12345678901234567890123456789012345678901234567890123456789012345678

Winamp Reversing
ovvero
fate fare ai vostri programmi
quello che vi pare

by .+Ma.

********** INTRO **********

Lo scopo di questo tute è mostrare quanto sia possibile fare con un po' di manipolazione del PE... e una mente malata (il nick non e' casuale :) Se vi scatenate un po', forse potete riuscire anche voi a combinare qualcosa di altrettanto skizzato... vi sfido a farlo! Adesso sedetevi, fate un bel respiro e provate a seguire il filo del discorso: alla fine del tute, ve lo garantisco, guarderete i vostri programmi da un diverso punto di vista. E ora pronti, partenza... via.

CONSUMO: 2 Coke 33cl
Succo ai frutti tropicali
Pizza (YUM!)
MUSICA: Afterhours - Non è per sempre (l'album, naturalmente crittato ;)

********** TOOLS **********

File XORer 0.3 build 009 [by JFL]
Sadd v1.0b by NeuRaL_NoiSE (thx!)
Pebrowse v4.0 by PRUDENS inc. (http://www.spywindows.com)

**********DOCS**********

Tutto quello che riuscite a trovare sul PE, in particolare:

-"PE-Crypters : uno sguardo da vicino al c.d. "formato" PE" by Kill3xx
-"Come inserire una nuova sezione e una nuova funzione in un PE" by Pusillus
-I tutorial di NeuRaL_NoiSE. Molti trattano manipolazione del PE... e anche se ne beccate uno che non c'entra nulla varra' comunque la pena leggerlo ;)

********** THE TUTE, AT LAST :) **********

Tempo fa mi era venuta un'idea decisamente malata: modificare winamp per fare in modo che potesse leggere mp3 in formato crittato. La cosa non mi sembrava impossibile, ma necessitava di due componenti essenziali: un cracker motivato e del tempo libero. Al momento, mi mancavano entrambe le componenti, mentre ora mi manca solo la seconda... La soluzione e' stata semplice: mi e' bastato togliere un po' di ore al sonno per rendere possibile il "progetto winamp" e scrivere questo tutorial (bambini, non fatelo a casa)!

La prima domanda che mi e' stata fatta - praticamente da tutti coloro a cui avevo raccontato di questo progetto - era: "ma a che serve crittare le mp3?". Erm... come facevo a spiegare loro che era solo un esercizio di stile, una operazione fine a se stessa, praticamente una questione di principio? Loro probabilmente (tutte menti illuminate) mi avrebbero capito... ma io dovevo assolutamente trovare qualcosa che mi convincesse dell'utilita' di quello che stavo facendo :) Le conclusioni a cui sono giunto sono le seguenti:

- Ormai la maggior parte dei fornitori di spazio web gratuito controlla che non siano presenti mp3 all'interno di un sito, altrimenti lo sega. Siccome il trucco di cambiare l'estensione dei file e' ormai vecchiotto, una

  buona idea potrebbe essere quella di crittare i file in modo che non siano
  facilmente riconoscibili;

- Se si vuole fare uno scambio massiccio di mp3 su CD spedendole per posta,
  si rischia parecchio... ora, forse la crittazione dei file all'interno del
  cd non e' proprio l'idea piu' intelligente, pero' e' sempre un'idea... Si
  noti che, mentre nel caso precedente e' possibile decrittare l'mp3 dopo
  averla scaricata da Internet, in questo e' una vera palla decrittarsi un
  CD intero e rimasterizzarselo: un lettore che decritta on the fly sarebbe
  il massimo;

Risolto il problema del "perche'", ora non mi restava altro che decidere il
"come"... infatti, la seconda domanda che mi e' stata fatta era: "Ma perche'
non ti programmi direttamente un player di mp3 crittate?". Anche qui, la mia
prima risposta era: "perche' sono un reverse engineer, e se non rovescio
qualcosa non mi diverto"... ma non era abbastanza convincente. Allora, ho
deciso che la versione ufficiale sarebbe stata la seguente:

- Dopo aver capito come modificare Winamp per fargli leggere mp3 crittate al
posto di quelle normali, sara' possibile eseguire la stessa operazione con
praticamente qualsiasi altro programma. Risultato: per quanto semplice possa
essere il vostro algoritmo di crittazione, sara' comunque un formato
proprietario piu' complicato da decrittare rispetto a quelli standard, in
quanto riconosciuto SOLO dalla VOSTRA versione del programma.

... bello, eh? :)
A questo punto, si trattava solo di mettersi all'opera. Schematizzando, il
mio lavoro (e il vostro, da questo momento) si e' ridotto a questi passi:

a) Studio del programma
   ovvero: "ma proprio una dll mi doveva capitare?"
b) Aggiunta di una nuova sezione
   ovvero: "lo spazio c'e' ma mi piace giocare col PE" (fa pure rima! :)
c) Esperimenti
   ovvero: "evitiamo di crittarci l'HDD, la ram, il monitor e la tastiera"
d) API hook
   ovvero: "un so se il termine e' giusto, ma suonava bene"
e) Decrittazione
   ovvero: "tanta fatica per du' righe di codice? Mavvaff..."
f) Crittazione
   ovvero: "si'... ho fatto un bel lavoro ma ke kappero ascolto adesso?"
g) Ascolto
   ovvero: "Funziona? Naaaaaa... non e' possibile!"

Analizziamo ora tutto il processo, passo per passo.

****
a) Studio del programma
   ovvero: "ma proprio una dll mi doveva capitare?"
****

Ebbene si', la parte di codice di winamp che sovrintende al caricamento delle
mp3 e' contenuta proprio in una dll. Per l'esattezza, TUTTE le parti di
codice di winamp che sovrintendono al caricamento dei diversi tipi di file
audio si possono trovare all'interno di dll. In particolare, quella adibita
al caricamento delle mp3 si chiama in_mp3.dll.

Come si puo' scoprirlo? Semplicissimo, basta avviare winamp, aprire la
finestra di softice con un bel CTRL-D (heh, il double size mode di winamp
dovreteselezionarlo da menu ;) e quindi settare un bel breakpoint sull'API

ReadFile. Appena restituito il controllo all'esecuzione del programma, il caro vecchio Sice si riaprira' all'interno della chiamata all'API: premete F12 e vi troverete davanti la sezione di codice incriminata e il nome della DLL di cui fa parte.

Il fatto che quella su cui dobbiamo lavorare sia una dll comporta sia dei vantaggi sia degli svantaggi: un vantaggio non indifferente e' dato dal fatto che possiamo supporre che tutte le chiamate a readfile per la lettura dell'mp3 siano contenute al suo interno; addirittura, possiamo supporre che TUTTE le chiamate a readfile ci interessino (e un'analisi del codice ce lo confermera'); lo svantaggio, invece, e' che bisogna studiarsi un po' il funzionamento delle dll e del loro PE... ed e' proprio a questo punto che si passa alla fase successiva.

****
b) Aggiunta di una nuova sezione
   ovvero: "lo spazio c'e' ma mi piace giocare col PE" (fa pure rima! :)
****

La prima cosa che bisogna fare quando si decide di aggiungere del codice a un programma e' quella di scegliere se inserirlo alla fine di una sezione esistente (sempre che ci sia spazio) o all'interno di una nuova. Per fare questo, occorre innanzitutto dare un'occhiata allo stato attuale delle sezioni:

```
Disassembly of File: in_mp3.dll
Code Offset = 00001000, Code Size = 0001E000
Data Offset = 00025000, Data Size = 00005000

Number of Objects = 0005 (dec), Imagebase = 11000000h

Object01: .text    RVA: 00001000 Offset: 00001000 Size: 0001E000 Flags: 60...
Object02: .rdata   RVA: 0001F000 Offset: 0001F000 Size: 00006000 Flags: 40...
Object03: .data    RVA: 00025000 Offset: 00025000 Size: 00005000 Flags: C0...
Object04: .rsrc    RVA: 0005A000 Offset: 0002A000 Size: 00003000 Flags: 40...
Object05: .reloc   RVA: 0005D000 Offset: 0002D000 Size: 00003000 Flags: 42...
```

Good. La prima sezione, chiamata .text, e' quella che contiene il codice e, come si puo' vedere con un qualsiasi editor esadecimale, ha un casino di spazio libero alla fine. Infatti, sapendo che essa ha dimensione 1E000 e un offset di 1000 all'interno del file, ci basta dare un'occhiata nei paraggi dell'offset 1F000 con l'hexed per vedere una gran quantita' di zeri. Ne segue che possiamo inserire il codice alla fine della sezione gia' esistente, quindi... creiamo una nuova sezione. :)

(Lo so, mi odiate, ma dovete pensare che lo faccio per voi: ricordate che questa lezione cerca di essere il piu' generale possibile, quindi non posso dare per scontato che lo spazio ci sia sempre!)

Per creare una nuova sezione sono necessarie alcune conoscenze di base sul formato PE, conoscenze che mi mancavano la prima volta che ho fatto delle prove... le quali hanno avuto esito disastroso, naturalmente ;) Ad ogni modo, procuratevi della buona documentazione (su Ringzer0 dovrebbe esserci piu' o meno tutto quello che vi puo' servire) e vedrete che non avrete troppi problemi a seguirmi. Ah, un consiglio... fate un backup di in_mp3.dll o, meglio ancora, copiatela con un altro nome e modificate la copia (verra' in seguito visualizzata all'interno di winamp come un nuovo plugin!). Io ho chiamato la mia in_mal.dll :)

Innanzi tutto, e' necessario individuare le caratteristiche della nostra

nuova sezione, cioe' i dati che la caratterizzeranno all'interno dell'header PE. Per questo, recuperiamo innanzitutto un tool per "spiare" il pe originale (PEBrowse), poi una definizione della sections table del PE, come ad esempio quella del "kill3xx", il mitico testo di riferimento ;) Citandolo con ossequioso rispetto:

- SName: stringa di 8 byte con il nome della sezione. Scegliete il nome che preferite, io ho usato ".mala" :)

- SVirtualSize: contiene la dimensione fisica (vedi SizeOfRawData) dei dati arrotondata ad un multiplo del section aligment. Poichè il section alignment e' uguale a 1000 (come si puo' leggere all'interno di PEBrowse, nella sezione Optional Header), scegliamo 1000 anche per SVirtualSize.

- SVirtualAddress (RVA): permette di calcolare la posizione che avra' la sezione una volta caricata in memoria dal loader. Deve essere un multiplo del section alignment. In questo caso, poiche' l'ultima sezione ha RVA 5D000 e Size 3000, l'RVA di .mala sara' 5D000+3000= 60000, che e' gia' arrotondato al section alignment.

- SizeOfRawData: la dimensione fisicamente occupata dai dati su disco, allineata al file alignment. Poiche' file alignment=1000, scegliamo come per SVirtualSize una dimensione di 1000 byte.

- PointerToRawData: l'offset "fisico" a cui troverete i dati della sezione. Per questo e' sufficiente sommare l'offset dell'ultima sezione alla SizeOfRawData dell'ultima sezione, arrotondando il risultato al file alignment. Quindi: 2D000 + 3000 = 30000. Anche questo valore e' gia' arrotondato.

- SFlags: i flag che identificano le caratterestiche (codice,dati,ecc.) e quindi le i flags e le protezioni di pagina che verranno applicate (writable,readable,ecc.). In questo caso utilizziamo gli stessi del codice della prima sezione, cioe' 0x60000020.

Notate che mentre alcuni dei valori sono facilmente reperibili da Wdasm altri devono essere letti da PEBrowse... posto che all'interno di quest'ultimo prog c'e' praticamente tutto quello che vi serve, lasciate perdere Wdasm :)

Ora che vi siete sorbiti tutta la pappardella teorica e siete pronti a metter mano all'interno della dll con il vostro editor esadecimale... lasciatelo perdere: il programma Sadd del buon vecchio Neuro fa il lavoro duro per voi! Esso aggiorna automaticamente anche l'Image Size, un campo decisamente importante (in particolare sotto NT) che ancora non e' stato descritto: esso riporta la grandezza dell'immagine una volta in memoria e, naturalmente, dev'essere modificato quando vengono aggiunte delle sezioni.

Naturalmente, se aveste avuto bisogno di aggiungere all'interno della dll delle funzioni da esportare questo tool non vi sarebbe stato sufficiente. Per fortuna, pero', non e' il nostro caso: la parte di codice che aggiungeremo non dovra' essere chiamata da winamp, ma dallo stesso codice della dll (che fortuna, ora non ci interessa piu' lo "svantaggio" di dover lavorare con una dll!).

Benissimo: ora che abbiamo aggiunto la nostra nuova sezione, non ci resta altro da fare che provarla... per questo passiamo alla fase successiva.

****
c) Esperimenti

   ovvero: "evitiamo di crittarci l'HDD, la ram, il monitor e la tastiera"
****

Beh, in realta' la situazione non e' cosi' grave come sembra... pero' di solito preferisco procedere per piccoli passi e non mi va di inserire subito una routine di crittazione senza sapere esattamente cosa voglio fare. Quindi, facciamo il punto della situazione.

Vogliamo leggere mp3 crittate. Come? Beh, innanzittutto leggendo il file da disco, come gia' avevamo intuito quando avevamo fatto il breakpoint sull'api readfile. In seguito, una volta letto il file (o -piu' in generale- una volta letta una quantita' qualsiasi di byte presi da questo file), vorremmo avere la possibilita' di decrittarlo in memoria facendo in modo che il resto del programma non si accorga di nulla. Come, esattamente? Studiamoci un po' l'API:

```
BOOL ReadFile(

byte  HANDLE hFile,                    // handle of file to read
dword LPVOID lpBuffer,                 // address of buffer that receives data
dword DWORD nNumberOfBytesToRead,      // number of bytes to read
dword LPDWORD lpNumberOfBytesRead,     // address of number of bytes read
dword LPOVERLAPPED lpOverlapped        // address of structure for data
);
```

Good. Date un'occhiata ai parametri che vengono passati alla funzione: fra di essi potete notare l'indirizzo del buffer all'interno del quale vengono memorizzati i dati e il numero di byte letti. Perfetto, e' esattamente quello che ci serve: dopo aver letto una qualsiasi quantità di byte, potremo andare a modificarli in memoria sapendone l'indirizzo e il numero. Come? Semplice: intercettiamo tutte le chiamate a ReadFile e le reindirizziamo alla nostra parte di codice, all'interno del quale prima verra' chiamato readfile e poi verra' decrittato il blocco di byte letti da file e salvati in memoria. Al ritorno dalla nostra funzione, per il programma sara' come aver semplicemente eseguito la chiamata a ReadFile.

E' ora il momento di cominciare a scrivere la nostra funzione: almeno per ora essa sara' semplicemente una replica della chiamata a ReadFile, in modo da poter controllare con facilita' il passaggio dei parametri e prendere un po' di confidenza con gli elementi sullo stack. In pratica, si tratta solo di prendere dallo stack i parametri passati alla nostra call, ripusharli (wow, bel termine :)), chiamare ReadFile e scaricare dallo stack i parametri che possiamo chiamare "originali".

Occorre a questo punto una breve spiegazione teorica: quando viene chiamata una funzione che richiede dei parametri, questi vengono PUSHati sullo stack prima della call vera e propria. Ad esempio, nel caso di ReadFile che vuole cinque parametri, essi corrispondono agli ultimi cinque valori che sono stati PUSHati (e sono proprio quelli che ci interessano). Questo e' un concetto da tener sempre presente, in quanto puo' tornare MOOOLTO utile in varie situazioni, a partire da quando volete comprendere un disassemblato fino ad arrivare alla ricerca di numeri seriali in memoria.

Quando viene eseguita una call, in cima allo stack viene pushato un altro valore corrispondente all'indirizzo di ritorno della funzione: tale valore e' quello che permette al programma di ritornare, quando trova il comando RET, all'indirizzo successivo a quello della chiamata (e, in pratica, e' proprio quest'indirizzo che viene salvato nello stack).
Questo significa che, una volta all'interno della nostra funzione, il primo parametro (quello, cioe', all'indirizzo indicato da ESP) sara' l'indirizzo

di ritorno, mentre i VERI parametri della funzione ReadFile sono memorizzati a partire dall'indirizzo ESP+4. Per essere piu' precisi, all'indirizzo ESP+4 sara' memorizzato l'ULTIMO parametro pushato (ricordate che le push seguono un ordine inverso a quello mostrato nella definizione dell'API?), e gli altri seguono a distanza di 4 byte l'uno dall'altro. A questo punto, il codice per replicare la chiamata a ReadFile potrebbe essere il seguente:

```
55             push ebp                       // salva il contenuto di ebp
                                                 NOTA: da questo momento tutti
                                                 gli elementi nello stack si
                                                 "spostano" di 4 byte!
8BEC           mov ebp, esp                   // usiamo ebp come valore fisso
                                                 per leggere gli elementi
                                                 nello stack
8B4518         mov eax, dword ptr [ebp+18]    // salva in eax il 5o parametro
50             push eax                       // PUSH 5o parametro
8B4514         mov eax, dword ptr [ebp+14]    // salva in eax il 4o parametro
50             push eax                       // PUSH 4o parametro
8B4510         mov eax, dword ptr [ebp+10]    // salva in eax il 3o parametro
50             push eax                       // PUSH 3o parametro
8B450C         mov eax, dword ptr [ebp+0C]    // salva in eax il 2o parametro
50             push eax                       // PUSH 2o parametro
8B4508         mov eax, dword ptr [ebp+08]    // salva in eax il 1o parametro
50             push eax                       // PUSH 1o parametro
FF1520F00111   Call dword ptr [1101F020]      // Chiama ReadFile
5D             pop ebp                        // recupera il valore originario
                                                 di ebp
************************************************
     INSERT DECRYPTION ALGORITHM HERE!!!
************************************************
C21400         ret 0014                       // torna indietro eliminando gli
                                                 ultimi 5 elementi dello stack
```

Notate l'ultimo comando, il RET 0014: questo e' dovuto al fatto che sullo stack, dopo l'indirizzo di ritorno, saranno sempre presenti anche i 5 parametri della funzione ReadFile, pushati dal programma originale! E' quindi necessario eliminarli riportando lo stack nello stesso stato in cui si sarebbe trovato se ci fosse stata la semplice call a ReadFile.

A questo punto non ci resta altro che fare una prova: prima di reindirizzare tutte le chiamate, naturalmente, ho provato con una sola, quella che si trova all'indirizzo 110014DD, la quale legge la tag id3 x la visualizzazione del titolo della canzone. Essa viene chiamata una sola volta all'inizio della riproduzione, quindi risulta indicata per delle prove in quanto non puo' creare troppo casino :)

```
* Reference To: KERNEL32.ReadFile, Ord:0218h
                                   |
:110014DD FF1520F00111             Call dword ptr [1101F020]
```

poiche' la nostra funzione inizia all'indirizzo 11060000 (ricordiamo che il valore si ottiene semplicemente sommando imagebase ed RVA), possiamo sostituire la call all'indirizzo 100014DD con una

```
:110014DD E81EEB0500               call 11060000
:110014E2 90                       nop
```

Per calcolare il valore relativo della chiamata e' sufficiente sottrarre al valore 11060000 l'indirizzo dell'istruzione successiva alla call. In questo caso, basta fare 11060000-110014E2=0005EB1E. Ricordando che il valore sotto

forma di opcode dev'essere ribaltato, e' esattamente quello che vedete qui sopra. A questo punto eseguite il programma (ricordandovi di abilitare la vostra dll alla riproduzione di mp3 e di disabilitare l'originale) e... non ci crederete, ma funziona! :)

****
d) API hook
   ovvero: "un so se il termine e' giusto, ma suonava bene"
****

Ora, io non so se questo termine va bene... pero' l'immagine a cui pensavo era proprio quella: "agganciare" le chiamate all'API e reindirizzarle dove desideravo. Mah, speriamo che vada bene :)

Quest'operazione l'abbiamo appena eseguita, nella scorsa sezione, per fare la nostra prova: adesso bisogna ripeterla per tutte le chiamate a ReadFile. Cercando all'interno del disassemblato la stringa "ReadFile" ho trovato 11 ricorrenze, 9 delle quali erano assolutamente identiche fra di loro

```
FF1520F00111            Call dword ptr [1101F020]
```

mentre le altre 2 erano differenti, rispettivamente una CALL EDI e una CALL ESI, i cui valori erano stati identificati, rispettivamente, dai comandi

```
8b3D20f00111            mov edi, dword ptr [1001F020]
```

e

```
8b3520f00111            mov esi, dword ptr [1001F020]
```

Bene, ora dobbiamo ripetere tutti quei pallosissimi calcoli di prima per il calcolo dei valori relativi nelle nove call da reindirizzare? Beh, purtoppo si'... ma c'e' un piccolo trucco per semplificare le cose :)
Poiche' il comando occupa 5 byte (1 di operatore + 4 di operando) possiamo risolvere la semplice equazione:

```
Operando = Indirizzo della funzione chiamata - (Indirizzo del comando + 5)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^    ^^^^^^^^^^^^^^^^^^^^^^^^^^
           questo valore rimane sempre fisso    questo val invece cambia!
```

che equivale a:

```
Operando = (Indirizzo della funzione chiamata - 5) - Indirizzo del comando
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^    ^^^^^^^^^^^^^^^^^^^^^^
              questo valore rimane sempre fisso          questo cambia
```

Nulla di incredibile... pero' ci semplifica i calcoli! Questo torna comodo in particolare in casi come il nostro, in cui bisogna reindirizzare da vari punti le chiamate a una stessa funzione. A questo punto ci bastera' fare, per ogni calcolo, la semplice sottrazione 1105FFFB-(indirizzo della call). I valori, alla fine, sono i seguenti:

```
01) 11001087: FF1520F00111 --> E874EF050090
02) 110014DD: FF1520F00111 --> E81EEB050090
03) 110015A8: FF1520F00111 --> E853EA050090
04) 11001649: FF1520F00111 --> E8B2E9050090
05) 110017be: FF1520F00111 --> E83DE8050090
07) 11007e05: FF1520F00111 --> E8F681050090
08) 11008333: FF1520F00111 --> E8C87C050090
09) 110083f1: FF1520F00111 --> E80A7C050090
```

11) 110088b2: FF1520F00111 --> E84977050090

A questo punto restano soltanto le due call "diverse"... un modo per patchare queste chiamate consiste nel modificare il comando che salva l'indirizzo in ESI (o in EDI) con un comando che salva in questi registri il valore 11060000.

06) 110077f8: CALL EDI: modificare a 110077f0 mov edi, dword ptr etc.
              8b3D20f00111 --> BF0000061190
10) 11008665: CALL ESI: modificare a 11008650 mov esi, dword ptr etc.
              8b3520f00111 --> BE0000061190

Et voila'... ecco fatto! Queste sono le sostituzioni da fare, un po' pallose ma alla fine funziona ancora tutto... e scusate se e' poco ;) A questo punto, possiamo trasformare la nostra funzione farlocca in una che faccia davvero qualche cosa di utile... chesso', decrittare un file mp3!

****
e) Decrittazione
   ovvero: "tanta fatica per du' righe di codice? Mavvaff..."
****

In precedenza, nella sezione C, mostrando il codice della nostra funzione avevo lasciato uno spazio vuoto all'interno del quale inserire l'algoritmo di decrittazione. E' giunto ora il momento di scriverne uno, decidendo innanzi tutto che tipo di crittazione desideriamo effettuare.

Innanzi tutto e' necessario specificare il fatto che tale algoritmo non puo' agire direttamente sul file intero, ma solo sulle porzioni di codice che vengono lette man mano dal programma. Per questo motivo, non e' sicuro (ne' facile da implementare in fase di crittazione) lavorare spostando i byte, mentre e' decisamente preferibile manipolarli con delle operazioni piu' semplici. Nel mio caso ho deciso di utilizzare un semplicissimo xor, ma nessuno vi impedisce di crearvi direttamente il vostro algoritmo seguendo la falsariga del mio.

Prima di mostrarvi il codice, occorre fare qualche premessa: prima di tutto, ci occorreranno due elementi presenti nello stack, per l'esattezza il secondo e il quarto parametro, rispettivamente l'indirizzo a partire dal quale sono memorizzati i dati letti da file e il numero di byte effettivamente letti. Poiche' quando si raggiunge la fine del file tale numero e' uguale a zero, ci conviene fare un controllo su di esso per evitare di crittarci porzioni varie di memoria (ehehe... Neu sa ke mi e' capitato ;). Infine, siccome la chiave per lo xor puo' essere scelta a piacere, lascero' all'interno del codice una etichetta "val" che potra' essere sostituita al momento giusto.. addirittura, potreste chiamare una dialogbox per chiedere una password, a partire dalla quale verra' generata la chiave di decrittazione!

Eccovi il codice commentato:

```
56                  push esi                    // Salva i valori contenuti
51                  push ecx                    // nei 3 registri che vengono
50                  push eax                    // usati in seguito.
8b742414            mov esi, dword ptr [esp+14] // esi=indirizzo di partenza
8b4c241c            mov ecx, dword ptr [esp+1c] //
8b09                mov ecx, dword ptr [ecx]    // ecx=numero di byte da decr

85c9                test ecx, ecx               // ecx=0? Allora STOP.
740c                jz endloop

loop:
```

```
8a06                mov al, byte ptr [esi]      // leggi 1 byte
34[val]             xor al, [VAL]               // fai uno xor del byte con
                                                   il valore VAL
8806                mov byte ptr [esi], al      // rimetti  il byte dov'era
46                  inc esi
49                  dec ecx
85c9                test ecx, ecx               // son finiti i byte da
                                                   decrittare?
75f4                jnz loop

endloop:
58                  pop eax
59                  pop ecx
5e                  pop esi                     // recupera i valori dei reg.
```

Ecco fatto! Nulla di particolarmente complicato, come potete vedere... ma il bello e' che all'interno di quel loop potete metterci praticamente quello che vi pare :) Io, ad esempio, ho scelto il valore esadecimale ED, e il disasm finale della mia funzione e' questo:

```
**************************LAST VERSION DISASSEMBLY************************

:11060000 55                        push ebp
:11060001 8BEC                      mov ebp, esp
:11060003 8B4518                    mov eax, dword ptr [ebp+18]
:11060006 50                        push eax
:11060007 8B4514                    mov eax, dword ptr [ebp+14]
:1106000A 50                        push eax
:1106000B 8B4510                    mov eax, dword ptr [ebp+10]
:1106000E 50                        push eax
:1106000F 8B450C                    mov eax, dword ptr [ebp+0C]
:11060012 50                        push eax
:11060013 8B4508                    mov eax, dword ptr [ebp+08]
:11060016 50                        push eax

* Reference To: KERNEL32.ReadFile, Ord:0218h
                                    |
:11060017 FF1520F00111              Call dword ptr [1101F020]
:1106001D 5D                        pop ebp
:1106001E 56                        push esi
:1106001F 51                        push ecx
:11060020 50                        push eax
:11060021 8B742414                  mov esi, dword ptr [esp+14]
:11060025 8B4C241C                  mov ecx, dword ptr [esp+1C]
:11060029 8B09                      mov ecx, dword ptr [ecx]
:1106002B 85C9                      test ecx, ecx
:1106002D 740C                      je 1106003B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:11060039(C)
|
:1106002F 8A06                      mov al, byte ptr [esi]
:11060031 34ED                      xor al, ED
:11060033 8806                      mov byte ptr [esi], al
:11060035 46                        inc esi
:11060036 49                        dec ecx
:11060037 85C9                      test ecx, ecx
:11060039 75F4                      jne 1106002F

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

```
|:1106002D(C)
|
:1106003B 58                    pop eax
:1106003C 59                    pop ecx
:1106003D 5E                    pop esi
:1106003E C21400                ret 0014
```

****
f) Crittazione
   ovvero: "si'... ho fatto un bel lavoro ma ke kappero ascolto adesso?"
****

Bene, ora avete una dll che consente al buon vecchio winamp di leggere le mp3 crittate... e guai a chi dice "ma io non ho mp3 crittate!" :)

Se avete riciclato il mio algoritmo di xor non c'e' nessun problema, basta scrivervi un semplicissimo programma che vi XORi i file... oppure potete utilizzare il File XORer di JFL dalla mia pagina (http://malattia.cjb.net, sezione tools/downloads)... io personalmente ho optato per quest'ultima possibilita' (Ehehe... secondo voi perche' avevo scelto un algoritmo cosi' stupido? :))
Se l'algoritmo che avete scelto e' proprietario... beh, cavoli vostri, mi sa proprio che dovrete scrivervi un vostro tool! >:)))

Ora, io so che siete tutti degli studenti intelligenti... pero' per favore, non venitemi a scrivere dicendomi che da quando avete crittato la vostra playlist non riuscite piu' a sentire le vostre canzoni... ok? ;)

****
g) Ascolto
   ovvero: "Funziona? Naaaaaa... non e' possibile!"
****

Et voila'... funziona! Davvero! La dimostrazione e' data dal fatto che giusto adesso mi sto ascoltando un album tutto crittato (lo so che voi non lo potete sentire, ma fidatevi, ok?)... se devo essere onesto, all'inizio non ci credevo neanche io! :)

**********OUTRO**********

Woa, e' finita! Ragazzi, ho fatto piu' fatica a scrivere il tute che a completare il progetto! Adesso fate i bravi, mettete a frutto gli insegnamenti dello zio +Ma e modificate i vostri programmini come vi pare e piace... questo era solo un esempio, adesso date sfogo alla vostra fantasia e fatemi vedere di cosa siete capaci! Per ogni info, complimento, accidente, contributo, offerta generosa potete mandarmi una mail a malattia@gmx.net. Se per caso questo indirizzo fosse morto, controllate quello che c'e' sulla mia pagina: http://malattia.cjb.net.

**********RINGRAZIAMENTI**********

Un grazie a Ringzer0, il gruppo di crecher piu' migliore dell'universo, a #crack-it tutto, e in particolare a:

Neuro, ke mi ha dato un casino di suggerimenti sul PE;
Pusi, ke mi ha dato retta quando farneticavo riguardo alla crittazione di mp3;
Suby, ke mi ha dato del malato quando farneticavo riguardo alla crittazione

      di mp3;
tutti quelli che mi hanno cagato quando farneticavo riguardo alla crittazione
      di mp3;
tutti quelli che non mi hanno mandato a cagare quando farneticavo riguardo
      alla crittazione di mp3;
tutti quelli che son riusciti a leggere fino a questo punto :)

byez,

  .+Ma.