Live Capture Procedures

Author: Dr Craig S Wright GSE GSM LLM MStat

Abstract / Lead

This article takes the reader through the process of carving files from a hard drive. We explore the various partition types and how to determine these (even on formatted disks), learn what the starting sector of each partition is and also work through identifying the length the sector for each partition. In this, we cover the last two bytes of the MBR and why they are important to the forensic analyst. This process is one that will help the budding analyst or tester in gaining an understanding of drive partitions and hence how they can recover and carve these from a damaged or formatted drive.

.

Introduction

This article takes the reader through the process of carving files from a hard drive. We explore the various partition types and how to determine these (even on formatted disks), learn what the starting sector of each partition is and also work through identifying the length the sector for each partition. In this, we cover the last two bytes of the MBR and why they are important to the forensic analyst. This process is one that will help the budding analyst or tester in gaining an understanding of drive partitions and hence how they can recover and carve these from a damaged or formatted drive.

The format of this article is a step by step process that is designed to take the reader through the analysis of a hard drive. Although the process may vary somewhat for each drive, the fundamentals remain the same and following these steps will allow the analyst to recover drive partitions that have been damaged or formatted even when the automated tools fail

The beginning

There are a number of commands we shall be using in this article that are fairly standard on most Linux distro's. In this artile, it is assumed that the analyst has already creates a bitwise image of the hard disk drive to be examined using "dd" or a similar tool.

The commands we will start with to copy our MBR (master boot record):

- `dd if=Image.dd of=MBR.img bs=512 count=1`
- `ls -al *img`
- `khexedit MBR.img &`

Here, we first extract the MBR from our image file (in this case IMG.dd) and extract the data to a file called MBR.img. Note that we have extracted only the first 512 bytes and we can validate this image file using the command "ls -al *img".

Master Boot Record (MBR)

In a most drive formats (there are exceptions with some RISC systems etc) that we will analyse, each Partition entry is always 16 bytes in length. More, the end of any MBR Marker is 0x55AA (ALWAYS)!

Partition	Offset	Byte Place
1st	0x01BE	446
2nd	0x01CE	462
3rd	0x01DE	478
4th	0x01EE	492

Table 1 The HDD table

We see from the file "MBR.img" the partition information displayed in table 1 and figure 1. The offset for each of the partition locations remains the same. In this way we can easily determine where the required partition data resides and hence extract and analyse it.

What are the partition types?

Each drive is divided into a number of partitions (these are the things we see in Windows as C:, D: etc). These are defined in table 2 from the offset location defined in table 1.

Offset (Dec)	Length bytes	Content
0	1	State of partition: 0x00 if active, 0x80 if not active
1	1	The head where the partition starts
2	2	The sector and cylinder where the partition is started
4	1	Type of partition (see table 3)
5	1	Head on which the partition ends
6	2	Sector and cylinder where the partition ends
8	4	Distance in sectors from the partition start table to the starting sector (the 1st sector of the partition)
12	4	Number of sectors contained in the partition (Length of the partition)

Table 2 The partition types

And the form of the partition (w it is formatted as and more) is set through the values displayed in table 3. This is held in offset 4 qas detailed in table 2.

Hex	Partition Type
0X01	FAT 12
0X0E / 0X06	FAT 16
0X0C / 0X0B	FAT 32

0X82	Linux Swap
0X83	Linux Native
0X05	Extended
0X07	NTFS
0X0F	Microsoft Extended

Table 3 The partition types

As one of my students pointed out, you can find the partition types on Wiki...

- http://en.wikipedia.org/wiki/Master_boot_record
- http://en.wikipedia.org/wiki/Partition_type

The Partition Table

If we view the MBR in a hex editor (such as khexedit in Linux), we see the following partition values in figure 1.

- Partition 1 80 01 01 00 06 1F 7F 96 3F 00 00 00 E1 0C 00 00
- Partition 2 00 00 41 97 05 1F BF 0B 20 85 0C 00 60 99 03 00

The offset values displayed in table 1 are important. These are always the same and allow us to extract the partition information displayed in figure 1.

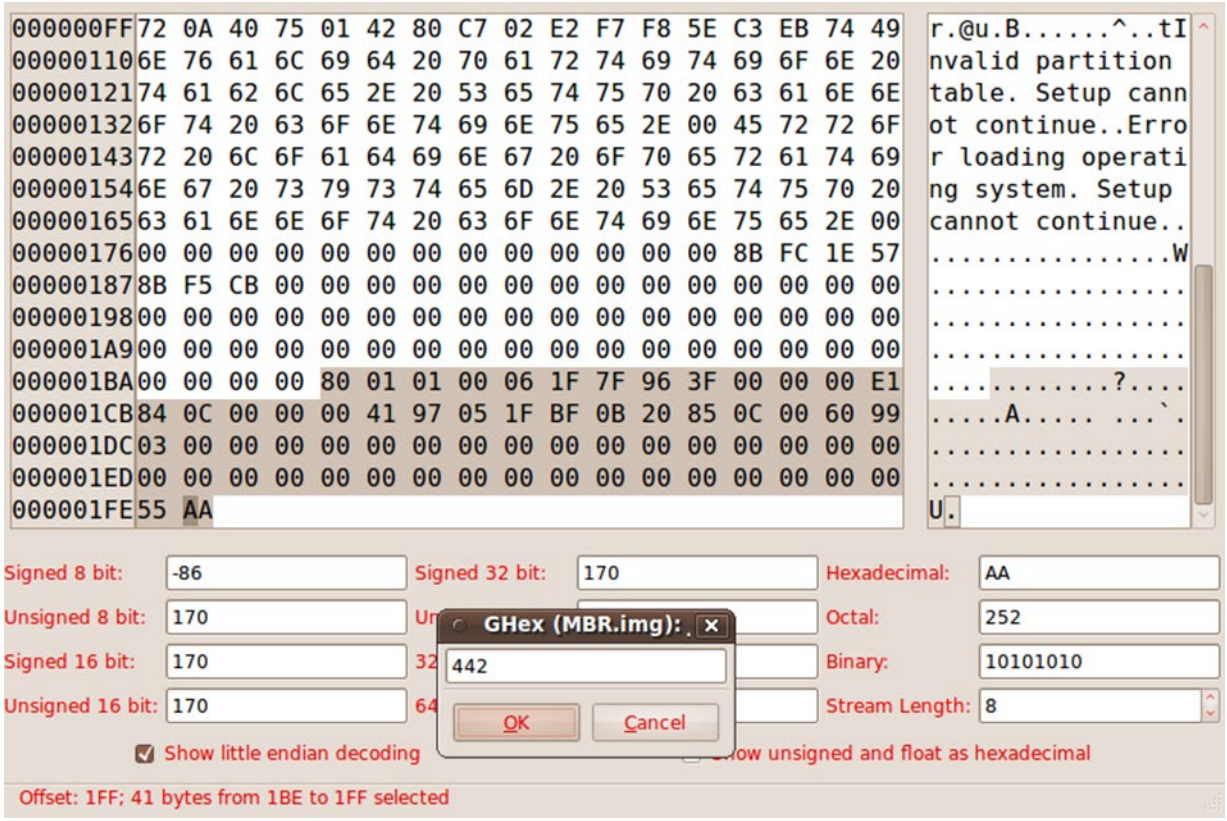

Figure 1 The MBR in KHexedit

Notice that the end of the partition entry in figure 1 is set using the value 0x55AA. As stated, the MBR Marker is always defined with the value 0x55AA.

Partition 1

We can extract the data for partition one (1) from the MBR (table 4).

80	01	01	00	06	1F	7F	96	3F	00	00	00	E1	84	0C	00
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 4 Partition 1

The data in table 4 is highlighted in figure 3. This is why the set offsets are important. Given a set offset and a defined byte length (table 1), we can always carve the partition information (such as that displayed for the example drive in table 4) from the MBR.

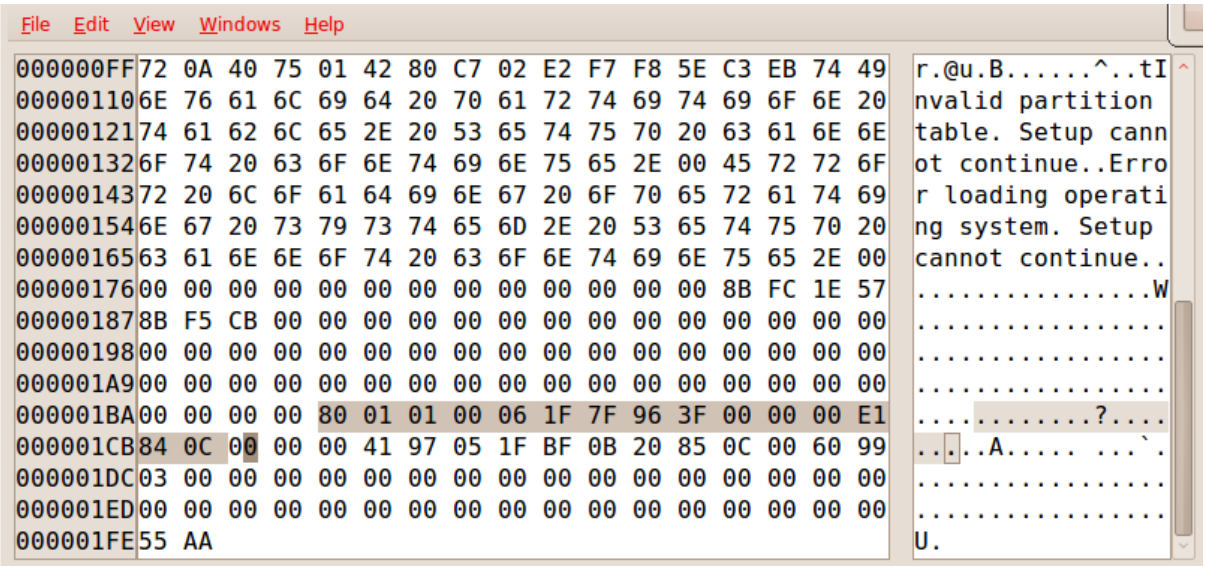

Figure 2 The MBR in KHexedit

Now, if we take the descriptions listed in table 2, we can extend our description of partition 1 with a list of the data displayed in table 5.

80	01	01	00	06	1F	7F	96	3F	00	00	00	E1	84	0C	00
0	1	2		4	5	6		8				C (or 12)			
State of Partition = 0X80 = Active	Head where partition begins = 0X01	Sector and Cylinder where the partition starts		Type of Partition = 0X06 = FAT16	Head where the partition ends	Sector and cylinder where the partition ends		Distance in sectors from the partition start table to the starting sector (the 1st sector of the partition)				Number of sectors contained in the partition (Length of the partition)			

Table 5 Partition 1 with definitions

The data in the table is displayed in “little endian” format in Intel systems (and those such as AMD following this standard). This means that the byte order is displayed in reverse (table 6).

80	01	01	00	06	1F	7F	96	3F	00	00	00	E1	84	0C	00
0	1	2		4	5	6	7	8				C (or 12)			
Active		0X0001		0X06				0X0000003F = 63 Sectors Little endian = backwards				0X000C84E1 = 820,449			

Table 6 Partition 1

The 3 key areas for forensics in the MBR include:

1. The partition type (offset 4)
2. The logical starting point for the partition as an offset in sectors (offset 8)
3. The length of the partition in sectors (offset 12)

So you can see that the data contained within this small section of the hard drive expands to provide a good deal of encoded information detailing and describing the drives features.

If we look at table 6, we see section 8 which we have expanded and calculated to determine the number of sectors used by the partition.

	Sector Distance		Number of Sectors
8	0X0000003F = 63	12	0X000C84E1

Table 7

Looking at the partition information that we extracted for partition 1 we can see at offset 4 the value of the drive format type defined for this device (table 8). Here the value 0x06 can be matched to table 3 and we see that the partition has a FAT 16 format type.

80	01	01	00	06	1F	7F	96	3F	00	00	00	E1	84	0C	00
0	1	2		4	5	6		8				C			

The Partition type is 0X06 (or FAT 16)

0X0E / 0X06	FAT 16

Table 8 The drive type

In table 9 we see how the partition length and and starting position are defined.

80	01	01	00	06	1F	7F	96	3F	00	00	00	E1	84	0C	00
0	1	2		4	5	6	7	8				C (or 12)			
Active		0X0001		0X06				0X0000003F = 63 Sectors Little endian = backwards				0X000C84E1 = 820,449			

The logical starting point for the partition as an offset in sectors

The length of the partition in sectors

	Sector Distance		Number of Sectors
8	0X0000003F = 63	12	0X000C84E1 = 820,449

Table 9 The size and location of the drive

First, the partition starts 0x3F (or decimal 63) sectors into the drive. The value at sector 8 is 0x3F000000. This value is stored in little endian format and as such is written in reverse order.

Sector C (12 decimal) can be seen to contain the value 0x000C84E1 with provides us with the length of the partition (in decimal 820,449) in sectors.

80	01	01	00	06	1F	7F	96	3F	00	00	00	E1	84	0C	00
0	1	2		4	5	6	7	8				C (or 12)			
Active		0X0001		0X06				0X0000003F = 63 Sectors Little endian = backwards				0X000C84E1 = 820,449			

Offset zero tells us if the partition is bootable or active or not
LILO or GRAB can ignore this flag and boot anyway

	Sector Distance		Type of Partition
0	0X80 = Active	4	0X06 = FAT16

Table 10 The State of Partition = 0X80 = Active

The value at section 0 on the partition data in our first partition is 0x80. This value flags the partition as being active and as such can be used as a boot device.

80	01	01	00	06	1F	7F	96	3F	00	00	00	E1	84	0C	00
0	1	2		4	5	6	7	8				C (or 12)			
Active		0X0001		0X06				0X0000003F = 63 Sectors Little endian = backwards				0X000C84E1 = 820,449			

Offset 4 has a value of 0X06
This corresponds to FAT 16

So the type of partition is FAT 16

	Sector Distance		Type of Partition
0	0X80 = Active	4	0X06 = FAT16

Table 11 State of Partition = 0X80 = Active

Taken together, sections 0 and 4 (table 11) allow us to determine that the first partition is a FAT 16 format drive primary boot partition.

Verification - MMLS

We can check our results using the command "mmls". For what we are doing, we are not using this command for the exercise, just as a check.

```
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  Meta    0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000000   0000000062   0000000063   Unallocated
02:  00:00   0000000063   0000820511   0000820449   DOS FAT16 (0x06)
03:  Meta    0000820512   0001056383   0000235872   DOS Extended (0x05)
04:  Meta    0000820512   0000820512   0000000001   Extended Table (#1)
05:  -----   0000820512   0000820574   0000000063   Unallocated
06:  01:00   0000820575   0001026143   0000205569   DOS FAT16 (0x06)
07:  Meta    0001026144   0001056383   0000030240   DOS Extended (0x05)
08:  Meta    0001026144   0001026144   0000000001   Extended Table (#2)
09:  -----   0001026144   0001026206   0000000063   Unallocated
10:  02:00   0001026207   0001056383   0000030177   DOS FAT12 (0x01)
```

Figure 3 MMLS can validate the results

For the most part, "mmls" will do all of the steps we have completed so far and more. Then reason for doing this exercise manually is twofold:

1. Sometimes the automated tools will fail. Commands such as "mmls" work well most of the time, but do fail in situations where we really need to obtain the data.
2. The use of a manual process teaches far more than running a tool ever can.

In using the tool to validate our checks we can see if we have made any foolish errors, but at the same time learn more about the system and how it is designed.

The highlighted data in figure 3 (on line 02) shows us that our calculations are correct. This shows the partition as a DOS FAT 16 partition as we have manually calculated.

What is the starting sector of each partition?

In the hex editor, you have found the 1st and 2nd primary partitions located at offsets 446 and 462 respectively (figure 2). We know this as the partitions are always set at a predefined location. The hex editor here displays the 3rd and 4th partitions are all 0's as they are unused.

Each partition entry is 16 bytes long. This is always the case. When we have extended partitions, these are similarly defined at later locations in the drive. We will cover this in a follow up article to this one.

As the values defined are all zero's, we can see that the 3rd and 4th partitions are empty and do not exist. There are extended partitions (defined within Partition 2) contained in this drive that we will continue the analysis on next article (table 13).

Back to Partition 1

80	01	01	00	06	1F	7F	96	3F	00	00	00	E1	84	0C	00
0	1	2		4	5	6		8				C			

	Starting Sector	Length of Partition	Partition Type
#1	0X0000003F = 63	0X003FFA86 = 820,449	0X06 = FAT16

Table 12 back to partition 1

If we extract the 16 bytes beginning at offset 446, we can examine the three sections we need. The 4-byte fields are stored in little endian order or backwards (table 12). From this we determine that the initial partition begins 63 sectors into the drive image.

Again, the length of the image is 0X003FFA86 which calculates to 820,449 sectors and the Partition type is 0X06 (or FAT 16).

Partition 2

In the examples above, we looked at Partitions 1 and 2. Notice there are more partitions. We will cover extended partitions shortly.

00	00	41	97	05	1F	BF	0B	20	85	0C	00	60	99	03	00
0	1	2		4	5	6		8				C (or 12)			

	Starting Sector	Length of Partition	Partition Type
#1	0X0000003F = 63	0X003FFA86 = 820,449	0X06 = FAT16
#2	0X000C8520 = 820,512	0X00039960 = 235,872	0X05 = Extended

Table 13 Partition 2

If we extract the 16 bytes beginning at offset 462 this time, we can examine the three sections we need. As always, the 4-byte fields are stored in little endian order or backwards.

From this (figure 13) we see the initial partition begins 820,512 sectors into the drive image. Next, we can see that the length of the image is 0X00039960 which calculates to 235,872 sectors.

Now, partition 2 has a value for the Partition type of 0X05 (or Extended). We can guess that Linux was installed first as a Microsoft Extended Partition would have been stored using the value 0X0F.

Doing this, we have identified the first two partitions. Now we need to go to the extended partition and find the others that are contained later in the drive. Doing this, we will find another 512 byte section stored later in the drive which we can analyse in the same manner. This we will leave to the follow-up article.

What are the last two bytes of the MBR?

This question is simple. The final 2-bytes of the MBR are always 0X55AA (figure 4).

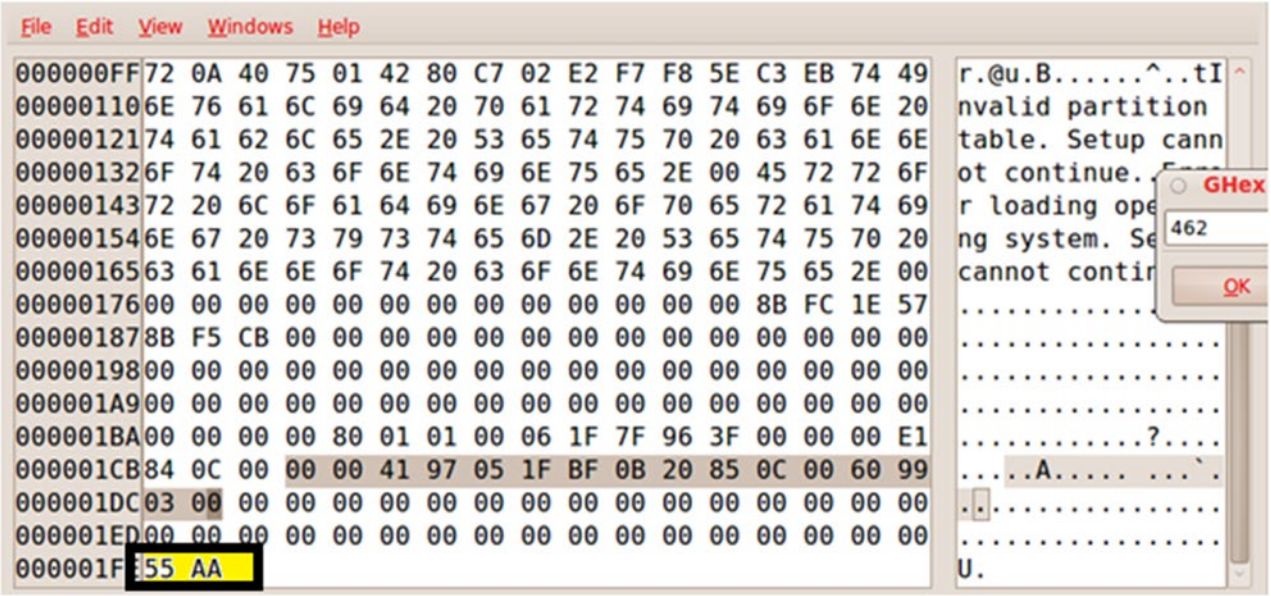

Figure 4 The MBR ends at 0x55AA

This fact allows us to find extended partitions throughout the drive. We can seek the value 0x55AA and look to see if other partition information exists on the drive. We will do this in the next article when we examine the extended partitions.

To conclude…

In a follow-up article to this one, we will continue into the Extended Partitions. In this process we will take what we have learnt of extracting the data from the MBR and now how we can find the other partitions. More, we will extend this into actually carving the partitions and when we have a good idea of just how we can find and calculate the partitions (including formatted ones), we will extend this process to recovering lost data.

Author's bio

About the Author:

Dr Craig Wright (Twitter: Dr_Craig_Wright) is a lecturer and researcher at Charles Sturt University and executive vice –president (strategy) of CSCSS (Centre for Strategic Cyberspace+ Security Science) with a focus on collaborating government bodies in securing cyber systems. With over 20 years of IT related experience, he is a sought-after public speaker both locally and internationally, training Australian and international government departments in Cyber Warfare and Cyber Defence, while also presenting his latest research findings at academic conferences.

In addition to his security engagements Craig continues to author IT security related articles and books. Dr Wright holds the following industry certifications, GSE, CISSP, CISA, CISM, CCE, GCFA, GLEG, GREM and GSPA. He has numerous degrees in various fields including

a Master's degree in Statistics, and a Master's Degree in Law specialising in International Commercial Law. Craig is working on his second doctorate, a PhD on the Quantification of Information Systems Risk.