Data Recovery

Logical Recovery
Chapter 4
Written By
Ann LeFlore

What is Logical Recovery

- Logical recovery includes rebuilding files that have been corrupted by user or virus attacks, recover data from HDD with bad sectors, file deletion, partition deletion, and problems with the OS.
- File recovery: large files are allocated consecutive clusters DR software can rebuild the files with out the FAT or NTSF entry. Files are fragmented due to large size. Most files are assigned consecutive cluster.
- Large files such as databases, email, large text documents are easily fragmented due to their size.
- When file allocation has been overwritten by another file it is practically impossible to recovery data from these files. One possibility to recover files is the advanced technology MFM that reads the drive magnetization.

Hard Drive Bad Sectors

- Bad sectors are sectors on the HDD that can not be used due to permanent damage or the OS inability to successfully access these sectors.
- Physical disk surface damage results in sectors being stuck in a magnetic or digital state that can not be reversed. Programs have the ability to mark these sectors so the OS skips them in the future.
- The FW of the disk controller remaps logical bad sectors to a different sectors.
- Two types of remapping of bad sectors: One the P-List done by the factor and two the G-List done by the HDD microcode. A variety of program exist to read the HDD SMART information displaying how many sectors are reallocated and how many spare sectors on the disk surface.
- Bad sectors are caused by: one platter surface is failing, two corruption by the track data, three Read/Write heads are failing, and four system area is damaged from bad translation of the recorded medium.

Boot Sector

- Boot sectors or boot block is a sector on the HDD containing code for booting programs, and OS stored in other parts of the disk.
- The BIOS selects the boot device, it copies the first sector MBR, VBR or any executable code to the address location 0x7C00.
- Boot sectors describes special kinds of programs executed by the computer system immediately after power up or reset. The programs are stored in non-volatile memory such as Flash ROM.
- Boot block is the first program executed by the CPU, Flash ROM or NOR Flash, that have dedicated sectors that store the boot programs.
- Boot sectors refers to dedicated hardware type of sectors. The sectors have an extra level of protection to guard against accidental erase and overwrite to avoid scenarios when the system completely fails to boot at the beginning of the boot sequence.
- Boot Blocks stored in Flash ROM have a size from 1kiB to 512kiB which performs extra hardware initialization, minimal hardware test, and checks for sophisticated boot loaders, OS loader or other programs that can be executed at start up.

Boot Sector

- Award BIOS and AMI BIOS have a dedicated area about 8kiB called a boot block which start first, checks if the main BIOS are present then either starts the main BIOS or enters into a special recovery mode which can recovery the main BIOS from a floppy disk or other media. Other systems depending on their design may have a similar software called Boot Block.
- Boot block refers to the same program as the boot loader. There are no rules to define what should be called a boot block or what should be called a boot loader.
- Generally small and simple exercitation of the boot program are referred to as boot blocks and more sophisticated boot programs are referred to as boot loaders.
- Some ROM based system initialize their programs stored in memory using a boot ROM instead.
- There are several major types of boot sectors that can be encountered on storage devices

Boot Sector

- Master boot record MBR is the first sector on the HDD that has been partitioned. The MBR sector contains code to locate the active partition and execute the volume boot record.
- Volume boot record VBR is the first sector on the HDD that has not been partitioned or the first sector of an individual partition on a HDD that has been partitioned. It contains code to execute the OS, or stand alone programs installed on the HDD within the partition.
- To be a valid boot sector the two byte hex word 0xAA55 called the boot sector signature must be present at the end of the sector. Otherwise the BIOS or MBR code will report an error message and stop bootstrapping process.

Boot Sector

- CD ROM have their own structure for boot sectors, for IBM PC compatible systems the structure is subject to El Torito specifications.
- Non-IBM compatible system may have different boot sector formats on their disk devices.
- Boot sector term does not refer to a special type of NOR or Flash ROM sectors that is intended to store the initial boot programs. Boot sectors in Flash ROM are often implemented with extra protection to avoid accidental erase and over write to the boot block program.
- Boot sectors on a disk device are physical properties of sectors storing boot programs and have the same properties as the other sectors.

Boot Sector

- The BIOS are ignorant to the distinction between VBR, MBR and partitioning. The FW loads and runs the frist sector on the HDD.
- VBR are first sectors on floppy or USB flash drive. MBR is first sector on HDD.
- The code in the MBR understand the disk formatting and partitioning and is responsible for loading and running the VBR where the primary partition is set to boot. The VBR loads a second stage boot loader from another location on the disk.
- Whatever is stored in the first sector is not required to immediately load any bootstrap code fro the OS. The BIOS passes control as long as the sector meets the qualification signature of 0xAA55 in the last two bytes.

Master Boot Record

- The MBR is created when the first partition on the HDD is created. The MBR is the most important data structure on the HDD. The MBR is the first sector on every disk. The location is always track and cylinder 0,side and head 0 and sector 1
- MBR contains the partition table for the disk and a small amount of executable code.
- Executable code examines the partition table and identifies the system partition.
- The MBR finds the system partition starting location on the disk and loads copy of the partition boot sector into memory.
- The MBR transfers ownership to executable code in the partition table boot sector.

Master Boot Record

- The HEX dump of the MBR shows the sector is in two parts.
- The first part of the MBR which occupies the first 446 bytes of the sector. The disk signature (FD 4E F2 14) is at the end of the MBR code.
- The second part of the MBR is the partition table. Physical Sector: Cyl 0, Side 0, Sector 1

Volume Boot Record

- VBR known as volume boot sector, partition boot record or partition boot sector is a boot sector introduced by IBM PC.
- VBR are found on a partitioned data storage device such as HDD, or un-partitioned data storage device such as floppy and CD-ROM. The VBR contains the code for bootstrapping programs stored in other parts of the device.
- Non-partitioned storage devices it is the first sector on the device.
- Partitioned storage devices first sector on an individual partition where the first sector is the MBR.

Volume Boot Record

- The code in the VBR is invoked either by the machine's FW or code stored directly in the MBR. Code in VBR and MBR is loaded the same way.
- Invoking a VBR from boot manager is known as chain loading.
- Some dual boot process system like NTLDR take copies of the bootstrap code that individual OS install into a single partition VBR and store them in disk files loading the VBR from file after the boot loader has determined OS bootstrap.
- In file system FAT and NTSF the VBR contains a BOS parameter block which specifies the location and layout of the disk data structure for the file system

BIOS Parameter Block

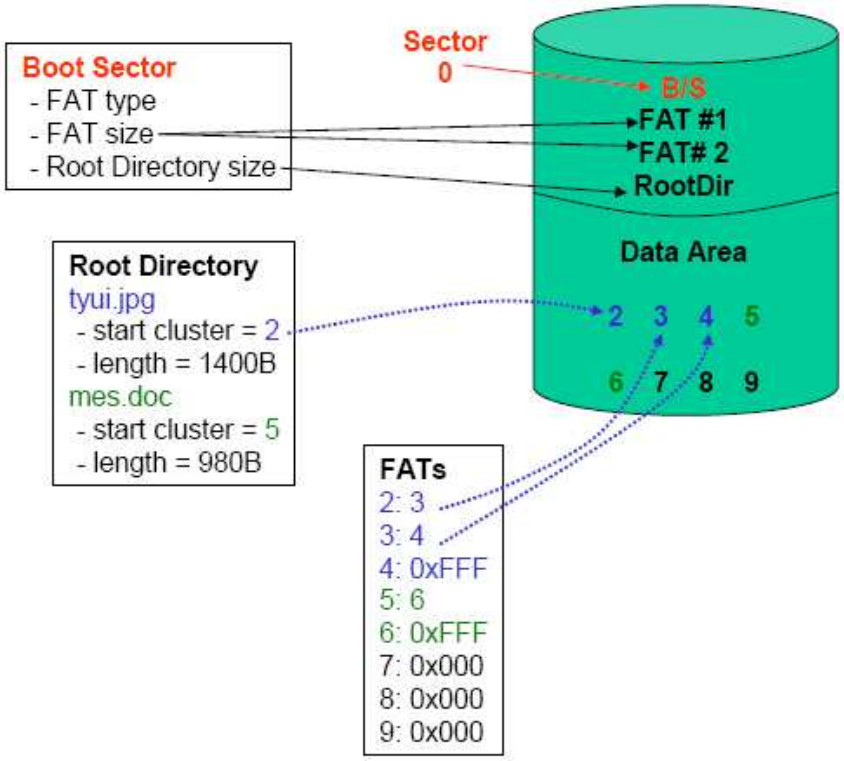
- BPR is the data structure in the VBR describing the physical layout of a data storage volume.
- Partitioned devices the BPB describes the volume partition
- Un-partitioned devices the BPB describes the entire medium
- Basic BPB appear and used on any partition – certain file systems also make use of the BPB to describe basic file system structures
- File systems making use of the BIOS parameter block include FAT16, FAT32, HPFS and NTSF – due to the different types of fields and amount of data they contain the length of the BPB is different for each file system

FAT File System

- FAT file allocation table organize the file system on the disk
- The table stores information about files on the HDD in the form of sequences and numbers defining where each part of each file is located.
- The first FAT as 12 digits and worked with diskettes and logic disks in volume no more than 16 MB.
- In MS-DOS version 3.0 FAT tables became 16 digits in order to support larger capacity disks.
- Disk volumes of 2 047 GBytes FAT 32 Bit tables are used.
- Data structures supported: Clusters, directories and file allocation table.

Boot Sector FAT, Root Directory and Files

- File tyui.jpg occupies clusters 2, 3, and 4 – the file size is 1,536 bytes = 3 clusters on the disk and cluster 4 includes 136 bytes of slack space.
- File mes.doc occupies clusters 5 and 6 – file size is 980 bytes = 2 clusters and has 44 bytes of slack space in cluster 6
- Cluster 7, 8, and 9 are un-allocated

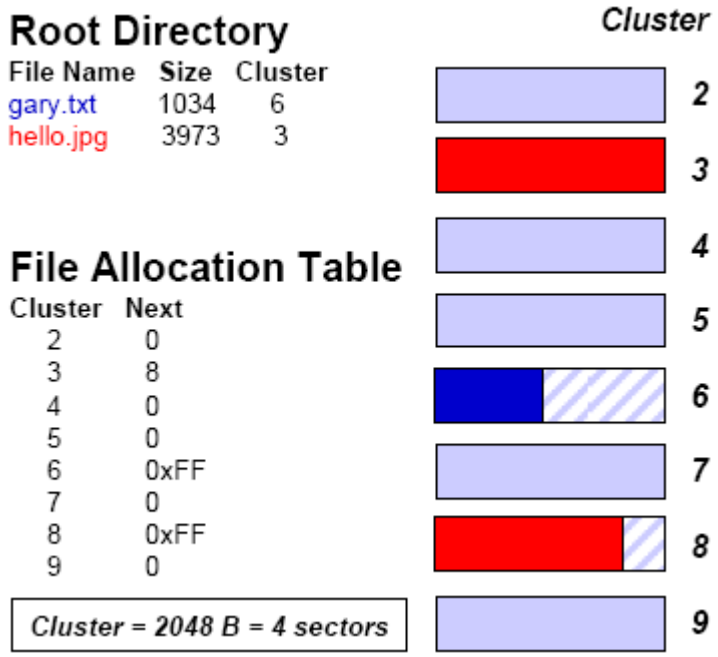


Fat Clusters and Sectors

- A cluster is a group of consecutive sectors
- A sector is usually 512 B
- A cluster is 1, 2, 4, 8, 16, 32, or 64 sectors that range from 512 B to 32 KB
- Each cluster has an address
- The first cluster has an address of 2
- There is no addressable cluster 0 or 1

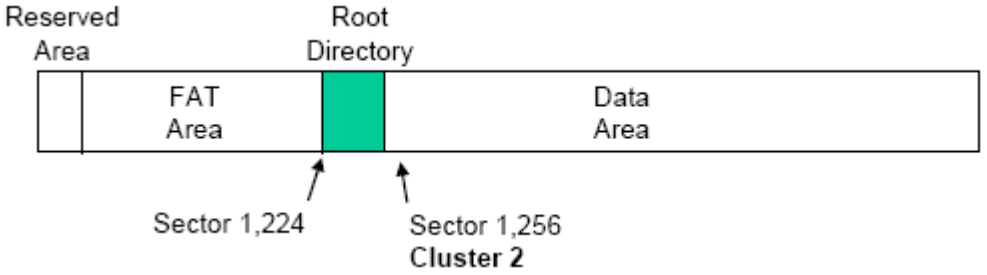
FAT, Slack and Unallocated Space

- Clusters 3, 6, and 8 are allocated; clusters 2, 4, 5, 7, and 9 are unallocated
- Clusters 6 and 8 are only partially filled; the unused portion is slack space
- File gary.txt:
  - logical size is 1,034 bytes
  - physical size is 2,048 bytes (slack = 1,014 B)
- File hello.jpg:
  - logical size is 3,973 bytes
- physical size is 4,096 bytes (slack = 123 B)



What is the First FAT Cluster

- The first cluster is Cluster 2
- Actual location of cluster 2 is different in FAT12/16 and FAT32
- Assume cluster size = 2,048 B (4 sectors)
- Assume that data area starts at sector 1224
- First sectors of data area are reserved for the Root Directory
  - Size is established at boot time
- Cluster 2 starts after Root Directory
- Root directory is set at 32 sectors
  - Occupies sectors 1,224-1,255
  - Cluster 2 starts at sector 1,256
  - Cluster 3 starts at sector 1,260
  - Cluster 4 at 1,264...



FAT Boot Sector

- First sector of a FAT system is the boot sector
  - Contains most of the information with which to determine
    - the file system type, and
    - size and location of data structures
- Boot sector format is different for FAT12/16 and FAT32

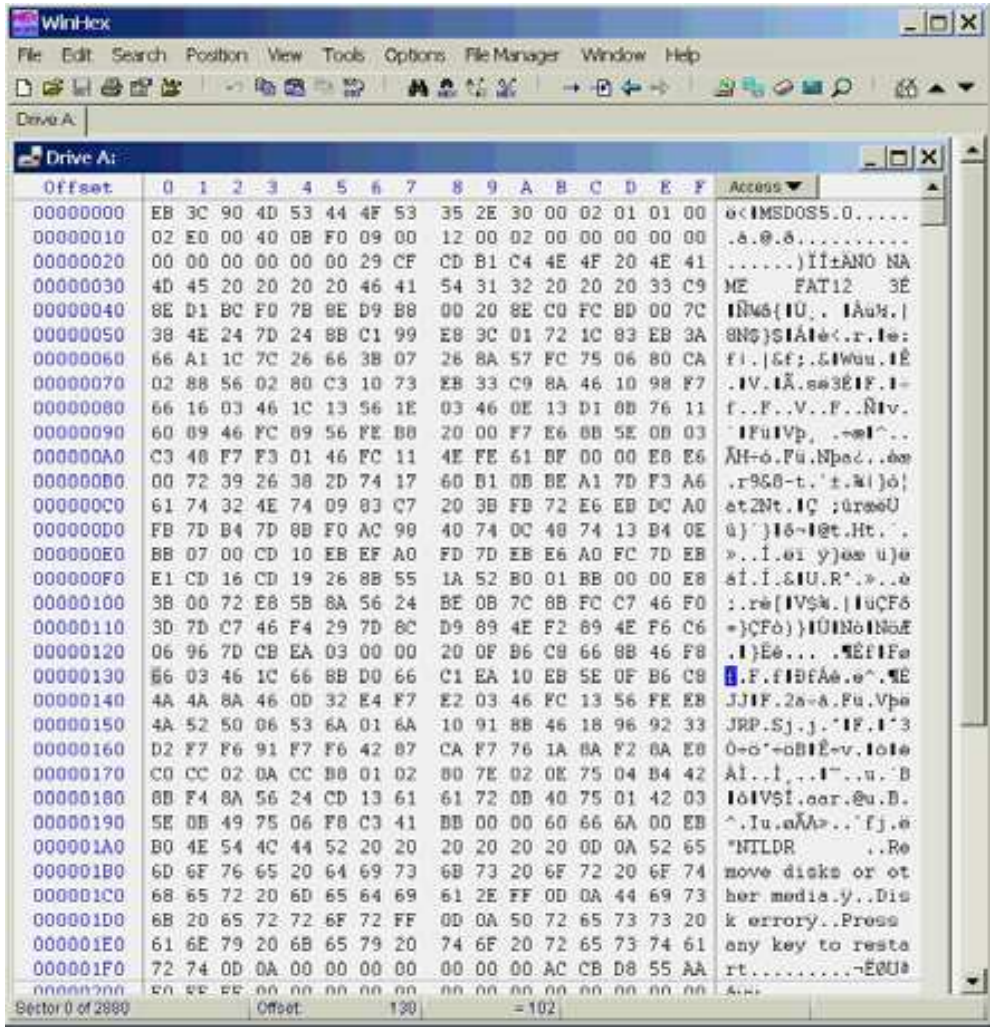
FAT Boot Sectors Bytes 0-35

Bytes	Purpose
0-2	Assembly code instructions to jump to boot code (mandatory in bootable partition_
3-10	OEM name in ASCII
11-12	Bytes per sector (512, 1024, 2048, or 4096)
13	Sectors per cluster (Must be a power of 2 and cluster size must be <=32 KB)
14-15	Size of reserved area, in sectors
16	Number of FATs (usually 2)
17-18	Maximum number of files in the root directory (FAT12/16; 0 for FAT32)
19-20	Number of sectors in the file system; if 2 B is not large enough, et to 0 and use 4B value in bytes 32-35 below
21	Media type (0xf0=removable disk, 0xf8=fixed disk)
22-23	Size of each FAT, in sectors, for FAT12/16; 0 for FAT32
24-25	Sectors per track in storage device
26-27	Number of heads in storage device
28-31	Number of sectors before the start partition
32-35	Number of sectors in the file system; the field will be 0 if the 28 files above (bytes 19-20) is non-zero

FAT Boot Sector (FAT12/16)

Bytes	Purpose
0-35	See FAT Boot Sector Bytes 0-35
36	BIOS INT 13h (low level disk services) drive number
37	Not used
38	Extended boot signature to validate next three fields (0x29)
39-42	Volume serial number
43-53	Volume label, in ASCII
54-61	File system type level, in ASCII (Generally “FAT”, “FAT12” or “FAT16”)
62-509	Not used
510-511	Signature value (0xaa55)

FAT12 Boot Sector



Boot Sector Interpretation

- 00-02: eb 3c 90 Instructions to jump to boot code
- 03-0a: 4d 53 44 4f 53 35 2e 30
- Name string (MSDOS5.0)
- 0b-0c: 00 02 Bytes/sector (0x0200 = 512)
- 0d : 01 Sectors/cluster (1)
- 0e-0f: 01 00 Size of reserved area (1 sector)
- 10 : 02 Number of FATs (2)
- 11-12: e0 00 Max. number of root directory entries (0x00e0 = 224)
- 13-14: 40 0b Total number of sectors (0x0b40 = 2,880)
- 15 : f0 Media type (removable)
- 16-17: 09 00 FAT size (0x0009 = 9 sectors)
- 18-19: 12 00 Sectors/track (0x0012 = 18)
- 1a-1b: 02 00 Number of heads (0x0002 = 2)
- 1c-1f: 00 00 00 00 Number of sector before partition (0)
- 20-23: 00 00 00 00 Total number of sectors (0 because 2B value not equal 0)
- 24 : 00 Drive number (0)
- 25 : 00 Unused
- 26 : 29 Extended boot signature
- 27-2a: cf cd b1 c4 Volume serial number (C4B1-CDCF)
- 2b-35: 4e 4f 20 4e 41 4d 45 20 20 20 20
- Volume label ("NO NAME ")
- 36-3d: 46 41 54 31 32 20 20 20
- File system type label ("FAT12 ")
- 3e-1fd : [snip] Not used
- 1fe-1ff: 55 aa Signature value (0xaa55)

FAT Capacity

- FAT12 allocates 12 bits per FAT entry
  - Limits addressing to 4,096 (212) clusters
- This (removable) device is configured so that:
  - 1 cluster = 1 sector
  - 1 sector = 512 B
- This FAT12 table is limited in capacity to 2,097,152 bytes (2 MB)
  - I.e., 4K clusters of 512 B each
- This device has 2,880 sectors
  - 512 B clusters yields a device capacity of 1.44 MB
  - Corresponds to what we would expect for a floppy

Sector Assignment

Sector	Address	Function
0	0x0000-0x1ff	Boot Sector
1-9	0x0200-0x13ff	File Allocation Table – primary
10-18	0x1400-0x25ff	File allocation Table – secondary
19-32	0x2600-0x41ff	Root Directory
33-2879	0x4200-0x167fff	File storage space

- Boot Sector is 1 sector (0x200 bytes)
- There are two FATs, each 9 sectors (0x1200 bytes)
- The Root Directory can contain 224 entries, each 32 bytes (7168, or 0x1c00, bytes; 14 sectors)
- File storage starts at sector #33 (1+9+9+14), byte #0x4200 (0x200+0x1200+0x1200+0x1c00)

Root Directory

- Contains file names and metadata
  - Located immediately after FAT(s) in FAT12/16 or in a location specified in the FAT32 boot sector
- Supports 8.3 names or long file names
- New entries are added to the directory using a first-available or next-available strategy
  - First-available: Finds first unallocated entry in the directory (e.g., Win98)
  - Next-available: Finds next available entry from the last allocated entry; at end of directory chain, start again at beginning (e.g., WinXP

Root Directory Entries

- The Root Directory is a series of entries describing files
- Each entry is 32 bytes and contains
  - single short (8.3) filename (SFN),
  - attributes,
  - MAC times,
  - start cluster,
  - size,
  - And other metadata.
  - Additional 32B entries contain the file's long filename (LFN)

Root Directory Entry Format SFN

Root Directory SFN Entry Data Structure	
Bytes	**Purpose**
0	First character of file name (ASCII) *or* allocation status (0x00=unallocated, 0xe5=deleted)
1-10	Characters 2-11 of the file name (ASCII); the "." is implied between bytes 7 and 8
11	File attributes (see File Attributes table)
12	Reserved
13	File creation time (in tenths of seconds)*
14-15	Creation time (hours, minutes, seconds)*
16-17	Creation date*
18-19	Access date*
20-21	High-order 2 bytes of address of first cluster (0 for FAT12/16)*
22-23	Modified time (hours, minutes, seconds)
24-25	Modified date
26-27	Low-order 2 bytes of address of first cluster
28-31	File size (0 for directories)

File Attributes	
Flag Value	**Description**
0000 0001 (0x01)	Read-only
0000 0010 (0x02)	Hidden file
0000 0100 (0x04)	System file
0000 1000 (0x08)	Volume label
0000 1111 (0x0f)	Long file name
0001 0000 (0x10)	Directory
0010 0000 (0x20)	Archive

** Bytes 13-22 are unused by DOS*

Example Root Directory

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000FF120	42	70	00	67	00	00	00	FF	FF	FF	FF	0F	00	D3	FF	FF
000FF130	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	00	FF	FF	FF	FF
000FF140	01	62	00	65	00	6C	00	69	00	6E	00	0F	00	D3	5F	00
000FF150	67	00	61	00	79	00	6C	00	65	00	00	00	2E	00	6A	00
000FF160	42	45	4C	49	4E	5F	7E	31	4A	50	47	20	00	96	CF	82
000FF170	FC	34	FC	34	00	00	31	B0	B6	32	8D	00	B6	6A	05	00
000FF180	42	6A	00	70	00	67	00	00	00	FF	FF	0F	00	56	FF	FF
000FF190	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	00	FF	FF	FF	FF
000FF1A0	01	6B	00	65	00	73	00	73	00	6C	00	0F	00	56	65	00
000FF1B0	72	00	5F	00	67	00	61	00	72	00	00	00	79	00	2E	00
000FF1C0	4B	45	53	53	4C	45	7E	31	4A	50	47	20	00	AB	CF	82
000FF1D0	FC	34	FC	34	00	00	31	B0	B6	32	E4	00	29	6A	05	00
000FF1E0	E5	54	00	68	00	75	00	6D	00	62	00	0F	00	A4	73	00
000FF1F0	2E	00	64	00	62	00	00	00	FF	FF	00	00	FF	FF	FF	FF
000FF200	E5	48	55	4D	42	53	20	20	44	42	20	26	00	C1	CF	82
000FF210	FC	34	FC	34	00	00	D3	8D	DC	34	3B	01	00	20	00	00

Three files shown here:
BELIN_~1.JPG @ offset 0xff160 (belin_gayle.jpg entry starts @ offset 0xff140)
KESSLE~1.JPG @ offset 0xff1c0 (kessler_gary.jpg entry starts @ offset 0xff1a0)
?HUMBS.DB @ offset 0xff200 (Thumbs.db; deleted)

FAT Comparison Table

Attribute	FAT12	FAT16	FAT32
Used For	Floppies; small hard drives	Small to large hard drives	Large to very large hard drives
Size of Each FAT Entry	12 bits	16 bits	28 bits
Maximum Number of Clusters	~4,096	~65,536	~268,435,456
Supported Cluster Sizes	512 B to 4 KB	2 KB to 32 KB	4 KB to 32 KB
Maximum Volume Size	16,736,256 B (16 MB)	2,147,123,200 B (2 GB)	~2^{41} B (2 TB)

The NTFS File System

- The NTSF boot record is loaded in memory location 0000:7C00 by the MBR code.
- The location 7C0Bh through 7C53h are filled with the NTSF BPB
- The next 303 bytes 7C54h through 7D82h contain sectors executable code
- The main program and various subroutines load the N.T.L.D.R. or Bootstrap sectors into memory
- The first 3 bytes are the Jump Instructions
- Only the first two bytes EB 52 have even been used to form the actual JMP instructions.
- The third byte 90h is the NOP instruction
- The next 8 bytes are the OEM ID or system name followed by the BPB
- The NTFS BPB has many fields in common with the FAT16 and FAT32 boot records: Bytes per Sector, Sectors per Clusters and others.

The NTFS File System

- The old Media Descriptor Byte F8 does not contain the System ID or the Volume Label Fields.
- In the NTFS Media Descriptor byte there is a number of system files: NTOSKRNL.EXE and BOOT.INI mentioned in the code which follows the Initial Boot sector.
- The old 4-byte Volume Serial Number has been replaced with a new eight-byte NTFS volume serial number
- The last 125 bytes of the Boot Records first sector contain an Error message.
- The Message Offset bytes and the word-sized signature ID of AA55h which is the HEX word for Intel 86x CPUs are stored in memory with the lowest-byte first and highest-byte last to make CPU processing quicker

The NTFS File System

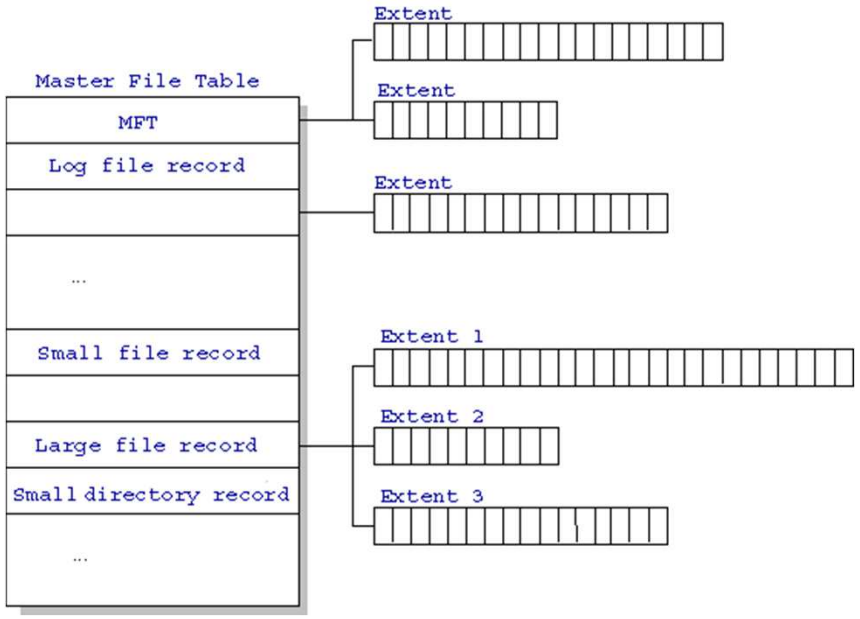
- The six physical sectors directly following the NTFS Boot Sector contain the code which interfaces with the NTLDR file in order to boot up the OS partition
- Code execution is pass from the Boot Sector of a bootable NTFS volume to offset A6h of the NTLDR Bootstrap code
- The bytes 8C and C8 comprise the first CPU instruction MOV, AX, CS, part of the Bootstrap code stores intermediate calculations and data about the partition.
- A JMP instruction at EB12 followed by 90 90 is a program execution and always jumps directly into the offset sector at 69Ah from the NTFS boot record
- The last sector of the Bootstrap code and the 7th sector of the entire NTFS Boot record where the HEX offsets are stored that start the NTLDR section; at 0D75h is the last byte of the code.
- The OS makes a backup copy of the NTFS VBR which is stored in the last sector of the partition. The total sector count in the NBR/ERR partition table is always 1 sector more than the total sectors in the volume count found in the boot record.

NTFS $Boot Sectors

- Windows 2000 and XP boot record contains a single sector plus the Bootstrap code consisting of seven sectors.
- All 16 sectors of the NTFS boot record area that are loaded into memory
- The first 16 bytes are the $Boot that occupy cluster 0 through 1 on systems with 8 sector clusters
- The 2nd sectors always begins with the 16 HEX byte Unicode for the 5 character NTLDR
- 3rd through 6th sectors have no outstanding features
- 7th sectors end with 138 zero byte
- Newly formatted NTFS volume $Boot is immediately followed by the $MFT that contains a number of sectors of FF bytes which are part of the $Bitmap, NTSF volumes contains Metadata system files in the middle of the partition. $MFTMirr is the backup copy of the first four $MFT and $LogFile records.

NTFS $Boot Sectors

- Each file in he NTFS volume is represented by a record that resides in the MFT
- NTFS reserves they first 16 records of the table for special information
- The 1st records describes the MFT
- The mirror records follows
- If the 1st record is corrupted MFT reads the 2nd records to find the MFT mirror file which record is identical to the 1st record in the MFT



NTFS $Boot Sectors

- The location for the data segments for both the MFT and the MFT mirror file are recorded in the boot sector.
- A duplicate boot sector is located at the logical center of the disk
- 3rd record of the MFT is the log file used for file recovery
- 17th record of the MFT are for each file and directory also view by the NTFS on the volume
- The MFT allocates a certain amount of space for each file record
- The attributes of the file are written to the allocated space in the MFT
- Small files and directories 1500 bytes or small can entirely be contained within the MFT
- Directory records are housed within the MFT just like file records.
- Instead of data, directories contain index information
- Small directory records reside entirely within the MFT structure
- Large directories are organized into B-trees having records with pointers to external clusters containing directory entries that could not be contained within the MFT structure

NTFS File Attributes

- The NTFS file system views each file and folder as a set of file attributes.
- Elements include file name, security information, and data
- Each attribute is identified by an attribute type code and an attribute name
- Resident attributes are file attributes that fit within the MFT file record
- Information like filename and time stamp is always included in the MFT file record
- When files are too large to fit into the MFT file record the attributes are non-resident
- Non-resident attributes are allocated one or more clusters of disk space somewhere in the volume
- NTFS creates the attribute list attribute to describe the location of all the attribute records

NTFS File Attributes

Attribute Type	Description
Standard Information	Includes information such as timestamp and link count
Attribute List	List the location of all attribute records that do not fit in the MFT record
File Name	A repeatable attribute for both long and short file names. The long name of the file can be up to 255 Unicode characters. The short name is the 8.3 case insensitive names for the file
Security Descriptor	Describes who owns the file and who can access it
Data	Contains file data. NTFS allows multiple data attributers per file Each file typically has one unnamed data attribute
Object ID	A Volume unique file identifier. Used by the distributed link tracking service. Not all files have object identifiers
Logged Tool Stream	Similar to a data stream, but operations are logged to the NTFS log file just like NTFS metadata changes. This is used by EFS
Reparse Point	Used for volume mount points. They are also used by Installable File Systems filter drivers to mark certain files as special to that driver
Index Root	Used to implement folders and other indexes
Index Allocation	Used to implement folders and other indexes
Bit map	Used to implement folders and other indexes
Volume Information	Used only in the #Volume system file. Contains the volume version
Volume Name	Used only in the $Volume system file. Contains the volume label

NTFS System Files

- NTFS includes several system files
- All are hidden from view on the NTFS volume
- A system file is one used by the file system to store its metadata and to implement the file system
- System files are places on the volume by the format utility

1. Master File table - $MFT contains one base file record for each file and folder on the NTFS volume.
2. Master File Table 2 - $MftMirr is a duplicate image of the first four records of the MFT
3. Log File - $LogFile contains a list of transaction steps used for NTFS recoverability
4. Volume - $Volume contains information about the volume such as the volume label and the volume versin
5. Attribute definitions - $AttrDef is the table of attribute names, numbers, and descriptions
6. Root file index name - $ is the root folder
7. Cluster bitmap - $Bitmap is a representation of the volume showing which cluster are in use
8. Boot sector - $Boot includes the BPB used to mount the volume and additional bootstrap loader code used in the volume bootable
9. Bad cluster file - $BadClus contains bad clusters for the volume
10. Security file - $Secure contains unique security descriptions for all files within a volume
11. Up case table - $Upcase converts lower case characters to matching Unicode uppercase characters
12. NTFS extension file - $Extend used for various optional extensions such as quotas, reparse point data, and object identifiers

NTFS Multiple Data Stream

- NTFS supports multiple data streams; the stream name identifies a new data attribute on the file
- Data streams are unique sets of file attributes
- Data streams have separate opportunistic locks, file locks, and sized but common permission
- This feature enables you to manage data as a single unit: myfile.dat:**stream2**
- A library of files can exist where the files are defined as alternate streams:
- library:file1
- :file2
- :file3

- A file can be associated with more than one application at a time
- program:source_file
- :doc_file
- :object_file
- :executable_file
- When you copy an NTFS file to a FAT Volume such as a floppy disk data streams and other attributes not supported by FAT are lost

NTFS Compressed Files

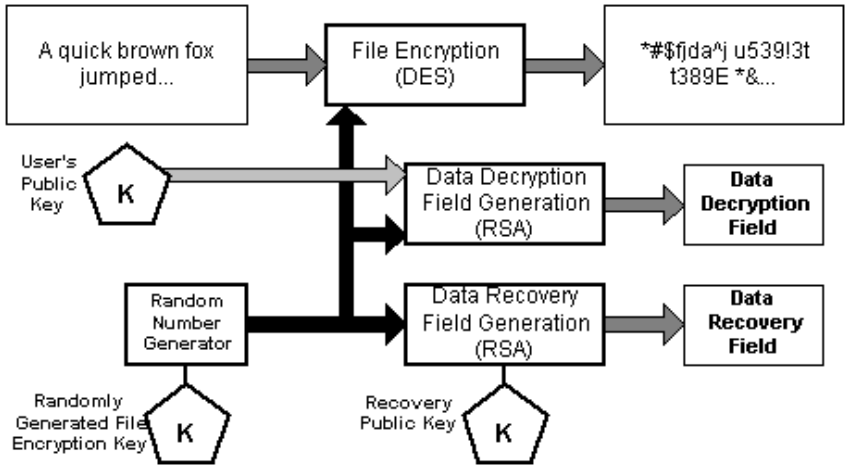
- File compressed on an NTFS volume can be read and written by any Windows-based application without first being decompressed by another program
- Decompression occurs automatically when the file is read.
- The file is compressed again when it is closed or saved
- The compression algorithms in NTFS are designed to support cluster sized up to 4 KB
- Sizes larger than 4 KB on the NTFS volume compression function are not available
- Each NTFS data stream contains information that indicates wheather any part of the stream is compressed

EFS Encrypting File System

- The EFS provides the core file encryption technology used to store encrypted files on NTFS volumes.
- Users work with files and folders just like they would with any file not encrypted
- The system automatically decrypts the files or folders when access is requested
- When file is saved it is encrypted again
- EFS features are invokes through Windows Explorer or by command line utility cipher.exe
- EFS uses symmetric key encryption in combination with public key technology to protect files
- File data is encrypted with symmetric algorithm DESX
- The key used is File Encryption Key FEK which is encrypted with a public and private key algorithm RSA and stored along with the file

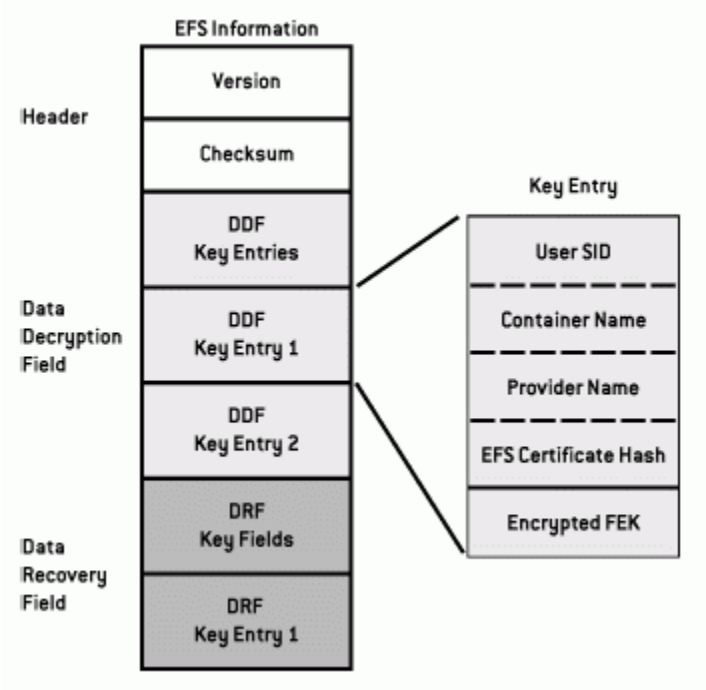
EFS Encrypting File System

- Encrypted file NTFS creates a log file Efs0.log in system volume information folder as the encrypted file. EFS uses a 1024-bit RSA algorithm to encrypt FEK
- EFS created Data Decryption Field DDF for the current user when it places the FEK and encrypts it with the public key
- If recovery agent is defined by the system policy EFS creates a Data Recovery Field DRF and places it with the FEK encrypted with public key for the recover agent
- A separate DRA is created for every recovery agent defined
- Temporary file Efs0.tmp is created when the file is being encrypted. Once encrypted the original file is over written and the temporary file is deleted.



$EFS Attribute

- When NTFS encrypts a file, it sets a flag Encrypted (0x4000) for the file and creates an $EFS attribute for the file where it stores the DDFs and DDRs. The attribute has an Attribute ID = 0x100 in NTFS and can be lengthy, occupying from 0.5K to several kilobytes depending on number of DDFs and DRFs



NTFS Boot Sector

Byte Offset	Field Length	Filed Name
0x00	3 bytes	Jump Instruction
0x03	LONGLONG	OEM ID
0x08	25 bytes	BPB
0x24	48 bytes	Extended BPB
0x54	426 bytes	Bootstrap Code
0x01FE	WORD	End of Sector Marker

NTFS Boot Sector

- NTFS Volume the data fields that follow the BPB form an extended BPB
- Data in the fields enables the Ntldr to find the MFT during start up
- MFT are not located in a predefined sector
- A boot sector in a NTFS volume formatted looks like:

- Bytes 0x00- 0x0A are the jump instruction and the OEM ID (shown in bold print).
- Bytes 0x0B-0x53 are the BPB and the extended BPB.
- The remaining code is the bootstrap code and the end of sector marker (shown in bold print).

HFS File System

Feature	HFS	HFS Plus	Benefit/Comment
User visible name	Mac OS Standard	Mac OS Extended	
Number of allocated blocks	16 bits worth	32 bits worth	Radical decrease in disk space used on large volumes, and a larger number of files per volume
Long file name	31 characters	255 characters	Obvious user benefit; also improves cross-platform compatibility
File name encoding	Mac Roman	Unicode	Allows for international-friendly file names including mixed script names
File/folder attributes	Support for fixed size attributes File Info and Extended Field Info	Allows for future metadata extensions	Future systems may use metadata for a richer finder experience
OS start-up support	System Folder ID	Also supports a dedicated start up file	May help non-Mac OS systems to boot from HFS Plus volumes
Catalog node size	512 bytes	4 Kb	Maintains efficiency in the face of the other changes and larger catalog records
Maximum file size	2^{31} bytes	2^{63} bytes	Obvious user benefit especially for multimedia content creators

HFS Terminology

- Most of the data structures on an HFS Plus volume do not depend on the size of a sector with the exception of the journal
- Journals relay on accessing individual sectors the sector size is stored in the jhdr_size field of the journal header
- HFS allocates space in units called allocation block
- Allocation blocks are a group of consecutive bytes
- The size in bytes of an allocation block is a power of two greater than or equal to 512 which is set when the volume is initialized
- Allocation blocks are identified by a 32-bit allocation block number
- There can be 2^{32} allocation blocks on a volume
- Current implementation of the file system are optimized for 4K allocation blocks
- All of the volume structure including the volume header are part of one or more allocation blocks
- To avoid fragmentation disk space is typically allocated to files in groups of allocation blocks or clumps
- Clumps are multiples of allocation block size

HFS Terminology

- HFS Plus volume must have a volume header which contains information about the volume: date, time, volume creation and number of files on the volume
- Volume header is located at 1024 bytes from the start of the volume
- Alternate volume header is stored starting at 1024 bytes before the end of the volume
- The first 1024 bytes of volume before volume header and the last 512 bytes after the alternate volume header are reserved
- All allocation blocks containing the volume header and alternate volume header are marked as used in the allocation file.
- The actual number of allocation blocks marked this way depends on the allocation block size
- HFS Plus volume contains 5 special files that store the file system structures required to access the file system payload, folders, user files and attributes
- The special files are catalog file, extents overflow file, allocation file, attributes file and start up file

HFS Terminology

- Catalog file describes the folder and file hierarchy on a volume
- Attributes file contain additional data for a file or folder
- Extent is a range of allocation blocks allocated to some fork represented by a pair of numbers – the first eight extents of each fork are stored in the volume catalog file and additional extents are stored in the extents overflow file
- Allocation files specifies whether an allocation block is used or free and performs the same role as the HFS volume bitmap
- Bad block file prevents the volume from using certain allocation blocks because the portion of the media that stores those blocks is defective – it is neither a special file or user file

Recommended Logical Recovery Programs

- Active Partition Recovery: http://www.partition-recovery.com/
- Acronis Migrate Easy: http://www.acronis.com/homecomputing/products/migrateeasy/index.html
- Acronis Disk Director Suit: http://www.acronis.com/homecomputing/products/diskdirector/
- Active Partition and File Recovery: http://www.partition-recovery.com/
- Active File Recovery: http://www.file-recovery.net/
- Get Data Back NTSF: http://www.runtime.org/data-recovery-software.htm
- Get Data Back FAT: http://www.runtime.org/data-recovery-software.htm
- RStudio: http://rstudio.org/
- File Scavenger: http://www.quetek.com/prod02.htm
- MHDD: http://hddguru.com/software/2005.10.02-MHDD/mhdd_manual.en.html
- Victoria: http://www.freenew.net/windows/victoria-hdd-utility-43/30961.htm
- Doc Regenerator: http://www.doc-regenerator.com-http.com/
- Partition Find and Mount: http://findandmount.com/
- Partition Table Doctor: http://www.ptdd.com/
- RAID Reconstruction: http://www.runtime.org/raid.htm
- USF Explorer: http://www.ufsexplorer.com/
- XLS Regenerator: http://xls-regenerator.en.softonic.com/
- WinHex: http://x-ways.net/winhex/
- Test Disk: http://www.cgsecurity.org/wiki/TestDisk