Introduction to MS Word Document Structure Analysis

Document number	2023001
Author	P.Leclercq - pl@osix.be
Owner	P.Leclercq
Version	1.0
Date	2023-04-15
Status	Draft
Security level	Public

Content

Goal

- Describe the structure of the Microsoft Office Word *.docx* and *.docm* formats to help the analysis of Office malware.

Method

- Explain the reference model described in Microsoft documentation;
- Illustrate the practical decoding of the structure by creating small pieces of Python code applied on simple file examples;
- Leverage the use of the best forensics tools for decoding OLE files: olefile, olemap and oledump.py.

Scope

This document describes the file format of *.docx* and *.docm* documents created by Microsoft Word >= 2007 version. This format is based on the Open Office XML format.

References

- Microsoft (2021). [MS-CFB]: Compound File Binary File Format (https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-cfb/53989ce4-7b05-4f8d-829b-d08d6148375b)
- Microsoft (2023). [MS-OVBA]: Office VBA File Format Structure (https://learn.microsoft.com/en-us/openspecs/office_file_formats/ms-ovba/575462ba-bf67-4190-9fac-c275523c75fc)
- Lagadec, P. (2020). olefile (https://www.decalage.info/olefile)
- Stevens, D. (2020). oledump.py (https://blog.didierstevens.com/programs/oledump-py/)
- The code and example files included in this document are freely available at https://github.com/plecbe/OfficeDocAnalysis/

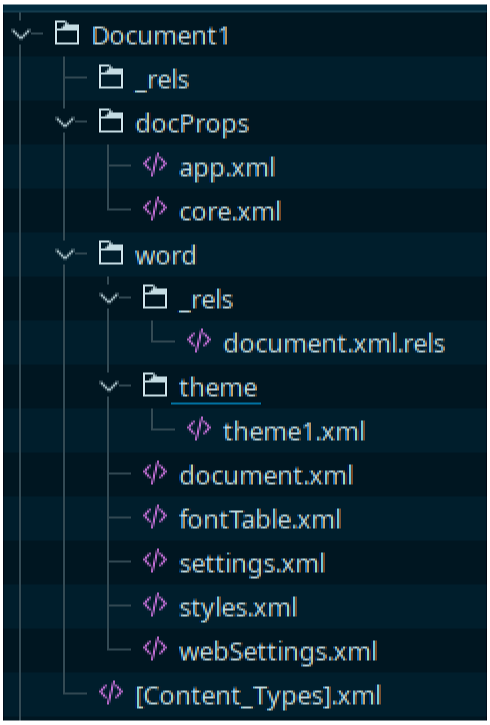
Base structure

A *.docx* file is basically a zip-compressed container containing several directories of XML and binary files. It can be uncompressed by any archiving utility able to manage the pkzip format.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

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 ¢..( .........
```

This is the ASCII dump of the first bytes of a *.docx* file. The first 2 bytes correspond to the "magic number" of a pkzip archive: PK.

A simple *Document1.docx* file, representing a single page document, contains the following hierarchy of directories and files:



Most of the files contain metadata about the document.

The following paragraphs describe the typical structure of a basic document. Note that the file and directory names can vary slightly with the nature and version of the software used to create it.

Structure of a simple document

[Content_Types].xml

This file lists all the various other xml files and directories contained in the *.docx* compressed file. A document can contain various media types like images, text, word art... Therefore, this file lists them and refers to their MIME types. For example, the line `<Default Extension="xml" ContentType="application/xml" />` means that the .xml files will contain documents of type application/xml.

MIME types are defined and standardized in IETF's [RFC 6838](RFC 6838).

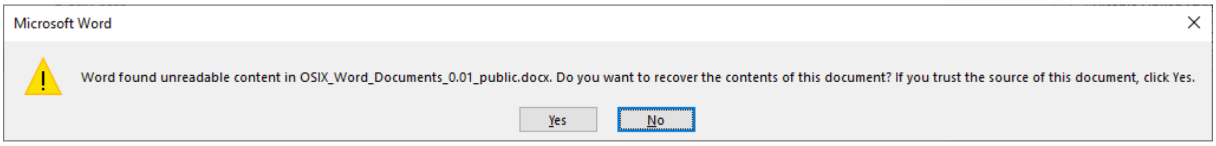
The base MIME type of the xml file containing the main document is `application/vnd.openxmlformats-officedocument.wordprocessingml.styles+xml`.

This is the content of a basic *Content_Types* file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-
 types">
  <Default Extension="rels" ContentType="application/vnd.openxmlformats-
  package.relationships+xml" />
  <Default Extension="xml" ContentType="application/xml" />
  <Override PartName="/word/document.xml"
  ContentType="application/vnd.openxmlformats-
  officedocument.wordprocessingml.document.main+xml" />
```

```
  <Override PartName="/word/styles.xml"
  ContentType="application/vnd.openxmlformats-
  officedocument.wordprocessingml.styles+xml" />
  <Override PartName="/word/settings.xml"
  ContentType="application/vnd.openxmlformats-
  officedocument.wordprocessingml.settings+xml" />
  <Override PartName="/word/webSettings.xml"
  ContentType="application/vnd.openxmlformats-
  officedocument.wordprocessingml.webSettings+xml" />
  <Override PartName="/word/fontTable.xml"
  ContentType="application/vnd.openxmlformats-
  officedocument.wordprocessingml.fontTable+xml" />
  <Override PartName="/word/theme/theme1.xml"
  ContentType="application/vnd.openxmlformats-officedocument.theme+xml" />
  <Override PartName="/docProps/core.xml"
  ContentType="application/vnd.openxmlformats-package.core-properties+xml"
  />
  <Override PartName="/docProps/app.xml"
  ContentType="application/vnd.openxmlformats-officedocument.extended-
  properties+xml" />
</Types>
```

Word only accepts a restricted set of MIME types. When it encounters a type it does not know, it displays an error message like follows when opening the file:



It means you cannot add arbitrary content to the *.docx* container and hope Word will ignore it.

Furthermore, if you add an *.xml* file containing an acceptable MIME type, and list it in the *[Content_Types].xml* file but do not reference it properly in the .rels file (see below), Word will open the document without error, but will silently drop the extraneous content when you save the document. It means that it will not persist files that have been maliciously inserted this way as hidden payload.

_rels

The _rels directory in the root directory is empty in our simple case.

docProps

The docProps typically contains two files describing the document properties.

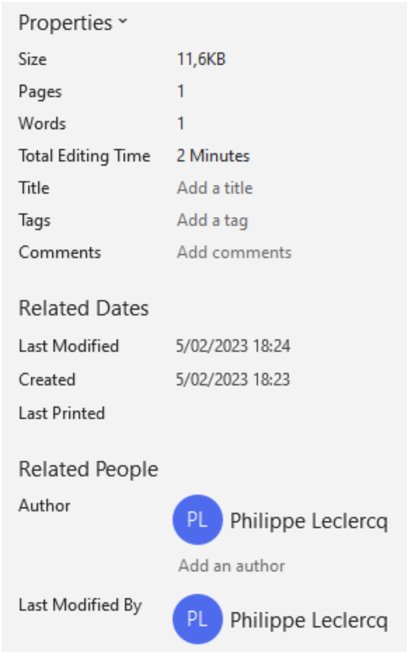
core.xml

The core.xml file contains the author, title, subject, version, creation and modification dates.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<cp:coreProperties
xmlns:cp="http://schemas.openxmlformats.org/package/2006/metadata/core-
properties" xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:dcmitype="http://purl.org/dc/dcmitype/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dc:title />
  <dc:subject />
```

```
  <dc:creator>Philippe Leclercq</dc:creator>
  <cp:keywords />
  <dc:description />
  <cp:lastModifiedBy>Philippe Leclercq</cp:lastModifiedBy>
  <cp:revision>1</cp:revision>
  <dcterms:created xsi:type="dcterms:W3CDTF">2023-02-
  05T17:23:00Z</dcterms:created>
  <dcterms:modified xsi:type="dcterms:W3CDTF">2023-02-
  05T17:24:00Z</dcterms:modified>
</cp:coreProperties>
```

These are the properties that are displayed when you click on File -> Info -> Properties in Word.



It means that anybody who can unzip the *.docx* file can modify these values with a simple editor and re-zip it to change the properties without even running Word.

Don't blindly trust the properties, they can easily be spoofed.

app.xml

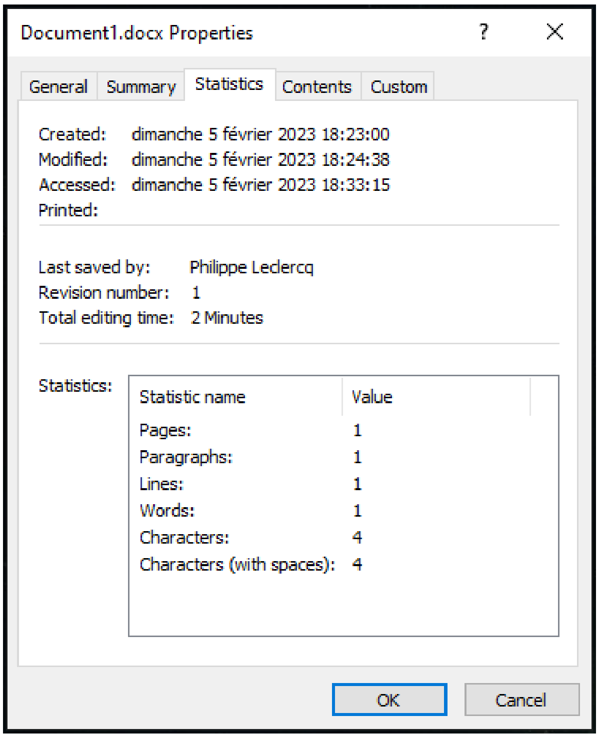
The *app.xml* lists the application and version with which the document has been created and extended properties for the document, like number of lines, words, characters, whether the document has been shared, the Office template used as a base for the styles in the document, whether the hyperlinks have been changed and are up to date...

You can find the complete list of extended properties here: http://www.datypic.com/sc/ooxml/e-extended-properties_Properties.html

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Properties
 xmlns="http://schemas.openxmlformats.org/officeDocument/2006/extended-
 properties"
 xmlns:vt="http://schemas.openxmlformats.org/officeDocument/2006/docPropsVT
 ypes">
  <Template>Normal.dotm</Template>
  <TotalTime>1</TotalTime>
  <Pages>1</Pages>
```

```
  <Words>0</Words>
  <Characters>4</Characters>
  <Application>Microsoft Office Word</Application>
  <DocSecurity>0</DocSecurity>
  <Lines>1</Lines>
  <Paragraphs>1</Paragraphs>
  <ScaleCrop>false</ScaleCrop>
  <Company />
  <LinksUpToDate>false</LinksUpToDate>
  <CharactersWithSpaces>4</CharactersWithSpaces>
  <SharedDoc>false</SharedDoc>
  <HyperlinksChanged>false</HyperlinksChanged>
  <AppVersion>16.0000</AppVersion>
</Properties>
```

You can find these extended properties by clicking File -> Info -> Properties -> Advance Properties in Word.



Note that if they are absent, Word proposes to rebuild them when you open the file. If you see this proposal when you open the document, you can suspect that it has been tampered with.

word

In this directory you can find the content of the document itself and the various attributes related to its rendering, like the themes, styles and fonts.

theme

This directory contains the theme files.

theme1.xml

This file defines the attributes of the theme used to create the document. It contains the definition of the fonts used, their color and typeface, the attributes of default lines, gradients, shadows… and can reference external files.

This is an excerpt of a theme file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<a:theme xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
name="Office Theme">
  <a:themeElements>
    <a:clrScheme name="Office">
      <a:dk1>
        <a:sysClr val="windowText" lastClr="000000" />
      </a:dk1>
      <a:lt1>
        <a:sysClr val="window" lastClr="FFFFFF" />
      </a:lt1>
      <a:hlink>
        <a:srgbClr val="0563C1" />
      </a:hlink>
      <a:folHlink>
        <a:srgbClr val="954F72" />
      </a:folHlink>
    </a:clrScheme>
    <a:fontScheme name="Office">
      <a:majorFont>
        <a:latin typeface="Calibri Light" panose="020F0302020204030204"
        />
        <a:font script="Jpan" typeface="游ゴシック Light" />
        <a:font script="Arab" typeface="Times New Roman" />
        <a:font script="Hebr" typeface="Times New Roman" />
        <a:font script="Thai" typeface="Angsana New" />
      </a:majorFont>
      <a:minorFont>
        <a:latin typeface="Calibri" panose="020F0502020204030204" />
        <a:font script="Jpan" typeface="游明朝" />
        <a:font script="Arab" typeface="Arial" />
        <a:font script="Hebr" typeface="Arial" />
        <a:font script="Thai" typeface="Cordia New" />
      </a:minorFont>
    </a:fontScheme>
    <a:fmtScheme name="Office">
      <a:fillStyleLst>
        <a:solidFill>
          <a:schemeClr val="phClr" />
        </a:solidFill>
      </a:fillStyleLst>
    </a:fmtScheme>
  </a:themeElements>
  <a:extLst>
    <a:ext uri="{05A4C25C-085E-4340-85A3-A5531E510DB2}">
      <thm15:themeFamily
      xmlns:thm15="http://schemas.microsoft.com/office/thememl/2012/main"
      name="Office Theme" id="{62F939B6-93AF-4DB8-9C6B-D6C7DFDC589F}"
      vid="{4A3C46E8-61CC-4603-A589-7422A47A8E4A}" />
    </a:ext>
  </a:extLst>
</a:theme>
```

fontTable.xml

This file describes the fonts used in the document and refers to the xml definitions of various Word/Office versions for the fonts. The description includes name, character set, pitch, color…

Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<w:fonts xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships" xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
xmlns:w16cex="http://schemas.microsoft.com/office/word/2018/wordml/cex"
xmlns:w16cid="http://schemas.microsoft.com/office/word/2016/wordml/cid"
xmlns:w16="http://schemas.microsoft.com/office/word/2018/wordml"
xmlns:w16sdtdh="http://schemas.microsoft.com/office/word/2020/wordml/sdtdatahash"
xmlns:w16se="http://schemas.microsoft.com/office/word/2015/wordml/symex"
mc:Ignorable="w14 w15 w16se w16cid w16 w16cex w16sdtdh">
 <w:font w:name="Calibri">
   <w:panose1 w:val="020F0502020204030204" />
   <w:charset w:val="00" />
   <w:family w:val="swiss" />
   <w:pitch w:val="variable" />
   <w:sig w:usb0="E4002EFF" w:usb1="C000247B" w:usb2="00000009"
   w:usb3="00000000" w:csb0="000001FF" w:csb1="00000000" />
 </w:font>
 <w:font w:name="Times New Roman">
   <w:panose1 w:val="02020603050405020304" />
   <w:charset w:val="00" />
   <w:family w:val="roman" />
   <w:pitch w:val="variable" />
   <w:sig w:usb0="E0002EFF" w:usb1="C000785B" w:usb2="00000009"
   w:usb3="00000000" w:csb0="000001FF" w:csb1="00000000" />
 </w:font>
</w:fonts>
```

settings.xml

This file contains settings you can find in the File -> Settings screen, e.g., compatibility settings, separators…

Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<w:settings xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships" xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
xmlns:v="urn:schemas-microsoft-com:vml" xmlns:w10="urn:schemas-microsoft-com:office:word"
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
xmlns:w16cex="http://schemas.microsoft.com/office/word/2018/wordml/cex"
xmlns:w16cid="http://schemas.microsoft.com/office/word/2016/wordml/cid"
xmlns:w16="http://schemas.microsoft.com/office/word/2018/wordml"
```

```
xmlns:w16sdtdh="http://schemas.microsoft.com/office/word/2020/wordml/sdtdat
ahash"
xmlns:w16se="http://schemas.microsoft.com/office/word/2015/wordml/symex"
xmlns:sl="http://schemas.openxmlformats.org/schemaLibrary/2006/main"
mc:Ignorable="w14 w15 w16se w16cid w16 w16cex w16sdtdh">
  <w:zoom w:percent="100" />
  <w:defaultTabStop w:val="720" />
  <w:characterSpacingControl w:val="doNotCompress" />
  <m:mathPr>
    <m:mathFont m:val="Cambria Math" />
    <m:brkBin m:val="before" />
    <m:brkBinSub m:val="--" />
    <m:smallFrac m:val="0" />
    <m:dispDef />
    <m:lMargin m:val="0" />
    <m:rMargin m:val="0" />
    <m:defJc m:val="centerGroup" />
    <m:wrapIndent m:val="1440" />
    <m:intLim m:val="subSup" />
    <m:naryLim m:val="undOvr" />
  </m:mathPr>
  <w:themeFontLang w:val="en-GB" />
  <w:clrSchemeMapping w:bg1="light1" w:t1="dark1" w:bg2="light2"
  w:t2="dark2" w:accent1="accent1" w:accent2="accent2" w:accent3="accent3"
  w:accent4="accent4" w:accent5="accent5" w:accent6="accent6"
  w:hyperlink="hyperlink" w:followedHyperlink="followedHyperlink" />
  <w:decimalSymbol w:val="." />
  <w:listSeparator w:val="," />
  <w14:docId w14:val="5F83821A" />
  <w15:chartTrackingRefBased />
  <w15:docId w15:val="{DB3CC0D1-20A9-4059-BF02-57784854D971}" />
</w:settings>
```

styles.xml

This file contains the description of all the styles that can be used in the document, like headings, normal, paragraph and character styles, table styles, numbering….

Excerpt:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<w:styles xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationship
s" xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
xmlns:w16cex="http://schemas.microsoft.com/office/word/2018/wordml/cex"
xmlns:w16cid="http://schemas.microsoft.com/office/word/2016/wordml/cid"
xmlns:w16="http://schemas.microsoft.com/office/word/2018/wordml"
xmlns:w16sdtdh="http://schemas.microsoft.com/office/word/2020/wordml/sdtdat
ahash"
xmlns:w16se="http://schemas.microsoft.com/office/word/2015/wordml/symex"
mc:Ignorable="w14 w15 w16se w16cid w16 w16cex w16sdtdh">
  <w:docDefaults>
    <w:rPrDefault>
      <w:rPr>
        <w:lang w:val="en-GB" w:eastAsia="en-US" w:bidi="ar-SA" />
      </w:rPr>
```

```xml
    </w:rPrDefault>
  </w:docDefaults>
  <w:latentStyles w:defLockedState="0" w:defUIPriority="99"
  w:defSemiHidden="0" w:defUnhideWhenUsed="0" w:defQFormat="0"
  w:count="376">
    <w:lsdException w:name="Normal" w:uiPriority="0" w:qFormat="1" />
    <w:lsdException w:name="heading 1" w:uiPriority="9" w:qFormat="1" />
    <w:lsdException w:name="heading 2" w:semiHidden="1" w:uiPriority="9"
    w:unhideWhenUsed="1" w:qFormat="1" />
    <w:lsdException w:name="index 1" w:semiHidden="1" w:unhideWhenUsed="1"
    />
    <w:lsdException w:name="toc 1" w:semiHidden="1" w:uiPriority="39"
    w:unhideWhenUsed="1" />
    <w:lsdException w:name="header" w:semiHidden="1" w:unhideWhenUsed="1"
    />
    <w:lsdException w:name="footer" w:semiHidden="1" w:unhideWhenUsed="1"
    />
    <w:lsdException w:name="List Number" w:semiHidden="1"
    w:unhideWhenUsed="1" />
    <w:lsdException w:name="Title" w:uiPriority="10" w:qFormat="1" />
    <w:lsdException w:name="Default Paragraph Font" w:semiHidden="1"
    w:uiPriority="1" w:unhideWhenUsed="1" />
    <w:lsdException w:name="Body Text" w:semiHidden="1"
    w:unhideWhenUsed="1" />
    <w:lsdException w:name="Table Simple 1" w:semiHidden="1"
    w:unhideWhenUsed="1" />
  </w:latentStyles>
  <w:style w:type="paragraph" w:default="1" w:styleId="Normal">
    <w:name w:val="Normal" />
    <w:qFormat />
  </w:style>
  <w:style w:type="character" w:default="1"
  w:styleId="DefaultParagraphFont">
    <w:name w:val="Default Paragraph Font" />
    <w:uiPriority w:val="1" />
    <w:semiHidden />
    <w:unhideWhenUsed />
  </w:style>
</w:styles>
```

webSettings

This file contains some settings for web publishing and viewing.

Example:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<w:webSettings xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationship
s" xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
xmlns:w16cex="http://schemas.microsoft.com/office/word/2018/wordml/cex"
xmlns:w16cid="http://schemas.microsoft.com/office/word/2016/wordml/cid"
xmlns:w16="http://schemas.microsoft.com/office/word/2018/wordml"
xmlns:w16sdtdh="http://schemas.microsoft.com/office/word/2020/wordml/sdtdat
ahash"
```

```
xmlns:w16se="http://schemas.microsoft.com/office/word/2015/wordml/symex"
mc:Ignorable="w14 w15 w16se w16cid w16 w16cex w16sdtdh">
  <w:optimizeForBrowser />
  <w:allowPNG />
</w:webSettings>
```


document.xml

Finally, the real content of the document is contained in this file. It contains the text and images you insert in your document in the <body> tag, with some additional styles and formatting information.

Our simple document, containing the "abcd" string, looks like follows:


```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<w:document
xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanv
as" xmlns:cx="http://schemas.microsoft.com/office/drawing/2014/chartex"
xmlns:cx1="http://schemas.microsoft.com/office/drawing/2015/9/8/chartex"
xmlns:cx2="http://schemas.microsoft.com/office/drawing/2015/10/21/chartex"
xmlns:cx3="http://schemas.microsoft.com/office/drawing/2016/5/9/chartex"
xmlns:cx4="http://schemas.microsoft.com/office/drawing/2016/5/10/chartex"
xmlns:cx5="http://schemas.microsoft.com/office/drawing/2016/5/11/chartex"
xmlns:cx6="http://schemas.microsoft.com/office/drawing/2016/5/12/chartex"
xmlns:cx7="http://schemas.microsoft.com/office/drawing/2016/5/13/chartex"
xmlns:cx8="http://schemas.microsoft.com/office/drawing/2016/5/14/chartex"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:aink="http://schemas.microsoft.com/office/drawing/2016/ink"
xmlns:am3d="http://schemas.microsoft.com/office/drawing/2017/model3d"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:oel="http://schemas.microsoft.com/office/2019/extlst"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationship
s" xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDra
wing"
xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDr
awing" xmlns:w10="urn:schemas-microsoft-com:office:word"
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
xmlns:w16cex="http://schemas.microsoft.com/office/word/2018/wordml/cex"
xmlns:w16cid="http://schemas.microsoft.com/office/word/2016/wordml/cid"
xmlns:w16="http://schemas.microsoft.com/office/word/2018/wordml"
xmlns:w16sdtdh="http://schemas.microsoft.com/office/word/2020/wordml/sdtdat
ahash"
xmlns:w16se="http://schemas.microsoft.com/office/word/2015/wordml/symex"
xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGrou
p"
xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocessingInk"
xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShap
e" mc:Ignorable="w14 w15 w16se w16cid w16 w16cex w16sdtdh wp14">
  <w:body>
    <w:p w14:paraId="38BB770E" w14:textId="26DE8476" w:rsidR="00F23B73"
    w:rsidRDefault="00C70005">
      <w:r>
      <w:t>abcd</w:t>
      </w:r>
    </w:p>
```

```
    <w:sectPr w:rsidR="00F23B73">
    <w:pgSz w:w="11906" w:h="16838" />
    <w:pgMar w:top="1440" w:right="1440" w:bottom="1440" w:left="1440"
    w:header="708" w:footer="708" w:gutter="0" />
    <w:cols w:space="708" />
    <w:docGrid w:linePitch="360" />
    </w:sectPr>
  </w:body>
</w:document>
```

The ***xmldump.py*** tool, present in the remnux distribution, a Linux distribution aimed at reverse engineering and malware analysis (https://remnux.org/), helps viewing the text in a simple form.

```
remnux@remnux:~/Documents/OfficeDocs/Docwithinsertedtext/word$ xmldump.py text document.xml
Abcdef.
```

_rels

_rels is a directory containing the files describing the relationships between the document file and other files.

document.xml.rels

This file describes the relationships between the files describing the document metadata or other resources, like the settings, themes, styles and fonts. Each relationship has a unique id and specifies the referenced xml target. For our document, it looks like this:

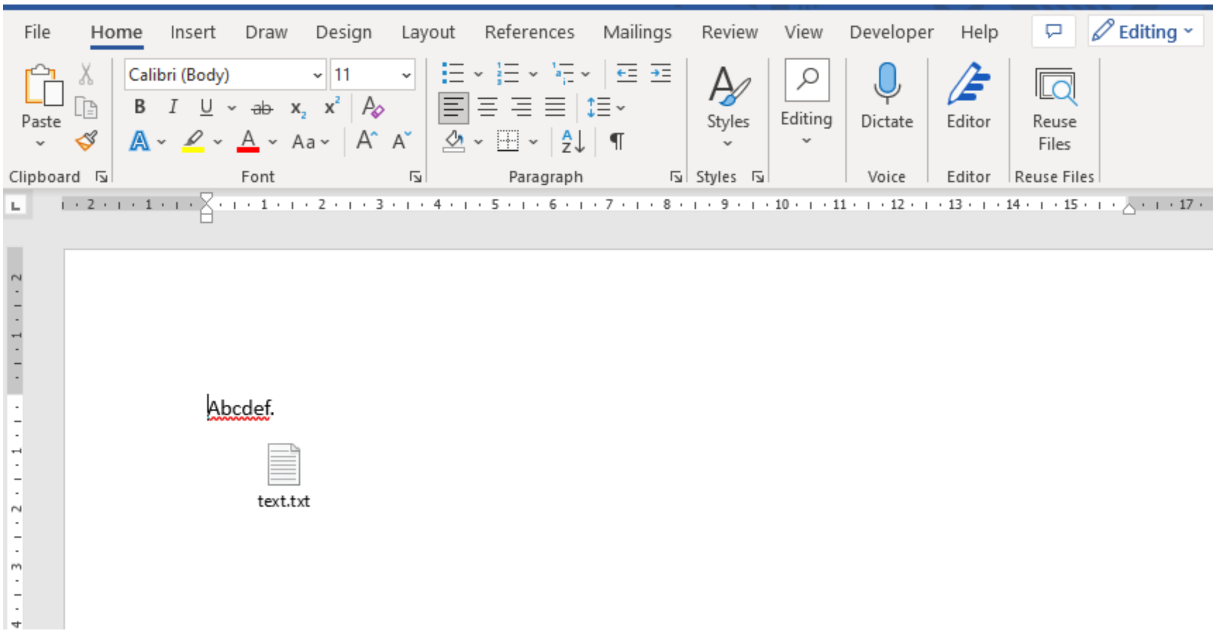
```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId3"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
  s/webSettings" Target="webSettings.xml" />
  <Relationship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

Documents with embedded objects

If you use the Insert->Object command or if you drag and drop an external file into your Word document, you insert an OLE (Object Linking and Embedding) object.

This file adds a picture in a new *word/media* directory and a *.bin* file in a new *word/embeddings* directory to the *.docx* container.

This is an example of a Word file containing a text file object.



Content_Types

In the *[Content_Types].xml* file, you will find the following additional lines listing the MIME type of the .bin file, the .bin file and the thumbnail picture:

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

```
</Types>
```

Document.xms.rels

In the *word/document.xml.rels*, the following highlighted relations are added:

```
<?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/relationship
  s/webSettings" Target="webSettings.xml" />
  <Relationship Id="rId7"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
  s/theme" Target="theme/theme1.xml" />
  <Relationship Id="rId2"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
  s/settings" Target="settings.xml" />
  <Relationship Id="rId1"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
  s/styles" Target="styles.xml" />
  <Relationship Id="rId6"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
  s/fontTable" Target="fontTable.xml" />
  <Relationship Id="rId5"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
  s/oleObject" Target="embeddings/oleObject1.bin" />
  <Relationship Id="rId4"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
  s/image" Target="media/image1.emf" />
</Relationships>
```

The document itself contains the references to the thumbnail picture and the description of the OLE object.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<w:document
xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanv
as" xmlns:cx="http://schemas.microsoft.com/office/drawing/2014/chartex"
xmlns:cx1="http://schemas.microsoft.com/office/drawing/2015/9/8/chartex"
xmlns:cx2="http://schemas.microsoft.com/office/drawing/2015/10/21/chartex"
xmlns:cx3="http://schemas.microsoft.com/office/drawing/2016/5/9/chartex"
xmlns:cx4="http://schemas.microsoft.com/office/drawing/2016/5/10/chartex"
xmlns:cx5="http://schemas.microsoft.com/office/drawing/2016/5/11/chartex"
xmlns:cx6="http://schemas.microsoft.com/office/drawing/2016/5/12/chartex"
xmlns:cx7="http://schemas.microsoft.com/office/drawing/2016/5/13/chartex"
xmlns:cx8="http://schemas.microsoft.com/office/drawing/2016/5/14/chartex"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:aink="http://schemas.microsoft.com/office/drawing/2016/ink"
xmlns:am3d="http://schemas.microsoft.com/office/drawing/2017/model3d"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:oel="http://schemas.microsoft.com/office/2019/extlst"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationship
s" xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDra
wing"
xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDr
awing" xmlns:w10="urn:schemas-microsoft-com:office:word"
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
```

```
xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
xmlns:w16cex="http://schemas.microsoft.com/office/word/2018/wordml/cex"
xmlns:w16cid="http://schemas.microsoft.com/office/word/2016/wordml/cid"
xmlns:w16="http://schemas.microsoft.com/office/word/2018/wordml"
xmlns:w16sdtdh="http://schemas.microsoft.com/office/word/2020/wordml/sdtdat
ahash"
xmlns:w16se="http://schemas.microsoft.com/office/word/2015/wordml/symex"
xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGrou
p"
xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocessingInk"
xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShap
e" mc:Ignorable="w14 w15 w16se w16cid w16 w16cex w16sdtdh wp14">
  &lt;w:body&gt;
    <w:p w14:paraId="3B913E80" w14:textId="59D115F3" w:rsidR="002837DF"
    w:rsidRDefault="00516570">
      &lt;w:proofErr w:type="spellStart" /&gt;
      &lt;w:r&gt;
        &lt;w:t&gt;Abcdef&lt;/w:t&gt;
      &lt;/w:r&gt;
      &lt;w:proofErr w:type="spellEnd" /&gt;
      &lt;w:r&gt;
        &lt;w:t&gt;.&lt;/w:t&gt;
      &lt;/w:r&gt;
    &lt;/w:p&gt;
    <w:p w14:paraId="4543FAEC" w14:textId="316C54D2" w:rsidR="00516570"
    w:rsidRDefault="00516570">
      &lt;w:r&gt;
        <w:object w:dxaOrig="1534" w:dyaOrig="994"
        w14:anchorId="23FECA4B">
          <v:shapetype id="_x0000_t75" coordsize="21600,21600" o:spt="75"
          o:preferrelative="t" path="m@4@5l@4@11@9@11@9@5xe" filled="f"
          stroked="f">
            &lt;v:stroke joinstyle="miter" /&gt;
            &lt;v:formulas&gt;
              &lt;v:f eqn="if lineDrawn pixelLineWidth 0" /&gt;
              &lt;v:f eqn="sum @0 1 0" /&gt;
              &lt;v:f eqn="sum 0 0 @1" /&gt;
              &lt;v:f eqn="prod @2 1 2" /&gt;
              &lt;v:f eqn="prod @3 21600 pixelWidth" /&gt;
              &lt;v:f eqn="prod @3 21600 pixelHeight" /&gt;
              &lt;v:f eqn="sum @0 0 1" /&gt;
              &lt;v:f eqn="prod @6 1 2" /&gt;
              &lt;v:f eqn="prod @7 21600 pixelWidth" /&gt;
              &lt;v:f eqn="sum @8 21600 0" /&gt;
              &lt;v:f eqn="prod @7 21600 pixelHeight" /&gt;
              &lt;v:f eqn="sum @10 21600 0" /&gt;
            &lt;/v:formulas&gt;
            <v:path o:extrusionok="f" gradientshapeok="t"
            o:connecttype="rect" />
            &lt;o:lock v:ext="edit" aspectratio="t" /&gt;
          &lt;/v:shapetype&gt;
          <v:shape id="_x0000_i1025" type="#_x0000_t75"
          style="width:76.85pt;height:49.85pt" o:ole="">
            &lt;v:imagedata r:id="rId4" o:title="" /&gt;
          &lt;/v:shape&gt;
```

```
            <o:OLEObject Type="Embed" ProgID="Package"
            ShapeID="_x0000_i1025" DrawAspect="Icon" ObjectID="_1738538003"
            r:id="rId5" />
          </w:object>
        </w:r>
      </w:p>
      <w:sectPr w:rsidR="00516570">
        <w:pgSz w:w="11906" w:h="16838" />
        <w:pgMar w:top="1417" w:right="1417" w:bottom="1417" w:left="1417"
        w:header="708" w:footer="708" w:gutter="0" />
        <w:cols w:space="708" />
        <w:docGrid w:linePitch="360" />
      </w:sectPr>
    </w:body>
</w:document>
```

OLE File Format

The OLE object is contained in the *word/embeddings/oleObject1.bin* file, which, as its extension suggests, is a binary file.

It starts with the classical 8-byte OLE file signature (in hexadecimal) D0 CF 11 E0 A1 B1 1A E1 (resembles the word DOCFILE):

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

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  ................
--------   -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Between binary code, you can catch some recognizable (Unicode) strings mentioning OLE package, and the content of the embedded text file:

```
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿR o o t
 E n t r y                                             ÿÿ
ÿÿÿÿÿÿ
         À       F                    
€         C o m p O b j
                 ÿÿÿÿÿÿÿÿÿÿÿÿ
       L           O b j I n f o
                                ÿÿÿÿ
                    O l e 1 0 N a t i v e
                            ÿÿÿÿÿÿÿÿÿÿÿÿ
               •               þÿÿÿþÿÿÿ
```

```
   OLE
Package     ▯   Package ô9²q
                              @ ▯ ▯
                                 `     text.txt C:\Users\plc
\AppData\Local\Microsoft\Windows\INetCache\Content.Word
\text.txt   ▯ w   C:\Users\plc\AppData\Local\Temp
\{A49ACDDC-335F-427C-8E1E-0BA95559D2D2}\{48F89947-8786-4F93-
A433-014810EA148D}\text.txt ▯   This is a text file.
v   C : \ U s e r s \ p l c \ A p p D a t a \ L o c a l
\ T e m p \ { A 4 9 A C D D C - 3 3 5 F - 4 2 7 C - 8 E 1 E -
 0 B A 9 5 5 5 9 D 2 D 2 } \ { 4 8 F 8 9 9 4 7 - 8 7 8 6 -
 4 F 9 3 - A 4 3 3 - 0 1 4 8 1 0 E A 1 4 8 D }
\ t e x t . t x t ▯   t e x t . t x t L   C : \ U s e r s
\ p l c \ A p p D a t a \ L o c a l \ M i c r o s o f t
\ W i n d o w s \ I N e t C a c h e
\ C o n t e n t . W o r d
\ t e x t . t x t
```

An OLE file is a Compound File Binary (CFB) file. This format allows one file to contain different kinds of objects.

OLE file storage

An OLE file is a ***Microsoft Compound File Binary*** (CBF) file. Its structure is documented on the Microsoft site (https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-cfb/53989ce4-7b05-4f8d-829b-d08d6148375b) and the downloadable documentation (https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/Windows_Protocols.zip)

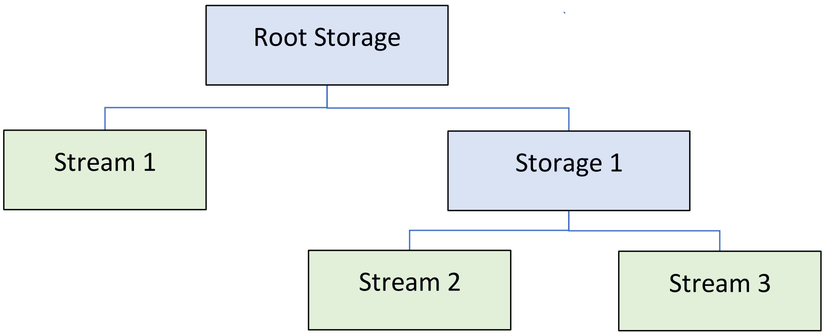
This format is also used for the pre-2007 versions of MS Office to store *.doc* and *.xls* documents.

A CBF is a mini filesystem designed to store different objects in a single file. It is divided into equal-length sectors (512 or 4096 bytes). It starts with a one-sector header giving the general characteristics of the file structure, and, as a normal filesystem, contains different structures describing to what object a sector belongs.

Header	Sector 0	Sector 1	Sector 2	…

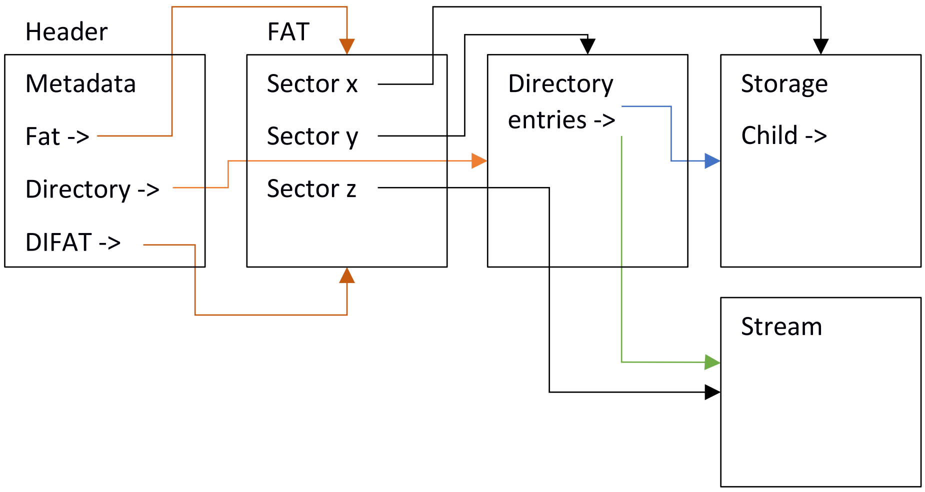
If an object is large enough to occupy several sectors, its sectors are organized into sector chains. The allocated sectors and the sector chains are described in a special structure called FAT (FAT Allocation Table).

Objects within a CFB file can be ***storages***, which act as directories or ***streams***, that act as files. An CFB file can for example be logically organized as follows:



Schematically, the path to find an object in a CBF file is the following:

Header -> DIFAT -> FAT -> Directory -> stream content



OLE file header

The header starts with the magic number D0 CF 11 E0 A1 B1 1A E1, signature of a CBF file. The first digits resemble the word DOCFILE.

Then it contains several numbers describing the structure of the file.

The first interesting value is Sector Shift, containing the exponent of 2 representing the size of a sector. For a version 3 OLE file, this number is 9, meaning the sector size is 2^9 = 512 bytes. For version 4, it is 12 (0x0C), meaning the sector size is 2^12 = 4096 bytes.

Other interesting values are the First DIFAT Sector Location and First Directory Sector Location, giving access to the DIFAT and Directories.

Using the description in the Microsoft documentation, we can write a Python program parsing and displaying the header portion of the CBF file.

Note:

The code presented here is for educational use only. It is voluntarily kept simple, non-optimized, has only been tested on a small sample of Office files, and does not cope with malformed documents. It should not be used in production.

```
import struct
```

```
import argparse

# Some constants
MAGICOLESIG = b'\xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1'   # OLE file signature
NUMBER_DIFAT_ENTRIES_IN_HEADER = 109                  # Number of DIFAT entries in
header
UNALLOCATED = 0xFFFFFFFF                              # Unallocated entry

# Main program
def main():
    # Parse OLE filename
    parser = argparse.ArgumentParser()
    parser.add_argument("filename", help="OLE file to be parsed")
    args = parser.parse_args()
    P_filename = args.filename
    print ("Filename: ",P_filename)

    # Open the OLE file in binary mode
    with open(P_filename, 'rb') as f:
        # Check we find the magic number in the first 8 bytes
        magic = f.read(8)
        if magic != MAGICOLESIG:
            print("!!!! Not an OLE file !!!!")
            f.close()
            exit(1)
        # Read the header information from the file
        clsid = f.read(16)
        minor_version = struct.unpack('<H', f.read(2))[0]
        major_version = struct.unpack('<H', f.read(2))[0]
        byte_order = struct.unpack('<H', f.read(2))[0]
        sector_shift = struct.unpack('<H', f.read(2))[0]
        mini_sector_shift = struct.unpack('<H', f.read(2))[0]
        reserved = f.read(6)
        directory_sector_count = struct.unpack('<I', f.read(4))[0]
        fat_sector_count = struct.unpack('<I', f.read(4))[0]
        first_directory_sector_id = struct.unpack('<I', f.read(4))[0]
        transaction_signature_number = struct.unpack('<I', f.read(4))[0]
        mini_stream_cutoff_size = struct.unpack('<I', f.read(4))[0]
        first_mini_fat_sector_id = struct.unpack('<I', f.read(4))[0]
        mini_fat_sector_count = struct.unpack('<I', f.read(4))[0]
        first_difat_sector_id = struct.unpack('<I', f.read(4))[0]
        difat_sector_count = struct.unpack('<I', f.read(4))[0]
        difat_entries = []
        # Read the 109 DIFAT entries in the header
        for i in range(NUMBER_DIFAT_ENTRIES_IN_HEADER):
            difat_entries.append(struct.unpack('<I', f.read(4))[0])

        # Compute sector size
        mySectorSize = 2 ** sector_shift

        # Print the header information
        print("======== CBF header - size: " + str(mySectorSize) + " bytes
========")
        print("  Field                          | Offset | Size |    Expected
| Value")
        print("Magic number                     |    0x0 |    8 | 0xD0 CF 11 E0 A1 B1 1A
E1     | ", end="")
        for i in range(8):
            print("0x{0:X} ".format(magic[i]), end="")
        print()
        print("CLSID                            |    0x8 |   16 | all 0s
| ", end="")
        for i in range (16):
            print ("{0:02X}".format(clsid[i]), end="")
        print("")
        print("Version                          |   0x18 |    4 | 3.62 or 4.62
| {0:d}.".format(major_version) + "{0:d}".format(minor_version))
```

```
        print("Byte Order                    |   0x1C |    2 | 0xFFFE
| 0x{0:X}".format(byte_order))
        if major_version == 3:
            print("Sector Shift                  |   0x1E |    2 | 0x0009 -> sector
size=2^9=512 bytes: {0:d}".format(sector_shift))
        else:
            print("Sector Shift                  |   0x1E |    2 | 0x000C -> sector
size=2^12=4096 bytes: {0:d}".format(sector_shift))
        print("Mini Sector Shift             |   0x20 |    2 | 0x0006 -> mini stream
sector size=2^6=64 bytes): {0:d}".format(mini_sector_shift))
        print("Reserved                      |   0x22 |    6 | all 0s
| ", end="")
        for i in range(6):
            print("0x{0:X} ".format(reserved[i]), end="")
        print()
        print("Directory Sector Count        |   0x28 |    4 | 0 if major version is
3       | {0:d}".format(directory_sector_count))
        print("FAT Sector Count              |   0x2C |    4 |
| {0:d}".format(fat_sector_count))
        print("First Directory Sector ID     |   0x30 |    4 |
| {0:d} - 0x{0:X}".format(first_directory_sector_id))
        print("Transaction Signature Number|   0x34 |    4 |
| {0:d}".format(transaction_signature_number))
        print("Mini Stream Cutoff Size       |   0x38 |    4 | 4096
| {0:d} - 0x{0:X}".format(mini_stream_cutoff_size))
        print("First Mini FAT Sector ID      |   0x3C |    4 |
| {0:d} - 0x{0:X}".format(first_mini_fat_sector_id))
        print("Mini FAT Sector Count         |   0x40 |    4 |
| {0:d} - 0x{0:X}".format(mini_fat_sector_count))
        print("First DIFAT Sector ID         |   0x44 |    4 | 0xFFFFFFFE = end of
chain     | {0:d} - 0x{0:X}".format(first_difat_sector_id))
        print("DIFAT Sector Count            |   0x48 |    4 |
| {0:d} - 0x{0:X}".format(difat_sector_count))
        print("DIFAT Entries in header       |   0x4C |109 x 4 |")

        # Find last non-empty DIFAT entry
        myMaxEntry = 0
        for i in range(NUMBER_DIFAT_ENTRIES_IN_HEADER):
            if difat_entries[i] != UNALLOCATED:
                myMaxEntry = i
        # Print the allocated DIFAT entries in header
        for i in range(myMaxEntry + 1):
            print("                              |   0x{0:X} |
| 0x{1:X}".format(0x4C + i*4, difat_entries[i]))
#
if __name__ == "__main__":
    main()

#EOF
```

The result for our document containing an embedded text file is:

```
remnux@remnux:~/Docwithinsertedtext/word/embeddings$ python3 MyOleFileParser.py oleObject1.bin
Filename:  oleObject1.bin
======== CBF header - size: 512 bytes ========
  Field                      | Offset | Size |    Expected                  | Value
Magic number                 |    0x0 |    8 | 0xD0 CF 11 E0 A1 B1 1A E1    | 0xD0 0xCF 0x11 0xE0 0xA1 0xB1 0x1A 0xE1
CLSID                        |    0x8 |   16 | all 0s                       | 00000000000000000000000000000000
Version                      |   0x18 |    4 | 3.62 or 4.62                 | 3.62
Byte Order                   |   0x1C |    2 | 0xFFFE                       | 0xFFFE
Sector Shift                 |   0x1E |    2 | 0x0009 -> sector size=2^9=512 bytes: 9
Mini Sector Shift            |   0x20 |    2 | 0x0006 -> mini stream sector size=2^6=64 bytes): 6
Reserved                     |   0x22 |    6 | all 0s                       | 0x0 0x0 0x0 0x0 0x0 0x0
Directory Sector Count       |   0x28 |    4 | 0 if major version is 3      | 0
FAT Sector Count             |   0x2C |    4 |                              | 1
First Directory Sector ID    |   0x30 |    4 |                              | 1 - 0x1
Transaction Signature Number|   0x34 |    4 |                              | 0
Mini Stream Cutoff Size      |   0x38 |    4 | 4096                         | 4096 - 0x1000
First Mini FAT Sector ID     |   0x3C |    4 |                              | 2 - 0x2
Mini FAT Sector Count        |   0x40 |    4 |                              | 1 - 0x1
First DIFAT Sector ID        |   0x44 |    4 | 0xFFFFFFFE = end of chain    | 4294967294 - 0xFFFFFFFE
DIFAT Sector Count           |   0x48 |    4 |                              | 0 - 0x0
DIFAT Entries in header      |   0x4C |109 x 4 |
                             |   0x4C |                                     | 0x0
```

Two tools you should use to analyze OLE files are the ***oletools*** by P.Lagadec (https://www.decalage.info/python/oletools) and ***oledump.py*** by D.Stevens (https://blog.didierstevens.com/programs/oledump-py/). They are both present in the remnux distribution.

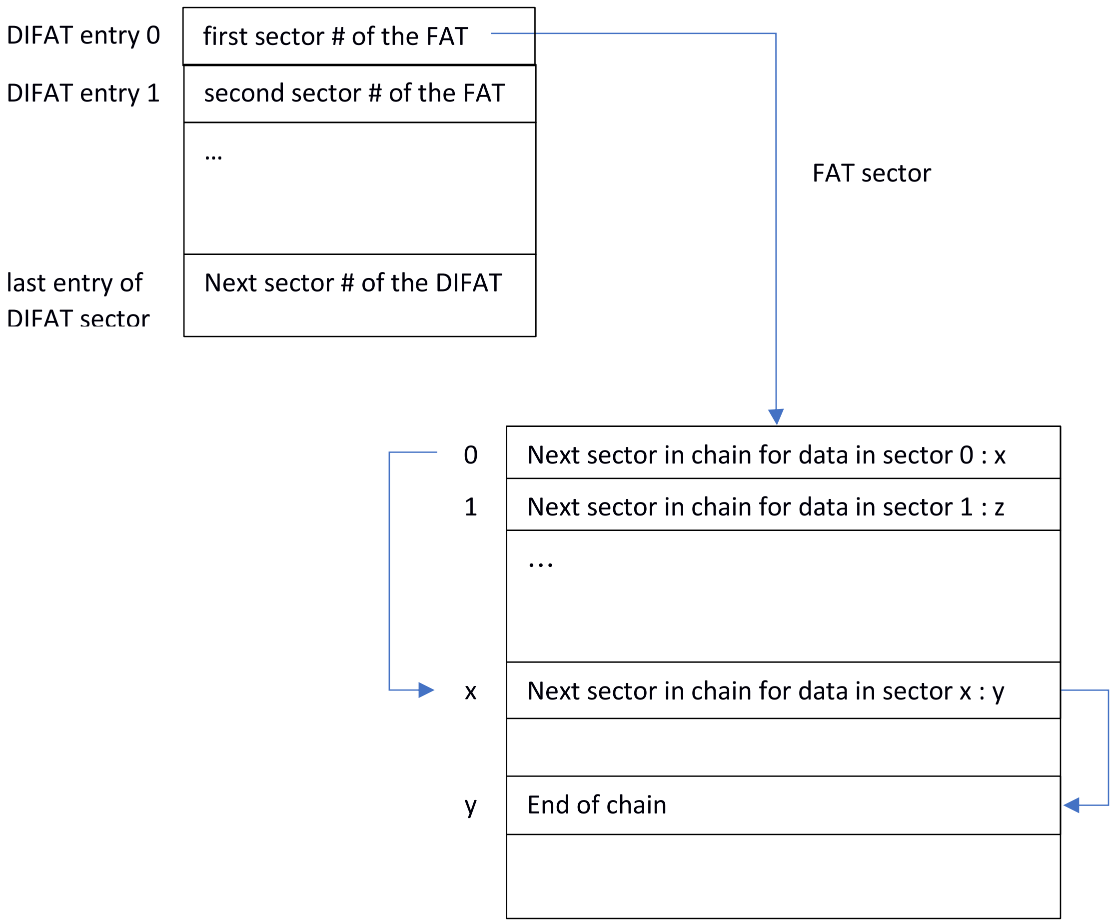
You can use the ***olemap*** utility to display the OLE file header:

```
remnux@remnux:~/Docwithinsertedtext/word/embeddings$ olemap oleObject1.bin
olemap 0.55 - http://decalage.info/python/oletools
-------------------------------------------------------------------------------
FILE: oleObject1.bin

OLE HEADER:
+-------------------------+---------------+-----------------------------------+
|Attribute                |Value          |Description                        |
+-------------------------+---------------+-----------------------------------+
|OLE Signature (hex)      |D0CF11E0A1B11AE1|Should be D0CF11E0A1B11AE1        |
|Header CLSID             |               |Should be empty (0)                |
|Minor Version            |003E           |Should be 003E                     |
|Major Version            |0003           |Should be 3 or 4                   |
|Byte Order               |FFFE           |Should be FFFE (little endian)     |
|Sector Shift             |0009           |Should be 0009 or 000C             |
|# of Dir Sectors         |0              |Should be 0 if major version is 3  |
|# of FAT Sectors         |1              |                                   |
|First Dir Sector         |00000001       |(hex)                              |
|Transaction Sig Number   |0              |Should be 0                        |
|MiniStream cutoff        |4096           |Should be 4096 bytes               |
|First MiniFAT Sector     |00000002       |(hex)                              |
|# of MiniFAT Sectors     |1              |                                   |
|First DIFAT Sector       |FFFFFFFE       |(hex)                              |
|# of DIFAT Sectors       |0              |                                   |
+-------------------------+---------------+-----------------------------------+

CALCULATED ATTRIBUTES:
+-------------------------+---------------+-----------------------------------+
|Attribute                |Value          |Description                        |
+-------------------------+---------------+-----------------------------------+
|Sector Size (bytes)      |512            |Should be 512 or 4096 bytes        |
|Actual File Size (bytes) |3072           |Real file size on disk             |
|Max File Size in FAT     |66048.0        |Max file size covered by FAT       |
|Extra data beyond FAT    |0              |Only if file is larger than FAT    |
|                         |               |coverage                           |
|Extra data offset in FAT |00000C00       |Offset of the 1st free sector at   |
|                         |               |end of FAT                         |
|Extra data size          |0              |Size of data starting at the 1st   |
|                         |               |free sector at end of FAT          |
+-------------------------+---------------+-----------------------------------+

To display the FAT or MiniFAT structures, use options --fat or --minifat, and -h for help.
```

DIFAT

The DIFAT (Double Indirect File Allocation Table) will give access to the FAT, which in turn will describe the sector chains belonging to objects.

Each DIFAT entry gives the sector number of a FAT sector.



The first part of the DIFAT (109 first values) is contained in the header, so we will read these entries first. Then, if the file is large enough and additional DIFAT entries are needed, the last DIFAT entry contains the sector number of the next DIFAT sector. A special value of End of Chain (0xFFFFFFFE) stored in the last entry means that no more DIFAT sector is needed. When stored in the header in the First DIFAT Sector Location, End of Chain means that there is no separate DIFAT sector, and that all DIFAT entries are stored in the header.

So, our additional code in the program will first build the complete DIFAT, by reading the first 109 entries in the header, and, if needed, by following the pointer in the last DIFAT entry to find the next DIFAT sector until the next sector pointer is End of Chain.

```
        ENDOFCHAIN = 0xFFFFFFFE                             # End of chain value

        # Build DIFAT sector chain
        print("======== DIFAT map outside header ========")
        myDifat_chain = [0]                                 # First DIFAT sector is
the header
        myNextDifatSector = first_difat_sector_id           # Take next DIFAT
sector number
        while myNextDifatSector != ENDOFCHAIN:
            print("Next DIFAT sector: {0:X}".format(myNextDifatSector))
            # While we are not at the end, add sector number to DIFAT chain
            myDifat_chain.append(myNextDifatSector)
            # Compute offset to next DIFAT sector
            myOffset = ((1 + myNextDifatSector) * mySectorSize)
            f.seek(myOffset, 0)
            # Append DIFAT entries to difat_entries
            for i in range((mySectorSize // 4) - 1):
```

```
                myOffset = f.tell()
                myFatSector = struct.unpack('<I', f.read(4))[0]
                difat_entries.append(myFatSector)
                print("DIFAT entry = FAT sector: offset 0x{0:X} - Value
0x{1:X}".format(myOffset, myFatSector))
            # Last entry of the DIFAT sector is pointer to next DIFAT sector
            myNextDifatSector = (struct.unpack('<I', f.read(4))[0])

        # Print DIFAT map
        print("DIFAT sector chain: ", end="")
        print(myDifat_chain)
```

For our example file, the result is the following:

```
remnux@remnux:~/Docwithinsertedtext/word/embeddings$ python3 MyOleFileParser.py oleObject1.bin
Filename:  oleObject1.bin
======== CBF header - size: 512 bytes ========
  Field                       | Offset | Size |    Expected                     | Value
Magic number                  |    0x0 |    8 | 0xD0 CF 11 E0 A1 B1 1A E1       | 0xD0 0xCF 0x11 0xE0 0xA1 0xB1 0x1A 0xE1
CLSID                         |    0x8 |   16 | all 0s                          | 00000000000000000000000000000000
Version                       |   0x18 |    4 | 3.62 or 4.62                    | 3.62
Byte Order                    |   0x1C |    2 | 0xFFFE                          | 0xFFFE
Sector Shift                  |   0x1E |    2 | 0x0009 -> sector size=2^9=512 bytes: 9
Mini Sector Shift             |   0x20 |    2 | 0x0006 -> mini stream sector size=2^6=64 bytes): 6
Reserved                      |   0x22 |    6 | all 0s                          | 0x0 0x0 0x0 0x0 0x0 0x0
Directory Sector Count        |   0x28 |    4 | 0 if major version is 3         | 0
FAT Sector Count              |   0x2C |    4 |                                 | 1
First Directory Sector ID     |   0x30 |    4 |                                 | 1 - 0x1
Transaction Signature Number|     0x34 |    4 |                                 | 0
Mini Stream Cutoff Size       |   0x38 |    4 | 4096                            | 4096 - 0x1000
First Mini FAT Sector ID      |   0x3C |    4 |                                 | 2 - 0x2
Mini FAT Sector Count         |   0x40 |    4 |                                 | 1 - 0x1
First DIFAT Sector ID         |   0x44 |    4 | 0xFFFFFFFE = end of chain       | 4294967294 - 0xFFFFFFFE
DIFAT Sector Count            |   0x48 |    4 |                                 | 0 - 0x0
DIFAT Entries in header       |   0x4C |109 x 4 |
                              |   0x4C |                                          | 0x0
======== DIFAT map outside header ========
DIFAT sector chain: [0]
```

Since the DIFAT sector count is 0 and the first DIFAT sector is End of Chain, there is no additional DIFAT sector; all entries are in the header.

You can use the olefile with debug mode (***olefile -d***) to list the DIFAT:

```
remnux@remnux:~/Docwithinsertedtext/word/embeddings$ olefile -d oleObject1.bin
olefile version 0.46 2018-09-09 - https://www.decalage.info/en/olefile

DEBUG    File size: 3072 bytes (C00h)
DEBUG    fmt_header size = 76, +FAT = 512
DEBUG    (b'\xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1', b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00', 62, 3, 65534, 9, 6, 0, 0, 0, 1, 1, 0,
4096, 2, 1, 4294967294, 0)
DEBUG    Minor Version = 62
DEBUG    DLL Version   = 3 (expected: 3 or 4)
DEBUG    Byte Order    = FFFE (expected: FFFE)
DEBUG    Sector Size   = 512 bytes (expected: 512 or 4096)
DEBUG    MiniFAT Sector Size   = 64 bytes (expected: 64)
DEBUG    Number of Directory sectors = 0
DEBUG    Number of FAT sectors = 1
DEBUG    First Directory sector  = 1h
DEBUG    Transaction Signature Number    = 0
DEBUG    Mini Stream cutoff size = 1000h (expected: 1000h)
DEBUG    First MiniFAT sector      = 2h
DEBUG    Number of MiniFAT sectors = 1
DEBUG    First DIFAT sector        = FFFFFFFEh
DEBUG    Number of DIFAT sectors   = 0
DEBUG    Maximum number of sectors in the file: 5 (5h)
DEBUG    _check_duplicate_stream: sect=1h in FAT
DEBUG    _check_duplicate_stream: sect=2h in FAT
DEBUG    Loading the FAT table, starting with the 1st sector after the header
DEBUG    len(sect)=436, so 109 integers
index        0        1        2        3        4        5        6        7
    0:        0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
    8: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   10: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   18: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   20: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   28: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   30: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   38: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   40: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   48: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   50: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   58: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   60: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
   68: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
DEBUG    isect = 0
DEBUG    isect = FFFFFFFF
DEBUG    found end of sector chain
DEBUG    No DIFAT, because file size < 6.8MB.
```

FAT

The FAT is an array of sector numbers, describing sector chains belonging to the same object, or unallocated sectors.

Each FAT entry represents one sector, and contains the next sector allocated to the same object.

As the header occupies 1 sector, the first sector allocated to an object is sector 1 of the file, so the first entry in the FAT, with index 0, represents the second sector of the file, with index 1. So, a sector number in the FAT can be converted to a byte offset in the file by the formula: (sector number + 1) * sector size.

Again, an End of Chain value stored in a FAT entry means that this is the last sector of an object.

Furthermore, other special values are used in the FAT to characterize sectors belonging to special structures: 0xFFFFFFFC means the sector is a DIFAT sector, 0xFFFFFFFD means the sector is a FAT sector, and 0xFFFFFFFF means the sector is unallocated.

To continue our program, we can now build a map of the sector occupation by constructing the FAT.

We will walk the DIFAT to find successive FAT sectors, and for each FAT entry in each FAT sector, interpret the contained value as the next sector in a chain. We will also tag the sectors with their content (DIFAT, FAT, data…) along the line.

```
        DIFATSECTOR = 0xFFFFFFFC                                    # DIFAT sector
        FATSECTOR = 0xFFFFFFFD                                      # FAT sector

        myAllocatedSectors = []
        # Build FAT
        print("======== FAT map ========")
        # Loop on DIFAT entries
        for i in range(len(difat_entries)):
            if difat_entries[i] == ENDOFCHAIN:
```

```
                print("DIFAT entry: " + str(i), end=" | ")
                print("End of chain - 0xFFFFFFFE")
                break
            else:
                if difat_entries[i] != UNALLOCATED:  #Ignore unallocated DIFAT
entries
                    print("DIFAT entry: " + str(i))
                    # Each DIFAT entry is the sector number containing FAT entries
                    print("First FAT sector: {0:d} -
0x{0:X}".format(difat_entries[i]))
                    # Compute offest to FAT sector
                    myOffset = (1 + difat_entries[i]) * mySectorSize
                    # Build FAT map
                    f.seek(myOffset, 0)
                    # Each FAT entry is 4 bytes
                    for j in range(mySectorSize // 4):
                        # Every DIFAT entry (i) corresponds to one full sector of
FAT entries.
                        # Each FAT entry is 4 bytes, so the number of FAT entries
per DIFAT entry is sector
                        #  size/4.
                        # And the 0th FAT entry contains the next sector for the
sector 0
                        mySectorNumber = j + (i * (mySectorSize // 4))
                        myNextSector = struct.unpack('<I', f.read(4))[0]
                        if myNextSector == FATSECTOR:
                            type = "FAT"
                        elif myNextSector == DIFATSECTOR:
                            type = "DIFAT"
                        elif myNextSector == ENDOFCHAIN:
                            type = "End of Chain"
                        elif myNextSector == UNALLOCATED:
                            type = "Free"
                        else:
                            type = "Data"
                        myAllocatedSectors.append({"number" : mySectorNumber,
"next" : myNextSector, "ptroffset" : myOffset + 4*j, "type" : type, "type2" : ""})
        # Print FAT map
        print("      Sector       | Pointer offset | Sector offset | Next sector |
Type")
        # Find last non-free sector
        myMaxSector = 0
        for i in range(len(myAllocatedSectors)):
            if myAllocatedSectors[i]["type"] != "Free":
                myMaxSector = i
        # Tag some sectors with their content
        # Directory sectors
        mySector = first_directory_sector_id
        while mySector != ENDOFCHAIN:
            for i in range(len(myAllocatedSectors)):
                if myAllocatedSectors[i]["number"] == mySector:
                    myAllocatedSectors[mySector]["type2"] = " - Directory"
            mySector = myAllocatedSectors[mySector]["next"]

        # Mini FAT sectors
        mySector = first_mini_fat_sector_id
        while mySector != ENDOFCHAIN:
            for i in range(len(myAllocatedSectors)):
                if myAllocatedSectors[i]["number"] == mySector:
                    myAllocatedSectors[mySector]["type2"] = " - Mini FAT"
            mySector = myAllocatedSectors[mySector]["next"]

        for i in range(myMaxSector+1):
            print(" {0:>7d} - 0x{0:>5X} |     0x{1:08X} |    0x{2:08X} |  0x{3:>8X}
| {4} {5}".format(myAllocatedSectors[i]["number"],
myAllocatedSectors[i]["ptroffset"], (1 + myAllocatedSectors[i]["number"]) *
mySectorSize, myAllocatedSectors[i]["next"], myAllocatedSectors[i]["type"],
myAllocatedSectors[i]["type2"]))
```

For our OLE file, the result is the following:

```
======== FAT map ========
DIFAT entry: 0
First FAT sector: 0 - 0x0
      Sector       | Pointer offset | Sector offset | Next sector | Type
       0 - 0x    0 |     0x00000200 |    0x00000200 |  0xFFFFFFFD | FAT
       1 - 0x    1 |     0x00000204 |    0x00000400 |  0xFFFFFFFE | End of Chain  - Directory
       2 - 0x    2 |     0x00000208 |    0x00000600 |  0xFFFFFFFE | End of Chain  - Mini FAT
       3 - 0x    3 |     0x0000020C |    0x00000800 |  0x       4 | Data
       4 - 0x    4 |     0x00000210 |    0x00000A00 |  0xFFFFFFFE | End of Chain
```

Sector 0 (after the header, so in fact sector 1 of the file, starting at 512th byte, or 0x200) is a FAT sector, as indicated by the first (and only) entry of the DIFAT.
Sector 1 is a sector containing directory entries (see below) and is the only one; the content of the FAT for sector 1 is End of Chain.
Sector 2 is a sector containing mini FAT entries (see below) and is the only one.
Sector 3 contains user data, and its follower is sector 4, which is the last in the data chain.
All other entries are unallocated, which means the file only contains 6 sectors (including the header), so its size should be 6 x 512 bytes = 3072 bytes.

```
remnux@remnux:~/Docwithinsertedtext/word/embeddings$ ls -l oleObject1.bin
-rw-rw-r-- 1 remnux remnux 3072 Jan  1  1980 oleObject1.bin
```

A discrepancy between the on-disk size and the size reported by the FAT is a warning sign. It is possible that a hidden payload is contained in the file, and you should examine the sectors belonging to the file and not mapped in the FAT.

You can use the ***olemap --fat*** command to dump the FAT.

```
remnux@remnux:~/Docwithinsertedtext/word/embeddings$ olemap --fat oleObject1.bin
olemap 0.55 - http://decalage.info/python/oletools
-------------------------------------------------------------------------------
FILE: oleObject1.bin

FAT:
+--------+-----------+--------+--------+
|Sector #|Type       |Offset  |Next #  |
+--------+-----------+--------+--------+
|       0|FAT Sector |00000200|FFFFFFFD|
|       1|End of Chain|00000400|FFFFFFFE|
|       2|End of Chain|00000600|FFFFFFFE|
|       3|<Data>     |00000800|       4|
|       4|End of Chain|00000A00|FFFFFFFE|
+--------+-----------+--------+--------+
```

Directory entries

We know now where the data is in the file, we know which sectors belong to the DIFAT and FAT, but we don't know yet to which user defined objects the other data belong.

To discover which objects are stored in the file, we must decode the directory entries.

A directory entry contains, among other elements, the name (in UTF-16) and type of the object, i.e., whether it is a stream (a file) or a storage. The following extract of the Microsoft documentation specifies the details of a directory entry:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">Directory Entry Name (64 bytes)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">Directory Entry Name Length</td><td colspan="8">Object Type</td><td colspan="8">Color Flag</td></tr>
<tr><td colspan="32">Left Sibling ID</td></tr>
<tr><td colspan="32">Right Sibling ID</td></tr>
<tr><td colspan="32">Child ID</td></tr>
<tr><td colspan="32">CLSID (16 bytes)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">State Bits</td></tr>
<tr><td colspan="32">Creation Time</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">Modified Time</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">Starting Sector Location</td></tr>
<tr><td colspan="32">Stream Size</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

As illustrated in a section above, the storage and streams hierarchy is organized as a tree. Storages can contain streams and storages, which in turn can also contain streams and other storages and so on. Therefore, the directory entries are also organized in trees. Each directory entry will contain a pointer to its predecessor in the same storage, a pointer to its follower in the same storage, and if the object is a storage, a pointer to its first contained object. It will also give the object sizen and a pointer to the object content if it is a stream.

The first directory entry, whose sector number is given by the First Directory Sector Location value in the file header, is the Root Directory entry. Its name is "Root Entry". Its CLSID can be used for OLE activation of the object's application.

The CLSID is a GUID, defined by rfc4122. In short, the interpretation of the 16 bytes on a little endian machine is the following:

```
# Address:  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
```

```
# Order:    03 02 01 00 05 04 07 06 08 09 0A 0B 0C 0D 0E 0F
```

The code to dump the directory entries will first build the sector chain for the directory entries. Starting with the First Directory Sector ID value in the header, we will follow the FAT to build the sector chain. Then, we will divide each sector in this chain into 128-byte chunks (the length of a directory entry) and make a list of them. Note that, if the number of directory entries is not an exact multiple of the number of sectors, there can be unallocated directory entries in the list.

To build a linear list and print the directory entries, we start with the Root Entry, and we use a classical recursive tree traversal algorithm to walk all the entries, following the left and right sibling pointers, and the child pointer if it is a storage. The directory entry names are coded in UTF16 (each character takes 2 bytes), so we translate them to ASCII to print them.

The unallocated directory entries, since they are not linked by other entries, are not inserted in the tree, but are still present in the list of directory entries.

```
# Parse CLSID
# CLSID is a mixed endian array
# Address:  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
# Order:    03 02 01 00 05 04 07 06 08 09 0A 0B 0C 0D 0E 0F
def parse_clsid(P_clsid):
    clsid2=""
    for i in range(4):
        clsid2 += "{0:02X}".format(P_clsid[3-i])
    clsid2 += "-"
    for i in range(5, 3, -1):
        clsid2 += "{0:02X}".format(P_clsid[i])
    clsid2 += "-"
    for i in range(7, 5, -1):
        clsid2 += "{0:02X}".format(P_clsid[i])
    clsid2 += "-"
    for i in range(8,10):
        clsid2 += "{0:02X}".format(P_clsid[i])
    clsid2 += "-"
    for i in range(10,16):
        clsid2 += "{0:02X}".format(P_clsid[i])
    return(clsid2)

# Parse directory entries (recursively)
# The directory entries are organized in a tree
# Each entry can have a left and a right sibling
# and storages can have a child
#                   +-----------------------+
#                   | dir entry2 (storage) |
#  dir entry 1 <--- | left sibling         |
#                   | right sibling        |----> dir entry 3
#                   | child --+            |
#                   +---------|------------+
#                             V
#                       dir entry 4
# We will recursively traverse the tree, starting with each left sibling until end
of chain
# then dumping the current entry, then the right sibling, then the potential child
#

def dump_entry(P_DirEntries, P_Index, P_Indent = 0):
    # Parse directory entry
    DirEntry = P_DirEntries[P_Index]
    myOffset = DirEntry["offset"]
    directory_entry_name = DirEntry["data"][0:64]
    directory_entry_name_length = struct.unpack_from('<H', DirEntry["data"],64)[0]
    object_type = struct.unpack_from('<c', DirEntry["data"],66)[0]
    color_flag = struct.unpack_from('<c', DirEntry["data"],67)[0]
    left_sibling = struct.unpack_from('<I', DirEntry["data"],68)[0]
```

```
    right_sibling = struct.unpack_from('<I', DirEntry["data"],72)[0]
    child_id  = struct.unpack_from('<I', DirEntry["data"],76)[0]
    clsid = DirEntry["data"][80:96]
    state_bits = struct.unpack_from('<I', DirEntry["data"],96)[0]
    creation_time = struct.unpack_from('<Q', DirEntry["data"],100)[0]
    modified_time = struct.unpack_from('<Q', DirEntry["data"],108)[0]
    starting_sector_location = struct.unpack_from('<I', DirEntry["data"],116)[0]
    stream_size = struct.unpack_from('<Q', DirEntry["data"],120)[0]

    # Add attributes
    P_DirEntries[P_Index]["start"] = starting_sector_location
    P_DirEntries[P_Index]["type"] = object_type
    P_DirEntries[P_Index]["id"] = P_Index
    P_DirEntries[P_Index]["size"] = stream_size

    # If left sibling exists, dump it
    if left_sibling != UNALLOCATED:
        dump_entry(P_DirEntries, left_sibling, P_Indent)

    # Print directory entry - UTF16 but can contain non printable chars
    print("0x{0:>08X} ".format(myOffset), end="|")
    print("{0:>3d} ".format(P_Index), end="|")
    myDirname = directory_entry_name.decode("utf-16le").rstrip("\x00")
    myDirname = P_Indent * "   " + myDirname
    myLen = 0
    for i in range(len(myDirname)):
        if myDirname[i].isprintable():
            print(myDirname[i], end="")
            myLen += 1
        else:
            print("x\{0:02X}".format(ord(myDirname[i])), end="")
            myLen += 4
    for i in range(32 - myLen):
        print(" ", end="")
    print("|", end="")
    if object_type == b'\x00':
        print(" Unalloc ", end="|")
    elif object_type == b'\x01':
        print(" Storage ", end="|")
    elif object_type == b'\x02':
        print(" Stream  ", end="|")
    elif object_type == b'\x05':
        print(" Root    ", end="|")
    else:
        print(" Unknown ", end="|")
    print(" 0x{0:08X} ".format(stream_size), end="|")
    print(" 0x{0:>8X} ".format(starting_sector_location), end="|")
    print(" 0x{0:>8X} ".format(child_id), end="|")
    print(" 0x{0:>8X} ".format(left_sibling), end="|")
    print(" 0x{0:>8X} ".format(right_sibling), end="|")
    clsid2 = parse_clsid(clsid)
    if clsid2 != "00000000-0000-0000-0000-000000000000":
        print(clsid2)
    else:
        print()

    # If right sibling exists, dump it
    if right_sibling != UNALLOCATED:
        dump_entry(P_DirEntries, right_sibling, P_Indent)
    # If it is a storage entry, dump child - start with root storage
    if object_type == b'\x05' and child_id != 0:
        dump_entry(P_DirEntries, child_id, P_Indent)
    else:
        if object_type == b'\x01' and child_id != 0:
            dump_entry(P_DirEntries, child_id, P_Indent + 1)
    return(P_DirEntries)
```

```
        # Build directory sector chain
```

```
        # Start with the first directory sector number in the header
        myDirSectorChain = [first_directory_sector_id]
        myDirectorySector = first_directory_sector_id
        # Follow the FAT to find the next sectors
        while myDirectorySector != ENDOFCHAIN:
            for i in range(len(myAllocatedSectors)):
            # Find next directory sector from the FAT
                if myAllocatedSectors[i]["number"] == myDirectorySector:
                    if myAllocatedSectors[i]["type"] == "Data":
                        myDirectorySector = myAllocatedSectors[i]["next"]
                        myDirSectorChain.append(myDirectorySector)
                    elif myAllocatedSectors[i]["type"] == "End of Chain":
                        myDirectorySector = ENDOFCHAIN
                    else:
                        myDirectorySector = ENDOFCHAIN
                        print("!!! Error in directory sector chain - unexpected
sector type")
                    break

        # Print directory sector chain
        print("======== Directory entries ========")
        print("Directory sector chain: [", end="")
        for i in range(len(myDirSectorChain)):
            print("0x{0:X} ".format(myDirSectorChain[i]), end="")
        print("]")

        # Dump the directory entries for each directory sector
        print("=== Dump dir entries ===")
        myDirEntryList = []
        # Loop through all the directory sectors
        for i in range(len(myDirSectorChain)):
            myOffset = (myDirSectorChain[i] + 1) * mySectorSize
            f.seek(myOffset)
            # Each directory entry is 128 bytes
            for j in range(mySectorSize // 128):
                myDirEntryList.append({"data":f.read(128), "offset": myOffset + j *
128})
        print("    Offset   | Id | Name                                      |  Type    |    Size
| 1st sector | Child       | Left        |    Right     |  CLSID")
        # Dump the content of the directory entries
        dump_entry(myDirEntryList, 0, 0)
```

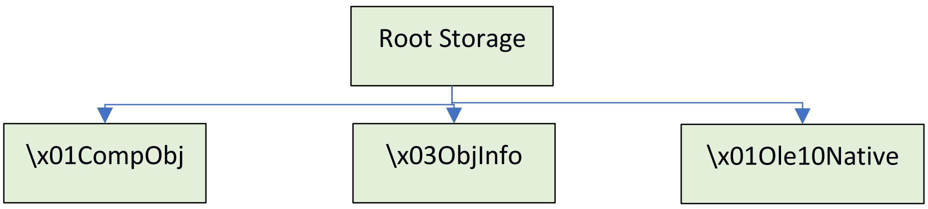
This is the result obtained with our example file.

```
======== Directory entries ========
Directory sector chain: [0x1 ]
=== Dump dir entries ===
   Offset   | Id | Name                          | Type   |    Size    | 1st sector | Child      | Left       |   Right    |  CLSID
0x00000400 |  0 |Root Entry                     | Root   | 0x00000380 | 0x       3 | 0x       2 | 0xFFFFFFFF | 0xFFFFFFFF |0003000C-0000-0000-C000-000000000046
0x00000480 |  1 |x\01CompObj                    | Stream | 0x0000004C | 0x       0 | 0xFFFFFFFF | 0xFFFFFFFF | 0xFFFFFFFF |
0x00000500 |  2 |x\03ObjInfo                    | Stream | 0x00000006 | 0x       2 | 0xFFFFFFFF | 0x       1 | 0x       3 |
0x00000580 |  3 |x\01Ole10Native                | Stream | 0x00000295 | 0x       3 | 0xFFFFFFFF | 0xFFFFFFFF | 0xFFFFFFFF |
```

There is only one storage, the Root Entry. Its child is entry number 2, x\03ObjInfo. The left sibling of this entry is entry 1, x\01CompObj, which has no other sibling. The right sibling of entry 1 is entry 3, x\01Ole10Native, which in turn has no other siblings.

The CLSID indicates that a generic object is embedded.

We can therefore represent the storage and streams tree like follows:



You can use the ***oledump.py --storages*** command to list all the directory entries, including the storages.

The 'O' indicates an embedded object.

Earlier, we wrote that the directory entry has a pointer to the stream content. It could be a sector number, and the stream would be a chain of sectors. But then all streams would be multiple of 512 bytes, and a lot of space could be wasted. There is a value in the file header called the Mini Stream Cutoff Size. If the size of the stream is larger than this, the stream content will be full sectors. If it is smaller,

```
remnux@remnux:~/Docwithinsertedtext/word/embeddings$ oledump.py --storages oleObject1.bin
  1: R          'Root Entry'
  2:          76 '\x01CompObj'
  3: O       661 '\x01Ole10Native'
  4:           6 '\x03ObjInfo'
```

then the content will be further sliced into mini sectors. The size of the mini sector is given by 2^ Mini Sector Shift in the header, normally 64 bytes. But as the mini sectors are smaller than full sectors, their allocation must be described by yet another structure: the mini FAT.

Mini FAT

The structure of the mini FAT is exactly the same as the FAT: it is an array of mini sectors, containing the mini sector number of the following mini sector, or End of Chain if this is the last mini sector of the stream.

The code to build the mini FAT sectors is rather straightforward: start with the first mini fat sector given by the First Mini FAT Sector Location in the header and follow the sector chain in the FAT to build the mini FAT list.

```
        # Mini sectors
        myMiniSectorSize = 2 ** mini_sector_shift
        # Get first minifat sector
        myMiniFatSectorId = first_mini_fat_sector_id
        myMiniFatEntries = []
        while myMiniFatSectorId != ENDOFCHAIN:
            # Compute offset to Mini FAT sector
            myOffset = (myMiniFatSectorId + 1) * mySectorSize
            f.seek(myOffset)
            # Each Mini FAT sector entry is 4 bytes
            # and represents the next mini sector of the current mini sector
            for i in range(mySectorSize // 4):
                myNextMiniSector = struct.unpack('<I', f.read(4))[0]
                myMiniFatEntries.append(myNextMiniSector)
            # Find next minifat sector
            for i in range(len(myAllocatedSectors)):
                if myAllocatedSectors[i]["number"] == myMiniFatSectorId:
                    if myAllocatedSectors[i]["type"] == "Data":
                        myMiniFatSectorId = myAllocatedSectors[i]["next"]
                    elif myAllocatedSectors[i]["type"] == "End of Chain":
                        myMiniFatSectorId = ENDOFCHAIN
                    else:
                        myMiniFatSectorId = ENDOFCHAIN
                        print("!!! Error in MiniFAT sector chain - unexpected
sector type")
                    break

        print("====== Mini FAT map ======")
        print("Mini sector |   Next")
        # Find last non-free sector
        myMaxSector = 0
        for i in range(len(myMiniFatEntries)):
            if myMiniFatEntries[i] != UNALLOCATED:
```

```
            myMaxSector = i
    # Print allocated mini FAT entries
    if len(myMiniFatEntries) > 0:
        for i in range(myMaxSector + 1):
            print(" 0x{0:>8X} | 0x{1:>8X}".format(i, myMiniFatEntries[i]))
```

The result on our example file is the following:

```
====== Mini FAT map ======
Mini sector |    Next
 0x        0 | 0x        1
 0x        1 | 0xFFFFFFFE
 0x        2 | 0xFFFFFFFE
 0x        3 | 0x        4
 0x        4 | 0x        5
 0x        5 | 0x        6
 0x        6 | 0x        7
 0x        7 | 0x        8
 0x        8 | 0x        9
 0x        9 | 0x        A
 0x        A | 0x        B
 0x        B | 0x        C
 0x        C | 0x        D
 0x        D | 0xFFFFFFFE
```

We have indeed 3 streams: one corresponding to mini sectors 0 and 1, one having only mini sector 2, and the third one covering the mini sectors 3 to 13 (0xD).
You can view the mini FAT with the ***olemap --minifat*** command.

```
remnux@remnux:~/Docwithinsertedtext/word/embeddings$ olemap --minifat oleObject1.bin
olemap 0.55 - http://decalage.info/python/oletools
-------------------------------------------------------------------------------
FILE: oleObject1.bin

MiniFAT:
+--------+-----------+--------+--------+
|Sector #|Type       |Offset  |Next #  |
+--------+-----------+--------+--------+
|       0|Data       |N/A     |       1|
|       1|End of Chain|N/A    |FFFFFFFE|
|       2|End of Chain|N/A    |FFFFFFFE|
|       3|Data       |N/A     |       4|
|       4|Data       |N/A     |       5|
|       5|Data       |N/A     |       6|
|       6|Data       |N/A     |       7|
|       7|Data       |N/A     |       8|
|       8|Data       |N/A     |       9|
|       9|Data       |N/A     |       A|
|       A|Data       |N/A     |       B|
|       B|Data       |N/A     |       C|
|       C|Data       |N/A     |       D|
|       D|End of Chain|N/A    |FFFFFFFE|
+--------+-----------+--------+--------+
```

When we will dump the stream content, we will use the mini FAT to walk the mini sector chains.

Streams

Finally, we will get to the data by dumping the streams.

First, we read all the streams mini sectors to get all stream data in memory.

To do that, we start with the first stream sector number contained in the Root Directory entry and we read all the mini streams contained in this sector. A mini stream is 64 bytes, so we have to perform (sector size / 64) reads. Then, we go to the next stream sector by following the sector chain in the FAT, we read it, and so on.

To dump the content of the streams, we take each directory entry, and every time we encounter an entry of type stream, we take its starting sector location, dump the sector content, follow the sector chain to find the next one and so on. As explained in a section above, if the size of the entry is smaller than the Mini Stream Cutoff Size, the sector number is a mini sector number, and the chain is located in the mini FAT. If it is larger, the sector number is a 'normal' file sector, and the chain is to be followed in the FAT.

We also take care of not handling the unallocated

```
        # Streams
        print("====== Streams ======")
        # Read all streams data blocks, one mini sector at a time
        # Start with first sector of the mini stream stored in the Root Directory
entry
        myMiniStreamSector = myDirEntryList[0]["start"]
        myMiniStream = []
        myMiniStreamChain = []
        while myMiniStreamSector != ENDOFCHAIN :
            # Build the sector chain for the mini streams
            myMiniStreamChain.append(myMiniStreamSector)
            # Compute offset to sector containing mini stream
            myOffset = (myMiniStreamSector + 1) * mySectorSize
            f.seek(myOffset)
            # Read all the mini sectors in the sector
            for i in range(mySectorSize // myMiniSectorSize):
                myMiniStream.append({"data": f.read(myMiniSectorSize), "offset":
myOffset})
                myOffset += myMiniSectorSize
            # Find next data sector from FAT
            for i in range(len(myAllocatedSectors)):
                if myAllocatedSectors[i]["number"] == myMiniStreamSector:
                    if myAllocatedSectors[i]["type"] == "Data":
                        myMiniStreamSector = myAllocatedSectors[i]["next"]
                        # myMiniStreamChain.append(myMiniStreamSector)
                    elif myAllocatedSectors[i]["type"] == "End of Chain":
                        myMiniStreamSector = ENDOFCHAIN
                    else:
                        myMiniStreamSector = ENDOFCHAIN
                        print("!!! Error in Mini streams sector chain - unexpected
sector type")
                    break
        # Print streams sector chain
        print("Mini streams sector chain: [", end="")
        for i in range(len(myMiniStreamChain)):
            print("0x{0:X} ".format(myMiniStreamChain[i]), end="")
        print("]")
        # Dump the content of the data streams and mini streams
        for i in range(len(myDirEntryList)):
            # Ignore non allocated entries which were not handled by dump_entry
            if "type" in myDirEntryList[i]:
                # Only handle data streams
                if myDirEntryList[i]["type"] == b'\x02':
                    # Get starting sector number
                    myIndex = myDirEntryList[i]["start"]
                    # Get stream size
                    mySize = myDirEntryList[i]["size"]
                    print("Directory entry 0x{0:X} - size: {1} -
0x{1:X}".format(myDirEntryList[i]["id"],mySize))
```

```
                    if mySize < mini_stream_cutoff_size:
                        # Data is in the mini streams
                        # The starting sector number is a mini sector number
                        while myIndex != ENDOFCHAIN:
                            # Dump the data bytes, 16 per line
                            for j in range(myMiniSectorSize):
                                if j % 16 == 0:
                                    print("0x{0:08X}:
".format(myMiniStream[myIndex]["offset"] + j), end="")
                                    myAsciidump =""
                                myByte = myMiniStream[myIndex]["data"][j]
                                print("{0:02X}".format(myByte), end=" ")
                                if chr(myByte).isprintable():
                                    myAsciidump += chr(myByte)
                                else:
                                    myAsciidump += "."
                                if j % 16 == 15:
                                    print("  " + myAsciidump)
                            # Find the next mini sector in the mini FAT
                            myIndex = myMiniFatEntries[myIndex]
                    else:
                        # Data is in the normal sectors
                        # The starting sector number is a 'normal' sector
                        while myIndex != ENDOFCHAIN:
                            f.seek((1 + myIndex) * mySectorSize)
                            myData = f.read(mySectorSize)
                            for j in range(mySectorSize):
                                if j % 16 == 0:
                                    print("0x{0:08X}: ".format(((1 + myIndex) *
mySectorSize) + j), end="")
                                    myAsciidump =""
                                myByte = myData[j]
                                print("{0:02X}".format(myByte), end=" ")
                                if chr(myByte).isprintable():
                                    myAsciidump += chr(myByte)
                                else:
                                    myAsciidump += "."
                                if j % 16 == 15:
                                    print("  " + myAsciidump)
                            # Find next data sector in the FAT
                            for i in range(len(myAllocatedSectors)):
                                if myAllocatedSectors[i]["number"] == myIndex:
                                    if myAllocatedSectors[i]["type"] == "Data":
                                        myIndex = myAllocatedSectors[i]["next"]
                                    elif myAllocatedSectors[i]["type"] == "End of
Chain":
                                        myIndex = ENDOFCHAIN
                                    else:
                                        myIndex = ENDOFCHAIN
                                        print("!!! Error in data streams sector
chain - unexpected sector type")
                                    break
                print("---------------------------------")
```

The result with our example file is the following:

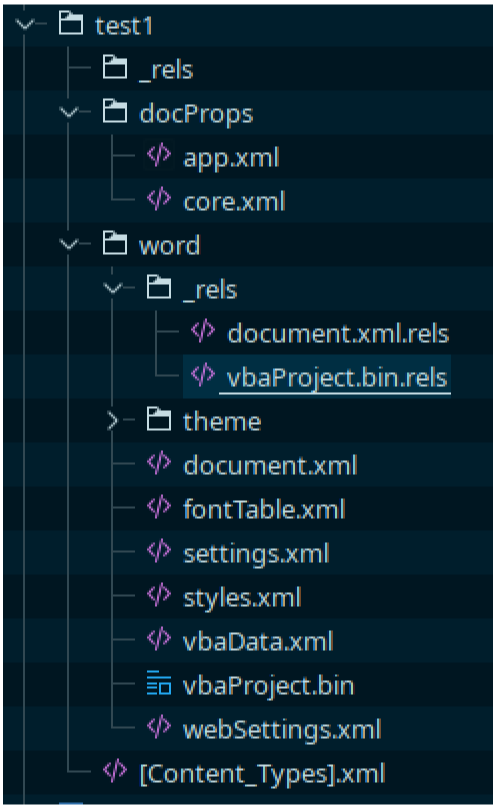
```
====== Streams ======
Mini streams sector chain: [0x3 0x4 ]
--------------------------------
Directory entry 0x1 - size: 76 - 0x4C
0x00000800: 01 00 FE FF 03 0A 00 00 FF FF FF FF 0C 00 03 00   ..þÿ....ÿÿÿÿ....
0x00000810: 00 00 00 00 C0 00 00 00 00 00 00 46 0C 00 00 00   ....À......F....
0x00000820: 4F 4C 45 20 50 61 63 6B 61 67 65 00 00 00 00 00   OLE Package.....
0x00000830: 08 00 00 00 50 61 63 6B 61 67 65 00 F4 39 B2 71   ....Package.ô9²q
0x00000840: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00000850: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00000860: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00000870: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
--------------------------------
Directory entry 0x2 - size: 6 - 0x6
0x00000880: 40 00 03 00 01 00 00 00 00 00 00 00 00 00 00 00   @...............
0x00000890: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x000008A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x000008B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
--------------------------------
Directory entry 0x3 - size: 661 - 0x295
0x000008C0: 91 02 00 00 02 00 74 65 78 74 2E 74 78 74 00 43   ......text.txt.C
0x000008D0: 3A 5C 55 73 65 72 73 5C 70 6C 63 5C 41 70 70 44   :\Users\plc\AppD
0x000008E0: 61 74 61 5C 4C 6F 63 61 6C 5C 4D 69 63 72 6F 73   ata\Local\Micros
0x000008F0: 6F 66 74 5C 57 69 6E 64 6F 77 73 5C 49 4E 65 74   oft\Windows\INet
0x00000900: 43 61 63 68 65 5C 43 6F 6E 74 65 6E 74 2E 57 6F   Cache\Content.Wo
0x00000910: 72 64 5C 74 65 78 74 2E 74 78 74 00 00 00 03 00   rd\text.txt.....
0x00000920: 77 00 00 00 43 3A 5C 55 73 65 72 73 5C 70 6C 63   w...C:\Users\plc
0x00000930: 5C 41 70 70 44 61 74 61 5C 4C 6F 63 61 6C 5C 54   \AppData\Local\T
0x00000940: 65 6D 70 5C 7B 41 34 39 41 43 44 44 43 2D 33 33   emp\{A49ACDDC-33
0x00000950: 35 46 2D 34 32 37 43 2D 38 45 31 45 2D 30 42 41   5F-427C-8E1E-0BA
0x00000960: 39 35 35 35 39 44 32 44 32 7D 5C 7B 34 38 46 38   95559D2D2}\{48F8
0x00000970: 39 39 34 37 2D 38 37 38 36 2D 34 46 39 33 2D 41   9947-8786-4F93-A
0x00000980: 34 33 33 2D 30 31 34 38 31 30 45 41 31 34 38 44   433-014810EA148D
0x00000990: 7D 5C 74 65 78 74 2E 74 78 74 00 16 00 00 00 54   }\text.txt.....T
0x000009A0: 68 69 73 20 69 73 20 61 20 74 65 78 74 20 66 69   his is a text fi
0x000009B0: 6C 65 2E 0D 0A 76 00 00 00 43 00 3A 00 5C 00 55   le...v...C.:.\.U
0x000009C0: 00 73 00 65 00 72 00 73 00 5C 00 70 00 6C 00 63   .s.e.r.s.\.p.l.c
0x000009D0: 00 5C 00 41 00 70 00 70 00 44 00 61 00 74 00 61   .\.A.p.p.D.a.t.a
0x000009E0: 00 5C 00 4C 00 6F 00 63 00 61 00 6C 00 5C 00 54   .\.L.o.c.a.l.\.T
0x000009F0: 00 65 00 6D 00 70 00 5C 00 7B 00 41 00 34 00 39   .e.m.p.\.{.A.4.9
0x00000A00: 00 41 00 43 00 44 00 44 00 43 00 2D 00 33 00 33   .A.C.D.D.C.-.3.3
0x00000A10: 00 35 00 46 00 2D 00 34 00 32 00 37 00 43 00 2D   .5.F.-.4.2.7.C.-
0x00000A20: 00 38 00 45 00 31 00 45 00 2D 00 30 00 42 00 41   .8.E.1.E.-.0.B.A
0x00000A30: 00 39 00 35 00 35 00 35 00 39 00 44 00 32 00 44   .9.5.5.5.9.D.2.D
0x00000A40: 00 32 00 7D 00 5C 00 7B 00 34 00 38 00 46 00 38   .2.}.\.{.4.8.F.8
0x00000A50: 00 39 00 39 00 34 00 37 00 2D 00 38 00 37 00 38   .9.9.4.7.-.8.7.8
0x00000A60: 00 36 00 2D 00 34 00 46 00 39 00 33 00 2D 00 41   .6.-.4.F.9.3.-.A
0x00000A70: 00 34 00 33 00 33 00 2D 00 30 00 31 00 34 00 38   .4.3.3.-.0.1.4.8
0x00000A80: 00 31 00 30 00 45 00 41 00 31 00 34 00 38 00 44   .1.0.E.A.1.4.8.D
0x00000A90: 00 7D 00 5C 00 74 00 65 00 78 00 74 00 2E 00 74   .}.\.t.e.x.t...t
0x00000AA0: 00 78 00 74 00 08 00 00 00 74 00 65 00 78 00 74   .x.t.....t.e.x.t
```

You can use the ***oledump.py -s <stream>*** to dump each stream:

```
remnux@remnux:~/Docwithinsertedtext/word/embeddings$ oledump.py -s 1 oleObject1.bin
00000000: 01 00 FE FF 03 0A 00 00  FF FF FF FF 0C 00 03 00  ................
00000010: 00 00 00 00 C0 00 00 00  00 00 00 46 0C 00 00 00  ...........F....
00000020: 4F 4C 45 20 50 61 63 6B  61 67 65 00 00 00 00 00  OLE Package.....
00000030: 08 00 00 00 50 61 63 6B  61 67 65 00 F4 39 B2 71  ....Package..9.q
00000040: 00 00 00 00 00 00 00 00  00 00 00 00              ............
```

Document with macros

A *.docm* Word document containing macros has the same structure as a simple document, with some additional files containing the description of the code, the macro code itself, and a relationship file.



Here, our *test1.docm* document contains the *word\rels\vbaproject.bin.rels*, *word\vbaData.xml* and *vbaProject.bin*. The presence of this kind of files should immediately warn you about a potential danger of active code.

The inserted macro is very simple:

```
Sub AutoOpen()
	Megabox "Hello, world"
EndSub
```

It is meant to display the Hello, world message in a message box when the document is opened (and, of course, if the macros are enabled in Word, which should not be the case in a secure environment).

Content_Types

As expected, the *[Content_Types].xml* file contains lines specifying the MIME types of the document, the macro files and listing the xml files describing them:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
  <Default Extension="bin" ContentType="application/vnd.ms-office.vbaProject" />
  <Default Extension="rels" ContentType="application/vnd.openxmlformats-package.relationships+xml" />
  <Default Extension="xml" ContentType="application/xml" />
  <Override PartName="/word/document.xml" ContentType="application/vnd.ms-word.document.macroEnabled.main+xml" />
```

```
  <Override PartName="/word/vbaData.xml" ContentType="application/vnd.ms-word.vbaData+xml" />
  <Override PartName="/word/styles.xml" ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.styles+xml" />
  <Override PartName="/word/settings.xml" ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.settings+xml" />
  <Override PartName="/word/webSettings.xml" ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.webSettings+xml" />
  <Override PartName="/word/fontTable.xml" ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.fontTable+xml" />
  <Override PartName="/word/theme/theme1.xml" ContentType="application/vnd.openxmlformats-officedocument.theme+xml" />
  <Override PartName="/docProps/core.xml" ContentType="application/vnd.openxmlformats-package.core-properties+xml" />
  <Override PartName="/docProps/app.xml" ContentType="application/vnd.openxmlformats-officedocument.extended-properties+xml" />
</Types>
```

The type of our document is now `application/vnd.ms-word.document.macroEnabled.main+xml`.

vbaProject.bin.rels

This additional file contains the relationship between the document and the macro files.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId1" Type="http://schemas.microsoft.com/office/2006/relationships/wordVbaData" Target="vbaData.xml" />
</Relationships>
```

vbaData.xml

This file contains a very long list of xml schemas used by macros, and the name of the macro, in the present case PROJECT.NEWMACROS.AUTOOPEN.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<wne:vbaSuppData
xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas" xmlns:cx="http://schemas.microsoft.com/office/drawing/2014/chartex"
xmlns:cx1="http://schemas.microsoft.com/office/drawing/2015/9/8/chartex"
xmlns:cx2="http://schemas.microsoft.com/office/drawing/2015/10/21/chartex"
xmlns:cx3="http://schemas.microsoft.com/office/drawing/2016/5/9/chartex"
xmlns:cx4="http://schemas.microsoft.com/office/drawing/2016/5/10/chartex"
xmlns:cx5="http://schemas.microsoft.com/office/drawing/2016/5/11/chartex"
xmlns:cx6="http://schemas.microsoft.com/office/drawing/2016/5/12/chartex"
xmlns:cx7="http://schemas.microsoft.com/office/drawing/2016/5/13/chartex"
xmlns:cx8="http://schemas.microsoft.com/office/drawing/2016/5/14/chartex"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:aink="http://schemas.microsoft.com/office/drawing/2016/ink"
xmlns:am3d="http://schemas.microsoft.com/office/drawing/2017/model3d"
xmlns:o="urn:schemas-microsoft-com:office:office"
```

```
xmlns:oel="http://schemas.microsoft.com/office/2019/extlst"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationship
s" xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDra
wing"
xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDr
awing" xmlns:w10="urn:schemas-microsoft-com:office:word"
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
xmlns:w16cex="http://schemas.microsoft.com/office/word/2018/wordml/cex"
xmlns:w16cid="http://schemas.microsoft.com/office/word/2016/wordml/cid"
xmlns:w16="http://schemas.microsoft.com/office/word/2018/wordml"
xmlns:w16sdtdh="http://schemas.microsoft.com/office/word/2020/wordml/sdtdat
ahash"
xmlns:w16se="http://schemas.microsoft.com/office/word/2015/wordml/symex"
xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGrou
p"
xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocessingInk"
xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShap
e" mc:Ignorable="w14 w15 w16se w16cid w16 w16cex w16sdtdh wp14">
  <wne:mcds>
  <wne:mcd wne:macroName="PROJECT.NEWMACROS.AUTOOPEN"
  wne:name="Project.NewMacros.AutoOpen" wne:bEncrypt="00" wne:cmg="56" />
    </wne:mcds>
</wne:vbaSuppData>
```

vbaProject.bin

This binary file contains the code of the macro.

Depending on the platform and default settings, the characters can be encoded as ASCII or Unicode characters. Between what appears as garbled text, you can find various strings referencing the Office libraries used, the name and content of the macros.

It also starts with the classical 8-byte OLE header:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

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  ................
```

Analysis of this binary file gives the following:

```
remnux@remnux:~/test1/word$ oledump.py vbaProject.bin
  1:       413 'PROJECT'
  2:        71 'PROJECTwm'
  3: M    1196 'VBA/NewMacros'
  4: m     932 'VBA/ThisDocument'
  5:      2576 'VBA/_VBA_PROJECT'
  6:      1128 'VBA/__SRP_0'
  7:       104 'VBA/__SRP_1'
  8:        84 'VBA/__SRP_2'
  9:       107 'VBA/__SRP_3'
 10:       578 'VBA/dir'
```

There are 2 streams and one storage in the root directory: PROJECT, PROJECTwm and VBA. The VBA storage contains the NewMacros, ThisDocument, _VBA_PROJECT, __SRP_0, __SRP_1, __SRP_2, __SRP_3 and dir streams.

The ***PROJECT*** stream specifies the project properties.

```
remnux@remnux:~/test1/word$ oledump.py -s 2 vbaProject.bin
00000000: 54 68 69 73 44 6F 63 75  6D 65 6E 74 00 54 00 68  ThisDocument.T.h
00000010: 00 69 00 73 00 44 00 6F  00 63 00 75 00 6D 00 65  .i.s.D.o.c.u.m.e
00000020: 00 6E 00 74 00 00 00 4E  65 77 4D 61 63 72 6F 73  .n.t...NewMacros
00000030: 00 4E 00 65 00 77 00 4D  00 61 00 63 00 72 00 6F  .N.e.w.M.a.c.r.o
00000040: 00 73 00 00 00 00 00                              .s.....
```

The ***PROJECTwm*** stream lists the module names in ASCII and UTF-16.
The ***_VBA_PROJECT*** stream specifies the VBA version used to create the project and some performance cache data.

The ***dir*** stream specifies the project properties, references and module properties.

The ***__SRP_**** streams specify performance cache.

The ***NewMacros*** and ***ThisDocument*** streams are the user created module streams, with NewMacros containing the code, as indicated by the preceding M.

The VBA code itself is compressed by a run-length algorithm.
You can decompress the VBA code to clear text with the ***oledump.py -v*** command:

```
remnux@remnux:~/test1/word$ oledump.py -s 3 -v vbaProject.bin
Attribute VB_Name = "NewMacros"
Sub AutoOpen()
Attribute AutoOpen.VB_ProcData.VB_Invoke_Func = "Project.NewMacros.AutoOpen"
'
' AutoOpen Macro
'
'
MsgBox "Hello, world."
End Sub
```

Summary

To safely analyze suspicious Office >= 2007 documents, here are some general tips:

- Check the content with an unzipper (unzip, 7zip, pkzip…)
- Check for embedded code (often in *.bin* files)
- Use *oledump.py* to locate streams containing embedded OLE files and VBA macros
- Check the on-disk size and the size reported by the FAT match to detect hidden payload. Use *olemap –fat* and *oledump.py -u*

- Decode and deobfuscate the executable code