Win2000 bootsector reversing (versione Fat32)
---
written by: Albe (-albe-@libero.it)

Tool usati

- una qualsiasi utility per dumpare un settore del vostro hard disk ... io ho usato WinHex (www.winhex.com), un programmino utile anche per altre occasioni.
- Ida (www.datarescue.com), il miglior disassemblatore :)

Introduzione (sui bootsector)

Prima di iniziare a guardare il codice, vediamo brevemente cosa è un bootsector: a cosa serve, dove si trova, e come funziona.
La funzione di un settore di boot (bleah, non userò più la traduzione italiana, lo prometto) è quella di caricare il sistema operativo. Ogni OS ha un proprio bootsector, e due bootsector di due OS differenti, pur avendo caratteristiche comuni, posso essere parecchio diversi tra loro. Per esempio, un bootsector potrebbe caricare un file binario e lanciarlo; un altro invece potrebbe leggere altri settori del disco (indipendentemente dal filesystem che verrà usato dall'OS), e passare il controllo al codice contenuto in essi.
Come ho detto, tutti i bootsector devono avere delle caratteristiche comuni, in particolare due: devono assumere che CS sia 0x0000 e IP sia 0x7C00 (questo è l'indirizzo al quale il bios carica un bootsector, come riportato nelle BIOS Boot Spec); devono terminare con i byte 0x55 e 0xAA, per essere riconosciuti come bootsector validi. Per il resto, in un bootsector ci si può piazzare ciò che si vuole (sempre rimanendo entro i 510 byte di codice).

Un bootsector di un disco formattato con Fat (12,16 o 32) contiene, a partire dall'undicesimo byte, un "Bios Parameter Block" (BPB): una serie di informazioni sul disco e sulla Fat. Sono informazioni che generalmente servono al bootsector per caricare un file (loader). Grazie a questi valori, infatti, si può capire dove si trova la Fat, e quindi si può risalire alla posizione di un file sul disco. Il bootsector di Win2000, ad esempio, fa uso del BPB per caricare il file "ntldr". Vediamo, di seguito, il BPB della Fat32 (una volta caricato in memoria):

```
 offset | nome e dimensione

7C0B   BytesPerSector       DW    ?
7C0D   SectorsPerCluster    DB    ?
7C0E   ReservedSectors      DW    ?
7C10   NumberOfFATs         DB    ?
7C11   RootEntries          DW    ?  ;valore ignorato dalla Fat32
7C13   TotalSectors         DW    ?  ;grandezza della partizione in settori
7C15   MediaDescriptor      DB    ?  ;0xF8 per gli hard disk
7C16   SectorsPerFAT        DW    ?  ;0 nel caso di Fat32
7C18   SectorsPerTrack      DW    ?
7C1A   Heads                DW    ?
7C1C   HiddenSectors        DD    ?
7C20   BigTotalSectors      DD    ?
7C24   BigSectorsPerFat     DD    ?
7C28   ExtFlags             DW    ?
7C2A   FS_Version           DW    ?
7C2C   RootDirStrtClus      DD    ?
7C30   FSInfoSec            DW    ?
7C32   BkUpBootSec          DW    ?
7C34   Reserved             DW    6 DUP (?)
```

Introduzione (sulla Fat)

Per capire ciò che fa il bootsector di Win2000 (sempre riferito alla sua versione per Fat32), è bene sapere anche qualcosina sulla Fat.
La Fat (File Allocation Table) è originariamente il filesystem dell'MS-DOS. Le sue componenti principali sono due.

Innanzitutto, c'è una vera e propria "Tabella di allocazione dei file", che consiste in un certo numero di "entry" (una per ogni cluster sul vostro hard disk (cluster = unità base in cui vengono memorizzati i file)); ogni entry (12 bit per la Fat12, 16 bit per la Fat16 e 32 bit per la Fat32) fornisce informazioni su un determinato cluster (in pratica c'è una corrispondenza biunivoca tra i cluster e le entry della File Allocation Table). In particolare, leggendo il valore di una entry, si può capire ad esempio se un cluster è danneggiato, o se ci troviamo alla fine di un file, oppure dove si trova il cluster successivo di un file...oppure ancora se il cluster è libero.
I valori delle entry (per quanto riguarda Fat32) possono essere:

```
 Available 00000000h
 Reserved 00000001h
 User Data 00000002h - 0FFFFFFF6h
 Bad Cluster 0FFFFFFF7h
 End Of File 0FFFFFF8h - 0FFFFFFFFh
```

Una seconda struttura della Fat è la directory. Per la Fat, una directory non è altro che un file un po' speciale; al suo interno ci sono alcune informazioni sui file contenuti nella directory stessa. Le informazioni sui file sono raggruppate in entry da 32 byte ciascuna, ed ogni entry è organizzata in questo modo:

```
offset | nome e dimensione

00   FileName   DB 11 DUP (?)
0B   Attributes   DB ?
0C   Reserved   DB ?
0D   Creation (millisecond stamp)  DB ?
0E   Creation Time   DW ?
10   Creation Date   DW ?
```

```
12   Last Access Date  DW ?
14   High 16-bit of cluster number DW ?  ;solo in Fat32
16   Last Write Time  DW ?
18   Last Write Date  DW ?
1A   Starting Cluster  DW ?
1C   File Size (in bytes)  DD ?
```

Come funziona il tutto?
Il filesystem per prima cosa carica da qualche parte un settore di una directory (il bootsector di Win2000 caricherà a 8200h un settore della directory root). Poi cerca, all'interno di questo settore caricato, la entry del file desiderato (confrontando ogni filename). Una volta trovata la entry, il filesystem legge il valore "Starting Cluster" (e, nel caso Fat32, legge anche il valore "High 16-bit of cluster number") e carica il cluster iniziale. A questo punto il filesystem deve sapere con quale cluster continuare la lettura del file. Per calcolare il cluster successivo, carica il settore della Tabella di allocazione dei file all'interno del quale si trova la entry del cluster attuale (il bootsector di Win2000 lo carica a 7E00h), legge la entry del cluster attuale e ottiene, quindi, il cluster successivo (oppure la fine del file, o il bad cluster marker).

Chiarisco con un esempio. Questa è una parte del primo settore della mia root directory (sul mio hard disk, è il settore numero 7924):

```
4E 4F 4E 41 4D 45 20 20 20 20 20 08 00 00 00 00  NONAME
00 00 00 00 00 00 5B 75 9C 2B 00 00 00 00 00 00        [u£+
4E 54 44 45 54 45 43 54 43 4F 4D 27 18 38 85 75  NTDETECTCOM'8àu
9C 2B 9C 2B 01 00 0F 6A 84 2B 02 00 A4 86 00 00  £+£+ ¤jä+ ñå
42 4F 4F 54 20 20 20 20 49 4E 49 26 18 7E 85 75  BOOT    INI&~àu
9C 2B 9C 2B 01 00 0F 5D 9A 2B 0B 00 C0 00 00 00  £+£+ ¤]Ü+ +
4E 54 4C 44 52 20 20 20 20 20 20 27 08 7F 85 75  NTLDR      'àu
9C 2B 9C 2B 01 00 0F 6A 84 2B 0C 00 B0 49 03 00  £+£+ ¤jä+ ¦I
```

Il bootsector di Win2000 carica questo settore a partire dall'indirizzo 7E00h, e ci cerca dentro la stringa "NTLDR   ". Quando ha trovato questa stringa, sa di essere arrivato alla entry del file ntldr, e quindi cerca il numero del cluster iniziale del file:

```
4E 54 4C 44 52 20 20 20 20 20 20 27 08 7F 85 75  NTLDR      'àu
9C 2B 9C 2B 01 00 0F 6A 84 2B 0C 00 B0 49 03 00  £+£+ ¤jä+ ¦I
            |---|             |---|
       High 16 bit of      Starting cluster
       cluster number
```

Il numero del cluster iniziale del file è quindi 65548. A partire da questo numero, il bootsector ricava i settori da leggere: in questi settori ci sta una prima parte di ntldr.

Dopo aver letto questi primi settori, il bootsector deve sapere il cluster successivo del file: nella tabella di allocazione dei file, viene cercata la entry corrispondente al cluster 65548. Il numero trovato in quella entry, se compreso tra 2 e 0FFFFFFF6h, costituisce il cluster successivo del file.
Il ciclo si ripete finchè il valore letto nella entry di un cluster è maggiore di 0FFFFFFF6h o minore di 2.

CHS / LBA

Vedremo che il bootsector di Win2000 carica in memoria un secondo settore. Per fare questo, naturalmente, deve sapere *dove* si trova questo settore.
Per indicare la posizione di un settore sul disco esistono due modi.

L' LBA (Logical Block Addressing) individua un settore attraverso un unico numero (a 32 bit). Questo numero parte dallo zero, ed arriva a (max_sector_number-1), dove max_sector_number è l'ultimo settore dell'hard disk. L' LBA è un "addressing mode" lineare e semplice da usare, *ma* è supportato in modo "nativo" solo a partire dagli hard disk ATA-2 (E-IDE). Nel caso degli hard disk precedenti bisogna usare il CHS.

Il CHS individua un settore attraverso 3 "coordinate": Cylinder, Head, Sector. Con questi tre valori, ogni settore è puntato in modo univoco. Il CHS è l' "addressing mode" del tradizionale int 13h (anche se poi sono state scritte estensioni dell'int 13h per supportare LBA).

Si capisce subito quanto siano necessarie delle routine per convertire un'espressione LBA in un'espressione CHS: infatti spesso è più comodo ragionare sempre in LBA, anche nel caso degli hard disk vecchi e dei floppy, e poi tradurre l'indirizzo LBA in indirizzo CHS. Questo è anche ciò che fa il bootsector di Win2000.
Le formule per la traduzione da LBA a CHS sono le seguenti:

```
1. Sector = (LBA mod SPT) + 1   ; SPT = sectors per track
2. Head = (LBA / SPT) mod HPC  ; HPC = head per cylinder
3. Cylinder = (LBA / SPT) / HPC
```

C'è anche la formula inversa:

```
LBA = (((Cylinder * HPC) + Head) * SPT) + Sector - 1
```

A noi (cioè, al bootsector di Win2000) servono le prime tre espressioni.
(Nota: un buon articolo sugli hard disk è quello di Thomas Kjoersen (vedi in fondo))

The bootsector (Fat32)

Ok, eccoci giunti alla parte più interessante del tutorial, ovvero l'analisi del codice. Prima di vedere il disassemblato, vi elenco brevemente quello che fa questo bootsector:

  1) Dopo aver skippato il BPB, inizializza i segmenti.
  2) Chiama l'int 13h, funzione 08h, per avere informazioni sul drive corrente.
  3) Calcola il valore (cylinders * heads_per_cylinder * sectors_per_track), che corrisponde al numero totale di settori presenti sull'hard disk. Questo valore viene memorizzato all'indirizzo [bp-8].
  4) Esegue dei controlli per verificare se alcune informazioni contenute nel BPB sono corrette.
  5) Legge un settore (il cui numero è ricavato dal BPB) chiamando una subroutine; memorizza questo settore all'indirizzo

8000h; salta al codice contenuto a 8000h. Nel caso del mio hard disk, il settore letto è il numero 75 (se avete installato Win2000 nella partizione C, probabilmente sarà così anche da voi).

Ecco invece quello che fa la subroutine per la lettura di un settore (i cui parametri sono: EAX = settore iniziale da leggere, in LBA; CX = numero di settori da leggere; BX = indirizzo in cui memorizzare i settori):

1) Confronta il settore iniziale da leggere (EAX) con il numero di settori totali dell'hard disk (per i commenti, vedere alla fine).
2) Legge il settore, utilizzando la funzione 42h o la funzione 02h dell'int 13h.

Consiglio di guardare il codice tenendo sempre come reference la Ralph Brown Interrupt List.
Ecco il codice commentato:

```
;------------------------ [Start here] ---------------------------------
;-----------------------------------------------------------------------
;Win2000 Bootsector (sector #1) reversed
;-----------------------------------------------------------------------

  org 7c00h

  jmp skipBPB

Int13Flag db 90h
Id  db 'MSWIN4.1',0
BytesPerSector dw 200h
SectorsPerCluster db 8
ReservedSector dw 20h
NumberOfFATs db 2
RootEntries dw 0
TotalSectors dw 0  ;grandezza della partizione in settori
MediaDescriptor db 0F8h  ;0xF8 per gli hard disk
SectorsPerFAT dw 0  ;0 nel caso di Fat32
SectorsPerTrack dw 63
Heads  dw 255
HiddenSectors dd 63
BigTotalSectors dd 8257347
BigSectorsPerFat dd 8056
ExtFlags dw 0
FS_Version dw 0
RootDirStrtClus dd 2
FSInfoSec dw 1
BkUpBootSec dw 6
Reserved dw 6 dup (0)
Drive  db 80h
HeadTemp db 0
Signature db 29h
SerialNumber dw 17ECh
VolumeLabel db 'NONAME     '
FileSystemID db 'FAT32   '

skipBPB:
  xor cx,cx
  mov ss,cx  ;Inizializza lo stack
  mov sp,7BF4h ;mette SP da qualche parte dove
     ;non può fare danni :)

  mov es,cx  ;Inizializza gli altri segmenti
  mov ds,cx
  mov bp,7C00h ;BP verrà usato per puntare al
     ;bpb.
  mov [bp+Int13Flag],cl ;Per ora, setta il flag
      ;dell'int13 extension a 0.
  mov dl,[bp+Drive] ;Prende il drive number (80h)
  mov ah,8
  int 13h  ;Ottiene alcune informazioni sul
     ;drive corrente
  jnc allRight ;Se non ci sono stati errori, salta
     ;ad allRight
  mov cx,0FFFFh ;Assume dei valori massimi??
  mov dh,cl

allRight:
 ;Nella parte di codice successiva, rielabora i valori ottenuti
 ;con la chiamata precedente all'int 13h, per ottenere il numero
 ;di settori totali.
 ;  TotalSectors = (MaxHead+1) * MaxSector * (MaxCylinder+1)
 ;Memorizza TotalSectors in [bp-8]

  movzx eax,dh  ;EAX = maximum head number
  inc ax  ;EAX = actual number of heads
     ;(incrementa AX perchè le "head"
     ;vengono contate a partire da 0).
  movzx edx,cl  ;EDX = maximum sector number
     ;(qui non ha bisogno di incrementare
     ;EDX, perchè i settori vengono contati
     ;a partire da 1 e non da 0).
  and dl,3Fh  ;isola i bit 0-5 (perchè gli altri bit
     ;si riferiscono al cylinder number).
  mul dx  ;EAX = sectors * heads
  xchg cl,ch  ;Qui ottiene in CX il massimo numero
  shr ch,6  ;di cylinder.
  inc cx  ;CX = actual number of cylinders
  movzx ecx,cx
```

```
  mul ecx  ;EAX = sectors * heads * cylinders
     ;(EAX = numero totale di settori)
  mov [bp-8],eax


  cmp word ptr [bp+SectorsPerFAT],0 ;esegue dei controlli
  jnz noNTLDR
  cmp word ptr [bp+FS_version],0
  ja noNTLDR
  mov eax, [bp+HiddenSectors]
  add eax, 0Ch ;calcola quale è il secondo settore
     ;da caricare (nel mio caso, vuole
     ;caricare il settore n° 75).
  mov bx,8000h ;lo carica all'offset 8000h
  mov cx,1  ;legge 1 solo settore
  call readSector ;carica il settore
  jmp near ptr 8000h ;salta al codice del settore caricato

;------------------------------------------------------------------------

discError:
  mov al,byte ptr ds:[DiscErrorPtr]
     ;il byte a [DiscErrorPrt] contiene
     ;il low-byte dell'offset della stringa
     ;"Errore Disco"

showMsg:
  mov ah,7Dh  ;quando arriviamo qua, abbiamo solo
     ;il low-byte dell'offset della stringa
     ;da scrivere, quindi dobbiamo calcolare
     ;l'offset intero (mettendo 7Dh negli
     ;8 bit più significativi).
  mov si,ax

showMsgLoop:    ;qui viene mostrata la stringa di
     ;errore
  lodsb
  test al,al  ;se AL = 0, siamo arrivati alla fine
     ;della stringa di reboot
  jz reboot
  cmp al,0FFh  ;se AL = 0FFh, siamo arrivati alla fine
     ;di una stringa di errore
  jz pressKey ;(e quindi visualizziamo la stringa
     ;di reboot)
  mov ah,0Eh  ;visualizza un carattere
  mov bx,7
  int 10h
  jmp showMsgLoop

pressKey:
  mov al,byte ptr ds:[PressKeyPtr]
     ;pointer alla stringa
     ;"Premere un tasto per riavviare"
  jmp showMsg

noNTLDR:
  mov al,byte ptr ds:[NoNTLDRPtr]
     ;pointer alla stringa
     ;NTLDR mancante"
  jmp showMsg

reboot:
  cbw
  int 16h  ;attende la pressione di un tasto
  int 19h  ;esegue un reboot

;------------------------------------------------------------------------

readSector:
  pushad   ;salviamo un po' tutto... :)

  cmp eax, [bp-8] ;in [bp-8] c'era il numero totale di
     ;settori presenti sull'hard disk. Ora
     ;io mi domando, come fa EAX ad essere
     ;maggiore di questo valore qui??
     ;Questo jb, almeno sul mio computer,
     ;è *sempre* preso (salta sempre a 7D34)
  jb LBAtoCHS ;Per fare la controprova, ho modificato
     ;il bootsector in modo che si
     ;interrompesse se il jb non era preso
     ;(ho messo un "mov al,0" / "int 16h" /
     ;"int 19h") ... risultato, Win2000 ha
     ;bootato come se niente fosse.

;****************** [Parte non eseguita ] ******************
  push large 0
  push eax
  push es
  push bx
  push large 10010h
  cmp byte ptr [bp+2],0
  jnz 7D1F
  mov ah,41h
```

```
  mov bx,55AAh
  mov dl,[bp+40h]
  int 13h
  jb near ptr 7D28+1
  cmp bx,0AA55h
  jnz near ptr 7D28+1
  test cl,1
  jz near ptr 7D28+1
  inc byte ptr [bp+2]

7D1F:
  mov ah,42h
  mov dl, [bp+40h]
  mov si,sp
  int 13h

7D28:
  mov al,0F9h
  pop eax
  pop eax
  pop eax
  pop eax
  jmp done
;************************************************************

LBAtoCHS:
  xor edx,edx
  movzx ecx, word ptr [bp+SectorsPerTrack]
  div ecx  ;EAX = sector # / sectors_per_track

  inc dl  ;In DL a questo punto abbiamo il
     ;numero del settore da leggere
     ;In pratica abbiamo DL=(EAX mod ECX)+1
     ;(confronta con le formule LBA/CHS)

  mov cl,dl  ;Sistemiamo già il sector # in CL
     ;per l'int 13h.

  mov edx,eax
  shr edx,16  ;Mette in DX:AX il valore che prima
     ;era contenuto in EAX (questo è
     ;necessario per dividere un numero a
     ;32 bit per una word)

  div word ptr [bp+1Ah] ;Divide DX:AX per l' head #
     ;Dopo la div, abbiamo:
     ;DL = numero della head da leggere
     ;AX = numero del cylinder da leggere

  xchg dl,dh  ;Qui sistemiamo un po' le cose nei
  mov dl, [bp+40h] ;registri giusti per l'int 13h.
  mov ch,al  ;(consiglio di consultare la
  shl ah,6  ;Ralph Brown's int list)
  or cl,ah
  mov ax,0201h ;Chiamiamo l'int 13h per leggere
  int 13h  ;un settore.

done:
  popad
  jb discError ;Se ci sono stati errori, salta
  add bx,200h  ;Altrimenti si porta a 512 byte dopo
  inc eax  ;incrementa EAX (nel caso bisognasse
     ;leggere il settore successivo)
  dec cx  ;Decrementa CX (che conteneva il
     ;numero di settori da leggere)
  jnz readSector
  ret

;---------------------------------------------------------------------------

FileName db 'NTLDR      ' ;Stringa che verrà usata successivamente
  db 49 dup (0)

NoNTLDRString db 0Dh,0Ah,'NTLDR mancante',0FFh
DiscErrorString db 0Dh,0Ah,'Errore disco',0FFh
PressKeyString db 0Dh,0Ah,'Premere un tasto per riavviare',0Dh,0Ah,0

  db 11 dup (0)

NoNTLDRPtr db 0ACh ;7DACh è l'indirizzo di NoNTLDRString
DiscErrorPtr db 0BDh ;7DBDh è l'indirizzo di DiscErrorString
PressKeyPtr db 0CCh ;7DCCh è l'indirizzo di PressKeyString

  dw 0
  dw 0AA55h  ;boot marker

;------------------------ [End here] -----------------------------------


Mie considerazioni
------------------

Come ho già scritto nei commenti, c'è tutta una parte in "readSector" che, almeno nel mio caso, non viene mai eseguita. Ho
```

fatto la controprova, mettendo sotto il "jb LBAtoCHS" le istruzioni:

```
  mov ah,0
 int 16h
 int 19h
```

che avrebbero dovuto aspettare un tasto e poi rebootare. Invece Win2000 si è caricato in modo normale (ha bootato come se niente fosse), e questo significa che (giustamente) il jb è *sempre preso* (salta sempre a LBAtoCHS). Come spiegarsi questa cosa? Ho fatto delle ipotesi:
  1) il jb potrebbe *non* essere preso quando nel BPB ci sono dei valori sbagliati. Quindi verrebbe usata la funzione 42h dell'int 13h per leggere in modo assoluto dal disco (cioè, senza fare la conversione LBA --> CHS, poichè questa conversione dipende dai valori contenuti nel BPB ... e se questi valori sono sbagliati, non leggiamo di certo il settore giusto). Ma mi sembra un'eventualità abbastanza remota, visto che i valori del BPB sono calcolati al momento del format, e poi non vengono più cambiati.
  2) la Microsoft ha fatto questo bootsector riutilizzando codice vecchio, senza togliere parti inutili :PPP
Se è vera questa seconda ipotesi, allora un'altra parte inutile di questo bootsector è quella che riguarda l' "Int13Flag": questo byte dovrebbe segnalare (nel caso sia uguale a 1) che l'hard disk supporta l'estensione dell'int 13h (l'estensione che permette di leggere i settori in LBA con la funzione 42h), ma dal momento che poi il bootsector utilizza solo la funzione 02h (e non la 42h) dell'int 13h, l' "Int13Flag" non serve più.
(Questo flag compare già nel bootsector di Win95, e in quel caso però viene utilizzato...quindi c'è da pensare davvero ad un riutilizzo di vecchio codice).

Avendo provato solo sul mio computer, comunque, non posso concludere niente...magari col vostro hard disk la situazione è diversa.

Il secondo settore

Ok, abbiamo visto che il bootsector carica un altro settore e salta al codice in esso contenuto. Però, a questo punto, siamo ancora lontani dall'aver caricato ed eseguito ntldr. Il compito di caricare ed eseguire ntldr è affidato, come vedremo, al settore che viene caricato a 0000:8000h dal bootsector.
I passi principali di questo secondo settore sono:
  1) calcolare quale è il primo settore della root directory
  2) caricare il settore della root dir
  3) cercare la entry del file "NTLDR" all'intero della root directory
  4) se bisogna passare ad un nuovo settore della root dir, ricalcolare il nuovo settore e saltare al punto 2
  5) una volta trovata la entry, trovare il primo cluster del file
  6) caricare, un cluster alla volta, il file
  7) saltare all'indirizzo a cui si era caricato il file.

```
;------------------------ [Start here] ---------------------------------
;-----------------------------------------------------------------------
;Win2000 Bootsector (sector #2) reversed
;-----------------------------------------------------------------------

  org 8000h

 ;Nel codice successivo, viene calcolato il primo settore della
 ;root directory, e viene memorizzato il numero di questo settore
 ;a [bp-4].

  movzx eax,byte ptr [bp+NumberOfFATs]
  mov ecx, [bp+BigSectorsPerFat]
  mul ecx
  add eax, [bp+HiddenSectors]
  movzx edx, word ptr [bp+ReservedSectors]
  add eax,edx   ;EAX = primo settore della root directory
  mov [bp-4],eax ;memorizza questo (importante)
     ;valore a [bp-4]

 ;A [bp-12] avremo ogni volta il cluster attualmente in uso. La prima
 ;volta, carica 0FFFFFFFFh per segnalare che questo valore non è ancora
 ;stato inizializzato.

  mov dword ptr [bp-12],0FFFFFFFFh ;
  mov eax,[bp+RootDirStrtClus] ;EAX = numero del primo
       ;cluster della root directory

  cmp eax,2  ;Controlla che sia un cluster valido
  jb near ptr noNTLDR ;(i cluster validi e che contengono
     ;dati sono compresi tra 2 e 0FFFFFFF6h)
  cmp     eax, 0FFFFFF8h
  jnb     near ptr noNTLDR

 ;Ciò che viene fatto nel codice successivo è "sistemare" il valore del
 ;cluster (sottraendo 2), moltiplicarlo per il numero di settori per
 ;cluster, e aggiungere il numero del primo settore della root
 ;directory.
 ;In questo modo si ottiene il settore della root directory da caricare
 ;(per cercarci dentro la entry del file ntldr).

calculateRootSector:
  push eax
  sub eax,2
  movzx ebx,byte ptr [bp+SectorsPerCluster]
  mov si,bx  ;tiene in SI il valore SectorPerCluster
     ;perchè ci serve dopo
  mul ebx
  add eax, [bp-4]

loadRootSector:
  mov bx,8200h ;carica il settore della root dir a 8200h
```

```
  mov di,bx
  mov cx,1
  call near ptr readSector

searchNTLDR:
  cmp [di],ch  ;se il byte è zero, significa che
     ;siamo arrivati alla fine della
     ;directory (senza trovare il file
     ;cercato), quindi salta
  jz NTLDRNotFound ;a NTLDRNotFound

  mov cl,11  ;Cerca la stringa "NTLDR     "
     ;composta da 11 caratteri
  push si  ;In pratica qui cerca la entry del
  mov si,FileName ;file ntldr, facendo passare tutti
  repe cmpsb  ;i nomi dei file nella root dir
  pop si
  jz NTLDRFound ;Se l'ha trovata, salta
  add di,cx  ;Altrimenti, sistema DI per puntare
  add di,15h  ;alla entry del file successivo.
  cmp di,bx  ;Controlla se c'è bisogno di caricare
     ;un nuovo settore della root dir
  jb searchNTLDR
  dec si  ;SI contiene il numero di settori per
     ;cluster. Se SI==0, vuol dire che
     ;dobbiamo ricalcolare il cluster
     ;successivo, mentre se SI!=0, ci basta
     ;caricare il settore successivo.
  jnz loadRootSector

  pop eax
  call getNextCluster
  jb calculateRootSector ;se non ci sono stati errori,
     ;e se non siamo alla fine della dir,
     ;calcola il root sector successivo
     ;e cerca la stringa "NTLDR      ".

NTLDRNotFound:
  add sp,4  ;sistema lo stack (avevamo pushato eax)
  jmp near ptr noNTLDR ;visualizza il msg di errore

;--------------------------------------------------

loadingSegment dw 2000h

NTLDRFound:
  add esp,4  ;sistema lo stack (avevamo pushato eax)
  mov si,[di+9] ;DI+9 = 16 bit più significativi del
     ;FirstFileCluster, ovvero il numero
     ;del primo cluster del file

  mov di,[di+15] ;DI+15 = 16 bit meno significativi del
     ;FirstFileCluster
  mov ax,si
  shl eax,16
  mov ax,di  ;memorizza in EAX il numero del
     ;primo cluster

  cmp eax,2  ;Controlla che sia un cluster valido
  jb near ptr NoNTLDR
  cmp eax,0FFFFFFF8h
  jnb near ptr NoNTLDR

loadCluster:
  push eax  ;Usa le stesse formule di prima per
  sub eax,2  ;ottenere il settore del cluster
  movzx ecx,byte ptr [bp+SectorsPerCluster] ;da caricare
  mul ecx
  add eax,[bp-4]
  mov bx,0
  push es  ;Carica il settore a partire da
     ;2000h:0000h
  mov es, word ptr ds:[loadingSegment]
  call near ptr readSector ;Carica n settori
     ;(con n = SectorsPerCluster).
     ;Quindi, con questa chiamata a
     ;readSector, carica un intero cluster
     ;del file NTLDR.
  pop es
  pop eax
  shr bx,4  ;Aggiunge 32 a [loadingSegment]
     ;(ha diviso 512 per 16 perchè andiamo
     ;a modificare il valore di un segmento
     ;e non di un offset)
  add word ptr ds:[loadingSegment],bx
  call getNextCluster
  jz jmpToNTLDR ;Se siamo alla fine del file, esegue
     ;NTLDR.
  jnz loadCluster

jmpToNTLDR:
  mov dl,[bp+Drive]
  jmp     far ptr 2000h:0
```

```
;-------------------------------------------------

getNextCluster:
;Questa routine ottiene il cluster successivo di un file (nel nostro caso,
;calcola il cluster successivo della root directory).
;
;Input:
; EAX ---> cluster attuale
;
;Output:
; EAX ---> cluster successivo
; ZF = 1 se siamo alla fine del file
;


  shl eax,2  ;moltiplica il cluster # per 4, perchè
     ;ogni entry nella FAT è di 32 bit
     ;(4 byte)
  call calculateNextCluster
  mov eax, dword ptr es:[bx+di]  ;ottiene il numero del
         ;cluster successivo
  and     eax, 0FFFFFFFh
  cmp eax, 0FFFFFF8h
  ret

;-------------------------------------------------

calculateNextCluster:
;Questa routine calcola la posizione, all'interno di un settore FAT, della
;entry del cluster specificato. Se è necessario, questa routine carica in
;memoria un nuovo settore della FAT.
;
;Input:
; EAX ---> (cluster attuale)*4
;
;Output:
; BX + DI ---> pointer alla entry del cluster successivo
;

  mov di,7E00h ;assume che il settore FAT sia stato
     ;caricato a 7E00h

 ;Nel prossimo pezzo di codice, si vuole calcolare dove sta la entry
 ;del cluster successivo (cioè, del cluster che vogliamo trovare).
 ;Per far questo, vengono usate queste formule:
 ; NextClusterSector = ClusterNumber / BytesPerSector
 ; (NextClusterSector rappresenta il settore FAT in cui si
 ; trova la entry che vogliamo trovare)
 ; NextClusterEntry = ClusterNumber mod BytesPerSector
 ; (NextClusterEntry rappresenta la posizione della entry
 ; all'interno del settore FAT).

  movzx ecx,word ptr [bp+BytesPerSector]
  xor edx,edx
  div ecx  ;EAX = cluster number / bytes per sector
     ;dopo la div, abbiamo:
     ;EAX = settore della FAT all'interno del
     ;quale c'è l'entry del cluster
     ;successivo (NextClusterSector)
     ;DX = offset, all'interno del
     ;settore FAT attuale, del valore del
     ;cluster che vogliamo (NextClusterEntry)

  cmp eax,[bp-12] ;Controlla se NextClusterSector è
     ;lo stesso settore FAT precedente.
     ;Se è lo stesso, non ha bisogno di
     ;ricaricarlo
  je skipReload

  mov [bp-12],eax ;altrimenti memorizza il settore FAT
     ;attuale

 ;Nel codice successivo, NextClusterSector verrà "sistemato"
 ;in modo da poter essere letto attraverso la routine readSector.
 ;Nel caso che la ActiveFat non sia la prima, viene anche aggiunta
 ;la posizione della ActiveFat.
 ;In pratica, abbiamo:
 ;
 ; NextClusterSector += (HiddenSectors + ReservedSectors);
 ; if (ActiveFat!=0)
 ; {
 ;  NextClusterSector += (BigSectorsPerFat*ActiveFat);
 ; }
 ;
 ;La prima "sistemazione" è necessaria perchè noi leggiamo il settore
 ;a partire dal primo settore dell'hard disk, non dal primo settore
 ;della partizione attuale.

  add eax,[bp+HiddenSectors]
  movzx ecx,word ptr [bp+ReservedSectors]
  add eax,ecx  ;Qui ha sistemato NextClusterSector
     ;aggiungendo HiddenSectors e
```

```
    ;ReservedSectors
  movzx   ebx, word ptr [bp+ExtFlags]
  and bx,0Fh  ;Isola i primi 4 bit dell'ExtFlags
    ;Questi 4 bit contengono il valore
    ;della ActiveFat
  jz ActiveFatZero ;Se la ActiveFat è 0 (la prima),
    ;allora non deve sistemare il
    ;NextClusterSector
  cmp     bl, [bp+NumberOfFats]
  jnb     near ptr noNTLDR
  push    dx
  mov     ecx, eax
  mov     eax, [bp+BigSectorsPerFat]
  mul     ebx
  add     eax, ecx ;Ha sistemato NextClusterSector
  pop     dx

ActiveFatZero:
  push    dx
  mov     bx, di  ;Legge un settore e lo memorizza a
  mov     cx, 1  ;7E00h
  call    near ptr readSector
  pop     dx

skipReload:
  mov     bx, dx  ;Ritorna NextClusterEntry in BX
  retn

;--------------------------------------------------

  db 182 dup (0)

  dw 0AA55h

;------------------------ [End here] -----------------------------------
```

Mie conclusioni - parte 2

Ok, questo secondo settore non è scritto male. Mi sembra che usi un codice abbastanza agile, soprattutto nelle routine per calcolare il cluster successivo.
Ho confrontato questi due settori con il bootsector del floppy (Fat12): mi sono accorto che il bootsector Fat12 carica ed esegue ntldr usando un solo settore di codice (cioè il settore di boot stesso, e nient'altro). Sarebbe interessante provare a riscrivere il bootsector Fat32 in modo da caricare ntldr, come nel caso Fat12, con un solo settore di codice.

Referenze

- Thomas Kjoersen, "File Allocation Table - How It Seems To Work" (http://home.no.net/tkos)
- Thomas Kjoersen, "Harddisks, Partitioning and Booting" (http://home.no.net/tkos)
- Mr. X, "Heroic attempt to disassemble the Windows 95 Boot Sector" (sito?)
- Ralph Brown, "Interrupt List"

Saluti

Un saluto a tutta la gentaglia di #asm e #crack-it sulla rete Azzurra!
Un ringraziamento particolare a Quake2^Am, che mi ha proposto di reversare questo bootsector, e a _d31m0s_ per le informazioni sui file di sistema di Win2000, in particolare su NTLDR.
Ringrazio infine tutte le persone che mi hanno dato consigli o segnalato errori.

ciaoo!

Albe
-albe-@libero.it

Disclaimer
