Chapter 9. FAT Concepts and Analysis

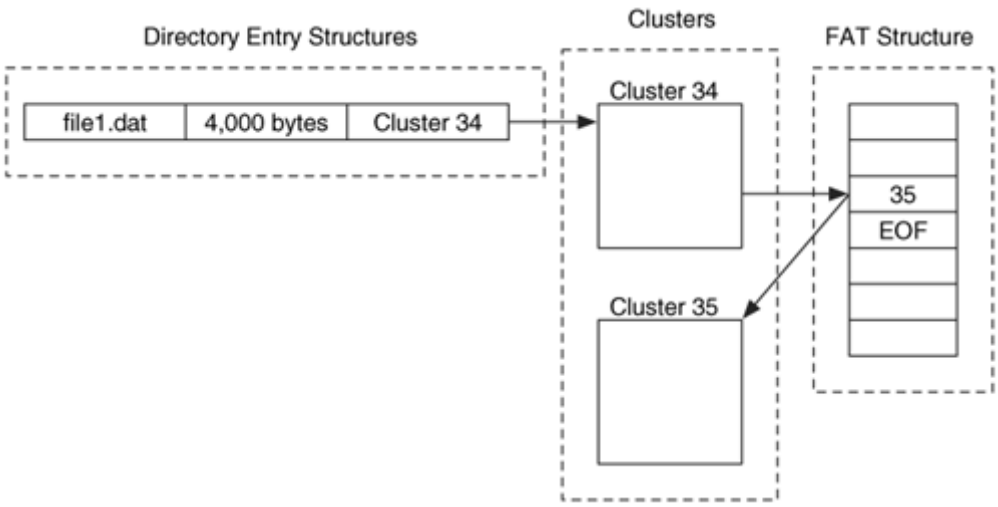
The File Allocation Table (FAT) file system is one of the most simple file systems found in common operating systems. FAT is the primary file system of the Microsoft DOS and Windows 9x operating systems, but the NT, 2000, and XP line has defaulted to the New Technologies File System (NTFS), which is discussed later in the book. FAT is supported by all Windows and most Unix operating systems and will be encountered by investigators for years to come, even if it is not the default file system of desktop Windows systems. FAT is frequently found in compact flash cards for digital cameras and USB "thumb drives." Many people are familiar with the basic concepts of the FAT file system but may not be aware of data hiding locations, addressing issues, and its more subtle behaviors. The goal of this chapter is to provide the general concepts and analysis techniques associated with FAT by using the five-category model. Chapter 10, "FAT Data Structures," discusses the low-level data structures. You can choose to read the two chapters in parallel, read them in sequential order, or skip the data structures chapter all together.

Introduction

One of the reasons the FAT file system is considered simple is because it has a small number of data structure types. Unfortunately, it also means that there have been some modifications over the years to give it new features (although they are not quite as confusing as those done to DOS partitions are). The FAT file system does not clearly follow the five-category model that was previously described; therefore, the following sections may seem awkward (in fact, trying to explain the file system in this way makes it more complex than it needs to be). There are two important data structures in FAT (the File Allocation Table and directory entries) that serve multiple purposes and belong to multiple categories of the model. For these data structures, part of the data structure will be explained in one section, and the rest of it will be explained in another section. They are described in more detail in the next chapter. It is important to describe the FAT file system using the categories so that it is easier to compare it with more advanced file systems that more clearly follow the model. The FAT file system does not contain any data that falls into the application category.

The basic concept of a FAT file system is that each file and directory is allocated a data structure, called a directory entry, that contains the file name, size, starting address of the file content, and other metadata. File and directory content is stored in data units called clusters. If a file or directory has allocated more than one cluster, the other clusters are found by using a structure that is called the FAT. The FAT structure is used to identify the next cluster in a file, and it is also used to identify the allocation status of clusters. Therefore it is used in both the content and metadata categories. There are three different versions of FAT: FAT12, FAT16, and FAT32. The major difference among them is the size of the entries in the FAT structure. The relationships between these data structures will be examined in more detail, but we can see this relationship in Figure 9.1.

Figure 9.1. Relationship between the directory entry structures, clusters, and FAT structure.



The layout of the FAT file system has three physical sections to it, which can be seen in Figure 9.2. The first section is the reserved area, and it includes data in the file system category. In FAT12 and FAT16 this area is typically only 1 sector in size, but the size is defined in the boot sector. The second section is the FAT area, and it contains the primary and backup FAT structures. It starts in the sector following the reserved area, and its size is calculated based on the number and size of FAT structures. The third section is the data area, and it contains the clusters that will be allocated to store file and directory content.

Figure 9.2. Physical layout of a FAT file system.

File System Category

The data in the file system category describe the general file system, and they are used to find the other important data structures. This section describes the general concepts about where FAT stores the data in this category and how we can analyze the data.

General Concepts

In a FAT file system, the file system category of data can be found in the boot sector data structure. The boot sector is located in the first sector of the volume, and it is part of the reserved area of the file system. Microsoft refers to some of the data in the first sector as belonging to the BIOS Parameter Block (BPB), but for simplicity I'll use the term boot sector.

The boot sector contains data that belong to all categories in the model, so I will wait until we get to each of those categories to describe those values. There is no field that identifies the file system as FAT12, FAT16, or FAT32. The type can be determined only by performing calculations from the boot sector data. I will show that calculation at the end of the chapter because it relies on concepts that have not yet been discussed.

A FAT32 file system boot sector contains additional data, including the sector address of a backup copy of the boot sector and a major and minor version number. The backup copy of the boot sector can be used if the version in sector 0 becomes corrupt, and the Microsoft documentation says that it should always be in sector 6 so that tools can automatically find it if the default copy goes bad. The data structure for the boot sector is discussed in Chapter 10.

FAT32 file systems also have an FSINFO data structure that contains information about the location of the next available cluster and the total amount of free clusters. The data are not guaranteed to be accurate, and are there only as a guide for the operating system. Its data structure is described in the next chapter.

Essential Boot Sector Data

One of the first things we need to know when analyzing a FAT file system is the location of the three physical layout areas. The reserved area starts in sector 0 of the file system, and its size is given in the boot sector. For FAT12/16 the reserved area is typically only 1 sector, but FAT32 will typically reserve many sectors.

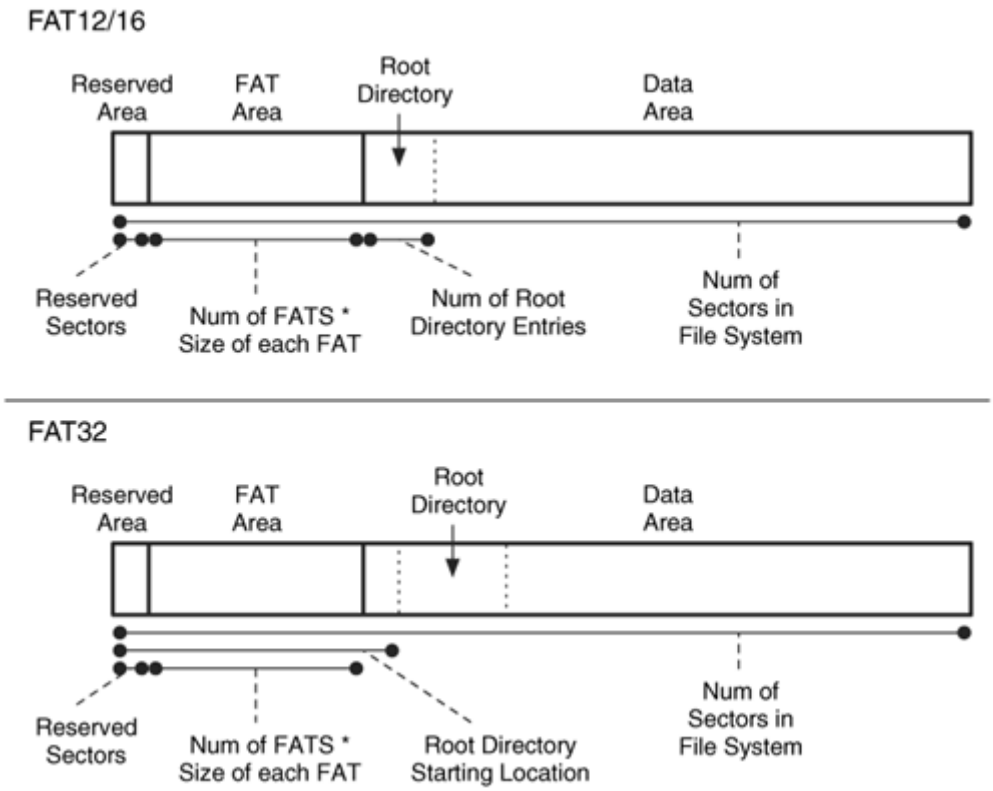
The FAT area contains one or more FAT structures, and it begins in the sector after the reserved area. Its size is calculated by multiplying the number of FAT structures by the size of each FAT; both of these values are given in the boot sector.

The data area contains the clusters that will store file and directory contents and begins in the sector after the FAT area. Its size is calculated by subtracting the starting sector address of the data area from the total number of sectors in the file system, which is specified in the boot sector. The data area is organized into clusters, and the number of sectors per cluster is given in the boot sector.

The layout of the data area is slightly different in FAT12/16 and FAT32. In FAT12/16 the beginning of the data area is reserved for the root directory, but in FAT32 the root directory can be anywhere in the data area (although it is rare for it to not be in the

beginning of the data area). The dynamic size and location of the root directory allows FAT32 to adapt to bad sectors in the beginning of the data area and allows the directory to grow as large as it needs to. The FAT12/16 root directory has a fixed size that is given in the boot sector. The starting address for the FAT32 root directory is given in the boot sector, and the FAT structure is used to determine its size. Figure 9.3 shows how the various boot sector values are used to determine the layout of FAT12/16 and FAT32 file systems.

Figure 9.3. FAT file system layout and data from the boot sector that is used to calculate the locations and sizes.



Non-essential Boot Sector Data

In addition to the layout information, the boot sector contains many non-essential values. The non-essential values are those that are not needed for the file system to save and retrieve files, and they are there for convenience and may not be correct. One such value is an eight-character string called the OEM name that may correspond to what tool was used to make the file system, but it is an optional value. For example, a Windows 95 system sets it to "MSWIN4.0," a Windows 98 system sets it to `MSWIN4.1`, and a Windows XP or 2000 system sets it to "MSDOS5.0." I found that the Linux `mkfs.msdos` sets it to `mkdosfs`, some USB tokens have random values, and some compact flash cards in digital cameras have names that resemble the camera model. Anyone with a hex editor can

change this value, but it may help you to determine what type of computer formatted a floppy. Some versions of Windows require that this value be set.

FAT file systems have a 4-byte volume serial number that is, according to the Microsoft specification, determined at file system creation time by using the current time, although the operating system that creates the file system can choose any value. My testing has shown different behavior with different versions of Windows. I tested a Windows 98 system, and it had the behavior reported by Craig Wilson [Wilson 2003], where the serial number is the result of adding the date and time fields in a specific order. This calculation is described in the "Boot Sector" section of Chapter 10. Windows XP did not create volume serial numbers using the same algorithm. Windows uses this value with removable media to determine when a disk has been changed.

There is also an eight-character type string that contains "FAT12," "FAT16," "FAT32," or "FAT." Most tools that create a FAT file system will set this string properly, but it does not need to be correct. The only way to determine the actual type is by calculating some numbers, which we will do later. The last identifying label is an eleven-character volume label string that the user can specify when he creates the file system. The volume label is also saved in the root directory of the file system, and I have observed that when a label is added in XP, the label is written to only the root directory and not the boot sector.

Boot Code

The boot code for a FAT file system is intertwined with file system data structures [Microsoft 2003e]. This is unlike the Unix file system, which has completely separate boot code. The first three bytes of the boot sector contain a jump instruction in machine code that causes the CPU to jump past the configuration data to the rest of the boot code. As you can see from the data structure in the next chapter, the boot sector is 512 bytes, and bytes 62 to 509 in FAT12/16 and bytes 90 to 509 in FAT32 are not used. These bytes contain the boot code, and FAT32 can use the sectors following the boot sector for additional boot code.

It is common for FAT file systems to have boot code even though they are not bootable file systems. The boot code displays a message to show that another disk is needed to boot the system. The FAT boot code is called from the boot code in the MBR of the disk, and the FAT boot code locates and loads the appropriate OS files.

Example Image

I will be using data from an example FAT32 image throughout this section. The Sleuth Kit (TSK) has a tool called `fsstat` that displays much of the data in the file system category of data. Here is the output from running `fsstat` on our example image:

```
# fsstat –f fat fat-4.dd
FILE SYSTEM INFORMATION
```

```
-----------------------------------------
File System Type: FAT
OEM Name: MSDOS5.0
Volume ID: 0x4c194603
Volume Label (Boot Sector): NO NAME
Volume Label (Root Directory): FAT DISK
File System Type Label: FAT32

Backup Boot Sector Location: 6
FS Info Sector Location: 1
Next Free Sector (FS Info): 1778
Free Sector Count (FS Info): 203836
Sectors before file system: 100800

File System Layout (in sectors)
Total Range: 0 - 205631
* Reserved: 0 – 37
** Boot Sector: 0
** FS Info Sector: 1
** Backup Boot Sector: 6
* FAT 0: 38 - 834
* FAT 1: 835 - 1631
* Data Area: 1632 - 205631
** Cluster Area: 1632 - 205631
*** Root Directory: 1632 - 1635

CONTENT-DATA INFORMATION
-----------------------------------------
Sector Size: 512
Cluster Size: 1024
Total Cluster Range: 2 – 102001
[REMOVED]
```

We can see from the previous output that there are 38 reserved sectors until the first FAT. In the reserved area are a backup boot sector and a FSINFO data structure. There are two FAT structures, and they span from sectors 38 to 834 and 835 to 1,631. The data area starts in sector 1,632, and it has clusters that are 1,024 bytes in size.

Analysis Techniques

The purpose of analyzing the file system category of data is to determine the file system layout and configuration details so that more specific analysis techniques can be conducted. In the process we also may find evidence that is specific to the case. For example, we may find which OS formatted the disk or hidden data.

To determine the configuration of a FAT file system, we need to locate and process the boot sector, whose data structure is given in Chapter 10. Processing the boot sector is simple because it is located in the first sector of the file system and has basic fields. There are two versions of the boot sector, but both are clearly documented. Using the information from the boot sector, we can calculate the locations of the reserved area, the FAT area, and the data area.

The FAT32 FSINFO data structure might also provide some clues about recent activity, and its location is given in the boot sector. It is typically located in sector 1 of the file system. Backup copies of both data structures also exist in FAT32.

Analysis Considerations

As we have seen, the data in this category provides structural data about the file system, and there is little data that the user has control over. Therefore, you will not likely find many smoking guns here. There are no values that show when or where the file system was created, but the OEM label and volume label, which are nonessential, may give some clues because different tools have different default values. We will later see that there is a special file that has a name equal to the volume label, and it might contain the file system creation time.

There are several places that are not used by the file system, and they could contain data that has been hidden by the user. For example, there are over 450 bytes of data between the end of the boot sector data and the final signature. Windows generally uses this space to store boot code for the system, but it is not needed for non-bootable file systems.

FAT32 file systems typically allocate many sectors to the reserved area, but only a few are used for the primary boot sector, the backup boot sector, and the FSINFO data structure. Therefore, these could contain hidden data. In addition, the FAT32 FSINFO data structure has hundreds of unused bytes. The OS generally wipes the sectors in the reserved area when it creates the file system.

There also could be hidden data between the end of the file system and the end of the volume. Compare the number of sectors in the file system, which is given in the boot sector, with the number of sectors that are in the volume to find volume slack. Note that it is relatively easy for someone to create volume slack because they need to modify only the total number of sectors value in the boot sector.

FAT32 file systems have a backup boot sector, and it should be located in sector 6. The primary and backup copies could be compared to identify inconsistencies. If the primary copy is corrupt, the backup should be examined. If the user modified any of the labels or other values in the primary boot sector by using a hex editor, the backup copy might contain the original data.

Analysis Scenario

During a raid on a suspect's house, a hard disk is found in a drawer. During the acquisition process, we realize that the first 32 sectors of the disk are damaged and cannot be read. The suspect probably put the drive in the drawer after it failed and used a new drive, but we want to examine it for evidence. The suspect's computer was running Windows ME and, therefore, using a FAT file system. This scenario shows how we can find the file systems even though the partition table does not exist.

To find the start of a FAT file system, we will search for the signature values of 0x55 and 0xAA in the final two bytes of the boot sector. We should expect a considerable number of false hits by doing only this search. If a disk contained random data, we would expect, on average, to find this signature every 65,536 (i.e., 2^{16}) sectors. We can reduce the number of false hits by using a larger signature or by using other data. This scenario shows how the latter method works well with FAT32 because there is a pattern of these signatures in the reserved area of the file system. Of course, automated tools can do this for us more quickly, but we are going to do it by hand.

We will use the `sigfind` tool from TSK to look for the signature. Any tool that searches for hexadecimal values should work. The `sigfind` tool prints the sector in which the signature was found and gives the distance since the previous hit. Here is the output, along with commentary:

```
# sigfind –o 510 55AA disk-9.dd
Block size: 512 Offset: 510
Block: 63 (-)
Block: 64 (+1)
Block: 65 (+1)
Block: 69 (+4)
Block: 70 (+1)
Block: 71 (+1)
Block: 75 (+4)
Block: 128504 (+128429)
Block: 293258 (+164754)
[REMOVED]
```

The first hit for the signature is in sector 63, which makes sense because the first partition typically starts in sector 63. We read the sector and apply the boot sector data structure. We learn that it has a backup boot sector in sector 6 and FSINFO in sector 1 of the file system. We also learn that there are 20,482,812 sectors in the file system. The FSINFO data structure has the same signature as the boot sector, so sector 64 is also a hit.

Similarly, sectors 69 and 70 are hits because they are the backup copies of the boot sector and FSINFO, which are located six sectors from the original. Blocks 65 and 71 are all zeros except for their signatures. The hit in block 128,504 is a false hit and is random data when we view it. Therefore, based on the location of the boot sector and the relative location of the backup copies, we can assume that there is a FAT file system from disk sector 63 to 20,482,874. We will now view more of the `sigfind` output:

```
[REMOVED]
Block: 20112453 (+27031)
Block: 20482875 (+370422)
Block: 20482938 (+63)
Block: 20482939 (+1)
Block: 20482940 (+1)
Block: 20482944 (+4)
Block: 20482945 (+1)
Block: 20482946 (+1)
```

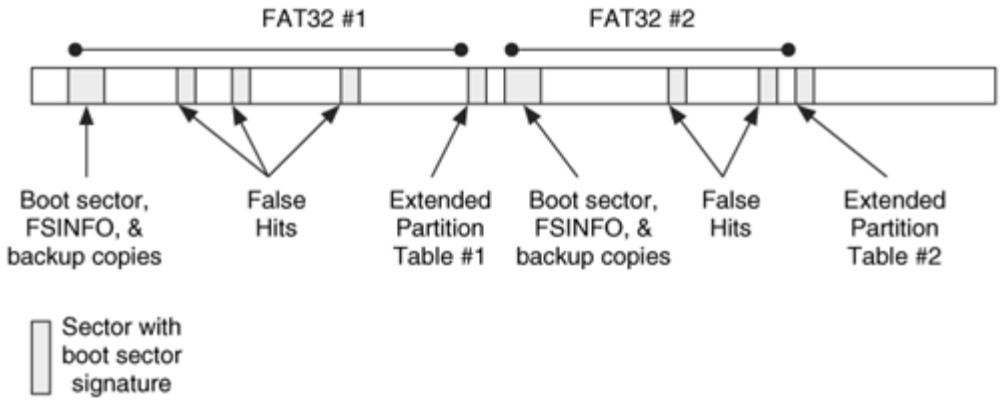
```
Block: 20482950 (+4)
Block: 20513168 (+30218)
```

In the output that I removed, there were many false hits, and the output shown has sector 20,482,875 with a hit. That sector follows the end of the previous file system, which ended in 20,482,874. The sequence of hits following 20,482,875 is different from what we previously saw, though, because the next hit is 63 sectors away, and then there are several that are close together. We view sector 20,482,875 to see if it is a false hit:

```
# dd if=disk-9.dd bs=512 skip=20482875 count=1 | xxd
0000000: 088c 039a 5f78 7694 8f45 bf49 e396 00c0 ...._xv..E.I....
0000016: 889d ddc0 6d36 60df 485d adf7 46d1 3224 ....m6`.H]..F.2$
0000032: 3829 95cd ad28 d2a2 dc89 f357 d921 cfde 8)...(.....W.!..
0000048: df8e 1fd3 303e 8619 641e 9c2f 95b4 d836 ....0>..d../...6
[REMOVED]
0000416: 3607 e7be 1177 db5f 11c9 fba1 c913 1a3d 6....w._.......=
0000432: da81 143d 00c7 7083 9d42 330c 0287 0001 ...=..p..B3.....
0000448: c1ff 0bfe ffff 3f00 0000 fc8a 3801 0000 ......?.....8...
0000464: c1ff 05fe ffff 3b8b 3801 7616 7102 0000 ......;.8.v.q...
0000480: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000496: 0000 0000 0000 0000 0000 0000 0000 55aa ..............U.
```

It could be easy to pass this off as a false hit, but notice the last four lines in the output and think back to Chapter 5, "PC-based Partitions," when we discussed DOS partitions. This sector contains an extended partition table, and the partition table starts at byte 446. DOS partition tables use the same signature value as FAT boot sectors. If we were to process the two non-zero entries in the table, we would learn that there is a FAT32 partition from sector 20,482,938 to 40,965,749 and an extended partition from sector 40,965,750 to 81,931,499. This confirms our `sigfind` output because we had a hit in sector 20,482,938 and hits 1, 6, and 7 sectors after that for the FSINFO data structure and backup copies. A graphical representation of this example can be found in Figure 9.4. It shows the two file systems we found and the locations of the various false hits and extended partition tables.

Figure 9.4. Results from searching for the FAT boot sector signature on a disk with no partition table.



In this example, we have shown how we can find a FAT32 file system if the boot sector exists. A search for only the 2-byte signature generates many false hits, but FAT32 makes it a little easier because we expect to find hits 1, 6, and 7 sectors away from the FSINFO data structure and backup copies. FAT12/16 is more difficult because there are no backup structures, but all we need to do is find the first hit. We can start by looking in sector 63. After we find a file system, we can use the file system length to skip ahead and start searching from there. We also can use any DOS extended partition table structures to help find file systems.

Content Category

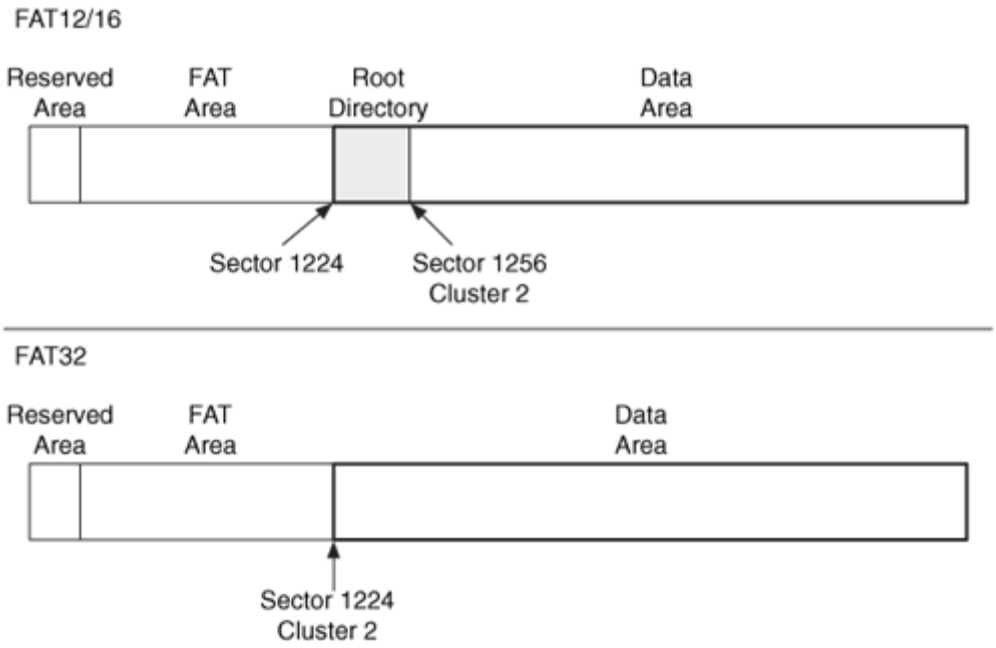
The content category includes the data that comprise file or directory content. A FAT file system uses the term cluster for its data units. A cluster is a group of consecutive sectors, and the number of sectors must be a power of 2, such as 1, 2, 4, 8, 16, 32, or 64. According to the Microsoft specification, the maximum cluster size is 32KB. Each cluster is given an address, and the address of the first cluster is 2. In other words, there are no clusters that have an address of 0 or 1. The clusters are all located in the data area region of the file system, which is the last of the three areas.

Finding the First Cluster

Finding the location of the first cluster, which is cluster 2, is harder than it sounds because it is not at the beginning of the file system; it is in the data area. The reserved and FAT areas, which occur before the data area, do not use cluster addresses. The FAT file system is an example where not every logical volume address has a logical file system address. As we will see in Chapter 11, this was changed in NTFS where the first cluster is also the first sector of the file system.

The procedure for finding the sector address of cluster 2 is different for FAT12/16 and FAT32. Cluster 2 in a FAT32 file system starts with the first sector of the data area. For example, consider a file system with 2,048-byte clusters and a data area that starts in sector 1,224. The sector address of cluster 2 will be sector 1,224, and the sector address of cluster 3 will be 1,228. We can see this in the bottom of Figure 9.5.

Figure 9.5. In a FAT12/16 file system, cluster 2 follows the root directory, and in a FAT32 file system, cluster 2 is the first sector of the data area.



With a FAT12 and FAT16 file system, the first sectors of the data area are for the root directory, which is allocated when the file system is created and has a fixed size. The number of root directory entries is given in the boot sector, and cluster 2 starts in the next sector. For example, consider a FAT16 file system with 32 sectors allocated for the root directory. If the data area starts in sector 1,224, the root directory spans from sectors 1,224 to 1,255. If we had 2048-byte clusters, cluster 2 would start in sector 1,256 and cluster 3 would start in sector 1,260. We can see this in the top of Figure 9.5.

Cluster and Sector Addresses

As we just discussed, the cluster addresses do not start until the data area. Therefore, to address the data in the reserved and FAT areas, we have to either use two different addressing schemes or use the lowest common denominator, which is the sector address (the logical volume address). Using the sector address for everything is relatively simple and is what all tools and operating systems internally use because they need to know where the data is located relative to the start of the volume. Some tools, including TSK,

show all addresses to the user in sector addresses so that only one addressing scheme is needed.

To convert between cluster and sector addresses, we need to know the sector address of cluster 2, and we need to know how many sectors there are per cluster. The basic algorithm for calculating the sector address of cluster C is

```
(C – 2) * (# of sectors per cluster) + (sector of cluster 2)
```

To reverse the process and translate a sector S to a cluster, the following is used:

```
( (S – sector of cluster 2) / (# of sectors per cluster) ) + 2
```

Cluster Allocation Status

Now that we know where to find the clusters, we need to determine which ones are allocated. The allocation status of a cluster is determined using the FAT structure. There are typically two copies of the FAT, and the first one starts after the reserved area of the file system. The FAT is discussed in more detail in Chapter 10, but I will provide the needed details here.

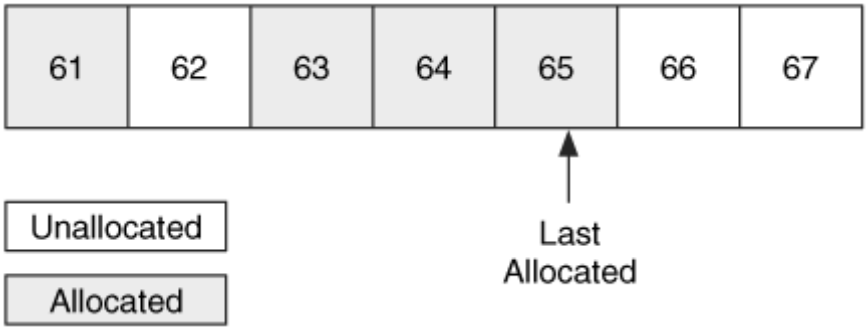
The FAT is used for many purposes, but the basic concept is that it has one entry for every cluster in the file system. For example, table entry 481 corresponds to cluster 481. Each table entry is a single number whose maximum value depends on the FAT version. FAT12 file systems have a 12-bit table entry, FAT16 file systems have a 16-bit table entry, and FAT32 file systems have a 32-bit table entry (although only 28 of the bits are used).

If the table entry is 0, the cluster is not allocated to a file. If the table entry is 0xff7 for FAT12, 0xfff7 for FAT16, or 0x0fff fff7 for FAT32, the cluster has been marked as damaged and should not be allocated. All other values mean that the cluster is allocated, and the meaning of the value will be discussed later in the "Metadata Category" section.

Allocation Algorithms

The OS gets to choose which allocation algorithm it uses when it allocates the clusters. I tested Windows 98 and Windows XP systems, and it appeared that a next available algorithm was being used in both. The next available algorithm searches for the first available cluster starting from the previously allocated cluster. For example, if cluster 65 is allocated to a new file and then cluster 62 is unallocated, the next search will start at cluster 66 and will not immediately reallocate cluster 62. We can see this in Figure 9.6. There are many factors that could affect the allocation of clusters, and it is difficult to identify the exact algorithms used.

Figure 9.6. The search for an unallocated cluster starts from the last allocated cluster, not the start of the file system.



To find an unallocated cluster that it can allocate, the OS scans the FAT for an entry that has a 0 in it. Recall that a FAT32 file system has the FSINFO data structure in the reserved area that identifies the next free cluster so that can be used as a guide by the operating system. To change a cluster to unallocated status, the corresponding entries in the FAT structures are located and set to 0. Most operating systems do not clear the cluster contents when it is unallocated unless they implement a secure wiping feature.

Analysis Techniques

Analysis of the content category is performed to locate a specific data unit, determine its allocation status, and do something with the content. Locating a specific data unit in FAT is more complex than with other file systems because cluster addresses do not start at the beginning of the file system. When we are trying to locate a data unit prior to the start of the data area, we need to use sector addresses. For the data units in the data area, we can use either sector or cluster addresses.

The allocation status of each cluster can be determined by looking at the cluster's entry in the FAT. Entries with a zero value are unallocated and non-zero entries are allocated. If we wanted to extract the contents of all unallocated clusters, we would read the FAT and extract each cluster with a zero in the table. The data units prior to the data area are not listed in the FAT; therefore, they do not have an official allocation state, although most are used by the file system. Test your tools to determine whether they consider any data prior to the data area to be unallocated.

Analysis Considerations

With a FAT file system, the clusters marked as bad should be examined because many disks handle bad sectors at the hardware level, and the operating system does not see them. Bad data units should be examined with any type of file system, but Microsoft notes that some copyright applications store data in FAT clusters that are marked as bad, so the ScanDisk tool in Windows will not verify that sectors marked as bad are indeed

bad [Microsoft 2004b]. Some versions of the `format` command in Windows will preserve the bad status of a cluster when they reformat a file system [Microsoft 2003c].

The size of the data area may not be a multiple of the cluster size, so there could be a few sectors at the end of the data area that are not part of a cluster. These could be used to hide data or could contain data from a previous file system. Figure 9.7 shows an example of this where there is an odd number of sectors in the data area and each cluster includes 2 sectors. The final sector is gray and does not have a cluster address.

Figure 9.7. The final sectors in the Data Area may not fit into a full cluster and, therefore, could contain hidden data or data from a previous file system.

To determine if there are unused sectors, subtract the sector address of cluster 2 from the total number of sectors and divide by the number of sectors in a cluster. If there is a remainder from the division, there are unused sectors:

```
(Total number of sectors – Sector address of cluster 2) / (Number of
sectors per cluster)
```

Data also could be hidden between the end of the last valid entry in the primary FAT structure and the start of the backup copy and between the end of the last entry in the backup FAT and the start of the data area. To calculate how much unused space there is, we need to compare the size of each FAT, which is given in the boot sector, with the size needed for the number of clusters in the file system. For example, in the FAT32 file system that was previously analyzed using `fsstat`, we saw that it had 797 sectors allocated to each FAT. Each table entry in the FAT32 file system is four bytes and, therefore, 128 entries exist in each 512-byte sector. Each table has room for

```
797 sectors * 128 (entries / sector) = 102,016 entries
```

The `fsstat` output also shows that there were 102,002 clusters, so there are 14 unused table entries for a total size of 64 bytes.

Not every sector in the volume is assigned a cluster address, so the results from doing a logical volume search versus a logical file system search may be different. Test your

tools to see if they search the boot sector and FAT areas. An easy way to test this is to search for the phrase "FAT" and see if it finds the boot sector (first verify that your boot sector has the string in it, though).

Analysis Scenario

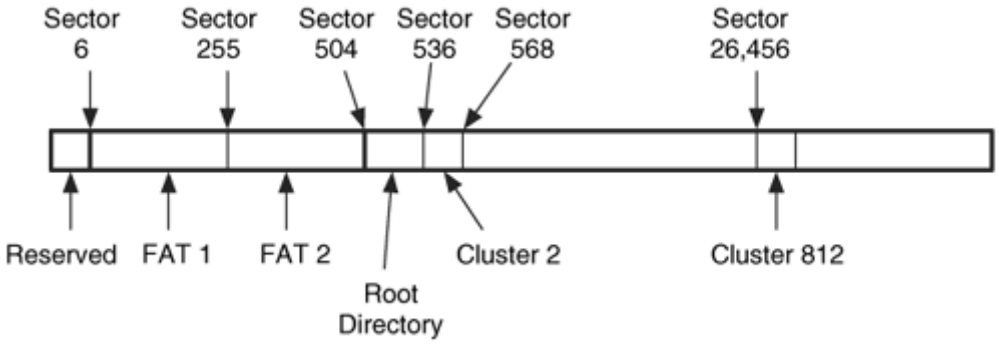
We have a FAT16 file system and need to locate the first sector of cluster 812. All we have is a hex editor that does not know about the FAT file system.

Our first step is to view the boot sector, which is located in sector 0 of the file system. We process it and learn that that there are six reserved sectors, two FATs, and each FAT is 249 sectors. Each cluster is 32 sectors, and there are 512 directory entries in the root directory.

Now we need to do some math. The first FAT starts in sector 6 and ends in sector 254. The second FAT starts in sector 255 and ends in 503. Next is the root directory. There are 512 entries in the root directory and (as we will later see) each entry is 32 bytes, so the directory needs 16,384 bytes, which comprise 32 sectors. Therefore, the root directory will be in sectors 504 to 535, and the data area will begin in sector 536.

The first cluster in the data area has an address of 2. We are looking for cluster 812, which is the 810th cluster in the data area, and each cluster is 32 sectors. Therefore, cluster 812 is 25,920 sectors from the start of the data area. Finally, we add the data area starting address and determine that cluster 812 starts in sector 26,456 and extends to sector 26,487. We see this layout in Figure 9.8.

Figure 9.8. Layout of our scenario example where we are looking for cluster 812.



Metadata Category

The metadata category includes the data that describe a file or directory, including the locations where the content is stored, dates and times, and permissions. We will use this category of data to obtain additional information about a file or to identify suspect files. In a FAT file system, this information is stored in a directory entry structure. The FAT structure also is used to store metadata information about the layout of a file or directory.

Directory Entries

The directory entry is a data structure that is allocated for every file and directory. It is 32 bytes in size and contains the file's attributes, size, starting cluster, and dates and times. The directory entry plays a role in both the metadata and file name category because the name of the file is located in this structure. Directory entries can exist anywhere in the data area because they are stored in the clusters allocated to a directory. In the FAT file system, a directory is considered a special type of file. The directory entry data structure is described in detail in Chapter 10.

When a new file or directory is created, a directory entry in the parent directory is allocated for it. Because each directory entry is a fixed size, we can imagine the contents of a directory to be a table of directory entries. Directory entries are not given unique numerical addresses, like clusters are. Instead, the only standard way to address a directory entry is to use the full name of the file or directory that allocated it. We will discuss this more in the "Directory Entry Addresses" section.

The directory entry structure has an attributes field, and there are seven file attributes that can be set for each file. However, the OS (or your analysis tool) might ignore some of them. I will cover the essential attributes first—these attributes cannot be ignored because they affect the way that the directory entry must be processed. The directory attribute is used to identify the directory entries that are for directories. The clusters allocated to a directory entry should contain more directory entries. The long file name attribute identifies a special type of entry that has a different layout. It will be discussed in detail in the "File Name" section. The final essential attribute is for the volume label, and only one directory entry, by specification, should have this attribute set. I have observed that with Windows XP, the volume label a user specifies is saved in this location and not in the boot sector. If none of these attributes are set, the entry is for a normal file.

There are four non-essential attributes that can be set for each file or directory. The impact of these being set depends on how the OS wants to enforce them. The read only attribute should prevent a file from being written to, but I found that directories in Windows XP and 98 can have new files created in them when they are set to read only. The hidden attribute may cause files and directories to not be listed, but there is typically a setting in the OS that causes them to be shown. The system attribute should identify a file as a system file, and Windows typically sets the archive attribute when a file is

created or written to. The purpose of this attribute is that a backup utility can identify which files have changed since the last backup.

Each directory entry has three times in it: created, last accessed, and last written. One of the strange traits of FAT is that each of these times has a widely different granularity. The created timestamp is optional and accurate to a tenth of a second; the access timestamp is also optional and accurate to the day; and the written timestamp is required by the specification and accurate to two seconds. There are no specifications for which operations cause each time to be updated, so each OS that uses FAT has its own policy with respect to when it updates the times. The Windows 95+ and NT+ lines update all times, but DOS and Windows 3.1 update only the last modified times. The times are stored with respect to the local time zone, which means that you do not have to convert the time based on where the computer was located.

The allocation status of a directory entry is determined by using the first byte. With an allocated entry, the first byte stores the first character in the file name, but it is replaced with 0xe5 when the entry becomes unallocated. This is why FAT recovery tools require you to enter the first letter of the file name.

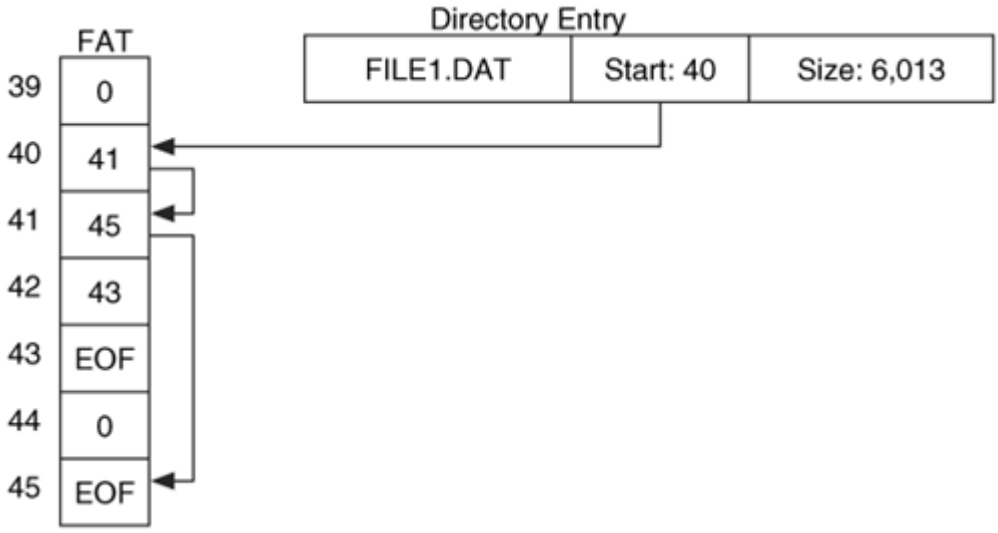
Cluster Chains

The directory entry contains the starting cluster of the file, and the FAT structure is used to find the remaining clusters in the file. Previously I noted that if a FAT entry is nonzero, its cluster is allocated. The non-zero entry contains the address of the next cluster in the file, an end of file marker, or a value to show that the cluster has bad sectors. To find the next cluster in a file, you simply look at the cluster's entry in the FAT and determine if it is the last cluster in the file or if there is another one. The FAT data structure layout and values are discussed in more detail in Chapter 10.

Using the FAT to find cluster addresses is analogous to a treasure hunt. You are given clues about the location of the first site. When you get to the first site, you find a treasure and directions to the second site. This process repeats until you find the final site.

For example, consider a file that is located in clusters 40, 41, and 45. If we wanted to read the contents of this file, we would first examine the starting cluster field in the directory entry, which should have a value of 40. The size of the file will show that more than one cluster is needed for the file, so the FAT entry for cluster 40 is referred to. The entry contains the value 41, which means that cluster 41 is the second cluster in the file. The file size shows that one more cluster is needed, and we examine the FAT entry for cluster 41 to find the value of 45. The FAT entry for cluster 45 contains the End of File (EOF) marker because it is the final cluster in the file. The sequence of clusters is sometimes called a cluster chain, and we can see an example cluster chain in Figure 9.9.

Figure 9.9. A FAT showing a cluster chain from cluster 40 to 41 to 45.



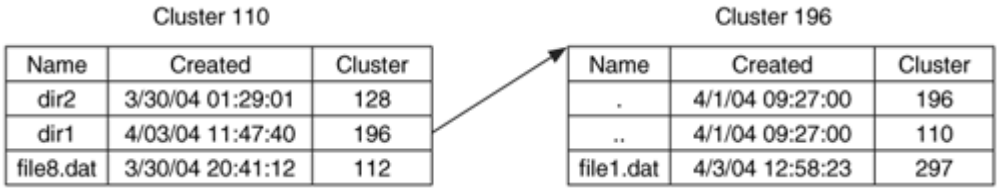
The maximum size of a FAT file system is based on the size of the FAT entries. Each entry has a maximum address that it can store for the next cluster in the chain. Therefore, there is a maximum cluster address. The FAT32 file system uses only 28-bits of the 32-bit entry, so it can describe only 268,435,456 clusters (actually, the maximum is a little less because there are some reserved values for EOF and bad clusters).

Directories

When a new directory is created, a cluster is allocated to it and wiped with 0s. The size field in the directory entry is not used, and should always be 0. The only way to determine the size of the directory is to use the starting cluster from the directory entry and follow the cluster chain in the FAT structure until the end of file marker is found.

The first two directory entries in a directory are for the . and .. directories. People who use the command line are used to these names because the . name is used to address the current directory, and .. is used to address the parent directory. These entries are actual directory entries with the directory attribute set, but Windows does not seem to update the values after they are created. The written, accessed, and created times seem to reflect the time that the directory was created. This behavior could be used to verify the creation date on a directory because it should be the same value as the . and .. entries. We cannot confirm the last written date of the directory, though, because the . and .. entries are not updated for every directory modification. We can see this situation in Figure 9.10 where the `dir1` created time is different from the . and .. entries. The user could have done this to hide some activity, or an application on the system may have modified it.

Figure 9.10. The created time in the directory entry for the directory does not match the '.' and '..' entries.



Directory Entry Addresses

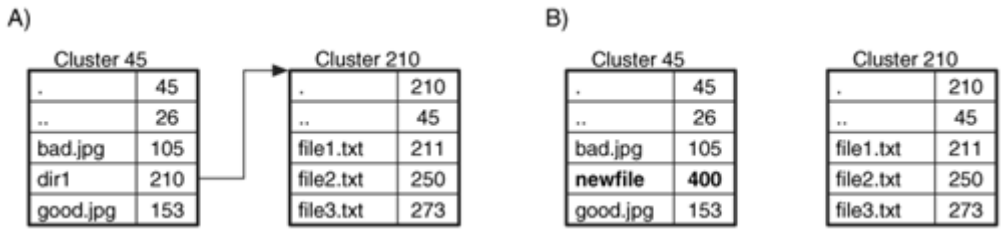
As I previously mentioned, the only standard way to address a directory entry is to use the full name of the file or directory that allocated it. There are at least two problems with this, and they are addressed in this section.

Consider a scenario where we want to find all allocated directory entry structures. To do this, we start in the root directory and recursively look in each of the allocated directories. For each directory, we examine each 32-byte structure and skip over the unallocated ones. The address of each structure is the name of the directory that we are currently looking at plus the name of the file.

This works great for normal file system usage, but investigators also want to find unallocated directory entries. This is where we find our first problem. The first letter of the name is deleted when the directory entry is unallocated. If a directory had two files `A-1.DAT` and `B-1.DAT` that were deleted, both entries would have the same name in them, `_-1.DAT`. Therefore, we have a unique naming conflict.

The second problem occurs when a directory is deleted and its entry is reallocated. In this case, there is no longer a pointer to the files and directories in the deleted directory, so they have no address. Consider Figure 9.11(A), where a directory entry for a directory points to cluster 210. That directory is deleted, and the directory entry is later reallocated, and it now points to cluster 400, as shown in Figure 9.11(B). The unallocated directory entries in cluster 210 still exist, but we cannot find them by simply following the directory tree, and if we do find them, we do not have an address for them. These are called orphan files.

Figure 9.11. A) shows an allocated directory and its contents. B) shows the state when the directory is deleted and the parent directory's directory entry is reallocated.

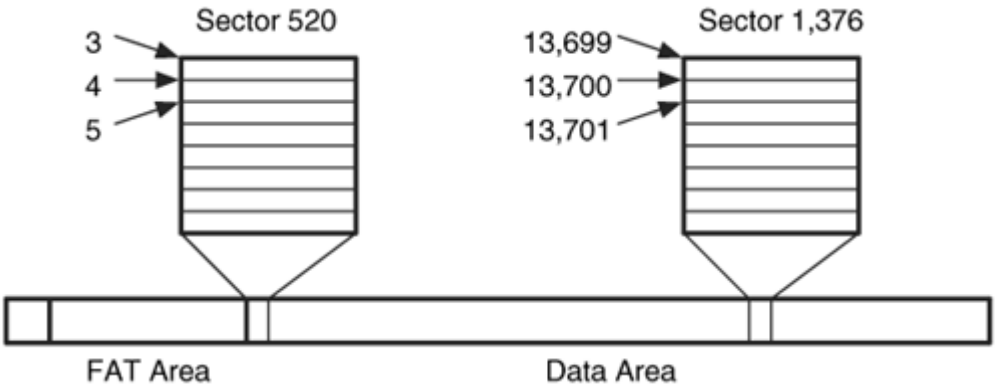


To find orphan files, we need to examine every sector of the data area. There is no standard method of doing this, but one method is to examine the first 32-bytes of each sector, not cluster, and compare it to the fields in a directory entry. If they are in a valid range, the rest of the sector should be processed. By searching each sector, you may find entries in the slack space of clusters that have been allocated to other files. A similar technique is to search the first 32 bytes of each cluster to find the . and .. entries that are the first two entries in every directory (we will show an example of this later). This technique will only find the first cluster of a directory and not any of its fragments.

One way to solve these problems is to use a different addressing method. For example, TSK uses a method that is also used by several Unix systems where it assumes that every cluster and sector could be allocated by a directory. Therefore, we can imagine that every sector is divided into 32-byte entries that could store a directory entry, and we assign a unique address to each entry. Therefore, the first 32-byte entry of the first sector in the data area would have an address of 0, and the second entry would have an address of 1.

This works, but there is a slight problem. Every directory and file will have a numerical address except for the root directory. Recall that the location and size of the root directory is given in the boot sector and not in a directory entry. The fix that TSK uses is to assign the root directory with an address of 2 (because that is what other Unix-based file systems use) and assign the first entry in the first sector an address of 3. We can see this in Figure 9.12 where sectors 520 and 1,376 are expanded to show the entry addresses inside them. Each 512-byte sector can store 16 directory entry structures, so sector 520 will have entries 3 to 18.

Figure 9.12. Addresses that are assigned to directory entries based on the sector and location in the sector.



Example Image

To show the data that exists for a typical FAT file, the output from `istat` is given here for a file in our example image. The directory entry for this file is analyzed in Chapter 10, but here is the formatted output:

```
# istat -f fat fat-4.dd 4
Directory Entry: 4
Allocated
File Attributes: File, Archive
Size: 8689
Name: RESUME-1.RTF

Directory Entry Times:
Written:        Wed Mar 24 06:26:20 2004
Accessed:       Thu Apr 8 00:00:00 2004
Created:        Tue Feb 10 15:49:40 2004

Sectors:
1646 1647 1648 1649 1650 1651 1652 1653
1654 1655 1656 1657 1658 1659 1660 1661
1662 1663
```

We see that this file has allocated 18 sectors, which is equivalent to nine clusters in this file system. The created, last written, and last accessed dates and times also are given, as are the attributes for the file. Because the file name is stored in the directory entry, we know the name of the file given only its address, but we do not know its full path.

Allocation Algorithms

In the metadata category, there are two general types of data that are allocated. The directory entries need to be allocated for new files and directories, and the temporal information for each file and directory needs to be updated. As with most allocation strategies, the actual behavior is OS-dependent and cannot be enforced by the file system data structures. A specific OS should be tested before conclusions are drawn based on the allocation algorithms. I will give my observations from Windows 98 and XP.

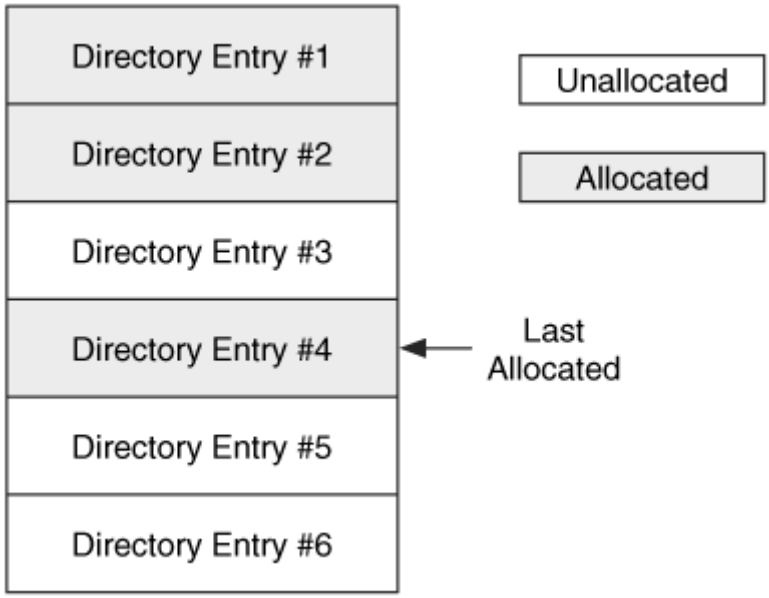
Directory Entry Allocation

Windows 98 uses a first-available allocation strategy and starts its search for an unallocated directory entry from the beginning of the directory. Windows XP uses a next-available allocation method and starts its search for an unallocated directory entry from the last allocated directory entry. Windows XP restarts its scan from the beginning of the directory when it gets to the end of the cluster chain. When Windows 98 or XP cannot find an unallocated directory entry, a new cluster is allocated.

The difference in the allocation methods can be seen in Figure 9.13, where entry 3 was unallocated after entry 4 was allocated. When the next entry is allocated, Windows 98

starts from the beginning of the cluster and allocates directory entry 3. Windows XP starts at entry 4 and allocates entry 5.

Figure 9.13. Directory entry 4 was just allocated. Windows 98 allocates entry 3 next, but Windows XP allocates entry 5 next.



When a file is renamed in Windows, a new entry is made for the new name, and the entry for the old name is unallocated. A similar behavior also can be seen when a new file or directory is created in Windows by using the right mouse button and choosing 'New.' This results in an entry for the default name, `New Text Document.txt`, for example, which is unallocated when the user enters a more unique name.

An interesting behavior of Windows XP is that when a file is created from a Windows application, two entries are created. The first entry has all the values except the size and the starting cluster. This entry becomes unallocated, and a second entry is created with the size and starting cluster. This occurs when saving from within an application, but not from the command line, drag and dropping, or using the 'New' menu option.

When a file is deleted, the first byte of the directory entry is set to 0xe5. Windows does not change any other values in the directory entry, but other OSes might. The clusters that were allocated for the file are unallocated by setting their corresponding entries in the FAT structure to 0. Fortunately, Windows keeps the starting cluster of the cluster chain in the directory entry so some file recovery can be performed until the clusters are allocated to new files and overwritten.

Time Value Updating

There are three time values in a directory entry: last accessed, last written, and created. There are few requirements in the FAT specification, but Microsoft Knowledge Base

article 299648 describes the behavior in Windows [Microsoft 2003d]. The time values are non-essential and could be false.

The created time is set when Windows allocates a new directory entry for a new file. The "new file" part is important because if the OS allocates a new directory entry for an existing file, even if original location was on a different disk, the original creation time is kept. For example, if a file is renamed or is moved to another directory or disk, the creation time from the original entry is written to the new entry. There is one known exception to this rule, and it is if the move is done from the command line of a 2000/XP system to a different volume. In this case, the created time is set to the time of the move. If a file is copied, a new file is being created, and a new creation time is written to the new entry. The creation time corresponds to the time when the first directory entry was allocated for the file, regardless of the original location (unless the file was moved to a new file system by using the command line).

The written time is set when Windows writes new file content. The write time is content-based and not directory entry-based and follows data as it is copied around. If files are moved or copied in Windows, the new directory entry has the written time from the original file. Changing the attributes or name of a file does not result in an update to this time. Windows updates this time when an application writes content to the file, even if the application is doing an automatic save and no content has changed.

In summary, if you move a file in Windows, the resulting file will have the original written and original creation time unless the move is to a different volume and the command line is used. If you copy a file, the resulting file will have the original written time and a new creation time. This can be confusing because the creation date is after the written time. If you create a new file from an application in Windows, it is common for the written time to be a little later than the creation time.

The last accessed date, which is only accurate to the day, is updated the most frequently. Anytime the file is opened, the date is updated. When you right-click on the file to see its properties, its access date is updated. If you move a file to a new volume, the access date on the new file is updated because Windows had to read the original content before it could write it to the new volume. If you move a file within the same volume, though, the access date will not change because the new file is using the same clusters. If you copy or move a file, the access date on both the source and destination is updated. The one exception to this simple rule is that under Windows XP, the access date is not updated when the file is copied or when the "copy" menu features are used. Alternatively, Windows 98 does update the access time of the source file when the destination file was created. Some versions of Windows can be configured to not update the last access date.

For directories, I observed that the dates were set when the directory was created and were not updated much after that. Even when new clusters were allocated for the directory or new files were created in the directory, the written times were not updated.

Analysis Techniques

The reason for analyzing the metadata category of data is to determine more details about a file or directory. To do this in FAT, we must locate a specific directory entry, process its contents, and determine where the file or directory content is stored. Locating all entries in FAT is difficult because only allocated directory entries have a full address, which is their path and file name. We will see an example of this in the "Scenarios" section. Unallocated entries might not have full addresses; therefore, a tool-specific addressing scheme will likely be used.

After a directory entry has been located, it is relatively straightforward to process it. To identify all clusters that a file has allocated, we use the starting cluster from the directory entry and the FAT structure to locate the other clusters.

Analysis Considerations

When analyzing the metadata from a FAT file system, there are some special considerations that should be taken into account. These considerations have to deal with the directory entry time values, allocation routines, and consistency checks.

One consideration, which can work to the benefit of the investigator, is that times are stored without respect to time zone, so it does not matter what time zone your analysis system is set to. This also makes dealing with daylight savings easier because you don't need to figure out whether you need to add or subtract an hour based on what month it is.

Another consideration is that the last accessed and the created dates and times are optional by specification and might be 0. Like the times on most file systems, it is easy to change the times on a file. In fact, there is a Microsoft document that tells developers to save the access time of a file before opening it so that it can be restored if the application was not able to read it [Microsoft 2003a]. Additional application-level data should be used to confirm temporal data. Further, the time updating is not even consistent within Microsoft platforms.

Windows sets the last written time to the last time that the data content itself was changed on the local system. Copying a file can result in a creation time that corresponds to the time that the copy occurred and a written date that corresponds to the last written date of the original location. This type of scenario is fairly common and not necessarily a sign of tampering. The resolution of the last access date is only accurate to the day, so it does not help much when trying to reconstruct the events on a system.

The allocation routines for new directory entries in an XP system allow you to see the names of deleted files for longer than would be expected because a first available algorithm is not used. In reality, this may not allow you to recover any more files than if a first available algorithm had been used because the clusters of the file will be reallocated just as quickly. Therefore, you might not recover the original file content even if you can see the name.

The DEFRAG utility that comes with Windows compacts directories and removes unused directory entries. It also moves the file content around to make it more contiguous. Therefore, recovery after running DEFRAG is difficult because the directory entries of the deleted files are gone and the clusters have been moved around.

The content of slack space is OS-dependent, but Microsoft Windows' systems have not added non-zero RAM slack to a file for several years. The unused sectors in the last cluster of a file allocated by Windows typically contain deleted data. It is entirely within the FAT specification for an operating system to clear all sectors when it allocates a cluster to a file, but this is rare.

Microsoft Windows does not show any data after it finds a directory entry with all zeros. Therefore, it is relatively easy for someone to create a directory with only a few files and then use the rest of the directory space for hiding data. The allocated size of a directory should be compared to the number of allocated files. A logical file system search should find any data that is hidden in these locations.

While doing research for this book, I found some interesting things about directory entries with the volume label attribute, and they can be used to hide data and provide some clues. The access and created times in the volume label entry are frequently set to 0, but the last written time is usually set by Windows to the current time when the file system is created. So you may be able to get some information about when the file system was created, although this value can of course be modified.

With Windows XP, a volume label directory entry can be used to hide data. It is, after all, a normal directory entry and has fields to store the starting cluster of a cluster chain. If you manually edit the starting cluster field to contain the address of an unused cluster and then go to the FAT and manually edit it to create a cluster chain, the ScanDisk program in Windows XP will not raise any warnings or errors. The cluster chain will stay allocated. Typically, if a directory entry is not referencing a cluster chain, the ScanDisk program will place it in a lost clusters directory. The ScanDisk program in Windows 98 will detect that clusters have been allocated to the volume label and remove them.

Another interesting behavior of Windows XP and the volume label attribute is that you can have as many entries as you want and in any location. The specification for one entry in the root directory is not enforced. Therefore, you can have multiple entries in multiple directories that are flagged as volume labels and hiding data. The ScanDisk program in Windows 98 detects this and alerts the user. This process is currently manual, but tools could be developed in the future to do this. I have a test image on the Digital Forensic Tool Testing (DFTT) site for FAT volume labels [Carrier 2004b].

Analysis Scenarios

File System Creation Date

We encounter a FAT file system with only a few files and directories in it. This could be because the file system was not used much, because it was recently formatted, or because there is data being hidden from the investigator. We can test the second hypothesis by looking at the times of the directory entry with the volume label attribute set in the root directory. Our analysis tool lists the directory entry as a normal file, and we see that it was formatted two weeks before the disk was acquired. This explains why there are few file names listed, but there could still be hidden data and data from the previous file system. The next scenario will look for data from the previous file system.

Searching for Deleted Directories

We have a FAT file system and want to recover deleted directories. This scenario can occur during a normal investigation where we want to find deleted files or when a FAT file system has been recently formatted and we want to find the directories from before the format. For this example, we are going to extract the unallocated space and then search for the signature associated with the . directory entry that starts a directory.

I am going to do this in TSK, but you can use any tool that allows you to search only unallocated space and that allows you to search for a hexadecimal value. With TSK, we must extract the unallocated space using `dls`.

```
# dls –f fat fat-10.dd > fat-10.dls
```

We are going to search for the first four bytes of a directory, which corresponds to the ASCII values for ". " (a period and three spaces). This is 0x2e202020 in hexadecimal. We search our unallocated space for this signature using `sigfind`.

```
# sigfind –b 512 2e202020 fat-10.dls
Block size: 512 Offset: 0
Block: 180 (-)
Block: 2004 (+1824)
Block: 3092 (+1088)
Block: 3188 (+96)
Block: 19028 (+15840)
[REMOVED]
```

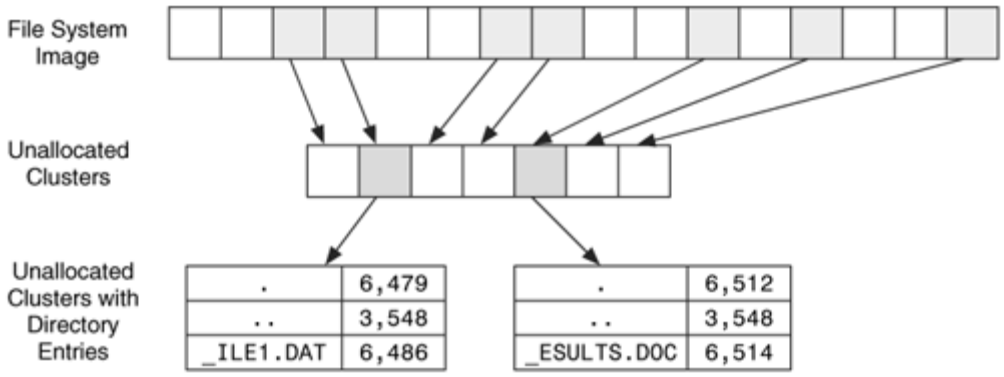
We view the contents of sector 180 in the `fat-10.dls` image (not the original file system image). It has the following contents:

```
# dd if=fat-10.dls skip=180 count=1 | xxd
0000000: 2e20 2020 2020 2020 2020 2010 0037 5daf . ..7].
0000016: 3c23 3c23 0000 5daf 3c23 4f19 0000 0000 <#<#..].<#0.....
0000032: 2e2e 2020 2020 2020 2020 2010 0037 5daf .. ..7].
0000048: 3c23 3c23 0000 5daf 3c23 dc0d 0000 0000 <#<#..].<#......
0000064: e549 4c45 312e 4441 5420 2020 0000 0000 .ILE1.DAT ....
0000080: 7521 7521 0000 0000 7521 5619 00d0 0000 u!u!....u!V.....
[REMOVED]
```

Each directory entry is 32 bytes, and there are three entries shown here. The first two are for the . and .. entries. If we interpret the first two entries, we see that the . entry points to cluster 6,479 (0x194f) and the .. entry points to cluster 3,548 (0x0ddc). The third entry is for a file, and it starts in cluster 6,486 (0x1956) with a size of 53,248 (0xd000) bytes. File recovery can be performed on this file now that we know its starting address and size.

The steps that were performed in this scenario are shown in Figure 9.14. We extracted the unallocated space and then searched for clusters that start with a . directory entry.

Figure 9.14. Process used to find deleted directories based on the directory entry structure.



File Name Category

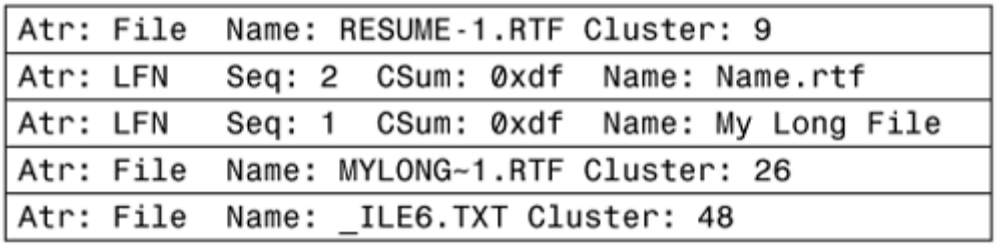
Typically, analysis in the file name category allows us to map a file name with a metadata structure. FAT does not separate the file name address and metadata address, so we will not be doing that here. In fact, we already saw the basics of file and directory names in the "Metadata Category" section because the file name is used as a metadata address. To recap, the directory entry data structure contains the name of a file using the 8.3 naming convention, where the name of the file has eight characters and the extension has three characters. In this section we focus on how FAT handles long file names.

If a file is given a name that is longer than eight characters or if the name has special values, a long file name (LFN) type of directory entry is added. Files with an LFN also have a normal, short file name (SFN), directory entry. The SFN entry is needed because LFN entries do not contain any of the time, size, or starting cluster information. They contain only the file name. The LFN data structure is discussed in detail in Chapter 10. When an LFN entry becomes unallocated, the first byte in the data structure is set to 0xe5.

The SFN and LFN data structures have an attribute field in the same location, and the LFN entries use a special attribute value. The remaining bytes in the entry are used to store 13 Unicode characters encoded in UTF-16, which are 2 bytes each. If a file name needs more than 13 characters, additional LFN entries are used. All LFN entries precede the SFN entry in the directory and contain a checksum that can be used to correlate the LFN entries with the SFN entry. The LFN entries are also in reverse order so that the first part of the file name is closest to the SFN entry.

Figure 9.15 shows a list of directory entries that exist in our example image. The directory entries are parsed by hand in Chapter 10, but overall, there are two allocated files in this directory, and one has a long file name. It has two LFN directory entries preceding its SFN entry, and the entries have the `My Long File Name.rtf` name in backwards order. Notice that the checksum values are the same for each entry, which is calculated based on the SFN.

Figure 9.15. Directory entries from the example image where there are three files, and one file has a long file name and another has been deleted.



Here is the output from running the `fls` tool on this directory in our example file system image:

```
# fls -f fat fat-2.dd
r/r 3: FAT DISK     (Volume Label Entry)
r/r 4: RESUME-1.RTF
r/r 7: My Long File Name.rtf (MYLONG~1.RTF)
r/r * 8: _ile6.txt
```

We can see the LFN on the third line and the deleted file on the fourth output line. Recall that the metadata addresses are assigned based on the directory entry slot. There is an address gap between `RESUME-1.RTF and My Long File Name.rtf` because of the LFN entries in slots 5 and 6.

LFN entries have Unicode characters, which allow them to have international characters in addition to the American English characters in ASCII. Prior to the introduction of the LFN entries, the international community was able to create file names with their native symbols by using code pages. ASCII uses only the first 128 values out of a total 256.

Code pages use the additional 128 entries and assign characters to them. The SFN entry may contain values from a code page instead of normal ASCII. A listing of code pages can be found at Microsoft [Microsoft 2004a].

Allocation Algorithms

The file names are stored in the same data structures as the metadata, so the allocation algorithms for the file name category are the same as for the metadata category. The one difference is that the LFN entries must come before the SFN, so the OS will look for enough room for all entries. Therefore, when using a first available strategy, the OS will skip over entries that do not have other unallocated entries around them.

The deletion routines are the same as were previously discussed in the "Metadata Category" section, specifically that the first byte of the entry is replaced with 0xe5. With an SFN entry, this will overwrite the first character of the name and an LFN directory entry will have its sequence number overwritten. Typically, when we recover deleted files we need to supply the first character of the short name, but if the file also has a long name, we can use its first character. When a directory is deleted, the entries inside it may or may not be changed.

Analysis Techniques

Analysis of the file name category is performed to find a specific file name. To do this in FAT, we first locate the root directory, which varies depending on the version of FAT. With FAT12/16, the root directory starts in the sector following the FAT area, and its size is given in the boot sector. With FAT32, the starting cluster address of the root directory is given in the boot sector, and the FAT structure is used to determine its layout.

We process the contents of a directory by stepping through each 32-byte directory entry structure. If the attribute for the structure is set for an LFN entry, its contents are stored and the next entry is examined. This storage process repeats until we find an entry that is not an LFN and whose checksum matches that of the long name entries. The allocation status of an entry is determined using the first byte, which works like a flag.

When we find a file or directory of interest, we will likely want to find its corresponding metadata. This is very easy with FAT because all the metadata for a file is located in the directory entry. This means that the metadata associated with a file name will not become out of sync after the file is deleted. The metadata associated with an unallocated file name will be accurate until both are overwritten.

Analysis Considerations

Because the file name and metadata are in the same data structure, we will always have a name for unallocated entries that we find. If LFNs are used, the entire long name can be recovered, although we may lose part of the name when the first byte is changed to 0xe5.

I observed with a Windows XP system that the scandisk tool for FAT file systems is not very exhaustive. I was able to create LFN entries that did not correspond to any SFN entries, and no error was given. This allowed me to save small amounts of data and not raise any alarms, and the data would not be shown when the directory contents were listed. Depending on how the keyword search of an analysis tool is done, the values hidden in the LFN entries may not be found.

As mentioned in the "Metadata Category" section, an attacker may also be able to hide data at the end of a directory. Some OSes will stop processing directory entries after finding one that is all zeros. Data could be hidden after that all zero entry, although another implementation of FAT could skip over that zero entry and continue to process entries until the end of the directory.

If you are using the keyword search mechanism of a search tool to look for a file name, make sure that you use the Unicode version of long file names. The SFN is stored in ASCII, but the LFN entries are in Unicode. You also will have to keep in mind that the entire name might not be stored sequentially, and it might be broken up into smaller pieces in multiple parts of multiple LFN entries (see the scenario in the next section).

Analysis Scenarios

To illustrate some of the analysis techniques, this section contains two scenarios. One is about searching for a file name, and the other is about the relative ordering of directory entries.

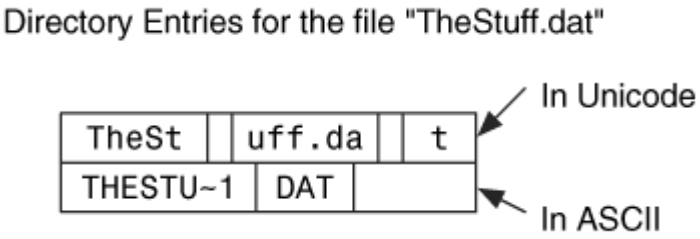
File Name Searching

Suppose that we want to find all references to a file named `TheStuff.dat`. We are interested in both allocated and deleted instances of this file. How do we do this if our analysis tool does allow us to search for file names?

If we do a keyword search of the file system for the name, we might find the content of files that reference our file. However, we won't find the file itself because the name is stored in an LFN entry and the SFN entry has a modified version of the name. Therefore, we can either search for "THESTUF~1DAT" in ASCII or modify our search for the long name entry.

If we want to find the long version of the name, we need to split the name up into chunks. Each long name entry can hold 13 characters and is broken up into chunks of five, six, and two characters. Therefore, our long entry will be broken up into "TheSt," "uff.da," and "t" in Unicode. To reduce the number of false positives, we should search for the longest string, which is "uff.da." A graphical representation of the directory entries for this file is shown in Figure 9.16.

Figure 9.16. Performing an ASCII or Unicode keyword search for a file name may not find its directory entry as shown here.



Directory Entry Ordering

While investigating a system, we find a directory full of contraband graphic images, and we want to determine how they got there.[1] For example, were they downloaded individually or copied from a CD? We notice that the directory entries in the directory are in alphabetical order, and we find this strange because Windows does not typically create directory entries in a sorted order. There are also no unallocated entries in the directory.

[1] Thanks to Eoghan Casey for suggesting the basic concept of this example.

We develop several hypotheses about this directory, including

- It was copied or moved to this location, and the new entries were created in alphabetical order.
- It was created by opening a ZIP file, and the unzip program created them in alphabetical order.
- The user downloaded each of these files in alphabetical order because that is the way they were listed on the remote server.

To test the first scenario, we can look at the created and last written times of the files. Files that are the result of a copy will have a created time that is more recent than its last written time. In this case, all the files have a created time that is equal to the last written time, so copying does not seem likely if we can trust the temporal data.

To test if the directory was moved, we set up a system that is similar to the one being investigated and conduct some tests. We create a directory with the same names as those we are investigating and create them in a non-alphabetical order. To test the moving hypothesis, we set the sorting option in the file manager to sort by name and then move the files by dragging and dropping. This results in a directory with entries that are almost in alphabetical order, but not quite. The entries are sorted more than they were in the original location, but not fully sorted. We repeat the tests by using different window sorting options, but still do not get the desired effect. We do find that if each file was moved individually in the sorted order, the new directory is sorted.

The second scenario was that the files were created from opening a ZIP file. We run more tests by creating a ZIP file that contains the files in question and then extracting them to a new directory. We find that the files are extracted from the ZIP file in the order that they were added to the ZIP file. Therefore, if they were added in alphabetical order, they would be created in that order. We also notice that the resulting files have the same created and written times and they are equal to the original last written time of the file. Therefore, we do not find any basic evidence to show that this scenario could not have occurred.

The last scenario was that the files were created from downloading them in the sorted order. We run tests and find that this could occur, but the time values raise doubts about this scenario. The created and written times of the different files are days and sometimes months apart. Therefore, the user would have had to download these files in the sorted order over the course of months. This could be a reasonable situation if the files were released on a periodic basis and the naming convention of the files was based on when they were released.

From our three scenarios, we conclude that the last two are the most likely. We will investigate both of these scenarios in more detail to find supporting evidence. For example, we can search for ZIP files or evidence of long-term downloading of these files. There are probably many other scenarios that could have caused this situation, and this is not meant to be an exhaustive set of tests. The results could vary by OS and ZIP versions.

The Big Picture

So far, we have seen the details of the FAT file system organized by the data categories. This is not always the most intuitive way to think about the file system, so I will summarize with some examples of how a file is created and deleted.

File Allocation Example

To review how a file is created in a FAT file system, we will step through the process to create the `dir1\file1.dat` file. In this example, the `dir1` directory already exists, the cluster size is 4,096 bytes, and the file size is 6,000 bytes.

1. We read the boot sector from sector 0 of the volume and locate the FAT structures, data area, and root directory.

2. We need to find the `dir1` directory, so we process each directory entry in the root directory and look for one that has `dir1` as its name and the directory attribute set. We find the entry, and it has a starting cluster of 90.

3. We read the contents of `dir1`'s starting cluster, cluster 90, and process each directory

entry in it until we find one that is unallocated.

4. We find an available entry and set its allocation status by writing the name `file1.txt`. The size and current time are also written to the appropriate fields.

5. We need to allocate clusters for the content, so we search the FAT structure. We allocate cluster 200 for the file by setting its entry to the EOF value.

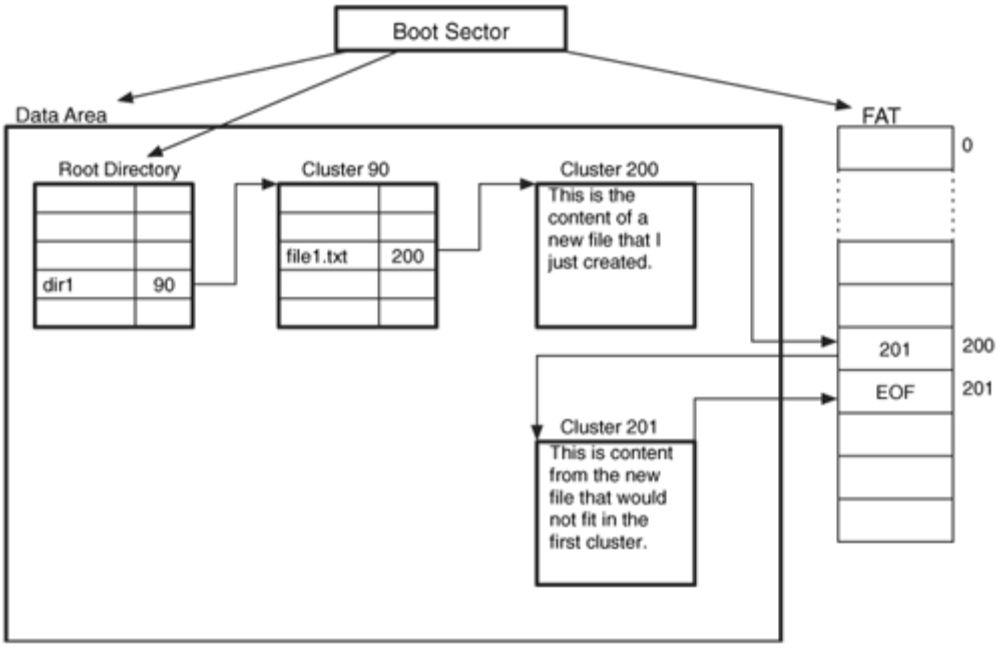
6. We write the address of cluster 200 in the starting cluster field of the directory entry. The first 4,096 bytes of the file content are written to the cluster. There are 1,904 bytes left, so a second cluster is needed.

7. We search the FAT structure for another cluster and allocate cluster 201.

8. The FAT entry for the first cluster, cluster 200, is changed so that it contains 201. The last 1,904 bytes of the file are written to cluster 201.

We can see the state of the file system after the file creation in Figure 9.17.

Figure 9.17. File system state after allocating the 'dir1\file1.txt' file.



File Deletion Example

We now repeat the previous example, but we are going to delete the `dir1\file1.txt file`.

1. We read the boot sector from sector 0 of the volume and locate the FAT structures, data area, and root directory.

2. We locate the `dir1` directory by processing each directory entry in the root directory and looking for one that has `dir1` as its name and that has the directory attribute set.

3. We process the contents of `dir1's` starting cluster, cluster 90, to find a directory entry that has the name `file1.txt`. We find that it has a starting cluster of 200.

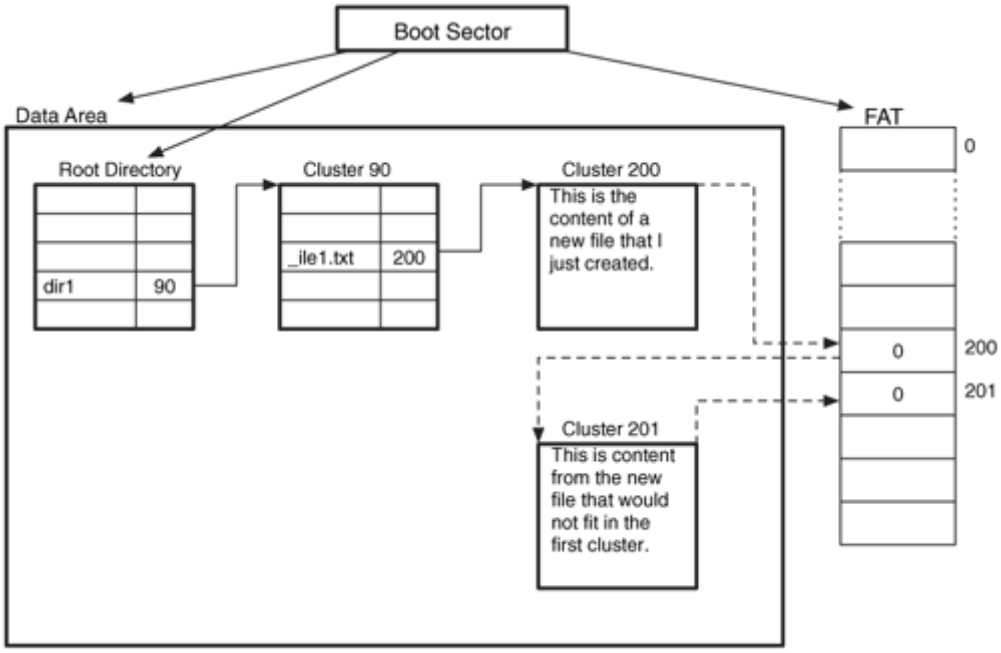
4. We use the FAT structure to determine the cluster chain for the file. In this case, the file has allocated clusters 200 and 201.

5. We set the FAT entries for clusters 200 and 201 to 0.

6. We unallocated the directory entry for `file1.txt` by changing the first byte to 0xe5.

This final state can be seen in Figure 9.18 where the pointers that have been cleared have dashed lines and the first letter of the file name is missing.

Figure 9.18. File system state after deleting the 'dir1\file1.txt' file.



Other Topics

There are a few topics that do not apply to any specific category, and they are discussed in this final section. We discuss file recovery techniques and what should be performed during a file system consistency check. We also examine how to determine the file system type based on the number of clusters in the file system.

File Recovery

Tools have existed for file recovery from a FAT file system since the early DOS days, but there is little documented theory on how it should be done. Recall that when a file is deleted from within Windows, the directory entry is marked as unused and the FAT entries for the clusters are set to 0. We saw an example diagram of this in Figure 9.18. To recover the file, we have the starting location and the size of the file. We have no information about the remaining clusters in the file.

We can try to recover the file data by reading the data from the known starting cluster. A recovery tool (or person) has two options when it comes to choosing the remaining clusters to read. It can blindly read the amount of data needed for the file size and ignore the allocation status of the data, or it can read only from the unallocated clusters.

The second option will succeed more often than the first option because it will recover some fragmented files. To explain the differences I will use Figure 9.19, which shows six clusters from a file system and three different scenarios. The file is 7,094 bytes in size and each cluster is 2,048 bytes, so we know it allocated four clusters. We also know that the starting cluster of the file was cluster 56. The light gray clusters represent the actual location of the file content for each scenario.

Figure 9.19. Three example scenarios for discussing file recovery.

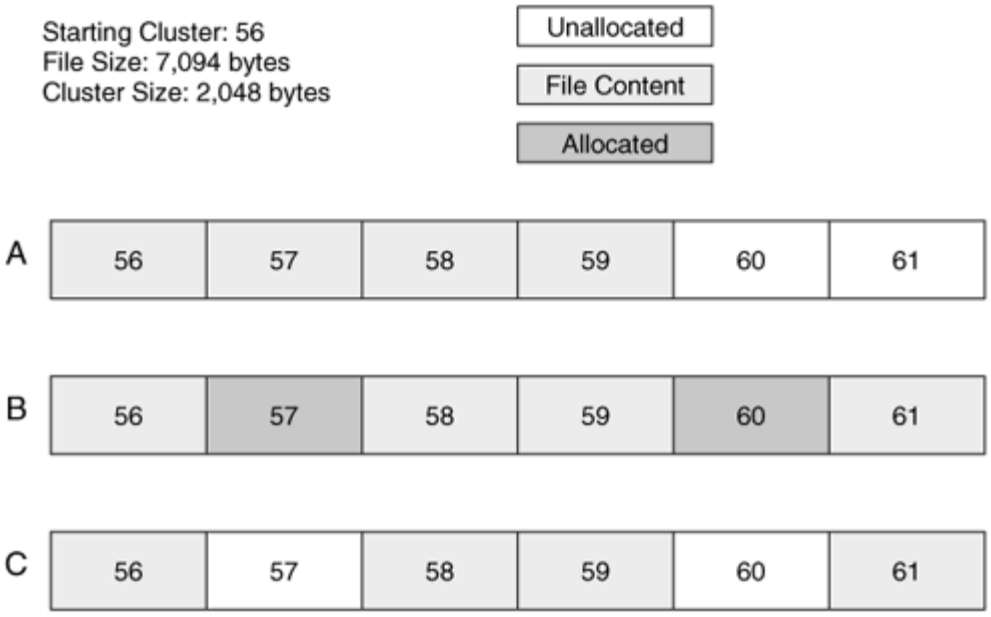


Figure 9.19(A) shows the scenario where the four clusters of the file are consecutive. In this scenario, both options will correctly recover clusters 56 to 59. Figure 9.19(B) shows the scenario where the file was fragmented into three chunks and the clusters in between the fragments, clusters 57 and 60, are still allocated to another file when the recovery takes place. In this scenario, option 1 will recover clusters 56 to 59 and incorrectly include the contents of cluster 57. Option 2 will correctly recover clusters 56, 58, 59, and 61. Figure 9.19(C) shows the scenario where the file allocated the same fragments as Figure 9.19(B), but the clusters in between the fragments are not allocated when the recovery takes place. In this scenario, both options will incorrectly recover clusters 56 to 59 like they did in the first scenario. There are other scenarios that could occur during file recovery, but they involve part of the file being overwritten, in which case no technique will accurately recover the data. Therefore, recovery option 2, where the allocation status of clusters is considered, can recover deleted files from more scenarios than option 1. Eoghan Casey performed some file recovery tests with WinHex and EnCase and found that WinHex version 11.25 used the first option and EnCase version 4 used the second option [Casey 2004].

A final note on recovery is that multiple cluster directories are likely to be fragmented. This is because the second cluster is only allocated when it is needed and it is highly unlikely that the next consecutive cluster will be available (because that cluster was probably allocated to one of the files in the directory). Therefore, directories are harder to recover, unless the directory is the result of a copy or the file system has been recently defragmented. When a directory is copied and its size is known, then consecutive clusters could be allocated for it.

If a user frequently runs defragmenting software on her file system, recovery of the files deleted after the last defragmenting will be easier because more of the files will be in consecutive clusters. On the other hand, files that were deleted prior to the defragmenting will be very difficult to recover because the clusters could have been reallocated.

I have created a FAT file recovery test image, and it is on the DFTT site [Carrier 2004a]. The test image has several deleted files and directories that you can use to test your recovery tools.

Determining the Type

In the beginning of this chapter, I mentioned that there is no single field that identifies a FAT file system as FAT12, FAT16, or FAT32. Now that we have discussed all the areas and concepts for a FAT file system, I'll cover the methods for calculating the type.

The official method determines the type based on the number of clusters in the file system. To get the number of clusters, we need to determine how many sectors there are in the data area and then divide that number by the number of sectors per cluster.

For FAT12 and FAT16, the root directory takes the first sectors of the data area, and cluster 2 starts after it. For FAT32, the root directory is dynamic, and cluster 2 starts at the beginning of the data area. There is a value in the boot sector that identifies how many entries there are in the root directory, and for FAT32 this value is 0. We can calculate the number of sectors in the root directory by multiplying the number of entries by 32 bytes (the size of each directory entry), dividing by the number of bytes per sector, and rounding up (if needed):

```
((NUM_ENTRIES * 32) + (BYTES_PER_SECTOR – 1)) / (BYTES_PER_SECTOR)
```

The previous calculation will be zero for FAT32. We determine the number of sectors that are allocated to clusters by taking the total number of sectors in the file system and subtracting the size of the reserved area, the size of the FAT area, and the size of the root directory. In other words, we are subtracting the sector address of cluster 2 from the total number of sectors.

```
TOTAL_SECTORS – RESERVED_SIZE – NUM_FAT * FAT_SIZE – ROOTDIR_SIZE
```

Lastly, we divide the number of sectors in the data area by the number of sectors in a cluster. If this value is less than 4,085, the file system should be FAT12; if the value is 4,085 or greater and less than 65,525, the file system should be FAT16, and any size greater than or equal to 65,525 should be FAT32. These values correspond to the maximum number of clusters that a FAT can store. Recall that there are 8 reserved values for end of file, 1 for bad clusters, and the unused entries in slots 0 and 1.

When formatting a volume in Windows, you are typically given a choice if you want FAT16 or FAT32. Windows will chose the cluster size such that the resulting number of clusters will fall into the previous range. For example, a 4-kilobyte cluster could be used if you format a file system as FAT16, and a 2-kilobyte cluster could be used if you format it as FAT32.

Consistency Check

When investigating a file system, it is useful to perform some consistency checks to identify corrupt file systems and hidden data. For the boot sector and other data structures in the reserved area of a FAT file system, a consistency check should verify that the defined values are in the appropriate range, and it should examine the unused locations for non-zero values. For example, there are many sectors in the reserved area that are not used in every file system. If a backup boot sector is available for a FAT32 file system, a consistency check should compare the two and report any differences.

The backup and primary FATs should be compared to verify that they have the same values. Each entry that is marked as bad should be examined because most hard disks fix errors before the operating system notices them. Floppy disks may have valid bad sectors, though. The space between the FAT entry for the last cluster and the end of the sector allocated to the FAT should be examined for each FAT because this space is not used by the file system and could contain hidden data. Space between the end of the last cluster and the end of the file system might exist that does not have a cluster address.

The root directory and its subdirectories should be examined, and each cluster chain in the FAT should be checked to make sure that an allocated directory entry points to the start of it. The reverse check also should be done to make sure that allocated directory entries point to allocated clusters. If multiple directory entries point to a cluster chain, Microsoft recommends that both files be copied to a new location and the original versions deleted [Microsoft 2003b]. The length of the cluster chain should be the number of clusters needed for the size of the file.

Any directory entries that are marked as volume labels should not have a starting cluster, and there should only be one volume label entry in the file system. The checksums for the allocated LFN directory entries should be examined and compared to the allocated SFN entry. If a corresponding SFN entry cannot be found, the LFN entries should be examined. This could be a result of a file system crash, using an OS that doesn't support long file names, or they could contain hidden data. Any directory entries in a directory that are all zeros or random data and have allocated entries before and after it could be the result of a wiping tool. Also check if there are any directory entries after a null entry. Some OSes will not show entries after a null entry.

Summary

FAT file systems have relatively few data structures but present challenges with respect to file recovery and temporal-based auditing. The data structures for the file system are well defined in the specification, but guidelines for allocation and the updating of times are not. Furthermore, there are many systems that use the FAT file system, and each may choose different algorithms. It is a good idea to test and observe the allocation algorithms for the types of systems that you are examining. If you have not been reading Chapter 10 along with this chapter, then I recommend you read the next chapter if you are interested in the layout of the data on disk.

FAT file systems will be around for some time to come and are well suited for an investigator to examine by hand. Many hex editors offer support for FAT file systems and allow you to examine each of the data structures. Unfortunately, a suspect can use those same hex editors to modify the file system contents.

After reading this chapter, you may be questioning the five-category data model, but it should be more intuitive from here on.

Bibliography

Bates, Jim. "File Deletion in MS FAT Systems." September 23, 2002. http://www.computer-investigations.com/arts/tech02.html.

Brouwer, Andries. "The FAT File System." September 20, 2002. http://www.win.tue.nl/~aeb/linux/fs/fat/fat.html.

Carrier, Brian. "FAT Undelete Test #1." Digital Forensic Tool Testing, February 2004a. http://dftt.sourceforge.net/.

Carrier, Brian. "FAT Volume Label Test #1." Digital Forensic Tool Testing, August 2004b. http://dftt.sourceforge.net/.

Casey, Eoghan. "Tool Review—WinHex." Journal of Digital Investigation, 1(2), 2004.

Landis, Hale, "How It Works: DOS Floppy Disk Boot Sector." May 6, 2002. http://www.ata-atapi.com/hiwdos.htm.

Microsoft. "FAT: General Overview of On-Disk Format." FAT32 File System Specification, Version 1.03, December 6, 2000. http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx.

Microsoft. "Last Access Date." MSDN Library, February 2003a. http://msdn.microsoft.com/library/en-us/win9x/lfn_5mg5.asp?frame=true.

Microsoft. "How to Fix Cross-linked Files." Microsoft Knowledge Base Article—83140, May 10, 2003b. http://support.microsoft.com/default.aspx?scid=kb;en-us;83140.

Microsoft. "MS-DOS FORMAT Does Not Preserve Clusters Marked Bad." Knowledge Base Article—103548, May 6, 2003c. http://support.microsoft.com/default.aspx?scid=kb;en-us;103548.

Microsoft. "Description of NTFS Date and Time Stamps for Files and Folders." Microsoft Knowledge Base Article 299648, July 3, 2003d. http://support.microsoft.com/default.aspx?scid=kb;en-us;299648.

Microsoft. "Detailed Explanation of FAT Boot Sector." Microsoft Knowledge Base Article Q140418, December 6, 2003e. http://support.microsoft.com/kb/q140418/.

Microsoft. "Encodings and Code Pages." Global Development and Computing Portal, 2004a. http://www.microsoft.com/globaldev/getWR/steps/wrg_codepage.mspx.

Microsoft. "How to Cause ScanDisk for Windows to Retest Bad Clusters." Microsoft Knowledge Base Article—127055, December 16, 2004b. http://support.microsoft.com/default.aspx?scid=kb;en-us;127055.

Microsoft. "Windows 2000 Server Operations Guide (Part 1)." n.d. http://www.microsoft.com/resources/documentation/windows/2000/server/reskit/en-us/serverop/part1/sopch01.mspx.

Wilson, Craig. "Volume Serial Numbers & Format Verification Date/Time." Digital Detective White Paper, October 2003. http://www.digital-detective.co.uk/documents/Volume%20Serial%20Numbers.pdf.