Digital Forensics

Federico Conti

2024/25

Contents

console.dd

The purpose of this analysis is to investigate a provided disk image `console.dd`, which appears to be an unpartitioned FAT filesystem containing only a single JPEG file. There is suspicion that the volume was recently reformatted to conceal prior data. The goal is to reconstruct the original partition scheme and recover as much of the original content as possible.

Initial Setup

```
diff console.dd.sha256 <(sha256sum console.dd)

file console.dd
# Output:
 console.dd: DOS/MBR boot sector, code offset 0x3c+2, OEM-ID "mkfs.fat", sectors/cluster 4,
 root entries 512, sectors 8192 (volumes <=32 MB), Media descriptor 0xf8, sectors/FAT 6,
 sectors/track 32, serial number 0xb9e28db8, unlabeled, FAT (12 bit)

fdisk -l console.dd
# Output:
    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

sudo mount -oro console.dd /mnt/console1

strings -e S xbox.jpg | hexdump
# Output:
    00000000  ff d8 ff e0 0a 4a 46 49  46 0a 32 22 33 2a 37 25  |.....JFIF.2"3*7%|

# SHA xbox.jpg
    88f83bef7713f5e38aa59da9b71ec53081fe373593758879a4ce06a658ccead0  xbox.jpg
```

- The image is detected as a FAT12 filesystem.
- No active partitions were listed, reinforcing that the current filesystem is unpartitioned.
- Upon mounting, only a single JPEG file (`xbox.jpg`) was present.
- The JPEG magic number (`FFD8 FFE0`) confirms the file type.

At this stage, the disk image appears as an unpartitioned FAT12 filesystem with a single valid JPEG file. However, further analysis is required to detect remnants of any previous partitioning.

Partition Scheme Identification

```
mmstat console.dd
# Output:
    gpt

mmls console.dd
# Output:
    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
```

```
img_stat console.dd
# Output:
    IMAGE FILE INFORMATION
    --------------------------------------------
    Image Type: raw

    Size in bytes: 4194304
    Sector size:    512
```

Surprisingly,- `mmstat` detected traces of a GUID Partition Table (GPT), which is inconsistent with a simple FAT12 format. This indicates remnants of a previous partition scheme.

Findings

- A previous GPT structure is partially detectable.
- There's evidence of a Linux filesystem starting at sector 2048.
- The presence of unallocated spaces and partition table metadata confirms that the disk was likely reformatted over an existing partitioned structure.

Analysing the disk image with ImHex revealed further information on the GPT structure:

- It appears that the disk was quickly reformatted to Logical Block Addressing (LBA) 0 with a FAT12 filesystem. This action overwritten the primary GPT header, rendering it partially corrupted and unable to identify the original partition entries.
- Despite this, remnants of the GPT structure are still detectable, suggesting that the disk previously contained a more complex partition scheme.

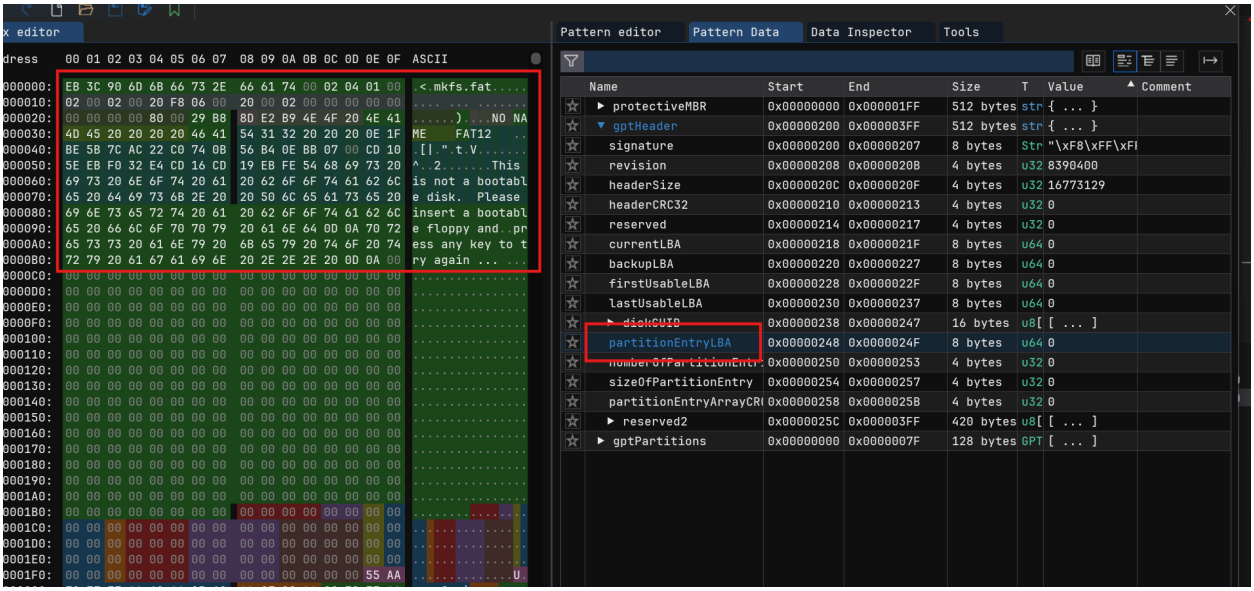

Figure 1: fake FAT12 partiotion wrap GPT protective MBR

Given:

1. The disk image size is 4,194,304 bytes.
2. The sector size is 512 bytes.

We confirm there are exactly 8192 sectors. According to the GPT standard, the backup GPT header resides at the last sector (LBA 8191).

Below is the Pattern used to identify the Secondary GPT Scheme

```
struct PartitionEntry {
    u8  bootIndicator;
    u8  startCHS[3];
    u8  partitionType;
    u8  endCHS[3];
    u32 relativeSectors;
    u32 totalSectors;
};

struct GPTHeader {
    char signature[8];
    u32  revision;
    u32  headerSize;
    u32  headerCRC32;
    u32  reserved;
    u64  currentLBA;
    u64  backupLBA;
    u64  firstUsableLBA;
    u64  lastUsableLBA;
    u8   diskGUID[16];
    u64  partitionEntryLBA; // redirected to 0
    u32  numberOfPartitionEntries;
    u32  sizeOfPartitionEntry;
    u32  partitionEntryArrayCRC32;
    u8   reserved2[420];
};

struct GPTPartitionEntry {
    u8  partitionTypeGUID[16];
    u8  uniquePartitionGUID[16];
    u64 firstLBA;
    u64 lastLBA;
    u64 flags;
    u16 partitionName[36];
};

//SECONDARY

// LBA = fdisk -l console.dd
GPTHeader Secondary_GPTHeader @ (8191 * 512);
GPTPartitionEntry Secondary_gptPartitions[128] @ (Secondary_GPTHeader.partitionEntryLBA * 512);
```

Result of GPT Analysis

- The secondary GPT header was successfully located at LBA 8191.
- The partition entries were parsed, revealing that:

  Entry 2 defines a partition of type Linux File System. This partition starts at sector 2048, consistent with previous findings from `mmls`.

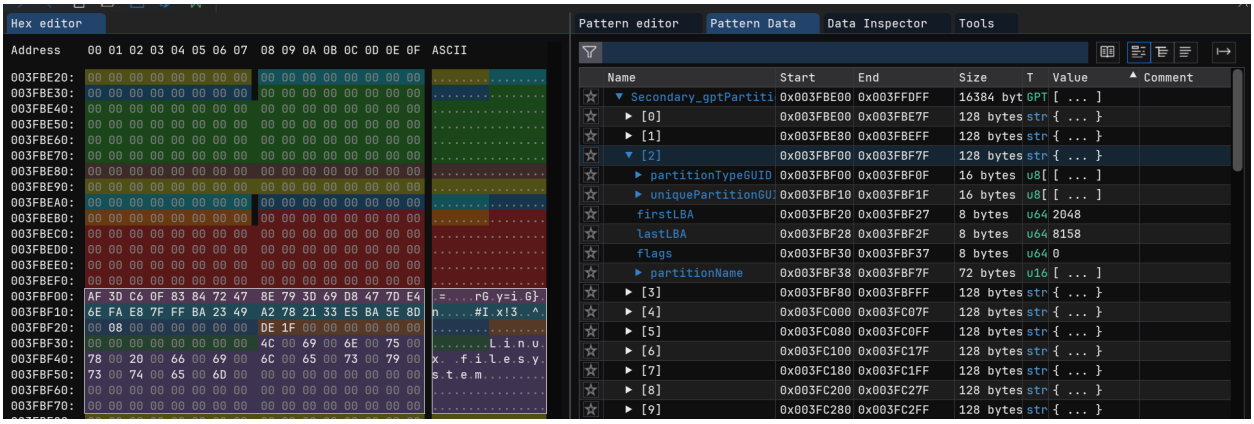

Figure 2: find original partition in third GPT backup entry

Data Recovery

The partition was identified as NTFS, confirming that the original system was likely Microsoft-based.

```
fsstat -o 2048 console.dd
# Output:
    File System Type: NTFS
    Version: Windows XP
    Cluster Size: 4096
    Total Sector Range: 0 - 6109
```

Upon inspection of the mounted partition, a file named ps5.jpg was discovered. The file ps5.jpg is a valid JPEG image, confirming successful recovery of at least part of the original data stored prior to the reformatting attempt.

```
dd if=console.dd of=ntfs.dd bs=512 skip=2048 count=6110

mount -oro  ntfs.dd /mnt/console1
# Output:
    file ps5.jpg
    ps5.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI), density 72x72, segment length 16, ba
# SHA ps5.jpg
    200ff11c196aeeaecd7a4021aa40a47459a252b5776ecdd3104f2e5537eb75a2  ps5.jpg
```

Advanced File System Analysis

To ensure a thorough investigation, advanced forensic tools were employed to enumerate files, recover orphaned data, and extract detailed metadata from the NTFS Master File Table (MFT).

```
fls -rp console.dd -o 2048
#
r/r 4-128-1:    $AttrDef
```

```
...
r/r 0-128-1:    $MFT
...
r/r 64-128-2:   ps5. # uniquely identifies the file or directory within the Master File Table.

V/V 65: $OrphanFiles
-/r * 16:       $OrphanFiles/OrphanFile-16
...
```

Presence of multiple orphaned files under $OrphanFiles, suggesting incomplete deletions or filesystem inconsistencies prior to the reformat.

To recover any residual files not linked within the filesystem, foremost was executed:

```
foremost -i console.dd -o recover/
```

Result:

- The carving process did not detect any deleted files.
- All files recovered by foremost were consistent with those already identified through filesystem analysis (ps5.jpg and xbox.jpg).
- No additional user files, fragments, or hidden data were found beyond the active filesystem entry

```
88f83bef7713f5e38aa59da9b71ec53081fe373593758879a4ce06a658ccead0  00000049.jpg
200ff11c196aeeaecd7a4021aa40a47459a252b5776ecdd3104f2e5537eb75a2  00006128.jpg
```

For detailed file metadata, the NTFS Master File Table (MFT) was extracted and analyzed using specialized tools.

```
icat -r -o 2048 console.dd 0 > MFT.bin
MFTECmd.exe -f MFT.bin --csv .\ --csvf console_mft.csv
```

- The file ps5.jpg was confirmed as an active file (InUse=True) with a creation and modification date of April 11, 2023.
- No Alternate Data Streams (ADS) or special attributes were detected. (parsing with TimelineExplorer.exe)

corrupted.dd

This section details the forensic analysis conducted on the disk image `corrupted.dd`. The objective was to investigate the file system structure, identify signs of corruption, recover inaccessible data, and locate specific string patterns within the image.

Partition Scheme Identification

```
file corrupted.dd
# Output:
    DOS/MBR boot sector, code offset 0x3c+2, OEM-ID "mkfs.fat", Bytes/sector 2048, FATs 3,
    root entries 512, sectors 720 (volumes <=32 MB), Media descriptor 0xf8, sectors/FAT 1,
    sectors/track 16, serial number 0xc8269037, label: "BILL", FAT (12 bit)

fsstat corrupted.dd
# Output:
    ...
    File System Type: FAT12
    OEM Name: mkfs.fat
    Volume ID: 0xc8269037
    Volume Label (Boot Sector): BILL
    Volume Label (Root Directory): BILL
    File System Type Label: FAT12
    ...
    File System Layout (in sectors)
    Total Range: 0 - 719
    * Reserved: 0 - 0
    ** Boot Sector: 0
    * FAT 0: 1 - 1
    * FAT 1: 2 - 2
    * FAT 2: 3 - 3
    * Data Area: 4 - 719
    ** Root Directory: 4 - 11
    ** Cluster Area: 12 - 719

    METADATA INFORMATION
    ------------------------------------------
    Range: 2 - 45831
    Root Directory: 2

    CONTENT INFORMATION
    ------------------------------------------
    Sector Size: 2048
    Cluster Size: 2048
```

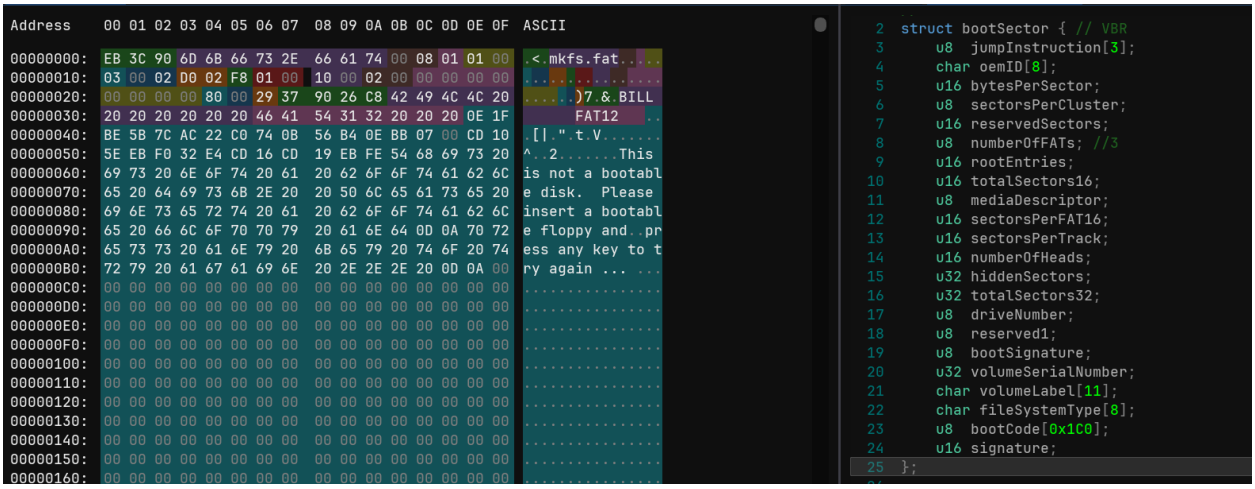

Figure 3: analysis of the FAT filesystem

A sector 0 directly contains a FAT12 Boot Sector, so there is no MBR/GPT table listing partitions, and it is not a bootable image.

- **Volume Label**: `BILL`
- **Sector Size**: 2048
- **Cluster Size**: 2048
- **Num FATs**: 3

Analysis Process

Using The Sleuth Kit (TSK) to inspect the file system, three `.TXT` files were identified:

```
fls -r -p corrupted.dd | grep '\.TXT'
# Output:
    r/r 45: HOMEWORK.TXT
    r/r 32: NETWORKS.TXT
    r/r * 36:       _EADME.TXT
```

HOMEWORK.TXT (pseudo-inode 45)

Status: Allocated
Size: 6 bytes
Sector: 619
Readability:
Can read it both by using `icat` and by mounting the image.

NETWORKS.TXT (pseudo-inode 32)

Status: Allocated
Size: 17,465 bytes
Starting Sector: 345
Readability:
Cannot read it by mounting the image, but you can read it using `icat`.
Explanation: This indicates that the mounted file system has issues following the cluster chain, likely due to corruption in the FAT.

EADME.TXT (pseudo-inode 36)

Status: Deleted
Size: 60,646 bytes
Sectors: 457 to 486
Readability:
Since this file is deleted, it is expected not to appear in the mounted file system. However, you can recover it using `icat` if the data has not been overwritten.

Using TSK, detailed metadata about the `NETWORKS.TXT` file associated with pseudo-inode 32 was identified.

```
istat corrupted.dd 32
# Output:
    Directory Entry: 32 #pseudo inode by TSK
    Allocated
    File Attributes: File, Archive
    Size: 17465
    Name: NETWORKS.TXT
    ...
    Sectors: 345
```

Fix FAT Table

Rebuild the cluster chain in the FAT for corrupted files, particularly for `NETWORKS.TXT`.

Analyzing the first FAT it was discovered that FAT0 was overwritten with non-FAT data, likely a fragment of a GIF file.

```
xxd -s $((2048)) -l 2048 corrupted.dd | less
```

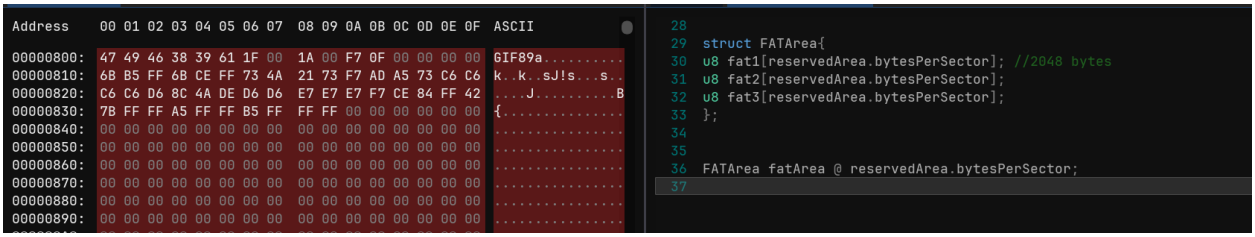

Figure 4: analysis of the first FAT

A known good copy of FAT2 was used to restore FAT0:

```
cp corrupted.dd corrupted_fixed.dd
dd if=corrupted_fixed.dd of=corrupted_fixed.dd bs=2048 skip=3 seek=1 count=1 conv=notrunc
```

After fixing FAT:

- The cluster chain has been rebuilt.
- Both `.TXT` files can now be mounted correctly, allowing the recovery of the hash for `NETWORKS.TXT`.

```
sha256sum *.TXT
9b4a458763b06fefc65ba3d36dd0e1f8b5292e137e3db5dea9b1de67dc361311  HOMEWORK.TXT
e9207be4a1dde2c2f3efa3aeb9942858b6aaa65e82a9d69a8e6a71357eb2d03c  NETWORKS.TXT
```

zxgio

Inside the file `corrupted.dd`, there are some occurrences of the string `zxgio` (without quotes). Below is the analysis:

```
strings -t d corrupted.dd | grep -E zxgio
    512 zxgio
   2832 zxgio
 724025 zxgio
1267712 zxgio
```

From fsstat on intact image:

Offset (byte)	Sector	File System Area
512	0	Boot Sector
2832	1	FAT 0 (corrupted)
724025	353	Cluster Area (slack space)
1267712	619	Cluster Area (inside `HOMEWORK.TXT` 619-619 (1) -> EOF)

1. **Verify sector 353**:
   - Sector 353 contained the string in slack space (confirmed via dd and xxd).
   - Offset 1,267,712 confirmed within HOMEWORK.TXT.
   - No occurrence found within actual data of NETWORKS.TXT.

```
istat corrupted_fixed.dd 32
# Output:
    Directory Entry: 32
    Size: 17465
    Name: NETWORKS.TXT
    Sectors: 345 346 347 348 349 350 351 352 353

icat corrupted_fixed.dd 32 | grep zxgio
# Output:
    NULL
```

2. **Slack Space Inspection**:
   - It can be confirmed that the string is contained in the slack space of cluster 353

```
dd if=corrupted_fixed.dd bs=2048 skip=353 count=1 of=sector353.bin
xxd sector353.bin | less
# Output:
    tware....zxgio..
```

Unlocated Space

The image `corrupted.dd` has a size of 721 sectors, while the FAT12 file system only uses sectors 0-719. Sector 720, being outside the file system, was extracted and analyzed.

```
dd if=corrupted.dd bs=2048 skip=720 count=1 of=unused_sector720.bin

file unused_sector720.bin
# Output:
  ASCII text

echo "asci text" | base64 -d > hidden_file

file hidden_file
```

```
# Output:
    GIF image data, version 89a, 86 x 33
```

Results:

- The sector contains a long ASCII string without line terminators.
- Analysis revealed it to be Base64 encoding.
- Decoding the string produced a file recognized as a GIF image.

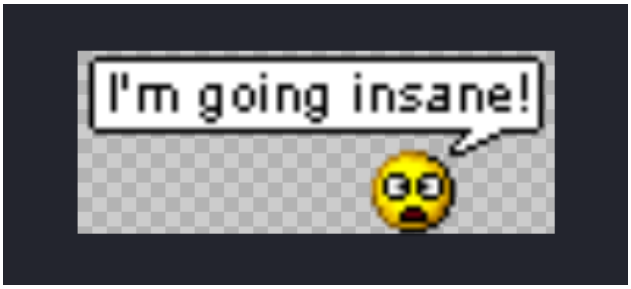

Figure 5: gif found in Unlocated Space

strange.dd

This section analyzes the disk image `strange.dd`, which exhibits an unusual dual file system configuration. The image uses a GPT partition scheme with a single partition labeled as `Microsoft basic data`. Within this partition, both FAT32 and Ext3 file systems coexist, creating an ambiguous setup that challenges traditional forensic tools. The analysis explores the partition scheme, identifies the file systems, and extracts their contents.

Partition Scheme Identification

- The disk image strange.dd uses a GPT partition scheme.
- It includes a Protective MBR and GPT header.
- Only one partition entry is defined in Primary GPT Entries, labeled as `Microsoft basic data`.

```
fdisk -l strange.dd
#Output
    Disk strange.dd: 8 GiB, 8589934592 bytes, 16777216 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: gpt
    Disk identifier: B5D4692F-0DB0-4356-B0E5-5A70BACB2347

mmls strange.dd
#Output
         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
```

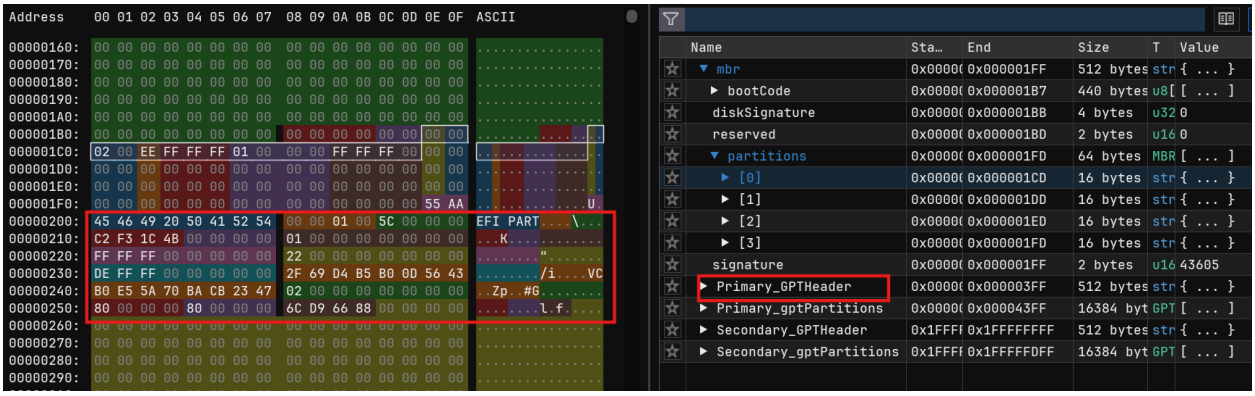

Figure 6: Primary GPT Header

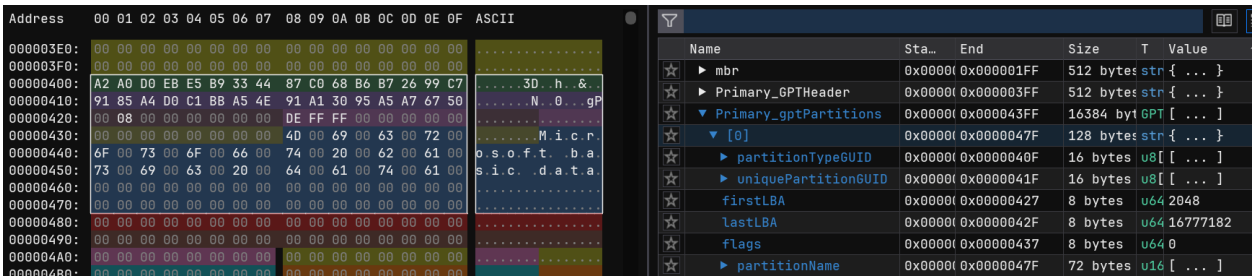

Figure 7: entry[0] in Primary GPT Entries

File System Type Identification

Extract only the partiotion part to better analyze it

`mmcat strange.dd 4 > microsoftdata.dd`

The `disktype` command reveals something unusual about this partition: Both file systems appear to coexist within the same partition space of ~8 GiB.

```
fsstat microsoftdata.dd
#Output
    Multiple file system types detected (EXT2/3/4 or FAT)

disktype microsoftdata.dd
#Output
--- microsoftdata.dd
Regular file, size 7.999 GiB (8588868608 bytes)
    FAT32 file system (hints score 3 of 5)
        Unusual sector size 2048 bytes
        Volume size 7.931 GiB (8515354624 bytes, 519736 clusters of 16 KiB)
        Volume name "FAT32LABEL"
    Ext4 file system
        Volume name "ext3label"
        UUID 66A53AB6-90CE-4122-8B31-8102D0845496 (DCE, v4)
        Last mounted at "/dir/dev1"
        Volume size 7.999 GiB (8588865536 bytes, 2096891 blocks of 4 KiB)
```

Potential Scenarios: - Possible file system-in-file system nesting (e.g., FAT embedded within an EXT partition). - Hidden Volumes: There could be a FAT file system hidden at a specific offset within the EXT3 partition.

ImHex analysis

The analysis confirms the following:

1. Recognizes the FAT32 boot sector at the start of the partition.
    - FAT32 characteristics:
        - Sector size: 2048 bytes.
        - Contains only 1 FAT table instead of the standard 2.
        - No backup boot sector is present.
        - Volume label: "FAT32LABEL".
2. An Ext3 SuperBlocks file system resides in the 1024 offset:

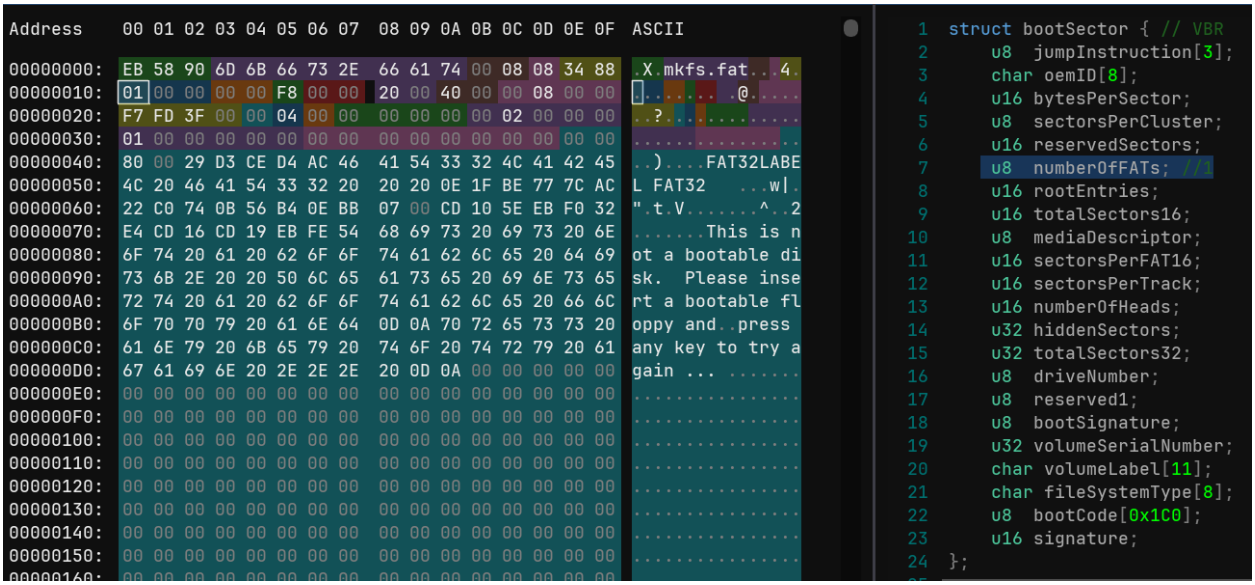

Figure 8: FAT boot sector

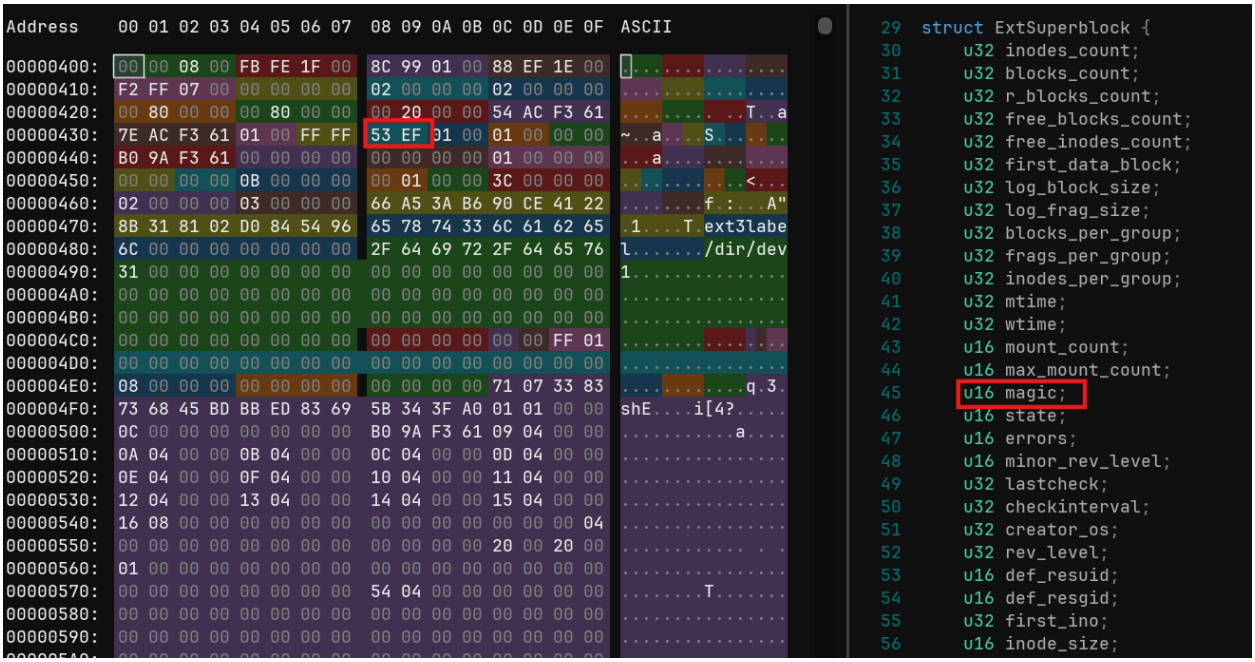

Figure 9: Ext3 First SuperBlock

This dual file system setup demonstrates forensic techniques (TSK), making it challenging to identify and analyze the true contents of the disk image.

After some attempts to get some hints from strings, here's probably the more informative combination of parameters:

```
strings --radix=d microsoftdata.dd | grep -i -E 'ext3|hint|jpg'

#Output
   1144 ext3label
```

```
4206644 ext3_nashorn_1.jpg
4206672 ext3_nashorn_2.jpg
4206700 ext3_nashorn_3.jpg
4268084 ext3_nashorn_1.jpg
4268112 ext3_nashorn_2.jpg
4273272 ext3label
4321332 ext3_nashorn_1.jpg
4321360 ext3_nashorn_2.jpg
4321388 ext3_nashorn_3.jpg
73506912 FAT32_~1JPG
73507008 FAT32_~2JPG
73507104 FAT32_~3JPG
134217848 ext3label
402653304 ext3label
671088760 ext3label
939524216 ext3label
1207959672 ext3label
3355443320 ext3label
3623878776 ext3label
6576668792 ext3label
6576670208 There is a  hint here
```

Analysing the HINT

```
xxd -s 6576670208 -l 512 microsoftdata.dd
```

This is the suggested paper: **4.2. Example B: Ext3 and FAT32**

An **Ambiguous File System Partition** is a deliberately crafted partition where show:

- it is possible to create ambiguous file system partitions by integrating a guest file system into the structures of a host file system: integrating a fully functional FAT32 into Ext3.
- Traditional forensic tools may detect one, both, or even get confused.

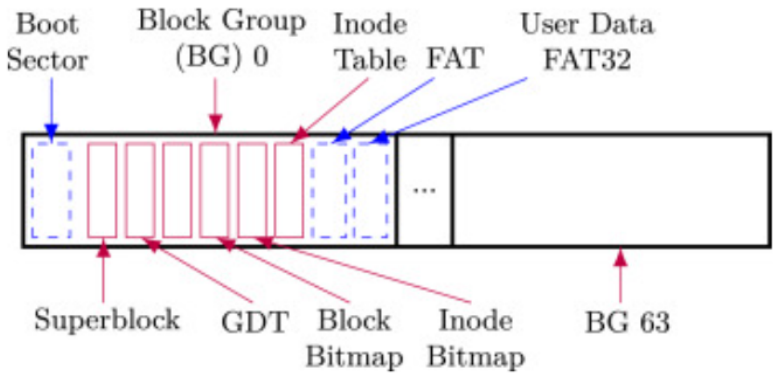

Figure 10: overview of the construction

"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."

This technique demonstrates the challenges in identifying and analyzing the true contents of a disk image, requiring advanced forensic methods to uncover hidden data.

Map Both File Systems Precisely

1. **FAT**

```
fsstat -f fat microsoftdata.dd
#Output
   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 # FAT0
   * 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
```

- size of the FS is 2048 * 4193782 = 8588865536 bytes.

2. **Ext3**

```
fsstat -f ext3 microsoftdata.dd
#Output
   ....
   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
   ....
```

- he FS size is 4096 * 2096890 = 8588861440 bytes.

To protect FAT32 data from being overwritten by the Ext3 file system, the group descriptor table, the superblock (and their copies) and the respective block bitmaps had to be manipulated. Vice versa, clusters occupied by the Ext3 file system had to be marked as bad in the FAT.

- Ext3 avoids overwriting Fat32 by marking 0 free blocks in the first group.
- all groups have almost all free blocks except for the first one, where there are no free blocks.

Summary

Offset (Hex)	Offset (Dec)	Content
0x00000000	0	FAT32 Boot Sector
0x00000400	1024	Ext3 Superblock
0x00008834	34868	FAT
0x00008C34	35892	FAT32 Data Area Start
0x003FFDF5	4193781	FAT32 Data Area End

File System extraction

```
sudo mount -o loop,ro -t ext3 microsoftdata.dd /mnt/strange/ext3
#Output
```

```
    sha256sum ext3_nashorn_*
    8b79029a06610f29ba1c16e4cd4cf498e196e3a7f67a53efebb32f720f3d472d  ext3_nashorn_1.jpg
    0cb84374324e13606bb22b4164323bb487f9088e4a2cc700673180256174e294  ext3_nashorn_2.jpg
    193067cecbd63195bfab2f3f702cc44ff3c6e6fa8de5335a405fbeb9955c3512  ext3_nashorn_3.jpg

sudo mount -o loop,ro -t vfat microsoftdata.dd /mnt/strange/fat32
#Output
    sha256sum fat32_nashorn_*
    8b79029a06610f29ba1c16e4cd4cf498e196e3a7f67a53efebb32f720f3d472d  fat32_nashorn_1.jpg
    0cb84374324e13606bb22b4164323bb487f9088e4a2cc700673180256174e294  fat32_nashorn_2.jpg
    193067cecbd63195bfab2f3f702cc44ff3c6e6fa8de5335a405fbeb9955c3512  fat32_nashorn_3.jpg
```

Failed attempts

Try to mount the image by jumping directly to the Ext3 superblock FAIL

```
mount -o ro,loop,offset=1024 microsoftdata.dd /mnt/strange
#Output
    mount: /mnt/strange: wrong fs type, bad option, bad superblock on /dev/loop0, m
    missing codepage or helper program, or other error.
        dmesg(1) may have more information after failed mount system call.
```

Let's go look for the backuop superblock and mount with resepct offest also fail:

- To protect FAT32 data from being overwritten by the Ext3 file system, the group descriptor table, the superblock (and their backup) and the respective block bitmaps are manipulated,

```
dumpe2fs microsoftdata.dd | grep -i superblock
#Output
    Primary superblock at 0, Group descriptors at 1-1
    Backup superblock at 32768, Group descriptors at 32769-32769
    ...
    ...
    ...
```

Zeroing out the first 512 bytes allowed successful mounting of the Ext3 file system; however, the FAT32 file system specifications were lost in the process.

```
cp microsoftdata.dd microsoftdata_clean.dd

dd if=/dev/zero of=microsoftdata_clean.dd bs=512 count=1 conv=notrunc

sudo mount -o ro,loop microsoftdata_clean.dd /mnt/strange
```