Caradoc: a Pragmatic Approach to PDF Parsing and Validation

IEEE Security & Privacy LangSec Workshop 2016

Guillaume Endignoux Olivier Levillain Jean-Yves Migeon

École Polytechnique, France
EPFL, Switzerland
ANSSI, France

Thursday 26th May, 2016

Portable Document Format ?

A commonly used format, but many security issues:

- 500+ reported vulnerabilities in Adobe Reader[1] (since 1999).
- Discrepancies between implementations.
- Syntax facilitates polymorphism[2] (PDF+ZIP, PDF+JPEG, etc.).

[1] `http://www.cvedetails.com`
[2] See for example PoC||GTFO

Portable Document Format ?

A commonly used format, but many security issues:

- 500+ reported vulnerabilities in Adobe Reader[1] (since 1999).
- Discrepancies between implementations.
- Syntax facilitates polymorphism[2] (PDF+ZIP, PDF+JPEG, etc.).

In our work, we aim at verifying PDFs from syntactic level.

Two approaches to validate files:

- **Blacklist**: does not detect new malware...
- **Whitelist**: higher rejection rate, but accepted files are clean.

[1] `http://www.cvedetails.com`
[2] See for example PoC||GTFO

Table of contents

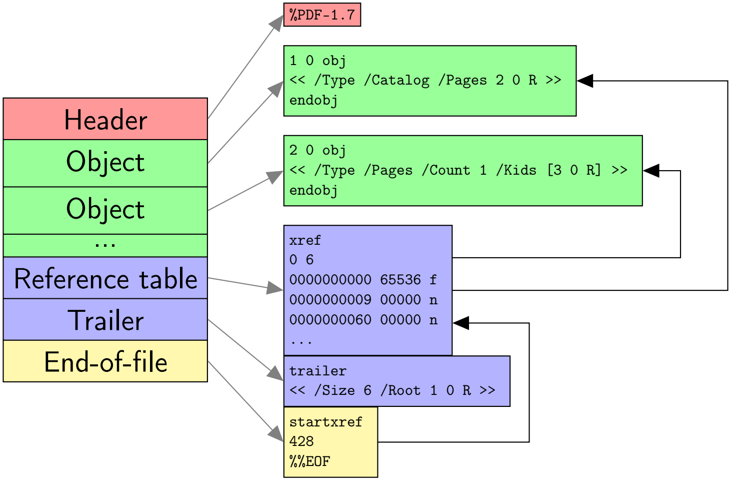
Table of contents

PDF syntax 101

A PDF document is made of objects:

- `null`
- booleans: `true`, `false`
- numbers: `123`, `-4.56`
- strings: `(foo)`
- names: `/bar`
- arrays: `[1 2 3]`, `[(foo) /bar]`
- dictionaries: `<< /key (value) /foo 123 >>`
- references: `1 0 obj ... endobj` and `1 0 R`
- streams: `<< ... >> stream ... endstream`

Structure of a PDF file



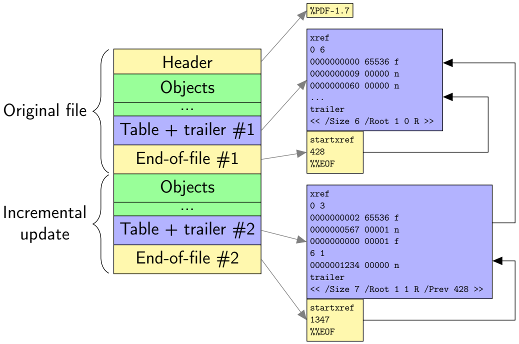
```
%PDF-1.7

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

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

xref
0 6
0000000000 65536 f
0000000009 00000 n
0000000060 00000 n
...

trailer
<< /Size 6 /Root 1 0 R >>

startxref
428
%%EOF
```

Organization of a simple PDF file.

Structure of a PDF file

More complex structures:

- incremental updates,
- object streams,
- linearization.



```
%PDF-1.7
```

```
xref
0 6
0000000000 65536 f
0000000009 00000 n
0000000060 00000 n
...
trailer
<< /Size 6 /Root 1 0 R >>
```

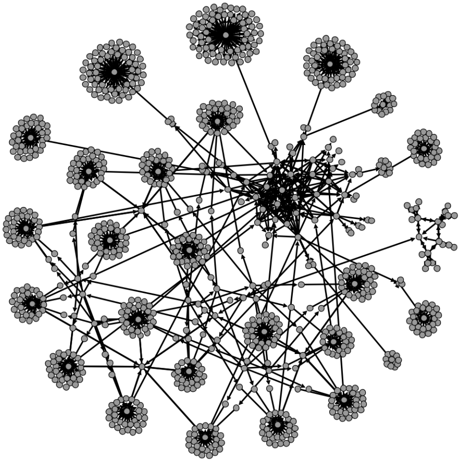
```
startxref
428
%%EOF
```

```
xref
0 3
0000000002 65536 f
0000000567 00001 n
0000000000 00001 f
6 1
0000001234 00000 n
trailer
<< /Size 7 /Root 1 1 R /Prev 428 >>
```

```
startxref
1347
%%EOF
```

Incremental update.

Logical structure of a PDF file

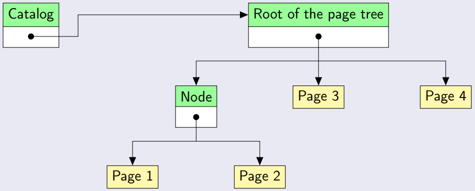


Document of 17 pages (about 1000 objects).

Graph organization

The graph of objects is organized into sub-structures, especially trees.

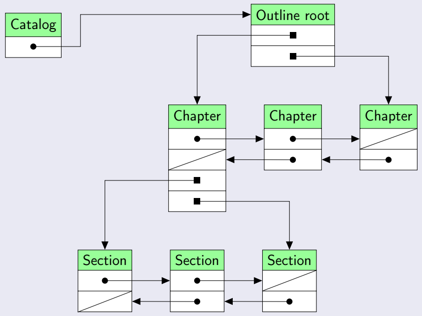
Page tree.



Graph organization

The table of contents uses doubly-linked lists.

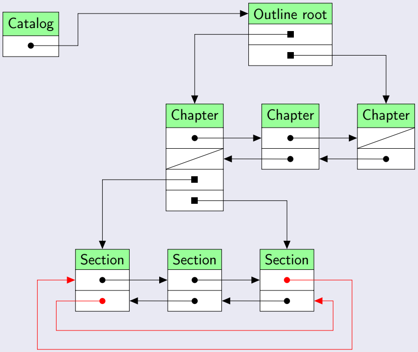
Table of contents.



Problematic structure

An attacker may write an invalid structure.

Invalid table of contents.



Demonstration

Demonstration: two examples

Loop in the outline structure

https://github.com/ANSSI-FR/caradoc/blob/master/test_files/negative/outlines/cycle.pdf

Polymorphic file

https://github.com/ANSSI-FR/caradoc/blob/master/test_files/negative/polymorph/polymorph.pdf

These files were reported to software editors.

Demonstration

These problems may lead to several attacks:

- Attacks on the structure (denial of service).
- Evasion techniques (attacks taking advantage of implementation discrepancies).

Table of contents

Solution proposals

Caradoc verifies a document at three levels:

- File syntax.
- Objects consistency (type checking).
- Higher-level verifications (graph, etc.).

Syntax restriction

At **syntax** level, guarantee extraction of objects without ambiguity:

- Grammar formalization[3] (BNF).
- Structure restrictions (no updates, no *linearization*, etc.).
- Systematic rejection of “corrupted” files.

[3] `https://github.com/ANSSI-FR/caradoc/tree/master/doc/grammar`

Syntax restriction

At **syntax** level, guarantee extraction of objects without ambiguity:

- Grammar formalization[3] (BNF).
- Structure restrictions (no updates, no *linearization*, etc.).
- Systematic rejection of "corrupted" files.

> *When a conforming reader reads a PDF file with a damaged or missing cross-reference table, it* ***may attempt*** *to rebuild the table by scanning all the objects in the file.*
>
> — ISO 32000-1:2008, annex C.2

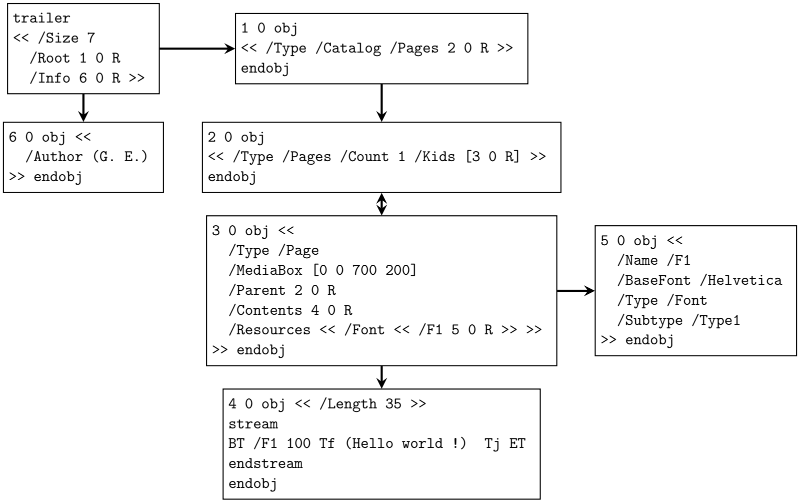
[3] `https://github.com/ANSSI-FR/caradoc/tree/master/doc/grammar`

Type checking

At **object** level: guarantee semantic consistency.

For this purpose: *type checking* algorithm.

Type checking



```
trailer
<< /Size 7
  /Root 1 0 R
  /Info 6 0 R >>
```

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

```
6 0 obj <<
  /Author (G. E.)
>> endobj
```

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

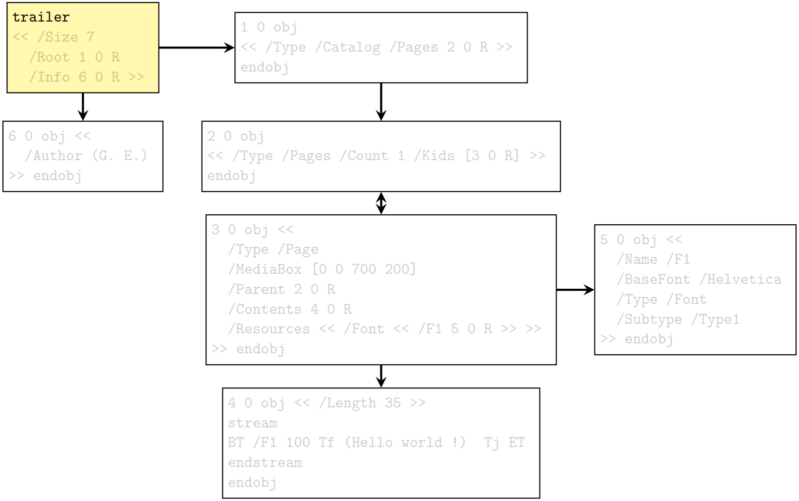
```
3 0 obj <<
  /Type /Page
  /MediaBox [0 0 700 200]
  /Parent 2 0 R
  /Contents 4 0 R
  /Resources << /Font << /F1 5 0 R >> >>
>> endobj
```

```
5 0 obj <<
  /Name /F1
  /BaseFont /Helvetica
  /Type /Font
  /Subtype /Type1
>> endobj
```

```
4 0 obj << /Length 35 >>
stream
BT /F1 100 Tf (Hello world !)  Tj ET
endstream
endobj
```

Example on a Hello World file.

Type checking



```
trailer
<< /Size 7
  /Root 1 0 R
  /Info 6 0 R >>
```

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

```
6 0 obj <<
  /Author (G. E.)
>> endobj
```

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

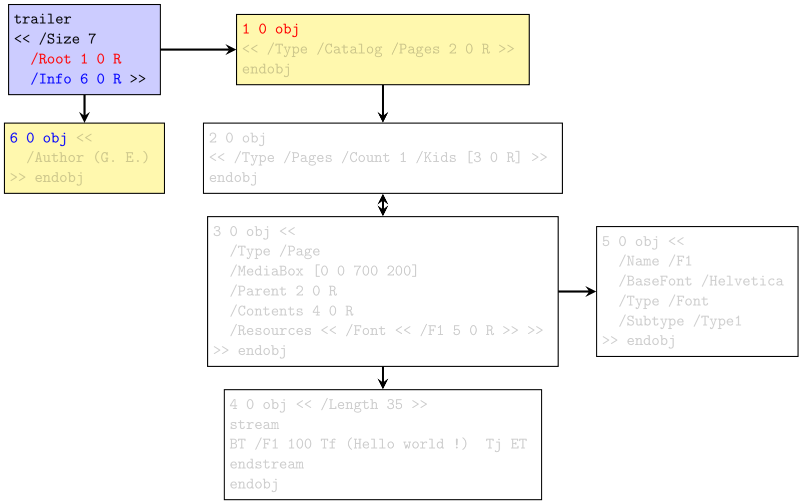
```
3 0 obj <<
  /Type /Page
  /MediaBox [0 0 700 200]
  /Parent 2 0 R
  /Contents 4 0 R
  /Resources << /Font << /F1 5 0 R >> >>
>> endobj
```

```
5 0 obj <<
  /Name /F1
  /BaseFont /Helvetica
  /Type /Font
  /Subtype /Type1
>> endobj
```

```
4 0 obj << /Length 35 >>
stream
BT /F1 100 Tf (Hello world !)  Tj ET
endstream
endobj
```

Constraint propagation.

Type checking



```
trailer
<< /Size 7
  /Root 1 0 R
  /Info 6 0 R >>
```

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

```
6 0 obj <<
  /Author (G. E.)
>> endobj
```

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

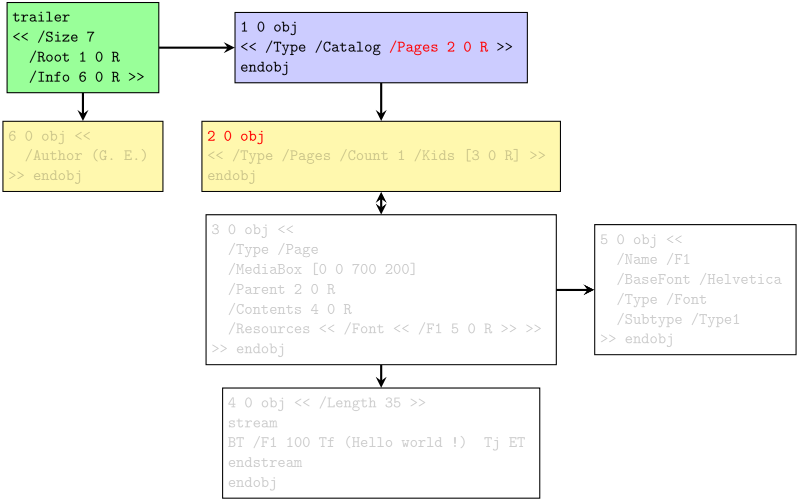
```
3 0 obj <<
  /Type /Page
  /MediaBox [0 0 700 200]
  /Parent 2 0 R
  /Contents 4 0 R
  /Resources << /Font << /F1 5 0 R >> >>
>> endobj
```

```
5 0 obj <<
  /Name /F1
  /BaseFont /Helvetica
  /Type /Font
  /Subtype /Type1
>> endobj
```

```
4 0 obj << /Length 35 >>
stream
BT /F1 100 Tf (Hello world !)  Tj ET
endstream
endobj
```

Constraint propagation.

Type checking



```
trailer
<< /Size 7
  /Root 1 0 R
  /Info 6 0 R >>
```

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

```
6 0 obj <<
  /Author (G. E.)
>> endobj
```

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

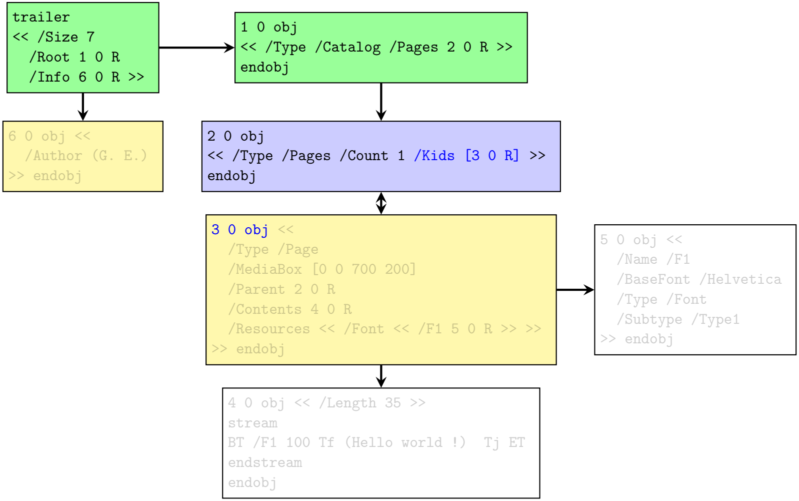
```
3 0 obj <<
  /Type /Page
  /MediaBox [0 0 700 200]
  /Parent 2 0 R
  /Contents 4 0 R
  /Resources << /Font << /F1 5 0 R >> >>
>> endobj
```

```
5 0 obj <<
  /Name /F1
  /BaseFont /Helvetica
  /Type /Font
  /Subtype /Type1
>> endobj
```

```
4 0 obj << /Length 35 >>
stream
BT /F1 100 Tf (Hello world !)  Tj ET
endstream
endobj
```

Constraint propagation.

Type checking



```
trailer
<< /Size 7
  /Root 1 0 R
  /Info 6 0 R >>
```

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

```
6 0 obj <<
  /Author (G. E.)
>> endobj
```

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

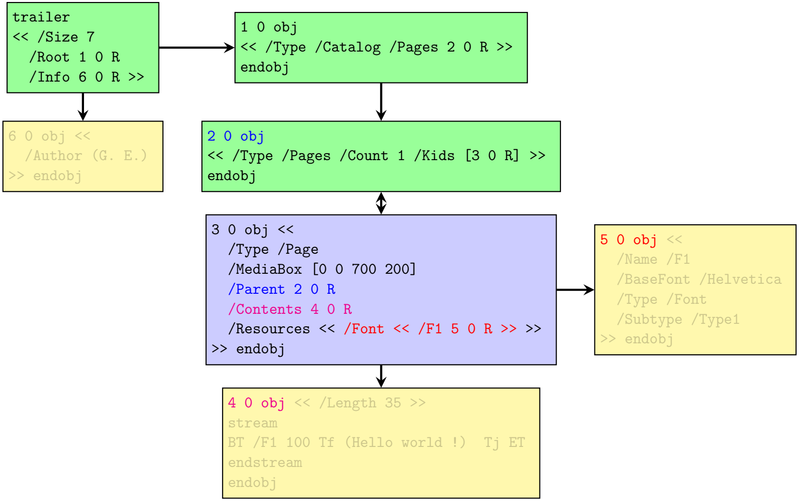
```
3 0 obj <<
  /Type /Page
  /MediaBox [0 0 700 200]
  /Parent 2 0 R
  /Contents 4 0 R
  /Resources << /Font << /F1 5 0 R >> >>
>> endobj
```

```
5 0 obj <<
  /Name /F1
  /BaseFont /Helvetica
  /Type /Font
  /Subtype /Type1
>> endobj
```

```
4 0 obj << /Length 35 >>
stream
BT /F1 100 Tf (Hello world !)  Tj ET
endstream
endobj
```

Constraint propagation.

Type checking



```
trailer
<< /Size 7
  /Root 1 0 R
  /Info 6 0 R >>
```

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

```
6 0 obj <<
  /Author (G. E.)
>> endobj
```

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

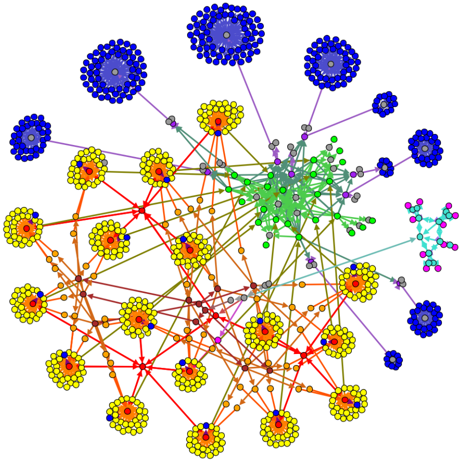
```
3 0 obj <<
  /Type /Page
  /MediaBox [0 0 700 200]
  /Parent 2 0 R
  /Contents 4 0 R
  /Resources << /Font << /F1 5 0 R >> >>
>> endobj
```

```
5 0 obj <<
  /Name /F1
  /BaseFont /Helvetica
  /Type /Font
  /Subtype /Type1
>> endobj
```

```
4 0 obj << /Length 35 >>
stream
BT /F1 100 Tf (Hello world !)  Tj ET
endstream
endobj
```

Constraint propagation.

Type checking



🟪	action
🟥	page
🟧	destination
🟨	annotation
🟩	resource
🟦	outline
🟦	content stream
🟪	font
🟫	name tree
⬜	other

Types of a 17-page document.

More complex verifications

At a higher level:

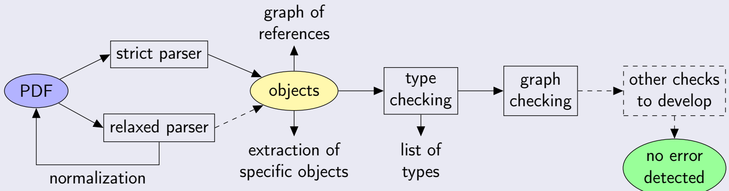
- Verification of tree structures (page tree, outlines, etc.).
- Other verifications easily integrable in the future (fonts, images, existing analyses, etc.).

Table of contents

Implementation

Implementation in OCaml from the PDF specification[4].

Validation workflow.



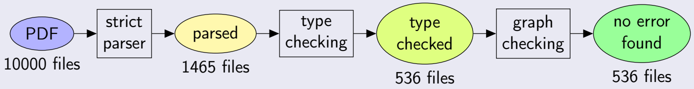
[4]https://www.adobe.com/devnet/pdf/pdf_reference.html

Real-world files

10K files collected from random queries on a web search engine.

Some files are directly accepted.

Direct validation.

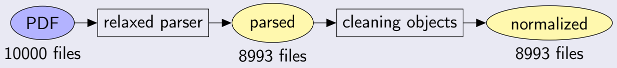


Normalization

Many files do not pass the first stage... But they can be normalized beforehand.

The relaxed parser supports common structures: incremental updates, object streams, etc.

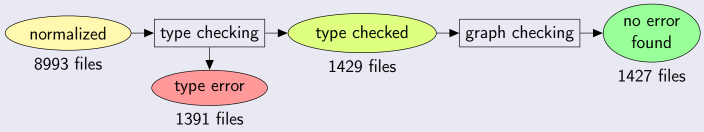
Normalization.



Some files were not normalized: encryption, unrecoverable syntax errors, etc.

Normalization

Validation after normalization.



Our type-checker detected typos:

- `/Blackls1` instead of `/BlackIs1`,
- `/XObjcect` instead of `/XObject`.

We identified incorrect tree structures in the wild.

Future work

What remains to be done:

- Complete the set of types.
- Check compression filters.
- Check graphic content.
- Check fonts, images, etc.

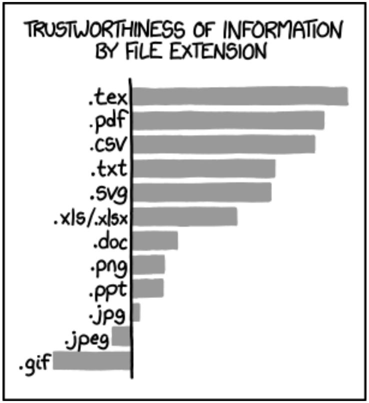
Conclusion

Summary of our contributions:

- We identified novel issues in PDF parsers.
- We proposed and formalized a simplified syntax for PDF.
- We implemented Caradoc to parse and validate PDF files.

Project page: `https://github.com/ANSSI-FR/caradoc`

Questions ?



https://xkcd.com/1301/