Ethan Roche Woodward (20094256)

FSF Assignment 2

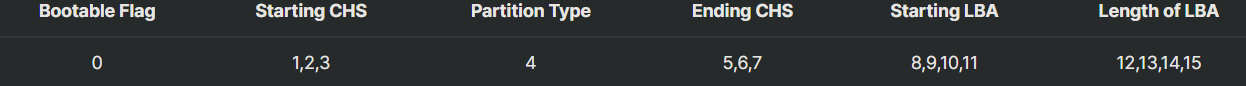
# Partition Table:

To get my laptop’s partition table I did the command “dd --localwrt if=//./physicaldrive1 of=partition.dd bs=512 skip=0 count=1” which took the first sector of my physical drive with Windows. This is the partition table. The hex for the partition table starts at byte 446 and my MBR partition was follows:

A picture containing text

Description automatically generated

I used this template to decode my MBR:



Byte 4 here shows that the disk is a GPT partitioned disk. This means the disk has a protective GPT MBR with a GPT or EFI header after. This matches with what using Windows’ disk management utility shows me along with mmls from the sleuth kit.

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

The first byte(Byte 0) indicates that the drive isn’t bootable but this is only so older/legacy BIOSes cannot boot from it as they will consider the drive as damaged.

The next three bytes point to where the GPT header is located. They point to CHS(0,0,2) . This is decoded by converting the hex values into binary then moving the first two bits from the Sector value to the start of the Cylinder Value and then simply converting the resulting binary into decimal.

Text, letter

Description automatically generated

Bytes 5,6 and 7 point to the ending CHS(Cylinder Head Sector). These are decoded in the same way as the Starting CHS. The decoded values are as follows:

Text

Description automatically generated

The following four bytes are the Starting LBA, or the Logical Block Address. They are in little endian so had to be reordered, leaving me with the Starting Logical Block Address being 1. Therefore CHS(0,0,2) = LBA 1. The next four bytes were the length of the LBA which was 4,294,967,295.

Text

Description automatically generatedText

Description automatically generated

As my MBR had a GPT header, I needed to decode entries from the next 10 sectors using the following templates:

GPT Header:

A screenshot of a computer

Description automatically generated with medium confidence

Series of entries describing each partition:

A screenshot of a computer screen

Description automatically generated with medium confidence

## GPT Header:

Bytes 0 to 7 are the Signature value. Decoding the hex values gives “EFI PART” which identifies the GPT as an EFI compatible header.

Text

Description automatically generated

The next four bytes are referring to the Version/Revision of the GPT Header. This decodes to be version 1 which is the current one.

Text

Description automatically generated

Following this is four bytes corresponding to the size of the GPT header.

Text

Description automatically generated

Bytes 16-19 are the CRC 32 checksum in little endian. It can also be confirmed by zeroing the bytes and then running a checksum analysis on a hex editor.

Text

Description automatically generated Graphical user interface

Description automatically generated with medium confidence

Bytes 20-23 are reserved and will always be 00 00 00 00.

24-31 are the LBA (Logical Block Address) of the current header. This is in little endian and is normally 01, as mine is.

Text

Description automatically generated

32-39 are the LBA of the other header structure which is a backup of the main header. These are stored in little endian and when converted to decimal gives Sector 500,118,191.

Text

Description automatically generated

The next 8 bytes are for the LBA of the start of the partition area. For my GPT I got Sector 34.

Text

Description automatically generated

Following this are the 8 bytes for the LBA of end of the partition area. For me this was Sector 500,118,158.

Text

Description automatically generated

52-71 is the Disk GUID (Globally Unique Identifier) of this disk drive.

Text

Description automatically generated

Then there’s 72-79 which have the LBA of the start of the partition table. This ended up as 2.

Text

Description automatically generated

80-83 gives the number of areas in the partition table, this being 128 for me.

Text

Description automatically generated

The next four bytes give the size of each entry in the partition table. This is usually 128 and that is what I also have.

Text

Description automatically generated

Then 88-91 is the CRC 32 checksum of the Partition table.

Text

Description automatically generated

From here 92 until the end of the header is all reserved and padded with all zero bytes.

## Partitions:

Slot 000:

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

Slot 001:

Text

Description automatically generated

Slot 002:

Text

Description automatically generated

Slot 003:

Text

Description automatically generated

For all of these, the start LBA and ending LBA along with the names are the exact same as what doing “mmls \\.\physicaldrive1” gives:

Text

Description automatically generated

Using Microsoft’s “PARTITION\_INFORMATION\_GPT structure” page (PARTITION\_INFORMATION\_GPT - Win32 apps, 2021) and the Partition type of each partition, I was able to figure out the partition types. Slot 000 is an EFI system partition, Slot 1 is a Microsoft Reserved Partition, Slot 2 is a partition type made by windows and can be assigned a drive letter by windows and Slot 3 is a Microsoft recovery partition.

Using that same page, I was also able to know what each attribute meant which is shown in the image of each decoded partition.

A picture containing diagram

Description automatically generated

Mapping of MBR and GPT.

# MFT:

This is the Master File Table and has information about all the files and directories, including itself. Before I began decoding the MFT, I had to make an image of the supplied file by using the dd command. From here I used fsstat on the image file to find the MFT. I then used “blkcat -h image.dd 4” to look at the hex of the MFT. To take out and decode the MFT entry itself, I took the first 42 bytes and followed this template along with using information from “BASIC CONCEPTS | NTFS DATA STRUCTURES” (Basic Concepts | NTFS Data Structures, 2022):

Table

Description automatically generated

Looking at the first four bytes, the values are the values for the word “FILE” which is the signature of an MFT entry.

Bytes 4-5 contain the offset to the fixup array in little endian. This means that the fixup array is 48 bytes into the MFT entry.



6-7 show the number of entries in the fixup array which in this case is 3.



After this, 8-15 has the $Logfile Sequence Number which records when updates are made to the file system to more quickly fix a corrupt system if it happens. For the supplied file this was 0.

The Sequence Value in 16-17 is raised by one when an entry is allocated. For this file, the value was 1 meaning this entry has only been allocated once.

The Link Count in 18-19 goes up by one for each directory with an entry for this MFT entry. This value is 1 for the MFT entry.

Then Bytes 20-21 show the offset to the first attribute which in this case is byte offset 56. (Note, “==” used when changing from little endian and “:” after used when converting to decimal).



22-23 is the flags and for this entry the flags are 1 which reveals that the entry is in use by the system.

From this is the Used size of the MFT entry, bytes 24-27, which is 408 bytes and then the allocated size of the MFT entry in 28-31 which is 1024 bytes.

Bytes 32-39 have the file reference to base record as 0 which indicates that this is a base entry.

Bytes 40 and 41 show that the next attribute ID that will be assigned is 4, telling me that 1-3 will already be assigned to attributes.

After this I went to byte 48 as it is the start of the fixup array. The first two bytes (04 00/0x0004) are the signature value. The two following bytes need to be used to replace the signature value at the last two bytes of each sector. Both of them have 0x0004 which gets replaced with 0x0000. The first attribute is in byte 56, the value of which is 10(hex) in the case for this file and the attributes end at 504 marked by 0xFFFF FFFF.



# MFT Entries & Associated Attributes:

To decode these entries, I used fls and istat from TSK along with NTFS FILE TYPES - NTFS.COM (NTFS File Types - NTFS.com, n.d.) to understand the attributes what they mean.

Entry 7: $Boot

This file is 8192 bytes large and it’s parent MFT entry is entry 5. It has one link. The values of it’s $STANDARD\_INFORMATION Attribute show it’s flags, owner ID, Security ID, and created, modified, MFT modified and Accessed dates and time.

This entry is flagged as a hidden entry and a system entry. Both the owner and security IDs are 0.

The times are as follows:

Text

Description automatically generated

The $FILE\_NAME attribute simply gives the name of the entry as $BOOT, it’s parent entry, sequence and both allocated and actual sizes. For this entry the Parent MFT entry is 5 and so is it’s sequence number. The entry is allocated and uses 8192 bytes.

The entry also has a $SECURITY\_DESCRIPTOR attribute and a $DATA attribute. The Security Descriptor attribute is 100 bytes, and the Data attribute is 8192 bytes with a starting address of 0 and length of 2 clusters (0 and 1). $STANDARD\_INFORMATION, $FILE\_NAME and $SECURITY\_DESCRIPTOR are all Resident while $DATA is Non-Resident. This means that the Data is too large to fit into the MFT record, so it is allocated clusters of disk space somewhere else on the volume.