Caradoc: une boîte à outils pour décortiquer et analyser sereinement les fichiers PDF

SSTIC 2017

Guillaume Endignoux[1,2], Olivier Levillain[3]
@gendignoux, @pictyeye

[1]EPFL, [2]Kudelski Security, [3]ANSSI

7 juin 2017

Portable Document Format ?

Format très courant, mais nombreux problèmes de sécurité :

- 500+ vulnerabilitiés dans Adobe Reader (depuis 1999).
- Divergences entre implémentations.
- Syntaxe facilite le polymorphisme, e.g. PoC||GTFO (PDF+ZIP, PDF+JPEG...).
- Collisions SHA-1...

Portable Document Format ?

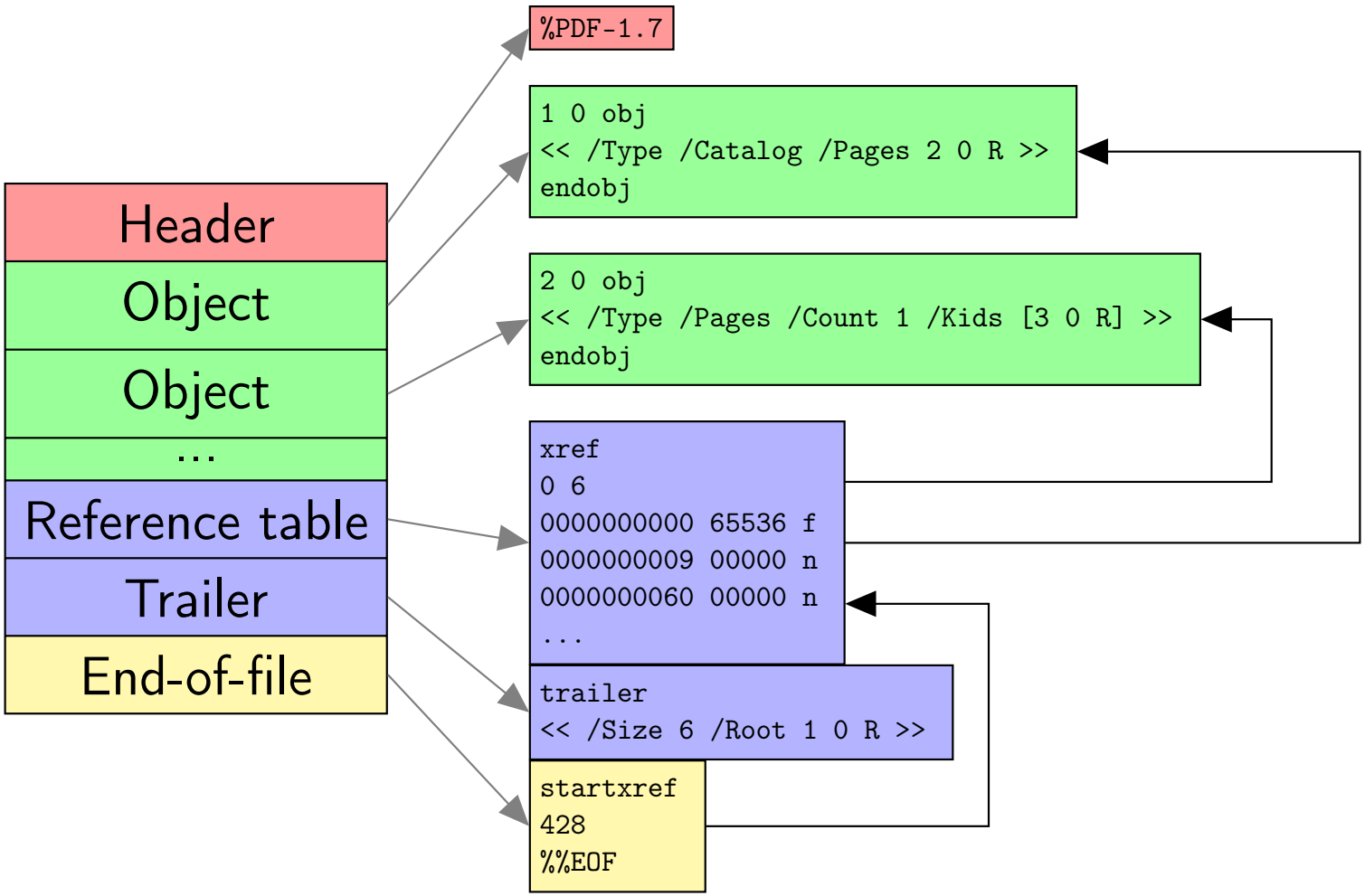
Format très courant, mais nombreux problèmes de sécurité :

- 500+ vulnerabilitiés dans Adobe Reader (depuis 1999).
- Divergences entre implémentations.
- Syntaxe facilite le polymorphisme, e.g. PoC||GTFO (PDF+ZIP, PDF+JPEG...).
- Collisions SHA-1...

Caradoc :

- Projet commencé à l'ANSSI en 2015.
- Idée de base : l'étape de *parsing* est importante.
- Article académique au LangSec Workshop 2016.

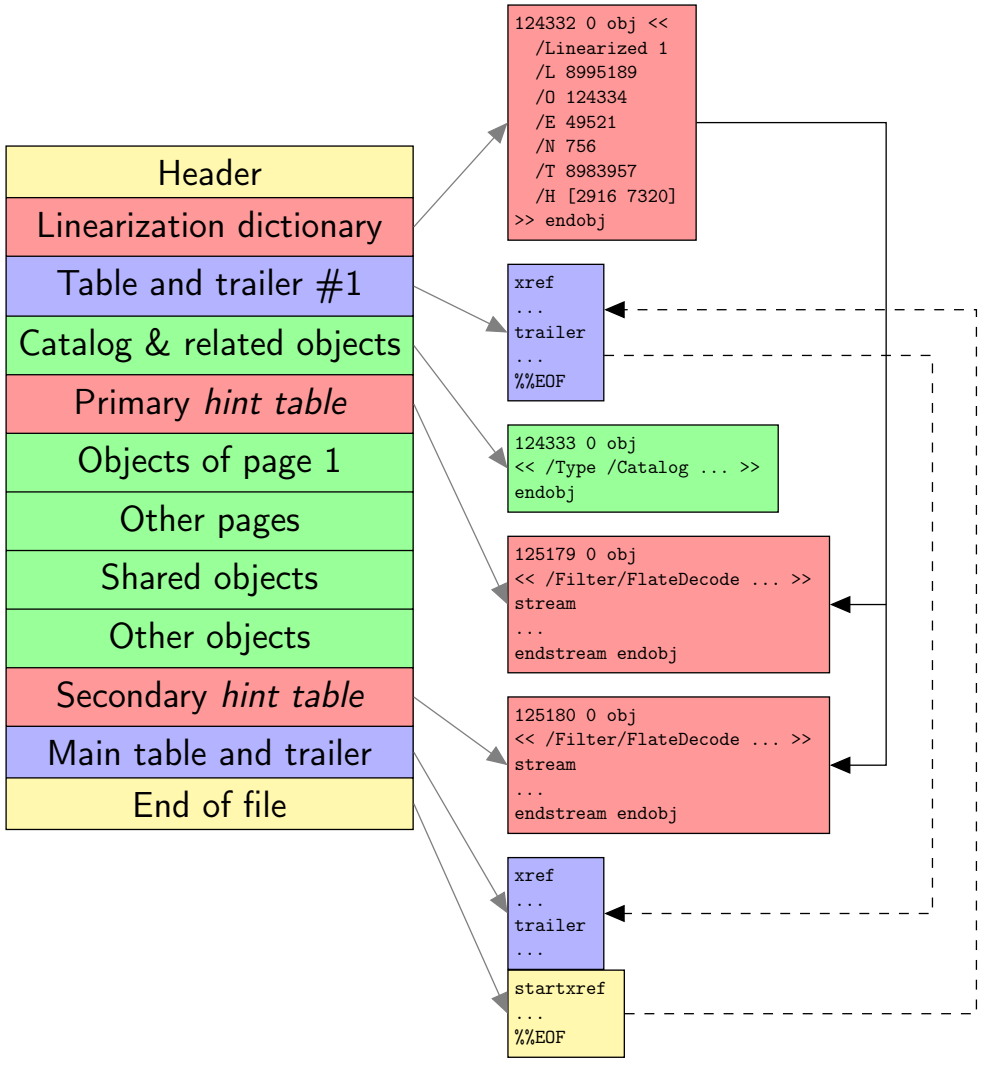
Syntaxe PDF 101



Structure d'un fichier PDF simple.

Structures plus complexes :

- mises-à-jour incrementales,
- objets compressés,
- linéarisation.



Fichier linéarisé.

Démonstration

Démo : quelques fichiers à la « limite » de la norme.

- En jouant seulement sur la structure, pas de JavaScript !

Exemples d'attaques (plusieurs bugs remontés) :

- Évasion : divergences lecteur PDF vs. détecteur de malware.
- Attaques sur le *parser* : déni de service, crash.

Validation avec Caradoc

Caradoc = implémentation en OCaml directement depuis la norme PDF.

Un *parser* robuste :

- Langage OCaml : typage fort, *pattern matching* exhaustif.
- Nombreux tests unitaires.
- Tests sur 10K fichiers réels.

Validation avec Caradoc

Caradoc = implémentation en OCaml directement depuis la norme PDF.

Un *parser* robuste :

- Langage OCaml : typage fort, *pattern matching* exhaustif.
- Nombreux tests unitaires.
- Tests sur 10K fichiers réels.

Caradoc valide :

- Syntaxe du fichier.
- Cohérence des objets (typage).
- Graphe des références (arbre des pages...).
- Instructions graphiques (syntaxe).

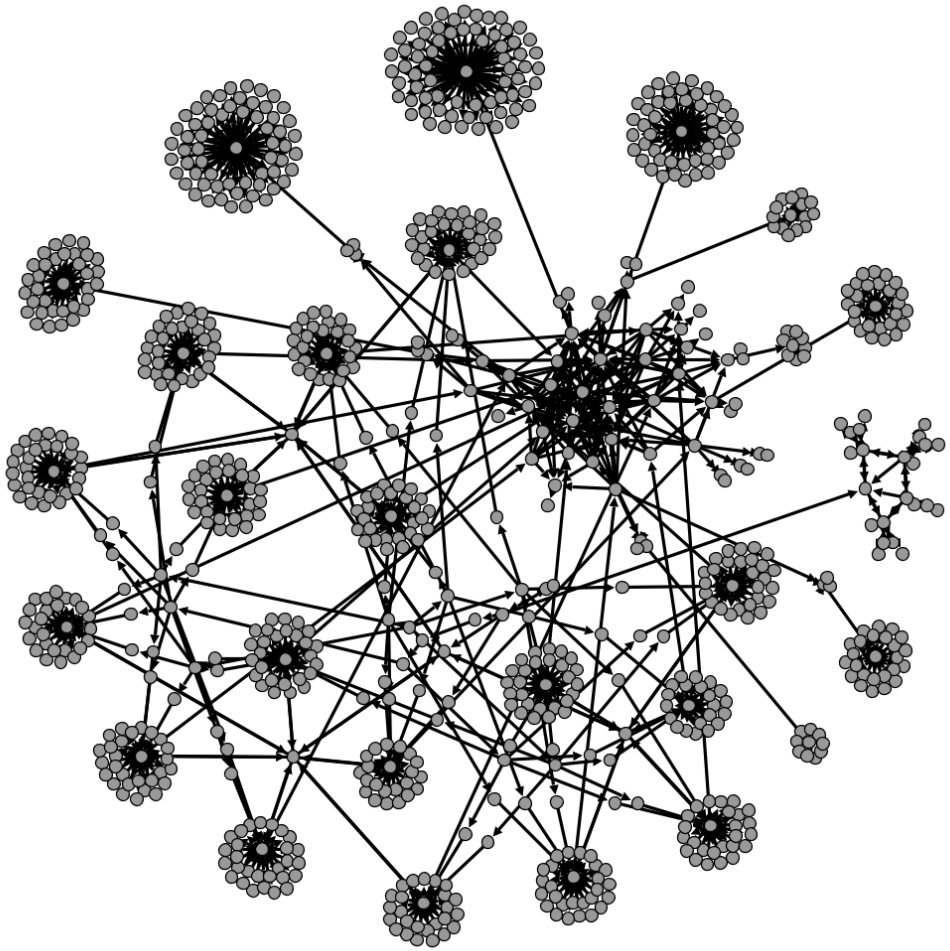
Restriction de syntaxe

Au niveau de la **syntaxe**, garantir extraction d'objets sans ambiguité :

- Grammaire formalisée (BNF).
- Restrictions sur la structure.
- Rejet des fichiers « corrompus ».

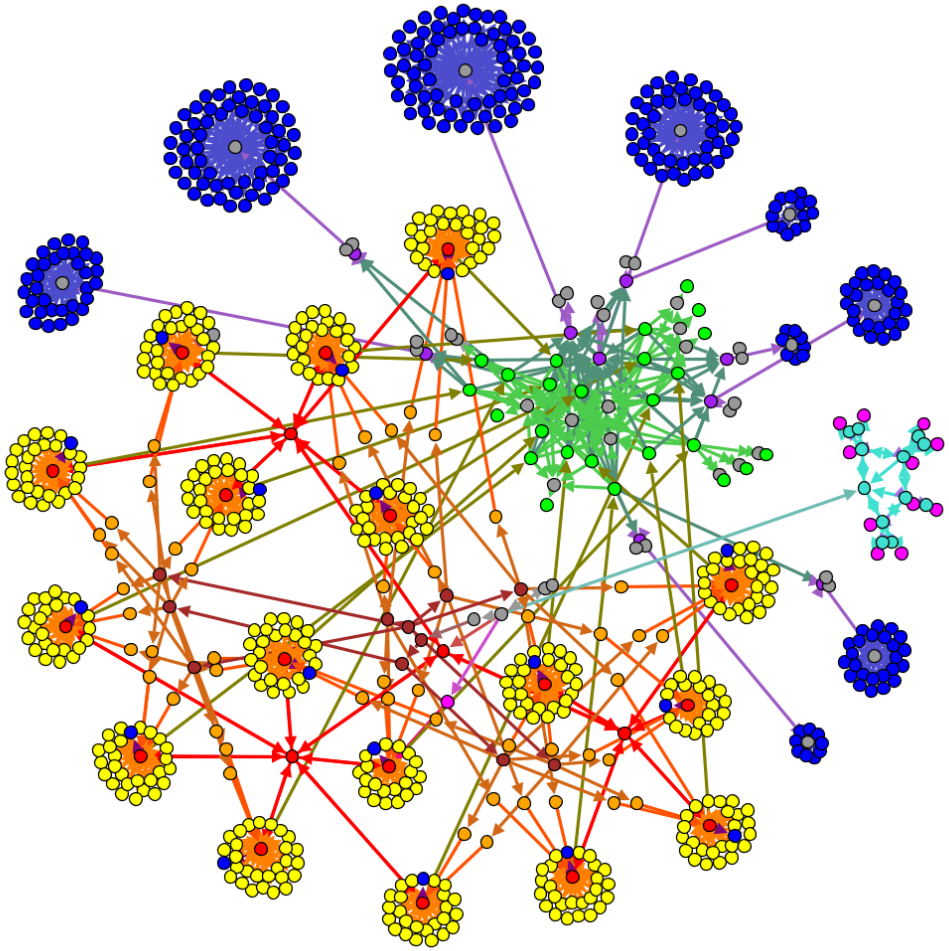
> *When a conforming reader reads a PDF file with a damaged or missing cross-reference table, it **may attempt** to rebuild the table by scanning all the objects in the file.*
>
> — ISO 32000-1:2008, annex C.2

Typage



Document de 17 pages (environ 1000 objets).

Typage

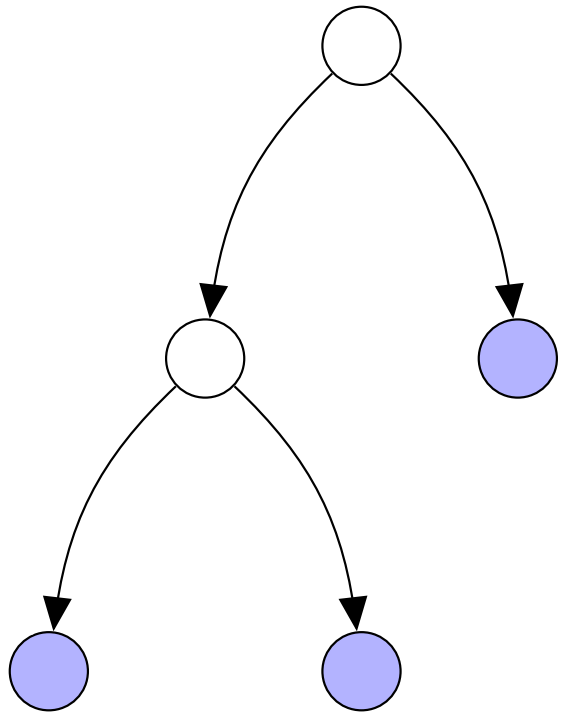


	action
	page
	destination
	annotation
	resource
	outline
	content stream
	font
	name tree
	other

Types d'un document de 17 pages.

Structure de graphe

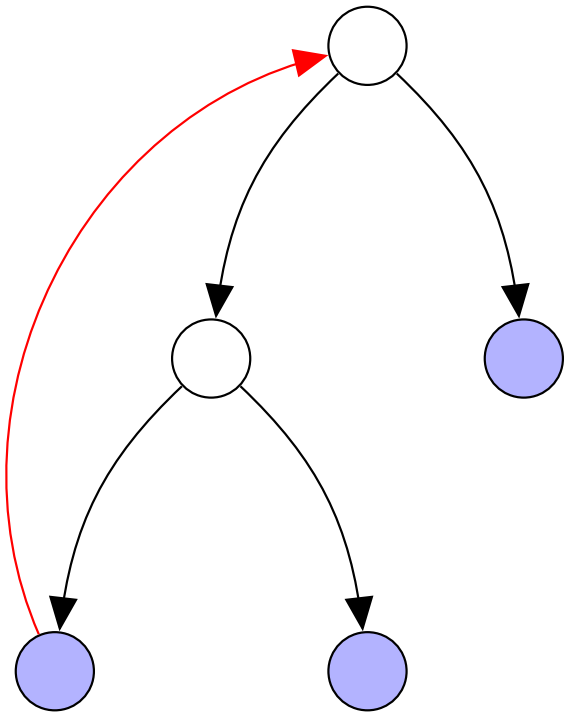
Références entre objets forment un graphe $\Rightarrow$ vérification des arbres.



Arbre des pages (feuilles sont les pages).

Structure de graphe

Références entre objets forment un graphe ⇒ vérification des arbres.



Structure recursive (cf. démo crash dans certains lecteurs).

Prise en main

Code source sur GitHub

```
$ git clone https://github.com/ANSSI-FR/caradoc
```

Statistiques générales

```
$ caradoc stats file.pdf
```

Validation stricte

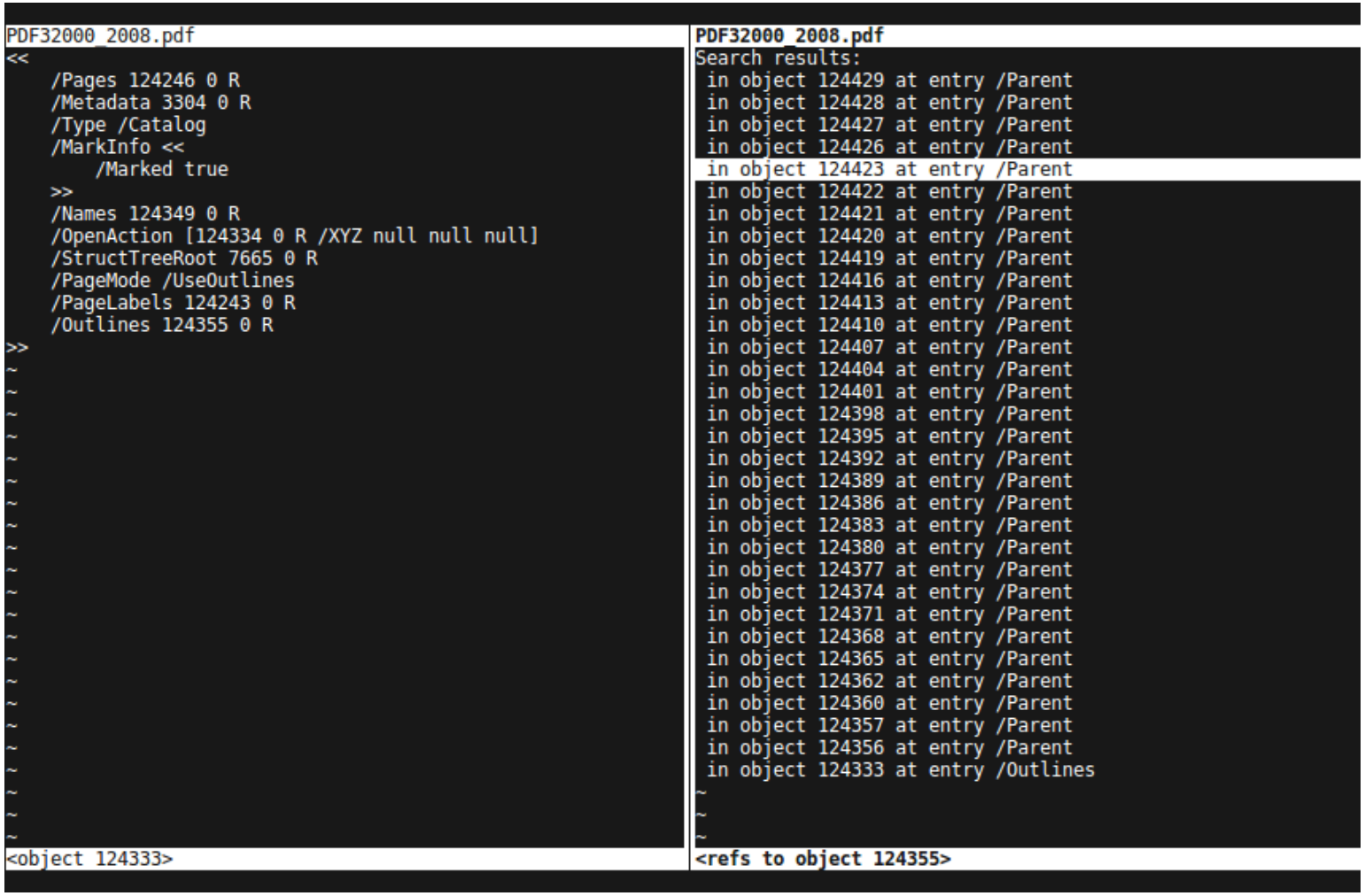
```
$ caradoc stats --strict file.pdf
```

Autres : table des objets, extraction d'objet, décodage de flux, recherche...

Interface interactive

Interface interactive en console (style ncurses)

```
$ caradoc ui file.pdf
```



Fichiers réels : enseignements

Évaluation : 10K fichiers aléatoires depuis un moteur de recherche.

Le *parser* strict rejette des fonctionnalités fréquentes :

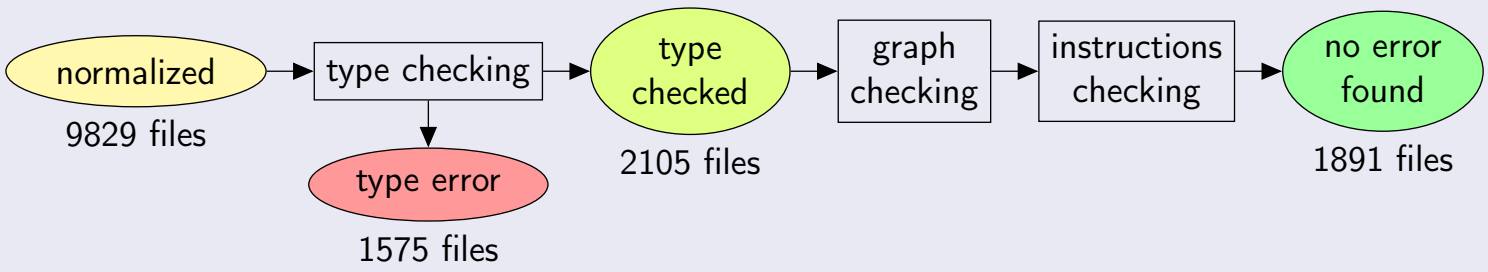
Fonctionnalité	% de fichiers
mises-à-jour incrémentales	65%
objets compressés	37%
objets non référencés	28%
chiffrement	5%

⇒ Solution : **normaliser** avec le *parser* souple au préalable !

```
$ caradoc cleanup --out file.clean.pdf file.pdf
```

Fichiers réels : enseignements

Validation après normalisation.



Typage détecte des fautes de frappe :

- `/Blackls1` au lieu de `/BlackIs1`,
- `/XObjcect` au lieu de `/XObject`.

Quelques structures d'arbre incorrectes détectées dans la nature.

Conclusion

Caradoc :

- *parser* robuste,
- base saine pour des analyses fiables,
- prise en charge des cas les plus courants.

Limitations :

- toute la norme n'est pas encore implémentée (env. 700 pages),
- *parser* « souple » accepte peu d'exceptions à la norme,
- analyses fiables de haut niveau restent à écrire/adapter.

Questions ?

- Caradoc : github.com/ANSSI-FR/caradoc
- LangSec Workshop : spw16.langsec.org/papers.html
- PDF cheat sheet : github.com/gendx/pdf-cheat-sheets
- Corpus de PDF « bizarres » : github.com/gendx/pdf-corpus

Quelques articles de blog :
https://gendignoux.com/blog/tags.html#pdf

Twitter : @gendignoux
GitHub : @gendx

Remerciements : Jean-Yves Migeon

Installation

Code source sur GitHub

```
$ git clone https://github.com/ANSSI-FR/caradoc
```

Dépendences

```
# apt-get install ocaml opam
# apt-get install zlib1g-dev libgmp-dev pkg-config m4
$ opam init
$ opam install ocamlfind cryptokit ounit menhir curses
```

Compilation

```
$ make
```

Validation

Caradoc est un validateur de fichiers PDF.

Statistiques générales

```
$ caradoc stats file.pdf
```

Validation stricte

```
$ caradoc stats --strict file.pdf
```

Normalisation

```
$ caradoc cleanup --out file.clean.pdf file.pdf
```

Verbose/debug, attention beaucoup de données !

```
$ caradoc <commande> --verbose --debug
```

Export de données

Caradoc est aussi utile comme *parser* pour extraire diverses données.

Métadonnées

```
$ caradoc stats file.pdf
```

Données générales : *trailer*, table d'objets

```
$ caradoc trailer file.pdf
$ caradoc xref file.pdf
```

Extraction d'un objet, décodage de flux

```
$ caradoc object --num 123 file.pdf
$ caradoc object --num 123 --decoded-stream stream.bin file.pdf
```

Export de données

Graphe des références

```
$ caradoc extract --dot graphe.dot file.pdf
```

Types des objets

```
$ caradoc extract --types types.txt file.pdf
```