Fuzzer

This README file describe how to use our tools, including TypeOracle, Cooper+TypeOracle and Favocado+TypeOracle, to fuzz PDF Readers.

TypeOracle

This fuzzer uses the type information reasoned by TypeOracle to fuzz PDF Readers

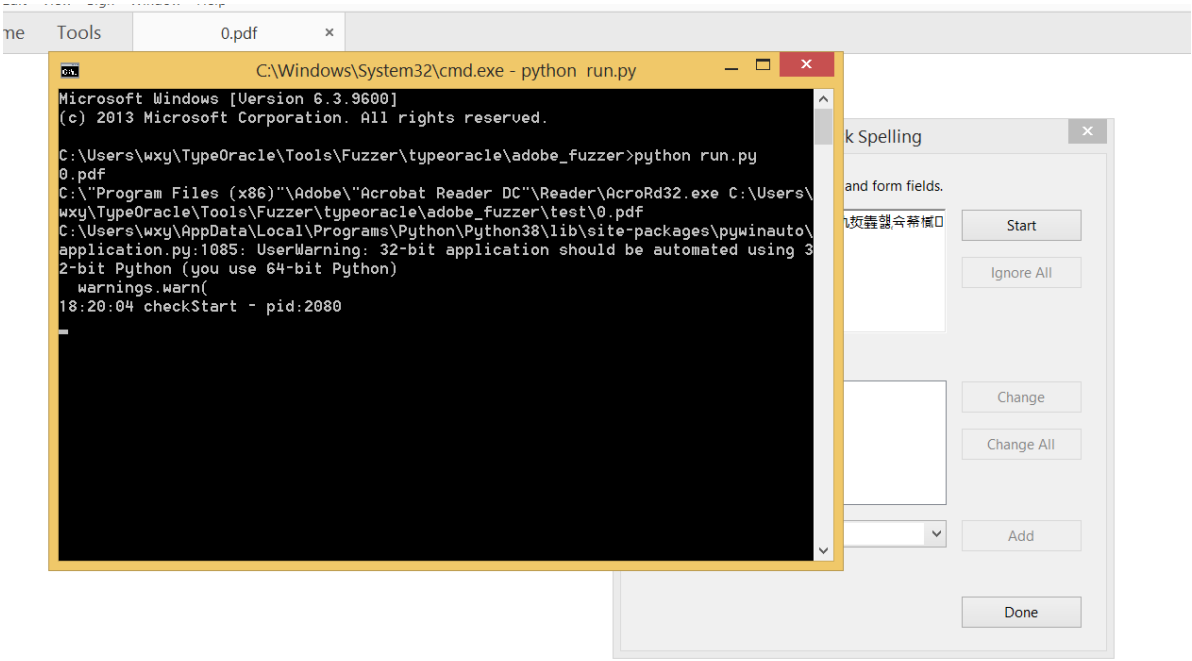
folder structure

```
- config
  - blacklist.txt (binding calls that don't test)
  - constant.txt (some constant variables)
  - delist.txt (binding calls that test less)
  - objlst.txt (list of built-in objects)
  - strlst.json (some special strings)
  - template.pdf (PDF template)

- data (the folder stores the type information of binding calls)

- save (save the test PDFs that trigger unormal behavior)

- test (save all test PDFs)

- generate.py (generate Javascript codes)

- monitor.py (monitor and interact with PDF Reader)

- mPDF.py (embeded Javascript codes to PDFs)

- pattern.py (parse type information file of every binding call)

- rand.py (generate random value)

- relationship.json (some binding calls that have relationships)

- run.py
```

how to reproduce

1. execute run.py to start fuzz

```
python run.py
```



2. results are stored in `save` folder

```
crash: PDFs make PDF Reader crash
halt: PDFs make PDF Reader exit without any notice
hang: PDFs make PDF Reader hang
stop: PDFs make PDF Readerstop execute Javascript codes
error: PDFs make our fuzz make mistakes
```

Cooper+TypeOracle

The fuzzer use Cooper integrated with TypeOracle's type information to generate test PDFs, and use our monitor tool to monitor and interact with PDF Reader. The source code of Cooper can be found at https://github.com/TCA-ISCAS/Cooper.

folder structure

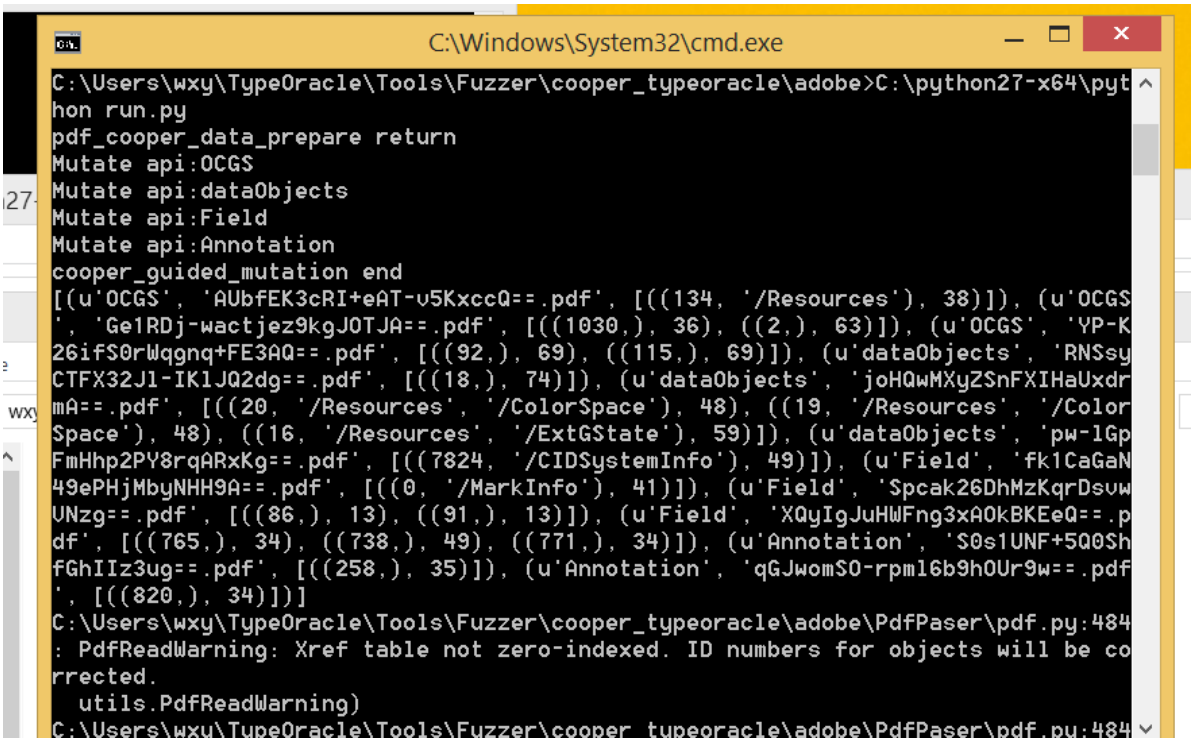
```
- config

- Cooper

- data (the folder stores the type information of binding calls)

- PdfData (result that parse PDF's of Cooper)

- PdfJsTemplate (Cooper's original Javascript generation template)

- PdfPaser (Cooper's utility to parse PDF's objects)

- PdfSamples (sample provided by Cooper's authors)

- save (save the test PDFs that trigger unormal behavior)
```

```
- test (save all test PDFs)

- generate.py (generate Javascript codes)

- grammar.py

- monitor.py (monitor and interact with PDF Reader)

- myPDF.py (embeded Javascript codes to PDFs)

- pattern.py (parse type information file of every binding call)

- PdfExecutionLog.py

- PdfMutation.py

- PdfSolution.py

- rand.py (generate random value)

- relationship.json (some binding calls that have relationships)

- run.py
```

how to reproduce

1. use python2 to execute run.py to launch fuzzer

```
C:\python27-x64\python run.py
```



2. results are stored in `save` folder

```
crash: PDFs make PDF Reader crash
halt: PDFs make PDF Reader exit without any notice
hang: PDFs make PDF Reader hang
stop: PDFs make PDF Readerstop execute Javascript codes
error: PDFs make our fuzz make mistakes
```

Favocado+TypeOracle

The fuzzer use Favocado integrated with TypeOracle's type information to generate test PDFs, and use our monitor tool to monitor and interact with PDF Reader. The source code of Favocado can be found at https://github.com/favocado/Favocado.

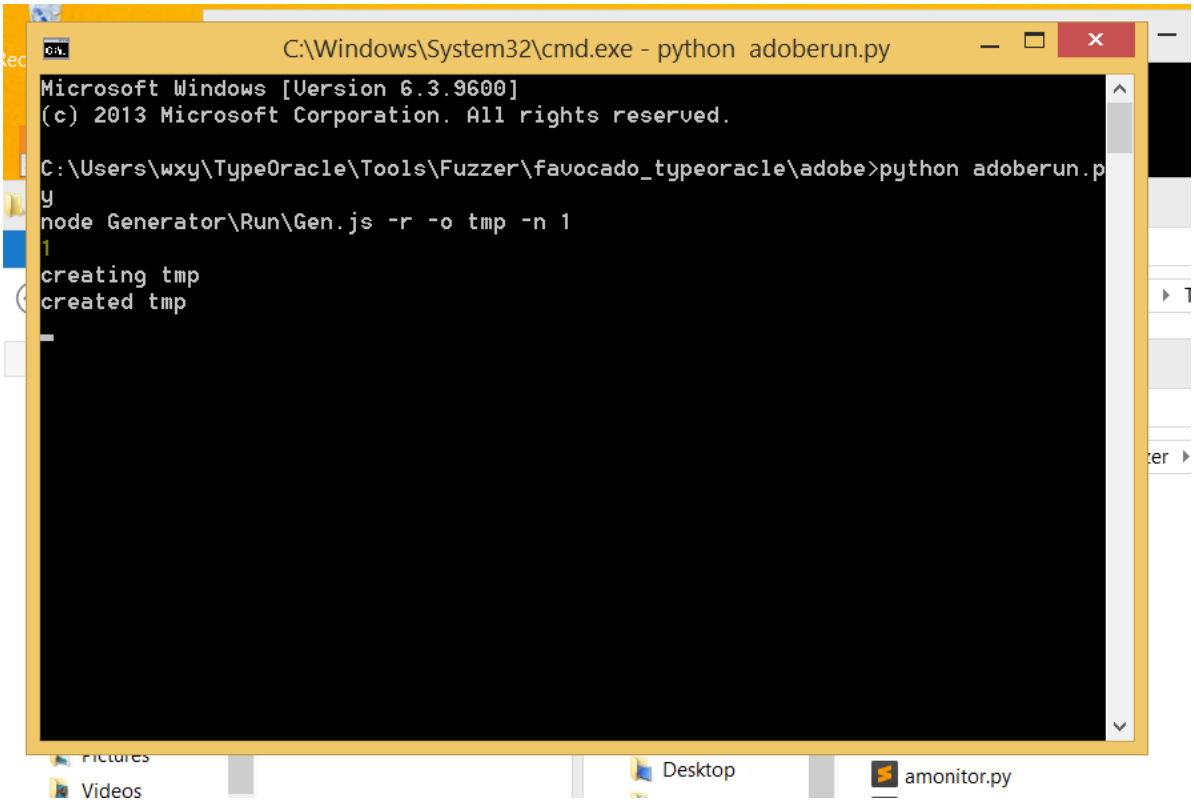
folder structure

```
- Generator (folders that store the major part of favocado, integrating with
our's type information)

- save (save the test PDFs that trigger unormal behavior)

- test (save all test PDFs)

- tmp (save the javascript codes)

- adoberun.py/foxitrun.py (launch the fuzzer)

- amonitor.py/fmonitor.py (monitor and interact with PDF Reader)

- generate.py (execute the command to generate javascript codes)
```

how to reproduce

1. execute adoberun.py/foxitrun.py

```
python adoberun.py
python foxitrun.py
```



2. results are stored in `save` folder

```
crash: PDFs make PDF Reader crash
halt: PDFs make PDF Reader exit without any notice
hang: PDFs make PDF Reader hang
stop: PDFs make PDF Readerstop execute Javascript codes
error: PDFs make our fuzz make mistakes
```