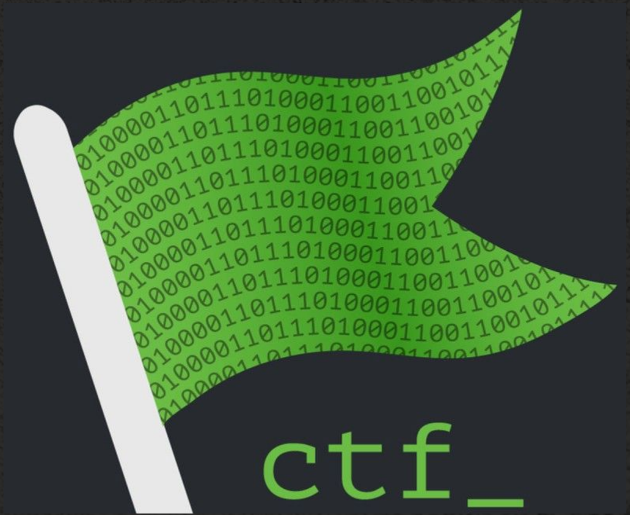
CTF Workshop

Digital Forensic

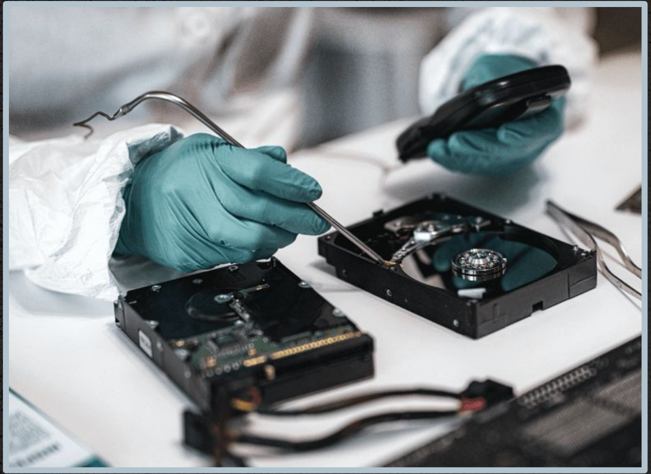
1. File Forensic
2. Image Forensic
3. Office File Forensic

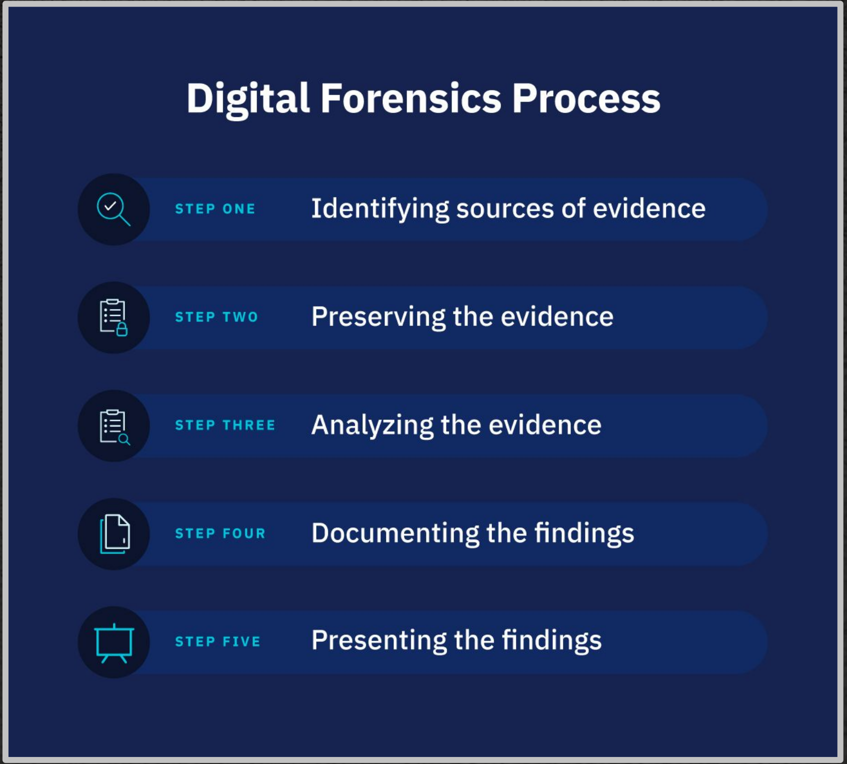


./forensic_intro

Forensic is the activity of recovering digital trail left on device or network.

Many methods to find data which was deleted, not stored, or worse covertly recorded.

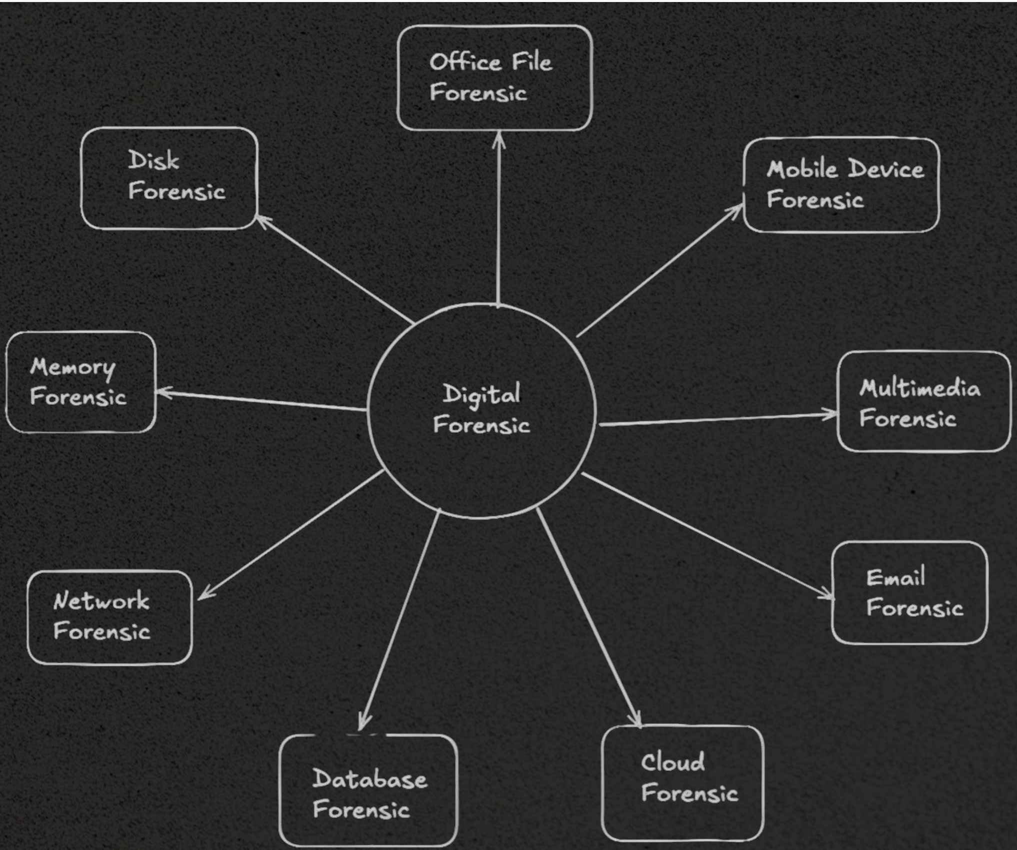




Digital Forensics Process

- STEP ONE Identifying sources of evidence
- STEP TWO Preserving the evidence
- STEP THREE Analyzing the evidence
- STEP FOUR Documenting the findings
- STEP FIVE Presenting the findings

./forensic_intro



./forensic_intro

Usually some similar themes:

- Look for little weird tricks
  - Can a zip file appended to JPEG ?
  - Can a file both a PDF and an exe ?

- Application of off-the-shelf software
  - Oh it's a dump of virtual memory
  - There's a Python script somewhere to parses dump of virtual memory to rebuild all process memory from PTEs

- File Format Identification
  - Magic bytes, header data and trailer data (89 50 4E 47)
  - Corrupted file hex signature

- Filesystem (Disk Image), PCAP, Memory Dump, Syslog and etc

./forensic_archive_files

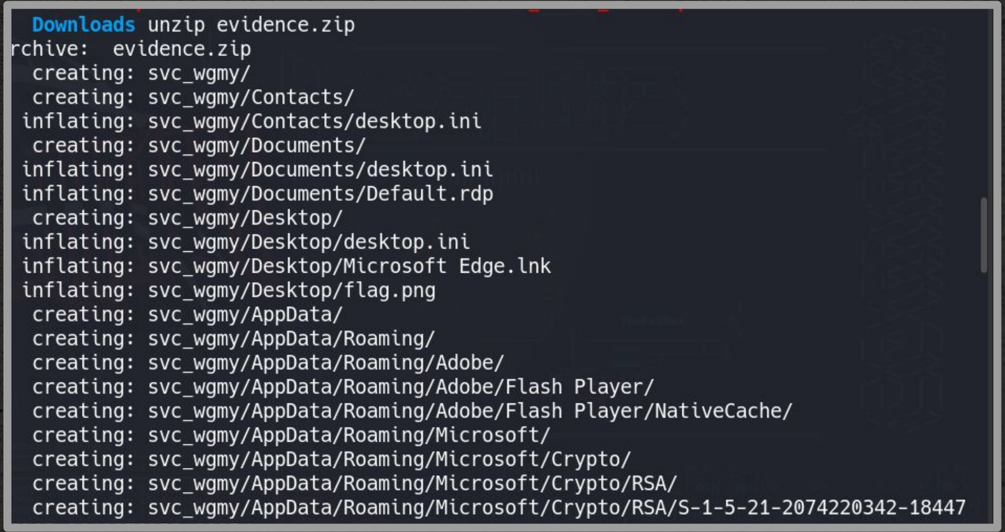
- CTF Challenges usually contained in a zip, 7z, rar, tar or tgz file
- Goal: To extract a file from the archive and file the flag from a file that is embedded or hidden

1. Zip file
   - $ unzip
   - $ zipdetails -v
   - $ zipinfo

2. RAR file
   - $ unrar x

3. 7z file
   - $ 7z x

4. tar.gz file
   - $ tar xzvf



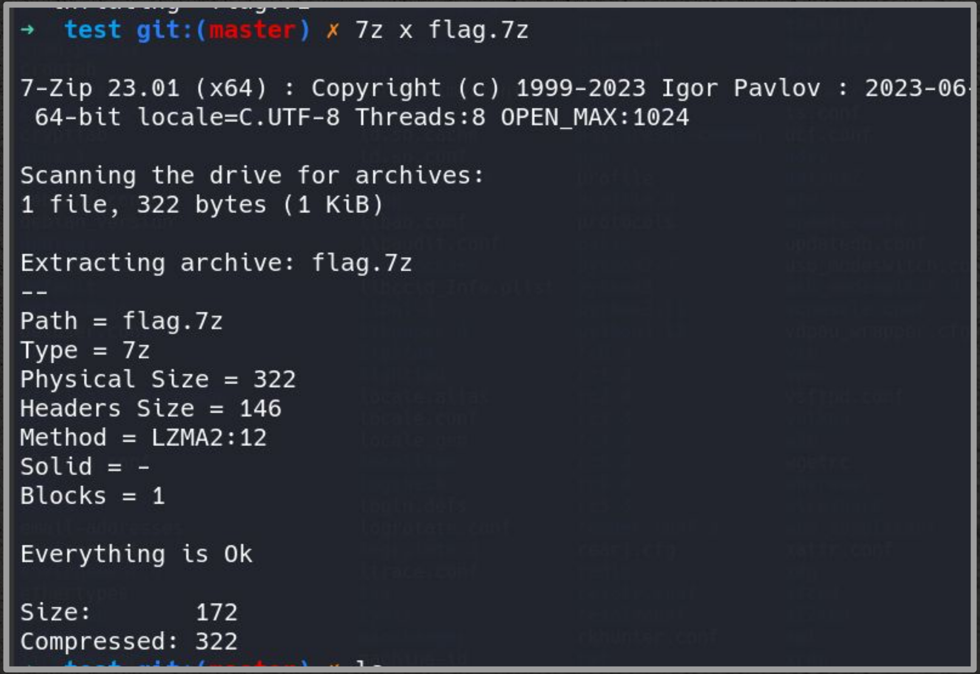
```
  Downloads unzip evidence.zip
rchive:   evidence.zip
   creating: svc_wgmy/
   creating: svc_wgmy/Contacts/
  inflating: svc_wgmy/Contacts/desktop.ini
   creating: svc_wgmy/Documents/
  inflating: svc_wgmy/Documents/desktop.ini
  inflating: svc_wgmy/Documents/Default.rdp
   creating: svc_wgmy/Desktop/
  inflating: svc_wgmy/Desktop/desktop.ini
  inflating: svc_wgmy/Desktop/Microsoft Edge.lnk
  inflating: svc_wgmy/Desktop/flag.png
   creating: svc_wgmy/AppData/
   creating: svc_wgmy/AppData/Roaming/
   creating: svc_wgmy/AppData/Roaming/Adobe/
   creating: svc_wgmy/AppData/Roaming/Adobe/Flash Player/
   creating: svc_wgmy/AppData/Roaming/Adobe/Flash Player/NativeCache/
   creating: svc_wgmy/AppData/Roaming/Microsoft/
   creating: svc_wgmy/AppData/Roaming/Microsoft/Crypto/
   creating: svc_wgmy/AppData/Roaming/Microsoft/Crypto/RSA/
   creating: svc_wgmy/AppData/Roaming/Microsoft/Crypto/RSA/S-1-5-21-2074220342-18447
```

./forensic_archive_files

5. XZ file
    - $ xz -d

6. bz2 file
    - $ bzip2 -d

7. gzip file
    - $ gzip -d

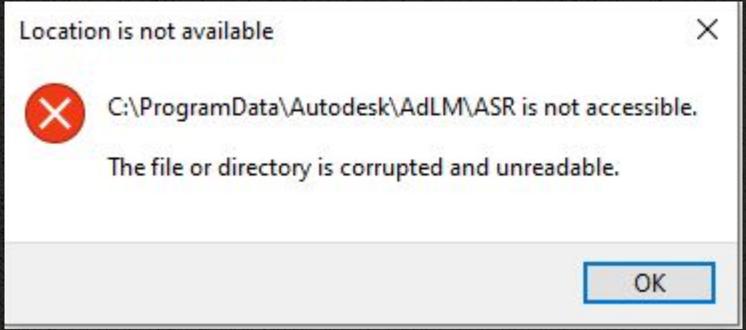


```
→  test git:(master) ✗ 7z x flag.7z

7-Zip 23.01 (x64) : Copyright (c) 1999-2023 Igor Pavlov : 2023-06
 64-bit locale=C.UTF-8 Threads:8 OPEN_MAX:1024

Scanning the drive for archives:
1 file, 322 bytes (1 KiB)

Extracting archive: flag.7z
--
Path = flag.7z
Type = 7z
Physical Size = 322
Headers Size = 146
Method = LZMA2:12
Solid = -
Blocks = 1

Everything is Ok

Size:       172
Compressed: 322
```

./forensic_file_analysis

What is File Forensic:

- The practise of analyzing digital files to recover evidence or understand file properties and contents



Location is not available

C:\ProgramData\Autodesk\AdLM\ASR is not accessible.

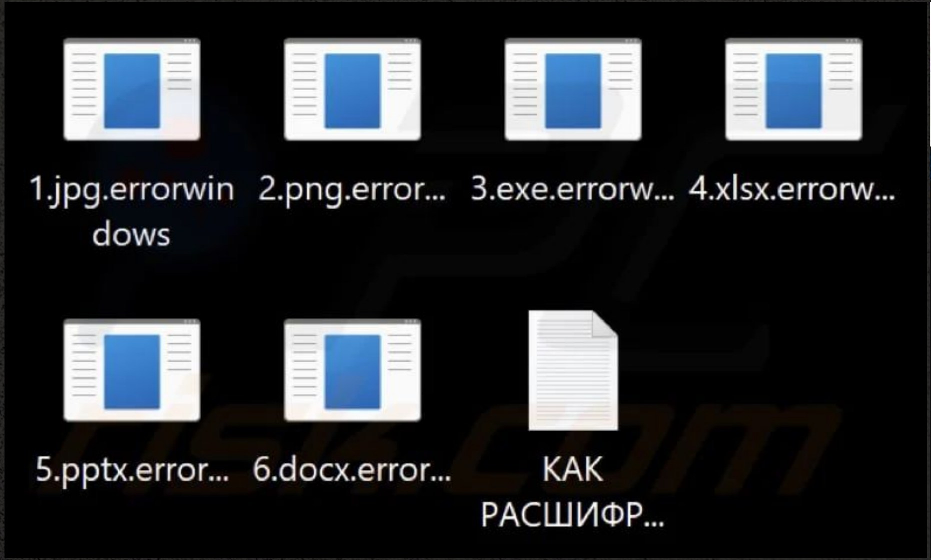
The file or directory is corrupted and unreadable.

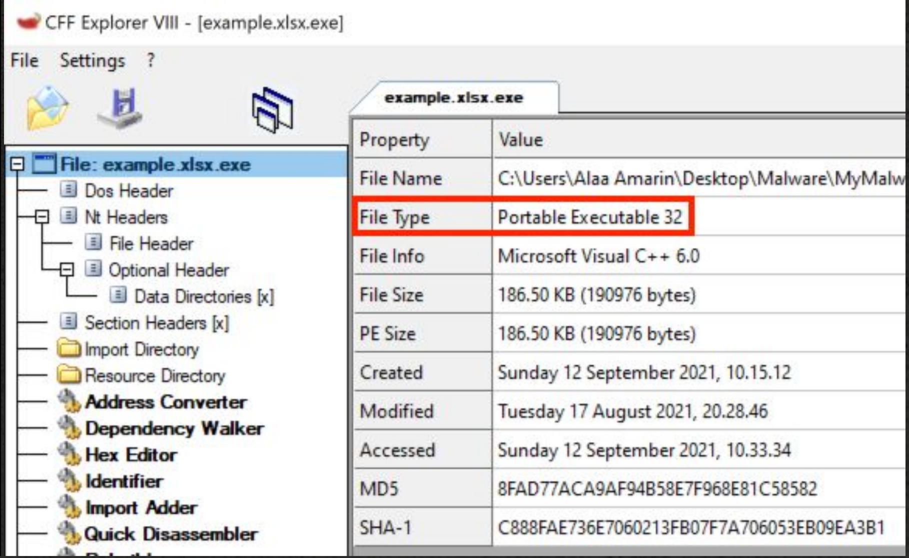
Purpose:

- Recover deleted or hidden information
- Understand file creation and modification details
- Identify malicious software or unauthorized changes

- Files can sometimes come without an extension, or with incorrect ones.
- File extensions aren’t reliable alone; file signatures, or magic numbers, accurately identify file types for consistent and correct data parsing

./forensic_file_analysis





Property	Value
File Name	C:\Users\Alaa Amarin\Desktop\Malware\MyMalw
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 6.0
File Size	186.50 KB (190976 bytes)
PE Size	186.50 KB (190976 bytes)
Created	Sunday 12 September 2021, 10.15.12
Modified	Tuesday 17 August 2021, 20.28.46
Accessed	Sunday 12 September 2021, 10.33.34
MD5	8FAD77ACA9AF94B58E7F968E81C58582
SHA-1	C888FAE736E7060213FB07F7A706053EB09EA3B1

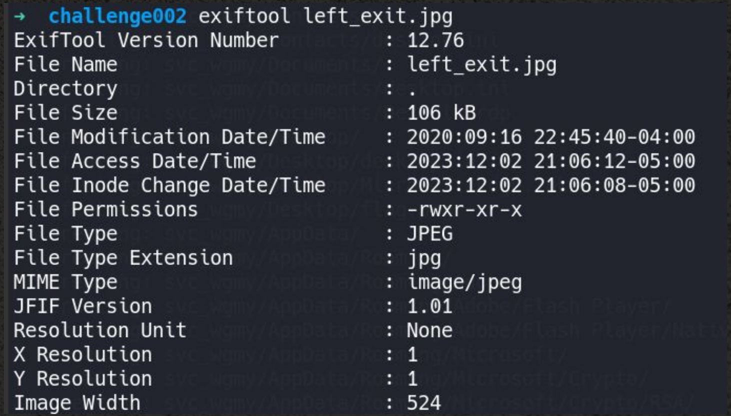
./forensic_file_analysis

Tools for file analysis:

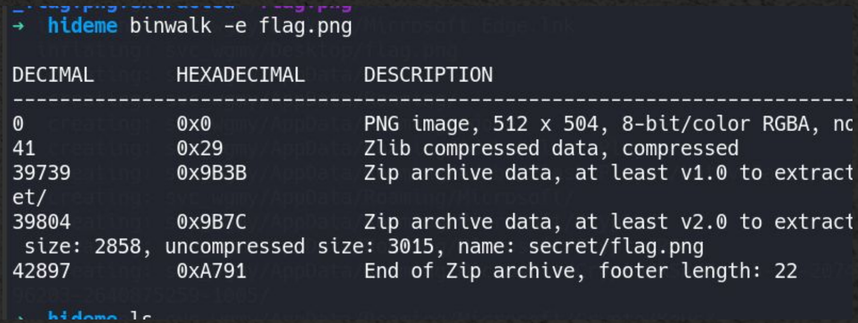
1. **$ exiftool**
- Extract all metadata of a digital file

2. **$ ghex** (for advanced use $ xxd)
- View, edit data from any file
- Also used by kids who cheat at computer games, by adding score or lives to saved games.

3. **$ binwalk**
- File extraction (embedded file within the main file)
- Signature Scanning (Magic Hex)
- To extract $ dd if=<input> of=<input> bs=<block size> skip=<read after certain bytes>



```
➜  challenge002 exiftool left_exit.jpg
ExifTool Version Number         : 12.76
File Name                       : left_exit.jpg
Directory                       : .
File Size                       : 106 kB
File Modification Date/Time     : 2020:09:16 22:45:40-04:00
File Access Date/Time           : 2023:12:02 21:06:12-05:00
File Inode Change Date/Time     : 2023:12:02 21:06:08-05:00
File Permissions                : -rwxr-xr-x
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : None
X Resolution                    : 1
Y Resolution                    : 1
Image Width                     : 524
```



```
➜  hideme binwalk -e flag.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 512 x 504, 8-bit/color RGBA, nc
41            0x29            Zlib compressed data, compressed
39739         0x9B3B          Zip archive data, at least v1.0 to extract
et/
39804         0x9B7C          Zip archive data, at least v2.0 to extract
 size: 2858, uncompressed size: 3015, name: secret/flag.png
42897         0xA791          End of Zip archive, footer length: 22
```

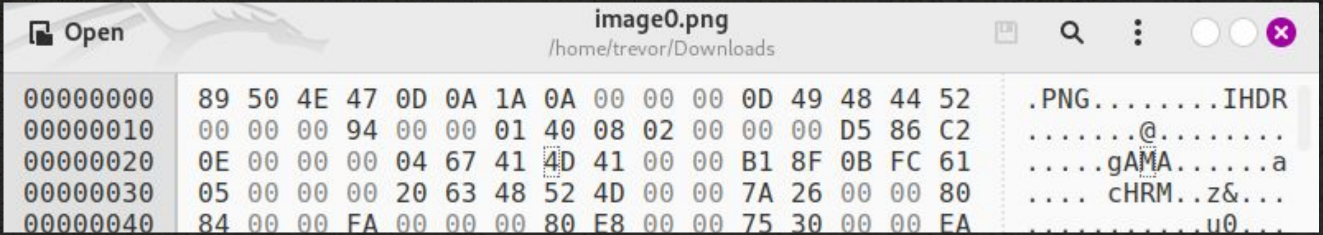
./forensic_file_analysis

File Signatures and Magic Hex

- Know the Magic Hex Signature (Header, Trailer, Body)
- Magic hex are typically 2-4 long, found at the beginning of a file
- https://gist.github.com/leommoore/f9e57ba2aa4bf197ebc5
- https://www.garykessler.net/library/file_sigs.html
- https://asecuritysite.com/forensics/png?file=%2Flog%2Fbasn0g01.png

Example: PNG Image

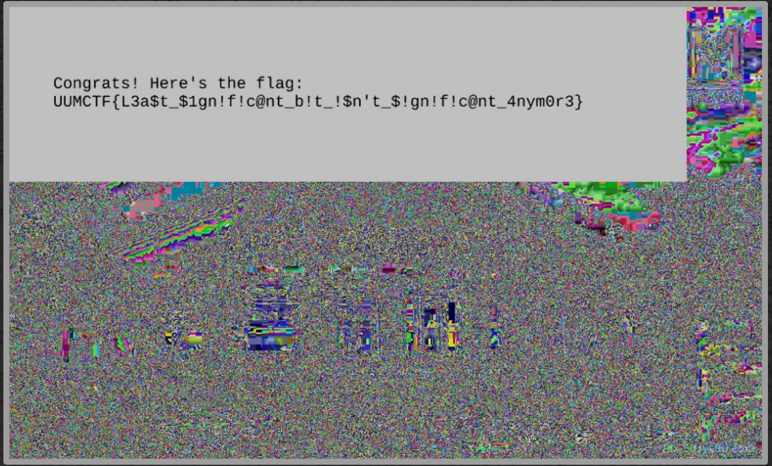
Header: 89 50 4E 47 (.PNG)
Trailer: AE 42 60 82 (IEND)



```
00000000  89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52   .PNG........IHDR
00000010  00 00 00 94 00 00 01 40 08 02 00 00 00 D5 86 C2   .......@........
00000020  0E 00 00 00 04 67 41 4D 41 00 00 B1 8F 0B FC 61   .....gAMA......a
00000030  05 00 00 00 20 63 48 52 4D 00 00 7A 26 00 00 80   .... cHRM..z&...
00000040  84 00 00 EA 00 00 00 80 E8 00 00 75 30 00 00 EA   ...........u0...
```

For Scanning Signature Analysis:
[PNG file, sig: 89504E470D0A1A0A] → File type identifier

./forensic_steganography



```
Congrats! Here's the flag:
UUMCTF{L3a$t_$1gn!f!c@nt_b!t_!$n't_$!gn!f!c@nt_4nym0r3}
```

The art of hiding data in images or audio

Popular CTF challenge and it might be a separate category by itself

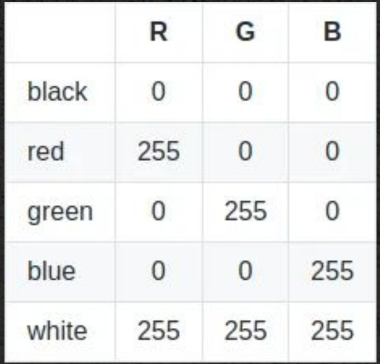
Common Methods:

- LSB (Least Significant Bit)
- Discrete Fourier Transform (DFT)
- Palette-Based Technique

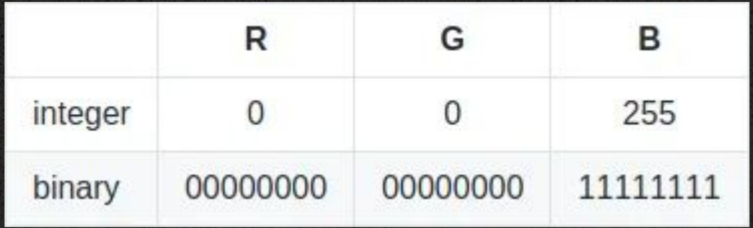
./forensic_steganography

Understanding How LSB Works:

- Each image has pixels with 3 channel of RGB
- Each channel needs 1 byte (8 bits of 1's and 0's)

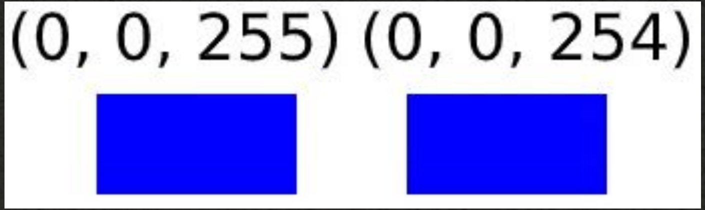


	R	G	B
black	0	0	0
red	255	0	0
green	0	255	0
blue	0	0	255
white	255	255	255



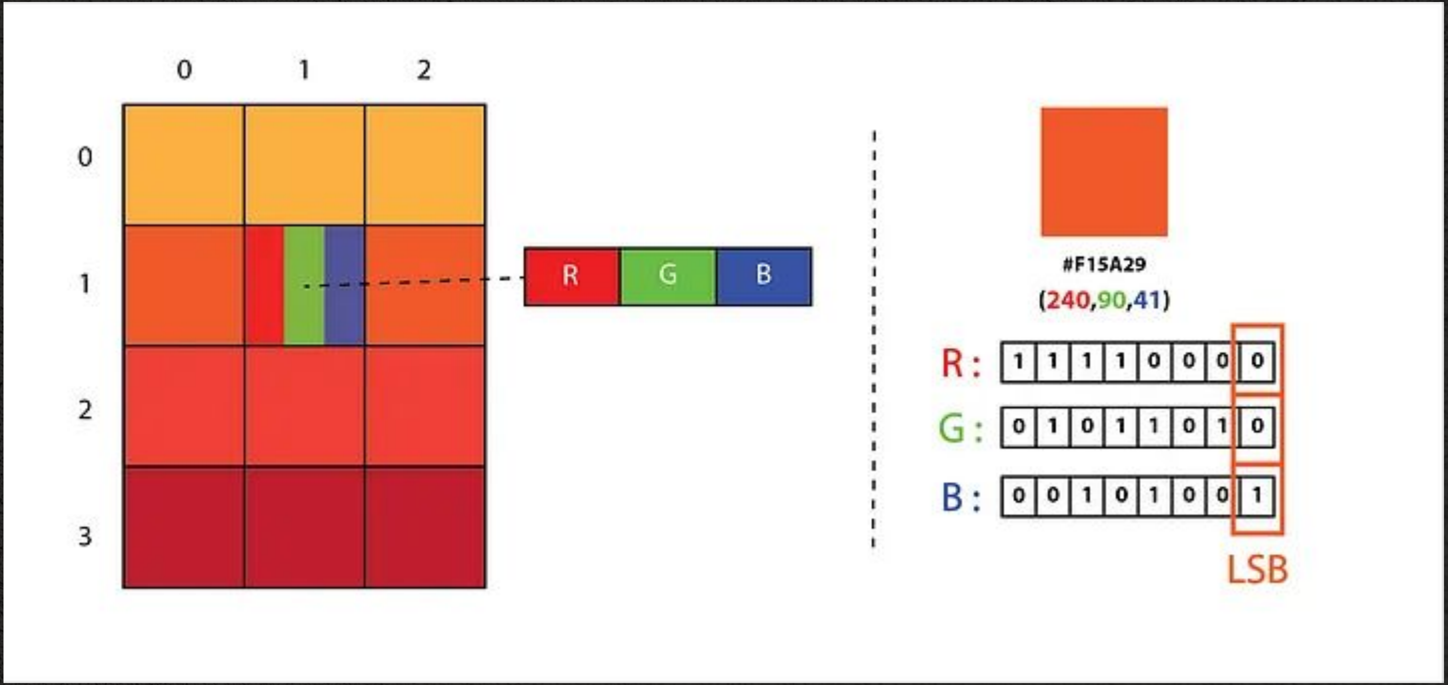
	R	G	B
integer	0	0	255
binary	00000000	00000000	11111111

If we change a single bit of the pixel, the last one (LSB), the result doesn't appeal to be very different.



So message are decoded in binary from ASCII:
Example: Letter 'A' -> ASCII value 97 -> 01100001
First pixel : 0 1 1; Second pixel: 0 0 0; Third pixel: 0 1

./forensic_steganography

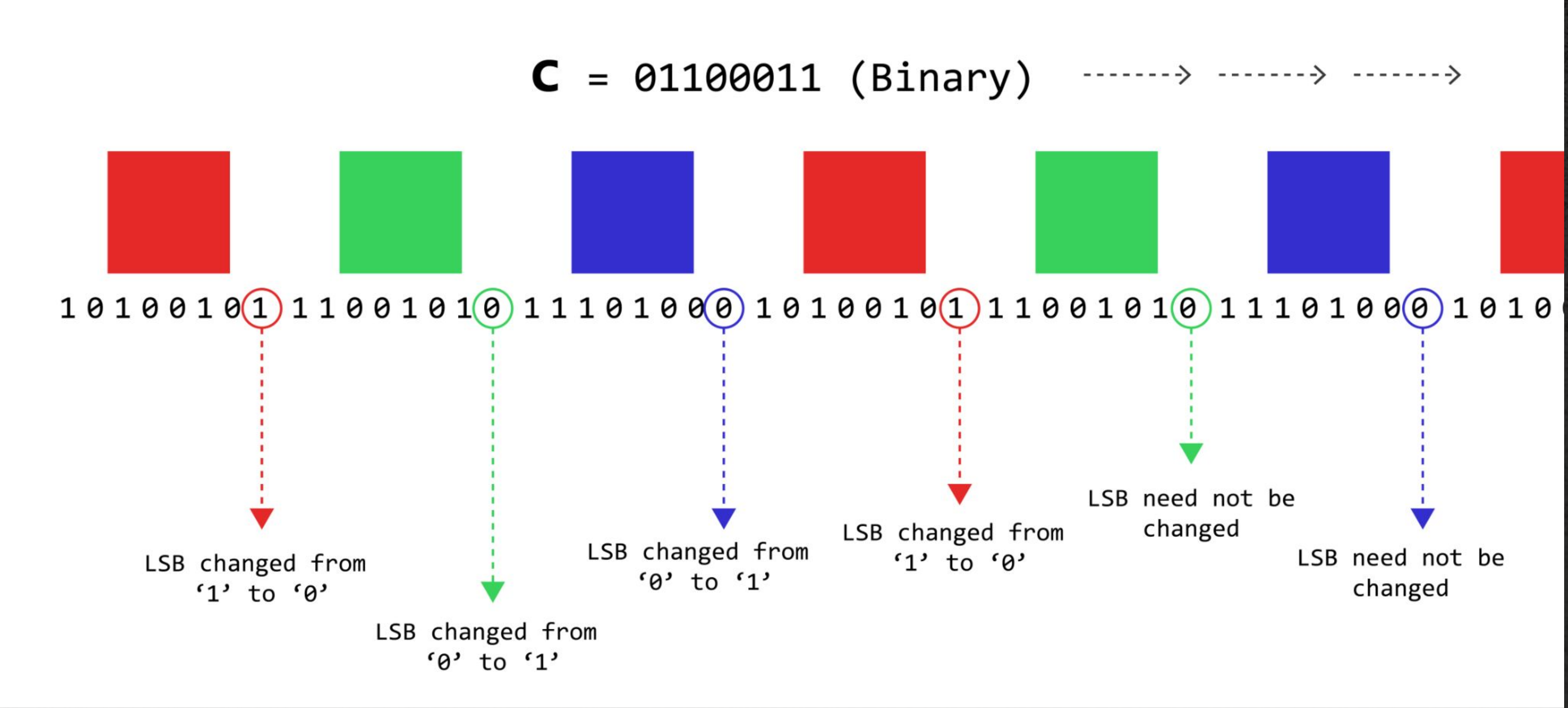


R : 1 1 1 1 0 0 0 0

G : 0 1 0 1 1 0 1 0

B : 0 0 1 0 1 0 0 1

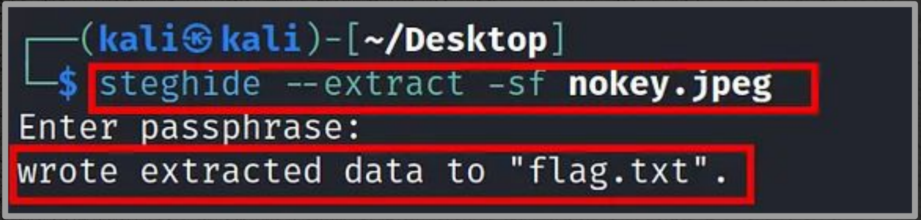
./forensic_steganography



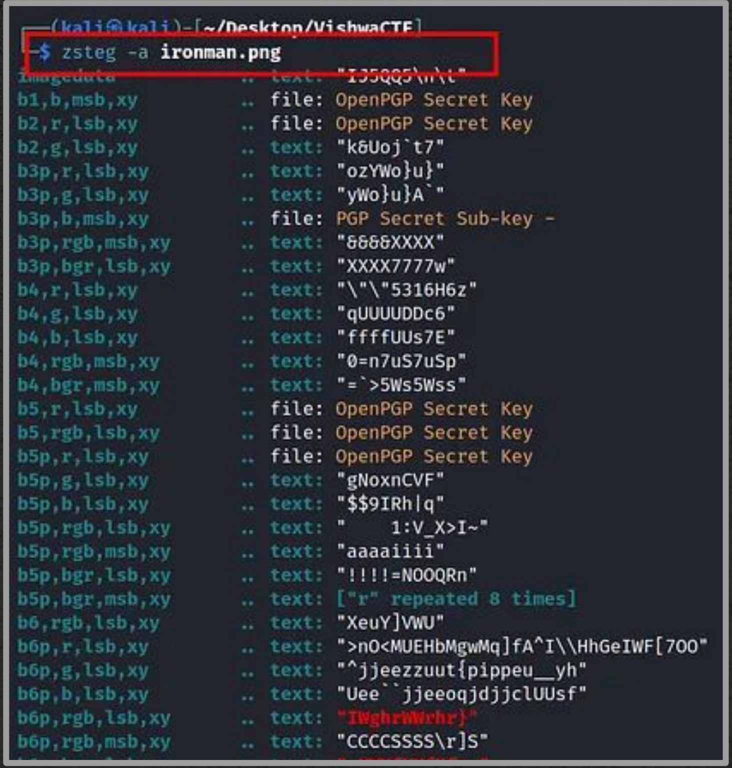
./forensic_steganography

Common tools for steganography challenge:

- Strings
- File
- Exiftool
- Binwalk
- Zsteg
- Steghide
- Stegsolve



```
┌──(kali㉿kali)-[~/Desktop]
└─$ steghide --extract -sf nokey.jpeg
Enter passphrase:
wrote extracted data to "flag.txt".
```



```
┌──(kali㉿kali)-[~/Desktop/VishwaCTF]
└─$ zsteg -a ironman.png
imagedata           .. text: "I35QQ5\n\t"
b1,b,msb,xy         .. file: OpenPGP Secret Key
b2,r,lsb,xy         .. file: OpenPGP Secret Key
b2,g,lsb,xy         .. text: "k&Uoj`t7"
b3p,r,lsb,xy        .. text: "ozYWo}u}"
b3p,g,lsb,xy        .. text: "yWo}u}A`"
b3p,b,msb,xy        .. file: PGP Secret Sub-key -
b3p,rgb,msb,xy      .. text: "&&&&XXXX"
b3p,bgr,lsb,xy      .. text: "XXXX7777w"
b4,r,lsb,xy         .. text: "\"\"5316H6z"
b4,g,lsb,xy         .. text: "qUUUUDDc6"
b4,b,lsb,xy         .. text: "ffffUUs7E"
b4,rgb,msb,xy       .. text: "0=n7uS7uSp"
b4,bgr,msb,xy       .. text: "=`>5Ws5Wss"
b5,r,lsb,xy         .. file: OpenPGP Secret Key
b5,rgb,lsb,xy       .. file: OpenPGP Secret Key
b5p,r,lsb,xy        .. file: OpenPGP Secret Key
b5p,g,lsb,xy        .. text: "gNoxnCVF"
b5p,b,lsb,xy        .. text: "$$9IRh|q"
b5p,rgb,lsb,xy      .. text: "    1:V_X>I~"
b5p,rgb,msb,xy      .. text: "aaaaiiii"
b5p,bgr,lsb,xy      .. text: "!!!!=NOOQRn"
b5p,bgr,msb,xy      .. text: ["r" repeated 8 times]
b6,rgb,lsb,xy       .. text: "XeuY]VWU"
b6p,r,lsb,xy        .. text: ">nO<MUEHbMgwMq]fA^I\\HhGeIWF[7OO"
b6p,g,lsb,xy        .. text: "^jjeezzuut{pippeu__yh"
b6p,b,lsb,xy        .. text: "Uee``jjeeoqjdjjclUUsf"
b6p,rgb,lsb,xy      .. text: "IWghrWWrhr}"
b6p,rgb,msb,xy      .. text: "CCCCSSSS\r]S"
```

./forensic_office_files

- OLE -> Object Linking and Embedding
- Allows to construct objects, which can linked or embedded within other documents or applications
- Acts like mini file system (compound document)
- Newer .docx, .xlsx are zipped XML format

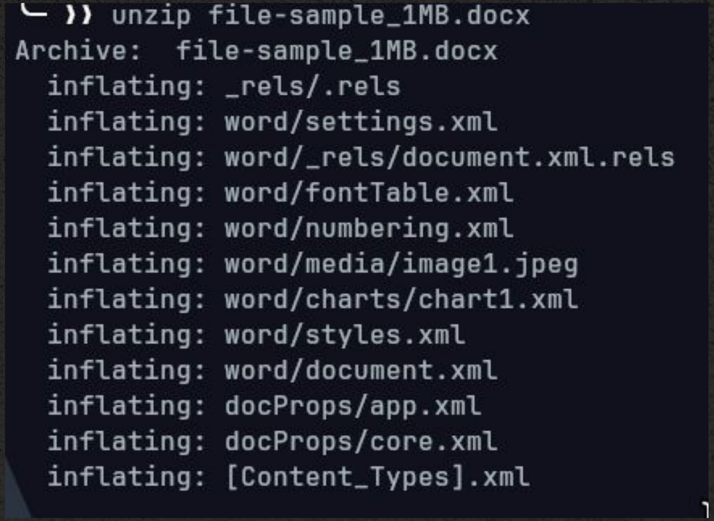
Threats in Office Files

- Malicious macros (VBA)
- Embedded executables
- Suspicious links
- Auto-execution triggers

./forensic_office_files

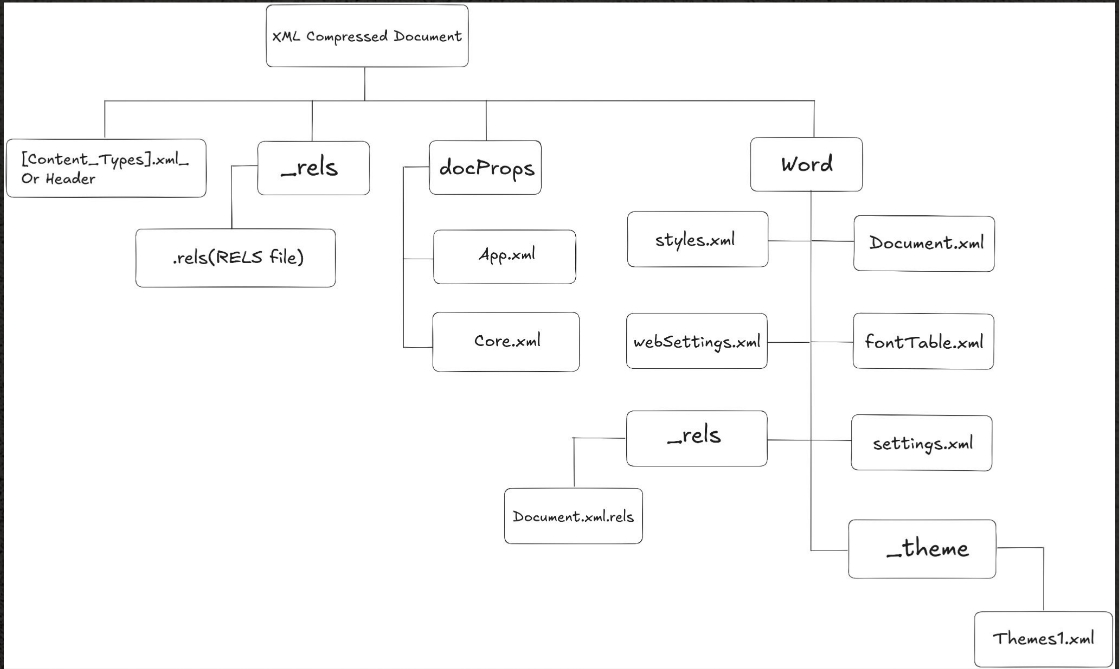
- Modern office files are XML-based archive file format
- Two methods to extract the contents:
  - unzip
  - oletools

- Crucial to understand the metadata structure of Office files
- Example:
  - docProps/core.xml is for file properties
  - word/styles.xml for formatting details



```
╰─ ❯❯ unzip file-sample_1MB.docx
Archive:  file-sample_1MB.docx
  inflating: _rels/.rels
  inflating: word/settings.xml
  inflating: word/_rels/document.xml.rels
  inflating: word/fontTable.xml
  inflating: word/numbering.xml
  inflating: word/media/image1.jpeg
  inflating: word/charts/chart1.xml
  inflating: word/styles.xml
  inflating: word/document.xml
  inflating: docProps/app.xml
  inflating: docProps/core.xml
  inflating: [Content_Types].xml
```

./forensic_office_files



Understand OOXML Structure

1. Key Metadata Files

-> Located in the docProps/ directory:
- core.xml - metadata
- app.xml - stores info like number of pages

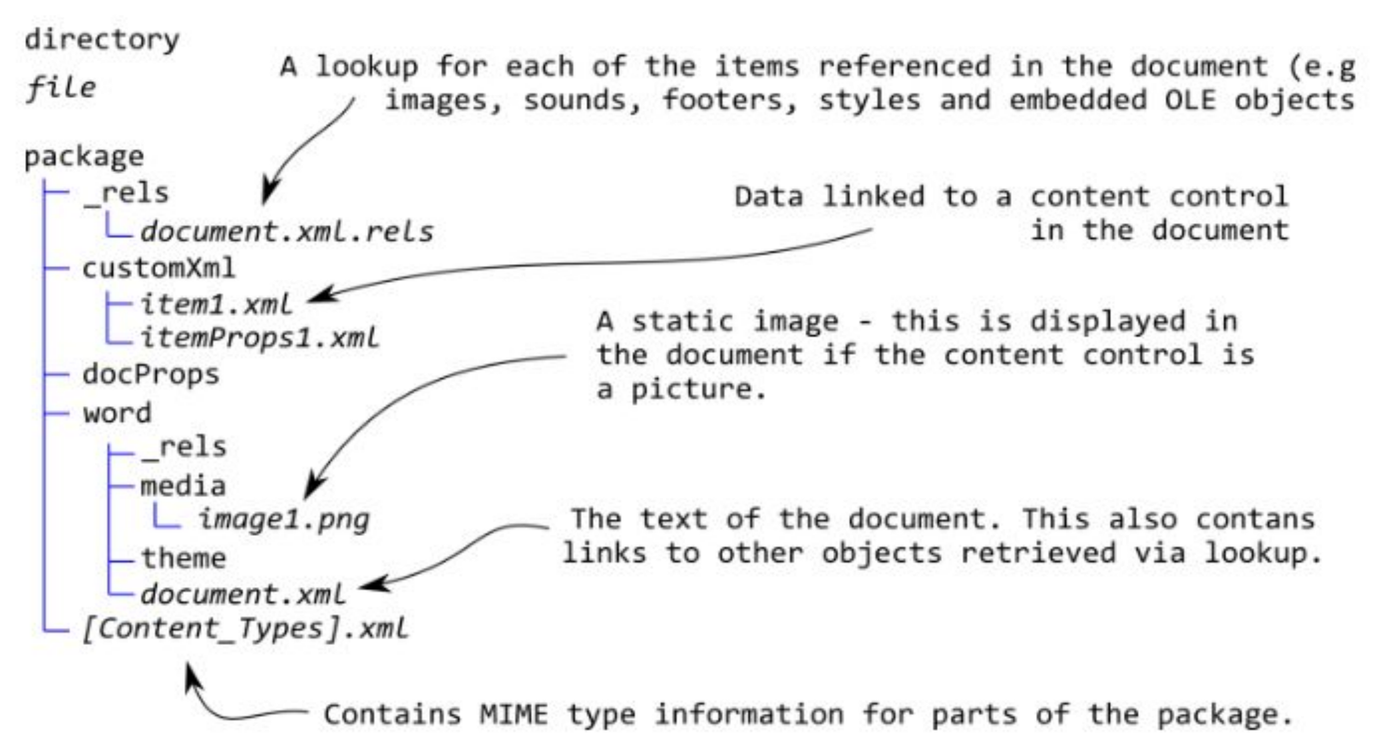
These metadata generated by Office, not OS

2. Two Types of Metadata

-> Internal (OOXML): From core.xml
-> External (File container): From filesystem

It is used to uncover authorship, editing history, or potential tampering

./forensic_office_files



directory

file

```
package
├─ _rels
│   └─ document.xml.rels
├─ customXml
│   ├─ item1.xml
│   └─ itemProps1.xml
├─ docProps
├─ word
│   ├─ _rels
│   ├─ media
│   │   └─ image1.png
│   ├─ theme
│   └─ document.xml
└─ [Content_Types].xml
```

A lookup for each of the items referenced in the document (e.g images, sounds, footers, styles and embedded OLE objects

Data linked to a content control in the document

A static image - this is displayed in the document if the content control is a picture.

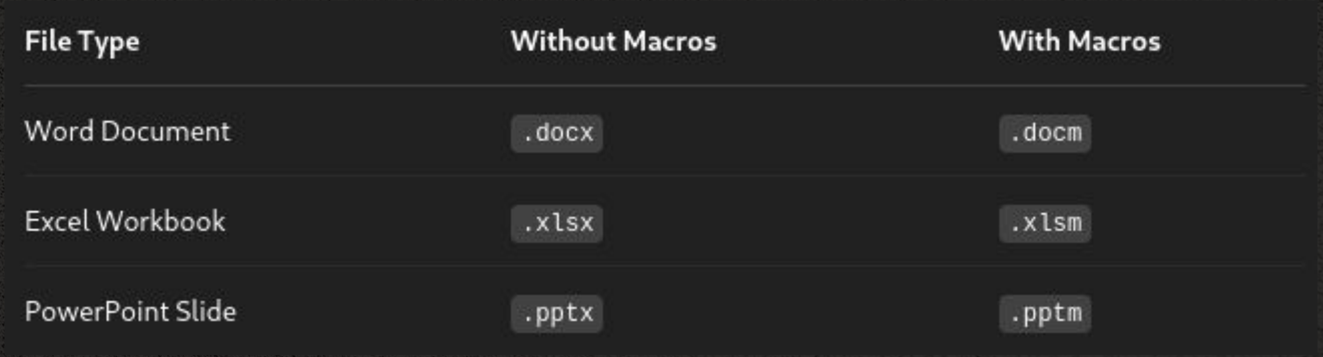
The text of the document. This also contans links to other objects retrieved via lookup.

Contains MIME type information for parts of the package.

./forensic_office_files

VBA Macros

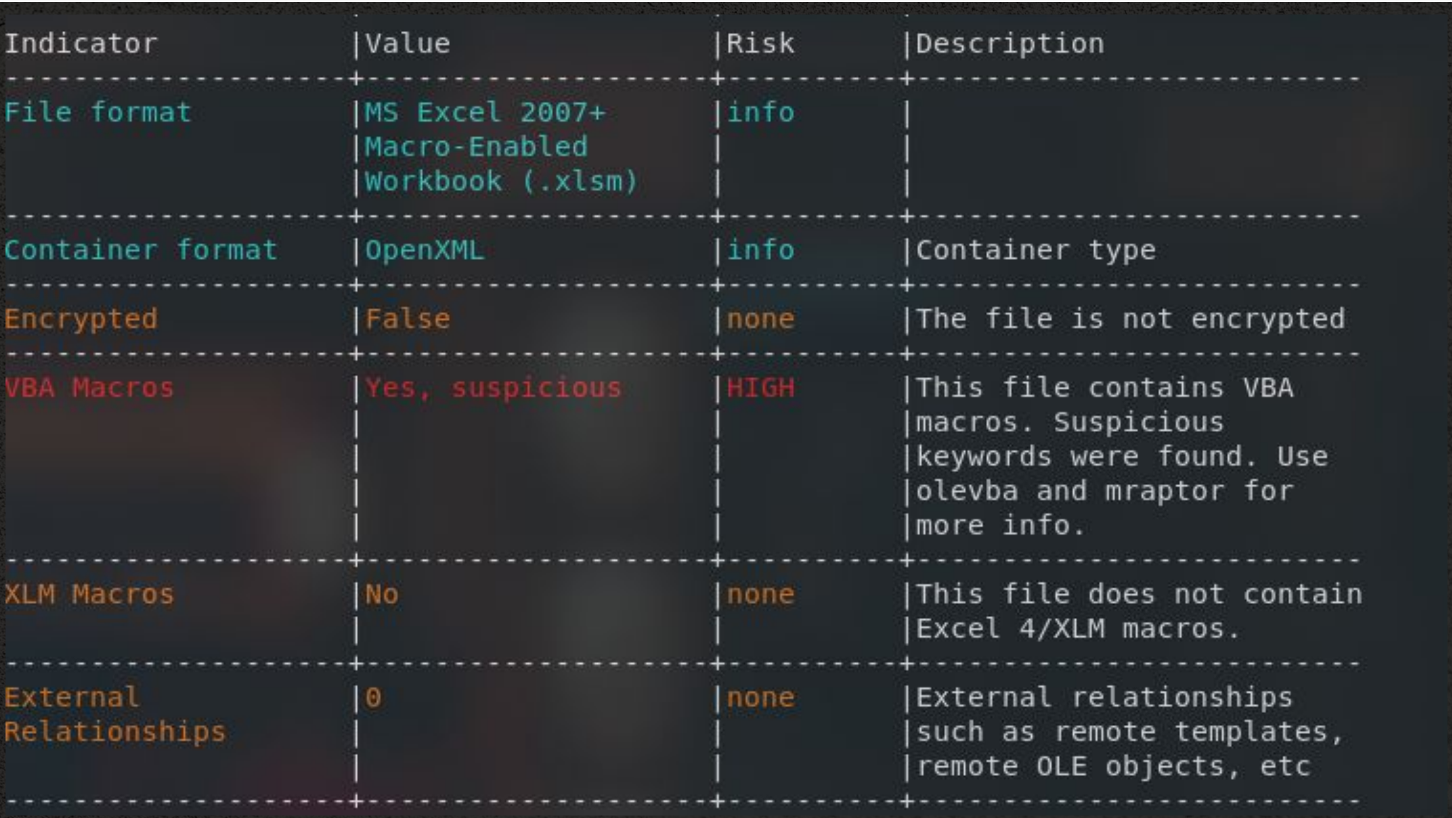
- Often used for malware which provide easy way to execute VB script by opening the file
- Macro-enabled files always have an ‘m’ at the end of the extension



File Type	Without Macros	With Macros
Word Document	`.docx`	`.docm`
Excel Workbook	`.xlsx`	`.xlsm`
PowerPoint Slide	`.pptx`	`.pptm`

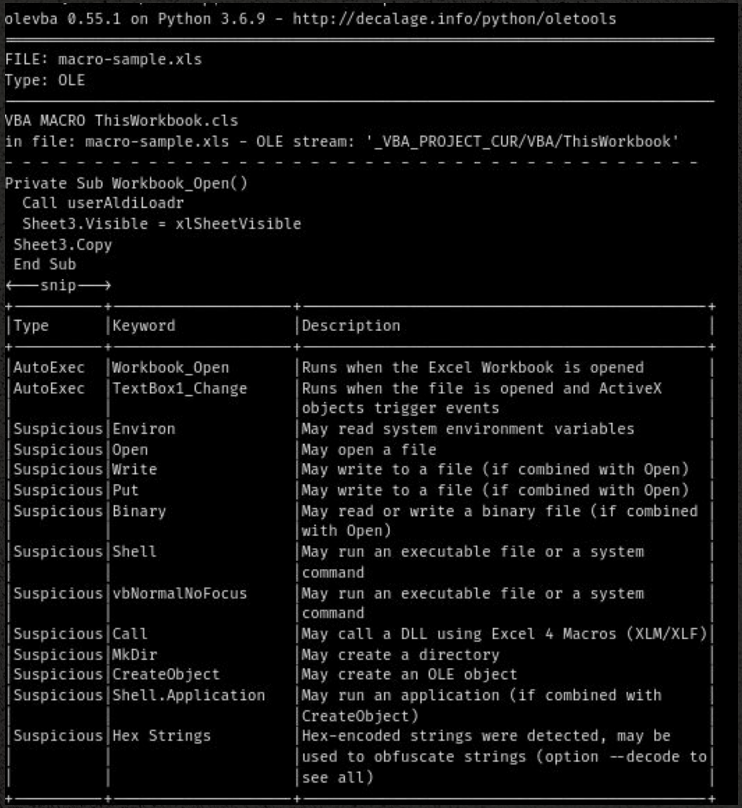
- OleVBA is a tool to detect and analyze VBA macros and able to find suspicious code and decode strings to allow deeper analysis

./forensic_office_files



```
Indicator           |Value               |Risk      |Description
--------------------+--------------------+----------+--------------------------
File format         |MS Excel 2007+      |info      |
                    |Macro-Enabled       |          |
                    |Workbook (.xlsm)    |          |
--------------------+--------------------+----------+--------------------------
Container format    |OpenXML             |info      |Container type
--------------------+--------------------+----------+--------------------------
Encrypted           |False               |none      |The file is not encrypted
--------------------+--------------------+----------+--------------------------
VBA Macros          |Yes, suspicious     |HIGH      |This file contains VBA
                    |                    |          |macros. Suspicious
                    |                    |          |keywords were found. Use
                    |                    |          |olevba and mraptor for
                    |                    |          |more info.
--------------------+--------------------+----------+--------------------------
XLM Macros          |No                  |none      |This file does not contain
                    |                    |          |Excel 4/XLM macros.
--------------------+--------------------+----------+--------------------------
External            |0                   |none      |External relationships
Relationships       |                    |          |such as remote templates,
                    |                    |          |remote OLE objects, etc
--------------------+--------------------+----------+--------------------------
```

./forensic_office_files



```
olevba 0.55.1 on Python 3.6.9 - http://decalage.info/python/oletools
===============================================================================
FILE: macro-sample.xls
Type: OLE
-------------------------------------------------------------------------------
VBA MACRO ThisWorkbook.cls
in file: macro-sample.xls - OLE stream: '_VBA_PROJECT_CUR/VBA/ThisWorkbook'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Private Sub Workbook_Open()
  Call userAldiLoadr
  Sheet3.Visible = xlSheetVisible
 Sheet3.Copy
 End Sub
<—snip—>
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |
+----------+--------------------+---------------------------------------------+
|AutoExec  |Workbook_Open       |Runs when the Excel Workbook is opened       |
|AutoExec  |TextBox1_Change     |Runs when the file is opened and ActiveX     |
|          |                    |objects trigger events                       |
|Suspicious|Environ             |May read system environment variables        |
|Suspicious|Open                |May open a file                              |
|Suspicious|Write               |May write to a file (if combined with Open)  |
|Suspicious|Put                 |May write to a file (if combined with Open)  |
|Suspicious|Binary              |May read or write a binary file (if combined |
|          |                    |with Open)                                   |
|Suspicious|Shell               |May run an executable file or a system       |
|          |                    |command                                      |
|Suspicious|vbNormalNoFocus     |May run an executable file or a system       |
|          |                    |command                                      |
|Suspicious|Call                |May call a DLL using Excel 4 Macros (XLM/XLF)|
|Suspicious|MkDir               |May create a directory                       |
|Suspicious|CreateObject        |May create an OLE object                     |
|Suspicious|Shell.Application   |May run an application (if combined with     |
|          |                    |CreateObject)                                |
|Suspicious|Hex Strings         |Hex-encoded strings were detected, may be    |
|          |                    |used to obfuscate strings (option --decode to|
|          |                    |see all)                                     |
+----------+--------------------+---------------------------------------------+
```

1. Auto-Execute Functions

- **Workbook_Open( ):** Runs automatically when file is open
- **TextBox1_Change:** Triggers when a specific TextBox is changed

2. Suspicious Element

- Environment variable access
- File operations
- Binary file operations (exe, dll)
- Shell command execution
- CreateObject capabilities
- Application execution
- Hex string encoding (possible obfuscation)

./forensic_office_files

Analyze VBA from Office files

1. oleid

- Static analysis, summary of security-relevant
- Detect VBA macros
- Exploit techniques used

```
$ oleid example.doc
```

2. olevba

- Extracts/Analyze VBA script macros
- Embedded OLE objects
- Useful for analyzing documents from phishing emails

```
$ olevba example.doc
```

Example Malicious Document Analysis Challenge:
https://0x251e-challenge.github.io/challenges/posts/total-wreck-spreadsheets/

./forensic_office_files

JS Embedding in PDF Files

Knowing PDF Structure and JS Embedding:

- Header (%PDF-1.4) -> indicates pdf version
- Body -> metadata objects, page content, interactive elements
- Cross-Reference Table (xref) -> Maps objects to their locations within the file
- Trailer (%%EOF)

Possible ways embedding JS into PDF

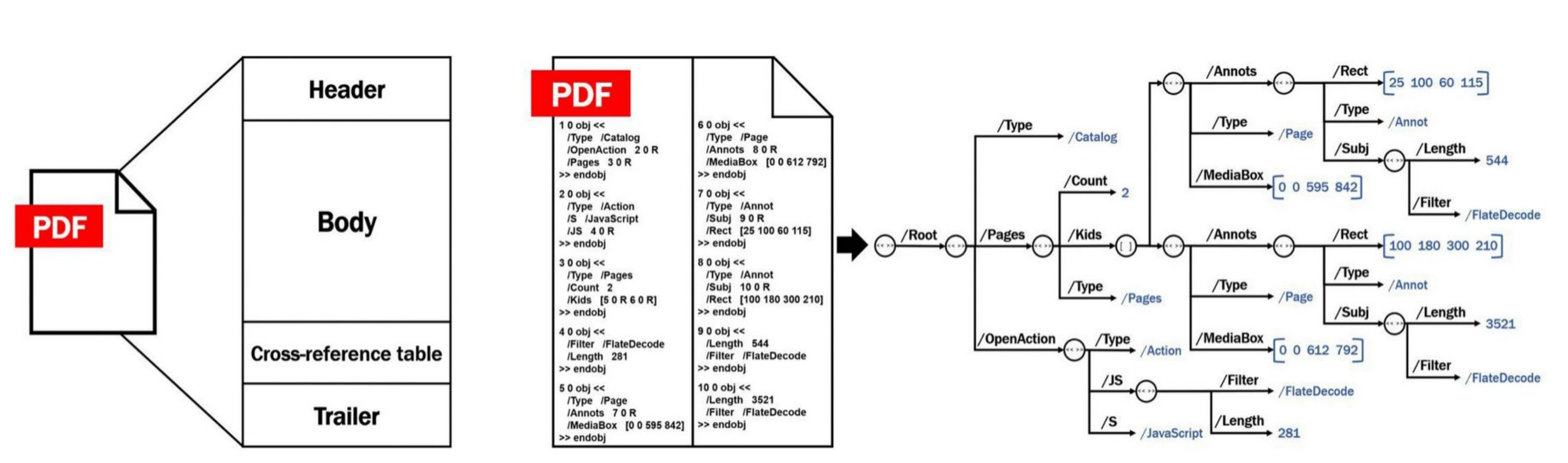
- Catalog Object: **OpenAction** will execute JS script when PDF is opened
- Annotations: **Button** or **Links** can trigger JS which exploit buffer overflow or XSS

Tools:

- PDFiD: Detects JS elements, embedded files, auto-actions
- pdf-parser.py: Analyze PDF objects to find JS payloads

./forensic_office_files

JS Embedding in PDF Files

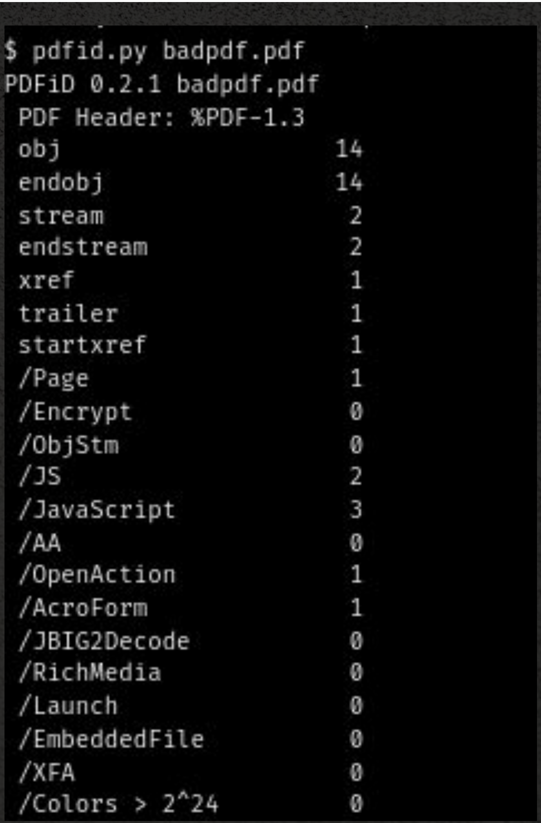


```
1 0 obj <<
  /Type   /Catalog
  /OpenAction   2 0 R
  /Pages   3 0 R
>> endobj

2 0 obj <<
  /Type   /Action
  /S   /JavaScript
  /JS   4 0 R
>> endobj

3 0 obj <<
  /Type   /Pages
  /Count   2
  /Kids   [5 0 R 6 0 R]
>> endobj

4 0 obj <<
  /Filter   /FlateDecode
  /Length   281
>> endobj

5 0 obj <<
  /Type   /Page
  /Annots   7 0 R
  /MediaBox   [0 0 595 842]
>> endobj

6 0 obj <<
  /Type   /Page
  /Annots   8 0 R
  /MediaBox   [0 0 612 792]
>> endobj

7 0 obj <<
  /Type   /Annot
  /Subj   9 0 R
  /Rect   [25 100 60 115]
>> endobj

8 0 obj <<
  /Type   /Annot
  /Subj   10 0 R
  /Rect   [100 180 300 210]
>> endobj

9 0 obj <<
  /Length   544
  /Filter   /FlateDecode
>> endobj

10 0 obj <<
  /Length   3521
  /Filter   /FlateDecode
>> endobj
```

./forensic_office_files

PDF Element Actions:

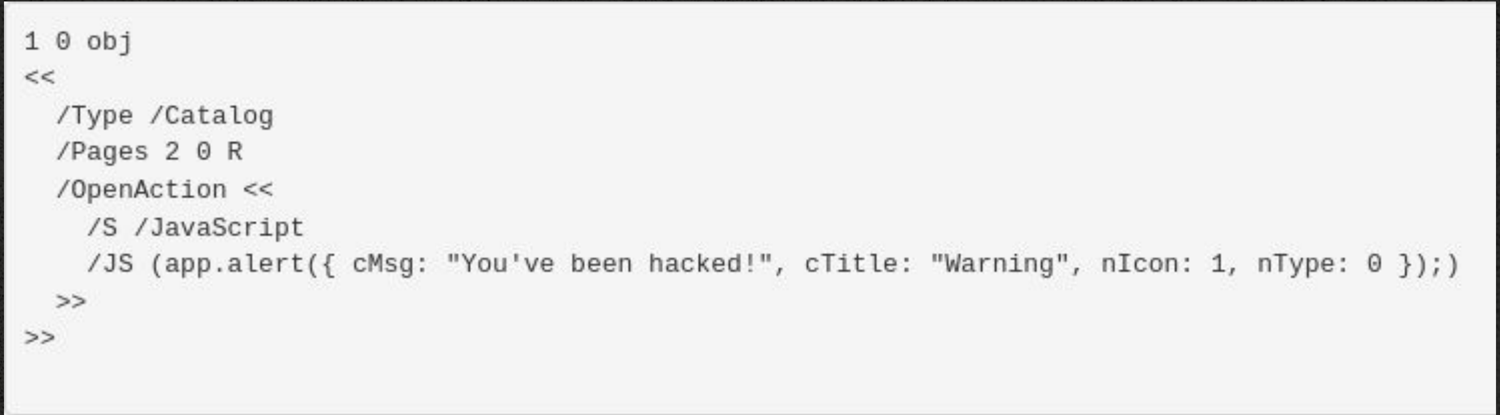
- OpenAction /AA - the function of this element is to carry out an action for e.g. execute a script
- /JavaScript /JS - link to the JavaScript that will run when the PDF is opened
- /Names - names of files that will likely be referred to by the PDF itself
- /EmbeddedFile - shows the other files embedded within the PDF file itself e.g., scripts
- /URI /SubmitForm - Links to other URLs on the internet e.g., possible link to a 2nd stage payload/additional tools for malware to run
- /Launch - Similar to OpenAction, can be used to run embedded scripts within the PDF file itself or run new additional files that have been downloaded by the PDF



```
$ pdfid.py badpdf.pdf
PDFiD 0.2.1 badpdf.pdf
 PDF Header: %PDF-1.3
 obj                    14
 endobj                 14
 stream                  2
 endstream               2
 xref                    1
 trailer                 1
 startxref               1
 /Page                   1
 /Encrypt                0
 /ObjStm                 0
 /JS                     2
 /JavaScript             3
 /AA                     0
 /OpenAction             1
 /AcroForm               1
 /JBIG2Decode            0
 /RichMedia              0
 /Launch                 0
 /EmbeddedFile           0
 /XFA                    0
 /Colors > 2^24          0
```

./forensic_office_files

Example:

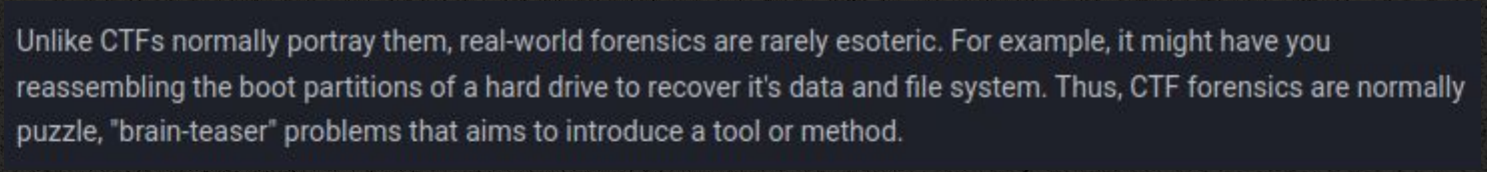


```
1 0 obj
<<
  /Type /Catalog
  /Pages 2 0 R
  /OpenAction <<
    /S /JavaScript
    /JS (app.alert({ cMsg: "You've been hacked!", cTitle: "Warning", nIcon: 1, nType: 0 });)
  >>
>>
```

If the PDF was opened with a web browser, it will show a alert message after opening it.

Use peepdf to analyze the object
$ peepdf -i example.pdf

./real_world_forensic



Unlike CTFs normally portray them, real-world forensics are rarely esoteric. For example, it might have you reassembling the boot partitions of a hard drive to recover it's data and file system. Thus, CTF forensics are normally puzzle, "brain-teaser" problems that aims to introduce a tool or method.

CTF forensics may seem like games, they build the mindset and skills needed for real investigations.

Out in the real world, you won’t just chase flags

you’ll **uncover truths, recover evidence, and solve incidents** that matter.

Also, we just only covered like less than 5% of the whole digital forensic.

THE END...WEEEEEE

AND HAPPY HACKING 🚩🚩

KEEP TRYING AND GIT GUD AT IT