Chapter 4
Volume Analysis

Learning Objectives
The objectives of this chapter are to:

- Understand how disk works and how data is structured on the surface of disk drive platters
- Demonstrate an understanding of concepts fundamental to disk partitioning
- Know about common types of disk partitioning systems
- Understand fundamental concepts of the most commonly encountered partition system and DOS-style partition system
- Know how to interpret partition-table hexdumps and how the data is laid out on a DOS-style disk
- Know how to use a The Sleuth Kit (TSK) utility mmls

When conducting digital forensic investigations, the most common source of digital evidence is hard disk or hard disk drive (HDD). This chapter is devoted to the fundamentals of hard disk. Following a brief introduction to hard disk geometry or internal structure, the concept of disk partitioning is introduced. Afterwards, the most common partition system, DOS-style partitions, is presented. Finally, this chapter gives an introduction to analysis techniques for disk volumes.

4.1 Hard Disk Geometry and Disk Partitioning

Some important concepts and definitions that will be used throughout this part of file system forensic analysis are first presented in Table 4.1 for reference.

© Springer Nature Switzerland AG 2018

X. Lin, *Introductory Computer Forensics*,
https://doi.org/10.1007/978-3-030-00581-8_4

Table 4.1 Definitions and common concepts

Table	A table is a collection of related data arranged in a predefined structure.
Table entry	A table contains many entries (or rows) of the same size, all of them assembled in an orderly manner. In most cases, entry number starts with 0, for example, 0, 1, 2, ... A unique number is assigned to each entry, and entry numbers are in increasing order.
Cluster	A group of data blocks, typically 4 kB and the smallest allocation of storage space assigned by an operating system.
Cluster chain	A group of clusters which make up an entire file.
Data block	The smallest unit of storage of a device, typically 512 bytes. Also known as a sector.
Dataset	A collection of data blocks (e.g. disk image) from which files are to be recovered.
File header	The first data block of a file which contains a beginning-of-file marker.
File fragment	A group of sequential data blocks that make up only part of the complete file and separate from the rest of the file by unrelated data blocks.
File footer	The last data block of a file which contains an end-of-file marker.

4.1.1 Hard Disk Geometry

There are various storage devices, for example, hard disks, USB drives, SD cards, Solid State Drives (SSD), floppy disks, tapes, CD-ROM, DVD, and hard disk is the most commonly used one. For simplicity, unless otherwise specified, we will use "disks" to refer to hard disks since hard disk is the storage device we are focusing on in the book. As shown in Fig. 4.1a, a disk consists of platters. Each platter has two magnetic surfaces, where data is stored. Disks use magnetic storage technology, and each platter surface requires its own dedicated head to read/write data. The surface of the disk is smooth and shinning, but actually much complicated. A magnetic film on the surface memorizes all information. The head magnetizes microscopic particles on platter surfaces to write data, and reads section of film that stores the data in sequence of 1 or 0. Each 1 or 0 is called a bit. Each square meter of the disk's surface can hold billion bits.

Data is stored into tiny concentric tracks on the disk's surface, which are arranged from the inner to the outer edge of the platter. A group of tracks with the same radius is called a *cylinder* (in Fig. 4.1a the dashed-line tracks belong to a cylinder). The number of tracks in a cylinder is twice the number of platters. In other words, it is the same as the number of disk's surfaces. Each track is divided into *sectors*. Each sector has a size of 512 bytes. It is important to note that sectors are always 512 bytes large; and sectors of other sizes can only be found in a few of the most modern hard drives available. Unless otherwise specified, assume that all sectors are 512 bytes. Newer hard drives use 4096 byte (4 kB) sectors, known as Advanced Format standard [7]. However, most of today's hard drives still use 512-byte sector as the basic unit of data storage.

Disks manufactured are initially blank; they don't contain tracks and sectors. Hence, disks need to be formatted in order to organize the disk surface into tracks and sectors. Most disks manufactured today have their disks preformatted. The

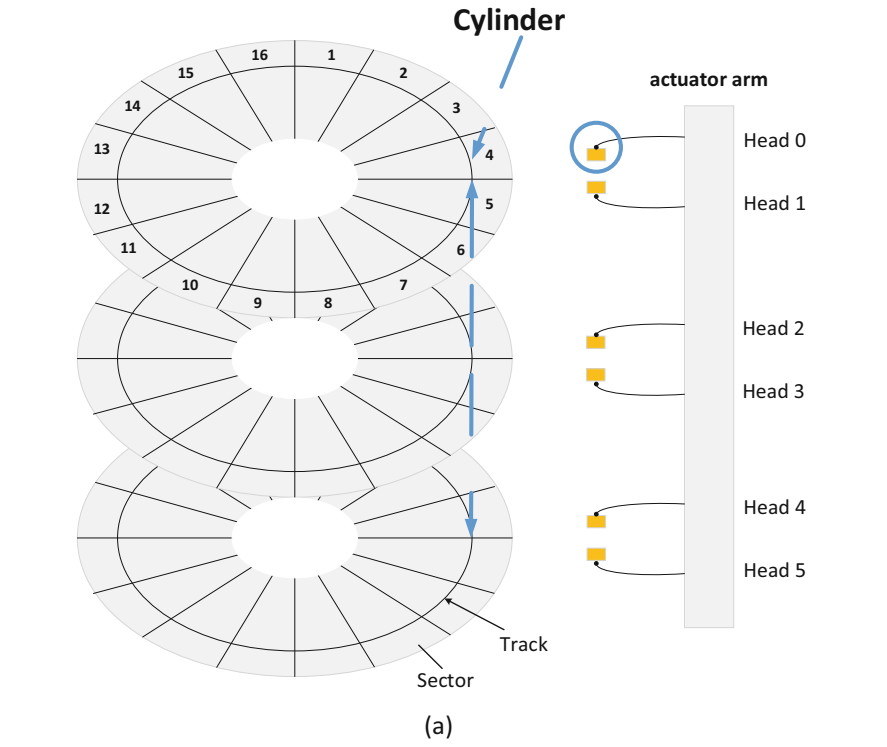


Fig. 4.1 Hard disk geometry. (**a**) Cylinder-head-sector. (**b**) An example of disk volume with size of 1801 sectors (or 1801X521 Bytes)

process of creating data structures including tracks and sectors directly to the storage medium or the disk surface is called disk formatting, often referred to as "*low-level formatting*". In other words, the process of formatting a disk applies addressing data to the platter's surface.

The smallest addressable unit (block) in a disk is sector. These blocks of data on the disk are mapped by two types of addresses:

- CHS (Cylinder-Head-Sector) address
- LBA (Logical Block Address) address

CHS sector addressing is an older method that is used to refer to the different disk's characteristics (cylinder, head, and sector). It no longer maintains a physical relationship of them. As it stands, a CHS address contains three information: Cylinder number, head number, and sector number. These three information can uniquely locate a sector on a disk according to their positions in a track, which is the sector number (or S in CHS). The track is determined by the head (or H in CHS) and cylinder (C in CHS) numbers. LBA is introduced to better address the *Hard Disk Drive* (HDD); although CHS is still being used by many utilities like partitioning. Thus, LBA supports CHS. LBA addressing is to sequentially number sectors, for example, the first sector on the disk is given an address of 0, i.e., sector 0. Since LBA supports CHS, CHS addresses can be converted to LBA addresses using the following formula:

$$\begin{aligned} \text{LBA} = &(((\text{CYLINDER} * \text{heads\_per\_cylinder}) + \text{HEAD}) * \text{sectors\_per\_track}) \\ &+ \text{SECTOR} - 1, \end{aligned}$$

where LBA is the LBA address, CYLINDER, HEAD, and SECTOR stand for cylinder, head, and sector numbers of the CHS address.

For example, consider a disk that reports 16 heads per cylinder and 63 sectors per track. If we have a CHS address of cylinder 2, head 3, and sector 4, its conversion to LBA would be as follows:

$$(((2 * 16) + 3) * 63) + 4 - 1 = 2208.$$

Hence, The LAB address of CHS = (2, 3, 4) is Sector 2208.

In CHS addressing, data is accessed by referring to the cylinder number, head number, and sector number where the data is stored. However, LBA maintains a complete mapping of sequence numbers (LBA addresses) and their locations including all tracks and sectors (CHS addresses) in the disk.

The collection of addressable sectors on a disk is called disk volume. Figure 4.1b shows an example of disk volume. Normally, a physical disk forms a volume, which is the most common case for our today's computers. Nevertheless, a disk volume can contain multiple physical disks. In Fig. 4.2, two physical disks can be combined and set up as a logical disk volume, for example, by Logical Volume Manager (LVM) in Linux. It creates the appearance of one large disk.

In a logical disk volume, the sectors can be addressed in two ways using LBA sector addressing. The first is called *logical disk volume address*, which is a sector address that is the distance relative to the starting of the disk volume. The second is known as *physical address*. It is only applied to a single physical disk. The physical address is the distance relative to the starting of the disk. If a disk volume only

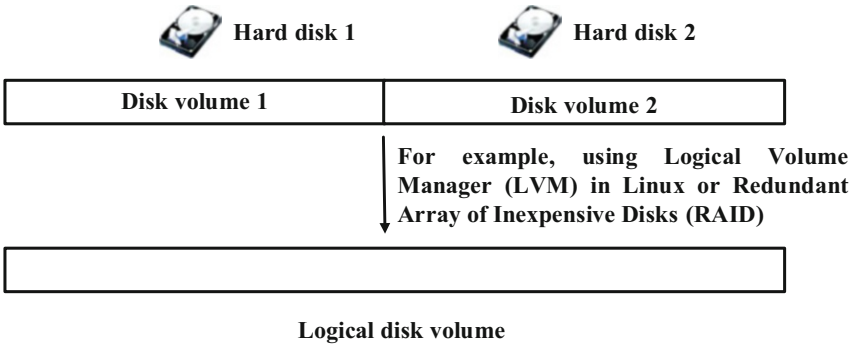


Fig. 4.2 An example logical disk volume with two physical disks

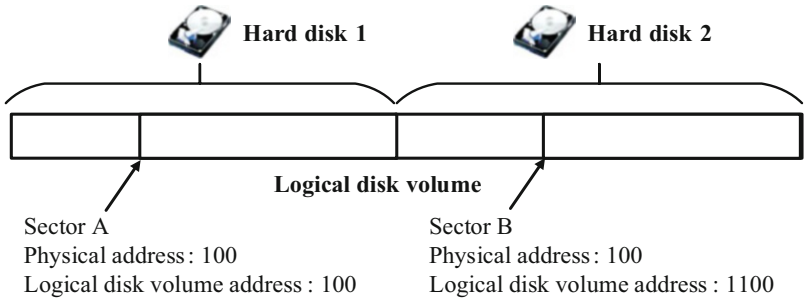


Fig. 4.3 Sector addressing in logical disk volume

contains one physical disk, physical address and logical disk volume address are the same in such a case. However, they become different when a disk volume is formed by multiple disks, as shown in Fig. 4.3. In the example shown in Fig. 4.3, a disk volume contains two physical disks (the size of 1000 sectors each). Logical disk volume address is the distance relative to the starting of the disk volume, which is the starting of disk 1. Therefore, Sector A's physical address and logical disk volume address are the same, 100, since it belongs to Disk 1. However, Sector B's logical disk volume address will be 1100 because it is located at offset 100 in sectors in Disk 2. Thus its distance to the starting of disk 1 should consider adding the size of Disk 1, although its physical address is still 100.

4.1.2 Disk Partitioning

A hard disk is usually divided into multiple parts, called *partitions*. Partitions are used to separate disk into logical storage units. It acts like minidisks (or multiple logical storage) of the HDD [5, 8]. This allows different file systems to be used on each partition. File system is some special data structures, which allow files stored on

a disk to be easily accessed. It will be detailed in the next chapter. This is also particularly useful since the capacity of modern disks has increased dramatically and becomes very large, but some file system cannot handle large disks. Some other benefits include [1, 2]:

- Allow images of the disk to be backup
- Easy to recover or prevent corrupted file systems
- Easy to share data among different Operating Systems through a partition dedicated for data and formatted with a file system supported by all OSes
- Improve data access performance
- "Short Stroking", which reduces the average *seek time* that a head requires to move between the tracks in order to read/write

However, there are also some disadvantages using multiple partitions:

- "Fragmentation" is likely to happen because of the size reduction in contiguous free data blocks (or clusters) which can be occupied on each partition.
- Constraints such as ability to use the full capacity of the disk when the disk is divided between two partitions. For instance, you cannot copy a 6 GB DVD image file to a 3 GB disk partition.

The partition will need to be formatted before use. The process of *formatting* is also known as "making a file system" on a disk partition or logical drives, which will be discussed in the next chapter. The formatted partition is also called *volume*, which is a storage area that can be accessed by a single Operating System (OS). Usually, a file system occupies the entire storage space allocated to a partition, which it resides in, and hence, the two terms *partition* and *volume* are often used interchangeably. However, they are not the same. First, it is possible that not the entire partition or logical drive is used when formatting. Some space is left unformatted and inaccessible to OS. The leftover storage space is also known as *volume slack*. Second, the difference between a volume and partition is that volume exists at a logical operating system level, while partition exists in physical media level [3].

There are many partitioning systems, including GUID (Globally Unique IDentifier) Partition Table (GPT), PC-based Partitions (or DOS-style partitions). Among them, the most common partition system used today is the DOS-style partitions, which will be discussed in detail in the subsequent section. Although DOS-style partitions are commonly used, GPT partitions have become popular, and have also been widely used in the latest OSes, such as Windows Server 2003 with SP1 and later [6]. Here we focus on DOS-style partitions.

4.1.3 DOS-Style Partitions

One popular disk partition system is called the **DOS** (Disk Operating System)-style partition system, also known as PC-based or **Master Boot Record** (**MBR**) partition style. DOS-style is usually called MBR style because it reserves the first 512-byte

Table 4.2 Basic structure of MBR sector [3]

Byte range (within MBR sector) in hexadecimal (bytes)	Length in decimal (bytes)	Relative byte offsets (within MBR sector) (in hexadecimal)	Description
0x000–0x1BD	446	0x000	Code area
0x1BE–0x1FD	64	0x1BE	Partition table with four 16-byte entries
0x1FE–0x1FF	2	0x1FE	Boot record signature (0xaa55)
MBR total size: 446 + 64 + 2 = 512 bytes			

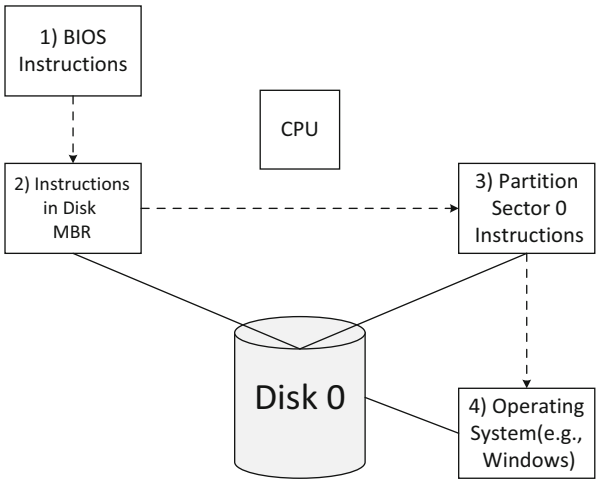


Fig. 4.4 Computer boot process

sector for the **MBR**. Disks using this type of partition system are called **MBR disks**. The **MBR** is not a partition, but a section of the disk that contains the **Partition Table**, as shown in Table 4.2, and code for initializing boot (such as **bootloader**, loading the operating system kernel files).

The code for initializing boot in MBR or the bootloader reads the MBR partition table, and searches for an "active" or bootable partition. If one is found, the bootloader will load the operating system kernel files found in the partition. If none of the partitions on a disk is found active or bootable, an error message "Missing operating system" appears. The entire booting process of a computer can be found as follows: As shown in Fig. 4.4, assume that our computer is off, let us turn on the computer. The computer's **BIOS** (Basic Input Out System), which contains instructions and setup for how your computer system should boot and how it operates, first triggers a **POST** (Power-on self-test). If the test fails, the computer will grind to a halt. This could be caused by a number of factors such as hardware failure, missing keyboard, etc. Once the POST passes, the BIOS boot sequence tells the computer to look for MBR of the first recognized storage device,

Table 4.3 Structure of a partition table entry [3]

Relative byte offsets (*within entry*) (in hexadecimal)	Length in decimal (bytes)	Byte range in hexadecimal (bytes)	Contents
0x0	1	0x0	Boot indicator (0x80 = *active*)
0x1	3	0x1–0x3	Starting CHS values
0x4	1	0x4	Partition-type descriptor (e.g. 0x06 = FAT16, 0x07 = NTFS, 0x0B = FAT32)
0x5	3	0x5–0x7	Ending CHS values
0x8	4	0x8–0xb	LBA address of the starting sector
0xc	4	0xc–0xf	Partition size (in sectors)
Partition total size: 1 + 3 + 1 + 3+ 4 + 4 = 16 bytes			

for example, Disk 0 in Fig. 4.4. Afterwards, the bootloader in MBR takes control of the execution, loads the OS kernel files, and transfers the control to the OS.

The Partition Table is a table that illustrates the layout of a physical disk. It is traditionally composed of four 16-byte entries, as shown in Table 4.3, each of them representing one primary partition. It is possible to divide the hard disk into more than four partitions by using **extended partition concept**, where one partition can be defined as extended and further divided into multiple **logical partitions** (or **Logical drives**). Each partition can be formatted and assigned with drive letters available for data storage. In such a case, an extended partition logically acts like a hard disk. The process of formatting is usually known as "making a file system" on a disk partition or logical drives. In other words, a specific file system structure, which is detailed later in next chapter, will be created after formatting. A partition or logical drive must be formatted before it can be available for data storage.

The way in which different OSes manage these partitions varies. Particularly, there are currently two mainstream ways: Windows-style and Linux-style. For example, in Windows, each partition is assigned with a drive letter, and can be formatted with a file system to be accessible as a volume, as shown in Fig. 4.5a. However, in Unix, different volumes are mounted to different directories (or folders) so they can be ready to use. The terms "folder" and "directory" are used interchangeably throughout this book. Also, it is worth noting that a different file system can be used on each partition, and some partitions can be hidden (like recovery partitions) which are invisible to computer users in a file browser such as Windows Explorer or DOS command line interface.

Figure 4.6 shows a hex dump of a MBR by using a Linux utility *xxd*. The left side (the seven digits before each colon) is the offset address, in hexadecimal format, which is used to locate individual bytes (starting at byte offset 0). The middle is the hex dump data, and the right is the ASCII interpretation of the dump data. Each byte represents a two-digit hexadecimal number. Table 4.2 explains what the MBR's content means, and where each variable is delimited. Table 4.3 shows the layout of partition table entry.

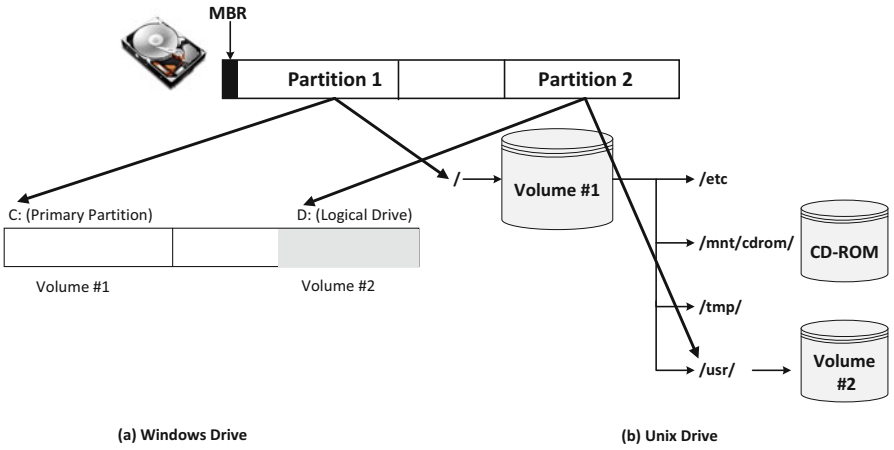


Fig. 4.5 Volumes in Windows and Unix

```
0000000: 33c0 8ed0 bc00 7c8e c08e d8be 007c bf00  3.....|......|..
0000010: 06b9 0002 fcf3 a450 681c 06cb fbb9 0400  .......Ph.......
0000020: bdbe 0780 7e00 007c 0b0f 850e 0183 c510  ....~..|........
0000030: e2f1 cd18 8856 0055 c646 1105 c646 1000  .....V.U.F...F..
0000040: b441 bbaa 55cd 135d 720f 81fb 55aa 7509  .A..U..]r...U.u.
0000050: f7c1 0100 7403 fe46 1066 6080 7e10 0074  ....t..F.f`.~..t
0000060: 2666 6800 0000 0066 ff76 0868 0000 6800  &fh....f.v.h..h.
0000070: 7c68 0100 6810 00b4 428a 5600 8bf4 cd13  |h..h...B.V.....
0000080: 9f83 c410 9eeb 14b8 0102 bb00 7c8a 5600  ............|.V.
0000090: 8a76 018a 4e02 8a6e 03cd 1366 6173 1cfe  .v..N..n...fas..
00000a0: 4e11 750c 807e 0080 0f84 8a00 b280 eb84  N.u..~..........
00000b0: 5532 e48a 5600 cd13 5deb 9e81 3efe 7d55  U2..V...]...>.}U
00000c0: aa75 6eff 7600 e88d 0075 17fa b0d1 e664  .un.v....u.....d
00000d0: e883 00b0 dfe6 60e8 7c00 b0ff e664 e875  ......`.|....d.u
00000e0: 00fb b800 bbcd 1a66 23c0 753b 6681 fb54  .......f#.u;f..T
00000f0: 4350 4175 3281 f902 0172 2c66 6807 bb00  CPAu2....r,fh...
0000100: 0066 6800 0200 0066 6808 0000 0066 5366  .fh....fh....fSf
0000110: 5366 5566 6800 0000 0066 6800 7c00 0066  SfUfh....fh.|..f
0000120: 6168 0000 07cd 1a5a 32f6 ea00 7c00 00cd  ah.....Z2...|...
0000130: 18a0 b707 eb08 a0b6 07eb 03a0 b507 32e4  ..............2.
0000140: 0500 078b f0ac 3c00 7409 bb07 00b4 0ecd  ......<.t.......
0000150: 10eb f2f4 ebfd 2bc9 e464 eb00 2402 e0f8  ......+..d..$...
0000160: 2402 c349 6e76 616c 6964 2070 6172 7469  $..Invalid parti
0000170: 7469 6f6e 2074 6162 6c65 0045 7272 6f72  tion table.Error
0000180: 206c 6f61 6469 6e67 206f 7065 7261 7469   loading operati
0000190: 6e67 2073 7973 7465 6d00 4d69 7373 696e  ng system.Missin
00001a0: 6720 6f70 6572 6174 696e 6720 7379 7374  g operating syst
00001b0: 656d 0000 0063 7b9a f8e8 7499 0000 8020  em...c{...t.... 
00001c0: 2100 07fe ffff 0008 0000 0000 1f01 00fe  !.....Partition..
00001d0: ffff 07fe ffff 0008 1f01 b022 b01c 0000  ..........table"...
00001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001f0: 0000 0000 0000 0000 0000 0000 0000 55aa  ..............U.
```

Fig. 4.6 Dump data of MBR

Table 4.4 Structure of a CHS address

Relative byte offsets (*within CHS address*) (in hexadecimal)	Length in decimal (bytes)	Contents
0x0	1	Head
0x1	1	Sector is in bits 5-0; bits 9-8 of cylinder are in bits 7-6
0x2	1	Bits 7-0 of cylinder
CHS total size: 3 bytes		

Table 4.5 Limitations of CHS

CHS	Minimum value	Maximum value	Number of bits	Number of values
Sector	1	63	6	63
Head	0	255	8	256
Cylinder	0	1023	10	1024

In the partition table, a CHS address is represented by three-byte values. The structure of CHS address is shown in Table 4.4. As shown in Table 4.2, the partition table can be found at byte offsets 0x1BE to 0x1FD, and the partition table has four entries (16 bytes each). If any partition table entry's 16 bytes are all 0, it means that the corresponding partition doesn't exist. It can be observed in Fig. 4.6 that the partition will go through next exists, and obviously, it is the first partition.

Note that BIOS imposes the following limitations to CHS (Table 4.5):

It is worth emphasizing that in CHS addressing the sector numbers always start at 1. It may seem counterintuitive, but there is no sector 0 in CHS addressing.

Therefore, to calculate the CHS address, you will first need to break a CHS value down into three sections: Head, sector, and cylinder. Each is mainly represented by 1 byte in the partition; however, they have different bit length that BIOS imposes. For this example, we will calculate the starting CHS address of the first partition.

By observing Fig. 4.6, we know that the dump data of that address is "20 21 00", where 0x20 (or the first byte) is the head, 0x21 (or the second byte) represents the sector, and 0x00 (or the third byte) represents the cylinder plus the two bits at the top of 0x21. By converting the 2 digit hexadecimals repetition into binary, we see its 8 bits binary counterpart. Since length of head is 8 bits, the head is 32 when converting to decimal. However, sector consists of 6 bits, so we only take the first 6 bits from the least significant bits (or low 6 bits). Therefore, the sector is 16 in decimal. The remaining top 2 bits from 0x21 are added to cylinder as its two most significant bits, who has the length of 10 bits. This makes the cylinder 0. The detailed parsing process can be found in Fig. 4.7.

Assuming that the system discussed here is using little-endian. It means to obtain the real value of a multiple byte number, you need to reverse or flip the order of raw data. For example, the dump data "00 08 00 00" of LBA address is read from the least significant byte (or the last 00) so the real value is 0x00000800 or 2048 in decimal. To calculate the size of the partition, which usually is represented in MB or GB, we first need to parse out the number of sectors in partition, which is

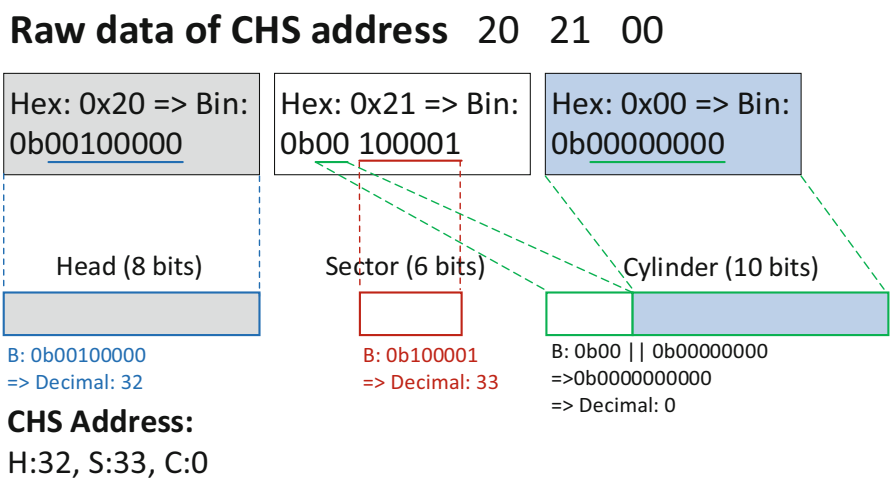


Fig. 4.7 CHS address parsing

Table 4.6 Partition table for partition entry #0

Starting CHS address	Cylinder: 0, head: 32, sector: 33
Ending CHS address	Cylinder: 1023, head: 254, sector: 63
Starting LBA address	0x00000800 => 2048
Size of the partition (MB)	0x011f0000 => 18808832 * 512 bytes = 9184 Mega-Bytes = 8.96875 GB Since we know the number of sectors in partition is 18808832
Type of partition	0x07 => NTFS

0x011f0000 or 18808832 in decimal. Then, we can obtain the partition size in bytes by multiplying it by 512. This is because one sector is 512 bytes long.

If you have followed the example, you should have the following answers for the first partition in Table 4.6.

Figure 4.8 shows the layout of the disk whose MBR partition table is analyzed above, which is displayed in Disk Management Console. It can be seen from Fig. 4.8 that the partition information including partition size and type matches our analysis results.

Also, The Sleuth Kit (TSK) provides a utility named *mmls*, which reads the partition table and displays the layout of a disk (or disk volume system). Finally, Fig. 4.9 shows an example output of The Sleuth Kit (TSK)'s *mmls* tool, detailing information about the layout of a disk.

Please note that the output of *mmls* on the disk image describes the starting, ending and size of the partitions (in sectors). One thing to note about the size of the partition is that the size is indicated by the number of sectors which a partition contains. For example, the size of the first partition is 248,223 sectors. You would need to convert the size of a partition from sectors to some commonly used measurements, such as megabytes (MB).

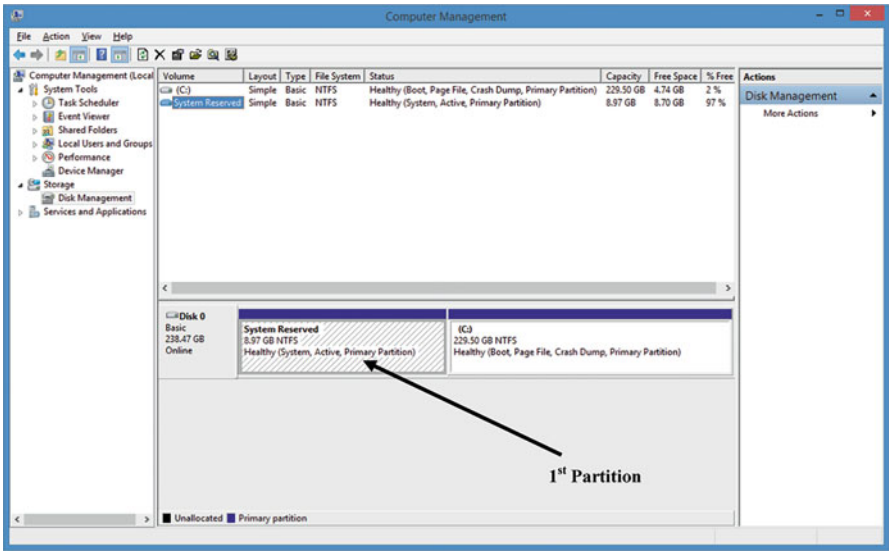


Fig. 4.8 The disk layout in disk management console

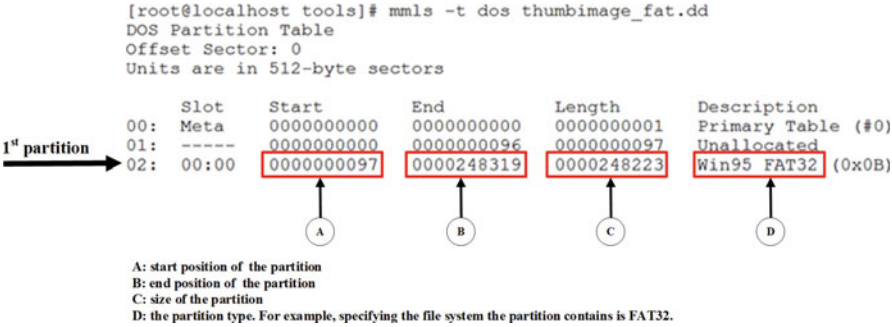


Fig. 4.9 Example output from The Sleuth Kit (TSK)'s mmls tool details information about the layout of the disk

4.1.4 Sector Addressing in Partitions

After disk partitioning, the sectors in partitions can be addressed in two ways: LBA sector addressing and a sector address, which is the distance relative to the start of a partition, also known as *logical partition address*.

Figure 4.10 shows a disk partitioned into two partitions. It is worth noting that not all sectors have partition addresses, specifically for these in non-partitioned area. For example, Sector A doesn't belong to any partition, and has a physical address of 36, but not partition address. Whereas, Sector B has both: Physical address is 1058, and partition address is 38. A partition address is the distance relative to the starting

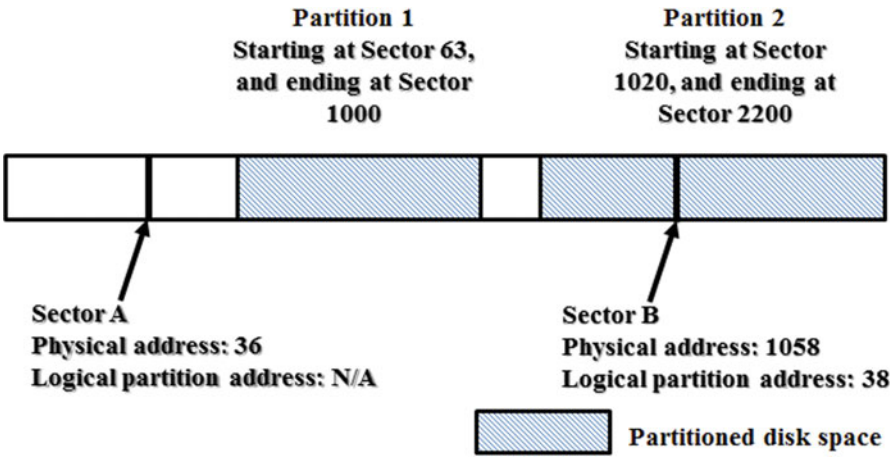


Fig. 4.10 An example disk layout

of a partition. The Partition 2 starts at sector 1020, and the physical address of Sector B is 1058. The distance between them is 38, which is the difference between two numbers. It can be obtained by subtracting 1020 from 1058. Hence the partition address of sector B is 38. Note that the (LBA) sector address starts at 0.

4.2 Volume Analysis

Next, we will discuss basics of volume analysis and commonly used volume analysis techniques.

4.2.1 Disk Layout Analysis

When it comes to forensic analysis of a disk, knowing how data is laid out on the disk is the first and foremost important thing. In doing so, we would need to locate the partition table, for example, the partition table in MBR can be found at byte offsets 0x1BE to 0x1FD. Also, from Table 4.3, we know how the partition table is organized and structured. Therefore, we should be able to figure out the information about each partition on a disk, including locations, types, and sizes. In other words, we are able to discover the layout of the disk, as shown in Fig. 4.11.

We can also check if any disk space is not used when disk partitioning. Extra care or attention should be paid to these non-assigned sectors, which can be used to hide data.

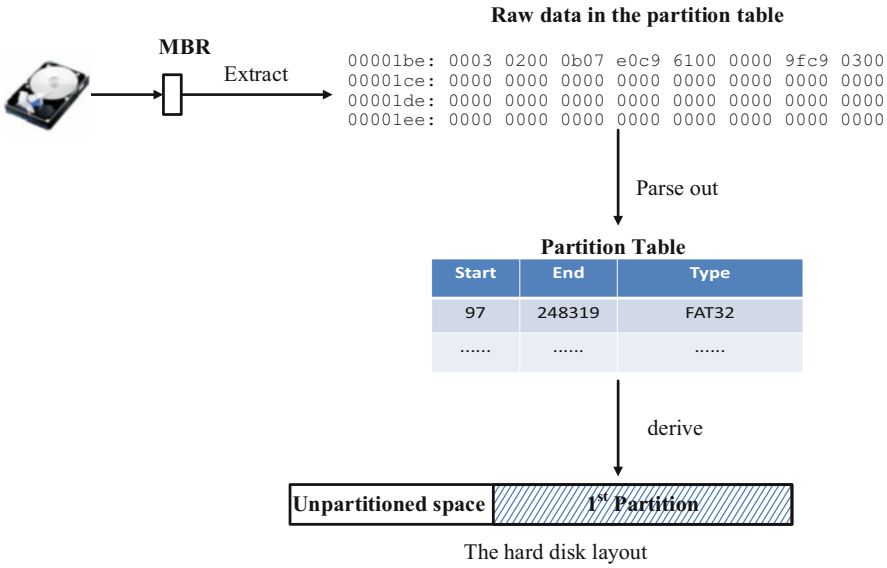
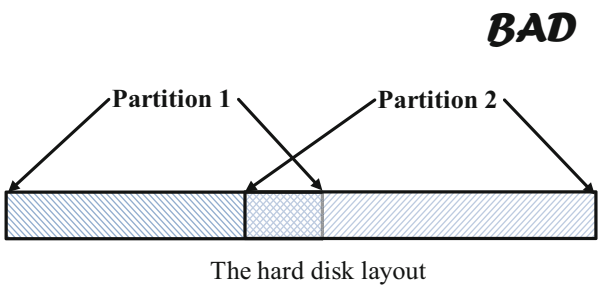


Fig. 4.11 Disk layout analysis

Fig. 4.12 Partition consistency check



We can also check if each entire partition or logical drive is used when formatting, which will be discussed in next chapter. This can be done by comparing the sizes of partition and file system made on it. If the size of file system is smaller than one of partition, it means there exists volume slack on disk.

4.2.2 Partition Consistency Check

By convention, a disk partition utility usually creates partitions in a way where one immediately follows the other if multiple partitions have to be created on a disk and partitions occupy the entire disk space. However, something inconsistent could occur due to many reasons, for example, Fig. 4.12 shows two partitions are overlapped, which could result unexpected file system corruption. Such kind of

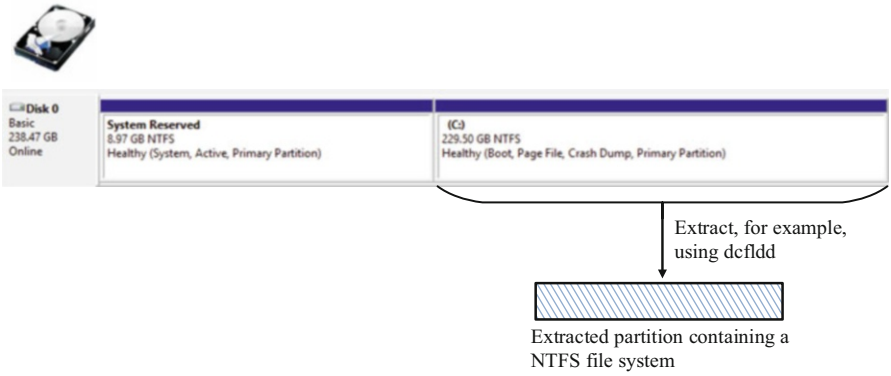


Fig. 4.13 Partition extraction

disk layout is invalid, and in principle, partitions should not overlap to each other. A further investigation should be launched to determine the cause if there is any inconsistency to the disk layout.

4.2.3 Partition Extraction

After we figure out the layout of the disk, we can conduct any further investigation by extracting each individual partition, for example, using *dcfldd*. These extracted partitions can be further analyzed, for example, using any file system analysis tools, which will be discussed in next chapter (Fig. 4.13).

4.2.4 Deleted Partition Recovery

Due to various reasons, accidental deletion, formatting, or even trail obfuscation against the investigation by criminals, a partition could be deleted. Hence, it is crucial to recover deleted partition(s) either at uncovering evidence during a forensic investigation or for maintaining business operations. Fortunately, when a partition on a disk is deleted, the partition contents aren't actually erased. Instead, the respective partition table entries are all zeroed, indicating the pre-partitioned but now deleted area is free and cannot be assessable. In order to recover the deleted partition, we need to figure out these important information about the partition, including the start point, the size, and type, and then put back to the zeroed partition table entries.

By convention, a disk is partitioned in a way where one partition immediately follows the other if multiple partitions have to be created on a disk and partitions occupy the entire disk space. Therefore, it should not be very difficult to figure the

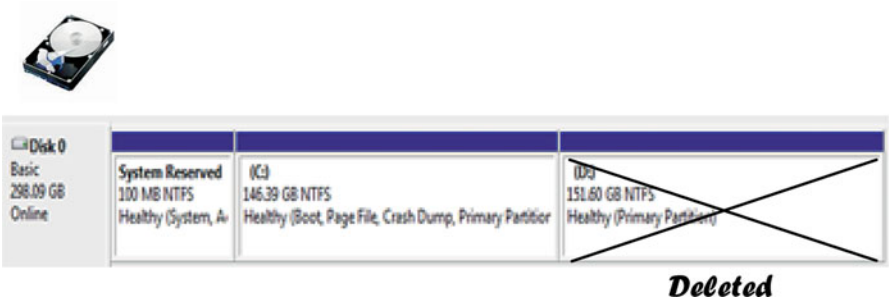


Fig. 4.14 Partition deletion

location of a possible deleted partition by taking a look at the existing partitions. Afterwards, by further analyzing the possible area for the deleted partition, which most likely is a type of file system, we should also determine the partition type. For example, if we figure out the type of file system on the deleted partition is FAT, then the partition type is 0x0b (Fig. 4.14).

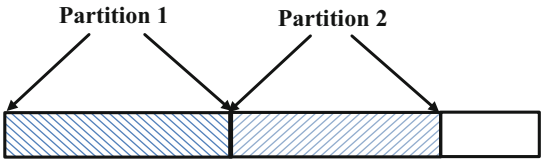
Review Questions

1. What does MBR stand for? _____
2. When conducting digital forensic investigations what is the most common source of digital evidence? _____

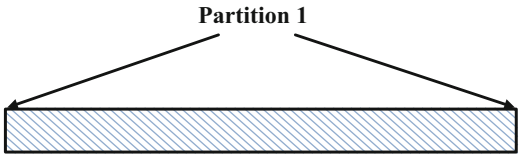
   (a) hard disk
   (b) Internet
   (c) partitions
   (d) unpartitioned disk space

3. Which of the following disk layout is invalid? _____

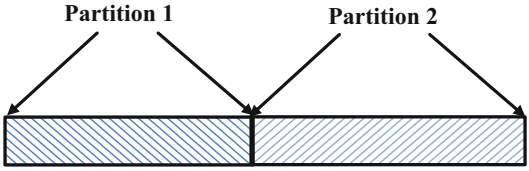
   (a)



   (b)

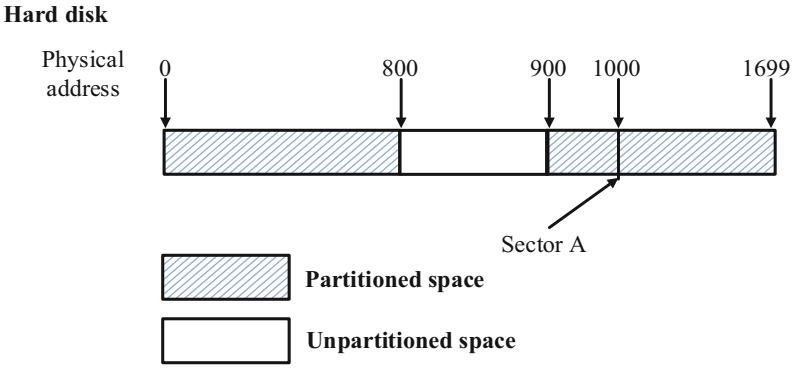


(c)



(d) None of the above

4. In MBR, how many entries are in the partition table? _____
5. What is the logical disk volume address (or ask logical partition address) for the Sector A shown in the figure below? _____ (If the address doesn't exist, N/A should be given.)



6. When is the MBR created? _____

   (a) Low-level Format
   (b) High-level Format
   (c) Partitioning
   (d) OS Install

7. BIOS stands for _____.
8. Consider a disk that reported 8 heads per cylinder and 63 sectors per track. If we had a CHS address of cylinder 6, head 3, and sector 18, the LBA address is _____.
9. After _____, MBR takes control of the booting process of a computer.

   (a) BIOS
   (b) partition boot section
   (c) master boot record
   (d) Operation System

10. How big, in bytes, is MBR? _____
11. In CHS, C stands for _____.
12. According to Fig. 4.6, fill out the following table with details of the second partition.

Partition table for partition entry #1	
Starting CHS address	Cylinder: _____, head: _____, sector: _____
Ending CHS address	Cylinder: _____, head: _____, sector: _____
Starting LBA address	_____
Size of the partition (GB)	_____
Type of partition	_____

4.3 Practice Exercise

The objective of this exercise is to give you a better understanding of how disk partitioning works.

4.3.1 Setting Up the Exercise Environment

In order to begin this exercise, you need to prepare the following disk images:

Download extended DOS partition testing tool, 1-extend-part.zip, where a file called ext-part-test-2.dd inside the zip archive (1-extend-part.zip) is a disk image for the purpose of learning extended partition concepts, and upload it to your Forensics Workstation [4].

To download this tool, go to http://dftt.sourceforge.net/test1/index.html.

4.3.2 Exercises

Part A: Analyze the MBR of the Disk Image "ext-part-test-2.dd", and Fill Out the Following Table with Details of the First Partition (Table 4.7)

Part B: Analyze the First Extended Partition, and Fill Out the Following Table with Details of the Partition (Table 4.8)

Please note: An extended partition acts like a disk, and the first sector is an MBR. You can figure out the layout of the extended partition by analyzing its MBR.

Table 4.7 Details of the first partition in the disk image "ext-part-test-2.dd"

First partition	
Start LBA address	
Number of sectors in partition	
Size of the partition (MB)	
Type of partition	

Table 4.8 Details of the first extended partition in the disk image "ext-part-test-2.dd"

First extended partition	
Start LBA address	
Number of sectors in partition	
Size of the partition (MB)	

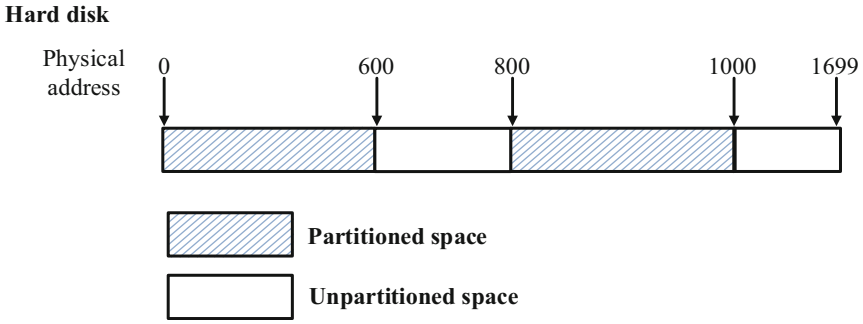


Fig. 4.15 An example layout of hard disk

Part C: Find Out the Layout of the Disk Image "ext-part-test-2.dd"
An example of the disk layout is given below (Fig. 4.15).

Q1. By looking at the disk layout which you have figured out, is there any unpartitioned space? _____ (Yes/No). (Unpartitioned space can be used to hide data.)

Please note that in order to figure out the layout of a disk, you will need to determine the locations of all the partitions.

Part D: Use *dcfldd* to Extract the First Partition Image from the Disk Image Provided
Q2. Use the spaces provided to write down the command(s) you issued.

_____

_____

Part E: Validate Your Answers Using *mmls*
Once you complete the above lab exercises, you will be able to validate your answers by using TSK utility *mmls*. The basic syntax of the command *mmls* is shown below.

mmls –t dos ext-part-test-2.dd

where '-t' option specifies the *mmls* utility has been used to display the layout of a disk that was partitioned using DOS partitioning, and the last argument specifies the filename of the disk image. In the example shown in Fig. 4.8, we known that we can determine the location of each partition and its type.

4.4 Helpful Tips

Here we provide some tips and tricks for obtaining a structured data in a disk image. In a digital investigation, it is common that we need to parse out meaningful information from the raw data found on a disk. In doing so, two important information are required:

- Location of the data, for example, the partition table stored at byte offset of 0x1BE
- Data structure of the data, for example, the partition table data structure depicted in Table 4.3 (Fig. 4.16)

For example, suppose that we have an image of MBR. In order to gather all the partitions information, we need to know the location of the partition table as well as the partition table data structure. As shown in Table 4.2, we know that the partition table starts at byte offset of 446 (or 1BE in Hex). We also know the detailed data structure according to Table 4.3. Thus, we should be able to obtain the partition information such as the starting LBA address, size of partition, and partition type.

Using the image of MBR in Fig. 4.6 as an example, we are able to fetch the partition table in its raw format, shown in Fig. 4.17.

According to Table 4.2, each partition table has four partition entries and one partition entry has a length of 16 bytes. For simplicity, we will look at the first partition entry as our example, where the raw data is "80 20 21 00 07 f. ff ff 00 08 00 00 00 00 1f 01". According to Table 4.3, the first byte indicates if the partition is bootable or not. It is set to 0x80 if bootable, or 0x00 if not. In our example here, the partition is bootable. The next 3 bytes (20 21 00) is the CHS address for starting position of the partition, followed by 1 byte (07) that stands for the partition type. The 3 bytes afterwards (fe ff ff) is the CHS address for ending position of the

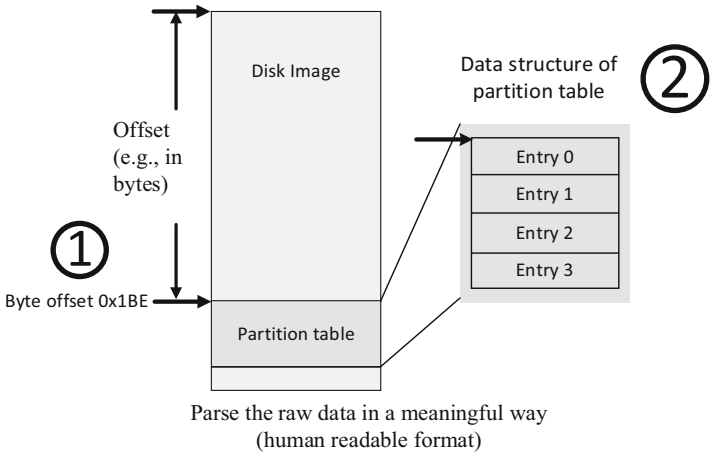


Fig. 4.16 How to obtain a structured data (e.g., partition table) from a disk image

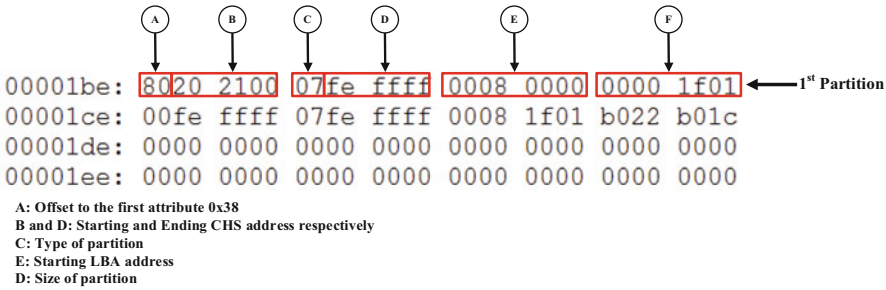


Fig. 4.17 DOS partition table

partition. The 4 bytes afterwards (00 08 00 00) is the LBA address for starting position of the partition, and the last 4 bytes (00 00 1f 01) of the 16-byte-entry counts the number of sectors in that partition.

It is important to emphasize that a byte offset is an address (or distance in bytes) relative to a position, for example, the beginning of MBR or the partition table. For example, in Table 4.3 it states that the starting CHS address of the first partition is between the byte offset of 1 and 3. This is referring to the beginning of the partition entry (or the partition table since we are discussing the first partition), which is at byte offset of 446 in MBR. In other words, the offset of starting CHS address is between 447 and 449 in MBR.

Example Consider the first partition and we first obtain the raw data of the first partition table entry, shown as follows:

Raw data	80	20	21	00	07	fe	ff	ff	00	08	00	00	00	00	1f	01
Byte offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Bytes 1–3 (20 21 00) show starting CHS address value. Then, we have cylinder, which has 10 bits, where bits 9–8 of cylinder are in bits 7–6 in the second byte and bits 7–0 of cylinder in the third byte,

$(00\ 000000000)_2 = 0$.

Also, we have head, which has 8 bits from the entire first byte of the CHS address value,

$0x20 = 32$.

Finally, we can obtain sector, which has 6 bits. These bits are from bits 5–0 in the second byte,

$(100001)_2 = 33$.

Thus, starting CHS address is (C = 0, H = 32, S = 33).

Similarly, we can obtain that ending CHS address is (C = 1023, H = 254, S = 63).

Next, Bytes 8–11 (00 08 00 00) show LBA address of the first sector in the partition. Note that little endian conversion applies here since we are currently using a Little-Endian machine. This applies to any multibyte values later.

0x00000800 = 2048

The size of the partition in sectors is determined by the last four bytes of the partition entry (00 00 1f 01), which is 0x011f0000 = 18808832 sectors.

Finally, we can obtain the size of the partition in GB,

18808832 X 512 / 1024 / 1024 = 8.96875 GB.

References

1. Disk partitioning. [Online] Available at: http://en.wikipedia.org/wiki/Disk_partitioning
2. Master boot record. [Online] Available at: http://en.wikipedia.org/wiki/Master_boot_record
3. Brian Carrier. File System Forensic Analysis. Addison-Wesley Professional; 1 edition (Mar 27 2005) ISBN-10: 0321268172
4. Digital Forensics Tool Testing Images. [Online] Available at: http://dftt.sourceforge.net/
5. Partitions and Volumes. [Online] Available at: http://www.yale.edu/pclt/BOOT/PARTITIO.HTM
6. GUID Partition Table. https://en.wikipedia.org/wiki/GUID_Partition_Table
7. Brian D. Carrier. Volume analysis of disk spanning logical volumes. Digital Investigation, vol. 2, no. 2, June 2005, pp. 78-88.
8. Anthony Sammes, Brian Jenkinson. Forensic Computing: A Practitioner's Guide. Springer-Verlag London, 2007. (ISBN: 9781846283970)