CS 798: Digital Forensics and Incident Response

Lecture 5 - File Forensics

Diogo Barradas

Winter 2025

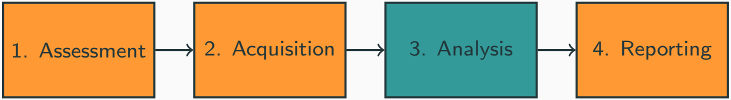
University of Waterloo

Recall evidence collection...

Outline

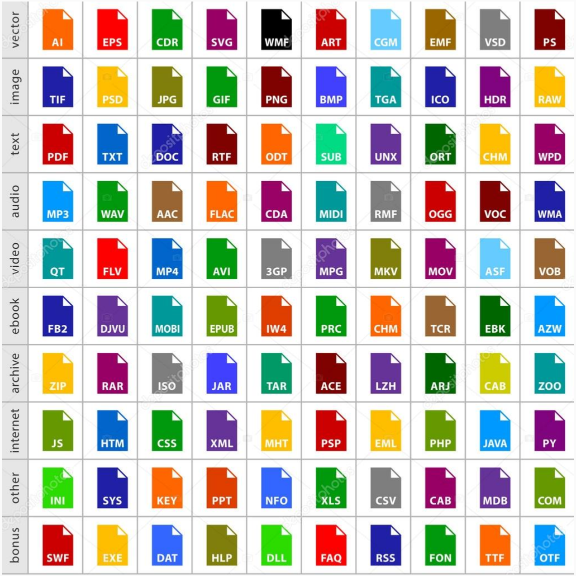
1. A primer on file formats
2. Syntatic analysis of files
3. Semantic analysis of files

Shifting our focus to Analysis



- How to analyze evidence from computers and networks?

Files! (*bazillions* of them)



A primer on file formats

File formats and file forensics

- A file format is a standard way that information is encoded for storage in a computer file
- In file forensics, we aim at recovering information from files by decoding their raw data from storage
- There are two broad file format families:
  - Text files: Essential to determine the text encoding scheme and structure (if any)
  - Binary files: Essential to determine the file format (including endianess)

Text files

- Raw bytes represent characters using an encoding
  - Some are printable,others non-printable (e.g.,linefeed, end of file)
  - ASCII and Unicode are the most common encoding schemes
- ASCII is the common code text representation
  - American Standard Code for Information Interchange
  - Proposed by ANSI in 1963, finalized in 1968
- Assigns a numerical value to characters in American English
  - Originally, 1 character was encoded in 7 bits
  - But limited, e.g., for European languages or mathematical symbols
- Extended ASCII Character Set
  - Extended ASCII uses 8-bits to encode a single character

Standard ASCII
The first 32 characters are control codes.

D	H		D	H		D	H	
0	00	Null	33	21	!	81	51	Q
1	01	Start of heading	34	22	"	82	52	R
2	02	Start of text	35	23	#	83	53	S
3	03	End of text	36	24	$	84	54	T
4	04	End of transmit	37	25	%	85	55	U
5	05	Enquiry	38	26	&	86	56	V
6	06	Acknowledge	39	27	'	87	57	W
7	07	Audible bell	40	28	(	88	58	X
8	08	Backspace	41	29	)	89	59	Y
9	09	Horizontal tab	42	2A	*	90	5A	Z
10	0A	Line feed	43	2B	+	91	5B	[
11	0B	Vertical tab	44	2C	,	92	5C	\
12	0C	Form feed	45	2D	-	93	5D	]
13	0D	Carriage return	46	2E	.	94	5E	^
14	0E	Shift out	47	2F	/	95	5F	_
15	0F	Shift in	48	30	0	96	60	`
16	10	Data link escape	49	31	1	97	61	a
17	11	Device control 1	50	32	2	98	62	b
18	12	Device control 2	51	33	3	99	63	c
19	13	Device control 3	52	34	4	100	64	d
20	14	Device control 4	53	35	5	101	65	e
21	15	Neg. acknowledge	54	36	6	102	66	f
22	16	Synchronous idle	55	37	7	103	67	g
23	17	End trans. block	56	38	8	104	68	h
24	18	Cancel	57	39	9	105	69	i
25	19	End of medium	58	3A	:	106	6A	j
26	1A	Substitution	59	3B	;	107	6B	k
27	1B	Escape	60	3C	<	108	6C	l
28	1C	File separator	61	3D	=	109	6D	m
29	1D	Group separator	62	3E	>	110	6E	n
30	1E	Record separator	63	3F	?	111	6F	o
31	1F	Unit separator	64	40	@	112	70	p
32	20	Blank space	65	41	A	113	71	q
			66	42	B	114	72	r
			67	43	C	115	73	s
			68	44	D	116	74	t
			69	45	E	117	75	u
			70	46	F	118	76	v
			71	47	G	119	77	w
			72	48	H	120	78	x
			73	49	I	121	79	y
			74	4A	J	122	7A	z
			75	4B	K	123	7B	{
			76	4C	L	124	7C	¦
			77	4D	M	125	7D	}
			78	4E	N	126	7E	~
			79	4F	O	127	7F	⌂
			80	50	P			

Extended ASCII
(DOS)

D	H		D	H		D	H	
128	80	Ç	174	AE	«	220	DC	▄
129	81	ü	175	AF	»	221	DD	▌
130	82	é	176	B0	░	222	DE	▐
131	83	â	177	B1	▒	223	DF	▀
132	84	ä	178	B2	▓	224	E0	α
133	85	à	179	B3	│	225	E1	β
134	86	å	180	B4	┤	226	E2	Γ
135	87	ç	181	B5	╡	227	E3	π
136	88	ê	182	B6	╢	228	E4	Σ
137	89	ë	183	B7	╖	229	E5	σ
138	8A	è	184	B8	╕	230	E6	µ
139	8B	ï	185	B9	╣	231	E7	τ
140	8C	î	186	BA	║	232	E8	Φ
141	8D	ì	187	BB	╗	233	E9	Θ
142	8E	Ä	188	BC	╝	234	EA	Ω
143	8F	Å	189	BD	╜	235	EB	δ
144	90	É	190	BE	╛	236	EC	∞
145	91	æ	191	BF	┐	237	ED	ø
146	92	Æ	192	C0	└	238	EE	∈
147	93	ô	193	C1	┴	239	EF	∩
148	94	ö	194	C2	┬	240	F0	≡
149	95	ò	195	C3	├	241	F1	±
150	96	û	196	C4	─	242	F2	≥
151	97	ù	197	C5	┼	243	F3	≤
152	98	ÿ	198	C6	╞	244	F4	⌠
153	99	Ö	199	C7	╟	245	F5	⌡
154	9A	Ü	200	C8	╚	246	F6	÷
155	9B	¢	201	C9	╔	247	F7	≈
156	9C	£	202	CA	╩	248	F8	°
157	9D	¥	203	CB	╦	249	F9	∙
158	9E	₧	204	CC	╠	250	FA	·
159	9F	ƒ	205	CD	═	251	FB	√
160	A0	á	206	CE	╬	252	FC	^{n}
161	A1	í	207	CF	╧	253	FD	^{2}
162	A2	ó	208	D0	╨	254	FE	■
163	A3	ú	209	D1	╤	255	FF	
164	A4	ñ	210	D2	╥			
165	A5	Ñ	211	D3	╙			
166	A6	ª	212	D4	╘			
167	A7	º	213	D5	╒			
168	A8	¿	214	D6	╓			
169	A9	⌐	215	D7	╫			
170	AA	¬	216	D8	╪			
171	AB	½	217	D9	┘			
172	AC	¼	218	DA	┌			
173	AD	¡	219	DB	█			

Unicode

- ASCII is nice and simple if you use American English, but it is quite limited for the rest of the world
    - Their native symbols cannot be represented
- Unicode helps solve this problem by using more than 1 byte to store the numerical version of a symbol
- The version 4.0 Unicode standard supports over 96,000 characters, which requires 4-bytes per character instead of the 1 byte that ASCII requires

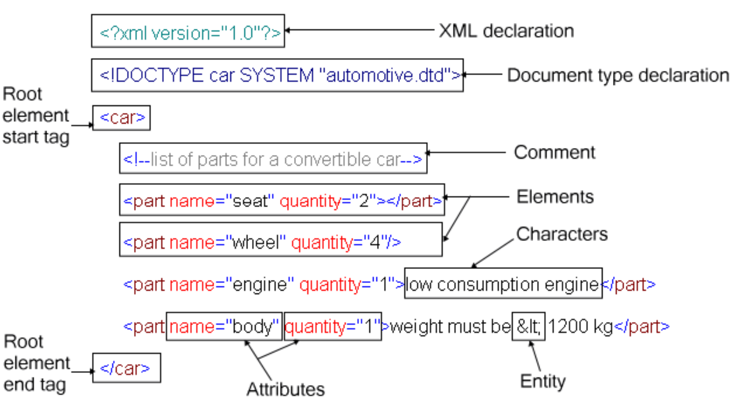
Unicode encodings – UTF-32, UTF-16, UTF-8

- There are three ways of storing a Unicode character:
  - UTF-32: uses a 4-byte value for each character
  - UTF-16: most used chars. in 2-byte value, lesser-used 4-bytes
  - UTF-8: uses 1, 2, or 4 bytes (most frequently used in 1 byte)
- Tradeoff between number of characters that can be represented, and space and processing efficiency
- UTF-8 is frequently used because it has the least amount of wasted space and because ASCII is a subset of UTF-8

Character	UTF-16	UTF-8
A	0041	41
c	0063	63
Ö	00F6	C3 B6
亜	4E9C	E4 BA 9C
𝄞	D834 DD1E	F0 9D 84 9E

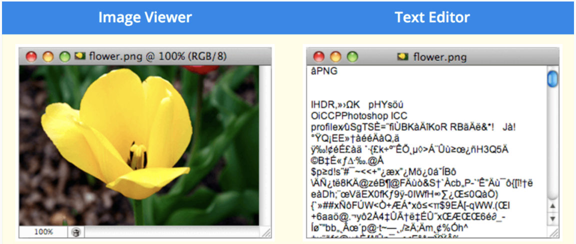
Text files can have structure

- **Example:** XML file



Binary files

- In binary files, bytes represent custom data
  - Binary files may contain both textual and custom binary data
- Binary file formats may include multiple types of data in the same file, such as image, video, and audio data
  - This data can be interpreted by supporting programs, but will show up as garbled text in a text editor

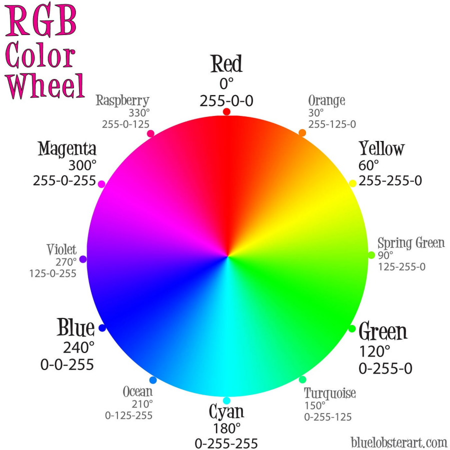


BMP image files

- BMP image: array of pixels, each encodes a specific color
  - Binary files may contain both textual and custom binary data
- The resolution indicates width and height of image
  - Ex: 750×491

Pixel color encoding

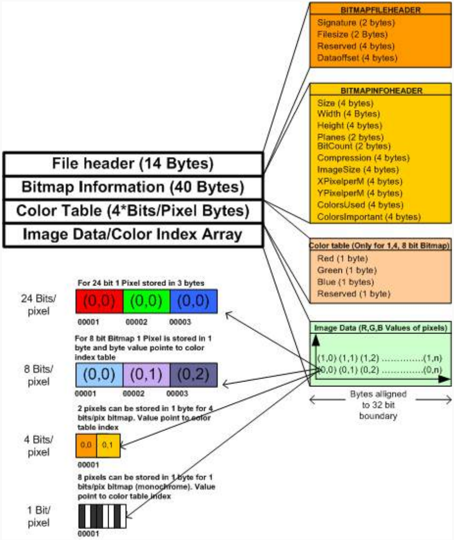
- 24-bit RGB image files
  - Each pixel encoded by 3 byte values for red, green, and blue



- **Signature:** 0x42 0x4D – “BM”

Offset	0	1	2	3	4	5	6	7 -	8	9	A	B	C	D	E	F	ASCII
00000000	42	4D	F6	04	00	00	00	00	00	00	76	00	00	00	28	00	BMц........v...(.
00000010	00	00	30	00	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	01	00	04	00	00	00	..0...0.........
00000020	00	00	80	04	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	00	00	00	00	00	00	..Ђ.............
00000030	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	80	............Ђ..Ђ
00000040	00	00	00	80	80	00	80	00	00	00	80	00	80	00	80	80	...ЂЂ.Ђ...Ђ.Ђ.ЂЂ
00000050	00	00	C0	C0	C0	00	80	80	80	00	00	00	FF	00	00	FF	..AAA.ЂЂЂ...я..я
00000060	00	00	00	FF	FF	00	FF	00	00	00	FF	00	FF	00	FF	FF	...яя.я...я.я.яя
00000070	00	00	FF	FF	FF	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	..яяя.яяяяяяяяяя

Size: 1270 bytes



PDF header format

```
%PDF-1.4
1 0 obj << /Type /Catalog /Outlines 2 0 R /Pages 3 0 R >> endobj
2 0 obj << /Type /Outlines /Count 0 >> endobj
3 0 obj << /Type /Pages /Kids [4 0 R] /Count 1 >> endobj
4 0 obj << /Type /Page /Parent 3 0 R /MediaBox [0 0 612 792]
        /Contents 5 0 R /Resources << /ProcSet 6 0 R
        /Font  << /F1 7 0 R >> >>   >> endobj
5 0 obj << /Length 73 >>   stream
        BT /F1 24  Tf 100 100 Td (Hello World) Tj ET  endstream   endobj
6 0 obj    [/PDF /Text] endobj
7 0 obj << /Type /Font /Subtype /Type1 /Name /F1
        /BaseFont /Helvetica /Encoding /MacRomanEncoding >> endobj
xref
0 8
0000000000 65535 f
0000000009 00000 n
0000000074 00000 n
0000000120 00000 n
0000000179 00000 n
0000000364 00000 n
0000000466 00000 n
0000000496 00000 n
trailer
     << /Size 8 /Root 1 0 R >>
     startxref
     625
%%EOF
```

1: Header

2: Body

Not Encoded Text

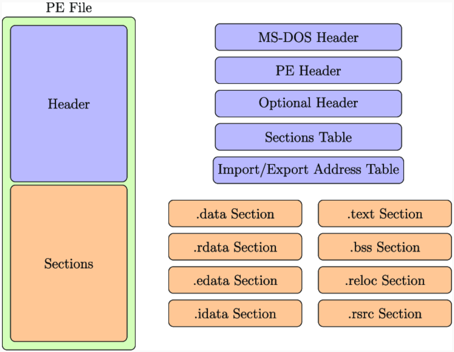
3: xref Table

4: trailer

Executable file format

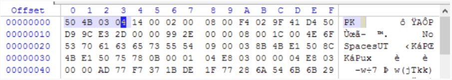
- **Portable Executable** (PE) format used to encode executable files on Windows (`.exe`, `.dll`)

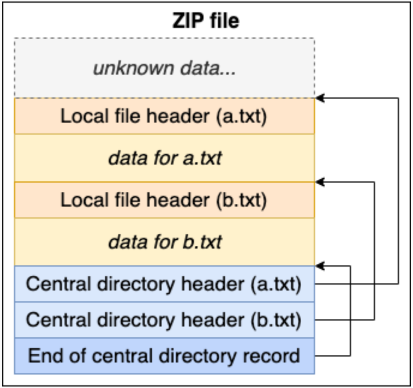
- Common sections are:
  - `.text` (for code)
  - `.data` (read/write data)
  - `.rdata` (read-only data)
  - `.reloc` (relocation data)



Compressed file formats (e.g., ZIP)

- **ZIP files** store multiple files
- Signature: 0x50 0x 4B – “PK”
  - Phil Katz invented ZIP



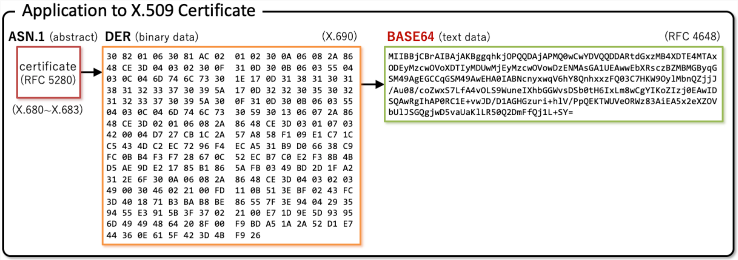


Common binary file formats

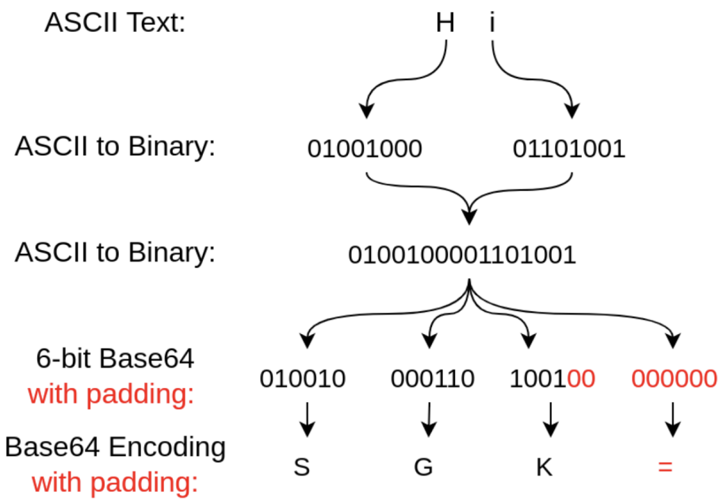
- Image file formats (JPG, GIF, BMP, PNG)
- Archive files (ZIP, TGZ)
- Filesystem images (EXT4)
- Packet captures (PCAP, PCAPNG)
- Memory dumps
- PDF
- Video (MKV, MP4) or Audio (WAV, MP3, FLAC)
- Microsoft Office formats (RTF, OLE, OOXML)

Binary to text encoding schemes

- Sometimes it is necessary to encode binary objects into text
  - E.g., when shipping a binary file across the network, some protocols could interpret byte sequences as control characters
- Base64 is a popular encoding scheme (but there are more...)
  - 64 characters are present in most character sets
  - Used to encode email attachments and certificates



Base64 encoding mechanism



(Anthony Critelli, CC BY-SA 4.0)

Base64 encoding mechanism

- To encode text to base64:

```
barradas@Vitrea ~> echo -n 'welcome to the dfir course' | base64
d2VsY29tZSB0byB0aGUgZGZpciBjb3Vyc2U=
barradas@Vitrea ~>
```

- To decode base64:

```
barradas@Vitrea ~> echo -n d2VsY29tZSB0byB0aGUgZGZpciBjb3Vyc2U= | base64 -d
welcome to the dfir course
barradas@Vitrea ~>
```

Base64 encoding mechanism

- To encode text to base64:

```
barradas@Vitrea ~> echo -n 'welcome to the dfir course' | base64
d2VsY29tZSB0byB0aGUgZGZpciBjb3Vyc2U=
barradas@Vitrea ~>
```

- To decode base64:

```
barradas@Vitrea ~> echo -n d2VsY29tZSB0byB0aGUgZGZpciBjb3Vyc2U= | base64 -d
welcome to the dfir course
barradas@Vitrea ~>
```

Recall that base64 is not an encryption algorithm!

Syntatic analysis of files

Syntactic analysis

- The ability to parse the format of a file so as to identify its internal structure and components
- Techniques:
  - Raw file inspection
  - File format discovery
  - String extraction
  - Encrypted file cracking
  - File header repairing

Inspection of a file's raw bytes

- Use an hex editor to read file contents, e.g., WinHex, xxd

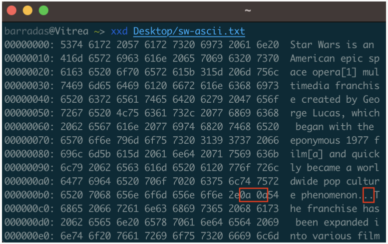
```
barradas@Vitrea ~> xxd Desktop/myimage.png | head
00000000: 8950 4e47 0d0a 1a0a 0000 000d 4948 4452  .PNG........IHDR
00000010: 0000 0200 0000 0200 0806 0000 00f4 78d4  ..............x.
00000020: fa00 0000 0473 4249 5408 0808 087c 0864  .....sBIT....|.d
00000030: 8800 0000 0970 4859 7300 000e c400 000e  .....pHYs.......
00000040: c401 952b 0e1b 0000 0019 7445 5874 536f  ...+......tEXtSo
00000050: 6674 7761 7265 0077 7777 2e69 6e6b 7363  ftware.www.inksc
00000060: 6170 652e 6f72 679b ee3c 1a00 0020 0049  ape.org..<... .I
00000070: 4441 5478 9ced dd79 b865 5579 e7f1 6f51  DATx...y.eUy..oQ
00000080: 05c5 2485 cc73 274e 200e 8108 28a8 6925  ..$..s'N ...(.i%
00000090: 7627 a44d 9410 7dec c76e 2740 2451 1b85  v'.M..}..n'@$Q..
barradas@Vitrea ~>
```

- Use the `file` utility to match a file's signature

```
barradas@Vitrea ~> file Desktop/myimage.png
Desktop/myimage.png: PNG image data, 512 x 512, 8-bit/color RGBA, non-interlaced
```

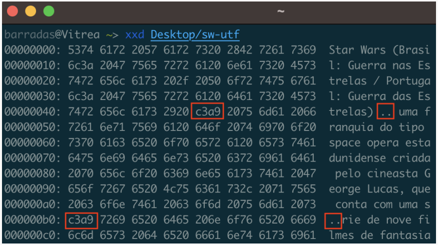
Remarks about the encoding scheme

- ASCII encoded file



- Special non-printable character

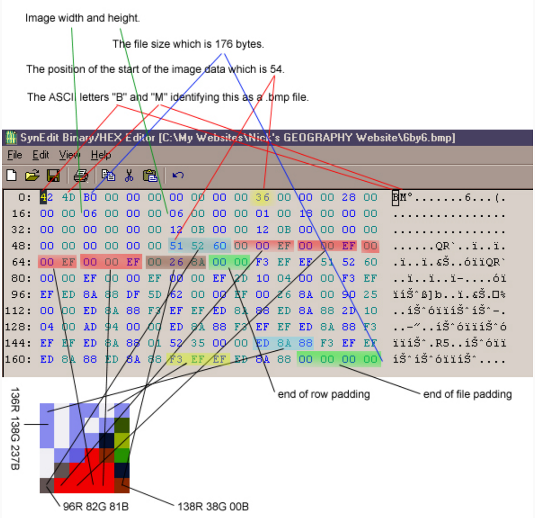
- UTF-8 encoded file



- Character “é” = 0xc3a9

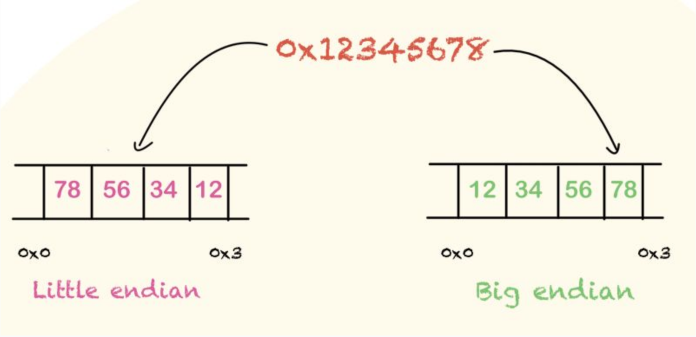
Inspecting a binary file

- Manual inspection
  - Parse the file according to the file format specification



Remarks about endianess

- Numbers can be stored as a sequence of one or more bytes
- Endianness deals with the order in which bytes are stored
  - We encounter two different approaches:



x86 and ARM make use of Little Endian.

Why do we care?

- In the sequence below, the two highlighted bytes represent a 16-bit integer (8 bit x 2 = 16 bits or 2 bytes)

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	01	23	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

- In a big-endian system, the value would be calculated as:
  - Big-endian calculation: **0x0123 = 291**
- In a little-endian system, the value would be calculated as:
  - Little-endian calculation: **0x2301 = 8961**

How to decode a file not knowing its format?

- Consider a file with the following byte sequence (hex):

```
48 65 6c 6c 6f 20 57 6f 72 6c 64 0a
```

- Is it a text file? ASCII? Base64?

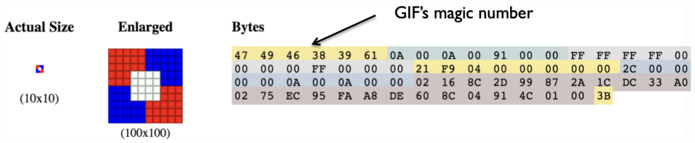
```
‘H’ ‘e’ ‘l’ ‘l’ ‘o’ ‘ ’ ‘W’ ‘o’ ‘r’ ‘l’ ‘d’ ‘\n’
```

- Or a binary file? Unsigned ints? Other format?

```
‘18533’ ‘27756’ ‘28448’ ‘22383’ ‘29292’ ‘25610’
```

Magic numbers

- Look for magic numbers: consist of constant numerical or text value used to identify a file format or protocol
  - E.g., GIF files start with sequence: 0x47 49 46 38 39 61
- Remember the `file` utility?

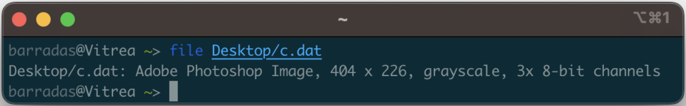


- Magic numbers of common file formats:
  - `http://www.garykessler.net/library/file_sigs.html`

Opening files with specialized programs

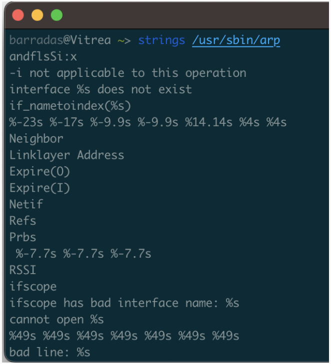
- After determining the file type, use a corresponding program to open it
  - E.g., examine file `c.dat`
- Remember the `file` utility?

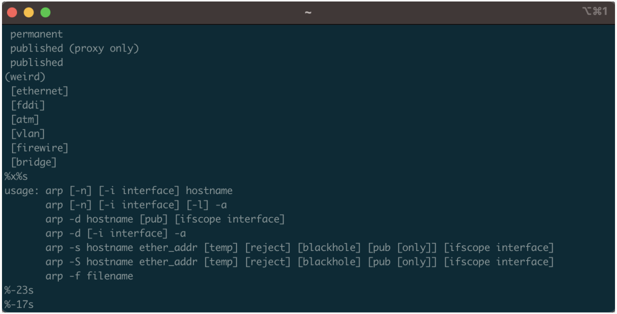
`c.dat` when opened in GIMP



String extraction from files

- The `strings` utility: prints a file's readable characters
  - Useful for looking at data fiedls without the originating program, searching executables for strings, etc.



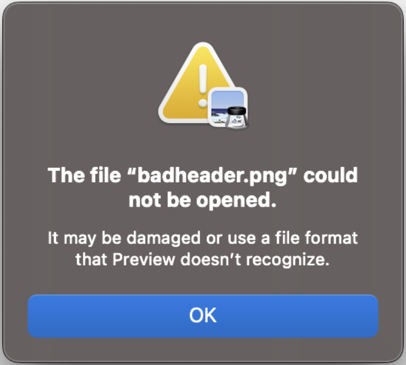


What if files are encrypted?

- Might be worth trying a password cracking software
  - Essentially, a password cracker works by trial and error
- Cracking approaches:
  - Brute force: try every possible key / password until succeeds
  - Dictionary attacks: attempt to reduce the number of trials required and will usually be attempted before brute force
- Examples of tools:
  - For encrypted zip files: `fcrackzip`
  - For encrypted pdf files: `pdfcrack`

What if files are corrupted?

- Specific metadata of a file may have been removed or tampered with thereby preventing its decoding
  - E.g., opening file `badheader.jpg` returns this error



Repairing a corrupted file

- Check if file metadata is consistent with the file format spec

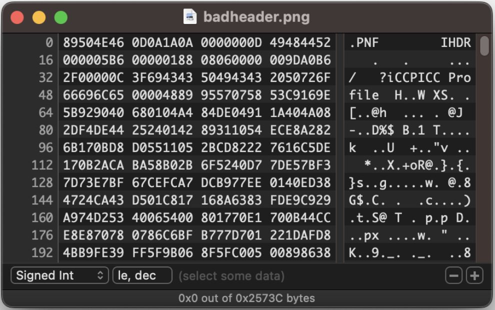
89 50 4E 47 0D 0A 1A 0A ‰PNG....
PNG Portable Network Graphics file
Trailer: 49 45 4E 44 AE 42 60 82 (IEND®B`,...)

Repairing a corrupted file

- Check if file metadata is consistent with the file format spec

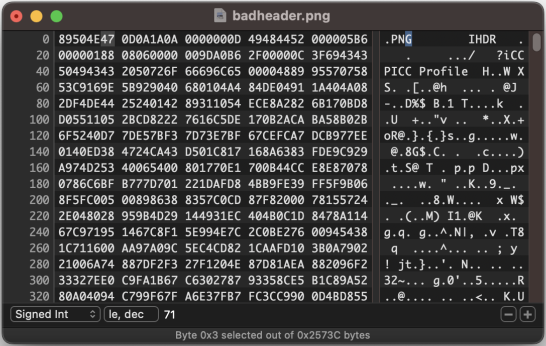
89 50 4E 47 0D 0A 1A 0A ‰PNG....
PNG [Portable Network Graphics file](#)
Trailer: 49 45 4E 44 AE 42 60 82 (IEND®B`,...)

- Our header looks wrong: 89 50 4E 46 0D 0A 1A 0A ...



Apply the fix

- Fix the wrong header byte



- Verify the fix by reopening the file using the viewer
- This case was easy, but may require additional effort

Summary of tips for interpreting file content

- Extension not entirely reliable
- Open the file and check between (text / binary)
- Look for known header and footer information
  - Especially file format signatures (e.g., magic numbers)
- Use tools that know how to interpret specific file format
- If the file format is unknown, we analyze it manually

Semantic analysis of files

Semantic analysis

- The ability to interpret and acquire information from the data content of a given file
- Presupposes the ability to parse the file's internal structures
- Some examples:
    - Vulnerability analysis
    - Image processing
    - Provenance analysis

Image processing

- Forensic image processing involves the computer restoration and enhancement of imagery
- It aims to maximize information extraction from imagery that is noisy, incomplete, or over/under exposed
- It also involves measurement of objects pictured on images

Image processing

- Forensic image processing involves the computer restoration and enhancement of imagery
- It aims to maximize information extraction from imagery that is noisy, incomplete, or over/under exposed
- It also involves measurement of objects pictured on images

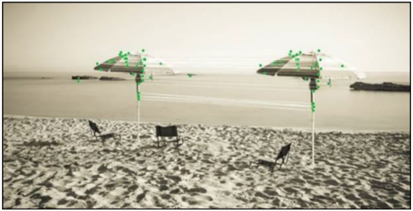
CSI-style “Enhance!”

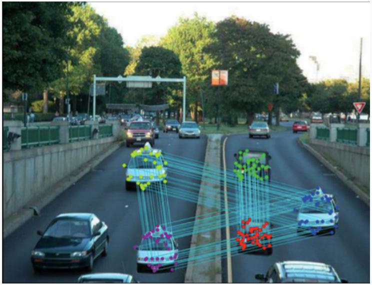
Tampering detection

- In the context of digital imaging, tampering recalls the intentional manipulation of images for malicious purposes

Example of tampering detection technique

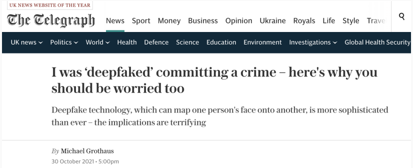
- Structural analysis: helps detect copy-move of an image region
  - A structural analysis algorithm splits the image into segments
  - A histogram is built count the number of matching segments which are separated by the same distance
  - The higher the number of pairs located at the same distance, the higher is the probability that those pairs belong to copied regions





Deepfake detection

- Deepfakes make image tampering detection difficult


UK NEWS WEBSITE OF THE YEAR

The Telegraph | News | Sport | Money | Business | Opinion | Ukraine | Royals | Life | Style | Trave

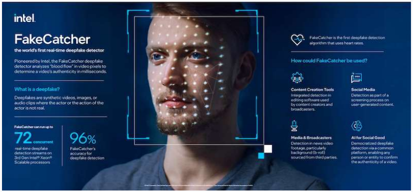
UK news | Politics | World | Health | Defence | Science | Education | Environment | Investigations | Global Health Security

I was 'deepfaked' committing a crime – here's why you should be worried too

Deepfake technology, which can map one person's face onto another, is more sophisticated than ever – the implications are terrifying

By Michael Grothaus

30 October 2021 • 5:00pm



Provenance analysis

The Story Of Luka Magnotta

Don't [redacted] *With Cats* tells the gripping story of one of Canada's worst murderers, Luka Magnotta.

In the three-part docuseries, we are introduced to Baudi Moovan (Deanna Thompson) and John Green who recount everything that lead up to Luka's eventual demise.

You see, before Luka Magnotta was a murderer, he was an animal abuser. Back in 2010, a mystery man uploaded a video onto Youtube titled *1 boy 2 kittens*. The video featured a male in a hooded jacket, vacuum sealing two kittens until they suffocated and died. The clip sparked outrage online and lead to Baudi and John creating the 'Find the Kitten Vacuumer...for Great Justice' Facebook group.

Provenance analysis

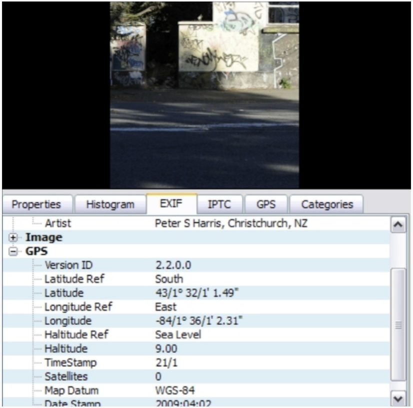
- Images & video frames: Tell one's location by looking at them

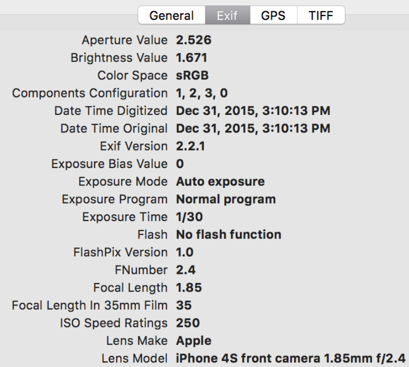
The Facebook group consisted of people from all over the world, determined to uncover who the animal abuser was.

Together they analysed Luka's video frame-by-frame. They looked at plug sockets on the wall to narrow down the killer's location. They listened to background sounds to determine the languages heard. The group even went so far as to find the online sellers of blankets and vacuums seen in shots to find some clue of who the cat killer was.

Metadata

- Image metadata can also give us a lot of information
    - (If available)





Source device identification

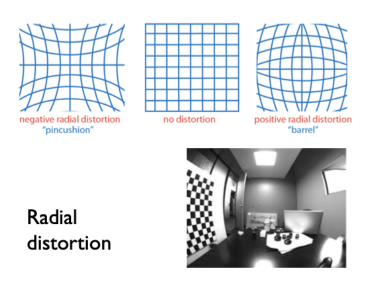
- In a court of law, the device used for acquisition of a particular image can represent crucial evidence
- Helpful clues on the source imaging device might be found in the file's header (EXIF), or by checking (if present) a watermark
- However, since this information can be easily modified or removed, we may need to employ blind techniques

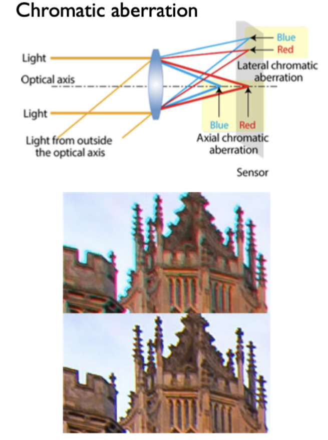
Blind image forensics techniques

- Retrieve information on the source device at two levels:
  - Try to distinguish between different camera models
  - Try to distinguish between single devices, even different devices using the same camera model – harder
- Examine the traces left by the different processing steps in the image acquisition and storage phases

Acquisition of artifacts produced by lenses

- Dirt / deformations
- Lens distortion
- Chromatic aberration





More relevant sources of info for file provenance

- Embedded cryptographic signatures
- References to the program that modified the file
- References about place and time of the file
- References about the author of the file
- Indirect references from other sources (e.g., logs)
- Watermarks
- ...

Takeaways

- Interpretation of file contents is one of the first steps in a typical forensic analysis procedure
- To interpret file contents it may be necessary to understand how the data is represent in its raw form
- In addition, many files are structured according to a particular layout, and thus it is necessary to learn its specific format in order to properly interpret them

Pointers

- **Textbook:**
  - Carrier – Chapter 2.1
- **Other resources:**
  - A. Piva, "An Overview on Image Forensics", ISRN Signal Processing, January 2013
  - Judith A. Redi et al. "Digital image forensics: a booklet for beginners", Multimedia Tools and Applications, 2010
- **Acknowledgements:**
  - Slides adapted from Nuno Santos's Forensics Cyber-Security course at Técnico Lisbon