5 PC-based Partitions

The last chapter provided an overview of volume analysis and why it's important. Now we're going to leave the abstract discussion of volumes and dive into the details of the partition systems used in personal computers. In this chapter, we will look at DOS partitions, Apple partitions, and removable media. For each system, we review how it works and look at its data structure. If you are not interested in the data structure details, you can skip those sections. This chapter also covers special considerations that should be made when analyzing these systems. The next chapter will examine server-based partitioning systems.

DOS Partitions

The most commonly encountered partition system is the DOS-style partition. DOS partitions have been used with Intel IA32 hardware (i.e., i386 / x86) for many years, yet there is no official specification. There are many Microsoft and non-Microsoft documents that discuss the partitions, but there is no standard reference.

In addition to there being no standard reference, there is also no standard name. Microsoft now calls disks using this type of partition system *Master Boot Record* (MBR) disks. This is in comparison to a *GUID Partition Table* (GPT) disk that is used with the *Extensible Firmware Interface* (EFI) and the 64-bit Intel Itanium-based systems (IA64), which are discussed in the next chapter[Microsoft 2004a]. Starting with Windows 2000, Microsoft also differentiates between basic and dynamic disks. A *basic disk* refers to

Carrier_05.qxd 2/22/05 3:07 PM Page 82

either an MBR or a GPT disk, and the partitions in the disk are independent and stand-alone. Dynamic disks, which are discussed in Chapter 7, “Multiple Disk Volumes,” also can be either MBR or GPT disks, and the partitions can be combined and merged to form a single, large partition. Basic disks have traditionally been associated with DOS partitions, probably because GPT disks are not yet as common. Therefore, using the current terminology, this chapter covers basic MBR disks. However, we will use the simple term DOS partitions for this book.

DOS partitions are used with Microsoft DOS, Microsoft Windows, Linux, and IA32-based FreeBSD and OpenBSD systems. DOS partitions are the most common but also the most complex partitioning system. They were originally designed in the 1980s for small systems and have been improved (i.e., hacked) to handle large modern systems. In fact, there are two different partitioning methods that are used in this system. This section will give an overview of the partitioning system, show the data structures in the system, show what tools can list the layout, and discuss investigation considerations.

GENERAL OVERVIEW

In this section, we will examine the DOS partition concepts and boot code location. The data structures are discussed in the following section.

Basic MBR Concepts

A disk that is organized using DOS partitions has an MBR in the first 512-byte sector. The MBR contains boot code, a partition table, and a signature value. The boot code contains the instructions that tell the computer how to process the partition table and locate the operating system. The partition table has four entries, each of which can describe a DOS partition. Each entry has the following fields:

- Starting CHS address
- Ending CHS address
- Starting LBA address
- Number of sectors in partition
- Type of partition
- Flags

Each table entry describes the layout of a partition in both CHS and LBA addresses. Recall that the CHS addresses only work for disks less than 8 GB in size, but the LBA addresses allow disks to be *terabytes* (TB) in size.

The type field in the partition identifies what type of data should exist in the partition. Common examples include FAT, NTFS, and FreeBSD. The next section has a more comprehensive list of partition types. The type value is used differently by different OSes. Linux, for example, does not care about it. You can put a FAT file system inside of a partition that has a type of NTFS, and it will mount it as FAT. Microsoft Windows, on the other hand, relies on it. Windows will not try to mount a file system in a partition if it does not support the partition type. Therefore, if a disk has a FAT file system inside a partition with a Linux file system type, the user will not see the FAT file system from within Windows. This behavior can be used to hide partitions from Windows. For example, some tools will add a bit to a partition type that Windows supports so that it will not be shown when Windows boots again.

Each entry also contains a flag field that identifies which partition is the "bootable" one. This is used to identify where the operating system is located when the computer is booting. Using the four entries in the MBR, we can describe a simple disk layout with up to four partitions. Figure 5.1 shows such a simple disk with two partitions and the MBR in the first sector.

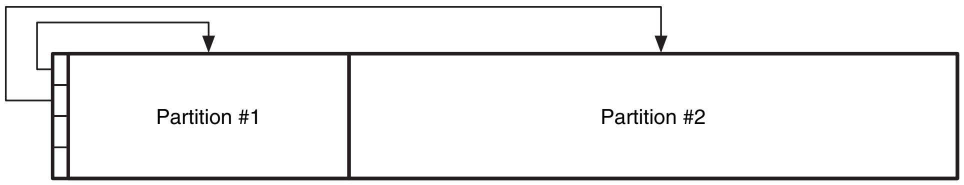


Figure 5.1 A basic DOS disk with two partitions and the MBR.

Extended Partition Concepts

The MBR is a simple method of describing up to four partitions. However, many systems require more partitions than that. For example, consider a 12GB disk that the user wants to divide into six 2GB partitions because he is using multiple operating systems. We cannot describe the six partitions by using the four partition table entries.

The solution to this design problem is what makes DOS partitions so complex. The basic theory behind the solution is to use one, two, or three of the entries in the MBR for normal partitions and then create an "extended partition" that will fill up the remainder of the disk. Before we move on, some definitions may be helpful. A *primary file system partition* is a partition whose entry is in the MBR and the partition contains a file system or other structured data. A *primary extended partition* is a partition whose entry is in the

MBR, and the partition contains additional partitions. We can see this in Figure 5.2, which has three primary file system partitions and one primary extended partition.

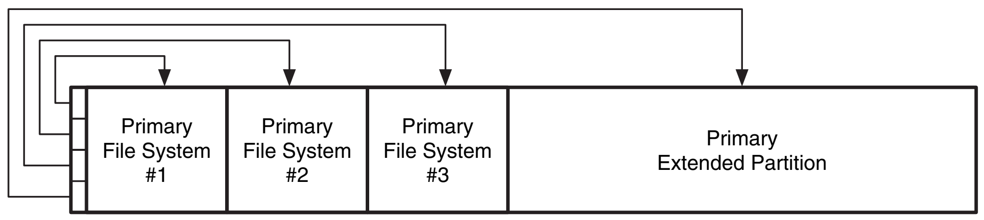


Figure 5.2 A DOS disk with three primary file system partitions and one primary secondary partition.

To consider what goes inside a primary extended partition, you should basically forget about everything we have discussed so far. In the MBR, we saw a central partition table that described several partitions. Here we see a linked list of partitions. The basic theory is that we are going to precede every file system partition with data that describe how big the file system partition is and where we can find the next partition. All these partitions should be located inside of the primary extended partition, which is why it must be as large as possible.

A *secondary file system partition*, also called a logical partition in Windows, is located inside the primary extended partition bounds and contains a file system or other structured data. Secondary file system partitions are equivalent to the partitions that are described in the MBR except that they are in an extended partition. A *secondary extended partition* is a partition that contains a partition table and a secondary file system partition. The secondary extended partitions wrap around the secondary file system partitions and describe where the secondary file system partition is located and where the next secondary extended partition is located.

Figure 5.3 shows an example of how secondary partitions work. Secondary Extended #1 contains a partition table that points to Secondary File System #1 and Secondary Extended #2. Secondary Extended #2 contains a partition table that points to Secondary File System #2. It also could point to another secondary extended partition, and this process could repeat until we are out of disk space.

Carrier_05.qxd 2/22/05 3:07 PM Page 85

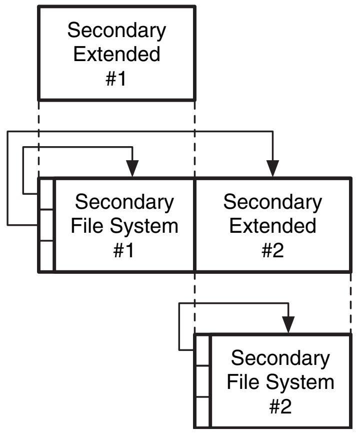


Figure 5.3 The basic theory and layout behind the secondary extended and file system partitions.

Putting the Concepts Together

Now let's put these two partitioning methods together. If we need one to four partitions, we can create them using only the MBR, and we do not need to worry about extended partitions. If we want more than four partitions, we must create up to three primary file system partitions in the MBR and then allocate the rest of the disk to a primary extended partition.

Inside the primary extended partition, we use the linked-list partitioning method. We can optimize the linked-list design that I described in the last section by not making the initial secondary extended partition. Instead, put a partition table at the beginning of the primary extended partition. It can describe one secondary file system and one secondary extended partition.

Consider an example. We have a 12GB disk and we want to break it up into six 2GB partitions. We create the first three 2GB partitions using the first three entries in the MBR, and the remaining 6GB is allocated to a primary extended partition, which spans from 6GB to 12GB.

We need to allocate three more partitions using the linked-list method. We use the partition table in the first sector of the primary extended partition, make a secondary file system partition that spans from 6GB to 8GB, and make a secondary extended partition that spans from 8GB to 10GB. A partition table is inside the secondary extended partition, and it has entries for a secondary file system partition that spans from 8GB to 10GB and an entry for another secondary extended partition that spans from 10GB to 12GB.

A partition table is inside the last secondary extended partition, and it has an entry for the final file system partition, which spans from 10GB to 12GB. We see this in Figure 5.4.

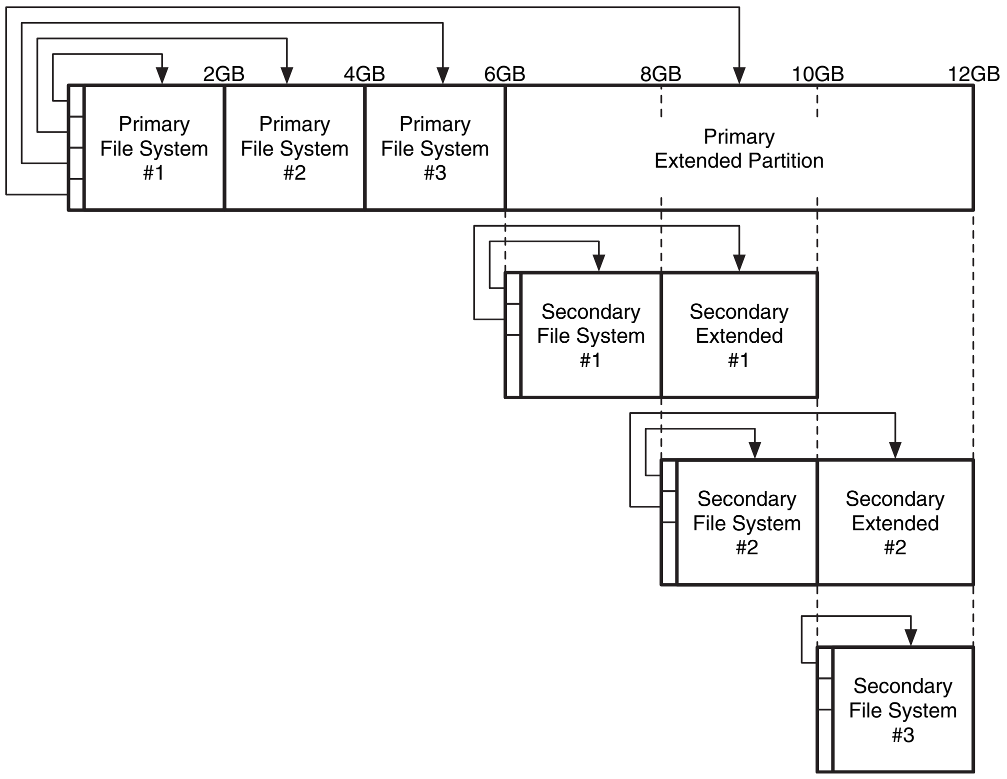


Figure 5.4 The layout required for a disk with six file system partitions.

As I have described it and as most documents claim, an extended partition table should have, at most, one entry for a secondary file system partition and one entry for a secondary extended partition. In practice, most operating systems will not generate an error if more than two entries are being used. In fact, in July 2003, I released a 160 MB disk image [Carrier 2003] with six 25 MB DOS partitions to the CFTT Yahoo! Groups list (`http://groups.yahoo.com/group/cftt/`). The image had a primary extended partition table with two secondary file system partition entries and one secondary extended partition entry. Some forensic tools properly handled the third partition entry, while others ignored it or claimed that the 25 MB partition was a 1 TB partition. This example shows how something as common as DOS partitions can cause problems with analysis tools.

Extended partitions have special types that are used in their partition table entries. To make this confusing partition scheme even more confusing, there is more than one type of extended partition, and they do not differentiate between primary and secondary extended partitions. The common types of extended partitions are "DOS Extended," "Windows 95 Extended," and "Linux Extended."

Boot Code

The boot code in a DOS disk exists in the first 446 bytes of the first 512-byte sector, which is the MBR. The end of the sector contains the partition table. The standard Microsoft boot code processes the partition table in the MBR and identifies which partition has the bootable flag set. When it finds such a partition, it looks in the first sector of the partition and executes the code found there. The code in the start of the partition will be operating system-specific. Boot sector viruses insert themselves into the first 446 bytes of the MBR so that they are executed every time the computer is booted.

It is becoming much more common to have multiple operating systems on a computer. There are two ways to handle this. Windows handles this by having code in the bootable partition that allows a user to select which OS to load. In other words, the boot code in the MBR executes first and loads the Windows bootable code. The Windows bootable code allows a user to choose a different partition from which to boot.

The other method is to change the code in the MBR. The new MBR code presents the user with a list of options, and the user chooses which partition to boot from. This typically requires more code and uses some of the unused sectors that exist before the first partition starts.

Summary

The DOS partition system is complex because each partition table has only four entries in it. Other partition systems discussed later in this chapter and the next have larger partition tables and are, therefore, less complex. The following high-level steps are necessary to list the layout information of a disk with DOS partitions:

1. The Master Boot Record is read from the first sector of the disk, and the four partition table entries are identified and processed.
2. When an entry for an extended partition is processed, the first sector of the extended partition is read and its partition table entries are processed in the same manner as the MBR.

3. When an entry for a non-extended partition is processed, its starting sector and size are displayed. The ending sector address can be determined by adding the starting sector address and the size together and subtracting one.

DATA STRUCTURES

The previous section reviewed the DOS partition system. This section provides a detailed discussion of the structures that make the system work. If you are not interested in data structures, you can skip this; however, there is an interesting example of extended partitions. This section is organized into three subsections describing the MBR, extended partitions, and show tool output from an example image.

MBR Data Structure

DOS Partition tables exist in the MBR and in the first sector of each extended partition. Conveniently, they all use the same 512-byte structure. The first 446 bytes are reserved for assembly boot code. Code needs to exist in the MBR because it is used when the computer is started, but the extended partitions do not need it and could contain hidden data. The MBR layout in tabular form can be found in Table 5.1.

Table 5.1 Data structures for the DOS partition table.

Byte Range	Description	Essential
0–445	Boot Code	No
446–461	Partition Table Entry #1 (see Table 5.2)	Yes
462–477	Partition Table Entry #2 (see Table 5.2)	Yes
478–493	Partition Table Entry #3 (see Table 5.2)	Yes
494–509	Partition Table Entry #4 (see Table 5.2)	Yes
510–511	Signature value (0xAA55)	No

The partition table has four 16-byte entries. The entries' structures are given in Table 5.2. Note that the CHS addresses are essential for older systems that rely on them, but are not essential on newer systems.

Table 5.2 Data structure for DOS partition entries.

Byte Range	Description	Essential
0–0	Bootable Flag	No
1–3	Starting CHS Address	Yes
4–4	Partition Type (see Table 5.3)	No
5–7	Ending CHS Address	Yes
8–11	Starting LBA Address	Yes
12–15	Size in Sectors	Yes

The bootable flag is not always necessary. The standard boot code for a system with only one OS looks for an entry whose flag is set to 0x80. For example, if a system has Microsoft Windows on it and the disk is partitioned into two partitions, the partition with the operating system on it (`C:\windows`, for example) will have the bootable flag set. On the other hand, if the boot code prompts the user to choose which partition to boot from, the bootable flag is not necessary. Although, some boot programs will set the bootable flag after the user chooses to boot that partition.

The starting and ending CHS addresses have an 8-bit head value, a 6-bit sector value, and a 10-bit cylinder value. In theory, either the CHS addresses or the LBA addresses need to be set for each partition, but not both. It is up to the OS and the code that is used to boot the system to determine which values need to be set. For example, Windows 98 and ME use the CHS addresses for partitions in the first 7.8GB of the disk, but Windows 2000 and beyond always ignore the CHS addresses [Microsoft 2003]. Some partitioning tools set both when possible for backward compatibility. The usage of these fields is application-dependent.

The partition type field identifies the file system type that should be in the partition. A list of common partition types is given in Table 5.3. A more detailed list of partition types can be found in *Partition types* [Brouwer 2004].

Table 5.3 Some of the type values for DOS partitions.

Type	Description
0x00	Empty
0x01	FAT12, CHS
0x04	FAT16, 16–32 MB, CHS

continues

Table 5.3 Some of the type values for DOS partitions (continued).

Type	Description
0x05	Microsoft Extended, CHS
0x06	FAT16, 32 MB–2GB, CHS
0x07	NTFS
0x0b	FAT32, CHS
0x0c	FAT32, LBA
0x0e	FAT16, 32 MB–2GB, LBA
0x0f	Microsoft Extended, LBA
0x11	Hidden FAT12, CHS
0x14	Hidden FAT16, 16–32 MB, CHS
0x16	Hidden FAT16, 32 MB–2GB, CHS
0x1b	Hidden FAT32, CHS
0x1c	Hidden FAT32, LBA
0x1e	Hidden FAT16, 32 MB–2GB, LBA
0x42	Microsoft MBR. Dynamic Disk
0x82	Solaris x86
0x82	Linux Swap
0x83	Linux
0x84	Hibernation
0x85	Linux Extended
0x86	NTFS Volume Set
0x87	NTFS Volume Set
0xa0	Hibernation
0xa1	Hibernation
0xa5	FreeBSD
0xa6	OpenBSD

Type	Description
0xa8	Mac OSX
0xa9	NetBSD
0xab	Mac OSX Boot
0xb7	BSDI
0xb8	BSDI swap
0xee	EFI GPT Disk
0xef	EFI System Partition
0xfb	Vmware File System
0xfc	Vmware swap

Notice how many partition types exist for Microsoft file systems in the 0x01 to 0x0f range. The reason is that Microsoft operating systems use the partition type to determine how to read and write data from the partition. Recall from Chapter 2, "Computer Foundations," that Windows can use either INT 13h or the extended INT 13h BIOS routines. The extended INT 13h routines are needed for accessing disks larger than 8.1GB and use LBA addressing instead of CHS. Therefore, the FAT16 0x04 and 0x0E types are the same except that the OS should use the extended routines for the latter type. Similarly, 0x0B and 0x0C types are the normal and extended versions of FAT32 and 0x05, and 0x0F types are the normal and extended for extended partitions [Microsoft 2004b]. The "hidden" versions of these partition types have a 1 instead of a 0 in the upper nibble, and various tools create them.

To illustrate the MBR and the partition tables, we will extract the sectors from an actual system and parse the structures by hand. The system is a dual boot Microsoft Windows and Linux system, and it has eight file system partitions.

The first example is from the first sector of the disk. This output is from the `xxd` tool in Linux, but similar data can be found using a hex editor in Windows or UNIX. The following command was used in Linux:

```
# dd if=disk3.dd bs=512 skip=0 count=1 | xxd
```

The left column is the byte offset in decimal, the middle eight columns are the data in hexadecimal format, and the final column is the data translated into ASCII. The data are from an IA32-based system, which is little-endian and stores numbers with the least significant byte at the lowest address. Therefore, the order of the bytes in the middle columns may need to be reversed. The MBR of the disk is as follows:

```
# dd if=disk3.dd bs=512 skip=0 count=1 | xxd
0000000: eb48 9010 8ed0 bc00 b0b8 0000 8ed8 8ec0  .H.............
[REMOVED]
0000384: 0048 6172 6420 4469 736b 0052 6561 6400  .Hard Disk.Read.
0000400: 2045 7272 6f72 00bb 0100 b40e cd10 ac3c   Error.........<
0000416: 0075 f4c3 0000 0000 0000 0000 0000 0000  .u..............
0000432: 0000 0000 0000 0000 0000 0000 0000 0001  ................
0000448: 0100 07fe 3f7f 3f00 0000 4160 1f00 8000  ....?.?...A`....
0000464: 0180 83fe 3f8c 8060 1f00 cd2f 0300 0000  ....?..`.../....
0000480: 018d 83fe 3fcc 4d90 2200 40b0 0f00 0000  ....?.M.".@.....
0000496: 01cd 05fe ffff 8d40 3200 79eb 9604 55aa  .......@2.y...U.
```

The first 446 bytes contain boot code. The 0xAA55 signature value can be seen in the last two bytes of the sector (although they are reversed in the output because of the endian ordering). The partition table is in bold and starts with the 0x0001 at offset 446. Each line in the output has 16 bytes, and each table entry is 16 bytes. Therefore, the second entry begins one line below the first entry with 0x8000. Using the structure previously outlined, the four partition table entries are shown in Table 5.4. The values are shown in hexadecimal format with the decimal value in parenthesis of important values.

Table 5.4 The contents of the primary partition table in the example disk image.

#	Flag	Type	Starting Sector	Size
1	0x00	0x07	0x0000003f (63)	0x001f6041 (2,056,257)
2	0x80	0x83	0x001f6080 (2,056,320)	0x00032fcd (208,845)
3	0x00	0x83	0x0022904d (2,265,165)	0x000fb040 (1,028,160)
4	0x00	0x05	0x0032408d (3,293,325)	0x0496eb79 (76,999,545)

Using Table 5.4 and the partition type field in Table 5.3, we can guess what type of data are in each partition. The first partition should be for an NTFS file system (type 0x07), the second and third partitions should be for Linux file systems (0x83), and the fourth partition is an extended partition (0x05). The second entry is set to be bootable. The extended partition should have been expected because it was previously mentioned that there would be a total of eight partitions. The disk layout from this partition table is shown in Figure 5.5.

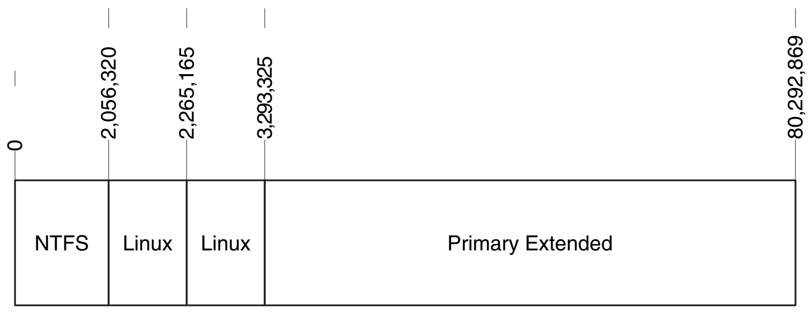


Figure 5.5 Disk layout after processing the first partition table in example (not to scale).

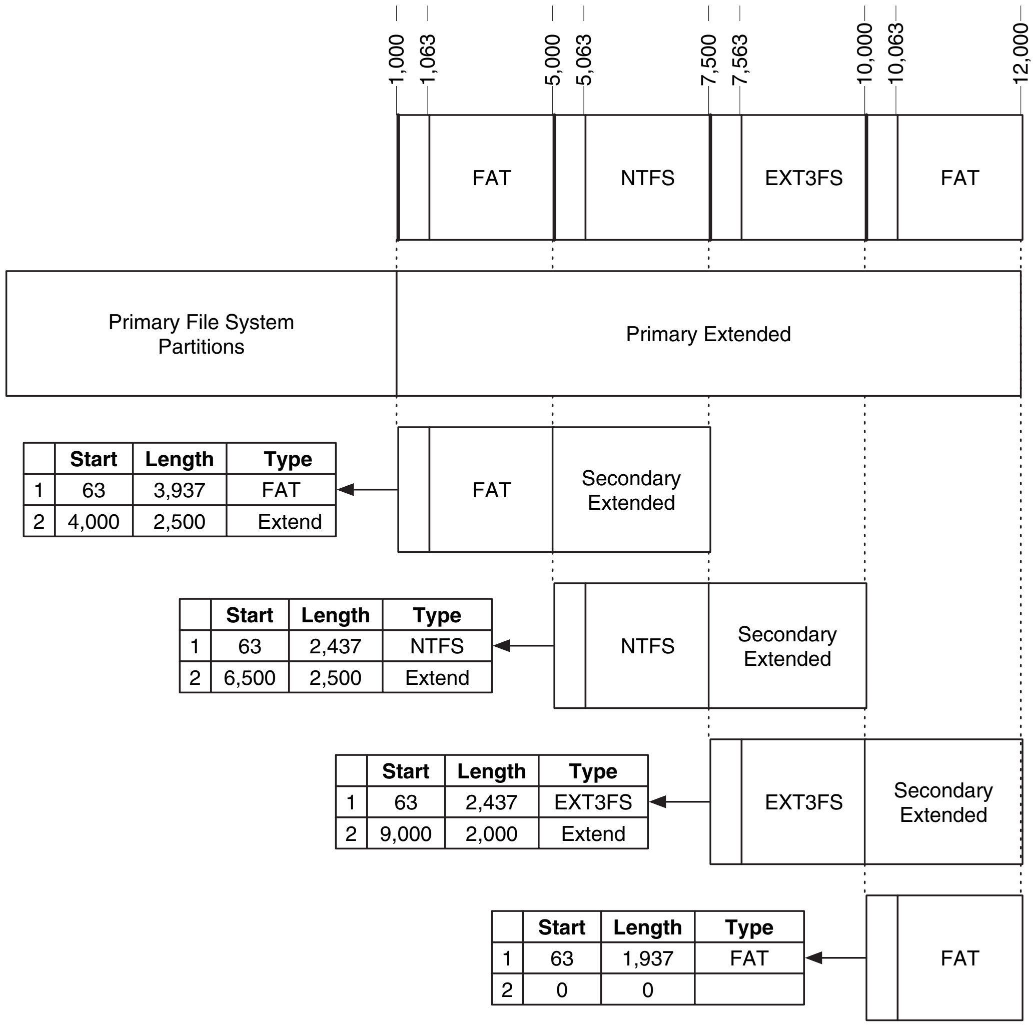
Extended Partition Data Structures

Recall that the extended partitions use the same structure in the first sector as the MBR does, but they use it to make a linked list. The partition table entries are slightly different, though, because the starting sector addresses are relative to other places on the disk besides the beginning of the disk. Furthermore, the starting sector of a secondary file system partition is relative to a different place than the starting sector of a secondary extended partition.

The starting address for a secondary file system entry is relative to the current partition table. This is intuitive because the secondary extended partitions work as wrappers around the file system partitions; therefore, they have the starting address relative to themselves. On the other hand, the starting address for a secondary extended partition entry is relative to the primary extended partition.

Let's step through the example shown in Figure 5.6. It has a primary extended partition that starts in sector 1,000 with a length of 11,000 sectors. Its partition table has two entries. The first is for a FAT file system with a starting sector of 63, which is added to the sector of the current partition table to get 1,063. The second entry is for an extended partition and its starting sector is 4,000. That is added to the start of the primary extended partition, which is sector 1,000, and we get sector 5,000.

Now let's jump ahead to that secondary extended partition (in sector 5,000). The first partition table entry is for an NTFS file system, and its starting value is 63, which is added to the address of the current partition table and to get sector 5,063. The second entry is for an extended partition, and its starting value is 6,500, which is added to the sector of the primary extended partition and to get sector 7,500.



	Start	Length	Type
1	63	3,937	FAT
2	4,000	2,500	Extend

	Start	Length	Type
1	63	2,437	NTFS
2	6,500	2,500	Extend

	Start	Length	Type
1	63	2,437	EXT3FS
2	9,000	2,000	Extend

	Start	Length	Type
1	63	1,937	FAT
2	0	0	

Figure 5.6 Disk with three secondary extended partitions. Note that the starting location of the secondary extended partitions is relative to the start of the primary extended partition, sector 1000.

We'll do one more round to make sure it is clear. The next extended partition starts in sector 7,500. The first entry is for an EXT3FS file system with a starting value of 63, which is added to 7,500 to get sector 7,563. The second entry is for a secondary extended partition, and its starting value is 9,000, which is added to 1,000 to get sector 10,000.

Return to the actual system that we parsed by hand. The following are the contents of the first sector of the primary extended partition, which is located in sector 3,293,325:

```
# dd if=disk3.dd bs=512 skip=3293325 count=1 | xxd
[REMOVED]
0000432: 0000 0000 0000 0000 0000 0000 0000 0001  ................
0000448: 01cd 83fe 7fcb 3f00 0000 0082 3e00 0000  ......?.....>...
0000464: 41cc 05fe bf0b 3f82 3e00 40b0 0f00 0000  A.....?.>.@.....
0000480: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000496: 0000 0000 0000 0000 0000 0000 0000 55aa  ..............U.
```

The four partition table entries are highlighted, and we see that the final two entries are empty. The first two partition table entries are parsed into the contents of Table 5.5 (the partition numbering is continued from Table 5.4):

Table 5.5 The contents of the primary extended partition table in the example disk image.

#	Flag	Type	Starting Sector	Size
5	0x00	0x83	0x0000003f (63)	0x003e8200 (4,096,572)
6	0x00	0x05	0x003e823f (4,096,575)	0x000fb040 (1,028,160)

Entry #5 has a type for a Linux file system (0x83), so it is a secondary file system partition, and its starting sector is relative to the start of the current extended partition (sector 3,293,325).

```
3,293,325 + 63 = 3,293,388
```

Entry #6 has a type for a DOS Extended partition, so its starting sector is relative to the start of the primary extended partition, which is the current partition.

```
3,293,325 + 4,096,575  = 7,389,900
```

The disk layout, as we know it, can be found in Figure 5.7. Before we continue, note the sizes of the two partitions. In the MBR, the primary extended partition had a size of 76,999,545 sectors. In this table, the size of the next secondary extended partition is only 1,028,160 sectors. Recall that the primary extended partition has a size of all the secondary file systems and secondary extended partitions, but the secondary extended partitions have a size that is equal to the size of only the next secondary file system partition plus the size needed for a partition table.

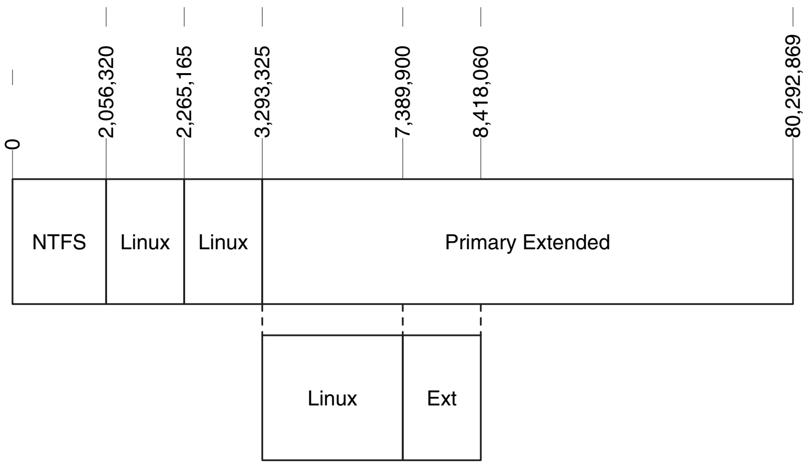


Figure 5.7 Disk layout after processing the second partition table (not to scale).

We can continue the example by examining the next secondary extended partition, which is located in sector 7,389,900. Its contents are shown in Table 5.6.

Table 5.6 The contents of the first secondary extended partition table in the example disk image.

#	Flag	Type	Starting Sector	Size
7	0x00	0x82	0x0000003f (63)	0x000fb001 (1,028,097)
8	0x00	0x05	0x004e327f (5,124,735)	0x000fb040 (1,028,160)

Entry #7 is for a Linux swap partition, so it is a secondary file system, and its starting sector address is relative to the current extended partition, which is sector 7,389,900.

```
7,389,900 + 63 = 7,389,963
```

Entry #8 is for a DOS Extended file system, so its starting sector address is relative to the primary extended partition, which is sector 3,293,325.

```
3,293,325 + 5,124,735 = 8,418,060
```

The disk layout with the information from this partition table can be found in Figure 5.8. The full contents of the example partition table are given in the next section when we look at tools that print the partition table contents.

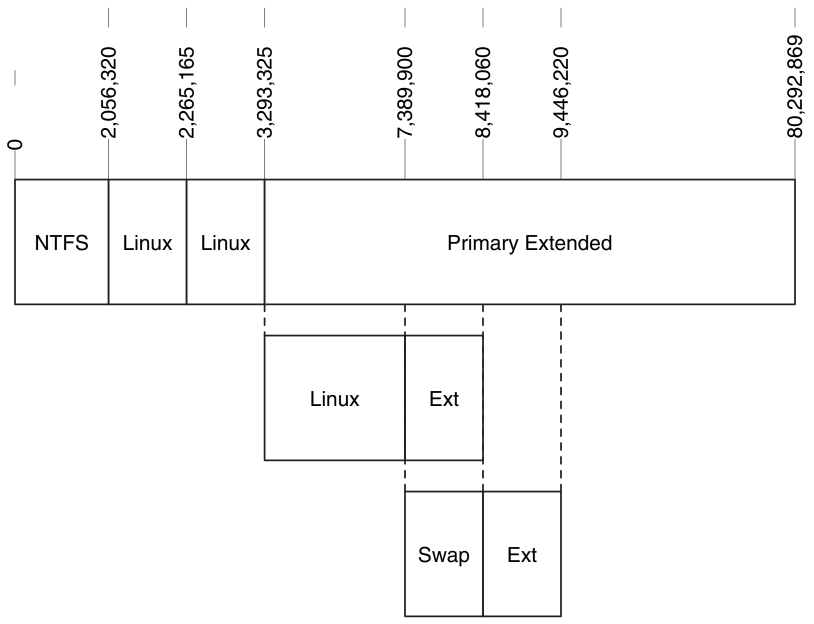


Figure 5.8 Disk layout after processing the third partition table (not to scale).

Example Image Tool Output

Now that the internal structure of the partition system is known, we will show how some of the analysis tools process them. For those who actually enjoy parsing the structure by hand and never use a tool, you can skip this section. Two Linux tools will be shown here. Other Windows tools, such as full forensic analysis tools and hex editors, also perform this function.

The `fdisk` command comes with Linux and is different from the tool with the same name that comes with Windows. `fdisk` can be run on a Linux device or a disk image file generated by `dd`. The `-l` flag forces it to list the partitions instead of going into interactive mode where the partitions could also be edited. The `-u` flag forces the output to be in sectors instead of cylinders. The output of the DOS Partitioned disk that we parsed by hand is as follows:

```
# fdisk -lu disk3.dd
Disk disk3.dd: 255 heads, 63 sectors, 0 cylinders
Units = sectors of 1 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
disk3.dd1            63   2056319   1028128+   7  HPFS/NTFS
disk3.dd2   *   2056320   2265164    104422+  83  Linux
disk3.dd3       2265165   3293324    514080   83  Linux
disk3.dd4       3293325  80292869  38499772+   5  Extended
disk3.dd5       3293388   7389899   2048256   83  Linux
disk3.dd6       7389963   8418059    514048+  82  Linux swap
disk3.dd7       8418123   9446219    514048+  83  Linux
disk3.dd8       9446283  17639369   4096543+   7  HPFS/NTFS
disk3.dd9      17639433  48371714  15366141   83  Linux
```

We can observe several things from this output. The output lists only the primary extended partition (disk3.dd4). The secondary extended partition in which the Linux swap partition is located is not displayed. This is acceptable for most circumstances because only the primary and secondary file system partitions are needed for an investigation, but it should be noted that you are not seeing all partition table entries.

The `mmls` tool in The Sleuth Kit provides slightly different information. Sectors that are unused by a partition are marked as such, the location of the partition tables is marked, and the extended partition locations are noted. Using the same disk as we used for the first `fdisk` example, the following is seen:

```
# mmls -t dos disk3.dd
Units are in 512-byte sectors
     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Table #0
01:  -----   0000000001   0000000062   0000000062   Unallocated
02:  00:00   0000000063   0002056319   0002056257   NTFS (0x07)
03:  00:01   0002056320   0002265164   0000208845   Linux (0x83)
04:  00:02   0002265165   0003293324   0001028160   Linux (0x83)
05:  00:03   0003293325   0080292869   0076999545   DOS Extended (0x05)
06:  -----   0003293325   0003293325   0000000001   Table #1
```

```
07:  -----   0003293326   0003293387   0000000062   Unallocated
08:  01:00   0003293388   0007389899   0004096512   Linux (0x83)
09:  01:01   0007389900   0008418059   0001028160   DOS Extended (0x05)
10:  -----   0007389900   0007389900   0000000001   Table #2
11:  -----   0007389901   0007389962   0000000062   Unallocated
12:  02:00   0007389963   0008418059   0001028097   Linux Swap (0x82)
13:  02:01   0008418060   0009446219   0001028160   DOS Extended (0x05)
14:  -----   0008418060   0008418060   0000000001   Table #3
15:  -----   0008418061   0008418122   0000000062   Unallocated
16:  03:00   0008418123   0009446219   0001028097   Linux (0x83)
17:  03:01   0009446220   0017639369   0008193150   DOS Extended (0x05)
18:  -----   0009446220   0009446220   0000000001   Table #4
19:  -----   0009446221   0009446282   0000000062   Unallocated
20:  04:00   0009446283   0017639369   0008193087   NTFS (0x07)
21:  04:01   0017639370   0048371714   0030732345   DOS Extended (0x05)
22:  -----   0017639370   0017639370   0000000001   Table #5
23:  -----   0017639371   0017639432   0000000062   Unallocated
24:  05:00   0017639433   0048371714   0030732282   Linux (0x83)
```

The 'Unallocated' entries are for the space in between partitions and for the space between the end of the partition table and the beginning of the first partition. The output of `mmls` gives both the ending address and the size, so it can be easily used to extract the partitions with `dd`.

The output of `mmls` is sorted by the starting sector of the partition, so the first column is only a counter for each entry and has no correlation to the partition table entry. The second column shows what partition table the partition was found in and which entry in the table. The first number shows which table, 0 being the primary table and 1 being the primary extended table, and the second number shows which entry in the table. The sorted output helps to identify sectors that are not partitioned. For example, consider this image:

```
# mmls -t dos disk1.dd
Units are in 512-byte sectors
     Slot   Start       End         Length      Description
00:  -----  0000000000  0000000000  0000000001  Table #0
01:  -----  0000000001  0000000062  0000000062  Unallocated
02:  00:00  0000000063  0001028159  0001028097  Win95 FAT32 (0x0B)
03:  -----  0001028160  0002570399  0001542240  Unallocated
04:  00:03  0002570400  0004209029  0001638630  OpenBSD (0xA6)
05:  00:01  0004209030  0006265349  0002056320  NTFS (0x07)
```

In this output, we see that the NTFS partition is in a slot that is before the OpenBSD partition, but the NTFS partition starts after the OpenBSD partition. We can also see that there is no entry '00:02,' and the 1,542,240 sectors in between the FAT and OpenBSD partitions are also marked as unallocated.

ANALYSIS CONSIDERATIONS

This section includes a few characteristics that can be taken into consideration when analyzing a DOS-based disk. The partition table and boot code require only one sector, yet 63 are typically allocated for both the MBR and extended partitions because the partitions start on a cylinder boundary. Therefore, sector 0 of the extended partition or MBR is used for code and the partition table, but sectors 1-62 may not be used. The unused area can be used by additional boot code, but it also may contain data from a previous installation, zeros, or hidden data. Windows XP does not wipe the data in the unused sectors when it partitions a disk.

As most partitions start at sector 63 (which you can use to your advantage if you are desperate to recover the contents of the first partition), the partition table is missing and the tools discussed in Chapter 4, "Volume Analysis," do not work. Try extracting data from sector 63 onward. This method includes other partitions in the image; however, you may be able to identify the actual size of the partition from file system data. The partition can be extracted with dd as follows:

```
# dd if=disk.dd bs=512 skip=63 of=part.dd
```

In theory, extended partitions should have only two entries: one secondary file system partition and another secondary extended partition. Most partitioning tools follow this theory, but it is possible to create a third entry by hand. Microsoft Windows XP and Red Hat 8.0 showed the "extra" partition when there were more than two in an extended partition, although neither OS would allow you to create such a configuration. Test your analysis tools to ensure that they are showing all of the partitions when this "invalid" configuration exists.

The value in the partition type field of the partition table is not always enforced. Windows uses this field to identify which partitions it should try to mount, but users are given access to all partitions in operating systems, such as Linux. Therefore, a user could put a FAT file system in a partition whose type is for laptop hibernation. They would not be able to mount it in Windows, but would in Linux.

Some versions of Windows only create one primary partition in the MBR and then rely on extended partitions for the remaining partitions. In other words, they do not create three primary partitions before creating an extended partition.

When parts of a partition table have become corrupt, it may be necessary to search for the extended partition tables. To find the extended partitions, a search for 0xAA55 in the last two bytes of a sector could be conducted. Note that this signature value exists at the same location in the first sector of a NTFS and FAT file system, and the remainder of the sector must be examined to determine if it is a partition table or a file system boot sector. If a sector is found to be a boot sector of a file system, a partition table may exist 63 sectors prior to it.

SUMMARY

DOS-based partitions are the most common for current computer investigations. Unfortunately, they are also the most complex to understand because they were not originally designed for the size of modern systems. Fortunately, tools exist to easily list the layout of the disk and extract the used and unused space. Many UNIX systems that run on IA32-compatible platforms use DOS partitions in addition to their own partition systems. Therefore, every investigator needs a solid understanding of DOS partitions.

APPLE PARTITIONS

Systems running the Apple Macintosh operating system are not as common as those running Microsoft Windows, but they have been increasing in popularity with the introduction of Mac OS X, a UNIX-based operating system. The partitions that we will describe here can be found in the latest Apple laptops and desktops running OS X, older systems that are running Macintosh 9, and even the portable iPod devices that play MP3 audio. The partition map also can be used in the disk image files that a Macintosh system uses to transmit files. The disk image file is similar to a zip file in Windows or a tar file in Unix. The files in the disk image are stored in a file system, and the file system may be in a partition.

The design of the partition system in an Apple system is a nice balance between the complexity of DOS-based partitions and the limited number of partitions that we will see in the BSD disk labels. The Apple partition can describe any number of partitions, and the data structures are in consecutive sectors of the disk. This section will give an overview of the Apple partitions, the details of the data structures, and discuss how to view the details.

GENERAL OVERVIEW

The Apple partitions are described in the partition map structure, which is located at the beginning of the disk. The firmware contains the code that processes this structure, so the map does not contain boot code like we saw in the DOS partition table. Each entry in the partition map describes the starting sector of the partition, the size, the type, and the volume name. The data structure also contains values about data inside of the partition, such as the location of the data area and the location of any boot code.

The first entry in the partition map is typically an entry for itself, and it shows the maximum size that the partition map can be. Apple creates partitions to store hardware drivers, so the main disk for an Apple system has many partitions that contain drivers and other non-file system content. Figure 5.9 shows an example layout of an Apple disk with three file system partitions and the partition for the partition map.

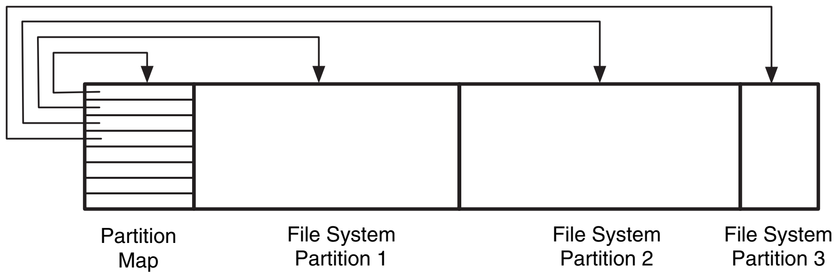


Figure 5.9 An Apple disk with one partition map partition and three file system partitions.

We will later see that BSD systems have a different partition structure called the disk label. Even though Mac OS X is based on a BSD kernel, it uses an Apple partition map and not a disk label.

DATA STRUCTURES

Now that we have examined the basic concepts of an Apple partition, we can look at the data structures. As with other data structures in this book, they can be skipped if you are not interested. This section also contains the output of some analysis tools using an example disk image.

Partition Map Entry

The Apple partition map contains several 512-byte data structures, and each partition uses one data structure. The partition map starts in the second sector of the disk and continues until all partitions have been described. The partition data structures are laid out in consecutive sectors, and each map entry has a value for the total number of partitions. The 512-byte data structure is shown in Table 5.7.

Table 5.7 Data structure for Apple partition entries.

Byte Range	Description	Essential
0–1	Signature value (0x504D)	No
2–3	Reserved	No
4–7	Total Number of partitions	Yes
8–11	Starting sector of partition	Yes
12–15	Size of partition in sectors	Yes
16–47	Name of partition in ASCII	No
48–79	Type of partition in ASCII	No
80–83	Starting sector of data area in partition	No
84–87	Size of data area in sectors	No
88–91	Status of partition (see table 5-8)	No
92–95	Starting sector of boot code	No
96–99	Size of boot code in sectors	No
100–103	Address of boot loader code	No
104–107	Reserved	No
108–111	Boot code entry point	No
112–115	Reserved	No
116–119	Boot code checksum	No
120–135	Processor type	No
136–511	Reserved	No

The type of partition is given in ASCII and not as an integer as other partition schemes use. The status values for each partition apply to both older A/UX systems and modern Macintosh systems. A/UX is an older operating system from Apple. The status value can have one of the values shown in Table 5.8 [Apple 1999].

Table 5.8 Status value for Apple partitions.

Type	Description
0x00000001	Entry is valid (A/UX only)
0x00000002	Entry is allocated (A/UX only)
0x00000004	Entry in use (A/UX only)
0x00000008	Entry contains boot information (A/UX only)
0x00000010	Partition is readable (A/UX only)
0x00000020	Partition is writable (Macintosh & A/UX)
0x00000040	Boot code is position independent (A/UX only)
0x00000100	Partition contains chain-compatible driver (Macintosh only)
0x00000200	Partition contains a real driver (Macintosh only)
0x00000400	Partition contains a chain driver (Macintosh only)
0x40000000	Automatically mount at startup (Macintosh only)
0x80000000	The startup partition (Macintosh only)

The data area fields are used for file systems that have a data area that does not start at the beginning of the disk. The boot code fields are used to locate the boot code when the system is starting.

To identify the partitions in an Apple disk, a tool (or person) reads the data structure from the second sector. It is processed to learn the total number of partitions, and then the other partition information from it is collected. The first entry is usually the entry for the partition map itself. The next sector is then read, and the process continues until all partitions have been read. Here are the contents of the first entry in the partition map:

```
# dd if=mac-disk.dd bs=512 skip=1 | xxd
0000000: 504d 0000 0000 000a 0000 0001 0000 003f  PM............?
0000016: 4170 706c 6500 0000 0000 0000 0000 0000  Apple..........
```

```
0000032: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000048: 4170 706c 655f 7061 7274 6974 696f 6e5f  Apple_partition_
0000064: 6d61 7000 0000 0000 0000 0000 0000 0000  map.............
0000080: 0000 0000 0000 003f 0000 0000 0000 0000  .......?........
0000096: 0000 0000 0000 0000 0000 0000 0000 0000  ................
[REMOVED]
```

Apple computers use Motorola PowerPC processors and, therefore, store data in big-endian ordering. As a result, we will not need to reverse the order of numbers like we did with DOS partitions. We see the signature value of 0x504d in bytes 0 to 1 and the number of partitions in bytes 4 to 7, which is 10 (0x0000000a). Bytes 8 to 11 show us that the first sector of the disk is the starting sector for this partition and that its size is 63 sectors (0x3f). The name of the partition is "Apple," and the type of partition is "Apple_partition_map." Bytes 88 to 91 show that no flags for this partition are set. Other entries in the partition map that are not for the partition map itself have status values set.

Example Image Tool Output

You can view an Apple partition map with `mmls` in The Sleuth Kit. The `fdisk` command in Linux will not show the contents of a partition map. Here are the results from running `mmls` on a 20GB iBook laptop:

```
# mmls -t mac mac-disk.dd
MAC Partition Map
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Unallocated
01:  00      0000000001   0000000063   0000000063   Apple_partition_map
02:  -----   0000000001   0000000010   0000000010   Table
03:  -----   0000000011   0000000063   0000000053   Unallocated
04:  01      0000000064   0000000117   0000000054   Apple_Driver43
05:  02      0000000118   0000000191   0000000074   Apple_Driver43
06:  03      0000000192   0000000245   0000000054   Apple_Driver_ATA
07:  04      0000000246   0000000319   0000000074   Apple_Driver_ATA
08:  05      0000000320   0000000519   0000000200   Apple_FWDriver
09:  06      0000000520   0000001031   0000000512   Apple_Driver_IOKit
10:  07      0000001032   0000001543   0000000512   Apple_Patches
11:  08      0000001544   0039070059   0039068516   Apple_HFS
12:  09      0039070060   0039070079   0000000020   Apple_Free
```

In this output, the entries are sorted by starting sector, and the second column shows in which entry in the partition map the partition was described. In this case, the entries were already in sorted order. We can see in entry 12 that Apple reports the sectors that are not currently allocated. Entries 0, 2, and 3 were added by `mmls` to show what space the partition map is using and which sectors are free. The drivers listed here are used by the system when it is booting.

An alternative tool that can be used on a raw disk image is the `pdisk` tool with the `-dump` flag on OS X:

```
# pdisk mac-disk.dd -dump
mac-disk.dd  map block size=512
   #:                 type name                length   base    ( size )
   1:  Apple_partition_map Apple                   63 @ 1
   2:       Apple_Driver43*Macintosh               54 @ 64
   3:       Apple_Driver43*Macintosh               74 @ 118
   4:     Apple_Driver_ATA*Macintosh               54 @ 192
   5:     Apple_Driver_ATA*Macintosh               74 @ 246
   6:       Apple_FWDriver Macintosh              200 @ 320
   7:   Apple_Driver_IOKit Macintosh              512 @ 520
   8:        Apple_Patches Patch Partition        512 @ 1032
   9:            Apple_HFS untitled          39068516 @ 1544    ( 18.6G)
  10:           Apple_Free                          0+@ 39070060

Device block size=512, Number of Blocks=10053
DeviceType=0x0, DeviceId=0x0
Drivers-
1: @ 64 for 23, type=0x1
2: @ 118 for 36, type=0xffff
3: @ 192 for 21, type=0x701
4: @ 246 for 34, type=0xf8ff
```

As was mentioned in the Introduction, Apple disk image files (which are different from forensic disk image files) also can contain a partition map. A disk image file is an archive file that can save several individual files. It is similar to a zip file in Windows or a tar file in Unix. The disk image file can contain a single partition with a file system, or it can contain only a file system and no partitions. The layout of a test disk image file (files with an extension of `.dmg`) has the following layout:

```
# mmls -t mac test.dmg
MAC Partition Map
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Unallocated
```

```
01:  00      0000000001   0000000063   0000000063   Apple_partition_map
02:  -----   0000000001   0000000003   0000000003   Table
03:  -----   0000000004   0000000063   0000000060   Unallocated
04:  01      0000000064   0000020467   0000020404   Apple_HFS
05:  02      0000020468   0000020479   0000000012   Apple_Free
```

ANALYSIS CONSIDERATIONS

The only unique characteristic of Apple partitions is that there are several unused fields in the data structure that could be used to hide small amounts of data. Also data could be hidden in the sectors between the last partition data structure and the end of the space allocated to the partition map. As with any partitioning scheme, anything could be in the partitions that have an official looking name or that claim to have a given type.

SUMMARY

The Apple partition map is a fairly simple structure and is easy to understand. The data structures are all located in one place, and the maximum number of partitions is based on how the disk was originally partitioned. The `mmls` tool allows us to easily identify where the partitions are located if we are using a non-Apple system, and the `pdisk` tool can be used on an OS X system.

REMOVABLE MEDIA

Most removable media also have partitions, but they use the same structures that hard disks use. The exception to this rule are floppy disks that are formatted for FAT12 in a Windows or UNIX system. They do not have partition tables, and each entire disk is treated like a single partition. If you image a floppy disk, you can directly analyze the image as a file system. Some of the small USB storage tokens (sometimes called 'thumb drives') do not have partitions and contain one file system, but some of them do have partitions.

Larger removable media, such as Iomega ZIP disks, do have partition tables. The partition table on a ZIP disk will depend on whether it has been formatted for a Mac or a PC. A PC-formatted disk will have a DOS-based partition table and by default will only have one partition in the fourth slot.

Flash cards, which are commonly used in digital cameras, also typically have a partition table. Many flash cards have a FAT file system and can be analyzed using normal investigation tools. Here is DOS-based partition table from a 128MB flash card:

```
# mmls -t dos camera.dd
DOS Partition Table
Units are in 512-byte sectors
     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000001   0000000031   0000000031   Unallocated
02:  00:00   0000000032   0000251647   0000251616   DOS FAT16 (0x06)
```

Putting flash cards in a USB or Firewire reader and using `dd` in Linux can easily image them.

CD-ROMs are more complex because there exist many possible variations. Most CDs use the ISO 9660 format so that multiple operating systems can read the contents of the CD. The ISO 9660 naming requirements are strict, and there are extensions to ISO 9660, such as Joliet and Rock Ridge, which are more flexible. CDs are complex to describe because one CD may have data in a basic ISO 9660 format and in a Joliet format. If a CD is also an Apple hybrid disc, the data could also be in an Apple HFS+ format. The actual content of the files is only saved once, but the data are pointed to by several locations.

Recordable CDs, or CD-Rs, have a notion of a session. A CD-R can have one or more sessions on it, and the purpose of the sessions is that you can continue to add data to CD-R more than once. A new session is made each time data are burned to the CD-R. Depending on the operating system in which the CD is used, each session may show up as though it was a partition. For example, I used an Apple OS X application to create a CD with three sessions. When the CD was used in an OS X system, all three of the sessions were mounted as file systems. When the CD was used in a Linux system, the last session was the default session to be mounted, but the other two could be mounted by specifying them in the mount command. The `readcd` tool (`http://freshmeat.net/projects/cdrecord/`) can be used to determine the number of sessions on a CD. When the CD was used in a Microsoft Windows XP system, the system said it was invalid, although Smart Project's ISO Buster program (`http://www.isobuster.com`) in Windows could see all three sessions. Different results may occur if the multiple session CD was created from within Windows. It is important with CD-Rs to use a specialized CD analysis tool to view the contents of all sessions and not rely on the default behavior of your analysis platform.

Some CDs also contain the partition systems of the native operating system. For example, a hybrid CD is one that is in an ISO format and an Apple format. Inside the session are an Apple partition map and HFS+ file system. Standard Apple investigation techniques can be applied to these disks. For example, here is the result of running `mmls` on hybrid disk:

```
# mmls -t mac cd-slice.dd
MAC Partition Map
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Unallocated
01:  00      0000000001   0000000002   0000000002   Apple_partition_map
02:  -----   0000000001   0000000002   0000000002   Table
03:  -----   0000000003   0000000103   0000000101   Unallocated
04:  01      0000000104   0000762559   0000762456   Apple_HFS
```

Many bootable CDs also have a native partition system. Sparc Solaris bootable CDs have a Volume Table of Contents structure in the ISO volume, and Intel bootable CDs can have a DOS-based partition table at the beginning of the CD. These structures are used after the operating system has been booted from the CD and the code required to boot the system is in the ISO format.

BIBLIOGRAPHY

Agile Risk Management. "Linux Forensics—Week 1 (Multiple Session CDRs)." March 19, 2004. `http://www.agilerm.net/linux1.html`.

Apple. "File Manager Reference." March 1, 2004. `http://developer.apple.com/documentation/Carbon/Reference/File_Manager/index.html`.

Apple. "Inside Macintosh: Devices." July 3, 1996. `http://developer.apple.com/documentation/mac/Devices/Devices-2.html`.

Brouwer, Andries. "Minimal Partition Table Specification." September 16, 1999. `http://www.win.tue.nl/~aeb/partitions/partition_tables.html`.

Brouwer, Andries. "Partition Types." December 12, 2004. `http://www.win.tue.nl/~aeb/partitions/partition_types.html`.

Carrier, Brian. "Extended Partition Test." *Digital Forensic Tool Testing Images*, July 2003. `http://dftt.sourceforge.net/test1/index.html`.

Apple. "The Monster Disk Driver Technote." November 22, 1999. `http://developer.apple.com/technotes/tn/pdf/tn1189.pdf`.

CDRoller. *Reading Data CD*, n.d. `http://www.cdroller.com/htm/readdata.html`.

ECMA. "Volume and File Structure of CDROM for Information Interchange." *ISO Spec*, September 1998. `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-119.pdf`.

Landis, Hale. "How it Works: Master Boot Record." May 6, 2002. `http://www.ata-atapi.com/hiwmbr.htm`.

Landis, Hale. "How it Works: Partition Types." December 12, 2004. `http://www.ata-atapi.com/hiwtab.htm`.

Microsoft. "Basic Disks and Volumes Technical Reference." *Windows Server 2003 Technical Reference*, 2004. `http://www.microsoft.com`.

Microsoft. "Managing GPT Disks in Itanium-based Computers." *Windows® XP Professional Resource Kit Documentation*, 2004a. `http://www.microsoft.com`.

Microsoft. "MS-DOS Partitioning Summary." *Microsoft Knowledge Base Article 69912*, December 20, 2004b. `http://support.microsoft.com/default.aspx?scid=kb;EN-US;69912`.

Stevens, Curtis, and Stan Merkin. "El Torito: Bootable CD-ROM Format Specification 1.0." January 25, 1999. `http://www.phoenix.com/resources/specs-cdrom.pdf`.