This work is licensed under a Creative Commons license

Attribution-NonCommercial-NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)

You are free to:

Share copy and redistribute the material in any medium or format.

Under the following terms:

Attribution You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial You may not use the material for commercial purposes.

NoDerivatives If you remix, transform, or build upon the material, you may not distribute the modified material.

The FAT File System Family

Giovanni Lagorio

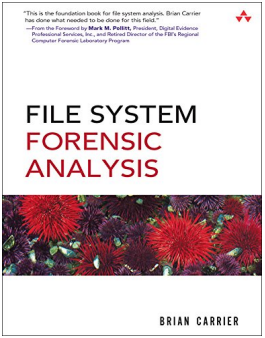
giovanni.lagorio@unige.it
https://csec.it/people/giovanni_lagorio
X/GitHub/...: zxgio

DIBRIS - Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
University of Genova, Italy

Outline

File System Forensics Analysis

The "bible" for this part of the course is Brian Carrier's "File System Forensics Analsysis" [Car05], where you can find more details



Introduction

First versions of FAT were developed in 1977/1978 (!), FAT12 in 1980

- No ACLs, no quota, no journal
- Still very common; lightweight and suited for embedded solutions

Three versions: FAT12, FAT16 and FAT32

- The number indicates the # of bits used to identify clusters
  - FAT12 can address $2^{12} = 4096$ clusters. Windows permits cluster sizes from 512 bytes to 8 KB, which limits FAT12 to 32 MB
  - FAT16 can address $2^{16} = 65,536$ clusters
  - FAT32 can address 2^{28} clusters (top 4 bits used for other purposes)

  Actually, first 2 & last 16 are reserved: usable clusters are slightly less
- Timestamps are in local time
- Filenames are in MSDOS 8.3 format
  - VFAT: a backward compatible extension allowing Unicode long names
- File sizes are stored as 32-bit integers

exFAT/FAT64

exFAT/FAT64

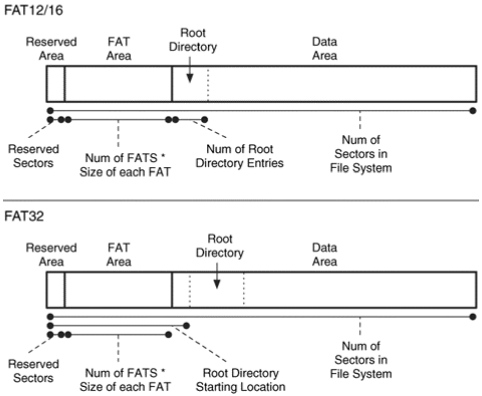
- introduced in 2006, but specification published in 2019
- some advanced functionality (e.g. optional ACLs), can be used where NTFS is unfeasible
- a bitmap tracks free clusters, improving the performance of allocation and deletion operations
- specifically designed for flash drives
- as the name implies, file sizes stored as 64 bit integers

We'll study the "classic" FAT family

Outline

Organization

- The VBR contains the so-called BIOS Parameter Block
- The root directory of FAT12/16 has a fixed location and size
- FAT32 boot sector includes the locations of the root directory, FSINFO structure (that keeps track of free clusters, to optimize allocations), and boot-sector backup (should be 6)



Boot sector FAT12/16

FAT16 Boot Sector

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	Jump Instruction			OEM ID								Bytes / sect		sect / cluster	reserved sectors	
10	no / FATS	Root entries		Small Sectors		Media descr iptor	Sectors / FAT		Sectors / Track		Number / heads	Hidden sectors				
20	large Sectors				Physical drive number	rese rved	ext boot sig	Volume Serial Number				Volume Label (deprecated)				
30	Volume label						File System Type									
40	OS Boot Code															
50																
60																
70																
...																
1D0																
1E0																
1F0															55	AA

https://www.writeblocked.org/resources/FAT_cheatsheet.pdf

"small sectors"/"large sectors"="# of sectors": only one is used, the other is 0
In green the Bios Parameter Block, in yellow the BPB extended

Root entries and *Sectors/FAT* (AKA size in sectors) are 0 for FAT32...

Boot sector FAT32

FAT32 Boot Sector

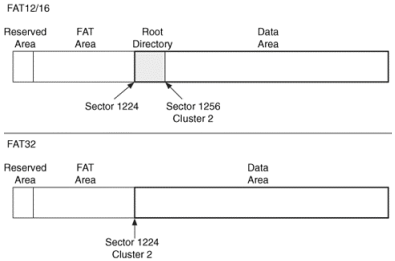
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	Jump Instruction			OEM ID								Bytes / sect		sect / cluster	reserved sectors	
10	no / FATS	0x0000		0x0000		Media descriptor	0x0000		Sectors / Track		Number / heads	Hidden sectors				
20	large Sectors Total sectors in volume					Sectors / FAT			0x0000		File System Version	Root(first) Cluster Number				
30	FS Info sector			Backup boot sector		Reserved										
40	Phys Drive num	0x00		Extd boot sig		Volume Serial Number			Volume Label (deprecated) normally “NO NAME ”							
50	Vol Label			System ID “FAT32”								Boot code				
60	Boot code															
70																
80																
90																
…																
1D0																
1E0																
1F0															55	AA

https://www.writeblocked.org/resources/FAT_cheatsheet.pdf

How to locate the areas

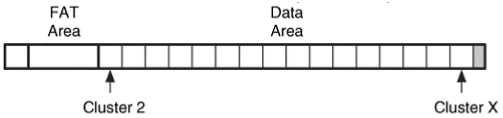
1. The reserved area starts at sector 0, its size is in the VBR (1 means only the VBR is reserved)
   - Usually 1 for FAT12/16; FAT32 uses more because of FSINFO
2. FAT area follows the reserved area, and its size is calculated by multiplying the number of tables by their size
3. then, there is the Data Area

Clusters are only in Data Area, numbered from 2 (!!!) and after the root directory for FAT12/16



Final sectors

The size of the data area may not be a multiple of the cluster size, so there could be unused sectors at the end:



Data could be also hidden after the last valid entry in a FAT table

Outline

Directory entry

A directory entry is 32 bytes and stores

- file name (8.3) — first byte `0xe5` means deleted (`0x05` used to encode `0xe5`)
- attributes: RO 1, hidden 2, sys 4, vol. label 8, dir 16, archived 32
- size, a 32-bit integer (0 for directories)
- starting cluster
- time stamp information
  - created, last accessed, and last written with widely different granularity
  - created/access timestamps are optional

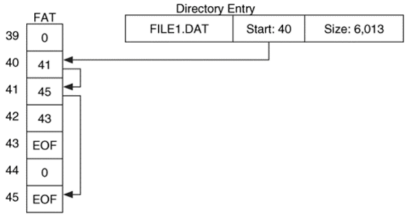
FAT Directory Entry

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	File name								Extension			attr ibute	reserved	10ms create time[1]	create time	
10	create date		last access date		unused		modified time		modified date		start cluster		File Size			

1. The 10millisecond create time is technically only used in FAT32.

`https://www.writeblocked.org/resources/FAT_cheatsheet.pdf`

Cluster chains



`fsstat` decodes this chains (in *sectors*); special values:

- `0` → not allocated
- `0xf...ff0-0xf...f6` → reserved
- `0xf...ff7` → damaged
- `0xf...ff8-0xf...fff` → EOF

FAT entries start at 0, but. . .

The first addressable cluster #2. Entry 0 typically stores a copy of the media type, and entry 1 stores the dirty-status of the file system

Exercise

Demo/Exercise

In `eighties.dd` (SHA256: `cc121c3a...`) and `eighties-all-files.dd` (SHA256: `e5f16884...`) you'll find two very similar FAT16 (not VFAT) file systems. In the former all files have been deleted.

Using ImHex,

1. find out:
   - Sector and cluster sizes
   - Number of reserved sectors
   - Locations of: FAT1, FAT2, Root Dir. (=Data Area), first cluster (#2)

   compare these results with the output of `fsstat`
2. check the FAT entries for `48.gif` in the two dd-images, and compare the results of `istat` on "inode" 5

VFAT/Long File Name support

When a file name does not follow MSDOS 8.3 ASCII convention

- additional entries (with "attributes" `0xf`) can store the long name
- those precede the main entry, which stores all other metadata

Long File Name

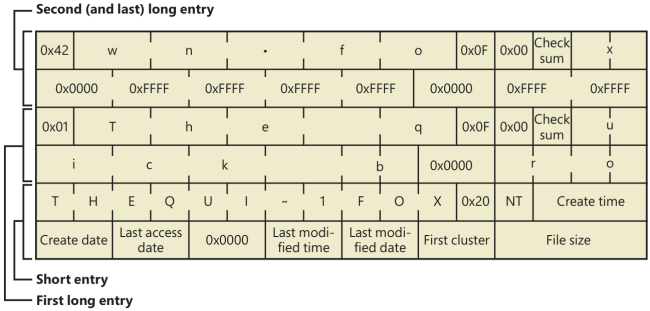
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	file name (Unicode 2 bytes/char)											0x0F	reserved	Check sum	file name	
10	file name											0x0000	file name			

`https://www.writeblocked.org/resources/FAT_cheatsheet.pdf`

Byte 0 is actually a sequence number (the final one is ORed with `0x40`) or `0xe5` if unallocated; for example...

LFN example

"The quick brown.fox", as `THEQUI~1.FOX` in 8.3 convention.



From [ARIS21]

Standard directory entries

When a new directory is created, it contains

- . and ..

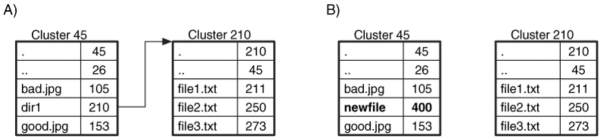
Cluster 110

Name	Created	Cluster
dir2	3/30/04 01:29:01	128
dir1	4/03/04 11:47:40	196
file8.dat	3/30/04 20:41:12	112

Cluster 196

Name	Created	Cluster
.	4/1/04 09:27:00	196
..	4/1/04 09:27:00	110
file1.dat	4/3/04 12:58:23	297

- those entries can be helpful for carving deleted directories



Since the size of a directory is always 0, the only way to know how many cluster to read is following the cluster chain

Exercise

Demo/Exercise

In `eighties-vfat.dd` (SHA256: `62258f92ebb42226...`) and `eighties-vfat-all-files.dd` (SHA256: `fe46141b98d227cb...`) you'll find two very similar, and familiar, VFAT FAT16 file systems.

As with the previous exercise, in the former all files have been deleted.
Yet, `fls -rp eighties-vfat.dd` can show the full, long name, for *some deleted files* but not for others, that are listed under `$OrphanFiles`.

1. Can you explain why?
   Hint: `eighties-vfat-all-files.dd` contains some clues
2. Using ImHex, can you manually recover the full names from `eighties-vfat.dd`?

References

[ARIS21] Andrea Allievi, Mark Russinovich, Alex Ionescu, and David Solomon.
Windows Internals, Part 2, 7th Edition.
Microsoft Press, 2021.

[Car05] Brian Carrier.
File System Forensic Analysis.
Addison-Wesley Professional, 2005.