Università di Genova | DIBRIS DIPARTIMENTO DI INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Report - Volume & File Systems Analysis

Digital Forensics Course - University of Genova - a.a 2023/24

Author:
Cattaneo Kevin - S4944382

Index

Checksum sha256 of images

In the first place I ran the command and obtained OK for all the three files.

```
sha256sum --check *.sha256
console.dd: OK
corrupted.dd: OK
strange.dd: OK
```

console.dd

Image overview

First of all I try to obtain the information about the size of each sector:

```
fdisk -l console.dd

Disk console.dd: 4 MiB, 4194304 bytes, 8192 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00000000
```

Then I tried to achieve some more information about the image:

```
fstat console.dd:

FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT12

OEM Name: mkfs.fat
Volume ID: 0xb9e28db8
Volume Label (Boot Sector): NO NAME
Volume Label (Root Directory):
File System Type Label: FAT12

Sectors before file system: 0

File System Layout (in sectors)
Total Range: 0 - 8191
* Reserved: 0 - 0
** Boot Sector: 0
* FAT 0: 1 - 6
* FAT 1: 7 - 12
* Data Area: 13 - 8191
```

```
** Root Directory: 13 - 44
** Cluster Area: 45 - 8188
** Non-clustered: 8189 - 8191

METADATA INFORMATION
--------------------------------------------
Range: 2 - 130870
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 2048
Total Cluster Range: 2 - 2037

FAT CONTENTS (in sectors)
--------------------------------------------
49-76 (28) -> EOF
```

Analyzing the contents

I inspected the files in the file system:

```
fls -rp console.dd
```

```
r/r 4:  xbox.jpg
v/v 130867:     $MBR
v/v 130868:     $FAT1
v/v 130869:     $FAT2
V/V 130870:     $OrphanFiles
```

An image has shown up, so I extract it:

```
icat console.dd 4 > xbox.jpg
```

Without any surprise, the image represents an Xbox gaming console in front perspective.
XBOX.JPG

Since the readme.txt tells that something “fishy” is happening, I investigated further into the listing of the partitions:

```
mmls console.dd
```

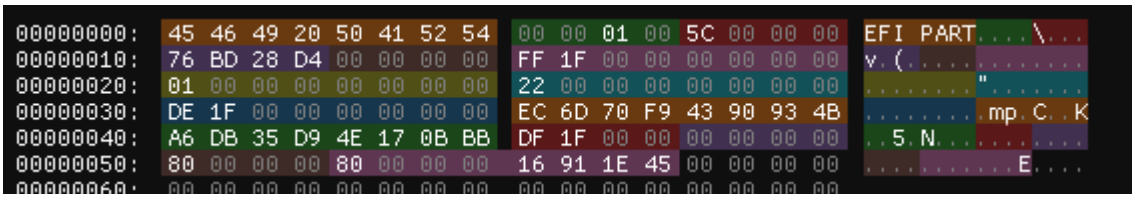
```
GUID Partition Table (EFI)
Offset Sector: 0
Units are in 512-byte sectors

      Slot      Start        End          Length       Description
000:  -------   0000000000   0000002047   0000002048   Unallocated
```

```
001:  002       0000002048   0000008158   0000006111   Linux
filesystem
002:  Meta      0000008159   0000008190   0000000032   Partition
Table
003:  -------   0000008159   0000008191   0000000033   Unallocated
004:  Meta      0000008191   0000008191   0000000001   GPT Header
```

What I discovered is that there is a second file system within the original one called “Linux Filesystem”. Moreover I see a GPT Header, so probably the original file was a GPT image that has been reformatted to be a pure FAT.

```
dd if=console.dd of=gpt_header.dd bs=512 skip=28191 count=1
```

And I viewed it via ImHex, following a pattern of mine: [GPT Pattern].



To gather some information about the labeled “Linux filesystem”, I will extract it, by looking respectively to the sector size information, the starting sector and the length of sectors occupied by the filesystem:

```
dd if=console.dd of=hidden.dd bs=512 skip=2048 count=6111
```

Now I am investigating the new filesystem, again the command:

```
fsstat hidden.dd

FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: NTFS
Volume Serial Number: 46ACAF237FF4C000
OEM Name: NTFS
Version: Windows XP

METADATA INFORMATION
--------------------------------------------
First Cluster of MFT: 4
First Cluster of MFT Mirror: 381
Size of MFT Entries: 1024 bytes
Size of Index Records: 4096 bytes
Range: 0 - 65
Root Directory: 5

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
```

```
Cluster Size: 4096
Total Cluster Range: 0 - 762
Total Sector Range: 0 - 6109

$AttrDef Attribute Values:
$STANDARD_INFORMATION (16)   Size: 48-72   Flags: Resident
$ATTRIBUTE_LIST (32)   Size: No Limit   Flags: Non-resident
$FILE_NAME (48)   Size: 68-578   Flags: Resident,Index
$OBJECT_ID (64)   Size: 0-256   Flags: Resident
$SECURITY_DESCRIPTOR (80)   Size: No Limit   Flags: Non-resident
$VOLUME_NAME (96)   Size: 2-256   Flags: Resident
$VOLUME_INFORMATION (112)   Size: 12-12   Flags: Resident
$DATA (128)   Size: No Limit   Flags:
$INDEX_ROOT (144)   Size: No Limit   Flags: Resident
$INDEX_ALLOCATION (160)   Size: No Limit   Flags: Non-resident
$BITMAP (176)   Size: No Limit   Flags: Non-resident
$REPARSE_POINT (192)   Size: 0-16384   Flags: Non-resident
$EA_INFORMATION (208)   Size: 8-8   Flags: Resident
$EA (224)   Size: 0-65536   Flags:
$LOGGED_UTILITY_STREAM (256)   Size: 0-65536   Flags: Non-resident
```

I've discovered that it is a NTFS file system from Windows XP, and not from a so-called "Linux Filesystem", so it's just a label. Again I try to discover the files in the image:

```
fls hidden.dd

r/r 4-128-1:    $AttrDef
r/r 8-128-2:    $BadClus
r/r 8-128-1:    $BadClus:$Bad
r/r 6-128-1:    $Bitmap
r/r 7-128-1:    $Boot
d/d 11-144-2:   $Extend
r/r 2-128-1:    $LogFile
r/r 0-128-1:    $MFT
r/r 1-128-1:    $MFTMirr
r/r 9-128-2:    $Secure:$SDS
r/r 9-144-3:    $Secure:$SDH
r/r 9-144-4:    $Secure:$SII
r/r 10-128-1:   $UpCase
r/r 10-128-2:   $UpCase:$Info
r/r 3-128-3:    $Volume
r/r 64-128-2:   ps5.jpg
V/V 65: $OrphanFiles
```

A new picture "ps5.jpg" appeared, so then I mounted read-only the image to retrieve it:

```
mkdir pictures
sudo mount -oro hidden.dd pictures
```

By exploring the newly created folder I analyzed the new picture: *ps5.jpg* that represents the PS5 gaming console from a rear perspective.
PS5.JPG

Even if the timestamps are not always reliable, the hypothesis of re-formatting of the image can be also verified by the creation/modified/access timestamps of the two console pictures in the two different images.

```
stat a/xbox.jpg
  File: a/xbox.jpg
  Size: 12808           Blocks: 28         IO Block: 2048
regular file
Device: 700h/1792d      Inode: 4           Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/
root)
Access: 2023-04-11 02:00:00.000000000 +0200
Modify: 2023-04-11 18:35:08.000000000 +0200
Change: 2023-04-11 18:35:09.590000000 +0200
 Birth: -

stat b/ps5.jpg
  File: b/ps5.jpg
  Size: 4258            Blocks: 16         IO Block: 4096
regular file
Device: 701h/1793d      Inode: 64          Links: 1
Access: (0777/-rwxrwxrwx)  Uid: (    0/    root)   Gid: (    0/
root)
Access: 2023-04-11 18:30:51.771508200 +0200
Modify: 2023-04-11 18:30:51.771656300 +0200
Change: 2023-04-11 18:30:51.771656300 +0200
 Birth: -
```

The ps5 image was present even before the xbox picture, so the possibility of the FAT being placed after, in time, the NTFS is concrete.
Again, the timestamps are not reliable, even because we are talking of different filesystems and maybe timestamps could be interpreted in different ways. It is just a niche information.

corrupted.dd

The following analysis will use provided questions (in *italic*) as guidelines.

Image overview

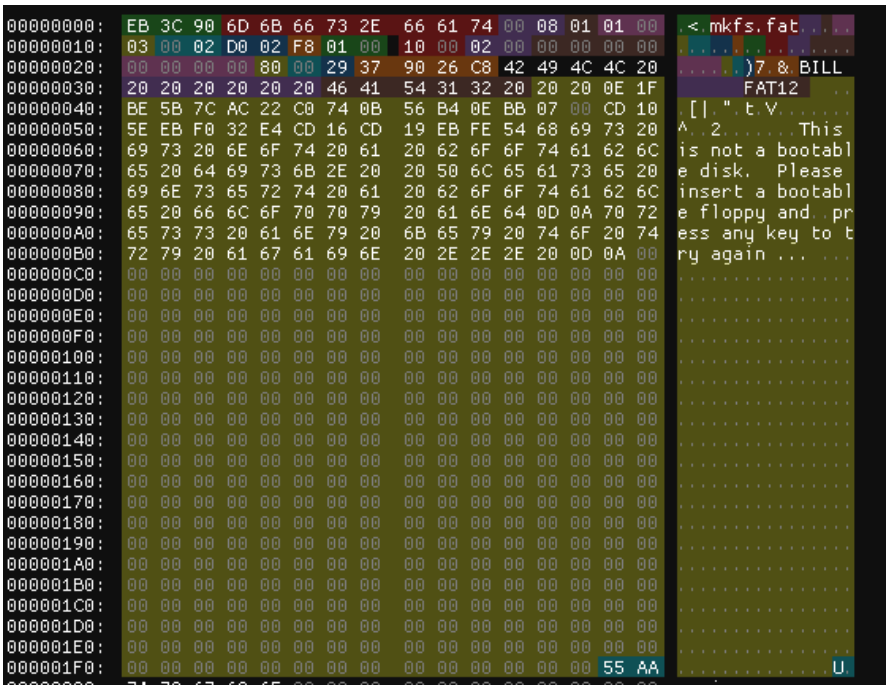
Is the image partitioned/bootable? If partitioned, list all partitions.
To find if it is bootable I open the image with ImHex editor.
The image is not bootable, also hinted by the code "**This is not a bootable disk (...)**" in the bootcode.
To find if it is partitioned:

```
mmls corrupted.dd
```

that shows no partitions; also by looking with ImHex editor I discover that the image is a pure FAT, without MBR and so without partition tables. So the image is not partitioned.

Inside, you'll find a FAT FS:
The following informations are retrieved through ImHex, applying a pattern of mine: [FAT12 Pattern]
- What is the volume label?
BILL, moreover it is located in the zone considered deprecated for the volume name
- What is the sector size?
2048 bytes
- What is the cluster size?
one sector, so 2048 bytes
- How many FAT tables are present?
3

Overview on ImHex:



Analyzing the contents

Are there sectors that are unused by the FS? If yes, what do they contain?

```
fsstat corrupted.dd
```

The filesystem occupies a total range from 0 to 719 sectors, so 720 sectors are used.

```
fdisk -b 2048 -l corrupted.dd
```

There is just one sector unused since the image contains 721 sectors. I analyzed the last sector:

```
dd if=corrupted.dd of=last_sector bs=2048 skip=720 count=1
cat last_sector
R0lGODlhVgAhALMLAP///wAAAP//AP/kAOu9ArZ9A9KfAYBUALOzs7F1AWwADgAAAA
AAAAAAAAAAAAAAACH/C05FVFNDQVBFMi4wAwEAAAAh+QQFCgALACwAAAAAVgAhAAAE
/3DJSau9OOvNd/hgKI5kaZ5o+lUB4L5wLM90bd+4G1Bt7v/AIGw36QFaRuEriWMqYz
2ixIh8Wq+0KG951IlkoGXUS/1AyeKo+bhmc49b3VDefY+792rdLnfWx3k6cXt/dHSB
iIaHh2WIYX+DSYF8hYmEhYR6lmCRc5hpYot/baJIayGlp4JFSnc5rlhNg0Gkr36xWb
O4u7hSUyrAwcImUCMdx8jJx5K+ys7P0GXQ09TJatXY2QXbBwc8K9nh0NsDAwTn4OLq
zwUDAu/v5gTN6/Ua7fD58vT2/RQFAkDAAzHPn8EK7QIgUBgg4BID/A6uSwgAwRGHFg
NAlHiQoo6AC0QXbuTo74A7ge8IjiRpz2S5fPEGGEgQkWW2AwTcBfxQjgBNmyXPDQig
oGgAAjNrAsWGE92HmT+XljxgoGpSpVLDiagWAQAh+QQFCgALACw4ABcACQAIAAAEEn
ABsGpFweqAtKReKI6klVVnBAAh+QQFCgALACw4ABcACQADAAAECxCsOVFYklIZkMYR
ACH5BAUKAAsALDgAFwAJAAMAAAQMcC0kJQAyVL1unVIEADs=
```

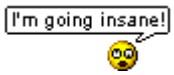
It's a base64, that I decoded:

```
base64 --decode last_sector
GIF89aV! [...]
```

I discover that it is a GIF:

```
base64 --decode last_sector > extract.gif
```

Link to GIF



How many files/deleted-files with extension TXT are there?

```
fls corrupted.dd | grep TXT | wc -l
```

There are 3 files.

Can you read them by readonly-mounting them/by using TSK? Why?

By read-only mounting them I discover that HOMEWORK.TXT and NETWORKS.TXT are **visible**, but NETWORK.TXT seems **truncated**. The last one I found previously with *fls*, *README.TXT*, whose directory entry number is 36 is the one **deleted**, so it is not present when mounting.

With the TSK I can extract it:

```
icat -r corrupted.dd 36 > README.TXT
```

HOMEWORK.TXT

NETWORK.TXT (truncated)

README.TXT (extracted)

One of them should correspond to SHA256
e9207be4a1dde2c2f3efa3aeb9942858b6aaa65e82a9d69a8e6a71357eb2d03c
... which one?
If you cannot extract such content, something is corrupted (hint hint); find the root cause and fix the file system before continuing.
You must explain how you found and fixed the problem.

In the first place I tried to run sha256sum on the three files found before and as expected no hash corresponds, so I need to repair the image, I tried 🙂.

```
fsstat corrupted.dd
```

By looking at *fsstat,* every fat content (files) is considered one sector long, that is strange. This can be also seen by looking to a file stats:

```
istat corrupted.dd 45
```

By analyzing the image with ImHex editor I discovered that there is a GIF in the VBR slack space of the image and after that other two FATs.
I extract the content of the sectors after the boot sector (sector 0):

```
dd if=corrupted.dd of=image.gif bs=2048 skip=1 count=1
dd if=corrupted.dd of=fat1 bs=2048 skip=2 count=1
dd if=corrupted.dd of=fat2 bs=2048 skip=3 count=1

diff -s fat1 fat2
Files fat1 and fat2 are identical
```

I see no difference with the two partitions, so fat2 is probably the backup of fat1.
So there are only two FAT tables. Through ImHex I can see that:

```
reserved_sectors = 1
num_fats = 3
```

So I modified those values, as follows:

```
reserved_sectors = 2
num_fats = 2
```

And saving as another image from ImHex as *correct.dd.*

By mounting the new image I can retrieve again HOMEWORK.TXT and NETWORKS.TXT (but not README.TXT since it is deleted as before, again I could retrieve that with *icat*). Now NETWORK.TXT is no longer truncated.

Inside the file corrupted.dd there are some occurrences of the string "zxgio" (without quotes); can you list their offset in bytes?
For each occurrence of such a string, determine its location with respect to the FAT file system. For instance, is the string contained inside a file? Is it in some unused/slack space? In other areas?

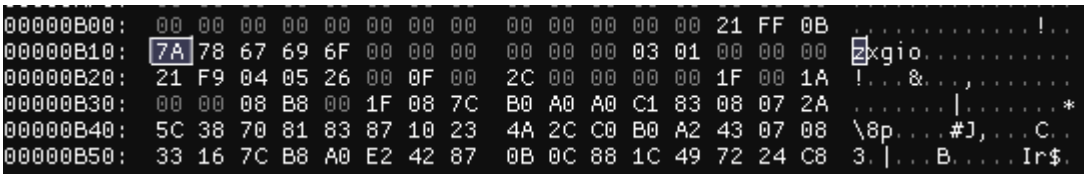
```
strings -tx corrupted.dd | grep zxgio
    200 zxgio
```

```
    b10 zxgio
  b0c39 zxgio
 135800 zxgio
```

In order to search where those strings are located I will compare the offset to the zone that I already identified in the filesystem and then I confirmed them with ImHex.

0x00000200
The first occurrence is in the reserved zone since 512 < 2048 (first sector, the boot sector 0), by observing with ImHex I found it in the slack space after the VBR.

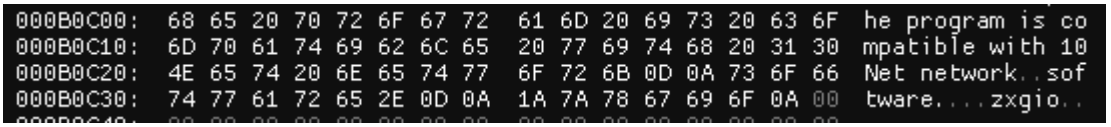
0x00000B10
The second occurrence is at 2832, so the second sector, that previously was identified as a fat table, is written in the first part of the GIF image content, that is after the heading (magic number etc.) of the GIF.



0x000B0C39
The third occurrence is at 724025 > 8192 that is where the data directories start, so it is in the data entries. I found it in the slack space of NETWORK.TXT, and can be verified by retrieving it from the corrected image, since in the *corrupted.dd* the file is truncated and the slack space is missing.

```
icat -rs correct.dd 32
[...]ʕzxgio
```

NETWORK.TXT (complete with slack)



0x00135800
The fourth one is at 1267712, again in the data entries. This one is easier to find since it's inside the content of HOMEWORK.TXT.

```
icat corrupted.dd 45
zxgio
```

HOMEWORK.TXT

strange.dd

Image overview

What partition scheme has been used for this image?

```
mmstat strange.dd
```

GPT

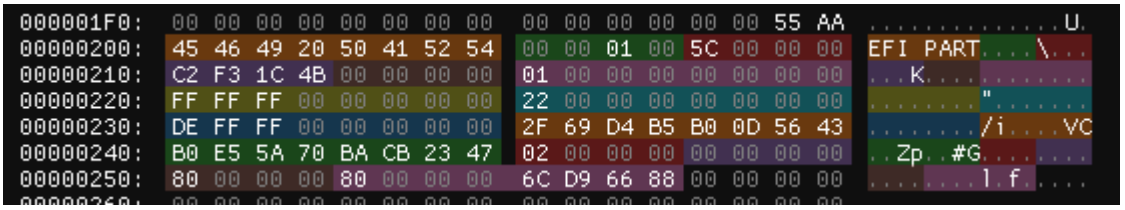
Can you identify the file system type for each partition and extract the files that are contained?

```
img_stat microsoft
IMAGE FILE INFORMATION
--------------------------------------------
Image Type: raw

Size in bytes: 8588869120
Sector size:    512
```

It is 8GB of size, as the long extraction time already hinted...

```
mmls strange.dd
GUID Partition Table (EFI)
Offset Sector: 0
Units are in 512-byte sectors

      Slot      Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Safety
Table
001:  -------   0000000000   0000002047   0000002048   Unallocated
002:  Meta      0000000001   0000000001   0000000001   GPT Header
003:  Meta      0000000002   0000000033   0000000032   Partition
Table
004:  000       0000002048   0016777182   0016775135   Microsoft
basic data
005:  -------   0016777183   0016777215   0000000033   Unallocated

dd if=strange.dd of=microsoft bs=512 skip=2048 count=16775135
```

I analyze the GPT header with ImHex with a pattern of mine: [GPT Pattern].



I have two entries in the GPT and also a protective MBR part pointed out by the 55AA before the GPT header part.

While investigating about the content I see that *fsstat* nor *fls* can't recognize the file system used, but EXT or FAT are suggested, I tried both combinations:

```
fsstat microsoft
Cannot determine file system type (EXT2/3/4 or FAT)

fsstat -f ext microsoft
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: Ext3
Volume Name: ext3label
Volume ID: 965484d00281318b2241ce90b63aa566

Last Written at: 2022-01-28 09:42:38 (CET)
Last Checked at: 2022-01-28 08:26:40 (CET)

Last Mounted at: 2022-01-28 09:41:56 (CET)
Unmounted properly
Last mounted on: /dir/dev1

Source OS: Linux
Dynamic Structure
Compat Features: Journal, Ext Attributes, Resize Inode, Dir Index
InCompat Features: Filetype,
Read Only Compat Features: Sparse Super, Large File,

Journal ID: 00
Journal Inode: 8

METADATA INFORMATION
--------------------------------------------
Inode Range: 1 - 524289
Root Directory: 2
Free Inodes: 524274

CONTENT INFORMATION
--------------------------------------------
Block Range: 0 - 2096890
Block Size: 4096
Free Blocks: 2027400

BLOCK GROUP INFORMATION
--------------------------------------------
Number of Block Groups: 64
Inodes per group: 8192
Blocks per group: 32768

Group: 0:
```

```
  Inode Range: 1 - 8192
  Block Range: 0 - 32767
  Layout:
    Super Block: 0 - 0
    Group Descriptor Table: 1 - 1
    Data bitmap: 513 - 513
    Inode bitmap: 514 - 514
    Inode Table: 515 - 1026
    Data Blocks: 1027 - 32767
  Free Inodes: 8181 (99%)
  Free Blocks: 0 (0%)
  Total Directories: 2

Group: 1:
  Inode Range: 8193 - 16384
  Block Range: 32768 - 65535
  Layout:
    Super Block: 32768 - 32768
    Group Descriptor Table: 32769 - 32769
    Data bitmap: 33281 - 33281
    Inode bitmap: 33282 - 33282
    Inode Table: 33283 - 33794
    Data Blocks: 33795 - 65535
  Free Inodes: 8192 (100%)
  Free Blocks: 31555 (96%)
  Total Directories: 0

Group: 2:
  Inode Range: 16385 - 24576
  Block Range: 65536 - 98303
  Layout:
    Data bitmap: 65536 - 65536
    Inode bitmap: 65537 - 65537
    Inode Table: 65538 - 66049
    Data Blocks: 65538 - 65537, 66050 - 98303
  Free Inodes: 8192 (100%)
  Free Blocks: 32254 (98%)
  Total Directories: 0

Group: 3:
  Inode Range: 24577 - 32768
  Block Range: 98304 - 131071
  Layout:
    Super Block: 98304 - 98304
    Group Descriptor Table: 98305 - 98305
    Data bitmap: 98817 - 98817
    Inode bitmap: 98818 - 98818
    Inode Table: 98819 - 99330
```

```
    Data Blocks: 99331 - 131071
  Free Inodes: 8189 (99%)
  Free Blocks: 31690 (96%)
  Total Directories: 0
```

I omit other Groups since the Free Inode percentages are equal to 100%.
I have block size of 4096 and 2096890 blocks, so its product gives 8.588.861.440, that is 8 GB.

```
fsstat -f fat microsoft
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT32

OEM Name: mkfs.fat
Volume ID: 0xacd4ced3
Volume Label (Boot Sector): FAT32LABEL
Volume Label (Root Directory): FAT32LABEL
File System Type Label: FAT32
Next Free Sector (FS Info): 4295003172
Free Sector Count (FS Info): 0

Sectors before file system: 2048

File System Layout (in sectors)
Total Range: 0 - 4193782
* Reserved: 0 - 34867
** Boot Sector: 0
** FS Info Sector: 1
** Backup Boot Sector: 0
* FAT 0: 34868 - 35891
* Data Area: 35892 - 4193782
** Cluster Area: 35892 - 4193779
*** Root Directory: 35892 - 35899
** Non-clustered: 4193780 - 4193782

METADATA INFORMATION
--------------------------------------------
Range: 2 - 266105029
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 2048
Cluster Size: 16384
Total Cluster Range: 2 - 519737
```

```
FAT CONTENTS (in sectors)
-----------------------------------------
35892-35899 (8) -> EOF
35900-36059 (160) -> EOF
36060-36227 (168) -> EOF
36228-36379 (152) -> EOF
```

I have sector size of 2048 and 4193782 sectors, so its product gives 8.588.865.536, that is 8 GB again. Since the original image is 8GB I can't have two distinct filesystems of 8GB each, maybe one is embedded in another.

Analyzing the contents

```
fls -f fat microsoft
r/r 3:  FAT32LABEL  (Volume Label Entry)
r/r 6:  fat32_nashorn_1.jpg
r/r 9:  fat32_nashorn_2.jpg
r/r 12: fat32_nashorn_3.jpg
v/v 266105027:  $MBR
v/v 266105028:  $FAT1
V/V 266105029:  $OrphanFiles
```

Extract pictures:

```
icat -f fat microsoft 6 > nashorn_1.jpg
icat -f fat microsoft 9 > nashorn_2.jpg
icat -f fat microsoft 12 > nashorn_3.jpg
```

And then:

```
fls -f ext microsoft
d/d 11: lost+found
r/r 24577:      ext3_nashorn_1.jpg
r/r 24578:      ext3_nashorn_2.jpg
r/r 24579:      ext3_nashorn_3.jpg
V/V 524289:     $OrphanFiles
```

Extract pictures:

```
icat -f ext microsoft 24577 > enashorn_1.jpg
icat -f ext microsoft 24578 > enashorn_2.jpg
icat -f ext microsoft 24579 > enashorn_3.jpg
```

They are images of rhinos. No difference between the images per filesystem type, just different names.

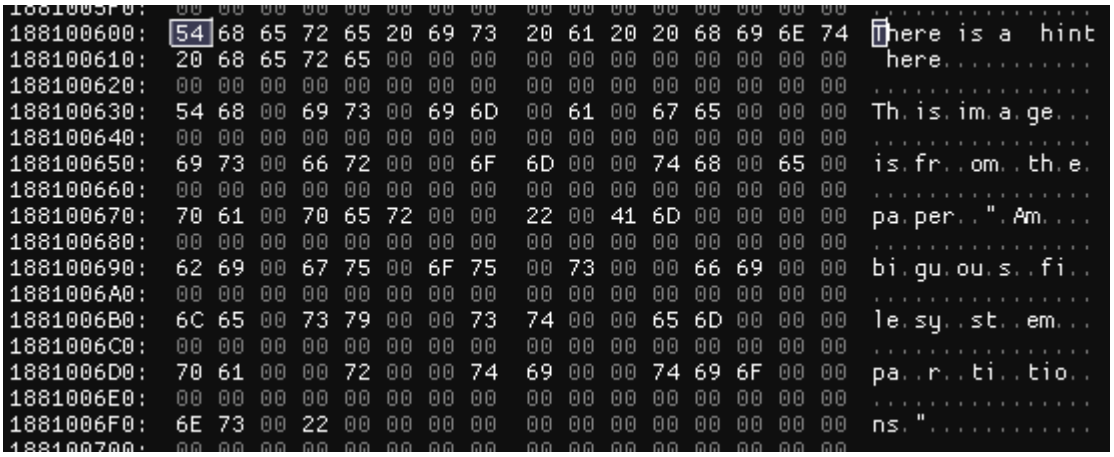
nashorn_1.jpg
nashorn_2.jpg
nashorn_3.jpg

Then I've checked for hints in the pictures, but nothing appears:
`strings nashorn_*.jpg | grep hint`

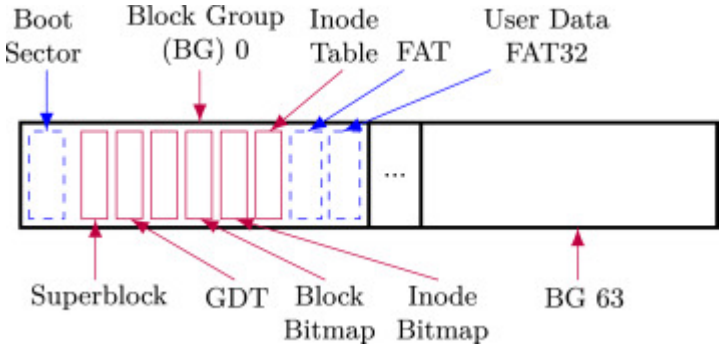
What is so strange about this image?
The hypothesis was about an embedding of some kind of the two filesystems, but after I searched for hints it was much clearer.

```
strings --radix=d strange.dd | grep hint
6577718784 There is a  hint here
```

Searching the offset with ImHex



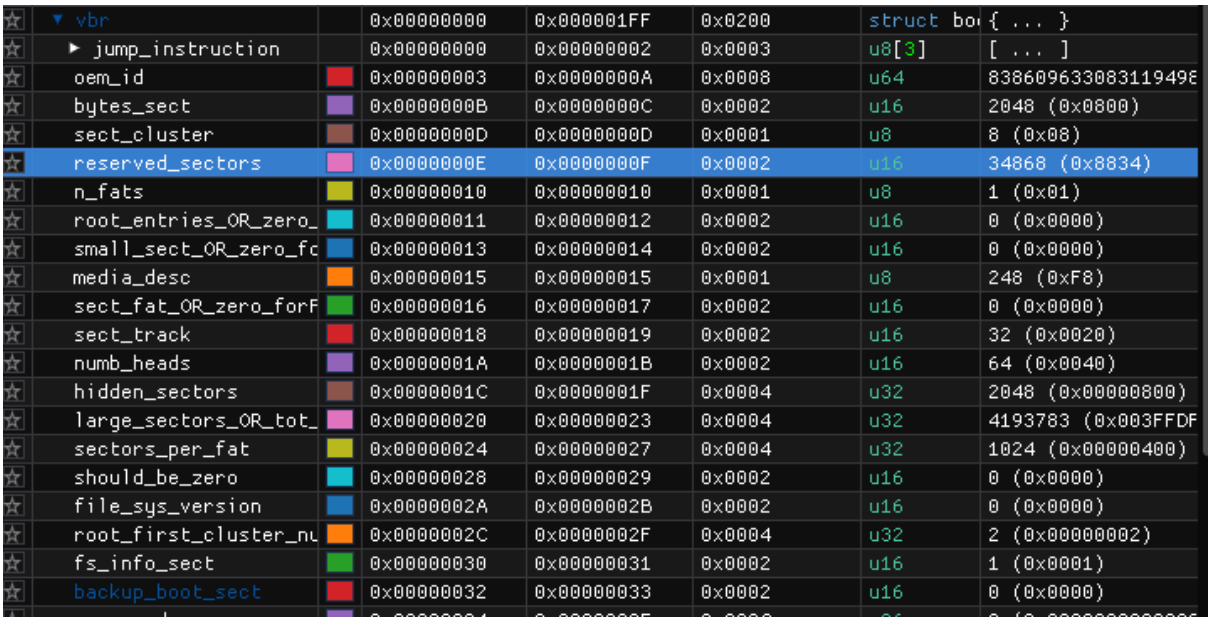
It was referring to the following article:
https://www.sciencedirect.com/science/article/pii/S2666281722000804



This figure is from the article and shows the work done on the image

From the article I can conclude that the file system type used are both **EXT3** and **FAT32**, via **overlapping**, where the EXT3 is used as host filesystem, while FAT32 as guest, so FAT is embedded into EXT. From section 4.2 of the article it can be understood how they could achieve the image configuration.
"The combination of Ext3 and FAT32 appears convenient because the superblock of Ext3 has a fixed offset of 1024 bytes, which **provides enough space for another data structure**

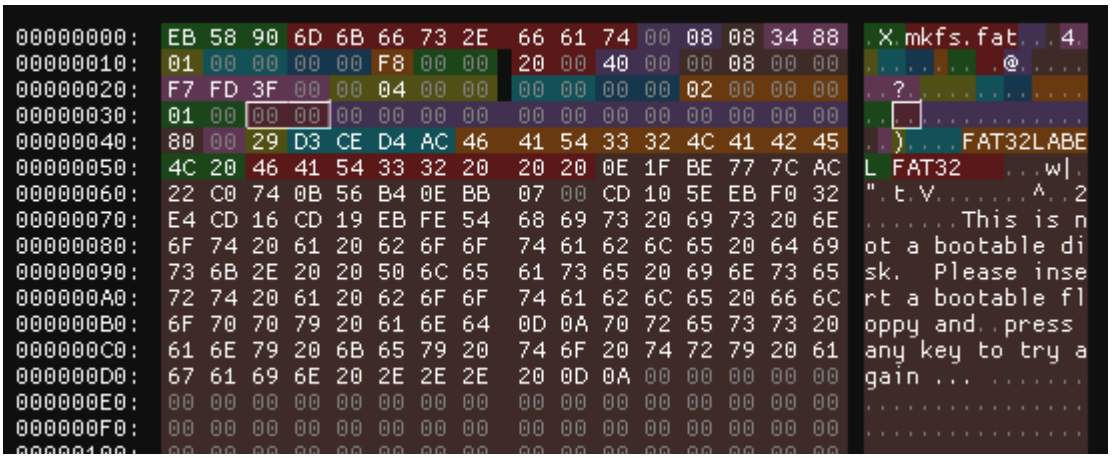
to be placed before. Therefore, the Ext3 file system serves as the host file system for the combination with FAT32."
“[...] However, some adjustments still had to be made. First, the FAT32 boot sector had to be edited. For simplification the backup boot sector position value is set to 0x00 (indicating no backup boot sector). Furthermore, the size of the reserved area had to be adjusted. The offset of 17434 blocks (Ext3) corresponds to **34868** sectors (FAT32) resulting in a size of 0x8834. If the sector size is too small, this number is too large to be stored in the corresponding data structure, since only two bytes are available for the value.”
This can be verified on ImHex, by applying a pattern of mine: [FAT32 Pattern].



▼ vbr	0x00000000	0x000001FF	0x0200	struct bo	{ ... }
▶ jump_instruction	0x00000000	0x00000002	0x0003	u8[3]	[...]
oem_id	0x00000003	0x0000000A	0x0008	u64	838609633083119498
bytes_sect	0x0000000B	0x0000000C	0x0002	u16	2048 (0x0800)
sect_cluster	0x0000000D	0x0000000D	0x0001	u8	8 (0x08)
reserved_sectors	0x0000000E	0x0000000F	0x0002	u16	34868 (0x8834)
n_fats	0x00000010	0x00000010	0x0001	u8	1 (0x01)
root_entries_OR_zero_	0x00000011	0x00000012	0x0002	u16	0 (0x0000)
small_sect_OR_zero_fc	0x00000013	0x00000014	0x0002	u16	0 (0x0000)
media_desc	0x00000015	0x00000015	0x0001	u8	248 (0xF8)
sect_fat_OR_zero_forF	0x00000016	0x00000017	0x0002	u16	0 (0x0000)
sect_track	0x00000018	0x00000019	0x0002	u16	32 (0x0020)
numb_heads	0x0000001A	0x0000001B	0x0002	u16	64 (0x0040)
hidden_sectors	0x0000001C	0x0000001F	0x0004	u32	2048 (0x00000800)
large_sectors_OR_tot_	0x00000020	0x00000023	0x0004	u32	4193783 (0x003FFDF
sectors_per_fat	0x00000024	0x00000027	0x0004	u32	1024 (0x00000400)
should_be_zero	0x00000028	0x00000029	0x0002	u16	0 (0x0000)
file_sys_version	0x0000002A	0x0000002B	0x0002	u16	0 (0x0000)
root_first_cluster_nu	0x0000002C	0x0000002F	0x0004	u32	2 (0x00000002)
fs_info_sect	0x00000030	0x00000031	0x0002	u16	1 (0x0001)
backup_boot_sect	0x00000032	0x00000033	0x0002	u16	0 (0x0000)

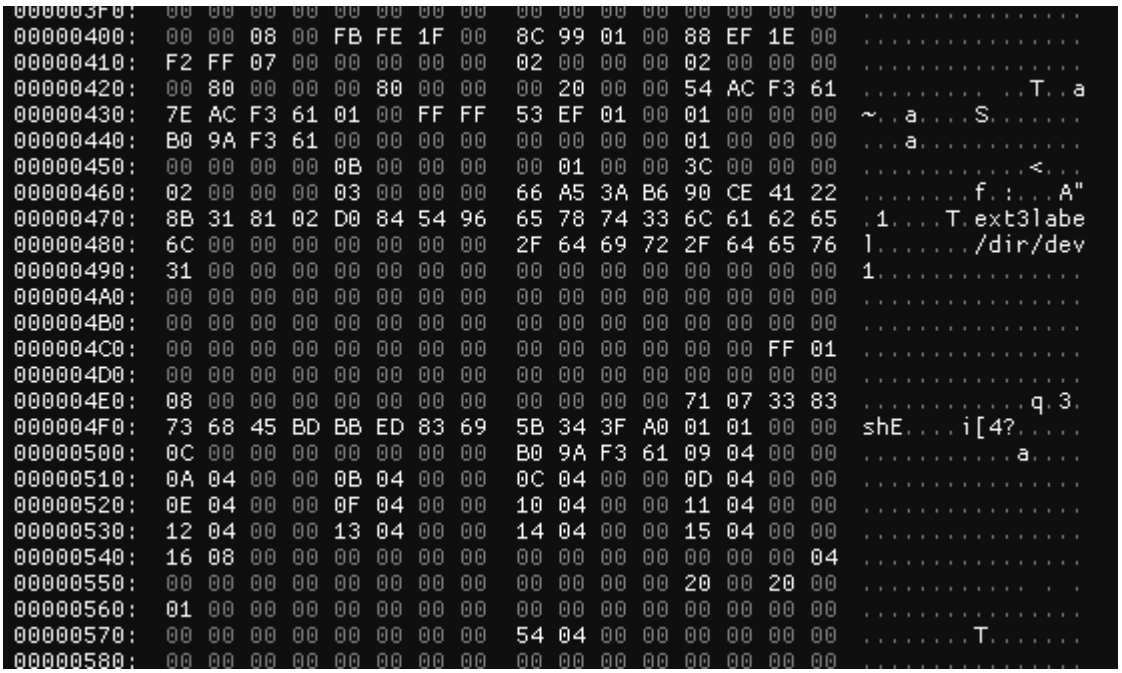
The highlighted “**reserved_sectors**” and “**backup_boot_sect**” represent the value pointed out by the article, respectively size of **0x8834** and **0x0000** as backup boot sector position value.

The overlapping can also be seen via ImHex:



```
00000000:  EB 58 90 6D 6B 66 73 2E  66 61 74 00 08 08 34 88   .X.mkfs.fat...4.
00000010:  01 00 00 00 00 F8 00 00  20 00 40 00 00 08 00 00   ........ .@.....
00000020:  F7 FD 3F 00 00 04 00 00  00 00 00 00 02 00 00 00   ..?.............
00000030:  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000040:  80 00 29 D3 CE D4 AC 46  41 54 33 32 4C 41 42 45   ..)....FAT32LABE
00000050:  4C 20 46 41 54 33 32 20  20 20 0E 1F BE 77 7C AC   L FAT32   ...w|.
00000060:  22 C0 74 0B 56 B4 0E BB  07 00 CD 10 5E EB F0 32   ".t.V.......^..2
00000070:  E4 CD 16 CD 19 EB FE 54  68 69 73 20 69 73 20 6E   .......This is n
00000080:  6F 74 20 61 20 62 6F 6F  74 61 62 6C 65 20 64 69   ot a bootable di
00000090:  73 6B 2E 20 20 50 6C 65  61 73 65 20 69 6E 73 65   sk.  Please inse
000000A0:  72 74 20 61 20 62 6F 6F  74 61 62 6C 65 20 66 6C   rt a bootable fl
000000B0:  6F 70 70 79 20 61 6E 64  0D 0A 70 72 65 73 73 20   oppy and..press
000000C0:  61 6E 79 20 6B 65 79 20  74 6F 20 74 72 79 20 61   any key to try a
000000D0:  67 61 69 6E 20 2E 2E 2E  20 0D 0A 00 00 00 00 00   gain ... .......
000000E0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000000F0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

First sector with FAT32 header



Then the EXT3 header

"Furthermore, the blocks used for the FAT and the actual user data had to be marked as allocated in Ext3. **Therefore, all data blocks of block group 0 were marked as allocated in the allocation bitmap of block group 0.**"
In fact, what we have already seen is that Group 0 has been marked with all blocks allocated (also the one that the ext doesn't occupy with superblock, GDT etc.), from the previous *fsstat* command:

```
Group: 0:
  Inode Range: 1 - 8192
  Block Range: 0 - 32767
  Layout:
    Super Block: 0 - 0
    Group Descriptor Table: 1 - 1
    Data bitmap: 513 - 513
    Inode bitmap: 514 - 514
    Inode Table: 515 - 1026
    Data Blocks: 1027 - 32767
  Free Inodes: 8181 (99%)
  Free Blocks: 0 (0%)
  Total Directories: 2
```

"A GUID Partition Table (GPT) with one entry was created on both hard disks with gdisk". That's because it shows **two** entries in the GPT header.

Moreover, as reported in the extract from section 5.2.2 of the article here, we have some more confirmations:
"For all three combinations **The Sleuth Kit showed the message cannot determine file system type** (<file system A> or <file system B>). The Sleuth Kit is therefore **unable to**

determine the file system but gives the user the possibility to choose from the most probable file systems. If one of the offered file systems is then passed as an option to The Sleuth Kit, all file system data is displayed correctly."
[...]
"For read-only operations, we expect the OS functions should work without any intervention on both the guest and host file system."
That is exactly what happened when the -f fat or -f ext option was passed, so the OS shows one file system independent from the other without any problem.