

FORENSIC ANALYSIS OF RESILIENT FILE SYSTEM IN WINDOWS SERVER 2012



By

Aemun Iqbal

A thesis submitted to the faculty of Information Security Department, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment of the requirements for the degree of MS in Information Security

AUGUST 2016

SUPERVISOR CERTIFICATE

It is to certify that the final copy of MS thesis has been evaluated by me, found as per the specified format and error free.

Dated: _____

Thesis Supervisor
(**Col Dr Imran Rashid**)

ABSTRACT

File system by-large stores a wealth of information. It possesses numerous areas where sensitive information can be hidden or encrypted by criminals related to their crime so that when they are caught there is no information that can be used against them. There are many areas in a file system where information can be hidden. In order to find these hidden areas, one must know the working and layout of the file system. This working and layout can be known by conducting forensic analysis of the file system under consideration. Forensic analysis offers the potential for a more comprehensive assessment of file system. This thesis provides an in-depth analysis of the newly proposed Resilient File system. Resilient file system has not been analyzed forensically for its available artifacts and other modifications in its build up. Therefore it will be analyzed forensically and the gathered artifacts will provide sound knowledge of all the new things resilient file system is offering and the major changes that have been done in it.

DECLARATION

I hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification either at this institution or elsewhere.

DEDICATION

“In the name of Allah, the most Beneficent, the most Merciful”

I dedicate this thesis to my loving parents, sweet husband and teachers who supported and encouraged me at each step throughout my struggle.

ACKNOWLEDGMENT

I am highly thankful to Allah for giving me the strength, guidance and blessing in completing this thesis. I am highly indebted to my parents who constantly pushed and encouraged me to achieve what I have always aspired. Their continuous guidance and prayers helped me climb these steep steps and have made me what I am today. I am also thankful to my husband who always found solutions to my problems and never discouraged or stopped me from accomplishing what I have always dreamed to accomplish and making it easy for me every time. He has supported me throughout this endeavor. I would like to convey my gratitude to my supervisor, Dr. Imran Rashid, for his supervision and support. I am thankful to Asst Prof Mian Muhammad Waseem Iqbal. His invaluable help in the form of constructive comments, guidance and suggestions throughout the thesis work kept me motivated and helped me great deal in completing this research.

TABLE OF CONTENTS

1 INTRODUCTION

1.1	Overview	1
1.2	Motivation and Problem Statement	2
1.3	Objectives	2
1.4	Thesis Organization	2

2 LITERATURE REVIEW

2.1	Introduction	4
2.2	Level of research already carried out on the proposed topic	4
2.3	Summary	7

3 RESILIENT FILE SYSTEM

3.1	Introduction	8
3.2	Getting file system info using ‘fsutil’ command.....	8
3.3	Master Boot Record (MBR)	10
3.4	Volume boot record (VBR)	12
3.4.1	NTFS VBR.....	13
3.4.2	ReFS VBR.....	14
3.4.2.1	File System Recognition Structure FSRS	15
3.5	Master File Table	17
3.5.1	ReFS Master File Table	17
3.5.2	NTFS Master File Table	17
3.6	Attributes	18
3.6.1	ReFS attributes	18
3.6.1.1	File System Metadata	18
3.6.1.2	Security Descriptor Stream	19
3.6.1.3	Volume Direct IO File.....	20
3.6.1.4	\$I30 Index Attribute	21

3.6.1.5	Upcase Table	21
3.6.2	NTFS Attributes	22
3.7	Security identifier	23
3.7.1	Security Identifier in ReFS	23
3.7.2	Security Identifier in NTFS	24
4	TEST CASE SCENARIOS FOR ARTIFACTS GATHERING	
4.1	Introduction	25
4.2	.txt Scenarios	25
4.2.1	File creation.....	25
4.2.2	Permissions changed.....	27
4.2.3	Modifying content.....	28
4.2.4	Renaming txt file.....	31
4.2.5	Copying txt file	32
4.2.6	Deleting txt file	34
4.2.7	SHIFT + Delete txt file (permanent delete)	36
4.2.8	Scenario analysis for .txt file	39
4.3	Alternate data stream	40
4.3.1	ADS in the form of text	41
4.3.1.1	Scenario analysis for ADS in the form of text.....	46
4.3.2	ADS in the form of executable	46
4.3.3	ADS in the form of image file	47
4.4	.JPG scenarios	48
4.4.1	Copying jpg file	48
4.4.2	Renaming image file	50
4.4.3	Permissions changed.....	51
4.4.4	Deleting jpg file	52
4.4.5	SHIFT + Delete jpg file (permanent delete)	53

4.4.6	Scenario analysis for jpg file	54
4.5	Folder scenarios	55
4.5.1	Folder creation	55
4.5.2	Renaming folder.....	57
4.5.3	Permissions changed.....	59
4.5.4	Adding content.....	61
4.5.5	Compressing folder.....	65
4.5.6	Deleting folder	67
4.5.7	Shift+ Deleting a folder	69
4.6	Exploring deletion in ReFS in detail	71
4.6.1	Deletion in image file.....	71
4.6.2	Deletion in txt file	73
4.6.3	Deletion in doc file.....	74
4.7	Exploring the trimming of filename after simple deletion.....	76
4.8	1GB FILE	78
4.9	Comparison of ReFS artifacts with NTFS artifacts.....	81
5	CONCLUSION AND FUTURE WORK	
5.1	Overview	83
5.2	Overview of Research.....	83
5.3	Findings	83
5.4	Future Work	84
5.5	Conclusion.....	84
	BIBLIOGRAPHY.....	86

LIST OF TABLES

Table 3-1 : File system info using fsutil commands.....	9
Table 3-2: FSRS parts in Microsoft documentation	16
Table 4-1: Scenario analysis for .txt file	40
Table 4-2: Scenario analysis for ADS in the form of text	46
Table 4-3: Scenario analysis for jpg file	55
Table 4-4: Metadata offsets for folders	56
Table 4-5: Metadata changes for folders after renaming.....	58
Table 4-6: Metadata changes for folders after permission change.....	60
Table 4-7:Metadata changes after content addition in folder	62
Table 4-8: Metadata changes after image deletion	72
Table 4-9: Metadata changes after txt file deletion	73
Table 4-10: Metadata changes after doc file deletion.....	75
Table 4-11: Filename trimming after simple deletion	76

Introduction

1.1 Overview

A file system is the fundamental structure used by a computer for consolidating data on a hard disk. Before the installation of different programs and important data storage on the new hard disk, user has to first partition and format the disk using a file system. File system tells the storage area how the information will be stored and retrieved. Information would be large chunk of disorganized data without file system as there would be no system to tell where one piece of information ends and where the new one starts. Therefore file systems hold prime importance in computer forensics perspective as it stores all the information and the ways through which information can be addressed.

Windows provides three file system options, namely: NTFS, FAT32, and the older and rarely-used FAT (also known as FAT16). To date, NTFS is the most advanced feature-rich and widely used file system. With the advent of Windows 8, Microsoft engineered a new file system ReFS (Resilient File System), codenamed "Protogon", which makes use of NTFS as a base but at the same time, is built for new generation of storage technologies and circumstances. This new file system facilitates cloud storage the most as it ensures data and scales efficiently to handle data sets far larger than NTFS. The platform server used by resilient file system is Windows Server 2012. Windows Server 2012, codenamed "Windows Server 8", is the sixth release of Windows Server. It is the server version of Windows 8 and succeeds Windows Server 2008 R2. [1]

With the growing trends in technology, this new file system will soon be implemented in cloud environments as it is specifically intended for managing extremely large data volumes and its main focus is on data integrity. Therefore it is of fundamental importance to analyze it forensically so that storage areas can be identified, deleted files recovered and images carved.

This research will make an endeavor to conduct detailed forensic analysis of resilient file system designed for Windows 8 and released in Windows Server 2012. The important artifacts and useful findings gathered during study of this new generation file system will further help in malware detection and presentation of digital evidence in court of law during forensic investigations.

1.2 Motivation and Problem Statement

Very less information is available presently related to the newly proposed resilient file system. This information lists only certain features that the file system is offering that do not include in-depth analysis of the structure and working of file system. The manufacturers of this file system have only underlined major changes which this file system encompasses. Moreover resilient file system has not been analyzed forensically for its available artifacts and other modifications in its build up. Forensic analysis offers the potential for a more comprehensive assessment of file system. Therefore resilient file system will be analyzed forensically and the gathered artifacts will be compared to new technology file system to have sound knowledge of all the new things resilient file system is offering and the major changes that have been done in it.

1.3 Objectives

The main objectives of thesis are:

- Analysis of Resilient file system – its working and structure
- Forensically available artifacts in Resilient file system
- Comparison of forensically available artifacts of resilient file system with those of NTFS

1.4 Thesis Organization

The purpose of this research is to elaborate the working and structure of resilient file system. Information gathered during this research will be used by forensic investigators for collection of digital evidence.

Chapter 2 will discuss the literature review. This chapter carries the information that helps in understanding the underlying features this file system contains.

Chapter 3 contains the structure of resilient file system. What makes the file system base and attributes that are included in this file system.

Chapter 4 discusses all the scenarios through which working of resilient file system was uncovered. Various operations, such as copy, rename, modify, etc are applied to different files to know the way resilient file system works with files.

Chapter 5 concludes this research thesis with the recommendations for future work.

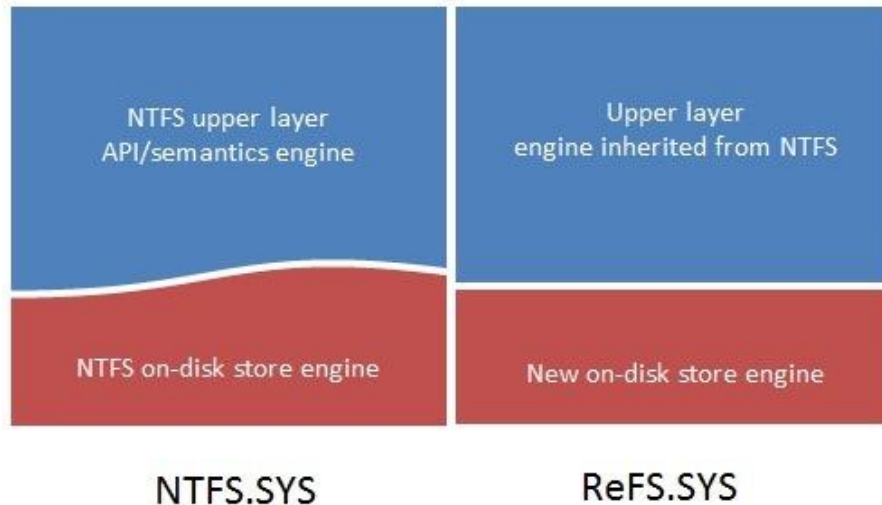
Literature Review

2.1 Introduction

Forensic analysis provides an in-depth view of the system under consideration. This analysis is very helpful for digital forensic examiners as it helps in extracting information from computers and compromised systems. This chapter provides information on the level of research that has already been carried out on the proposed topic.

2.2 Level of research already carried out on the proposed topic

- Microsoft's official site [2] lists Resilient File System under the New Features and Enhancements section in Windows 8 and Windows Server 2012 cookbook. The key features mentioned by Microsoft for ReFS are: integrity, availability, scalability, app compatibility and proactive error identification (data integrity scanner called scrubber).
- Resilient file system has not been developed from scratch, but it has been reimagined and built on the parts of NTFS. ReFS implements the file system interface (read, write, open, close, change notification, etc.), maintains in-memory file and volume state, enforces security, and maintains memory caching and synchronization for file data like NTFS. This reuse ensures a high degree of compatibility with the features of NTFS that are being carried forward. Underneath this reused portion, the NTFS version of the code-base uses a newly architected engine that implements on-disk structures such as the Master File Table (MFT) to represent files and directories. ReFS combines this reused code with a brand-new engine, where a significant portion of the innovation behind ReFS lies. [1] Graphically, it looks like this:



- Log structured file system implementation was rejected for ReFS. This approach is unsuitable for the type of general-purpose file system required by Windows. NTFS relies on a journal of transactions to ensure consistency on the disk. That approach updates metadata in-place on the disk and uses a journal on the side to keep track of changes that can be rolled back on errors and during recovery from a power loss. The main disadvantages of a journaling system are that writes can get randomized and, more importantly, the act of updating the disk can corrupt previously written metadata if power is lost at the time of the write, a problem commonly known as torn write.

To maximize reliability and eliminate torn writes, an allocate-on-write approach has been used, that never updates metadata in-place, but rather writes it to a different location.
- Data integrity is provided by creating checksums. All ReFS metadata is checksummed at the level of a B+ tree page, and the checksum is stored independently from the page itself. This allows ReFS to detect all forms of disk corruption, including lost and misdirected.

When the metadata for a ReFS directory is corrupted, subfolders and their associated files are automatically recovered. ReFS identifies and recovers the files while ReFS remains online. Unrecoverable corruption of the ReFS directory metadata affects only those files that are in the directory in which the corruption has occurred. [3]

- Data integrity resembling that of ReFS has been implemented in ZFS, which is a combined file system and logical volume manager designed by Sun Microsystems for use in their Solaris operating system. [4] It has been designed to protect the user's data on disk against silent data corruption which is achieved by using a (Fletcher-based) checksum or a (SHA-256) hash throughout the file system tree. [5] In addition to this ZFS has a repair tool called "scrub", which is used for file system validation and file system automatic repairing.
- ReFS can use checksums to detect if data has changed since last written and is able to detect and recover from corruption quickly. In fact, when data is written to disk, it is written to a new location on disk rather than over the top of existing data. Once successfully written, the file system can free the space used by the old data stream. ReFS is able to recover from corruption within the file system rapidly without limiting availability of the volume.[6] Additional protection of data streams can be done by enabling Integrity Streams. When configured to do so, checksums are used against written data and updates are done using copy-on-write. You may enable Integrity Streams on particular folders, volumes, or even granularly on a per-file basis.
- ReFS can handle up to 1 Yottabyte (YB).
 $1\text{GB} = 10^9$
 $1\text{YB} = 10^{24}$
 This is like 1 quadrillion GB. ReFS is built to scale to 262,000 Exabytes per volume, containing 18 quintillion files per volume. Compare this with NTFS which is built to handle only 16 Exabytes. [7]
- An algorithm of reconstructing directory tree above deleted files was proposed in this [8] for NTFS. Furthermore, through detailed analysis of the theory of internal structure of the NTFS file system, the storage principle of Data Runs in attribute 80 of MFT were presented.
- Methods of information hiding and detection in FAT and NTFS were explored in this paper [9]. Information hiding methods included hidden files and folders,

deleted files, hidden/ deleted partitions, alternate data streams (in NTFS), hiding data in areas such as slack space, file slack space, bad clusters and steganography. Forensic toolkits were referred to as the methods for detecting, recovering and viewing this hidden information in NTFS.

- Brian Carrier's book [10] lists EnCase by Guidance Software as the most widely used computer investigation software. The Forensic Toolkit (FTK) is Windows-based and can acquire and analyze disk, file system, and application data. ProDiscover by Technology Pathways, SMART by ASR Data (Linux based) and The Sleuth Kit (TSK, Unix based) are some of the toolkits that have been used for analyzing NTFS time and again.
- A carving method for the continuously allocated compressed files was proposed in this paper [11]. Most of the file carving tools cannot recover NTFS compressed files because NTFS supports a compression function for internal files. An algorithm for the implementation of this carving method for compressed files in corresponding tools was also introduced in this paper.

2.3 Summary

Current research on resilient file system does not provide information regarding the structure and functioning of the file system. Without knowing the base of file system and areas where data can be hidden, it is very difficult for a forensic analyst to find hidden information. Therefore there is a need to conduct forensic analysis of resilient file system which will be useful for future forensic investigations.

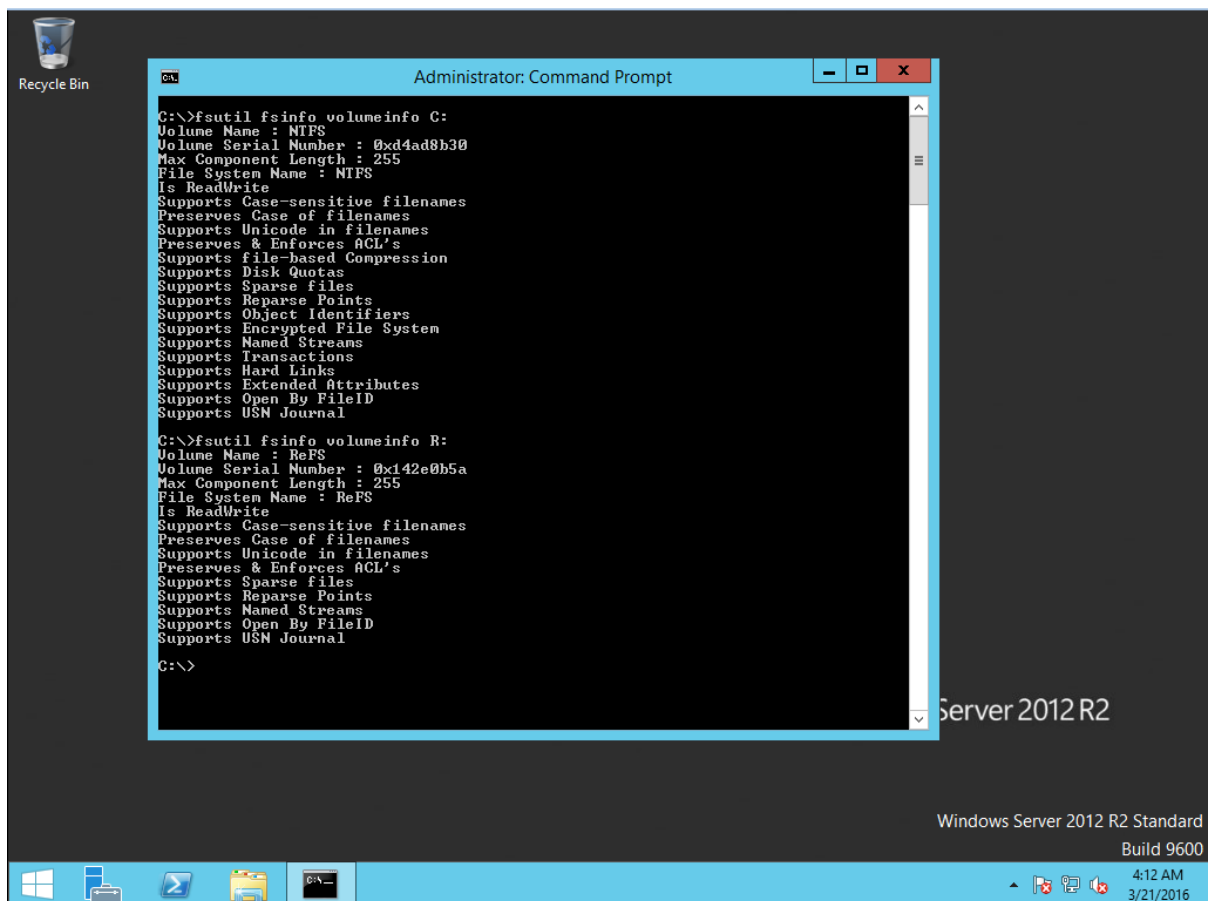
Resilient File System

3.1 Introduction

This chapter contains information related to resilient file system. Commands that were used to get information of the file system, master boot record and volume boot record that are file system specific have been explored in this chapter.

3.2 Getting file system info using 'fsutil' command

First of all windows command line command 'fsutil' is used to get basic file system information. Following are the results and snapshots:



```

Administrator: Command Prompt
C:\>fsutil fsinfo volumeinfo C:
Volume Name : NTFS
Volume Serial Number : 0xd4ad8b30
Max Component Length : 255
File System Name : NTFS
Is ReadWrite
Supports Case-sensitive filenames
Preserves Case of filenames
Supports Unicode in filenames
Preserves & Enforces ACL's
Supports file-based Compression
Supports Disk Quotas
Supports Sparse files
Supports Reparse Points
Supports Object Identifiers
Supports Encrypted File System
Supports Named Streams
Supports Transactions
Supports Hard Links
Supports Extended Attributes
Supports Open By FileID
Supports USN Journal

C:\>fsutil fsinfo volumeinfo R:
Volume Name : ReFS
Volume Serial Number : 0x142e0b5a
Max Component Length : 255
File System Name : ReFS
Is ReadWrite
Supports Case-sensitive filenames
Preserves Case of filenames
Supports Unicode in filenames
Preserves & Enforces ACL's
Supports Sparse Files
Supports Reparse Points
Supports Named Streams
Supports Open By FileID
Supports USN Journal

C:\>
  
```

Windows Server 2012 R2 Standard
Build 9600
4:12 AM
3/21/2016

Following are the results for *fsutil fsinfo sectorinfo* and *fsutil fsinfo volumeinfo* commands applied to the NTFS and ReFS drives respectively:

Command: <i>fsutil fsinfo sectorinfo</i>	NTFS	ReFS
LogicalBytesPerSector	512	512
PhysicalBytesPerSectorForAtomicity	512	512
PhysicalBytesPerSectorForPerformance	512	512
FileSystemEffectivePhysicalBytesPerSectorForAtomicity	512	512
Device Alignment	Aligned<0x000>	Aligned<0x000>
Partition alignment on device	Aligned<0x000>	Aligned<0x000>
Command: <i>fsutil fsinfo volumeinfo</i>		
Volume Name	NTFS	ReFS
Volume Serial Number	0xd4ad8b30	0x142e0b5a
Max Component Length	255	255
File System Name	NTFS	ReFS
Is ReadWrite	Yes	Yes
Supports Case-sensitive filename	Yes	Yes
Preserves Case of filenames	Yes	Yes
Supports Unicode in filenames	Yes	Yes
Preserves & Enforces ACL's	Yes	Yes
Supports file-based Compression	Yes	No
Supports Disk Quotas	Yes	No
Supports Sparse files	Yes	Yes
Supports Reparse Points	Yes	Yes
Supports Object Identifiers	Yes	No
Supports Encrypted File System	Yes	No
Supports Named Streams	Yes	Yes
Supports Transactions	Yes	No
Supports Hard Links	Yes	No
Supports Extended Attributes	Yes	No
Supports Open By FileID	Yes	Yes
Supports USN Journal	Yes	Yes

Table 3-1: File system info using fsutil commands

3.3 Master Boot Record (MBR)

Windows Server 2012 R2 contains one disk drive which has been formatted into two partitions. First partition is formatted using NTFS (primary partition) and second partition is formatted using ReFS. Starting part of the drive is by default system reserved. From here the hexadecimal of each partition will be analyzed through WinHex which is a hexadecimal editor.

Physical disk drive is opened in the hexadecimal editor for looking at the master boot record.

The MBR is located at the very first, starting sector of a physical disk. A generic MBR has the following three parts:

1. The Bootstrap Code Area/Bootloader
2. Partition Table
3. Boot Record Signature

The 512 bytes of MBR are distributed as:

512 bytes = 446 bytes (bootstrap code) + 64 bytes (partition table) + 2 bytes (boot signature). These bytes are shown here:

WinHex - [Hard disk 0]

Search Navigation View Tools Specialist Options Window Help

Hard disk 0 Drive C:

Partitioning style: MBR

Name	Ext.	Size	Created	Modified	Record changed	Attr.	1st sector
Start sectors		1.0 MB					0

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
00000000	33	C0	8E	D0	BC	00	7C	8E	C0	8E	D8	BE	00	7C	BF	00	3A ZD4 ZAZ0% z
00000010	06	B9	00	02	FC	F3	A4	50	68	1C	06	CB	FB	B9	04	00	' uoxPh Eû'
00000020	BD	BE	07	80	7E	00	00	7C	0B	0F	85	0E	01	83	C5	10	%€~ ... fÅ
00000030	E2	F1	CD	18	88	56	00	55	C6	46	11	05	C6	46	10	00	ání 'V UZF EF
00000040	B4	41	BB	AA	55	CD	13	5D	72	0F	81	FB	55	AA	75	09	'A»*UÍ r ûU'u
00000050	F7	C1	01	00	74	03	FE	46	10	66	60	80	7E	10	00	74	-Á t pF f'€~ t
00000060	26	66	68	00	00	00	00	66	FF	76	08	68	00	00	68	00	&fh fÿv h h
00000070	7C	68	01	00	68	10	00	B4	42	8A	56	00	8B	F4	CD	13	h h 'BŠV <óÍ
00000080	9F	83	C4	10	9E	EB	14	B8	01	02	BB	00	7C	8A	56	00	ÿfÅ žë . » ŠV
00000090	8A	76	01	8A	4E	02	8A	6E	03	CD	13	66	61	73	1C	FE	Šv ŠN Šn Í fas p
000000A0	4E	11	75	0C	80	7E	00	80	0F	84	8A	00	B2	80	EB	84	N u €~ € „Š *€ë„
000000B0	55	32	E4	8A	56	00	CD	13	5D	EB	9E	81	3E	FE	7D	55	U2äŠV í jež >p}U
000000C0	AA	75	6E	FF	76	00	E8	8D	00	75	17	FA	B0	D1	E6	64	*unÿv è u ú°Næd
000000D0	E8	83	00	B0	DF	E6	60	E8	7C	00	B0	FF	E6	64	E8	75	èf °Bæ'èl °ÿædèn
000000E0	00	FB	B8	00	BB	CD	1A	66	23	C0	75	3B	66	81	FB	54	û, »Í f#Au;f ûT
000000F0	43	50	41	75	32	81	F9	02	01	72	2C	66	68	07	BB	00	CPAu2 ù r,fh »
00000100	00	66	68	00	02	00	00	66	68	08	00	00	00	66	53	66	fh fh fsf
00000110	53	66	55	66	68	00	00	00	00	66	68	00	7C	00	00	66	SfUfh fh f
00000120	61	68	00	00	07	CD	1A	5A	32	F6	EA	00	7C	00	00	CD	ah í Z2ôè í
00000130	18	A0	B7	07	EB	08	A0	B6	07	EB	03	A0	B5	07	32	E4	· è ¶ è µ 2ä
00000140	05	00	07	8B	F0	AC	3C	00	74	09	BB	07	00	B4	0E	CD	<8-< t » ' í
00000150	10	EB	F2	F4	EB	FD	2B	C9	E4	64	EB	00	24	02	E0	F8	èòóëÿ+Éädè \$ àø
00000160	24	02	C3	49	6E	76	61	6C	69	64	20	70	61	72	74	69	\$ ÅInvalid parti
00000170	74	69	6F	6E	20	74	61	62	6C	65	00	45	72	72	6F	72	tion table Error
00000180	20	6C	6F	61	64	69	6E	67	20	6F	70	65	72	61	74	69	loading operati
00000190	6E	67	20	73	79	73	74	65	6D	00	4D	69	73	73	69	6E	ng system Missi
000001A0	67	20	6F	70	65	72	61	74	69	6E	67	20	73	79	73	74	g operating syst
000001B0	65	6D	00	00	00	63	7B	9A	6A	BC	95	07	00	00	80	20	em c(šj4* €
000001C0	21	00	07	BE	12	2C	00	08	00	00	00	F0	0A	00	00	BE	! %, , ø %
000001D0	13	2C	07	FE	FF	FF	00	F8	0A	00	00	40	28	01	00	FE	, pÿÿ ø @ (p
000001E0	FF	FF	07	FE	FF	FF	00	38	33	01	00	B8	EC	01	00	00	ÿÿ pÿÿ 83 ,i
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	ÿ*
00000200	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

- 0x00000000-0x000001BD is the bootstrap code area.
- 0x000001BE-0x000001FD is the partition table. And
- 0x000001FE-0x000001FF is the boot signature

At 0x000001BE, the value 80 at the start of partition table signifies that the partition is bootable; otherwise there is a value of 00 if the partition is not bootable. This bootable partition is the NTFS partition. The byte at offset 0x000001C2 represents the partition's file system and therefore we can assume that it will be unique for every file system. 07 is an indication for NTFS. However this byte is the same in the next partition (byte at offset 000001D2) which shows that ReFS is built on NTFS.

0000001A0	67 20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74	g operating syst
0000001B0	65 6D 00 00 00 63 7B 9A 6A BC 95 07 00 00 80 20	em c{šj4• €
0000001C0	21 00 07 BE 12 2C 00 08 00 00 <u>00 F0 0A 00</u> 00 BE	! % , ø %
0000001D0	13 2C 07 FE FF FF 00 F8 0A 00 <u>00 40 28 01</u> 00 FE	, pÿÿ ø @(p
0000001E0	FF FF 07 FE FF FF 00 38 33 01 <u>00 B8 EC 01</u> 00 00	ÿÿ pÿÿ 83 ,i
0000001F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA	U*
000000200	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

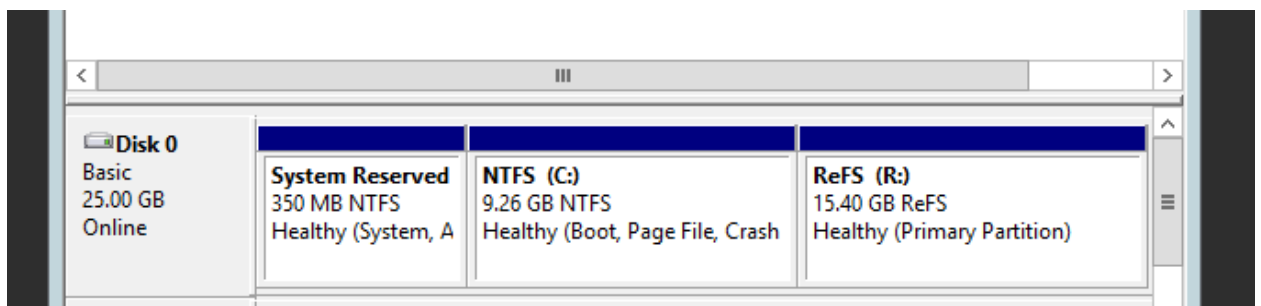
The partition table has four basic partitions, where the last partition is used as an extended partition to support further partitioning. We can see in the above partition table that our last partition is not being used. Size of each partition can be calculated from the partition table.

00 F0 0A 00 (underlined in red) is in Little Endian. Converting it to Big Endian it becomes 00 0A F0 00 which are 716800 bytes/sectors (starts at sector 2048) when converted into decimal. Thus the first partition is of **350MB** in size.

00 40 28 01 becomes 01 28 40 00 which are 19415040 sectors. Second partition is of **9480MB = 9.25GB**

00 B8 EC 01 becomes 01 EC B8 00 which are 32290816 sectors. Third partition is of **15767MB = 15.39GB**

Further verifying our results using the windows disk management:



3.4 Volume boot record (VBR)

The volume boot record exists in the first sectors of the partition. As we have two partitions each for NTFS and ReFS we examine and compare their first sectors for the volume boot record.

3.4.1 NTFS VBR

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Hex	ASCII	
00000000	EB	52	90	4E	54	46	53	20	20	20	20	00	02	08	00	00	EB 52 90 4E 54 46 53 20 20 20 20 00 02 08 00 00	eR NTFS	
00000010	00	00	00	00	00	00	00	00	FF	3F	28	01	00	00	00	00	00 00 00 00 00 00 00 00 FF 3F 28 01 00 00 00 00	ø ? ý ø	
00000020	00	00	00	00	80	00	80	00	FF	3F	28	01	00	00	00	00	00 00 00 00 80 00 80 00 FF 3F 28 01 00 00 00 00	€ € ý?(
00000030	00	00	0C	00	00	00	00	00	02	00	00	00	00	00	00	00	00 00 0C 00 00 00 00 00 02 00 00 00 00 00 00 00	ö	0<-Ôš-ÔH
00000040	F6	00	00	00	01	00	00	00	30	8B	AD	D4	9A	AD	D4	48	F6 00 00 00 01 00 00 00 30 8B AD D4 9A AD D4 48	ú3ĂŽĐw úhĂ	
00000050	00	00	00	00	FA	33	C0	8E	D0	BC	00	7C	FB	68	C0	07	00 00 00 00 FA 33 C0 8E D0 BC 00 7C FB 68 C0 07	hf Ě^ f > N	
00000060	1F	1E	68	66	00	CB	88	16	0E	00	66	81	3E	03	00	4E	1F 1E 68 66 00 CB 88 16 0E 00 66 81 3E 03 00 4E	TFSu 'As'Uí r ú	
00000070	54	46	53	75	15	B4	41	BB	AA	55	CD	13	72	0C	81	FB	54 46 53 75 15 B4 41 BB AA 55 CD 13 72 0C 81 FB	U*u ÷Á u éY fi	
00000080	55	AA	75	06	F7	C1	01	00	75	03	E9	DD	00	1E	83	EC	55 AA 75 06 F7 C1 01 00 75 03 E9 DD 00 1E 83 EC	h 'HŠ <ô í	
00000090	18	68	1A	00	B4	48	8A	16	0E	00	8B	F4	16	1F	CD	13	18 68 1A 00 B4 48 8A 16 0E 00 8B F4 16 1F CD 13	ŸfĂ žX rá; uŮĚ	
000000A0	9F	83	C4	18	9E	58	1F	72	E1	3B	06	0B	00	75	DB	A3	9F 83 C4 18 9E 58 1F 72 E1 3B 06 0B 00 75 DB A3	Á. Z3Ů^ +Ě	
000000B0	0F	00	C1	2E	0F	00	04	1E	5A	33	DB	B9	00	20	2B	C8	0F 00 C1 2E 0F 00 04 1E 5A 33 DB B9 00 20 2B C8	fÿ ŽĂÿ è	
000000C0	66	FF	06	11	00	03	16	0F	00	8E	C2	FF	06	16	00	E8	66 FF 06 11 00 03 16 0F 00 8E C2 FF 06 16 00 E8	K +Ěwi, »Í f#Au-	
000000D0	4B	00	2B	C8	77	EF	B8	00	BB	CD	1A	66	23	C0	75	2D	4B 00 2B C8 77 EF B8 00 BB CD 1A 66 23 C0 75 2D	f ůTCPAu\$ ů r	
000000E0	66	81	FB	54	43	50	41	75	24	81	F9	02	01	72	1E	16	66 81 FB 54 43 50 41 75 24 81 F9 02 01 72 1E 16	h » hR h fSfSf	
000000F0	68	07	BB	16	68	52	11	16	68	09	00	66	53	66	53	66	68 07 BB 16 68 52 11 16 68 09 00 66 53 66 53 66	U h, fa í 3Ăž	
00000100	55	16	16	16	68	B8	01	66	61	0E	07	CD	1A	33	C0	BF	55 16 16 16 68 B8 01 66 61 0E 07 CD 1A 33 C0 BF	'ô ůó^ép f`	
00000110	0A	13	B9	F6	0C	FC	F3	AA	E9	FE	01	90	90	66	60	1E	0A 13 B9 F6 0C FC F3 AA E9 FE 01 90 90 66 60 1E	f; f fh	
00000120	06	66	A1	11	00	66	03	06	1C	00	1E	66	68	00	00	00	06 66 A1 11 00 66 03 06 1C 00 1E 66 68 00 00 00	fP Sh h 'BŠ	
00000130	00	66	50	06	53	68	01	00	68	10	00	B4	42	8A	16	0E	00 66 50 06 53 68 01 00 68 10 00 B4 42 8A 16 0E	<ôí fY[ZfYfY	
00000140	00	16	1F	8B	F4	CD	13	66	59	5B	5A	66	59	66	59	1F	00 16 1F 8B F4 CD 13 66 59 5B 5A 66 59 66 59 1F	, fÿ ŽĂÿ	
00000150	0F	82	16	00	66	FF	06	11	00	03	16	0F	00	8E	C2	FF	0F 82 16 00 66 FF 06 11 00 03 16 0F 00 8E C2 FF	u* faĂ;ô è	
00000160	0E	16	00	75	BC	07	1F	66	61	C3	A1	F6	01	E8	09	00	0E 16 00 75 BC 07 1F 66 61 C3 A1 F6 01 E8 09 00	;ú è ôëý<š-< t	
00000170	A1	FA	01	E8	03	00	F4	EB	FD	8B	F0	AC	3C	00	74	09	A1 FA 01 E8 03 00 F4 EB FD 8B F0 AC 3C 00 74 09	' » í èòĂ A di	
00000180	B4	0E	BB	07	00	CD	10	EB	F2	C3	0D	0A	41	20	64	69	B4 0E BB 07 00 CD 10 EB F2 C3 0D 0A 41 20 64 69	sk read error oc	
00000190	73	6B	20	72	65	61	64	20	65	72	72	6F	72	20	6F	63	73 6B 20 72 65 61 64 20 65 72 72 6F 72 20 6F 63	curred BOOTMGR	
000001A0	63	75	72	72	65	64	00	0D	0A	42	4F	4F	54	4D	47	52	63 75 72 72 65 64 00 0D 0A 42 4F 4F 54 4D 47 52	is compressed	
000001B0	20	69	73	20	63	6F	6D	70	72	65	73	73	65	64	00	0D	20 69 73 20 63 6F 6D 70 72 65 73 73 65 64 00 0D	Press Ctrl+Alt+	
000001C0	0A	50	72	65	73	73	20	43	74	72	6C	2B	41	6C	74	2B	0A 50 72 65 73 73 20 43 74 72 6C 2B 41 6C 74 2B	Del to restart	
000001D0	44	65	6C	20	74	6F	20	72	65	73	74	61	72	74	0D	0A	44 65 6C 20 74 6F 20 72 65 73 74 61 72 74 0D 0A	š s ž Ů*	
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00 00 00 00 00 00 00	B O O T M G R	
000001F0	00	00	00	00	00	00	00	8A	A7	01	BF	01	00	00	55	AA	00 00 00 00 00 00 8A 01 A7 01 BF 01 00 00 55 AA	\$ I 3 0 Ô \$	
00000200	07	00	42	00	4F	00	4F	00	54	00	4D	00	47	00	52	00	07 00 42 00 4F 00 4F 00 54 00 4D 00 47 00 52 00		
00000210	04	00	24	00	49	00	33	00	30	00	00	D4	00	00	00	24	04 00 24 00 49 00 33 00 30 00 00 D4 00 00 00 24		

The first three bytes in the volume boot record are the jump instructions which are used to jump to the executable assembly code which resides within this block of VBR. The next eight bytes (0x03 to 0x0A) are the OEM ID or system name. System name is followed by BPB (BIOS Parameter Block). The last 125 bytes contain error messages and signature ID or magic number. [12]

Following this above sector is the code for BOOTMGR (shown in the figure below). This code tells the machine which operating system should be loaded on it. Error message can be seen in this sector after 0D 0A.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000001D0	44	65	6C	20	74	6F	20	72	65	73	74	61	72	74	0D	0A	Del to restart
0000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	Š Š ě U*
0000001F0	00	00	00	00	00	00	8A	01	A7	01	BF	01	00	00	55	AA	B O O T M G R
000000200	07	00	42	00	4F	00	4F	00	54	00	4D	00	47	00	52	00	\$ I 3 0 Ô \$
000000210	04	00	24	00	49	00	33	00	30	00	00	D4	00	00	00	24	
000000220	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000230	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000240	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000250	00	00	00	00	00	00	E9	C0	00	90	05	00	4E	00	54	00	éÀ N T
000000260	4C	00	44	00	52	00	07	00	42	00	4F	00	4F	00	54	00	L D R B O O T
000000270	54	00	47	00	54	00	07	00	42	00	4F	00	4F	00	54	00	T G T B O O T
000000280	4E	00	58	00	54	00	00	00	00	00	00	00	00	00	00	00	N X T
000000290	00	00	00	00	00	00	00	00	00	00	0D	0A	41	6E	20	6F	An o
0000002A0	70	65	72	61	74	69	6E	67	20	73	79	73	74	65	6D	20	perating system
0000002B0	77	61	73	6E	27	74	20	66	6F	75	6E	64	2E	20	54	72	wasn't found. Tr
0000002C0	79	20	64	69	73	63	6F	6E	6E	65	63	74	69	6E	67	20	y disconnecting
0000002D0	61	6E	79	20	64	72	69	76	65	73	20	74	68	61	74	20	any drives that
0000002E0	64	6F	6E	27	74	0D	0A	63	6F	6E	74	61	69	6E	20	61	don't contain a
0000002F0	6E	20	6F	70	65	72	61	74	69	6E	67	20	73	79	73	74	n operating syst
000000300	65	6D	2E	00	00	00	00	00	00	00	00	00	00	00	00	00	em.
000000310	00	00	00	00	00	00	00	9A	02	66	0F	B7	06	0B	00	66	š f . f
000000320	0F	B6	1E	0D	00	66	F7	E3	66	A3	52	02	66	8B	0E	40	ŕ f÷áfflR f< @
000000330	00	80	F9	00	0F	8F	0E	00	F6	D9	66	B8	01	00	00	00	€ù öÜf,
000000340	66	D3	E0	EB	08	90	66	A1	52	02	66	F7	E1	66	A3	86	fóäë f;R f÷áffl+
000000350	02	66	0F	B7	1E	0B	00	66	33	D2	66	F7	F3	66	A3	56	f . f3Öf÷ófflV
000000360	02	E8	A2	04	66	8B	0E	4E	02	66	89	0E	26	02	66	03	èc f< N f% & f
000000370	0E	86	02	66	89	0E	2A	02	66	03	0E	86	02	66	89	0E	† f% * f † f%
000000380	0E	00	66	00	0F	86	00	66	00	0F	00	00	66	00	0F	00	f + f% \ f +

3.4.2 ReFS VBR

Looking at the starting sectors of the ReFS partition, it is observed that only 64 bytes are being used which is very small as compared to that used by the NTFS.

Also ReFS does not have boot code and Bios parameter block BPB, as it is apparent from the snapshot taken. It might be because ReFS is not bootable.

OEM ID of four bytes is present on the same location as it was in NTFS, after the first three jump instructions, but the jump instructions are null in ReFS case. It might be possible that when Microsoft makes ReFS bootable then the values of these three bytes will be changed.

There is no signature ID 55 AA at the end of the sector.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	00	00	00	52	65	46	53	00	00	00	00	00	00	00	00	00
00000010	46	53	52	53	00	02	6C	D6	00	00	EC	01	00	00	00	00
00000020	00	02	00	00	80	00	00	00	01	02	00	00	0A	00	00	00
00000030	00	00	00	00	00	00	00	00	5A	0B	2E	14	2D	2E	14	DC
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000200	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000210	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

ReFS
FSRS 1Ö i
€
z . - . ü

OEM ID or System name

3.4.2.1 File System Recognition Structure FSRS

There is text FSRS in place of BPB in ReFS which stands for File System Recognition Structure. Microsoft's definition on FSRS reveals that "a file system data structure and file system recognition APIs that may allow an operating system to identify a partition of a storage device as having a valid file system, even if the operating system does not know how to access the file system a priori". [13]

Its main goal is to give Windows an additional option to identify an otherwise unrecognized file system.

This is achieved by writing a data structure on the logical disk sector zero which would then be recognized by the operating system and notify the user that the media contains a

valid but unrecognized file system and is not a RAW volume if the drivers for the file system are not installed. [14]

FILE_SYSTEM_RECOGNITION_STRUCTURE is the data structure that tells the operating system to achieve its goals regarding the recognition of an otherwise unrecognized file system stored in the volume's boot sector (logical disk sector zero). Checksum validation code is also stored within this data structure. At the application level, file system recognition is achieved through the use of FSCTL_QUERY_FILE_SYSTEM_RECOGNITION device I/O control code.

FILE_SYSTEM_RECOGNITION_STRUCTURE has the following parts defined in the Microsoft documentation:

Type	Offset	Length	Contents	Description
Jmp	0x00	3 bytes	00 00 00	Jump instruction
FsName	0x03	4 bytes	52 65 46 53	File system name
MustBeZero	0x07	9 bytes	00 00 00 00 00 00 00 00 00	Reserved space that contains all zeros
Identifier	0x10	16 bytes	46 53 52 53 00 02 6C D6 00 00 EC 01 00 00 00 00	Structure identifier arranged in Little endian.
Length	0x20	2 bytes	00 02	The number of bytes in the structure
Checksum	0x16	2 bytes		Checksum calculated from FsName to last byte, excluding Jmp and Checksum

Table 3-2: FSRS parts in Microsoft documentation

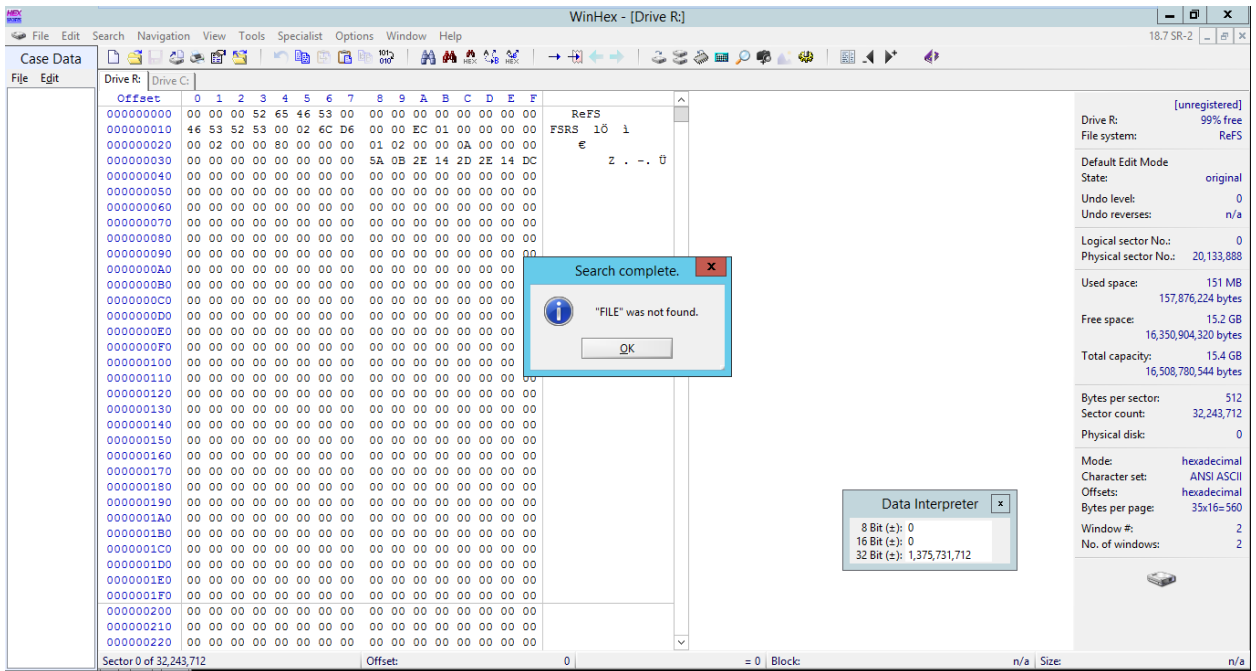
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
266700000	00	00	00	52	65	46	53	00	00	00	00	00	00	00	00	00	ReFS
266700010	46	53	52	53	00	02	6C	D6	00	00	EC	01	00	00	00	00	FSRS lÖ i
266700020	00	02	00	00	80	00	00	00	01	02	00	00	0A	00	00	00	€
266700030	00	00	00	00	00	00	00	00	5A	0B	2E	14	2D	2E	14	DC	z . - . Ü
266700040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

3.5 Master File Table

This section of the chapter describes and compares the master file table of NTFS as well as ReFS. Master file table is the main area where all the records of files and folders exist in NTFS. Let's look if resilient file system makes use of such structure or has any other thing in store for us.

3.5.1 ReFS Master File Table

ReFS does not have an MFT and therefore no entries related to the master file table either. On searching for 'FILE' or 'FILE0' entries (that signify the start of every MFT entry) in the file system, the result returned is as follows:



3.5.2 NTFS Master File Table

NTFS has a master file table which contains entries of every file on the drive along with other general information. Each entry starts with FILE or FILE0 and is 1024 bytes in size. NTFS MFT has sixteen reserved MFT entries for file system metadata files. They are:

0. \$MFT - Master file table.
1. \$MftMirr - Master file table mirror.
2. \$LogFile - Log file.

3. \$Volume - Volume contains information such as the volume label and the volume version.
4. \$AttrDef - Attribute definitions.
5. “.” Or \$ - The root folder
6. \$Bitmap - Cluster bitmap which represents the volume by showing free and unused clusters.
7. \$Boot - Boot sector, includes the BIOS Parameter Block used to mount the volume and additional bootstrap loader code used if the volume is bootable.
8. \$BadClus - Bad cluster file, which contains a list of bad clusters for the volume.
9. \$Secure - Security file which contains unique security descriptors for all files within a volume.
10. \$Upcase - Upcase table which converts lowercase characters to matching Unicode uppercase characters.
11. \$Extend - NTFS extension file, which is used for future use.

12 – 15 are reserved for future use.

3.6 Attributes

File system views each file (or folder) as a set of file attributes. Elements such as the file's name, its security information, and even its data, are all file attributes. Each attribute is identified by an attribute type code and, optionally, an attribute name. In this section ReFS and NTFS attributes are discussed.

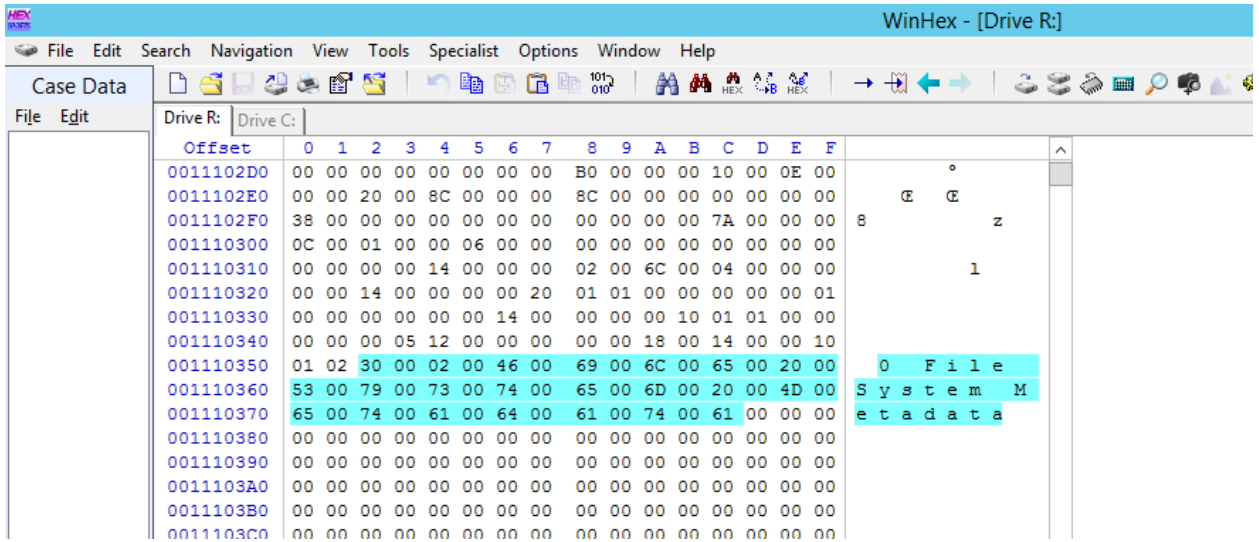
3.6.1 ReFS attributes

ReFS like NTFS, contains some attributes which are not similar to NTFS attributes but have something in common to them. Following attributes are found on clean ReFS drive:

3.6.1.1 File System Metadata

Metadata is data about data. File system metadata contains internal information (data) about the data stored on the volume. The elegance of the metadata system is that by storing

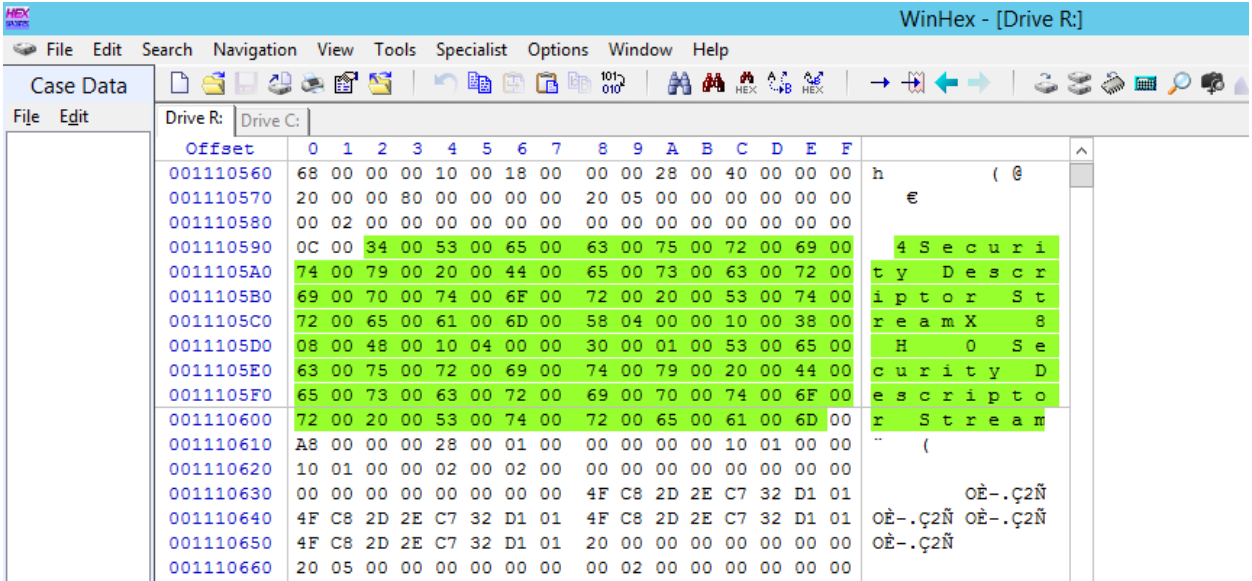
internal information in files, it is possible to expand on the capabilities of the file system. On examination of ReFS drive, file system metadata attribute is found as shown below:



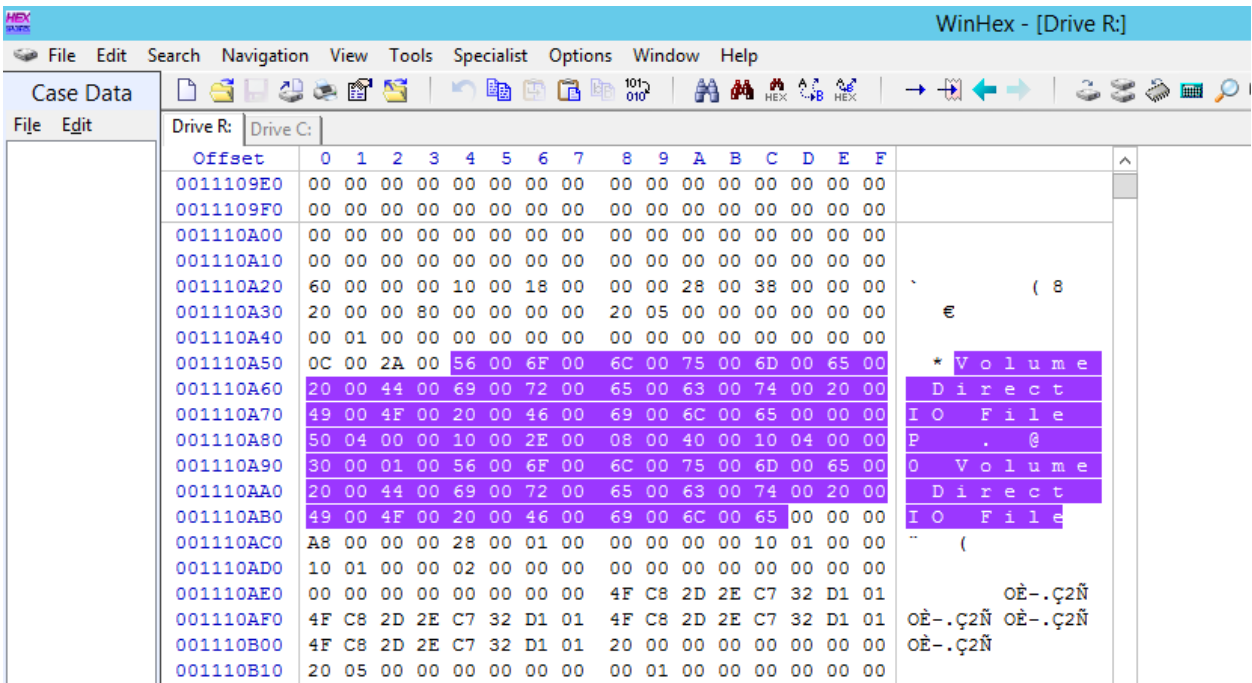
This 'File System Metadata' block appears in the same area where other attributes appear. This block can be used to contain the file information and data related to it.

3.6.1.2 Security Descriptor Stream

The attribute 'Security Descriptor Stream' at offset 0x1110592 in ReFS is analogous to '\$SECURITY_DESCRIPTOR' in NTFS. In NTFS security descriptor is a data structure that controls the access control and security properties of the file. From here, it can be inferred that the security descriptor stream in ReFS would also be used for access control, security and ownership related things.

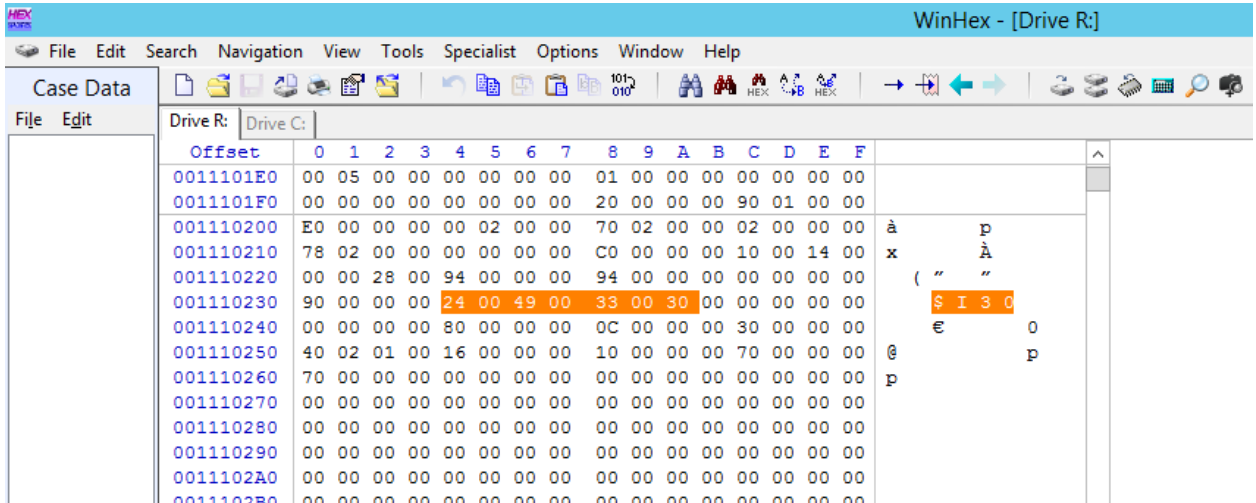


3.6.1.3 Volume Direct IO File



‘Volume Direct IO File’ attribute in ReFS can be compared to ‘\$VOLUME_VERSION, \$VOLUME_NAME and \$VOLUME_INFORMATION’ attributes present in NTFS. These volume attributes in NTFS are self-explanatory. It can be inferred from here that ReFS ‘Volume Direct IO File’ attribute will also be used, more or less, for the same purpose.

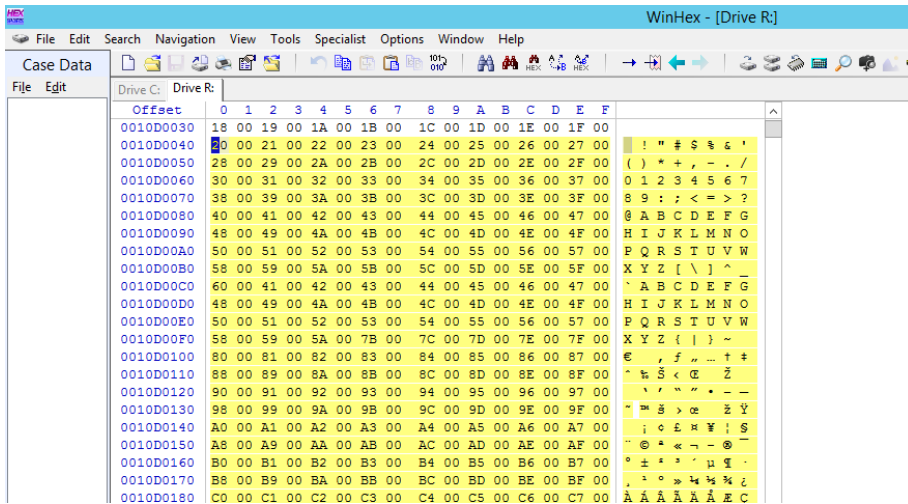
3.6.1.4 \$I30 Index Attribute



\$I30 is defined as “An INDX buffer in the NTFS file system that tracks the contents of a folder”. [15] It will be used for the same purpose in ReFS too.

3.6.1.5 Uppcase Table

ReFS has an Uppcase table at offset 0x10D0040 on clean drive, which is similar to the Uppcase table found in NTFS (file system metadata file number 10). It contains each uppercase character in the Unicode alphabet. Like in NTFS, ReFS also makes use of this Uppcase table to convert lowercase characters to uppercase characters. Structure of the ReFS Uppcase table is the same as in NTFS as both are used for the same purpose and Unicode alphabets are independent of file systems. Uppcase table in ReFS can be seen as follows:

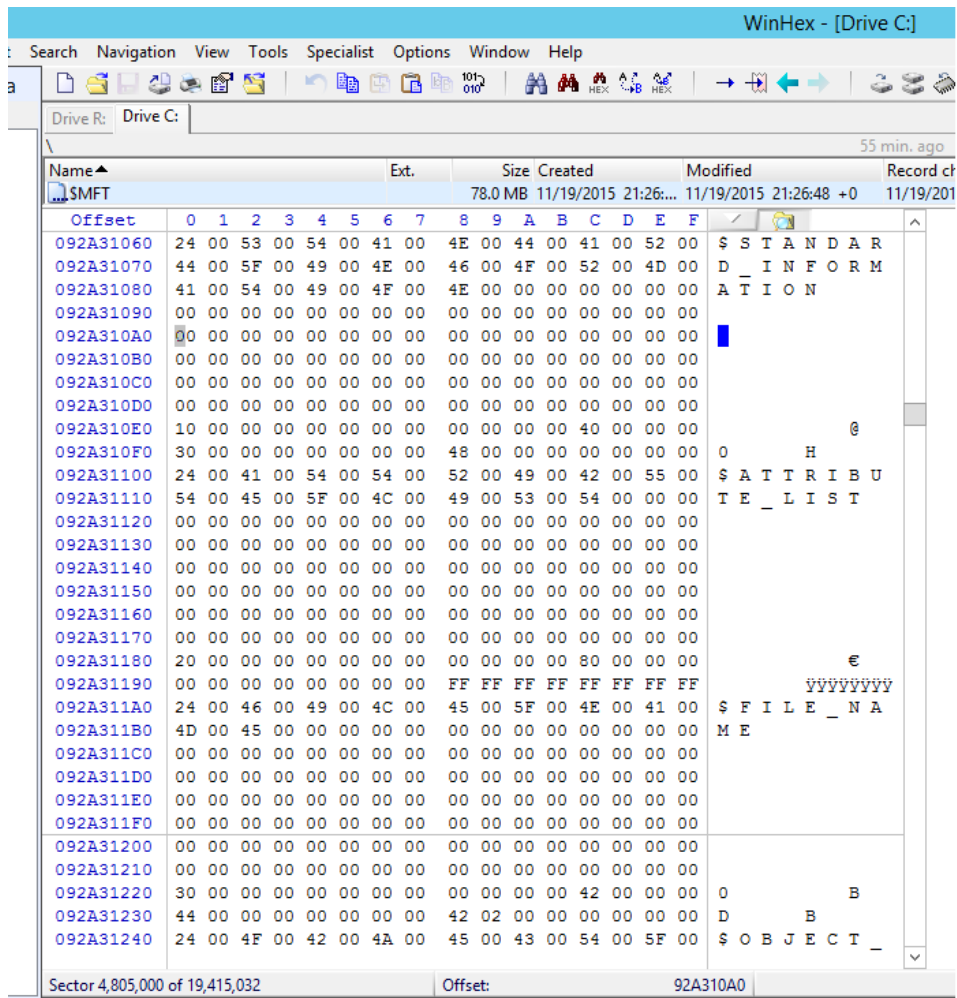


3.6.2 NTFS Attributes

“File system forensic analysis” book by Brian Carrier lists following attributes as standard attribute types:

\$STANDARD_INFORMATION, \$ATTRIBUTE_LIST, \$FILE_NAME,
 \$VOLUME_VERSION, \$OBJECT_ID, \$SECURITY_DESCRIPTOR,
 \$VOLUME_NAME, \$VOLUME_INFORMATION, \$DATA, \$INDEX_ROOT,
 \$INDEX_ALLOCATION, \$BITMAP, \$SYMBOLIC_LINK, \$REPARSE_POINT,
 \$EA_INFORMATION, \$EA, \$LOGGED_UTILITY_STREAM.

These attributes are data structures that store specific types of data. Each attribute has its own internal structure and they are self-explanatory. They are found at different offsets throughout the file system.



3.7 Security identifier

A Security Identifier (commonly abbreviated SID) is a unique, immutable identifier of a user, user group, or other security principal. [16] The security descriptor is essential to prevent unauthorized access to files. It stores information about:

- Owner of the file
- Permissions the owner has granted to other users
- What actions should be logged (auditing)

Following section describes NTFS and ReFS security identifiers.

3.7.1 Security Identifier in ReFS

The security identifier string found in ReFS is as follows:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001130530	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001130540	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001130550	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001130560	B8	00	00	00	10	00	5A	00	00	00	70	00	48	00	00	00	
001130570	30	00	02	00	53	00	2D	00	31	00	2D	00	35	00	2D	00	
001130580	32	00	31	00	2D	00	37	00	34	00	38	00	38	00	32	00	
001130590	36	00	36	00	37	00	34	00	2D	00	32	00	34	00	39	00	
0011305A0	33	00	35	00	35	00	35	00	35	00	37	00	35	00	2D	00	
0011305B0	38	00	37	00	35	00	39	00	31	00	38	00	33	00	34	00	
0011305C0	37	00	2D	00	35	00	30	00	30	00	00	00	00	00	00	00	
0011305D0	02	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011305E0	02	DA	48	6C	C8	32	D1	01	DA	05	7A	A2	B3	9B	D1	01	
0011305F0	DA	05	7A	A2	B3	9B	D1	01	DA	05	7A	A2	B3	9B	D1	01	
001130600	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Removing zeroes from the security identifier, we get:

S-1-5-21-748826674-2493555575-875918347-500

The above string is interpreted as follows:

S for security id

1 Revision level

5 Identifier Authority (48 bit) 5 = logon id

21 Sub-authority (21 = NT non unique)

748826674-2493555575-875918347 Domain or local computer identifier

500 user id/Relative ID (RID)

3.7.2 Security Identifier in NTFS

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0C000E590	00	00	00	00	00	00	00	00	06	00	00	10	00	00	00	00	
0C000E5A0	2B	01	53	00	2D	00	31	00	2D	00	35	00	2D	00	32	00	+ S - 1 - 5 - 2
0C000E5B0	31	00	2D	00	37	00	34	00	38	00	38	00	32	00	36	00	1 - 7 4 8 8 2 6
0C000E5C0	36	00	37	00	34	00	2D	00	32	00	34	00	39	00	33	00	6 7 4 - 2 4 9 3
0C000E5D0	35	00	35	00	35	00	35	00	37	00	35	00	2D	00	38	00	5 5 5 5 7 5 - 8
0C000E5E0	37	00	35	00	39	00	31	00	38	00	33	00	34	00	37	00	7 5 9 1 8 3 4 7
0C000E5F0	2D	00	35	00	30	00	30	00	D5	30	01	00	00	00	0C	00	- 5 0 0 00
0C000E600	68	00	52	00	00	00	00	00	39	00	00	00	00	00	02	00	h R 9
0C000E610	D3	F9	76	4A	01	23	D1	01	EE	B2	F8	47	00	8D	D1	01	ÒùvJ #Ñ Ì*øG Ñ
0C000E620	EE	B2	F8	47	00	8D	D1	01	EE	B2	F8	47	00	8D	D1	01	Ì*øG Ñ Ì*øG Ñ
0C000E630	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

The security identifier string in NTFS is S-1-5-21-748826674-2493555575-875918347-500, which is same as that found in ReFS.

Test case scenarios for artifacts gathering

4.1 Introduction

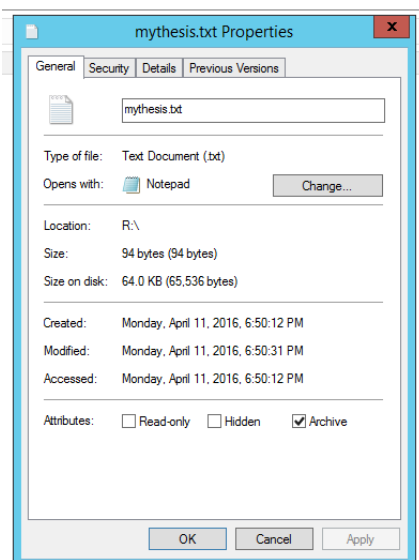
In this chapter, various scenarios such as creation, modification, deletion, etc are taken in consideration and performed on different files to know the working and structure of files in resilient file system. Main purpose of the research was to know the working and structure of this new file system which will provide assistance in criminal investigations.

4.2 .txt Scenarios

.txt is file extension for a text file. It contains unformatted text. It can be created through notepad which is included in Microsoft Windows by default. Different scenarios are applied to .txt file to check how resilient file system works with general files.

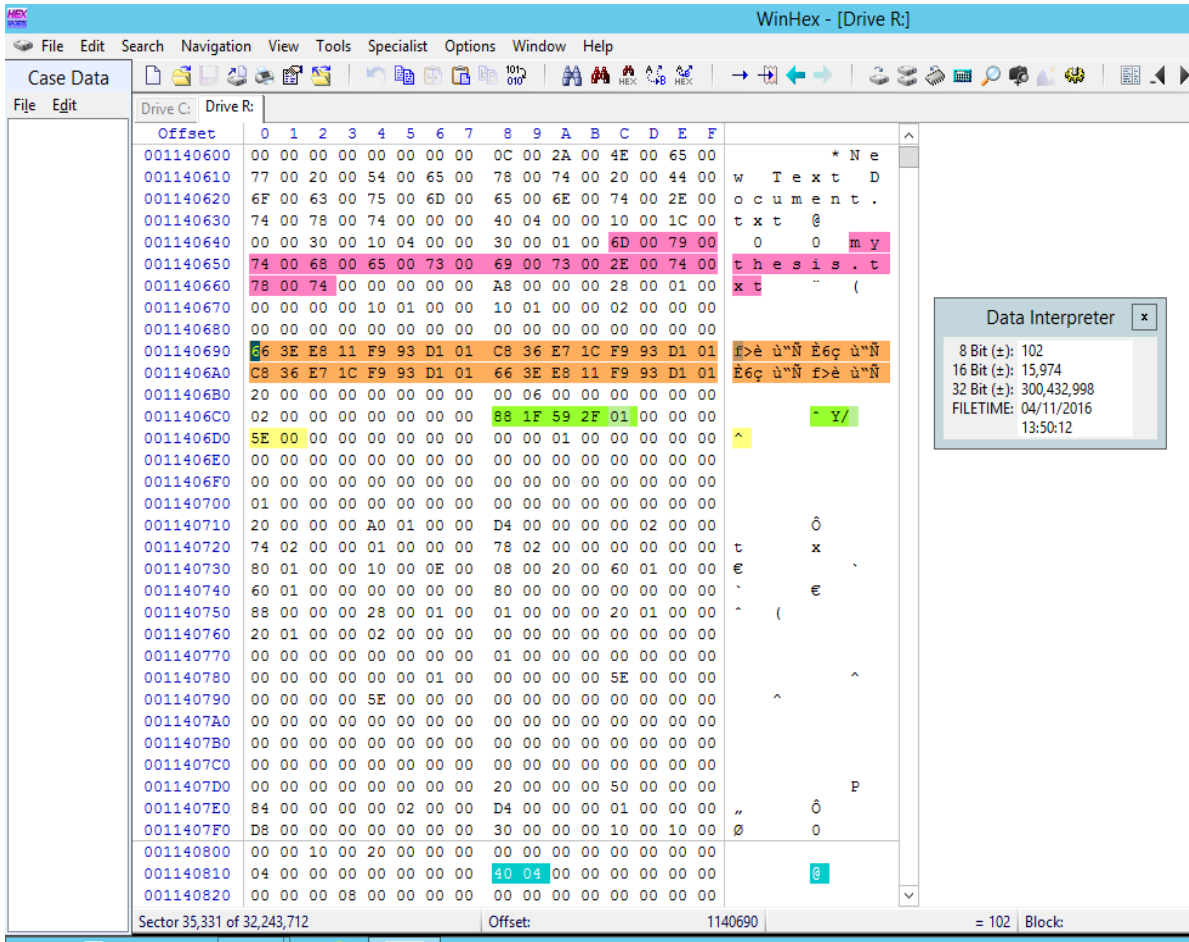
4.2.1 File creation

A new txt file is created in ReFS drive with content “*I am currently carrying out my research in Resilient File system which was introduced in 2012.*” and saved as “*mythesis.txt*”. Its size is 94 bytes (as observed through properties of file).



Official Microsoft documentation states that ReFS stores metadata in 16KB blocks so that it can support volume sizes of 2^{64} bytes. [17] ReFS storage engine uses B+ trees exclusively as the single common on-disk structure to represent all information on the disk. [18]

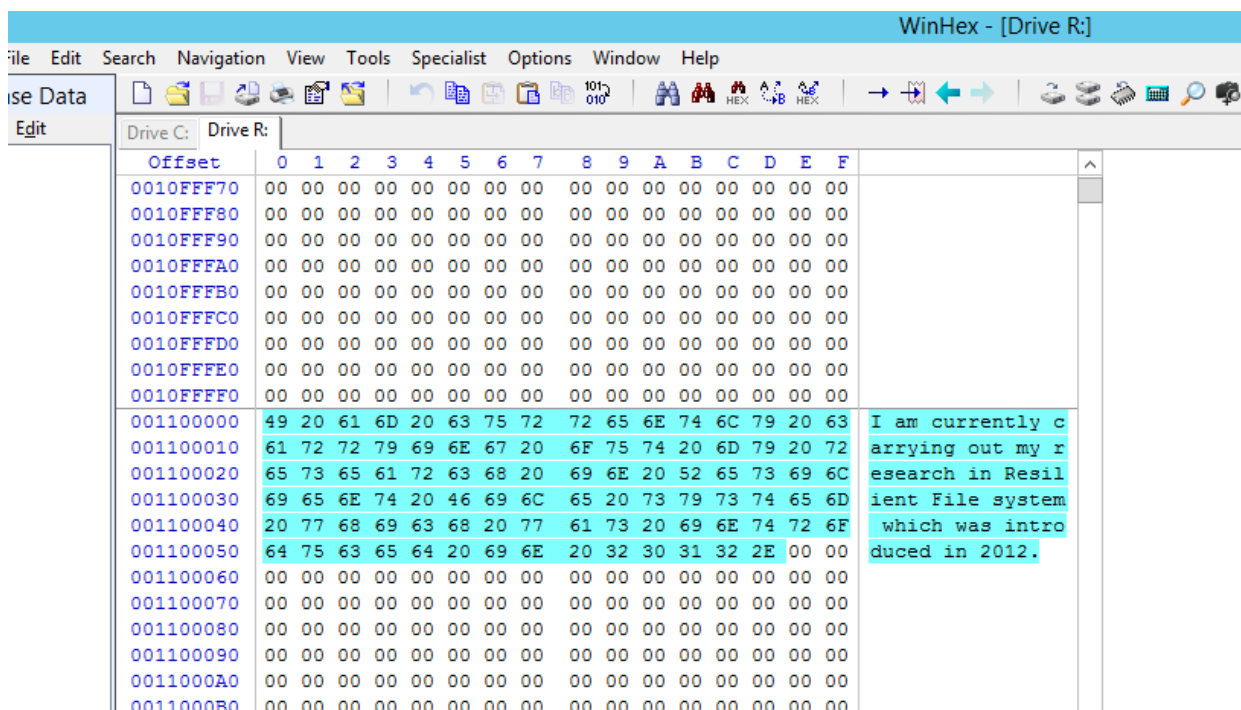
Observing the file in WinHex (hexadecimal editor), filename 'mythesis.txt' is found at offset 0x114064C (pink). A copy of filename is found at offset 0x1140AAC. MACE times (orange) are found at offset 0x1140690 which is in Windows 64-bit Little Endian. MACE times ('Modify', 'Access', 'Create' and 'Entry Modified') are time stamps of files, therefore they are present immediately after the filename. File permissions are present in green. File size is present in yellow (5E in hexadecimal and 94 bytes in decimal) and blue highlights the offset where the content of file is stored. It is file pointer in Little Endian. From this file pointer we can calculate the offset of the file content area.



In the metadata block of the file above, file pointer is 40 04 in Little Endian.

- Converting 40 04 to Big Endian, we get 04 40
- 440 hexadecimal is equal to 1088 in decimal
- Multiplying 1088 with 16,384 (because all the metadata is stored in blocks of 16KB), we get 17825792
- Converting 17825792 decimal to hexadecimal, we get 1100000
- 1100000 is the offset of the file content.

Now looking at the file content area at offset 0x1100000(in blue) in the editor:

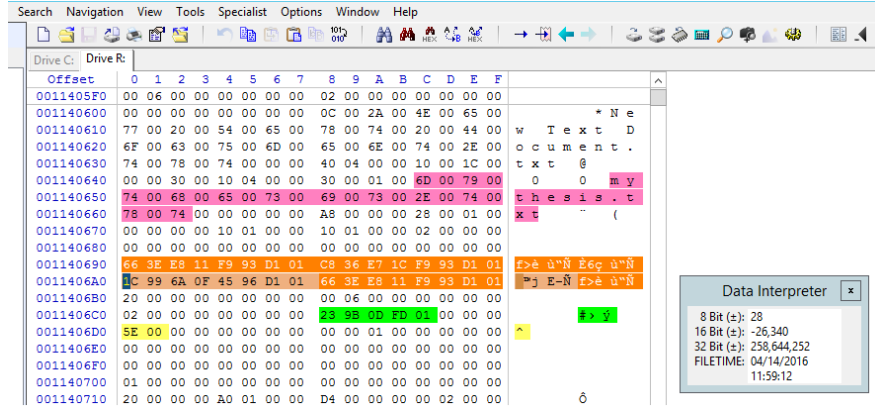


4.2.2 Permissions changed

We observe that when file permissions are changed, the values highlighted green in the drive change. This shows that these five bytes define file permissions as they change when file permissions are changed.

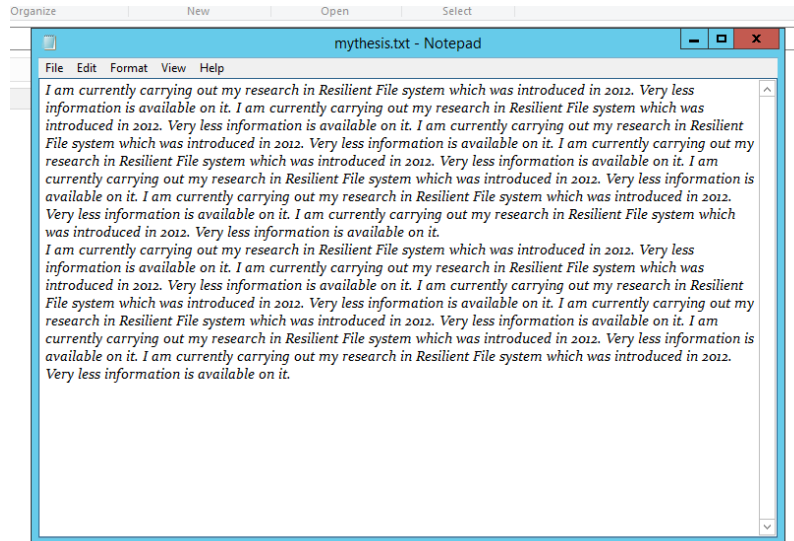
Further examining the drive closely, it is observed that the ‘accessed’ part of the MACE times (eight bytes) found at offset 0x11406A0 has changed (orange). ‘Accessed’ part of the MACE time refers to the time when the entry was accessed. As we know that after

changing permission this entry has been accessed therefore these eight bytes of the MACE times have been modified only. Modified, entry modified and created times are same. Rest of the metadata block is same as when the file was created.



4.2.3 Modifying content

More content is added to the previous 'mythesis.txt' file. File size, as shown in the properties window is 1.74 KB (1,783 bytes), now. We know that in NTFS, files containing data more than 512 bytes are non-resident and subsequently stored at different location. Let's examine what happens in ReFS:



After adding content to the previous file, it is observed that the file metadata is found at three different offsets. It can be seen here:

0011405F0	00 06 00 00 00 00 00 00 02 00 00 00 00 00 00	
001140600	00 00 00 00 00 00 00 00 0C 00 2A 00 4E 00 65 00	* Ne
001140610	77 00 20 00 54 00 65 00 78 00 74 00 20 00 44 00	w Text D
001140620	6F 00 63 00 75 00 6D 00 65 00 6E 00 74 00 2E 00	ocument .
001140630	74 00 78 00 74 00 00 00 40 04 00 00 10 00 1C 00	txt @
001140640	00 00 30 00 10 04 00 00 30 00 01 00 6D 00 79 00	o o my
001140650	74 00 68 00 65 00 73 00 69 00 73 00 2E 00 74 00	thesis . t
001140660	78 00 74 00 00 00 00 00 A8 00 00 00 28 00 01 00	xt " (
001140670	00 00 00 00 10 01 00 00 10 01 00 00 02 00 00 00	
001140680	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
001140690	66 3E E8 11 F9 93 D1 01 C8 36 E7 1C F9 93 D1 01	f>è ù"Ñ È6ç ù"Ñ
0011406A0	18 A3 74 85 56 96 D1 01 66 3E E8 11 F9 93 D1 01	£t...V-Ñ f>è ù"Ñ
0011406B0	20 00 00 00 00 00 00 00 00 06 00 00 00 00 00 00	
0011406C0	02 00 00 00 00 00 00 00 1D 0E 34 12 01 00 00 00	4
0011406D0	5E 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00	^
0011406E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

001144610	77 00 20 00 54 00 65 00 78 00 74 00 20 00 44 00	w Text D
001144620	6F 00 63 00 75 00 6D 00 65 00 6E 00 74 00 2E 00	ocument .
001144630	74 00 78 00 74 00 00 00 40 04 00 00 10 00 1C 00	txt @
001144640	00 00 30 00 10 04 00 00 30 00 01 00 6D 00 79 00	o o my
001144650	74 00 68 00 65 00 73 00 69 00 73 00 2E 00 74 00	thesis . t
001144660	78 00 74 00 00 00 00 00 A8 00 00 00 28 00 01 00	xt " (
001144670	00 00 00 00 10 01 00 00 10 01 00 00 02 00 00 00	
001144680	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
001144690	66 3E E8 11 F9 93 D1 01 C8 36 E7 1C F9 93 D1 01	f>è ù"Ñ È6ç ù"Ñ
0011446A0	EA 1B AA 31 45 96 D1 01 66 3E E8 11 F9 93 D1 01	è *1E-Ñ f>è ù"Ñ
0011446B0	20 00 00 00 00 00 00 00 00 06 00 00 00 00 00 00	
0011446C0	02 00 00 00 00 00 00 00 69 DB CD A2 01 00 00 00	iÜÍç
0011446D0	5E 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00	^
0011446E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

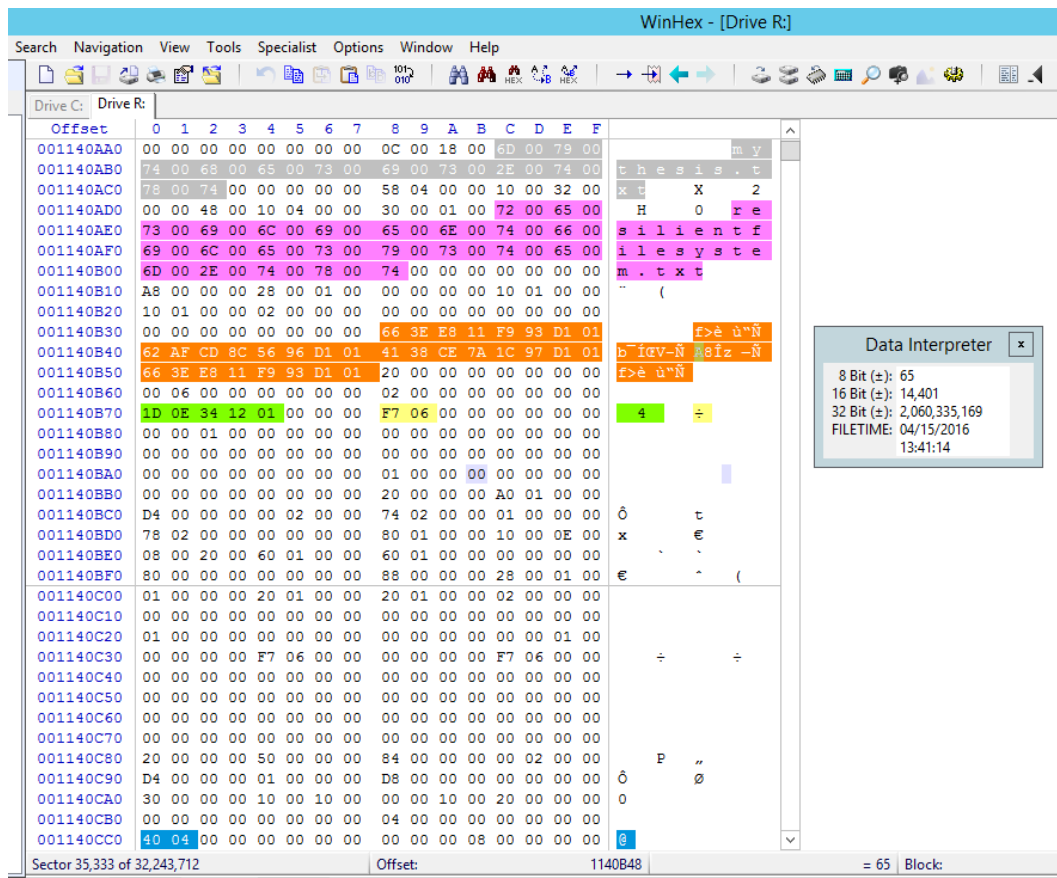
001148600	00 00 00 00 00 00 00 00 0C 00 2A 00 4E 00 65 00	* Ne
001148610	77 00 20 00 54 00 65 00 78 00 74 00 20 00 44 00	w Text D
001148620	6F 00 63 00 75 00 6D 00 65 00 6E 00 74 00 2E 00	ocument .
001148630	74 00 78 00 74 00 00 00 40 04 00 00 10 00 1C 00	txt @
001148640	00 00 30 00 10 04 00 00 30 00 01 00 6D 00 79 00	o o my
001148650	74 00 68 00 65 00 73 00 69 00 73 00 2E 00 74 00	thesis . t
001148660	78 00 74 00 00 00 00 00 A8 00 00 00 28 00 01 00	xt " (
001148670	00 00 00 00 10 01 00 00 10 01 00 00 02 00 00 00	
001148680	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
001148690	66 3E E8 11 F9 93 D1 01 62 AF CD 8C 56 96 D1 01	f>è ù"Ñ b̄ÍGV-Ñ
0011486A0	62 AF CD 8C 56 96 D1 01 66 3E E8 11 F9 93 D1 01	b̄ÍGV-Ñ f>è ù"Ñ
0011486B0	20 00 00 00 00 00 00 00 00 06 00 00 00 00 00 00	
0011486C0	02 00 00 00 00 00 00 00 1D 0E 34 12 01 00 00 00	4
0011486D0	F7 06 00 00 00 00 00 00 00 00 01 00 00 00 00 00	÷
0011486E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

At offset 0x114064C is the original file metadata. A duplicate of just the file name is found after 460 bytes at 0x1140AAC.

This point to one of the key features of ReFS that Microsoft has stated “Allocate on write transactional model for robust disk updates (also known as copy on write)”. [4] ReFS employ’s an allocation-on-write update strategy for metadata, which allocates new chunks for every update transaction. We have observed this fact above. Through this, built-in resiliency is obtained.

4.2.4 Renaming txt file

The same file ‘mythesis.txt’ is now renamed to be ‘resilientfilesystem.txt’. Checking out the metadata changes as follows:



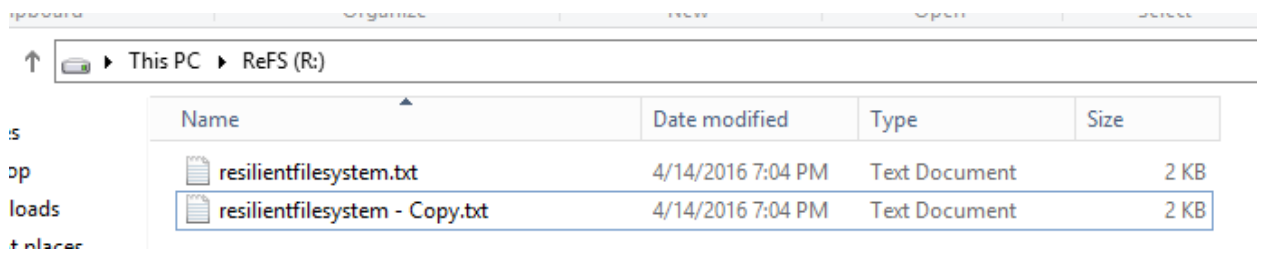
The file metadata is now found at offset 0x1140AA0 with the original file name (gray) and the renamed filename (pink). Because of the addition of the modified filename, all the file metadata can be seen to be pushed down. Entry modified time in MACE times has been

modified whereas all other times are same. Below this block, a duplicate filename entry can be found as shown below:

001140F40	02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
001140F50	0C 00 2E 00 72 00 65 00 73 00 69 00 6C 00 69 00	. r e s i l i
001140F60	65 00 6E 00 74 00 66 00 69 00 6C 00 65 00 73 00	e n t f i l e s
001140F70	79 00 73 00 74 00 65 00 6D 00 2E 00 74 00 78 00	y s t e m . t x
001140F80	74 00 00 00 00 00 00 00 00 00 00 00 00 00 00	t

4.2.5 Copying txt file

In this scenario, a copy of ‘resilientfilesystem.txt’ is made and pasted in the drive. Its name is ‘resilientfilesystem – Copy.txt’.



The copied file ‘resilientfilesystem – Copy.txt’ is found at four different offsets. These offsets are:

- First offset: 0x1144FBC
- Second offset: 0x1148FBC
- Third offset: 0x114CFBC
- Fourth offset: 0x1170FBC

Checking the metadata changes in each one of them:

First instance of the copied file is found at offset 0x1144FBC.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
001144F30	20	00	00	80	00	00	00	00	00	06	00	00	00	00	00	00
001144F40	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001144F50	0C	00	2E	00	72	00	65	00	73	00	69	00	6C	00	69	00
001144F60	65	00	6E	00	74	00	66	00	69	00	6C	00	65	00	73	00
001144F70	79	00	73	00	74	00	65	00	6D	00	2E	00	74	00	78	00
001144F80	74	00	00	00	00	00	00	00	70	00	00	00	10	00	18	00
001144F90	00	00	28	00	48	00	00	00	20	00	00	80	00	00	00	00
001144FA0	00	06	00	00	00	00	00	00	03	00	00	00	00	00	00	00
001144FB0	00	00	00	00	00	00	00	00	0C	00	3C	00	72	00	65	00
001144FC0	73	00	69	00	6C	00	69	00	65	00	6E	00	74	00	66	00
001144FD0	69	00	6C	00	65	00	73	00	79	00	73	00	74	00	65	00
001144FE0	6D	00	20	00	2D	00	20	00	43	00	6F	00	70	00	79	00
001144FF0	2E	00	74	00	78	00	74	00	60	04	00	00	10	00	40	00
001145000	08	00	50	00	10	04	00	00	30	00	01	00	72	00	65	00
001145010	73	00	69	00	6C	00	69	00	65	00	6E	00	74	00	66	00
001145020	69	00	6C	00	65	00	73	00	79	00	73	00	74	00	65	00
001145030	6D	00	20	00	2D	00	20	00	43	00	6F	00	70	00	79	00
001145040	2E	00	74	00	78	00	74	00	A8	00	00	00	28	00	01	00
001145050	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00
001145060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001145070	78	02	E3	07	D9	97	D1	01	78	02	E3	07	D9	97	D1	01
001145080	78	02	E3	07	D9	97	D1	01	78	02	E3	07	D9	97	D1	01
001145090	20	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00
0011450A0	03	00	00	00	00	00	00	00	88	1F	59	2F	01	00	00	00
0011450B0	F7	06	00	00	00	00	00	00	00	00	01	00	00	00	00	00
0011450C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Data Interpreter [x]

8 Bit (±): 120
 16 Bit (±): 632
 32 Bit (±): 132,317,816
 FILETIME: 04/16/2016 12:10:56

It is observed from the metadata blocks that ‘resilientfilesystem – Copy.txt’ exists with an entry of the original filename (grey) from which the file was copied ‘resilientfilesystem.txt’. Just above the copied filename, a duplicate of copied filename entry exists, 21 bytes after the first instance of ‘resilientfilesystem – Copy.txt’ is found.

File metadata at first offset 0x1144FBC is different only in terms of MACE times whereas the metadata at the other three offsets (0x1148FBC, 0x114CFBC and 0x1170FBC) is identical. Metadata at first offset is of the file that has been copied whereas other three offsets have the metadata of the original file when it was created.

Comparing metadata of first offset with that of the three offsets, it is observed that file metadata at first offset differs in MACE time values for ‘file altered’ (0x08) and ‘entry modified’ (0x10) time. It contains the time when the copied file was created.

File size, permissions and file pointer for the file metadata at all four offsets is same. File pointer is ‘58 04’ as shown below (blue):

0011451C0	84 00 00 00 00 02 00 00	D4 00 00 00 01 00 00 00	„	ô
0011451D0	D8 00 00 00 00 00 00 00	30 00 00 00 10 00 10 00	ø	o
0011451E0	00 00 10 00 20 00 00 00	00 00 00 00 00 00 00 00		
0011451F0	04 00 00 00 00 00 00 00	58 04 00 00 00 00 00 00		x
001145200	00 00 00 08 00 00 00 00	00 00 00 00 00 00 00 00		
001145210	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		

Calculating the file content offset from file pointer 58 04 which is in Little Endian.

- Converting 58 04 to Big Endian, we get 04 58
- 458 hexadecimal is equal to 1112 in decimal
- Multiplying 1112 with 16,384, we get 18219008
- Converting 18219008 decimal to hexadecimal, we get 1160000
- 1160000 is the offset where file content for the copied file is found.

File content for original file is at the same offset where it was prior to the copying of file i.e. 0x1100000.

4.2.6 Deleting txt file

In this scenario, file ‘resilientfilesystem – Copy.txt’ is simply deleted from ReFS drive. It still exists in recycle bin. Now when the file is searched, it is found at seven different offsets, four of them being exactly the same offsets as discussed in copied file section (0x1144FBC, 0x1148FBC, 0x114CFBC and 0x1170FBC), whereas remaining three offsets which have been added after the file has been deleted are:

- 0x114001E
- 0x117500C
- 0X117900C

The first most interesting offset where ‘resilientfilesystem – Copy.txt’ is found is 0x114001E, as shown below:

Above 0x117500C where 'resilientfilesystem - Copy.txt' is found, at 0x1174F54 original filename of the file exists (before the file was copied) 'resilientfilesystem.txt' (grey).

Twenty one bytes above 0x117500C, part of file name has been cut out 'entfilesystem - Copy.txt'. MACE times (yellow), file permissions (orange) and file size (green) are found after the deleted full filename entry.

At offset 0x117900C (4000 bytes after 0x117500C), an exact duplicate of 0x117500C metadata block, is found.

All the three metadata block offsets, possess the same file pointer '58 04' which was above calculated to be offset '0x1160000'. Looking at '0x1160000' offset, it is observed that even after file deletion, content still exists at the same location where it was originally when the file had been copied.

4.2.7 SHIFT + Delete txt file (permanent delete)

In this scenario, a new file is created 'forensic_investigations.txt' and content is placed in it. It is found at three offsets. First offset has no file pointer and zero size because when the file was created at the beginning, no content was placed in it. Therefore it has no pointer to content as well as zero file size, as shown here:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0011A1450	0C	00	2A	00	4E	00	65	00	77	00	20	00	54	00	65	00	* New Te
0011A1460	78	00	74	00	20	00	44	00	6F	00	63	00	75	00	6D	00	x t Docum
0011A1470	65	00	6E	00	74	00	2E	00	74	00	78	00	74	00	42	42	ent . t x t B B
0011A1480	60	00	00	00	10	00	18	00	04	00	28	00	38	00	00	00	` (8
0011A1490	20	00	00	80	00	00	00	00	00	06	00	00	00	00	00	00	€
0011A14A0	06	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011A14B0	0C	00	2A	00	4E	00	65	00	77	00	20	00	54	00	65	00	* New Te
0011A14C0	78	00	74	00	20	00	44	00	6F	00	63	00	75	00	6D	00	x t Docum
0011A14D0	65	00	6E	00	74	00	2E	00	74	00	78	00	74	00	FF	FF	ent . t x t ŷŷ
0011A14E0	60	04	00	00	10	00	3A	00	00	00	50	00	10	04	00	00	` : P
0011A14F0	30	00	01	00	66	00	6F	00	72	00	65	00	6E	00	73	00	0 forensic
0011A1500	69	00	63	00	5F	00	69	00	6E	00	76	00	65	00	73	00	i c _ i n v e s
0011A1510	74	00	69	00	67	00	61	00	74	00	69	00	6F	00	6E	00	t i g a t i o n
0011A1520	73	00	2E	00	74	00	78	00	74	00	00	00	00	00	00	00	s . t x t
0011A1530	A8	00	00	00	28	00	01	00	00	00	00	00	10	01	00	00	-- (
0011A1540	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
0011A1550	00	00	00	00	00	00	00	00	F5	35	51	AF	C6	9B	D1	01	ø5Q E N
0011A1560	F5	35	51	AF	C6	9B	D1	01	71	65	9C	C3	C6	9B	D1	01	ø5Q E N qeαE N
0011A1570	F5	35	51	AF	C6	9B	D1	01	20	00	00	00	00	00	00	00	ø5Q E N
0011A1580	00	06	00	00	00	00	00	00	06	00	00	00	00	00	00	00	
0011A1590	88	1F	59	2F	01	00	00	00	00	00	00	00	00	00	00	00	^ Y/
0011A15A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

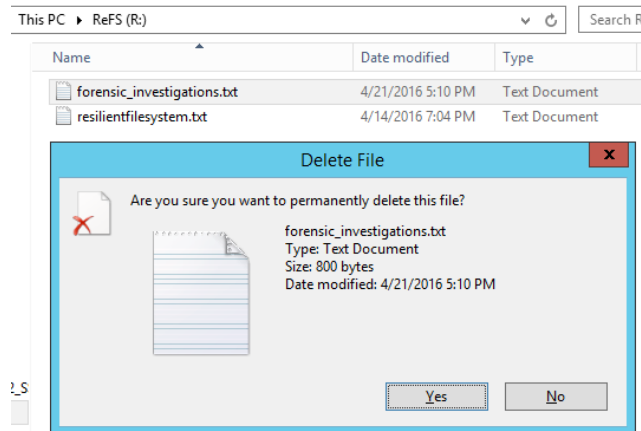
A duplicate of only the filename is found 480 bytes from where the first instance of 'forensic_investigations.txt' exists.

At the second offset both file size and file pointer are present because the content had been added in it after file creation but the MACE time values for file altered and entry modified timestamps are different then the first offset as shown:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0011A5450	0C	00	2A	00	4E	00	65	00	77	00	20	00	54	00	65	00	* New Te
0011A5460	78	00	74	00	20	00	44	00	6F	00	63	00	75	00	6D	00	xt Docum
0011A5470	65	00	6E	00	74	00	2E	00	74	00	78	00	74	00	42	42	ent.txt BB
0011A5480	60	00	00	00	10	00	18	00	04	00	28	00	38	00	00	00	` (8
0011A5490	20	00	00	80	00	00	00	00	00	06	00	00	00	00	00	00	€
0011A54A0	06	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011A54B0	0C	00	2A	00	4E	00	65	00	77	00	20	00	54	00	65	00	* New Te
0011A54C0	78	00	74	00	20	00	44	00	6F	00	63	00	75	00	6D	00	xt Docum
0011A54D0	65	00	6E	00	74	00	2E	00	74	00	78	00	74	00	FF	FF	ent.txt ŸŸ
0011A54E0	60	04	00	00	10	00	3A	00	00	00	50	00	10	04	00	00	` : P
0011A54F0	30	00	01	00	66	00	6F	00	72	00	65	00	6E	00	73	00	0 forensic
0011A5500	69	00	63	00	5F	00	69	00	6E	00	76	00	65	00	73	00	ic inves
0011A5510	74	00	69	00	67	00	61	00	74	00	69	00	6F	00	6E	00	tigation
0011A5520	73	00	2E	00	74	00	78	00	74	00	00	00	00	00	00	00	s.txt
0011A5530	A8	00	00	00	28	00	01	00	00	00	00	00	10	01	00	00	-- (
0011A5540	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
0011A5550	00	00	00	00	00	00	00	00	F5	35	51	AF	C6	9B	D1	01	05Q E>N
0011A5560	0D	71	07	CF	C6	9B	D1	01	0D	71	07	CF	C6	9B	D1	01	q iE>N q iE>N
0011A5570	F5	35	51	AF	C6	9B	D1	01	20	00	00	00	00	00	00	00	05Q E>N
0011A5580	00	06	00	00	00	00	00	00	06	00	00	00	00	00	00	00	
0011A5590	88	1F	59	2F	01	00	00	00	20	03	00	00	00	00	00	00	* Y/
0011A55A0	00	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	

File metadata at third offset (0x11A94F4) is identical to the metadata found at second offset.

The file 'forensic_investigations.txt' is then shift deleted (deleted permanently) from ReFS drive, as shown below:



After permanent deletion ‘forensic_investigations.txt’ is found at the same three offsets which have been discussed above. MACE time values are identical across the three offsets. The only difference in the metadata of the first offset is file size and file pointer, which were originally not present when the file was created, but are now present after deletion, as shown:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0011A1450	0C	00	2A	00	4E	00	65	00	77	00	20	00	54	00	65	00	* New Te
0011A1460	78	00	74	00	20	00	44	00	6F	00	63	00	75	00	6D	00	xt Docum
0011A1470	65	00	6E	00	74	00	2E	00	74	00	78	00	74	00	42	42	ent.txt BB
0011A1480	60	00	00	00	10	00	18	00	04	00	28	00	38	00	00	00	` (8
0011A1490	20	00	00	80	00	00	00	00	00	06	00	00	00	00	00	00	€
0011A14A0	06	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011A14B0	0C	00	2A	00	4E	00	65	00	77	00	20	00	54	00	65	00	* New Te
0011A14C0	78	00	74	00	20	00	44	00	6F	00	63	00	75	00	6D	00	xt Docum
0011A14D0	65	00	6E	00	74	00	2E	00	74	00	78	00	74	00	FF	FF	ent.txt yy
0011A14E0	60	04	00	00	10	00	3A	00	04	00	50	00	10	04	00	00	` : P
0011A14F0	30	00	01	00	66	00	6F	00	72	00	65	00	6E	00	73	00	0 forens
0011A1500	69	00	63	00	5F	00	69	00	6E	00	76	00	65	00	73	00	ic_inves
0011A1510	74	00	69	00	67	00	61	00	74	00	69	00	6F	00	6E	00	tigation
0011A1520	73	00	2E	00	74	00	78	00	74	00	00	00	00	00	00	00	s.txt
0011A1530	A8	00	00	00	28	00	01	00	00	00	00	00	10	01	00	00	-- (
0011A1540	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
0011A1550	00	00	00	00	00	00	00	00	F5	35	51	AF	C6	9B	D1	01	85Q E>N
0011A1560	0D	71	07	CF	C6	9B	D1	01	0D	71	07	CF	C6	9B	D1	01	q IE>N q IE>N
0011A1570	F5	35	51	AF	C6	9B	D1	01	20	00	00	00	00	00	00	00	85Q E>N
0011A1580	00	06	00	00	00	00	00	00	06	00	00	00	00	00	00	00	
0011A1590	88	1F	59	2F	01	00	00	00	20	03	00	00	00	00	00	00	^ Y/
0011A15A0	00	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011A1600	00	00	00	00	00	00	00	00	04	00	00	00	00	00	00	00	
0011A16D0	00	00	00	00	00	00	00	00	04	00	00	00	00	00	00	00	
0011A16E0	6C	04	00	00	00	00	00	00	00	00	00	08	00	00	00	00	l
0011A16F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Another feature that comes into play after permanent deletion is that file header, MACE times and file permissions of ‘forensic_investigations.txt’ file is found at offset 0x11AD454, but it is important to note here that they are without filename entry. File size and file pointer are missing in this entry. It can be deduced here that this might be the entry which existed at first offset when the file was originally created (before deletion) because that entry also didn’t have file size and file pointer.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0011AD430	20	00	00	80	00	00	00	00	00	06	00	00	00	00	00	00
0011AD440	05	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0011AD450	0C	00	2A	00	4E	00	65	00	77	00	20	00	54	00	65	00
0011AD460	78	00	74	00	20	00	44	00	6F	00	63	00	75	00	6D	00
0011AD470	65	00	6E	00	74	00	2E	00	74	00	78	00	74	00	42	42
0011AD480	60	00	00	00	10	00	18	00	00	00	28	00	38	00	00	00
0011AD490	20	00	00	80	00	00	00	00	00	06	00	00	00	00	00	00
0011AD4A0	06	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0011AD4B0	0C	00	2A	00	4E	00	65	00	77	00	20	00	54	00	65	00
0011AD4C0	78	00	74	00	20	00	44	00	6F	00	63	00	75	00	6D	00
0011AD4D0	65	00	6E	00	74	00	2E	00	74	00	78	00	74	00	FF	FF
0011AD4E0	50	04	00	00	10	00	2E	00	08	00	40	00	10	04	00	00
0011AD4F0	30	00	01	00	4E	00	65	00	77	00	20	00	54	00	65	00
0011AD500	78	00	74	00	20	00	44	00	6F	00	63	00	75	00	6D	00
0011AD510	65	00	6E	00	74	00	2E	00	74	00	78	00	74	00	00	00
0011AD520	A8	00	00	00	28	00	01	00	00	00	00	00	10	01	00	00
0011AD530	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00
0011AD540	00	00	00	00	00	00	00	00	F5	35	51	AF	C6	9B	D1	01
0011AD550	F5	35	51	AF	C6	9B	D1	01	F5	35	51	AF	C6	9B	D1	01
0011AD560	F5	35	51	AF	C6	9B	D1	01	20	00	00	00	00	00	00	00
0011AD570	00	06	00	00	00	00	00	00	06	00	00	00	00	00	00	00
0011AD580	88	1F	59	2F	01	00	00	00	00	00	00	00	00	00	00	00
0011AD590	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0011AD5A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0011AD5B0	00	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00

As shown above, at 0x11AD454 and 0x11AD4B4, identical file headers which were present before file deletion exist here with same MACE times (orange) and file permissions (green) that the file originally had before deletion. But file size (red) and file pointer are not present.

Despite ‘forensic_investigations.txt’ file’s permanent deletion from the ReFS drive it is observed that the file and file content still exist at their positions although at front end they have been deleted, but in the hexadecimal they are easily visible.

Forensic investigators while manually traversing the drive can easily file deleted as well as permanently deleted data as they are only deleted from front end but are present at the hexadecimal level.

4.2.8 Scenario analysis for .txt file

In this section of the chapter, all scenarios applied on the text file are analyzed and data is presented in tabular format:

Operation	Filename (pink)	Artifacts Gathered				
		Metadata Offset	MACE times (orange)	Permission (green)	File size (yellow)	File pointer (blue)
Creation	mythesis.txt	0x114064C	✓	✓	5E	40 04
Permission changed	mythesis.txt	0x114464C	Entry modified part changes	Changed values as compared to above	5E	40 04
Modifying content	mythesis.txt	0x114864C	✓	✓	F7 06	40 04
Renaming file	resilientfilesyste m.txt	0x1140ADC	Entry modified part changes	✓	F7 06	40 04
Copying file	resilientfilesyste m - Copy.txt	0x1144FBC	✓	✓	F7 06	58 04
Simple delete	resilientfilesyste m - Copy.txt	0x114001E	Contains file size, time of deletion, path of file and name of the deleted file			
	resilientfilesyste m.txt entfilesystem - Copy.txt resilientfilesyste m - Copy.txt	0x117900C	✓	✓	F7 06	58 04
Shift delete	resilientfilesyste m - Copy.txt	0x1144FBC 0x1148FBC 0x114CFBC	✓	✓	✓	✓

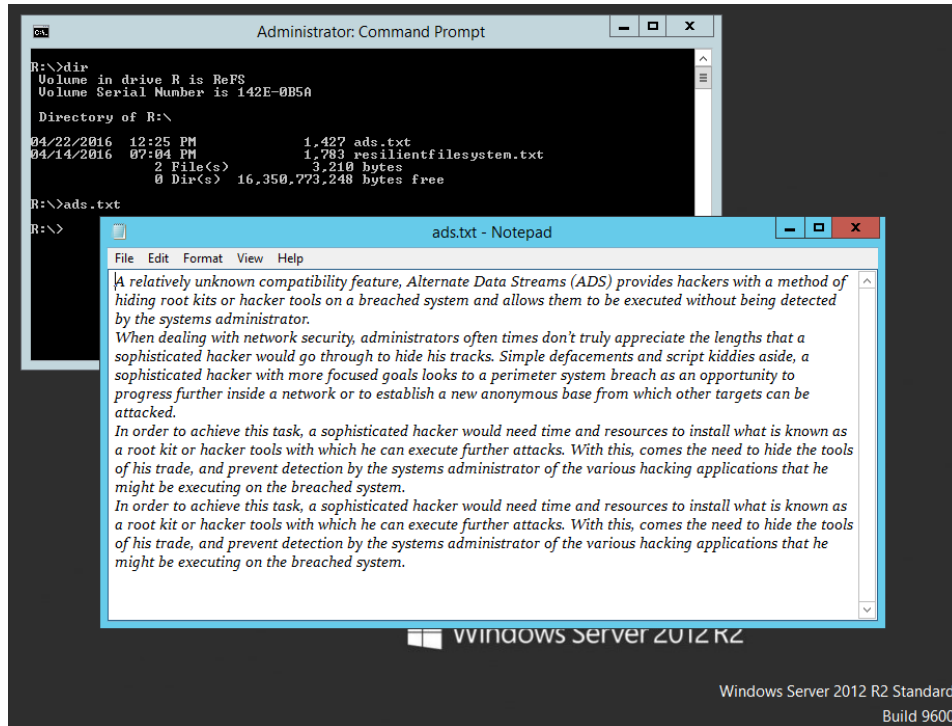
Table 4-1: Scenario analysis for .txt file

4.3 Alternate data stream

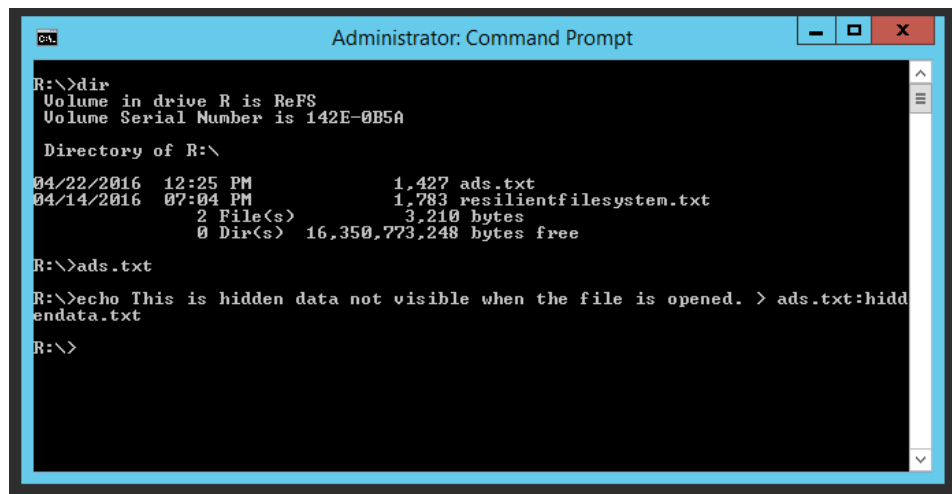
ADS is the ability to fork file data into existing files without affecting their functionality, size, or display to traditional file browsing utilities like Windows Explorer. [19] NTFS supports ADS in the form of text, file, image and executable files. Microsoft's official documentation states that resilient file system does not support alternate data stream. Let's check if it supports or not.

4.3.1 ADS in the form of text

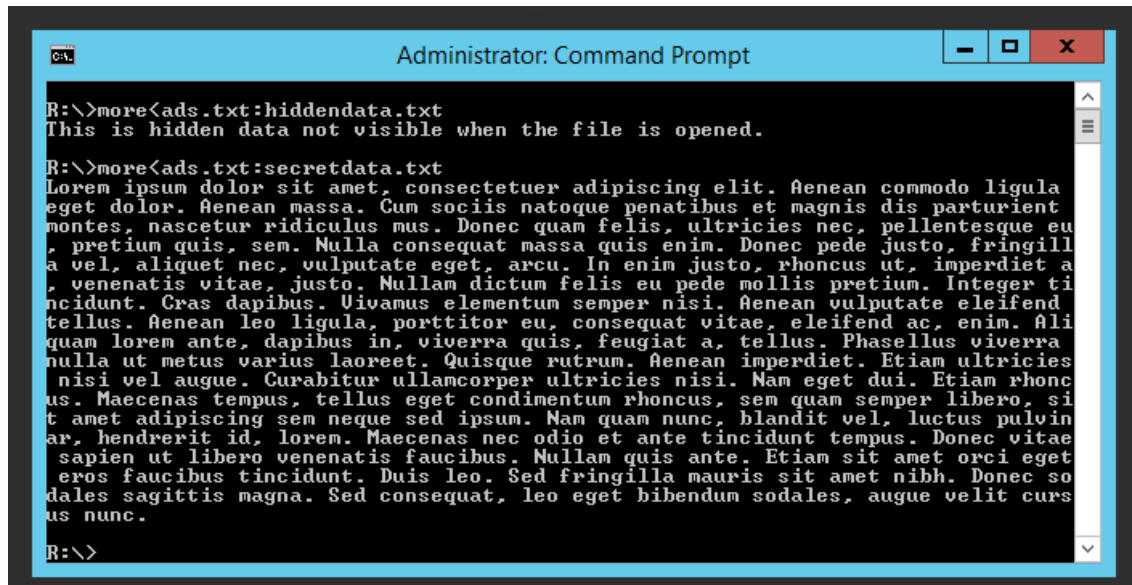
A file named 'ads.txt' is created in R:\ drive for this scenario. File size is 1427 bytes. It is opened from command prompt:



Alternate Data Stream is then created, and data is attached to it. The data is 'This is hidden data not visible when the file is opened.' The name of the hidden alternate stream is 'hiddendata.txt'.



Another alternate stream is attached to the file. Its name is 'secretdata.txt' and the data contents are long.



```
Administrator: Command Prompt
R:\>more<ads.txt:hidddata.txt
This is hidden data not visible when the file is opened.

R:\>more<ads.txt:secretdata.txt
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula
 eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient
 montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu
 , pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringill
 a vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a
 , venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer ti
 ncidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend
 tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Ali
 quam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra
 nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies
 nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhonc
 us. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, si
 t amet adipiscing sem neque sed ipsum. Nam quam nunc, blandit vel, luctus pulvin
 ar, hendrerit id, lorem. Maecenas nec odio et ante tincidunt tempus. Donec vitae
 sapien ut libero venenatis faucibus. Nullam quis ante. Etiam sit amet orci eget
 eros faucibus tincidunt. Duis leo. Sed fringilla mauris sit amet nibh. Donec so
 dales sagittis magna. Sed consequat, leo eget bibendum sodales, augue velit curs
 us nunc.

R:\>
```

Looking at 'ads.txt' at hexadecimal level in WinHex, it is found at five different offsets:

- 0x1161554
- 0x1165554
- 0x1169554
- 0x116D554
- 0x11AD554

First four offsets appear after every 4000 bytes successively whereas last offset appears after 40,000 bytes.

File metadata at the first four offsets contains 'ads.txt' MACE times, file size, file permissions and pointer to file content. Last offset does not contain file size and file pointer to 'ads.txt' with it. It can be deduced that the metadata at the last offset was in fact written the very first time when the file 'ads.txt' was initially created because only then the file was empty and hence possesses zero file size and zero file pointer. Immediately after the creation of ads.txt data was added to it and therefore that data is present in the first four offsets.

Moreover all the five offsets differ in ‘file altered’ and ‘entry modified’ timestamps of MACE time. All have same ‘file creation’ and ‘file read’ timestamps.

In the following three ‘ads.txt’ file metadata offsets: 0x1161554, 0x1169554 and 0x116D554, the hidden file named ‘hiddendata.txt’, its size and its content is present along with ‘ads.txt’ metadata.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001161510	0C	00	2A	00	4E	00	65	00	77	00	20	00	54	00	65	00	* New Te
001161520	78	00	74	00	20	00	44	00	6F	00	63	00	75	00	6D	00	x t Docum
001161530	65	00	6E	00	74	00	2E	00	74	00	78	00	74	00	33	41	ent . t x t 3A
001161540	38	04	00	00	10	00	12	00	00	00	28	00	10	04	00	00	8 (
001161550	30	00	01	00	61	00	64	00	73	00	2E	00	74	00	78	00	o ads . t x
001161560	74	00	00	00	00	00	00	00	A8	00	00	00	28	00	01	00	t " (
001161570	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00	
001161580	01	00	00	00	00	00	00	00	03	00	00	00	00	00	00	00	
001161590	AB	E7	F0	11	68	9C	D1	01	3C	C5	D5	40	70	9C	D1	01	<çø hœÑ <ÃÕ@pœÑ
0011615A0	3C	C5	D5	40	70	9C	D1	01	AB	E7	F0	11	68	9C	D1	01	<ÃÕ@pœÑ <çø hœÑ
0011615B0	20	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00	
0011615C0	07	00	00	00	00	00	00	00	88	1F	59	2F	01	00	00	00	" Y/
0011615D0	93	05	00	00	00	00	00	00	00	00	01	00	00	00	00	00	"
0011615E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011615F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001161600	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001161610	20	00	00	00	48	00	00	00	2C	02	00	00	01	03	00	00	H ,
001161620	74	02	00	00	01	00	00	00	78	02	00	00	00	00	00	00	t x
001161630	28	00	00	00	10	00	00	00	02	00	10	00	18	00	00	00	(
001161640	5D	04	00	00	00	00	00	00	00	00	02	08	08	00	00	00	l
001161650	4A	8C	01	9C	10	E0	A7	94	01	00	00	00	20	01	00	00	JG œ àS"
001161660	20	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
001161670	00	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00	
001161680	00	00	00	00	00	00	01	00	00	00	00	00	93	05	00	00	" "
001161690	00	00	00	00	93	05	00	00	00	00	00	00	00	00	00	00	" "
0011616A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011616B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011616C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011616D0	00	00	00	00	00	00	00	00	20	00	00	00	50	00	00	00	P
0011616E0	84	00	00	00	00	02	00	00	D4	00	00	00	01	00	00	00	" Õ
0011616F0	D8	00	00	00	00	00	00	00	30	00	00	00	10	00	10	00	ø 0
001161700	00	00	10	00	20	00	00	00	00	00	00	00	00	00	00	00	
001161710	04	00	00	00	00	00	00	00	50	04	00	00	00	00	00	00	p
001161720	00	00	00	08	00	00	00	00	00	00	00	00	00	00	00	00	
001161730	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Looking at the file metadata above, of the first offset 0x1161554 (pink), file size is ‘93 05’ (yellow) which is 1427 bytes, file permissions are present at 0x11615C8 (green), MACE times are present at 0x1161590 (orange) and ‘50 04’ (blue) is the file pointer which points to offset ‘0x114000’ where ‘ads.txt’ file’s content is present as shown below:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001140000	41	20	72	65	6C	61	74	69	76	65	6C	79	20	75	6E	6B	A relatively unk
001140010	6E	6F	77	6E	20	63	6F	6D	70	61	74	69	62	69	6C	69	n timer compatibili
001140020	74	79	20	66	65	61	74	75	72	65	2C	20	41	6C	74	65	ty feature, Alte
001140030	72	6E	61	74	65	20	44	61	74	61	20	53	74	72	65	61	rnate Data Strea
001140040	6D	73	20	28	41	44	53	29	20	70	72	6F	76	69	64	65	ms (ADS) provide
001140050	73	20	68	61	63	6B	65	72	73	20	77	69	74	68	20	61	s hackers with a
001140060	20	6D	65	74	68	6F	64	20	6F	66	20	68	69	64	69	6E	method of hidin
001140070	67	20	72	6F	6F	74	20	6B	69	74	73	20	6F	72	20	68	g root kits or h
001140080	61	63	6B	65	72	20	74	6F	6F	6C	73	20	6F	6E	20	61	acker tools on a
001140090	20	62	72	65	61	63	68	65	64	20	73	79	73	74	65	6D	breached system
0011400A0	20	61	6E	64	20	61	6C	6C	6F	77	73	20	74	68	65	6D	and allows them
0011400B0	20	74	6F	20	62	65	20	65	78	65	63	75	74	65	64	20	to be executed
0011400C0	77	69	74	68	6F	75	74	20	62	65	69	6E	67	20	64	65	without being de
0011400D0	74	65	63	74	65	64	20	62	79	20	74	68	65	20	73	79	ected by the sv

Scrolling down from the offset 0x1161554 (pink), 'hiddendata.txt' its content and file size are found.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001161710	04	00	00	00	00	00	00	00	50	04	00	00	00	00	00	00	P
001161720	00	00	00	08	00	00	00	00	00	00	00	00	00	00	00	00	
001161730	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001161740	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001161750	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001161760	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001161770	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001161780	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001161790	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011617A0	00	00	00	00	00	00	00	00	00	00	00	00	20	00	00	00	
0011617B0	B0	00	00	00	10	00	28	00	00	00	38	00	77	00	00	00	° (8 w
0011617C0	77	00	00	00	00	00	00	00	B0	00	00	00	68	00	69	00	w ° hi
0011617D0	64	00	64	00	65	00	6E	00	64	00	61	00	74	00	61	00	ddendata
0011617E0	2E	00	74	00	78	00	74	00	00	00	00	00	6B	00	00	00	.txt k
0011617F0	0C	00	00	00	30	00	00	00	00	00	00	00	00	00	00	00	0
001161800	3B	00	00	00	00	00	00	00	3B	00	00	00	00	00	00	00	; ;
001161810	3B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	; ;
001161820	02	00	00	00	54	68	69	73	20	69	73	20	68	69	64	64	This is hidd
001161830	65	6E	20	64	61	74	61	20	6E	6F	74	20	76	69	73	69	en data not visi
001161840	62	6C	65	20	77	68	65	6E	20	74	68	65	20	66	69	6C	ble when the fil
001161850	65	20	69	73	20	6F	70	65	6E	65	64	2E	20	0D	0A	00	e is opened.
001161860	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Highlighted in red is '5D 04' which points to 'secretdata.txt' file metadata. It is file pointer to the second hidden stream. '5D 04' is 0x1174000 offset. Navigating to offset 0x1174000, it is observed that '5D 04' is also present there.

001173FE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
001173FF0	00 00 00 00 20 00 00 00 A0 01 00 00 50 02 00 00	P
001174000	5D 04 00 00 00 00 00 00 96 00 00 00 00 00 00 00]
001174010	00 00 00 00 00 00 00 00 00 06 00 00 00 00 00 00	
001174020	01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Scrolling a little down it is observed that ‘hiddendata.txt’, its content and file size is present along with ‘secretdata.txt’, its content and file size. ‘3B’ is the size of ‘hiddendata.txt’ which is 59 bytes. ‘5B 05’ is the file size of ‘secretdata.txt’ which is 1371 bytes. It is important to note here that the actual size of hiddendata.txt is 56 bytes and secretdata.txt is 1368 bytes respectively. It is because three identical bytes ‘20 0D 0A’ (yellow) are present at the end of both hidden streams content. It can be deduced here that these three bytes show that they belong to alternate data stream.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0011741E0	00	00	38	00	77	00	00	00	77	00	00	00	00	00	00	00
0011741F0	B0	00	00	00	68	00	69	00	64	00	64	00	65	00	6E	00
001174200	64	00	61	00	74	00	61	00	2E	00	74	00	78	00	74	00
001174210	00	00	00	00	6B	00	00	00	0C	00	00	00	30	00	00	00
001174220	00	00	00	00	00	00	00	00	3B	00	00	00	00	00	00	00
001174230	3B	00	00	00	00	00	00	00	3B	00	00	00	00	00	00	00
001174240	00	00	00	00	00	00	00	00	02	00	00	00	54	68	69	73
001174250	20	69	73	20	68	69	64	64	65	6E	20	64	61	74	61	20
001174260	6E	6F	74	20	76	69	73	69	62	6C	65	20	77	68	65	6E
001174270	20	74	68	65	20	66	69	6C	65	20	69	73	20	6F	70	65
001174280	6E	65	64	2E	20	0D	0A	00	D0	05	00	00	10	00	28	00
001174290	00	00	38	00	97	05	00	00	97	05	00	00	00	00	00	00
0011742A0	B0	00	00	00	73	00	65	00	63	00	72	00	65	00	74	00
0011742B0	64	00	61	00	74	00	61	00	2E	00	74	00	78	00	74	00
0011742C0	00	00	00	00	8B	05	00	00	0C	00	00	00	30	00	00	00
0011742D0	00	00	00	00	00	00	00	00	5B	05	00	00	00	00	00	00
0011742E0	5B	05	00	00	00	00	00	00	5B	05	00	00	00	00	00	00
0011742F0	00	00	00	00	00	00	00	00	02	00	00	00	4C	6F	72	65
001174300	6D	20	69	70	73	75	6D	20	64	6F	6C	6F	72	20	73	69
001174310	74	20	61	6D	65	74	2C	20	63	6F	6E	73	65	63	74	65
001174320	74	75	65	72	20	61	64	69	70	69	73	63	69	6E	67	20
001174330	65	6C	69	74	2E	20	41	65	6E	65	61	6E	20	63	6F	6D
001174340	6D	6E	64	6E	20	6C	69	67	75	6C	61	20	65	67	65	74

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0011747E0	75	72	69	73	20	73	69	74	20	61	6D	65	74	20	6E	69
0011747F0	62	68	2E	20	44	6F	6E	65	63	20	73	6F	64	61	6C	65
001174800	73	20	73	61	67	69	74	74	69	73	20	6D	61	67	6E	61
001174810	2E	20	53	65	64	20	63	6F	6E	73	65	71	75	61	74	2C
001174820	20	6C	65	6F	20	65	67	65	74	20	62	69	62	65	6E	64
001174830	75	6D	20	73	6F	64	61	6C	65	73	2C	20	61	75	67	75
001174840	65	20	76	65	6C	69	74	20	63	75	72	73	75	73	20	6E
001174850	75	6E	63	2E	20	0D	0A	00	00	00	00	00	00	00	00	00
001174860	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

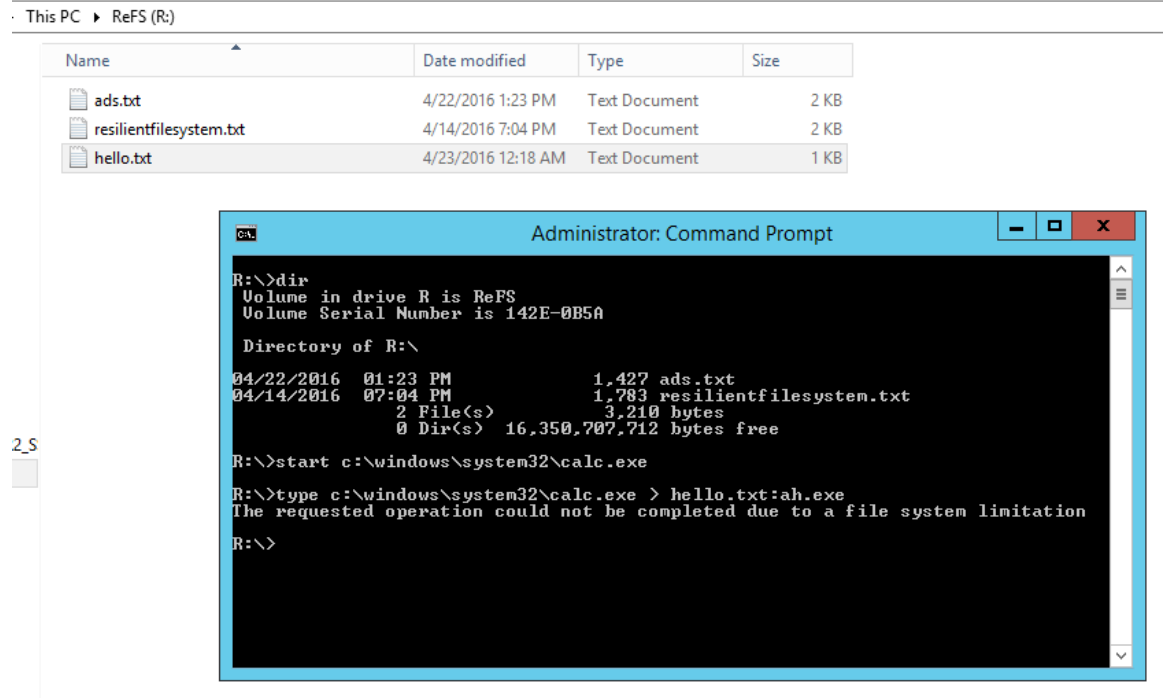
4.3.1.1 Scenario analysis for ADS in the form of text

File name (pink)	Alternate stream name	Artifacts Gathered					
		Metadata Offset	MACE times (orange)	Permissi on (green)	File size (yellow)	File pointer (blue)	Hidden data (purple)
ads.txt	hiddendata.txt secretdata.txt	0x1161554	✓	✓	93 05	50 04	Content and size of hiddendata.txt present with file pointer to secretdata.txt
		0x1165554	Entry modified and file altered time changes	✓	93 05	BC 04	
		0x1169554	Entry modified and file altered time changes	✓	93 05	BC 04	Content and size of hiddendata.txt present
		0x116D554	Entry modified and file altered time changes	✓	93 05	✓	Content and size of hiddendata.txt present
		0x11AD554	Entry modified and file altered time changes	✓	X	X	

Table 4-2: Scenario analysis for ADS in the form of text

4.3.2 ADS in the form of executable

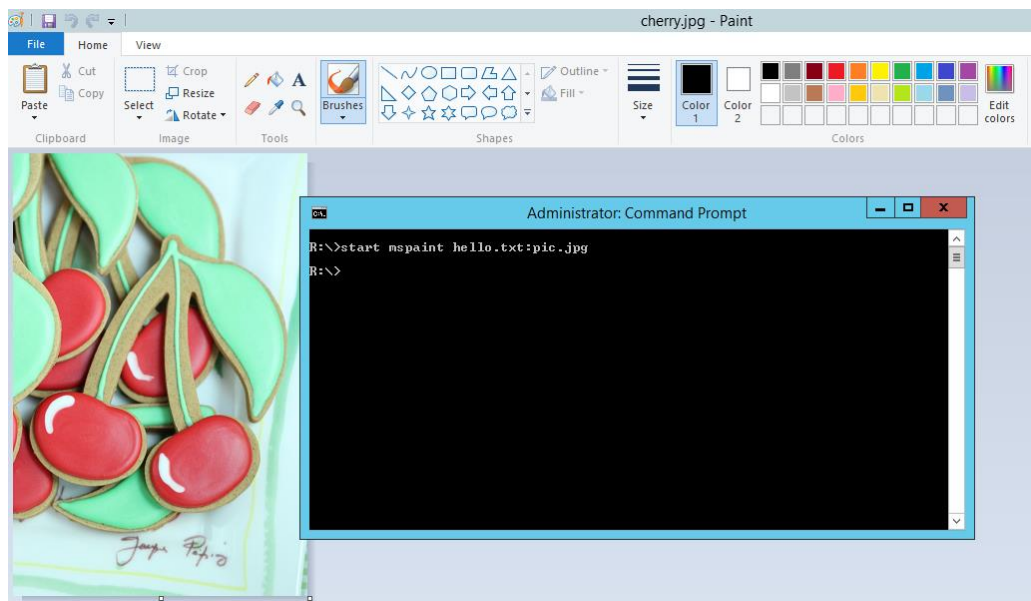
‘hello.txt’ file is created for this purpose. Calculator executable is tried to attach with it but the following error occurs when the command is executed: ‘*The requested operation could not be completed due to a file system limitation*’.



Thus attaching an exe file as alternate data stream in ReFS is not possible.

4.3.3 ADS in the form of image file

A jpg image named “cherry.jpg” is attached to “hello.txt” file as alternate data stream in ReFS drive. It is then opened and displayed in command prompt.



Thus attaching an image file as alternate data stream in ReFS is possible.

4.4 .JPG scenarios

In this scenario, jpg image file is taken and various operations are performed on it.

4.4.1 Copying jpg file

For this scenario, a jpg file named 'verse.jpg' is copied and pasted in ReFS drive. The image file metadata as seen in WinHex is as follows:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001262340	0A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262350	0C	00	12	00	76	00	65	00	72	00	73	00	65	00	2E	00	verse.
001262360	6A	00	70	00	67	00	00	00	38	04	00	00	10	00	16	00	jpg 8
001262370	08	00	28	00	10	04	00	00	30	00	01	00	76	00	65	00	(0 ve
001262380	72	00	73	00	65	00	2E	00	6A	00	70	00	67	00	00	00	rse.jpg
001262390	A8	00	00	00	28	00	01	00	00	00	00	00	10	01	00	00	" (
0012623A0	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
0012623B0	00	00	00	00	00	00	00	00	79	D7	B7	CC	59	9D	D1	01	y*.iY N
0012623C0	03	49	33	0C	88	4B	CF	01	03	49	33	0C	88	4B	CF	01	I3 ^KI I3 ^KI
0012623D0	79	D7	B7	CC	59	9D	D1	01	20	00	00	00	00	00	00	00	y*.iY N
0012623E0	00	06	00	00	00	00	00	00	0A	00	00	00	00	00	00	00	
0012623F0	88	1F	59	2F	01	00	00	00	E1	7B	01	00	00	00	00	00	^ Y/ á{
001262400	00	00	02	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262410	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262420	00	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00	
001262430	00	00	00	00	00	00	00	00	20	00	00	00	A0	01	00	00	
001262440	D4	00	00	00	00	02	00	00	74	02	00	00	01	00	00	00	ô t
001262450	78	02	00	00	00	00	00	00	80	01	00	00	10	00	0E	00	x €
001262460	08	00	20	00	60	01	00	00	60	01	00	00	00	00	00	00	€ ` `
001262470	80	00	00	00	00	00	00	00	88	00	00	00	28	00	01	00	€ ^ (
001262480	01	00	00	00	20	01	00	00	20	01	00	00	02	00	00	00	
001262490	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012624A0	01	00	00	00	00	00	00	00	00	00	00	00	00	00	02	00	
0012624B0	00	00	00	00	E1	7B	01	00	00	00	00	00	E1	7B	01	00	á{ á{
0012624C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012624D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012624E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012624F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262500	20	00	00	00	50	00	00	00	84	00	00	00	00	02	00	00	P "
001262510	D4	00	00	00	01	00	00	00	D8	00	00	00	00	00	00	00	ô ø
001262520	30	00	00	00	10	00	10	00	00	00	10	00	20	00	00	00	0
001262530	00	00	00	00	00	00	00	00	08	00	00	00	00	00	00	00	4
001262540	BC	04	00	00	00	00	00	00	00	00	00	08	00	00	00	00	
001262550	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262560	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Filename is found at 0x1262354 (pink) and its duplicate is found after 16 bytes. In text file a duplicate of just the file name was found farther down the metadata of the file (after the file pointer). MACE timestamps possess same timestamps when the file was created and

read, whereas entry modified and file altered timestamps are same. File created and file read timestamps possess the latest time when the file was copied whereas entry modified and file altered timestamps possess past values (year 2014 in this case).

Image file properties (green) are present at offset 0x12623F0. File size 'E1 7B 01' (yellow) which when converted to big Endian becomes '01 7B E1'. Converting it in decimal, we get the value 97,249 bytes which is exactly the value shown in file properties window.

Moving down we see file pointer (blue) 'BC 04' which is in Little Endian.

- Converting BC 04 to Big Endian, we get 04 BC
- 4BC hexadecimal is equal to 1212 in decimal
- Multiplying 1212 with 16,384 (because all the metadata is stored in blocks of 16KB), we get 19857408
- Converting 19857408 decimal to hexadecimal, we get 12F0000
- 12F0000 is the offset of the image file coding area.

Now looking at the image file coding area at offset 0x12F0000 (in blue) in the editor:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0012EFFF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012F0000	FF	D8	FF	E2	02	1C	49	43	43	5F	50	52	4F	46	49	4C	y0yã ICC_PROFIL
0012F0010	45	00	01	01	00	02	0C	6C	63	6D	73	02	10	00	00	00	E lcms
0012F0020	6D	6E	74	72	52	47	42	20	58	59	5A	20	07	DC	00	01	mntrRGB XYZ Ü
0012F0030	00	19	00	03	00	29	00	39	61	63	73	70	41	50	50	4C) 9acspAPPL
0012F0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012F0050	00	00	00	00	00	00	00	00	00	00	F6	D6	00	01	00	00	öç
0012F0060	00	00	D3	2D	6C	63	6D	73	00	00	00	00	00	00	00	00	Ö-lcms
0012F0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012F0080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012F0090	00	00	00	00	00	00	00	0A	64	65	73	63	00	00	00	FC	desc ü
0012F00A0	00	00	00	5E	63	70	72	74	00	00	01	5C	00	00	00	0B	^cprt \
0012F00B0	77	74	70	74	00	00	01	68	00	00	00	14	62	6B	70	74	wtpt h bkpt
0012F00C0	00	00	01	7C	00	00	00	14	72	58	59	5A	00	00	01	90	rXYZ
0012F00D0	00	00	00	14	67	58	59	5A	00	00	01	A4	00	00	00	14	gXYZ »
0012F00E0	62	58	59	5A	00	00	01	B8	00	00	00	14	72	54	52	43	bXYZ , rTRC
0012F00F0	00	00	01	CC	00	00	00	40	67	54	52	43	00	00	01	CC	i @gTRC i
0012F0100	00	00	00	40	62	54	52	43	00	00	01	CC	00	00	00	40	@bTRC i @
0012F0110	64	65	73	63	00	00	00	00	00	00	00	03	63	32	00	00	desc c2
0012F0120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012F0130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012F0140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012F0150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012F0160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012F0170	74	65	78	74	00	00	00	00	46	42	00	00	58	59	5A	20	text FB XYZ
0012F0180	00	00	00	00	00	00	F6	D6	00	01	00	00	00	00	D3	2D	öç Ö-
0012F0190	58	59	5A	20	00	00	00	00	00	00	03	16	00	00	03	33	XYZ 3
0012F01A0	00	00	02	A4	58	59	5A	20	00	00	00	00	00	00	6F	A2	»XYZ öç
0012F01B0	00	00	38	F5	00	00	03	90	58	59	5A	20	00	00	00	00	öç XYZ
0012F01C0	00	00	62	99	00	00	B7	85	00	00	18	DA	58	59	5A	20	b™ ... ÜXYZ
0012F01D0	00	00	00	00	00	24	A0	00	00	0F	84	00	00	B6	CF		\$ „ ¶İ
0012F01E0	63	75	72	76	00	00	00	00	00	00	1A	00	00	00	CB		curv È
0012F01F0	01	C9	03	63	05	92	08	6B	0B	F6	10	3F	15	51	1B	34	É c ' k ö ? Q 4
0012F0200	21	F1	29	90	32	18	3B	92	46	05	51	77	5D	ED	6B	70	!ñ) 2 ;'F Qw]ikp
0012F0210	7A	05	89	B1	9A	7C	AC	69	BF	7D	D3	C3	E9	30	FF	F6	z ½š -i;ÖÄéOyÿ

Image file content pointer is present after '08' (red) attribute whereas in text file, file pointer was present after '04' attribute. So file pointer attribute values are different for text file and image files.

4.4.2 Renaming image file

The same image file 'verse.jpg' is now renamed to be 'new_verse.jpg'. Observing the metadata changes:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001266330	20	00	00	80	00	00	00	00	00	06	00	00	00	00	00	00	€
001266340	0A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001266350	0C	00	12	00	76	00	65	00	72	00	73	00	65	00	2E	00	verse.
001266360	6A	00	70	00	67	00	00	00	40	04	00	00	10	00	1E	00	jpg ©
001266370	00	00	30	00	10	04	00	00	30	00	01	00	6E	00	65	00	o o ne
001266380	77	00	5F	00	76	00	65	00	72	00	73	00	65	00	2E	00	verse.
001266390	6A	00	70	00	67	00	00	00	A8	00	00	00	28	00	01	00	jpg (
0012663A0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00	
0012663B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012663C0	79	D7	B7	CC	59	9D	D1	01	03	49	33	0C	88	4B	CF	01	y× ·ÏY Ñ I3 ^KÏ
0012663D0	19	01	88	D0	79	9D	D1	01	79	D7	B7	CC	59	9D	D1	01	^Ëy Ñ y× ·ÏY Ñ
0012663E0	20	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00	
0012663F0	0A	00	00	00	00	00	00	00	88	1F	59	2F	01	00	00	00	^ Y/
001266400	E1	7B	01	00	00	00	00	00	00	00	02	00	00	00	00	00	á{
001266410	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001266420	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001266430	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001266440	20	00	00	00	A0	01	00	00	D4	00	00	00	00	02	00	00	ô
001266450	74	02	00	00	01	00	00	00	78	02	00	00	00	00	00	00	x
001266460	80	01	00	00	10	00	0E	00	08	00	20	00	60	01	00	00	·
001266470	60	01	00	00	00	00	00	00	80	00	00	00	00	00	00	00	€
001266480	88	00	00	00	28	00	01	00	01	00	00	00	20	01	00	00	^ (
001266490	20	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
0012664A0	00	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00	
0012664B0	00	00	00	00	00	00	02	00	00	00	00	00	E1	7B	01	00	á{
0012664C0	00	00	00	00	E1	7B	01	00	00	00	00	00	00	00	00	00	á{
0012664D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012664E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012664F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001266500	00	00	00	00	00	00	00	00	20	00	00	00	50	00	00	00	P
001266510	84	00	00	00	00	02	00	00	D4	00	00	00	01	00	00	00	ø
001266520	D8	00	00	00	00	00	00	00	30	00	00	00	10	00	10	00	ø
001266530	00	00	10	00	20	00	00	00	00	00	00	00	00	00	00	00	
001266540	08	00	00	00	00	00	00	00	BC	04	00	00	00	00	00	00	4
001266550	00	00	00	08	00	00	00	00	00	00	00	00	00	00	00	00	

Image file metadata of the renamed file is found at offset 0x126637C (pink) different from where the original image file metadata existed. Original name of the image file 'verse.jpg' is also present 16 bytes above new name of the image. MACE timestamp values are exact

duplicate of the original image file. Image size, permissions and file pointer are also exactly identical to the original file.

A duplicate of the renamed filename 'new_verse.jpg' is found after 460 bytes (0x12667DC) from the first offset (0x126637C).

0012667B0	00 00 28 00 28 00 00 00	20 00 00 80 00 00 00 00	((€
0012667C0	00 06 00 00 00 00 00 00	0A 00 00 00 00 00 00 00	
0012667D0	00 00 00 00 00 00 00 00	0C 00 1A 00 6E 00 65 00	ne
0012667E0	77 00 5F 00 76 00 65 00	72 00 73 00 65 00 2E 00	w_verse.
0012667F0	6A 00 70 00 67 00 00 00	00 00 00 00 00 00 00 00	jpg
001266800	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

4.4.3 Permissions changed

Opening the properties window of the image file and changing the permissions in the security tab is what is done in this scenario. The metadata of 'new_verse.jpg' as seen through WinHex is as follows:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Hex	ASCII
001262330	20	00	00	80	00	00	00	00	00	06	00	00	00	00	00	00	00	€
001262340	0A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262350	0C	00	12	00	76	00	65	00	72	00	73	00	65	00	2E	00	00	verse.
001262360	6A	00	70	00	67	00	00	00	40	04	00	00	10	00	1E	00	00	pe
001262370	00	00	30	00	10	04	00	00	30	00	01	00	6E	00	65	00	00	o ne
001262380	77	00	5F	00	76	00	65	00	72	00	73	00	65	00	2E	00	00	w_verse.
001262390	6A	00	70	00	67	00	00	00	A8	00	00	00	28	00	01	00	00	jpg
0012623A0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00	00	(
0012623B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012623C0	79	D7	B7	CC	S9	9D	D1	01	03	49	33	0C	88	4B	CF	01	00	yx·iY N i3 ·KI
0012623D0	9C	00	ED	64	8F	9D	D1	01	79	D7	B7	CC	S9	9D	D1	01	00	æ id N yx·iY N
0012623E0	20	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00	00	
0012623F0	0A	00	00	00	00	00	00	00	25	9B	A0	FC	01	00	00	00	00	š> ũ
001262400	E1	7B	01	00	00	00	00	00	00	00	02	00	00	00	00	00	00	á{
001262410	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262420	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262430	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262440	20	00	00	00	A0	01	00	00	D4	00	00	00	00	02	00	00	00	ô
001262450	74	02	00	00	01	00	00	00	78	02	00	00	00	00	00	00	00	t x
001262460	80	01	00	00	10	00	0E	00	08	00	20	00	60	01	00	00	00	é
001262470	60	01	00	00	00	00	00	00	80	00	00	00	00	00	00	00	00	é
001262480	88	00	00	00	28	00	01	00	01	00	00	00	20	01	00	00	00	· (
001262490	20	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	00	
0012624A0	00	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00	00	
0012624B0	00	00	00	00	00	02	00	00	00	00	00	00	E1	7B	01	00	00	á{
0012624C0	00	00	00	00	E1	7B	01	00	00	00	00	00	00	00	00	00	00	á{
0012624D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012624E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012624F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262500	00	00	00	00	00	00	00	00	20	00	00	00	50	00	00	00	00	P
001262510	84	00	00	00	02	00	00	00	D4	00	00	00	01	00	00	00	00	.. ô
001262520	D8	00	00	00	00	00	00	00	30	00	00	00	10	00	10	00	00	ø
001262530	00	00	10	00	20	00	00	00	00	00	00	00	00	00	00	00	00	
001262540	08	00	00	00	00	00	00	00	BC	04	00	00	00	00	00	00	00	4
001262550	00	00	00	08	00	00	00	00	00	00	00	00	00	00	00	00	00	

'new_verse.jpg' is now found at two different offset. One offset is of the file which was renamed 0x126637C (as discussed in the renaming image file section above) whereas the other offset is new where the metadata of the changed permission of image file resides.

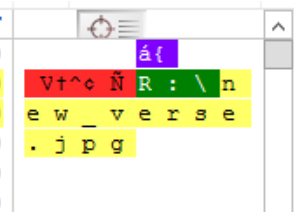
The metadata at this offset 0x126237C differs from renamed metadata in file permissions (green) and ‘entry modified’ (light orange) MACE timestamp (orange) only. Apart from these all other metadata is same. There is a duplicate filename after 460 bytes at 0x12627DC just like the renamed metadata had.

4.4.4 Deleting jpg file

In this scenario, file ‘new_verse.jpg’ is simply deleted from ReFS drive. It still exists in recycle bin.

The first offset where the image metadata is found after deletion is 0x116001E, shown below:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
001160000	01	00	00	00	00	00	00	00	E1	7B	01	00	00	00	00	00
001160010	20	56	86	5E	A2	9D	D1	01	52	00	3A	00	5C	00	6E	00
001160020	65	00	77	00	5F	00	76	00	65	00	72	00	73	00	65	00
001160030	2E	00	6A	00	70	00	67	00	00	00	00	00	00	00	00	00
001160040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001160050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00



File size of the deleted file has been highlighted in purple, starting eight bytes (red) show the time when file was deleted. The next six bytes (green) show the path of file, where it was residing and remaining bytes show the filename (yellow). This metadata after image deletion is identical to the text file metadata after deletion.

Image is also found at offset: 0x126237C, 0x126637C and at 0x126A37C. At offset 0x126237C metadata of image file is present when its permissions were changed and there is no change in it. At offset 0x126637C metadata of image is present when it was renamed. Metadata is same except that the ‘entry modified’ part of MACE time has now changed to be the same as that of when the file permissions were changed. Initially when file was renamed and MACE times were observed, it was the same as that of the original file, but now it has changed. Another fact is that the filename duplicate of this offset (0x126637C) which was ‘new_verse.jpg’ and existed after 460 bytes, at offset 0x12667DC, now exists after 472 bytes at 0x12667E8. It has now been cut short after file deletion. It is now ‘rse.jpg’ (brown).

0012667B0	00 00 28 00 18 00 00 00	20 00 00 80 00 00 00 00	(€
0012667C0	00 06 00 00 00 00 00 00	0A 00 00 00 00 00 00 00		
0012667D0	01 00 00 00 00 00 00 00	02 07 00 00 00 00 00 00		
0012667E0	00 00 00 00 00 00 00 00	72 00 73 00 65 00 2E 00		r s e .
0012667F0	6A 00 70 00 67 00 00 00	00 00 00 00 00 00 00 00	j p g	
001266800	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		

Metadata at 0x126A37C offset has only been added now when the file has been deleted and sent to recycle bin. It is identical to the metadata of the above offset and contains the cut out filename 'rse.jpg', exactly 472 bytes after the 0x126A37C offset.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00126A2F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00126A300	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00126A310	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00126A320	48	00	00	00	10	00	18	00	04	00	28	00	20	00	00	00	H
00126A330	20	00	00	80	00	00	00	00	00	06	00	00	00	00	00	00	€
00126A340	0A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00126A350	0C	00	12	00	76	00	65	00	72	00	73	00	65	00	2E	00	v e r s e .
00126A360	6A	00	70	00	67	00	00	00	40	04	00	00	10	00	1E	00	j p g @
00126A370	04	00	30	00	10	04	00	00	30	00	01	00	6E	00	65	00	o o n e
00126A380	77	00	5F	00	76	00	65	00	72	00	73	00	65	00	2E	00	w _ v e r s e .
00126A390	6A	00	70	00	67	00	00	00	A8	00	00	00	28	00	01	00	j p g (
00126A3A0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00	
00126A3B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00126A3C0	79	D7	B7	CC	59	9D	D1	01	03	49	33	0C	88	4B	CF	01	y x · i Y Ñ I 3 ^ K I
00126A3D0	9C	00	ED	64	8F	9D	D1	01	79	D7	B7	CC	59	9D	D1	01	id Ñ y x · i Y Ñ
00126A3E0	20	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00	
00126A3F0	0A	00	00	00	00	00	00	00	25	9B	A0	FC	01	00	00	00	è > ù
00126A400	E1	7B	01	00	00	00	00	00	00	00	02	00	00	00	00	00	á {
00126A410	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

00126A7B0	00 00 28 00 18 00 00 00	20 00 00 80 00 00 00 00	(€
00126A7C0	00 06 00 00 00 00 00 00	0A 00 00 00 00 00 00 00		
00126A7D0	01 00 00 00 00 00 00 00	02 07 00 00 00 00 00 00		
00126A7E0	00 00 00 00 00 00 00 00	72 00 73 00 65 00 2E 00		r s e .
00126A7F0	6A 00 70 00 67 00 00 00	00 00 00 00 00 00 00 00	j p g	
00126A800	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		

Another thing is observed where a section of filename has been cut out. 14 bytes above the cut out filename are two identical bytes '02 07' which are found only where the section of cut out filename is found. It can be deduced here that they show that the file has been deleted.

4.4.5 SHIFT + Delete jpg file (permanent delete)

Following metadata changes are observed when the image is permanently deleted:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
00126E350	0C	00	12	00	76	00	65	00	72	00	73	00	65	00	2E	00		v e r s e .
00126E360	6A	00	70	00	67	00	00	00	40	04	00	00	10	00	1E	00		j p c @
00126E370	04	00	30	00	10	04	00	00	30	00	01	00	6E	00	65	00		0 0 n e
00126E380	77	00	5F	00	76	00	65	00	72	00	73	00	65	00	2E	00		w _ v e r s e .
00126E390	6A	00	70	00	67	00	00	00	A8	00	00	00	28	00	01	00		j p g (
00126E3A0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00		
00126E3B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00126E3C0	79	D7	B7	CC	59	9D	D1	01	03	49	33	0C	88	4B	CF	01		y * - i Y N I 3 - R I
00126E3D0	9C	00	ED	64	8F	9D	D1	01	9	D7	B7	CC	59	9D	D1	01		e i d N * - i Y N
00126E3E0	20	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00		
00126E3F0	0A	00	00	00	00	00	00	00	25	9B	A0	FC	01	00	00	00		> u
00126E400	E1	7B	01	00	00	00	00	00	00	00	02	00	00	00	00	00		á {
00126E410	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00126E420	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00126E430	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00126E440	20	00	00	00	A0	01	00	00	D4	00	00	00	00	02	00	00		ô
00126E450	74	02	00	00	01	00	00	00	78	02	00	00	00	00	00	00		t x
00126E460	80	01	00	00	10	00	0E	00	08	00	20	00	60	01	00	00		e ,
00126E470	60	01	00	00	00	00	00	00	80	00	00	00	00	00	00	00		` e
00126E480	88	00	00	00	28	00	01	00	01	00	00	00	20	01	00	00		^ (
00126E490	20	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00		
00126E4A0	00	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00		
00126E4B0	00	00	00	00	00	02	00	00	00	00	00	00	E1	7B	01	00		á { á {
00126E4C0	00	00	00	00	E1	7B	01	00	00	00	00	00	00	00	00	00		á {
00126E4D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00126E4E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00126E4F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00126E500	00	00	00	00	00	00	00	00	20	00	00	00	50	00	00	00		P
00126E510	84	00	00	00	00	02	00	00	D4	00	00	00	01	00	00	00		„ ô
00126E520	D8	00	00	00	00	00	00	00	30	00	00	00	10	00	10	00		ø 0
00126E530	00	00	10	00	20	00	00	00	00	00	00	00	00	00	00	00		
00126E540	08	00	00	00	00	00	00	00	BC	04	00	00	00	00	00	00		{
00126E550	00	00	00	08	00	00	00	00	00	00	00	00	00	00	00	00		

New offset (0x126E37C) has been added now, where the deleted file data identical to other offsets resides (as shown above). 0x116001E, 0x126237C, 0x126637C, 0x126A37C which have already been discussed in the above sections, possess metadata even after deletion. The only difference in these offsets is they no longer contain the section of cut out filename (at duplicate filename area) that they possessed when the file was deleted and sent to recycle bin. After permanent deletion full name of the file exists.

It might be noted here that even after permanent deletion, image file still exists in metadata. This is of great help to forensic examiners while searching for data in drives.

4.4.6 Scenario analysis for jpg file

All scenarios applied on the jpg file are analyzed and data is presented in tabular format:

Operation	Filename (pink)	Artifacts Gathered				
		Metadata Offset	MACE times (orange)	Permission (green)	File size (yellow)	File pointer (blue)
Copying	verse.jpg	0x1262354	✓	✓	E1 7B 01	BC 04
Renaming	new_verse.jpg	0x126637C	✓	✓	E1 7B 01	BC 04
Permissions changed	new_verse.jpg	0x126237C	Entry modified part changes	changed	E1 7B 01	BC 04
Simple delete	new_verse.jpg	0x116001E	Contains file size, time of deletion, path of file and name of the deleted file			
	new_verse.jpg rse.jpg	0x126637C	Entry modified part changes	✓	E1 7B 01	BC 04
	new_verse.jpg rse.jpg	0x126A37C	✓	✓	E1 7B 01	BC 04
Shift delete	new_verse.jpg	0x126E37C	✓	✓	✓	✓

Table 4-3: Scenario analysis for jpg file

4.5 Folder scenarios

Folders have different format than files. In this section of the chapter, different operations are performed on folder to see the changes resilient file system undergoes when these operations are performed on folder.

4.5.1 Folder creation

In this scenario a new folder named “forensics” is created in ReFS drive. Another subfolder named “filesystem_forensics” is created inside forensics folder immediately after its creation. Looking at the metadata changes of ReFS drive, it is observed that folders are found at many different offsets with \$I30 attribute. We know that \$I30 is index attribute for directories and lists directory contents. *“NTFS maintains an index of all files/directories that belong to a directory called the \$I30 attribute. Every directory in the file system contains an \$I30 attribute that must be maintained whenever there are changes to the directory's contents. When files or folders are removed from the directory, the \$I30 index records are re-arranged accordingly.”* [20] Therefore it is present with folder offset.

Metadata Block Offset	Folder name	Starting Bytes	Eighth Byte	\$I30
0x1160356	New folder	58 04	B4	✓
0x1164356	forensics	59 04	B5	✓
0x1168356	forensics	5A 04	B6	✓
0x1168574	New folder	5A 04	B6	
0x116C356	forensics	5B 04	B7	✓
0x116C574	filesystem_forensics	68 04	B6	
0x11A0356	New folder	68 04	B6	✓
0x11A4356	filesystem_forensics	69 04	B7	✓
0x12627BC	forensics			
0x12667BC	New folder			
0x126A7BC	forensics			
0x126E7BC	forensics			

Table 4-4: Metadata offsets for folders

\$I30 is present exactly 122 bytes before folder offset. The above table does not include MACE times which were almost same (with some second's difference, if any).

Drive R:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001160000	58	04	00	00	00	00	00	00	B4	00	00	00	00	00	00	00	X
001160010	00	00	00	00	00	00	00	00	03	07	00	00	00	00	00	00	
001160020	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001160030	E8	00	00	00	28	00	01	00	00	00	00	00	30	01	00	00	è (0
001160040	30	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	0
001160050	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001160160	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
001160170	00	00	00	00	00	00	00	00	97	25	12	DE	5D	9E	D1	01	— 8 P] žÑ
001160180	97	25	12	DE	5D	9E	D1	01	97	25	12	DE	5D	9E	D1	01	— 8 P] žÑ — 8 P] žÑ
001160190	97	25	12	DE	5D	9E	D1	01	00	00	00	10	00	00	00	00	— 8 P] žÑ
0011601A0	03	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011601B0	AE	A2	E1	AE	01	00	00	00	00	00	00	00	00	00	00	00	0cá8
0011601C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0011601E0	01	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00	
0011601F0	00	00	00	00	00	00	00	00	20	00	00	00	78	01	00	00	x
001160200	F8	00	00	00	00	02	00	00	70	02	00	00	02	00	00	00	ø p
001160210	78	02	00	00	00	00	00	00	C0	00	00	00	10	00	14	00	x À
001160220	00	00	28	00	94	00	00	00	94	00	00	00	00	00	00	00	(" "
001160230	90	00	00	00	24	00	49	00	33	00	30	00	00	00	00	00	\$ I 3 0
001160240	00	00	00	00	80	00	00	00	0C	00	00	00	30	00	00	00	€ o
001160250	40	02	01	00	16	00	00	00	10	00	00	00	70	00	00	00	@ p
001160260	70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	p
001160270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001160280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001160290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011602A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011602B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011602C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011602D0	00	00	00	00	00	00	00	00	98	00	00	00	10	00	0E	00	~
0011602E0	00	00	20	00	74	00	00	00	74	00	00	00	00	00	00	00	t t
0011602F0	38	00	00	00	00	00	00	00	00	00	00	00	66	00	00	00	8 f
001160300	0C	00	01	00	00	06	00	00	00	00	00	00	00	00	00	00	
001160310	00	00	00	00	97	25	12	DE	5D	9E	D1	01	97	25	12	DE	—§ P}žÑ —§ P}
001160320	5D	9E	D1	01	97	25	12	DE	5D	9E	D1	01	97	25	12	DE	}žÑ —§ P}žÑ —§ P}
001160330	5D	9E	D1	01	00	00	00	00	00	00	00	00	00	00	00	00	}žÑ
001160340	00	00	00	00	00	00	00	10	00	00	00	00	0A	00	00	00	
001160350	73	00	30	00	02	00	4E	00	65	00	77	00	20	00	66	00	s 0 New f
001160360	6F	00	6C	00	64	00	65	00	72	00	00	00	00	00	00	00	o l d e r
001160370	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Folder name length is present ten bytes before every folder name. For ‘New folder’ it is ‘0A’. For ‘forensics’ it is ‘09’.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00116C530	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00116C540	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00116C550	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00116C560	88	00	00	00	10	00	2C	00	00	00	40	00	48	00	00	00	^ , @ H
00116C570	30	00	02	00	66	00	69	00	6C	00	65	00	73	00	79	00	o files y
00116C580	73	00	74	00	65	00	6D	00	5F	00	66	00	6F	00	72	00	s t e m _ f o r
00116C590	65	00	6E	00	73	00	69	00	63	00	73	00	00	00	00	00	e n s i c s
00116C5A0	04	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00116C5B0	BE	7B	EF	F7	5D	9E	D1	01	BE	7B	EF	F7	5D	9E	D1	01	%{i-}žÑ %{i-}žÑ
00116C5C0	BE	7B	EF	F7	5D	9E	D1	01	BE	7B	EF	F7	5D	9E	D1	01	%{i-}žÑ %{i-}žÑ
00116C5D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

4.5.2 Renaming folder

For this scenario, main folder named “forensics” is now renamed to be “research”. Following are metadata changes in tabular form:

Metadata Block Offset	Folder name (before renaming)	Starting Bytes	Eighth Byte	\$I30	Folder name (after renaming)	Eighth Byte	Identical
0x1160356	New folder	58 04	B4	✓	research	BF	
0x1160574				✓	filesystem_forensics		
0x1164356	forensics	59 04	B5	✓	forensics	B5	✓
0x1168356	forensics	5A 04	B6	✓	forensics	B6	✓
0x1168574	New folder	5A 04	B6		New folder		
0x116C356	forensics	5B 04	B7	✓	forensics	B7	✓
0x116C574	filesystem_forensics	68 04	B6		filesystem_forensics		
0x11A0356	New folder	68 04	B6	✓	New folder	B6	✓
0x11A4356	filesystem_forensics	69 04	B7	✓	filesystem_forensics	B7	✓
0x12627BC	forensics				forensics		✓
0x12667BC	New folder				research		
0x126A7BC	forensics				forensics		✓
0x126E7BC	forensics				forensics		✓

Table 4-5: Metadata changes for folders after renaming

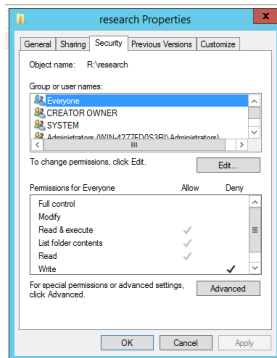
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F			
001160000	58	04	00	00	00	00	00	00	BF	00	00	00	00	00	00	00	X	z	
001160010	00	00	00	00	00	00	00	00	03	07	00	00	00	00	00	00			
001160020	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
001160030	E8	00	00	00	28	00	01	00	00	00	00	00	30	01	00	00	è	(0
001160040	30	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	0		
001160050	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			

Only the eighth byte of first offset where ‘research’ is found is different. All others are same. Only one new offset containing ‘filesystem_forensics’ is present after renaming (row highlighted in yellow). Two of the ‘New folder’ entries have been changed to ‘research’ whereas all others are almost same. MACE times have no difference after renaming folder.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001160160	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
001160170	00	00	00	00	00	00	00	00	97	25	12	DE	5D	9E	D1	01	-ε ρ] ζÑ
001160180	80	D5	FA	FD	5D	9E	D1	01	80	D5	FA	FD	5D	9E	D1	01	ε0úý] ζÑ ε0úý] ζÑ
001160190	80	D5	FA	FD	5D	9E	D1	01	10	00	00	00	00	00	00	00	ε0úý] ζÑ
0011601A0	03	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011601B0	AE	A2	E1	AE	01	00	00	00	00	00	00	00	00	00	00	00	εcás
0011601C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011601D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011601E0	01	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00	
0011601F0	00	00	00	00	00	00	00	00	20	00	00	00	78	01	00	00	x
001160200	F8	00	00	00	00	02	00	00	70	02	00	00	02	00	00	00	σ p
001160210	78	02	00	00	00	00	00	00	C0	00	00	00	10	00	14	00	x À
001160220	00	00	28	00	94	00	00	00	94	00	00	00	00	00	00	00	(" "
001160230	90	00	00	00	24	00	49	00	33	00	30	00	00	00	00	00	Σ I 3 0
001160240	00	00	00	00	80	00	00	00	0C	00	00	00	30	00	00	00	ε 0
001160250	40	02	01	00	16	00	00	00	10	00	00	00	70	00	00	00	@ p
001160260	70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	p
001160270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001160280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001160290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011602A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011602B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011602C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011602D0	00	00	00	00	00	00	00	00	98	00	00	00	10	00	0E	00	~
0011602E0	00	00	20	00	74	00	00	00	74	00	00	00	00	00	00	00	t t
0011602F0	38	00	00	00	00	00	00	00	00	00	00	00	62	00	00	00	8 b
001160300	0C	00	01	00	00	06	00	00	00	00	00	00	00	00	00	00	
001160310	00	00	00	00	97	25	12	DE	5D	9E	D1	01	80	D5	FA	FD	-ε ρ] ζÑ ε0úý
001160320	5D	9E	D1	01	80	D5	FA	FD	5D	9E	D1	01	80	D5	FA	FD] ζÑ ε0úý] ζÑ ε0úý
001160330	5D	9E	D1	01	00	00	00	00	00	00	00	00	00	00	00	00] ζÑ
001160340	00	00	00	00	10	00	00	10	00	00	00	00	08	00	00	00	
001160350	73	00	30	00	02	00	72	00	65	00	73	00	65	00	61	00	s 0 r e s e a
001160360	72	00	63	00	68	00	00	00	00	00	00	00	00	00	00	00	r c h
001160370	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

4.5.3 Permissions changed

'Research' folder permissions are changed as shown below:



Following are metadata changes after main folder permissions have been changed:

Metadata Block Offset	Folder name			Starting Bytes	Eighth Byte	Eighth Byte (after permission change)
	before renaming	after renaming	after permission change			
0x1160356	New folder	research	research	58 04	BF	C9
0x1160574		filesystem_forensics	filesystem_forensics			
0x1164356	forensics	forensics	research	59 04	B5	C3
0x1164574			filesystem_forensics			
0x1168356	forensics	forensics	research	5A 04	B6	C5
0x1168574	New folder	New folder	filesystem_forensics	5A 04		
0x116C356	forensics	forensics	forensics	5B 04	B7	B7
0x116C574	filesystem_forensics	filesystem_forensics	filesystem_forensics	68 04		
0x11A0356	New folder	New folder	filesystem_forensics	68 04	B6	C3
0x11A4356	filesystem_forensics	filesystem_forensics	filesystem_forensics	69 04	B7	C5
0x12627BC	forensics	forensics	research			
0x12667BC	New folder	research	research			
0x126A7BC	forensics	forensics	forensics			
0x126E7BC	forensics	forensics	forensics			

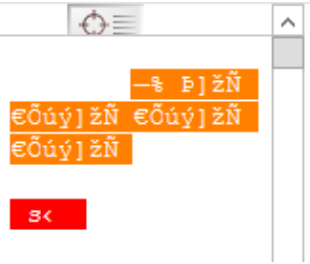
Table 4-6: Metadata changes for folders after permission change

One new offset containing (0x1164574) ‘filesystem_forensics’ folder name (highlighted in yellow in the above table) has been added when the permissions have been changed for main folder. Moreover it is observed that all the ‘New folder’ entries have been replaced with the subfolder name. MACE times are exactly the same.

One of the main differences is as follows:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001160160	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
001160170	00	00	00	00	00	00	00	00	97	25	12	DE	5D	9E	D1	01	— 3 P] žÑ
001160180	80	D5	FA	FD	5D	9E	D1	01	80	D5	FA	FD	5D	9E	D1	01	εōúý] žÑ εōúý] žÑ
001160190	80	D5	FA	FD	5D	9E	D1	01	10	00	00	00	00	00	00	00	εōúý] žÑ
0011601A0	03	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011601B0	AE	A2	E1	AE	01	00	00	00	00	00	00	00	00	00	00	00	⊗cá8
0011601C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
001160160	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00
001160170	00	00	00	00	00	00	00	00	97	25	12	DE	5D	9E	D1	01
001160180	80	D5	FA	FD	5D	9E	D1	01	80	D5	FA	FD	5D	9E	D1	01
001160190	80	D5	FA	FD	5D	9E	D1	01	10	00	00	00	00	00	00	00
0011601A0	03	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0011601B0	05	73	8B	20	01	00	00	00	00	00	00	00	00	00	00	00
0011601C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00



The bytes ‘AE A2 E1 AE 01’ have been changed to ‘05 73 8B 20 01’ where ever the main folder name ‘research’ appears. At offset 0x116C356, where ‘research’ folder’s original name ‘forensics’ is appearing the permissions are ‘AE A2 E1 AE 01’.

4.5.4 Adding content

For this scenario three files, one image file ‘open.jpg’, a word document ‘word.doc’ and a text file ‘cool.txt’ is added to the ‘filesystem_forensics’ subfolder to check their behavior inside folders created on ReFS drive.

Metadata Offset	Folder name	Offset	Sub folder name	Starting Bytes	Eighth byte	cool.txt	word.doc	open.jpg
0x1160356	research	0x1160574	filesystem_forensics	58 04	D8			
0x1164356	research	0x1164574	filesystem_forensics	59 04	D0			
0x1168356	research	0x1168574	filesystem_forensics	5A 04	D2			
0x116C356	research	0x116C574	filesystem_forensics	5B 04	D5			
		0x11A4356	filesystem_forensics	69 04	D0			
		0x11A8356	filesystem_forensics	6A 04	D1			
		0x11AC356	filesystem_forensics	6B 04	D1			
		0x11F0356	filesystem_forensics	7C 04	D8	0x11F1C0 4	0x11F1784	0x11F1304
		0x11F4356	filesystem_forensics	7D 04	D5		0x11F5784	0x11F5304
		0x11F8356	filesystem_forensics	7E 04	D9	0x11F9C 04	0x11F9784	0x11F9304

Table 4-7:Metadata changes after content addition in folder

It is observed that files added to the sub folder are located in the same order in the metadata in which they were added. ‘open.jpg’ was added before ‘word.doc’ therefore it is present before ‘word .doc’ as it is apparent from their offsets. ‘cool.txt’ was added to the folder in the last and it is apparent from its offset where ‘cool.txt’ is found.

There is one extra metadata entry for ‘doc’ and ‘jpg’ file. The shaded row in the above table is shown in the below images as seen through the editor.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0011EFFC0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011EFFD0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011EFFE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011EFFF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F0000	7C	04	00	00	00	00	00	00	D8	00	00	00	00	00	00	00	l
0011F0010	00	00	00	00	00	00	00	00	04	07	00	00	00	00	00	00	ø
0011F0020	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F0030	E8	00	00	00	28	00	01	00	00	00	00	00	30	01	00	00	è (0
0011F0040	30	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	0

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0011F01C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F01D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F01E0	07	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00	
0011F01F0	00	00	00	00	00	00	00	00	20	00	00	00	90	01	00	00	
0011F0200	E0	00	00	00	00	02	00	00	70	02	00	00	02	00	00	00	à p
0011F0210	78	02	00	00	00	00	00	00	C0	00	00	00	10	00	14	00	x À
0011F0220	00	00	28	00	94	00	00	00	94	00	00	00	00	00	00	00	(" "
0011F0230	90	00	00	00	24	00	49	00	33	00	30	00	00	00	00	00	\$ I 3 0
0011F0240	00	00	00	00	80	00	00	00	0C	00	00	00	30	00	00	00	€ o
0011F0250	40	02	01	00	16	00	00	00	10	00	00	00	70	00	00	00	è p
0011F0260	70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	p
0011F0270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F0280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F0290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F02A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F02B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F02C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F02D0	00	00	00	00	00	00	00	00	B0	00	00	00	10	00	0E	00	°
0011F02E0	00	00	20	00	8C	00	00	00	8C	00	00	00	00	00	00	00	€ €
0011F02F0	38	00	00	00	00	00	00	00	00	00	00	00	7A	00	00	00	8 z
0011F0300	0C	00	01	00	03	07	00	00	00	00	00	00	00	00	00	00	
0011F0310	00	00	00	00	BE	7B	EF	F7	5D	9E	D1	01	BE	7B	EF	F7	%{i÷}žÑ %{i÷
0011F0320	5D	9E	D1	01	BE	7B	EF	F7	5D	9E	D1	01	BE	7B	EF	F7]žÑ %{i÷}žÑ %{i÷
0011F0330	5D	9E	D1	01	00	00	00	00	00	00	00	00	00	00	00	00]žÑ
0011F0340	00	00	00	00	00	00	00	10	00	00	00	00	14	00	00	00	
0011F0350	FF	FF	30	00	02	00	66	00	69	00	6C	00	65	00	73	00	ÿÿ0 files
0011F0360	79	00	73	00	74	00	65	00	6D	00	5F	00	66	00	6F	00	system fo
0011F0370	72	00	65	00	6E	00	73	00	69	00	63	00	73	00	00	00	rensics
0011F0380	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0011F12D0	48	00	00	00	10	00	18	00	00	00	28	00	20	00	00	00	H (
0011F12E0	20	00	00	80	00	00	00	00	04	07	00	00	00	00	00	00	€
0011F12F0	04	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F1300	0C	00	10	00	6F	00	70	00	65	00	6E	00	2E	00	6A	00	open.j
0011F1310	70	00	67	00	FE	7F	00	00	38	04	00	00	10	00	14	00	pgp 8
0011F1320	08	00	28	00	10	04	00	00	30	00	01	00	6F	00	70	00	(0 op
0011F1330	65	00	6E	00	2E	00	6A	00	70	00	67	00	00	00	00	00	en.jpg
0011F1340	A8	00	00	00	28	00	01	00	00	00	00	00	10	01	00	00	" (
0011F1350	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
0011F1360	00	00	00	00	00	00	00	00	8F	B4	4C	E2	AC	A0	D1	01	'Lá-Ñ
0011F1370	0B	C4	13	5F	71	92	D1	01	0B	C4	13	5F	71	92	D1	01	Ä q'Ñ Ä q'Ñ
0011F1380	8F	B4	4C	E2	AC	A0	D1	01	20	00	00	00	00	00	00	00	'Lá-Ñ
0011F1390	04	07	00	00	00	00	00	00	04	00	00	00	00	00	00	00	
0011F13A0	56	95	C8	B0	01	00	00	00	7B	33	01	00	00	00	00	00	V•È° {3
0011F13B0	00	00	02	00	00	00	00	00	00	00	00	00	00	00	00	00	

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0011F1760	20	00	00	80	00	00	00	00	04	07	00	00	00	00	00	00	€
0011F1770	05	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F1780	0C	00	10	00	77	00	6F	00	72	00	64	00	2E	00	64	00	word.d
0011F1790	6F	00	63	00	00	C0	FF	FF	38	04	00	00	10	00	14	00	oc Àÿÿ8
0011F17A0	08	00	28	00	10	04	00	00	30	00	01	00	77	00	6F	00	(0 wo
0011F17B0	72	00	64	00	2E	00	64	00	6F	00	63	00	00	00	00	00	rd.doc
0011F17C0	A8	00	00	00	28	00	01	00	00	00	00	00	10	01	00	00	" (
0011F17D0	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
0011F17E0	00	00	00	00	00	00	00	00	56	B4	EA	BB	1A	A1	D1	01	V'ê» ;Ñ
0011F17F0	C0	62	4D	A1	1A	A1	D1	01	C0	62	4D	A1	1A	A1	D1	01	ÀbM; ;Ñ ÀbM; ;Ñ
0011F1800	56	B4	EA	BB	1A	A1	D1	01	20	00	00	00	00	00	00	00	V'ê» ;Ñ
0011F1810	04	07	00	00	00	00	00	00	05	00	00	00	00	00	00	00	
0011F1820	56	95	C8	B0	01	00	00	00	00	8C	00	00	00	00	00	00	V•È° 6
0011F1830	00	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0011F1BF0	06	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0011F1C00	0C	00	10	00	63	00	6F	00	6F	00	6C	00	2E	00	74	00	cool.t
0011F1C10	78	00	74	00	37	53	00	00	38	04	00	00	10	00	14	00	xt 7S 8
0011F1C20	08	00	28	00	10	04	00	00	30	00	01	00	63	00	6F	00	(0 co
0011F1C30	6F	00	6C	00	2E	00	74	00	78	00	74	00	00	00	00	00	ol.txt
0011F1C40	A8	00	00	00	28	00	01	00	00	00	00	00	10	01	00	00	" (
0011F1C50	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	
0011F1C60	00	00	00	00	00	00	00	00	8A	2D	7A	FB	7B	AB	D1	01	Š-zû{«Ñ
0011F1C70	9A	01	CB	DC	7B	AB	D1	01	9A	01	CB	DC	7B	AB	D1	01	š EÜ{«Ñ š EÜ{«Ñ
0011F1C80	8A	2D	7A	FB	7B	AB	D1	01	20	00	00	00	00	00	00	00	Š-zû{«Ñ
0011F1C90	04	07	00	00	00	00	00	00	06	00	00	00	00	00	00	00	
0011F1CA0	56	95	C8	B0	01	00	00	00	63	08	00	00	00	00	00	00	V•È° c
0011F1CB0	00	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	

4.5.5 Compressing folder

In this scenario a folder named 'current' is made inside the ReFS drive that contains two files 'new.txt' and 'open.jpg'. This folder is then compressed and named 'current_compressed.zip'. The metadata of the compressed folder is as follows:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001262800	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262810	10	00	00	10	00	00	00	00	70	00	00	00	10	00	12	00	
001262820	00	00	28	00	48	00	00	00	30	00	02	00	63	00	75	00	(H 0 c u
001262830	72	00	72	00	65	00	6E	00	74	00	00	00	00	00	00	00	r r e n t
001262840	05	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262850	DC	54	36	69	AD	A0	D1	01	DD	C0	61	7A	AD	A0	D1	01	ÚT6i- Ñ YÀaz- Ñ
001262860	DD	C0	61	7A	AD	A0	D1	01	DD	C0	61	7A	AD	A0	D1	01	YÀaz- Ñ YÀaz- Ñ
001262870	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262880	00	00	00	10	00	00	00	00	50	00	00	00	10	00	18	00	
001262890	04	00	28	00	28	00	00	00	20	00	00	80	00	00	00	00	
0012628A0	00	06	00	00	00	00	00	00	0B	00	00	00	00	00	00	00	
0012628B0	00	00	00	00	00	00	00	00	0C	00	16	00	63	00	75	00	c u
0012628C0	72	00	72	00	65	00	6E	00	74	00	2E	00	7A	00	69	00	r r e n t . z i
0012628D0	70	00	E7	01	00	C0	FF	FF	50	04	00	00	10	00	30	00	ç ÀÿÿP 0
0012628E0	00	00	40	00	10	04	00	00	30	00	01	00	63	00	75	00	@ 0 c u
0012628F0	72	00	72	00	65	00	6E	00	74	00	5F	00	63	00	6F	00	r r e n t _ c o
001262900	6D	00	70	00	72	00	65	00	73	00	73	00	65	00	64	00	m p r e s s e d
001262910	2E	00	7A	00	69	00	70	00	A8	00	00	00	28	00	01	00	. z i p " (
001262920	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00	
001262930	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262940	00	22	E6	B2	BF	BA	D1	01	B1	E7	28	B3	BF	BA	D1	01	"æ°ç°Ñ ±ç(°ç°Ñ
001262950	47	49	CE	C0	BF	BA	D1	01	00	22	E6	B2	BF	BA	D1	01	GIÏÀç°Ñ "æ°ç°Ñ
001262960	20	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00	
001262970	0B	00	00	00	00	00	00	00	88	1F	59	2F	01	00	00	00	^ Y/
001262980	EA	33	01	00	00	00	00	00	00	00	02	00	00	00	00	00	è3

The grey highlighted part of the screenshot shows the name of the original folder (current), its MACE times and 'current.zip' is when the folder was compressed. This current.zip was renamed to 'current_compressed.zip' immediately after its compression hence both current.zip and current_compressed.zip are found side by side and have the same MACE times. File permissions are highlighted in green whereas file size 'EA 33 01' (highlighted in yellow) converts to '01 33 EA' in big Endian and we get the value 78,826 in decimal. This is exactly the same file size as shown in file properties window. Below image shows file pointer '94 04' which converts to '1250000'.

001262A90	50 00 00 00 00 02 00 00	D0 00 00 00 02 00 00 00	P	Ð
001262AA0	D8 00 00 00 00 00 00 00	30 00 00 00 10 00 10 00	Ø	0
001262AB0	00 00 10 00 20 00 00 00	00 00 00 00 00 00 00 00		
001262AC0	04 00 00 00 00 00 00 00	94 04 00 00 00 00 00 00		"
001262AD0	00 00 00 08 00 00 00 00	30 00 00 00 10 00 10 00		0

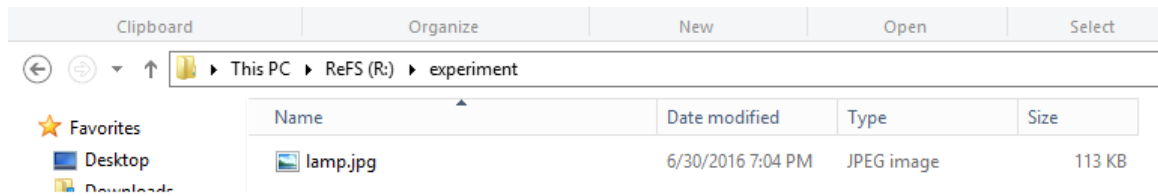
Looking at the file content pointer offset '1250000':

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
001250000	50	4B	03	04	14	00	00	00	08	00	CD	A9	9B	48	A0	94	PK	ie>H "
001250010	26	03	44	00	00	00	4A	00	00	00	0F	00	00	00	63	75	& D J	cu
001250020	72	72	65	6E	74	2F	6E	65	77	2E	74	78	74	0D	C2	81		urrent/new.txt
001250030	09	80	30	0C	04	C0	55	DE	05	44	3B	85	03	08	2E	D0		€0 ÀUP D;... .D
001250040	A8	0F	31	81	A4	5A	DC	5E	8F	5B	44	D5	D1	3D	B4	0E		" 1 «ZÜ^ [DÖÑ='
001250050	58	4F	26	FE	21	49	A5	58	C3	4E	15	E4	9B	4D	2E	C4		XO&p!I¥XÂN ä>M.Ä
001250060	6D	46	3B	40	C3	46	AB	DE	13	29	F1	48	A0	4C	73	19		mF;@ÄF«P)ñH Ls
001250070	3F	50	4B	03	04	14	00	00	00	08	00	CA	A0	89	48	F5		?PK È «HÖ
001250080	64	C7	E2	BA	32	01	00	7B	33	01	00	10	00	00	00	63		dÇá°2 {3 c
001250090	75	72	72	65	6E	74	2F	6F	70	65	6E	2E	6A	70	67	94		urrent/open.jpg"
0012500A0	7B	65	54	1D	4D	D0	E6	70	B9	10	9C	8B	3B	5C	DC	83		{eT Mðap³ æ<;\Üf
0012500B0	BB	6B	70	77	77	77	09	41	83	BB	3B	17	77	87	E0	0E		>kpwww Af>; w+à
0012500C0	09	09	EE	1E	3C	C1	35	04	0B	C1	C2	F2	0E	FD	F6	C7		i <Á5 ÁÄðžýöç
0012500D0	EE	BF	AD	9E	E9	73	BA	6B	A6	A7	BA	7A	EA	A9	AA	39		i¿-žés°k!S°zè€°9
0012500E0	3D	2F	DF	5F	7E	00	10	05	59	79	59	00	0E	0E	00	E0		=/B ~ YyY à
0012500F0	5E	0B	F0	B2	0E	48	01	60	78	F8	FF	0E	30	3C	18	01		^ 8° H `xøý 0<
001250100	0C	46	40	44	79	83	F8	4A	E8	C8	28	48	28	98	E8	10		F@DyføJèÈ(H(~è
001250110	08	26	3A	26	06	16	0E	21	2E	16	36	01	36	06	26	1E		&:& !. 6 6 &
001250120	29	1E	01	11	11	09	09	09	04	97	8C	82	8C	98	82	90) -E,€~,
001250130	98	84	F8	BF	41	E0	C0	08	08	88	08	88	68	6F	DE	A0		„ ø¿AàÀ ^ `hoP
001250140	11	63	61	62	11	FF	7F	D3	CB	20	80	85	04	B7	0B	5A		cab ý ÓÈ €... · Z
001250150	82	87	C3	02	40	58	70	F0	58	70	2F	DF	00	B2	57	39		,+Ä @XpðXp/B °W9
001250160	11	E0	E0	FE	93	F7	7F	08	0E	00	BD	CA	89	F8	06	09		ààp"- «È«ø
001250170	F9	95	4D	FA	5F	EB	FF	62	03	70	FF	C3	7E	59	03	D0		ù•Mú_ëyb pýÄ~Y Ð
001250180	E0	5F	D9	10	78	C8	6B	EF	9F	99	AA	80	2A	24	DE	E5		à_Ü xÈkiÝ°«*\$Pá
001250190	60	3A	38	C4	D7	42	59	4C	59	8C	B8	11	35	34	46	A4		`:8Ä×BYLYQ, 54F×
0012501A0	6C	9C	14	7F	10	5E	D9	30	C1	63	B0	5C	92	67	E9	80		læ ^ÙOÄc°\ 'gèÈ

Zipped folder contents are shown in the above image. It can be noted here that filename and their paths are present. Rest of the content is of the files that are present inside the current folder. 'PK' (highlighted red) is the format in which the folder has been zipped.

4.5.6 Deleting folder

For this scenario, a folder ‘experiment’ is created and an image file ‘lamp.jpg’ is placed inside it. This folder is then simply deleted (it is highlighted and deleted). By simple delete this folder goes to the recycle bin. Checking out the metadata changes:



All the metadata of the ‘experiment’ folder is similar to the metadata as discussed in the ‘folder creation’ section. The new entry of ‘experiment’ folder that appears after the folder has been deleted is present at offset 0x12F001E.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0012EFFC0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012EFFD0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012EFFE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012EFFF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012F0000	01	00	00	00	00	00	00	00	46	C0	01	00	00	00	00	00	EÀ
0012F0010	30	BE	95	78	C8	D5	D1	01	52	00	3A	00	5C	00	65	00	0%*xÈÖÑ R : \ e
0012F0020	78	00	70	00	65	00	72	00	69	00	6D	00	65	00	6E	00	x p e r i m e n
0012F0030	74	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	t
0012F0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Folder size of the deleted folder has been highlighted in purple (above), starting eight bytes (red) show the time when folder was deleted. The next six bytes (green) show the path of file, where it was residing and remaining bytes show the folder name (yellow).

At offset 0x 12B8356, ‘\$I30’ attribute exists which is created every time any changes occur to folder (as discussed above). This time the change is deletion of folder hence the index attribute comes into play.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0012B81E0	02	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00
0012B81F0	00	00	00	00	00	00	00	00	20	00	00	00	78	01	00	00
0012B8200	F8	00	00	00	00	02	00	00	70	02	00	00	02	00	00	00
0012B8210	78	02	00	00	00	00	00	00	C0	00	00	00	10	00	14	00
0012B8220	00	00	28	00	94	00	00	00	94	00	00	00	00	00	00	00
0012B8230	90	00	00	00	24	00	49	00	33	00	30	00	00	00	00	00
0012B8240	00	00	00	00	80	00	00	00	0C	00	00	00	00	30	00	00
0012B8250	40	02	01	00	16	00	00	00	10	00	00	00	70	00	00	00
0012B8260	70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0012B8270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0012B8280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0012B8290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0012B82A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0012B82B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0012B82C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0012B82D0	00	00	00	00	00	00	00	00	98	00	00	00	10	00	0E	00
0012B82E0	00	00	20	00	74	00	00	00	74	00	00	00	00	00	00	00
0012B82F0	38	00	00	00	00	00	00	00	00	00	00	00	66	00	00	00
0012B8300	0C	00	01	00	00	06	00	00	00	00	00	00	00	00	00	00
0012B8310	00	00	00	00	08	E3	12	15	56	D4	D1	01	08	E3	12	15
0012B8320	56	D4	D1	01	08	E3	12	15	56	D4	D1	01	08	E3	12	15
0012B8330	56	D4	D1	01	00	00	00	00	00	00	00	00	00	00	00	00
0012B8340	00	00	00	00	00	00	00	10	00	00	00	0A	00	00	00	00
0012B8350	73	00	30	00	02	00	65	00	78	00	70	00	65	00	72	00
0012B8360	69	00	6D	00	65	00	6E	00	74	00	00	00	00	00	00	00
0012B8370	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Another important fact is the addition of deleted folder ‘experiment’ in the vicinity of recycle bin. This deleted folder name appears after 2826 bytes of Recycle bin entry successively. It is shown:

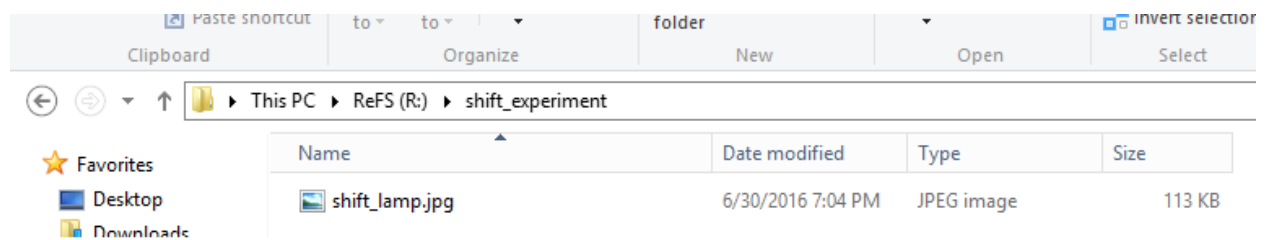
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
001268550	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001268560	78	00	00	00	10	00	1C	00	00	00	30	00	48	00	00	00
001268570	30	00	02	00	24	00	52	00	45	00	43	00	59	00	43	00
001268580	4C	00	45	00	2E	00	42	00	49	00	4E	00	00	00	00	00
001268590	01	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0012685A0	92	DA	48	6C	C8	32	D1	01	92	DA	48	6C	C8	32	D1	01
0012685B0	92	DA	48	6C	C8	32	D1	01	92	DA	48	6C	C8	32	D1	01
0012685C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0012685D0	06	00	00	10	00	00	00	00	60	00	00	00	10	00	18	00
0012685E0	04	00	28	00	38	00	00	00	20	00	00	80	00	00	00	00
0012685F0	00	06	00	00	00	00	00	00	02	00	00	00	00	00	00	00

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00126AD90	04	00	28	00	48	00	00	00	30	00	02	00	65	00	78	00
00126ADA0	70	00	65	00	72	00	69	00	6D	00	65	00	6E	00	74	00
00126ADB0	06	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00126ADC0	08	E3	12	15	56	D4	D1	01	08	E3	12	15	56	D4	D1	01
00126ADD0	08	E3	12	15	56	D4	D1	01	08	E3	12	15	56	D4	D1	01
00126ADE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00126ADF0	10	00	00	10	00	00	00	00	00	00	00	00	00	00	00	00
00126AE00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

The recycle bin entry appears at 0x1268576 whereas the deleted folder entry appears at 0x126AD9C.

4.5.7 Shift+ Deleting a folder

For this scenario, same folder that was used for delete folder scenario is renamed as 'shift_experiment' is copied to the ReFS drive and same image file 'shift_lamp.jpg' is placed inside it. This folder is then shift deleted (permanent delete). By shift delete this folder does not go to the recycle bin. It is permanently deleted. Checking out the metadata changes:



\$I30 attribute is present at offset 0x1300234 (highlighted in purple) along with the permanently deleted folder name (highlighted in pink) and MACE times (highlighted in orange), as shown below:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0013001F0	00	00	00	00	00	00	00	00	20	00	00	00	88	01	00	00	
001300200	E8	00	00	00	00	02	00	00	70	02	00	00	02	00	00	00	è p
001300210	78	02	00	00	00	00	00	00	C0	00	00	00	10	00	14	00	x À
001300220	00	00	28	00	94	00	00	00	94	00	00	00	00	00	00	00	(" "
001300230	90	00	00	00	24	00	49	00	33	00	30	00	00	00	00	00	\$ I 3 0
001300240	00	00	00	00	80	00	00	00	0C	00	00	00	30	00	00	00	€ 0
001300250	40	02	01	00	16	00	00	00	10	00	00	00	70	00	00	00	@ p
001300260	70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	p
001300270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001300280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001300290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0013002A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0013002B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	█
0013002C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0013002D0	00	00	00	00	00	00	00	00	A8	00	00	00	10	00	0E	00	..
0013002E0	00	00	20	00	84	00	00	00	84	00	00	00	00	00	00	00	" "
0013002F0	38	00	00	00	00	00	00	00	00	00	00	00	72	00	00	00	8 r
001300300	0C	00	01	00	00	06	00	00	00	00	00	00	00	00	00	00	
001300310	00	00	00	00	39	17	C1	96	D6	D5	D1	01	39	17	C1	96	9 Á-ÖÖÑ 9 Á-
001300320	D6	D5	D1	01	39	17	C1	96	D6	D5	D1	01	39	17	C1	96	ÖÖÑ 9 Á-ÖÖÑ 9 Á-
001300330	D6	D5	D1	01	00	00	00	00	00	00	00	00	00	00	00	00	ÖÖÑ
001300340	00	00	00	00	00	00	00	10	00	00	00	00	10	00	00	00	
001300350	73	00	30	00	02	00	73	00	68	00	69	00	66	00	74	00	s 0 s h i f t
001300360	5F	00	65	00	78	00	70	00	65	00	72	00	69	00	6D	00	_ e x p e r i m
001300370	65	00	6E	00	74	00	00	00	00	00	00	00	00	00	00	00	e n t
001300380	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Even after permanent deletion, the folder 'shift_experiment' appears in recycle bin entry's vicinity after 2826 bytes like it appeared in simple deletion scenario. It can be observed below:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001260550	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001260560	78	00	00	00	10	00	1C	00	00	00	30	00	48	00	00	00	x 0 H
001260570	30	00	02	00	24	00	52	00	45	00	43	00	59	00	43	00	0 \$ R E C Y C
001260580	4C	00	45	00	2E	00	42	00	49	00	4E	00	00	00	00	00	L E . B I N
001260590	01	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0012605A0	92	DA	48	6C	C8	32	D1	01	92	DA	48	6C	C8	32	D1	01	'ÓH1È2Ñ 'ÓH1È2Ñ
0012605B0	92	DA	48	6C	C8	32	D1	01	92	DA	48	6C	C8	32	D1	01	'ÓH1È2Ñ 'ÓH1È2Ñ
0012605C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001262D90	04	00	38	00	48	00	00	00	30	00	02	00	73	00	68	00	8 H 0 s h
001262DA0	69	00	66	00	74	00	5F	00	65	00	78	00	70	00	65	00	i f t _ e x p e
001262DB0	72	00	69	00	6D	00	65	00	6E	00	74	00	00	00	00	00	r i m e n t
001262DC0	07	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262DD0	39	17	C1	96	D6	D5	D1	01	C8	A9	C4	96	D6	D5	D1	01	9 Å-ÖÖÑ ÈèÀ-ÖÖÑ
001262DE0	C8	A9	C4	96	D6	D5	D1	01	C8	A9	C4	96	D6	D5	D1	01	ÈèÀ-ÖÖÑ ÈèÀ-ÖÖÑ
001262DF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001262E00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00	00	

Another important change in simple deletion and permanent deletion is the presence of ‘38 00 48’ before permanently deleted folder name appears, while in simple deletion scenario these bytes take the form ‘28 00 48’ (highlighted above in red for both scenarios). These bytes point out the difference in simple deletion and permanent deletion.

In permanent deletion, there is no such block that shows the path of folder, time of deletion of folder, folder size and name of folder that is deleted as in simple deletion scenario. From this we can distinguish whether a folder has been deleted simply or permanently.

We see that even after permanent deletion folder name exist at backend which is of great importance for forensic examiners because this can be used for gathering facts about deleted folders.

4.6 Exploring deletion in ReFS in detail

In this scenario, file deletion is explored in detail and explained in tabular form. For this scenario, different files are placed in ReFS drive to check their behavior. These files are first simply deleted and then permanently deleted. The differences are as follows:

4.6.1 Deletion in image file

First file is a jpg image file named as ‘hello.jpg’. Following things were observed in the hexadecimal of ReFS drive:

Metadata Block Offset	Filename				
	before deletion	after simple deletion		after permanent deletion	
0x157709C	hello.jpg	hello.jpg	00	hello.jpg	00
0x157B09C	hello.jpg	hello.jpg	00	hello.jpg	00
0x157F0C4		jpg	00	jpg	00
0x15730C4				jpg	04
0x 159001E	File path after simple deletion				
0x1580000	File contents				

Table 4-8: Metadata changes after image deletion

Initially when the image is placed in drive, image metadata is found at first two offsets.

When it is simply deleted and sent to recycle bin third and fifth offsets are added. At third offset cut out filename appears. Immediately above cut out filename ‘02 07’ bytes appear. At fifth offset file path after deletion appears.

Finally when the image is deleted from recycle bin, fourth offset comes into play where cut out filename appears. Before the file is permanently deleted, 56 bytes above the cut out filename ‘04’ appears which was previously ‘00’ when the file was copied and even after simple deletion.

File contents remain intact even after permanent deletion of file.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
001573050	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00
001573060	00	00	00	00	00	00	00	00	40	00	00	00	10	00	18	00
001573070	04	00	28	00	18	00	00	00	20	00	00	80	00	00	00	00
001573080	00	06	00	00	00	00	00	00	13	00	00	00	00	00	00	00
001573090	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00
0015730A0	00	00	00	00	00	00	00	00	6A	00	70	00	67	00	50	00
0015730B0	38	04	00	00	10	00	16	00	0C	00	28	00	10	04	00	00
0015730C0	30	00	01	00	68	00	65	00	6C	00	6C	00	6F	00	2E	00
0015730D0	6A	00	70	00	67	00	00	00	A8	00	00	00	28	00	01	00
0015730E0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00
0015730F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001573100	D2	ED	A1	BD	2C	DC	D1	01	96	50	E4	EA	B4	0C	CD	01
001573110	96	50	E4	EA	B4	0C	CD	01	D2	ED	A1	BD	2C	DC	D1	01
001573120	20	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00
001573130	13	00	00	00	00	00	00	00	88	1F	59	2F	01	00	00	00
001573140	CA	67	00	00	00	00	00	00	00	00	01	00	00	00	00	00
001573150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001573160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

It might be noted here that even after permanent deletion file contents and deletion path still remain at the same offsets and are not overwritten even when new files are placed in the drive. File metadata at offset other than that which had been added when the file was

permanently deleted (0x15730C4), are overwritten when any other file is placed in drive. The only trace of permanently deleted file exists at offset 0x15730C4 and deletion path at offset 0x159001E.

4.6.2 Deletion in txt file

Second file is a 'txt' file named as 'index.txt'. Following things were observed in the hexadecimal of ReFS drive:

Metadata Block Offset	Filename				
	before deletion	after simple deletion		after permanent deletion	
0x157709C	index.txt	index.txt	00	txt	04
0x157B09C	index.txt	index.txt	00	index.txt	00
0x157F09C	index.txt	index.txt	00	index.txt	00
0x15730C4		txt	00	txt	00
0x15B001E	File path after simple deletion				
0x15A0000	File contents				

Table 4-9: Metadata changes after txt file deletion

The same offset (0x15730C4) where permanently deleted metadata of 'hello.jpg' existed has now been overwritten and allocated to 'index.txt' when index.txt has been simply deleted.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
001573050	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00
001573060	00	00	00	00	00	00	00	00	40	00	00	00	10	00	18	00
001573070	00	00	28	00	18	00	00	00	20	00	00	80	00	00	00	00
001573080	00	06	00	00	00	00	00	00	14	00	00	00	00	00	00	00
001573090	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00
0015730A0	00	00	00	00	00	00	00	00	74	00	78	00	74	00	00	00
0015730B0	38	04	00	00	10	00	16	00	0C	00	28	00	10	04	00	00
0015730C0	30	00	01	00	69	00	6E	00	64	00	65	00	78	00	2E	00
0015730D0	74	00	78	00	74	00	00	00	A8	00	00	00	28	00	01	00
0015730E0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00
0015730F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001573100	6A	F5	85	26	41	DC	D1	01	EF	BF	12	1B	41	DC	D1	01
001573110	EF	BF	12	1B	41	DC	D1	01	6A	F5	85	26	41	DC	D1	01
001573120	20	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00
001573130	14	00	00	00	00	00	00	00	88	1F	59	2F	01	00	00	00
001573140	DC	01	00	00	00	00	00	00	00	00	01	00	00	00	00	00
001573150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001573160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

After permanent deletion, metadata at first offset is modified. A section of filename is cut out. Moreover the bytes '04' appear at the same position as they appeared in 'hello.jpg' scenario. This signifies that these bytes change from '00' to '04' only when permanent deletion occurs. Another important change is found where cut out filename appears bytes '48' change to '40'.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
001577050	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00
001577060	00	00	00	00	00	00	00	00	40	00	00	00	10	00	18	00
001577070	04	00	28	00	18	00	00	00	20	00	00	80	00	00	00	00
001577080	00	06	00	00	00	00	00	00	14	00	00	00	00	00	00	00
001577090	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00
0015770A0	00	00	00	00	00	00	00	00	74	00	78	00	74	00	00	00
0015770B0	38	04	00	00	10	00	16	00	0C	00	28	00	10	04	00	00
0015770C0	30	00	01	00	69	00	6E	00	64	00	65	00	78	00	2E	00
0015770D0	74	00	78	00	74	00	00	00	A8	00	00	00	28	00	01	00
0015770E0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00157B010	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00
00157B020	00	00	00	00	00	00	00	00	40	00	00	00	10	00	18	00
00157B030	00	00	28	00	18	00	00	00	20	00	00	80	00	00	00	00
00157B040	00	06	00	00	00	00	00	00	12	00	00	00	00	00	00	00
00157B050	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00
00157B060	00	00	00	00	00	00	00	00	48	00	00	00	10	00	18	00
00157B070	00	00	28	00	20	00	00	00	20	00	00	80	00	00	00	00
00157B080	00	06	00	00	00	00	00	00	14	00	00	00	00	00	00	00
00157B090	00	00	00	00	00	00	00	00	0C	00	12	00	69	00	6E	00
00157B0A0	64	00	65	00	78	00	2E	00	74	00	78	00	74	00	00	00
00157B0B0	38	04	00	00	10	00	16	00	08	00	28	00	10	04	00	00
00157B0C0	30	00	01	00	69	00	6E	00	64	00	65	00	78	00	2E	00
00157B0D0	74	00	78	00	74	00	00	00	A8	00	00	00	28	00	01	00
00157B0E0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00

4.6.3 Deletion in doc file

Third file is a 'doc' file named as 'assignment.doc'. Following things were observed in the drive:

Metadata Block Offset	Filename				
	before deletion	after simple deletion		after permanent deletion	
0x157B09C	assignment.doc	assignment.doc	00	ment.doc	04
0x157F09C	assignment.doc	assignment.doc	00	assignment.doc	00
0x15730A8		ment.doc	00	ment.doc	00
0x15770A8		ment.doc	00	ment.doc	00
0x130001E	File path after simple deletion				
0x1330000	File contents				

Table 4-10: Metadata changes after doc file deletion

It was observed that the first two offsets are the same that appeared in ‘index.txt’ scenario, which means that after permanent deletion of ‘index.txt’ that space is unallocated and is overwritten when new files are placed in the drive.

At third and fourth offset that are added when file is simply deleted, cut out filename appears. Above cut out filename ‘02 07’ appear. Also the byte ‘50’ is changed to ‘40’ after simple deletion where cut out filename is found.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00157B050	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00
00157B060	00	00	00	00	00	00	00	00	40	00	00	00	10	00	18	00
00157B070	04	00	28	00	18	00	00	00	20	00	00	80	00	00	00	00
00157B080	00	06	00	00	00	00	00	00	15	00	00	00	00	00	00	00
00157B090	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00
00157B0A0	00	00	00	00	00	00	00	00	6D	00	65	00	6E	00	74	00
00157B0B0	2E	00	64	00	6F	00	63	00	40	04	00	00	10	00	20	00
00157B0C0	0C	00	30	00	10	04	00	00	30	00	01	00	61	00	73	00
00157B0D0	73	00	69	00	67	00	6E	00	6D	00	65	00	6E	00	74	00
00157B0E0	2E	00	64	00	6F	00	63	00	A8	00	00	00	28	00	01	00
00157B0F0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00157F050	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00
00157F060	00	00	00	00	00	00	00	00	50	00	00	00	10	00	18	00
00157F070	00	00	28	00	28	00	00	00	20	00	00	80	00	00	00	00
00157F080	00	06	00	00	00	00	00	00	15	00	00	00	00	00	00	00
00157F090	00	00	00	00	00	00	00	00	0C	00	1C	00	61	00	73	00
00157F0A0	73	00	69	00	67	00	6E	00	6D	00	65	00	6E	00	74	00
00157F0B0	2E	00	64	00	6F	00	63	00	40	04	00	00	10	00	20	00
00157F0C0	08	00	30	00	10	04	00	00	30	00	01	00	61	00	73	00
00157F0D0	73	00	69	00	67	00	6E	00	6D	00	65	00	6E	00	74	00
00157F0E0	2E	00	64	00	6F	00	63	00	A8	00	00	00	28	00	01	00
00157F0F0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00

We can conclude based on the above scenarios that simple deletion cut shorts the file name in resilient file system. File contents remain intact. After permanent deletion, one byte changes to 04 and other changes to 40. Permanent deletion hides the file from file explorer and recycle bin but the metadata of the file still exists at hexadecimal level. This is very helpful for forensic examiners as deleted files can be easily recovered which would then help in further examinations.

4.7 Exploring the trimming of filename after simple deletion

For this scenario a txt file's name 'beautiful.txt' is repeatedly changed to check the cutting of filename when simple deletion occurs and it is placed in ReFS drive. The data collected is presented in tabular form as follows:

Filename	Filename after simple deletion
b.txt	Cut out filename does not exists
be.txt	Cut out filename does not exists
bea.txt	t
beau.txt	xt
beaut.txt	txt
beauti.txt	.txt
beautif.txt	f.txt
beautiful.txt	ful.txt
beautifullest.txt	fullest.txt

Table 4-11: Filename trimming after simple deletion

When file with filename 'beautiful.txt' is placed in ReFS drive and simply deleted which sends it to the recycle bin, the filename after deletion is cut short and appears as 'ful.txt'. More tests are performed by changing the number of alphabets in filename to check when the filename is cut.

It is noted that when the file name comprises of one alphabet only 'b.txt', the cut out filename does not come into play. In ReFS, when a file is placed in drive, the filename appears in the metadata along with its duplicate, as discussed in the '.txt experiments' section. It is this place where the cut out filename appears when file is deleted. In filename

that comprise one (b.txt) or two (be.txt) alphabets this duplicate name does not exist. Only the filename appears.

When the number of alphabets is increased to three, it is noted that the cut out name starts to appear. It must be noted here that this cut out name starts to appear from backwards (in the descending order of the filename).

For 'bea.txt' after deletion 't' appears above filename. This 't' is from the txt present at the end of filename.

For 'beau.txt' after deletion 'xt' appears. For 'beautif.txt' after deletion 'f.txt' appears.

Increasing alphabets in the filename and then deleting the file cut shorts the filename and increases the alphabets in the cut out filename. For 'beautifullest.txt' after deletion 'fullest.txt' appears.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0016B71F0	1E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0016B7200	0C	00	0A	00	62	00	2E	00	74	00	78	00	74	00	00	00	b . t x t
0016B7210	30	04	00	00	10	00	0E	00	08	00	20	00	10	04	00	00	0
0016B7220	30	00	01	00	62	00	2E	00	74	00	78	00	74	00	00	00	0
0016B7230	A8	00	00	00	28	00	01	00	00	00	00	00	10	01	00	00	..
0016B7240	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	(

Above screenshot is when 'b.txt' is placed in ReFS drive.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00161B1F0	1E	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00	
00161B200	02	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00161B210	30	04	00	00	10	00	0E	00	0C	00	20	00	10	04	00	00	0
00161B220	30	00	01	00	62	00	2E	00	74	00	78	00	74	00	00	00	0
00161B230	A8	00	00	00	28	00	01	00	00	00	00	00	10	01	00	00	..
00161B240	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	(

Above screenshot is when 'b.txt' is deleted from ReFS drive. It exists in recycle bin.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001527190	01	00	00	00	00	00	00	00	02	07	00	00	00	00	00	00	
0015271A0	00	00	00	00	00	00	00	00	74	00	78	00	74	00	00	00	text
0015271B0	38	04	00	00	10	00	16	00	0C	00	28	00	10	04	00	00	8 (
0015271C0	30	00	01	00	62	00	65	00	61	00	75	00	74	00	2E	00	0 beaut.
0015271D0	74	00	78	00	74	00	00	00	A8	00	00	00	28	00	01	00	text " (
0015271E0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00	
0015271F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00168B240	02	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00168B250	66	00	75	00	6C	00	6C	00	65	00	73	00	74	00	2E	00	fullest.
00168B260	74	00	78	00	74	00	00	00	48	04	00	00	10	00	26	00	text H &
00168B270	0C	00	38	00	10	04	00	00	30	00	01	00	62	00	65	00	8 0 be
00168B280	61	00	75	00	74	00	69	00	66	00	75	00	6C	00	6C	00	autifull
00168B290	65	00	73	00	74	00	2E	00	74	00	78	00	74	00	00	00	est.txt
00168B2A0	A8	00	00	00	28	00	01	00	00	00	00	00	10	01	00	00	" (
00168B2B0	10	01	00	00	02	00	00	00	00	00	00	00	00	00	00	00	

4.8 1GB FILE

Until now files with small sizes have been explored. In this section we look at how resilient file system deals with large files and what is its format of storing large files.

NTFS master file table stores data of small files and directories typically 512 bytes or smaller, within the master file table record and labels them as resident data. Files larger than this size are non-resident and possess data run(s) in them. In this scenario we check whether ReFS has data runs or not or resident and non-resident attributes.

An mp4 file 'zootopia.mp4' of size 1.65GB is placed in the drive. The metadata as seen through WinHex is as follows:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
001687380	00	00	00	00	00	00	00	00	0C	00	18	00	7A	00	6F	00	z o
001687390	6F	00	74	00	6F	00	70	00	69	00	61	00	2E	00	6D	00	o t o p i a . m
0016873A0	70	00	34	00	00	E0	FF	FF	40	04	00	00	10	00	1C	00	p 4 àÿÿè
0016873B0	08	00	30	00	10	04	00	00	30	00	01	00	7A	00	6F	00	o z o
0016873C0	6F	00	74	00	6F	00	70	00	69	00	61	00	2E	00	6D	00	o t o p i a . m
0016873D0	70	00	34	00	00	00	00	00	A8	00	00	00	28	00	01	00	p 4 " (
0016873E0	00	00	00	00	10	01	00	00	10	01	00	00	02	00	00	00	
0016873F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001687400	53	23	CB	FF	45	E2	D1	01	53	23	CB	FF	45	E2	D1	01	S#ËyEaÑ S#ËyEaÑ
001687410	53	23	CB	FF	45	E2	D1	01	53	23	CB	FF	45	E2	D1	01	S#ËyEaÑ S#ËyEaÑ
001687420	21	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00	!
001687430	20	00	00	00	00	00	00	00	88	1F	59	2F	01	00	00	00	Y/
001687440	D7	53	CA	69	00	00	00	00	00	00	CB	69	00	00	00	00	*SËi Èi
001687450	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001687460	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

The file 'zootopia.mp4' is found in the same format as other files (small size files) were found. Filename and its duplicate are found at offset 0x168738C. MACE times are highlighted in orange whereas file permissions are highlighted in green. 'D7 53 CA 69' is actual file size in Little Endian. We can calculate actual file size in bytes by converting 'D7 53 CA 69' to Big Endian '69 CA 53 D7' and finally converting it in decimal, we get '1774867415' bytes. '1774867415' bytes are equal to 1.65GB.

Further searching for the file content pointer, we also find file size on disk. Highlighted in yellow '2C A7 01' is the file size on disk. Calculating file size on disk is different than actual file size because it is multiplied by 16,384. 16,384 is multiplied because all the metadata is stored in blocks of 16KB.

In the metadata block of the file, file size on disk is 2C A7 01 in Little Endian.

- Converting 2C A7 01 to Big Endian, we get 01 A7 2C
- 1A72C hexadecimal is equal to 108332 in decimal
- Multiplying 108332 with 16,384 (because all the metadata is stored in blocks of 16KB), we get 1,774,911,488.

1,774,911,488 bytes is the file size on disk. This also makes 1.65GB.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0016875C0	00	00	00	08	00	00	00	00	30	00	00	00	10	00	10	00	0
0016875D0	00	00	10	00	20	00	00	00	00	00	00	00	00	00	00	00	
0016875E0	2C	A7	01	00	00	00	00	00	00	D0	03	00	00	00	00	00	,S D
0016875F0	00	00	00	08	00	00	00	00	00	00	00	00	00	00	00	00	
001687600	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
001687610	00	00	00	00	20	00	00	00	20	00	00	00	80	00	00	00	€

In the metadata block above, file content pointer is 00 D0 03 in Little Endian.

- Converting 00 D0 03 to Big Endian, we get 03 D0 00
- 3D000 hexadecimal is equal to 249856 in decimal
- Multiplying 249856 with 16,384 (because all the metadata is stored in blocks of 16KB), we get 4093640704

- Converting 4093640704 decimal to hexadecimal, we get F4000000
- F4000000 is the offset of the file content.

Now looking at the file content area at offset 0xF4000000 in the editor:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0F3FFFFFF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0F3FFFFFF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0F40000000	00	00	00	20	66	74	79	70	69	73	6F	6D	00	00	02	00	ftypisom
0F40000010	69	73	6F	6D	69	73	6F	32	61	76	63	31	6D	70	34	31	isomiso2avcImp41
0F40000020	00	56	BF	6E	6D	6F	6F	76	00	00	00	6C	6D	76	68	64	V_inmoov lmvhd
0F40000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	03	E8
0F40000040	00	63	5F	55	00	01	00	00	01	00	00	00	00	00	00	00	c_U
0F40000050	00	00	00	00	00	01	00	00	00	00	00	00	00	00	00	00	
0F40000060	00	00	00	00	00	01	00	00	00	00	00	00	00	00	00	00	
0F40000070	00	00	00	00	40	00	00	00	00	00	00	00	00	00	00	00	§
0F40000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0F40000090	00	00	00	03	00	25	9A	14	74	72	61	6B	00	00	00	5C	§ trak \
0F400000A0	74	6B	68	64	00	00	00	03	00	00	00	00	00	00	00	00	tkhd
0F400000B0	00	00	00	01	00	00	00	00	00	63	5F	08	00	00	00	00	c_
0F400000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00	
0F400000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00
0F400000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	§
0F400000F0	07	80	00	00	03	20	00	00	00	00	00	24	65	64	74	73	€ sedts
0F40001000	00	00	00	1C	65	6C	73	74	00	00	00	00	00	00	00	01	elst
0F40001010	00	63	5F	08	00	00	04	8E	00	01	00	00	00	25	99	8C	c_ Z §WE
0F40001020	6D	64	69	61	00	00	00	20	6D	64	68	64	00	00	00	00	mdia mdhd
0F40001030	00	00	00	00	00	00	00	00	00	36	9A	05	6D	02	BB	00	6§ # §
0F40001040	55	C4	00	00	00	00	2D	68	64	6C	72	00	00	00	00	00	UÄ -hdr
0F40001050	00	00	00	00	76	69	64	65	00	00	00	00	00	00	00	00	vide
0F40001060	00	00	00	56	69	64	65	6F	48	61	6E	64	6C	65	72	00	VideoHandler
0F40001070	00	00	25	99	37	6D	69	6E	66	00	00	00	14	76	6D	68	§?minf vmh
0F40001080	64	00	00	00	01	00	00	00	00	00	00	00	00	00	00	00	d
0F40001090	24	64	69	6E	66	00	00	00	1C	64	72	65	66	00	00	00	§dinf dref
0F400010A0	00	00	00	00	01	00	00	00	0C	75	72	6C	20	00	00	00	url
0F400010B0	01	00	25	98	F7	73	74	62	6C	00	00	00	97	73	74	73	§+stbl -sts
0F400010C0	64	00	00	00	00	00	00	00	01	00	00	00	87	61	76	63	d +avc
0F400010D0	31	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00	l
0F400010E0	00	00	00	00	00	00	00	00	00	07	80	03	20	00	48	00	€ H
0F400010F0	00	00	48	00	00	00	00	00	00	01	00	00	00	00	00	00	H

The first 12 bytes at 0xF4000000 show ‘...ftypisom’ which is file signature and describes that it is an ISO Base Media file (MPEG-4) v1. ISO base media file defines a general structure for time-based multimedia files such as video and audio. [21]

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
15DCA5260	03	00	00	03	00	00	03	00	00	03	00	00	03	00	00	03	
15DCA5270	00	00	03	00	00	03	00	00	03	00	00	03	02	0E	21	11	!
15DCA5280	45	00	14	50	01	47	21	11	45	00	14	50	01	47	00	00	E P G! E P G
15DCA5290	00	32	01	9F	F6	74	43	F6	FF	00	00	03	00	00	03	00	2 Yörcöy
15DCA52A0	00	03	00	00	03	00	00	03	00	00	03	00	00	03	00	00	
15DCA52B0	03	00	00	03	00	00	03	00	00	03	00	00	03	00	00	03	
15DCA52C0	00	00	04	9C	21	11	45	00	14	50	01	47	21	11	45	00	æ! E P G! E
15DCA52D0	14	50	01	47	00	00	00	32	01	9F	F8	6A	43	F6	FF	00	P G 2 YöjCöy
15DCA52E0	00	03	00	00	03	00	00	03	00	00	03	00	00	03	00	00	
15DCA52F0	03	00	00	03	00	00	03	00	00	03	00	00	03	00	00	03	
15DCA5300	00	00	03	00	00	03	00	00	04	9D	21	11	45	00	14	50	! E P
15DCA5310	01	47	21	11	45	00	14	50	01	47	00	00	00	3B	41	9B	G! E P G ;A>
15DCA5320	FB	35	08	2D	93	29	82	89	87	ED	FF	FE	A9	96	00	00	û5 -"),*#iyp@-
15DCA5330	03	00	00	03	00	00	03	00	00	03	00	00	03	00	00	03	
15DCA5340	00	00	03	00	00	03	00	00	03	00	00	03	00	00	03	00	
15DCA5350	00	03	00	00	03	00	00	0D	08	21	11	45	00	14	50	01	! E P
15DCA5360	47	21	11	45	00	14	50	01	47	00	00	00	32	01	9E	1A	G! E P G 2 ž
15DCA5370	6A	43	F6	FF	00	00	03	00	00	03	00	00	03	00	00	03	jCöy
15DCA5380	00	00	03	00	00	03	00	00	03	00	00	03	00	00	03	00	
15DCA5390	00	03	00	00	03	00	00	03	00	00	03	00	00	04	9C	21	æ!
15DCA53A0	11	45	00	14	50	01	47	21	11	45	00	14	50	01	47	21	E P G! E P G!
15DCA53B0	11	45	00	14	50	01	47	21	11	45	00	14	50	01	47	21	E P G! E P G!
15DCA53C0	11	45	00	14	50	01	47	21	11	45	00	14	50	01	47	21	E P G! E P G!
15DCA53D0	11	45	00	14	50	01	47	00	00	00	00	00	00	00	00	00	E P G
15DCA53E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
15DCA53F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
15DCA5400	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

While looking at file metadata, we observe that there is no data run or resident and non-resident attribute type notion in ReFS. All file data is located at file content pointer location. The end of 'zootopia.mp4' is found at 0x15DCA5398. Subtracting 15DCA53D7 from F4000000, we get 69CA53D7 which when converted to decimal becomes 1,774,867,415 bytes. This is equal to 1.65GB. From here it is proved that in resilient file system all file content is located at one location and it is not scattered. This avoids overhead which is created while looking for file data in NTFS when data is non-resident.

4.9 Comparison of ReFS artifacts with NTFS artifacts

The following table provides a comprehensive comparison of the artifacts of resilient file system that have been observed in this research with those of the artifacts of new technology file system that have been discussed time and again.

Sr. No.	ReFS Artifacts	NTFS Artifacts
1	NO MFT. ReFS has its own mechanism of storing file metadata. Master file table like structure is not present in refs.	Master file table is the chief structure that stores file metadata
2	All file attributes not present in ReFS that are present in NTFS. Five attributes have been observed during this research.	A set of seventeen well defined and well researched attributes exist in NTFS.
3	ReFS is not bootable.	NTFS is bootable
4	Log structured file system was rejected in ReFS case. ReFS uses an allocate on write strategy that never updates metadata in-place but writes it to a new location.	Log structure is implemented that updates metadata in-place and keeps a journal or log of transactions that notes every change that takes place in the metadata. Main disadvantage of journaling is that writes can get randomized and torn write can occur.
5	Check disk is not present in ReFS because repair (if needed) occurs on-the-fly.	Check disk is used to fix disk corruptions.
6	Designed to handle very large volumes upto 1 yottabyte and accommodate cloud storage	Not specifically designed for cloud storage but is or can be used for it
7	File system recognition structure is present whose main goal is to identify unrecognized file system	No such structure is present
8	No data run or non- resident data in ReFS. Large file's content is present at the same location. This avoids overhead which occurs in non-resident case.	Large files (more than 512 bytes) are non-resident and contain data runs in them that point to clusters where the remaining file data is present
9	Simple deletion trims the filename. File is fully recoverable	Simple deletion updates the flag in metadata of file. File is fully recoverable

Conclusion And Future Work

5.1 Overview

Forensic analysis has been performed in this thesis which is a branch of digital forensics that examines digital information found in computers and digital storage media. The aim of forensic analysis is to identify, preserve, recover, analyze and present facts and opinions about the digital information in a forensically sound manner without destroying evidence.

Present day involves a variety of computer crimes which are investigated by taking into consideration forensic analysis of that specific system.

5.2 Overview of Research

This research is about forensic analysis of a file system that is relatively new in the file system arena. Forensic facts related to it have not been discovered yet therefore there is a need to conduct analysis on it so that it's working and structure is identified. This new file system called Resilient file system abbreviated as ReFS has been built on NTFS. In other words it uses NTFS as a base. This gives us some help during its analysis.

5.3 Findings

The research carried out on resilient file system in Windows Server 2012 R2 gives us a basic view of the file system's underlying features. Moreover the scenarios that have been taken into consideration during the research give us the working of resilient file system when different operations are performed on file. Deleted files can be easily recovered, even when they are permanently deleted in resilient file system as they exist at hexadecimal level.

5.4 Future Work

This thesis focuses on the underlying structure and working of resilient file system. Behavior of applications on resilient file system has not been analyzed in it. Applications such as the working of antivirus should be explored in resilient file system and its results should be compared with NTFS.

5.5 Conclusion

Forensic examiners have to be aware of every possible area where crucial information can be hidden in order to recover and investigate digital evidence to be used in court. The scope of a forensic analysis can vary from simple information retrieval to reconstructing a series of events. And for all these investigation and further examinations, forensic investigators should know the working and structure of the file system in the computer or digital media that is under consideration.

Bibliography

- [1] Steven Sinofsky, “Building the next generation file system for Windows: ReFS”, 17 January 2012, <http://blogs.msdn.com/b/b8/archive/2012/01/16/building-the-next-generation-file-system-for-windows-refs.aspx>
- [2] “Resilient file system” <http://msdn.microsoft.com/en-us/library/windows/desktop/dn323741%28v=vs.85%29.aspx>
- [3] “Resilient File System Overview”, 1 November 2013, <http://technet.microsoft.com/en-us/library/hh831724.aspx>
- [4] http://en.wikipedia.org/wiki/ZFS#cite_note-endtoend-18
- [5] Bonwick, “ZFS End-to-End Data Integrity”, 08 Dec 2005, https://blogs.oracle.com/bonwick/entry/zfs_end_to_end_data
- [6] Martin Lucas, “Windows Server 2012: Does ReFS replace NTFS? When should I use it?”, 1 Jan 2013, <http://blogs.technet.com/b/askpfeplat/archive/2013/01/02/windows-server-2012-does-refs-replace-ntfs-when-should-i-use-it.aspx>
- [7] Ken Mizota, “Windows Resilient File System Forensics”, 22 August 2013, <http://encase-forensic-blog.guidancesoftware.com/2013/08/windows-resilient-file-system-forensics.html>
- [8] Liu Naiqi, Wang Zhongshan, HaoYujie, QinKe, “Computer Forensics Research and Implementation Based on NTFS File System”, ISECS International Colloquium on Computing, Communication, Control, and Management, 2008

[9] Jeremy Davis, Joe MacLean, David Dampier, “Methods of Information Hiding and Detection in File Systems”, Fifth International Workshop on Systematic Approaches to Digital Forensic Engineering, pp. 1–4, 2010

[10] Brian Carrier, “Digital investigation foundations”, in *Filesystem forensic analysis*, One Lake Street, Upper Saddle River, NJ, 2005, Chapter 1, Part I: Foundations, pp. 19-20

[11] Byeongyeong Yoo, Jungheum Park, Jewan Bang, Sangjin Lee, “A Study on a Carving Method for Deleted NTFS Compressed Files”, IEEE, 2010

[12] <http://thestarman.pcministry.com/asm/mbr/NTFSbrHexEd.htm>

[13] <https://www.google.com/patents/US8200895>

[14] <https://msdn.microsoft.com/en-gb/library/windows/desktop/dd442652%28v=vs.85%29.aspx>

[15] <https://www.fireeye.com/blog/threat-research/2012/09/striking-gold-incident-response-ntfs-indx-buffers-part-1.html>

[16] “Security Identifier”, https://en.wikipedia.org/wiki/Security_Identifier

[17] <https://technet.microsoft.com/en-us/library/hh831724.aspx>

[18] <https://blogs.msdn.microsoft.com/b8/2012/01/16/building-the-next-generation-file-system-for-windows-refs/>

[19] Ray Zadjmool, “Hidden Threat: Alternate Data Streams”, 24 March 2004,

http://www.windowsecurity.com/articles-tutorials/windows_os_security/Alternate_Data_Streams.html

[20] <http://www.osforensics.com/faqs-and-tutorials/how-to-scan-ntfs-i30-entries-deleted-files.html>

[21] https://en.wikipedia.org/wiki/ISO_base_media_file_format