ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

UDC 004.633 **doi: 10.32620/reks.2023.1.14**

Maksym BOIKO[1,2], Viacheslav MOSKALENKO[1,3]

[1] ***Sumy State University, Sumy, Ukraine***
[2] ***The National Anti-corruption Bureau of Ukraine, Kyiv, Ukraine***
[3] ***National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine***

SYNTACTICAL METHOD FOR RECONSTRUCTING HIGHLY FRAGMENTED OOXML FILES

A common task in computer forensics is to recover files that lack file system metadata. In the case of searching for file fragments in unallocated space, file carving is the most often used method, which is ideal for unfragmented data. However, such methods and the tools based on them are ineffective for recovering OOXML files with a high fragmentation level. These methods do not provide reliable determination of the correct order of fragments. Techniques for reconstructing documents based on the analysis of words and phrases are also ineffective in fragmented OOXML documents. The main reason is that OOXML files are ZIP archives and, as a result, store data on disk space in a compressed form. This paper proposes a syntactical method for reconstructing OOXML documents based on knowledge about the internal structure of this file type, regardless of their content. The details of the implementation of the reconstruction algorithm and the peculiarities of restoring certain types of local elements of the document were considered. The efficiency of the algorithm was tested on the Govdocs1 and NapierOne datasets. The proposed method was applied to 4096-byte data blocks, which correspond to the standard cluster size in different file systems. The experimental results confirmed the method's suitability for practical use with 82.97 % of recovered files, including 34.38 % reconstructed completely, 0.43 % excluding the last 21 bytes at most, and another 48.16 % excluding embeddings that require other approaches. In the latter case, obtaining a fully working document without displaying graphic images and other contents of different embeddings is possible. The presence in OOXML files of CRC-32 hashes of the uncompressed data stream of each local element allows us to confirm the correctness of information recovery and its integrity unambiguously. Simultaneously, the method's effectiveness depends mainly on data verification methods during the reconstruction of local elements that occupy at least three clusters in the file. Therefore, this method is supposed to be improved by developing new mechanisms for verifying XML elements.

Keywords: *digital forensics; data recovery; file carving; syntactical file carving; fragmentation; file reconstruction; Office Open XML; OOXML; DOCX file; ZIP archive; DEFLATE compression.*

1. Introduction

1.1 Motivation of research

When investigating economic crimes, computer forensic experts often face the issue of searching for and further examining electronic documents and their draft versions, the circumstances, and chronology of their creation, etc. Only one workstation of an office worker can have tens of thousands of available text files, one of the most common types of which are OOXML documents. The amount and nature of the deleted information depend on the features and procedure of the user's actions when working with data, operating system parameters, and the type of storage medium used, etc. As a rule, the most significant difficulties occur when searching for deleted files without file system metadata associated with the data blocks being analyzed.

There are many available data recovery techniques [2 – 4]. The most common and straightforward of the existing methods is searching for signatures of files' beginning and end. This method can be classified as general since it identifies a wide range of complex file types, including OOXML documents. However, many difficulties occur when working with fragmented data using this method.

Along with partially overwritten and/or damaged data, the most problematic situations occur when data blocks are out-of-order fragmented. Some studies [5] estimate the possible level of fragmentation of Microsoft Office text documents in NTFS file systems on workstations running Windows operating systems. Therefore, approximately 51 % of the DOCX files were split into three or more fragments. Simultaneously, 72 % of these documents had data blocks located on the disk space in a non-sequential order.

The techniques used in such circumstances are generally reduced to two separate tasks: identifying data blocks by type and further reconstruction based on the file content and/or its internal structure.

Although OOXML documents contain textual information, they are de facto ZIP archives. Therefore, analyzing the text in raw data blocks is impossible without preliminary processing. As a result, context-based statistical models [6 – 9], or their modifications are not very applicable when reconstructing files of this type.

On the other hand, even small damage up to 4096 bytes in the middle of a compressed bitstream led to at least an inaccurate reconstruction of the original texts [10]. As a result, incorrect and/or incomplete reassembly of OOXML file fragments into a single whole leads to partial and, in the worst cases, completes loss of access to the contents of the recovered document.

1.2. State of the Art

Researchers [11, 12] consider the problem of searching uncompressed data fragments containing the texts of DOCX documents in RAM. Thus, a memory dump was studied in [11], where the search was performed using keywords obtained from the previously unzipped internal contents of the OOXML file. The results of the experiments depended on the user's actions with the files. In the best case, extracting about 20.18 % of blocks with text data was possible. In [12], available parts of DOCX files were found by searching tags between which this data can be placed in the "document.xml" element. This method retrieved an average of 40.4 % of the textual content of documents.

In contrast to previous studies, the study [13] was conducted with compressed data using limited information about the internal structure of the OOXML package. Here, the authors searched for clusters containing the beginning of the "document.xml" element and then applied unsupervised learning techniques. On average, they achieved 54.35 % to 90.54 % of recovered documents for different input data. However, it seems that in this work, the authors did not consider the case of fragmenting data and limited themselves to the first part of the "document.xml" element without searching for other data blocks.

Some researchers [14] have developed docs' text recognition software that can work with corrupt the pictured texts. However, the proposed methodology does not solve the problem of recovering compressed text data.

The general concept of document reconstruction is presented in [9]. However, despite the prospects of recovering full texts and obtaining a wide range of other forensically important data, the issue of OOXML file reconstruction is not sufficiently studied. Existing works mostly ignore it. This is primarily because on disk space, an OOXML document can look like a set of fragments related to different types of data, primarily media files. As a rule, they are already compressed and stored in their formats, such as JPEG [15], GIF [16, 17], and BMP [18] and their different modifications [19, 20].

However, it is worth noting that works in related fields are devoted to a syntactical approach to analyzing file fragments [8, 21, 25]. This approach can be used to analyze the content of a specific object by using knowledge about its internal structure. An example of this approach is the reconstruction of JPEG, JPG, PNG, BMP graphic files [8, 21, 22, 23, 24, 25, 26], SQLite databases [27], DOC files [28], other graphics [29, 30], and text data [8, 31], etc.

Thus, using information about the internal structure and content of OOXML files for their reconstruction is a promising but little-studied approach.

1.3. The purpose and tasks of research

The study develops a syntactical method for reconstructing OOXML documents based on knowledge about the internal structure of ZIP archives, the internal structure of XML elements of the Microsoft Office package, and the features of the Deflate compression algorithm. This paper does not consider the recovery of embedded data in OOXML documents (for example, graphic images).

To achieve this goal, we must solve the following tasks:

- identify the key elements of the OOXML package as a ZIP archive to specify its fragments;
- develop a syntactical method for restoring an OOXML document without considering embeddings;
- analyze the effectiveness of the developed method on publicly available datasets.

The main contribution of the researched method is an approach to recovering highly fragmented OOXML files from unallocated space and RAM. Also, this method allows to achieve access to the partial texts of damaged OOXML documents, their internal metadata, etc.

Structurally, the paper consists of the following sections. An analysis of the OOXML file structure and a description of the syntactical method for reconstruction are presented in Section 2. Section 3 describes the datasets used to evaluate the method's effectiveness and provides the analysis. Section 4 contains a discussion of the obtained results. The last section provides the conclusions of the paper and directions for future research.

2. Syntactical method for reconstructing OOXML files

2.1. Analysis of OOXML files structure

Since the 90s, the main format of Microsoft Office has become binary files with the standard extensions *.doc, *.xls, *.ppt. However, starting with Microsoft Office 2007, XML-based files – Office Open XML

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

documents – began to be used by default [32]. The last ones had the standard extensions *.docx, *.xlsx, *.pptx.

In 2006, the Ecma International – European association for standardizing information and communication systems – adopted the OOXML files format as the Ecma-376 standard. In 2008, the specification was approved by the International Organization for Standardization and the International Electrotechnical Commission as ISO/IEC:29500.

Office Open XML documents are a regular ZIP archive that consists of a number of related elements. Such a ZIP archive is also called a package. It has a similar structure to Microsoft Office documents, spreadsheets, and presentations that differs in that the majority of their content is stored in XML elements of various structures in directories named "word", "xl", and "ppt", respectively. In the following, we will use examples based on Microsoft Word documents.

Fig. 1 shows the typical internal structure of a DOCX file, where [32]:

- "[Content_Types].xml" file – a content-type item that describes the content stored in the package elements, including media type, subtype, and other optional parameters;

- other XML files – elements that contain different parts of the document, such as main document story, properties, headers, footers, comments, metadata, etc. From the forensic viewpoint, a brief description of the most important elements of the OOXML package is given in Table 1;

- RELS files – package-relationship ZIP items that indicate the types of relationship between the initial and final parts of the package. These items do not affect the content of other elements.

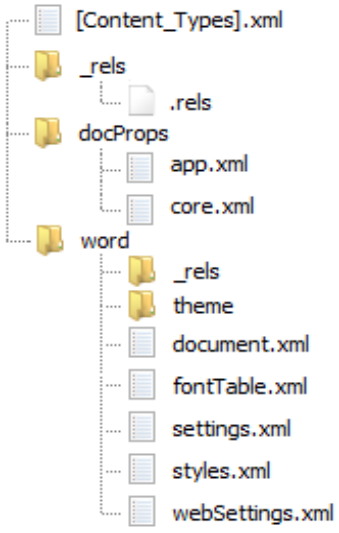

Fig. 1. Typical contents of a DOCX package

Simultaneously, each above elemen has its own clearly defined standard structure [32] and contains information that may be important in resolving issues occurring in computer forensics [33, 34].

Table 1

Descriptions of some XML elements

Title	Description
app.xml	Extended file metadata such as total editing time, hyperlinks, the number of pages, words, characters, etc.
core.xml	Basic file metadata such as the name of the document's author, the username who last modified it; content created, last modified, and last printed timestamps, etc.
document.xml	Main content of the document
comments.xml	Information about each comment, its author, date and time of its creation
people.xml	Contact information for each person who authored at least one comment or revision in the document
settings.xml	All document properties, including those that allow establishing links between different versions of files

2.2. Analysis of the structure of OOXML files as ZIP archives

The ZIP file format was introduced in 1989 by PKWARE. Details of the specification and a general description of the internal structure of this file type are provided by the developer in the form of application notes [35], which are periodically updated.

From the viewpoint of the internal structure of an OOXML document as a ZIP archive [32], it starts with a sequence of local file headers and file data. Each element of the archive has its header with its metadata. In some cases, descriptors may follow the file data. There are records of the central directory at the end of the archive file. They contain a copy of the metadata for each local element and its location, etc. All of this is summarized by the last record, which contains metadata of the central directory and information about its location. A simplified structure of a ZIP archive is shown in Fig. 2.

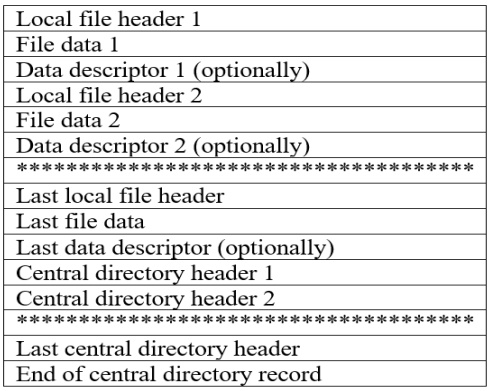

Fig. 2. The structure of a ZIP file

Additionally, each mentioned above element of the archive has its own signature [35]:

- local file header - 0x504b0304;
- central directory header - 0x504b0102;
- end of the central directory - 0x504b0506;
- descriptor - 0x504b0708.

2.3. Deflate compression method

Local elements of an OOXML file can be stored in an archive with or without compression [32]. As a rule, PNG, JPG, JPEG, GIF, TIFF files, XLSX tables, PPTX presentations, and other media data remain uncompressed. Simultaneously, XML, RELS elements, and embeddings in the form of BIN, EMF, WMF, PPT, DOC, XLS, DTTF, and DOCX files are stored in a compressed state.

Deflate is a typical compression method used in OOXML documents [32, 36]. One of the main features of this algorithm in the context of this work is the ability to restore part of the compressed data before the point of stream corruption [37].

2.4. Proposed reconstruction method

The proposed method is based on the use of knowledge about the internal structure of OOXML files, in particular:

- the internal structure of ZIP archives – to determine the locations of key clusters of an OOXML document and verify data. Every local element has a unique signature, detailed information on its exact location from the beginning of the file, and other metadata. Mentioned data are contained in the central directory, whose records, in turn, have their own signature and occupy the last clusters of the file;
- features of the deflate compression algorithm – to find the potential next cluster in the chain. Such clusters can be detected by appending their data to the end of the damaged compressed stream and the absence of errors during the subsequent decompressing of the resulting fragment;
- the internal structure of individual XML elements of the OOXML package – to filter out false positives. This is possible by comparing the decompressed stream with the typical structure of these elements.

The algorithm for reconstructing OOXML files consists of the following steps:

1. Search for clusters containing the central directory data and local headers.
2. Determining the correct location of the detected clusters.
3. Sequential iterative – by searching the cluster by cluster – determination of the location of other data blocks with their verification.

1) At the first stage, all clusters are actually divided into three groups, for which:

- searching for clusters containing signatures of central directory headers and its end – 0x504b0102 and 0x504b0506, respectively. These clusters are the last clusters of the file and contain detailed information about the location of local elements;
- searching for clusters containing signatures of local file headers – 0x504b0304. These clusters contain comprehensive information about each local element and the initial part of the compressed data.

2) At the next stage is to place the clusters with the central directory records in the correct order; for this purpose is to find the first record of the central directory in each detected cluster and to determine the relative offset of the local header in its offsets 42-45. Subsequently, the clusters are arranged in an ascending order of the detected indicators.

If the central directory is correctly restored, in the vast majority of cases, the first entry will contain data on the "[Content_Types].xml" element with an offset value of 0x00000000. All other elements will have successively increasing local offsets. Simultaneously, there may be extreme cases when the headers of the first central directory entry and/or its end (footer) are divided between clusters. These missing clusters can be found separately in this situation, but this is uncritical for file reconstruction.

Fig. 3 shows an example of a central directory, where its first entry and end are highlighted in red, and consistently increasing offsets of local headers and the central directory itself relative to the beginning of the file are highlighted in blue. The latter values are stored in Little-Endian byte order.

After reconstructing the central directory, it is necessary to analyze its entries and determine the sequence of local elements, their names, information about their compression type, compressed data sizes, CRC-32 hashes, the offset of the local element header relative to the beginning of the file, etc. After that, based on the offsets of local file headers, it is necessary to determine the location of each cluster where the local elements begin (Fig. 4) and set all unfilled cluster chains – their beginning, length, and end (Fig. 5).

3) In the third stage, the local elements of large sizes, namely their compressed data, are reconstructed separately one by one. For example, Fig. 6 shows a structure of the local header, additional field, and compressed content of a typical first element of an OOXML document named "[Content_Types].xml".

Depending on the number of unknown clusters, two similar algorithms are used separately for each local element, differing only at the data validation phase.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

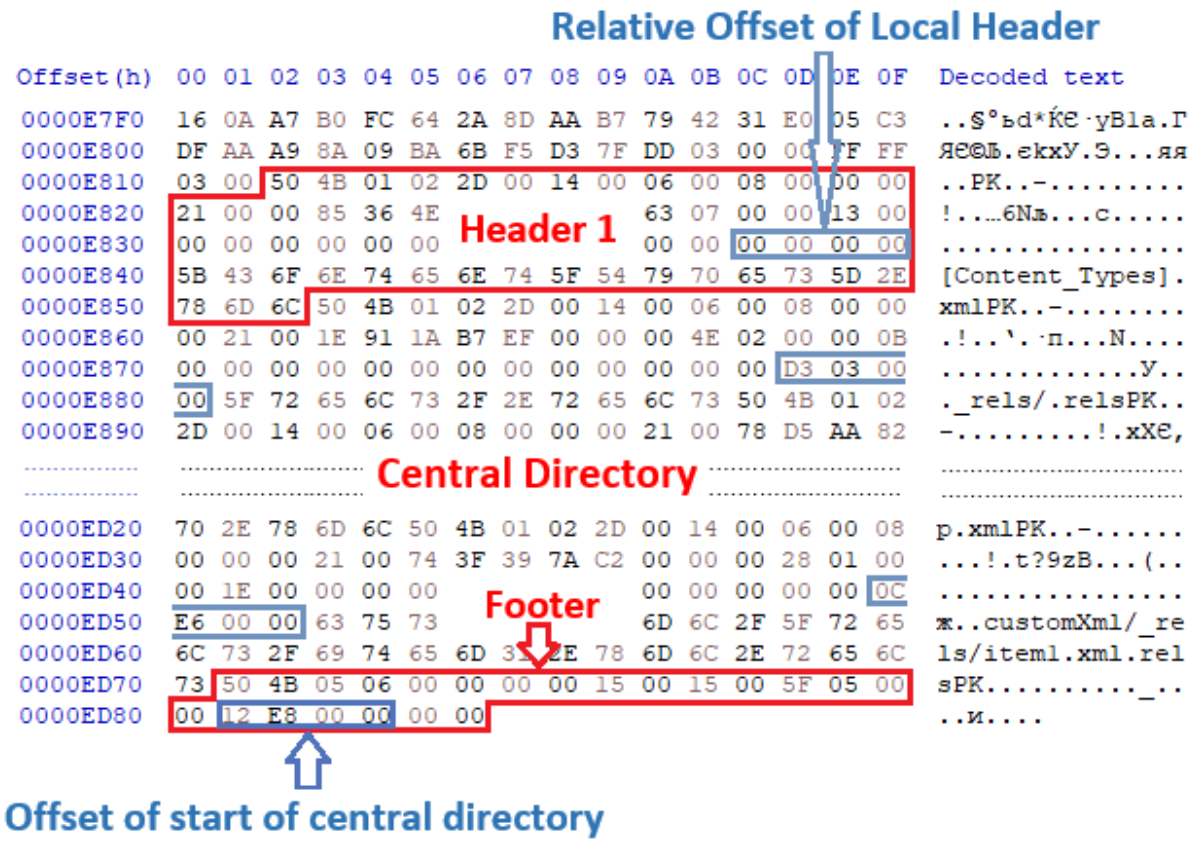

Fig. 3 Example of a central directory for a DOCX document

	Compression type		Offset	Cluster	File name
0	Deflated	-	0x0	0	b'[Content_Types].xml'
1	Deflated	-	0x3d3	0	b'_rels/.rels'
2	Deflated	-	0x6f3	0	b'word/document.xml'
3	Deflated	-	0x44cb	4	b'word/_rels/document.xml.rels'
4	Deflated	-	0x5816	5	b'word/footnotes.xml'
5	Deflated	-	0x5b02	5	b'word/endnotes.xml'
6	No compression	-	0x5dee	5	b'word/media/image1.png'
7	No compression	-	0x7311	7	b'word/media/image2.png'
8	No compression	-	0x813d	8	b'word/media/image3.png'
9	No compression	-	0xa042	10	b'word/media/image4.png'
10	Deflated	-	0xabfc	10	b'word/theme/theme1.xml'
11	Deflated	-	0xb270	11	b'word/settings.xml'
12	Deflated	-	0xc247	12	b'customXml/item1.xml'
13	Deflated	-	0xc367	12	b'customXml/itemProps1.xml'
14	Deflated	-	0xc4a7	12	b'word/numbering.xml'
15	Deflated	-	0xcb51	12	b'word/styles.xml'
16	Deflated	-	0xdada	13	b'word/webSettings.xml'
17	Deflated	-	0xde04	13	b'word/fontTable.xml'
18	Deflated	-	0xe043	14	b'docProps/core.xml'
19	Deflated	-	0xe2e4	14	b'docProps/app.xml'
20	Deflated	-	0xe60c	14	b'customXml/_rels/item1.xml.rels'

Fig. 4 Information about the initial clusters of local elements

Fig. 5 The top row shows the clusters divided into three groups based on the results of the first stage (blue indicates clusters with local headers, orange - with records of the central directory, and white - clusters whose location needs to be determined); the bottom row shows their locations based on the results of the second stage

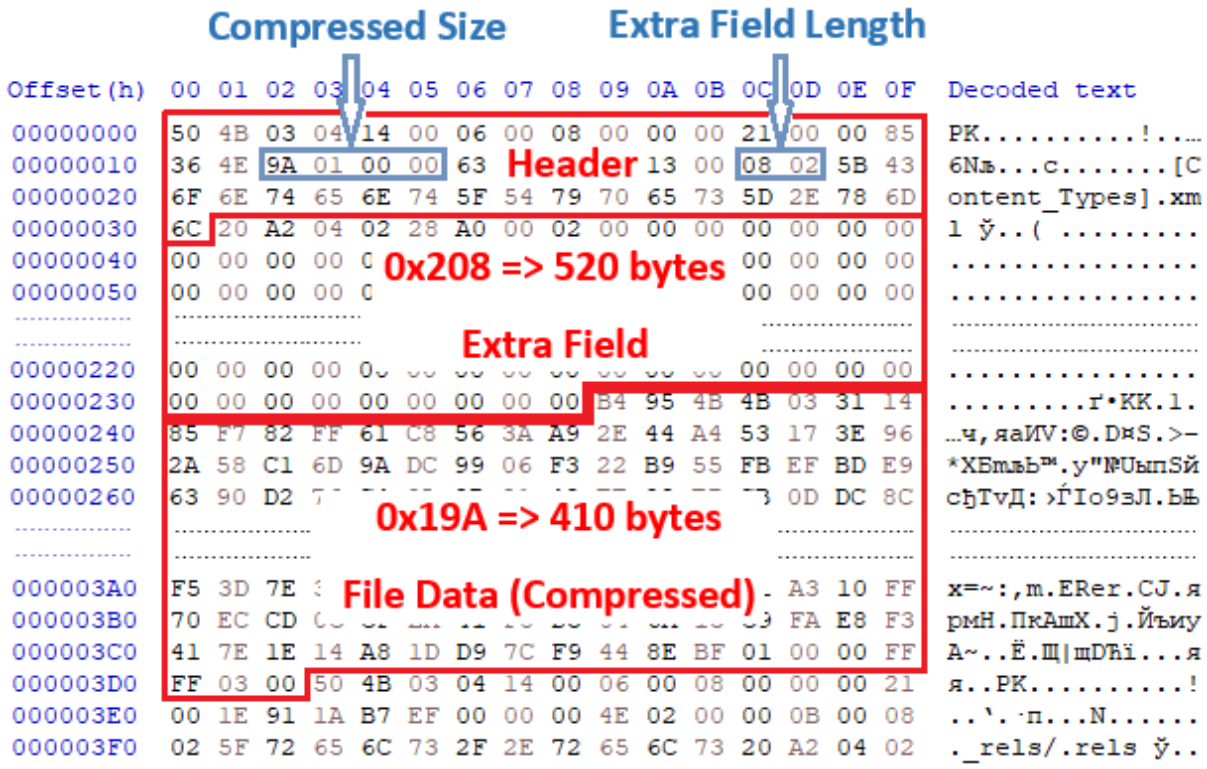

Fig. 6 Example of a structure of the local header, additional field,
and compressed content of a typical first element of an OOXML document

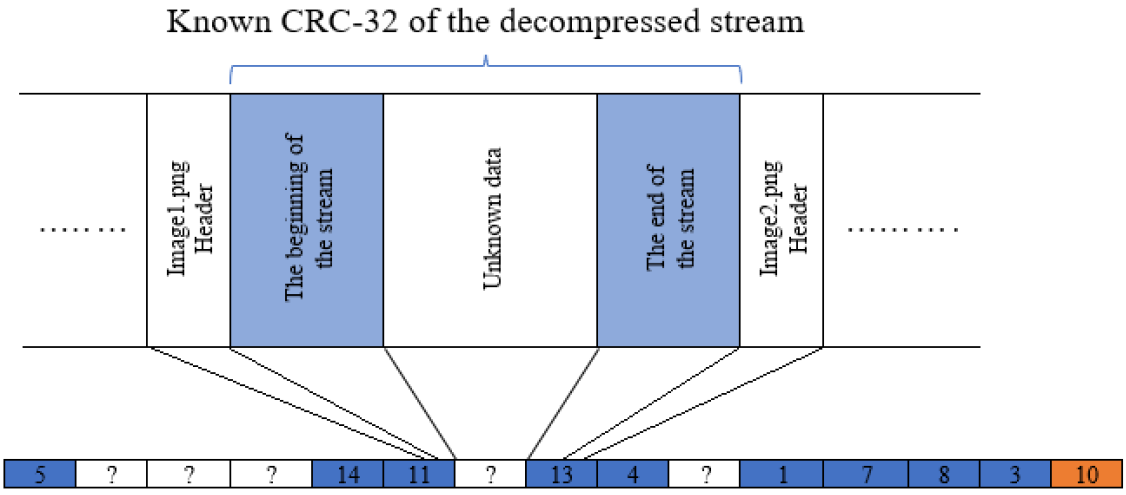

Fig. 7 Situation with one skipped cluster

Thus, when only one cluster remains unknown among the data of a local element, it is possible to define the skipped fragment by calculating the CRC-32 hashes of the uncompressed data stream of this local element. This case is simplified in Fig. 7.

Therefore, if the length of the unknown fragment is equal to one cluster, the following steps should be taken:

1) from the cluster preceding the unknown fragment, cut the last part starting from the beginning of the compressed data stream of the last local element present in this cluster;

2) add the data from the cluster whose location is unknown to the data obtained in step 1;

3) from the cluster that follows the unknown fragment, cut the first part of the data starting from the zero offset and ending with the local header of the first local element in this cluster, and then add this byte stream to the data obtained in step 2. Occasionally, the part of the data to be cut may end with the beginning of the descriptor (bytes '0x504b0708');

4) attempt to unzip the obtained compressed stream;

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

5) if the attempt fails, repeat the process from step 2 with the next cluster, the location of which is still undetermined;

6) in the case of a successful attempt to calculate the CRC-32 hashes of decompressed data;

7) if the CRC-32 hash value does not match the value specified in the central directory, repeat the process from step 2 with the next cluster, the location of which is still undetermined;

8) if the CRC-32 hash value matches the value specified in the central directory, the cluster search process is considered complete.

In other words, this is searched for a cluster for which there are no stream corruption points when filling the space between the beginning and end of the compressed data, and the uncompressed data has a CRC-32 hash value identical to that specified in the corresponding central directory entry, in the local element header and/or in the corresponding descriptor.

The logic of the search process is shown schematically in Fig. 8, and the input and the result are shown in Fig. 9.

Unlike the previous case, if there is a sequence of length k of unknown classifiers with their total number n, the number of possible combinations is n!/(n-k)!. It will tend to the value n^k for large values of n. In real cases, achieving a positive result for a full search task is unlikely and time-consuming.

In addition, OOXML documents belong to complex files and may contain embeddings of other data types, some of which are stored in uncompressed form. Such elements have not been studied in the context of this work and require the use of separate methods for their reconstruction.

Therefore, for local elements that occupy at least four clusters in a file, it is proposed to determine one next cluster in the chain at each step, for which the following actions should be performed (using XML files as an example):

1) from the cluster preceding the unknown fragment, cut the last part starting from the beginning of the compressed data stream of the last local element present in this cluster;

2) add the data from the cluster whose location is unknown to the data obtained in step 1;

3) attempt to unzip the obtained compressed stream;

4) if the attempt fails, repeat the process from step 2 with the next cluster, the location of which is still undetermined;

5) in the case of a successful attempt to compare the XML structure of the decompressed part of the data with the typical XML structure of the corresponding local element of the OOXML package;

6) in the case of a damaged XML structure, repeat the process from step 2 with the next cluster, the location of which is still undetermined;

7) in the case of a correct XML structure, the cluster search process is considered complete;

8) further repeat the search process until the last cluster in the chain is found;

9) add the data from all detected clusters to the data obtained in step 1, and add the first part of the data from the cluster that follows the end of the unknown fragment, starting from the zero offset and ending with the local header of the first element present in this cluster. Occasionally, the part of the data to be cut may end with the beginning of the descriptor (bytes '0x504b0708');

10) conduct verification by calculating the CRC-32 hash of decompressed data.

The logic of the search process is shown schematically in Fig. 10, and the input and result are shown in Fig. 11.

It is also possible that the header of a local element is located on the border of two clusters and is divided between them. This is not a problem because, in this case, according to the data of the central directory, it is necessary to sequentially calculate the offset of the local file header relative to the beginning of the archive and then to calculate the offset of the beginning of the compressed stream relative to the beginning of the cluster, and finally search for the corresponding data block.

If the last cluster of the local element is unknown, it is necessary to define the size of its compressed stream from the central directory, then determine the size of the last fragment, and use only the first part of the clusters of the corresponding length when searching.

It is worth paying more attention to the stage of verification of intermediate data, which is key when searching for clusters of local elements with two or more unknown fragments. Here, verifying the decompressed data by hashes is impossible, so the first step is to look for a cluster that may be the next in the chain. This uses a feature of the deflate algorithm when the decompressor can restore a part of the compressed local file to the beginning of the corrupt bitstream [37]. When another piece of information is added to the initial part of the compressed data, any bitstream is potentially valid [10]. As a result, these situations are possible when trying to decompress:

- an error;
- false positive decompressed data stream, where the first part of the content has a typical XML structure, and the last fragment contains an erroneous set of characters (Fig. 14);
- successfully unzipped data, where all the content has a typical XML structure (Fig. 15).

A option for verifying the content of the restored part of the local element to filter out false positives is to reconstruct the XML tree to its typical structure and check its integrity. This process is shown schematically in Fig. 12 on the example of the element "footnotes.xml",

the simplified structure of which is shown in Fig. 13. Simultaneously, it additionally checks for incrementally increasing identifiers specified in the attributes of the "w:footnote" element. Generally, the essence of this process is the same for all other elements of the OOXML package but differs in detail depending on the complexity of the internal structure of the XML tree.

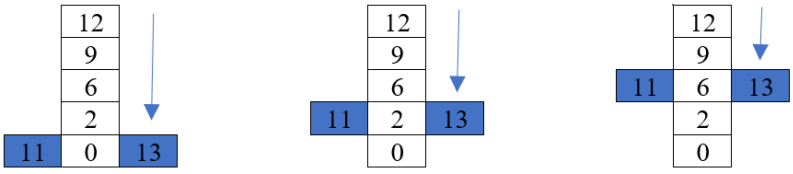

Fig. 8 The clusters marked in white are checked for compliance one by one

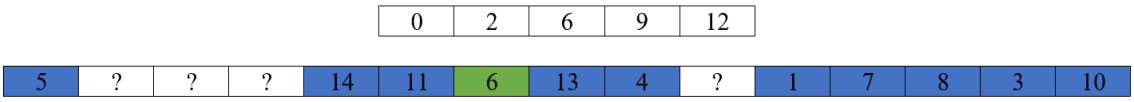

Fig. 9 The top row shows the clusters whose location was unknown, and the bottom row shows their location according to the interim results of the third stage (clusters with local headers and central directory records are highlighted in blue, and clusters whose location has been determined are highlighted in green)

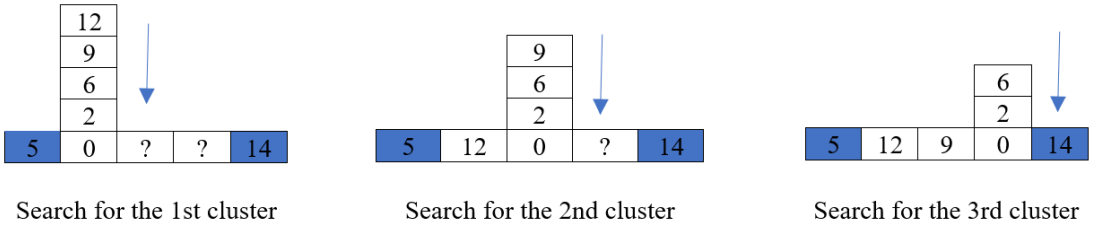

Fig. 10 The clusters marked in white are checked for compliance one by one

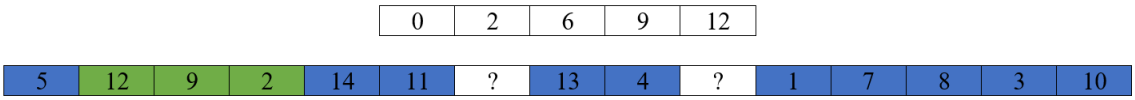

Fig. 11 The top row shows the clusters whose location was unknown, and the bottom row shows their location according to the interim results of the third stage (clusters with local headers and central directory records are highlighted in blue, clusters whose location has been determined are highlighted in green)

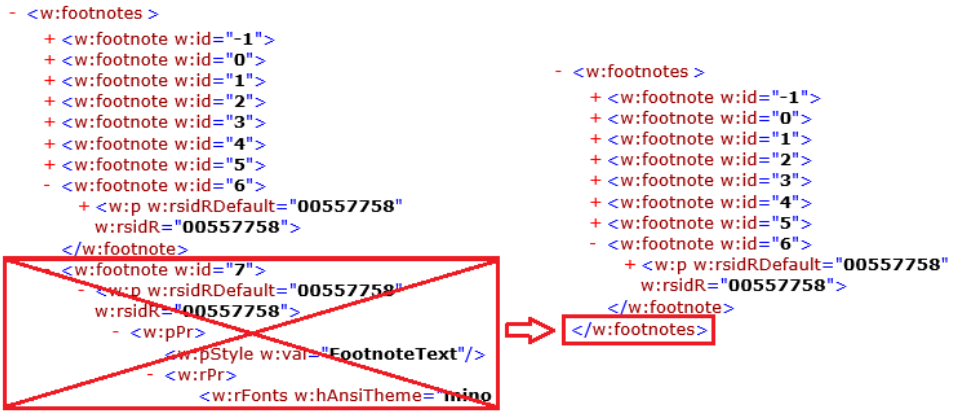

Fig. 12 Data manipulation before verification

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

```
<w:footnotes>
    <w:footnote w:id="##">
        <w:p>
            ********
            ********
        </w:p>
    </w:footnote>
</w:footnotes>
```

Fig. 13 Simplified internal structure of the "footnotes.xml" element

Fig. 14 An example of a false positive result during decompression

Fig. 15 An example of successful decompression

3. Results of experiments

3.1. Preparing data for testing

The input is a DOCX document evenly divided into N-byte size parts. The last fragment of the document after the last byte is filled with 0x00 characters up to the size of N bytes. After that, all the obtained fragments are randomly mixed and presented as an input array for further analysis. It is necessary to obtain the correct order of fragments and restore the original document, and if this is not possible, restore parts of its XML elements.

Since the standard cluster size in various file systems in the vast majority of real tasks is 4096 bytes, this value of N was used here.

3.2. Datasets description

In order to test the suitability of the proposed method for reconstructing OOXML documents, two datasets, Govdocs1 [38] and NapierOne [39], were used. The md5 hashes of all files from each dataset were compared before conducting the experiments. Also, files smaller than 4096 bytes were ignored.

Govdocs1 dataset [38] (created in 2009) contains different files. Among the 163 DOCX documents, 157 unique files are larger than 4096 bytes.

NapierOne dataset [39] (created in 2021) contains data from publicly available web resources from the gov.uk domain, including 5000 DOCX documents (4992 unique);

Tables 2, 3, and fig. 16 provide general information about file sizes in the mentioned datasets. Table 4 provides information about the size characteristics and compression type of the vast majority of local elements in DOCX files.

3.3. Analysis of results

To test the effectiveness of the proposed method, we used the Govdocs1 and NapierOne datasets described above, as well as the following software tools: Anaconda Navigator 2.3.2, python 3.8.8, HxD Hex Editor 2.5.0.0, Microsoft Office 365 version 16.0.16026.20002, Autopsy 4.20.0.

Table 2

Information about file sizes in datasets

Dataset	Unique files	File size, bytes	Av. file size, bytes	Av. file size, clusters
Govdocs1	157	10229–9494595	214150	52.8
NapierOne	4992	11451–14580050	303552	74.61

Table 3

Information about file sizes in datasets

Dataset		File size, clusters						
		1-8	9-16	17-24	25-32	33-40	41-48	49+
Govdocs1	Files	86	17	10	8	8	2	26
	%	54.78	10.83	6.37	5.10	5.10	1.27	16.56
NapierOne	Files	1071	1314	665	418	227	191	1106
	%	21.45	26.32	13.32	8.37	4.55	3.83	22.16

Fig. 16 Information about file sizes in datasets

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Table 4

Information about local elements

Element name	Files	Element compressed size, bytes			Compressed elements, %	Elements that occupy at least 3 clusters	
		min	max	average		Elements	%
[Content_Types].xml	157	358	613	410	100	0	0.0
	4992	346	1085	452	100	0	0.0
document.xml	158	695	243512	20080	100	85	53.8
	5296	348	2534430	26218	100	3708	70.02
*.rels	390	185	3576	302	100	0	0.0
	22098	184	15259	292	100	7	0.03
settings.xml	158	618	9330	1837	100	8	5.06
	5297	590	34186	2233	100	275	5.19
styles.xml, stylesWithEffects.xml	158	1527	8759	2720	100	6	3.8
	7293	1501	137209	4326	100	654	8.97
numbering.xml	119	567	22587	3068	100	10	8.4
	4238	464	17187	2618	100	272	6.42
app.xml	157	374	6495	629	100	1	0.64
	4992	308	15513	539	100	1	0.02
core.xml	157	327	540	404	100	0	0.0
	4992	311	822	382	100	0	0.0
header*.xml, footer*.xml	244	301	5468	540	100	1	0.41
	12205	301	479616	1175	100	159	1.30
footnotes.xml, endnotes.xml	160	356	2070	386	100	0	0.0
	7720	360	44797	591	100	18	0.23
word/theme/*.xml	157	1635	1808	1689	100	0	0.0
	5102	997	3241	1634	100	0	0.0
item*.xml, itemProps*.xml	80	133	1249	235	100	0	0.0
	16414	2	31603	409	100	5	0.03
webSettings.xml	158	164	3490	336	100	0	0.0
	5297	164	35288	533	100	14	0.26
fontTable.xml	158	380	836	540	100	0	0.0
	5297	380	1612	647	100	0	0.0
custom.xml	10	157	420	298	100	0	0.0
	2039	223	2066	444	100	0	0.0
word/diagrams/*.xml	0	–	–	–	–	–	–
	380	793	12616	1931	100	14	3.68
word/charts/*.xml	0	–	–	–	–	–	–
	492	162	6851	1576	100	1	0.2
*.png, *.jpeg, *.gif, *.jpg, *.dat, *.tiff, *.tif, *.svg, *.tmp	430	113	1812417	52843	0	369	85.81
	10338	35	3496512	81312	0.10	7857	76.00
*.bin, *.emf, *.wmf, *.ppt, *.doc, *.xls, *.dttf	99	242	565986	50820	100	59	60.20
	2526	142	9159432	162342	100	1268	50.20
*.docx	0	–	–	–	–	–	–
	45	10632	3380231	257913	100.0	45	100.0
*.xlsx, *.pptx	0	–	–	–	–	–	–
	176	8475	14478286	456897	0	176	100.0

Table 5

Information about the number of files with large local elements

Dataset	Number of files with elements that occupy at least 3 clusters	Number of files with XML elements that occupy at least 3 clusters	Number of files with uncompressed elements
Govdocs1	111 (70.70 %)	88 (56.05 %)	53 (33.76 %)
NapierOne	4314 (86.42 %)	3753 (75.18 %)	3329 (66.69 %)

Table 6 summarizes the overall results for the number of reconstructed files from each dataset, showing the number and percentage of:

a) fully reconstructed OOXML documents;

b) fully reconstructed OOXML documents that partially lack the end of the central directory – the last part of up to 21 bytes;

c) fully reconstructed OOXML documents excluding uncompressed embeddings in the following formats: PNG, JPEG, GIF, etc.;

d) fully reconstructed OOXML documents except for also compressed embeddings in formats such as EMF, WMF, DOC, XLS, PPT, DOCX, etc.;

e) OOXML documents whose reconstruction process failed with errors;

f) OOXML documents that were excluded from the analysis due to the presence of XLSX, PPTX embeddings.

Table 7 presents statistical information on the results of reconstructing individual local elements.

Table 8 shows the number of successfully recovered clusters of "document.xml" files and, accordingly, the percentage of their fragments partially recovered for cases where errors occurred during the reconstruction of this element. Additionally, the largest reconstructed fragments among the files from the Govdocs1 and NapierOne datasets were local elements of "document.xml", which occupied 60 and 223 clusters, respectively. At the same time, the reconstructed files themselves had sizes of 2319 and 702 clusters, respectively.

As a result of testing the proposed method's effectiveness, as seen in Table 6, it was possible to achieve an efficiency rate of 82.97 % for document reconstruction on two datasets in total. Simultaneously, 34.38 % of files were fully reconstructed, 0.43 % were reconstructed except for the last 21 bytes at most, and another 48.16 % of documents were reconstructed without errors except for embedding. In the latter case, the documents had fully recovered texts, internal metadata, parameters, etc., but had gaps in place of media data. In essence, this is a simplified version of detecting fragments of a document without images in unallocated space and then reconstructing the file. The simplification lies in the total dataset being thousands of times smaller than in real cases, and all fragments a priori belong to the same document.

The emphasis was placed on the recovery of XML elements that contain the main content of the OOXML document and are most common among the elements that occupy at least three clusters. As can be seen in Table 4, such elements include the "document.xml" file, which occupies at least 3 clusters in 69.5 % of cases; the "stylesWithEffects.xml", "settings.xml", "styles.xml", "numbering.xml" - 5.2 % to 8.9 %; as well as both structurally similar "header*.xml" and "footer*.xml", which are large in a little more than 1 % of cases in total. The elements "footnotes.xml", "endnotes.xml" and "webSettings.xml" were also accounted for. Other XML and RELS elements, which in few cases can reach large sizes or whose structure can differ significantly from the typical one, were ignored.

4. Discussion

Testing the algorithm's effectiveness showed that the proposed method of reconstructing OOXML documents works. The best effectiveness of reconstructing OOXML files was achieved on the Govdocs1 dataset. This could be due to the fact that the NapierOne dataset was created in 2021, and, as a result, the documents in it (in particular, the local elements "document.xml") have a more complex structure.

As can be seen in Table 7, the largest number of errors when restoring individual local elements – about 14 % – was observed when working with "document.xml" files. All other XML elements were successfully reconstructed in 99 % of the cases. This can be explained by the fact that most of the errors occurred at the last stage of the search when filtering out false positives, namely, during the process of comparing the XML structure of the decompressed part of the data with the typical XML structure of the corresponding local element. In the case of the "documents.xml" files, this number of errors was caused by their complex structure with different elements, including elements with graphic data, etc. Errors in the reconstruction of data blocks related to, for example, the "header*.xml" and "footer*.xml" files were also mainly caused by the presence of media data. Modifying methods similar to [40] may be helpful in the latter cases. The relatively simple structure of the other XML elements and/or their small sizes made achieving such efficiency in their reconstruction possible.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Table 6

Results of reconstructing OOXML files

Dataset (unique files)	Fully restored	Fully restored but without a footer	Fully restored, but with missing uncompressed embeddings of other data types	Fully recovered, but with missing compressed embeddings of other data types	With errors	Skipped files (with XLSX, PPTX embeddings)
Govdocs1 (157)	153 (97.45 %)				4 (2.55 %)	0 (0.00 %)
	99 (63.06 %)	1 (0.64 %)	36 (22.93 %)	17 (10.83 %)		
NapierOne (4992)	4119 (82.51 %)				783 (15.69 %)	90 (1.80 %)
	1671 (33.47 %)	21 (0.42 %)	2090 (41.87 %)	337 (6.75 %)		

Table 7

Information about reconstructing OOXML local elements

Element name	Files	Number of elements that occupy at least 3 clusters		With errors	
		Elements	%	Elements	%
document.xml	158	85	53.8	2	1.27
	5296	3708	70.02	754	14.24
*.rels	390	0	0.0	0	0.0
	22098	7	0.03	3	0.01
settings.xml	158	8	5.06	1	0.63
	5297	275	5.19	2	0.04
styles.xml, stylesWithEffects.xml	158	6	3.8	0	0.00
	7293	654	8.97	4	0.05
numbering.xml	119	10	8.4	1	0,84
	4238	272	6.42	1	0.02
app.xml	157	1	0.64	0	0.00
	4992	1	0.02	1	0.02
core.xml	157	0	0.0	0	0.00
	4992	0	0.0	0	0.00
header*.xml, footer*.xml	244	1	0.41	0	0.00
	12205	159	1,30	52	0.43
footnotes.xml, endnotes.xml	160	0	0.0	0	0.00
	7720	18	0.23	1	0.01
item*.xml, itemProps*.xml	80	0	0.0	0	0.00
	16414	5	0,03	1	0.01
webSettings.xml	158	0	0.0	0	0.00
	5297	14	0.26	0	0.00

Table 8

Information about reconstructing "document.xml" elements

Element name	With errors		Number of clusters occupied by elements with errors	Successfully identified clusters	
	Elements	%		Clusters	%
document.xml	2	1.27	41	6	14.63
	754	14.24	18215	2154	11.83

As can be seen in Table 8, the proposed method for reconstructing OOXML files allows extracting 85.76 % to 98.73 % of full texts from documents for different cases. A similar problem was solved in [13], where the authors obtained a result of 54.35 % to 90.54 % of the detected texts of documents in the RAM under other input data and conditions. However, it seems that the authors did not consider fragmented data blocks that did not contain the signatures of the local file headers 0x504b0304 since the key data structures were searched for by this expression, ignoring other significant parts of OOXML files. Also, fragments, where local header signatures were located on the borders of non-contiguous clusters, could be skipped. As for elements that occupy several non-contiguous data blocks in unallocated space, it is highly likely to skip text from intermediate fragments using the signature-based data recovery method.

In contrast to this work, the method proposed in this study allows for detecting the above non-contiguous data blocks. This possibility is proven by the fact that the largest successfully reconstructed "document.xml" elements were 60 and 223 clusters for the Govdocs1 and NapierOne datasets, respectively. In the former case, 60 fragments were found among about 2300 other clusters, and in the latter case, among 702 clusters. This can be roughly compared to searching for data blocks of the "document.xml" element among the unallocated space.

Additionally, the proposed method allowed the recovery of about 11 % of OOXML documents in which the reconstruction of the "document.xml" elements failed (Table 8).

During the study, if a document contains an unfilled sequence of 2-4 clusters size, identifying these missing fragments by completely searching through all possible options was not a goal. Although such a task is quite feasible with a limited number of clusters, it is far from real cases.

It is also worth paying attention to the following fact, which did not affect the results of the current study but may cause uncertainty when recovering from unallocated space OOXML files and ZIP archives in general. For example, if the header of the end of the central directory is located on the border of two non-contiguous clusters (less than 1 % of cases in general), when only one or two of its bytes (0x50 or 0x504b) are contained in the former of them, it is impossible without additional analysis to exactly determine whether these bytes are the beginning of the signature of the next record of the central directory or its end. As a result, there is an ambiguity about the actual archive size and number of its elements.

Conclusions

For the first time, a syntactical method for reconstructing OOXML files has been developed which can be used to recover highly fragmented OOXML documents. The method is based on the analysis of the internal structure and content of this file type and is suitable for searching fragments of OOXML files in unallocated space and RAM.

The high efficiency of OOXML file reconstruction using this particular method exceeds the results obtained by other researchers and has been experimentally proven. The method's effectiveness was evaluated on public datasets such as Govdocs1 and NapierOne.

The practical significance of the proposed method lies in reconstructing OOXML documents based on the use of knowledge about the internal structure of OOXML files, regardless of the language of the document and/or its content. Its main advantage over other research [11, 12] is that it works with compressed data streams and does not require decompressing the entire text for its reconstruction. The proposed method allows improving the OOXML file carving techniques and approaches used in [13] and to search for data blocks of OOXML documents that do not have clear markers, such as the signatures of local file headers, etc.

The obtained scientific results show its effectiveness at the level of 82.97 % of successfully reconstructed documents, among which 34.38 % of files were entirely reconstructed, 0.43 % were fully reconstructed except for the last 21 bytes at most, and another 48.16 % of documents were reconstructed without errors except for embeddings. Simultaneously, 85.76 % to 98.73 % of the analyzed OOXML files had fully recovered the main texts of the documents. Additionally, about 11 % of the main texts were restored from OOXML documents whose reconstruction was completed with errors.

Although the method presented in this paper was applied to data fragmented into 4096-byte chunks, which corresponds to the standard cluster size in various file systems, it is quite possible to apply it to data blocks of arbitrary sizes.

Theoretically, the proposed algorithm allows recovering any ZIP archive or a specific part of it that uses the deflate compression method. To achieve this, first it is necessary to study the structure of its local elements and determine the markers by which the data will be verified.

This study does not solve the issue of reconstructing OOXML documents that contain uncompressed embeddings in the form of files with a similar internal structure (for example, XLSX and PPTX files). Here, it is necessary to separate the local elements of the embedded file from similar or analogous elements of the main document.

Additionally, local elements such as PNG, JPEG, GIF, JPG, and EMF, etc., were not processed because they de facto belong to other data types with their own internal structure and require separate methods for their

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

reconstruction. In these cases, the text of the reconstructed files will be fully viewable, for example, in the Microsoft Word application but without displaying media data

Future research should be focused on increasing the proposed method's efficiency by reducing the number of errors when comparing XML structures at the last stage of the algorithm. This can be done by applying models and methods of intellectual analysis (for example, using dictionary-based techniques). Another important direction of research should be the question of recovering highly fragmented OOXML files in unallocated space and memory dumps.

Contribution of authors: conceptualization of the problem, supervision and editing of work – **Viacheslav Moskalenko**; development of the method, analysis and visualization of the results – **Maksym Boiko**.

All authors have read and agreed with the published version of the manuscript.

References

1. Cantrell, G., & Runs Through, J. The five levels of data destruction: A paradigm for introducing data recovery in a computer science course. *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, 2019, pp. 133-138. DOI: 10.1109/CSCI49370.2019.00029.

2. Ali, N. U. A., Iqbal, W., & Shafqat, N. Analysis of windows OS's fragmented file carving techniques: A systematic literature review. *Advances in Intelligent Systems and Computing*, 2019, vol. 800, pp. 63-67. DOI: 10.1007/978-3-030-14070-0_10.

3. Ramli, N. I. S., Hisham, S. I., & Razak, M. F. A. Survey of File Carving Techniques. *In Lecture Notes on Data Engineering and Communications Technologies*, 2021, vol. 72, pp. 815-825. DOI: 10.1007/978-3-030-70713-2_74.

4. Sari, S. A., & Mohamad, K. M. A Review of Graph Theoretic and Weightage Techniques in File Carving. *Journal of Physics: Conference Series*, 2020, vol. 1529, iss. 5, article no. 052011. DOI: 10.1088/1742-6596/1529/5/052011.

5. van der Meer, V., Jonker, H., & van den Bos, J. A contemporary investigation of NTFS file fragmentation. *Forensic Science International: Digital Investigation*, 2021, vol. 38(Suppl.), article no. 301125. DOI: 10.1016/j.fsidi.2021.301125.

6. Lee, H., Lee, H.-W. Block based Smart Carving System for Forgery Analysis and Fragmented File Identification. *Journal of Internet Computing and Services*, 2020, vol. 21, no. 3, pp. 93–102. DOI: 10.7472/jksii.2020.21.3.93.

7. Memon, N., & Pal, A. Automated reassembly of file fragmented images using greedy algorithms. *IEEE Transactions on Image Processing*, 2006, vol. 15, iss. 2, pp. 385-393. DOI: 10.1109/TIP.2005.863054.

8. Ravi, A., Kumar, T. R., & Mathew, A. R. A method for carving fragmented document and image files. *2016 International Conference on Advances in Human Machine Interaction (HMI)*, Kodigehalli, India, 2016, pp. 1-6. DOI: 10.1109/HMI.2016.7449170.

9. Shanmugasundaram, K., & Memon, N. Automatic reassembly of document fragments via context based statistical models. *19th Annual Computer Security Applications Conference*, 2003. Proceedings., Las Vegas, NV, USA, 2003, pp. 152-159. DOI: 10.1109/CSAC.2003.1254320.

10. Brown, R. D. Improved recovery and reconstruction of DEFLATEd files. *Digital Investigation*, 2013, vol. 10(Suppl.), pp. S21–S29. DOI: 10.1016/J.DIIN.2013.06.003.

11. Al-Sharif, Z. A., Bagci, H., Abu Zaitoun, T., & Asad, A. Towards the memory forensics of ms word documents. *Advances in Intelligent Systems and Computing*, 2018, vol. 558, pp. 179-185. DOI: 10.1007/978-3-319-54978-1_25.

12. Taşdelen, Kubilay & Süzen, Ahmet. Analysing and Carving MS Word and PDF Files from RAM Images on Windows. *Tehnički vjesnik*, 2022, vol. 29, no. 5, pp. 1714-1720. DOI: 10.17559/TV-20210218122046.

13. Ali, N. U. A., Iqbal, W., & Afzal, H. Carving of the OOXML document from volatile memory using unsupervised learning techniques. *Journal of Information Security and Applications*, 2022, vol. 65, article no. 103096. DOI: 10.1016/j.jisa.2021.103096.

14. Dergachov, K., Krasnov, L., Bilozerskyi, V. & Zymovin, A. Methods and algorithms for protecting information in optical text recognition systems. *Radioelectronic and Computer Systems*, 2022, no. 1, pp. 154-169. DOI: 10.32620/reks.2022.1.12.

15. *Standard ECMA TR/98 JPEG File Interchange Format (JFIF)*. Available at: https://www.ecma-international.org/publications-and-standards/technical-reports/ecma-tr-98/. (accessed 12 january 2023).

16. *G I F (tm) Graphics Interchange Format (tm) A standard defining a mechanism for the storage and transmission of raster-based graphics information.* CompuServe Inc., 1987. Available at: https://www.w3.org/Graphics/GIF/spec-gif87.txt. (accessed 12 january 2023).

17. Ali, Hamza A. & Ne'ma, Bashar M. Effective Variations on Opened GIF Format Images. *IJCSNS*, 2008, vol. 8. No. 5, pp. 70-75.

18. *Bitmap Image File (BMP), Version 5. Sustainability of Digital Formats: Planning for Library of Congress Collections.* Available at: https://www.loc.gov/

preservation/digital/formats/fdd/fdd000189.shtml. (accessed 12 january 2023).

19. Fedorchenko, I., Oliinyk, A., Stepanenko, A., Korniienko, S., Kharchenko, A., & Laktionov, V. Development of a method for compressing images on the basis of JPEG algorithm. *Technology Audit and Production Reserves*, 2020, vol. 2, no. 2(52), pp. 32-34. DOI: 10.15587/2706-5448.2020.202433.

20. Barannik, V., Krasnorutsky, A., Shulgin, S., Yeroshenko, V., Sidchenko, Y., & Hordiienko, A. Image compression based on classification coding of constant-pitched functions transformers. *Radioelectronic and Computer Systems*, 2021, no. 3, pp. 48-62. DOI: 10.32620/reks.2021.3.05.

21. Ali, R. R., & Mohamad, K. M. RX_myKarve carving framework for reassembling complex fragmentations of JPEG images. *Journal of King Saud University - Computer and Information Sciences*, 2021, vol. 33, iss. 1, pp. 21–32. DOI: 10.1016/J.JKSUCI.2018.12.007.

22. Chang, X., Wu, J., & Hao, F. JPEG fragment carving based on pixel similarity of MED-ED. *Chinese Control Conference (CCC)*, Guangzhou, China, 2019, pp. 8862-8866. DOI: 10.23919/ChiCC.2019.8865161.

23. Durmus, E., Korus, P., & Memon, N. Every Shred Helps: Assembling Evidence from Orphaned JPEG Fragments. *IEEE Transactions on Information Forensics and Security*, 2019, vol. 14, iss. 9, pp. 2372-2386. DOI: 10.1109/TIFS.2019.2897912.

24. Hilgert, J. N., Lambertz, M., Rybalka, M., & Schell, R. Syntactical Carving of PNGs and Automated Generation of Reproducible Datasets. *Digital Investigation*, 2019, vol. 29(Suppl.), pp. S22-S30. DOI: 10.1016/j.diin.2019.04.014.

25. Tang, Y., Fang, J., Chow, K. P., Yiu, S. M., Xu, J., Feng, B., Li, Q., & Han, Q. Recovery of heavily fragmented JPEG files. *DFRWS 2016 USA - Proceedings of the 16th Annual USA Digital Forensics Research Conference*, 2016. DOI: 10.1016/j.diin.2016.04.016.

26. Uzun, E., & Sencar, H. T. Jpg Scraper : An Advanced Carver for JPEG Files. *IEEE Transactions on Information Forensics and Security*, 2020, vol. 15, pp. 1846-1857. DOI: 10.1109/TIFS.2019.2953382.

27. Zhang, L., Hao, S., & Zhang, Q. Recovering SQLite data from fragmented flash pages. *Annales Des Telecommunications – Annals of Telecommunications*, 2019, vol. 74, pp. 251-460. DOI: 10.1007/s12243-019-00707-9.

28. Lin, W., & Xu, M. A Microsoft Word documents carving method based on interior virtual streams. *Advanced Materials Research*, 2012, vol. 433–440, pp. 3028-3032. DOI: 10.4028/www.scientific.net/AMR.433-440.3028.

29. Paixão, T. M., Berriel, R. F., Boeres, M. C. S., Koerich, A. L., Badue, C., de Souza, A. F., & Oliveira-Santos, T. Self-supervised deep reconstruction of mixed strip-shredded text documents. *Pattern Recognition*, 2020, vol. 107, article no. 107535. DOI: 10.1016/J.PATCOG.2020.107535.

30. Bhawal, S., & Tabassum, M. Forensic image reconstruction based on efficient morphological operational model. *Advances in Intelligent Systems and Computing*, 2019, vol. 814, pp. 297-307. DOI: 10.1007/978-981-13-1501-5_26.

31. Alothman, A. F., Wahab Sait A. R. Managing and Retrieving Bilingual Documents Using Artificial Intelligence-Based Ontological Framework. *Comput Intell Neurosci.*, 2022 vol. 2022, article no. 4636931. DOI: 10.1155/2022/4636931.

32. *Standard ECMA-376 Office Open XML File Formats*. Available at: https://www.ecma-international.org/publications-and-standards/standards/ecma-376/. (accessed 12 january 2023).

33. Didriksen, E. *Forensic Analysis of OOXML Documents*, 2014. Available at: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/198656/EDidriksen.pdf. (accessed 12 january 2023).

34. Fu, Z., Sun, X., Liu, Y., & Li, B. Forensic investigation of OOXML format documents. *Digital Investigation*, 2011, vol. 8, iss. 1, pp. 48-55. DOI: 10.1016/j.diin.2011.04.001.

35. *ZIP File Format Specification, version 6.3.10*, PKWare, Inc., 2022. Available at: https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT. (accessed 12 january 2023).

36. Fu, Z., Sun, X., Zhou, L., & Shu, J. New forensic methods for OOXML format documents. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, vol. 8389, pp. 503-513. DOI: 10.1007/978-3-662-43886-2_36.

37. Brown, R. D. Reconstructing corrupt DEFLATEd files. *Digital Investigation*, 2011, vol. 8(Suppl.), pp. S125-S131. DOI: 10.1016/j.diin.2011.05.015.

38. Garfinkel, S., Farrell, P., Roussev, V., & Dinolt, G. Bringing science to digital forensics with standardized forensic corpora. *Digital Investigation*, 2009, vol. 6(Suppl.), pp. S2-S11. DOI: 10.1016/j.diin.2009.06.016.

39. Davies, S. R., Macfarlane, R., & Buchanan, W. J. NapierOne: A modern mixed file data set alternative to Govdocs1. *Forensic Science International: Digital Investigation*, 2022, vol. 40, article no. 301330. DOI: 10.1016/J.FSIDI.2021.301330.

40. Chukhray, A., & Havrylenko, O. The method of student's query analysis while intelligent computer tutoring in SQL. *Radioelectronic and Computer Systems*, 2021, no. 2, pp. 87-96. DOI: 10.32620/reks.2021.2.07.

Radioelectronic and Computer Systems, 2023, no. 1(105)
ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Надійшла до редакції 12.09.2022, розглянута на редколегії 20.02.2023

СИНТАКСИЧНИЙ МЕТОД РЕКОНСТРУКЦІЇ OOXML-ФАЙЛІВ З ВИСОКИМ РІВНЕМ ФРАГМЕНТАЦІЇ

Максим Бойко, В'ячеслав Москаленко

Поширеною задачею комп'ютерно-технічної експертизи є відновлення файлів, для яких відсутні метадані файлової системи. Для пошуку фрагментів файлів у нерозподіленому просторі найчастіше застосовується методи відновлення за сигнатурами (карвінг), які ідеально підходять для нефрагментованих файлів. Однак подібні методи і основані на них інструменти, неефективні для відновлення OOXML-файлів, які мають високий рівень фрагментації. Дані методи не забезпечують достовірного визначення правильного порядку фрагментів. У зв'язку з тим, що OOXML-файли являють собою ZIP-архіви та, як наслідок, зберігають дані на дисковому просторі в стисненому вигляді, то в такому разі також неефективними є техніки реконструювання документів на базі аналізу слів, словосполучень тощо. У роботі пропонується синтаксичний метод реконструкції OOXML-документів, який базується на використанні знань про внутрішню структуру цього типу файлів незалежно від їх вмісту. Розглянуто деталі реалізації алгоритму відновлення та особливості відновлення окремих типів локальних елементів документу. Тестування ефективності алгоритму здійснювалося на наборах даних Govdocs1 і NapierOne. Пропонований метод розглянуто на прикладі розмірів блоків даних розміром 4096 байт, що відповідає стандартному розміру кластера різних файлових систем. Експериментальні результати підтвердили придатність методу для практичного використання з загальним показником у 82,97 % відновлених файлів, серед яких 34,38 % реконструйовано повністю, 0,43 % – за винятком останніх максимум 21 байт, ще 48,16 % – за винятком вкладень до документів, які потребують інших підходів. В останньому випадку досягнуто можливості отримання повністю робочого документа без відображення в ньому графічних зображень, вмісту інших вкладень тощо. Наявність в OOXML-файлах ґеш-кодів CRC-32 розархівованого потоку даних кожного локального елемента дозволяє однозначно підтвердити коректність відновлення інформації та її цілісність. При цьому ефективність методу суттєвим чином залежить від способів верифікації даних при реконструкції локальних елементів, що займають у файлі щонайменше три кластери. Тому даний метод передбачається розвивати шляхом розробки нових механізмів верифікації xml-елементів.

Ключові слова: комп'ютерно-технічна експертиза; відновлення даних; карвінг файлів; синтаксичний карвінг файлів; фрагментація; реконструкція файлів; Office Open XML; OOXML; файл DOCX; ZIP-архів; стиснення DEFLATE.

Бойко Максим Володимирович – асп. каф. комп'ютерних наук, Сумський державний університет, Суми, Україна; ст. детектив, Управління аналітики та обробки інформації, Національне антикорупційне бюро України, Київ, Україна.

Москаленко В'ячеслав Васильович – канд. техн. наук, доц., доц. каф. комп'ютерних наук, Сумський державний університет, Суми, Україна; докторант каф. комп'ютерних систем, мереж та кібербезпеки, Національний аерокосмічний університет ім. М. Є. Жуковського "Харківський авіаційний інститут", Харків, Україна.

Maksym Boiko – PhD student of Computer Sciences Department of Sumy State University, Sumy, Ukraine; senior detective, Information Processing and Analysis Department, the National Anti-corruption Bureau of Ukraine, Kyiv, Ukraine,
e-mail: mboiko25@gmail.com, ORCID: 0000-0003-0950-8399.

Viacheslav Moskalenko – PhD, Associate Professor of Computer Science Department of Sumy State University, Sumy, Ukraine; Doctoral Student of Department of Computer Systems, Networks and Cybersecurity, National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine,
e-mail: v.moskalenko@cs.sumdu.edu.ua, ORCID: 0000-0001-6275-9803, Scopus Author ID: 57189099775.