digital whisper
Technologic Papers

NTFS Forensic

מאת דני אודלר

הקדמה

"NTFS (ראשי תיבות של New Technology File System), היא מערכת קבצים בעיצוב מיקרוסופט, שיצאה לשוק ב-1993, יחד עם Windows NT 3.1. NTFS התפתחה מתוך מערכת הקבצים HPFS שעוצבה כחלק מהמיזם OS/2 המשותף ל--IBMולמיקרוסופט ועל פי עקרונות של מערכת ניהול הקבצים NTFS נועדה להחליף את מערכת הקבצים FAT והציגה שיפורים רבים על פניה.

NTFS כוללת מגוון שיפורים על פני FAT, כגון - תמיכה בקבצים גדולים במיוחד (עד 2^{64} בתים), רשימת בקרת גישה (המאפשרת לקבוע הרשאות גישה למשתמשים ברמה של קבצים בודדים), תיעוד פעולות (Journaling) דחיסת נתונים מובנית, התאוששות עצמית, יכולת שחזור נתונים במצב של כשל בדיסק, וחיפוש קבצים מהיר יותר באמצעות קיטלוג." (ויקיפדיה)

מספר דגשים לפני המשך קריאת המאמר:

1. הסקירה להלן תתייחס למבנה העדכני של NTFS המיושם ב-Windows8 וב-Windows10
2. הנחת הבסיס היא שמדובר במחשב עם BIOS ולא עם UEFI מכיוון ויש שוני במבנים שיתוארו בהמשך וחלוקת המחיצות ב-UEFI שונה ומוגדרת אחרת.

במערכת קבצים זו, **Sector** לרב מוגדר כ-512 בתים רציפים בדיסק. **Cluster** לרב מוגדר כ-8 סקטורים, (4096 בתים רציפים בדיסק), **ומחיצה** (Partition) הינה ייצוג לוגי לחלק מהדיסק. יתכן ויהיו מספר מחיצות בדיסק מסויים.

מבנה מערכת הקבצים הינו פרי יצירתה של מייקרוסופט והתיעוד איננו רב אם בכלל. אך עם זאת, במרוץ השנים נעשו מאמצים לפתח מימוש דומה עבור מערכות לינוקס אשר דרש כתיבת דרייבר NTFS, פרוייקט זה מספק מידע בעל ערך רב (ראה מקורות מס' 3).

ראשית צריך להבין שכל דבר ב-NTFS הוא קובץ, החל מקבצי ה-meta files של ה-NTFS ועד הקבצים הרגילים הפזורים בדיסק. כך למשל גם ה-record NTFS boot הינו קובץ לכל דבר.

מבנה הדיסק:

מה זה MBR?

MBR - Master Boot Record - נמצא פיזית ב-Cylinder 0, Head 0, Sector 1. ה-MBR תופס את ה-512 הבתים הראשונים בדיסק. 446 הבתים הראשונים מכילים קוד אתחול ובדיקת קיום של TPM. הקוד בעיקרו מעתיק את עצמו לזכרון וקופץ אל מיקום זה להמשך ריצה. הקוד גם מטפל בהצגת שגיאות כגון מחיצה לא מאותחלת ועוד. בהנחה שאין UEFI , כל הקוד רץ ב-real mode - אין זכרון וירטואלי ומרחב הכתובות לרב מוגבל ל-20 ביט.

64 הבתים הבאים מכילים את טבלת המחיצות אשר מחזיקה לכל מחיצה את מאפייניה כגון גודל, מיקום התחלה, סוג, אתחול. למעשה יש תמיכה בארבע מחיצות לעומת מערכות UEFI בה הדיסק פורמט עם טבלת GPT (מחוץ למאמר זה):

Description	Size (Bytes)
Bootstrap code area	446
Partition Entry #1	16
Partition Entry #2	16
Partition Entry #3	16
Partition Entry #4	16
Magic Number	2
Total:	**512**

לאחר שה-Bios מסיים את פעולות ה-Power-On Self Test הוא יטען את קוד ה-MBR לזכרון ויעביר את הריצה אליו. ה-MBR יעבור על ה-partition table ויחפש את המחיצה שמסומנת כ-bootable, לאחר שמצא אותה הוא יטען לזכרון את ה-VBR של אותה מחיצה. המחיצה הראשונה, לאו דווקא מחיצת האתחול, תתחיל בסקטור 2048 או בדיוק 1 מגהבייט מתחילת הדיסק.

ב-Windows8 ומעלה המחיצה הראשונה מוגדרת כשמורה system reserved וגודלה 350 מגהבייט, רב המחיצה הזאת תפוסה ע"י סביבת השיחזור של Windows ולשם כך היא נועדה.

מה זה VBR?

VBR - Volume Boot Record, מתחיל מייד עם תחילת המחיצה ואנו נתייחס כאן ל-VBR של מחיצת האתחול, כלומר אם יש לדגומא 4 מחיצות בדיסק אזי רב המחיצות יכילו VBR ואנו נתחייס בפרט למחיצת האתחול של Windows. 16 הסקטורים הראשונים של מחיצת האתחול מוקדשים לביצוע פעולות אתחול כאשר הסקטור הראשון הוא למעשה ה-VBR ומיד אחריו ישנם 9 סקטורים אשר מכילים את ה-Bootmgr Loader אשר מתעדכן מעת לעת לפי גרסאת וינדווז הרלוונטית.

בלוק ה-VBR מתחיל בקפיצה, אופקוד EB52, להיסט 0x54 ובכך מדלג על המשתני אתחול שלו (ראה איור בעמוד הבא). קוד הריצה מכיל ברובו קריאות לפסיקות INT13 BIOS אשר מפורשים כשרותי קריאה מהדיסק הקשיח במצב real mode, הרבה לפני שדרייברים של NTFS נטענים. אפשר לראות למשל בהיסט 0x7A, 0x9E,0x145 את הקריאות לפסיקה זו - CD13.

לאחר מכן קוד ה-VBR מעתיק לזכרון את 9 הסקטורים שבהם נמצא ה-bootmgr loader ומבצע בדיקה לגבי רכיב ה-TPM, ניתן לראות את האחרון בהיסט 0xE3 שם ישנם תווי אסקי TCPA, זה הוא קיצור של Trusted Computing Platform Alliance אשר קשור לבדיקות רכיב ה-TPM, Bitlocker למשל ישתמש במידע זה כדי להצפין דאטה.

בלוק ה-VBR מסתיים עם חתימה של 0x55AA:

```
x0008A800  x000  EB 52 90 4E 54 46 53 20 20 20 20 00 02 08 00 00   ëR NTFS    ....
567,296    x010  00 00 00 00 00 F8 00 00 3F 00 FF 00 00 A8 08 00   .....ø..?.ÿ..¨..
           x020  00 00 00 00 80 00 80 00 21 B0 C0 1B 00 00 00 00   ....€.€.!°À.....
           x030  00 00 0C 00 00 00 00 00 02 00 00 00 00 00 00 00   ................
           x040  F6 00 00 00 01 00 00 00 44 D4 89 5A FD 89 5A 1C   ö.......DÔ‰Zý‰Z.
           x050  00 00 00 00 FA 33 C0 8E D0 BC 00 7C FB 68 C0 07   ....ú3À‎Ð¼.|ûhÀ.
           x060  1F 1E 68 66 00 CB 88 16 0E 00 66 81 3E 03 00 4E   ..hf.Ë‰...f>..N
           x070  54 46 53 75 15 B4 41 BB AA 55 CD 13 72 0C 81 FB   TFSu.´A»ªUÍ.r.û
           x080  55 AA 75 06 F7 C1 01 00 75 03 E9 DD 00 1E 83 EC   Uªu.÷Á..u.éÝ..‎ì
           x090  18 68 1A 00 B4 48 8A 16 0E 00 8B F4 16 1F CD 13   .h..´H‎Š...‎‎ô..Í.
           x0A0  9F 83 C4 18 9E 58 1F 72 E1 3B 06 0B 00 75 DB A3   ‎ƒÄ.‎žX.rá;...uÛ£
           x0B0  0F 00 C1 2E 0F 00 04 1E 5A 33 DB B9 00 20 2B C8   ..Á.....Z3Û¹. +È
           x0C0  66 FF 06 11 00 03 16 0F 00 8E C2 FF 06 16 00 E8   fÿ.....‎‎ŽÂÿ...è
           x0D0  4B 00 2B C8 77 EF B8 00 BB CD 1A 66 23 C0 75 2D   K.+Èwï¸.»Í.f#Àu-
           x0E0  66 81 FB 54 43 50 41 75 24 81 F9 02 01 72 1E 16   f‎ûTCPAu$‎ù..r..
           x0F0  68 07 BB 16 68 52 11 16 68 09 00 66 53 66 53 66   h.».hR..h..fSfSf
           x100  55 16 16 16 68 B8 01 66 61 0E 07 CD 1A 33 C0 BF   U...h¸.fa..Í.3À¿
           x110  0A 13 B9 F6 0C FC F3 AA E9 FE 01 90 90 66 60 1E   ..¹ö.üóªéþ.‎‎‎f`.
           x120  06 66 A1 11 00 66 03 06 1C 00 1E 66 68 00 00 00   .f¡..f.....fh...
           x130  00 66 50 06 53 68 01 00 68 10 00 B4 42 8A 16 0E   .fP.Sh..h..´B‎Š..
           x140  00 16 1F 8B F4 CD 13 66 59 5B 5A 66 59 66 59 1F   ...‎‎‎‹ôÍ.fY[ZfYfY.
           x150  0F 82 16 00 66 FF 06 11 00 03 16 0F 00 8E C2 FF   .‎‚..fÿ.....‎‎ŽÂÿ
           x160  0E 16 00 75 BC 07 1F 66 61 C3 A1 F6 01 E8 09 00   ...u¼..faÃ¡ö.è..
           x170  A1 FA 01 E8 03 00 F4 EB FD 8B F0 AC 3C 00 74 09   ¡ú.è..ôëý‎‎‎‹ð¬<.t.
           x180  B4 0E BB 07 00 CD 10 EB F2 C3 0D 0A 41 20 64 69   ´.».Í.ëòÃ..A di
           x190  73 6B 20 72 65 61 64 20 65 72 72 6F 72 20 6F 63   sk read error oc
           x1A0  63 75 72 72 65 64 00 0D 0A 42 4F 4F 54 4D 47 52   curred...BOOTMGR
           x1B0  20 69 73 20 63 6F 6D 70 72 65 73 73 65 64 00 0D    is compressed..
           x1C0  0A 50 72 65 73 73 20 43 74 72 6C 2B 41 6C 74 2B   .Press Ctrl+Alt+
           x1D0  44 65 6C 20 74 6F 20 72 65 73 74 61 72 74 0D 0A   Del to restart..
           x1E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
           x1F0  00 00 00 00 00 00 8A 01 A7 01 BF 01 00 00 55 AA   ......‎‎Š.§.¿...Uª
```

משתני האתחול של ה-VBR נמצאים ב-82 בתים הראשונים בבלוק ומכילים מידע חשוב על המחיצה. המשתנים החשובים הם: גודל סקטור, גודל קלאסטר, מיקום תחילת טבלת ה-MFT וגודל מחיצה:

Sector	Boot sector (NTFS)					
x0008A800	Valid Boot Sector					
567,296	NTFS Signature:	NTFS		Physical drive #:	x80	128
	Bytes per sector:	x0200	512	Sectors in volume:	x00001BC0B021	465612833
	Sectors per cluster:	x08	8	1st MFT cluster:	x000C0000	786432
	Media descriptor:	xF8	248	1st MFT mirror cluster:	x00000002	2
	Sectors per FAT:	x0000	0	Clusters/file record:	x000000F6	246
	Sectors per track:	x003F	63	Clusters/index block:	x00000001	1
	Heads:	x00FF	255	Volume serial number:	x5A89D444	1518982212
	Hidden sectors:	x00000008A800	567296			

הקוד ב-bootmgr loader מכיל הודעות שגיאה שונות למטרות אתחול, ובעיקר תפקידו לטעון את קובץ ה-BOOTMGR של Windows ולהעביר את המשך העליה אליו. מכאן Windows נכנס לפעולה, יטען דרייברים של כתיבה/קריאה וימשיך לטעון את הקרנל:

Sector	Offset	Hex values	Ascii values
x0008A801	x000	07 00 42 00 4F 00 4F 00 54 00 4D 00 47 00 52 00	. . B . O . O . T . M . G . R .
567,297	x010	04 00 24 00 49 00 33 00 30 00 00 D4 00 00 00 24	. . $. I . 3 . 0 . . Ô . . . $
	x020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
	x030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
	x040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
	x050	00 00 00 00 00 00 E9 C0 00 90 05 00 4E 00 54 00	. . . . . é À . . . N . T .
	x060	4C 00 44 00 52 00 07 00 42 00 4F 00 4F 00 54 00	L . D . R . . . B . O . O . T .
	x070	54 00 47 00 54 00 07 00 42 00 4F 00 4F 00 54 00	T . G . T . . . B . O . O . T .
	x080	4E 00 58 00 54 00 00 00 00 00 00 00 00 00 00 00	N . X . T . . . . .
	x090	00 00 00 00 00 00 00 00 00 00 0D 0A 41 6E 20 6F	. . . . . A n o
	x0A0	70 65 72 61 74 69 6E 67 20 73 79 73 74 65 6D 20	p e r a t i n g s y s t e m
	x0B0	77 61 73 6E 27 74 20 66 6F 75 6E 64 2E 20 54 72	w a s n ' t f o u n d . T r
	x0C0	79 20 64 69 73 63 6F 6E 6E 65 63 74 69 6E 67 20	y d i s c o n n e c t i n g
	x0D0	61 6E 79 20 64 72 69 76 65 73 20 74 68 61 74 20	a n y d r i v e s t h a t
	x0E0	64 6F 6E 27 74 0D 0A 63 6F 6E 74 61 69 6E 20 61	d o n ' t . . c o n t a i n a
	x0F0	6E 20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74	n o p e r a t i n g s y s t
	x100	65 6D 2E 00 00 00 00 00 00 00 00 00 00 00 00 00	e m . . . . .
	x110	00 00 00 00 00 00 00 9A 02 66 0F B7 06 0B 00 66	. . . . . ▌ . f . · . . . f
	x120	0F B6 1E 0D 00 66 F7 E3 66 A3 52 02 66 8B 0E 40	. ¶ . . . f ÷ ã f £ R . f ▌ . @
	x130	00 80 F9 00 0F 8F 0E 00 F6 D9 66 B8 01 00 00 00	. € ù . . . . . ö Ù f ¸ . . . .
	x140	66 D3 E0 EB 08 90 66 A1 52 02 66 F7 E1 66 A3 86	f Ó à ë . . f ¡ R . f ÷ á f £ ▌
	x150	02 66 0F B7 1E 0B 00 66 33 D2 66 F7 F3 66 A3 56	. f . · . . . f 3 Ò f ÷ ó f £ V
	x160	02 E8 A2 04 66 8B 0E 4E 02 66 89 0E 26 02 66 03	. è ¢ . f ▌ . N . f ▌ . & . f .
	x170	0E 86 02 66 89 0E 2A 02 66 03 0E 86 02 66 89 0E	. ▌ . f ▌ . * . f . . ▌ . f ▌ .
	x180	2E 02 66 03 0E 86 02 66 89 0E 3E 02 66 03 0E 86	. . f . . ▌ . f ▌ . > . f . . ▌
	x190	02 66 89 0E 46 02 66 B8 90 00 00 00 66 8B 0E 26	. f ▌ . F . f ¸ . . . f ▌ . &
	x1A0	02 E8 90 09 66 0B C0 0F 84 BF FD 66 A3 32 02 66	. è . . f . À . ▌ ¿ ý f £ 2 . f

טבלת ה-MFT

זה הוא הרכיב המרכזי במערכת ה-NTFS. זו היא טבלה המורכבת מרשומות MFT, כאשר כל רשומה מתארת קובץ או תיקיה במערכת הקבצים. כל רשומה הינה בגודל של שני סקטורים כלומר 1024 בתים, אלא אם הוגדר אחרת ב-VBR.

כמו שכבר נאמר כל דבר בNTFS הינו קובץ ואף טבלת ה-MFT הינה קובץ אשר לה יש רשומה בתוך טבלת הMFT, למעשה הרשומה הראשונה בטבלה מצביעה לטבלה עצמה. הטבלה מתחילה בגודל קטן יחסית וגדלה לפי הצורך ככל שקבצים חדשים מתווספים לדיסק.

טבלת ה-MFT מתחילה לרב בקלאסטר 0xc0000 כאשר היסט זה הוא מתחילת המחיצה ונקבע במשתני אתחול של ה-VBR:

Sector	Name	Type	Attributes	Size
x00600000	**$MFT**	FILE	__sh__	262144
6,291,456	No: x0[x1] (x0), Parent directory: x5[x5], Run: 31:40 00 00 0C			
x00600002	**$MFTMirr**	FILE	__sh__	4096
6,291,458	No: x1[x1] (x1), Parent directory: x5[x5], Run: 11:01 02			
x00600004	**$LogFile**	FILE	__sh__	23265280
6,291,460	No: x2[x2] (x2), Parent directory: x5[x5], Run: 32:30 16 5A E9 0B			
x00600006	**$Volume**	FILE	__sh__	0
6,291,462	No: x3[x3] (x3), Parent directory: x5[x5], Run: Resident			
x00600008	**$AttrDef**	FILE	__sh__	2560
6,291,464	No: x4[x4] (x4), Parent directory: x5[x5], Run: 11:01 23			
x0060000A	.	DIR	__sh__	
6,291,466	No: x5[x5] (x5), Parent directory: x5[x5], Run: 11:01 24			
x0060000C	**$Bitmap**	FILE	__sh__	478048
6,291,468	No: x6[x6] (x6), Parent directory: x5[x5], Run: 31:75 8A FF 0B			
x0060000E	**$Boot**	FILE	__sh__	8192
6,291,470	No: x7[x7] (x7), Parent directory: x5[x5], Run: 11:02 00			
x00600010	**$BadClus**	FILE	__sh__	0
6,291,472	No: x8[x8] (x8), Parent directory: x5[x5], Run: Resident			
x00600012	**$Secure**	FILE	__sh__	
6,291,474	No: x9[x9] (x9), Parent directory: x5[x5], Run:			
x00600014	**$UpCase**	FILE	__sh__	131072
6,291,476	No: xA[xA] (xA), Parent directory: x5[x5], Run: 11:20 03			
x00600016	**$Extend**	DIR	__sh__	
6,291,478	No: xB[xB] (xB), Parent directory: x5[x5], Run:			
x00600018		FILE	__sh__	0
6,291,480	No: xC[xC] (xC), Parent directory: , Run: Resident			

קבצי Metadata בטבלת ה-MFT

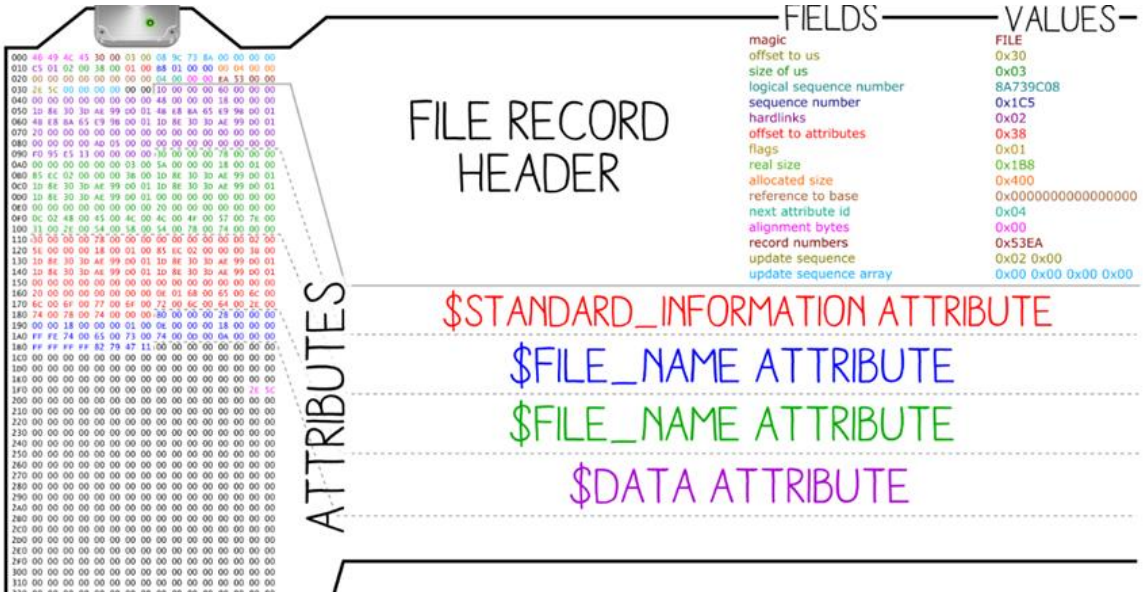
16 הרשומות הראשונות בטבלה מוקדשות לקבצי המערכת.

- **$MFT** - תאור טבלת ה-MFT
- **$MFTMirr** - מכיל גיבוי של מספר קבצי מטהדאטה עיקריים, למעשה ארבעת הקבצים הראשונים בטבלה.
- **$LogFile** - מכיל טרנזקציות שבוצעו על קבצים.
- **$Volume** - מכיל שם ותאור המחיצה.
- **$AttrDef** - מגדיר את ה-Attributes האפשריים לכל קובץ, גודלם ושמם.
- **$.** - מכיל את תיקיית השורש של המחיצה.
- **$Bitmap** - מערך המגדיר איזה קלאסטר תפוס, למשל אם ביט מספר 500 במערך דלוק אזי קלאסטר מספר 500 תפוס.
- **$Boot** - מכיל את ה-VBR.
- **$BadClus** - מכיל את הקלאסטרים שמוגדרים כלא תקינים.
- **$Secure** - מכיל אפשרויות של הרשאות גישה לקבצים.
- **$UPCase** - מכיל אות גדולה עבור כל תו Unicode.
- **$Extend** - הרחבות לקבצי מערכת כגון Reparse Point

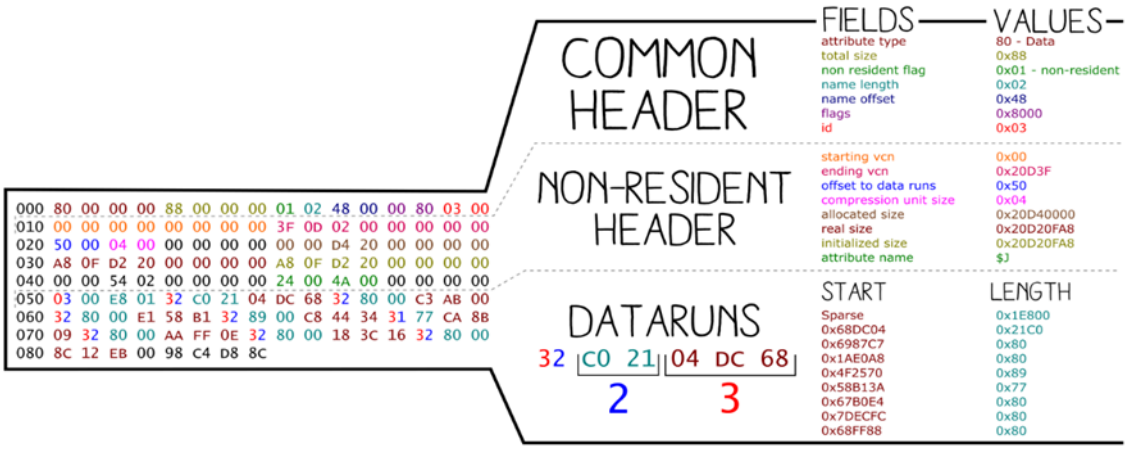
רשומת MFT

כל רשומת MFT היא סוג של מצביע לקובץ האמיתי או לחילופין מכילה בעצמה את תכולת הקובץ, ללא הצבעה. חשוב להבין שרשומות MFT אינן נמחקות אלא ממוחזרות, כלומר אם קובץ נמחק מהדיסק, רשומת ה-MFT שלו תתעדכן למצב מחוק אך לא תמחק בפועל.

כל רשומה מכילה Header ומספר attributes. כל attribute מכיל אף הוא header משל עצמו:

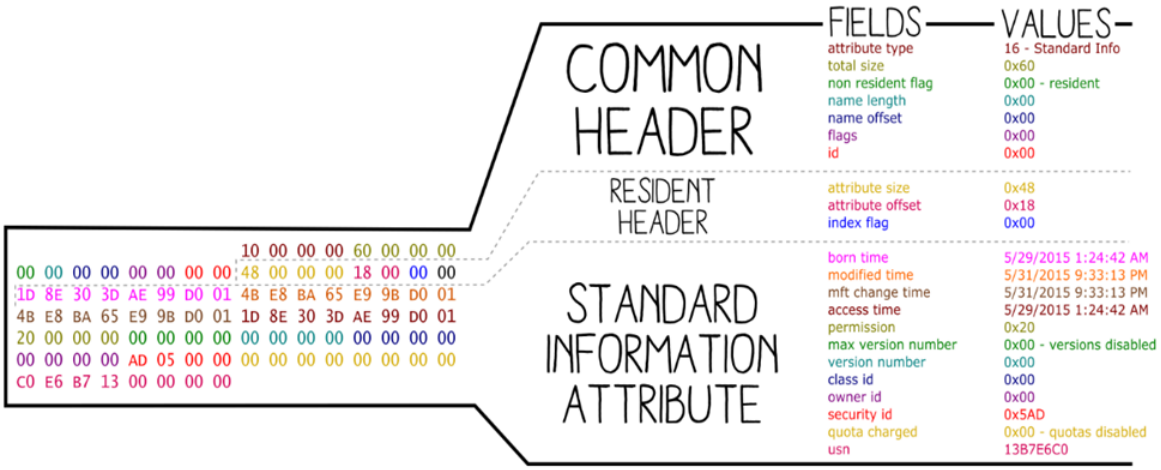


כל attribute מכיל בתחילתו תיאור של הטיפוס, גודל, והאם הוא מקומי או חיצוני. attribute מקומי משמעו כי כל הנתונים שלו נמצאים ברשומה עצמה. attribute חיצוני משמעו כי הנתונים שלו שמורים בקלאסטרים אחרים בדיסק; למשל תוכן של קובץ אשר גדול מכדי להישמר בתוך הרשומה עצמה, כמו באיור הבא:



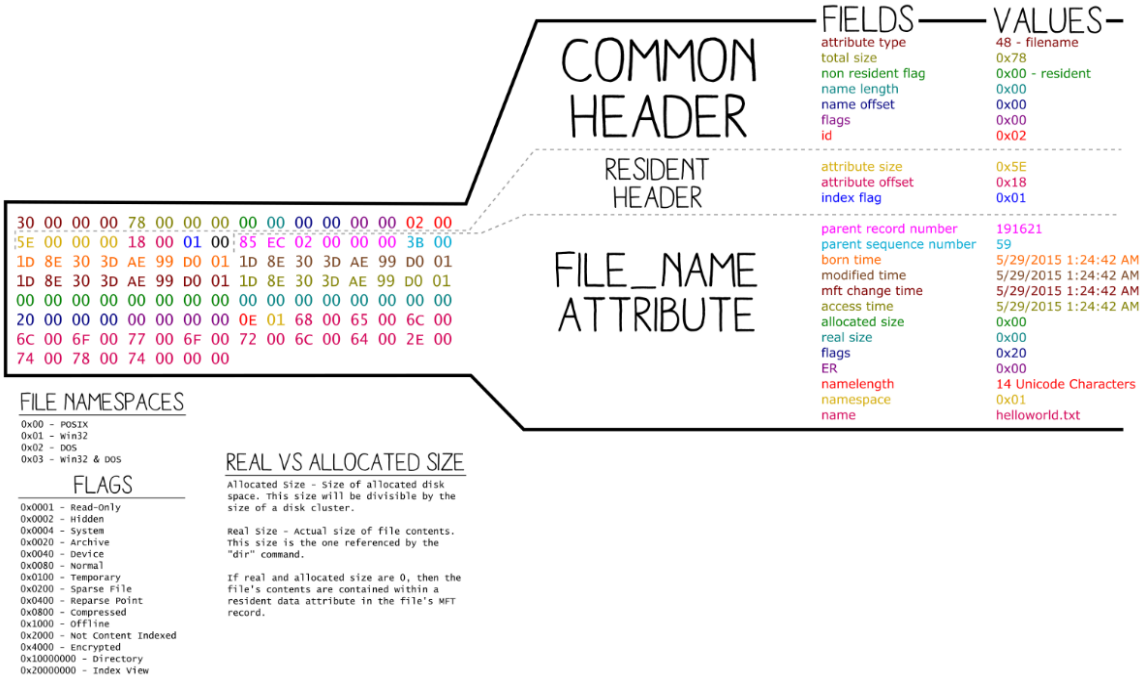
Standard Information Attribute

מכיל מידע כללי כגון דגלים, קריאה כתיבה ופתיחה אחרונה, הרשאות גישה. הוא נמצא באופן גורף בכל רשומות ה-MFT עבור קבצים וגם תיקיות. הוא נמצא תמיד ראשון בסדר הופעת ה-attributes מכיוון והסוג שלו הוא הנמוך ביותר 0x10.



File Name Attribute

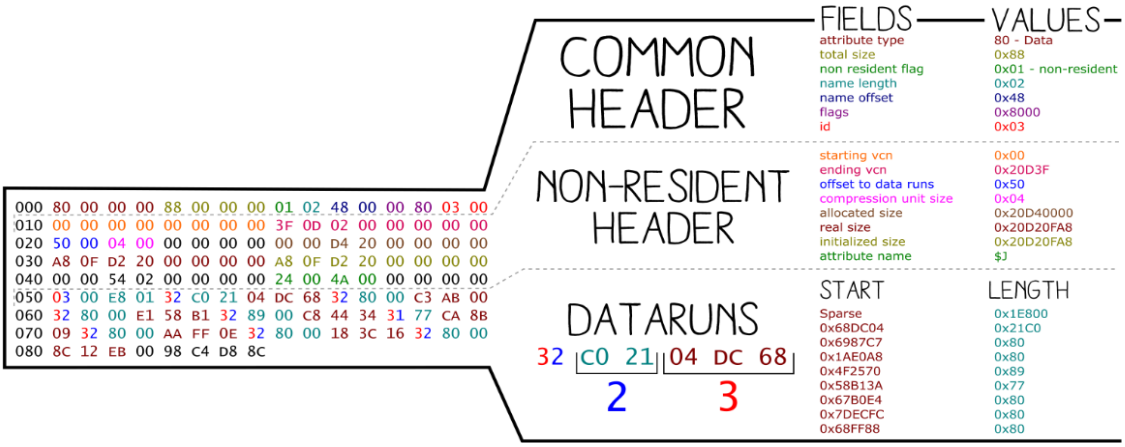
כל רשומת MFT תכיל לפחות attribute אחד כזה. מכיל את שם הקובץ ביוניקוד, סוג הקובץ, גודל הקובץ, גישה/כתיבה/קריאה אחרונה לקובץ (לא תמיד מעודכן), קישור לתיקיית האב.



Data Attribute

משמש לשמירת מידע מסוג כלשהוא. attribute זה יכול להופיע מספר פעמים לדוגמה במיקרה של ADS - Alternative Data Stream אנחנו נראה attribute עבור המידע של תוכן הקובץ ו-attribute נוסף ל-ADS.

(להרחבה - ראה את מאמרו של אפיק קסטיאל מגיליון מספר 17)

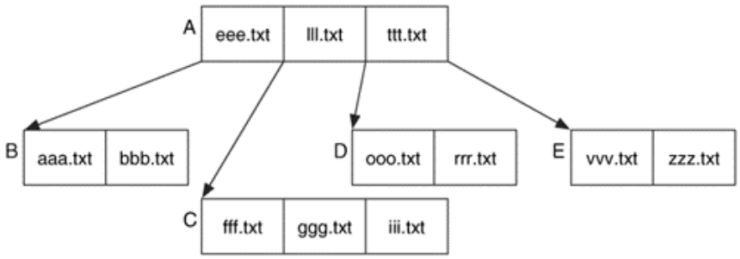


בדוגמה זו כאשר ה-attribute הינו חיצוני, נשמר מצביע אל הקלאסטרים אשר מכילים מידע זה. הקלאסטרים לא חייבים להיות רציפים לכן יתכנו מספר מצביעים. מצביע כזה נקרא DataRun כאשר המבנה של כל מצביע הוא כלהלן: הבייט הראשון אומר כמה בתים צריך לקרוא כדי לקבל את כמות הקלאסטרים וכמה בתים צריך לקרוא כדי לקבל את המיקום ההתחלתי של הקלאסטר הראשון. כלומר הבייט הראשון הוא זוג של מיקום וגודל. למשל באיור שבעמוד זה המידע יושב החל מקלאסטר 0x68dc04 וגודלו 0x21c0 קלאסטרים - כלומר כ-33mb.

בסעיף הבא נכיר עוד שני attributes וישנם עוד נוספים אשר יכולים להופיע ברשומת ה-MFT אך אלה פחות בשימוש ולא נרחיב עליהם כאן.

מבנה העץ של NTFS

המיבנה מאורגן ומוחזק כעץ מסוג B-TREE אשר מקל על אחסון וחיפוש של קבצים במערכת. העץ מוגדר כך שהילדים השמאליים של צומת קטנים מערך הצומת והילדים הימניים של צומת גדולים מערך הצומת. כך שהחיפוש מתבצע ביעילות מירבית. כל צומת יכול להחזיק מספר ערכים ומכאן למעשה נגזר שמספר הבנים של צומת הוא מספר הערכים בצומת + 1:

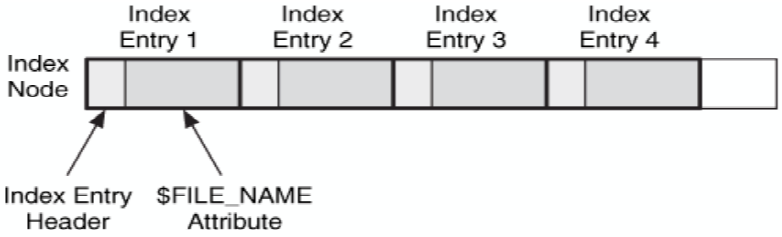


מכיוון והעץ נשמר מאוזן לאורך חיין, נוצר מצב בו הוספה או מחיקה של קובץ הינה פעולה מורכבת ויכולה לקחת זמן רב בגלל שיתכן וצריך להוריד או להוסיף צמתים כדי לשמור על האיזון. (שיעורי בית: נסו להוסיף לאיור העץ שבעמוד הקודם את הקובץ jjj.txt).

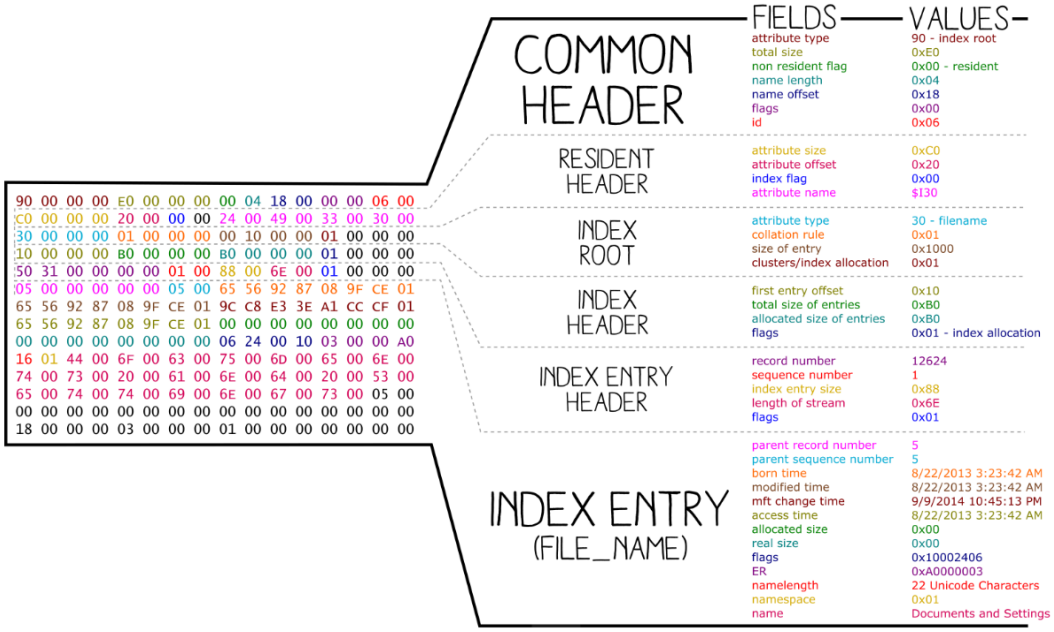
כמו שנאמר, כל צומת בעץ מכיל מספר ערכים, כל ערך כזה נשמר ב-index entry של הצומת.

צומת אינדקס בעץ מוגדר כאוסף של attributes אשר מאוחסנים באופן ממויין. שימוש העיקרי בצומת אינדקס נפוץ בהצגת תיקיה אשר מכילה למעשה attributes מסוג File_Name. המטרה לייצג בעץ את השמות של הקבצים ולא את התוכן הפיזי של הקבצים.

מבחינת NTFS כל צומת אינדקס (למשל תיקיה) מוגדר כרשומת MFT אשר מכילה attribute מסוג Index_Root. Attribute זה מכיל כניסות מסוג Index Entry. כמו באיור הבא:



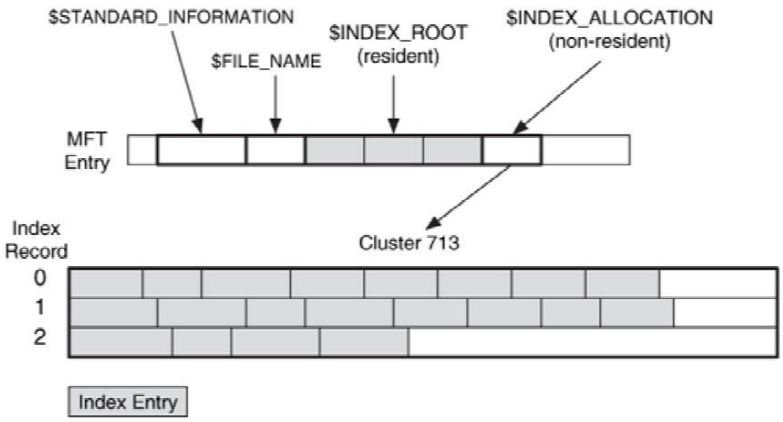
אפשר לזהות רשומה כזו במהירות ע"י עיון ב-header יופיע $I30 כיצוג אסקי בשדה attribute name. רשומה זו תהיה תמיד מקומית כלומר תכיל את ה-Index Entries בתוך הרשומה ולכן גודלה מוגבל בערך לכחצי מגודל רשומה כ-500 בתים, כלומר היא יכולה להחזיק מספר מועט יחסית של קבצים.



כל Index Entry מכיל דגל המסמל האם יש לו ילידים, אם כן המספר סידורי שלהם יופיע, מכיוון וה-index entries ממויינים בתוך הצומת אזי קל למצוא האם הערך נמצא בילדים.

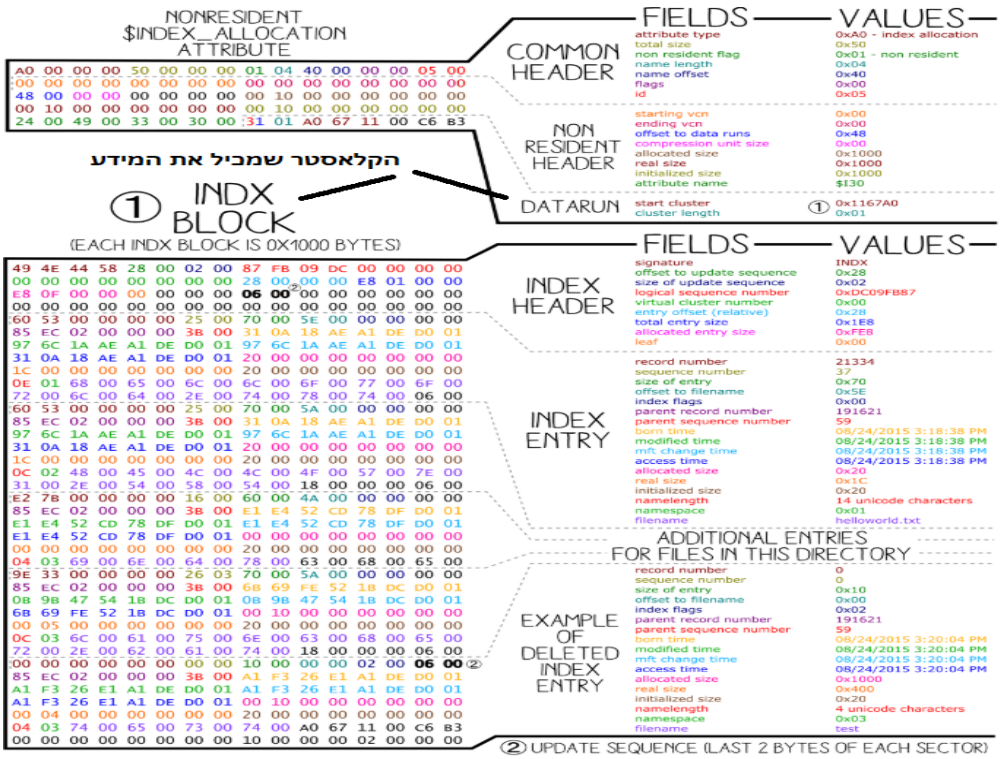
$Index Root

צומת אינדקס עם מספר גדול של קבצים יוגדר כרשומת MFT אשר מכילה attribute מסוג Index Allocation. רשומה זו תהיה תמיד חיצונית כלומר רשימת הקבצים שלה תשמר מחוץ לרשומה (אין מספיק מקום), באחד הקלאסטרים בדיסק. כמו באיור הבא:



קל לזהות את הקלאסטר שמכיל בלוק של אינדקסים ע"י מציאת חתימה - הייצוג האסקי INDX.

$Index_Allocation



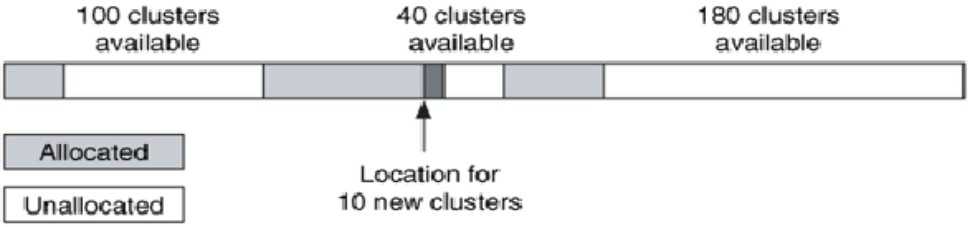
שימו לב לאיור - הדגל של קובץ מחוק מוגדר כ-0x2 לעומת דגל של קובץ רגיל שמוגדר 0. פרטי הקובץ אינם נמחקים ונשארים על הדיסק.

אלגוריתם הקצאת דאטה לקובץ חדש

נניח ויצרתם קובץ חדש או העתקתם קובץ חדש לדיסק אזי כפועל ישיר יש להקצות מקום עבור תוכן הקובץ. בזמן הקצאה האלגוריתם יחפש בטבלת ה-MFT את הרשומה הראשונה אשר מסומנת כמחוקה, אם מצא כזאת הוא ישתמש בה ואם לא מצא תוקצה רשומה חדשה והטבלה תגדל. כמו שהוסבר מקודם, אם תוכן הקובץ קטן יחסית, אזור ה-700 בתים, אז התוכן ישמר בתוך רשומת ה-MFT של הקובץ. אם התוכן גדול מכך, יתבצע אלגוריתם חיפוש והקצאה עבור המידע החדש.

אלגוריתם ההקצאה ישאף תמיד למצוא מיקום אופטימלי, כלומר אם הגודל שיש לשמור הוא למשל 4000 בתים אז המערכת תחפש קלאסטר בודד אשר נמצא איפשהו בין קלאסטרים תפוסים, גם אם זמן החיפוש יהיה ארוך יותר. יש העדפה לכתוב את המידע במצב של "סגירת חורים" מאשר לכתוב את המידע איפה שהוא בתוך 10 קלאסטרים פנויים רציפים שיכולים לשמש לשמירת קובץ גדול.

למשל בדוגמה הבאה רוצים להקצות מקום לקובץ בגודל 10 קלאסטרים ויש 3 chunks שאפשר לכתוב בהם את המידע. המידע יכתב במיקום הפנוי הכי קטן שהוא גדול מהכמות המינימלית הנדרשת.



חשוב להבין שלמעשה לכל קלאסטר במחיצה יש קישור חד ערכי מרשומת ה-MFT שלו (דרך ה-attribute $data), ואם אין קישור כזה אז המידע ימחק ע"י תוכנות כגון checkdisk. קלאסטר אשר מתפנה למעשה לא נמחק כלל אלא רשומת ה-MFT שלו פשוט מתעדכנת שקובץ זה כבר לא בשימוש. כזכור גם רשומת ה-MFT איננה נמחקת אלא ממוחזרת עבור קבצים עתידיים. בנוסף גם קובץ ה-$Bitmap מתעדכן ומסמן את הקלאסטר כפנוי כך שאלגוריתם ההקצאה יוכל לבדוק ביעילות אם קלאסטר מסויים תפוס.

דוגמה להקצאת קובץ

נרצה להקצות קובץ בתיקיית \home\test.txt כאשר גודל הקובץ 6000 בתים וגודל כל קלאסטר 4096 בתים.

1. קריאת ה-VBR מהסקטור הראשון במחיצה תביא לנו את כתובת תחיל טבלת ה-MFT.

2. תחילה נקצה MFT עבור הקובץ החדש. ההקצאה יכולה למחזר רשומה קיימת המסומנת כמחוקה או להוסיף בפועל רשומה חדשה. ישנו bitmap עזר ברשומת הMFT- הראשונה בטבלה אשר מסמל מי מהרשומות תפוסה. (לא להתבלבל עם ה-Bitmap של הקלאסטרים, רשומה מספר 6 בטבלת ה-MFT). ברגע שנבחרת רשומת MFT ה-bitmap מתעדכן.

3. נניח ומחזרנו רשומה קיימת, אזי תוכן הרשומה הישנה ימחק ויכתבו Standard_Info ו-File_Name חדשים לרשומה. הדגל של רשומה מחוקה יתעדכן לדגל של רשומה בשימוש.

4. בשלב הזה אנו צריכים להקצות שני קלאסטרים עבור תוכן הקובץ בעזרת Data attribute חיצוני חדש שיכנס לרשומה. בעזרת bitmap הקלאסטרים נמצא שני קלאסטרים פנויים רציפים אשר סוגרים חור של שני קלאסטרים במחיצה. (הקצאה אופטימלית). הדאטה יכתב לקלאסטרים, bitmap הקלאסטרים יתעדכן וב-Data attribute ברשומה ירשם מצביע לקלאסטרים (datarun).

5. בשלב הזה נרצה לשקף את הקובץ החדש בעץ ה-NTFS, על כן החיפוש בעץ יתחיל בשורש שלו - רשומה מספר 5 בטבלת ה-MFT. בעזרת קריאת ה-Index_Root וה-Index_Allocation של תיקיית השורש מבצעים חיפוש עד למציאת תיקיית היעד - להלן home. רשומת ה-MFT של תיקיית home תתעדכן בשדה של זמן גישה אחרון.

6. רשומת ה-MFT של תיקיית home תכיל אף היא attributes מסוג Index Root/Index Allocation ולפי כך ניתן בקלות למצוא לאיזה צומת בעץ ניתן להכניס את ה index entry אשר תייצג את קובץ היעד - test.txt. Index entry חדש יווצר ויוכנס לצומת והעץ יתאזן מחדש באם צריך. רשומת ה-index entry החדשה תכיל מצביע (record number) לרשומת ה-MFT שמייצגת את הקובץ test.txt.

7. בכל אחד מהשלבים הקודמים יווצרו רשומות בתוך קובץ המטה $LogFile ובז'ורנל השינויים ב-\UsrJrnl$\$Extend.

ניתן לחלק את התהליך לשני שלבים עיקריים - 1. חיפוש התנהגות אנומלית אצל הקבצים השונים. כמו שנראה בהמשך כלי השירות של מיקרוסופט checkdisk יודע לעשות עבודה טובה בתחום זה. 2. לשחזר את המידע החבוי, זהו תהליך לא פשוט כי בהרבה מיקרים הדאטה יהיה ללא מבנה מסויים או חתימות.

חילוץ בעזרת חתימות

ניתן לחפש את החתימות העיקריות שמעניינות אותנו כגון INDX – עבור צמתים בעץ, FILE – עבור רשומות ה-MFT. כיוון זה יכול להצליח כאשר אין הרבה פעילות בדיסק ואז בעצם נשארים עקבות שלא נדרסו שמהם אפשר אפילו לשחזר תיקיות שלמות, למשל בעזרת הצמתים של העץ אשר מכילים בלוקים שמתחילים ב INDX מבצעים bottom up reconstruction.

חילוץ מ-USN Journal

אחד ממקומות הלוג המרכזיים עבור הטרנזקציות שקרו במערכת הקבצים נמצא בקובץ המטהדאטה $Extend\$UsnJrnl . עבור כל טרנזקציה בקובץ נרשם תעוד של הסיבה והזמן. פירסור הלוג לא פשוט אבל יכול לתת הרבה אינדיקציות על מה שקרה במערכת הקבצים. לרב ישתמשו ב-לוג זה ביחד עם ה $LogFIle, רשומה מספר 3 בטבלת ה-MFT, כדי לקבל תמונה משלימה.

חילוץ מידע מהקרנל

נתחיל מהסוף: הקרנל של Windows שומר בזמן ריצה מספר views של הקבצים הפיזיים בדיסק, לאור זאת ניתן לחלץ מידע בעל ערך מדגימות זכרון ה-ram. אחד ממנגנוני הקרנל המרכזים הוא ה-cache manager אשר דואג לשמור בזכרון view של קבצים שניגשים אליהם הרבה. התהליך הזה מתבצע בעזרת מנגנון נוסף: ה-memory manager אשר אחד מיכולתיו הוא למפות section objects לקבצים והוא זה שקובע איזה חלק מהקובץ אכן נשאר בזכרון או ממופה החוצה - page out. מצד שני ה-cache manager ממפה את הקבצים שהוא צריך בעזרת אובייקט VACB אשר גודלו לרב 256KB.

ברמה הטיפה יותר גבוהה הקרנל מחזיק אובייקטים מסוג file_object אשר בעצם שקול לסוג של handle המוכר שפותחים בזמן גישה לקובץ. יש מספר שיטות למצוא בזכרון הקרנלי אובייקטי file_object לדוגמה סריקת ה-VADs אך לא ניכנס לכך במאמר זה.

לכל אובייקט file_object יש שדה SectionObjectPointer אשר מצביע למיפוי הקובץ/האם הקוד הוא דאטה או בר הרצה/אזור ה-cache הפעיל. שדה ה-SectionObjectPointer נמצא בשימוש גם ע"י ה- cache manager וה-memory manager. אחד מהשדות הנוספים ש-SectionObjectPointer מצביע אליו הוא מערך כניסות הדפים page table entry - PTE אשר ממנו אפשר לגזור באופן ישיר איזה דפים היו ממופים בזכרון לטובת מיפוי הקובץ ועם נתונים אלה אפשר לשחזר המון מידע של קבצים שהשתמשו בהם.

כמו שתואר קודם לכן, אחד המצביעים של SectionObjectPointer הוא מצביע לאזור ה-cache הפעיל - _SHARED_CACHE_MAP. ה-cache manager משתמש במבנה זה כדי לעקוב אחרי מצב המיפויים שלו דרך ה-VACBs שלו. מכיוון ומה שמעניין אותנו זה אזורי ה-cache שממופים לדפים בזכרון, paged in, אפשר לעבור על מערך ה-VACBs ולגזור באופן ישיר את הדפים הממופים (בזכרון ה-system cache) ואת ההיסט בקובץ עצמו שעבורו בוצע המיפוי.

VBR

כמו שתואר מעלה, ב-16 הסקטורים הראשונים נמצאים ה-VBR וה BooMgr Loader. חלק נכבד מהסקטורים האלה לא מאוכלס ולכן יש לבחון אזורים אלה. שיטה זו קלה לגילוי באבחון מהיר.

$BadClusters

רשומות MFT מסוג BAAD במקום FILE (ארבעה הבתים הראשונים ברשומה). יכול לשמש כממצא מחשיד מכיוון שהסבירות נמוכה מאד שמערכת ההפעלה תזהה קלאסטר פגום לפני הקושחה של הדיסק. לרב הדיסק יזהה קלאסטרים פגומים ויתקן אותם.

קלאסטרים יתומים

כל קלאסטר "פעיל" חייב להיות מוצבע ע"י רשומת MFT, דרך ה-Data attribute. רשומה זו צריכה להיות עם דגל של רשומה פעילה וגם MFT bitmap צריך לשקף זאת. קלאסטר יחשב כחשוד במידה ובמערך ה-Clusters bitmap הקלאסטר מסומן כתפוס אבל בפועל אין שום רשומת MFT פעילה שמצביעה אליו. אגב מצב כזה יזוהה מייד ע"י checkdisk ויתוקן.

Checkdisk יזהה עוד הרבה וריאציות של חוסר תאימות למשל:

1. רשומת MFT לא פעילה שמחזיקה הצבעה לקלאסטר - כמובן ששינוי זדוני כזה מסתכן בכך הרשומה פשוט תידרס/תמוחזר
2. קלאסטר יתום – אין רשומת MFT שמצביעה אליו וגם במערך ה Bitmap cluster אין סימון. קלאסטר כזה גם מסתכן בדריסה שרירותית בכל רגע נתון.
3. רשומת MFT לא פעילה עם דגל של רשומה פעילה במערך ה-MFT Bitmap
4. Attributes בתוך רשומה שלא תואמים את הסטנדרט
5. תוכן של attributes- למשל גודל שמוגדר ב-header אל מול הגודל האמיתי.

MFT Slack

רשומת MFT רגילה מגיע לגודל של כמה מאות בתים בהנחה וה-data attribute הוא חיצוני או פנימי ומכיל מידע מועט. מכיוון והגודל המקסימלי הוא 1024 בתים נוצר מקום עם שטח פנוי. שטח זה יכול להיות חשוד אך אין שום בטחון שהשטח זה לא ידרס למשל אם הקובץ גדל (התווסף מידע).

שיטות הסתרה ניתן לקרוא במאמר המעמיק של אור צ'צ'יק מגיליון 84.

סיכום

מערכת ה-NTFS הינה חלק מרכזי בכמעט כל פעולה שאנו מבצעים בזמן השימוש במחשב החל מתהליך העליה של מערכת ההפעלה ועד לכך שאתם מצליחים לקרוא את המאמר הנכחי. המערכת לא השתנתה הרבה ביחס לכמות השנים העצומה שהיא קיימת ומציגה פתח למחקרים רבים בתחום זה החל מפורנזיקה בסיסית, ניתוח יכולות ישנות שכבר לא בשימוש ועד לניצול חולשות בדרייבר ה-ntfs.sys. בכדי להבין קצת יותר ולחוש בידיים איך הדברים בנויים מומלץ מאד להוריד תוכנה אשר יודעת לפרסר מידע גולמי של מערכת הקבצים וכך לראות איך הכל עובד ברמות הנמוכות, למשל מה קורה כשמוסיפים מידע לקובץ ומה קורה כשמורידים מידע מקובץ וכו'.

על המחבר

דני אודלר עוסק בתחום מחקרי סייבר מגוונים - חקירת קרנל, פריסת קוד וחיפוש חולשות.
אשמח לקבל שאלות, הערות והארות לכתובת: danny.odler [kruhit] gmail.com

מקורות מידע

1. איורים 7,8,9,10,13,14:

https://github.com/Invoke-IR/ForensicPosters/tree/master/Posters/NTFS

2. איורים 11,12,15,16:

File System Forensic Analysis , Brian Carrier 2005

3. Linux-NTFS Project

https://flatcap.org/linux-ntfs

4. מידע שימושי על מערכת הקבצים:

http://www.ntfs.com/ntfs_basics.htm

5. מידע על checkdisk של Windows

https://en.wikipedia.org/wiki/CHKDSK