Computer Forensics Tutorial
Disk File Systems (FAT16, FAT32, NTFS)

José M. Rodríguez Justiniano
Computer Science
Jeffrey L. Duffany, Ph.D.
Computer Science Department
Polytechnic University of Puerto Rico

Abstract *— This tutorial is intended as in-class laboratory exercise for computer forensics classes at the Polytechnic University of Puerto Rico. It's specifically designed to provide basic understanding on the functionalities and capabilities of the tree most used file systems FAT16, FAT32, and NTFS. This document provides an inside or raw view of the files systems structure and how it handles data. It first covers the creation of a lab environment using openly available applications and the use of Hexadecimal Editors or Disk Editors to view and modify data.*

Key Terms *— Electronic Data, Forensics, File Systems, Hex Editor, Storage Device, Tutorial.*

INTRODUCTION

As defined a File System is a means to organize data expected to be retained after a program terminates by providing procedures to store, retrieve and update data, as well as manage the available space on the device which contains it. File systems are used on data storage devices such as hard disk drives, floppy disks, optical discs, or flash memory storage devices to maintain the physical location of the computer files.

The purpose of this tutorial is to provide a basic understanding of the most used files systems in the industry FAT16, FAT32 and NTFS. File Systems are normally an abstraction on the Operating Systems side, by using openly available software we can develop a lab environment that can be used to create and show the inner workings of the file systems.

The tutorial is designed around three free and openly available applications and they are:

- **Oracle Virtual Box:** is a general-purpose full virtualizer for x86 hardware, targeted at server, desktop and embedded use. Refer to Figure 1.

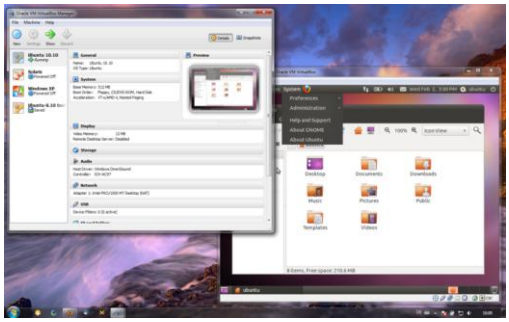

Figure 1
Oracle Virtual Box

- **HxD:** is a carefully designed and fast hex editor which, additionally to raw disk editing and modifying of main memory (RAM), handles files of any size. Refer to Figure 2.

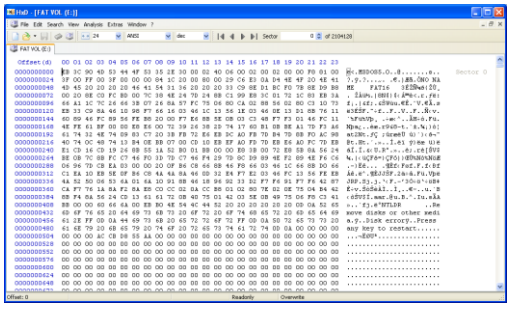

Figure 2
HxD Hex Editor

- **FTK Imager:** is a data preview and imaging tool that lets you quickly assess electronic evidence to determine if further analysis, can also create perfect copies (forensic images) of computer data without making changes to the original evidence. Refer to Figure 3.

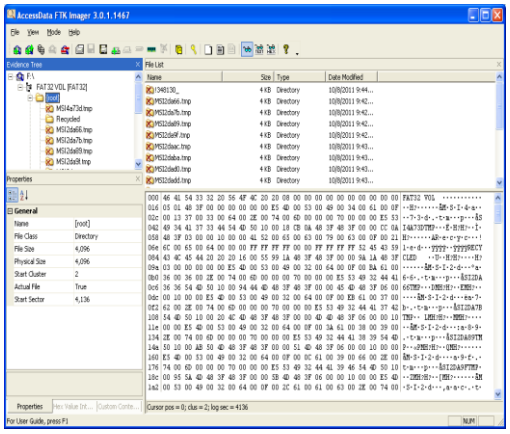

Figure 3
FTK Imager

DISK FILE SYSTEMS

A file system is the way in which files are named and where they are placed logically for storage and retrieval. The DOS, Windows, OS/2, Macintosh, and UNIX-based operating systems all have file systems in which files are placed somewhere in a hierarchical (tree) structure. A file is placed in a directory or subdirectory at the desired place in the tree structure.

File systems specify conventions for naming files. These conventions include the maximum number of characters in a name, which characters can be used, and, in some systems, how long the file name suffix can be. A file system also includes a format for specifying the path to a file through the structure of directories.

LAB ENVIRONMENT SETUP

Lab environment is created so that there is minimal impact on the user's workstation. By using Oracle Virtual Box we can create a virtual machine that has multiple hard drives therefore creating 3 partitions respectively with FAT16, FAT32 and NTFS file systems. Since is a best practice to work with images instead of working directly with production data, we use FTK Imager to create images of the newly created partitions. After the images are created we can also use FTK Imager to mount the image files.

FILE ALLOCATION TABLE (FAT16/32) FILE SYSTEMS

FAT16 was designed for Hard Drives that were larger than 16MB. It uses a 16-Bit Cluster addressing system that allows for Hard Drives sizes up to 4GB.

It was used by later MS-DOS versions (Earlier ones using FAT12), as well as Early Windows Versions.

FAT16 has a maximum File Size of 4GB, and a Maximum Volume Size of 4GB (On MS-DOS, and Windows 9x, they only support up to 32KB Clusters, making the maximum Volume Size they can support 2GB). Maximum Number of files on a FAT16 Volume is 65536, with the maximum viewable number of files and folders in the Root is 512.[1]

FAT32 was created to supersede FAT16, and was introduced by Windows 95 OSR2. It uses a 32-Bit Addressing System for Disk Clusters.

Maximum file size on a FAT32 Volume is 4GB (A Problem now being encountered for those with DVD Images, as FAT32 cannot handle them is above 4GB), but the Maximum Volume Size is under debate.

Windows XP will only format FAT32 Volumes up to 32GB; however, other utilities will theoretically format FAT32 Volumes up to 8 Terabytes in size. However, if a FAT32 Primary Partition is greater than 8GB, then there is no guarantee that it will be bootable.

Maximum number of files on a FAT32 Volume is 4,177,920, with the maximum number of Files and Folders standing at 65,534 per folder. [2]

FAT Root Directory, sometimes referred to as the Root Folder, contains an entry for each file and directory stored in the file system. This information includes the file name, starting cluster number, and file size. This information is changed whenever a file is created or subsequently modified. Root directory has a fixed size of 512 entries on a hard disk and the size on a floppy disk depends. With FAT32 it can be stored anywhere within the

partition, although in previous versions it is always located immediately following the FAT region.

The primary task of the File Allocation Tables is to keep track of the allocation status of clusters, or logical groupings of sectors, on the disk drive. There are four different possible FAT entries: allocated (along with the address of the next cluster associated with the file), unallocated, end of file, and bad sector.

In order to provide redundancy in case of data corruption, two FATs, FAT1 and FAT2, are stored in the file system. FAT2 is a typically a duplicate of FAT1. However, FAT mirroring can be disabled on a FAT32 drive, thus enabling any of the FATs to become the Primary FAT. This possibly leaves FAT1 empty, which can be deceiving.

Data Area: The Boot Record, FATs, and Root Directory are collectively referred to as the System Area. The remaining space on the logical drive is called the Data Area, which is where files are actually stored. It should be noted that when a file is deleted by the operating system, the data stored in the Data Area remains intact until it is overwritten.

Clusters: In order for FAT to manage files with satisfactory efficiency, it groups sectors into larger blocks referred to as clusters. A cluster is the smallest unit of disk space that can be allocated to a file, which is why clusters are often called allocation units. Each cluster can be used by one and only one resident file. Only the "data area" is divided into clusters, the rest of the partition is simply sectors. Cluster size is determined by the size of the disk volume and every file must be allocated an even number of clusters. Cluster sizing has a significant impact on performance and disk utilization. Larger cluster sizes result in more wasted space because files are less likely to fill up an even number of clusters.

The size of one cluster is specified in the Boot Record and can range from a single sector (512 bytes) to 128 sectors (65536 bytes). The sectors in a cluster are continuous; therefore each cluster is a continuous block of space on the disk. Note that only one file can be allocated to a cluster. Therefore if a 1KB file is placed within a 32KB cluster there are 31KB of wasted space. The formula for determining clusters in a partition is (# of Sectors in Partition) - (# of Sectors per Fat * 2) - (# of Reserved Sectors)) / (# of Sectors per Cluster).

Wasted Sectors (a.k.a. partition slack) are a result of the number of data sectors not being evenly distributed by the cluster size. It's made up of unused bytes left at the end of a file. Also, if the partition as declared in the partition table is larger than what is claimed in the Boot Record the volume can be said to have wasted sectors. Small files on a hard drive are the reason for wasted space and the bigger the hard drive the more wasted space there is. [3] Figure 4 illustrates the file system structure for FAT16 and FAT32, both are extremely similar.

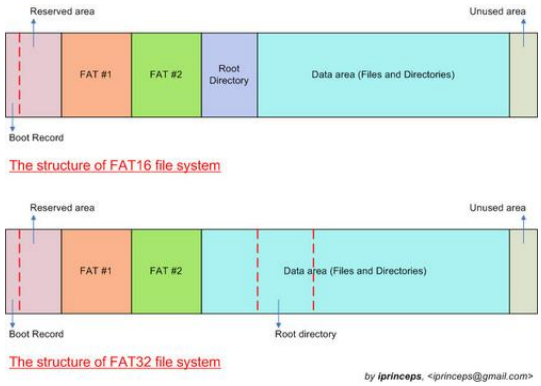

Figure 4
Structure of FAT16 and FAT32 File Systems

Boot Sector on non-partitioned devices, e.g., floppy disks, the boot sector is the first sector. For partitioned devices such as hard drives, the first sector is the Master Boot Record defining partitions, while the first sector of partitions formatted with a FAT file system is again the FAT boot sector.

Fragmentation: the FAT file system does not contain mechanisms which prevent newly written files from becoming scattered across the partition. Other file systems, e.g., HPFS, use free space bitmaps that indicate used and available clusters, which could then be quickly looked up in order to find free contiguous areas. Another solution is the linkage of all free clusters into one or more lists (as is done in Unix file systems). Instead, the FAT has

to be scanned as an array to find free clusters, which can lead to performance penalties with large disks.

Table 1
Common Boot Sector Structure used by all FAT versions[6]

Byte Offset	Length (bytes)	Description
0x00	3	Jump instruction. This instruction will be executed and will skip past the rest of the header if the partition is booted from.
0x03	8	OEM Name (padded with spaces 0x20). This value determines in which system disk was formatted. Common examples are IBM3.3, MSDOS5.0, mkdosfs, and FreeDOS.
0x0B	2	Bytes per sector; the most common value is 512. The BIOS Parameter Block starts here.
0x0D	1	Sectors per cluster. Allowed values are powers of two from 1 to 128.
0x0E	2	Reserved sector count. The number of sectors before the first FAT in the file system image. At least 1 for this sector, usually 32 for FAT32.
0x10	1	Number of file allocation tables. Almost always 2; RAM disks might use 1.
0x11	2	Maximum number of FAT12 or FAT16 root directory entries. **0** for FAT32, where the root directory is stored in ordinary data clusters.
0x13	2	Total sectors
0x15	1	Media descriptor 0xF0: 3.5" Double Sided 5.25" Double Sided 0xF8: Fixed disk (i.e. Hard disk) 0xF9: 3.5" Double sided, 5.25" Double sided, 0xFA: 5.25" Single sided, 0xFB: 3.5" Double sided, 0xFC: 5.25" Single sided 0xFD: 5.25" Double sided, 0xFE: 5.25" Single sided 0xFF: 5.25" Double sided, Same value of media descriptor should be repeated as first byte of each copy of FAT. Certain operating systems
0x16	2	Sectors per File Allocation Table for FAT12/FAT16, 0 for FAT32
0x18	2	Sectors per track for disks with geometry, e.g., 18 for a 1.44MB floppy
0x1A	2	Number of heads for disks with geometry, e.g., 2 for a double sided floppy
0x1C	4	Count of hidden sectors preceding the partition that contains this FAT volume. This field should always be zero on media that are not partitioned.
0x20	4	Total sectors (if greater than 65535; otherwise, see offset 0x13)

Formula's to assist in the location of important areas on the File System FAT16 and FAT32.

*Root Sectors = Root Directory Entries * 32 / Bytes Per Sector*

*FAT Sectors = Number of FATs * Sectors Per FAT*

Data Sectors = Total Sectors – (Reserved Sectors + FAT Sectors + Root Sectors)

Total Clusters = Data Sectors / Sectors Per Cluster

To locate the start sector of the data area you can use the following formula:

Data Area Start = Reserved Sectors + FAT Sectors + Root Sectors

Root Directory Details; FAT12 and FAT16 volumes have the root directory located immediately following the file allocation tables. The following formula gives you the starting sector number for the root directory:

Root Starting Sector = Reserved Sectors + FAT Sectors

On FAT32 volumes the root directory is made up of an ordinary cluster chain. A field in the Boot Record will tell you the initial cluster number. Once you've got the initial cluster number you can easily get the starting sector number for FAT32 as well:

*Root Starting Sector = ((Root Cluster – 2) * Sectors Per Cluster) + Data Area Start*

Now that we have a general understanding of FAT16 and FAT32 we can use HxD Hex editor to view the inner workings of the file systems. Figure 5 illustrates the first 36 bytes of the FAT file system all in sector 0 of the drive. Table 1 can also be used to identify the different parts of that section.

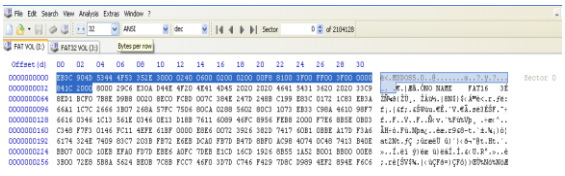

Figure 5
Boot Sector Common Structure of FAT16 and FAT32

With the information selected in Figure 2 and using Table 1 we can find the following:

OEM Name = MSDOS5.0

Bytes per Sector = 512 bytes

Sectors per cluster = 64 Sectors

With the information gathered we can determine the cluster size of the file system.

Cluster Size = (Bytes per Sector) x (Sectors per cluster) = (512 bytes) x (64 Sectors) = 32,768 bytes = 32KB

All of this data can be gathered using automated tools but the point of this exercise is to look at the raw data of the file system and understand it.

NTFS FILE SYSTEM

The Windows NT file system (NTFS) provides a combination of performance, reliability, and compatibility not found in the FAT file system. It is designed to quickly perform standard file operations such as read, write, and search - and even advanced operations such as file-system recovery - on very large hard disks.

Formatting a volume with the NTFS file system results in the creation of several system (metadata) files such as $MFT - Master File Table, $Bitmap, $LogFile and others, which contains information about all the files and folders on the NTFS volume.

The first information on an NTFS volume is the Partition Boot Sector ($Boot metadata file), which starts at sector 0 and can be up to 16 sectors long. This file describes the basic NTFS volume information and a location of the main metadata file - $MFT. [4]. Figure 6 illustrates the NTFS file structure.

Figure 6
Layout of an NTFS Volume

The NTFS file system includes security features required for file servers and high-end personal computers in a corporate environment. The NTFS file system also supports data access control and ownership privileges that are important for the integrity of critical data. While folders shared on a Windows NT computer are assigned particular permissions, NTFS files and folders can have permissions assigned whether they are shared or not. NTFS is the only file system on Windows NT that allows you to assign permissions to individual files.

The NTFS file system has a simple, yet very powerful design. Basically, everything on the volume is a file and everything in a file is an attribute, from the data attribute, to the security attribute, to the file name attribute. Every sector on an NTFS volume that is allocated belongs to some file. Even the file system metadata (information that describes the file system itself) is part of a file.

The NTFS file system views each file (or folder) as a set of file attributes. Elements such as the file's name, its security information, and even its data, are all file attributes. Each attribute is identified by an attribute type code and, optionally, an attribute name. See Table 2.

Table 2
Lists all of the File Attributes currently defined by the NTFS File System

Hex Code	Attribute Type	Description
0x10	Standard Information	Includes information such as timestamp and link count.
0x20	Attribute List	Lists the location of all attribute records that do not fit in the MFT record.
0x30	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. Additional names, or hard links, required by POSIX can be included as additional file name attributes.
0x50	Security Descriptor	Describes who owns the file and who can access it.
0x80	Data	Contains file data. NTFS allows multiple data attributes per file. Each file typically has one unnamed data attribute. A file can also have one or more named data attributes, each using a particular syntax.
0x40	Object ID	A volume-unique file identifier. Used by the distributed link tracking service. Not all files have object identifiers.
0x100	Logged Utility 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.
0xC0	Reparse	Used for volume mount points. They are

	Point	also used by Installable File System (IFS) filter drivers to mark certain files as special to that driver.
0x90	Index Root	Used to implement folders and other indexes.
0xA0	Index Allocation	Used to implement folders and other indexes.
0xB0	Bitmap	Used to implement folders and other indexes.
0x70	Volume Information	Used only in the $Volume system file. Contains the volume version.
0x60	Volume Name	Used only in the $Volume system file. Contains the volume label.

When a file's attributes can fit within the MFT file record, they are called resident attributes. For example, information such as filename and time stamp is always included in the MFT file record. When all of the information for a file is too large to fit in the MFT file record, some of its attributes are nonresident. The nonresident attributes are allocated one or more clusters of disk space elsewhere in the volume. If all attributes cannot fit into one MFT record NTFS creates additional MFST records and puts the Attribute List attribute to the first file's MFT record to describe the location of all of the attribute records.

When you format an NTFS volume, the format program allocates the first 16 sectors for the $Boot metadata file. First sector, in fact, is a boot sector with a "bootstrap" code and the following 15 sectors are the boot sector's IPL (initial program loader). To increase file system reliability the very last sector an NTFS partition contains a spare copy of the boot sector.

BIOS parameter block, often shortened to BPB, is a data structure in the Volume Boot Record describing the physical layout of a data storage volume. On partitioned devices, such as hard disks, the BPB describes the volume partition, whereas, on unpartitioned devices, such as floppy disks, it describes the entire medium.

The following table (Table 3) describes the fields in the BPB and the extended BPB on NTFS volumes. The fields starting at 0x0B, 0x0D, 0x15, 0x18, 0x1A, and 0x1C match those on FAT16 and FAT32 volumes.

Table 3

BPB and the extended BPB

Byte Offset	Field Length	Field Name
0x0B	WORD	Bytes Per Sector
0x0D	BYTE	Sectors Per Cluster
0x0E	WORD	Reserved Sectors
0x10	3 BYTES	always 0
0x13	WORD	not used by NTFS
0x15	BYTE	Media Descriptor
0x16	WORD	always 0
0x18	WORD	Sectors Per Track
0x1A	WORD	Number Of Heads
0x1C	DWORD	Hidden Sectors
0x20	DWORD	not used by NTFS
0x24	DWORD	not used by NTFS
0x28	LONGLONG	Total Sectors
0x30	LONGLONG	Logical Cluster Number for the file $MFT
0x38	LONGLONG	Logical Cluster Number for the file $MFTMirr
0x40	DWORD	Clusters Per File Record Segment
0x44	DWORD	Clusters Per Index Block
0x48	LONGLONG	Volume Serial Number
0x50	DWORD	Checksum

TFS includes several system files, all of which 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 placed on the volume by the Format utility. [5].

Now that we have a general understanding of NTFS file system we can use HxD hex editor to view the raw data of the file system. Figure 7 illustrates an NTFS partition with the BPB of the the Boot sector that according with Table 3 are bytes 0x0B to 0x50.

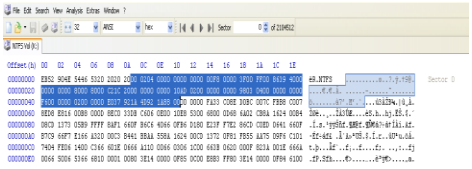

Figure 7

BPB of the Boot Sector

Using Table 3 we can find the following:

Sector Size has a value of 0x0200 (512 DEC) = Sector Size of 512 Bytes.

Sectors per cluster has a value of 0x0004 (04 DEC) = 4 Sectors per cluster.

$MFT's location should be in byte 0x30 and it's an 8 byte address.

The Hex Value is 0x000000000002AD10 = 175,376. $MFT is located at cluster 175,376.

With the information gathered we can determine the sector in which $MFT is located:

*Location of $MFT = Sectors per cluster * Starting Cluster of $MFT = 4 * 175,376 = 701,504*

$MFT is located at Sector 701,504.

All of this data can be gathered using automated tools but the point of this exercise is to look at the raw data of the file system and understand it.

CONCLUSION

This tutorial should be considered as a good recourse in the understanding of FAT16, FAT32 and NTFS file systems. Since Digital Forensics is an evolving field, knowing the inner workings and structure of the main file systems help a great deal in the process of data gathering and recovery. Giving the reader the necessary knowledge to understand how data is stored and handled by the different file systems.

REFERENCES

[1] "FAT16 - Computing Knowledgebase", Retrieved on Oct 19, 2011, http://pc.wikia.com/wiki/FAT16.

[2] "FAT32 - Computing Knowledgebase", Retrieved on Oct 19, 2011, http://pc.wikia.com/wiki/FAT32.

[3] "FAT - Forensics Wiki", Retrieved on Oct 19, 2011, http://www.forensicswiki.org/wiki/FAT#FATs.

[4] "NTFS.com File System Structure, Recovery Software, Hard Disk Internals.", Retrieved on Oct 21, 2011, http://www.ntfs.com/.

[5] "NTFS - Wikipedia, the free encyclopedia", Retrieved on Oct 21, 2011, http://en.wikipedia.org/wiki/NTFS.

[6] "File Allocation Table", Retrieved on Oct 21, 2011, http://www.isdaman.com/alsos/protocols/fats/nowhere/FAT.HTM.