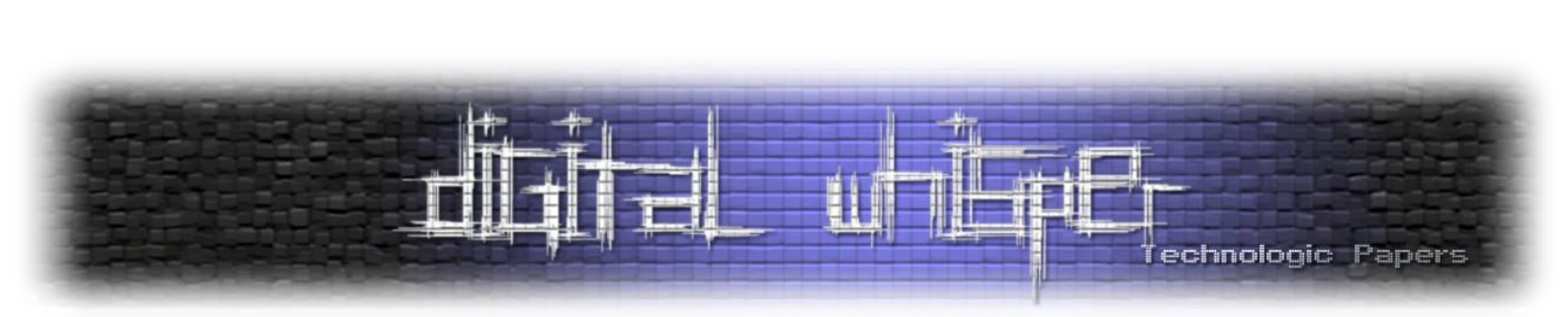

הנדסה-לאחור: שרשרת העלייה של Windows 7 חלק שני - VBR

מאת 0x3d5157636b525761

רקע

בחלק הקודם דיברנו על ה-MBR, על הטעינה שלו על ידי ה-BIOS ועל כל הפעולות שהוא ביצע והכין לפני העברת האחריות ל-VBR. בחלק זה נתמקד ב-VBR. ה-VBR (קיצור של Volume Boot Record) הוא הסקטור הראשון של המחיצה הלוגית. כאמור, הוא נטען על ידי ה-MBR, שהוא הסקטור הראשון של הדיסק הפיזי שעלה.

במהלך המדריך אני אעבוד על Windows 7 SP1, x64 ולמעשה אמשיך מהמצב בו הפסקנו לפני כן.

הערת צד: כל ניסוי שהקורא מחליט לבצע כתוצאה מקריאת מאמר זה - על אחריותו בלבד!

חשיפת ה-VBR

בניגוד לקריאת הדיסק הפיזי (PhysicalDrive0), המצב הרבה יותר קל. בהנחה שהכונן הראשי הוא C:

```
ActivePython 2.7.6.9 (ActiveState Software Inc.) based on
Python 2.7.6 (default, Feb 27 2014, 14:13:40) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> vbr = open('\\\\.\\C:', 'rb').read(512)
>>> open('C:\\vbr.bin', 'wb').write(vbr)
>>>
```

ניתוח ראשוני עם IDA

נפתח את הקובץ ב-IDA. כמקודם, IDA לא יודעת לנתח את הקובץ ישירות כי מדובר בקוד טהור ולא בפורמט מוגדר ולכן ננתח את הקובץ כקוד 16 ביט. כמובן, נבצע rebase לכתובת 7C00 (אליה נטען ה-VBR על ידי ה-MBR) ונלחץ על C בהתחלה כדי לנתח כקוד. כמקודם, ה-VBR מסתיים ב-0xAA55, כצפוי.

חלק א': ישר קופצים?

הדבר הראשון שנראה זה קפיצה מעל אזור די גדול אל - 0x7C54... מדוע? יש תשובה טובה לכך - אנחנו כבר לא בארץ ה-MBR הכיפי - מדובר כאן במחיצה, ולכן היא צריכה להכיל מערכת קבצים! כל מערכת קבצים מתחילה ב-header כלשהו שמתאר פרמטרים שונים בה, וכמובן - Windows עובד על NTFS. מכאן, כדאי להכנס למדריך כלשהו - אני נכנסתי אל http://ntfs.com/ntfs-partition-boot-sector.htm שמסביר יפה מאד איך ה-layout הבסיסי נראה. כמובן, לא ניכנס לכאן אל NTFS, אבל זה יספיק לענייננו.

הערת צד: המונח BPB שמופיע בהמון מקומות נקרא גם BIOS PARAMETER BLOCK, והוא מתאר את מערכת הקבצים. הוא שומש באופן מסורתי ב-FAT12 עבור floppies ולאחר מכן ב-FAT16, FAT32 ועכשיו גם ב-NTFS. יש לציין שישנן "גרסאות" ל-BPB, כאשר כל גרסא מרחיבה את הקודמת לה. זה של NTFS גדול במיוחד, בגודל 0x54 בתים!

לאחר שאנחנו מבינים מדוע קיימת קפיצה, אפשר לראות מה קורה לאחר מכן ב-0x7C54. אפשר לראות שהעניינים נראים די דומים (לפחות בהתחלה) למה שהיה עם ה-MBR:

```
seg000:7C54 ; ---------------------------------------------------------------------------
seg000:7C54
seg000:7C54 lblMain:                                ; CODE XREF: seg000:lblStart↑j
seg000:7C54                 cli
seg000:7C55 ;
seg000:7C55 ; Build stack
seg000:7C55 ;
seg000:7C55
seg000:7C55                 xor     ax, ax
seg000:7C57                 mov     ss, ax
seg000:7C59                 mov     sp, 7C00h
seg000:7C5C                 sti
seg000:7C5D ;
seg000:7C5D ; Make CS=DS=0x07C0
seg000:7C5D ;
seg000:7C5D                 push    7C0h
seg000:7C60                 pop     ds
seg000:7C61                 assume ds:nothing
seg000:7C61                 push    ds
seg000:7C62                 push    (offset lblStartParsingNTFS - offset lblStart)
seg000:7C65                 retf
seg000:7C66
seg000:7C66 lblStartParsingNTFS:                    ; DATA XREF: seg000:7C62↑o
seg000:7C66                 mov     ds:0Eh, dl
seg000:7C6A ;
seg000:7C6A ; Make sure the OEM is "NTFS"
seg000:7C6A ;
seg000:7C6A                 cmp     dword ptr ds:3, 'SFTN'
seg000:7C73                 jnz     short lblPrintErrorAndHangCaller
seg000:7C75 ;
seg000:7C75 ; Check for LBA mode reading
seg000:7C75 ;
seg000:7C75                 mov     ah, 41h ; 'A'
seg000:7C77                 mov     bx, 55AAh
seg000:7C7A                 int     13h             ; DISK - Check for INT 13h Extensions
seg000:7C7A                                         ; BX = 55AAh, DL = drive number
seg000:7C7A                                         ; Return: CF set if not supported
seg000:7C7A                                         ; AH = extensions version
seg000:7C7A                                         ; BX = AA55h
seg000:7C7A                                         ; CX = Interface support bit map
seg000:7C7C                 jb      short lblPrintErrorAndHangCaller
seg000:7C7E                 cmp     bx, 0AA55h
seg000:7C82                 jnz     short lblPrintErrorAndHangCaller
seg000:7C84                 test    cx, 1
seg000:7C88                 jnz     short loc_7C8D
seg000:7C8A
seg000:7C8A lblPrintErrorAndHangCaller:             ; CODE XREF: seg000:7C73↑j
seg000:7C8A                                         ; seg000:7C7C↑j ...
seg000:7C8A                 jmp     lblPrintErrorAndHang
```

ניתוח:

- בניית המחסנית: דומה מאד למה שהתבצע ב-MBR - איפוס SS והשמת 0x7C00 ב-SP. מכיוון שהמחסנית גדלה לכיוון כתובות נמוכות, אנחנו במצב טוב.
- שינוי CS ו-DS ל-0x07C0: זה למעשה יתן לנו יכולת עבודה טובה וקלה: מעתה כל אוגר הוא פשוט offset מבסיס התוכנית. הטכניקה של push-push-retf זהה לטכניקה ששומשה ב-MBR.
- על 0x0E שומרים את DL (ה-Drive Number). מי שפרסר כבר את NTFS יודע ש-16 ביט שם מכילים reserved sectors (ת'כלס זה 0).
- בשלב הבא מוודאים שה- OEM הוא NTFS (זה נמצא בבית השלישי במשך 8 בתים), ומוודאים שניתן לבצע קריאות LBA (בדומה למה שהיה ב-MBR). נקודה משעשעת: ראינו ב-MBR שיש תמיכה גם אם אין עבודה עם LBA, אבל כאן רואים ש-LBA הוא פיצ'ר חובה! לא מצאתי תיעוד לכך, אבל אני יכול לחשוב על שתי סיבות מרכזיות:
  - ✓ תאימות לאחור: ה-MBR יתאים גם למערכות הפעלה ישנות יותר, עם VBR שמאפשר CHS.
  - ✓ תמיכה במערכות הפעלה שונות: אין סיבה שה- MBR לא יטען VBR של לינוקס (EXT) למשל.
- בכל מקרה של כשלון קופצים אל 0x7C8A. איך אני יודע שזוהי הדפסה של שגיאה? התשובה היא שצריך להתסכל קצת מה קורה: ב-0x7C8A קופצים באופן בלתי-מותנה אל 0x7D6A ושם מבצעים קריאה פעמיים אל פונקציה שמדפיסה שגיאות (TTY, כמו שהיה ב-MBR) ולאחר מכן מבצעת HLT וקפיצה אינסופית. הנה הקוד החל מ-0x7D6A ועד הסוף:

```
seg000:7D6A
seg000:7D6A lblPrintErrorAndHang:                   ; CODE XREF: seg000:lblPrintErrorAndHangCaller↑j
seg000:7D6A                                         ; sub_7D1D+33↑j
seg000:7D6A                 mov     al, ds:1F8h
seg000:7D6D                 call    PrintMessage
seg000:7D70                 mov     al, ds:1FBh
seg000:7D73                 call    PrintMessage
seg000:7D76
seg000:7D76 lblHang:                                ; CODE XREF: seg000:7D77↓j
seg000:7D76                 hlt
seg000:7D76 sub_7D1D        endp
seg000:7D76
seg000:7D77 ; ---------------------------------------------------------------------------
seg000:7D77                 jmp     short lblHang
seg000:7D79
seg000:7D79 ; =============== S U B R O U T I N E =======================================
seg000:7D79
seg000:7D79
seg000:7D79 PrintMessage    proc near               ; CODE XREF: sub_7D1D+50↑p
seg000:7D79                                         ; sub_7D1D+56↑p
seg000:7D79                 mov     ah, 1
seg000:7D7B                 mov     si, ax
seg000:7D7D ;
seg000:7D7D ; Handle the next character
seg000:7D7D ;
seg000:7D7D
seg000:7D7D lblNextChar:                            ; CODE XREF: PrintMessage+10↓j
seg000:7D7D                 lodsb
seg000:7D7E                 cmp     al, 0
seg000:7D80                 jz      short lblFinishPrinting
seg000:7D82 ;
seg000:7D82 ; Perform TTY printing and handle the next character
seg000:7D82 ;
seg000:7D82                 mov     ah, 0Eh
seg000:7D84                 mov     bx, 7
seg000:7D87                 int     10h             ; - VIDEO - WRITE CHARACTER AND ADVANCE CURSOR (TTY WRITE)
seg000:7D87                                         ; AL = character, BH = display page (alpha modes)
seg000:7D87                                         ; BL = foreground color (graphics modes)
seg000:7D89                 jmp     short lblNextChar
seg000:7D8B ; ---------------------------------------------------------------------------
seg000:7D8B
seg000:7D8B lblFinishPrinting:                      ; CODE XREF: PrintMessage+7↑j
seg000:7D8B                 retn
seg000:7D8B PrintMessage    endp
seg000:7D8B
seg000:7D8B ; ---------------------------------------------------------------------------
```

- את הודעות השגיאה אני יודע להסיק מתוך ה-0x7D6A למשל: מכיוון ש-DS הוא 0x07C0 אז DS:1F8 הוא למעשה 0x7DF8. הבית שמופיע שם הוא 0x8C, וניתן לראות שבתחילת הפונקציה PrintMessage שמים בתוך AH את הערך 1, מה שאומר שכל AX יהיה 0x018C ולכן SI יהיה גם ערך זה. כמובן שאנחנו בסגמנט 0x07C0 ולכן ES:SI הוא 0x7D8C ושם כתוב ב-ANSI את המשפט המעצבן A disk read error occurred. לכן, PrintMessage מצפה שאוגר AL יחזיק את ה-offset של ההודעה להדפסה, בעוד 0x7DF8 משמשת כטבלת offset-ים עבור הודעות השגיאה.

חלק ב': ההכנות לקראת הטעינה הבאה!

קדימה לחלק הבא:

```
seg000:7C8D
seg000:7C8D lblGetDriveParams:                      ; CODE XREF: seg000:7C88↑j
seg000:7C8D                 push    ds
seg000:7C8E ;
seg000:7C8E ; Make room for the buffer on the stack
seg000:7C8E ; Size of buffer = 0x1A
seg000:7C8E ;
seg000:7C8E                 sub     sp, 18h
seg000:7C91                 push    1Ah             ; 0x1A because it's WORD more than 0x18 + the way PUSH works
seg000:7C94 ;
seg000:7C94 ; Function 48 (get drive paramters)
seg000:7C94 ; DL = drive number
seg000:7C94 ; DS:SI = buffer to fill
seg000:7C94 ;
seg000:7C94                 mov     ah, 48h ; 'H'
seg000:7C96                 mov     dl, ds:0Eh      ; DL = Drive number from [0x0E]
seg000:7C9A                 mov     si, sp
seg000:7C9C                 push    ss
seg000:7C9D                 pop     ds
seg000:7C9E                 assume ds:nothing
seg000:7C9E                 int     13h             ; DISK - IBM/MS Extension - GET DRIVE PARAMETERS (DL - drive, DS:SI - buffer)
seg000:7CA0 ;
seg000:7CA0 ; Clear buffer from stack
seg000:7CA0 ;
seg000:7CA0                 lahf
seg000:7CA1                 add     sp, 18h
seg000:7CA4                 sahf
seg000:7CA5                 pop     ax              ; Bytes per sector
seg000:7CA6                 pop     ds              ; Restore DS
```

ניתוח:

- בקטע קוד זה קוראים ל-int13 (דיסק) עם פונקציה מספר 0x48. ניתן לקרוא תיעוד מלא ב-RBIL, אבל בכל מקרה מה שחשוב לדעת זה שמספר הכונן (drive number) נמצא ב-DL, הבאפר שיתמלא נמצא על DS:SI וגודלו בבתים נשמר ב-WORD הראשון שלו (זה ה-Push 0x1A).

- **נקודה עדינה**: מדוע עושים SUB SP על 0x18? התשובה היא ש-SI יקבל את ערכו של SP. תזכורת חשובה: SP מצביע על ראש המחסנית, על הכתובת האחרונה שבשימוש (inclusive). ה-PUSH לאחר ה-SUB דוחף את הגודל שעליו דיברנו, ולכן בסך הכל הורדנו את SP ב-0x1A בתים (ישנם גדלים נוספים הנתמכים, זו הדרך שבה הפסיקה מבצעת versioning). אגב, כל הסיפור עובד כי המחסנית גדלה לכיוון כתובות נמוכות -- אם לא, היינו צריכים לבצע push 0x1A לפני פעולת ה-SUB.

- נשים לב שמנקים רק 0x18 ואז מבצעים POP AX. במבנה שחוזר ניתן לראות שזה בדיוק ה- bytes per sector. הנה המבנה המלא (כאשר הוא בגודל 0x1A):

Offset (HEX)	Description	Size (bytes)
0x0000	Size of buffer (0x1A)	2
0x0002	Information flags	2
0x0004	Number of physical cylinders on drive	2
0x0008	Number of physical heads on drive	2
0x000C	Number of physical sectors on drive	2
0x0010	Number of total sectors on drive	8
0x0018	Bytes per sector	2

אז למעשה מה שביצענו הוא לקרוא ל-GetDriveParams על DL, וממנו לשלוף את Bytes per sector. כמובן שיש צורך לבצע בדיקות שהקריאה הצליחה וכו', וזה בדיוק מה שיתבצע בהמשך הקוד:

```
000:7CA7 ;
000:7CA7 ; Check for failures and mismatches:
000:7CA7 ;    1. INT13 failure
000:7CA7 ;    2. Bytes per sector failure
000:7CA7 ;
000:7CA7                jb      short lblPrintErrorAndHangCaller
000:7CA9                cmp     ax, ds:0Bh      ; [0x0B] is exactly bytes per sector in the BPB
000:7CAD                jnz     short lblPrintErrorAndHangCaller
```

לאחר מכן, בהמשך ניתן לראות כמה הכנות ואז תחילת לולאה. אלו הן הכנות לטעינת BOOTMGR, ולמעשה לא מכילות המון לוגיקה. מייקרוסופט כתבו קוד יחסית גנרי, ולכן הוא מתייחס לנתונים מתוך ה-BPB. אף על פי כן, אנחנו נתייחס לנתונים האלה כקבועים:

```
seg000:7CAF ;
seg000:7CAF ; Copy bytes per sector to 0x7C0F and shift
seg000:7CAF ; Shifting by 4 is just like division by 16
seg000:7CAF ; This puts 0x20 in 0x7C0F
seg000:7CAF ;
seg000:7CAF                 mov     ds:0Fh, ax
seg000:7CB2                 shr     word ptr ds:0Fh, 4
seg000:7CB7 ;
seg000:7CB7 ; Preparations for loading BOOTMGR
seg000:7CB7 ;
seg000:7CB7                 push    ds
seg000:7CB8                 pop     dx              ;  DX = 0x07C0
seg000:7CB9                 xor     bx, bx
seg000:7CBB                 mov     cx, 2000h       ; CX = 16 sectors
seg000:7CBE                 sub     cx, ax          ; CX - AX = 15 sectors
seg000:7CC0                 inc     dword ptr ds:11h ; [0x7C11] = 1 (was zero)
```

יש כאן כמה דברים מעניינים שנזכור לשלב מאוחר יותר:

- ה-WORD בכתובת 0x7C0F מקבלת 0x20, שזה גודל סקטור חלקי 16. ניתן לראות כאן שמייקרוסופט עשו קוד גנרי (AX מכיל בשלב זה את גודל הסקטור) ולא סתם הציבו 0x20 במקום המתאים.
- אוגר DX מקבל את הערך 0x07C0.
- אוגר BX מאופס.
- אוגר CX מקבל גודל של 15 סקטורים, בהנחה שגודל סקטור הוא 512 (זה נכון לפי ה-BPB). דווקא כאן משעשע לראות שהקוד לא גנרי בכלל: אם גודל סקטור ישתנה בעתיד, הקוד הזה יידפק.
- ה- DWORD בכתובת 0x7C11 (שנמצאת ב-BPB) "הופך" למשתנה גלובאלי, והוא מקבל את הערך 1.

הקוד לאחר מכן ישר קורא לפונקציה שנמצאת בכתובת 0x7D1D , אז החלק הבא ינתח אותה.

חלק ג': ExtendedRead, שוב...

הפעם נשתולל לגמרי ונעשה חלקים די גדולים (הוספתי הערות כמובן):

```
seg000:7D1D
seg000:7D1D
seg000:7D1D ExtendedDiskRead proc near              ; CODE XREF: seg000:7CCF↑p
seg000:7D1D                 pushad                  ; All 32 bit registers are pushed
seg000:7D1F                 push    ds              ; Backup DS and ES
seg000:7D20                 push    es
seg000:7D21
seg000:7D21 lblReadAttempt:                         ; CODE XREF: ExtendedDiskRead+46↓j
seg000:7D21                 mov     eax, ds:11h     ; Loads the block counter
seg000:7D25                 add     eax, ds:1Ch     ; Adds the base block number
seg000:7D2A ;
seg000:7D2A ; Prepare buffer for extended read operation
seg000:7D2A ;
seg000:7D2A                 push    ds              ; Save DS
seg000:7D2B                 push    large 0         ; Starting block number HI
seg000:7D31                 push    eax             ; Starting block number LO
seg000:7D33                 push    es              ; Transfer buffer HI
seg000:7D34                 push    bx              ; Transfer buffer LO
seg000:7D35                 push    1               ; Number of blocks to transfer
seg000:7D38                 push    10h             ; Size = 0x10, reserved = 0
seg000:7D3B ;
seg000:7D3B ; Prepare parameters for extended read
seg000:7D3B ;
seg000:7D3B                 mov     ah, 42h ; 'B'   ; Function #52 - extended read
seg000:7D3D                 mov     dl, ds:0Eh      ; Drive number
seg000:7D41                 push    ss
seg000:7D42                 pop     ds              ; DS = 0 (because SS = 0)
seg000:7D43                 mov     si, sp          ; SI points to the buffer
seg000:7D45 ;
seg000:7D45 ; Perform the interrupt
seg000:7D45 ;
seg000:7D45                 int     13h             ; DISK - IBM/MS Extension - EXTENDED READ
seg000:7D47 ;
seg000:7D47 ; Cleanups
seg000:7D47 ;
seg000:7D47                 pop     ecx             ; 0x0110
seg000:7D49                 pop     bx              ; Old BX
seg000:7D4A                 pop     dx              ; Old ES
seg000:7D4B                 pop     ecx             ; Old EAX
seg000:7D4D                 pop     ecx             ; 0
seg000:7D4F                 pop     ds              ; Restore DS
```

ניתוח:

- תחילת הפונקציה מגבה את כל ה-General purpose registers וכן את DS ו-ES. אפשר לראות שבסוף הפונקציה יש POP-ים מתאימים.
- מכאן מתחילה לולאה שמשתמשת בכמה משתנים גלובאליים.
  - בכתובת 0x7C11 נשמר מספר, שבאיטרציה הראשונה הוא 1. אפשר לראות שהוא גדל כל פעם באחד (בכל איטרציה).
  - בכתובת 0x7C1C נשמר ערך שלא משתנה בין איטרציות (0x800).
  - מכיוון ששני הערכים מחוברים לתוך EAX (ומכיוון ש-EAX הופך למספר הבלוק בדיסק שממנו נבצע את הקריאה), ניתן להסיק שהערך שגדל כל פעם באחד הוא block counter ואילו הערך השני הוא "כתובת בסיס".
- מכאן מתבצעת הכנת ה-input buffer עבור הקריאה (פונקציה 0x42, פסיקה 0x13). עברנו במאמר על ה-MBR על פונקציה זו ועל המבנה של הבאפר בפירוט, ולכן לא אחזור עליו. כהרגלי, אפנה את הקורא המלומד אל RBIL ובו יש פירוט מלא.
- ביצוע הפסיקה וניקוי הבאפר ממנה. משעשע לראות שהניקוי הוא לא סתם ADD SP אלא ממש פקודות POP, כאשר ECX משמש כאוגר "זבל" ל-DWORD-ים סוררים.

אם לסכם, קראנו בלוק יחיד מהדיסק שנלקח מתוך EAX (והוא נלקח מתוך משתנה שמתחיל ב-0x32801 וגדל כל פעם באחד). הערכים הישנים של ES ו-BX נשמר ב-DX ו-BX, בהתאמה. להזכירכם, אלו הם הרגיסטרים ששומשו לשמירת הבלוק שנקרא מתוך הדיסק. כעת, אנחנו מצפים לוידוא שהקריאה הצליחה ולהכנה לקראת האיטרציה הבאה, וכך אכן מתבצע:

```
seg000:7D50 ;
seg000:7D50 ; Validation
seg000:7D50 ;
seg000:7D50                 jb      lblPrintErrorAndHang
seg000:7D54 ;
seg000:7D54 ; Post iteration operations
seg000:7D54 ;
seg000:7D54                 inc     dword ptr ds:11h ; Increase block counter
seg000:7D59                 add     dx, ds:0Fh       ; Increase buffer pointer
seg000:7D5D                 mov     es, dx
seg000:7D5F                 dec     word ptr ds:16h ; Decrease iteration flag
seg000:7D63                 jnz     short lblReadAttempt
seg000:7D65 ;
seg000:7D65 ; Restore registers and return
seg000:7D65 ;
seg000:7D65                 pop     es
seg000:7D66                 pop     ds
seg000:7D67                 popad
seg000:7D69                 retn
seg000:7D6A ; ---------------------------------------------------------------------------
seg000:7D6A
seg000:7D6A lblPrintErrorAndHang:                   ; CODE XREF: seg000:lblPrintErrorAndHangC
seg000:7D6A                                         ; ExtendedDiskRead+33↑j
seg000:7D6A                 mov     al, ds:1F8h
seg000:7D6D                 call    PrintMessage
seg000:7D70                 mov     al, ds:1FBh
seg000:7D73                 call    PrintMessage
seg000:7D76
seg000:7D76 lblHang:                                ; CODE XREF: seg000:7D77↓j
seg000:7D76                 hlt
seg000:7D76 ExtendedDiskRead endp
seg000:7D76
seg000:7D77 ; ---------------------------------------------------------------------------
seg000:7D77                 jmp     short lblHang
```

ניתוח:

- ביצוע JB כדי לוודא שהפסיקה הצליחה (אם היא נכשלה אז היא משנה את CF).
- הגדלה של ערך ES בערך שנמצא תחת 0x7C0F. ערך זה יהיה 0x20, כפי שניתחנו לפני כן.
- הורדה של הערך תחת 0x7C16. ערך זה יסמן לנו את מספר האיטרציות לביצוע - כאשר הוא מגיע לאפס, הפונקציה מסתיימת.
- כמובן, בכל מקרה של שגיאה תודפסנה הודעות שגיאה ונגיע ללולאה אינסופית של HLT ו-JMP. ראינו דוגמא דומה בניתוח של ה-MBR.

אם לסכם, פונקציה זו מבצעת קריאה סדרתית (block-by-block) של הדיסק. מספר הבלוקים נקבעים לפי המשתנים 0x7C11 ו-0x7C1C, מספר האיטרציות נקבע לפי 0x7C16, ובאפר היעד יהיה ES:BX.

חלק ד': יאללה ל-bootmgr!

עכשיו יהיה מעניין להסתכל על הקוד שקרא לפונקציה שכעת ניתחנו. הפלא-ופלא, זה בדיוק קוד ההמשך שלנו:

```
seg000:7CC5 ;
seg000:7CC5 ; Load BOOTMGR code
seg000:7CC5 ;
seg000:7CC5
seg000:7CC5 lblLoadChunkFromDisk:                   ; CODE XREF: seg000:7CD4↓j
seg000:7CC5                 add     dx, ds:0Fh      ; DX is increased by 0x20 (0x07E0, 0x0800, 0x0820, ..., 0x9A0)
seg000:7CC9                 mov     es, dx          ; ES = DX
seg000:7CCB                 inc     word ptr ds:16h ; 0x7C16 = 1 = one iteration
seg000:7CCF                 call    ExtendedDiskRead
seg000:7CD2                 sub     cx, ax          ; 0x2000 initially, each time decreased by 0x200
seg000:7CD4                 ja      short lblLoadChunkFromDisk
```

ניתוח:

- הערך של DX עולה כל פעם ב-0x20 (זה גודל סקטור חלקי 16 שחישבנו לפני כן). בהתחלה הוא היה 0x07C0, ולכן באיטרציה הראשונה הוא יקבל 0x07E0 וכן הלאה. לאחר מכן אנחנו מבינים ש-DX סתם היה אוגר ביניים, וכל מה שאמרתי עד כה היה עבור ES.
- **נקודה עדינה:** זה זמן טוב להיזכר ש-BX מאופס. מכיוון ש-ES:BX משמשים כבאפר שבו ייכתב המידע, בפועל המידע ייכתב לכתובות הפיזיות 0x7E00, 0x8000 וכן הלאה! לכן היה צורך לחלק את גודל הסקטור ב-16, וכאן ניתן לראות כבר שהקוד שמתמלא ייכנס היישר אל x7C000. נזכור גם ש-BX (ולמעשה, כל שאר האוגרים) לא מושפע מקריאה לפונקציה עקב השימוש ב-PUSHAD.
- ייחסנו חשיבות גדולה אל 0x7C16, והפונקציה ExtendedDiskRead מתייחסת אליו מאד ברצינות, אבל בפועל הוא תמיד יהיה 1... לכן קראתי לו בשם "Iteration flag" - משעשע לראות שמייקרוסופט כתבו פונקציית ExtendedRead גנרית מאד שתומכת במספר רב של איטרציות, אך בפועל קוראת לו כל פעם עם איטרציה אחת...

לאחר מכן בודקים תמיכה ב-Trusted platform:

```
seg000:7CD6 ;
seg000:7CD6 ; TCG, if possible
seg000:7CD6 ;
seg000:7CD6                 mov     ax, 0BB00h
seg000:7CD9                 int     1Ah             ; Trusted Computing Group call - TCG_StatusCheck
seg000:7CD9                                         ; Return: EAX = 0 if supported
seg000:7CD9                                         ; EBX = 41504354h ('TCPA')
seg000:7CD9                                         ; CH:CL = TCG BIOS Version
seg000:7CD9                                         ; EDX = BIOS TCG Feature Flags
seg000:7CD9                                         ; ESI = Pointer to Event Log
seg000:7CD9                                         ;
seg000:7CDB                 and     eax, eax
seg000:7CDE                 jnz     short lblRunBootMgr ; Best-effort
seg000:7CE0                 cmp     ebx, 'APCT'
seg000:7CE7                 jnz     short lblRunBootMgr ; Best-effort
seg000:7CE9                 cmp     cx, 102h
seg000:7CED                 jb      short lblRunBootMgr ; Best-effort
```

ניתוח:

- קריאה ל-TCG_StatusCheck (על ידי AX=0xBB00), מחזירה ב-EAX את הערך 0 אם יש תמיכה. אם לא, ממשיכים הלאה.
- בדיקה ש-EBX מחזיק את ערך החזרה הנכון. אם לא, ממשיכים הלאה.
- בדיקה שהגרסא היא 1.02 ומעלה. אם לא, ממשיכים הלאה.
- הייתי רוצה שנזכור אם המשפט "**אם לא, ממשיכים הלאה**", שחזר אחרי עצמו 3 פעמים. אנחנו ננצל את המשפט הזה להערות הסיום.

אם הכל טוב עד כה, נגיע ל-0x7CEF. בבלוק זה תתבצע קריאה אל TCG_CompactHashLogExtendEvent ולאחר מכן יבוצע fallback אל lblRunBootMgr. הקריאה מתוארת באתר של Trusted Computing Group תחת:

https://www.trustedcomputinggroup.org/files/resource_files/CB0B2BFA-1A4B-B294-D0C3B9075B5AFF17/TCG_PCClientImplementation_1-21_1_00.pdf

הנה הקוד שקורא לפונקציה זו:

```
seg000:7CEF ;
seg000:7CEF ; TCG_CompactHashLogExtendEvent
seg000:7CEF ;
seg000:7CEF                 push    ss
seg000:7CF0                 push    0BB07h          ; EAX = 0x0000BB07
seg000:7CF3                 push    ss
seg000:7CF4                 push    0E70h           ; ECX = 0x00000E70
seg000:7CF7                 push    ss
seg000:7CF8                 push    9               ; EDX = 0x00000009
seg000:7CFB                 push    ebx             ; EBX
seg000:7CFD                 push    ebx             ; Dummy ESP
seg000:7CFF                 push    ebp             ; EBP
seg000:7D01                 push    ss
seg000:7D02                 push    ss              ; ESI = 0x00000000 = informative value to be placed into the event field
seg000:7D03                 push    ss
seg000:7D04                 push    1B8h            ; EDI = 0x000001B8 = offset of buffer to be hashed
seg000:7D07                 popad
seg000:7D09                 push    cs
seg000:7D0A                 pop     es              ; ES = CS = segment of buffer to be hashed
seg000:7D0B                 int     1Ah             ; Trusted Computing Group call - TCG_StatusCheck
seg000:7D0B                                         ; Return: EAX = 0 if supported
seg000:7D0B                                         ; EBX = 41504354h ('TCPA')
seg000:7D0B                                         ; CH:CL = TCG BIOS Version
seg000:7D0B                                         ; EDX = BIOS TCG Feature Flags
seg000:7D0B                                         ; ESI = Pointer to Event Log
seg000:7D0B                                         ;
```

ניתוח:

- הקוד דוחף מלא ערכים למחסנית ואז מבצע POPAD, כדי להשפיע על כל האוגרים. נשים לב שהאוגרים יוצאים בסדר הפוך.
- כל השדות מתוארים הן בקוד והן באתר של ה-TCG. ספציפית, השדות המאד מעניינים הם המצביע לבאפר שעליו עושים HASH (כתובת 07C0:01B8) וגודלו (0xE70).

לאחר מכן מגיעים אל סוף התכנית, lblRunBootMgr:

```
seg000:7D0D
seg000:7D0D lblRunBootMgr:                          ; CODE XREF: seg000:7CDE↑j
seg000:7D0D                                         ; seg000:7CE7↑j ...
seg000:7D0D                 xor     ax, ax
seg000:7D0F ;
seg000:7D0F ; Zero filling
seg000:7D0F ;
seg000:7D0F                 mov     di, 1028h
seg000:7D12                 mov     cx, 0FD8h
seg000:7D15                 cld
seg000:7D16                 rep stosb
seg000:7D18 ;
seg000:7D18 ; Pass control to bootmgr
seg000:7D18 ;
seg000:7D18                 jmp     near ptr 7E7Ah
seg000:7D1B ; ---------------------------------------------------------------------------
seg000:7D1B                 nop
seg000:7D1C                 nop
```

כאן מתבצע Zero filling, אף על פי שלא ברור מדוע יש צורך במילוי הזיכרון באפסים ולכאורה ניתן להסתדר בלעדיו. בכל מקרה, לאחר מכן, מעבירים את השליטה אל הכתובת 0x7E7A, שאליה נטען BOOTMGR.

כאן למעשה נגמר הקוד הראשי של ה-VBR. את שתי הפונקציות (קריאה מהדיסק והדפסה) ניתחנו.

נקודות מעניינות ותובנות

- יש לי חוב קטן: איך אני יודע שמדובר ב-bootmgr בכלל? ישנן כמה תשובות טובות, והכנתי אותן בסגנון פסח, כי זה הסתדר לי לא רע:
  1. **חכם** - מה הוא אומר? דיבגתי ואכן ראיתי ש-bootmgr עולה.
  2. **רשע** - מה הוא אומר? הרי קיימת בקוד הודעת השגיאה "BOOTMGR is missing".
  3. **תם** - מה הוא אומר? שרשרת העלייה של Windows מתעדת שהרכיב הבא בתור הוא bootmgr ואני מאמין לתיעוד.
  4. ושאינו יודע לשאול? כנראה שלא הגיע לחלק זה של המאמר בכל מקרה...

- אני רוצה שניזכר בנקודה המעניינת על ה-TCG - שימו לב שמתבצעות מספר קריאות, אבל הן best-effort, כלומר, אם אין תמיכה - לא נורא. חלק גדול מה- bootkits עשו hooking על int 0x13 ועל ידי כך החזירו בלוקים שקריים מהדיסק, עם נתונים שלהם. ניתן בהחלט להמציא bootkit שיבצע hooking על הקריאות אל ה-TCG, מכיוון שאף על פי שהן נקראות - נראה שלשרשרת העלייה לא כל כך אכפת אם הן מצליחות או לא. יהיה מעניין כפרוייקט לעשות hooking על int 0x1A.

סיכום

- המשכנו את תהליך העלייה של Windows, הפעם התמקדנו בחלק השני: ה-VBR.
- התעמקנו בנושאים הבאים:
  - ה-BPB
  - קריאות אל ה-TPM
  - טעינת bootmgr מהדיסק עם Extended Read
  - העברת שליטה אל bootmgr
- הוספתי נספח של הקוד השלם. הייתי מוסיף IDB, אבל גרסאות שונות של IDA לא תומכות בהכרח בכל IDB וגם נחמד שזה יגיע יחד עם המסמך. מדובר בקוד הסופי, כולל ההערות, כמובן.

על המחבר

0x3d5157636b525761, עושה Reversing ופיתוח Low Level למחייתו. ניתן ליצור איתי קשר ב:
0x3d5157636b525761@gmail.com

נספח א': הקוד המלא כולל הערות

```
seg000:7C00 ;
seg000:7C00 ; Input MD5   : C1C0E5D5E2701CBDA3FD4292CD32D6C2
seg000:7C00 ; Input CRC32 : 2A6D0660
seg000:7C00
seg000:7C00 ; ---------------------------------------------------------------------------
seg000:7C00 ; File Name   : vbr.bin
seg000:7C00 ; Format      : Binary file
seg000:7C00 ; Base Address: 0000h Range: 0000h - 0200h Loaded length: 0200h
seg000:7C00
seg000:7C00                 .686p
seg000:7C00                 .mmx
seg000:7C00                 .model flat
seg000:7C00
seg000:7C00 ; ===========================================================================
seg000:7C00
seg000:7C00 ; Segment type: Pure code
seg000:7C00 seg000          segment byte public 'CODE' use16
seg000:7C00                 assume cs:seg000
seg000:7C00                 ;org 7C00h
seg000:7C00                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:7C00
seg000:7C00 lblStart:                               ; DATA XREF: seg000:7C62o
seg000:7C00                 jmp     short lblMain
seg000:7C02 ; ---------------------------------------------------------------------------
seg000:7C02                 nop
seg000:7C02 ; ---------------------------------------------------------------------------
seg000:7C03 g_dwOemLo       dd 'SFTN'               ; DATA XREF: seg000:7C6Ar
seg000:7C03                                         ; OEM ID
seg000:7C07                 dd '    '
seg000:7C0B                 dw 200h                 ; Bytes per sector
seg000:7C0D                 db    8                 ; Sectors per cluster
seg000:7C0E g_wReservedSectors dw 0                 ; DATA XREF: seg000:lblStartParsingNTFSw
seg000:7C0E                                         ; seg000:7C96r
seg000:7C0E                                         ; Reserved sectors
seg000:7C10                 db 0, 0, 0              ; 0 ; Zero
seg000:7C13                 dw 0                    ; Unused
seg000:7C15                 db 0F8h ; °             ; Media descriptor
seg000:7C16                 dw 0                    ; Zero
seg000:7C18                 dw 3Fh                  ; Sectors per track
seg000:7C1A                 dw 0FFh                 ; Number of heads
seg000:7C1C                 dd 32800h               ; Hidden sectors
seg000:7C20                 dd 0                    ; Unused
seg000:7C24                 dd 800080h              ; Unused
seg000:7C28                 dq 1866D7FFh            ; Total sectors
seg000:7C30                 dq 0C0000h              ; Logical cluster number for the file
$MFT
seg000:7C38                 dq 2                    ; Logical cluster number for the file
$MFTMirr
seg000:7C40                 dd 0F6h                 ; Clusters per file record segment
seg000:7C44                 db    1                 ; Clusters per index buffer
seg000:7C45                 db 0, 0, 0              ; 0 ; Unused
seg000:7C48                 dq 88B4B852B4B8448Ah    ; Volume serial number
seg000:7C50                 dd 0                    ; Checksum
seg000:7C54 ; ---------------------------------------------------------------------------
seg000:7C54
seg000:7C54 lblMain:                                ; CODE XREF: seg000:lblStartj
seg000:7C54                 cli
seg000:7C55 ;
seg000:7C55 ; Build stack
seg000:7C55 ;
```

```
seg000:7C55
seg000:7C55                 xor     ax, ax
seg000:7C57                 mov     ss, ax
seg000:7C59                 mov     sp, 7C00h
seg000:7C5C                 sti
seg000:7C5D ;
seg000:7C5D ; Make CS=DS=0x07C0
seg000:7C5D ;
seg000:7C5D                 push    7C0h
seg000:7C60                 pop     ds
seg000:7C61                 assume ds:nothing
seg000:7C61                 push    ds
seg000:7C62                 push    (offset lblStartParsingNTFS - offset lblStart)
seg000:7C65                 retf
seg000:7C66
seg000:7C66 lblStartParsingNTFS:                    ; DATA XREF: seg000:7C62o
seg000:7C66                 mov     ds:0Eh, dl      ; Save drive number in [0x0E]
seg000:7C6A ;
seg000:7C6A ; Make sure the OEM is "NTFS"
seg000:7C6A ;
seg000:7C6A                 cmp     dword ptr ds:3, 'SFTN'
seg000:7C73                 jnz     short lblPrintErrorAndHangCaller
seg000:7C75 ;
seg000:7C75 ; Check for LBA mode reading
seg000:7C75 ;
seg000:7C75                 mov     ah, 41h ; 'A'
seg000:7C77                 mov     bx, 55AAh
seg000:7C7A                 int     13h             ; DISK - Check for INT 13h Extensions
seg000:7C7A                                         ; BX = 55AAh, DL = drive number
seg000:7C7A                                         ; Return: CF set if not supported
seg000:7C7A                                         ; AH = extensions version
seg000:7C7A                                         ; BX = AA55h
seg000:7C7A                                         ; CX = Interface support bit map
seg000:7C7C                 jb      short lblPrintErrorAndHangCaller
seg000:7C7E                 cmp     bx, 0AA55h
seg000:7C82                 jnz     short lblPrintErrorAndHangCaller
seg000:7C84                 test    cx, 1
seg000:7C88                 jnz     short lblGetDriveParams
seg000:7C8A
seg000:7C8A lblPrintErrorAndHangCaller:             ; CODE XREF: seg000:7C73j
seg000:7C8A                                         ; seg000:7C7Cj ...
seg000:7C8A                 jmp     lblPrintErrorAndHang
seg000:7C8D ; ---------------------------------------------------------------------------
seg000:7C8D
seg000:7C8D lblGetDriveParams:                      ; CODE XREF: seg000:7C88j
seg000:7C8D                 push    ds
seg000:7C8E ;
seg000:7C8E ; Make room for the buffer on the stack
seg000:7C8E ; Size of buffer = 0x1A
seg000:7C8E ;
seg000:7C8E                 sub     sp, 18h
seg000:7C91                 push    1Ah             ; 0x1A because it's WORD more than 0x18 +
the way PUSH works
seg000:7C94 ;
seg000:7C94 ; Function 48 (get drive paramters)
seg000:7C94 ; DL = drive number
seg000:7C94 ; DS:SI = buffer to fill
seg000:7C94 ;
seg000:7C94                 mov     ah, 48h ; 'H'
seg000:7C96                 mov     dl, ds:0Eh      ; DL = Drive number from [0x0E]
seg000:7C9A                 mov     si, sp
seg000:7C9C                 push    ss
seg000:7C9D                 pop     ds
```

```
seg000:7C9E                 assume ds:nothing
seg000:7C9E                 int     13h             ; DISK - IBM/MS Extension - GET DRIVE
PARAMETERS (DL - drive, DS:SI - buffer)
seg000:7CA0 ;
seg000:7CA0 ; Clear buffer from stack
seg000:7CA0 ;
seg000:7CA0                 lahf
seg000:7CA1                 add     sp, 18h
seg000:7CA4                 sahf
seg000:7CA5                 pop     ax              ; Bytes per sector
seg000:7CA6                 pop     ds              ; Restore DS
seg000:7CA7 ;
seg000:7CA7 ; Check for failures and mismatches:
seg000:7CA7 ;    1. INT13 failure
seg000:7CA7 ;    2. Bytes per sector failure
seg000:7CA7 ;
seg000:7CA7                 jb      short lblPrintErrorAndHangCaller
seg000:7CA9                 cmp     ax, ds:0Bh      ; [0x0B] is exactly bytes per sector in
the BPB
seg000:7CAD                 jnz     short lblPrintErrorAndHangCaller
seg000:7CAF ;
seg000:7CAF ; Copy bytes per sector to 0x7C0F and shift
seg000:7CAF ; Shifting by 4 is just like division by 16
seg000:7CAF ; This puts 0x20 in 0x7C0F
seg000:7CAF ;
seg000:7CAF                 mov     ds:0Fh, ax
seg000:7CB2                 shr     word ptr ds:0Fh, 4
seg000:7CB7 ;
seg000:7CB7 ; Preparations for loading BOOTMGR
seg000:7CB7 ;
seg000:7CB7                 push    ds
seg000:7CB8                 pop     dx              ;  DX = 0x07C0
seg000:7CB9                 xor     bx, bx
seg000:7CBB                 mov     cx, 2000h       ; CX = 16 sectors
seg000:7CBE                 sub     cx, ax          ; CX - AX = 15 sectors
seg000:7CC0                 inc     dword ptr ds:11h ; [0x7C11] = 1 (was zero)
seg000:7CC5 ;
seg000:7CC5 ; Load BOOTMGR code
seg000:7CC5 ;
seg000:7CC5
seg000:7CC5 lblLoadChunkFromDisk:                   ; CODE XREF: seg000:7CD4j
seg000:7CC5                 add     dx, ds:0Fh      ; DX is increased by 0x20 (0x07E0,
0x0800, 0x0820, ..., 0x9A0)
seg000:7CC9                 mov     es, dx          ; ES = DX
seg000:7CCB                 inc     word ptr ds:16h ; 0x7C16 = 1 = one iteration
seg000:7CCF                 call    ExtendedDiskRead
seg000:7CD2                 sub     cx, ax          ; 0x2000 initially, each time decreased
by 0x200
seg000:7CD4                 ja      short lblLoadChunkFromDisk
seg000:7CD6 ;
seg000:7CD6 ; TCG, if possible
seg000:7CD6 ;
seg000:7CD6                 mov     ax, 0BB00h
seg000:7CD9                 int     1Ah             ; Trusted Computing Group call -
TCG_StatusCheck
seg000:7CD9                                         ; Return: EAX = 0 if supported
seg000:7CD9                                         ; EBX = 41504354h ('TCPA')
seg000:7CD9                                         ; CH:CL = TCG BIOS Version
seg000:7CD9                                         ; EDX = BIOS TCG Feature Flags
seg000:7CD9                                         ; ESI = Pointer to Event Log
seg000:7CD9                                         ;
seg000:7CDB                 and     eax, eax
seg000:7CDE                 jnz     short lblRunBootMgr ; Best-effort
```

```
seg000:7CE0                 cmp     ebx, 'APCT'
seg000:7CE7                 jnz     short lblRunBootMgr ; Best-effort
seg000:7CE9                 cmp     cx, 102h
seg000:7CED                 jb      short lblRunBootMgr ; Best-effort
seg000:7CEF ;
seg000:7CEF ; TCG_CompactHashLogExtendEvent
seg000:7CEF ;
seg000:7CEF                 push    ss
seg000:7CF0                 push    0BB07h          ; EAX = 0x0000BB07
seg000:7CF3                 push    ss
seg000:7CF4                 push    0E70h           ; ECX = 0x00000E70 = length of buffer to
be hashed
seg000:7CF7                 push    ss
seg000:7CF8                 push    9               ; EDX = 0x00000009 = PCR index
seg000:7CFB                 push    ebx             ; EBX
seg000:7CFD                 push    ebx             ; Dummy ESP
seg000:7CFF                 push    ebp             ; EBP
seg000:7D01                 push    ss
seg000:7D02                 push    ss              ; ESI = 0x00000000 = informative value to
be placed into the event field
seg000:7D03                 push    ss
seg000:7D04                 push    1B8h            ; EDI = 0x000001B8 = offset of buffer to
be hashed
seg000:7D07                 popad
seg000:7D09                 push    cs
seg000:7D0A                 pop     es              ; ES = CS = segment of buffer to be
hashed
seg000:7D0B                 int     1Ah             ; Trusted Computing Group call -
TCG_StatusCheck
seg000:7D0B                                         ; Return: EAX = 0 if supported
seg000:7D0B                                         ; EBX = 41504354h ('TCPA')
seg000:7D0B                                         ; CH:CL = TCG BIOS Version
seg000:7D0B                                         ; EDX = BIOS TCG Feature Flags
seg000:7D0B                                         ; ESI = Pointer to Event Log
seg000:7D0B                                         ;
seg000:7D0D
seg000:7D0D lblRunBootMgr:                          ; CODE XREF: seg000:7CDEj
seg000:7D0D                                         ; seg000:7CE7j ...
seg000:7D0D                 xor     ax, ax
seg000:7D0F ;
seg000:7D0F ; Zero filling
seg000:7D0F ;
seg000:7D0F                 mov     di, 1028h
seg000:7D12                 mov     cx, 0FD8h
seg000:7D15                 cld
seg000:7D16                 rep stosb
seg000:7D18 ;
seg000:7D18 ; Pass control to bootmgr
seg000:7D18 ;
seg000:7D18                 jmp     near ptr 7E7Ah
seg000:7D1B ; ---------------------------------------------------------------------------
seg000:7D1B                 nop
seg000:7D1C                 nop
seg000:7D1D
seg000:7D1D ; =============== S U B R O U T I N E =======================================
seg000:7D1D
seg000:7D1D
seg000:7D1D ExtendedDiskRead proc near              ; CODE XREF: seg000:7CCFp
seg000:7D1D                 pushad                  ; All 32 bit registers are pushed
seg000:7D1F                 push    ds              ; Backup DS and ES
seg000:7D20                 push    es
seg000:7D21
seg000:7D21 lblReadAttempt:                         ; CODE XREF: ExtendedDiskRead+46j
```

```
seg000:7D21                 mov     eax, ds:11h     ; Loads the block counter
seg000:7D25                 add     eax, ds:1Ch     ; Adds the base block number
seg000:7D2A ;
seg000:7D2A ; Prepare buffer for extended read operation
seg000:7D2A ;
seg000:7D2A                 push    ds              ; Save DS
seg000:7D2B                 push    large 0         ; Starting block number HI
seg000:7D31                 push    eax             ; Starting block number LO
seg000:7D33                 push    es              ; Transfer buffer HI
seg000:7D34                 push    bx              ; Transfer buffer LO
seg000:7D35                 push    1               ; Number of blocks to transfer
seg000:7D38                 push    10h             ; Size = 0x10, reserved = 0
seg000:7D3B ;
seg000:7D3B ; Prepare parameters for extended read
seg000:7D3B ;
seg000:7D3B                 mov     ah, 42h ; 'B'   ; Function #52 - extended read
seg000:7D3D                 mov     dl, ds:0Eh      ; Drive number
seg000:7D41                 push    ss
seg000:7D42                 pop     ds              ; DS = 0 (because SS = 0)
seg000:7D43                 mov     si, sp          ; SI points to the buffer
seg000:7D45 ;
seg000:7D45 ; Perform the interrupt
seg000:7D45 ;
seg000:7D45                 int     13h             ; DISK - IBM/MS Extension - EXTENDED READ
(DL - drive, DS:SI - disk address packet)
seg000:7D47 ;
seg000:7D47 ; Cleanups
seg000:7D47 ;
seg000:7D47                 pop     ecx             ; 0x0110
seg000:7D49                 pop     bx              ; Old BX
seg000:7D4A                 pop     dx              ; Old ES
seg000:7D4B                 pop     ecx             ; Old EAX
seg000:7D4D                 pop     ecx             ; 0
seg000:7D4F                 pop     ds              ; Restore DS
seg000:7D50 ;
seg000:7D50 ; Validation
seg000:7D50 ;
seg000:7D50                 jb      lblPrintErrorAndHang
seg000:7D54 ;
seg000:7D54 ; Post iteration operations
seg000:7D54 ;
seg000:7D54                 inc     dword ptr ds:11h ; Increase block counter
seg000:7D59                 add     dx, ds:0Fh      ; Increase buffer pointer
seg000:7D5D                 mov     es, dx
seg000:7D5F                 dec     word ptr ds:16h ; Decrease iteration flag
seg000:7D63                 jnz     short lblReadAttempt
seg000:7D65 ;
seg000:7D65 ; Restore registers and return
seg000:7D65 ;
seg000:7D65                 pop     es
seg000:7D66                 pop     ds
seg000:7D67                 popad
seg000:7D69                 retn
seg000:7D6A ; ---------------------------------------------------------------------------
seg000:7D6A
seg000:7D6A lblPrintErrorAndHang:                   ; CODE XREF:
seg000:lblPrintErrorAndHangCallerj
seg000:7D6A                                         ; ExtendedDiskRead+33j
seg000:7D6A                 mov     al, ds:1F8h
seg000:7D6D                 call    PrintMessage
seg000:7D70                 mov     al, ds:1FBh
seg000:7D73                 call    PrintMessage
seg000:7D76
```

```
seg000:7D76 lblHang:                                ; CODE XREF: seg000:7D77j
seg000:7D76                 hlt
seg000:7D76 ExtendedDiskRead endp
seg000:7D76
seg000:7D77 ; ---------------------------------------------------------------------------
seg000:7D77                 jmp     short lblHang
seg000:7D79
seg000:7D79 ; =============== S U B R O U T I N E =======================================
seg000:7D79
seg000:7D79
seg000:7D79 PrintMessage    proc near               ; CODE XREF: ExtendedDiskRead+50p
seg000:7D79                                         ; ExtendedDiskRead+56p
seg000:7D79                 mov     ah, 1
seg000:7D7B                 mov     si, ax
seg000:7D7D ;
seg000:7D7D ; Handle the next character
seg000:7D7D ;
seg000:7D7D
seg000:7D7D lblNextChar:                            ; CODE XREF: PrintMessage+10j
seg000:7D7D                 lodsb
seg000:7D7E                 cmp     al, 0
seg000:7D80                 jz      short lblFinishPrinting
seg000:7D82 ;
seg000:7D82 ; Perform TTY printing and handle the next character
seg000:7D82 ;
seg000:7D82                 mov     ah, 0Eh
seg000:7D84                 mov     bx, 7
seg000:7D87                 int     10h             ; - VIDEO - WRITE CHARACTER AND ADVANCE
CURSOR (TTY WRITE)
seg000:7D87                                         ; AL = character, BH = display page
(alpha modes)
seg000:7D87                                         ; BL = foreground color (graphics modes)
seg000:7D89                 jmp     short lblNextChar
seg000:7D8B ; ---------------------------------------------------------------------------
seg000:7D8B
seg000:7D8B lblFinishPrinting:                      ; CODE XREF: PrintMessage+7j
seg000:7D8B                 retn
seg000:7D8B PrintMessage    endp
seg000:7D8B
seg000:7D8B ; ---------------------------------------------------------------------------
seg000:7D8C aADiskReadError db 0Dh,0Ah
seg000:7D8C                 db 'A disk read error occurred',0
seg000:7DA9 aBootmgrIsMissi db 0Dh,0Ah
seg000:7DA9                 db 'BOOTMGR is missing',0
seg000:7DBE aBootmgrIsCompr db 0Dh,0Ah
seg000:7DBE                 db 'BOOTMGR is compressed',0
seg000:7DD6 aPressCtrlAltDe db 0Dh,0Ah
seg000:7DD6                 db 'Press Ctrl+Alt+Del to restart',0Dh,0Ah,0
seg000:7DF8                 db 8Ch
seg000:7DF9                 db 0A9h ; _
seg000:7DFA                 db 0BEh ; _
seg000:7DFB                 db 0D6h ; _
seg000:7DFC                 db    0
seg000:7DFD                 db    0
seg000:7DFE                 db  55h ; U
seg000:7DFF                 db 0AAh ; ¬
seg000:7DFF seg000          ends
seg000:7DFF
seg000:7DFF
seg000:7DFF                 end
```