e a
muovere i primi passi nell'affascinante mondo del CRaCKiNG.
Questo manuale è stato redatto in modo da rivolgersi ad un pubblico inesperto
o con una minima preparazione. Probabilmente in futuro ne uscirà una versione
per quelli di voi già più esperti. Per avere eventuali future release di
questo manuale scrivete a: nz2@usa.net
oppure visitate il sito http://alor.home.ml.org
Vi congedo con una massima di +ORC il più famoso cracker della rete.

              "If you give a man a crack he'll be hungry again
               tomorrow, but if you teach him how to crack, he'll
               never be hungry again"
                                                +ORC



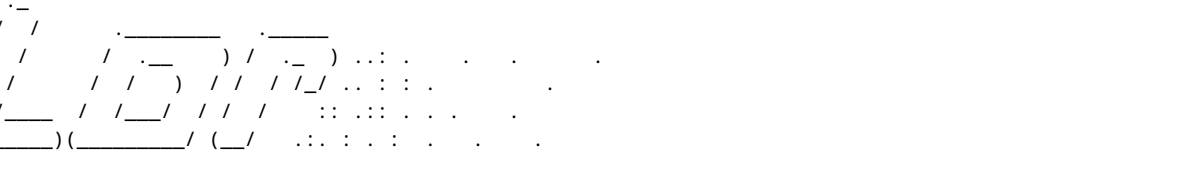
Greetings to: LordKasKo (my cracking teacher...)

                Ob1 (for the help in writing C source)

                The other NEURO ZONE 2 members (10t8or, DK2DEnd, LordKasKo,
                                                MaPHas, Ob1, TurboSnail, XXXX)

                All the Cracker on the NET from whom I have learned something.

==



            --==> ALoR <==----------------------- - -   -
                                  ICQ: 10666678   In IRC: WhiteFly

          e-mail: ALoR@thepentagon.com   www: http://ALoR.home.ml.org/

==