CONTEXT-BASED CARVING OF FRAGMENTED WORD DOCUMENTS FROM VOLATILE MEMORY USING MACHINE LEARNING TECHNIQUE

By

Noor Ul Ain Ali

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

July 2019

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

This thesis is dedicated to

MY MOTHER IN LAW,HUSBAND AND CHILDREN; ABDULLAH and IBRAHIM

for their love, support and great endurance. Also my MOTHER for always motivating and hearing me out

ACKNOWLEDGEMENTS

All praises be for you ALLAH; al-Hadi, al-Fattah, and al-Muizz

As my supervisor, I am grateful to AP Waseem Iqbal for he has guided me at every key moment and also gave me enough confidence to work independently majority of the time.Also for keeping up with the amount of breaks I took during the course of my work.

I'm thankful to my committee members; Dr Rabia Latif, PhD and Lec Narmeen for their support and guidance.

I'm also obliged to HoD and all faculty members of Information Security department, who have influenced and enhanced my research. Especially Lec Narmeen for she went above and beyond to accommodate me.

Finally, I would take this opportunity to show my deep gratitude to my parents, siblings, and friends, and coursemates who see good in me and always push me to be the best version of myself.

Noor Ul Ain Ali

SUPERVISOR CERTIFICATE

This is to certify that **Noor Ul Ain Ali** Student of **MSIS-15** Course Reg.No: **00000171974** has completed her MS Thesis title **"Context-based Carving of Fragmented Word Documents from Volatile Memory using Machine Learning Technique"** under my supervision. I have reviewed her final thesis copy and I am satisfied with her work.

Thesis Supervisor

(Asst Prof. Mian Muhammad Waseem Iqbal)

Dated: ______2019

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS Thesis written by **Noor Ul Ain Ali** Registration No. **00000171974**, of **Military College of Signals** has been vetted by undersigned, found complete in all respect as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial, fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have been also incorporated in the said thesis.

Signature: ______________________________

Name of Supervisor: ____________________

Date: ___________________________________

Signature (HOD): _______________________

Date: ______________________________

Signature (Dean/Principal): _______________

Date: ______________________________

ABSTRACT

With the rise in digital crimes nowadays, digital investigators are required to recover and analyse data from various digital resources. Since the files are often stored in fragments owing to memory constraints, the information related to file system and metadata of the file is required to recover a file. However, in many cases when the file system is destroyed intentionally or unintentionally, and the metadata is deleted as well, the recovery of the digital evidence is done by a special method known as carving. In file carving, files are recovered solely based on the information about the structure and content of the individual file rather than matching the system's information of the file

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

DEFINITION	**ACRONYM**
Application Program Interface	API
Random Access Memory	RAM
Bag Of Words Model	BOWM
Inverse Document frequency	IDF
Sector Identifier	SecID
Sector Allocation Table	SAT
Master Allocation Table	MAT
Microsoft Compound Document File Format	MCDFF
Random Access Memory	RAM
True positives	TP
True Negative	TN
False Positive	FP
False negative	FN
Term Frequency	TF

Chapter 1

INTRODUCTION

The main task of digital forensics investigator is to extract data including documents, folders, files, browsing history etc.Such artefacts are very helpful when determining the evidences in case of a digital crime.It is possible to recover the data using metadata and Application Program Interface (API) of a file system in a storage device. But, in cases where the API is damaged the technique of file craving is used for the extraction of data. However, in some cases the API is damaged and extraction of data is done using the technique of file carving [1]. The evidences are mainly extracted from storage media like hard disks and solid state drives. However, the interest of forensic investigators has recently shifted towards extraction of data from volatile memory i.e. Random Access Memory (RAM). The importance of RAM forensics can be determined by the fact that all important operations such as writing to a specific file or reading a file are all saved in RAM for a temporary time limit [2]. Due to the scattered inhomogeneous nature of RAM and the absence of metadata, carving of RAM is very useful for artefact extraction. During an investigation when a dump of physical memory i.e. RAM is taken, carving and extraction process on that memory dump can lead to important inferences about the investigation.

If the file is stored on adjacent blocks of memory it is comparatively easier to crave a file from memory. Whereas in reality, a file is saved as multiple fragments scattered across the memory because of the file structure being used and many other reasons [2]. This makes the carving of fragmented files difficult and requires something more than just matching the signatures of files to yield better results.Hence the conventional craving tools are not successful at recovering fragmented files from memory.

A universal approach regarding carving of fragmented files from memory is to perform extraction in two steps. Firstly, the fragments are grouped together in classes of similar type of files. For this purpose of fragment classification, three types of approaches are generally applied:

- Content based analysis approach: In this approach each block of data is first classified as a specific file type based on the content of the file. Then these blocks of similar contents are further grouped together to form a file.
- Distance-based approach: It performs the classification based on the differences between the values of adjacent blocks in memory and the frequency of the byte values. A set of files is used as a model frequency. When the distance between the model frequency and a non-classified block is below a threshold, then this block is assigned this particular type of file.
- Machine learning approach: In this approach statistical values are used as input to a classification algorithm which is used for classifying already formed machine learning algorithms.
- Binary Classifiers: In case of binary classifiers, files are classified based on one to one functions that distinguish one file from another

All possible combinations are checked by exhaustive search from each class for the reassembly of fragmented files in correct order. Image markers could be used in some cases as extra information of the file structure [3].
Secondly, another dimension of File carving is its usefulness for data recovery in times of accidental loss. MS Word format is the most widely used format of MS Office and is required for most of the digital documentation done using computers including text files in form of documents and sending emails [4]. The recovery of Microsoft Word files is an essential need for every user these days as loss of important Word documents can be very devastating for the owner.

1.1 PROBLEM STATEMENT AND OBJECTIVES

Microsoft File format is one of the format which is not much studied and researched because it is a compound format. Research has been conducted in either the extraction of images or text from documents but there has been no detailed research for the extraction of both components simultaneously.The extraction process becomes more complex in case of fragmented word documents and recovery of image and text is almost impossible.
Objectives of this thesis are:

- Literature review of carving techniques from volatile memory
- Proposing a technique using machine learning algorithms for context based carving of fragmented word documents from volatile memory

1.2 MOTIVATION

This research will be helpful for industry, academia researchers from all fields of life. This research will prove a way forward for forensics examiners to strengthen their tools and techniques for yielding best results while examining a crime scene involving volatile memory based word document usage.

1.3 CONTRIBUTIONS

Following are the main contributions of this research work:

- In depth literature review of carving techniques
- Ready reckoner document for researchers, academicians and forensics analysts to carve word base documents from volatile memory
- In this work, the extraction of Microsoft Word Document from RAM is done using file carving
- The work basically encompasses both Microsoft word format i.e DOC and DOCX which have before never been catered together.

1.4 THESIS STRUCTURE

Including the current Introduction chapter, this research work is composed of six chapters. Outline of the remaining chapters is as follow:

- Chapter 2: It contains extensive literature review done on carving of all type of formats available.
- Chapter 3: It contains all the background knowledge, including in detail analysis of DOC and DOCX file formats.
- Chapter 4: It contains the Methodology used for carving of both the formats.Also the experiments designed for carving.

- Chapter 5: It contains the experimental results

- Chapter 6: It contains Conclusion and Future Work.

To facilitate our readers we have included an extensive bibliography at the end.

Chapter 2

TECHNIQUES OF FILE CARVING AND TYPES OF FRAGMENTATION

This chapter extensively addresses the technique of file carving.In this section we briefly describe few terms used in this research.In the next section 2.2,the general idea behind fragmentation and why fragmentation happens. Then 2.3 explains clustering and two basic types of clustering.In section 2.4 we have the process of feature extraction followed by detail discussion about Microsoft Compound File Format2.5 and Open Office XML2.5.3

2.1 File Carving

Carving is recovery of the data which is lost without the information about the related file structure or metadata of the file. In digital forensics, the technique of file carving depends totally upon the content of file structure rather than on similarities of metadata of the file system. The unallocated space in the drive is basically analyzed for file carving. Unallocated space in a drive means the portion of the drive that is not holding any piece of information about the file or the file structure information like file allocation tables etc. In file carving the complete drive is considered if the drive is damaged or the file system is corrupt or missing. It is believed that under the limitations of constraint environment symmetric-key primitives are better choice to gain security, but their design and implementation should be efficient enough to comply with the scarce resources.

2.1.1 Difference between File Recovery and File Carving

There is a very huge difference that exists between recovery and carving of a file. When a file is deleted from memory, its file system information still remains on the disk and file recovery basically uses this file system information for extraction of data. By utilizing this information a great deal of files can be extracted. In order for file recovery to work efficiently and correctly the system information must be accurate. If the system information is incorrect or corrupted than the files are impossible to recover or if the file system is somehow formatted, even then the recovery technique will fail [1]

On the other hand file carving techniques makes use of raw data in the storage media and has nothing to do with the file system's structure or information. A file system basically deals with the arrangement and organization of files and the data a system may contain. Although carving is not at all affected by the types of file system used by the user of files but knowing the file system type could significantly help in the carving of data.

2.2 Fragmentation

Most of the Operating Systems don't use fragmentation because it makes the writing and reading process slower except the following situations in which the OS is bound to do so.

- If the disk has not enough space to write the file without been fragmented.This basically happens in drives that are used for a long period of time and it has been used up to its maximum capacity and has many files deleted and added randomly over a long period of time [5] .

- If a file already exists on a disk and new data is added to it, and the disk has some missed non allocated sectors at the end and no sectors are available for appending new data.In this type of scenario some file systems rearrange the original file but in most of the cases it will write the data to some other distant memory location. [6]

- If writing of a specific type of file in adjacent blocks of memory is not supported by the file system. This mostly happens in Unix based file systems.

Simon Garfinkel [2] studied the fragmentation statistics in which he investigated 350 disks that included NTFS, FAT, and UFS. From his study, he proved that the fragmentation rate for user files like emails,Microsoft Excel, JPEG and Microsoft Word is considerably higher than other files. According to the study , the fragmentation rate for Microsoft Word is *17* percent.

2.3 Clustering

Clustering is an unsupervised machine learning problem.clustering is like finding a pattern or groups of unlabbeled data.We can say that clustering is arranging similar members together in a group.So objects within a cluster are similar to each other but they are different from other clusters. There are maily two types of clustering:

- Partitioning clustering: Cluster numbers are specified in this type of clustering. It is implemented by K mean

- Hierarchical clustering: Cluster number is not specified in this type of clustering. It is Implemented by Hierarchical clustering

2.3.1 K-Mean Clustering Algorithm

For clustering larger data sets k-means clustering algorithm is deemed more efficient. MacQueen proposed this clustering algorithm , and is simply the best algorithm available. The K-Means algorithm differentiates objects based on unique features or attributes, divides it into k clusters, where k is constant. It defines one centroid per cluster ,the idea is to define k centroids, one per cluster. [7].The centroid is the starting point for partitioning every cluster.

Method

If k mean is performed in the following way :

- first we will partition the objects into k partitions, the partitions should have a non zero value.

- For each partition we created, we will define a centroid.

- Then we will assign objects to the clusters.

- Then using the distance formulae we will calculate the distance of each object to every centroid and assign the object to the cluster whose object-centroid distance is the closest.

- Now we re allot the clusters and again calculate centroid for each newly assigned cluster. [8]

2.3.2 Hierarchical Clustering Algorithm

Hierarchical Clustering Algorithm is a type of agglomerative algorithm that means it can have many variations depending upon the formulae used for calculating distance between clusters. The most common formulae for calculating distances of individual points is the Euclidean distance. There is no strict criteria or rule regarding which distance formulae be used, and depends mostly on dataset. Following are the types of hierarical clustering based on different distance formulaes: [9]:

Average Linkage Clustering

Average values are used in this case for calculating the dissimilarity of clusters.For calculating the average distance we calculate the distance between every object in observed cluster and all other objects of another cluster.The clusters with the smallest distance are joined together to form a new cluster.

Centroid Linkage Clustering

It uses the centroid ,which is the centre of all the objects, as the average value .

Ward's Method

Clusters are assigned by calculating the sum of squared difference from the centre of a cluster. The clusters are joined that produce the smallest squared sum
In order to perform both the K mean algorithm and the hierarchal algorithm we need to perform some preprocessing on our data. This preprocessing is basically making our textual data ready for input to the machine learning algorithm.

2.4 Feature Extraction

Computers can't really understand English and machine learning cannot deal with raw text. For machine learning we need to have well defined inputs and outputs. So we convert each word into a numeric value. Each of this numeric value is stored in a vector. This is known as feature extraction. [10]
It is a basic step for any machine learning technique. In order to reduce our dataset, we convert each word into a dense vector. A dense vector consists of 300 real values. In this way every document in converted into a list of vectors or numbers. Each of these lists have a special sequence of text and that sequence is known as a token. These token can either be single unique words or a sum of words.

2.4.1 Bag Of Words Model

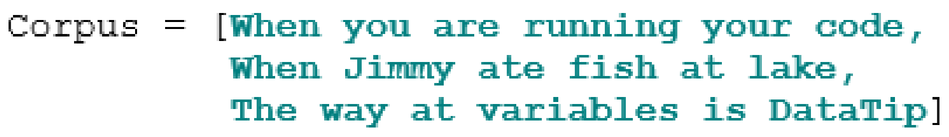
Bag of words *(BOWs)* approach is a way of feature extraction from text which are then used for machine learning. This approach is the most common and the feature extraction can be done in a number of ways. Bag of words is basically a collection of the occurrence of words in a document. The sequence of the document is not important. Only the frequency of the content is what interests us hence the name bag of words in used for this. The features are basically the word counts.The idea is that the documents having same frequency of words are similar. [11].

Steps for Bag of Words

Following steps are performed for feature extraction from bag of words model:

COLLECTION OF DATA

The first step is to collect our data. The data could be anything from random documents to emails to anything that contains text. For our Example we consider the following document corpus, and each line in the corpus refers to a different document.

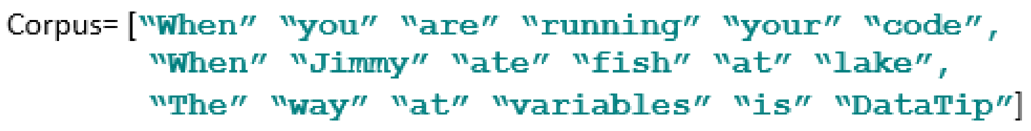


```
Corpus = [When you are running your code,
          When Jimmy ate fish at lake,
          The way at variables is DataTip]
```

Figure 2.1: The corpus used for the example

TOKENIZING

Before we start making our vocabulary one important thing is to tokenize our document corpus. Tokenizing is basically to split the document into individual words or letters. This is basically done so that each word appears as a separate component in our bag of words. Our example corpus would look like something like this if we tokenize it.



```
Corpus= ["When" "you" "are" "running" "your" "code",
         "When" "Jimmy" "ate" "fish" "at" "lake",
         "The" "way" "at" "variables" "is" "DataTip"]
```

Figure 2.2: Tokenizing the corpus

MAKING OF VOCABULARY

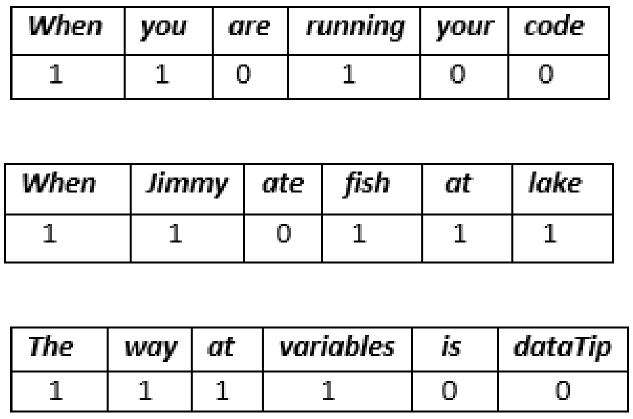
Designing the vocabulary comes next. The vocabulary is designed ignoring the punctuations and case sensitivity of words. Vocabulary can contain unique words from the documents or any word from the documents. Choosing the vocabulary is the most crucial step in bag of words and it directly impacts the result.

The vocabulary is selected as follows:

- *"When"*
- *"you"*
- *"Running"*
- *"Jimmy"*
- *"Lake"*
- *"Fish"*
- *"Way"*
- *"The"*
- *"Variables"*
- *"At"*

CREATION OF VECTORS

The fourth step is to create the vectors for the documents. One way is to simply use the Boolean representation I.e. *"0"* for absence of that word in vocabulary and *"1"* if that word is present in vocabulary. The binary vectors for the above documents are as follows: [12]



When	*you*	*are*	*running*	*your*	*code*
1	1	0	1	0	0

When	*Jimmy*	*ate*	*fish*	*at*	*lake*
1	1	0	1	1	1

The	*way*	*at*	*variables*	*is*	*dataTip*
1	1	1	1	0	0

Figure 2.3: Vectorizing each document

In this way the occurrences of the known words are recorded. Now this is a very small example and in real life the documents are large in number and hence the vectors will also be very large because the length of the vector is equal to the length of the vocabulary defined. Like for some books in a library the number of words in our defined vocabulary can be in thousand and millions and the known words from one book will be very less. So there would be a lot of zeros in a vector. This type of a vector is known as sparse vector. As it is evident that dealing with sparse vectors would lead to the use of large number of resources. So we will reduce the size of vocabulary.

REDUCTION OF VOCABULARY

For reducing the size of the vocabulary we can simply ignore words that are very common to appear in a text, for English language words like the articles the, are etc. These words don't have a significant meaning so we can ignore them. Also we can ignore the words that have spelling mistakes. A more sophisticated method for reducing vocabulary is that we make groups of words and store a group as single entry in the vocabulary. In this approach each group of text is called *"gram"*. So if we have two words in a group we would call it a bigram, a group of three words is called a trigram. Similarly if we have n number of groups then we will call this approach as n gram. [13]
Keep in view the above technique the bigrams for the above lines are as follows:
The use of bigrams and trigrams in BOW is a better approach because it captures the essence of the document better and also reduces the computation.

- *"When you"*
- *"When Jimmy"*
- *"The way"*

The use of bigrams and trigrams in BOW is a better approach because it captures the essence of the document better and also reduces the computation.

SCORING OF THE VOCABULARY

Once our vocabulary is decided and reduced the next thing is to score the occurrence of the vocabulary words in the example document. One simple way is the Boolean score i.e. *"1"* and *"0"*. Other scoring methods can be to count the number of times a word appears in a document or to calculate the frequency of the word in the document as compared to all the other words of the document.

TF-IDF

When we are scoring the words according to their frequencies,then the words dominating the scoring would be the ones that are least significant in their meaning like*"the"* but their occurrence frequency would be very high.So in order to neutralize this we use the term frequency-inverse document frequency approach.Term frequency is the frequency by which a word appears in a document.Inverse Document frequency (IDF) is to find that how rare is the word in all the documents under observation.The IDF of a known word is low whereas the IDF of a rare or unique word is high. [14]

2.5 Compound File Format

Compound file format as described by Microsoft is a binary file format and it contains a number of streams that are virtual in nature. These virtual streams consist of both control streams and user data streams [15]. The smallest portion of memory space in a compound document is a sector and it is *512* bytes in size. it is just serial arrangements of the sectors. The start of these sectors is called sector identifier and they start with index number zero.
These arrays of sectors of virtual streams have a special sector in start known as the header and its size is *512* bytes.It is a special purpose sector and has some very important information that is stored in the header which is actually crucial for the proper working of a compound file i.e. the signature for a compound file, size of the virtual streams, etc. The rest of the sectors are used for other purposes and are of different types. [16]

2.5.1 A Brief History of the DOC Format

Over thirty years ago Microsoft first used the Doc format MS-DOS.it was a strictly proprietary application used by only the Microsoft for making word documents until it opened the specifications for general public is 2006, It was reverse engineered and finally a number of similar applications were built. [16]
In the early 2000s there were a number of softwares that were compatible with Microsoft's office doc format, but the full editing functionality was missing in those competitive versions of document format.Since Office and specially the word has been the most common

feature of the Microsoft's office suit, it has maintained its dominance over the competitors for years.Since 2008 Microsoft has released a number of doc formats and the versions are revised each year.

2.5.2 FRAMEWORK FOR CARVING OF COMPOUND MICROSOFT WORD DOCUMENT

The structure of a compound file format is almost similar to a file system in real life. There exist a number of data streams which can be considered as files that are hieratically organized in storages, like the directories of a file system. There is always a root storage, it further has storages and stream, which are direct and indirect members of the root storage. Two direct members of the storage cannot have a same name where as two indirect storage or streams can have same names. The hieratical structure is shown below.

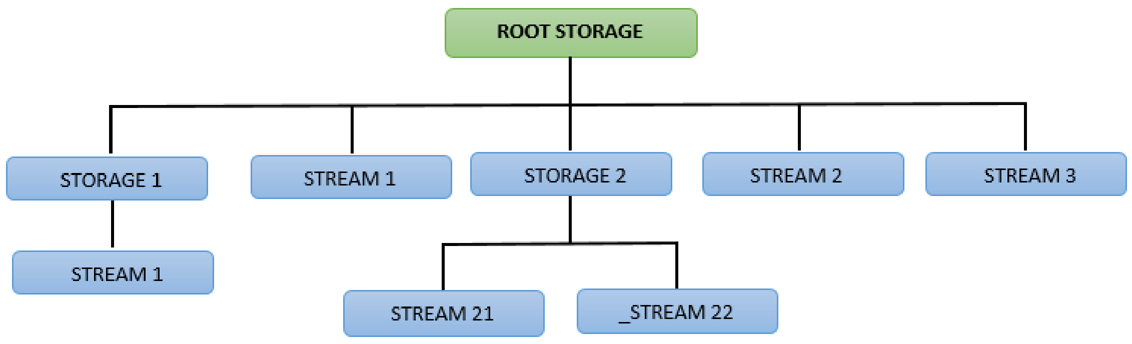


Figure 2.4: Internal Structure of a Compound File Format

SECTORS AND SECTOR IDENTIFIER

All the streams of a compound file format are divided into small chunks of data known as the sector. The file consists of one header that is the beginning of the file, followed by the number of sectors in that file. the header includes the size of every sector and then remain constant throughout the whole file.

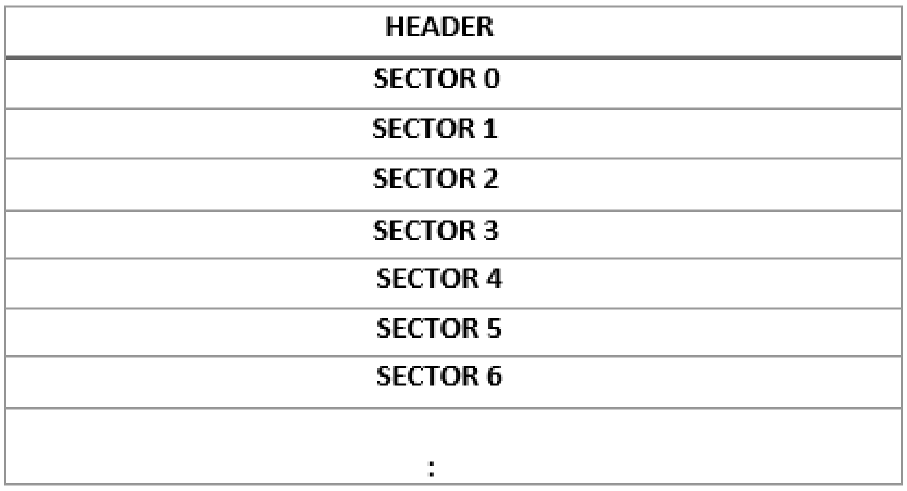


Figure 2.5: Sectors of a Compound File

The sectors are numbered according to their occurrence in the file. The sector with the index zero is called the sector ID $(SecID)$. If the secID has a non-negative value then it means that this sector belongs to some preexisting file.If secID has a negative value than it has a specific meaning. Following is a list of the valid possible values of secID's:

- If a secID value is*"-1"* it means the sector is a free sector.
- If a secID value is*"-2"* it means that the sector is an ending sector also known as a trailing sector.
- If a secID value is *"-3"* it means that the sector contains sector Allocation Table.
- If a secID value is *"-4"* it means that the sector contains Master Allocation table.

SECTOR CHAINS AND SECID CHAINS

The data stream is divided into a number of sectors and the list of these sectors is known as the sector Chain. The sectors can be placed anywhere in the file, without any sequence and can be at totally random places. The sector chain which is an array of the secID specifies the order of all the sectors of a stream. The stream also ends with the secID -2 indicating the end of file. The SecID chain for very stream us build using the sector Allocation table except for the short sector Allocation table, the SecID's of Master sector Allocation table because it uses itself for making it and the SAT table because it is built from the master Allocation Table.

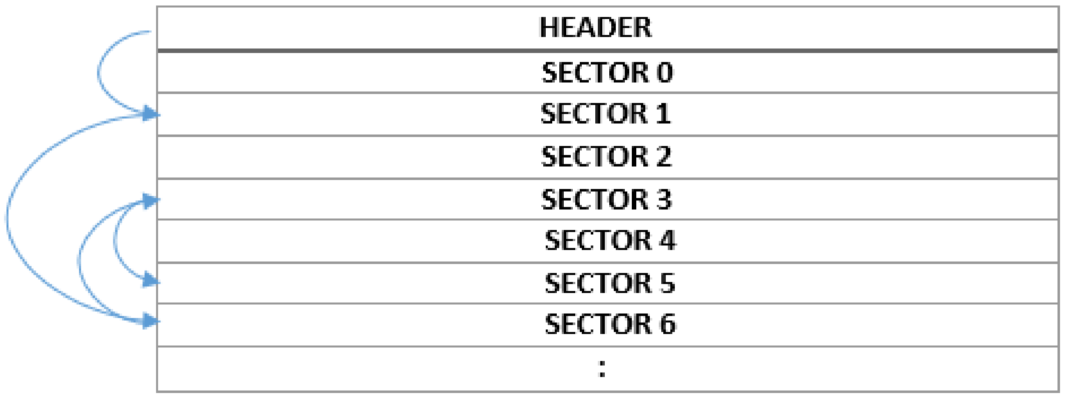
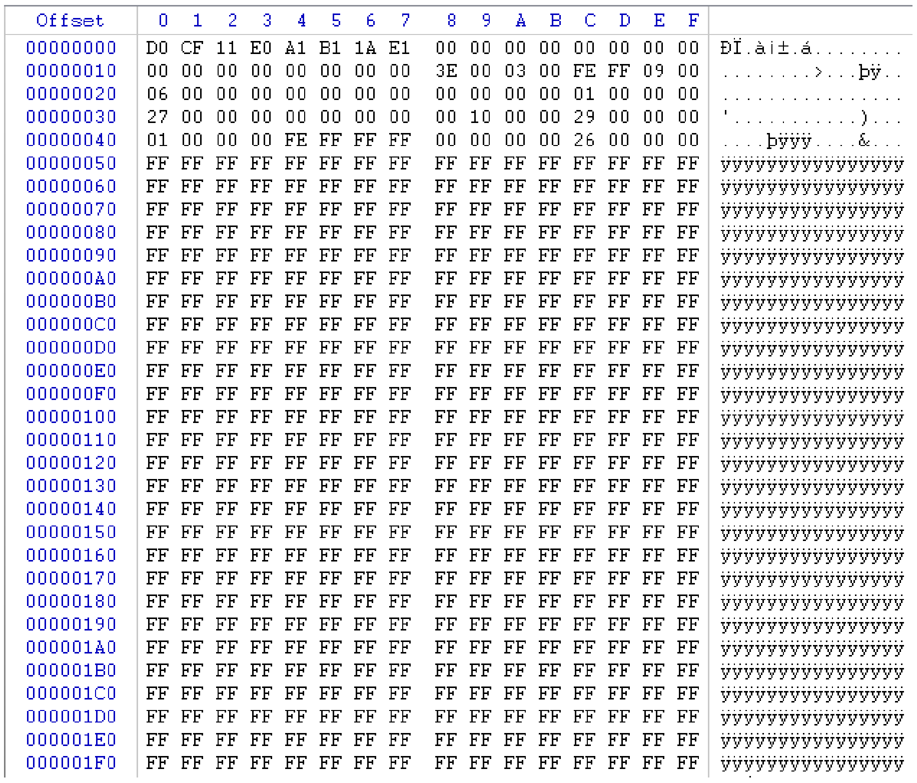


Figure 2.6: Example of a secID chain of a stream is [1,6,3,5,-2]

COMPOUND DOCUMENT HEADER

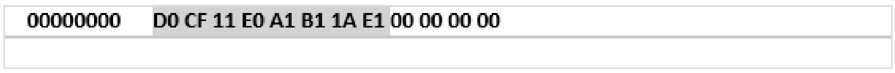
The header has all the required information necessary for reading a MCCFD file. Header is always present at the starting of a file and has a fixed length of 512 bytes, which means that the secID zero known as the sector identifier always starts at file offset of 512 bytes.



```
  Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
 00000000  D0 CF 11 E0 A1 B1 1A E1  00 00 00 00 00 00 00 00  ÐÏ.à¡±.á........
 00000010  00 00 00 00 00 00 00 00  3E 00 03 00 FE FF 09 00  ........>...þÿ..
 00000020  06 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  ................
 00000030  27 00 00 00 00 00 00 00  00 10 00 00 29 00 00 00  '...........)...
 00000040  01 00 00 00 FE FF FF FF  00 00 00 00 26 00 00 00  ....þÿÿÿ....&...
 00000050  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000060  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000070  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000080  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000090  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000000A0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000000B0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000000C0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000000D0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000000E0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000000F0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000100  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000110  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000120  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000130  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000140  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000150  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000160  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000170  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000180  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 00000190  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000001A0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000001B0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000001C0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000001D0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000001E0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
 000001F0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

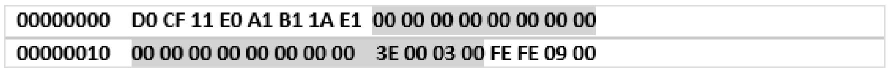
Figure 2.7: Header of a compound file

The first eight bytes are the fixed file identifier for a Microsoft Compound file format



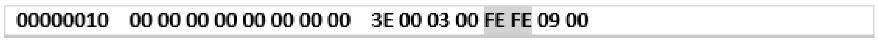
```
00000000    D0 CF 11 E0 A1 B1 1A E1 00 00 00 00
```

The next 16 bytes include a unique identifier followed by a four byte version no or revision number.



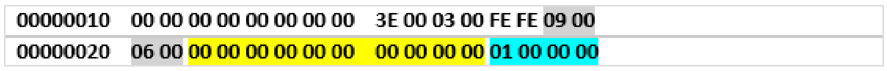
```
00000000   D0 CF 11 E0 A1 B1 1A E1 00 00 00 00 00 00 00 00
00000010   00 00 00 00 00 00 00 00   3E 00 03 00 FE FE 09 00
```

The byte order is identified by the next two bytes , it will always have the value FE FF.



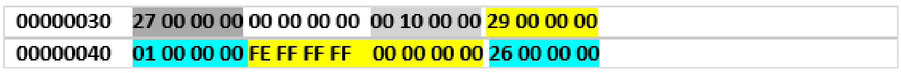
```
00000010   00 00 00 00 00 00 00 00   3E 00 03 00 FE FE 09 00
```

The following two bytes show the size of a sector and the bytes following it show the size of a short sector, the size of a sector is 64 bytes and size of a short sector is 64 bytes. After that there comes ten bits without any valid data so we can ignored them, then the last four bits show the number of sectors occupied by Sector Allocation table(SAT) and in this case it is just one.



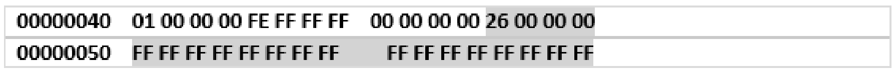
```
00000010   00 00 00 00 00 00 00 00   3E 00 03 00 FE FE 09 00
00000020   06 00 00 00 00 00 00 00   00 00 00 00 01 00 00 00
```

The first four bytes contain the SecID of the directory, In this case the directory starts at offset 00000027 which means sector 39.Next four bytes are ignored because they don't contain any valid data. The next bytes identify the minimum size a standard stream can occupy, it is 00001000 which turns to be 4096 bytes. Next four bytes determine the sector ID of the first sector occupied by SSAT which is 00000029 that becomes sector 41 and the next four bytes determine how many sectors are occupied by SSAT which in this case is just one. Then comes the sector ID of the MSAT and the sectors occupied by it. Its value is FEFFFFFF which is -2, and we know that it is the end of chain. So know that there is so MSAT in this example.



```
00000030   27 00 00 00 00 00 00 00 00 10 00 00 29 00 00 00
00000040   01 00 00 00 FE FF FF FF   00 00 00 00 26 00 00 00
```

The next 109 bytes contain the sectors occupied by MSAT. We can see that only one sector is valid because the SAT only occupies one sector. So all the rest if the sectors are given the special value of FFFFFFF i.e. -1 to indicate free sectors. The only sector used is sector 00000026 i.e. 38.



```
00000040   01 00 00 00 FE FF FF FF   00 00 00 00 26 00 00 00
00000050   FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF
```

MASTER SECTOR ALLOCATION TABLE

Since the master allocation is present inside the header so its sector chain would look like [0,-2]

SECTOR ALLOCATION TABLE

The most important control stream in case of compound file is a Sector Allocation Table(SAT). These SAT can be one or more than one in number. SAT is basically used for the arrangement of control streams in sectors and represents them in the form of chains.

2.5.3 The Introduction of Office Open XML (DOCX)

Due to the increase in the readily available open source office suits, Microsoft started working on a more open standard in early 2000s. This resulted in the formation of DOCX file format for documents, XLXS format for excel and PPTX for presentations.
These new standards were given the name of "Open Office XML" and were made using the extensible Markup language rather than the binary format. There were a number of advantages of these new standards like lesser chances of data corruption, very small document sizes and compressed images were handled without more compression.

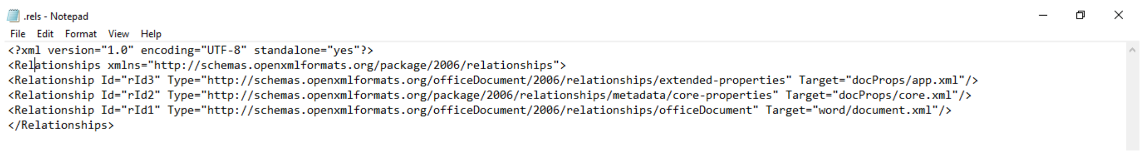
OPEN OFFICE FILE FORMAT

An Office Open XML (OOXML) document is basically a series or collection of word documents just like we have subfolders. One greatest advantage of this format is that it provides more flexibility and ease as to when we have to edit the document because everything is divided in parts.
In order to view the compressed components of a Microsoft Word file, we need to change the file extension from .DOCX to .ZIP, This will show the Microsoft Word document as series of compressed folders. Following is an example containing a traditional word document

and open office file format document containing the text *"This is a text document"*. We are breaking the individual components of the word document down for better understanding:

$_rels/.rels$

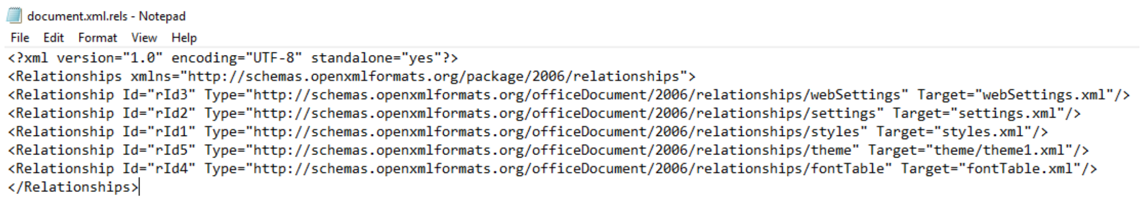
The rels tells Ms word as to where we can find the contents of the whole document. In this case, it references word/document.xml for document contents, docProps/app.xml for extended properties and docProps/core.xml for core properties like versions and authorship.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId3" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/extended-properties" Target="docProps/app.xml"/>
<Relationship Id="rId2" Type="http://schemas.openxmlformats.org/package/2006/relationships/metadata/core-properties" Target="docProps/core.xml"/>
<Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument" Target="word/document.xml"/>
</Relationships>
```

$_rels/document.xml.rels$

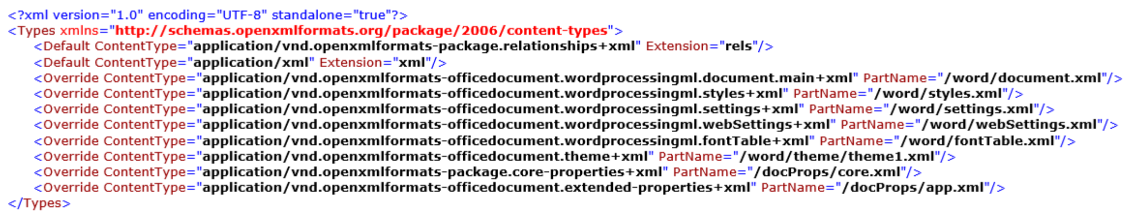
This file defines the objects that are defined in the document like images etc. We have five resources with the Id=rId1.rId2,rId3,rId4, rId5. rId1 references to style.xml in our case, rId2 is referring to settings.xml , rId3 shows websettings, rId4 has fontTable information and rId5 has theme in it.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId3" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/webSettings" Target="webSettings.xml"/>
<Relationship Id="rId2" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings" Target="settings.xml"/>
<Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styles" Target="styles.xml"/>
<Relationship Id="rId5" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/theme" Target="theme/theme1.xml"/>
<Relationship Id="rId4" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/fontTable" Target="fontTable.xml"/>
</Relationships>
```

$[Content_Types]$**.xml**

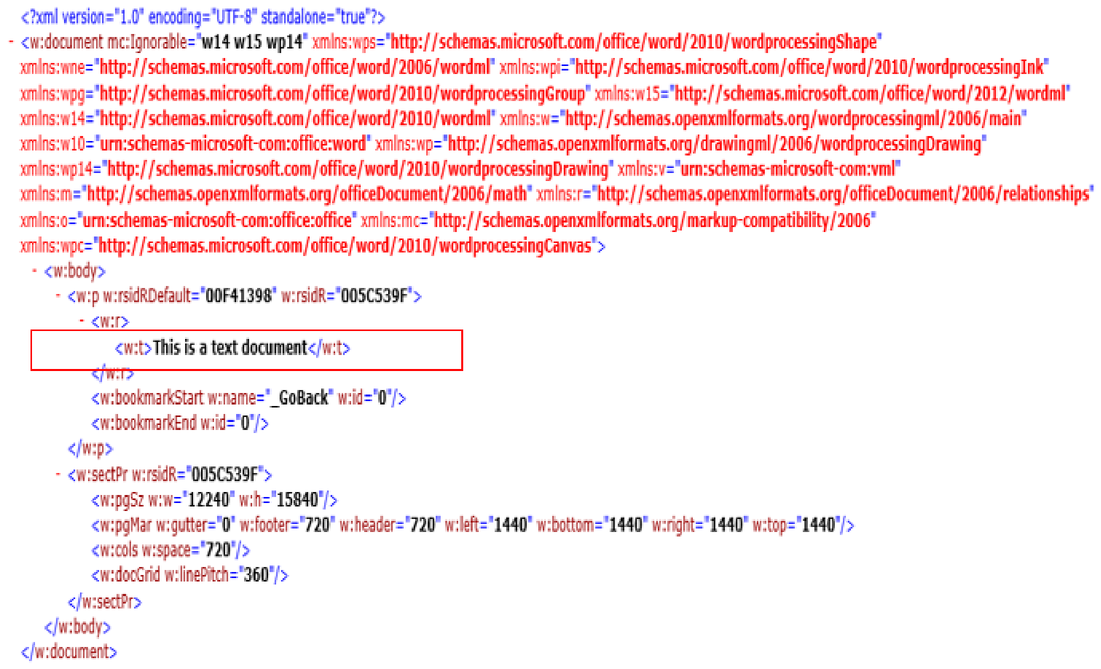
$[Content_Types]$.xml is a type of dictionary and it includes information about what types of media there exists inside the document. In our example since we are working with the textual content of the word file for now, so it becomes pretty simple:



```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
    <Default ContentType="application/vnd.openxmlformats-package.relationships+xml" Extension="rels"/>
    <Default ContentType="application/xml" Extension="xml"/>
    <Override ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml" PartName="/word/document.xml"/>
    <Override ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.styles+xml" PartName="/word/styles.xml"/>
    <Override ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.settings+xml" PartName="/word/settings.xml"/>
    <Override ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.webSettings+xml" PartName="/word/webSettings.xml"/>
    <Override ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.fontTable+xml" PartName="/word/fontTable.xml"/>
    <Override ContentType="application/vnd.openxmlformats-officedocument.theme+xml" PartName="/word/theme/theme1.xml"/>
    <Override ContentType="application/vnd.openxmlformats-package.core-properties+xml" PartName="/docProps/core.xml"/>
    <Override ContentType="application/vnd.openxmlformats-officedocument.extended-properties+xml" PartName="/docProps/app.xml"/>
</Types>
```

document.xml

Document .xml is the part where there is the main XML textual content of the document. Our simple example document has the text *"This is a text document"* inside it. We can see in the figure below that the text is written in the tag $< w : t >$ in the tag $< w : body >$.



```
  <?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <w:document mc:Ignorable="w14 w15 wp14" xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShape"
  xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml" xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocessingInk"
  xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGroup" xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
  xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml" xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
  xmlns:w10="urn:schemas-microsoft-com:office:word" xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"
  xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing" xmlns:v="urn:schemas-microsoft-com:vml"
  xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math" xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
  xmlns:o="urn:schemas-microsoft-com:office:office" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas">
  - <w:body>
    - <w:p w:rsidRDefault="00F41398" w:rsidR="005C539F">
      - <w:r>
          <w:t>This is a text document</w:t>
        </w:r>
        <w:bookmarkStart w:name="_GoBack" w:id="0"/>
        <w:bookmarkEnd w:id="0"/>
      </w:p>
    - <w:sectPr w:rsidR="005C539F">
        <w:pgSz w:w="12240" w:h="15840"/>
        <w:pgMar w:gutter="0" w:footer="720" w:header="720" w:left="1440" w:bottom="1440" w:right="1440" w:top="1440"/>
        <w:cols w:space="720"/>
        <w:docGrid w:linePitch="360"/>
      </w:sectPr>
    </w:body>
  </w:document>
```

The main node $< w : document >$ is representing the actual document, $< w : body >$ is the main body of the document and it contains paragraphs, and nested within these paragraphs are the dimension of the page defined by the tag $< w : sectPr >.< w : rsidR >$ is an internal attribute of the MS word so we can ignore it.

Paragraph Structure

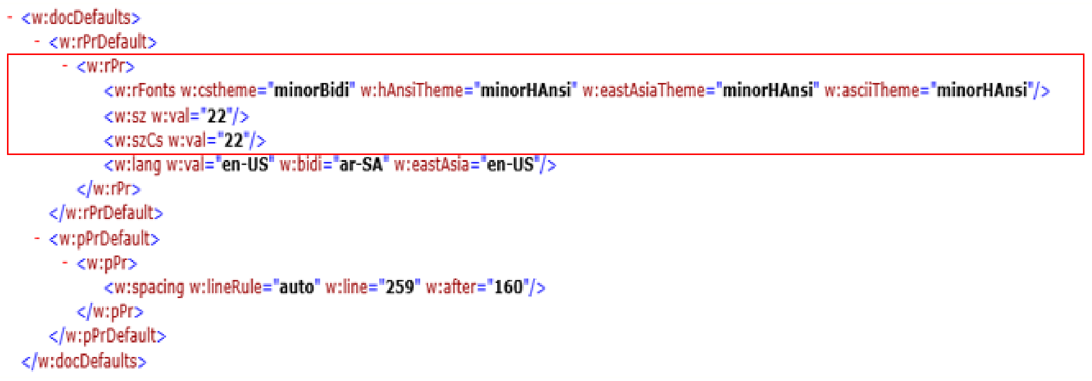
Every Word document has paragraphs, paragraphs are basically a collection of text having similar font, size color and editing etc.This set of same text is known as a run and a paragraphs can have single to multiple runs.

Text properties

Some of the text properties include color,style,size and font so on.There can be upto 40 tags used in specifying the text of a word file.As we mentioned earlier that each would have its own text properties.

Styles

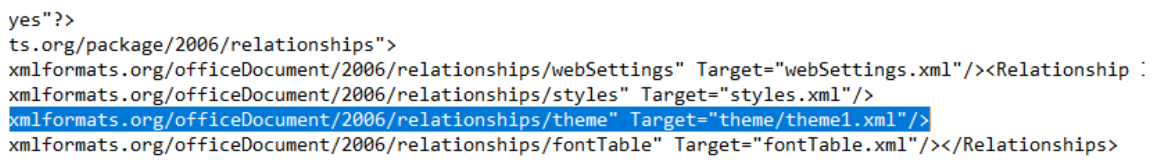
There are a variety of styles in a word document etc. All of these styles are stored in the xml file known as */word/styles.xml*. So when we choose a style for text than the type of this style can be found inside the tag labelled as $< w : pPr >$which is also known as paragraph properties.



```
- <w:docDefaults>
  - <w:rPrDefault>
    - <w:rPr>
        <w:rFonts w:cstheme="minorBidi" w:hAnsiTheme="minorHAnsi" w:eastAsiaTheme="minorHAnsi" w:asciiTheme="minorHAnsi"/>
        <w:sz w:val="22"/>
        <w:szCs w:val="22"/>
        <w:lang w:val="en-US" w:bidi="ar-SA" w:eastAsia="en-US"/>
      </w:rPr>
    </w:rPrDefault>
  - <w:pPrDefault>
    - <w:pPr>
        <w:spacing w:lineRule="auto" w:line="259" w:after="160"/>
      </w:pPr>
    </w:pPrDefault>
  </w:docDefaults>
```

Fonts

There are a variety of fonts in the word document and the reference of these fonts can be found in $word/_{r}els/Document.xml.rels$:



```
yes"?>
ts.org/package/2006/relationships">
xmlformats.org/officeDocument/2006/relationships/webSettings" Target="webSettings.xml"/><Relationship
xmlformats.org/officeDocument/2006/relationships/styles" Target="styles.xml"/>
xmlformats.org/officeDocument/2006/relationships/theme" Target="theme/theme1.xml"/>
xmlformats.org/officeDocument/2006/relationships/fontTable" Target="fontTable.xml"/></Relationships>
```

The default font name is found in $word/theme/themes1.xml$, also inside a theme tag exist a major font and a minor font tags.

Text alignment

Text alignment is done using the tag $< w : jc >$ with four modes of $< w : val >$ available. The modes are right, left , center and both. Default mode is the left mode, text always starts with the left of page's width. The center will align all the characters to appear at the center according to the of page's width. For right mode, text is aligned to the right of the page's margin. The both modes aligns the text equally on both the edges of the page's margin.

Tables

The tags used for making tables in XML are quite similar to the tags used in HTML i.e $< table >$ for XML tag matches with $< tr >$ for the html tag, etc.$< w : tbl >$ tag is the table itself. Its properties are stored in tag $< w : tblPr >$. Each table grid tag $< w : tblGrid >$ contain the column property mentioned in $< w : gridCol >$. Rows come one after the other as $< w : tr >$ tags and number of rows should be equal to the number of columns generally

Width size is specified in the $< w : tblW >$ tag, but giving the width, manual is not required because MS word's internal algorithm find the minimum width required for making a table of effective minimum size.

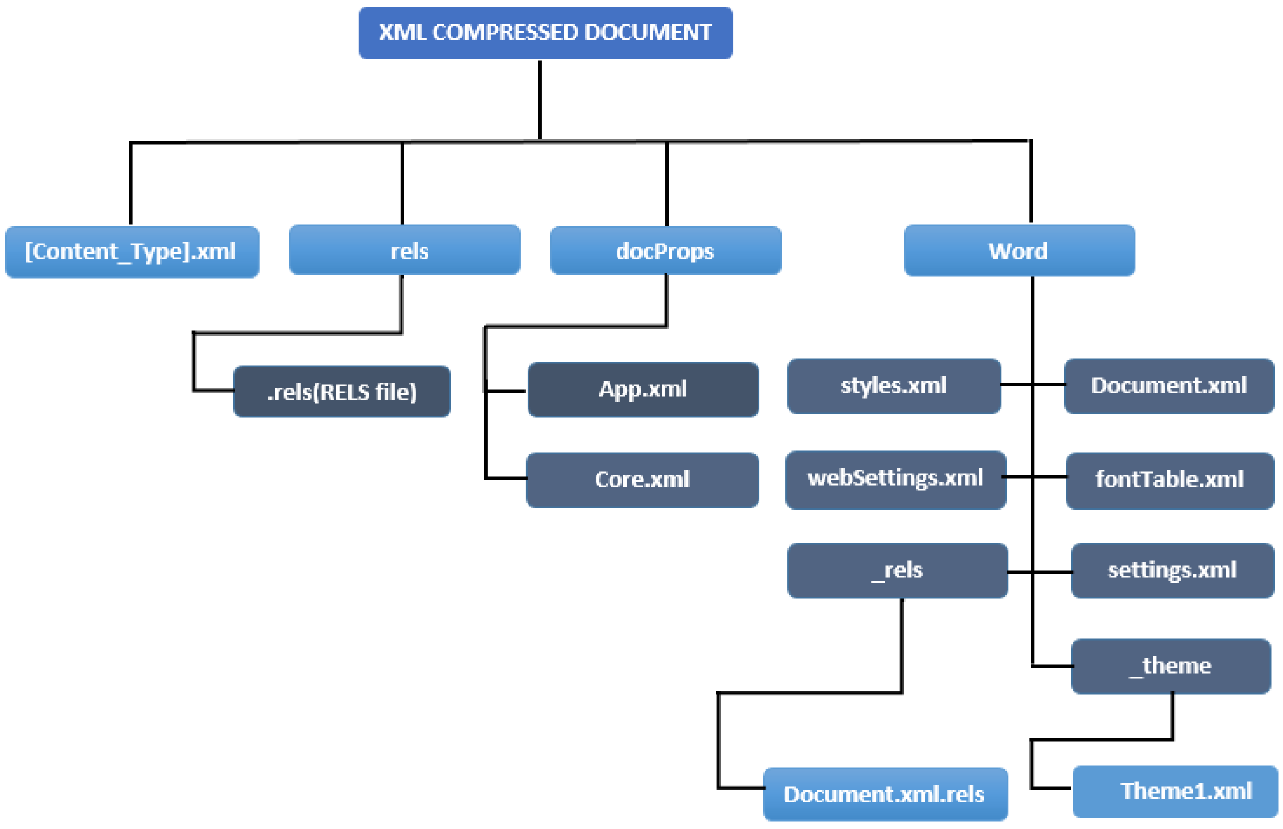


Figure 2.8: The main components of the Compressed Microsoft XML format

Chapter 3

EXISTING METHODOLOGIES FOR CARVING

Initially Foremost used header and footer carving and was developed by US airforce. [18]
Cohen proposed the idea that the carving of fragmented files was equivalent to a mapping function of recovered files and byte image of storage media, Cohen suggested that the recovery of fragmented files was possible by a generator that will generate all potential mapping functions. The acquired mappings were then fed to a validator for validation. Cohen also suggested that the disadvantage of using this approach was the processing required for finding all possible mappings and further solution was needed for this processing intensive problem. [19]
Brian Roux in his research determined that the text files particularly the ASCII files are overlooked in data carving applications because of the absence of the header and footer information in them. He further proposed a technique for rebuilding of ASCII file fragments using machine learning. [20]
A. Pal and N. D. Memon [21] the evolution of carving in detail and showed the problems with the existing recovery techniques in detail. They also used their technique to recover image and text but further suggested the need for a customized software for the recovery of other formats like video, executables and audio etc.
Xinyan Zha and Sartaj Sahni [22] have also explored Scalpel and pointed out that it spends a lot of time in searching for headers and footers. Hence, if the algorithms used by scalpel were changed to the one proposed by them, then its performance can be increased 17 times and the search time is drastically reduced as well.
Rainer Poisel, Simon Tjoa, and Paul Tavolato [23] carried out their research on extracting the digital images from the multimedia files using carving technique. They also incorporated the fragmented image files in their research and developed an open source carving tool.
In 2012 Wei Lin and Ming Xu [24]proposed a method for carving text document using virtual streams in a text document. Firstly they located the header and control streams for construction of word file's framework and later utilized this framework to find the fragmented

regions.

Azzat AI-Sadi, Manaf Bin Yahya, Ahmad Almulhem [25] in their work specifically addressed the problem of image fragmentation and its detection. They used the image pixel value for identification of fragments of different images from a pool of images using Naïve Bayes Multinomial Updateable, Multi Class, Random Forest and Bayes Net classifiers. These classifiers are selected due to their popular usage with images. The results they obtained further validated their choice of technique.

Luigi Sportiello and Stefano Zanero [3] carried out research on carving data using block classification. They used simple training and reduced the number of false negatives and positives in a block classifier of the single block. They also improved the context-based classification by labelling the unidentified blocks as miss-classified rather than using them in their classification that greatly improved results for them.

Vassil Roussev and Simson L. Garfinkel [26] in their work proposed that the practice of using only header and footer byte sequence in identification of specific file type for file carving is flawed mainly because it does not cater for the complex file structure of compound files like ZIP, DOC and PPT etc. They further supported their argument by examining the framework of these formats and reached to the conclusion that the data sets used for file carving using headers and footers were very limited and did not cater for compound and complex files. According to their study the most effective methods used for carving only worked on a specific kind of data set. They proposed that specific classification carvers are required for carving each type of complex file for a more practical outcome.

Ziad A. Al-Sharif, Dana N. Odeh and Mohammad I. Al-Saleh [27] proposed a technique to extract open PDF files from RAM based on the pointers in a pdf file that represent the start and end of a pdf file, they used these special pointers for locating and extracting the pdf files from memory. They performed many experiments and proved that it is possible to extract pdf files from memory even when the pdf viewer is closed.

Simson L Garfinkel and Micheal McCarrin [28] presented a relatively different method for carving files i.e. by using hashes. They proposed that for finding the targeted files on a system, instead of taking hashes of the whole file ,one can take hashes of individual blocks of files and still determine whether a file was fragmented or not. Previous efforts in this regard catered for only one file or one or two fragments but their research was based on a dataset of millions of files.

James Wagner, Alexander Rasin and Jonathan Grier [29] had their research on how to carve files from databases. According to them there existed a lot of database recovery tools but there is no such tool for carving databases specifically. They made a universal tool for carving databases that would also work for fragmented databases as well. Further they verified their tool as an efficient solution for recovery of slightly corrupted and deleted data from databases.

Amer Aljaedi, Dale Lindskog, Pavol Zavarsky, Ron Ruhl, Fares Almari [30] in their research carried out a comprehensive comparison between the two mostly used live memory forensics techniques which are live memory forensics and forensics analysis of memory dump taken from the kernel. It was proved that live memory forensics plays a more vital role in memory forensics as compared to image analysis and in their research they also proposed some important information that can be considered as forensics artefact which is usually missed in live analysis.

Aaron Walters and Nick L. Petroni Jr [31] described the important role of volatile memory in digital investigations and also pointed out various shortfalls in the existing techniques of live memory forensics. They further provided a mechanism for extraction of keying information from the disk without the knowledge of the password. They also discussed about Volatools, which is software for extraction of live responses from memory.

William C. Calhoun and Drue Coles [32] devised two algorithms for identifying the various kinds of file fragments. One algorithm used the byte frequency i.e. the frequency of occurrence of bytes, in order to identify the type of a certain file fragments. The other used the idea that two files which are of the same type would have same type of strings in them, this is known as the common longest substring method. Both these methods were not very efficient and required a larger dataset and efficiency improvements for practical purposes.

Husrev T. Sencar and Nasir Memon [33]work focused on identification of next fragment of an encrypted JPEG file, they used the JPEG file format and did bit pattern matching for identifying the next segment of an encrypted JPEG. Also they worked with fragments that cannot be linked to a header or their header is just missing.

Mehdi et. al also proposed a technique for the identification of the file type of the fragments in memory. The technique used by them extracted 15 features from byte frequency distribution and based on this they used both the SVM classifier and MLP classifier for classifying their test fragments. Their comparison proved that SVM is a better classifier when it comes to classifying different types of fragmented data. [34]

Binglong et al [34] worked specifically on finding the Document fragments and used the technique of enhanced string kernel (ESK) for its classification purposes. ESK is used to extract a small sequence of bytes from the document header fragment and uses it for matching with the other available fragments. This pattern matching would led in finding the remaining fragments of the file header.

Golden et all [35] proposed a new technique for carving. He reasoned that traditional cravers doesn't use the host machine for analysis, what they really do is make a copy of the whole disk and it can be very time consuming most of the time. So they introduced a technique known as the in place file carving. In this type of carving the investigator uses the host machine itself for investigating the files live without making a copy of them.

Nadeem et al [36] did a very extensive literature review on the existing techniques of carving and he proposed that there was a need of more realistic data sets for carving, he also said that special focus is required for carving of the fragmented files. Also said that there was a need to detect hidden or malicious injections in unoccupied spaces and carving should be used effectively for this purpose.

Zaid et al [37] performed a forensics analysis of RAM and he used the XML representation of the word document to find out which Word document the criminal was viewing or editing. He extracted the XML representation of the number of paragraphs and the textual content of the word file from RAM , he than matched these both to the available documents in the system. The document that matched was the document being viewed.

Hyukdon et al [38] proposed that there was a problem with the Microsoft Compound Document File Format $(MCDFF)$ that it is easy to insert information in the MCDFF but it is very hard to detect that information. He said that open source tools can be used for adding malicious contents in the MCDFF, so he presented the analysis of the possible exploits. Also formed a tool called Doc detector that helped in detection and analysis of the hidden malicious data.

Irfan et al [39] proposed a technique for a faster identification of file fragments, his technique involved selection of only the most frequently occurring frequencies for calculating the byte frequency distribution for content based file carving. Also the classification processing time was reduced by randomly selecting chunks of data rather than using every chunk consecutively.

After the summary of the techniques being used, we list down the existing data carving open source tools in Table below:

Table 3.1: Summary of Existing Carving Tools

Name	**Algorithm**	**Type of Artefact**
Scalpel	It is used by finding header and footers and extracting those header footer pairs	Non fragmented files
Foremost	It uses the technique of sequential finding of header and footer	Image files
Volatools	It basically views the memory pages	Extraction of data from a live resource

The most common tool used for carving of non-fragmented files is Scalpel. For the extraction of image files the tool used is called Foremost developed by US Army. Another well-formed tool for extraction of data from live memory is known as Volatools. Other than these we don't have any proper tool designed for the carving of data from memory.

All the research papers mentioned in section discuss the carving of data from memory using multiple techniques and procedures.Microsoft file format has been researched and studied very less because it is a compound format.To store the data directories and hierarchal structure of streams is used. Research has been conducted in either the extraction of images or text from documents but there has been no detailed research for the extraction of both components simultaneously.The extraction process becomes more complex in case of fragmented word documents and recovery of image and text is almost impossible.

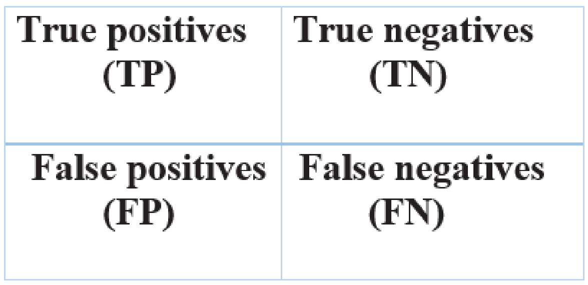
3.1 Quantitative Measures

Quantitative measure is required for evaluating the performance of the approach one uses. It is also required for comparison with the existing techniques and also for tracking improvement in your own approach. While choosing the evaluation measure it is important to consider all the choices available and selecting your quantitative measures accordingly.

In classification of text each text instance belongs to one of the available classes. So if each class is assigned a label than the two of the class cases would *"+1"* and *"-1"*. *"+1"* for positive and *"-1"*for negative. A simple case is that we count the number of labels assigned by the system that are correct.

[40] In text classification the commonly used performance evaluation measures are a function of these followings:

- True positives (TP): The system predicts *“+1”*for text that is actually present in the class.
- True negatives (TN): The system predicts *“-1”* for text that is not present in the class.
- False positives (FP): The system predicts *“-1”* for text that is actually present in the class.
- False negatives (FN): The system predicts *“+1”* for text that is not present in the class.



True positives (TP)	**True negatives (TN)**
False positives (FP)	**False negatives (FN)**

In order to find that how much relevant data is retrieved by the system, we use the measures precision and recall. These measures are defined below:

3.1.1 Precision

Precision is used to measure of how exact a classifier is performing. A classifier with less false positives is a good classifier, whereas a classifier with more false positive is a bad classifier. The basic question answered by precision is *“what is the correct proportion of positive identification?*”

Precision = Total number of relevant documents retrieved /Total number of documents that are retrieved.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.1}$$

Example: If in case we have one true positive and false positive the precision would be half i.e. 0.5

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{1}{1+1} = 0.5 \tag{3.2}$$

3.1.2 Recall

Recall measures how sensitive or complete a classifier is. High value of recall would mean lesser false negatives, while low value of recall would mean higher false negatives. Recall is used to answer *"How many actual positives were correctly identified?"*

Recall = Total number of relevant documents retrieved /Total number of relevant documents in the database.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.3}$$

Example: If consider in a scenario that the TP is one but the value of FN is 8 then our Recall would be 0.11.

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{1}{1+8} = 0.11 \tag{3.4}$$

In order to fully evaluate our model we need to consider both precision and recall. But the fact is that precision and recall are inversely related to each other that means increasing precision would reduce recall and vice versa. [41]

3.1.3 F-measure Metric

Many times Precision and recall are combined together to make a single metric known as F-measure. F mean is basically harmonic mean of both precision and recall.

$$\text{F} = \frac{2 * ((Precision * Recall)}{(Precision + Recall))} \tag{3.5}$$

PROPOSED METHODOLOGY

We are assuming that the metadata of the files in RAM is lost. So in order to recover them we would need to carve the Word Document from the RAM. This assumption is pretty logical because the data in the RAM is constantly overwritten as new files are viewed/edited or new processes are used. In many cases the carving of Microsoft Word Document using its own internal structure is the only option available to us. A Word Document consists of many components and these components can be uniquely identified from the RAM. These components may come from different files if more than one file is being viewed at a time. But the real challenge is to identify the parts belonging to different files. The challenge is overcome using SVM classifying, which will classify each component based on its content. The figure shows our model for investigation

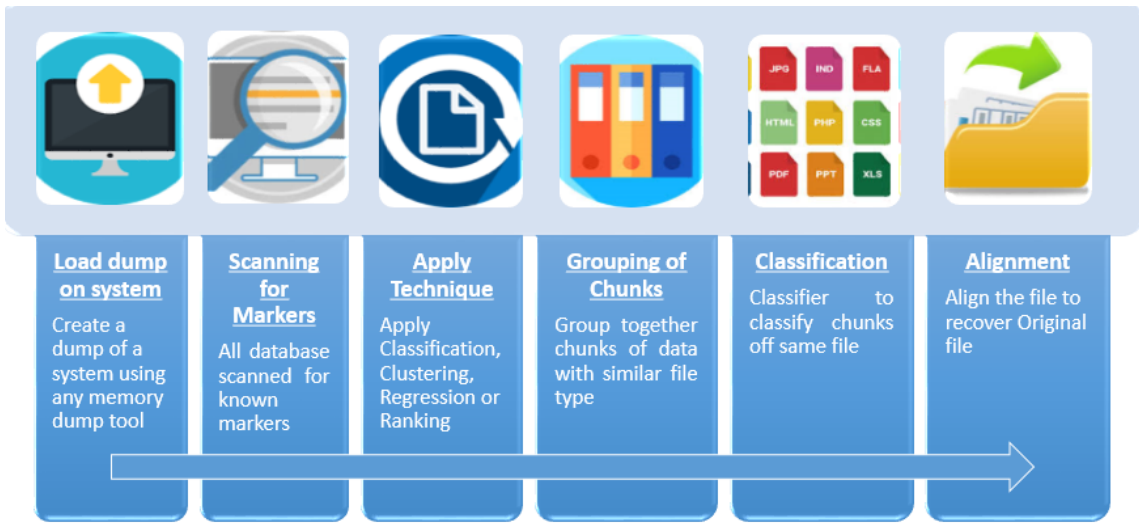
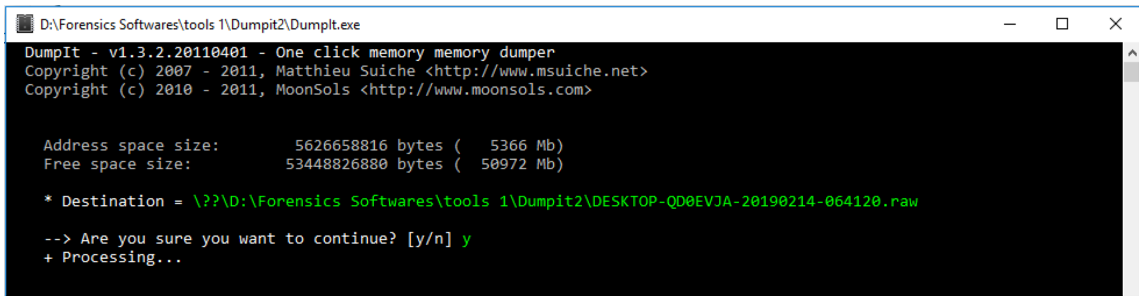

JPG
IND
FLA
HTML
PHP
CSS
PDF
PPT
XLS
Load dump on system
Create a dump of a system using any memory dump tool
Scanning for Markers
All database scanned for known markers
Apply Technique
Apply Classification, Clustering, Regression or Ranking
Grouping of Chunks
Group together chunks of data with similar file type
Classification
Classifier to classify chunks off same file
Alignment
Align the file to recover Original file

Figure 4.1: Model for Investigation

4.1 CREATING OF RAM DUMP

There are a number of methods and open source tools available for creating memory dumps. We performed our analysis on a virtual Windows 7 Operating System (OS) made using the *"VMware Workstation"*. We used a RAM of 1GB for this OS. The tool which we used for taking the memory dump was DumpIt. DumpIt is a command prompt application, which takes a live dump of your RAM and saves it as a ".rar" format *(raw binary format)*.



```
DumpIt - v1.3.2.20110401 - One click memory memory dumper
Copyright (c) 2007 - 2011, Matthieu Suiche <http://www.msuiche.net>
Copyright (c) 2010 - 2011, MoonSols <http://www.moonsols.com>


  Address space size:        5626658816 bytes (   5366 Mb)
  Free space size:          53448826880 bytes (  50972 Mb)

  * Destination = \??\D:\Forensics Softwares\tools 1\Dumpit2\DESKTOP-QD0EVJA-20190214-064120.raw

  --> Are you sure you want to continue? [y/n] y
  + Processing...
```

Figure 4.2: The Dump created using DumpIt

sectionCARVING OF OPEN OFFICE MICROSOFT WORD DOCUMENT OBJECTS One of the first component of every document was its header. We performed our analysis in the Win Hex, where we could see the hex format of the word file. We began with our manual analysis of the word file. As there is a fixed header for each type of file, we started by finding the header for our file. [**?**]

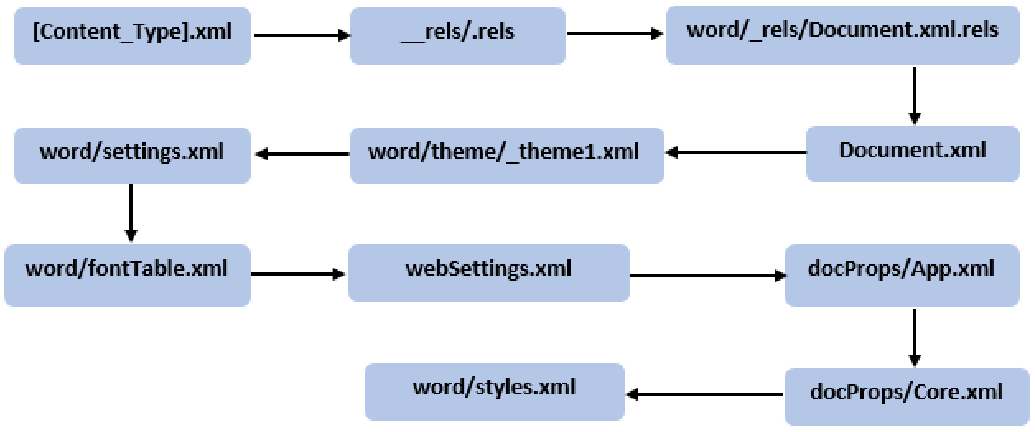
Since the DOCX is the part of the Microsoft Office Open XML Format (OOXML). There is no sub header for MS OOXML files as there is with XLS,DOC, and PPT.If we change the extension of any word document from.doc to .zip; look at the resultant file named $[Content_Types].xml$ to see the content types.In particular,look for the $< OverridePartName = tag >$, where you will find word, ppt, or xls, respectively.

The header looks like this: $504B030414000600$ followed by 18 additional bytes. The first two bytes of the Header $0x50$ $0x40$ are also the header for ZIP file, which is fine because we said that a DOCX file format is basically a collection of compressed ZIP folders.

One more thing that we deducted from our research was that there were different signatures in different Open Office Microsoft Word Documents. For this we made a dataset of 100 Microsoft Word Documents on different versions and observed the starting bytes of every document manually. We found that each version had a unique header, the starting 12 bytes were same for all the versions of Word Document, the next 8 bytes were different and unique for every version and the remaining 28 bytes were totally same in all versions. The common headers are listed below.

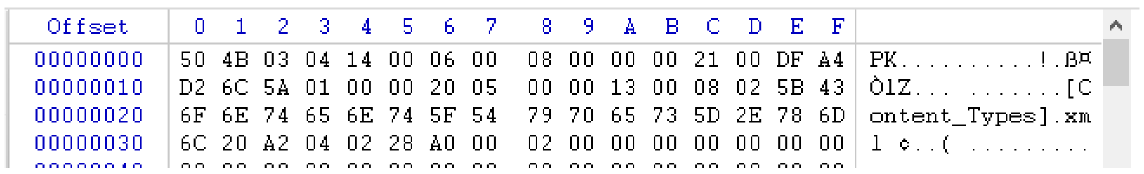
SIGNATURE	MICROSOFT VERSION
PK..........!.ß¤ÒlZ.......... [Content_Types].xml	Microsoft Office Professional Plus 2013
PK..........!.2‘oWf...¥....... [Content_Types].xml	
PK..........!.Œ.µ‡Š...-....... [Content_Types].xml	
PK..........!.ä$‰L}...)....... [Content_Types].xml	
PK..........!.>RHèq...¤....... [Content_Types].xml	
PK..........!.‡J˜3¢...-....... [Content_Types].xml	
PK..........!.-êï/r...Ù.......[Content_Types].xml	

In the Hex values the first component to appear was the $Content_Type.xml$ we have consider this the header for our convenience, after this came the $.relsfile$, then came the $Document.xml.rels$ from the Word folder, then came the $Document.xml$, the $Document.xml$ contains the actual text of the word document, followed by $theme1$, After this came the settings, $App.xml, core.xml, websettings, Fonttable, Core.xmlandstyles.xml$. After observing a number of Microsoft Word Document we found a pattern in the way the files and folders were aligned in the hex file. The following figures shows the flow in which the components of the XML document appear in file:



The first five components are constant for any word file and would appear in the same sequence but the later appear according to the use in the word file , For example if the font is set first than the Fontable.xml will appear first ,following the rest of the components.
As we know that Open Office XML representation uses the XML as its back end programming and MS word is just a compressed folder of different XML files, each of which represents a specific feature in the word document [?]. So we observed that the folder or file that appeared first in the hex file would end first in the file.

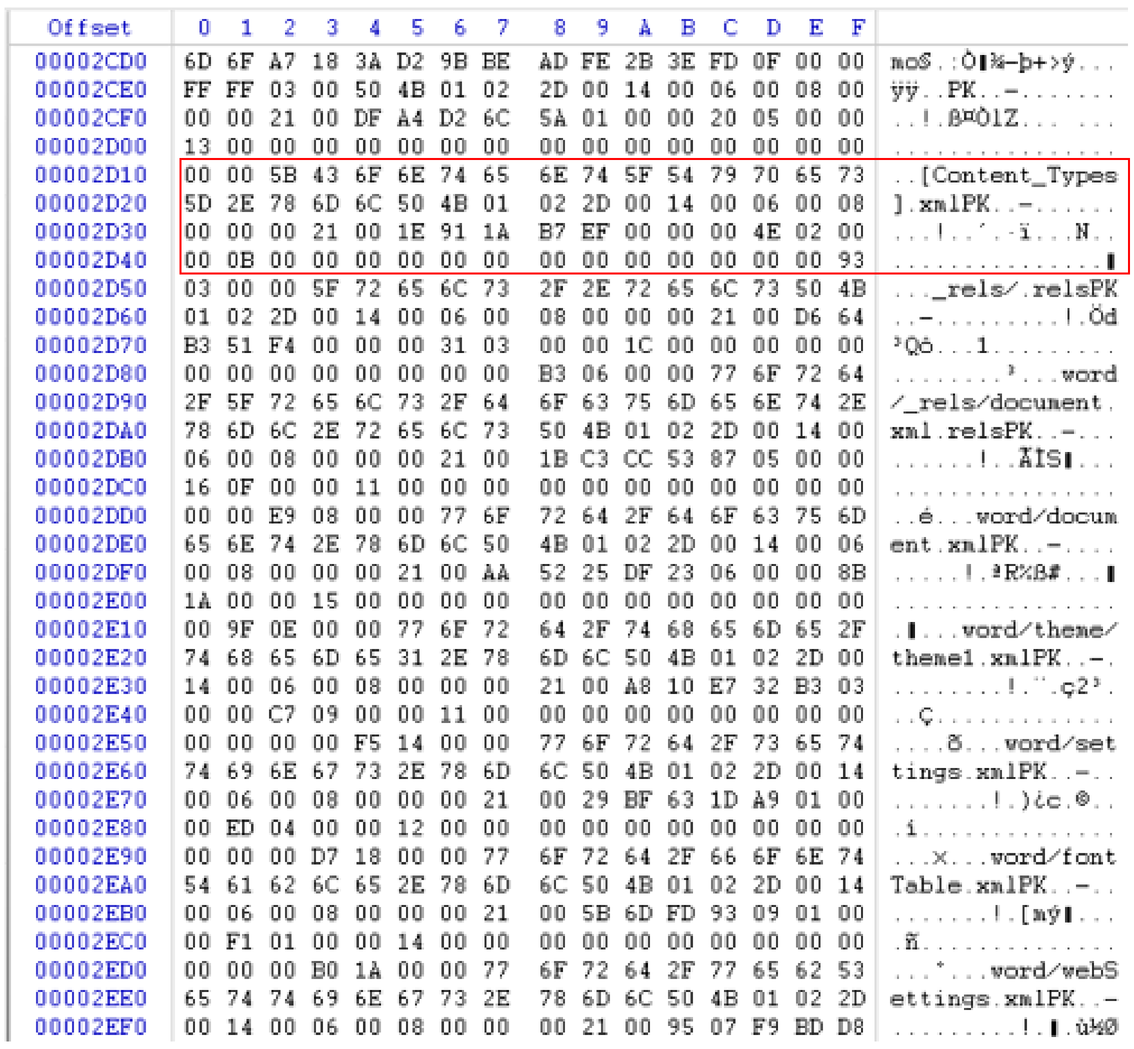
For Example if $[Content_Type].xml$ comes first in the compressed format than when the file will begin to end the first to appear in the ending section would be the $[Content_Type].xml$. So we can consider this like an xml wrapping.



```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000  50 4B 03 04 14 00 06 00  08 00 00 00 21 00 DF A4  PK..........!.ß¤
00000010  D2 6C 5A 01 00 00 20 05  00 00 13 00 08 02 5B 43  ÒlZ... .......[C
00000020  6F 6E 74 65 6E 74 5F 54  79 70 65 73 5D 2E 78 6D  ontent_Types].xm
00000030  6C 20 A2 04 02 28 A0 00  02 00 00 00 00 00 00 00  l ¢..( .........
```

Figure 4.3: Shows the header of an xml compressed file format

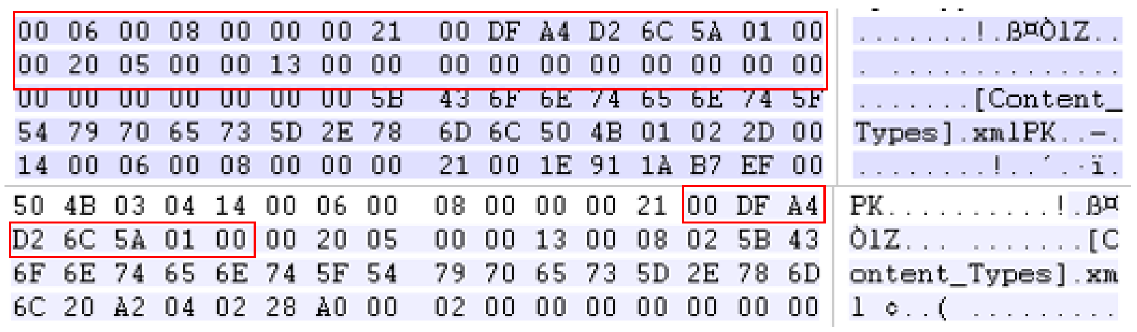
Figure below shows the ending of a Microsoft word document and we can see that the $[Content_Type].xml$ comes first in the ending because it appeared first in the start as well. Similarly whatever file or folder appears first in the starting of the file will also appear first in the ending of the file.



```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00002CD0  6D 6F A7 18 3A D2 9B BE  AD FE 2B 3E FD 0F 00 00  mo§.:Ò▮¾þ+>ý...
00002CE0  FF FF 03 00 50 4B 01 02  2D 00 14 00 06 00 08 00  ÿÿ..PK..-.......
00002CF0  00 00 21 00 DF A4 D2 6C  5A 01 00 00 20 05 00 00  ..!.ß¤ÒlZ... ...
00002D00  13 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00002D10  00 00 5B 43 6F 6E 74 65  6E 74 5F 54 79 70 65 73  ..[Content_Types
00002D20  5D 2E 78 6D 6C 50 4B 01  02 2D 00 14 00 06 00 08  ].xmlPK..-......
00002D30  00 00 00 21 00 1E 91 1A  B7 EF 00 00 00 4E 02 00  ...!..´.·ï...N..
00002D40  00 0B 00 00 00 00 00 00  00 00 00 00 00 00 00 93  ...............▮
00002D50  03 00 00 5F 72 65 6C 73  2F 2E 72 65 6C 73 50 4B  ..._rels/.relsPK
00002D60  01 02 2D 00 14 00 06 00  08 00 00 00 21 00 D6 64  ..-.........!.Öd
00002D70  B3 51 F4 00 00 00 31 03  00 00 1C 00 00 00 00 00  ³Qô...1.........
00002D80  00 00 00 00 00 00 00 00  B3 06 00 00 77 6F 72 64  ........³...word
00002D90  2F 5F 72 65 6C 73 2F 64  6F 63 75 6D 65 6E 74 2E  /_rels/document.
00002DA0  78 6D 6C 2E 72 65 6C 73  50 4B 01 02 2D 00 14 00  xml.relsPK..-...
00002DB0  06 00 08 00 00 00 21 00  1B C3 CC 53 87 05 00 00  ......!..ÃÌS▮...
00002DC0  16 0F 00 00 11 00 00 00  00 00 00 00 00 00 00 00  ................
00002DD0  00 00 E9 08 00 00 77 6F  72 64 2F 64 6F 63 75 6D  ..é...word/docum
00002DE0  65 6E 74 2E 78 6D 6C 50  4B 01 02 2D 00 14 00 06  ent.xmlPK..-....
00002DF0  00 08 00 00 00 21 00 AA  52 25 DF 23 06 00 00 8B  .....!.ªR%ß#...▮
00002E00  1A 00 00 15 00 00 00 00  00 00 00 00 00 00 00 00  ................
00002E10  00 9F 0E 00 00 77 6F 72  64 2F 74 68 65 6D 65 2F  .▮...word/theme/
00002E20  74 68 65 6D 65 31 2E 78  6D 6C 50 4B 01 02 2D 00  theme1.xmlPK..-.
00002E30  14 00 06 00 08 00 00 00  21 00 A8 10 E7 32 B3 03  ........!.¨.ç2³.
00002E40  00 00 C7 09 00 00 11 00  00 00 00 00 00 00 00 00  ..Ç.............
00002E50  00 00 00 00 F5 14 00 00  77 6F 72 64 2F 73 65 74  ....õ...word/set
00002E60  74 69 6E 67 73 2E 78 6D  6C 50 4B 01 02 2D 00 14  tings.xmlPK..-..
00002E70  00 06 00 08 00 00 00 21  00 29 BF 63 1D A9 01 00  .......!.)¿c.©..
00002E80  00 ED 04 00 00 12 00 00  00 00 00 00 00 00 00 00  .í..............
00002E90  00 00 00 D7 18 00 00 77  6F 72 64 2F 66 6F 6E 74  ...×...word/font
00002EA0  54 61 62 6C 65 2E 78 6D  6C 50 4B 01 02 2D 00 14  Table.xmlPK..-..
00002EB0  00 06 00 08 00 00 00 21  00 5B 6D FD 93 09 01 00  .......!.[mý▮...
00002EC0  00 F1 01 00 00 14 00 00  00 00 00 00 00 00 00 00  .ñ..............
00002ED0  00 00 00 B0 1A 00 00 77  6F 72 64 2F 77 65 62 53  ...°...word/webS
00002EE0  65 74 74 69 6E 67 73 2E  78 6D 6C 50 4B 01 02 2D  ettings.xmlPK..-
00002EF0  00 14 00 06 00 08 00 00  00 21 00 95 07 F9 BD D8  .........!.▮.ù½Ø
```

Figure 4.4: Shows the footer of an xml compressed file format

The most tedious task is to find the footer of a word file because there is no definite end point as to where the Word file would end. Particularly it has a complex hieratical structure and the structure changes with the change in the file. But we noticed a particular bytes appearing in ending of the file. These bytes are same as the unique bytes present in the header of each OOXML file. The figure below shows the unique bytes in a header.



```
00 06 00 08 00 00 00 21  00 DF A4 D2 6C 5A 01 00   ........!.ß¤Ò1Z..
00 20 05 00 00 13 00 00  00 00 00 00 00 00 00 00   . ..............
00 00 00 00 00 00 00 5B  43 6F 6E 74 65 6E 74 5F   .......[Content_
54 79 70 65 73 5D 2E 78  6D 6C 50 4B 01 02 2D 00   Types].xmlPK..-.
14 00 06 00 08 00 00 00  21 00 1E 91 1A B7 EF 00   ........!..´.·ï.
50 4B 03 04 14 00 06 00  08 00 00 00 21 00 DF A4   PK..........!.ß¤
D2 6C 5A 01 00 00 20 05  00 00 13 00 08 02 5B 43   Ò1Z... ........[C
6F 6E 74 65 6E 74 5F 54  79 70 65 73 5D 2E 78 6D   ontent_Types].xm
6C 20 A2 04 02 28 A0 00  02 00 00 00 00 00 00 00   l ¢..( .........
```

Figure 4.5: Shows the XML wrapping of an xml compressed file format

The figure above shows the ending portion of the OOXML file, we can observe that the header bytes are repeated in the ending of the file also, so we can consider this as a footer or ending.

Other than header and footer the important components that are extracted are $Document.xml.rels$, $_rels$ and $document.xml$. A number of these components is extracted from RAM based on the number of documents being viewed by the user. For further analysis all of these components are grouped together in a pool of data. .

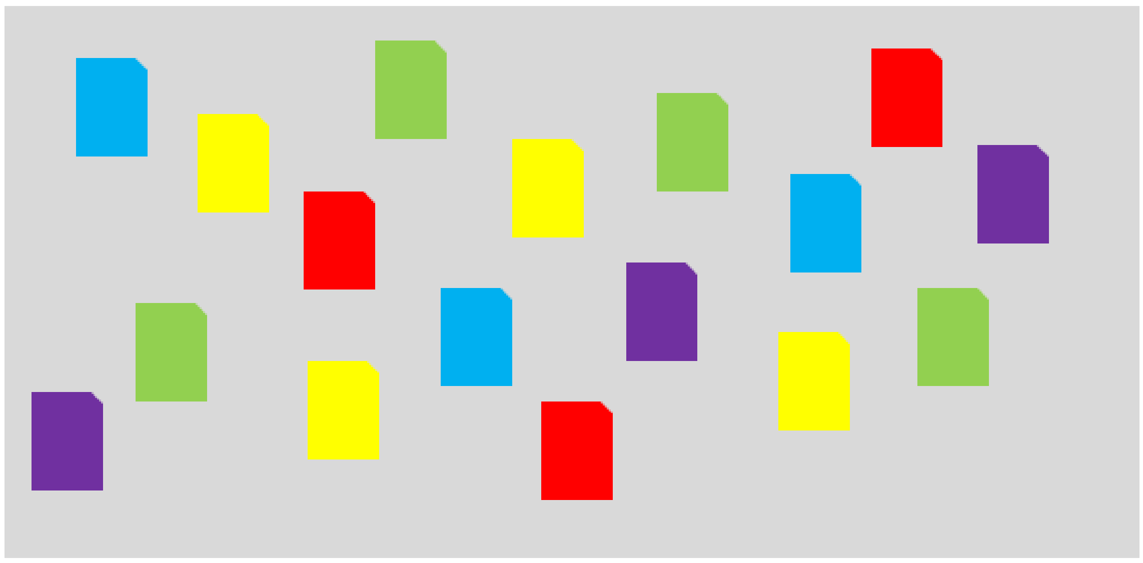


Figure 4.6: The figure shows grey space as the pool and all the colored files are components of word document

4.2 GROUPING SAME TYPE OF WORD COMPONENTS TOGETHER

Now comes the classification of the components. We now have to sort the components based on their textual content. As we can see that components which are headers have a certain type of text in them. Also the assumed footers have portion of the same header text. So we can save these headers and footers in a pair and separate them from the pool of the components. The remaining contents are also sorted. All $document.xml$ are grouped together, all $_rels$ are separated in a different pool, same happens with $[Content_Type]$ and $document.xml.rels$. So now we have different groups of similar components.

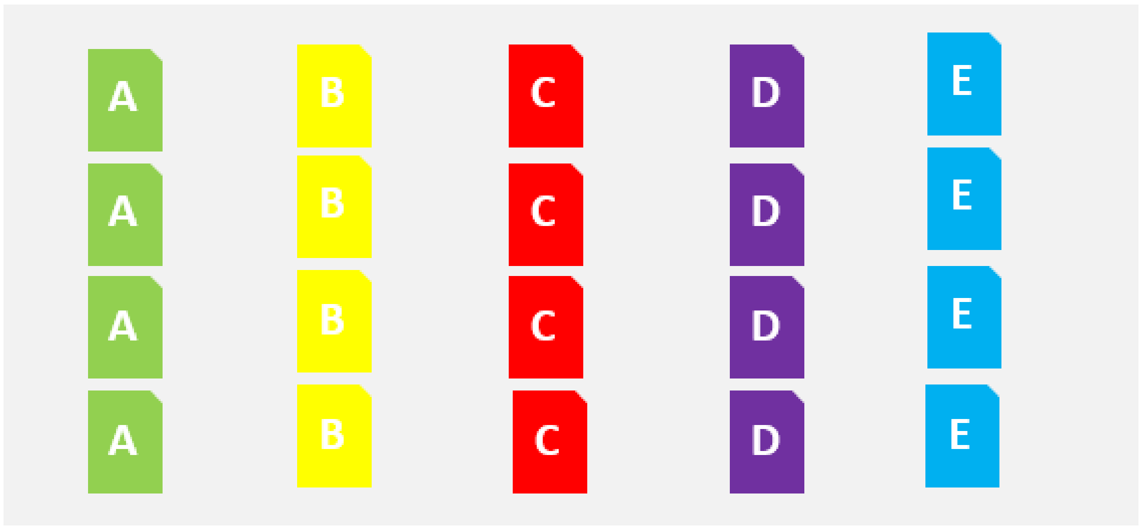


Figure 4.7: The figure shows grey space as the pool and all the colored files are components of word document

4.3 EXTRACTION OF TEXTUAL DATA

Since we have recovered different components of the word document, we analysis them further to find the textual data from the document. We first observe the $[Content_Type].xml$ data, we known from the basic structure of Open Office XML that it contains information about all the parts of the document but doesn't contain the actual content. The Open Office XML internal structure states that the content of the word document is present in the $document.xml$ life, which is a part of the word folder. But the component is ZIP encoded and the first thing we need to do is decode it using some ZIP decoder technique. After the decoding is done we further observe all the document.xml files. In the tag $< w : t >$ is the textual content of that word life. Tag $< w : t >$ is a run and it will be present inside a tag $< w : p >$ i.e. a paragraph. From this we can note down the text and the number of paragraphs for each of the word document.

4.4 TEXT SPELLING AND SPACING CORRECTION

The plain text take comes from decoding can have a number of spelling and spacing mistakes. Probably due to the fact that not every character is decoded properly. Some characters are missed hence some spacing and spelling issues can occur. So for now we would just simply correct the spaces and spelling mistakes that occur during the extraction of textual content. For this we simply pasted the content in a word document and it gave us the suggestions for the corrections. We rectified the mistakes and saved these textual instances for further processing.

For example an encoded sentence *"This is an experimental test document used for forensics analysis of RAM"* can be decoded as follows:

"T his is an e periment l text docu ment used for f o r e n sics analy sis of RAM" Like of the characters stay decoded and disrupt the whole sentence. With simple observation we can correct the spellings and spacing of the lines.

4.5 TEXT CLUSTERING

For a smaller size document the $document.xml$ can be found at one place in the RAM but for a large file the document.xml file might be present at multiple locations. For this purpose we would need to cluster together pieces of the document.xml to retrieve the contents of the original file. The technique that we would be using is called the text clustering. We would be grouping together text that have similar text instances in their contents. The idea behind this is that similar text instances are more likely to originate from the same word documents. We will be using the two most popular clustering algorithms for this the simple K mean and Hierarchical clustering, both of these are used using their default distance function. But before performing the text clustering we need to perform some preprocessing on the data.

4.5.1 Preprocessing

In order to differentiate between two texts and also to group together text from the same document we will first gather all the text instances. After gathering the data we tokenize each document i.e. we split each document into individual words.

After we have our data, we will analyze it and form our vocabulary. Vocabulary basically consists of known words or unique words from the documents. We can have as many words in our vocabulary as we want. The more the dataset is diverse, the more diverse vocabulary we can have.

After defining the vocabulary we will form vectors for each document. The vectors will be formed by putting *“1”* in place of the words present in the document and *“0”* for the words absent in the document.

After forming the vectors we will than parse these vectors or you can say shorten them up by removing the words that are known and are of lesser importance but their occurrence in any data is high e.g. like is are, the etc. By removing these common words we will be left with a shorter length of a vector.

One more thing we will do is that we will take the term frequency versus inverse document frequency tf-idf of the document. What this will do is that it will give us the frequency of words that are different in both of the documents. So we would know which word belongs to which document.

4.6 EVALUATION METRIC

For validating our results we will be using the evaluation metrics known as precision and Recall. These are text processing metrics based on true positives and false negatives.

In our scenario we have a pool of text documents and we have to separate the text that belong to one single document. The number of classes would be equal to the number of documents.

Following is the detailed overview of the whole model we will be following in our experiments performed in the next section. The part labelled *"A"* shows the process of extraction of the textual content from RAM.*"B"* shows the pre processing performed on the textual content before applying the machine learning technique and finally *"C"* Shows the details of hierarchal clustering for best results.

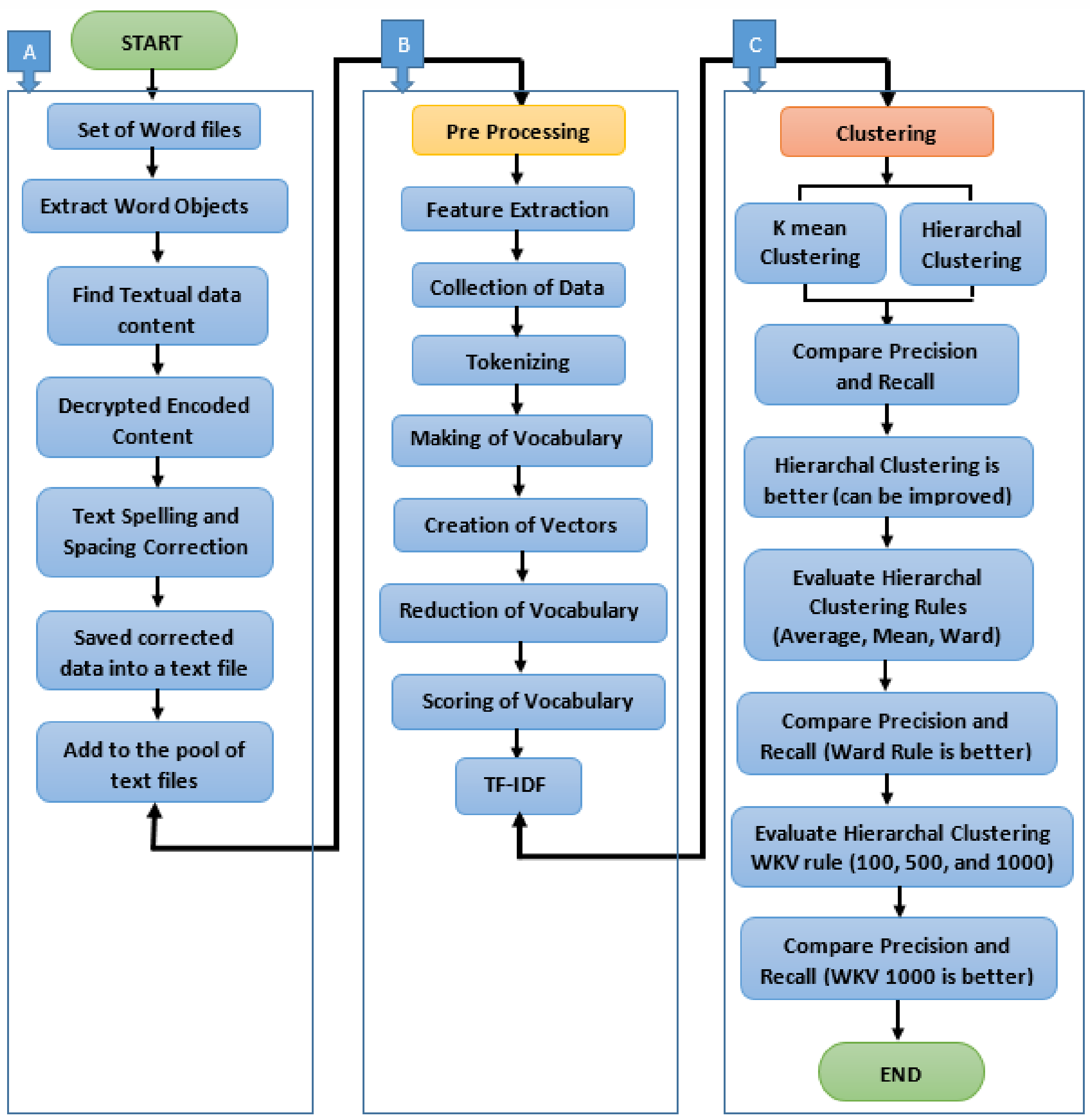

A
START
Set of Word files
Extract Word Objects
Find Textual data content
Decrypted Encoded Content
Text Spelling and Spacing Correction
Saved corrected data into a text file
Add to the pool of text files
B
Pre Processing
Feature Extraction
Collection of Data
Tokenizing
Making of Vocabulary
Creation of Vectors
Reduction of Vocabulary
Scoring of Vocabulary
TF-IDF
C
Clustering
K mean Clustering
Hierarchal Clustering
Compare Precision and Recall
Hierarchal Clustering is better (can be improved)
Evaluate Hierarchal Clustering Rules (Average, Mean, Ward)
Compare Precision and Recall (Ward Rule is better)
Evaluate Hierarchal Clustering WKV rule (100, 500, and 1000)
Compare Precision and Recall (WKV 1000 is better)
END

Figure 4.8: Detailed model of the technique

Chapter 5

EXPERIMENTS AND RESULTS

5.1 EXPERIMENTS

Our methodology consists of three parts, one is to extract the Open Office Microsoft Word Document components and second is to extract the textual content correctly from these components and thirdly we need to classify and group these textual contents correctly in order to find which content belongs to which word file. In order to achieve these goals we design our experiments as below.

Our experiment will have two portion i.e. to extract the Microsoft Word Document from RAM when it is being viewed or edited and to extract the Word Document when it is closed and not being viewed. So we need the memory dump in the two states below:

State 1: The RAM dump was taken while the MS Word document is viewed or is being edited.

State 2: The RAM dump is taken after the document is closed.

For our experiments we will be using a total of 30 word documents. These word documents are chosen at random from the internet and contain children stories, technical documents, daily news, fashion articles, food blogs etc., they include different versions of Word Document. The reason that we have chosen completely different types of documents in our experimental phase is because we can't simultaneously open thousands of word documents, which are required in terms we want to train our machine learning algorithm. So for simplicity we chose documents with diverse topics so that they can be segregated easily.

We have divided these documents into five subsets. Subset one ($S1$) includes just 1 file, Subset two ($S2$) consists of 5 files and subset three ($S3$) consists of 10 files, Subset four ($S4$) consists of 20 files and subset five (S5) consists of all 30 files. Subset two includes subset one and 4 new files. Subset three includes subset one, two and 5 new files, subset four includes subset one, two and three with 10 additional files and finally subset five contains all the previous subsets with 15 new files.

First all of these experiments are performed with the state 1 and then the same experiments are performed using the state 2. The purpose is to find out that how much data retains in the RAM when the RAM is not using our particular file. Before we began our experiments we checked if the whole of the word documents exits in RAM or some pieces or chunks are just loaded in RAM. For this purpose we manually constructed the whole word document from RAM. We rearranged all the fragments extracted from the RAM together than we were able to open to the word document. Hence proving our assumption that the whole document is loaded in the RAM.

5.1.1 Experiment 1

For our very first experiment we work with $S1$. We open this word file and take a memory dump using the DumpIt tool. First of all we need the components of the word document for the extraction of the textual contents for this we start analyzing the whole memory dump and look for our header. After finding the header we look for our remaining components. After we find the components we add all these to a pool of files we create.

Algorithm 1 Extraction of Components
Find the components of the word file

```
1: Input: A memory Dump
2: Result: A dataset of word components
3: Data: Creates simple text files
4: Find the header
5: Match the header with the version of Word Doc
6: Find the [Content_Type.xml] and store it as Obj
7: Find the Rels.xml and store it as Obj
8: Find the document.xml and store it as Obj
9: Create a pool of Objs
```

Then we further analyze these pool of files and differentiate our components. The components of similar types are grouped together. From all of these components the textual content is extracted and decode, the document.xml is then used for finding the actual text of the word document. This text is checked for errors and saved in a text file. Since we have just used a single file so we don't need to use any clustering technique in this experiment. Algorithm 1 shows the process of extraction of the components of the word file.

Algorithm 2 Extraction of Textual Data
Find the textual data from the components

```
1: Input: A pool of Objs
2: Result: A dataset of textual components
3: Data: Creates different sets of components.
4:  Analyze Objs
5: Differentiate the components
6: For each Obj ∈ S1
7:    If Obj has textual data then
8:          Extract its textual data
9:          If Data is encoded then
10:             Decode its textual data
11:         end
12:             if Fix missing spaces in text;
13:                 Fix spelling mistakes in text;
14:                 Save the text in a separate text file
15:                 Add the resulting file to pool of data;
16:             end if
17:   end if
```

5.1.2 Experiment 2

For experiment 2 we take the data set $S2$ which contains 5 Word files taken at random from our system. Since in this experiment we have more than one file so we will need to classify the components i.e. which component belongs to which file. We used the two most commonly used text clustering Algorithms available and the Algorithm 4 shows these techniques and we tested each of these techniques using the precision metrics we discussed earlier, since now we have more than one text document, we would need to perform some preprocessing on the textual contents before we can actually perform our Machine Learning Algorithms on it. This process is known as Preprocessing and is shown below:

Algorithm 3 Preprocessing of Textual Data
Preparing the textual data for the Machine Learning Algorithm.

```
1: Input: A pool of Textual data
2: Result: A reduced dataset of textual components.
3: Data: Unique textual Content
4: Collecting all textual instances into a Corpus
5: Tokenizing it
6: Making of vocabulary from Corpus
7: Making Vectors of Corpus
8: Reduction of Stop words
9: Scoring of Vectors
10: Finding Document Frequency DF
11: Finding TF-IDT
```

After the preprocessing is done the ending result is textual content which is unique in every textual instance. The preprocessing reduces the processing time by removing repetitive or common words like stop words, like is, am and the etc., which are mandatory to appear in every document but have no significant meaning that contribute towards the document. So removing and sorting our text is a very crucial step and can lead to drastic changes in results. After preprocessing we perform the Text Clustering, The Algorithm 4 basically shows the whole process behind the Text Clustering. We performed both K mean and Hierarchal Clustering in our experiments, because we were not sure which algorithm could provide us with better results.

Algorithm 4 Classification of Objects
Classify each component using Text Clustering

```
1: Input: A dataset of textual components
2: Result: Micro Average precision and Recall
3: Data: Sets of textual components
4: Apply Algorithm 1
5: Apply Algorithm 2
6: Apply Algorithm 3
7: Apply the K Means Text Clustering algorithm on pool
8: For each Cluster do
9:        Find the Macro Average Precision
10:       Find the Macro Average Recall
11:    end for
12: Apply the Hierarchical Text Clustering on pool
13: For each Cluster do
14:       Find the Macro Average Precision
15:       Find the Macro Average Recall
16:    end for
```

5.1.3 Experiment $3, 4 and 5$

The other experiments $3, 4 and 5$ were carried out using the subsets $S3$, $S4$ and $S5$ respectively, these contain a total of 30 files. There are no practical real life scenarios where there is a need of opening 30 or so files at once. These experiments were carried out just to check how well the clustering algorithm works and the impact of size of the data set on the precision of our technique.

5.2 RESULTS

The following section presents the results obtained from the experiments performed above. The results are divided into two parts. The first part contains results when the file was read or viewed. The second part contains results when the file was not being viewed.

5.2.1 When File was being Viewed

As we know from above experiments that the first thing we need are the XML components.The following table shows the number of components extracted from all the five sets of data that we have chosen. Point should be noted that there are almost 11 to 12 components in an XML file that contains vital information but since we are only focused on finding the content we would only extract 4 main components that would lead us to our content. This saves us from extra computation and processing.

Table 5.1: No. of Extracted XML components

DataSet	Number of XML components
$S1$	4
$S2$	20
$S3$	40
$S4$	60
$S5$	120

After receiving the XML components we decoded the XML components using ZIP decoding.The resulting decoding lead us to a number od textual files. Below is the number of Textual files generated from each data set. Number of text file generated form a single document mostly depended upon the number of lines in a document. But this cant be generalized for each document.

Table 5.2: No. of Textual Components

DataSet	Size	Number of Textual components
$S1$	12 KB	8
$S2$	76 KB	65
$S3$	172 KB	158
$S4$	696 KB	247
$S5$	976 KB	534

So After performing K mean clustering and Hierarchal Clustering we got the following performance metrics:

Table 5.3: PRECISION

Dataset	**Hierarchial Clustering**	**K-mean Clustering**
$S1$	0.998	0.767
$S2$	0.834	0.751
$S3$	0.910	0.655
$S4$	0.899	0.621
$S5$	0.821	0.600

The Table below shows the recall results :

Table 5.4: RECALL

Dataset	**Hierarchial Clustering**	**K-mean Clustering**
$S1$	0.978	0.802
$S2$	0.824	0.772
$S3$	0.870	0.733
$S4$	0.900	0.656
$S5$	0.868	0.627

The above results clearly indicate that the Hierarchial clustering algorithm performs better than the K mean algorithm.The range of precision for hierarchial clustering is from 0.821 to 0.998 and for k mean it is 0.600 and 0.767. Also the range for recall is 0.868 and 0.978 for hierarchial clustering and 0.637 and 0.802 for k mean.These number clearly show that which algorithm is the best.

The results are graphically shown as follows

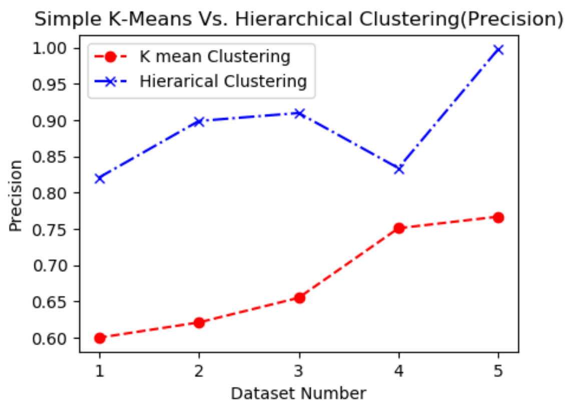


Figure 5.1: precision

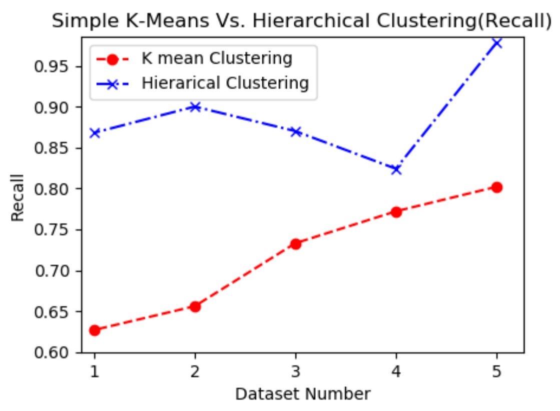


Figure 5.2: Recall

The final Table includes results for both scenarios, the one when the files are open and the one where the files are closed.We performed the experiment three times using the hierarchial clustering and averaged our results in the table below:

Table 5.5: Experiment performed right before the system is closed

Dataset	**Number of Objects**	**Scenario 1**	**Scenerio 2**	**Perc. carved**
$S1$	8	7.5	7	90.625
$S2$	65	61	58	91.538
$S3$	158	143	140	89.55
$S4$	247	222	217	88.866
$S5$	534	498	486	92.134
Avg	-	-	-	90.54

We can see that the percentage of carved files is 90.54 percent when the system is powered on.After that we perfrom the same experiment but after 10 mints of closing the system. The results are as follows:

Table 5.6: Experiment performed 10 mins after the system is closed

Dataset	**Number of Objects**	**Scenario 1**	**Scenerio 2**	**Perc. carved**
$S1$	8	6	5	68.75
$S2$	65	53	49	78.46
$S3$	158	137	129	84.177
$S4$	247	217	203	85.020
$S5$	534	483	465	88.7
Avg	-	-	-	81.021

The results of carving comes out to be 81.021 perc after 10 mints.Now we perform the same experiment after 30 minutes and the result is as follows:

Table 5.7: Experiment performed 30 mins after when the system was closed

Dataset	**Number of Objects**	**Scenario 1**	**Scenerio 2**	**Perc. carved**
$S1$	8	4	3	43.75
$S2$	65	39	30	53.07
$S3$	158	98	79	56.012
$S4$	247	152	138	58.706
$S5$	534	344	299	60.20
Avg	-	-	-	54.3476

Above are the results obtained by carving of Microsoft word document objects from RAM using 30 word files.We have considered two scenarios, one when the document is being viewed or open and second when the document is closed. Further we performed the same experiment thrice with some time difference.It can be seen that the number of textual content extracted has slight difference when the document is open or close. But the percentage of the carving decrease on whole when the time is passed.This is mainly because the RAM being a temporary memory stores the state of every function being performed so some data might be lost or overwritten with time.

When we performed the experiment after 30 mints than the percentage of carved files became 54.37 percent. So we can say in our case that the Digital Investigator has almost 50 percent chance of correctly carving the Microsoft Word Document from RAM. Also we have shown experimentally that the better Algorithm for separating carved textual content from RAM is the hierarchal Clustering.The total number of textual contents of all 30 word files is 534 and the size of the carved textual files from RAM range in size from $1KB$ to $8KB$.

This extraction technique is useful for the Digital Investigators and has shown promising results, but there are a few factors that resulted in such good results.One thing is the size of the data set,our data set is very small and has very small number of textual content that needed classification. Using a bigger data set would be complex and would certainly reduce the performance. Secondly the Word files were not choosen at random, the files were chosen in a way that they include different themes so that it becomes easier in classification because we would have distinct data with lesser similarity between content.

Chapter 6

CONCLUSION and FUTUREWORK DIRECTIONS

6.1 CONCLUSION

The rise in the digital crimes these days lays emphasis on the digital forensics in solving crimes.Proof extracted from digital machines is enough for proving a criminal guilty.While extracting these proofs from the system a crucial place for extraction is the RAM because it hold the current state about the system.So in cases where the meta data of the file is not present so extraction of file from RAM is a very challenging task because files are placed randomly in a RAM frames.Now if we carve these randomly placed instances separately then they would mean nothing in the court and wont be considered a valid proof.Our research enhances the technique of carving a specific type of file by clustering chunks of similar data together, based on similar features.In our research we use multiple files and extracted their chunks using the two most common clustering techniques i.e. K mean and Hierarchical clustering. Our results show that when it comes to clustering textual content of a word file then the hieratical clustering out performs the simple K-mean algorithm.Also when the experiments are repeated after some time than due to the change in the dynamics of the RAM, the performance of our technique reduces. This is mainly because the chunks no longer reside in the RAM or are scattered discontinuously across multiple pages in RAM.Hence the use of clustering technique for extraction of word file from RAM reduces the burden of the digital investigator ,which otherwise would have to go through each and every word document available on the system in order to find the concerned document.

The focus of this research is to study the extraction of multiple formats from the main memory using multiple tools and methods.It also emphasizes on analyzing MS Word document file format from craving point of view.For future research,it can be concluded from above discussion that the recovery of MS Word document from volatile memory when the document is fragmented is a promising field to explore for researchers and forensic investigators.

From the above discussion we have concluded that as future research,the recovery of Microsoft word document from volatile memory when the document is fragmented seems like a promising field to explore for forensic investigators and researchers

6.2 FUTURE DIRECTIONS

In this section, proposals for the future work are provided.

- One important aspect that should be explored is the extraction of Microsoft Word Document when we have the same type of content in two different Microsoft Word Document. Like same text or type of text originating from multiple sources.We did not go into this regime because it became a very tough text classification problem, each word document would become a class and we would have to find which chunk is a part of which class.

- Another thing is the extraction of images from the Microsoft Word Document, in order to keep our work simple, we ignored the images used in the Microsoft Word Document. The data set we choose intentionally had documents only containing the textual contents.But the images can be extracted in the same way by choosing the Open Office XML component that contains the information about the images used.So for future work we would like to extend our methodology to include the non textual data as well.

- We have just discussed the carving of Word Document from the RAM for simplicity. Same procedure can be performed on the main disk but it would require more resources and power because we would have to search through the whole disk many times for finding the components of the word documents.

- The current focus of our work included the investigations performed on a normal desktop environment, with limited number of users and smaller resources available. However in future we would like to check the applicability of our technique on a cloud environment with more number of users and complexity.

BIBLIOGRAPHY

[1] Beek, Christiaan. "Introduction to file carving." White paper. McAfee (2011).

[2] Garfinkel, Simson L. "Carving contiguous and fragmented files with fast object validation." digital investigation 4 (2007): 2-12.

[3] Sportiello, Luigi, and Stefano Zanero. "Context-based file block classification." IFIP International Conference on Digital Forensics. Springer, Berlin, Heidelberg, 2012.

[4] Cohen, Michael I. "Advanced carving techniques." Digital Investigation 4.3-4 (2007): 119-128.

[5] Wei Lin and Ming Xia. Advanced Materials Research Vols. 433-440 (2012) pp 3028-3032, 2012

[6] Azzat AI-Sadi, Manaf Bin Yahya, Ahmad Almulhem. “Identification of image fragments for file carving” . Proposed in World Congress on Internet Security (WorldCIS-2013)

[7] Deokar, Sanjivani Tushar. "Text documents clustering using k means algorithm." International Journal of Technology and Engineering Science [IJTES] TM 1.4 (2013): 282-286.

[8] Jing, Liping, et al. "Subspace clustering of text documents with feature weighting k-means algorithm." Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, Berlin, Heidelberg, 2005.

[9] Mythili, S., and E. Madhiya. "An analysis on clustering algorithms in data mining." International Journal of Computer Science and Mobile Computing 3.1 (2014): 334-340.

[10] Guyon, Isabelle, et al., eds. Feature extraction: foundations and applications. Vol. 207. Springer, 2008.

[11] Goldberg, Yoav. "Neural network methods for natural language processing." Synthesis Lectures on Human Language Technologies 10.1 (2017): 1-309

[12] Umer, Muhammad Fahad, and M. Sikander Hayat Khiyal. "Classification of textual documents using learning vector quantization." Information Technology Journal 6.1 (2007): 154-159.

[13] Martin, James H., and Daniel Jurafsky. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Upper Saddle River: Pearson/Prentice Hall, 2009.

[14] Compound file format introduction.

[15] Martin, James H., and Daniel Jurafsky. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Upper Saddle River: Pearson/Prentice Hall, 2009.

[16] Source: https://www.howtogeek.com/304622/what-is-a-.docx-file-and-how-is-it-different-from-a-.doc-file-in-microsoft-word

[17] Richard III, Golden G., and Vassil Roussev. "Scalpel: A Frugal, High Performance File Carver." DFRWS. 2005.

[18] Foremost 1.53 [Online]. Available: http://foremost.sourceforge.net

[19] Cohen, Michael I. "Advanced carving techniques." Digital Investigation 4.3-4 (2007): 119-128.

[20] Roux, Brian. "Reconstructing Textual File Fragments Using Unsupervised Machine Learning Techniques." (2008).

[21] Pal, Anandabrata, and Nasir Memon. "The evolution of file carving." IEEE signal processing magazine 26.2 (2009): 59-71.

[22] Zha, Xinyan, and Sartaj Sahni. "Fast in-place file carving for digital forensics." International Conference on Forensics in Telecommunications, Information, and Multimedia. Springer, Berlin, Heidelberg, 2010.

[23] Poisel, Rainer, Simon Tjoa, and Paul Tavolato. "Advanced file carving approaches for multimedia files." JoWUA 2.4 (2011): 42-58.

[24] Wei Lin and Ming Xia. Advanced Materials Research Vols. 433-440 (2012) pp 3028-3032, 2012

[25] Al-Sadi, Azzat, Manaf Bin Yahya, and Ahmad Almulhem. "Identification of image fragments for file carving." World Congress on Internet Security (WorldCIS-2013). IEEE, 2013.

[26] Roussev, Vassil, and Simson L. Garfinkel. "File fragment classification-the case for specialized approaches." 2009 Fourth international IEEE workshop on systematic approaches to digital forensic engineering. IEEE, 2009.

[27] Al-Sharif, Ziad A., Hasan Bagci, and Aseel Asad. "Towards the Memory Forensics of MS Word Documents." Information Technology-New Generations. Springer, Cham, 2018. 179-185.

[28] Garfinkel, Simson L., and Michael McCarrin. "Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb." Digital Investigation 14 (2015): S95-S105.

[29] Wagner, James, Alexander Rasin, and Jonathan Grier. "Database forensic analysis through internal structure carving." Digital Investigation 14 (2015): S106-S115.

[30] Aljaedi, Amer, et al. "Comparative analysis of volatile memory forensics: live response vs. memory imaging." 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing. IEEE, 2011.

[31] Walters, Aaron, and Nick L. Petroni. "Volatools: integrating volatile memory into the digital investigation process." Black Hat DC 2007 (2007): 1-18.

[32] Calhoun, William C., and Drue Coles. "Predicting the types of file fragments." digital investigation 5 (2008): S14-S20.

[33] Sencar, Husrev T., and Nasir Memon. "Identification and recovery of JPEG files with missing fragments." digital investigation 6 (2009): S88-S98.

[34] Amirani, Mehdi Chehel, Mohsen Toorani, and Sara Mihandoost. "Feature-based Type Identification of File Fragments." Security and Communication Networks 6.1 (2013): 115-128.

[35] Li, Binglong, Qingxian Wang, and Junyong Luo. "Forensic analysis of document fragment based on SVM." 2006 International Conference on Intelligent Information Hiding and Multimedia. IEEE, 2006.

[36] Zha, Xinyan, and Sartaj Sahni. "Fast in-place file carving for digital forensics." International Conference on Forensics in Telecommunications, Information, and Multimedia. Springer, Berlin, Heidelberg, 2010.

[37] Alherbawi, Nadeem, Zarina Shukur, and Rossilawati Sulaiman. "Systematic literature review on data carving in digital forensic." Procedia Technology 11 (2013): 86-92.

[38] Kwon, Hyukdon, et al. "A tool for the detection of hidden data in microsoft compound document file format." 2008 International Conference on Information Science and Security (ICISS 2008). IEEE, 2008.

[39] Ahmed, Irfan, et al. "Fast content-based file type identification." IFIP International Conference on Digital Forensics. Springer, Berlin, Heidelberg, 2011.

[40] Suominen, Hanna. "Performance evaluation measures for text mining." Handbook of research on text and web mining technologies. IGI Global, 2009. 724-747

[41] Buckland, Michael, and Fredric Gey. "The relationship between recall and precision." Journal of the American society for information science 45.1 (1994): 12-19.