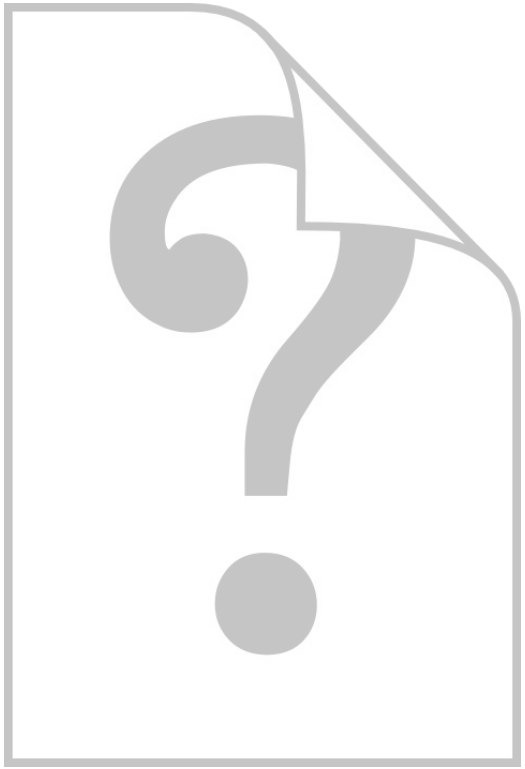
THE CHALLENGES OF FILE FORMATS

SHARING MY PERSPECTIVES WITH THE DIGIPRES COMMUNITY



ANGE ALBERTINI

My perspectives

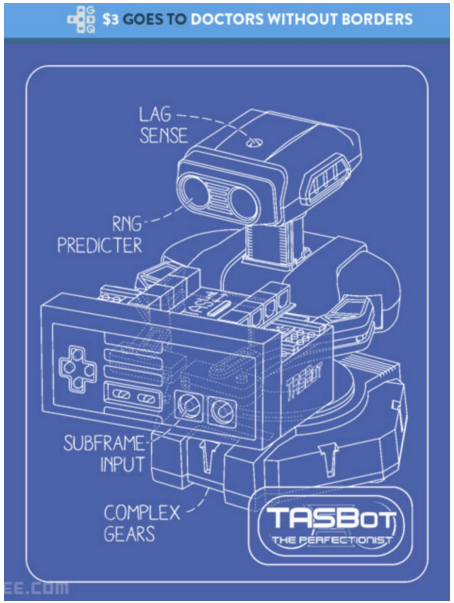
USE Load/save
share

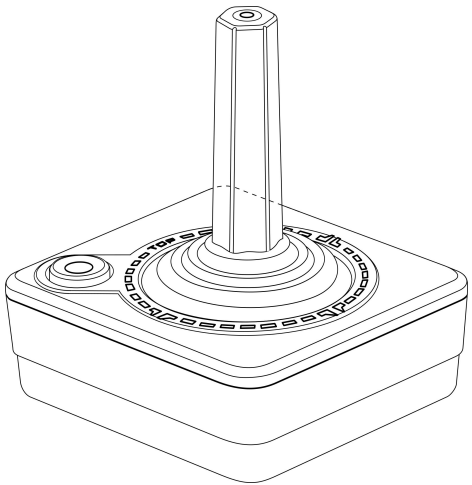
PRESERVE Documents
Video Games

DISSECT Malicious files

CRAFT Extreme files

TINKERING WITH COMPUTERS FOR 30+ YEARS...

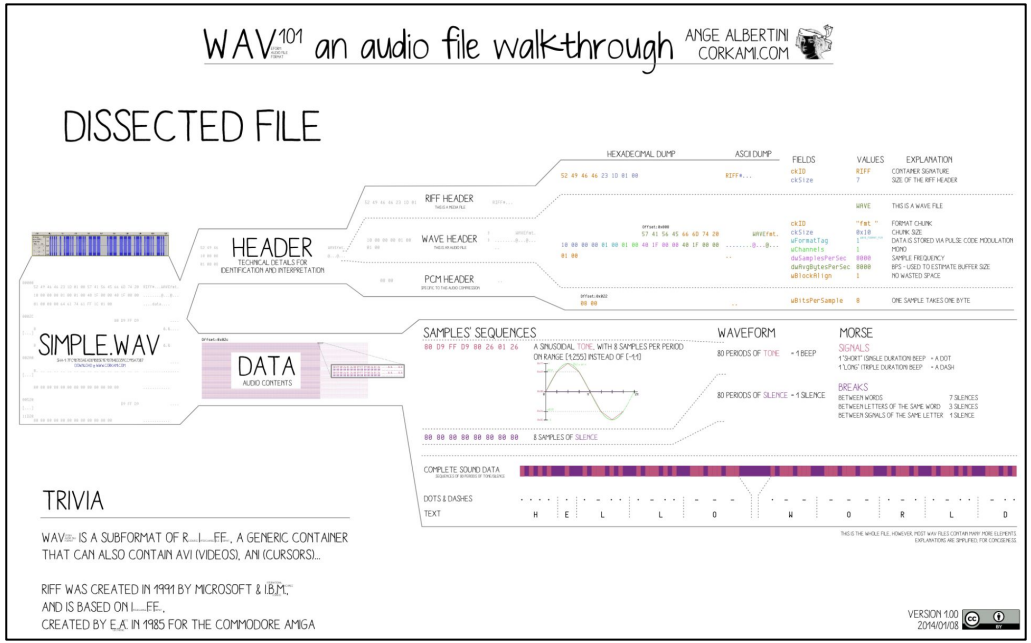


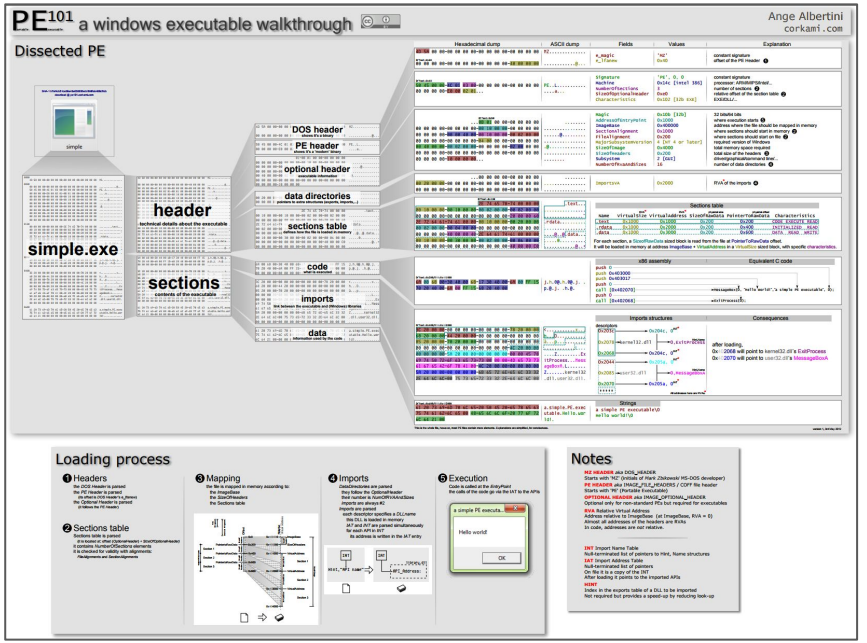


github.com/corkami/pics

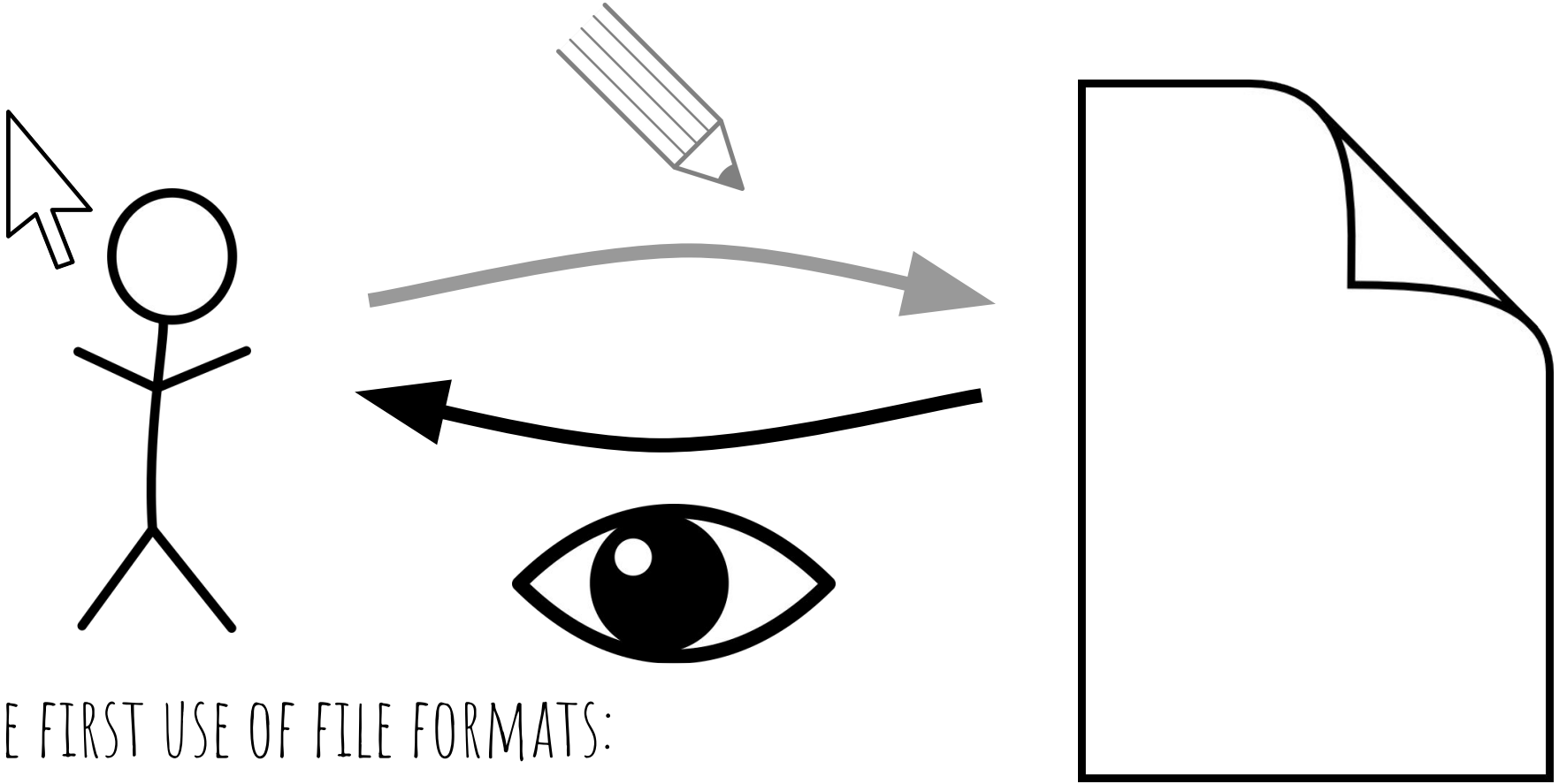
I DRAW MANY KINDS OF THINGS...

...AND DOCUMENT FILE FORMATS.





github.com/corkami/docs

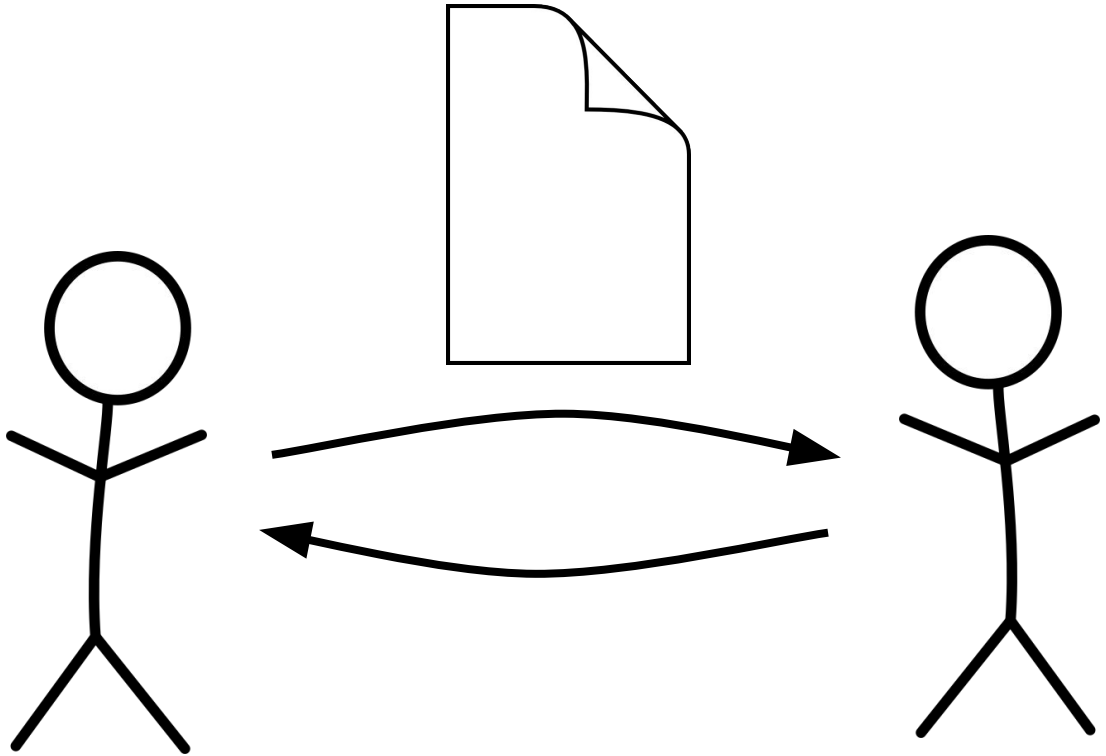


The first use of file formats:
"Save your progress"

Ever printed something
that looked *NOTHING* like
what you have on screen?

Story time:

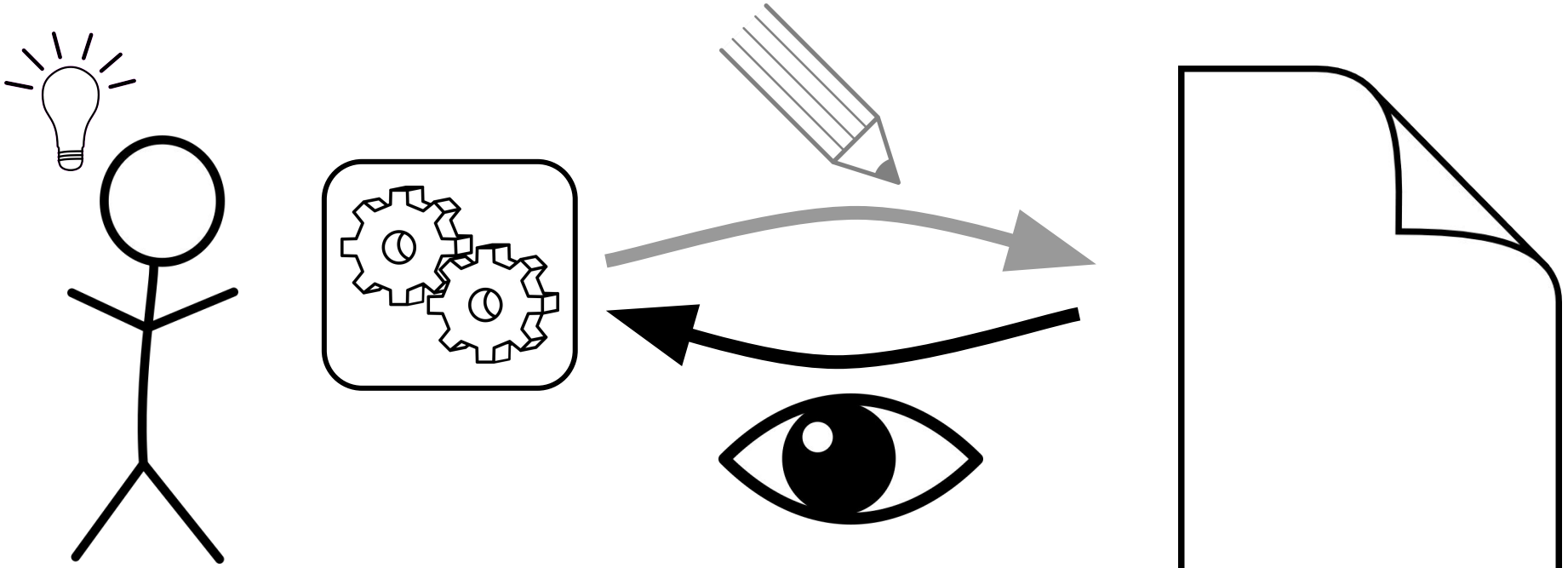
A document without vowels



Enabling people to share information:
a Wonderful digital language.

Ever tried to import someone else's **Word** document ?

"Ouch !"

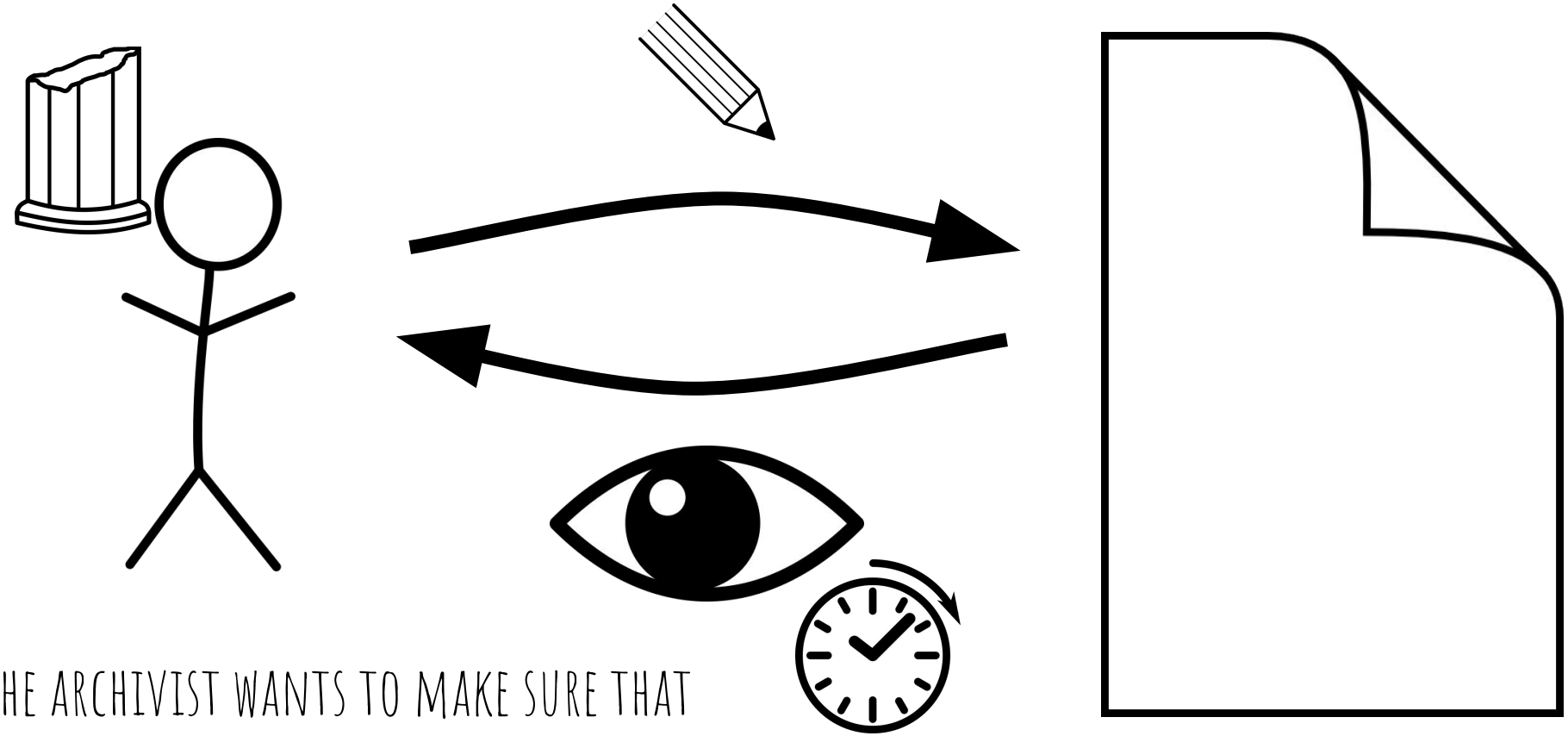


The developer relies on the specifications to add support in their library.

Preserve...

- Insert rant here -

Web pages anyone ? ;)



The archivist wants to make sure that their data will be re-usable *much* later.

Ever tried to re-import an *old* Word document ?

I'm also a Reverse Engineer

- Interested since 1989
- Video games preservation in 1999

comme endommagés dans la table d'allocation des fichiers. Par chance, il n'attaque que les IBM PC-XT. Pour s'en débarrasser, il faut rétablir les pistes de démarrage dans leur état d'origine. Avec un éditeur d'octets du type PC-Tools, vérifiez la présence des octets 33 C0 dans les zones 30 et 31 du secteur d'amorçage du disque dur ; s'ils sont bien présents, mieux vaut exécuter la commande SYS depuis une disquette Système saine ; à la fin de la première table d'allocation des fichiers du disque dur, remplacez les trois derniers octets (FF 7F FF) par FF 0F 00. Puis localisez le code du virus lui-même, qui commence par FF 06 F3 7D 8B 1E, et remplacez-le (ainsi que tous les octets qui suivent, jusqu'à 55 AA) par F6 si le formatage est dû à la commande FORMAT du système, ou par 00 s'il provient de PC-Tools. Si l'opération vous semble trop com-

Science & Vie Micro, November 1989

Instructions to **manually** remove a boot sector virus

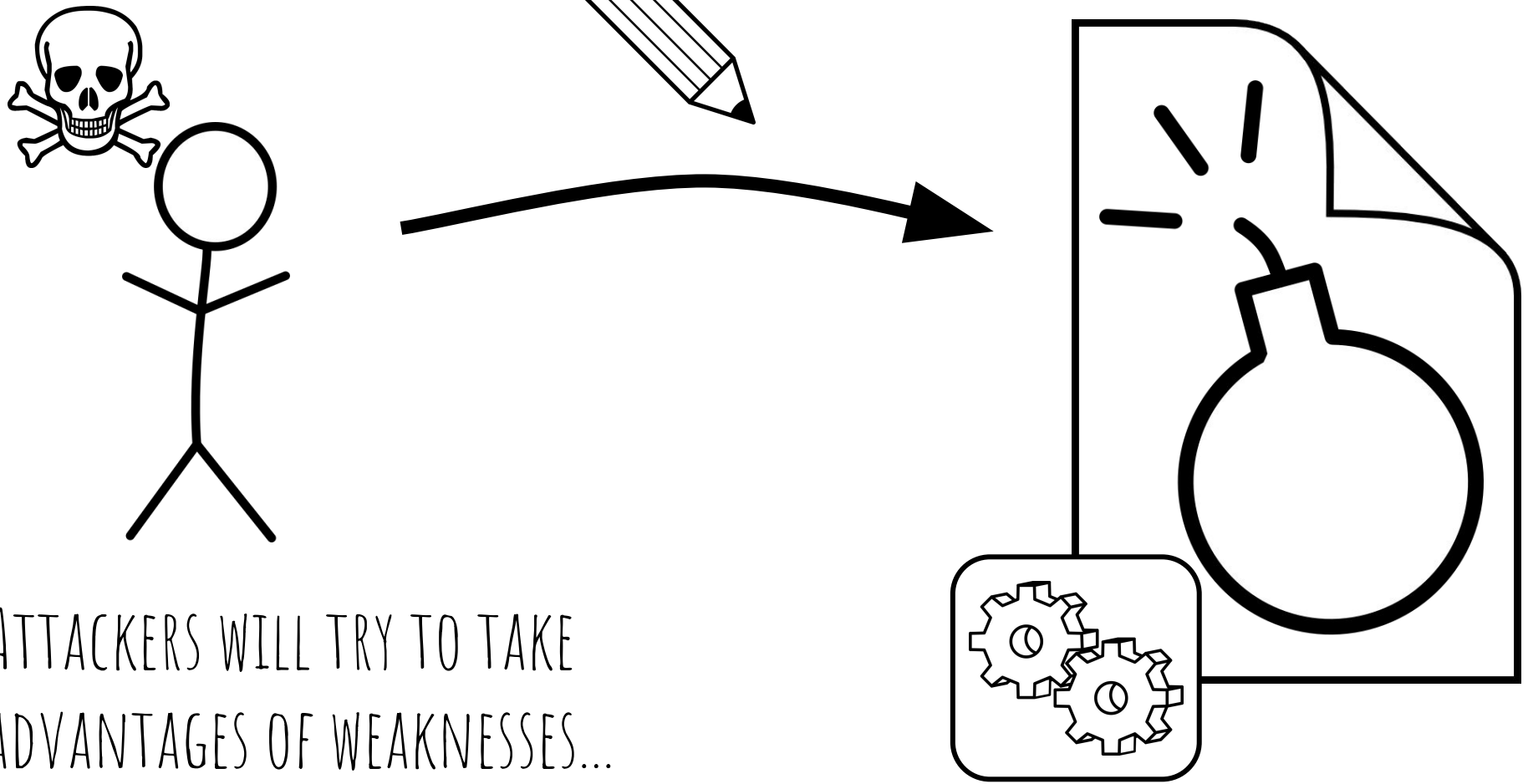
But...

We live in dark times!

Professionally

- 12 years of malware analysis
- Executables, documents...

Note: this talk reflects my **own** opinion, not my employer.



Attackers will try to take advantages of weaknesses...

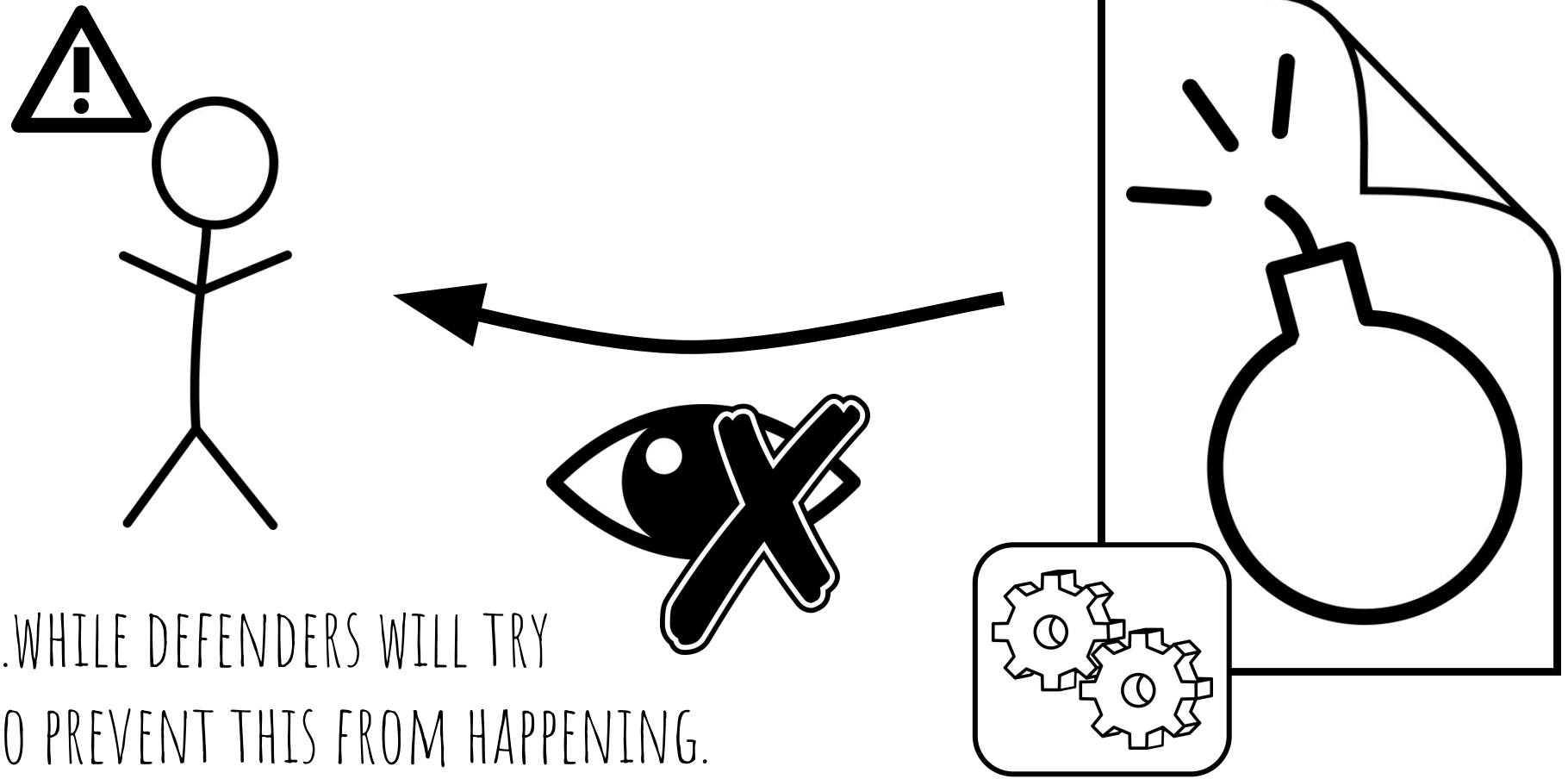
Black hats

Find (and sell) **vulnerabilities**
in software that can be exploited
to steal information or money,
or spy on people (and get them arrested...).

Goals: bypass security, hack into systems.

Find bugs to hack your target.

- Analyze code (source)
  or disassembly (the binary itself)
- Blind fuzzing (flip bits randomly and see what happens)
- Generational fuzzing (according to the structure of the file format)
- Smart fuzzing (modify/instrument/analyse code)



...while defenders will try
to prevent this from happening.

Do the same as the bad guys

But disclose and fix vulnerabilities (see Bug Bounties)

- Remove the low-hanging fruits from the tree.
- Controlled burn to prevent huge fires.

=> improve practices

IMHO file formats should receive this treatment too.

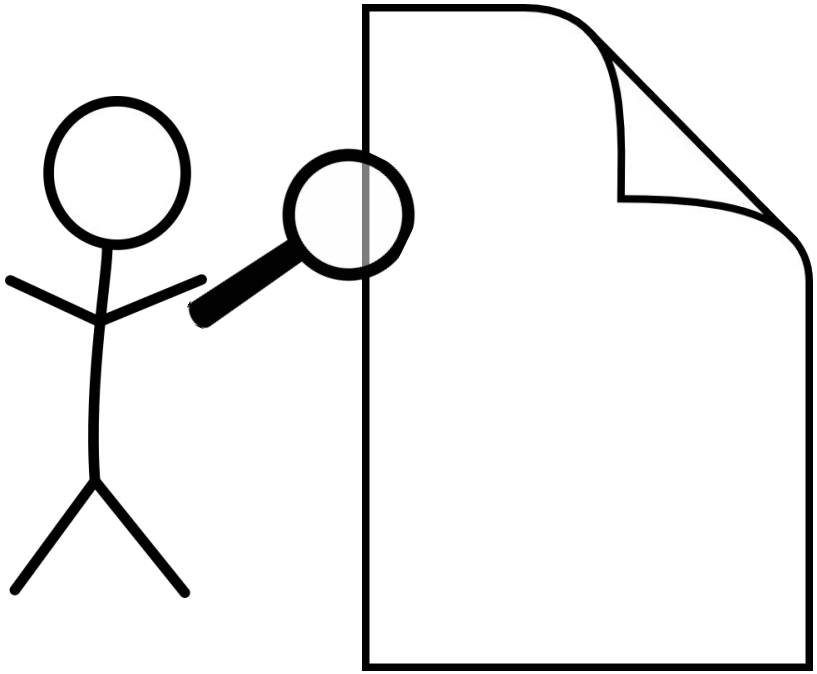
Anti-malware industry

1- Analyze files

2- Determine if they're corrupted, or malicious

3- come up with ways to detect them

4- improve defenses



Malware analyst or DFIR are looking for clues.
(typically, I open files with a hex editor first)

Digital Forensics - Incident Response

determine if and how a file or system was:

- hacked
- tampered with (casino machines)
- If data was stolen (copyright infrigment)

One more thing...

So far, just InfoSec things:
nothing for the DigiPres world.

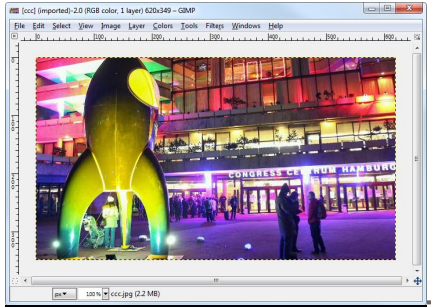
Professionally (next)

- I was fed up of being only retroactive to attackers' "innovation"
- I started to experiment and create my own files, from **scratch**.

And share them openly and freely.

Polyglots

Image



Java

A single file with multiple types:

It's not a gadget,

It's actually useful to hack people!

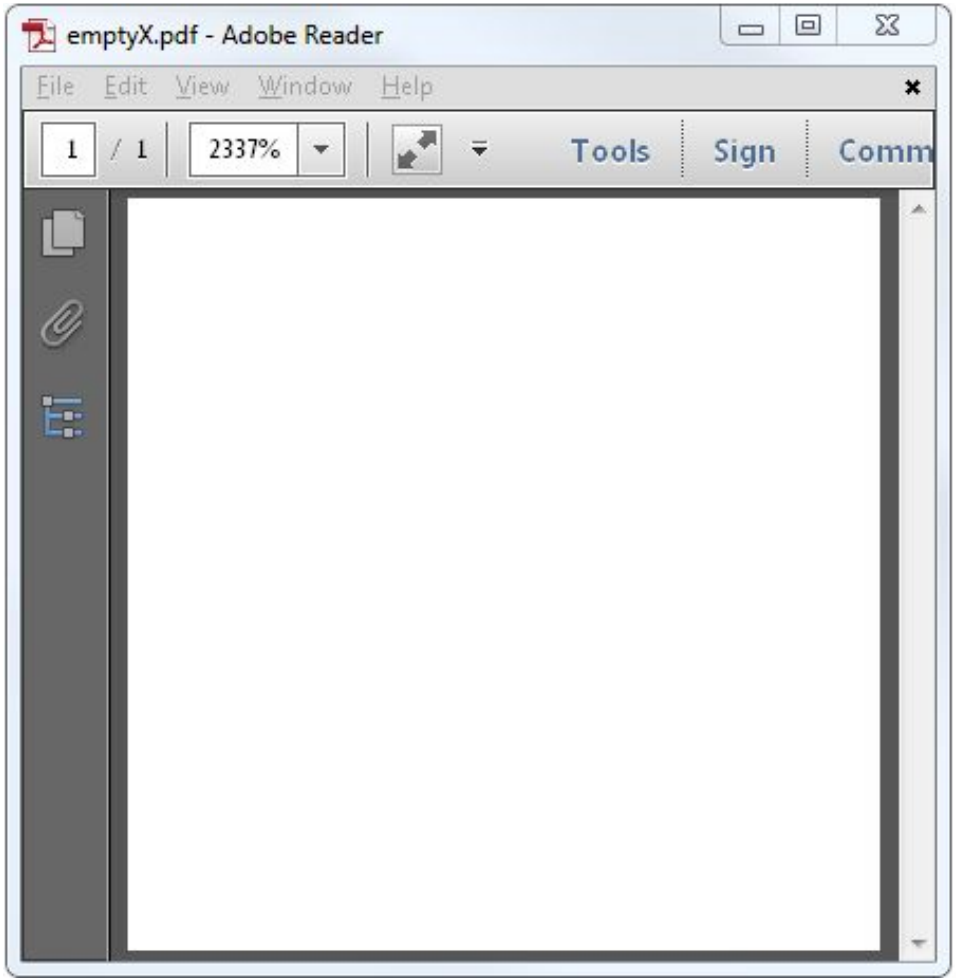
Used in the wild, in 2008!

https://en.wikipedia.org/wiki/GIFAR

"Too small to be Suspicious!"

(or even considered valid)

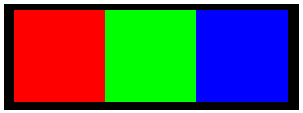
```
%PDF-\0trailer<</Root<</Pages<<>>>>>>
```

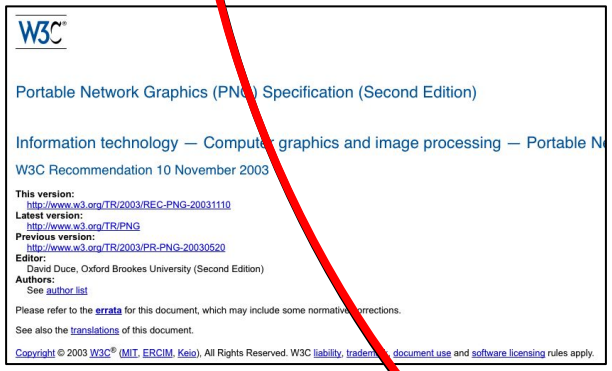


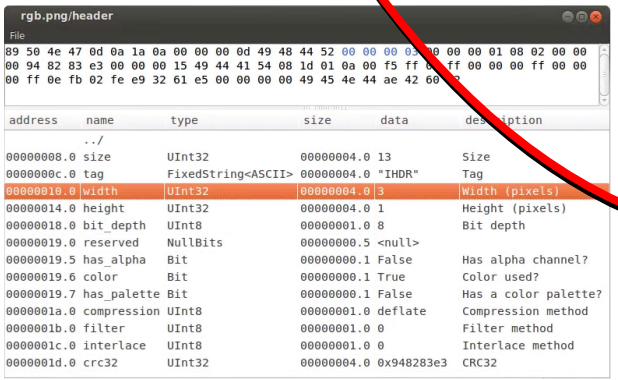
A mini PDF (Adobe-only, 36 bytes)

How do I do it ?

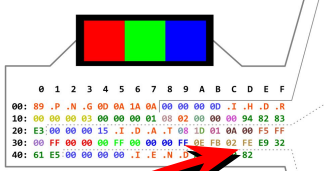
1. I study a simple file with the specs
2. I create my own script to reproduce it
   (Standard tools usually don't give you total control)
3. I can now create my own files, with full control
4. I can then experiment,
   and optionally, document and visualize.

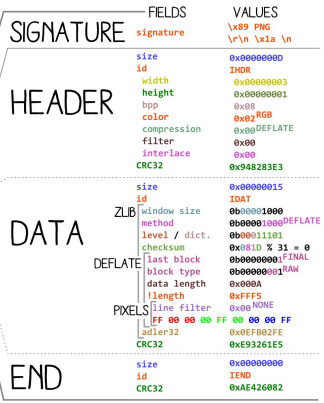


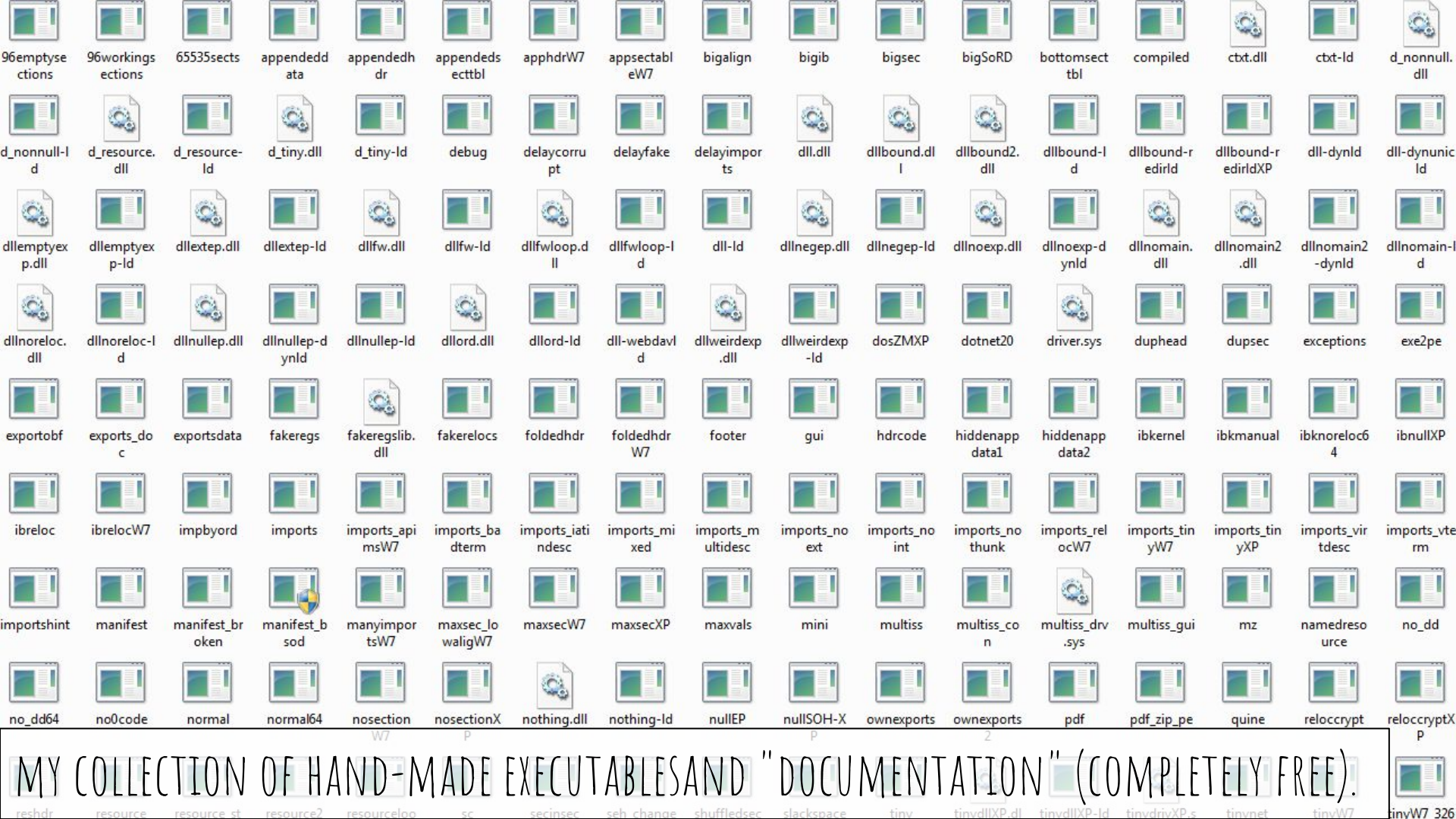





```python
import struct
import binascii
import zlib

def make(chunks):
  s = ["\x89PNG\x0d\x0a\x1a\x0a"]
  for type_, data in chunks:
    s += [
      struct.pack(">I", len(data)), # length
      type_,                        # type
      data,                         # data
      struct.pack(">I",             # CRC
          binascii.crc32(type_ + data)
            % 0x100000000)
      ]
  return "".join(s)

with open("rgb.png", "wb") as f:
  f.write(make([
    ["IHDR", # Image Header
      struct.pack(">IIBBBBB",
        3, # Width
        1, # Height
        8, # Depth: 8
        2, # Colour type: RGB
        0, # Compression: Deflate
        0, # Filter: adaptive filtering
        0  # Interlace: no
        )],
    ["IDAT", # Image Data
      zlib.compress(
      "\0" +          # no interlacing
      "\xFF\x00\x00" +  # Red pixel
      "\x00\xFF\x00" +  # Green pixel
      "\x00\x00\xFF"    # Blue pixel
      ,0)], # no compression
    ["IEND", # Image End
      ""]    # Empty chunk
    ]))
```





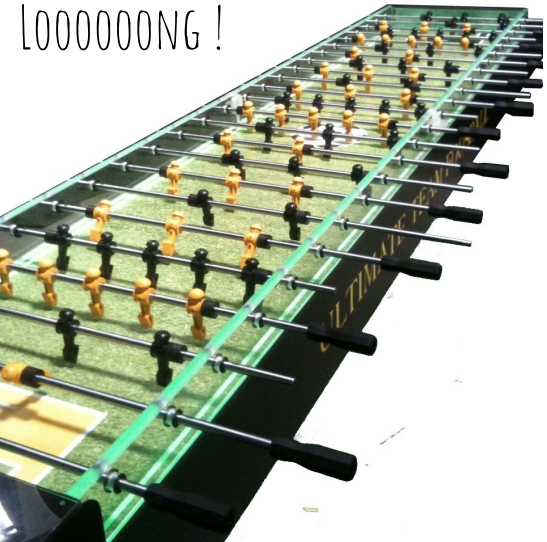

96emptysections
96workingsections
65535sects
appendeddata
appendedhdr
appendedsecttbl
apphdrW7
appsectableW7
bigalign
bigib
bigsec
bigSoRD
bottomsecttbl
compiled
ctxt.dll
ctxt-ld
d_nonnull.dll
d_nonnull-ld
d_resource.dll
d_resource-ld
d_tiny.dll
d_tiny-ld
debug
delaycorrupt
delayfake
delayimports
dll.dll
dllbound.dll
dllbound2.dll
dllbound-ld
dllbound-redirld
dllbound-redirldXP
dll-dynld
dll-dynunicld
dllemptyexp.dll
dllemptyexp-ld
dllextep.dll
dllextep-ld
dllfw.dll
dllfw-ld
dllfwloop.dll
dllfwloop-ld
dll-ld
dllnegep.dll
dllnegep-ld
dllnoexp.dll
dllnoexp-dynld
dllnomain.dll
dllnomain2.dll
dllnomain2-dynld
dllnomain-ld
dllnoreloc.dll
dllnoreloc-ld
dllnullep.dll
dllnullep-dynld
dllnullep-ld
dllord.dll
dllord-ld
dll-webdavld
dllweirdexp.dll
dllweirdexp-ld
dosZMXP
dotnet20
driver.sys
duphead
dupsec
exceptions
exe2pe
exportobf
exports_doc
exportsdata
fakeregs
fakeregslib.dll
fakerelocs
foldedhdr
foldedhdrW7
footer
gui
hdrcode
hiddenappdata1
hiddenappdata2
ibkernel
ibkmanual
ibknoreloc64
ibnullXP
ibreloc
ibrelocW7
impbyord
imports
imports_apimsW7
imports_badterm
imports_iatindesc
imports_mixed
imports_multidesc
imports_noext
imports_noint
imports_nothunk
imports_relocW7
imports_tinyW7
imports_tinyXP
imports_virtdesc
imports_vterm
importshint
manifest
manifest_broken
manifest_bsod
manyimportsW7
maxsec_lowaligW7
maxsecW7
maxsecXP
maxvals
mini
multiss
multiss_con
multiss_drv.sys
multiss_gui
mz
namedresource
no_dd
no_dd64
no0code
normal
normal64
nosectionW7
nosectionXP
nothing.dll
nothing-ld
nullEP
nullSOH-XP
ownexports
ownexports2
pdf
pdf_zip_pe
quine
reloccrypt
reloccryptXP
MY COLLECTION OF HAND-MADE EXECUTABLESAND "DOCUMENTATION" (COMPLETELY FREE).
reshdr
resource
resource_st
resource2
resourceloo
sc
secinsec
seh_change
shuffledsec
slackspace
tiny
tinydllXP.dl
tinydllXP-ld
tinydrivXP.s
tinynet
tinyW7
tinyW7_326

Victor Frankeinstein

Initially, I started with simpler stuff

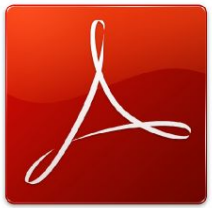
But... it excalated quickly :)

Looooooong !

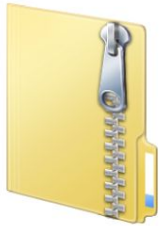


Dual headed cow

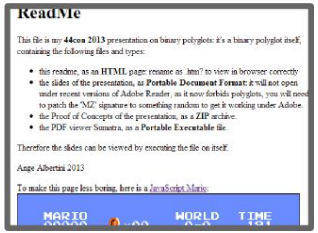
PDF viewer



PDF slides



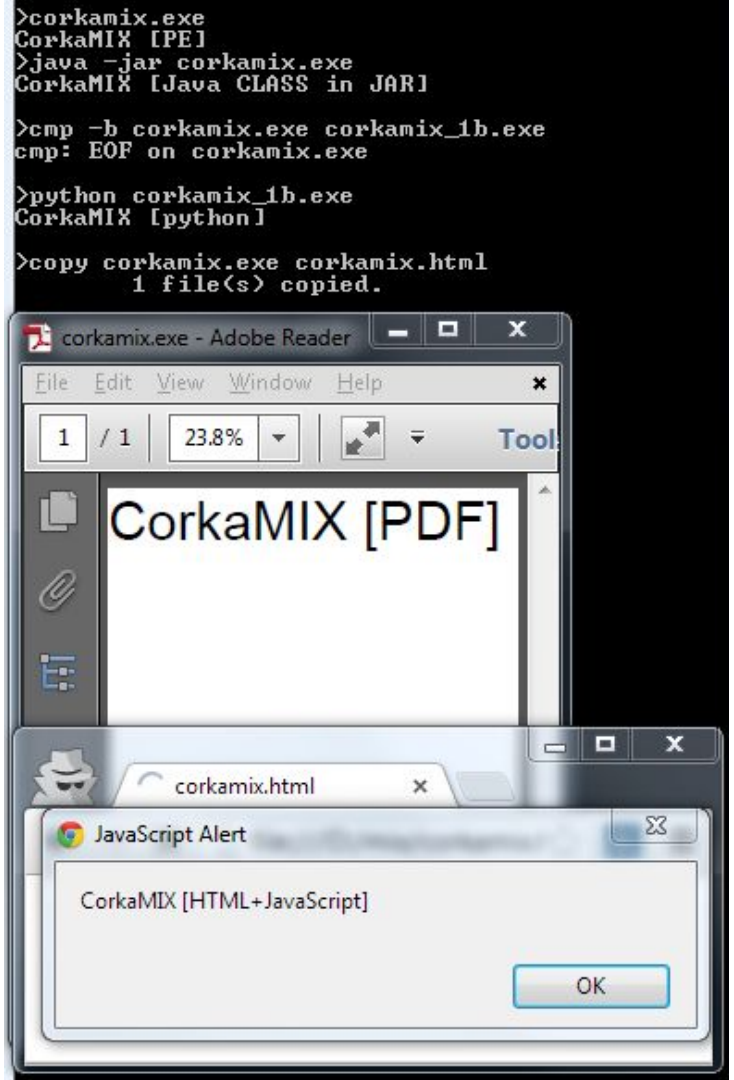
HTML



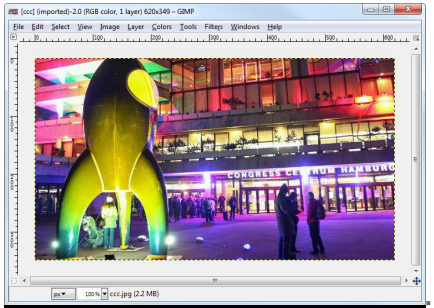
A PRESENTATION SLIDE DECK VIEWING ITSELF (PDF VIEWER AND PDF DOCUMENT)

HTML JavaScript Java Windows executable PDF

2 standard infection chains in a **single** file



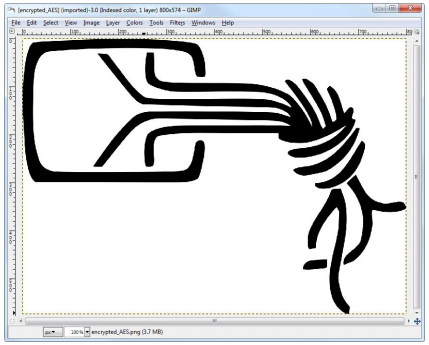
JPG



JAR
(ZIP + CLASS)

```
>java -jar ccc.jpg
Hello World! [Java]

>
```

AES_{K_1}



PNG

AES_{K_2}

FLV

3DES

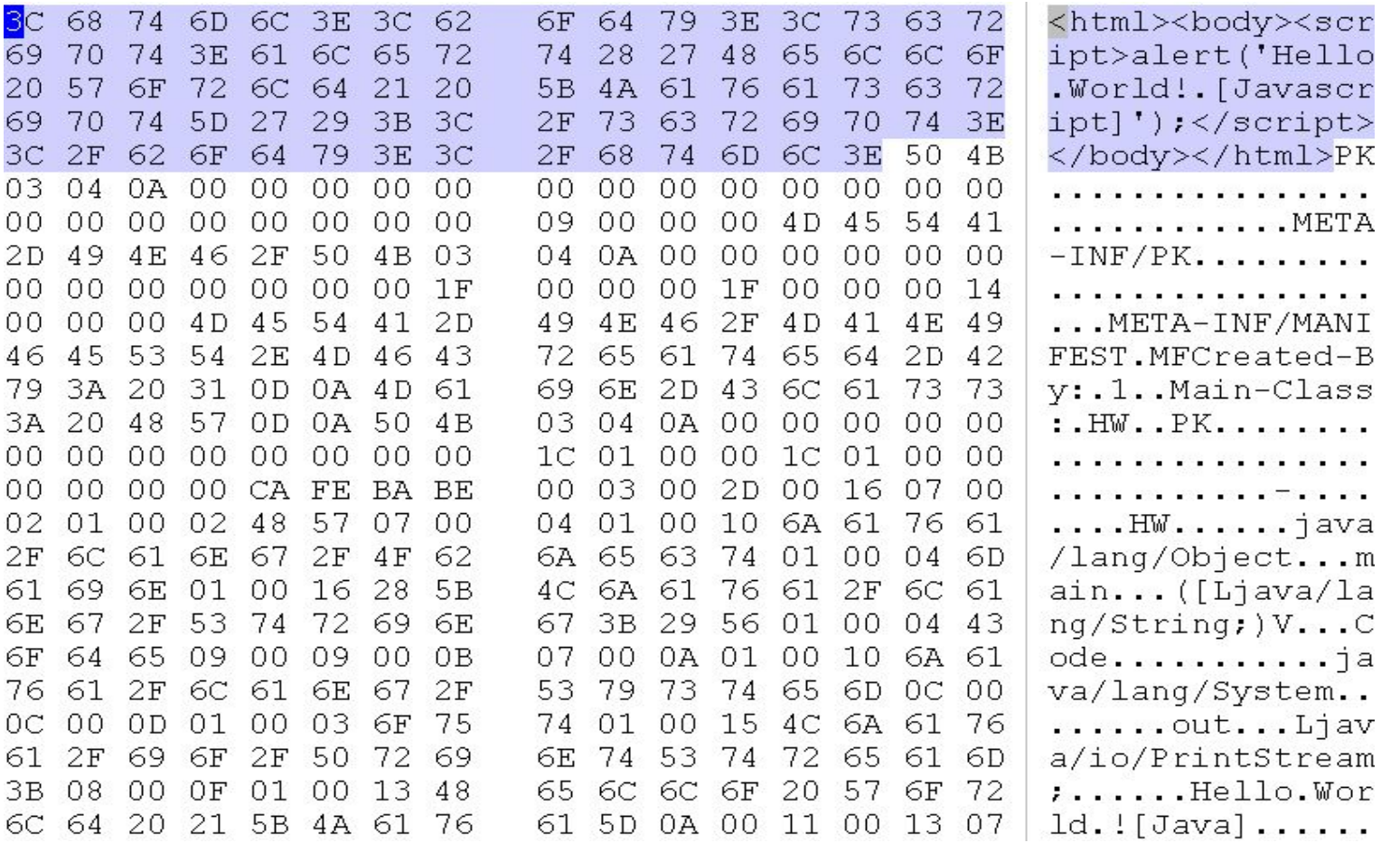
PDF

Mixing binary and cryptography

unicode //

```
\u002f\u002f<html>
\u002f\u002f    <body>
\u002f\u002f        <script>
\u002f\u002f            alert('Hello World! [Javascript]');
\u002f\u002f        </script>
\u002f\u002f    </body>
\u002f\u002f </html>

public class HW
{
    public static void main(String[] args)
    {
        System.out.println("Hello World! [Java]");
    }
}
```

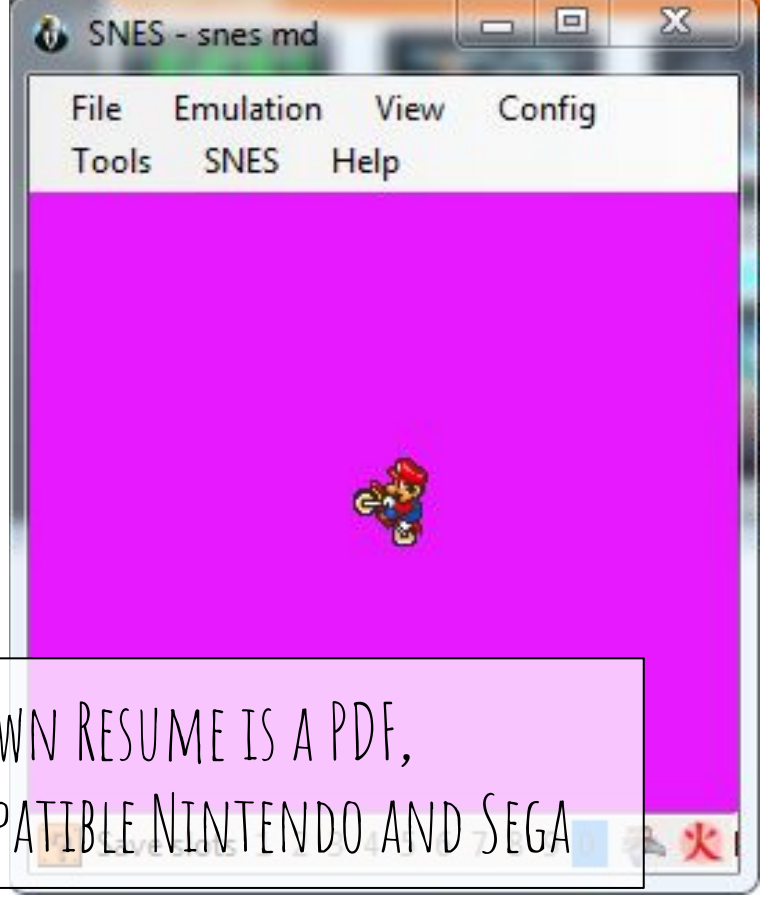
A Java & JavaScript polyglot - at source level



A Java & JavaScript polyglot - at binary level

=> Java = JavaScript

Yes, your management was right all along ;)

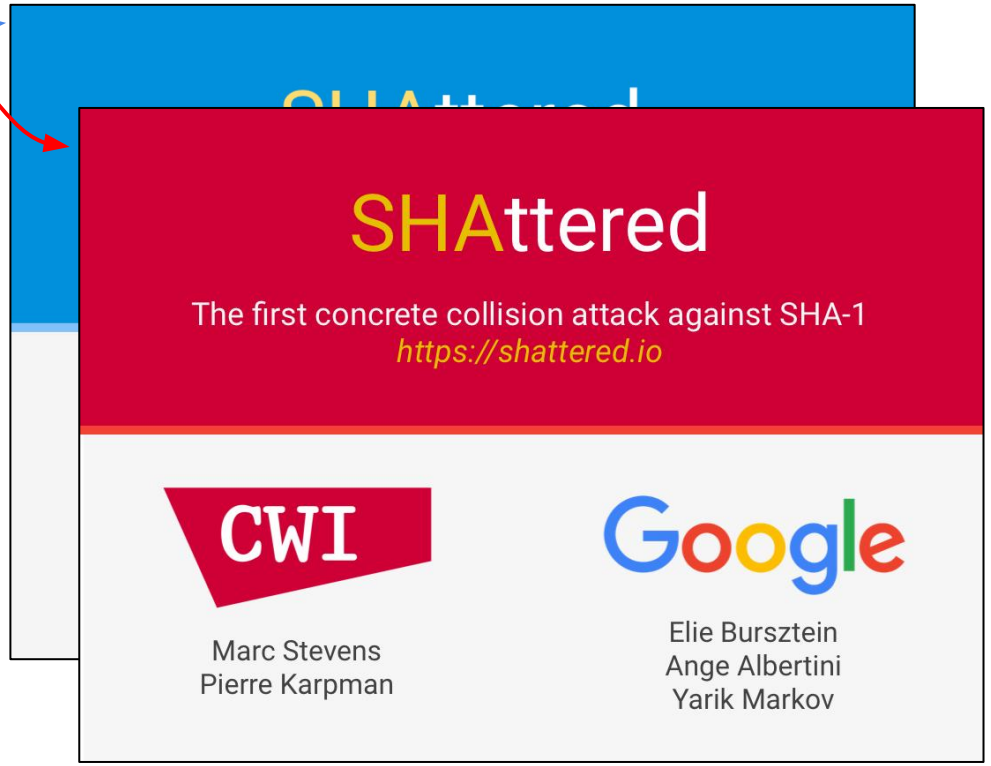


My own Resume is a PDF, compatible Nintendo and Sega

NINTENDO

Sega®

same SHA-1



I crafted the binary structure of the files with the first SHA1 collision.

These files violate the specifications!
And yet they work everywhere.

The JPEG File Interchange Format is entirely compatible with the standard JPEG interchange format; the only additional requirement is the mandatory presence of the APP0 marker right after the SOI marker. Note that JPEG interchange format requires (as does JFIF) that all table specifications used in the encoding process be coded in the bitstream prior to their use.

https://www.w3.org/Graphics/JPEG/jfif3.pdf

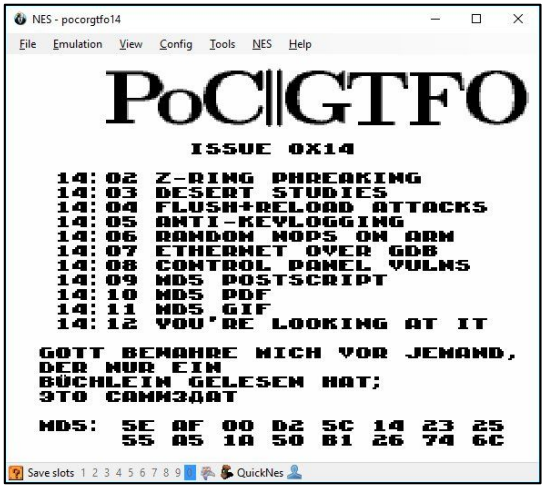
These files display their **own** MD5!

(not by me)

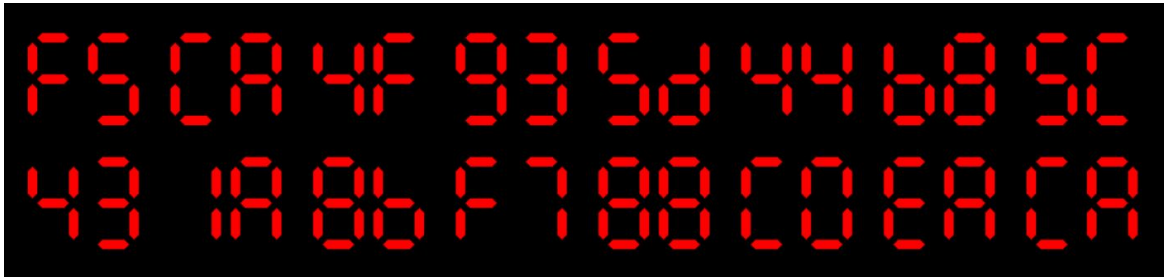
PDF

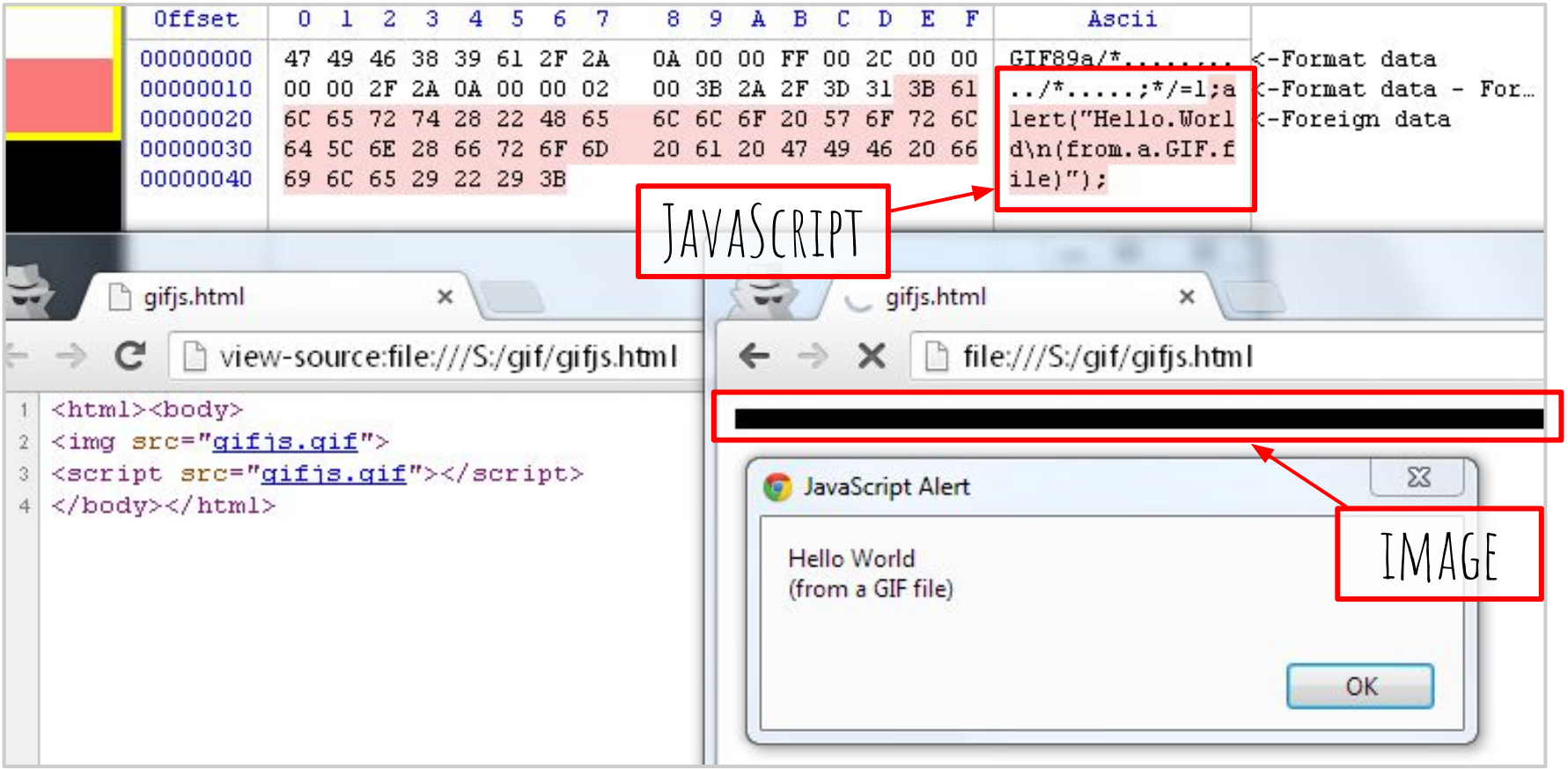
```
$ pdftotext -q md5text.pdf -
66DA5E07C0FD4C921679A65931FF8393


$ md5sum md5text.pdf
66da5e07c0fd4c921679a65931ff8393  md5text.pdf
```

Nintendo Rom



GIF





A JavaScript // GIF polyglot (useful to embed payload - also works with JPG or BMP)

Isn't this all

"useless?"

I remove potential traps by researching,
And I train myself with extreme files.
Also, some of these were used in the wild to attack people,
And I can reproduce key features into shareable files.

My perspective:

1- What is a file?

A sequence of byte(s)

Any parser can give an incomplete perspective.

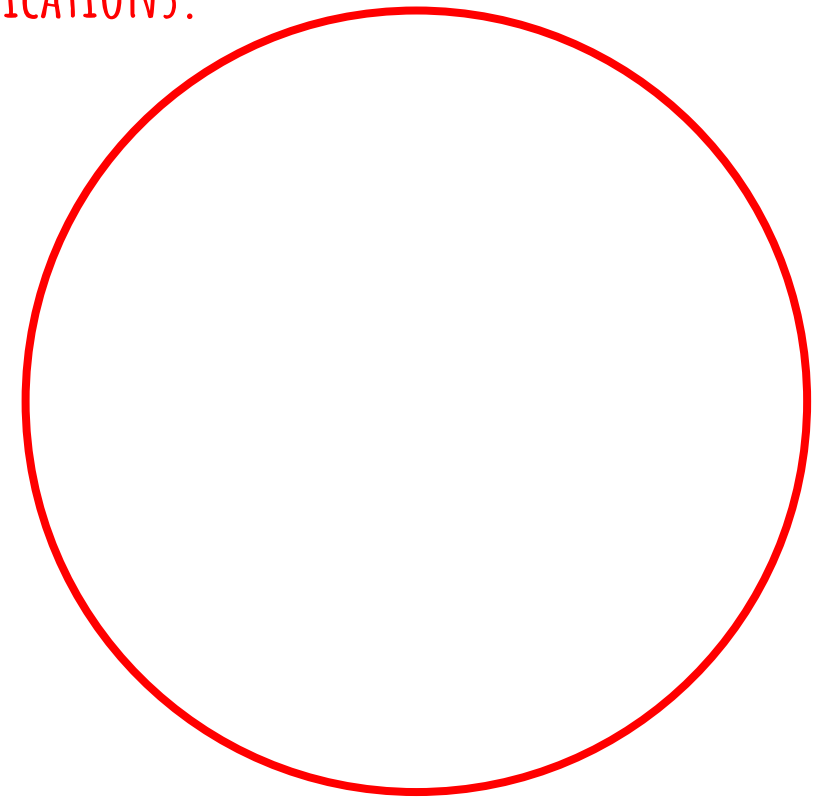
I open most files first with a hex editor (out of curiosity, at least)

Yes, I'm a hex-addict ;)

2- What is a **VALID** file?

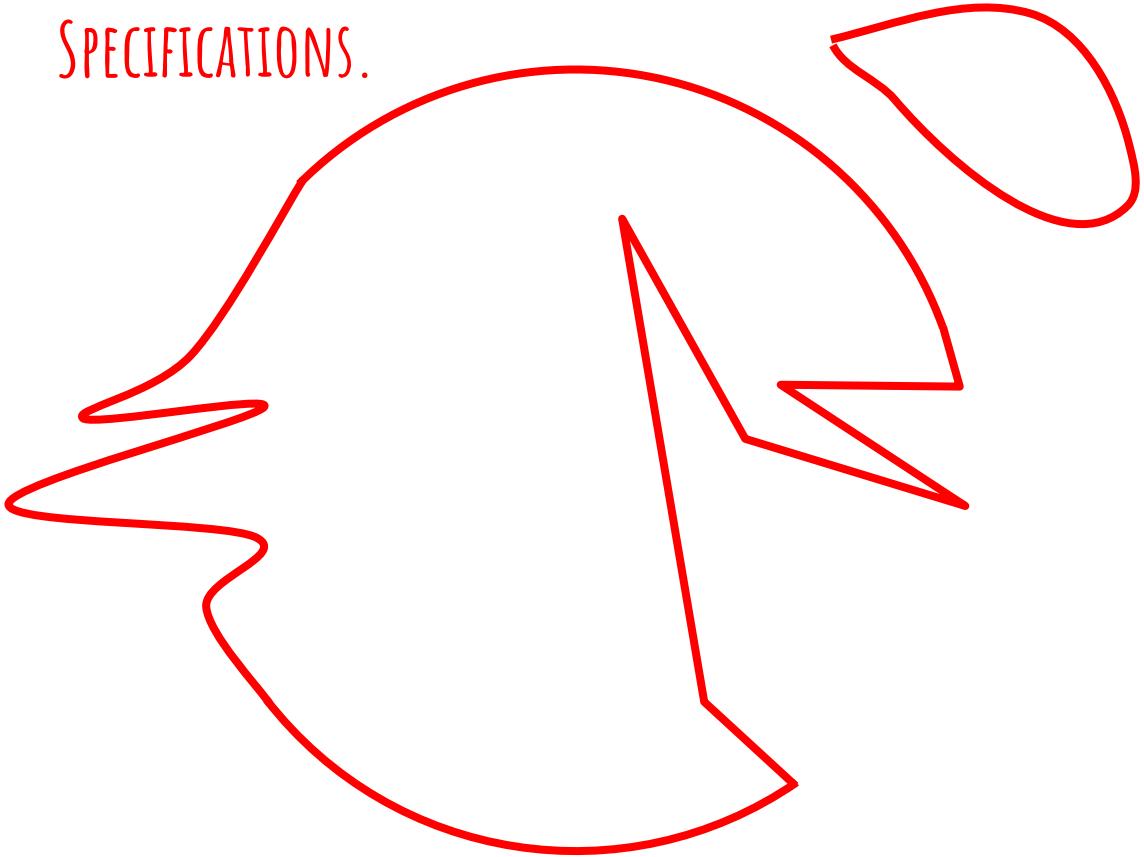
A file loaded successfully
by a parser/loader/processor.
A file in itself is **NOTHING**.

Specifications.

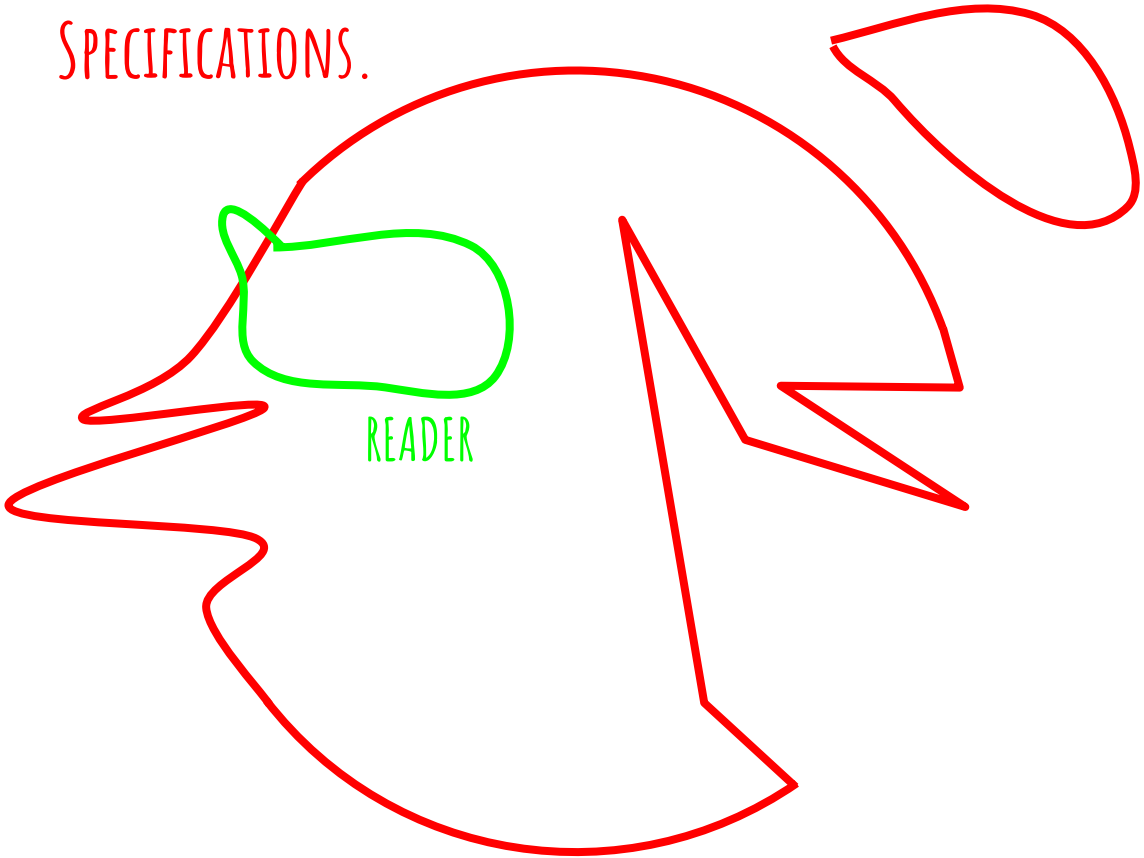


What we wished...

Specifications.



In reality, they are more complex,
often for no particular reason (see design by commitee)



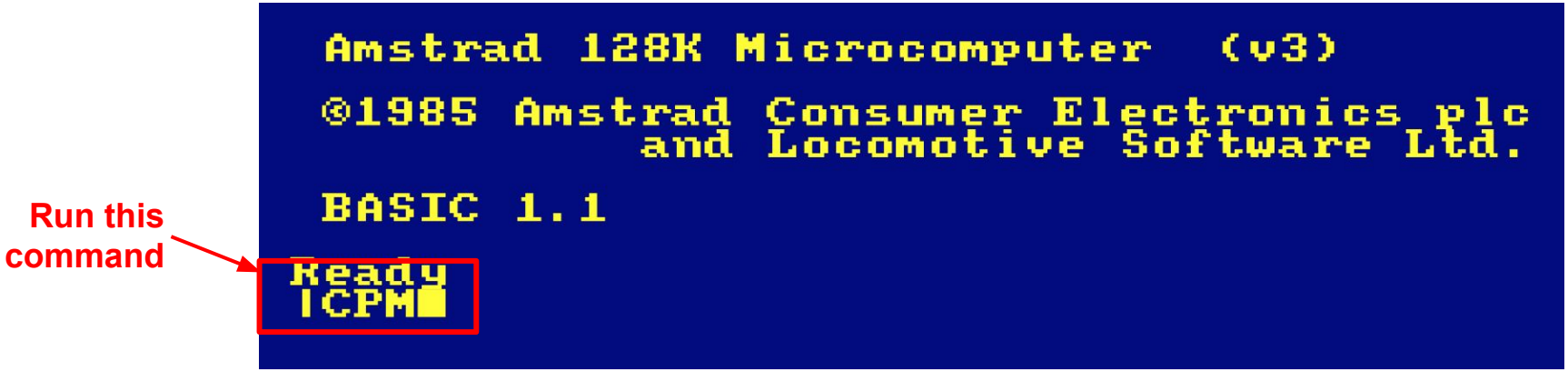
Now comes a software that takes these files as input.
This software defines validity (the parser/loader), not the specifications.

Specifications are irrelevant!

As long as the file 'works as intended'.

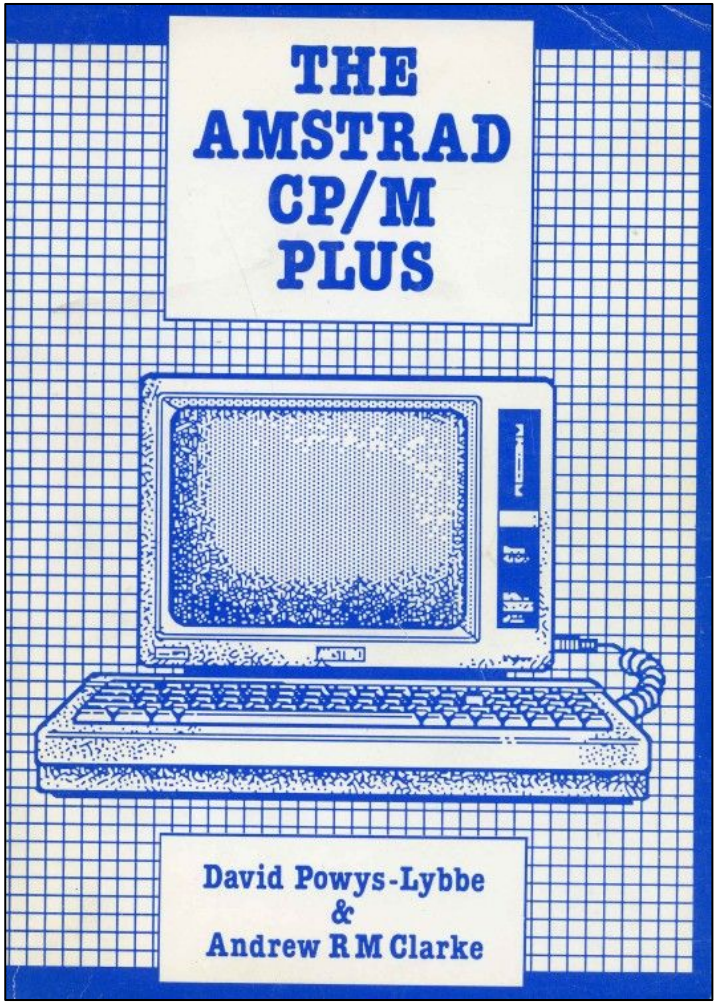
On this computer...

We'll launch...

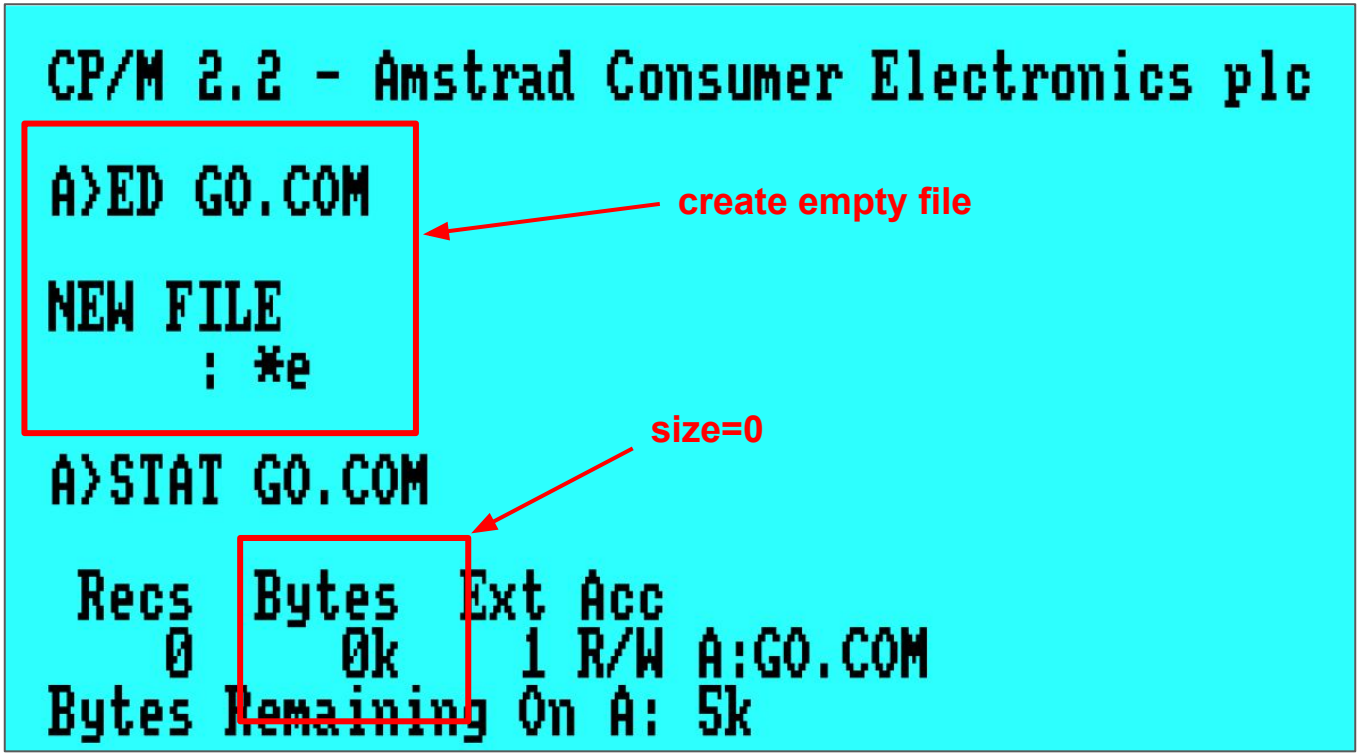

Amstrad 128K Microcomputer (v3)
©1985 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.
BASIC 1.1
Ready
|CPM
Run this command

...THIS OS.


Amsoft
AMSTRAD CONSUMER ELECTRONICS plc
3
UTILIDADES DE PROGRAMACION
CP/M PLUS NO SERIE 1006/5000-1232-
2408347
EL USO DE ESTOS PROGRAMAS ESTA SUJETO A LAS CONDICIONES DEL ACUERDO DE LICENCIA DE USUARIO FINAL ESTABLECIDAS EN EL APENDICE DEL MANUAL DE INSTRUCCIONES DEL USUARIO DE ESTE PRODUCTO
PROGRAMAS © COPYRIGHT DIGITAL RESEARCH INC. & AMSTRAD CONSUMER ELECTRONICS plc


THE AMSTRAD CP/M PLUS
David Powys-Lybbe
&
Andrew R M Clarke

Let's create... an EMPTY file!

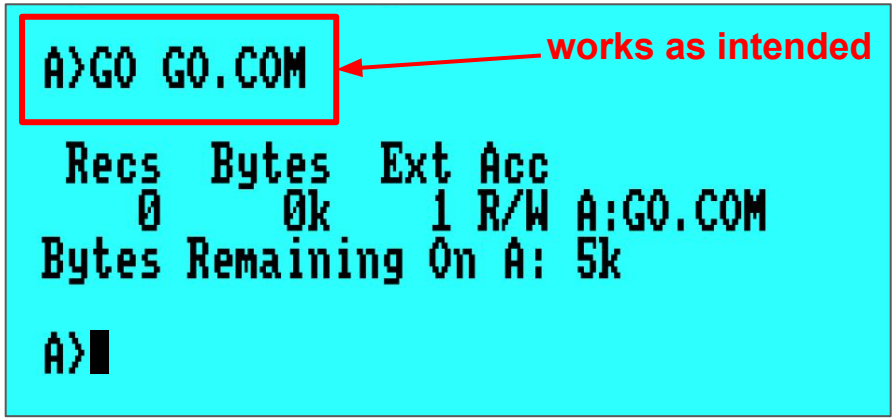


Is it valid?

Yes: **Transient Commands** are copied blindly and execution started at offset zero.

Does it do anything?

Transient Memory Area is **not**
Cleared between executions,
so the previous command is re-executed.



Under a commercial OS from 1985,

the **EMPTY** file is valid, useful and reliable.

It was even **SOLD** as a commercial program for £5.

Takeaway

- it's old-school, obsolete...
- And yet this example is **pure** simplicity...
  to prove that the software defines the rules!
- The specifications are volatile....
  the software is the ground truth.

Text files

What could go wrong?

Just a bad encoding maybe?

This is a ...
malicious Flash file !!

No, not base64 - it's directly executable as is!

But, aren't Flash files...
pure binary!?

```
CWSMIKI0hCD0Up0IZUnnnnnnnnnnnnnnnnnnnUU5nnnnnn3Snn7iiudIbEAt333swW0ssG03
sDDtDDDt0333333Gt333swwv3wwwFPOHtoHHvwHHFhH3D0Up0IZUnnnnnnnnnnnnnnnnnnnU
U5nnnnnn3Snn7YNqdIbeUUUfV13333333333333333s03sDTVqefXAxooooD0CiudIbEAt33
swwEpt0GDG0GtDDDtwwGGGGGsGDt33333www033333GfBDTHHHHUhHHHeRjHHHhHHUccUSsg
SkKoE5D0Up0IZUnnnnnnnnnnnnnnnnnnnUU5nnnnnn3Snn7YNqdIbe13333333333sUUe133
333Wf03sDTVqefXA8oT50CiudIbEAtwEpDDG033sDDGtwGDtwwDwttDDDGwtwG33wwGt0w33
333sG03sDDdFPhHHHbWqHxHjHZNAqFzAHZYqqEHeYAHlqzfJzYyHqQdzEzHVMvnAEYzEVHMH
bBRrHyVQfDQflqzfHLTrHAqzfHIYqEqEmIVHaznQHzIIHDRRVEbYqItAzNyH7D0Up0IZUnnn
nnnnnnnnnnnnnnnnnnUU5nnnnnn3Snn7CiudIbEAt33swwEDt0GGDDDGptDtwwG0GGptDDww0G
DtDDDGGDDGDDtDD33333s03GdFPXHLHAZZOXHrhwXHLhAwXHLHgBHHhHDEHXsSHoHwXHLXAw
XHLxMZOXHWHwtHtHHHHLDUGhHxvwDHDxLdgbHHhHDEHXkKSHuHwXHLXAwXHLTMZOXHeHwtHt
HHHHLDUGhHxvwTHDxLtDXmwTHLLDxLXAwXHLTMwlHtxHHHDxLlCvm7D0Up0IZUnnnnnnnnnn
nnnnnnnnnUU5nnnnnn3Snn7CiudIbEAtuwt3sG33ww0sDtDt0333GDw0w33333www033GdFP
DHTLxXThnohHTXgotHdXHHHxXTlWf7D0Up0IZUnnnnnnnnnnnnnnnnnnnUU5nnnnnn3Snn7C
iudIbEAtwwWtD333wwG03www0GDGpt03wDDDGDDD33333s033GdFPhHHkoDHDHTLKwhHhzoD
HDHTlOLHHhHxeHXWgHZHoXHTHNo4D0Up0IZUnnnnnnnnnnnnnnnnnnnUU5nnnnnn3Snn7Ciu
dIbEAt33wwE03GDDGwGGDDGDwGtwDtwDDGGDDtGDwwGw0GDDw0w33333www033GdFPHLRDXt
hHHHLHqeeorHthHHHXDhtxHHHLravHQxQHHHOnHDHyMIuiCyIYEHWSsgHmHKcskHoXHLHwhH
HvoXHLhAotHthHHHLXAoXHLxUvH1D0Up0IZUnnnnnnnnnnnnnnnnnnnUU5nnnnnn3SnnwWNq
dIbe13333333333333333WfF03sTeqefXA888ooooooooooooooooooooooooooooooooo
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
ooooooooooooooooooooooooooooooooooo888888880Nj0h
```

```
.F .W .S 07 8C 00 00 00 78 00 05 5F 00 00 0F A0
00 00 0C 01 00 3F 03 6D 00 00 00 88 2C 00 04 00
.m .e .s .s .a .g .e 00 .c .r .e .a .t .e .T .e
.x .t .F .i .e .l .d 00 .t .e .x .t 00 .H .e .l
.l .o .  .W .o .r .l .d .! 00 96 2A 00 07 32 00
00 00 07 64 00 00 00 06 00 00 00 00 00 00 00 00
06 00 00 00 00 00 00 00 00 07 01 00 00 00 08 00
07 06 00 00 00 08 01 3D 17 96 02 00 08 00 1C 96
04 00 08 02 08 03 4F 00 40 00 00 00
```

<= "Hello World" in Flash:
It's a lot of non-ASCII!

Flash can be compressed with ZIP's Deflate.
Which can use the Huffman algorithm,
in which case you supply a code dictionary.
You can craft such a dictionary to 'expand' your data,
But in return it's ascii-only.

https://molnarg.github.io/ascii-flash/ascii-flash.pdf

```
>>> zlib.decompress('\xf3H\xcd\xc9\xc9W\x08\xcf/\xcaIQ\xe4\x02\x00 \x91\x04H',-8)
'Hello World!\n'
>>> zlib.decompress("D0Up0IZUnnnnnnnnnnnnnnnnnnnUU5nnnnnn3SUUnUUUwCiudIbEAtwwwEt3
33wwG0swGpDDGpDDwDDDGtD33333s033333GdFPkWwwOaGOowgQ4", -8)
'Hello World!\n'
```

https://github.com/molnarg/ascii-zip

https://miki.it/blog/2014/7/8/abusing-jsonp-with-rosetta-flash/

Valid Flash files entirely in ASCII:

-> bypassed all filters

-> abused most websites

FLASH FILE TYPES

	TYPE	FILE STRUCTURE
	FLAT	FWS <Version:1> <FileLength:4> <uncompressed data...>
=>	ZLIB	CWS <Version:1> *<FileLength:4> <zlib data>
		<CMF:1> <FLG:1> <dict>* <deflate> *<adler32:4>
	LZMA	ZWS <Version:1> *<FileLength:4> <lzma data> *UNCOMPRESSED

`Version` AND `FileLength` ARE NOT CHECKED.

ADLER32 CHECKSUM

FOR EACH BYTE OF THE UNCOMPRESSED STREAM:

```
.. .. .. .. XX .. .. ..
S1 += XX
S2 += S1
```

FINAL RESULT:

```
ADLER32 = S2 << 16 | S1
```

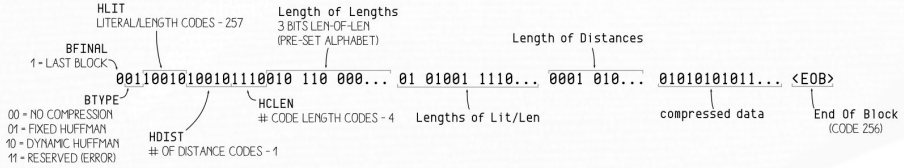
WITH BOTH S1 & S2 MODULO 65521 (LARGEST PRIME <2^16)

FLASH ALLOWS APPENDED DATA AFTER END MARKER:

1. ADJUST S1:
   - APPEND `0xFE` TO *uncompressed* DATA UNTIL S1 IS VALID (`[0-9a-zA-Z./]*`)
   (`0xFF` DOESN'T WORK WELL FOR HUFFMAN MANIPULATION)
2. ADJUST S2:
   - APPEND `0x00` UNTIL S2 IS VALID
   (APPENDING `0x00` DOESN'T AFFECT S1)

ZLIB STREAM START

STRUCTURE OF A DEFLATE BOCK



Specifications are blurry.

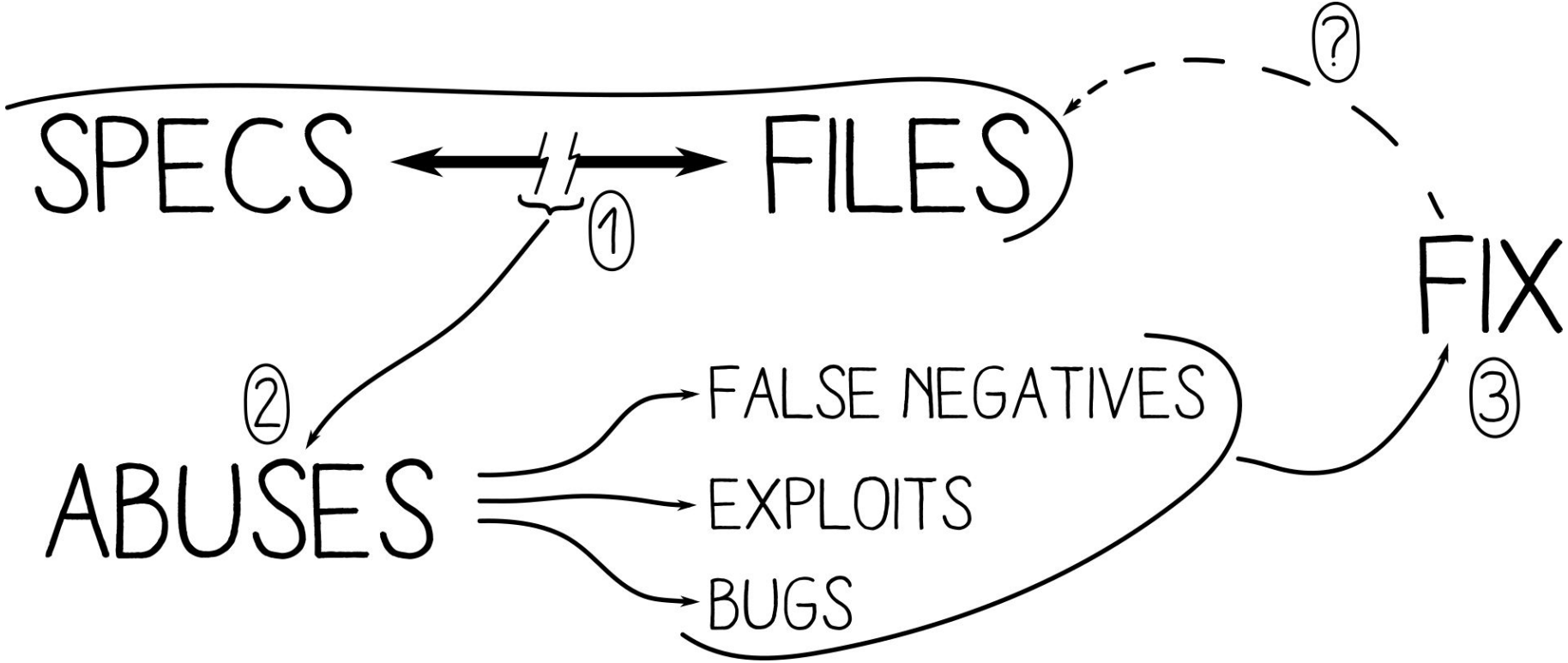
There's always a corner case that is not clarified.

Specifications are imperfect.

I know no perfect specifications.

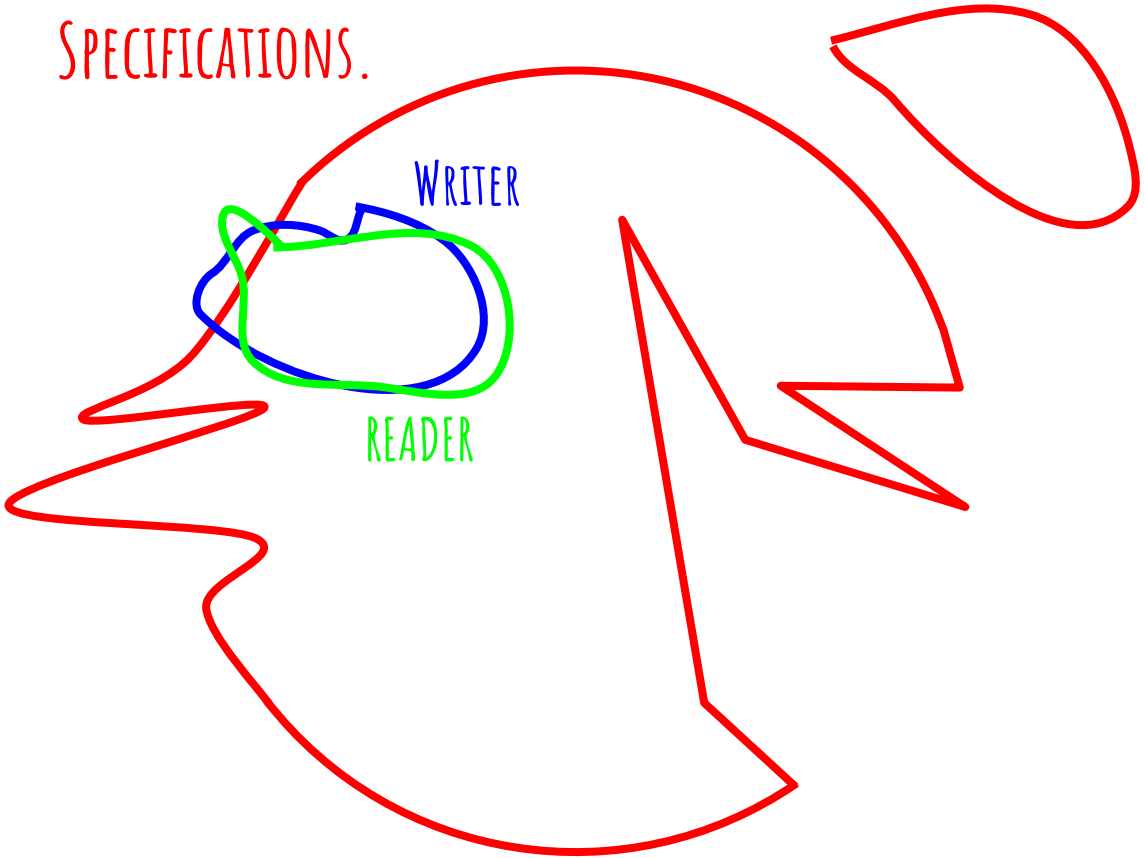
except for the empty file? :)

I could only make specifications **better** by finding problems before they were **finalized** and set in stone.


SPECS
FILES
?
FIX
③
①
②
ABUSES
FALSE NEGATIVES
EXPLOITS
BUGS

The **real** problem

- Unlike laws, specifications are not enforced.
- Not updated either. Set in stone.
- The problem remains the same - and grows with time.
- Survival of the fittest software:
  the Winner decides the rules.



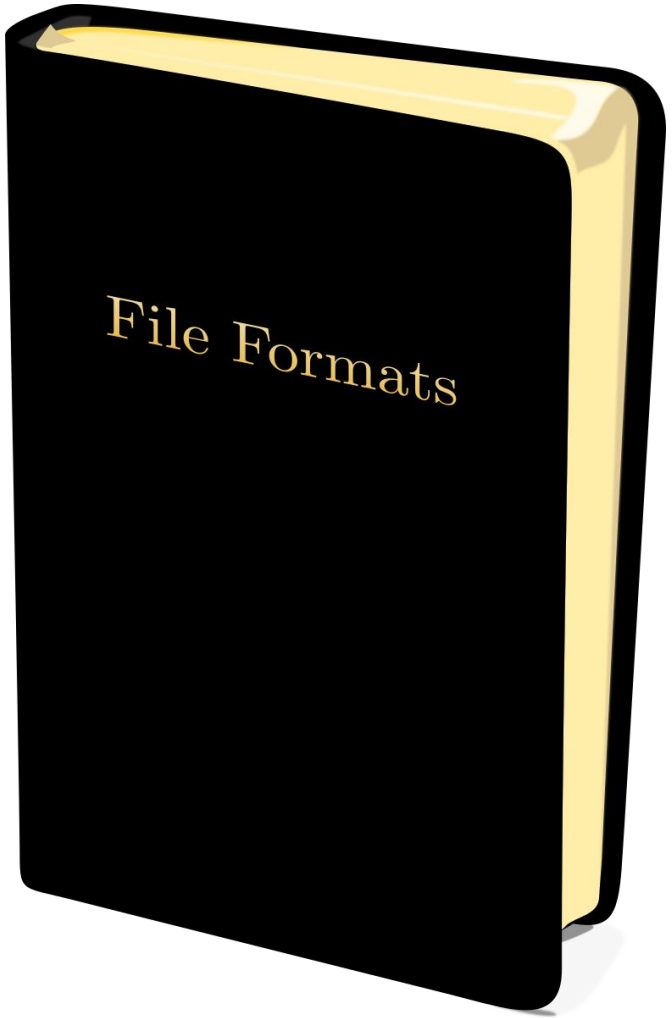
Now comes a software that creates these files.
Their planned features and actual abilities of readers and writers may not overlap.

Large Format Scanners:
Infinite "height" scans
-> image height fixed to 65535!

Tolerated by libjpeg,
So valid **everywhere**!

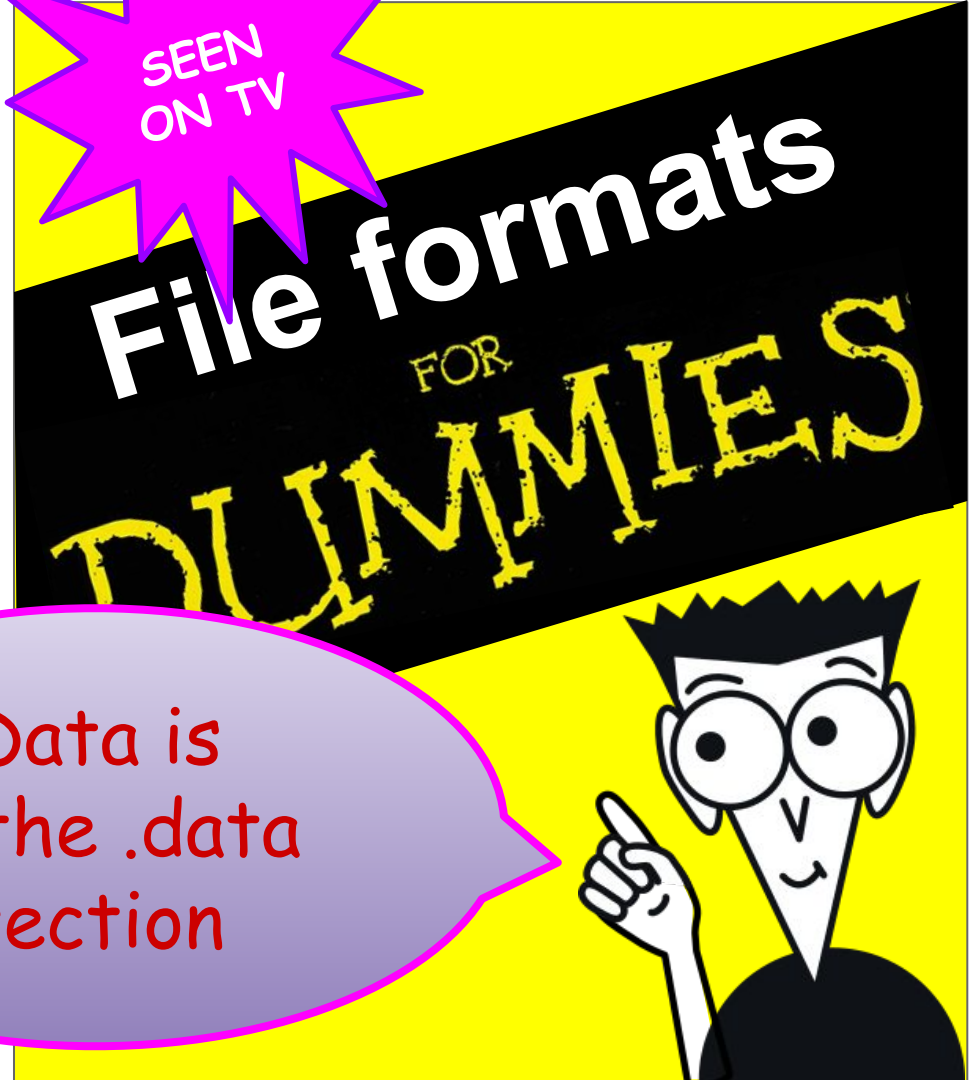
Detected by Anti-Virus, because it was used to exploit MS04-028.

Thou shalt use
Adobe PDF writer..



Specifications are not updated:
They become outdated and irrelevant.

Some specifications are even worse:
They're nothing but...
a "gentle introduction":
Almost useless from the start.



Divergences

As specifications are not perfect,
They are interpreted by different people in different ways:
So one file may work on one reader, not on the other one.

A NORMAL PDF

HEADER

```
%PDF-1.1
```

SIGNATURE & VERSION INFORMATION

XREF TABLE

CROSS REFERENCE

```
xref
0 5
0000000000 65535 f
0000000010 00000 n
0000000047 00000 n
0000000111 00000 n
0000000313 00000 n
```

CROSS REFERENCES
5 OBJECTS, STARTING AT INDEX 0
(STANDARD FIRST EMPTY OBJECT 0
OFFSET TO OBJECT 1, REV 0
TO OBJECT 2...
3...
4

TRAILER

```
trailer
<<
  /Root 1 0 R
>>

startxref
413
%%EOF
```

BODY

DICTIONARY << [ID VALUE]* >>

OBJECT REFERENCE: <OBJECT NUMBER> <REVISION NUMBER> R

IDENTIFIER (WITH /)

ARRAY

STREAM PARAMETERS: LENGTH, COMPRESSION.....

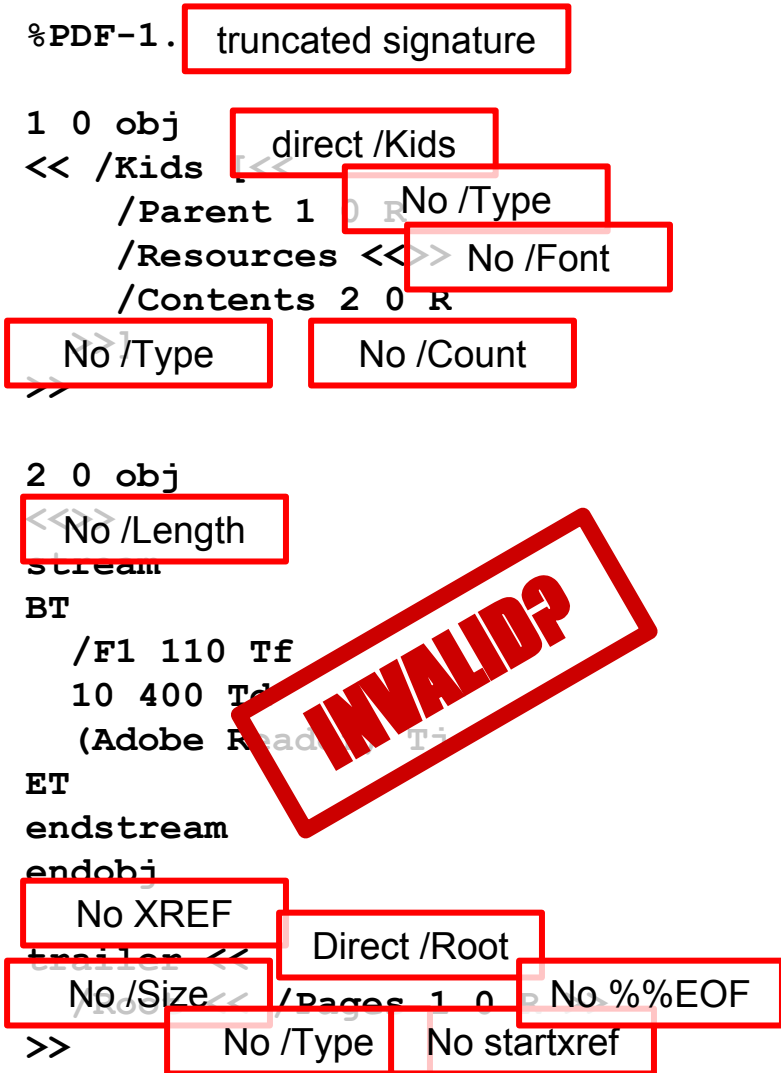
STRING

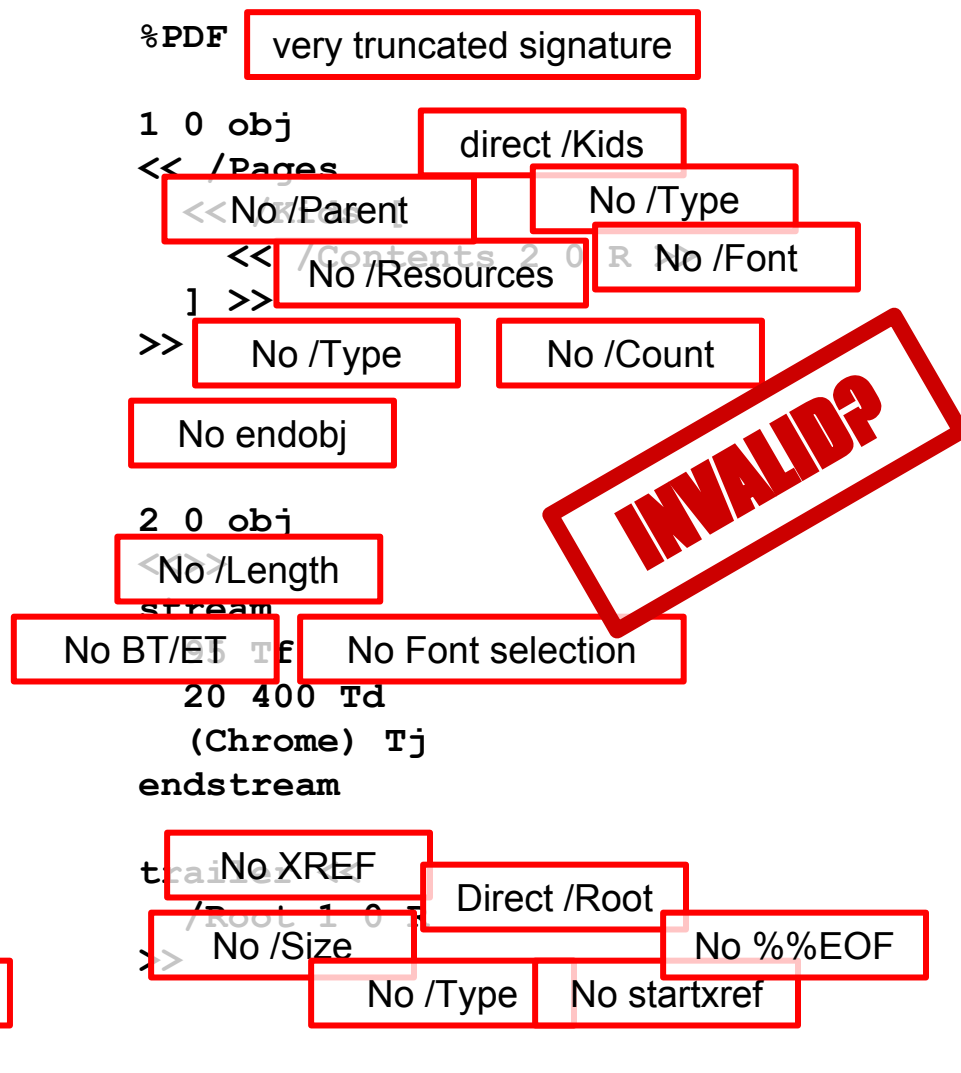
```
1 0 obj
<<
  /Pages 2 0 R
>>
endobj

2 0 obj
<<
  /Type /Pages
  /Count 1
  /Kids [3 0 R]
>>
endobj

3 0 obj
<<
  /Type /Page
  /Contents 4 0 R
  /Parent 2 0 R
  /Resources <<
    /Font <<
      /F1 <<
        /Type /Font
        /Subtype /Type1
        /BaseFont /Arial
      >>
    >>
  >>
>>
endobj

4 0 obj
<< /Length 50 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!)Tj
ET
endstream
endobj
```

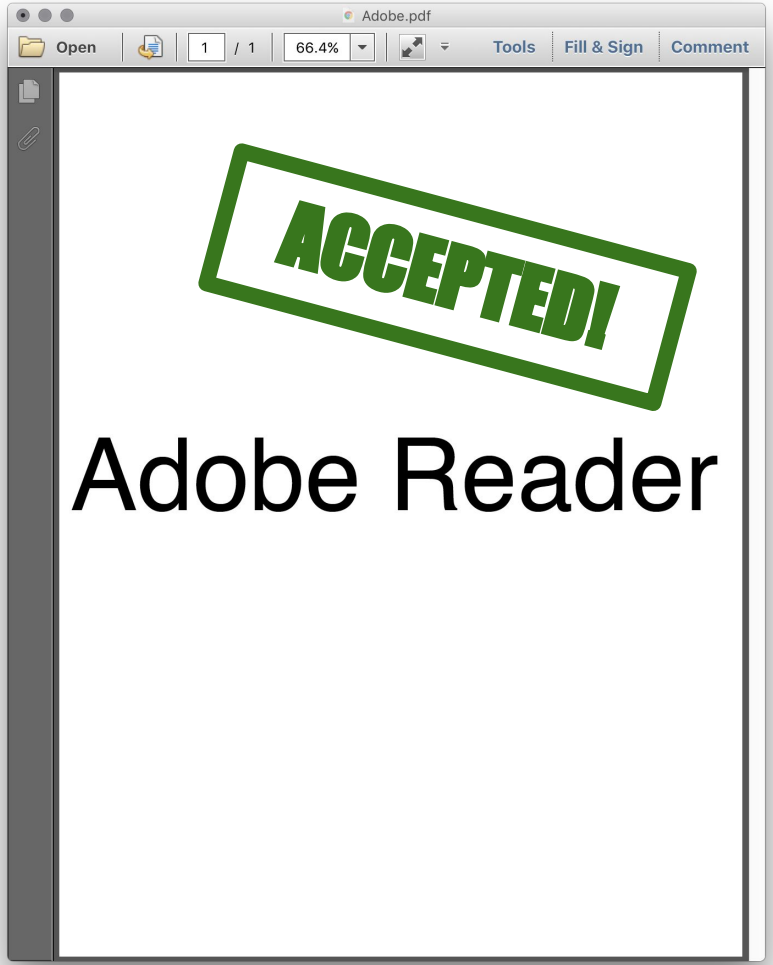
BEGIN TEXT
FONT F1 (ARIAL) SET TO SIZE 110
MOVE TO COORDINATE 10, 400
OUTPUT TEXT "HELLO WORLD!"
END TEXT

```
%PDF-1.

1 0 obj
<< /Kids [<<
    /Parent 1 0 R
    /Resources <<>>
    /Contents 2 0 R
  >>]
>>

2 0 obj
<<>>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Adobe Reader) Tj
ET
endstream
endobj

trailer <<
  /Root << /Pages 1 0 R >>
>>
```

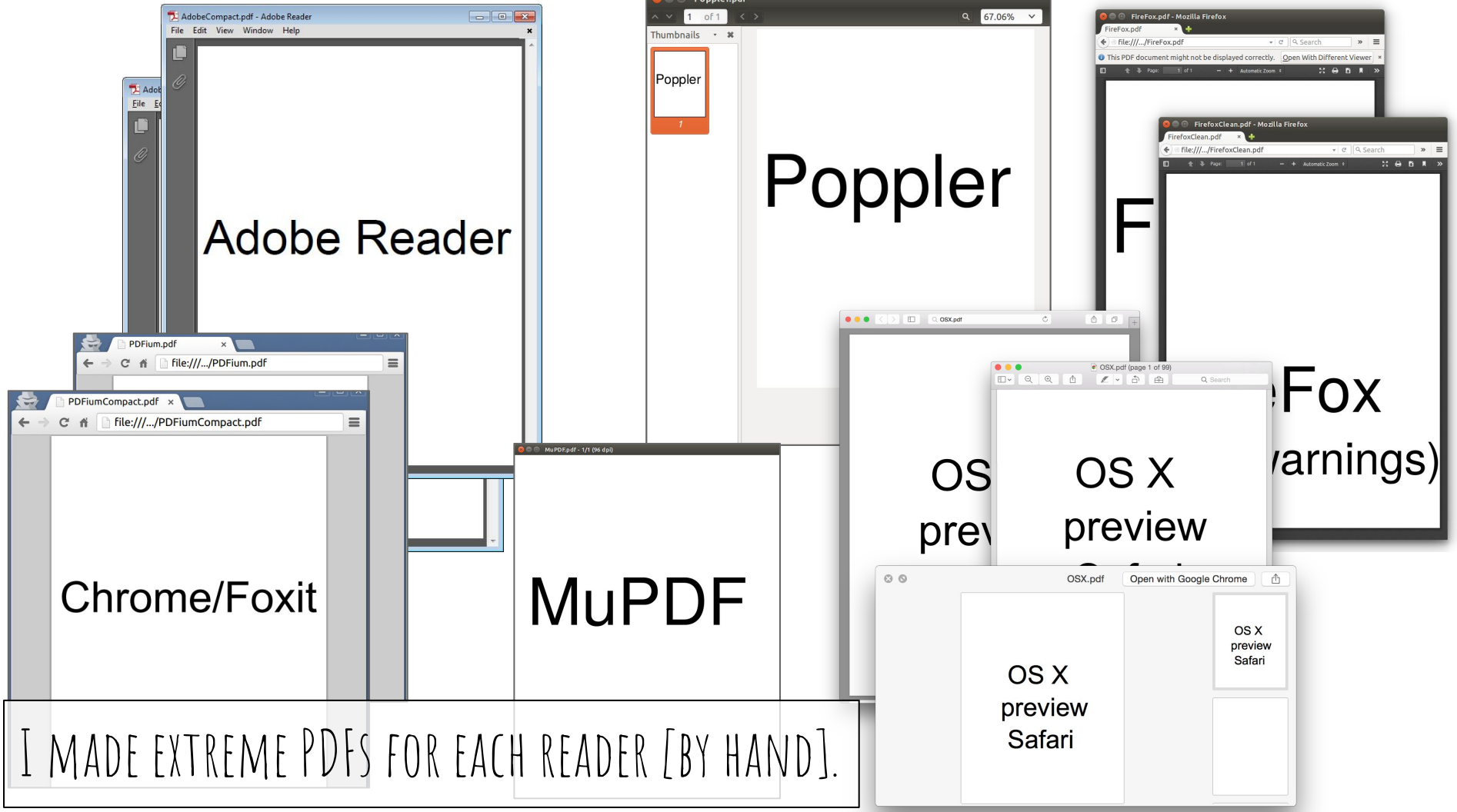
```
%PDF

1 0 obj
<< /Pages
  << /Kids [
    << /Contents 2 0 R >>
  ] >>
>>

2 0 obj
<<>>
stream
  95 Tf
  20 400 Td
  (Chrome) Tj
endstream

trailer <<
  /Root 1 0 R
>>
```

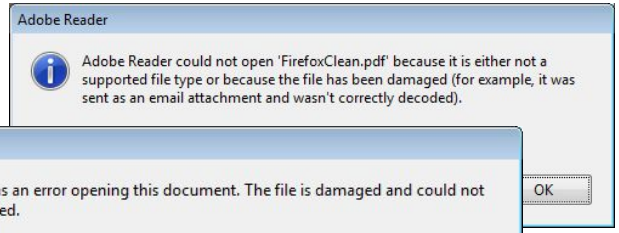

%PDF-1.
truncated signature
1 0 obj
<< /Kids [
direct /Kids
/Parent 1 0 R
No /Type
/Resources <<
No /Font
/Contents 2 0 R
>>]
No /Type
No /Count
>>
2 0 obj
<<
No /Length
>>
stream
BT
/F1 110 Tf
10 400 Td
(Adobe Read
Tj
ET
endstream
endobj
INVALID?
No XREF
trailer <<
Direct /Root
No /Size
/Root 1 0 R
/Pages 1 0
No %%EOF
>>
No /Type
No startxref

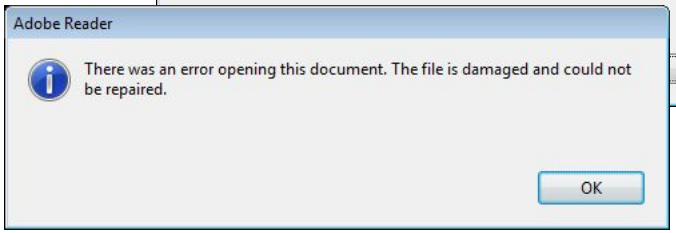

%PDF
very truncated signature
1 0 obj
<< /Pages
direct /Kids
<< /Kids [
No /Parent
No /Type
<< /Contents 2 0 R
No /Resources
No /Font
] >>
>>
No /Type
No /Count
No endobj
2 0 obj
<<
No /Length
>>
stream
No BT/ET
/F1 110 Tf
No Font selection
20 400 Td
(Chrome) Tj
endstream
INVALID?
trailer <<
No XREF
/Root 1 0 R
Direct /Root
>>
No /Size
No %%EOF
No /Type
No startxref

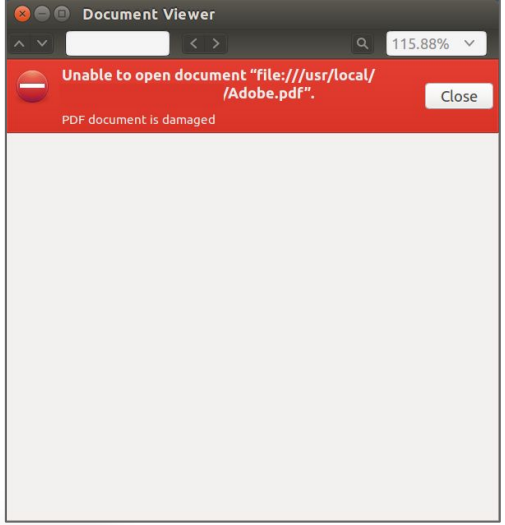

Adobe.pdf
Open
1
/ 1
66.4%
Tools
Fill & Sign
Comment
ACCEPTED!
Adobe Reader

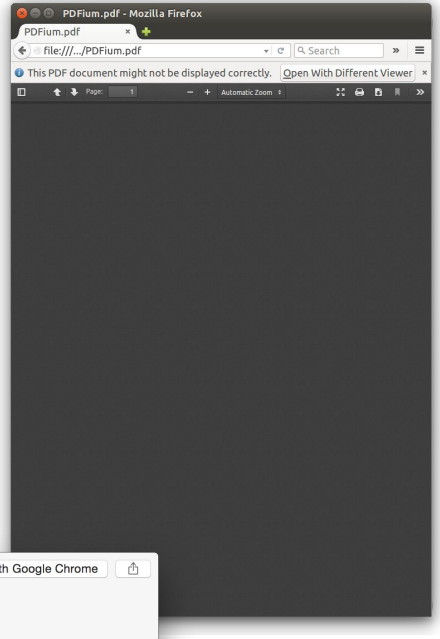

chrome.pdf
file:///Users/corkami/Downloads/chrome.pdf
ACCEPTED!
Chrome

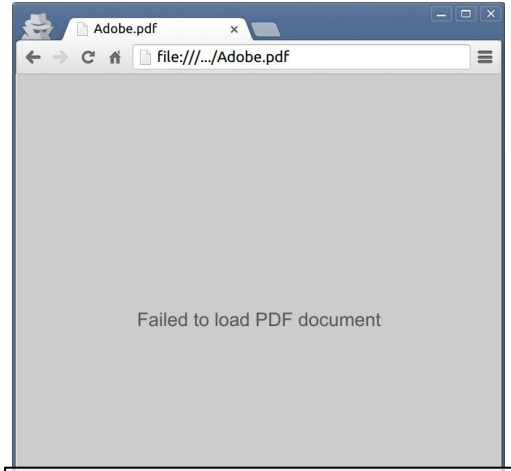

AdobeCompact.pdf - Adobe Reader
File Edit View Window Help
Adobe Reader
Poppler.pdf
1 of 1
67.06%
Thumbnails
Poppler
Poppler
FireFox.pdf - Mozilla Firefox
FireFox.pdf
file:///.../FireFox.pdf
This PDF document might not be displayed correctly. Open With Different Viewer
FireFoxClean.pdf - Mozilla Firefox
FireFoxClean.pdf
file:///.../FireFoxClean.pdf
F
Fox
varnings)
PDFium.pdf
file:///.../PDFium.pdf
PDFiumCompact.pdf
file:///.../PDFiumCompact.pdf
Chrome/Foxit
MuPDF.pdf - 1/1 (96 dpi)
MuPDF
OSX.pdf
OS
prev
OSX.pdf (page 1 of 99)
OS X
preview
OSX.pdf
Open with Google Chrome
OS X
preview
Safari
OS X
preview
Safari
I MADE EXTREME PDFS FOR EACH READER [BY HAND].

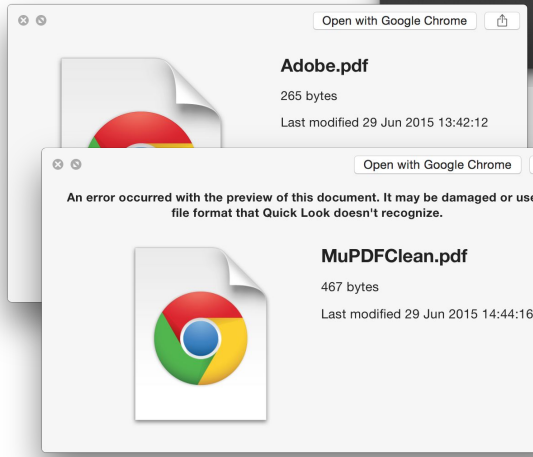






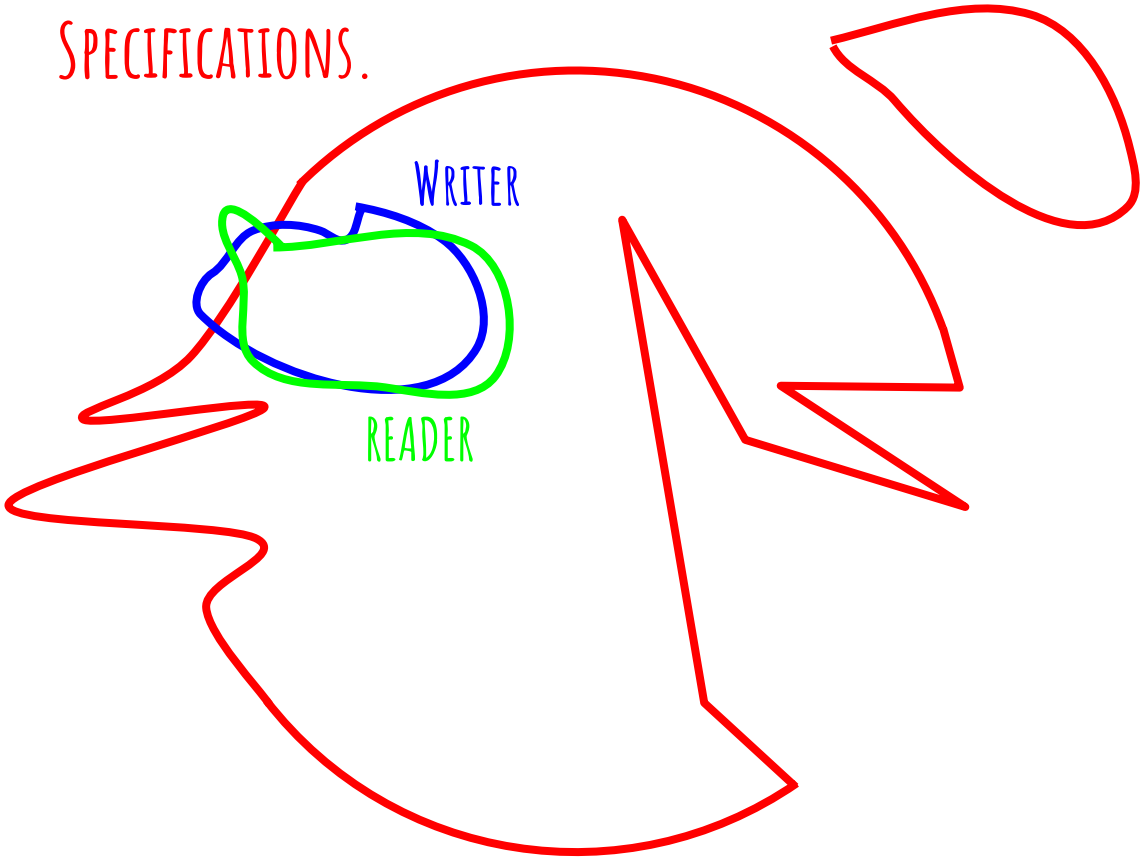






THESE EXTREME PDFS FAIL ON ANY OTHER READER.

Specifications.



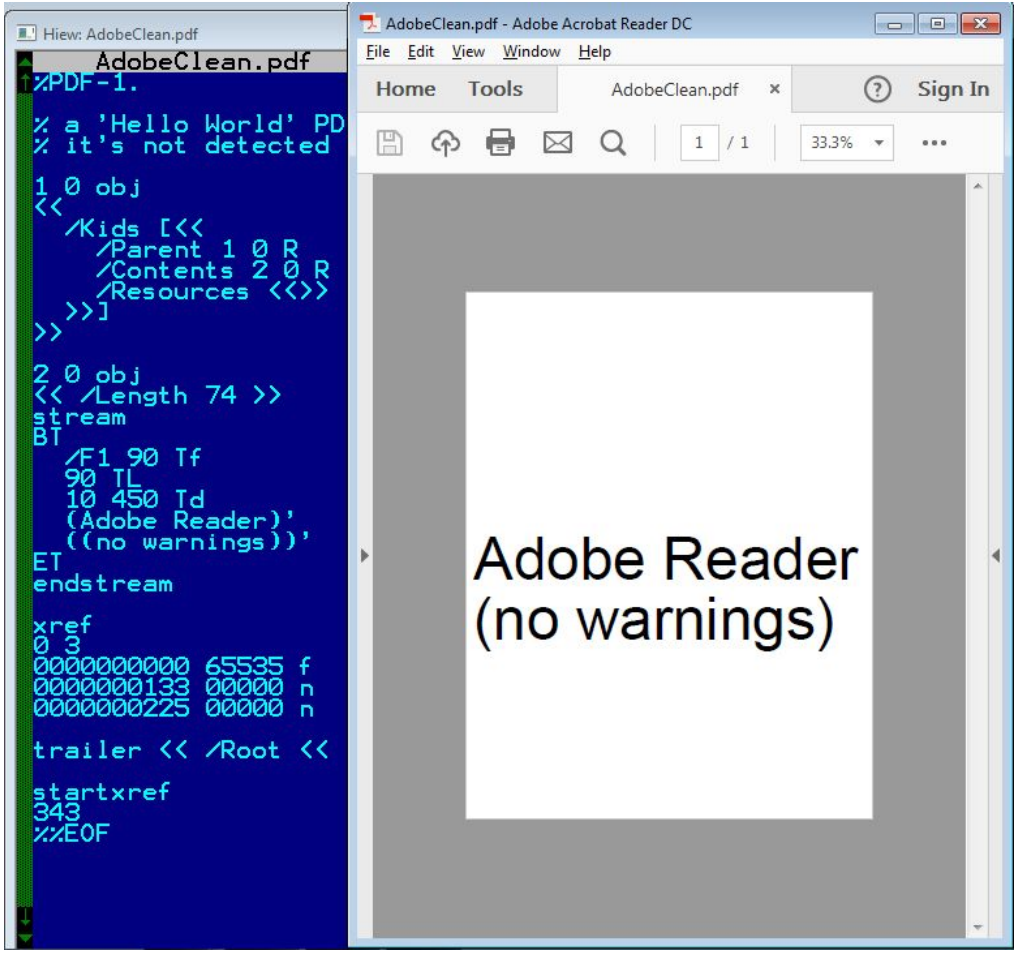
If one of these software becomes standard, the other software will have to adapt to it.

RECOVERY

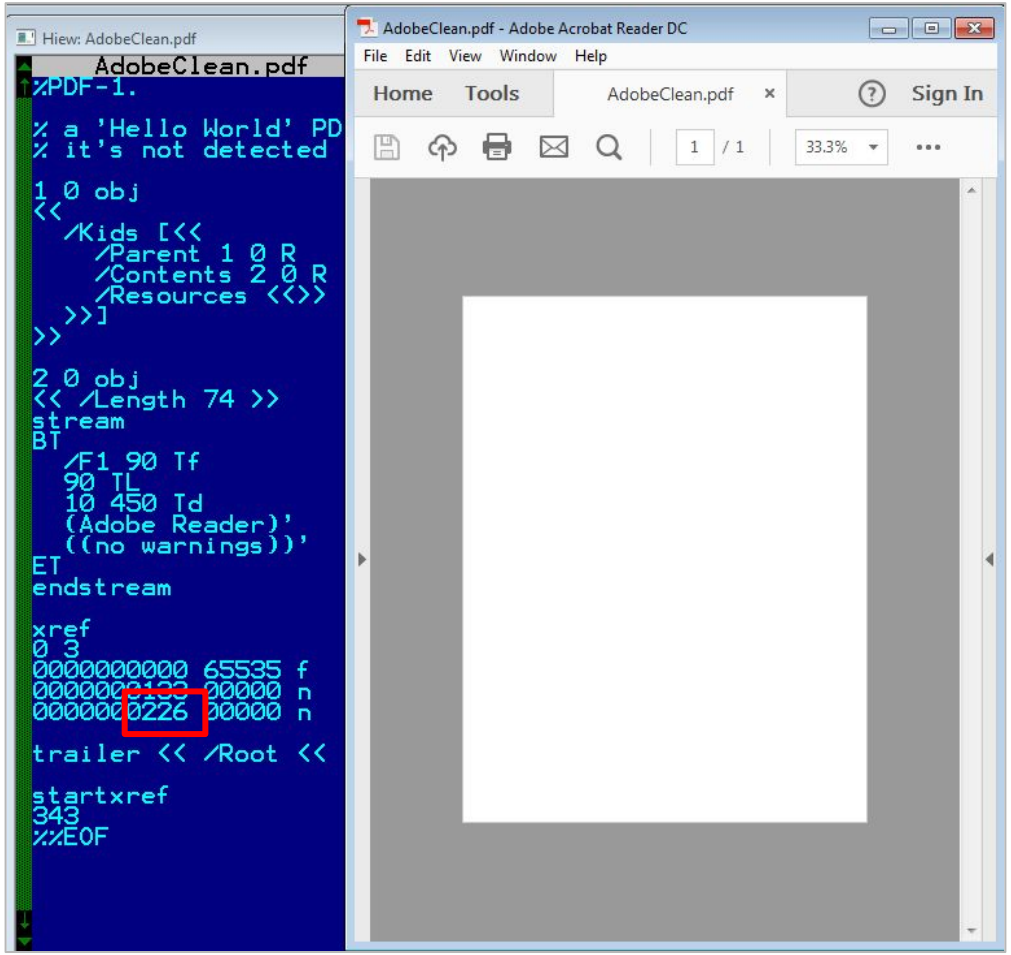
Since there's no official 'direction',
other softwares may have
to be taken into consideration.

Take a standard PDF.

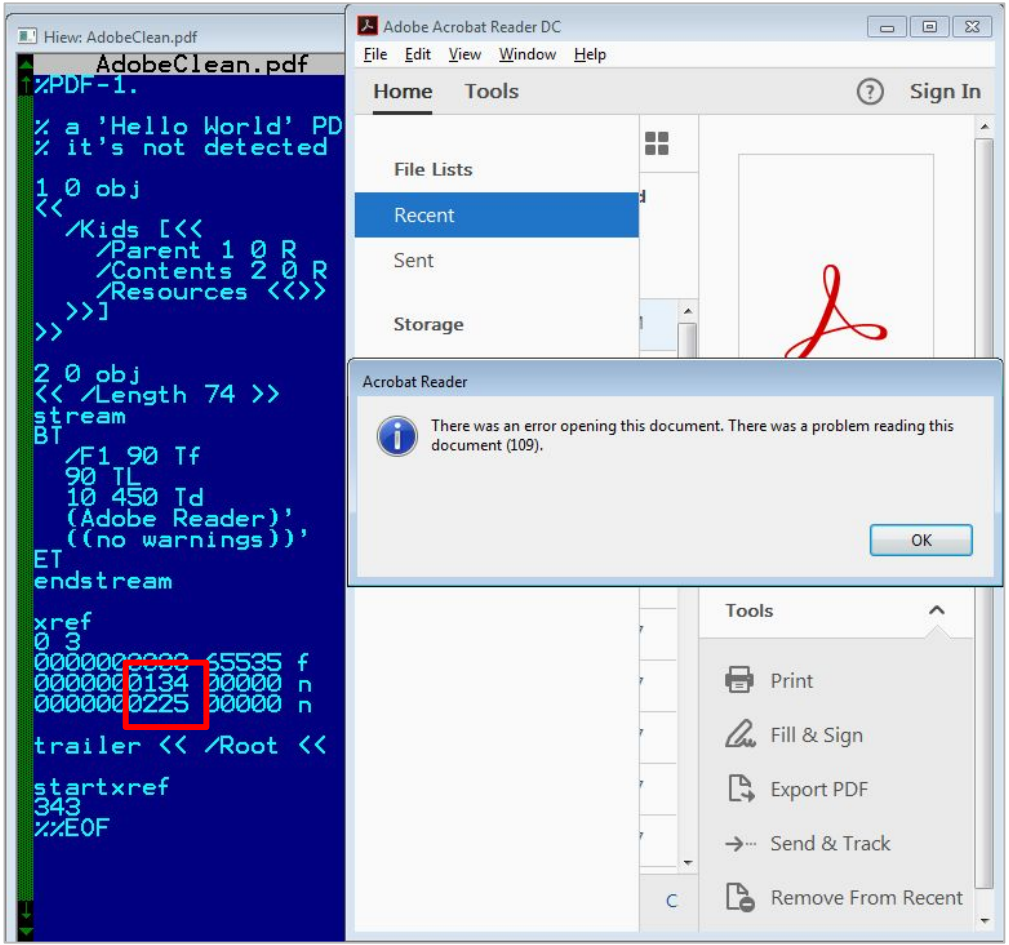
(it opens in Adobe with **NO** warnings)



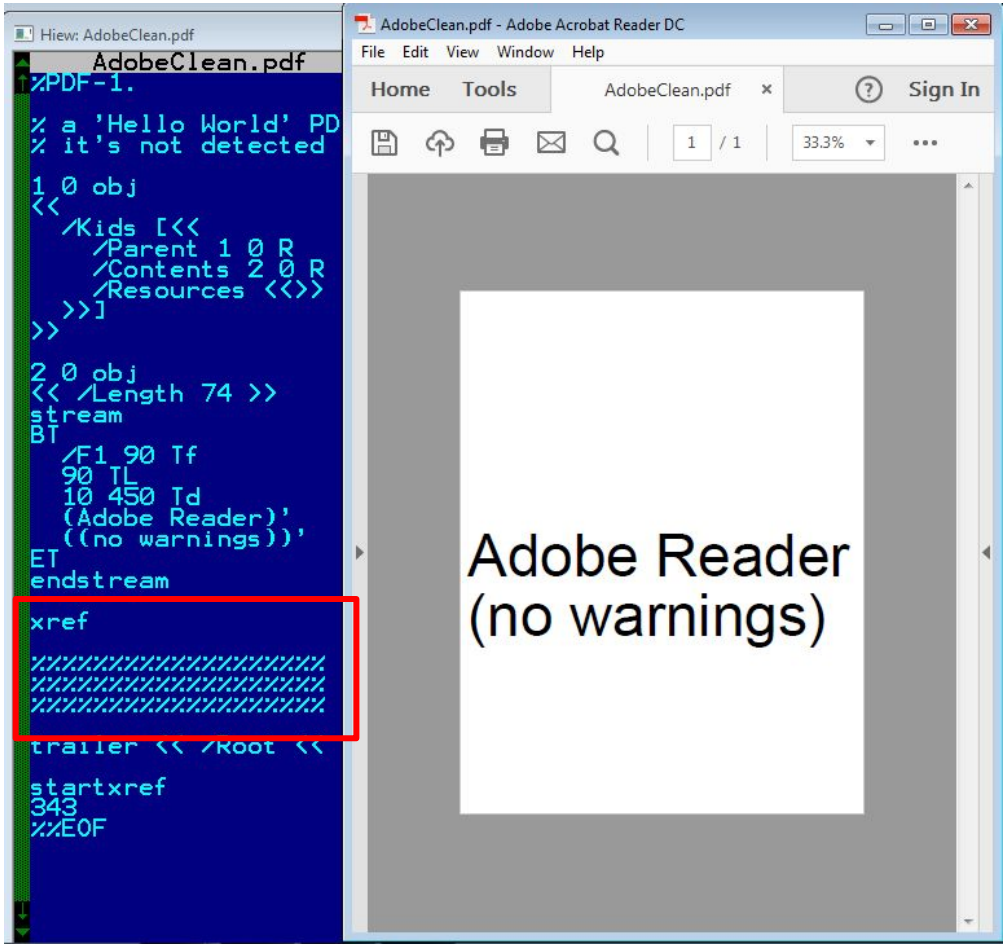
If you modify
its xref table,
it won't work correctly...



...OR MAYBE EVEN NOT OPEN AT ALL!



But if you **erase** the xref table, Adobe will fall back to **recovery** mode, and open the file without **any** warning!

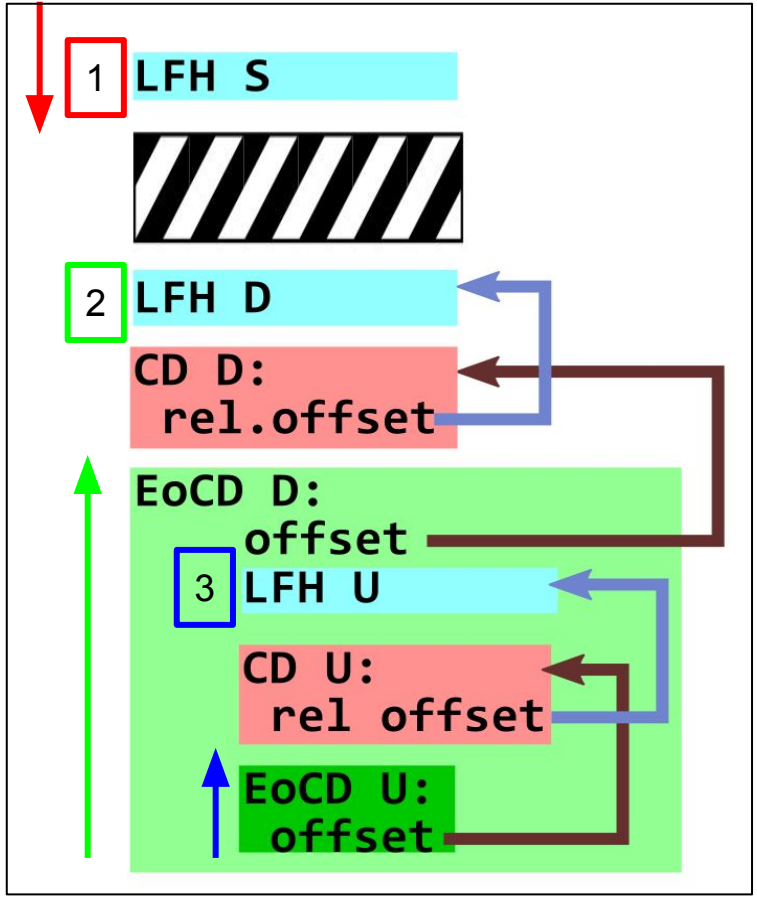


Schizophrenia

Different contents (clean & malicious) can be combined
In the same file, to bypass security or fool softwares.

Zip archives

3 different softwares will see
3 different archives
from the same file



This enabled a **critical** vulnerability
in all Android devices in 2013:
validate a content, execute a different one!

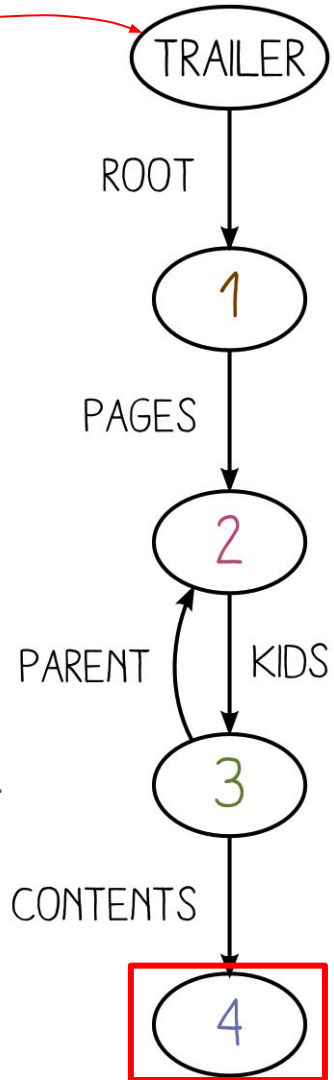
PDF

The trailer defines the start of the document tree.

See a different trailer -> see a completely different document!

Several trailers can co-exist in the same file, and

Parsers tolerate 'unused' objects (referenced by unseen trailers).



commented line - seen by PDFium

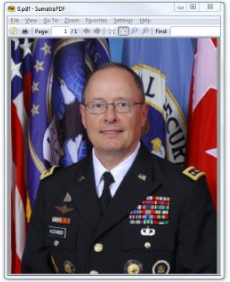
```
% trailer <</Root ...>>
```

Standard trailer - the only one seen by Adobe Reader

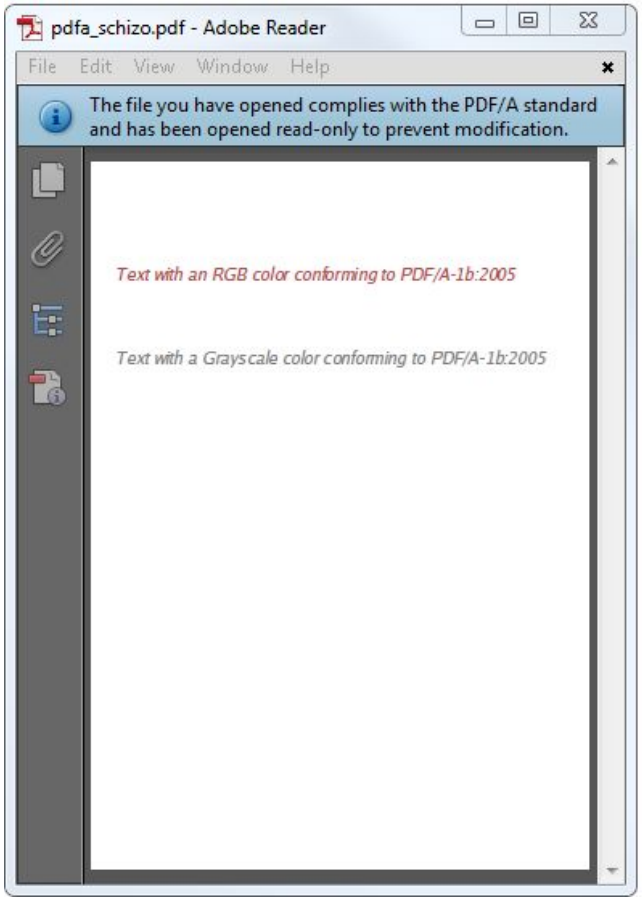
```
trailer <</Root ...>>
```

missing trailer keyword, but seen by Poppler

```
        <</Root ...>>
```

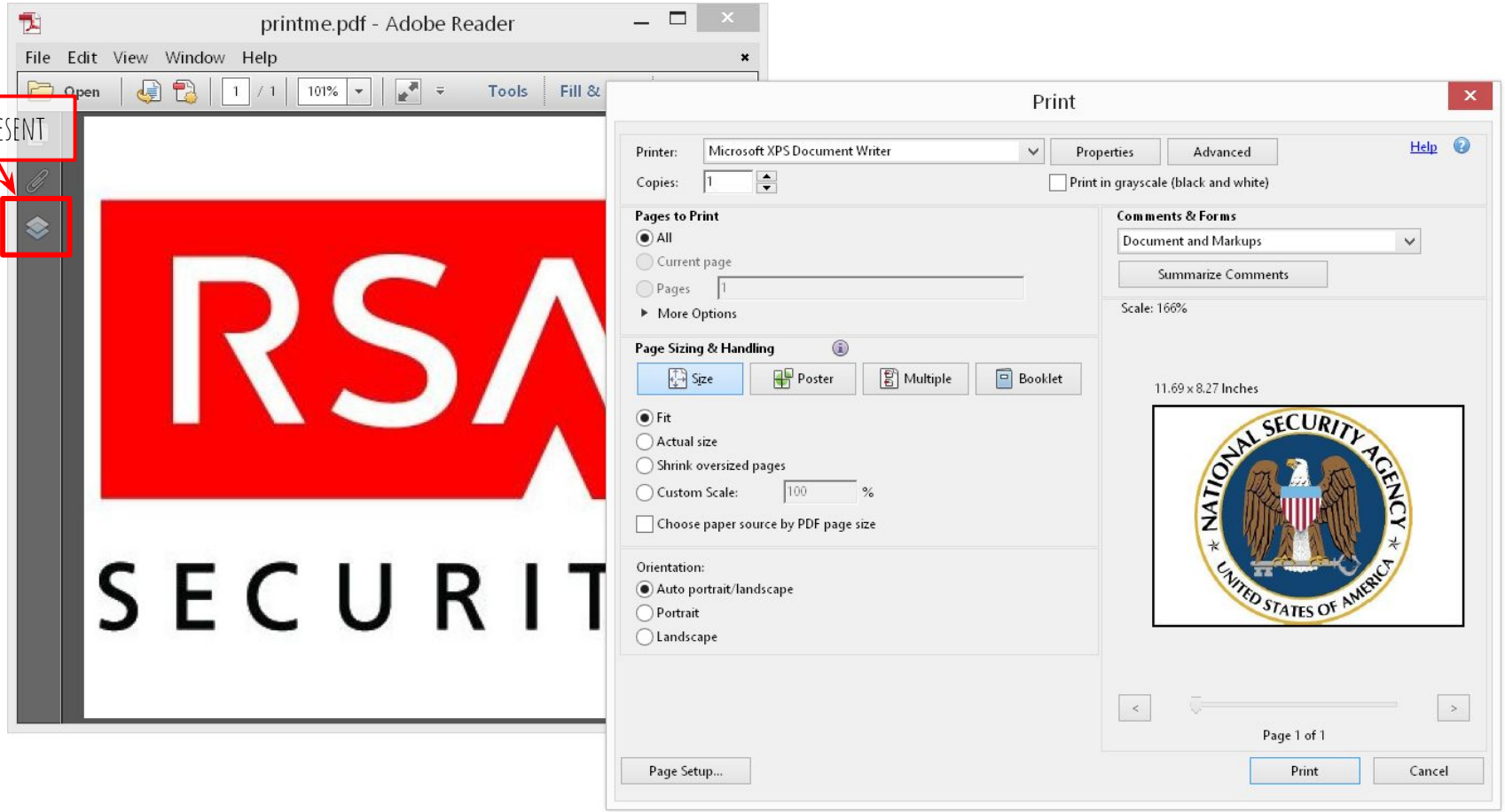


3 different documents as seen by 3 different readers in the same file (such things even happen accidentally in the wild!)



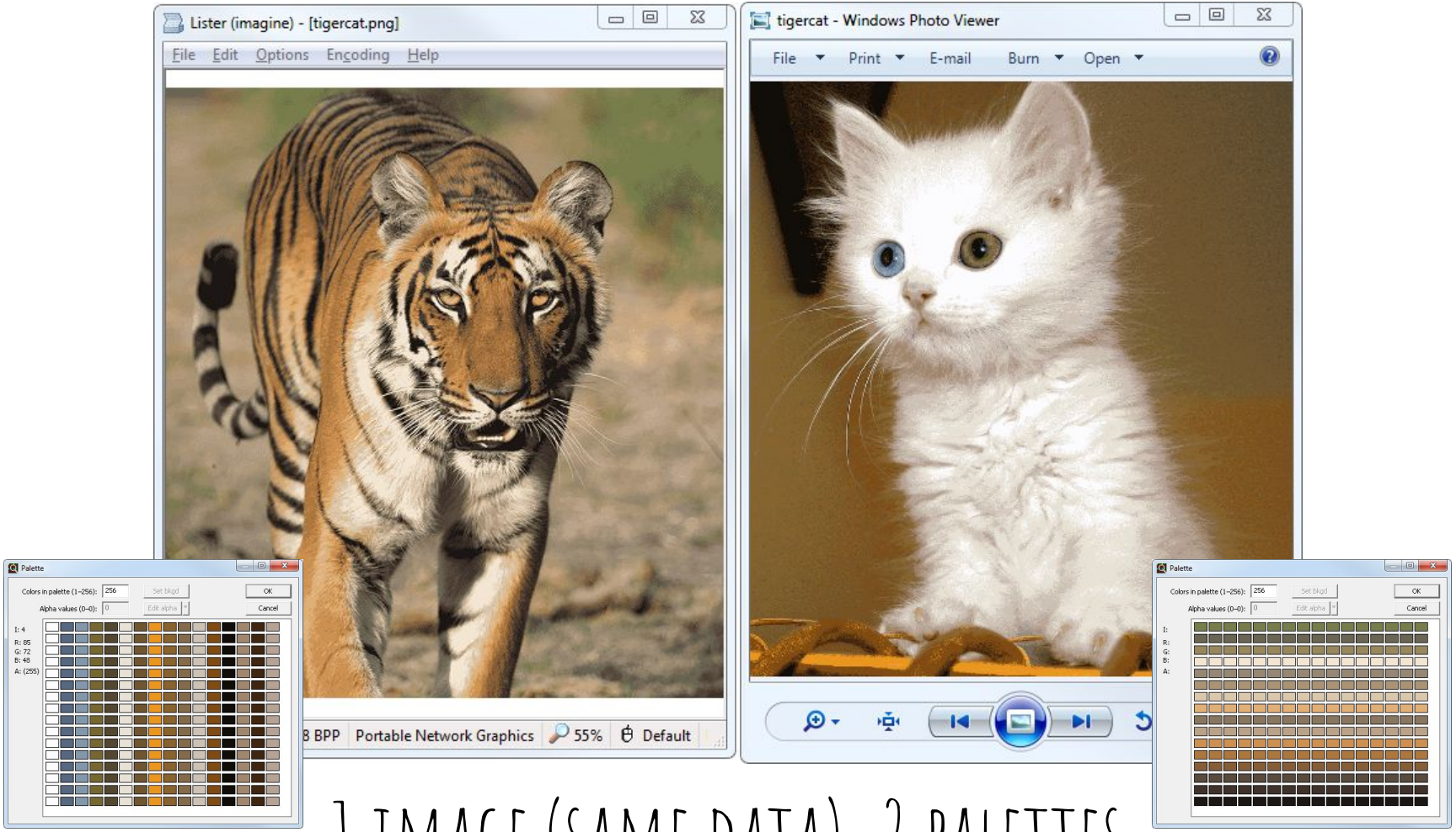
It used to work with PDF/A too

(OK for Adobe Reader, but not for Preflight)



What you see is not always what you print - when you use Layers [Optional Content Groups]!

Fun fact: you **can't** change the printing output with Adobe Reader ;)



1 IMAGE (SAME DATA), 2 PALETTES

Is this just an infinite vicious circle?

Do we need a great fire destroying our knowledge so that we build file formats in a smarter way?

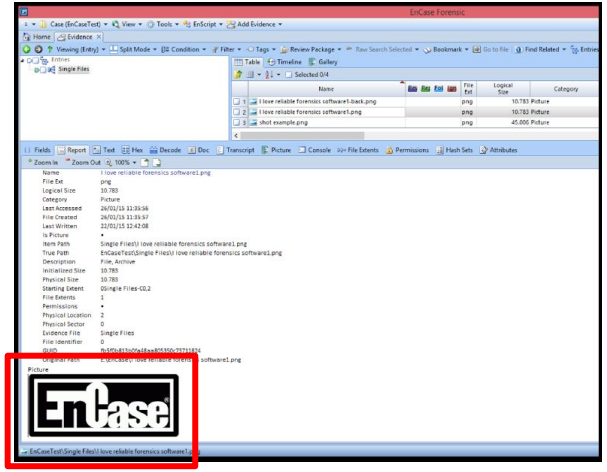
Or ultimately, we'll forget our roots and just maintain stacks of emulation layers...?

Law enforcements

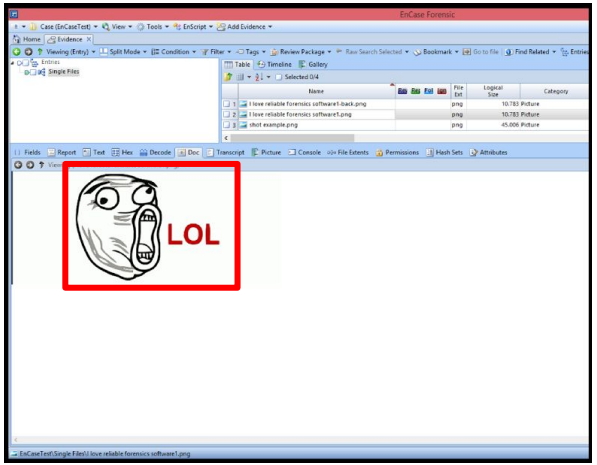
Like archivists, investigators rely on softwares to determine if someone is guilty or not.

These softwares rely on the same blurry specs...

They're vulnerable to the same problems.



Same file, two different tabs



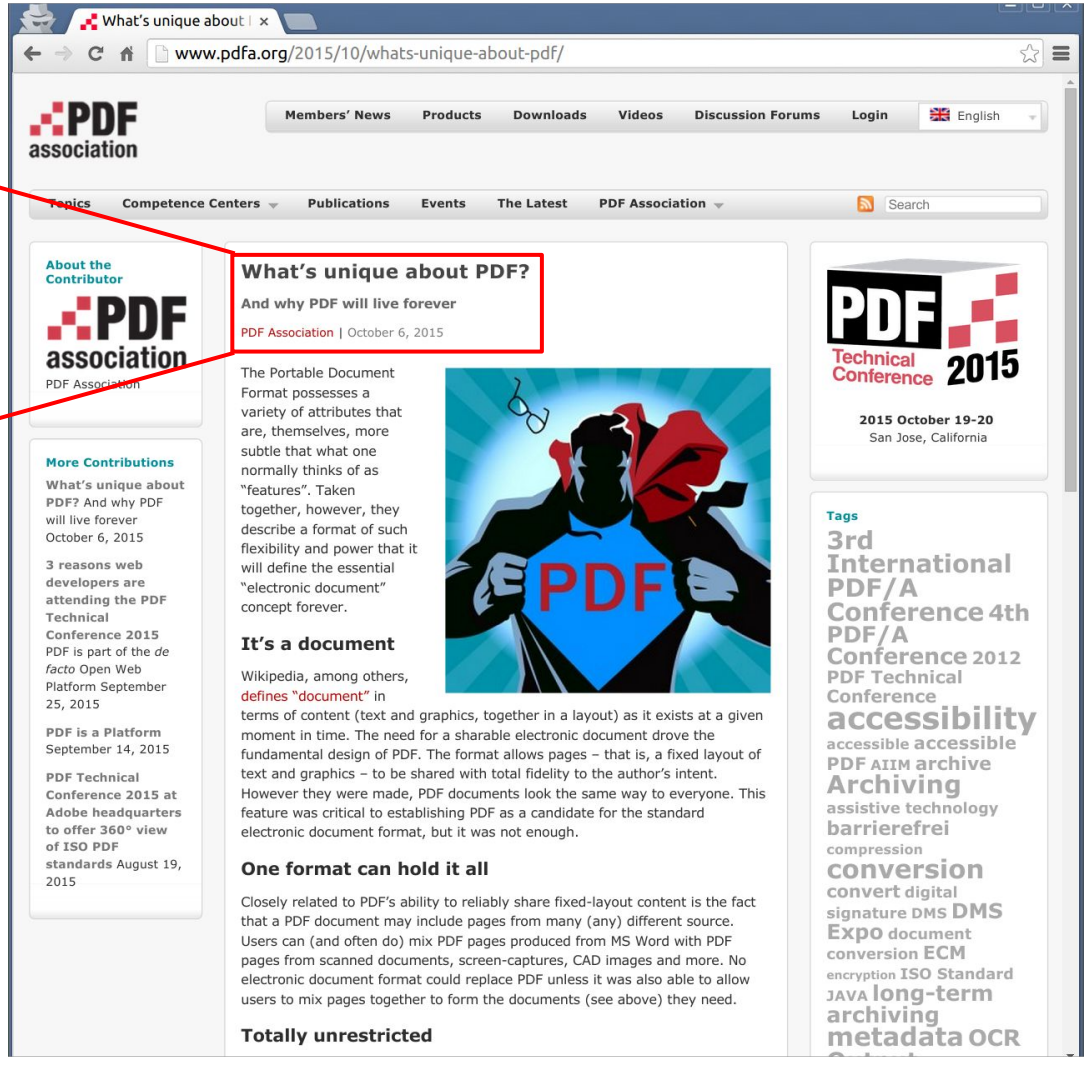
What's unique about PDF?

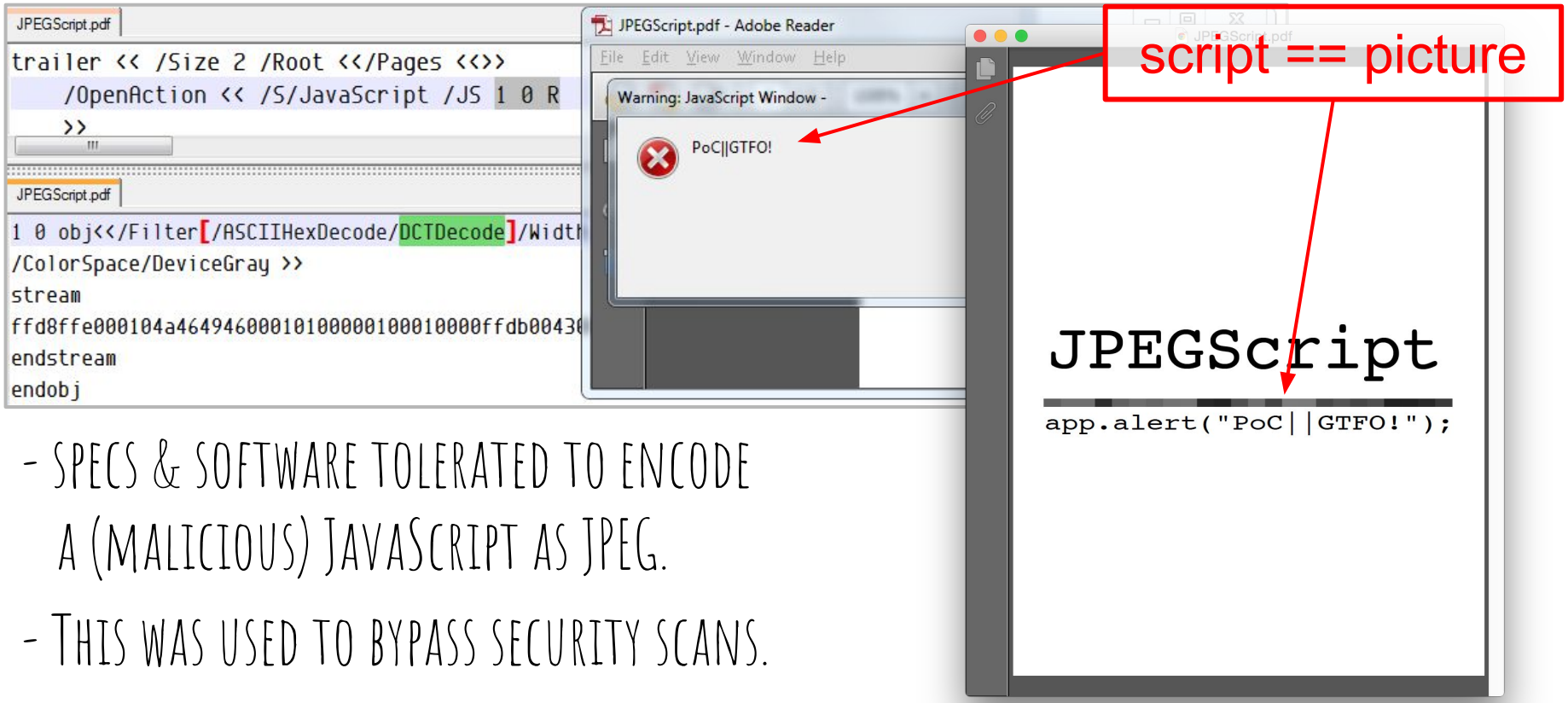
And why PDF will live forever

PDF Association | October 6, 2015

http://www.pdfa.org/2015/10/whats-unique-about-pdf/

- Flash is now dying, for security reasons.
- Adobe is out of the game for PDF.
- Looking at PDF 2.0, I'm very skeptical...
(many extra security risks)





- specs & software tolerated to encode a (malicious) JavaScript as JPEG.
- This was used to bypass security scans.
- This 'feature' was killed to improve security

-> specifications were never updated -> they're now **clearly** outdated.

PDF is great, but...

No major actor (Adobe) behind it anymore?

PDF 2.0 is too different?

Too permissive security-wise?

(Don't get me wrong, I really like to [ab]use PDF)

My point of view:

need to create better corpuses

To demonstrate the current state of things.

To determine the actual limits of file formats.

To make automation easier for everyone.

Atomic - Copyright-free - PII-free

Preservation

Content, or the files? And private data?

Preserving usually means "saving to a safer format"

This means we gave up on the original format...

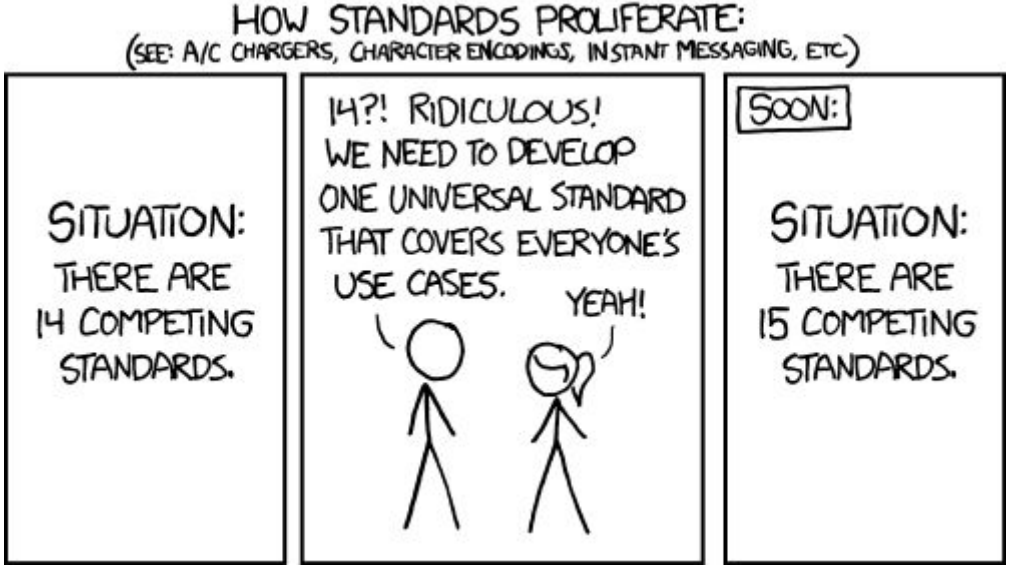
But which one is safe?

We need to preserve interesting files as-is too.

We don't need 'safer' formats

(whatever that means)

Yet another format?
Blurry specs,
Divergences,
Unclear corner cases...



We need a new process:
File formats should be alive!

Like cryptography: Updates, deprecations...
-> eventually uniform standardisation

With a date, a version number, a commit.
Up to date specifications and open validator.

Once things are properly documented, we could go back easily.

Why doesn't it happen?

Because we haven't proved **enough** yet how **broken** they are?

In cryptography, there are official competitions
to break the drafts before choosing the standard:
survival of the fittest **before** setting **standards** in stone.

Conclusion

- Files formats are awesome (Even PDF!)
  - Except when they don't work as expected :)
- Specifications are not challenged enough.
- formats authorship is not a liability.

We need to develop our expertises
and share our knowledge.

FEEDBACK?

THANK YOU FOR READING THAT FAR!

ANGE ALBERTINI
reverse engineering
VISUAL DOCUMENTATIONS
@angealbertini
ange@corkami.com
http://www.corkami.com

Extra resources

Funky files formats:

https://speakerdeck.com/ange/funky-file-formats-31c3

Schizophrenic files:

https://speakerdeck.com/ange/schizophrenic-files-v2

Abusing file formats:

https://archive.org/stream/pocorgtfo07#page/n17/mode/2up