Messing with binary formats

ΠΟΛΎΓΛΩΣΣΟΣ

44CON

Ange Albertini
2013/09/13

London, England

Welcome!

- this is the non-live version of my slides
  - more text
  - standard PDF file ;)

About me:

- Reverse engineer
- my website: http://corkami.com
  - reverse engineering &
    visual documentations

I just like to play with lego blocks

low-level ones, that is

block by block

generate files byte per byte

```
istruc IMAGE_DOS_HEADER
    at IMAGE_DOS_HEADER.e_magic,  db 'MZ'
    at IMAGE_DOS_HEADER.e_lfanew, dd NT_Signature - IMAGEBASE
iend
NT_Signature:
istruc IMAGE_NT_HEADERS
    at IMAGE_NT_HEADERS.Signature, db 'PE', 0, 0
iend
istruc IMAGE_FILE_HEADER
    at IMAGE_FILE_HEADER.Machine,         dw IMAGE_FILE_MACHINE_I386
    at IMAGE_FILE_HEADER.Characteristics, dw IMAGE_FILE_EXECUTABLE_IMAGE
iend
istruc IMAGE_OPTIONAL_HEADER32
```

Goals

- explore the format
- make sure that's how things work
- full control over the structure

a *complete* executable

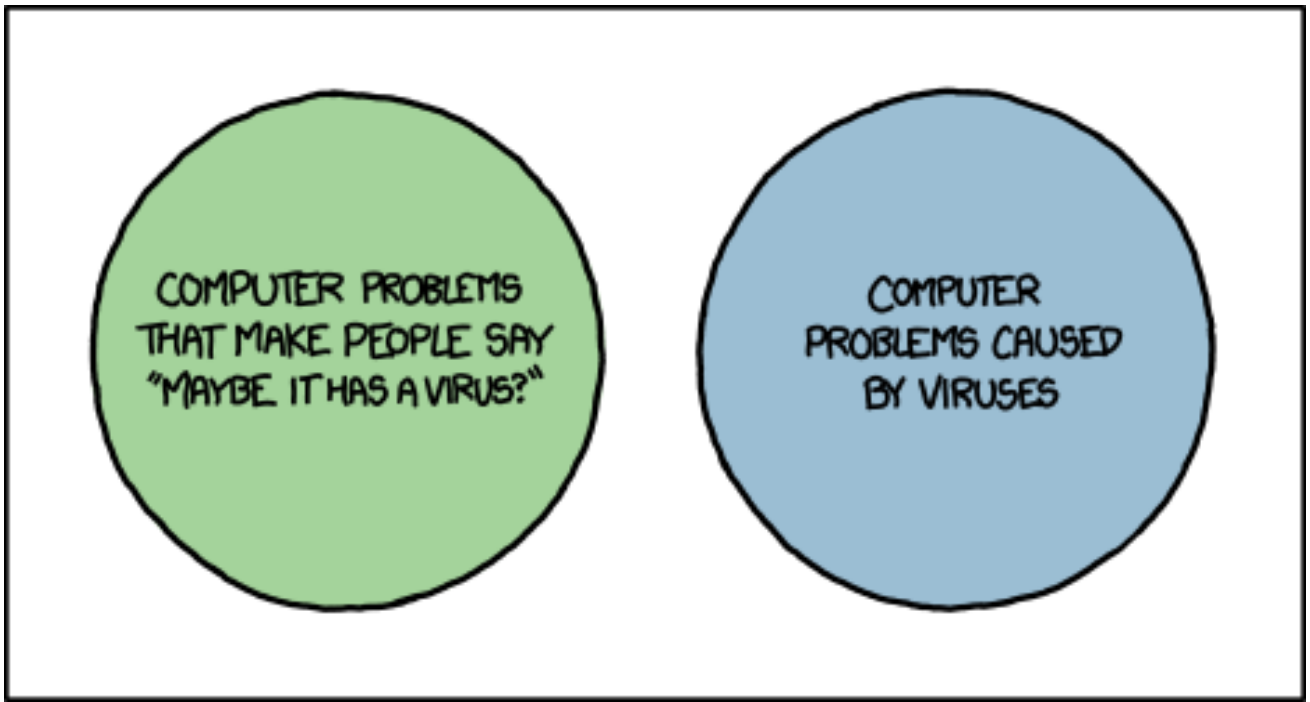

```
IMAGEBASE equ 400000h
org IMAGEBASE

istruc IMAGE_DOS_HEADER
    at IMAGE_DOS_HEADER.e_magic,  db 'MZ'
    at IMAGE_DOS_HEADER.e_lfanew, dd NT_Signature - IMAGEBASE
iend
NT_Signature:
istruc IMAGE_NT_HEADERS
    at IMAGE_NT_HEADERS.Signature, db 'PE', 0, 0
iend
istruc IMAGE_FILE_HEADER
    at IMAGE_FILE_HEADER.Machine,         dw IMAGE_FILE_MACHINE_I386
    at IMAGE_FILE_HEADER.Characteristics, dw IMAGE_FILE_EXECUTABLE_IMAGE
iend
istruc IMAGE_OPTIONAL_HEADER32
    at IMAGE_OPTIONAL_HEADER32.Magic,                  dw IMAGE_NT_OPTIONAL_HDR32_MAGIC
    at IMAGE_OPTIONAL_HEADER32.AddressOfEntryPoint,    dd EntryPoint - IMAGEBASE ; not strictly required
    at IMAGE_OPTIONAL_HEADER32.ImageBase,              dd IMAGEBASE ; not required under XP
    at IMAGE_OPTIONAL_HEADER32.SectionAlignment,       dd 1
    at IMAGE_OPTIONAL_HEADER32.FileAlignment,          dd 1
    at IMAGE_OPTIONAL_HEADER32.MajorSubsystemVersion, dw 4
    at IMAGE_OPTIONAL_HEADER32.SizeOfImage,            dd SIZEOFIMAGE
    at IMAGE_OPTIONAL_HEADER32.SizeOfHeaders,          dd SIZEOFIMAGE - 1 ; required for XP
    at IMAGE_OPTIONAL_HEADER32.Subsystem,              dw IMAGE_SUBSYSTEM_WINDOWS_CUI
iend
istruc IMAGE_DATA_DIRECTORY_16
iend

EntryPoint:
    push 42
    pop eax
    retn
```

result:

- a complete executable
- all bytes defined by hand

our problem

- is related to virus (malwares)
- they use many file formats
- it's critical to identify them reliably
  - and to tell whether corrupted or well-formed

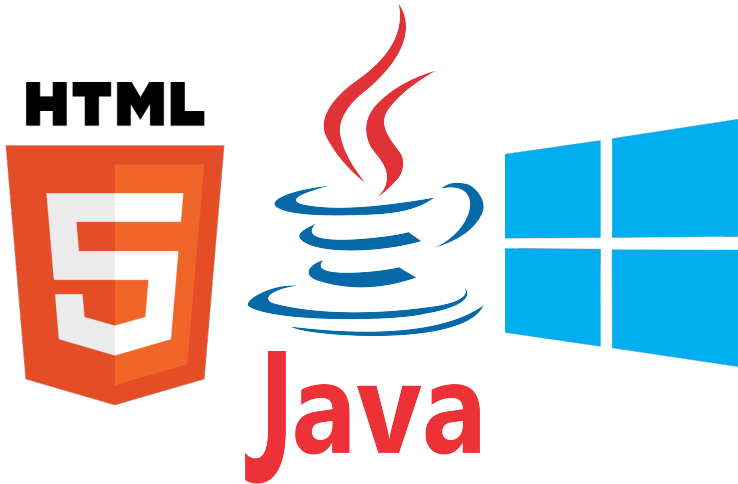


standard infection chain

the most common chain:

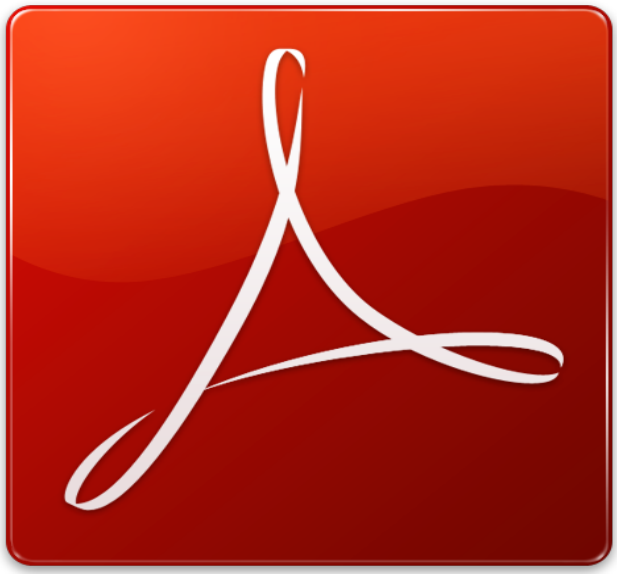
1. a web page, in HTML format
   a. launching an applet
2. an evil applet, in CLASS format
   a. exploiting a Java vulnerability
   b. dropping an executable
3. a malicious executable, in Portable Executable format

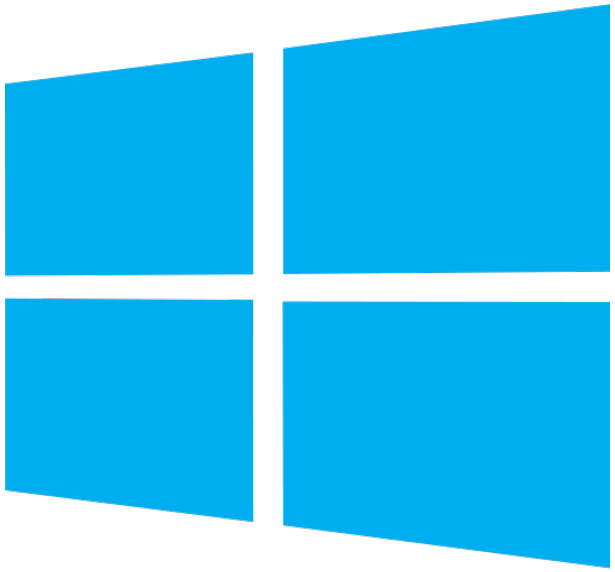
(a vast majority of malwares rely on an executable)



another classic chain

- open a PDF document
  - with an exploit inside
    - dropping or downloading a PE executable
- get a malicious executable on your machine





the challenge

it might look obvious:

- tell whether it's a PDF, a PE, a JAVA, an HTML...
- typical formats are clearly defined
  - Magic signature enforced at offset 0

reality

some formats have no header at all

- Command File (DOS 16 bits)
- Master Boot record

some formats don't need to start at offset 0

- Archives (Zips, Rars...)
- HTML
  - but text-only?

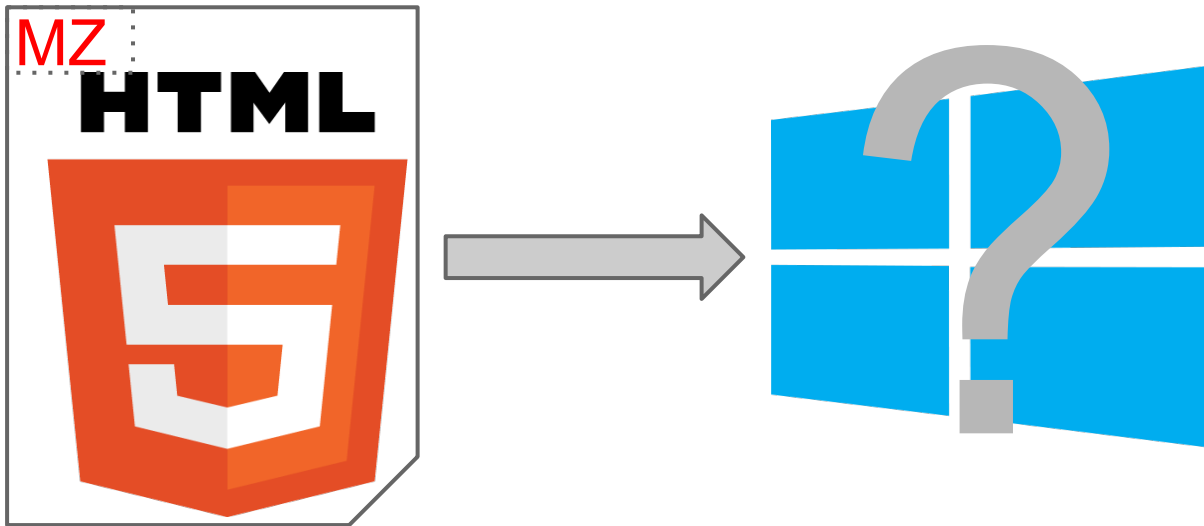
some formats accept a large controllable block early in their header

- Portable Executable
- PICT image

How did this start?

a real-life problem:

1. a (malicious) HTML page
2. started with 'MZ' (the signature of PE)
3. just scanned as a PE!
    a. wow, this PE is highly corrupted :)
    b. it must be clean :p



polyglots in the wild

+ =

```
<applet
    code=gifar.class
    archive=gifar.gif
    >
</applet>
```

GIFAR = GIF + JAR

- an uploaded image
  - an avatar in a forum
- with a malicious JAVA appended as JAR

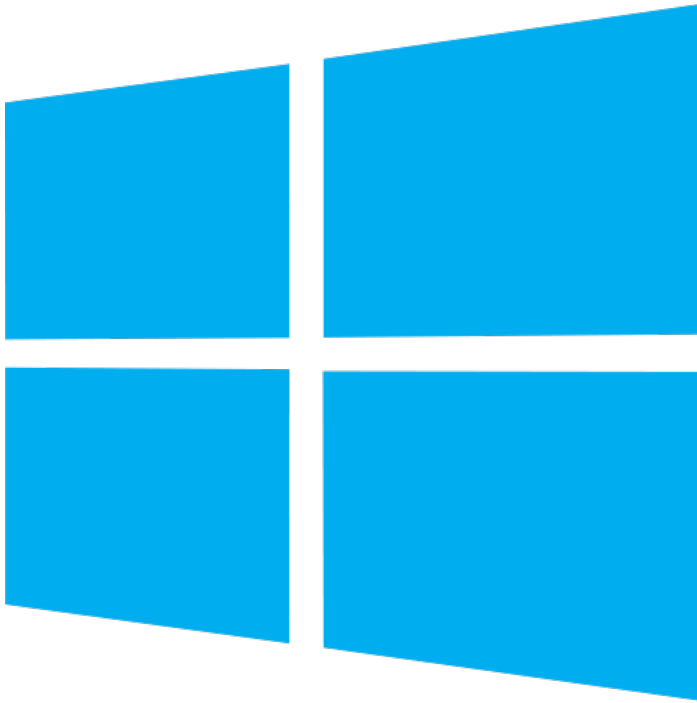
hosted on the server!

- bypass *same domain policy*
- now useable via its *JAVA=EVIL* payload

let's get started

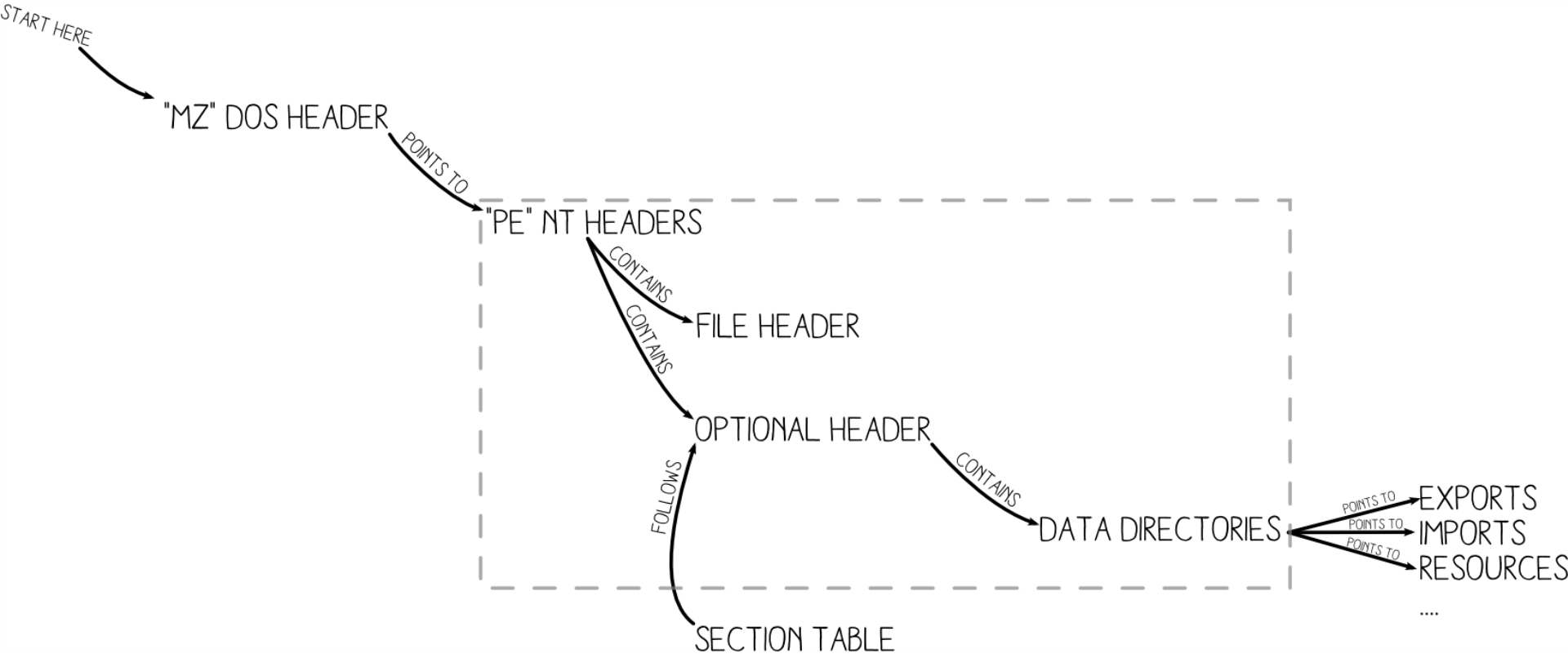
PE, the executable format of windows

- it's central to windows malware
- it enforces a magic signature at offset 0
  - game over for other formats?



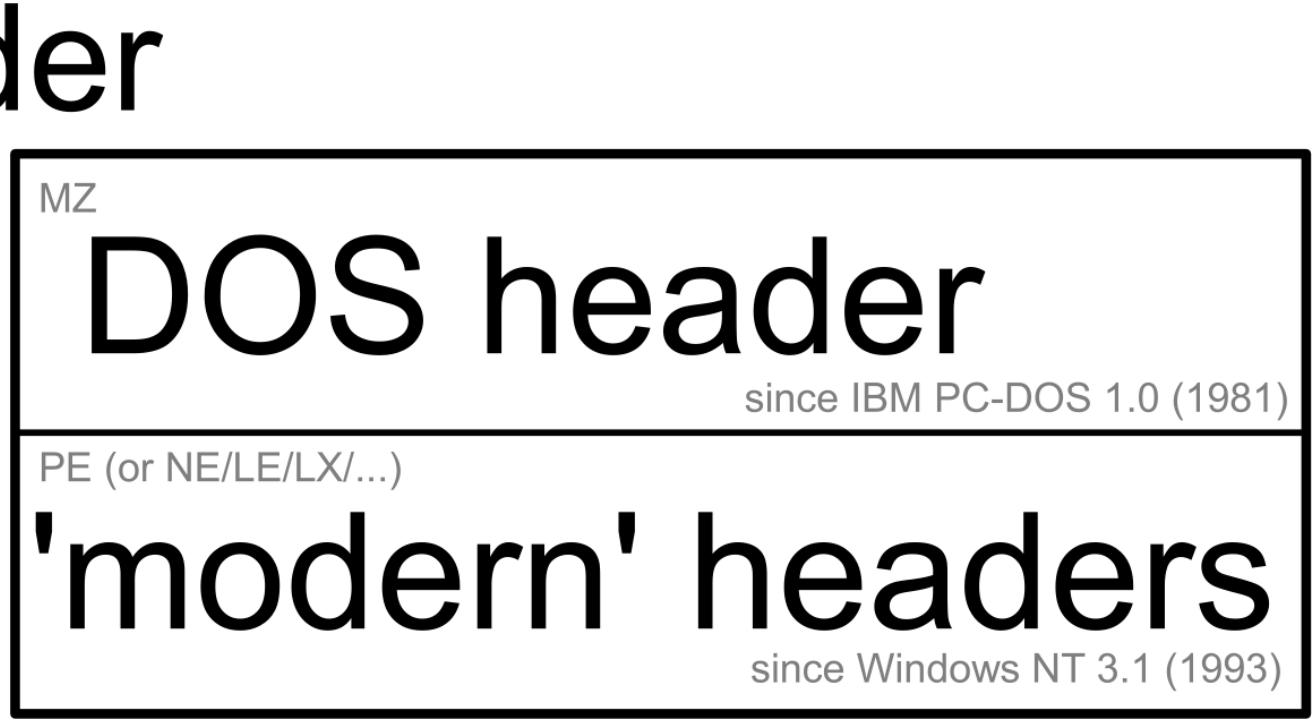
overview

- starts with a compulsory header
- made of sub-headers



a historical sandwich

1. a deprecated but required header
2. a modern header

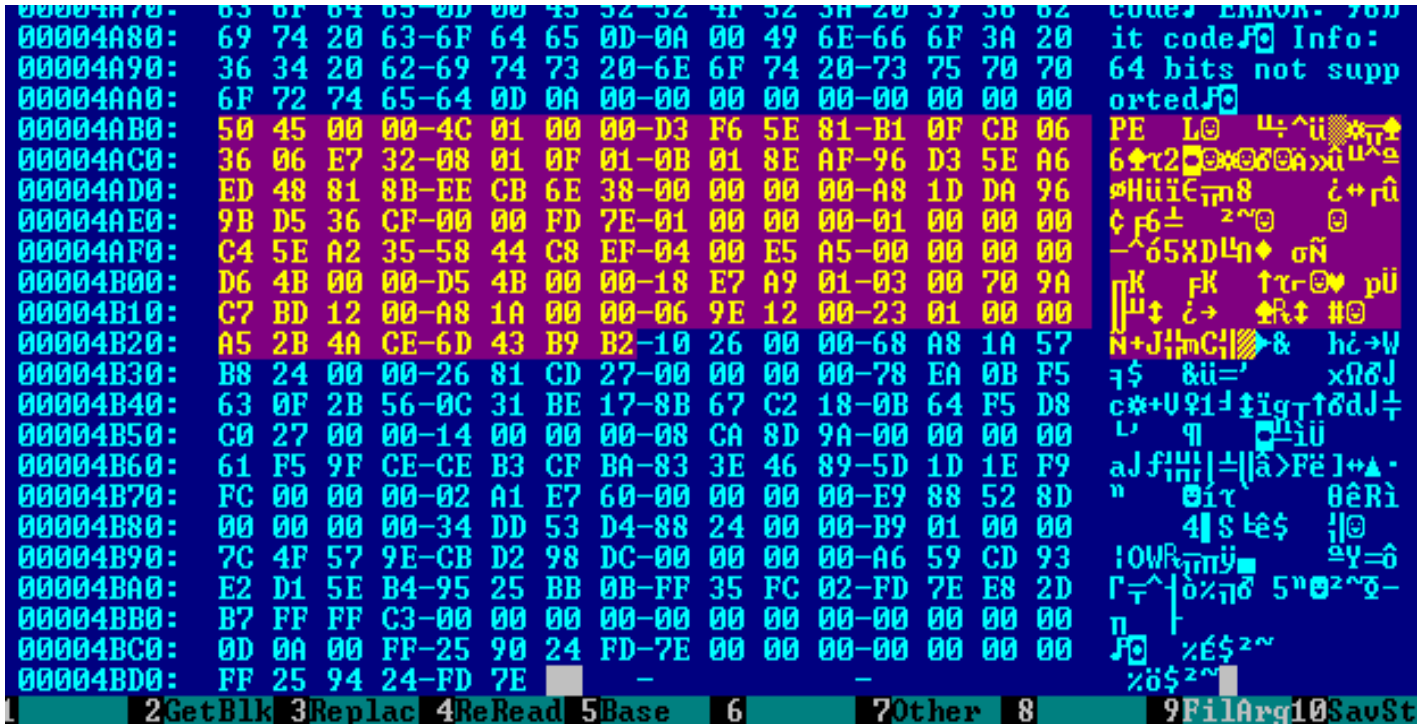


old header content

- almost completely ignored
- only required:
  - 2 byte signature
  - pointer to new header

the new header can be anywhere

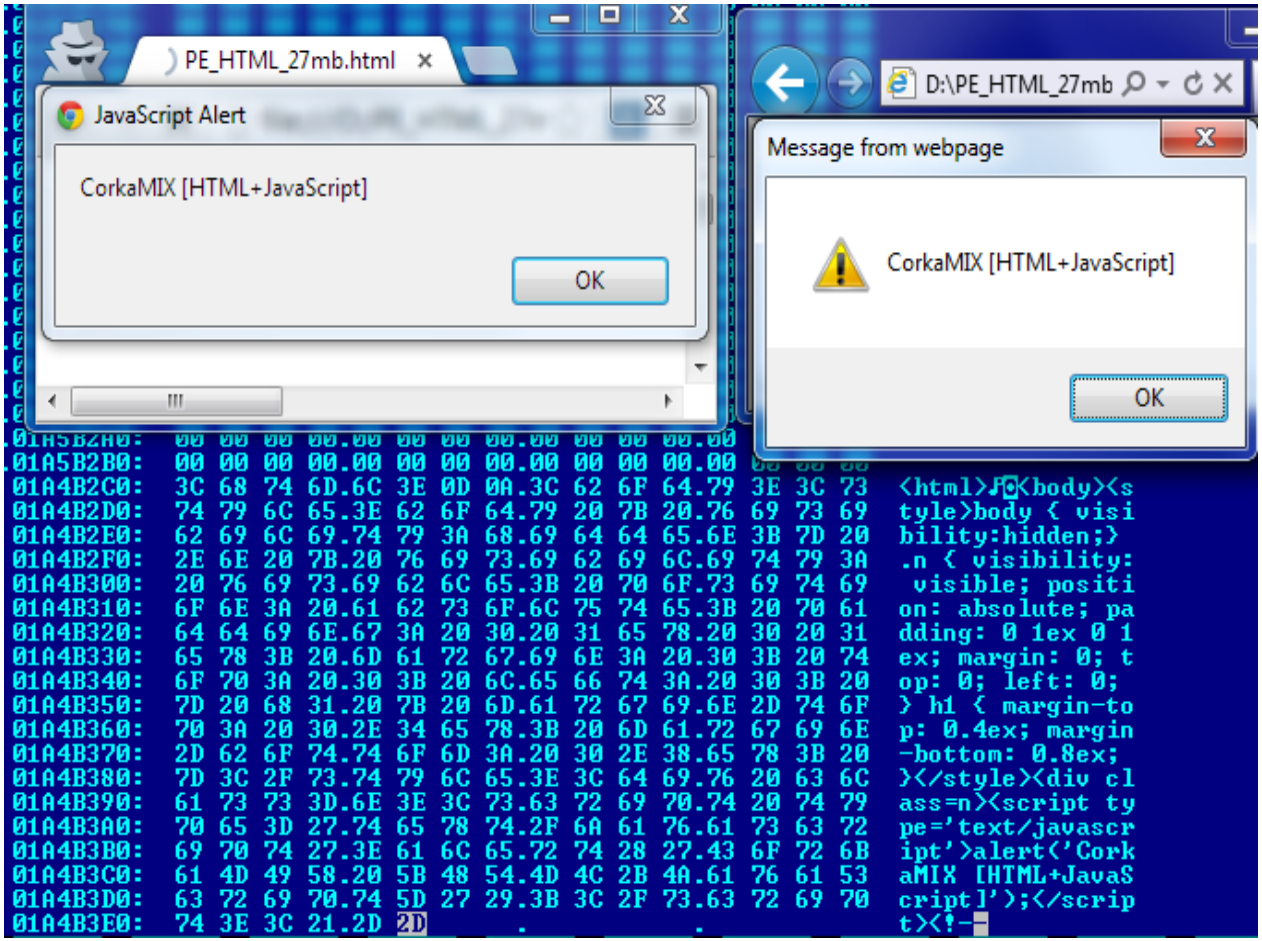
ex: at the end of the file!
such as ***Co***rkami ***S***tandard ***T***est



let's look at HTML format

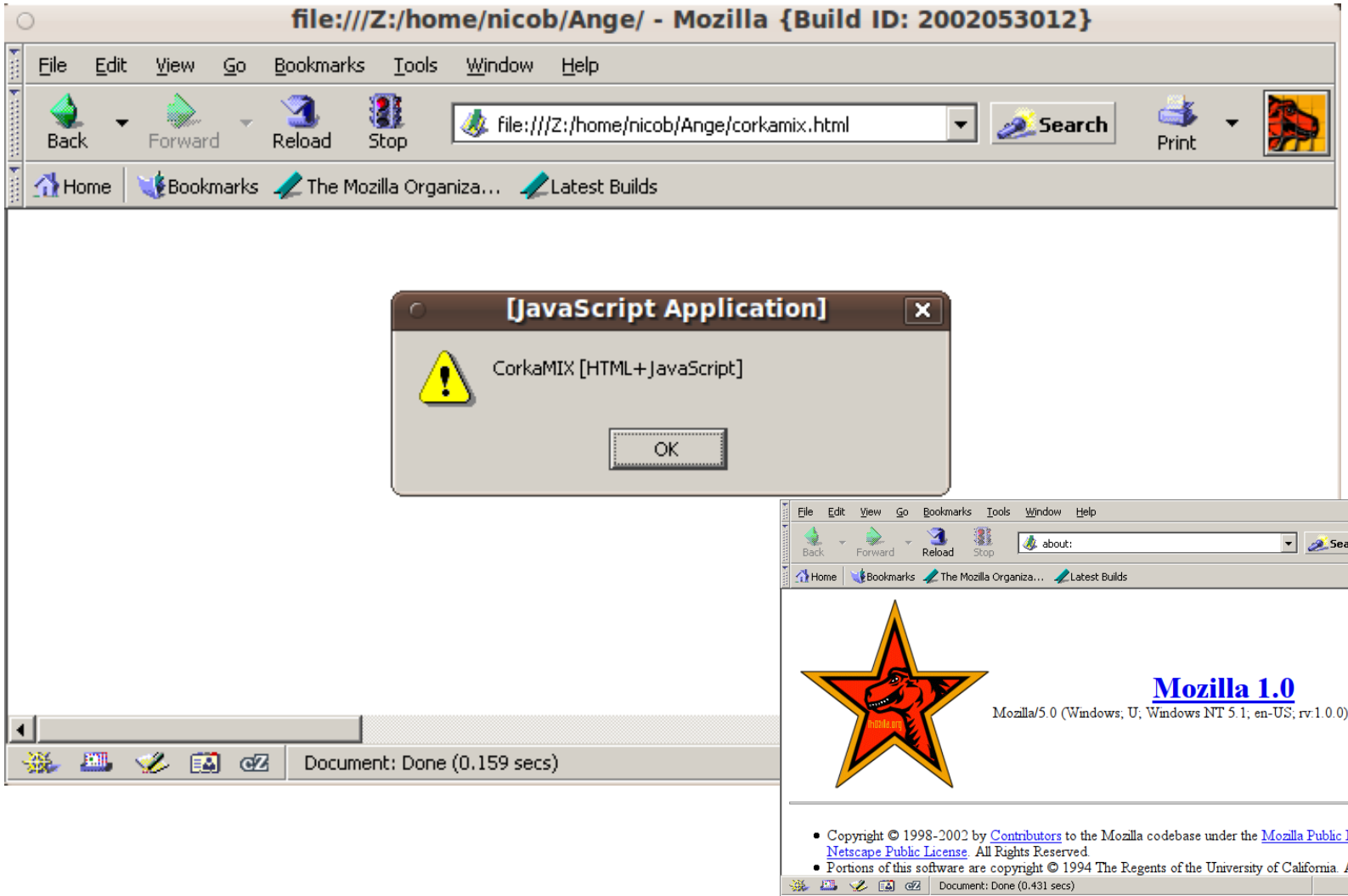
it enforces NOTHING!

anything before the <html> tag!
even 28 Mb of binary!

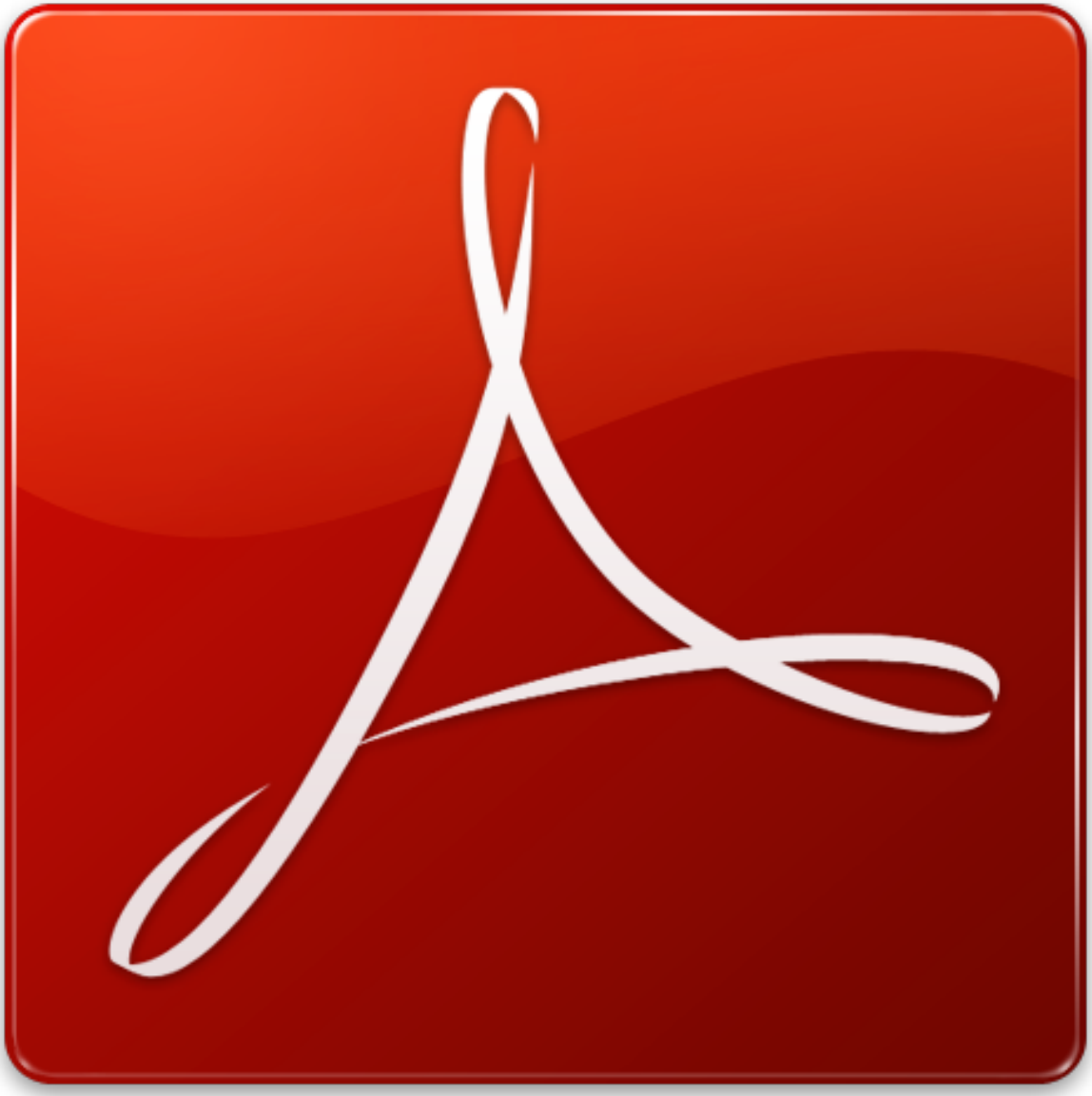


and it's been the same since Mozilla 1.0 in 2002

thanks to Nicolas Grégoire!



now, the PDF format

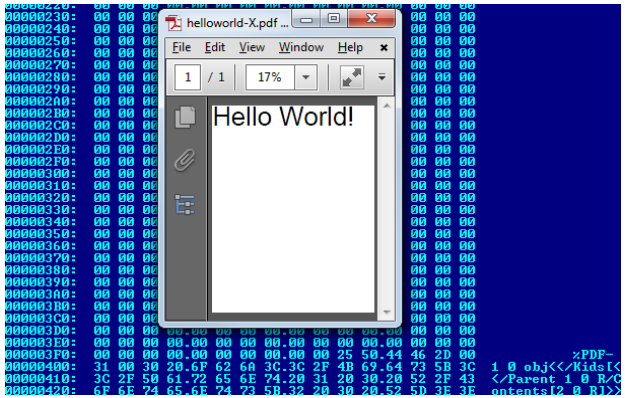


signature position?

- officially at offset 0
- officially tolerated until offset 1024
- wtf?
  - it get actually worse later

7.5.2 File Header

The first line of a PDF file shall be a *header* consisting of the 5 characters %PDF– followed by a version number of the form 1.N, where N is a digit between 0 and 7.

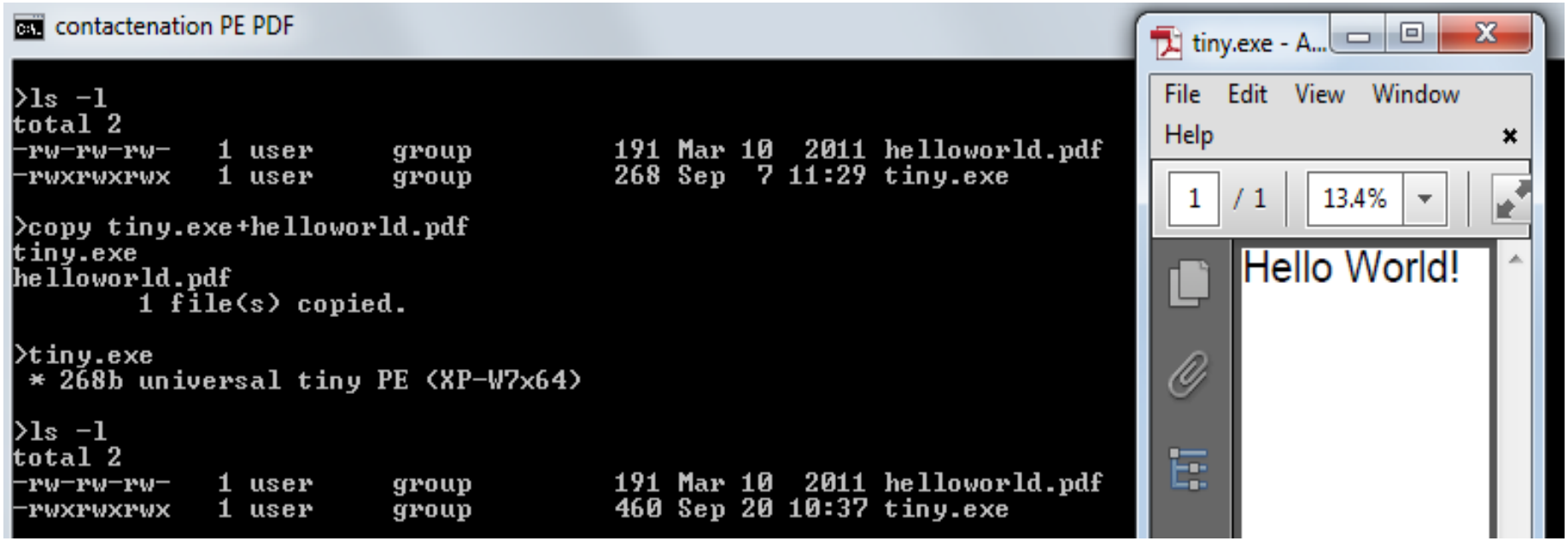


3.4.1, "File Header"

13. Acrobat viewers require only that the header appear somewhere within the first 1024 bytes of the file.

PDF trick 1

put a small executable within 1024 bytes
(just concatenate)

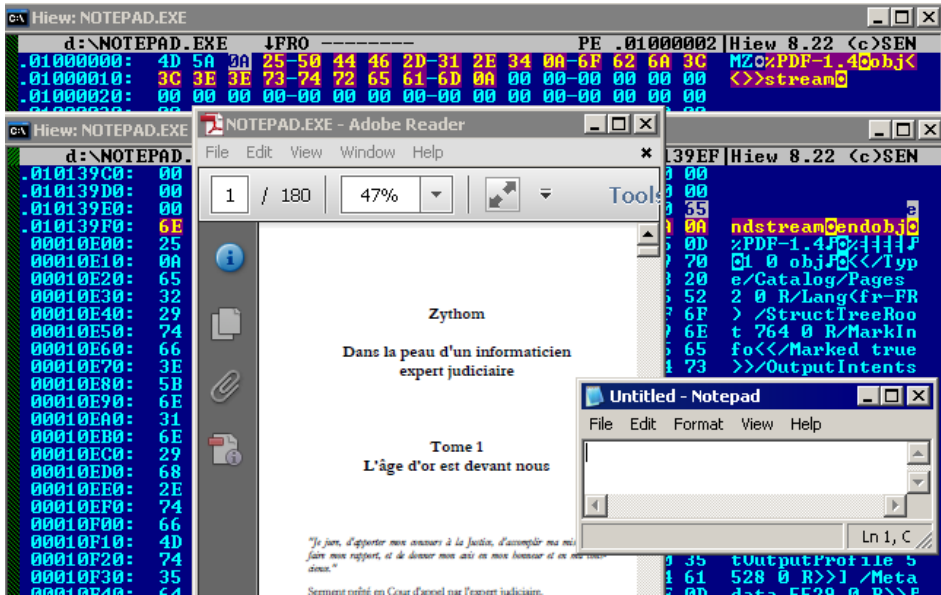


trick 2

1. start a fake PDF + object in a PE header
2. finish fake object at the end the PE
3. end fake object
4. put PDF real structure

works with real-life example!

(PE data might contain PDF keywords)



JAR = ZIP + Class

just enforced at the very end of the file

but CRCs are just ignored

it was too easy :p

```
>yasm -o test.jar zip.asm

>unzip -lv test.jar
Archive:  test.jar
 Length   Method    Size  Ratio   Date   Time   CRC-32    Name
--------  ------  ------- -----   ----   ----   ------    ----
       0  Stored        0   0%  00/00/80 00:00  00000000  META-INF/
      35  Stored       35   0%  00/00/80 00:00  deadbeef  META-INF/MANIFEST.MF
     299  Stored      299   0%  00/00/80 00:00  0badbabe  test.class
--------          -------  ---                          -------
     334              334   0%                          3 files

>unzip -t test.jar
Archive:  test.jar
    testing: META-INF/                      OK
    testing: META-INF/MANIFEST.MF     bad CRC 8391c53a  (should be deadbeef)
    testing: test.class               bad CRC 7846a510  (should be 0badbabe)
At least one error was detected in test.jar.

>java -jar test.jar
Java: Working! (with wrong CRCs)

>
```

Summary

Structure

1. start
    - PE Signature
        - %PDF + fake *obj* start
        - HTML comment start
2. next
    - PE (next)
    - HTML
    - PDF (next)
3. bottom
    - ZIP

it’s time for a real example!

an ***inception*** demo!
wait, what?

we're already in the demo!

the **live** version file is simultaneously:

- the PDF slides themselves
- a PDF viewer executable
  - ie, the file is loading itself
- the PoCs in a ZIP
- an HTML readme
  - with JavaScript mario

so, it works

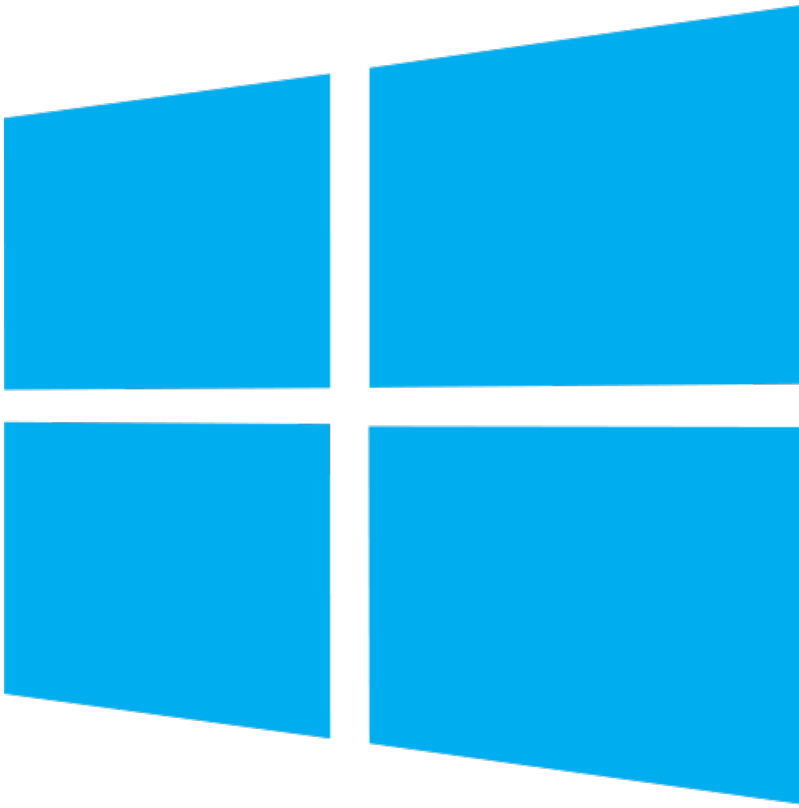
but it lacks something

- not artistic enough
- not advanced enough

let's build a 'well representative' (=nasty) PoC

the PE specs

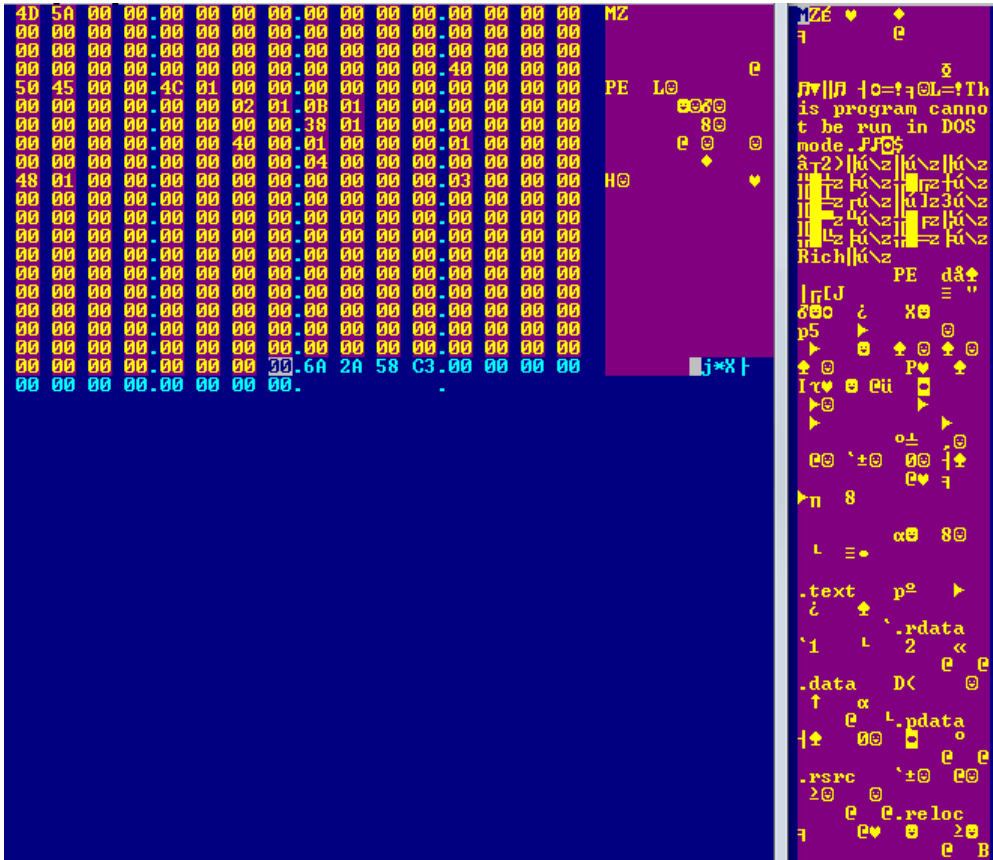
- Official MS specs = big joke
  - 'the gentle guide for beginners'
  - barely describes standard PEs



stripped down PE

many elements removed

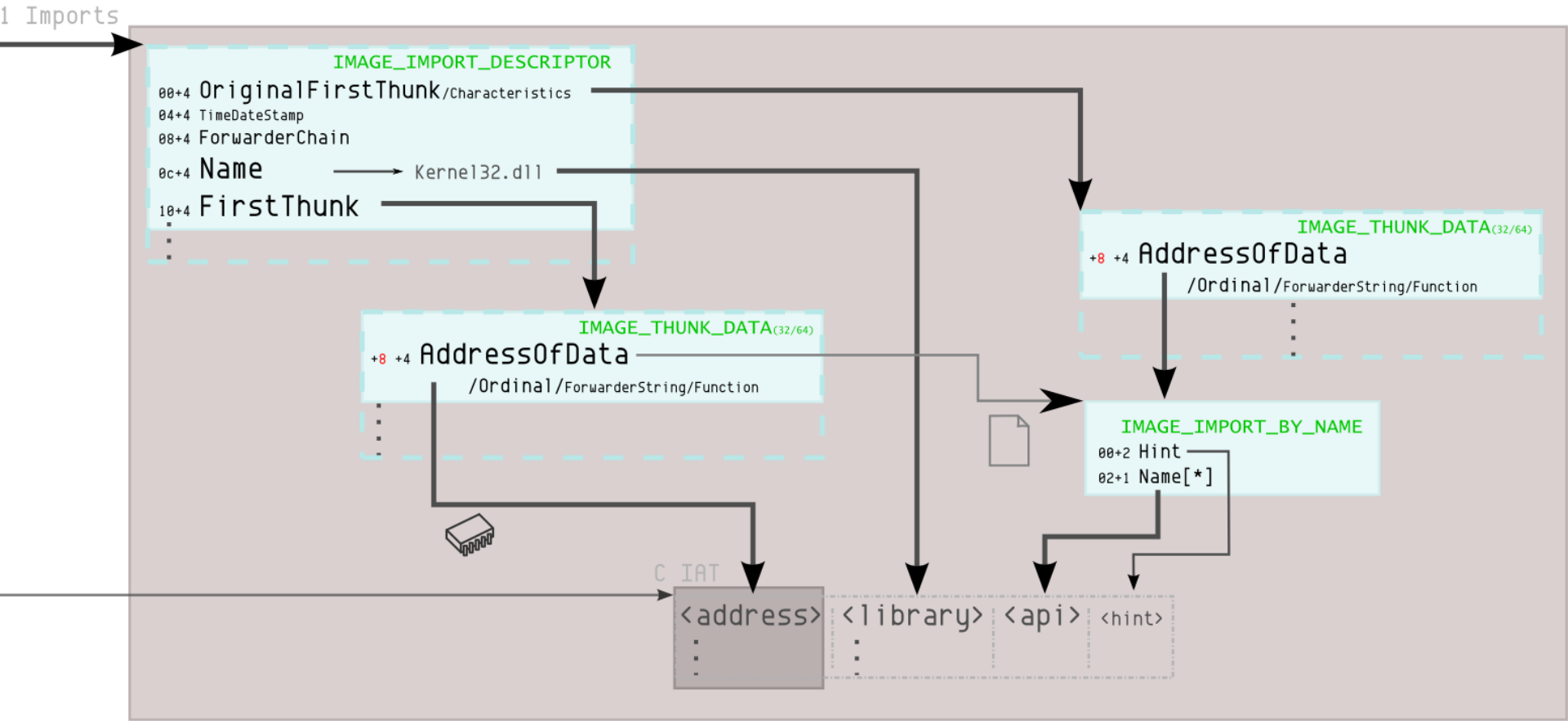
- including no sections



imports

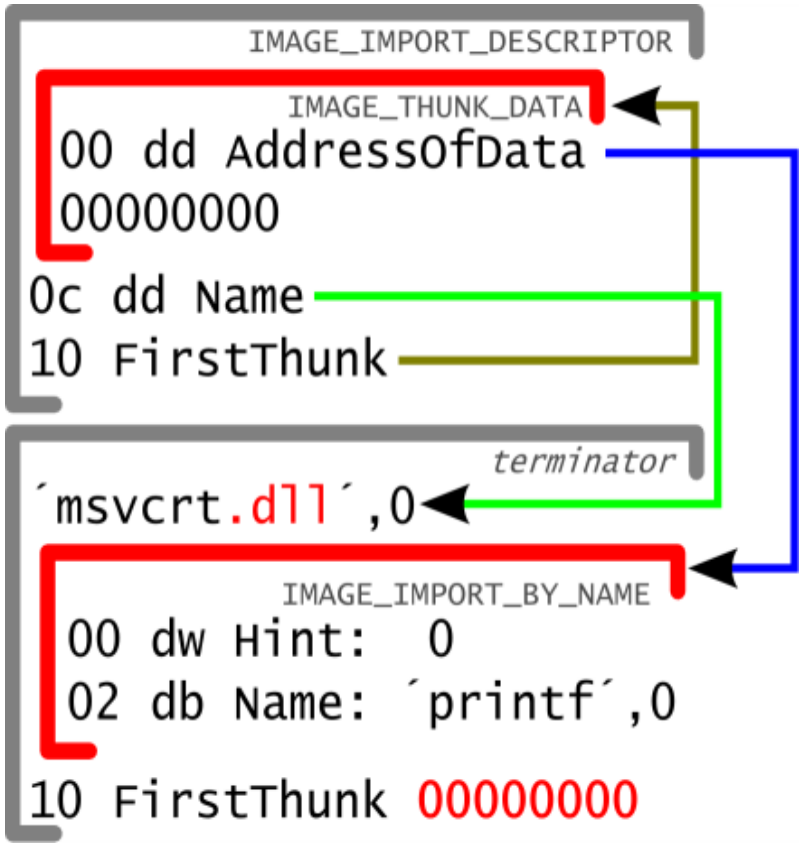
(imports = communication between executables and libraries)

imports are made of 3 lists



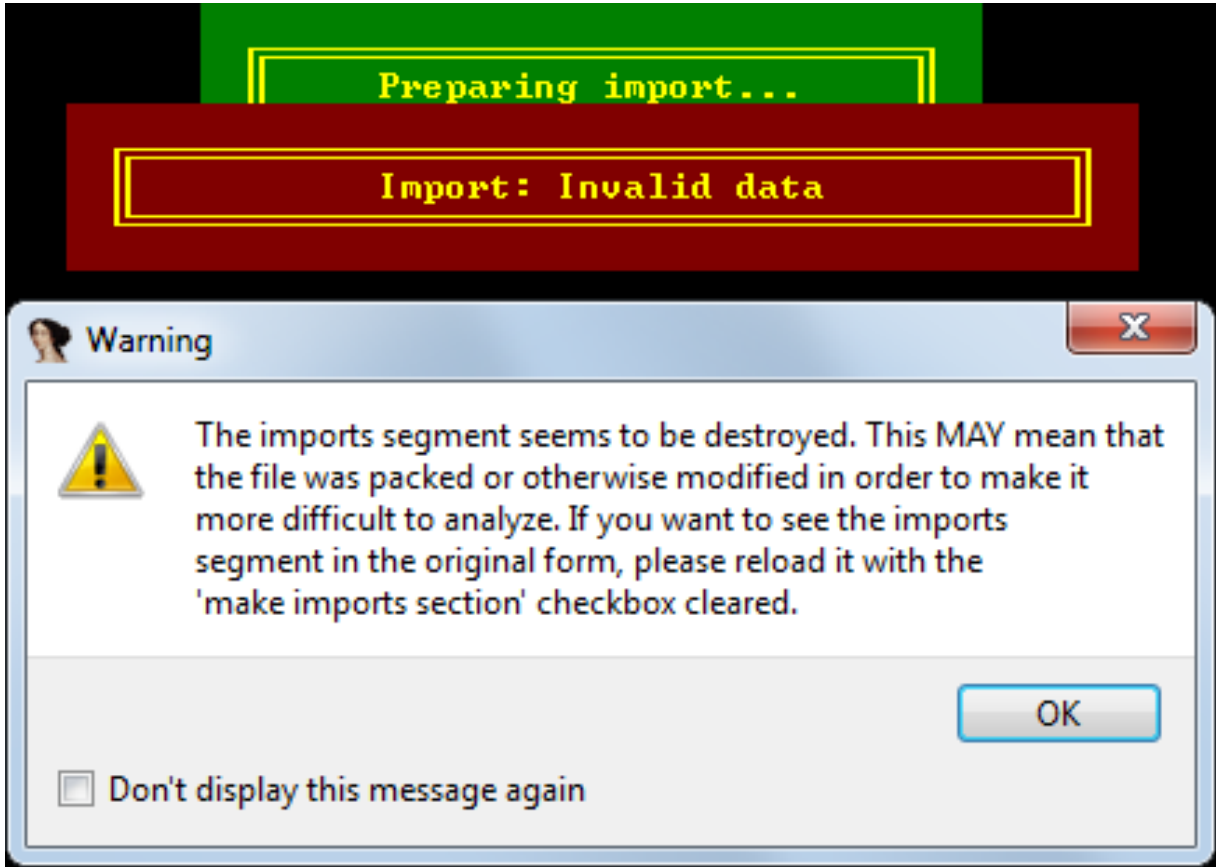
evil imports

- let's make these lists into each other
- with more extra tricks to fail parser!



ultimate import fail

- failing all tools
  - including IDA & Hiew
- now fixed :)



let's put some code

- some undocumented opcodes!
- big blank spaces in Intel official docs

Table A-2. One-byte Opcode Map: (00H — F7H) *

	0	1	2	3	4	5	6	7
D	Shift Grp 2[1A] Eb, 1	Ev, 1	Eb, CL	Ev, CL	AAM[i64] Ib	AAD[i64] Ib		XLAT/ XLATB

Table A-3. Two-byte Opcode Map: 08H — 7FH (First Byte is 0FH) *

	pfx	8	9	A	B	C	D	E	F
0		INVD	WBINVD		2-byte Illegal Opcodes UD2[1B]		NOP Ev		
1		Prefetch[1C] (Grp 16[1A])							NOP Ev
		vmovaps	vmovaps	cvtpi2ps	vmovntps	cvttps2pi	cvtps2pi	vucomiss	vcomiss

let's check AMD's

- miracle!

Table A-1. One-Byte Opcodes, Low Nibble 0–7h

Nibble[1]	0	1	2	3	4	5	6	7
D	Group 2[2]				AAM[3]	AAD[3]	SALC[3]	XLAT
	Eb, 1	Ev, 1	Eb, CL	Ev, CL				

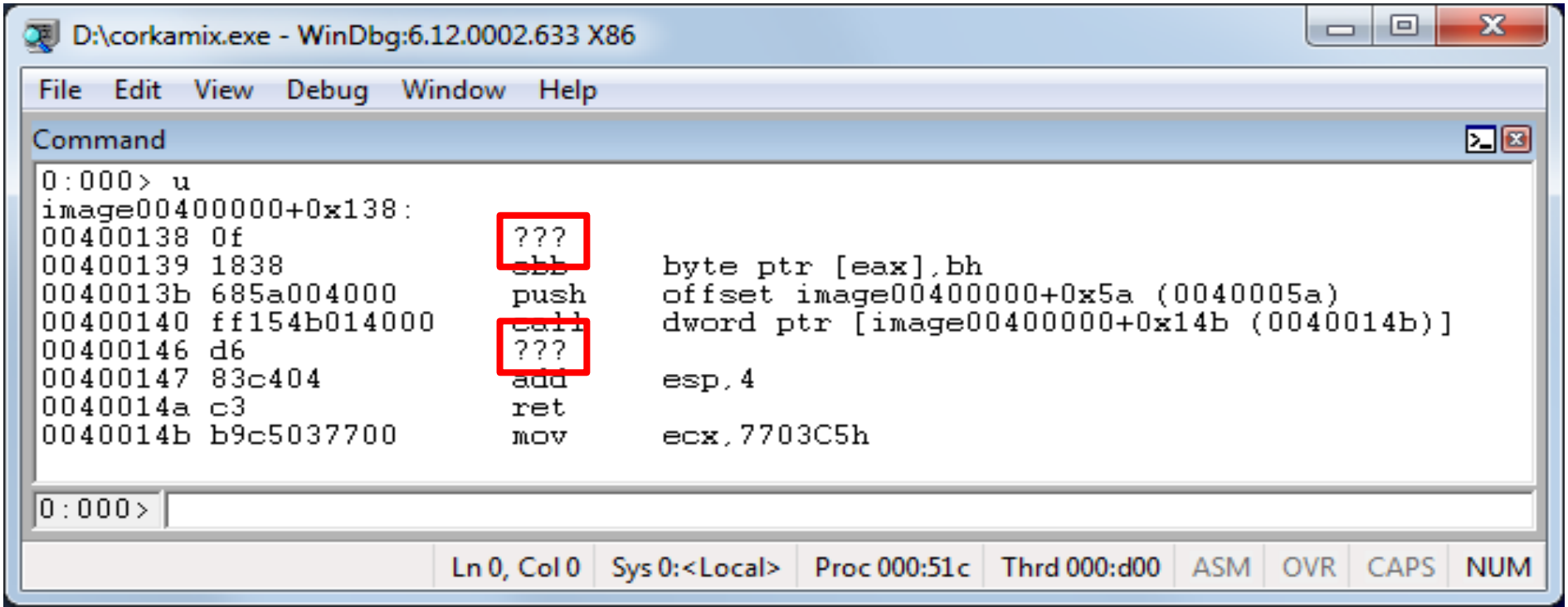
Table A-4. Second Byte of Two-Byte Opcodes, Low Nibble 8–Fh

Prefix	Nibble[1]	8	9	A	B	C	D	E	F
n/a	0	INVD	WBINVD	invalid	UD2	invalid	Group P[2] PREFETCH	FEMMS	3DNow! See "3DNow!™ Opcodes" on page 351
n/a	1	Group 16[2]	NOP[3]	NOP[3]	NOP[3]	NOP[3]	NOP[3]	NOP[3]	NOP[3]

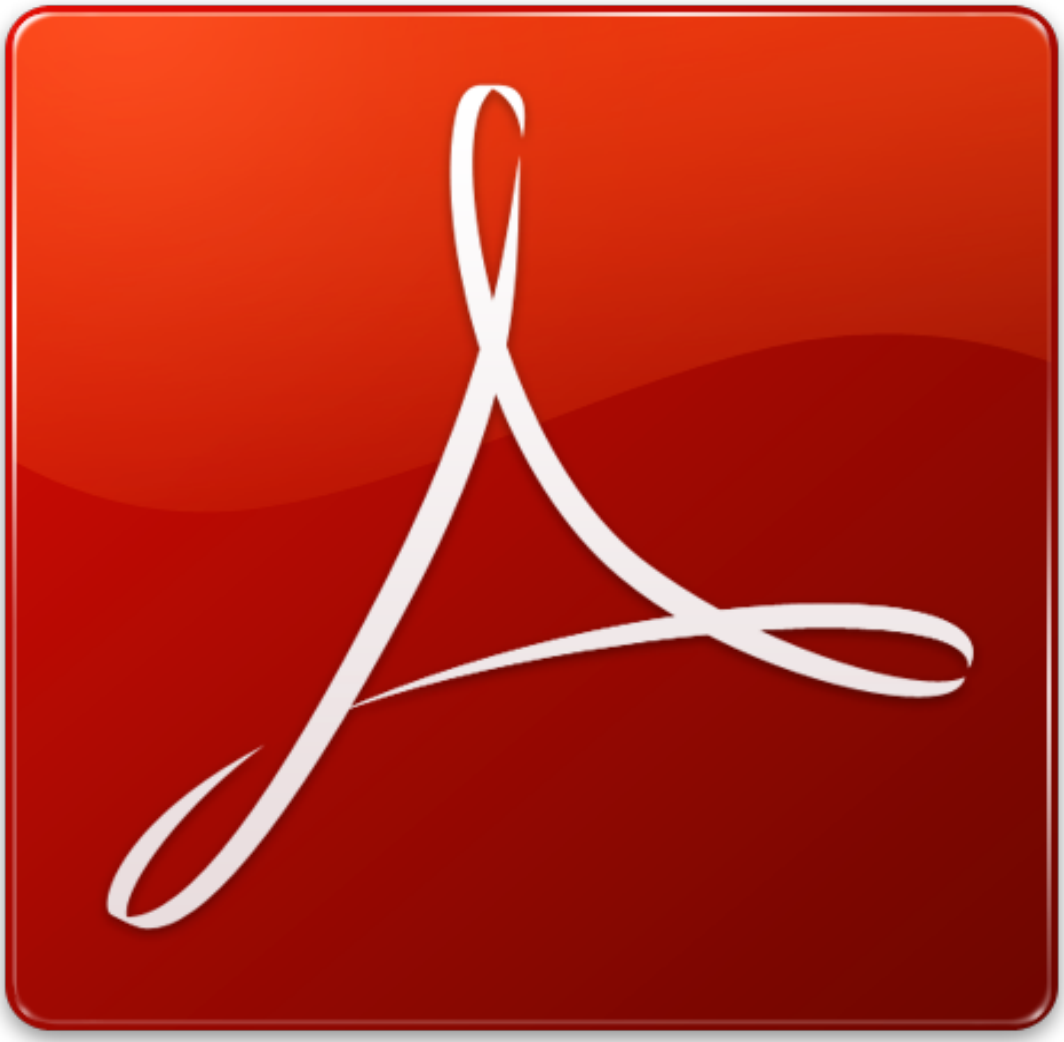
result in WinDbg

- '???' == clueless (tool/user)

don't rely (only) on official docs



messing with PDF

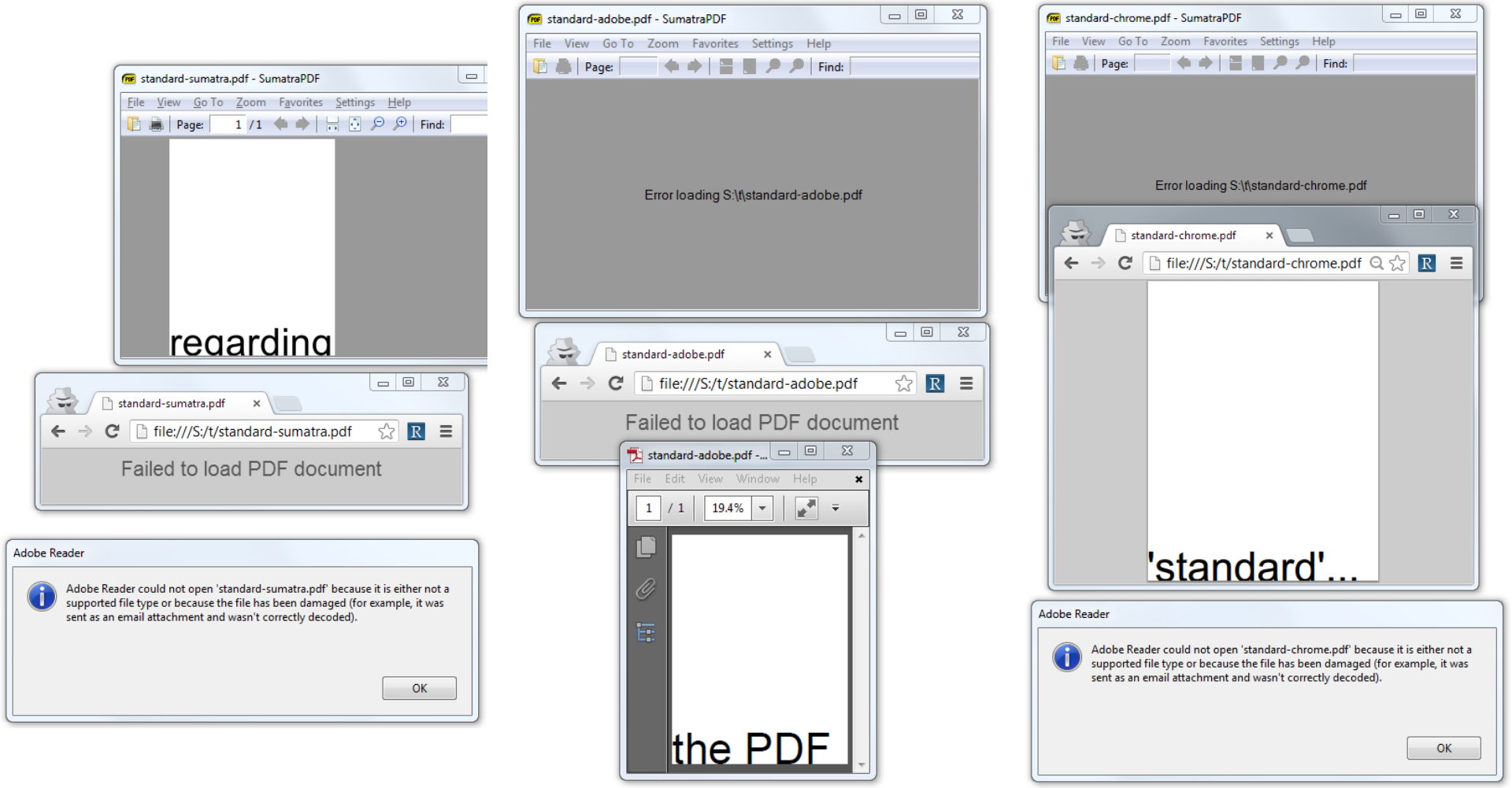


there is a so-called standard

and the reality of existing parsers

looking at: Adobe, MuPDF, Chrome

- 3 different files
  - working each on a specific viewer
  - failing on the other 2


standard-sumatra.pdf - SumatraPDF
File View Go To Zoom Favorites Settings Help
Page: 1 / 1 Find:
regarding
standard-sumatra.pdf
file:///S:/t/standard-sumatra.pdf
Failed to load PDF document
Adobe Reader
Adobe Reader could not open 'standard-sumatra.pdf' because it is either not a supported file type or because the file has been damaged (for example, it was sent as an email attachment and wasn't correctly decoded).
OK
standard-adobe.pdf - SumatraPDF
File View Go To Zoom Favorites Settings Help
Page: Find:
Error loading S:\t\standard-adobe.pdf
standard-adobe.pdf
file:///S:/t/standard-adobe.pdf
Failed to load PDF document
standard-adobe.pdf -...
File Edit View Window Help
1 / 1 19.4%
the PDF
standard-chrome.pdf - SumatraPDF
File View Go To Zoom Favorites Settings Help
Page: Find:
Error loading S:\t\standard-chrome.pdf
standard-chrome.pdf
file:///S:/t/standard-chrome.pdf
'standard'...
Adobe Reader
Adobe Reader could not open 'standard-chrome.pdf' because it is either not a supported file type or because the file has been damaged (for example, it was sent as an email attachment and wasn't correctly decoded).
OK

let's look inside

- MuPDF
  - no *%PDF* sig required
    - a PDF without a PDF sig ? WTF ?!?!
  - no *trailer* keyword required either
- Chrome
  - integer overflows: -4294967275 = 21
  - trailer in a comment
    - it can actually be almost ANYWHERE
    - even inside another object
- Adobe
  - looks almost sane compare to the other 2

```
10 0 obj
<<

  /Count 0
  /Kids [<<

    /Contents 11 0 R
    /Resources <<
      /Font <<
        /F1 <<
          /BaseFont /Arial
        >>
      >>
    >>
  >>]
>>

11 0 obj
<< >>
stream
BT
  /F1 140
  Tf
  (regarding)Tj
ET
endstream


-
<</Root<</Pages 10 0 R>>>>
```

```
%PDF-1.

30 0 obj
<<


  /Kids [<<
    /Parent 30 0 R
    /Contents 31 0 R
    /Resources <<>>



  >>]
>>

31 0 obj
<< >>
stream
BT
  /F1 150
  Tf 1 0 0 1 1 0
  Tm(the PDF)Tj
ET
endstream
endobj

trailer
<</Root<</Pages 30 0 R>>>>
```

```
%PDF

20 0 obj
<<
  /Pages <<

    /Kids [<<

      /Contents -4294967275 4294967296 R



  >>]
>>

21 0 obj
<< >>
stream
BT
  110
  Tf
  ('standard'...)Tj

endstream


% trailer
<</Root 20 0 R>>
```

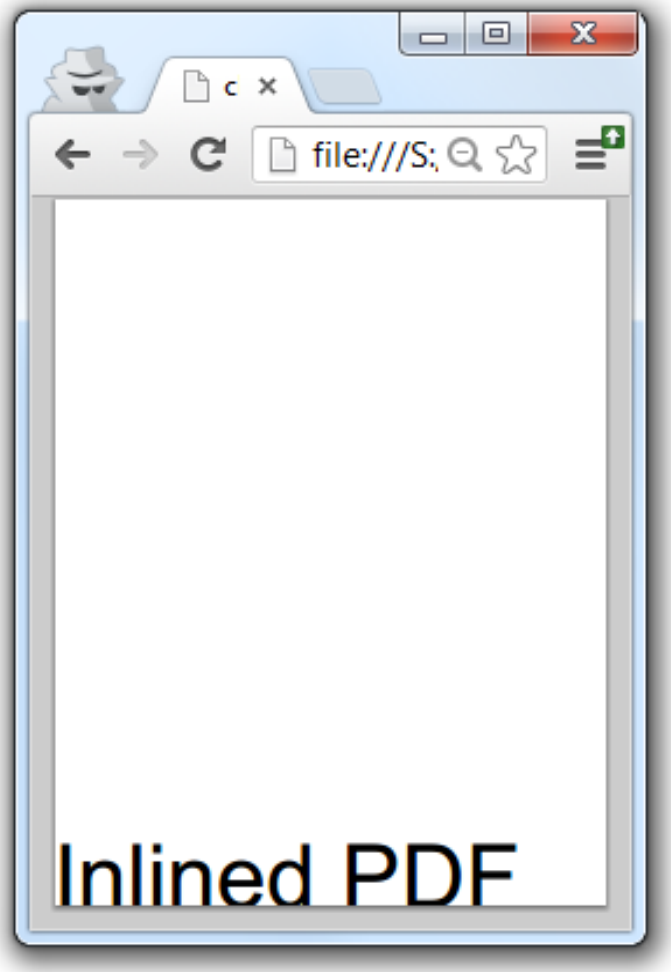
Chrome insanity++

(thx to Jonas Magazinius)

- a single object
- no 'trailer'
- inline stream
- brackets are not even closed
- * are required - it just checks for minimum space

All streams must be indirect objects (see Section 3.2.9, “Indirect Objects”) and the stream dictionary must be a direct object. The keyword **stream** that follows the stream dictionary should be followed by an end-of-line marker consisting of either a carriage return and a line feed or just a line feed, and not by a carriage

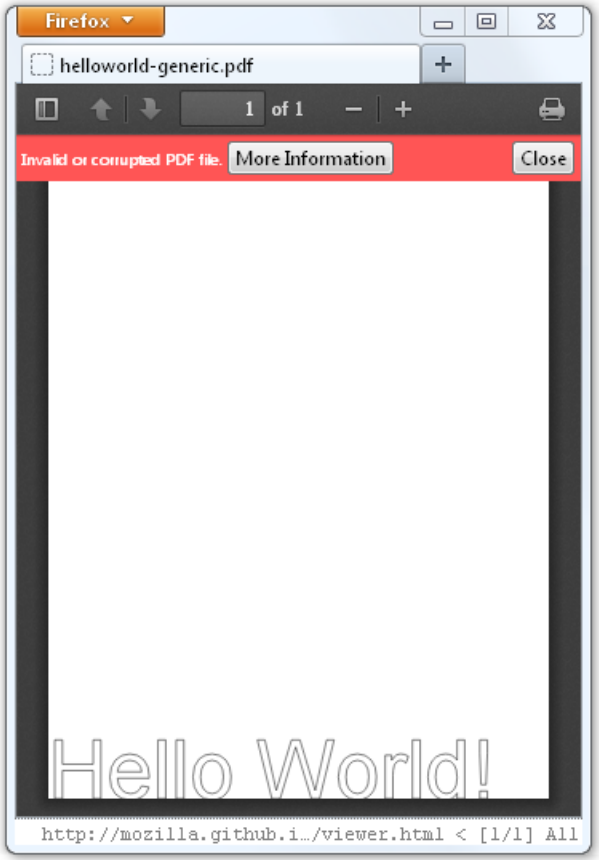
```
%PDF*****
1 0 obj
<<
    /Size 2
    /W[[]1/]
    /Root 1 0 R
    /Pages<<
        /Kids[<<
            /Contents<<>>
            stream
            BT{99
            Tf{Td(Inlined PDF)'
            endstream
        >>]
    >>
>>
stream
*
endstream
startxref%*******
```



PDF.JS

- very strict
  - 'too' strict / naive ?
  - I don't want to be their QA ;)
- requires a lot of information usually ignored
  - xref
  - /Length

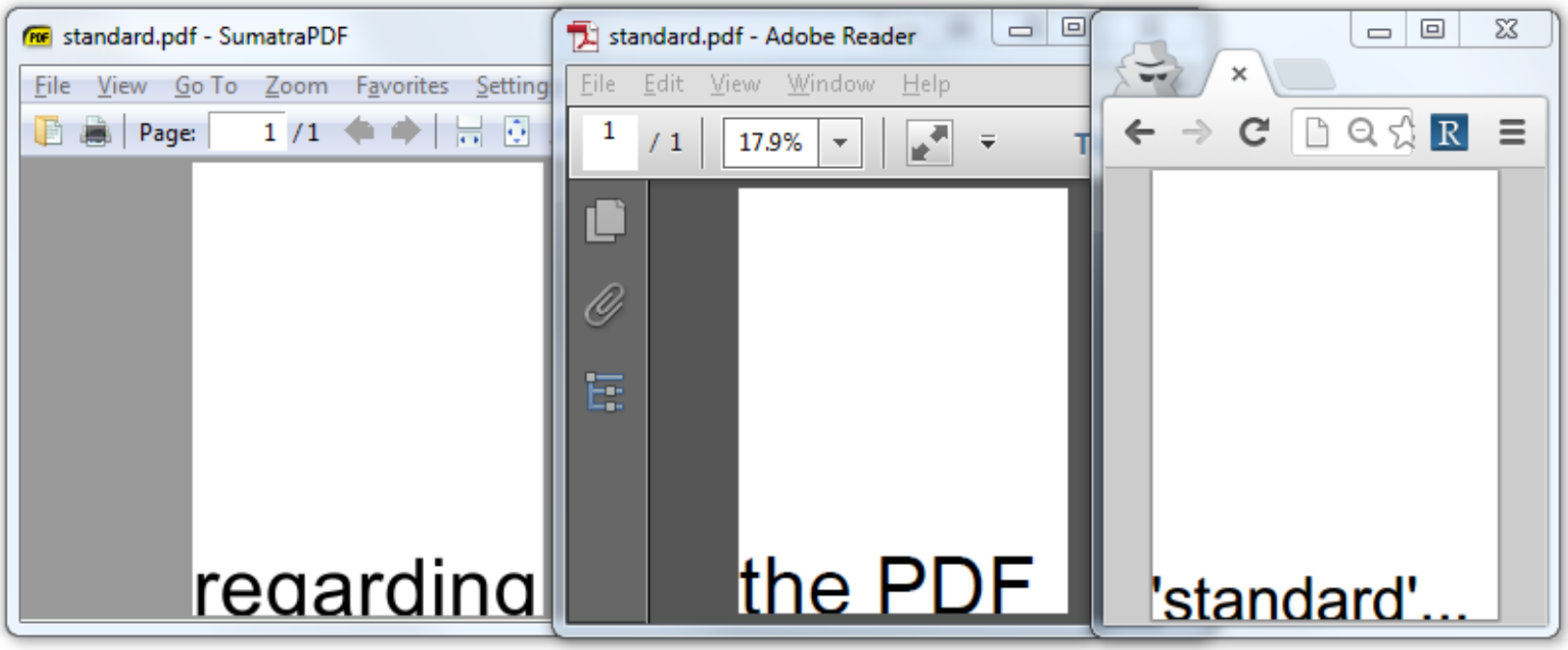
```
%PDF-1.1
1 0 obj
<<
%               /Type /Catalog
                ...
>>
endobj

2 0 obj
<<
                /Type /Pages
                ...
>>
endobj

3 0 obj
<<
                /Type /Page
                /Resources <<
                                /Font <<
                                                /F1 <<
                /Type /Font
                /Subtype /Type1
...
                                                >>
                                >>
                >>
>>
endobj

4 0 obj
<< /Length 47>>
stream
...

xref
0 1
0000000000 65535 f
0000000010 00000 n
...
```



let's play further

combine 3 documents in a ***single*** file

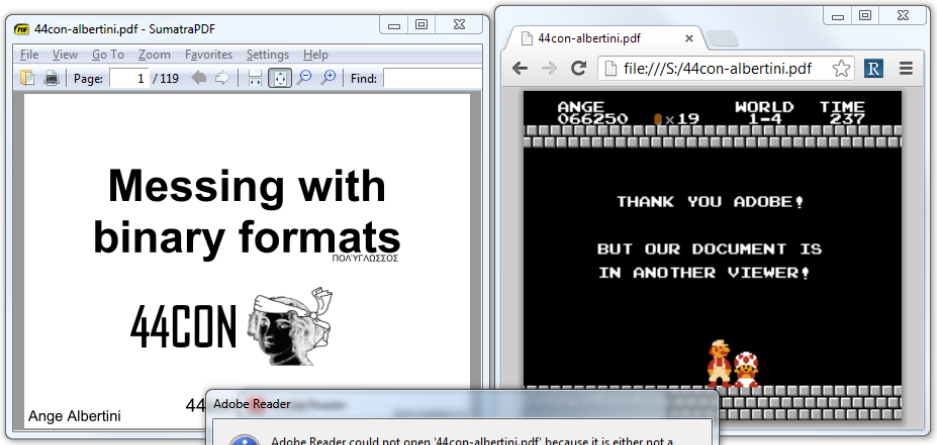
- it's actually 3 set of 'independant' objects
- objects are parsed
  - but not used



alternate reality demo

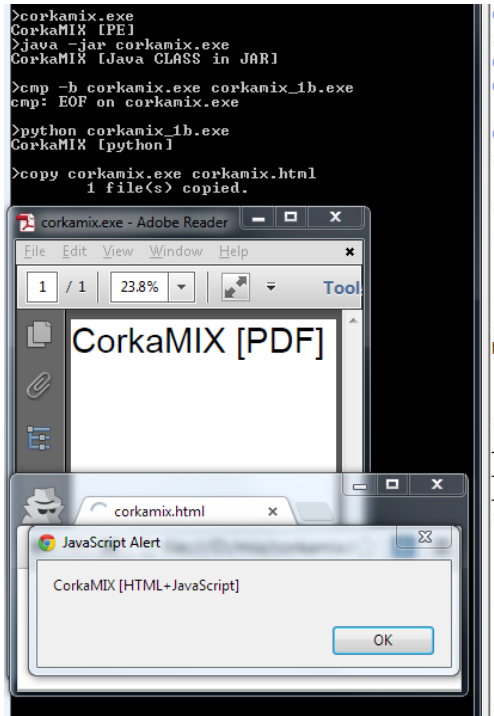
the live slide-deck contains 2 PDF

- bogus one under Chrome
- real one under MuPDF (Sumatra, Linux...)
- rejected under Acrobat
  - because of the PE signature (see later)



final PoC

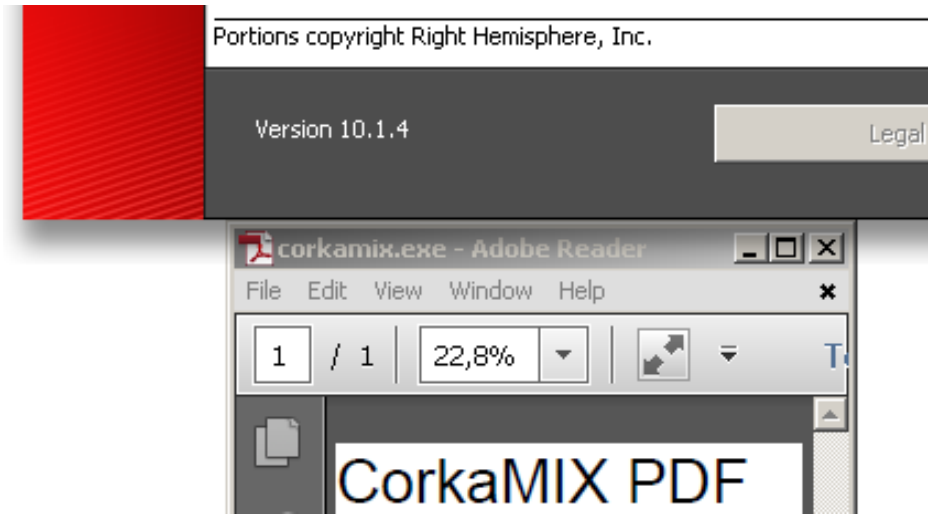
- combine most previously mentioned tricks
- many fails on many tools
- total control of the structure
  - the PDF 'ends' in the Java class

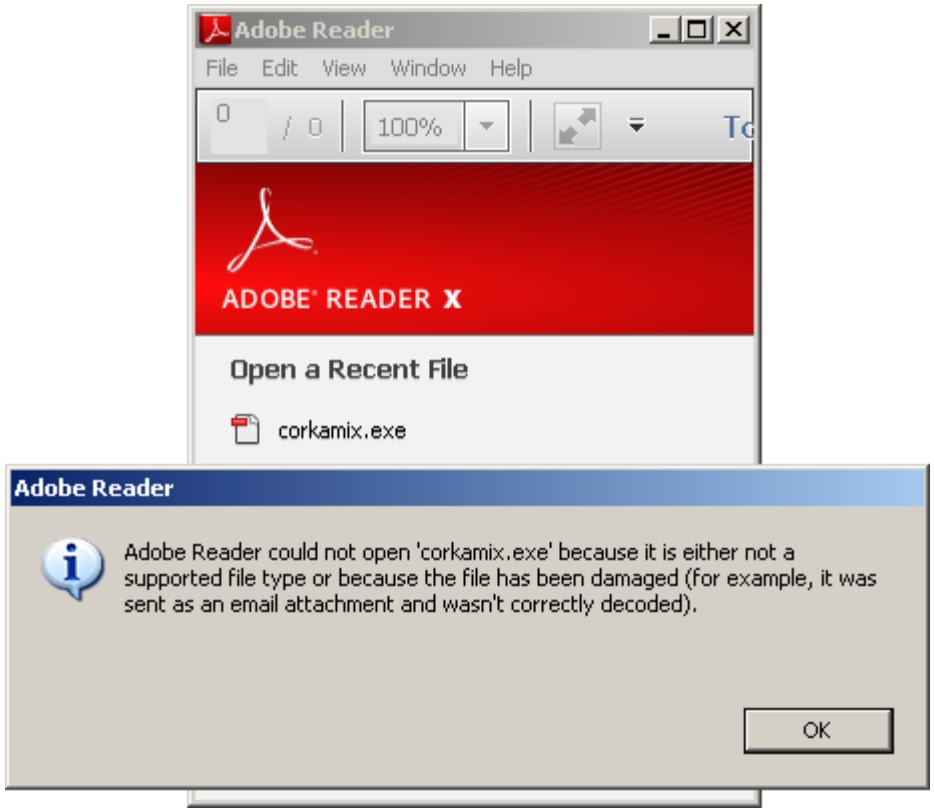



```
db 'MZ'
; [...]
db '%PDF-1.', 0ah
db 'obj<<>>stream', 0ah

db '<html>'
; [...]
    at IMAGE_NT_HEADERS.Signature, db 'PE',0,0
; [...]
    db 0fh, 018h, 111b << 3
    push msg
    call [__imp__printf]
    salc
; [...]
header:
    db 'PK', 3, 4
    dw 0ah ; version_needed
; [...]
_dd 0CAFEBABEh ; signature
_dw 3          ; major version
_dw 2dh        ; minor version
; [...]
            _dd 9 ; length of bytecode
                GETSTATIC 8
                LDC 14
                INVOKEVIRTUAL 16
                RETURN
            _dw 0 ; exceptions_count
            _dw 0 ; attributes_count
; [...]
```

Adobe rejects 'weird magics' after 10.1.5

not in their own specs :p



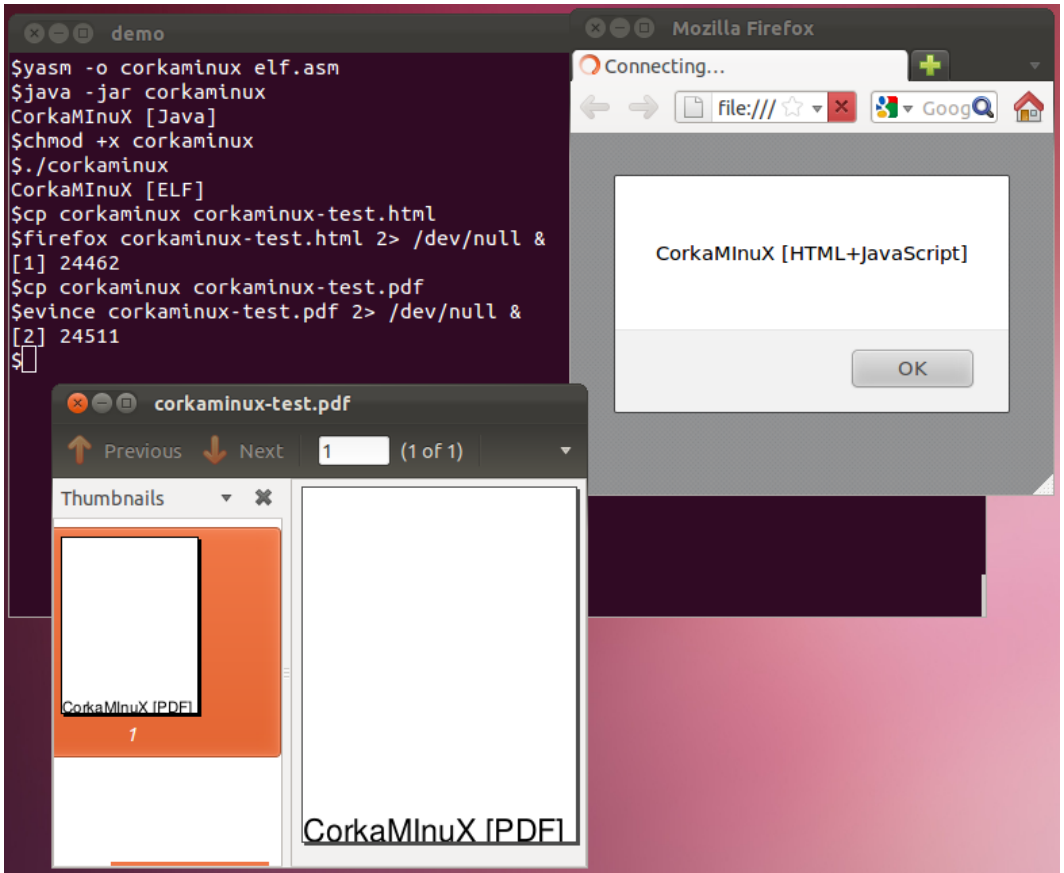


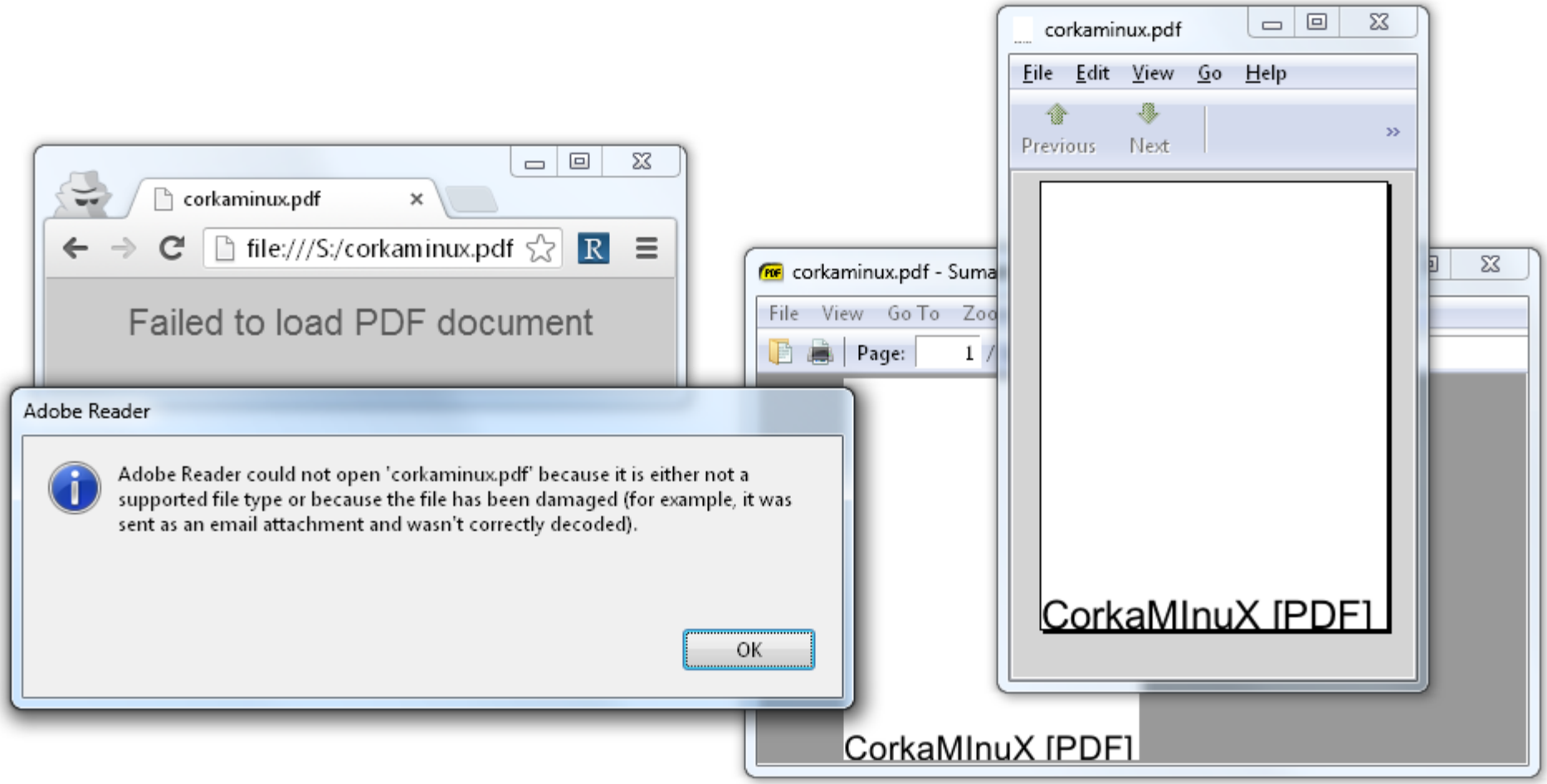
10.1.4

10.1.5

also in ELF/Linux flavor

- starring a signature-less PDF
  - which won't run on other viewers

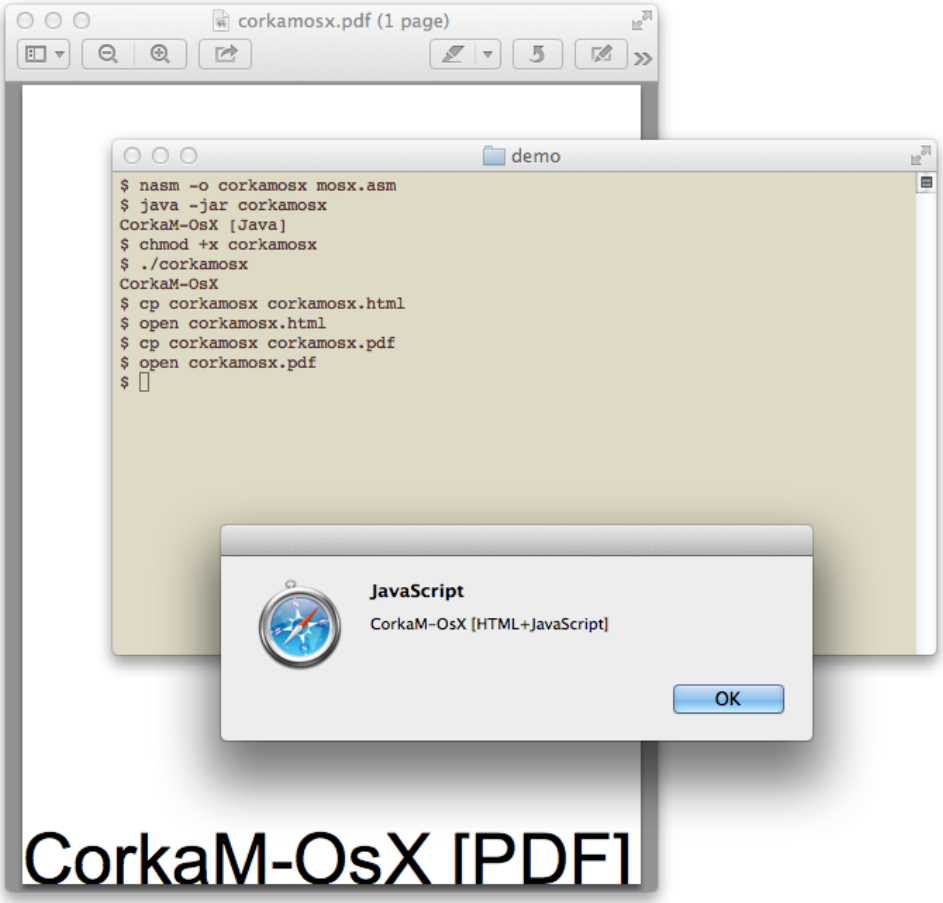



corkaminux.pdf
file:///S:/corkaminux.pdf
Failed to load PDF document
Adobe Reader
Adobe Reader could not open 'corkaminux.pdf' because it is either not a supported file type or because the file has been damaged (for example, it was sent as an email attachment and wasn't correctly decoded).
OK
corkaminux.pdf - Suma
File View Go To Zoo
Page: 1 /
CorkaMInuX [PDF]
corkaminux.pdf
File Edit View Go Help
Previous Next
CorkaMInuX [PDF]

and Apple too

PS: I don't have a Mac, this was built blindly

Thanks to Nicolas Seriot for testing



why should we care?

like washing powders

security tools are selected:

- speed
- {files} → {[clean/detected]}

file types not taken into consideration

type confusion

make the tool believe it's another type, which will fool the engine

engine with checksum caching will be fooled:

1. scanned as HTML, clean
2. reused as PE but malicious

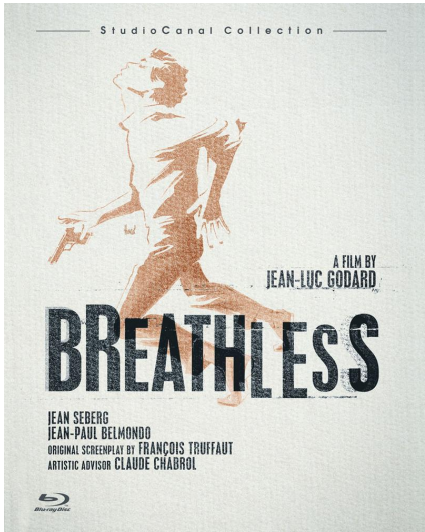
SHA256:	2a9c7a16cdb3c3f2285afaf61072dd5e7cc022e97f351cad6234a13e5216f389
SHA1:	e27faaa006229f8e4ab97fba7019dc9f2797f84d
MD5:	88cad2b56ab67b43794a0f7a4e690fd5
File size:	1.5 KB (1530 bytes)
File name:	corkamix.exe
File type:	PDF
Tags:	pdf

engine exhaustion

rankings in magazines are based on scanning time

→ scanning per file must stop arbitrarily

→ waste scanning cycle by adding extra formats



Weaknesses

- evasion
  - filters → exfiltration
  - *same origin policy*
  - detection
    - ex: clean PE but malicious PDF/HTML/...
    - exhaust checks
    - pretend to be corrupt
- DoS

Conclusion

Conclusion

- type confusion is bad
  - succinct docs too
  - lazy softwares as well

- go beyond the specs
  - Adobe: good

- suggestions
  - more extensions checks
  - isolate downloaded files
  - enforce magic signature at offset 0

thank YOU !

Questions ?

http://

reverseengineering

.stackexchange.com

@angealbertini

✉ ange@corkami.com

Bonus

Valid image as JavaScript

Highlighted by Saumil Shah

- abusing header and parsers laxisms
- turn a field into /*
- close comment after the picture data