Digital Forensics

Lab 04: Data Hiding & Data Recovery

Master Exam-Ready Study Notes — 2025/26

Contents

1 Lab 04 Overview & Objectives

This lab covers three core DF skills:

1. **Disk Partitioning** — creating and understanding partitions on a new disk.
2. **MBR Examination** — reading raw hex from Sector 0 and decoding partition entries.
3. **MBR Repair** — deliberately breaking and then repairing the MBR to understand boot failure forensics.

Why This Matters for DF If you receive a device (or a raw `dd` image) and there appears to be no data, the ability to **identify and recover partitions** from the MBR is critical. Recovering the partition table lets you access file systems, which in turn lets you recover files, metadata, names, locations — all providing forensic context.

2 Lab 4.1 — Creating Partitions on a Disk

2.1 Virtualisation Setup

Definition — Virtual Machine (VM) A **Virtual Machine** is a software-emulated computer running inside your real ("host") computer. It behaves like a completely separate machine with its own OS, disks, and memory. VMs are essential in DF because you can safely examine evidence without risking your host system.

The lab uses **VMware Workstation Player v17** to run a **Windows XP Professional** VM. Key steps:

Step	Action
1. Install VMware	Untick "Check for product updates" and "Join the VMware CEIP" during setup
2. Open VM	Open the `.vmx` file from the unzipped "Windows XP Pro – chrome" folder
3. Add a new disk	Edit Virtual Machine Settings → Add → Hard Disk → SCSI → Create new virtual disk
4. Disk size	Set to **1.01 GB**, stored as a **single file**
5. Launch	Play Virtual Machine → choose "Copied it"

2.2 Creating 4 Primary Partitions

Once the VM boots, you use **Disk Management** (right-click My Computer → Manage → Disk Management) to create partitions on the new 1.01 GB disk (Disk 1).

#	Volume Label	Size	File System	Drive Letter
1	Part 1	10 MB	NTFS	E:
2	Part 2	10 MB	NTFS	F:
3	Part 3	10 MB	NTFS	G:
4	Part 4	10 MB	NTFS	H:

Procedure for each partition:

1. Right-click unallocated space → New Partition.
2. Choose **Primary Partition** → Next.
3. Set size to **10 MB** → Next.
4. Assign drive letter → Next.
5. Format: NTFS, Quick format, Volume label = "Part N" → Finish.

EXAM KEY POINT — Why 4 Partitions Maximum? The MBR's partition table is **64 bytes** long, with each entry requiring **16 bytes**. $64 \div 16 = 4$ entries. That is why MBR-based disks support a **maximum of 4 primary partitions**. Additional "logical volumes" (extended partitions) can exist *within* a primary partition, but you **cannot boot an OS** from a logical volume.

Windows 10/11 Note When replicating this lab in Windows 10/11, you must select **MBR** (not GPT) when initialising the 1.01 GB disk. The partition wizard is called "New Simple Volume" instead of "New Partition."

3 Lab 4.2 — Examining the MBR

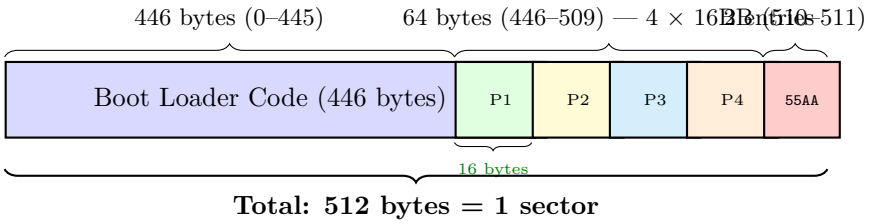
3.1 MBR Recap — What You Must Know

Definition — Master Boot Record (MBR) The **MBR** is the first 512 bytes (Sector 0) of a physical disk. It tells the computer how the disk is partitioned and which partition to boot from.

MBR Quick-Fire Q&A (Exam Favourites)

Question	Answer
What is the MBR for?	Defines the disk's partitions and contains boot loader code to start the OS
What sector is it located at?	**Sector 0** (the very first 512 bytes of the disk)
How many primary partitions can it define?	**4** (64 bytes ÷ 16 bytes per entry)
What are the last 2 bytes?	`55 AA` (the MBR signature at offset `0x1FE` = byte 510)

3.2 MBR Structure Diagram



3.3 Using a Sector Editor (SectEdit)

Roadkil's SectEdit is a hex-level sector editor that lets you view and modify raw disk sectors.

SectEdit Procedure

```
1. Extract SectEdit to C:\SectEdit
2. Right-click -> Run as Administrator
3. Untick "Protect my computer..."  -> OK
4. Choose "Physical 1" (the 1.01 GB disk)
   -> Automatically taken to Sector 0 (the MBR)
```

What you see is a **hex dump** of the MBR — 512 bytes displayed as hexadecimal values alongside their ASCII representations.

EXAM KEY POINT — Finding Partitions in the MBR Hex Dump

1. **Find** `55 AA` at the very end (offset `0x1FE`–`0x1FF`, bytes 510–511).
2. **Go back 64 bytes** from `55 AA` — this is the start of the partition table (byte 446).
3. **Each 16-byte chunk** defines one partition (4 chunks total).
4. Decode each chunk using the partition entry table below.

3.4 Partition Table Entry Structure (16 bytes)

This is the format for **each** of the 4 partition entries inside the MBR:

Offset	Length	Contents
0	1 byte	**Boot Indicator**: `0x80` = active (bootable), `0x00` = not bootable
1–3	3 bytes	Starting CHS (Cylinder-Head-Sector) values
4	1 byte	**Partition-type descriptor**: identifies the file system (e.g. `0x07` = NTFS)
5–7	3 bytes	Ending CHS values
8–11	4 bytes	**Starting Sector** — stored in **little-endian** (read right-to-left)
12–15	4 bytes	**Partition Size** (in sectors) — stored in **little-endian** (read right-to-left)

Definition — Little-Endian **Little-endian** means the **least significant byte is stored first** (at the lowest address). To read the actual value, you reverse the byte order. Example: stored as `C1 3E 00 00` → actual value = `00 00 3E C1`.

Definition — Partition-Type Descriptor Byte 4 of each entry identifies the file system. Common values:

Hex Value	File System
`0x07`	NTFS
`0x0B`	FAT32 (CHS)
`0x0C`	FAT32 (LBA)
`0x83`	Linux (Ext2/3/4)
`0x82`	Linux Swap
`0x05`	Extended Partition

4 Worked Example — Decoding a Partition Entry

This is the most exam-critical skill from Lab 04: reading raw hex bytes and computing partition details.

4.1 The Raw Hex Data (Partition 2 from the Lab)

The 16 bytes for Partition 2 ("Part 2") as seen in SectEdit:

```
00  01  01  07  FE  3F  01  C1  3E  00  00  C1  3E  00  00
```

(Note: The first byte in the lab sheet is listed as `00`, giving 16 bytes: positions 0 through 15.)

4.2 Step-by-Step Decode

Byte(s)	Hex	Field	Interpretation
0	`00`	Boot Indicator	`0x00` = **NOT bootable**. If this were `0x80`, the BIOS would try to boot from this partition.
1–3	`01 01 00`	Starting CHS	CHS addressing (less important on modern systems; LBA used instead).
4	`07`	Partition Type	`0x07` = **NTFS**. This tells us the file system format.
5–7	`FE 3F 01`	Ending CHS	Ending Cylinder-Head-Sector values.
8–11	`C1 3E 00 00`	Starting Sector	**Little-endian**: reverse → `00 00 3E C1`. Convert hex to decimal: see below.
12–15	`C1 3E 00 00`	Partition Size	**Little-endian**: reverse → `00 00 3E C1`. Convert hex to decimal: see below.

4.3 Converting the Starting Sector / Partition Size

Full Worked Calculation **Raw bytes (stored):** `C1 3E 00 00`
Step 1 — Reverse for little-endian:

$$\texttt{C1 3E 00 00} \xrightarrow{\text{reverse}} \texttt{00 00 3E C1}$$

Step 2 — Convert hex `3EC1` to decimal:

$$3 \times 16^3 + 14 \times 16^2 + 12 \times 16^1 + 1 \times 16^0$$

$$= 3 \times 4096 + 14 \times 256 + 12 \times 16 + 1 \times 1$$

$$= 12{,}288 + 3{,}584 + 192 + 1 = \mathbf{16{,}065} \text{ sectors}$$

Step 3 — Convert sectors to bytes:

$$16{,}065 \times 512 = \mathbf{8{,}225{,}280} \text{ bytes}$$

Step 4 — Convert to Kilobytes:

$$\frac{8{,}225{,}280}{1{,}024} = \mathbf{8{,}032} \text{ KB}$$

Step 5 — Convert to Megabytes:

$$\frac{8{,}032}{1{,}024} = 7.84 \approx \mathbf{8}\ \text{MB}$$

This matches the roughly 10 MB partition we created (the formatted usable size is slightly smaller due to file system overhead).

4.4 Verification

You can verify this by checking "My Computer" in the VM — the partition labelled "Part 2" should show approximately 8 MB of usable space, consistent with the calculated value. The difference between the 10 MB requested and 8 MB usable is due to NTFS metadata overhead (MFT, journalling, etc.).

5 Lab 4.2 Continued — Breaking & Repairing the MBR

5.1 Breaking the MBR (Intentional Corruption)

What Happens When You Corrupt the MBR Signature In SectEdit, the lab instructs you to:

1. Open **Physical Disc 0** (the main boot disk).
2. Change the last 2 bytes from `55 AA` to `AA 55` (swap them).
3. Save the sector.

Result: The VM **will not boot**. The BIOS checks for the `55 AA` signature at bytes 510–511 of Sector 0. If this signature is missing or incorrect, the BIOS does not recognise the sector as a valid MBR and refuses to boot.

EXAM KEY POINT — Why `55 AA` Matters The 2-byte signature `55 AA` at the end of the MBR is a **magic number** — a validity marker. The BIOS/firmware checks for it during the boot process:

- `55 AA` present → Valid MBR → proceed to boot.
- `55 AA` absent or wrong → Invalid MBR → **system fails to boot**.

The exact byte order matters: `55` at offset 510, `AA` at offset 511. Reversing them (`AA 55`) is enough to break the boot process entirely.

5.2 Repairing the MBR with Hiren's Boot CD

Since the system can no longer boot from the hard drive, you need to boot from an **external medium** (the Hiren's Boot CD ISO attached as a virtual CD).

Step	Action
1	Attach the Hiren's Boot CD `.iso` file to the VM's CD/DVD drive (VM Settings → CD/DVD → Use ISO image file)
2	Restart the VM — it boots from the CD instead of the corrupted hard drive
3	Select "**Mini Windows XP**" from Hiren's Boot CD menu
4	Open Hiren's Quick Launcher → Partition/Boot/MBR → Command Line → **MBRFix**
5	Run: `MbrFix.exe /drive 0 driveinfo` — displays drive information
6	Run: `Mbrfix /drive` — shows physical drive 0 info. Type `y` to confirm
7	Run: `Mbrfix /drive 0 fixmbr` — **repairs the MBR**. Type `y` to confirm
8	Restart the VM — it now boots normally into Windows
9	Verify: open SectEdit, check physical disc 0 — the `55 AA` signature should be restored

MBRFix Commands Summary

```
REM View drive information
MbrFix.exe /drive 0 driveinfo

REM Display drive info interactively
Mbrfix /drive
y
```

```
REM Repair the MBR signature and boot code
Mbrfix /drive 0 fixmbr
y
```

EXAM KEY POINT — DF Relevance of MBR Repair

- A suspect could deliberately corrupt the MBR to **hide data** by making the disk appear empty or unbootable.
- A DF investigator who understands MBR structure can **detect and repair** such corruption.
- Even without repairing, knowing the partition table format allows you to **manually locate partitions** from a hex dump of the disk image.
- Tools like MBRFix, `fdisk`, and `testdisk` can reconstruct or repair MBR data.

6 Logical vs. Extended Partitions

The lab mentions that additional "logical volumes" (logical/extended partitions) exist beyond the 4 primary partition limit:

	Primary Partition	Logical Volume / Extended Partition
Maximum count	4 (MBR limit)	Unlimited (within an extended partition)
Can boot OS?	**Yes**	**No** — an OS cannot be booted from a logical volume
Defined in	MBR partition table (16 bytes each)	Extended partition's own chain of Extended Boot Records (EBRs)
Drive letters	e.g. C:, D:	e.g. E:, F:, G:

If you need more than 4 partitions on an MBR disk, one primary partition is designated as an **extended partition**, which then contains multiple logical volumes inside it.

7 Additional Worked Example — Partition Size Calculation

For extra exam practice, here is the full calculation from Topic 3's lecture applied in the lab context:

Practice Problem: Partition Size from Hex **Given:** A partition entry's size field (bytes 12–15) reads: `4B 34 41 00`

Step 1 — Reverse for little-endian:

$$\texttt{4B 34 41 00} \xrightarrow{\text{reverse}} \texttt{00 41 34 4B}$$

Step 2 — Convert hex to decimal:
Break `0041344B` into individual hex digits and multiply by position:

$$0\times16^7 + 0\times16^6 + 4\times16^5 + 1\times16^4 + 3\times16^3 + 4\times16^2 + 4\times16^1 + 11\times16^0$$

$$= 0 + 0 + 4{,}194{,}304 + 65{,}536 + 12{,}288 + 1{,}024 + 64 + 11 = \mathbf{4{,}273{,}227} \text{ sectors}$$

Step 3 — Sectors to bytes:

$$4{,}273{,}227 \times 512 = 2{,}187{,}892{,}224 \text{ bytes}$$

Step 4 — Bytes to MB to GB:

$$\frac{2{,}187{,}892{,}224}{1{,}024} = 2{,}136{,}613 \text{ KB} \quad \rightarrow \quad \frac{2{,}136{,}613}{1{,}024} \approx 2{,}086 \text{ MB} \approx \mathbf{2} \text{ GB}$$

8 Complete Lab 04 Exam Quick-Reference

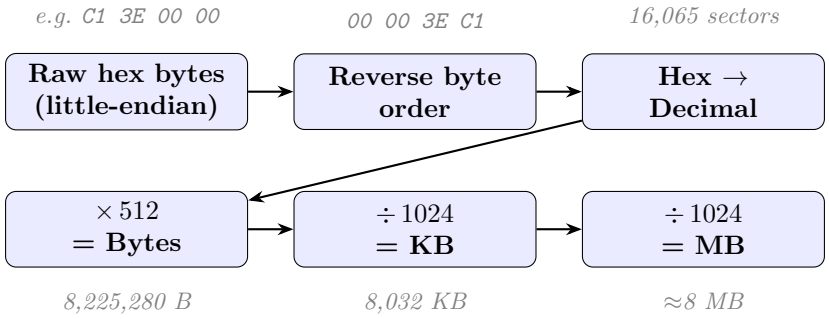
8.1 MBR Structure at a Glance

Section	Size	Byte Offsets
Boot Loader Code	446 bytes	0–445
Partition Entry 1	16 bytes	446–461
Partition Entry 2	16 bytes	462–477
Partition Entry 3	16 bytes	478–493
Partition Entry 4	16 bytes	494–509
MBR Signature (`55 AA`)	2 bytes	510–511
Total	**512 bytes**	**Sector 0**

8.2 Partition Entry Fields at a Glance

Offset	Size	Field	Key Values
0	1B	Boot Indicator	`80`=bootable, `00`=not
1–3	3B	Starting CHS	Cylinder/Head/Sector
4	1B	Partition Type	`07`=NTFS, `0B/0C`=FAT32, `83`=Linux
5–7	3B	Ending CHS	Cylinder/Head/Sector
8–11	4B	Starting Sector	Little-endian!
12–15	4B	Partition Size (sectors)	Little-endian!

8.3 Conversion Pipeline (Hex Partition Size → Human-Readable)



8.4 Tools Used in This Lab

Tool	Type	Purpose
VMware Player v17	Virtualisation	Run Windows XP VM safely
Disk Management	Windows built-in	Create/manage partitions via GUI
Roadkil's SectEdit	Sector editor	View and edit raw disk sectors (hex level)
Hiren's Boot CD	Bootable recovery ISO	Boot from CD when HDD won't boot; contains repair tools
MBRFix	CLI repair tool	View drive info and repair corrupted MBR

8.5 Key Concepts for Exam

Must-Know Summary

1. The MBR is **512 bytes** at **Sector 0**; it ends with `55 AA`.
2. It holds **4 partition entries** of **16 bytes each** (bytes 446–509).
3. Partition sizes and starting sectors are stored in **little-endian** — reverse the bytes before converting.
4. `0x80` = bootable; `0x00` = not bootable; `0x07` = NTFS.
5. Corrupting `55 AA` prevents booting; repairing it (e.g., with MBRFix) restores boot.
6. A DF investigator can **manually decode partition entries from a hex dump** to locate hidden or deleted partitions.
7. Logical/extended partitions exist **within** primary partitions and **cannot boot** an OS.
8. Conversion: hex (little-endian) → reverse → decimal sectors → $\times 512$ bytes → $\div 1024$ KB → $\div 1024$ MB.

End of Master Study Notes — Lab 04: Data Hiding & Data Recovery