Digital Forensics 1.0.1

Introduction: Post-mortem Digital Forensics

CIRCL *TLP:WHITE*

info@circl.lu

Edition May 2020

Thanks to:

AusCERT

JISC

Overview

1. Introduction
2. Information
3. Disk Acquisition
4. Disk Cloning / Disk Imaging
5. Disk Analysis
6. Forensics Challenges
7. Bibliography and Outlook

CIRCL
FORENSICS Training

1. Introduction

1.1 Admin default behaviour

- Get operational asap:
  - Re-install
  - Re-image
  - Restore from backup
    $\rightarrow$ Destroy of evidences

- Analyse the system on his own:
  - Do some investigations
  - Run AV
  - Apply updates
    $\rightarrow$ Overwrite evidences
    $\rightarrow$ Create big noise

$\rightarrow$ Negative impact on forensics

1.2 Preservation of evidences

- Finding answers:
  - → System compromised
  - → How, when, why
  - → Malware/RAT involved
  - → Persistence mechanisms
  - → Lateral movement inside LAN
  - → Detect the root cause of the incident
  - → Access sensitive data
  - → Data exfiltration
  - → Illegal content
  - → System involved at all

- Legal case:
  - → Collect & safe evidences
  - → Witness testimony for court

1.2 Preservation of evidences

- CRC not sufficient:
  - Example: Checksum

    $4711 \rightarrow 13$
  - Example: Collision

    $12343 \rightarrow 13$

- Cryptographic hash function:
  - Output always same size
  - Deterministic: if $m = m \rightarrow h(m) = h(m)$
  - 1 Bit change in m $\rightarrow$ max. change in $h(m)$
  - One way function: For $h(m)$ impossible to find m
  - Simple collision resistance: For given $h(m1)$ hard to find $h(m2)$
  - Strong collision resistance: For any $h(m1)$ hard to find $h(m2)$

1.3 Forensics Science

- Classical forensic

  Locard's exchange principle

  `https://en.wikipedia.org/wiki/Locard%27s_exchange_principle`

- Write down everything you see, hear, smell and do

- Chain of custody

  → `https://www.nist.gov/sites/default/files/documents/2017/04/28/Sample-Chain-of-Custody-Form.docx`

- Scope of the analysis

1.4 Forensic disciplines

- Reverse Engineering
- Code-Deobfuscation
- Memory Forensics
  - → `https://www.circl.lu/pub/tr-22/`
  - → `https://www.circl.lu/pub/tr-30/`
- Network Forensics
- Mobile Forensics
- Cloud Forensics
- Post-mortem Analysis
  - → `https://www.circl.lu/pub/tr-22/`
  - → `https://www.circl.lu/pub/tr-30/`

1.5 First Responder: Order of volatility

CPU registers → nanoseconds

CPU cache → nanoseconds

RAM memory → tens of nanoseconds

Network state → milliseconds

Processes running → seconds

Disk, system settings, data → minutes

External disks, backup → years

Optical storage, printouts → tens of ears

→ `https://www.circl.lu/pub/tr-22/`

1.5 First Responder: Be prepared

- Prepare your toolbox
  - Photo camera
  - Flash light, magnifying glasses
  - Labelling device, labels, tags, stickers
  - Toolkit, screwdriver kits
  - Packing boxes, bags, faraday bag
  - Cable kits, write blocker, storage devices
  - Anti-static band, network cables
  - Pens, markers, notepads
    $\rightarrow$ Chain of custody
- USB stick
  - 256 GB USB3
  - File system: exFAT
  - Memory dump: Dumpit
  - FTK Imager Lite
  - Encrypted Disk Detector - Edd

1.5 First Responder: First steps

- Did an incident occure
  - Talk with people
  - Take notes
- Mouse jiggler
- Identify potential evidences
  - Tower, desktop, laptop, tablets
  - Screen, printer, storage media
  - Router, switches, access point
  - Paper, notes, ...
- Powered-on versus powered-off
  - Shutdown: Lost of live data
  - Shutdown: Data on disk modified
  - Pull power: Corrupt file system
  - Live analysis: Modify memory and disk
  - Live analysis: Known good binaries?

1.5 First Responder: Live response

- Memory dump
- Live analysis:
  - → System time
  - → Logged-on users
  - → Open files
  - → Network -connections -status
  - → Process information -memory
  - → Process / port mapping
  - → Clipboard content
  - → Services
  - → Command history
  - → Mapped drives / shares
  - → !!! Do not store information on the subject system !!!
- Image of live system (Possible issues)
- Shutdown and image if possible

1.6 Post-mortem Analysis

- Hardware layer & acquisition
  - Best copy (in the safe)
  - Working copy (on a NAS)
  - Disk volumes and partitions
  - Simple tools: dd, dmesg, mount
- File system layer
  - FAT, NTFS
  - File system timeline
  - Restore deleted files
- Data layer
  - Carving: foremost, scalpel, testdisk/photorec
  - String search

1.7 Post-mortem Analysis

- OS layer
  - Registry
  - Event logs
  - Volume shadow copies
  - Prefetch files
- Application layer
  - AV logs
  - Browser history: IE, firefox, chrome
  - Email
  - Office files & PDFs
- Identify malware
  - TEMP folders
  - Startup folders
  - Windows tasks

1.8 Forensic Distributions

- Commercial
  - EnCase Forensic
  - F-Response
  - Forensic Toolkit
  - Helix Enterprise
  - X-Ways Forensics
  - Magnet Axiom

- Open source tools
  - Kali Linux
  - SANS SIFT
  - Digital Evidence and Forensics Toolkit - DEFT
  - PlainSight
  - Computer Aided INvestigative Environment - CAINE

CIRCL
FORENSICS Training

2. Information

2.1 Data in a binary system

- BIT $\rightarrow$ Binary digit
- Data stored in binary form

  `x Bits --> 01010000011010010110111001100111 --> y Bits`

  Bit x + 2 = 1

  Bit x + 3 = 0

  $\rightarrow$ What information is stored within this data?

- *"..... information is data arranged in a meaningful way for some perceived purpose ....."* $\rightarrow$ Interpretative rules

- Grouping, addressing and interpreting

```
-->   01010000   01101001   01101110   01100111   -->
      --------   --------   --------   --------
-->   Byte 117   Byte 118   Byte 119   Byte 120   -->
```

2.1 Data in a binary system

- Grouping examples:
  - Nibble: `0101 0000 0110 1001 0110 1110 0110 0111`
  - Byte: `01010000 01101001 01101110 01100111`
  - Word: `0101000001101001 0110111001100111`
  - Double Word: `01010000011010010110111001100111`

- Interpreting:
  - Integer: (Signed, Unsigned)
  - Endian: (Big, Little)
  - Floating Point
  - Binary Coded Decimal, Packed BCD
  - Encoding: (ASCII, ISO8859, Unicode 16L, 16B, 32L, 32B)
  - Binary: (ELF, MZ, PE, GIF, JPEG, ZIP, PDF, OLE, ...)
  - ...

2.2 Number Systems

- Decimal:

```
2145
|||| _   5 * 10^0 =       5
|||_ __   4 * 10^1 =      40
||_ ___   1 * 10^2 =     100
|_ ____   2 * 10^3 =   2,000
                      ------
                      2,145
```

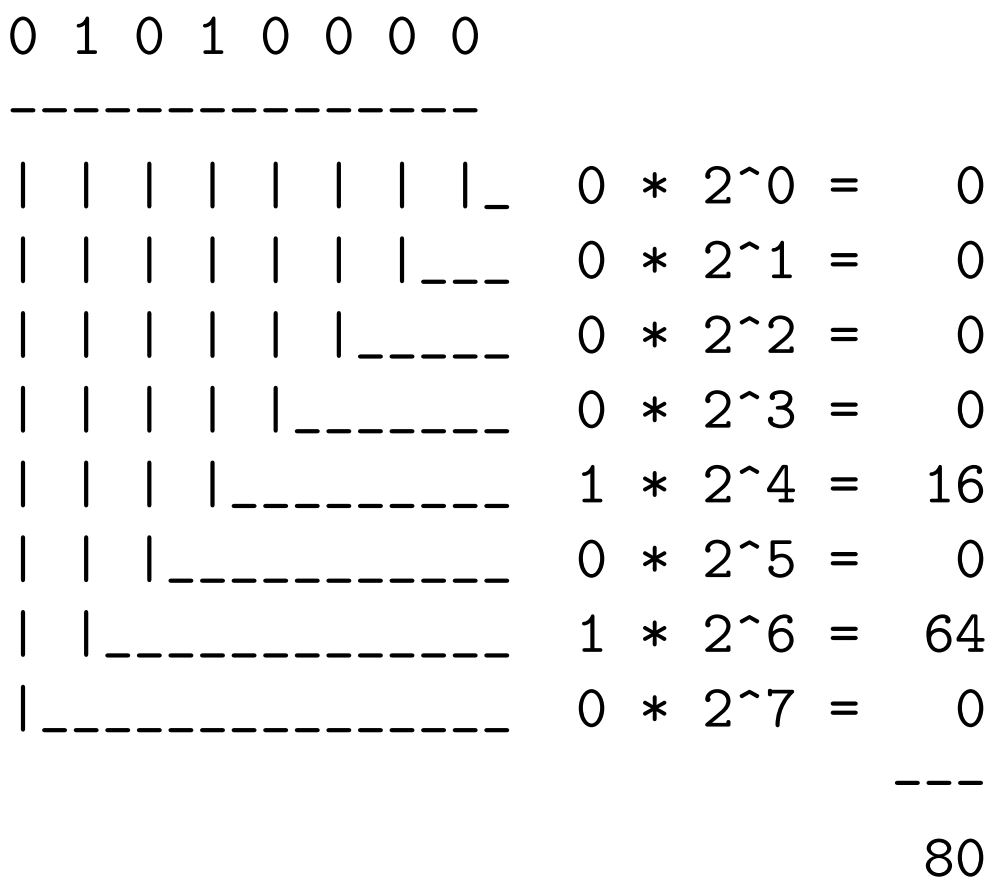
- Hexadecimal:

```
2A9F
|||| _  15 * 16^0 =      15
|||_ __  09 * 16^1 =     144
||_ ___  10 * 16^2 =   2,560
|_ ____  02 * 16^3 =   8,192
                      ------
                      10,911
```

- Binary:

```
1111
|||| _   1 *  2^0 =       1
|||_ __   1 *  2^1 =       2
||_ ___   1 *  2^2 =       4
|_ ____   1 *  2^3 =       8
                        = 15
```

2.3 Interpreting binary data: Integer



2.3 Interpreting binary data: Signed Integer

```
1 0 1 1 1 1 1 1
---------------
```

Two's complement:

```
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 1
```

1. Invert all single bits
2. Add the value 1

```
  |           |

 64           1
 --------------
            -65
```

2.4 Exercise: Signed Integer Bytes

```
1 1 0 1 1 1 0 0
---------------

  | | | | | | |

  ? ? ? ? ? ? ?
 --------------
              -
```

Two's complement:

1. Invert all single bits
2. Add the value 1

2.4 Exercise: Signed Integer Bytes

```
1 1 0 1 1 1 0 0
---------------

0 0 1 0 0 0 1 1
0 0 1 0 0 1 0 0

    |     |

   32     4
 -------------
            -36
```

Two's complement:

1. Invert all single bits
2. Add the value 1

2.5 From Bin to Hex

Example:

0001 1000	0101 0101	0000 1111	1010 0110
1 8	5 5	0 F	A 6

Exercise:

1001 0110	1010 0101	0000 1111	1100 0011

2.5 From Bin to Hex

Exercise:

1001 0110	1010 0101	0000 1111	1100 0011
---------	---------	---------	---------

Results:

1001 0110	1010 0101	0000 1111	1100 0011
---------	---------	---------	---------
9 6	A 5	0 F	C 3

2.6 Big Endian and Little Endian

Large values (Example 256): Big Endian representation:

2^:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Address:	10.000								10.001							

Large values (Example 256): Little Endian representation:

2^:	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Address:	10.000								10.001							

2.6 Exercise: Little Endian

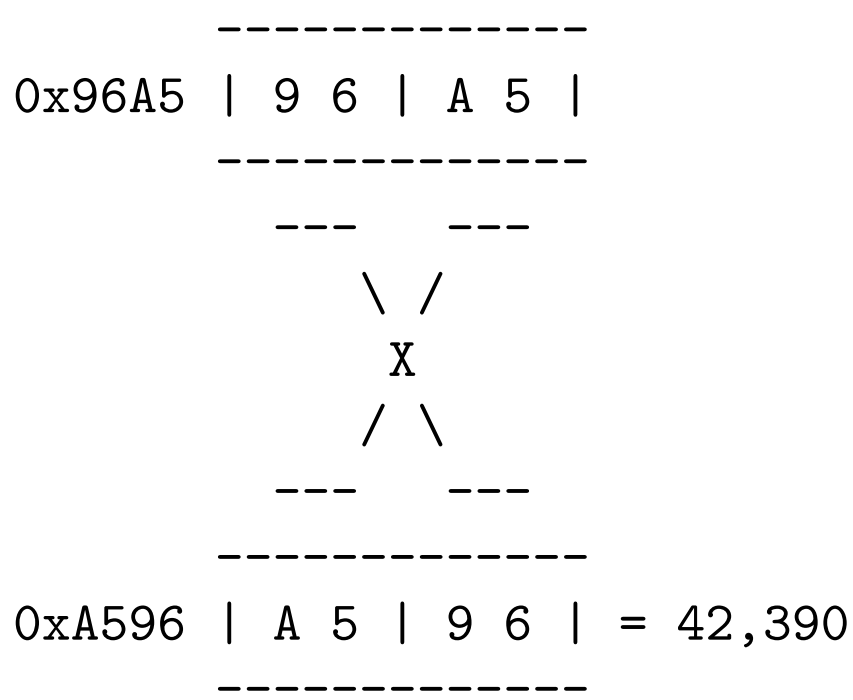
Read and interpret this little endian 'word'

```
            -------------
0x96A5 | 9 6 | A 5 |
            -------------
```

```
            -------------
0x      |      |      | =
            -------------
```

2.6 Exercise: Little Endian

Read and interpret this little endian 'word'



2.6 Exercise: Little Endian

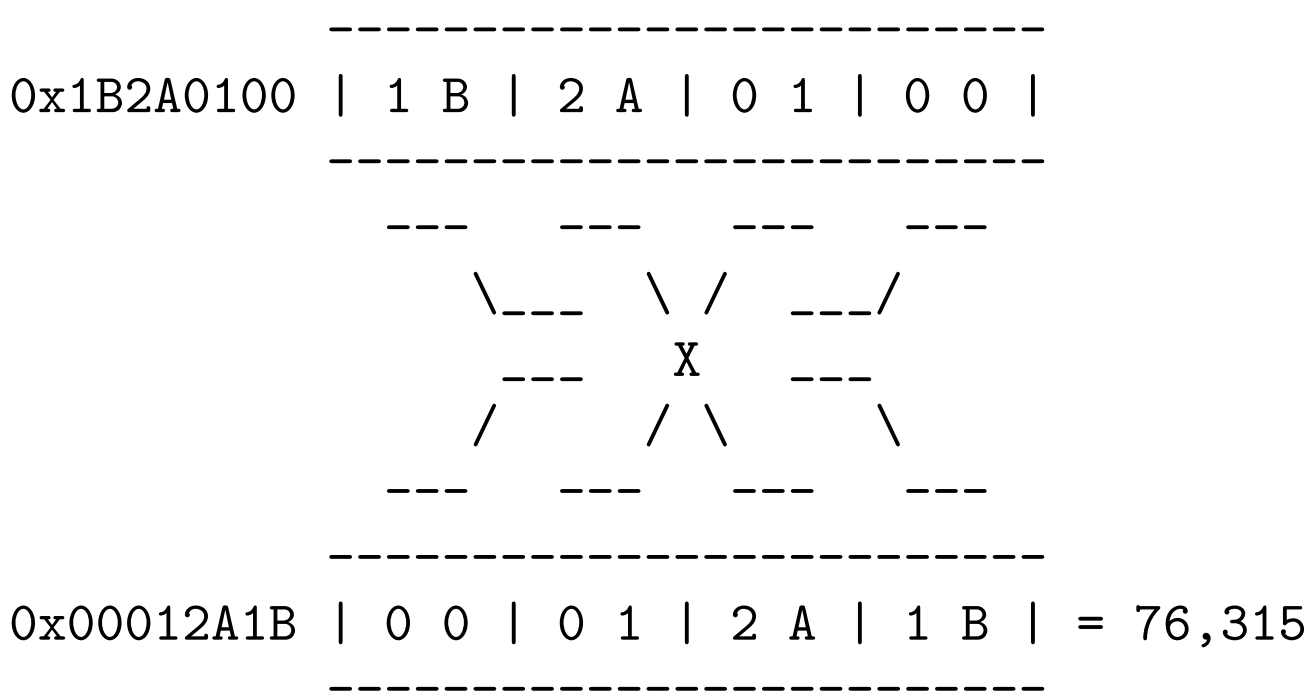
Read and interpret this little endian 'double word'

```
                 -------------------------
0x1B2A0100 | 1 B | 2 A | 0 1 | 0 0 |
                 -------------------------
```

```
                 -------------------------
0x               |     |     |     |     | =
                 -------------------------
```

2.6 Exercise: Little Endian

Read and interpret this little endian 'double word'



2.7 Example: Others

BCD / PBCD

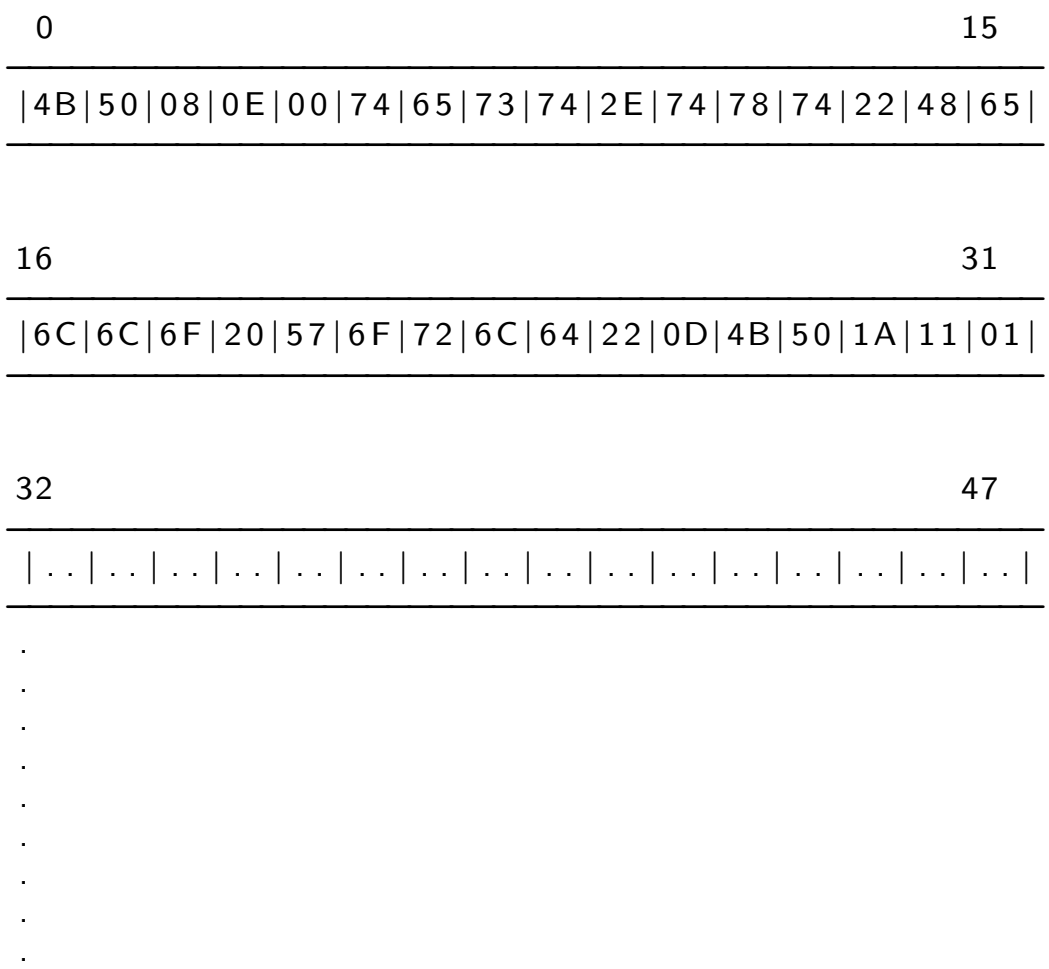
```
   2        9        1
-------- -------- --------
00000010 00001001 00000001
```

```
0110 1111 0000 1001
---- ---- ---- ----
 6    na   0    9
```

ASCII

```
01110000  01101001  01101110  01100111
--------  --------  --------  --------
  0x70      0x65      0x6E      0x67
   112       105       110       103
    p         i         n         g
```

2.8 Data structures: Example



2.8 Data structures: Example

```
 0                                                              15
-----------------------------------------------------------------
|4B|50|08|0E|00|74|65|73|74|2E|74|78|74|22|48|65|
-----------------------------------------------------------------
| K P | 8|  14 |

16                                                             31
-----------------------------------------------------------------
|6C|6C|6F|20|57|6F|72|6C|64|22|0D|4B|50|1A|11|01|
-----------------------------------------------------------------

32                                                             47
-----------------------------------------------------------------
|..|..|..|..|..|..|..|..|..|..|..|..|..|..|..|..|
-----------------------------------------------------------------
```

Offset	Size	Description
0	2	Header signature (ASCII)
2	1	Lenght of file name (Integer)
3	2	Lenght of data (Integer little endian)
5	—	Variable file name (ASCII)
5+	—	Data (Binary)

2.8 Data structures: Example

```
 0                                                 15
-------------------------------------------------------
|4B|50|08|0E|00|74|65|73|74|2E|74|78|74|22|48|65|
-------------------------------------------------------
| K P | 8|  14 | t  e  s  t  .  t  x t | "  H  e
16                                                31
-------------------------------------------------------
|6C|6C|6F|20|57|6F|72|6C|64|22|0D|4B|50|1A|11|01|
-------------------------------------------------------
  l  l  o     W  o  r  l  d  "   |
32                                                47
-------------------------------------------------------
|..|..|..|..|..|..|..|..|..|..|..|..|..|..|..|..|
-------------------------------------------------------
```

Offset	Size	Description
0	2	Header signature (ASCII)
2	1	Lenght of file name (Integer)
3	2	Lenght of data (Integer little endian)
5	—	Variable file name (ASCII)
5+	—	Data (Binary)

2.8 Data structures: Example

```
 0                                                 15
|4B|50|08|0E|00|74|65|73|74|2E|74|78|74|22|48|65|
| K P | 8|  14 | t  e  s  t  .  t  x  t | "  H  e
16                                                31
|6C|6C|6F|20|57|6F|72|6C|64|22|0D|4B|50|1A|11|01|
   l  l  o     W  o  r  l  d  "    | K P |26| 273 |
32                                                47
|..|..|..|..|..|..|..|..|..|..|..|..|..|..|..|..|
```

Offset	Size	Description
0	2	Header signature (ASCII)
2	1	Lenght of file name (Integer)
3	2	Lenght of data (Integer little endian)
5	—	Variable file name (ASCII)
5+	—	Data (Binary)

2.9 Data, files, context

- Sequence of Bits + Addressing + Interpretation → Information
- Information → Stored in files
- Where did you find the string "ping"?
  - Binary inside TEMP folder
  - Autorun folder
  - Registry
  - Browser history
  - Command line history

  → Data → Information → Knowledge

- Files contains data
- Files → Meta data describe files
- Files → File systems organize files and meta data

CIRCL
FORENSICS Training

3. Disk Acquisition

3.1 Storage devices / media

- IBM 305 RAMAC - IBM 350 Disk Storage
  - 1956: Random Access Method of Accounting and Control
  - 152 x 172 x 63 cm; 500 kg
  - 50.000 blocks of 100 Characters $\rightarrow$ 5MB

Image (c) www.chip.de - Image used solely for illustration purposes

3.1 Storage devices / media

ftp://ftp.seagate.com/techsuppt/misc/jet.txt

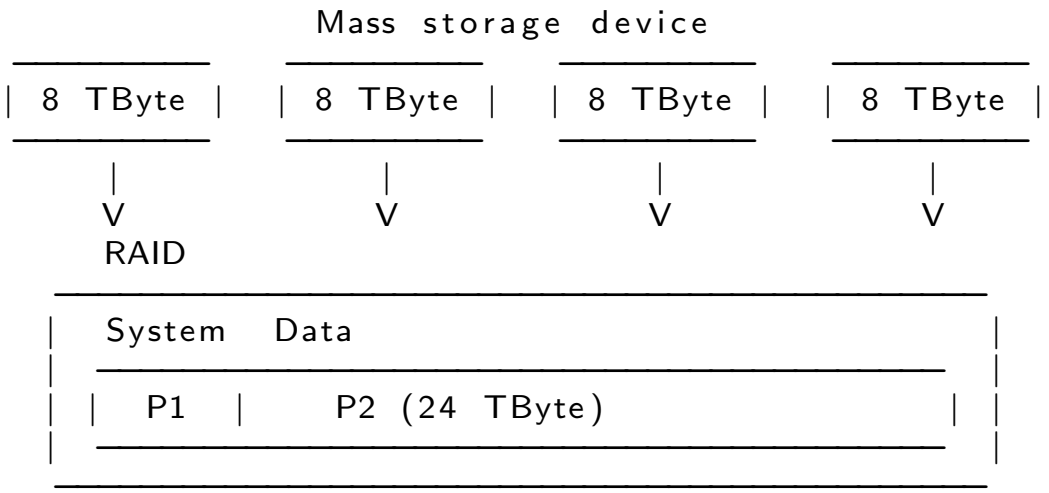
The incredible feat of a read/write head: Today's new generation of disc drives achieve the engineering equivalent of a Boeing 747 flying at MACH 4 just two meters above the ground, counting each blade of grass as it flies over. The read/write head floats at 12 millionths of an inch above the surface of the disc which is turning at 3,600 revolutions per minute. Read/write heads position precisely over information tracks which are 800 millionths of an inch apart and the data is electronically recorded at 20,000 bits per inch.

```
                        --.
                       / /    -- -- ---
                     /  /e
                   ./   /e         /---u O O OO
              /oAAAAAAAAAAAAAAAAAAA/
            {  @@@  oooooooooooooooo @@'
              \AAAAAA\AAAA\AAAAAAAAAA/
                     ue\    \
                         e\  \
                             \-\. = = ==

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

3.1 Storage devices / media

- Magnetic storage
  - Tapes
  - Floppy disks
    - 8" - 1971 - 80KB
    - 5.25" - 1976 - 360 KB
    - 3.5" - 1984 - 1.2 MB / - 1986 - 1.44 MB
  - Hard disks
    - IDE / EIDE, Firewire, PATA, SCSI
    - SATA, SAS Serial attached SCSI, USB, Thunderbolt
- Optical storage
  - Compact disks - CD
  - Digital versatile disk - DVD
  - Blu-ray disk
- Non-volatile memory
  - USB flash drive
  - Solid state drive
  - Flash memory cards

3.2 Physical- / Logical layers



Considerations: Disk duplication

Speed USB2: 480 Mbit/s
Capacity: 8 * 1024^4 * 8
Duration: ~40 hours per disk

Speed USB3.1: 10 Gbit/s
Capacity: 24 * 1024^4 * 8
Duration: ~5 hours per volume

(Theoretically)

A solution:

- Local NAS
- 10 GBit network
- USB 3.1 / 3.2
- 60+ TB mass storage
- Virtual appliance

3.3 ATA Disks

- ATA-3: Hard disk password
- ATA-4: HPA - Host Protected Area
  - Vendor area - benefit system vendors
  - Recovery data. persistent data
  - Controlled by firmware not OS
  - `READ_NATIVE_MAX_ADDRESS`
- ATA-6: DCO - Device Configuration Overlay
  - Benefit system vendors
  - Control reported capacity and disk features
  - Use disk from different manufacturers
  - Use disk with different number of sectors
    $\rightarrow$ Makes disks looking uniq
  - `DEVICE_CONFIGURATION_IDENTIFY`
- ATA-7: Serial ATA

3.4 Demo: Hidden Sectors

- New disk

```
dmesg
    sd 1:0:0:0: [sdb] 3904981168 512-byte logical blocks: (2.00 TB/1.82 TiB)

hdparm -N /dev/sdb
    max sectors   = 3907029168/3907029168, ACCESSIBLE MAX ADDRESS disabled
```

- Create hidden message

```
echo -n 'MySecret 123456' | dd of=/dev/sdb seek=3500000000

dd if=/bin/dd of=/dev/sdb seek=3500000001
     148+1 records in
     148+1 records out
     76000 bytes (76 kB, 74 KiB) copied, 0,022659 s, 3,4 MB/s
```

- Create HPA

```
hdparm --yes-i-know-what-i-am-doing -N p3000000000 /dev/sdb
    setting max visible sectors to 3000000000 (permanent)
    max sectors   = 3000000000/3907029168, ACCESSIBLE MAX ADDRESS enabled

Power cycle your device after every ACCESSIBLE MAX ADDRESS
```

3.4 Demo: Hidden Sectors

- Create partition and format

```
dmesg
    sd 1:0:0:0: [sdb] 3000000000 512-byte logical blocks: (1.54 TB/1.40 TiB)

fdisk /dev/sdb
    primary
    2048
    2999999999

mkfs.ntfs -L CIRCL.DFIR -f /dev/sdb1
    Creating NTFS volume structures.
    mkntfs completed successfully. Have a nice day.
```

- Investigate disk layout

```
fdisk -l /dev/sdb
    Device     Boot Start        End    Sectors  Size Id Type
    /dev/sdb1        2048 2999999999 2999997952  1,4T  7 HPFS/NTFS/exFAT
```

- Investigate last accessible sector

```
dd if=/dev/sdb skip=2999999999 status=none| xxd
    00000000: eb52 904e 5446 5320 2020 2000 0208 0000  .R.NTFS     .....
      .......
    000001f0: 0000 0000 0000 0000 0000 0000 0000 55aa  ..............U.
```

3.4 Demo: Hidden Sectors

- Try to access hidden message

```
dd if=/dev/sdb skip=3500000000 count=1 | xxd
    dd: /dev/sdb: cannot skip: Invalid argument
    0+0 records in
```

- Resize HPA

```
hdparm -N /dev/sdb
    max sectors   = 3000000000/3907029168, ACCESSIBLE MAX ADDRESS enabled

hdparm --yes-i-know-what-i-am-doing -N p3900000000 /dev/sdb
    max sectors   = 3900000000/3907029168, ACCESSIBLE MAX ADDRESS enabled

Power cycle your device after every ACCESSIBLE MAX ADDRESS
```

- Investigate disk layout and last sector

```
fdisk -l /dev/sdb
    Device     Boot Start        End    Sectors  Size Id Type
    /dev/sdb1        2048 2999999999 2999997952  1,4T  7 HPFS/NTFS/exFAT

dd if=/dev/sdb skip=2999999999 status=none | xxd | less
dd if=/dev/sdb skip=3899999999 status=none | xxd | less
```

3.4 Demo: Hidden Sectors

- Recover hidden message

```
dd if=/dev/sdb skip=3500000000 count=1 status=none
    00000000: 4d79 5365 6372 6574 2031 3233 3435 3600  MySecret 123456.
```

- Recover hidden dd command

```
dd if=/dev/sdb skip=$(( 3500000001*512 )) count=76000 bs=1 of=dd.exe

md5sum dd.exe
    36a70f825b8b71a3d9ba3ac9c5800683

md5sum /bin/dd
    36a70f825b8b71a3d9ba3ac9c5800683
```

- Feeback: kaplan(at)cert.at

```
https://www.schneier.com/blog/archives/2014/02/swap_nsa_exploi.html
https://en.wikipedia.org/wiki/Host_protected_area
```

- How it works

```
IDENTIFY DEVICE
SET MAX ADDRESS
READ NATIVE MAX ADDRESS
—> HPA aware software (like the BIOS)
```

3.5 Other Hidden Sectors

- Service area, negative sectors
  - Firmware
  - Bad sectors
  - ATA passwords

    ```
    hdparm --security-unlock "myPassWD" /dev/sdb
    ```
  - SMART data

- Self-Monitoring, Analysis and Reporting Technology - SMART

  ```
  apt install smartmontools
  smartctl -x /dev/sdb | less
  ```

  ```
  ......
  SMART Attributes Data Structure revision number: 16
  Vendor Specific SMART Attributes with Thresholds:
  ID# ATTRIBUTE_NAME          FLAGS    VALUE WORST THRESH  FAIL RAW_VALUE
    1 Raw_Read_Error_Rate     POSR-K   200   200   051     -    0
    3 Spin_Up_Time            POS--K   234   233   021     -    3258
    4 Start_Stop_Count        -O--CK   100   100   000     -    679
    5 Reallocated_Sector_Ct   PO--CK   200   200   140     -    0
    7 Seek_Error_Rate         -OSR-K   200   200   000     -    0
    9 Power_On_Hours          -O--CK   095   095   000     -    3802
    ......
  ```

3.6 Collecting information from devices

hdparm -I /dev/sdb

```
ATA device, with non-removable media
        Model Number:       WDC WD20NPVT-00Z2TT0
        Serial Number:      WD-WX11A9269540
        Firmware Revision:  01.01A01
        Transport:          Serial, SATA 1.0a, SATA Rev 2.6, SATA Rev 3.0
Standards:
        Supported: 8 7 6 5
        Likely used: 8
        .....
Security:
        Master password revision code = 65534    supported
        not     enabled
        not     locked
        not     frozen
        not     expired: security count
        374min for SECURITY ERASE UNIT.
```

hdparm -I /dev/sda

```
...
Commands/features:
        Enabled Supported:
        ...
           *    Data Set Management TRIM supported (limit 8 blocks)
           *    Deterministic read ZEROs after TRIM
```

3.7 How is the device connected

- Most relevant data with: dmesg

```
dmesg -T
    .....
    [Mi Aug  1 13:06:11 2018] usb-storage 1-1:1.0: USB Mass Storage device detected
    [Mi Aug  1 13:06:11 2018] scsi host1: usb-storage 1-1:1.0
    [Mi Aug  1 13:06:13 2018] scsi 1:0:0:0: Direct-Access USB Flash DISK
    [Mi Aug  1 13:06:13 2018] sd 1:0:0:0: Attached scsi generic sg1 type 0
    [Mi Aug  1 13:06:13 2018] sd 1:0:0:0: [sdb] 15826944 512-byte logical blocks
```

- Enumerate host hardware

```
lshw | less
    .....

lshw -businfo -class storage
    Bus info          Device          Class          Description
    ========================================================================
    pci@0000:04:00.0                  storage        Samsung Electronics Co Ltd
    usb@2:3           scsi0           storage
    usb@1:1           scsi1           storage

lshw -businfo -class disk
    Bus info          Device          Class          Description
    ========================================================================
    scsi@0:0.0.0      /dev/sda        disk           SD/MMC CRW
                      /dev/sda        disk
    scsi@1:0.0.0      /dev/sdb        disk           2TB 2000FYYZ-01UL1B2
```

3.7 How is the device connected

- Enumerate PCI bus

```
lspci -d ::0106                         # List SATA controller

lspci -d ::0108                         # List NVME controller
    04:00.0 Non-Volatile memory controller: Samsung Electronics Co Ltd Device a808

lspci -d ::0C03                         # List USB, FW, ... controller
    00:14.0 USB controller: Intel Corporation Sunrise Point-LP USB 3.0 xHCI Controller
    3b:00.0 USB controller: Intel Corporation JHL6540 Thunderbolt 3 USB Controller (C
    3e:00.0 USB controller: Fresco Logic FL1100 USB 3.0 Host Controller (rev 10)
    40:00.0 USB controller: Fresco Logic FL1100 USB 3.0 Host Controller (rev 10)
```

- Enumerate block devices

```
lsscsi -v

lsblk /dev/sdb
    NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
    sdb      8:16   0  1,8T  0 disk
     sdb1    8:17   0  1,8T  0 part /media/mich/031F0F30642CBB8B

lsblk -pd -o TRAN,NAME,SERIAL,VENDOR,MODEL,REV,WWN,SIZE,HCTL,SUBSYSTEMS /dev/sdb
    TRAN NAME      SERIAL             VENDOR   MODEL
    usb  /dev/sdb WD-WMC1P0H10ZEX WT055 WD 2000FYYZ-01UL1B2
             REV WWN                    SIZE HCTL       SUBSYSTEMS
            01.0 0x50014ee05979e023  1,8T 1:0:0:0     block:scsi:usb:pci
```

3.8 USB enumeration

- List attached USB device
  - USB bus
  - Device address
  - Vendor ID
  - Product ID
  - Product details

  ...

`lsusb`

```
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 002: ID 0bda:0328 Realtek Semiconductor Corp.
Bus 002 Device 003: ID 1b1c:1a0e Corsair
Bus 002 Device 004: ID 0951:162b Kingston Technology
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 004: ID 06cb:009a Synaptics, Inc.
Bus 001 Device 003: ID 04f2:b61e Chicony Electronics Co., Ltd
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

3.8 USB enumeration

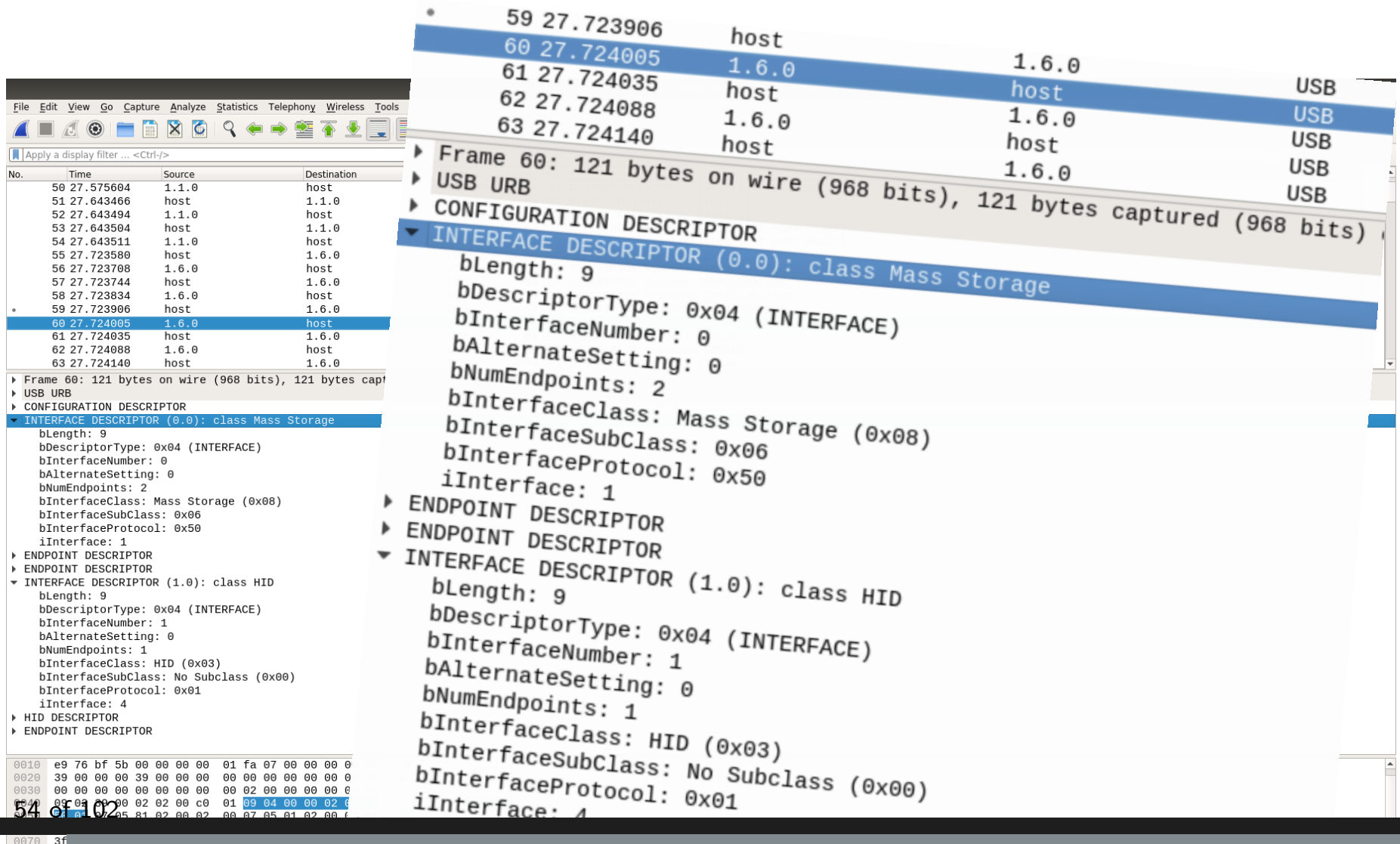
lsusb -t

```
/:  Bus 04.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/2p, 10000M
/:  Bus 03.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/2p, 480M
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/6p, 5000M
    |__ Port 1: Dev 4, If 0, Class=Mass Storage, Driver=usb-storage, 5000M
    |__ Port 2: Dev 3, If 0, Class=Mass Storage, Driver=uas, 5000M
    |__ Port 3: Dev 2, If 0, Class=Mass Storage, Driver=usb-storage, 5000M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/12p, 480M
    |__ Port 8: Dev 3, If 1, Class=Video, Driver=uvcvideo, 480M
    |__ Port 8: Dev 3, If 0, Class=Video, Driver=uvcvideo, 480M
    |__ Port 9: Dev 4, If 0, Class=Vendor Specific Class, Driver=, 12M
```

lsusb -v -d 0951:162b‘

```
...
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
      bNumEndpoints           2
      bInterfaceClass         8 Mass Storage
      bInterfaceSubClass      6 SCSI
      bInterfaceProtocol     80 Bulk-Only
...
```

3.9 USB Interface monitoring



CIRCL
FORENSICS Training

4. Disk Cloning / Disk Imaging

4.1 Disk cloning - imaging

- Clone disk-2-disk
  - Different sizes
  - Wipe target disk!
- Clone disk-2-image
  - Clear boundaries
  - One big file
  - Break file into chunks
- Image file format
  - RAW
  - AFF (Advanced Forensic Format)
  - EWF (Expert Witness Format)
  - Please no 3rd party formats
- Write-Blockers
  - Hardware

4.2 Connecting devices

- udev

```
udevadm info /dev/sda                    # userspace /dev
udevadm monitor
```

- /dev/

```
/dev/sd*                 # SCSI, SATA
/dev/hd*                 # IDE. EIDE
/dev/md*                 # RAID
/dev/nvme*n*             # NVME devices

/dev/sda1                # Partition 1 on disk 1
/dev/sda2                # Partition 2 on disk 1
...
```

- Block Devices
  - Attaching
  - Mounting

4.2 Read partition table

- `dmesg`

```
[106834.127269] sd 6:0:0:0: Attached scsi generic sg1 type 0
[106834.127503] sd 6:0:0:0: [sdb] 15826944 512-byte logical blocks: (8.10 GB/7.54 GiB)
[106834.130380] sd 6:0:0:0: [sdb] Write Protect is off
```

- `fdisk -l circl-dfir.dd`

```
Disk circl-dfir.dd: 1536 MB, 1536000000 bytes
4 heads, 7 sectors/track, 107142 cylinders, total 3000000 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
Disk identifier: 0x8f7e6594

          Device Boot      Start         End      Blocks   Id  System
circl-dfir.dd1               2048     3000000     1498976+   7  HPFS/NTFS/exFAT
```

- Exercise: Analyze output. Why 1498976? → Conclusions?

```
End:      echo $(( 3000000 * 512 ))                  --> 1536 MB, 1536000000 bytes
          echo $(( 3000000 * 512 / 1024 / 1024 ))    --> 1464

1498976: echo $(( 1498976 * 2 ))                     --> 2997952
```

4.2 Mounting

- `mount`

```
mkdir /mnt/ntfs                                  # Create mount point
mount /dev/sdb1 /mnt/ntfs                        # Mounting

mount -o ro,remount /dev/sdb1 /mnt/ntfs          # Re-mounting

umount /mnt/ntfs                                 # Un-mounting
umount /dev/sdb1                                 # Also un-mounting


# Mounting readonly, no journaling, no executable
mount -o ro,noload,noexec /dev/sdb1 /mnt/ntfs
mount -o ro,noload,noexec,remount /dev/sdb1 /mnt/ntfs


# Mounting with offset. mounting from image files
mount -o ro,noload,noexec,offset=$((512*2048)) circl-dfir.dd /mnt/ntfs


# Mounting NTFS file systems
mount -o ro,noload,noexec,offset=$((512*2048)),
      show_sys_files,streams_interface=windows circl-dfir.dd /mnt/ntfs
```

4.3 dd - disk imaging rudimentary

Copy files from: /mnt/ntfs/dd/

```
$ dd if=img_1.txt of=out_1.txt bs=512

     <input file>      <output file><block size>
                                      (default)
3+0 records in
3+0 records out
1536 bytes (1.5 kB) copied, 0.000126 s, 12.2 MB/s

$ ll
-rw-rw-r-- 1 hamm hamm 1536 May 16 11:20 img_1.txt
-rw-rw-r-- 1 hamm hamm 1536 May 16 11:16 out_1.txt



$ dd if=img_2.txt of=out_2.txt bs=512
3+1 records in
3+1 records out
1591 bytes (1.6 kB) copied, 0.00016048 s, 9.9 MB/s

$ ll
-rw-rw-r-- 1 hamm hamm 1591 May 16 11:20 img_2.txt
-rw-rw-r-- 1 hamm hamm 1591 May 16 11:26 out_2.txt
```

4.3 dd - disk imaging rudimentary

Demo: `skip` and `count` options

```
dd if=img_3.txt bs=512 skip=0 count=1 status=none | less
dd if=img_3.txt bs=512 skip=1 count=1 status=none | less
dd if=img_3.txt bs=512 skip=2 count=1 status=none | less
```

Exercise: Play with `bs`, `skip` and `count` options

```
dd if=img_3.txt bs=1 skip=$((512*3)) count=16 status=none
dd if=img_3.txt bs=16 skip=$((32*3)) count=1 status=none
```

Exercise: `dd | xxd | less`

```
dd if=img_3.txt bs=512 skip=3 count=1 status=none | xxd | less
```

```
0000000: 4f76 6572 6865 6164 2031 3233 3435 3637  Overhead 1234567
0000010: 3839 3020 204d 6573 7361 6765 2d31 2020  890  Message-1
0000020: 3039 3837 3635 3433 3231 2020 2020 2020  0987654321
0000030: 2020 2020 2020 20
```

Exercise: Find the secret password behind sector 3

4.3 dd - disk imaging rudimentary

Exercise: Continue an interrupted imaging process

```
dd if=img_2.txt of=broken.raw bs=512 skip=0 count=2 status=none

ll  img_2.txt  ..... 1591 Aug 13 14:40 img_2.txt*
ll broken.raw  ..... 1024 Aug 13 15:05 broken.raw

dd if=img_2.txt of=broken.raw bs=512 skip=2 seek=2 status=none

md5sum  img_2.txt f319b1cc9d424a923a8c83c3e67185f1
md5sum broken.raw f319b1cc9d424a923a8c83c3e67185f1
```

Error handling: Bad blocks

```
$ dd if=img_3.txt of=out_3.txt bs=512 conv=noerror,sync
```

Demo: Progress

```
Signaling: & and 'kill -10'
Signaling: & and 'kill -USR1'
Signaling: & and 'kill -USR1 $(pidof dd)'
Option:    status=progress
```

4.4 Disk acquisition

- Forensic features
  - Progress monitoring
  - Error handling & logging
  - Meta data
  - Splitting output files & support of forensic formats
  - Cryptographic hashing & verification checking
- Example: hashing

  `md5sum circl-dfir.dd` → `bd80672b9d1bef2f35b6e902f389e83`

  `sha1sum circl-dfir.dd` → `e5ffc7233a.....7e53b9f783`
- Tools
  - dd
  - ddrescue, gddrescue, dd_rescue
  - dc3dd - Department of Defense Cyber Crime Center
  - dcfldd - Defense Computer Forensic Labs
  - rdd-copy, netcat, socat, ssh
  - Guymager

4.5 Exercise: dc3dd

```
dc3dd if=/mnt/ntfs/carving/deleted.dd                    # Input file
      log=usb.log  -/                                    # Logging
      hash=md5 hash=sha1  -/                             # Hashing
      ofsz=$((8*1024*1024)) ofs=usb.raw.000              # Chunk files of 8MB


ls -l


cat usb.log


cat usb.raw.00* | md5sum                                 # Verify hashes
cat usb.raw.00* | sha1sum


dc3dd wipe=/dev/sdx                                      # Wipe a drive
```

4.6 SuashFS as forensic container

- Embedded systems
- Read only file system
- Supports very large files
- Adding files possible
- Deleting, modifying files not possible
- Compressed

  → Real case: 3*1TB disks stored in 293GB container
- Bruce Nikkel: `http://digitalforensics.ch/sfsimage/`

```
mksquashfs circl-dfir.dd case_123.sfs
mksquashfs analysis.txt case_123.sfs
unsquashfs -ll case_123.sfs
.....
mksquashfs analysis.txt case_123.sfs
.....
sudo mount case_123.sfs /mnt/
```

4.7 Exercise: Modify data on RO mounted device

```
mount
mount -o ro,remount /media/michael/7515-6AA5/
mount
```

Demo: Modify Document

```
strings -td /dev/sdb1
     .....
     299106 Hello World!
     .....

echo $((299106/512))
     584

dd if=/dev/sdb1 bs=512 skip=584 count=1 of=584.raw
ll
hexer 584.raw

dd of=/dev/sdb1 bs=512 seek=584 count=1 if=584.raw
mount
```

Demo: Review Document

4.7 Exercise: RO Countermeasures

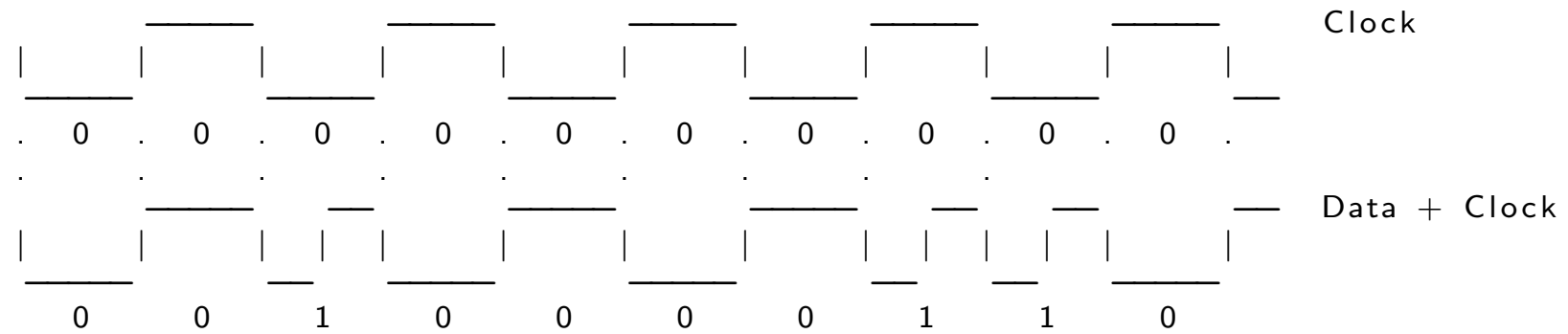
- Try on board methods:
  - `hdparm -r1 /dev/sdb`
  - `blockdev --setro /dev/sdb`
  - `udev` rules
    → Attack on block device still possible
- Try Forensics Linux Distributions:
  - Live Kali 2018_4 in forensic mode
  - SANS SIFT Workstation 3.0
  - DEFT X 8.2 DFIR Toolkit
    - Some distributions do not auto mount
      → Attack on block device still possible
- Kernel Patch: Linux write blocker (not tested)
  → `https://github.com/msuhanov/Linux-write-blocker`
- Hardware Write Blocker
  → Effectively block attack

CIRCL
FORENSICS Training

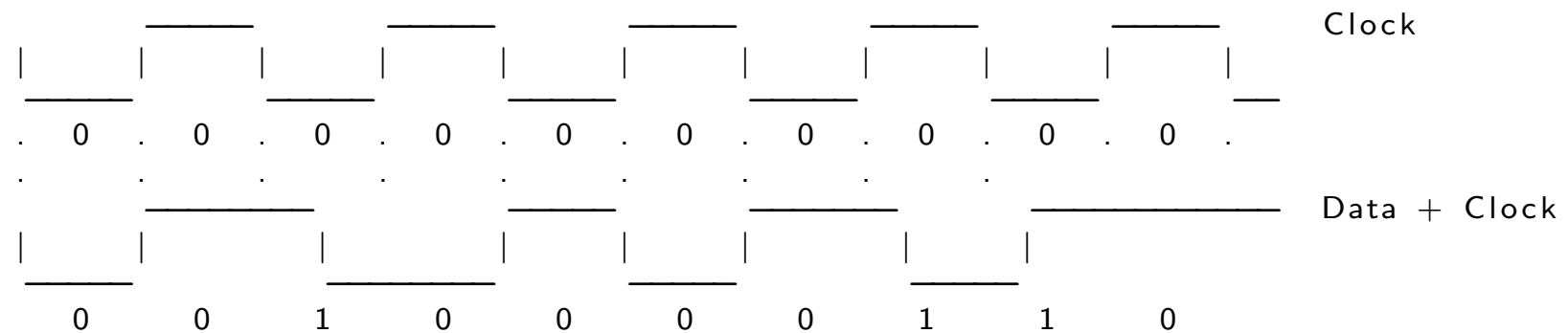
5. Disk Analysis

5.1 Low-Level Data Encoding

1. FM - Frequency Modulation



2. MFM - Modified Frequency Modulation (Double Density)



5.1 Low-Level Data Encoding

- RLL 2,7 - Run Length Limited
  - No more clock is stored
  - No less than 2 zeros in between two 1's
  - No mores than 7 zeros in between two 1's

Data chunk	RLL 2,7 code
000	000100
10	0100
010	100100
0010	00100100
11	1000
011	001000
0011	0001000

5.2 CHS - Cylinder Head Sector

Track, Head, Cylinder, Sector, Block, Cluster

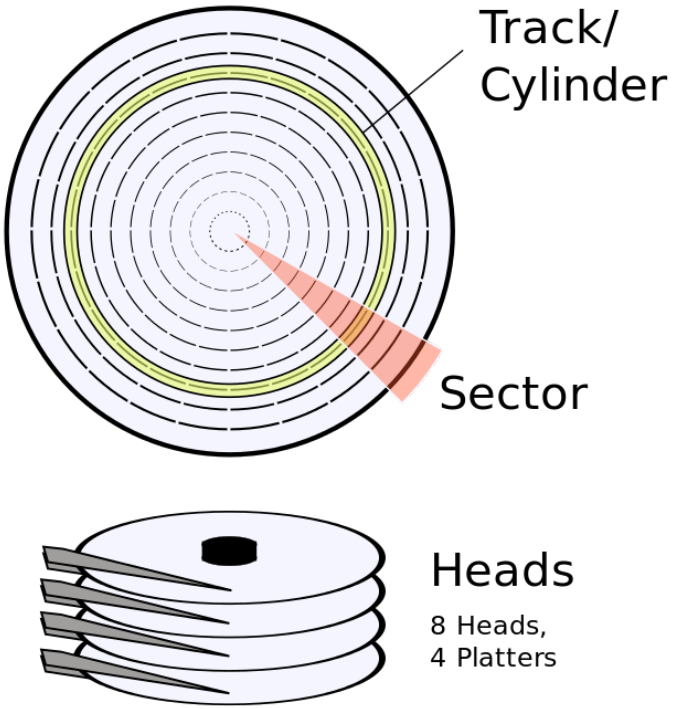


Image (c) wikipedia.org - Image used solely for illustration purposes

5.3 Low-Level: Sector Structur

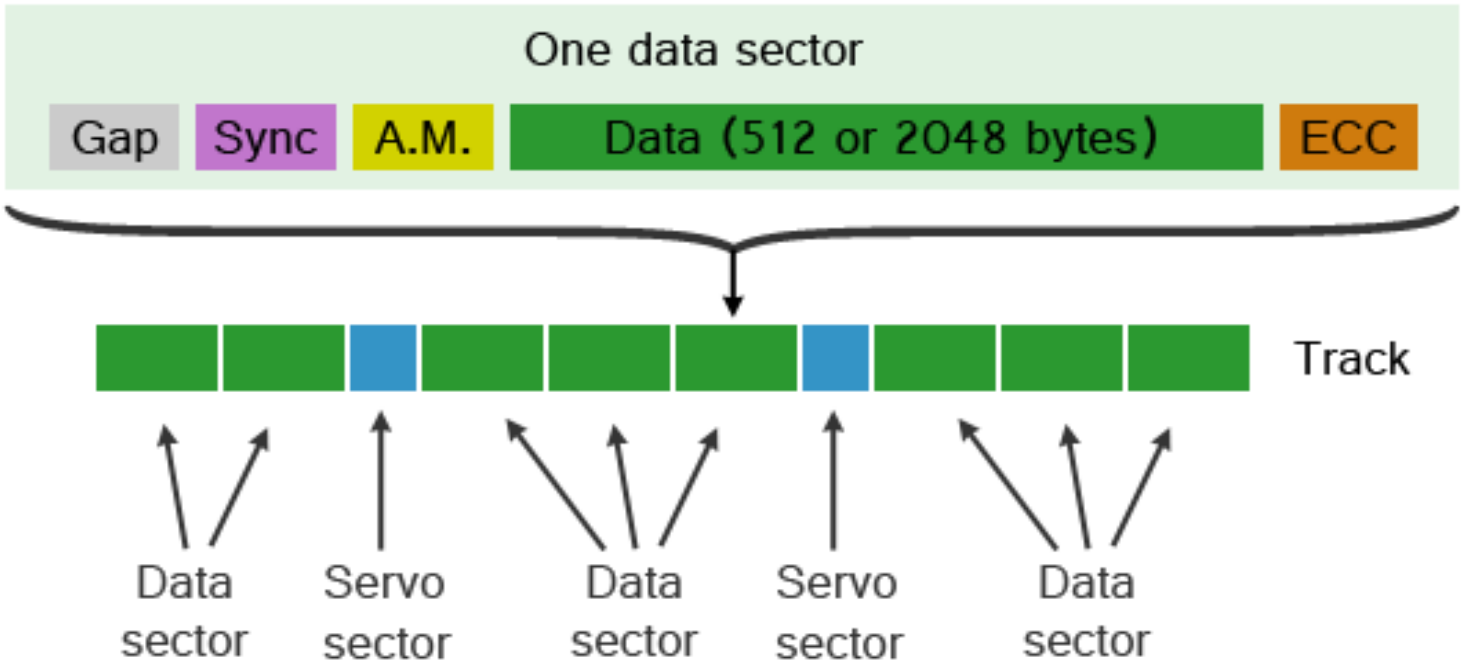


Image (c) forensicfocus.com - Image used solely for illustration purposes

5.4 Low-Level: Legacy considerations

Interleave Factor:

```
Interleave factor 1:1 ---> 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17
Interleave factor 2:1 ---> 01 10 02 11 03 12 04 13 05 14 06 15 07 16 08 17 09
Interleave factor 3:1 ---> 01 07 13 02 08 14 03 09 15 04 10 16 05 11 17 06 12
```

Zoned Bit Recording:

Zone:	12	11	10	09	08	07	06	05	04	03	02	01	00
Tracks:	100	120	140	155	170	185	195	205	210	210	215	218	220
Sectors:	132	132	132	132	132	132	132	132	100	100	100	100	100

Head and Cylinder Skewing:

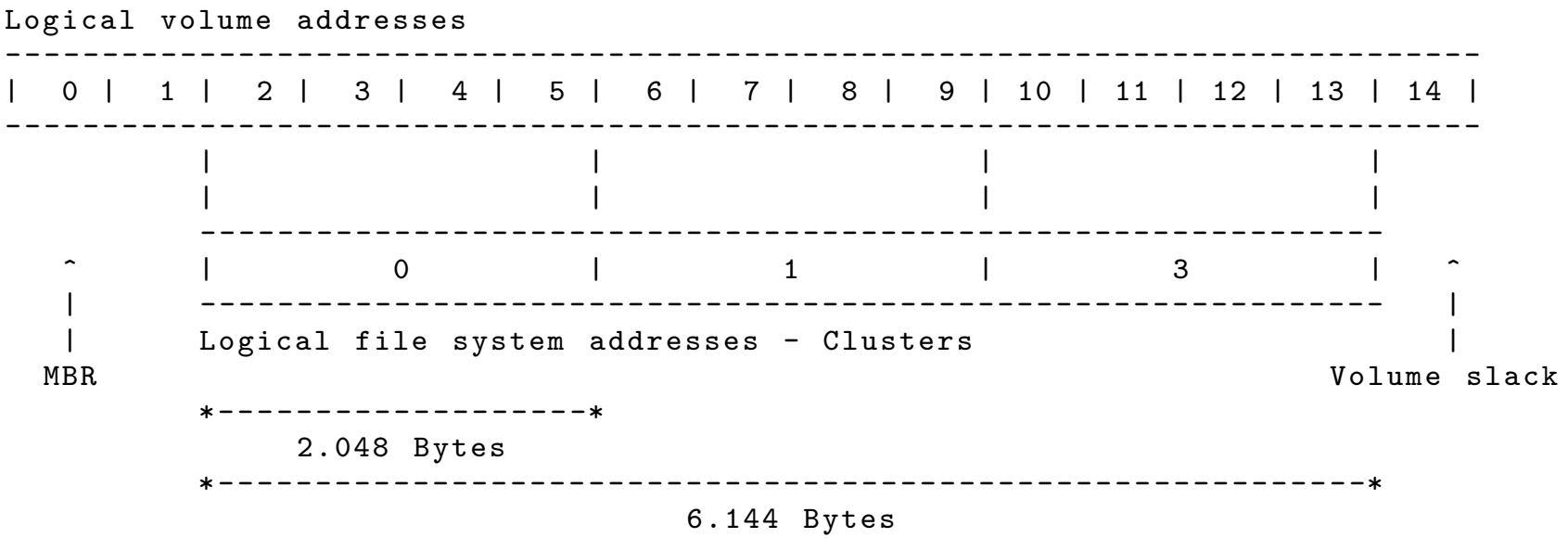
No skewing

```
Cylinder 0: Head 0: |01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17
            Head 1: |01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17
Cylinder 1: Head 0: |01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17
```

Head skew = 1, Cylinder skew = 4

```
Cylinder 0: Head 0: |01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17
            Head 1:  17|01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16
Cylinder 1: Head 0:  13 14 15 16 17|01 02 03 04 05 06 07 08 09 10 11 12
```

5.5 LBA - Logical Block Addressing



5.6 MBR - Master Boot Record

```
# dd if=/dev/sdc bs=512 count=1 skip=0 |xxd

0000000: fab8 0010 8ed0 bc00 b0b8 0000 8ed8 8ec0   ................
0000016: fbbe 007c bf00 06b9 0002 f3a4 ea21 0600   ...|.........!..
0000032: 00be be07 3804 750b 83c6 1081 fefe 0775   ....8.u........u
0000048: f3eb 16b4 02b0 01bb 007c b280 8a74 018b   .........|...t..
0000064: 4c02 cd13 ea00 7c00 00eb fe00 0000 0000   L.....|.........
0000080: 0000 0000 0000 0000 0000 0000 0000 0000   ................
0000096: 0000 0000 0000 0000 0000 0000 0000 0000   ................
...
...
0000432: 0000 0000 0000 0000 9af0 0200 0000 0020   ...............
0000448: 2100 0b1b 0299 0008 0000 0080 2500 00a8   !...........%...
0000464: 01a8 071a b327 0058 2900 00c0 5d00 001a   .....'.X)...]...
0000480: b427 076c dad2 0018 8700 00c0 6800 0000   .'.l........h...
0000496: 0000 0000 0000 0000 0000 0000 0000 55aa   ..............U.
```

000 — 439	0x000 — 0x1B7	Boot code
440 — 443	0x1B8 — 0x1BB	Disc signature
444 — 445	0x1BC — 0x1BD	Reserved
446 — 509	0x1BE — 0x1FD	Partitiontable
510 — 511	0x1FE — 0x1FF	0x55 0xAA

5.6 MBR - DOS Partition Table

```
# dd if=/dev/sdc bs=512 count=1 skip=0 |xxd

0000000: fab8 0010 8ed0 bc00 b0b8 0000 8ed8 8ec0   ................
0000016: fbbe 007c bf00 06b9 0002 f3a4 ea21 0600   ...|.........!..
0000032: 00be be07 3804 750b 83c6 1081 fefe 0775   ....8.u........u
0000048: f3eb 16b4 02b0 01bb 007c b280 8a74 018b   .........|...t..
0000064: 4c02 cd13 ea00 7c00 00eb fe00 0000 0000   L.....|.........
0000080: 0000 0000 0000 0000 0000 0000 0000 0000   ................
0000096: 0000 0000 0000 0000 0000 0000 0000 0000   ................
...
...
0000432: 0000 0000 0000 0000 9af0 0200 0000 0020   ...............
0000448: 2100 0b1b 0299 0008 0000 0080 2500 00a8   !...........%...
0000464: 01a8 071a b327 0058 2900 00c0 5d00 001a   .....'.X)...]...
0000480: b427 076c dad2 0018 8700 00c0 6800 0000   .'.l........h...
0000496: 0000 0000 0000 0000 0000 0000 0000 55aa   ..............U.
```

```
Partitiontable:
  Offset: 0      Size: 1 Value: 0x80     ---> Bootable
  Offset: 1      Size: 3 Value:          ---> Starting CHS address
  Offset: 4      Size: 1 Value: 0x0b     ---> FAT32
                                 0x07     ---> NTFS
  Offset: 5      Size: 3 Value:          ---> Ending CHS address
  Offset: 8      Size: 4 Value:          ---> Starting LBA address
  Offset:12      Size: 4 Value:          ---> LBA size in sectors
```

5.6 MBR - DOS Partition Table

```
0000432: 0000 0000 0000 0000 9af0 0200 0000 0020  ................
0000448: 2100 0b1b 0299 0008 0000 0080 2500 00a8  !...........%...
0000464: 01a8 071a b327 0058 2900 00c0 5d00 001a  .....'.X)...]...
0000480: b427 076c dad2 0018 8700 00c0 6800 0000  .'.l........h...
0000496: 0000 0000 0000 0000 0000 0000 0000 55aa  ..............U.

Partitiontable:
  Offset: 0      Size: 1 Value: 0x80     ---> Bootable
  Offset: 1      Size: 3 Value:          ---> Starting CHS address
  Offset: 4      Size: 1 Value: 0x0b     ---> FAT32
                                 0x07     ---> NTFS
  Offset: 5      Size: 3 Value:          ---> Ending CHS address
  Offset: 8      Size: 4 Value:          ---> Starting LBA address
  Offset:12      Size: 4 Value:          ---> LBA size in sectors

Addressable space:
  CHS: echo $((2**8 * 2**6 * 2**10 * 512 / 1024**2))  ==  8192 MByte
  LBA: echo $((2**32 * 512 / 1024**3))                ==  2048 GByte
```

- Exercise: Calculate the size of the partitions
  1. Take LBA size
  2. Apply Little Endian
  3. Apply sector size

5.6 MBR - DOS Partition Table

```
0000432: 0000 0000 0000 0000 9af0 0200 0000 0020  ................
0000448: 2100 0b1b 0299 0008 0000 0080 2500 00a8  !...........%...
0000464: 01a8 071a b327 0058 2900 00c0 5d00 001a  .....'.X)...]...
0000480: b427 076c dad2 0018 8700 00c0 6800 0000  .'.l........h...
0000496: 0000 0000 0000 0000 0000 0000 55aa  ..............U.
```

- Exercise: Calculate the size if the partitions

	LBA size	Little Endian		Sector size		
Part1:	0x00802500	0x00258000	2457600	* 512	1258291200	1.2 GB
Part2:	0x00c05d00	0x005dc000	6144000	* 512	3145728000	3.0 GB
Part3:	0x00c06800	0x0068c000	6864896	* 512	3514826752	3.4 GB

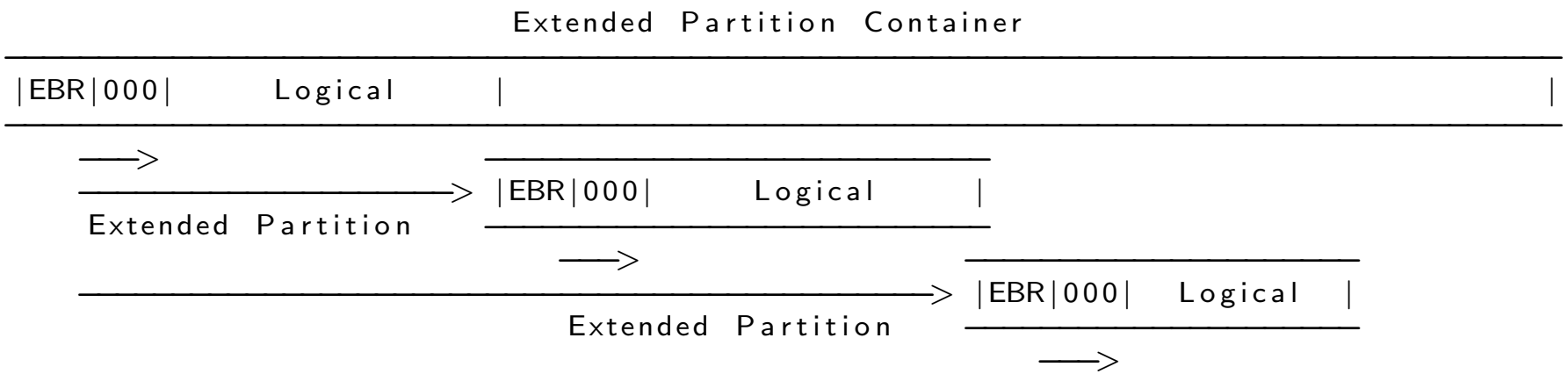
- Demo: Change partition type with hexeditor

  `fdisk -l /dev/sdb; hexedit /dev/sdb; F2, CTRL+x`

- Exercise: Find password in unused space before first partition

5.7 EBR - Extended Partitions

```
MBR:   000001b0: 0000 0000 0000 0000 d7b8 0cae 0000 0014
       000001c0: 0904 050f 823e 0008 0000 0000 0400 0000
```




```
EBR_01:  001001b0: 0000 0000 0000 0000 0000 0000 0000 0029
         001001c0: 0708 0717 0a2c 0008 0000 0040 0000 0018
         001001d0: 012c 051f 4206 0048 0000 0088 0100 0000

         EBR_02:  00A001B0: 0000 0000 0000 0000 0000 0000 0000 002C
                  00A001C0: 0930 071F 4206 0008 0000 0080 0100 001F
                  00A001D0: 4306 0503 8228 00D0 0100 0008 0200 0000

                  EBR_03:  03B001B0: 0000 0000 0000 0000 0000 0000 0000 0006
                           03B001C0: 410B 0703 8228 0008 0000 0000 0200 0000
                           03B001D0: 0000 0000 0000 0000 0000 0000 0000 0000
```

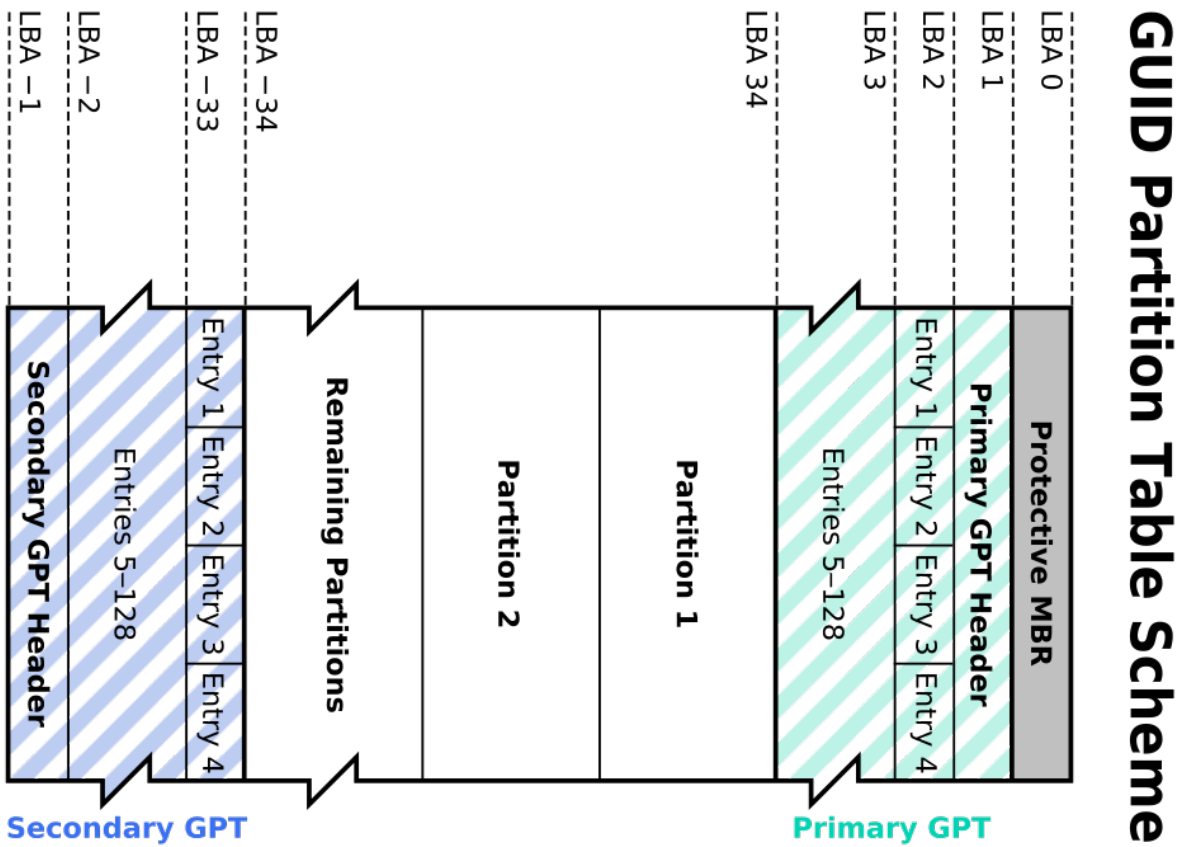
5.8 GPT - GUID Partition Table

- BIOS $\rightarrow$ UEFI - Unified Extensible Firmware Interface
- GUID - Globally Unique Identifier for each partition

  $\rightarrow$ GUID Partition Table
- Protective MBR at LBA0
  - One single entry covering the entire disk
  - Partition type 0xEE
    if 0xEE unknown $\rightarrow$ Not empty $\rightarrow$ Not formatted
- GPT header at LBA1
- GPT entries at LBA2 $\rightarrow$ LBA34
- GPT entries: 128 Bytes
- GPT backup at end of disk

5.8 GPT - GUID Partition Table

Figure: Image (c) wikipedia.org - Image used solely for illustration purposes



5.9 VBR - Volume Boot Record - Boot Sector

```
# dd if=/dev/sdc1 bs=512 count=1 skip=0 |xxd

0000000: eb58 906d 6b64 6f73 6673 0000 0208 2000  .X.mkdosfs.... .      # 0xeb 0x58 0x90
0000010: 0200 0000 00f8 0000 3e00 f800 0000 0000  ........>.......      # JMP  2+88 NOP
0000030: 0100 0600 0000 0000 0000 0000 0000 0000  ................
0000040: 0000 29a2 20e9 9c46 4154 2020 2020 2020  ..). ..FAT
0000050: 2020 4641 5433 3220 2020 0e1f be77 7cac    FAT32   ...w|.
0000060: 22c0 740b 56b4 0ebb 0700 cd10 5eeb f032  ".t.V.......^..2
...
...
00001f0: 0000 0000 0000 0000 0000 0000 0000 55aa  ..............U.


0 - 2           Size:  3    Jump to bootstrap code
3 - 10          Size:  8    OEM-ID: mkdosfs
11 - 12         Size:  2    Bytes per sector: 0x0002 -> 0x0200 (little endian)-> 512
13 (0xD)        Size:  1    Sectors per cluster: 0x08 -> 4096 bytes per cluster
50 (0x32) - 51  Size:  2    Boot sector backup: 0x0600 -> 0x0006 -> at sector 6
67 (0x43) - 70  Size:  4    Volume serial number: 0xa220e99c -> 0x9ce920a2
71 (0x47)       Size: 11    Volume label: FAT
82 (0x52)       Size:  8    Partition type: FAT32
90 (0x5A)- 509 (0x1FD)      Bootstrap code
510 (0x1FE)     Size:  2    Signature: 0x55AA
```

- Demo: Sleuthkit tools: `mmstat`, `mmls`, `fsstat`

CIRCL
FORENSICS Training

6. Forensics Challenges

6.1 Hide and recover data

- Situation:
  - USB stick image
  - One partition
  - Several unallocated sectors
- Challenge:
  - Hide a message in unallocated sector
  - Recover the message
  - Hide a binary in unallocated sectors
  - Recover the binary

6.1 Hide and recover data

1. Hide message in the last (unallocated) sector

```
echo -n "My secret message" | dd of=mbr_ex.raw seek=262143 status=none conv=notrunc
```

2. Read message from the last (unallocated) sector

```
dd if=mbr_ex.raw skip=262143 status=none | xxd | less
dd if=mbr_ex.raw skip=262143 status=none | strings
```

3. Hide a binary file between MBR and 1th partition

```
dd if=/bin/dd of=mbr_ex.raw seek=3 conv=notrunc
    76000 bytes (76 kB, 74 KiB) copied, 0,00173009 s, 43,9 MB/s
```

4. Recover the hidden binary file

```
dd if=mbr_ex.raw skip=0 count=4 | xxd | less

dd if=mbr_ex.raw bs=1 skip=$((3*512)) count=76000 of=ddddd.exe
    76000 bytes (76 kB, 74 KiB) copied, 0,12853 s, 591 kB/s

md5sum ddddd.exe /bin/dd
    36a70f825b8b71a3d9ba3ac9c5800683  ddddd.exe
    36a70f825b8b71a3d9ba3ac9c5800683  /bin/dd
```

6.2 Recovering corrupt MBR

- Situation:
  - USB stick image
  - Several partitions available
  - At least one partition do not mount

- Challenge:
  - Examine the partition table
  - Find the first sector of the partition
  - Fix the Master Boot Record - MBR
  - Analyze the other offsets
  - Analyze unallocated sectors

6.2 Recovering corrupt MBR

1. Examine the partition table

```
$ fdisk -l mbr/mbr_ex.raw
    Sector size (logical/physical): 512 bytes / 512 bytes
    Disklabel type: dos
    Disk identifier: 0x9392806f

    Device           Boot  Start    End Sectors Size Id Type
    mbr/mbr_ex.raw1        2050  67585   65536  32M  c W95 FAT32 (LBA)
    mbr/mbr_ex.raw2       67586 133119   65534  32M  c W95 FAT32 (LBA)
    mbr/mbr_ex.raw3      133120 262142  129023  63M  c W95 FAT32 (LBA)


$ mmls mbr/mbr_ex.raw
    DOS Partition Table
    Offset Sector: 0
    Units are in 512-byte sectors

          Slot      Start        End          Length       Description
    000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
    001:  -------   0000000000   0000002049   0000002050   Unallocated
    002:  000:000   0000002050   0000067585   0000065536   Win95 FAT32 (0x0c)
    003:  000:001   0000067586   0000133119   0000065534   Win95 FAT32 (0x0c)
    004:  000:002   0000133120   0000262142   0000129023   Win95 FAT32 (0x0c)
    005:  -------   0000262143   0000262143   0000000001   Unallocated
```

6.2 Recovering corrupt MBR

2. Investigate start of 1th partition

```
dd if=mbr/mbr_ex.raw skip=2050 count=1 status=none | xxd | less
dd if=mbr/mbr_ex.raw skip=2047 count=4 status=none | xxd | less
```

Fix LBA Start vaule of 1th partition entry

Calculation: 2048 = 0x00000800 => little endian: 0X00080000
Replace 0X02080000 with 0X00080000

```
hexedit -l 16 mbr/mbr_ex.raw
000001C0   21000C34  30040008  00000000  01000033
```

Review partition table and file system stats

```
mmls mbr/mbr_ex.raw
      Slot     Start        End          Length       Description
000:  Meta     0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------  0000000000   0000002047   0000002048   Unallocated
002:  000:000  0000002048   0000067583   0000065536   Win95 FAT32 (0x0c)
003:  -------  0000067584   0000067585   0000000002   Unallocated
004:  000:001  0000067586   0000133119   0000065534   Win95 FAT32 (0x0c)
005:  000:002  0000133120   0000262142   0000129023   Win95 FAT32 (0x0c)
006:  -------  0000262143   0000262143   0000000001   Unallocated
```

6.2 Recovering corrupt MBR

3. Investigate end of 1th and start of 2nd partition

```
fsstat -o 2048 mbr/mbr_ex.raw
    File System Type: FAT16
    Total Range: 0 - 65535
    ....
    --> Size of partition 1 is okay


sigfind -o 510 -l AA55 mbr/mbr_ex.raw
    Block: 0 (-)
    Block: 2048 (+2048)
    Block: 67586 (+65538)
    Block: 133120 (+65534)


fsstat -o 67586 mbr/mbr_ex.raw
    File System Type: FAT16
    Total Range: 0 - 65533
    ....
    --> Start of partition 2 is okay
    --> There are 2 unallocated sectors in between
    --> Size of partition 2 is okay
```

Investigate the sectors

```
dd if=mbr/mbr_ex.raw skip=67583 count=4 | xxd | less
```

6.2 Recovering corrupt MBR

```
005:  000:002   0000133120   0000262142   0000129023   Win95 FAT32 (0x0c)
006:  --------  0000262143   0000262143   0000000001   Unallocated
```

4. Investigate 3rd partition

```
sigfind -o 510 -l AA55 mbr/mbr_ex.raw
    Block: 0 (-)
    Block: 2048 (+2048)
    Block: 67586 (+65538)
    Block: 133120 (+65534)


fsstat -o 133120 mbr/mbr_ex.raw
    File System Type: FAT16
    Total Range: 0 - 129022
    ....
    --> Start of partition 3 is okay
    --> Size of partition 3 is okay
    --> There is 1 unallocated sector at end of disk
```

Investigate the last 2 sectors of disk

```
dd if=mbr/mbr_ex.raw skip=262142 | xxd | less
```

6.3 Lost in Hyperspace: USB stick investigation

- Situation:
  - USB stick image with one extended partition
  - Some logical partitions available
  - Countles partitions get mounted

- Challenge:
  - Analyze USB stick with standard tools
  - Analyze MBR with a hexeditor
  - Discover whats going wrong
  - Fix the broken values

6.3 Lost in Hyperspace: USB stick investigation

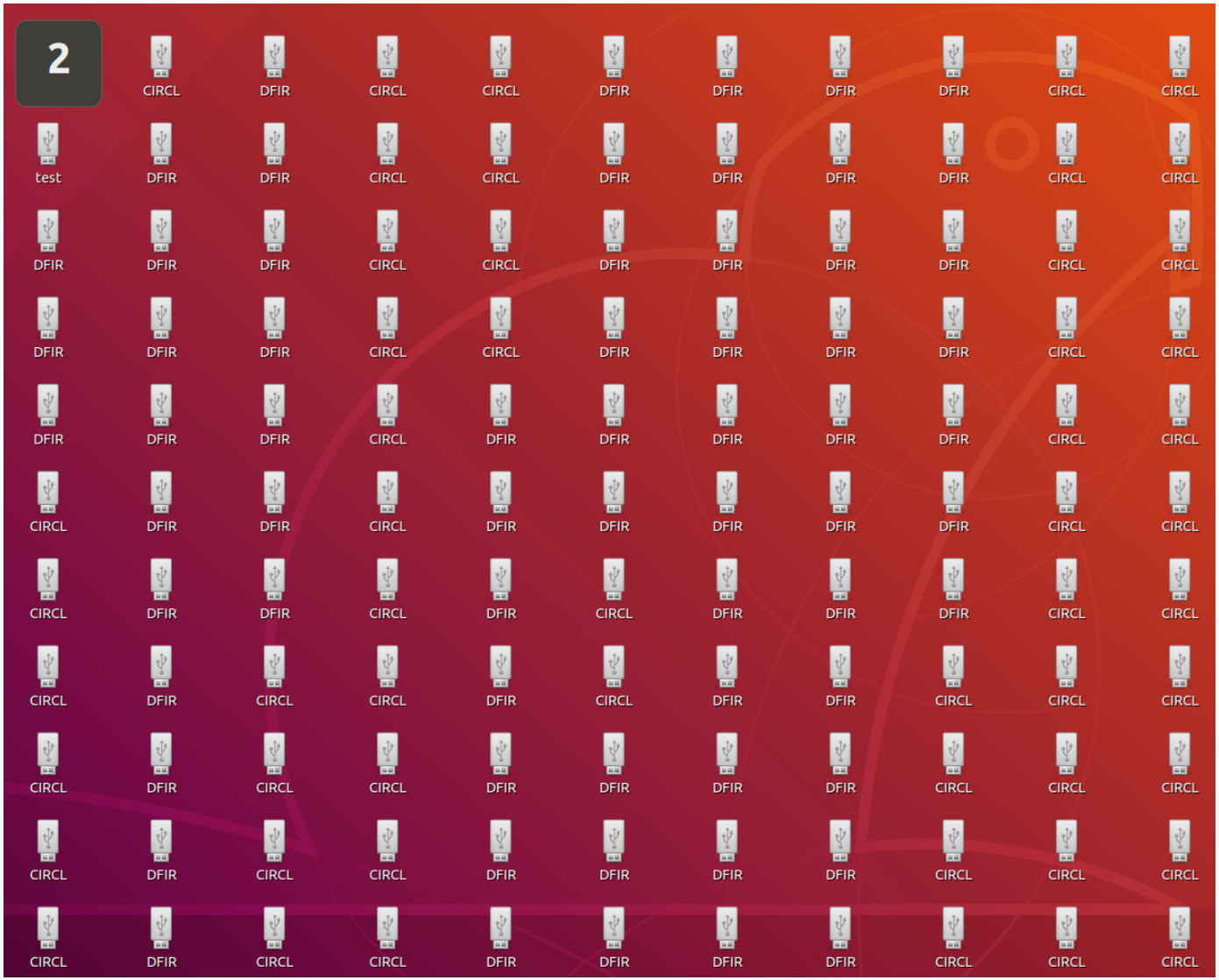
USB stick before manipulation:

```
# dmesg -T
     [Do Jan 23 21:40:07 2020] sd 1:0:0:0: [sdb] 250068992 512-byte logical blocks:
     [Do Jan 23 21:40:07 2020]  sdb: sdb1 < sdb5 sdb6 sdb7 >

# fdisk -l /dev/sdb
     Device     Boot  Start    End Sectors  Size Id Type
     /dev/sdb1          2048 264191  262144  128M  5 Extended
     /dev/sdb5          4096  20479   16384    8M  7 HPFS/NTFS/exFAT
     /dev/sdb6         22528 120831   98304   48M  7 HPFS/NTFS/exFAT
     /dev/sdb7        122880 253951  131072   64M  7 HPFS/NTFS/exFAT

# mount              /dev/sdb7 on /media/michael/DFIR
                     /dev/sdb6 on /media/michael/CIRCL
                     /dev/sdb5 on /media/michael/test

# df -ha | grep sdb
     /dev/sdb7             64M  2,5M   62M   4% /media/michael/DFIR
     /dev/sdb6             48M  2,5M   46M   6% /media/michael/CIRCL
     /dev/sdb5            8,0M  2,5M  5,6M  31% /media/michael/test
```

Manipulation 4 bytes:

```
# hexedit /dev/sdb
     .....
     03B001C0   41 0B 07 03  82 28 00 08  00 00 00 00  02 00 00 00  A....(..........
     03B001D0   00 00 05 00  00 00 00 48  00 00 00 88  01 00 00 00  .......H........
```

6.3 Lost in Hyperspace: WTF



6.3 Lost in Hyperspace: USB stick investigation

```
$ fdisk -l /dev/sdb

      Device         Boot   Start     End  Sectors   Size Id Type
      /dev/sdb1             2048  264191   262144   128M  5 Extended
      /dev/sdb5             4096   20479    16384     8M  7 HPFS/NTFS/exFAT
      /dev/sdb6            22528  120831    98304    48M  7 HPFS/NTFS/exFAT
      /dev/sdb7           122880  253951   131072    64M  7 HPFS/NTFS/exFAT
      /dev/sdb8            22528  120831    98304    48M  7 HPFS/NTFS/exFAT
      /dev/sdb9           122880  253951   131072    64M  7 HPFS/NTFS/exFAT
            .....
            .....
      /dev/sdb56           22528  120831    98304    48M  7 HPFS/NTFS/exFAT
      /dev/sdb57          122880  253951   131072    64M  7 HPFS/NTFS/exFAT
      /dev/sdb58           22528  120831    98304    48M  7 HPFS/NTFS/exFAT
      /dev/sdb59          122880  253951   131072    64M  7 HPFS/NTFS/exFAT
      /dev/sdb60           22528  120831    98304    48M  7 HPFS/NTFS/exFAT

$ mount
            .....
      /dev/sdb79 on /media/michael/DFIR25
      /dev/sdb82 on /media/michael/CIRCL28
      /dev/sdb86 on /media/michael/CIRCL33
            .....
            .....
      /dev/sdb162 on /media/michael/CIRCL68
      /dev/sdb163 on /media/michael/DFIR73
      /dev/sdb166 on /media/michael/CIRCL64
```

6.3 Lost in Hyperspace: USB stick investigation

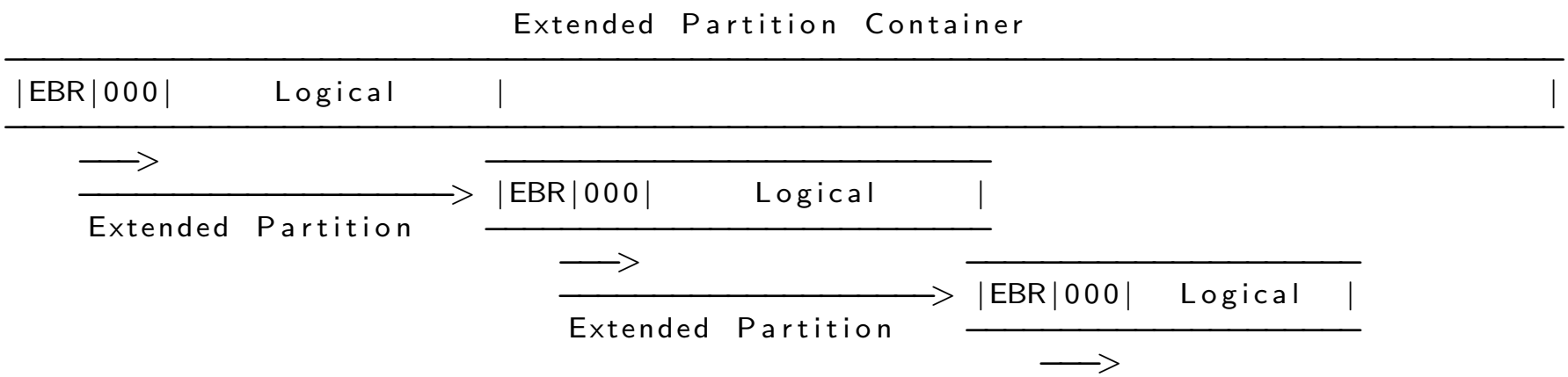
Do further investigations:

```
$ df -ha

      .....
/dev/sdb157      64M  2,5M   62M   4% /media/michael/DFIR72
/dev/sdb158      48M  2,5M   46M   6% /media/michael/CIRCL63
/dev/sdb159      64M  2,5M   62M   4% /media/michael/DFIR69
/dev/sdb160      48M  2,5M   46M   6% /media/michael/CIRCL67
/dev/sdb162      48M  2,5M   46M   6% /media/michael/CIRCL68
/dev/sdb163      64M  2,5M   62M   4% /media/michael/DFIR73
      .....

$ mmls /dev/sdb                    --> Nothing... WTF?
```

Any ideas how to proceed?

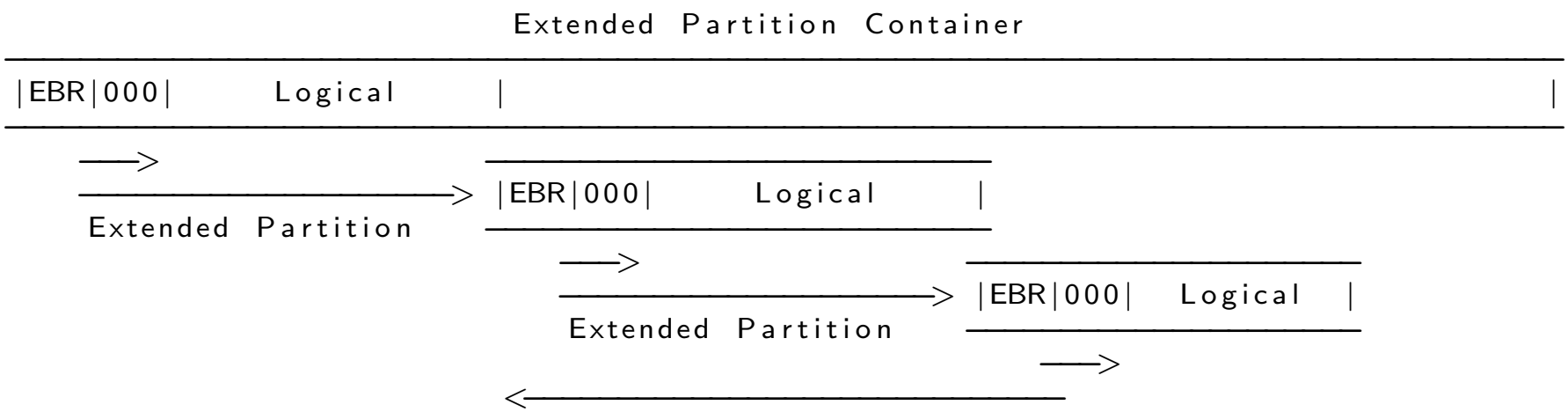
→ Use hexeditor to read the partition table

6.3 Lost in Hyperspace: Solution step 1




```
EBR_01:  001001b0: 0000 0000 0000 0000 0000 0000 0000 0029
         001001c0: 0708 0717 0a2c 0008 0000 0040 0000 0018
         001001d0: 012c 051f 4206 0048 0000 0088 0100 0000

         EBR_02:  00A001B0: 0000 0000 0000 0000 0000 0000 0000 002C
                  00A001C0: 0930 071F 4206 0008 0000 0080 0100 001F
                  00A001D0: 4306 0503 8228 00D0 0100 0008 0200 0000

                  EBR_03:  03B001B0: 0000 0000 0000 0000 0000 0000 0000 0006
                           03B001C0: 410B 0703 8228 0008 0000 0000 0200 0000
                           03B001D0: 0000 0000 0000 0000 0000 0000 0000 0000
```

6.3 Lost in Hyperspace: Solution step 2




```
EBR_01:  001001b0: 0000 0000 0000 0000 0000 0000 0000 0029
         001001c0: 0708 0717 0a2c 0008 0000 0040 0000 0018
         001001d0: 012c 051f 4206 0048 0000 0088 0100 0000

         EBR_02:  00A001B0: 0000 0000 0000 0000 0000 0000 0000 002C
                  00A001C0: 0930 071F 4206 0008 0000 0080 0100 001F
                  00A001D0: 4306 0503 8228 00D0 0100 0008 0200 0000

                  EBR_03:  03B001B0: 0000 0000 0000 0000 0000 0000 0000 0006
                           03B001C0: 410B 0703 8228 0008 0000 0000 0200 0000
                           03B001D0: 0000 0500 0000 0048 0000 0088 0100 0000
```

CIRCL
FORENSICS Training

6. Bibliography and Outlook

6.1 Outlook

CIRCL - DFIR 1.0.2

File System Forensics and Data Recovery

CIRCL - DFIR 1.0.3

Windows-, Memory- and File Forensics

6.2 Bibliography

- Digital Forensics with Kali Linux
  Shiva V.N. Parasram
  Packt Publishing
  ISBN-13: 978-1-78862-500-5

- Practical Forensic Imaging
  Bruce Nikkel
  No Starch Press
  ISBN-13: 978-1-59-327793-2

- Digital Forensics with Open Source Tools
  Cory Altheide, Harlan Carvey
  Syngress
  ISBN-13: 978-1-59-749586-8

6.2 Bibliography

- File System Forensic Analysis

  Brian Carrier
  Pearson Education
  ISBN-13: 978-0-32-126817-4

- Forensic Computing: A Practitioner's Guide

  Anthony Sammes, Brian Jenkinson
  Springer
  ISBN-13: 978-1-85-233299-0

Overview

1. Introduction
2. Information
3. Disk Acquisition
4. Disk Cloning / Disk Imaging
5. Disk Analysis
6. Forensics Challenges
7. Bibliography and Outlook