**Digital Incident Scene Investigation and Analysis**

**Disk Acquisition & Analysis**

**-Coursework1-**

**Student ID: M00900592**

**Module leader: David Neilson**

**December 2023**

Table of Contents

[1. Drive imaging 3](#_Toc157769100)

[1.Acquiring the image 3](#_Toc157769101)

[2. Acquisition form 5](#_Toc157769102)

[3. Image verification (FTK Image summary) 6](#_Toc157769103)

[4. Why imaging is such an important process in a digital incident scene investigation? 6](#_Toc157769104)

[2. Master Boot Record (MBR) 8](#_Toc157769105)

[1. Locate the Master Boot Record (MBR) and determine how many partitions are stored on the device. 8](#_Toc157769106)

[2. What is the size in bytes of the largest partition? 9](#_Toc157769107)

[3. What is the offset address for the start of this partition? 10](#_Toc157769108)

[3. Volume Boot Record (VBR) 13](#_Toc157769109)

[1. Determine the number of bytes per sector. 13](#_Toc157769110)

[2. Determine the number of sectors per cluster. 14](#_Toc157769111)

[3. Determine the number of reserved sectors. 14](#_Toc157769112)

[4. Determine the number of FAT’s. 15](#_Toc157769113)

[5. Determine the maximum number of entries allowed in the root directory. 16](#_Toc157769114)

[6. Determine the number of sectors per FAT. 16](#_Toc157769115)

[4. FAT Data Structures 18](#_Toc157769116)

[1. Calculate the starting byte offset of FAT1. 18](#_Toc157769117)

[2. Calculate the starting byte offset of FAT2. 19](#_Toc157769118)

[3. Calculate the starting byte offset of Root Directory. 20](#_Toc157769119)

[4. Calculate the starting byte offset of Cluster 2. 20](#_Toc157769120)

[5. Directory Entries 22](#_Toc157769121)

[1. Select one directory entry that is an allocated file and larger than one cluster. Provide a screenshot of the hexadecimal output. 22](#_Toc157769122)

[2. What is the filename of the entry? 23](#_Toc157769123)

[3. On what date and time was this file created? 25](#_Toc157769124)

[6. File Content & Extraction 27](#_Toc157769125)

[1. Calculate the absolute offset address for the start of and the end of the file content, verify your answers. 27](#_Toc157769126)

[2. Verify the allocated clusters for the file in the FAT and show all the cluster chains related to the file. 29](#_Toc157769127)

[3. Use two methods that can be used to extract the file content. Verify that both files are exactly the same. 31](#_Toc157769128)

[7. References 33](#_Toc157769129)

Table of Figures

[Figure 1. Acquisition form 5](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277955)

[Figure 2. Image summary 6](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277956)

[Figure 3. Master Boot Record 8](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277957)

[Figure 4. MBR - mmls verification 9](#_Toc153277958)

[Figure 5. Partition - mmls verification 9](#_Toc153277959)

[Figure 6. Size of the partition - verification mmls 10](#_Toc153277960)

[Figure 7. Start of the first partition 11](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277961)

[Figure 8. Partition start - verification mmls 11](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277962)

[Figure 9. MMLS Verification 12](#_Toc153277963)

[Figure 10. Bytes per sector 13](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277964)

[Figure 11. Sector size - verification fsstat 13](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277965)

[Figure 12. Sectors per cluster 14](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277966)

[Figure 13. Cluster size - verification fsstat 14](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277967)

[Figure 14. Number of reserved sectors 14](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277968)

[Figure 15. Reserved sectors - verification fsstat 15](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277969)

[Figure 16. Number of FATs 15](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277970)

[Figure 17. Number of FATs - Verification fsstat 15](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277971)

[Figure 18. Maximum number of entries allowed in the Root Directory 16](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277972)

[Figure 19. RD sectors - verification fsstat 16](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277973)

[Figure 20. Number of sectors per FAT 16](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277974)

[Figure 21. Sectors per FAT - verification fsstat 17](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277975)

[Figure 22. Full report fsstat 17](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277976)

[Figure 23. Starting byte offset address of FAT1. 18](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277977)

[Figure 24. Starting sector of first FAT 18](#_Toc153277978)

[Figure 25. Starting byte offset address of FAT2. 19](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277979)

[Figure 26.Starting sector of second FAT. 19](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277980)

[Figure 27. Strating byte offset address of Root Directory 20](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277981)

[Figure 28. RD starting sector - verification fsstat. 20](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277982)

[Figure 29. Starting byte offset address of Cluster 2 21](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277983)

[Figure 30. Starting sector for Cluster 2 21](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277984)

[Figure 31. Full report - fsstat 21](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277985)

[Figure 32. Allocated file larger than 1 cluster. 22](#_Toc153277986)

[Figure 33. i-node for the selected entry 23](#_Toc153277987)

[Figure 34 Entry details - istat report 23](#_Toc153277988)

[Figure 35. Filename and extension of the entry selected. 23](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277989)

[Figure 36. Name of the file - verification istat 24](#_Toc153277990)

[Figure 37. Creation date and time 25](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277991)

[Figure 38. Creation date and time - verification istat 26](#_Toc153277992)

[Figure 39. Bytes 26-27 27](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277993)

[Figure 40. Sector no. from where the file starts - verification istat. 28](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277994)

[Figure 41. End of the file 28](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277995)

[Figure 42. Sector where the file ends - verification istat 29](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277996)

[Figure 43. Cluster chain 30](#_Toc153277997)

[Figure 44. Export a file using FTK Imager 31](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277998)

[Figure 45. Retrieve the file using icat. 31](file:///C:\Users\boamf\Desktop\Coursework1%20DISIA\Coursework1%20-%20DISIA.docx#_Toc153277999)

[Figure 46. Hash verification using md5sums. 32](#_Toc153278000)

# Drive imaging

## Acquiring the image

Steps that must be carried out to acquire the image:

* **Step 1** – Gain custody of the evidence.
* **Step 2** – Initiate the completion of the Acquisition Form. (Section B, Case No., and Evidence No.)
* **Step 3** – Establish the conditions to safely obtain the image.
* **Step 4** – Obtain the image of the device using FTK Imager.
  + **Step 4.1** – Select the action needed to perform.
  + **Step 4.2** – Select the Source Evidence Type.
  + **Step 4.3** – Select the drive to obtain the image for.
  + **Step 4.4** – Select the destination of the image, set the Destination Image Type, fill in the details of the image and set the image fragment size.
  + **Step 4.5** – Make sure the “Verify Images after they are created” box is ticked.
* **Step 5** – Verify if the Image has been acquired correctly.
* **Step 6** – Fill in the “Evidence details” section (Section C) in the Acquisition Form, using the image summary.
* **Step 7** – Fill in the “Image Details" section (Section D) in the Acquisition Form.

**Step 1** – First step of the process was to obtain the custody of the Physical Device which had to acquire the Image for.

**Step 2** -After gaining custody of the physical disk, ensuring the completion of the Acquisition Form, note the corresponding time, date, and location as part of documenting the process (Section B). The Case number and Evidence number need to be specified as well.

**Step 3** – In order to obtain the Image of the disk without damaging the evidence, set up a write-blocker device in the first place. Firstly, insert the cable into the write-blocker, connect it to the computer and turn it on. Once the write-blocker is On, insert the USB (Evidence) into the write-blocker. Evidence has appeared on the computer, and the write-blocker successfully displayed data related to the evidence. These two occurrences signify that the imaging process can start safely.

**Step 4** – Using the software called FTK Imager, obtain the image following the next step by step method:

**Step 4.1** – In the main window of the software, select the Create disk image icon.

**Step 4.2** – Set the Source Evidence Type to Physical Drive.

**Step 4.3** – In the next section in the Source Drive selection Section select the drive that need to do an Image for, the evidence (USB no.4)

**Step 4.4** – The next section requires selecting the destination for the image (where it will be saved). Upon pressing the ADD button, the Destination Image Type should be set to Raw(dd). Subsequently, details of the image are filled in. Finally, the destination folder is selected, the image name is set, the Image Fragment Size is set to 0, and the Finish button is pressed.

**Step 4.5** - Also, it's important to ensure the 'Verify images after creation' option is selected. This step confirms the match between the created image and the evidence device, a process meticulously followed. Upon completing these steps, initiating the image creation process is done by pressing the start button.

**Step 5** – After the image is created, a new window appears, displaying the image's details. It's essential to check that the hashes correspond, a reason for selecting the 'Verify images after creation' option. Matching hashes indicate the image was obtained correctly, and the evidence is now successfully imaged.

**Step 6** – Once the image is acquired and its accuracy confirmed through hash verification, the next step involves completing the Evidence section of the acquisition form. Special attention is given to the Additional Information section, where unique details about the evidence itself must be specified.

**Step 7** – Lastly, the last section of the Acquisition Form (Section D), “Image details” section must be filled in with the required information.

## b. Acquisition form

A scan version of the Acquisition form that has been used for this piece of coursework has been inserted below. (Figure 1)

A document with writing on it

Description automatically generated

Figure . Acquisition form

## c. Image verification (FTK Image summary)

Below is a snippet from the image summary, showing that after the image creation, both hashes undergo verification. This validated information allows for a comparison between the image's hashes and the computed hashes. If these hashes match in both sections, it confirms the image's correct acquisition, as illustrated in the provided example. (Figure 2)

**A computer screen shot of a computer program

Description automatically generated**

Figure . Image summary

## d. Why imaging is such an important process in a digital incident scene investigation?

Imaging plays a crucial role in digital incident scene investigations by preserving and analyzing vital digital evidence. It works by making an exact copy of a digital device's contents, keeping the original data intact for analysis without any changes. [4]

To start, investigators carefully prepare the device and choose between two methods: logical imaging, which captures active data, or physical imaging, making a complete copy of the storage, no matter its status. [5]

Next, the selected imaging method creates a mirrored copy of the device's storage, storing it separately. This copy undergoes strict verification to ensure accuracy, including analyzing its unique code to confirm it's an exact match. [1]

Through detailed documentation, every step is recorded to support the investigation's integrity. Specialized tools, like forensic software and protective devices, secure and validate digital evidence, making imaging a key player in these investigations. [6]

Imaging goes beyond preserving evidence. It helps investigators dive deep into the data, uncovering crucial details about incidents and those involved. By creating easily shareable representations of evidence, it encourages teamwork among investigators and experts, speeding up analyses and improving shared understanding. [2]

Moreover, in legal proceedings, imaging's reliability solidifies the evidence's validity. Its careful handling and creation of image files support the investigation's credibility, ensuring fair outcomes in legal contexts. [3]

In essence, imaging stands as an essential tool in digital incident scene investigations, safeguarding evidence, enabling thorough analysis, encouraging collaboration, and ensuring credibility in legal settings. It's a critical factor in unraveling complex digital puzzles and enhancing security measures against future threats.

# Master Boot Record (MBR)

## a. Locate the Master Boot Record (MBR) and determine how many partitions are stored on the device.

Master Boot Record is located in the first sector, starting at the offset 00000000 up to the end of the first sector/start of the second sector. The end of the Master Boot record is marked with the following sequence of bytes: “55 AA” (Figure 3).

**A screenshot of a computer

Description automatically generated**

Figure . Master Boot Record

The partitions slots are shown in black colour, which highlights the bytes 446-509, the bytes that represent the partitions slots. Each partition slot is represented by 16 bytes which leads us to a total of 4 partition slots, but only 1 partition is being present and used in this case. (Figure 3)

**Verification using mmls:**



Figure 4. MBR - mmls verification

The mmls table displays that the primary table resides in sector 0 and spans a length of 1 sector, aligning with the description provided earlier.



Figure 5. Partition - mmls verification

There is a single partition shown in the mmls report, which also confirms the statement above.

## What is the size in bytes of the largest partition?

The size of each partition can be calculated using the bytes 12-15 (considering we start counting from byte 0), so the last 4 bytes of the partition.

In this case, the last 4 bytes for each partition are:

* Partition 1: 0x 80 A0 3B 00
* Partition 2: 0x 00 00 00 00
* Partition 3: 0x 00 00 00 00
* Partition 4: 0x 00 00 00 00

Given the list, Partition 1 is identified as the largest. To determine its size, the following calculations need to be performed:

- Reverse the last 4 bytes of the partitions, which will give us:

**0x 3B A0 80**

* Convert the value above in decimal:

0x 3B A0 80 = **3,907,712 sectors** (this is the value of the partition in sectors)

The value obtained is in sectors. To convert this to bytes, it's necessary to consider that each sector contains 512 bytes per sector (bps). Multiplying the number of sectors by 512 will give the size in bytes.  
 Size in sectors \* Bytes per sector = Size in bytes  
 3,907,712 \* 512 = **2,000,748,544 bytes**

**Verification using mmls:**

****

Figure 6. Size of the partition - verification mmls

The size of the partition in sectors is: 3907712, similar with the size calculated before.

## What is the offset address for the start of this partition?

To determine the offset address for the start of this partition, calculate the Starting Logical Block Address using the following method::

* Take the value of bytes 8-11: **0x 80 1F 00 00.**
* Reverse the value: 0x 1F 80.
* Convert the value into decimals to get the number of sectors until the start of the partition (file system): **8064 sectors.**
* To count the byte offset address for the start of the file system the following calculations must be performed:

Starting LBA \* Bytes per sector, and convert it into hexadecimal  
8064 sectors \* 512 bytes per sector = 4,128,768 bytes  
=> in HEX: **0x 3F 0000**

A screenshot of a computer

Description automatically generated To reach the start of the file system, press CTRL + G and enter the calculated value from earlier. (“EB 3C” is the start of the partition, Figure 7):

Figure . Start of the first partition

**A number on a black background

Description automatically generatedVerification using mmls:**

Figure . Partition start - verification mmls

The Partition begins in sector 8064, aligning with the previously provided description.

**Verification chapter 2 – mmls full report**

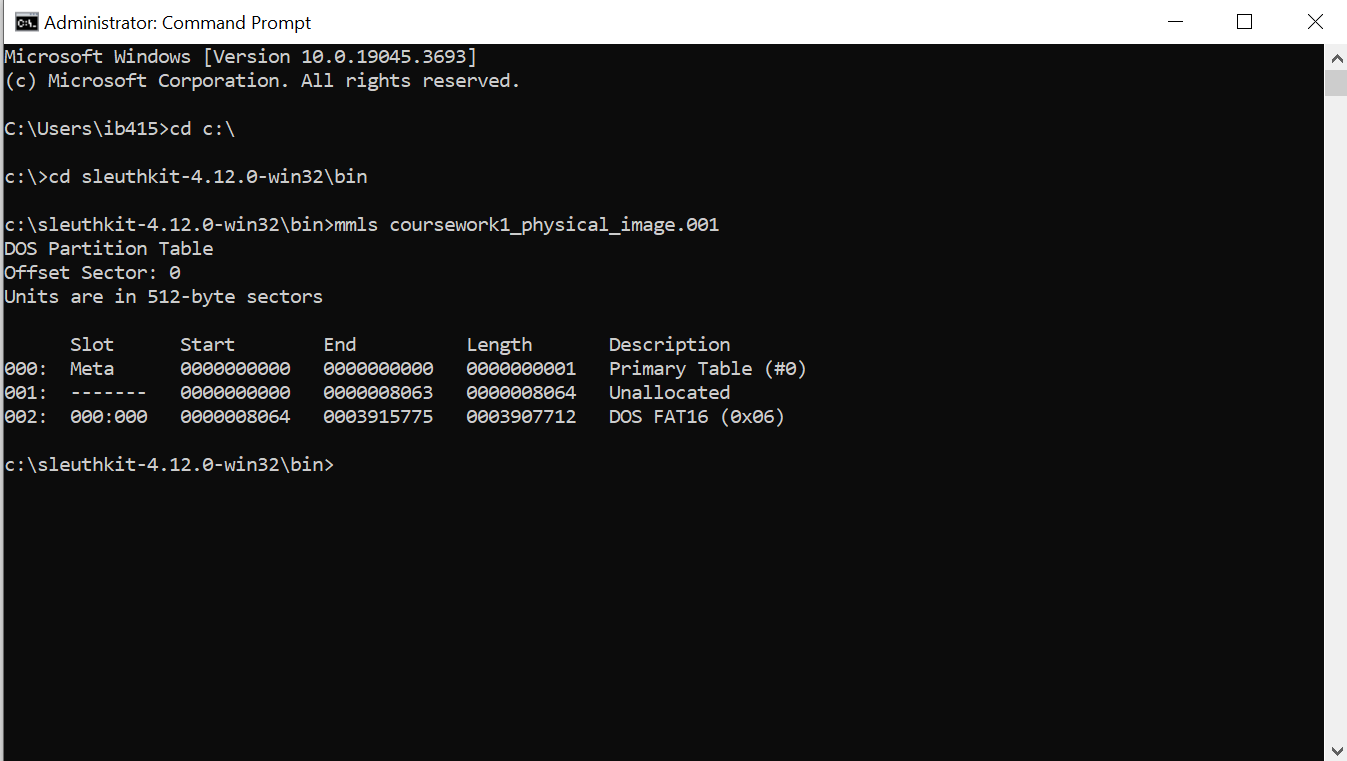


Figure 9. MMLS Verification

This full report confirms that the methods were executed accurately, ensuring the correctness of the obtained results.

# Volume Boot Record (VBR)

## Determine the number of bytes per sector.

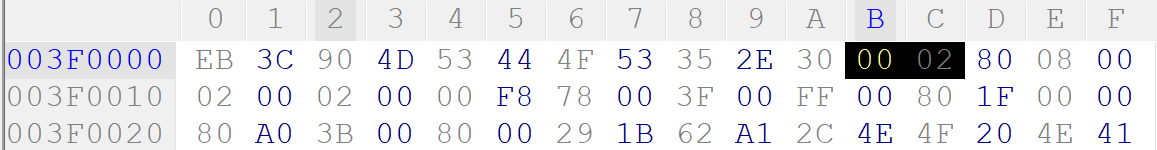
 To figure out the number of bytes per sector in the file system (at the start of the partition reached earlier), the values from bytes 11-12 (starting from 0) need to be examined. (Figure 10)

Figure . Bytes per sector

For calculating the number of bytes per sector, take the value from those two bytes, reverse it, and then convert it into decimal form.

Bytes 10-11: 0x 00 02 🡺 Reversed: 0x 02 00

The number of bytes per sector is: 0x 02 00 in decimal = 512 bps.

A black background with white text

Description automatically generated**Verification using fsstat:**

Figure . Sector size - verification fsstat

Using fsstat reveals that the sector size is 512 bytes, validating the earlier statement.

## Determine the number of sectors per cluster.

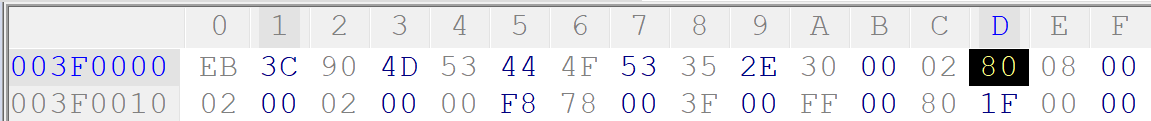
 To determine the number of sectors per cluster, the value of Byte 13 (Byte D) needs to be converted into decimal, which will provide the sectors-per-cluster value. (Figure 12):

Figure . Sectors per cluster

Byte 13 = 0x 80 🡺in decimal: **128 sectors per cluster**

**(65536 bytes/cluster)**

**Verification using fsstat:**

**A black background with white text

Description automatically generated**

Figure . Cluster size - verification fsstat

With fsstat, verification shows that the cluster size is 65536, confirming the earlier statement.

## Determine the number of reserved sectors.

To find out the number of reserved sectors, take the value of bytes 14-15, reverse it, and then convert it into decimal. In this instance, the value of bytes 14-15 is: 0x 08 00. (Figure 14)

A screenshot of a computer

Description automatically generated

Figure . Number of reserved sectors

The following calculation must be performed:

0x 08 00 🡺Reversed: 0x 08 🡺In decimal: **8 (number of reserved sectors)**

**Verification using fsstat:**

**A black background with white text

Description automatically generated**

Figure . Reserved sectors - verification fsstat

Fsstat can be used to confirm that there are 8 sectors for the Reserved area, including sector 0.

## Determine the number of FAT’s.

**A screenshot of a computer

Description automatically generated**

Figure . Number of FATs

The number of File Allocation Tables (FATs) is indicated by the value of Byte 16. In this scenario, Byte 16 is 0x 02 (as shown in Figure 16). To determine the required number, this value should be converted into decimal.

0x 02 🡺 in decimal: **2 (Number of FATs)**

**Verification using fsstat:**

**A black background with white numbers

Description automatically generated**

Figure . Number of FATs - Verification fsstat

* There are 2 FATs in the fsstat report**,** which confirms the statement above.

## Determine the maximum number of entries allowed in the root directory.

**A screenshot of a computer

Description automatically generated**

Figure . Maximum number of entries allowed in the Root Directory

The maximum number of entries allowed in the root directory is determined by the value of bytes 17-18, which in this case is 0x 00 02(Figure 18). For the precise calculation, take this value, reverse it, and then convert it into decimal as follows:

0x 00 02 🡺 Reversed: 0x 200 🡺 in decimal: **512 entries**

Considering that size of each entry is 32 bytes, the maximum size of the Root Directory will be no. of entries \* no. of bytes per entry:

512 entries \* 32 bytes per entry = **16 384 bytes (size of RD)**

In sectors this size will be: 16 384 bytes / 512 bytes per sector = **32 sectors (size of RD in sectors)**

**Verification using fsstat:**

****

Figure . RD sectors - verification fsstat

279 – 248 = 32 sectors (including the first sector)

## Determine the number of sectors per FAT.

**A screenshot of a calculator

Description automatically generated**

Figure . Number of sectors per FAT

To calculate the number of sectors per FAT, we will need the value of bytes 22-23, reverse it and convert it into decimal. In our case the value of bytes 22-23 is: 0x 78 00 (Figure 20).

0x 78 00 🡺 Reversed: 0x 78 🡺 in decimal: **120 sectors per FAT.**

A number on a black background

Description automatically generated**Verification using fsstat:**

Figure . Sectors per FAT - verification fsstat

* FAT 0 = 127 – 8 = 120 sectors (including the first sector)
* FAT 1 = 247 – 128 = 120 sectors (including the first sector)

**Verification Chapter 3 – fsstat report (full)**

**A screenshot of a computer

Description automatically generated**

Figure . Full report fsstat

This is the full report used to verify the answers above. (Figure 22)

# FAT Data Structures

## Calculate the starting byte offset of FAT1.

To calculate the starting byte offset of FAT1, begins from the start of the file system. Add the Reserved sectors located right before FAT1, and include the physical sector calculated in the previous steps. The procedure goes in this manner:

* Byte offset address for the start of the file system is:

**0x 3F 0000 (Sector 8064)** (calculated at point 2.3)

* The number of reserved sectors is: 8 sectors (calculated at point 3.3). To determine the value for addition to the previous address, multiply the number of sectors by the number of bytes per sector. For instance, 8 sectors multiplied by 512 bytes per sector results in 4096. When converted to hexadecimal, this becomes 0x1000. **(Sector 8 of the file system)**

As previously mentioned, FAT1 immediately follows the reserved area, which begins at the start of the file system. Additionally, it's essential to include the physical sector in the calculation. Therefore, the formula becomes: physical sector + reserved area equals the starting byte offset address for FAT1.

0x 3F 0000 + 0x 1000 = **0x 3F 1000 (Sector 8072/Sector 8 in the File system) (starting byte offset address of FAT1.)**

* A screenshot of a computer

  Description automatically generatedBy navigating to the obtained address, observation reveals that the desired point has been reached, indicated by the section describing the start of the FAT, marked as “F8 FF” (as shown in Figure 23).

**Verification using fsstat:**

Figure . Starting byte offset address of FAT1.

****

Figure . Starting sector of first FAT

* The first FAT of the system starts in sector 8.

## Calculate the starting byte offset of FAT2.

FAT2 is located immediately after FAT1, which, as previously mentioned, follows the reserved sectors positioned at the beginning of the file system. It was calculated earlier that reaching the first FAT requires adding the addresses of these sections to the physical sector. Given that FAT2 immediately follows FAT1, the size of FAT1 must be added to the previous sum to determine the starting point of FAT2. To achieve this, the following calculation is performed:

* Starting byte offset address of FAT1 + size of FAT1 (in hexadecimal) = starting byte offset address of FAT2.
* 0x 3F 1000 + 0x F000 (120 sectors FAT1 \* 512 bps)

= **0x 40 0000 (Sector 8192/Sector 128 of the File system)** (starting byte offset address of FAT2)

* **A screenshot of a computer

  Description automatically generated** Navigating to the obtained address reveals the same identifier “F8 FF” as in FAT1, confirming the correct location. This shows that the values of FAT1 and FAT2 match. (as illustrated in Figure 25)

Figure . Starting byte offset address of FAT2.

**Verification using fsstat:**

Figure .Starting sector of second FAT.

* The second FAT of the file system starts in sector 128.

## Calculate the starting byte offset of Root Directory.

* The Root Directory is the section immediately following FAT2. Since FAT1 and FAT2 are identical in size, finding the Root Directory involves adding the size of FAT2 (which is the same as FAT1) to the starting byte offset address of FAT2, calculated earlier. The procedure for this calculation goes as follows:
* Starting byte offset address of FAT2: 0x 40 0000.
* Size of FAT2 (similar with FAT1): 0x F000 (point 4.2)
* 0x 40 0000 + 0x F000 = **0x 40 F000(Sector 8312/Sector 248 in the File system)** (starting byte offset address of Root Directory)
* A screenshot of a computer code

  Description automatically generated By navigating to the calculated address, it becomes apparent that this is the beginning of the Root Directory. (as illustrated in Figure 27)

Figure . Strating byte offset address of Root Directory

**Verification using fsstat:**

Figure . RD starting sector - verification fsstat.

* The Root Directory starts in sector 248.

## Calculate the starting byte offset of Cluster 2.

The Content Section, specifically Cluster 2, immediately follows the Root Directory. To determine the starting byte offset address of the Content Section, we must add the size of the Root Directory, which was calculated earlier at point 3.5 (32 sectors), to the previously found address (RD address). The procedure for this calculation proceeds as follows:

* Starting byte offset address for RD: 0x 40 F000.
* Size of RD, in hexadecimal:

32 sectors \* 512 bps = 16384 bytes (0x 4000 in hex)

* 0x 40 F000 + 0x 4000 = **0x 41 3000(Sector 8344/Sector 280 in the file system) (starting byte offset address of Cluster 2)**
* A screenshot of a computer

  Description automatically generated When arriving at the calculated address, it becomes apparent that this marks the beginning of Cluster 2, representing the start of the content. (as shown in Figure 29)

Figure . Starting byte offset address of Cluster 2

**Verification using fsstat:**

Figure . Starting sector for Cluster 2

* Cluster 2/beginning of the content starts in sector 280.

**A screenshot of a computer

Description automatically generatedVerification Chapter 4 – fsstat full report (similar with chapter 3)**

Figure . Full report - fsstat

This is the full report that has been used to verify the answers above.

# Directory Entries

## Select one directory entry that is an allocated file and larger than one cluster. Provide a screenshot of the hexadecimal output.

A screenshot of a computer

Description automatically generated

Figure 32. Allocated file larger than 1 cluster.

In the example provided, each directory entry consists of 32 bytes. To select one entry from the Root Directory, two rows must be chosen. To determine the allocation status of the entry, attention should be given to the first byte. If the first byte matches "E5," it signifies that the entry has been deleted and is not allocated. The selection made in this context has a value of "49," indicating that the selected entry is allocated. (as illustrated in Figure 32)

To determine the size of the entry and ensure it is larger than one cluster, we need to consider the value of the last 4 bytes of the entry, which in this case is:

**0x 5B 87 09 00.**

* Reverse: 0x 09 87 5B
* Convert to decimal: **624,475 bytes (size of the entry)**
* According to point 3.2, the size of a cluster is equal with 65536 bytes, which means that this entry is 10 clusters big, so bigger than 1 cluster.

**Verification using fls and istat:**

The directory listings for the root directory were printed using "fls" to identify the i-node and gather additional information about the entry.

****

Figure . i-node for the selected entry

Due to the name being longer than 8 characters, the name obtained after converting the hex values to ASCII is shorter than the one displayed by "fls," which represents the file system used to store the entry. To go into the entry further, we utilize i-node 8 and examine it in more detail using "istat," as shown below:

A screen shot of a computer

Description automatically generated

Figure Entry details - istat report

Using the information provided by "istat" about the entry, it can be verified that the file is allocated, and the size of the file is 624,475 bytes.

## What is the filename of the entry?

A screenshot of a phone

Description automatically generated The filename, including the extension, is stored in bytes 1 to A. However, it's important to include the first byte of the entry, which contains the first character of the name. Therefore, we need to consider bytes 0 to 10. (as indicated in Figure 35)

Figure . Filename and extension of the entry selected.

To determine the name of the file, it is necessary to extract the value of bytes 0 to 10 and apply the ASCII table to convert it from hexadecimal to letters.

Bytes 0 – 10: 0x 49 4D 47 30 31 7E 31 4A 50 47

* ASCII Conversion:

49 – I

4D – M

47 – G

30 – 0

30 – 0

31 – 1

7E - ~

31 – 1

4A – J

50 – P

47 – G

In conclusion, the name of the file is: IMG001~1.JPG.

**Verification using istat:**

****

Figure . Name of the file - verification istat

By utilizing "istat," it is possible to confirm that the name obtained after conversion matches the name in the file system.

## On what date and time was this file created?

****

Figure . Creation date and time

To determine the creation time of the entry, it's required to extract the values of bytes 14-15 from the selected entry (as shown in Figure 37). Following this, the value should be reversed, converted into binary, and then used to calculate the hours, minutes, and seconds:

* Bytes 14-15: 0x 37 5F
* Reversed: 0x 5F 37
* Binary value: 0101 1111 0011 0111

To calculate the hours, minutes, and seconds, it's necessary to split the binary value as follows and convert them into decimal:

* Bits 0-4: seconds (multiply by 2)
* Bits 5-10: minutes
* Bits 11-15: hours
* Seconds: 10111 🡺 23\*2 = **46 seconds**
* Minutes: 111001 🡺 **57 minutes**
* Hours: 01011 🡺 **11 hours**

**Time of creation: 11:57:46 GMT**

To establish the creation date of the entry, it's essential to utilize bytes 16-17 (as indicated by the red square in Figure 37). The process involves reversing the value, converting it into binary, and then dividing the binary number as follows:

* Bits 0-4: Date
* Bits 5-8: Month
* Bits 9-15: Year (value + 1980)
* Bytes 16-17: 0x 64 46
* Reversed: 0x 46 64
* Binary: 0100 0110 0110 0100
* Bits 0-4: 00100 🡺 4th (day)
* Bits 5-8: 0011 🡺 3 (Month)
* Bits 9-15: 0100011 🡺 35 + 1980 = 2015 (Year)

**Creation date: 2015 – 03 – 04**

**Creation date and time: 2015 – 03 – 04 11:57:46 GMT**

**Verification using istat:**

**A black screen with white numbers

Description automatically generated**

Figure . Creation date and time - verification istat

By making use of the information supplied by "istat," it is possible to confirm that the creation date matches. However, there is a one-second discrepancy in the creation time. This discrepancy arises from the multiplication by 2 required to calculate the seconds. Multiplying by 2 always results in an even number, and consequently, for seconds represented by an odd number, there will be a 1-second difference between the calculated time and the time recorded in "istat."

# File Content & Extraction

## Calculate the absolute offset address for the start of and the end of the file content, verify your answers.

The initial step involves calculating the starting cluster number to pinpoint the file's location. This can be achieved by utilizing the values of bytes 26-27 from our selected entry (as shown in Figure 39). These values should be reversed and converted to decimal.

Figure . Bytes 26-27

Bytes 26-27: 0x 04 00

Reversed: 0x 00 04

Decimal: **4 (starting cluster number)**

To calculate the absolute offset address, certain previously calculated values are required:

* Bytes per sector: 512 bytes (point 3.1)
* Sectors per cluster: 128 sectors (point 3.2)
* Sector address for cluster 2: 280 (point 4.4)
* Starting cluster no. of file: 4 (point 6.1)
* Offset to start of partition: 8064 (0x 3F 0000) (point 2.3)
* Considering that the content begins in cluster 2, it's necessary to subtract 2 from the starting cluster number during the calculation.

Calculation:

* (Start cluster no. – 2) \*SPC + Sector number of cluster 2.
* (4 - 2) \* 128 + 280 = **536 (sector number from where the file starts)**
* Byte offset: Sector where file starts \* BPS = 536 \* 512 = 274 432 bytes
* Convert to hex: 0x 4 3000 (byte offset address where the file starts)
* Absolute offset: 0x 4 3000 + 0x 3F 0000 = **0x 43 3000 (absolute offset address for the start of the file)**

**A number on a black background

Description automatically generated Verification using istat:**

Figure . Sector no. from where the file starts - verification istat.

* To verify the calculation above, the first sector shown in the istat report must match the sector number from where the file starts, in our case 536.

To locate the end of the file content, it's necessary to add the file size to the starting address of the file and then subtract 1. This will lead to the end of the file.

* File size: 624475 bytes (point 5.1)
* Convert to hex: 0x 9 875B.
* (Start of file + file size) – 1 = end of content
* A screenshot of a computer

  Description automatically generated(0x 43 3000 + 0x 9875B) – 0x 1 = **0x 4C B75A (end of the file) (Figure 41)**

Figure . End of the file

Converting the offset address of the end of the file into decimal and dividing it by 512 gives the address in sectors. Subtracting the number of sectors of the physical section, which is 8064, from this address in sectors provides us with the sector where the file ends, facilitating the verification procedure. The process unfolds as follows:

* 0x 4C B75A in decimal: 5,027,674 bytes
* Value in sectors: 9189 (sector where the file ends, including the physical section; 8064 sector)
* 9189 – 8064 = **1755 (sector where the content of the file ends)**

**Verification using istat:**



Figure . Sector where the file ends - verification istat

The number of the last sector in istat where the file ends is 1755, which confirms the calculation done before.

## Verify the allocated clusters for the file in the FAT and show all the cluster chains related to the file.

Initially, the correct offset in FAT for the starting cluster must be determined:

* Cluster no. \* 2 = byte offset for cluster in FAT
* 4 \* 2 = 8 bytes = 0x 8.

Subsequently, the obtained address should be added to the address for FAT1:

* Byte offset + Absolute offset of FAT1 (point 4.1) = Starting cluster in FAT
* 0X 8 + 0x 3F 1000 = 0x 3F 1008 (Starting cluster in FAT)
* The 2 bytes at this location point to the next cluster = 0x 05 00
* Reversed: 0x 5 = 5 is the next cluster (Figure 21)

To establish the cluster chain and identify the cluster where the file ends, the following process is followed:

* Size of file: 624475 bytes
* Size of file in sectors: 624475/512 bps = 1219 sectors
* Sectors per cluster: 128 sectors (point 3.2)
* Cluster required to store the file: 1219/128 = 9.52 = 10 clusters required to store the file
* If the starting cluster number is 4, adding the 10 clusters required to store the file will lead to the file ending in cluster 13.
* Performing the same calculation using the current cluster, such as 5, as illustrated in the previous example, will lead to pointing to the next cluster, which will be 6. This process continues until the EOF marker, represented as 'FF FF,' is encountered. This marker indicates that the cluster containing it is the final cluster. In our case, this corresponds to cluster 13 within the FAT.

A screenshot of a computer

Description automatically generated

Figure 43. Cluster chain

* 0x 0D 00 is the value before the EOF, which converted into decimal will be 13, followed by the EOF marker in cluster 14, our last cluster from the chain. (Figure 43)

## Use two methods that can be used to extract the file content. Verify that both files are exactly the same.

**Method 1. FTK Imager**

Utilizing the FTK Imager software, the evidence (the image) should be added to the software. Inside partition 1, the root directory is selected. In the middle of the screen, you will find a list of root directory entries. Choose the desired file for extraction, right-click on it, and then proceed to export it to your designated destination (as shown in Figure 44).

**A screenshot of a computer

Description automatically generated**

Figure . Export a file using FTK Imager

**Method 2. Icat**

Figure . Retrieve the file using icat.

Utilizing icat (as shown in Figure 45), the file can be extracted from the image by employing the i-node provided by fls. This facilitates the retrieval of the file into the present folder.

**Verification using md5sums:**

**A screenshot of a computer program

Description automatically generated**

Figure . Hash verification using md5sums.

To verify the identity of the extracted images, md5sums were employed to generate hashes for both files (as illustrated in Figure 46). Given that the hashes for both files match, it can be affirmed that both methods have been successful, and the extracted files are identical.

# References

1. The Journal of Digital Forensics, Security and Law - <https://www.jdfsl.org/>
2. The International Journal of Digital Evidence - <https://dblp.org/db/journals/ijde/index>
3. The Scientific Working Group on Digital Evidence (SWGDE) - <https://www.swgde.org/>
4. Digital Forensics: A Practical Handbook for Computer Investigations\*\*, by Brian Carrier (Second Edition, Addison-Wesley, 2011)
5. The Cyber Forensics Handbook: A Field Guide to Computer Crime Scene Investigation\*\*, by Eoghan Casey (Second Edition, Elsevier, 2011)
6. Computer Forensics and Incident Response, by Christopher A. Carrier (Elsevier, 2019)