RQ3: Fuzzing Performance

This experiment is in the `Section V-D Fuzzing Performance` of our paper. The goal of this experiment is to measure the accuracy of the type information infered by TYPEORACLE. The result is shown in the `Fig. 7: Fuzzing performance of type guidance`, `Fig. 8: Fuzzing performance of coverage guidance` and `Fig. 9: Comparison and integration with state-of-art fuzzers` in our paper. We conduct three groups of experiments to evaluate the fuzzing performance of different configurations on both Adobe Reader and Foxit Reader. Each experiment is run for five times, and each time lasts for 48 hours. Figure 7, Figure 8 and Figure 9 present the experiment results, in which X-axis represents time, Y-axis represents the number of covered instructions, the lines represent mean of the five runs. Note that we do not count the instructions covered by base PDF files themselves, and only concentrate on the instructions covered by binding calls.

folder structure

We conduct three groups of experiments to evaluate the fuzzing performance of different configurations on both Adobe Reader and Foxit Reader.

Group 1: Type Guidance. This group of experiments is to assess the effect of using type information as fuzzing guidance. Figure 7a presents the experiment results on Adobe Reader, from which we have two conclusions. First, undocumented binding calls incur 43.09% coverage increment (comparing the gray dashed line with the gray dotted line). Second, type information provided by TYPEORACLE incurs 40.38% coverage increment than random testing (comparing the black solid line with the gray dashed line) and 34.88% coverage increment than that provided by Adobe's API documentation (comparing the black solid line with the black dash-dot line). The experiment results on Foxit Reader are shown in Figure 7b. As we can see, using the type information extracted by TYPEORACLE incurs 17.09% and 47.28% coverage increment compared with random testing and using Adobe's API documentation respectively.

Group 2: Coverage Guidance. We also assess the effect of using coverage feedback as fuzzing guidance. The fuzzing process is refracted to leverage the coverage feedback. Specifically, if a test case covers new paths, it will be preserved for further mutations in the subsequent iterations. In this group of experiments, we use the same base PDF file as in Group 1. The experiment results indicate that coverage guidance has limited impact on improving fuzzing performance. On Adobe Reader, it increases the coverage by 4.27% for random testing and by 1.36% for fuzzing with TYPEORACLE (Figure 8a). On Foxit Reader, it increases the coverage by 3.86% for random testing and by 1.00% for fuzzing with TYPEORACLE (Figure 8b).

Group 3: Comparison and Integration with State-of-Art Fuzzers. Gramatron is a state-of-art grammar-aware fuzzer that uses grammar automatons and aggressive mutation operators to synthesize complex syntactically-valid test cases. We extend Gramatron (denoted as Gramatron+) to support binding call invocations. We use Gramatron and Gramatron+ for the generation of scripts injected into the base PDF file and treat them as standalone baselines. By comparison, TYPEORACLE achieves 309.18% and 72.56% more coverage than Gramatron, and 31.56% and 12.65% more coverage than Gramatron+ (Figures 9a and 9b). Favocado and Cooper are two state-of-art fuzzers that consider binding calls during fuzzing. Favocado mixes binding calls with various Javascript templates. Cooper simultaneously mutates both binding calls and PDF page elements. They extract parameter types from Adobe's API documentation, hence cannot handle

undocumented and inconsistent binding calls. TYPEORACLE can be complementary with them by providing more complete and accurate parameter type information. Compared with the original Favocado, the integration of Favocado and TYPEORACLE (denoted as Favocado+TYPEORACLE) increases coverage by 10.19% on Adobe Reader and by 17.30% on Foxit Reader (Figures 9c and 9d). Compared with the original Cooper, the integration of Cooper and TYPEORACLE (denoted as Cooper+TYPEORACLE) increases coverage by 21.37% on Adobe Reader and by 55.12% on Foxit Reader (Figures 9e and 9f).

The following is the folder structure.

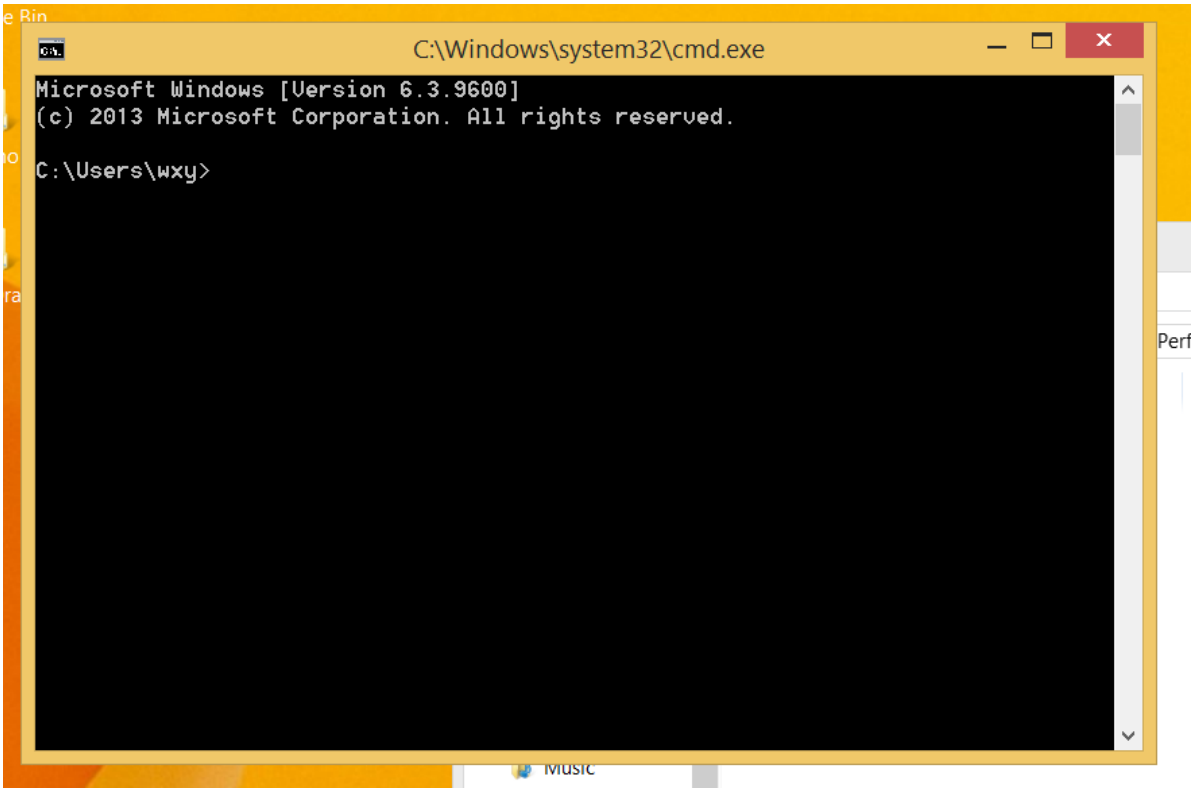
```
Group1(Figure 7(a), Figure 7(b))
  - adobe reader (Figure 7(a))
    - data/:
      - document: documented binding calls + random testing
      - random: all binding calls + random testing
      - error_message: all binding calls + error message
      - path_len: all binding calls + path length
      - adobe_manual: documented binding calls + Adobe Manual
      - typeoracle: all binding calls + TypeOracle

    - utility/:
      - draw_all.py: the script file to parse coverage information

    - result/:
      - adobe_type.pdf: Figure 7(a)


  - foxit reader (Figure 7(b))
    - data/:
      - adobe_manual: fuzzing all binding calls using Adobe Manual
      - random: randomly fuzzing all (documented+undocumented) binding calls
      - error_message: all binding calls + error message
      - path_len: all binding calls + path length
      - typeoracle_foxit: fuzzing all binding calls using TypeOracle of Foxit

    - utility/:
      - draw_all.py: the script file to parse coverage information

    - result/:
      - foxit_type.pdf: Figure 7(b)

Group2(Figure 8(a), Figure 8(b))
  - adobe reader (Figure 8(a))
    - data/:
      - random: randomly fuzzing all (documented+undocumented) binding calls
      - coverage_random: randomly fuzzing all (documented+undocumented) binding
calls + coverage guidance
      - typeoracle: fuzzing all binding calls using TypeOracle
      - coverage_typeoracle: fuzzing all binding calls using TypeOracle +
coverage guidance

    - utility/:
      - draw_all.py: the script file to parse coverage information

    - result/:
```

```
      - adobe_cov.pdf: Figure 8(a)

  - foxit reader (Figure 8(b))
    - data/:
      - random: randomly fuzzing all (documented+undocumented) binding calls
      - coverage_random: randomly fuzzing all (documented+undocumented) binding
calls + coverage guidance
      - typeoracle_foxit: fuzzing all binding calls using TypeOracle
      - coverage_typeoracle: fuzzing all binding calls using TypeOracle +
coverage guidance

    - utility/:
      - draw_all.py: the script file to parse coverage information

    - result/:
      - foxit_cov.pdf: Figure 8(b)

Group3(Gramatron) (Figure 9(a), Figure 9(b))
  - adobe reader (Figure 9(a))
    - data/:
      - gramatron: fuzzing using naive Gramatron
      - gramatron_plus: fuzzing all binding calls using Gramatron
      - typeoracle: fuzzing all binding calls using TypeOracle

    - utility/:
      - draw_all.py: the script file to parse coverage information

    - result/:
      - adobe_gramatron.pdf: Figure 9(a)

  - foxit reader (Figure 9(b))
    - data/:
      - gramatron: fuzzing using naive Gramatron
      - gramatron_plus: fuzzing all binding calls using Gramatron
      - typeoracle: fuzzing all binding calls using TypeOracle

    - utility/:
      - draw_all.py: the script file to parse coverage information

    - result/:
      - foxit_gramatron.pdf: Figure 9(b)

Group3(Favocado) (Figure 9(c), Figure 9(d))
  - adobe reader (Figure 9(c))
    - data/:
      - favocado: fuzzing all binding calls using Favocado
      - favocado_typeoracle: fuzzing all binding calls using combination of
TypeOracle and Favocado

    - utility/:
      - draw_all.py: the script file to parse coverage information

    - result/:
      - adobe_favocado.pdf: Figure 9(c)
```

```
  - foxit reader (Figure 9(d))
    - data/:
      - favocado: fuzzing all binding calls using Favocado
      - favocado_typeoracle: fuzzing all binding calls using combination of
TypeOracle and Favocado

    - utility/:
      - draw_all.py: the script file to parse coverage information

    - result/:
      - foxit_favocado.pdf: Figure 9(d)

Group3(Cooper) (Figure 9(e), Figure 9(f))
  - adobe reader (Figure 9(e))
    - data/:
      - cooper: fuzzing all binding calls using Cooper
      - cooper_typeoracle: fuzzing all binding calls using combination of
TypeOracle and Cooper

    - utility/:
      - draw_all.py: the script file to parse coverage information

    - result/:
      - adobe_cooper.pdf: Figure 9(e)


  - foxit reader (Figure 9(f))
    - data/:
      - cooper: fuzzing all binding calls using Cooper
      - cooper_typeoracle: fuzzing all binding calls using combination of
TypeOracle and Cooper

    - utility/:
      - draw_all.py: the script file to parse coverage information

    - result/:
      - foxit_cooper.pdf: Figure 9(f)
```

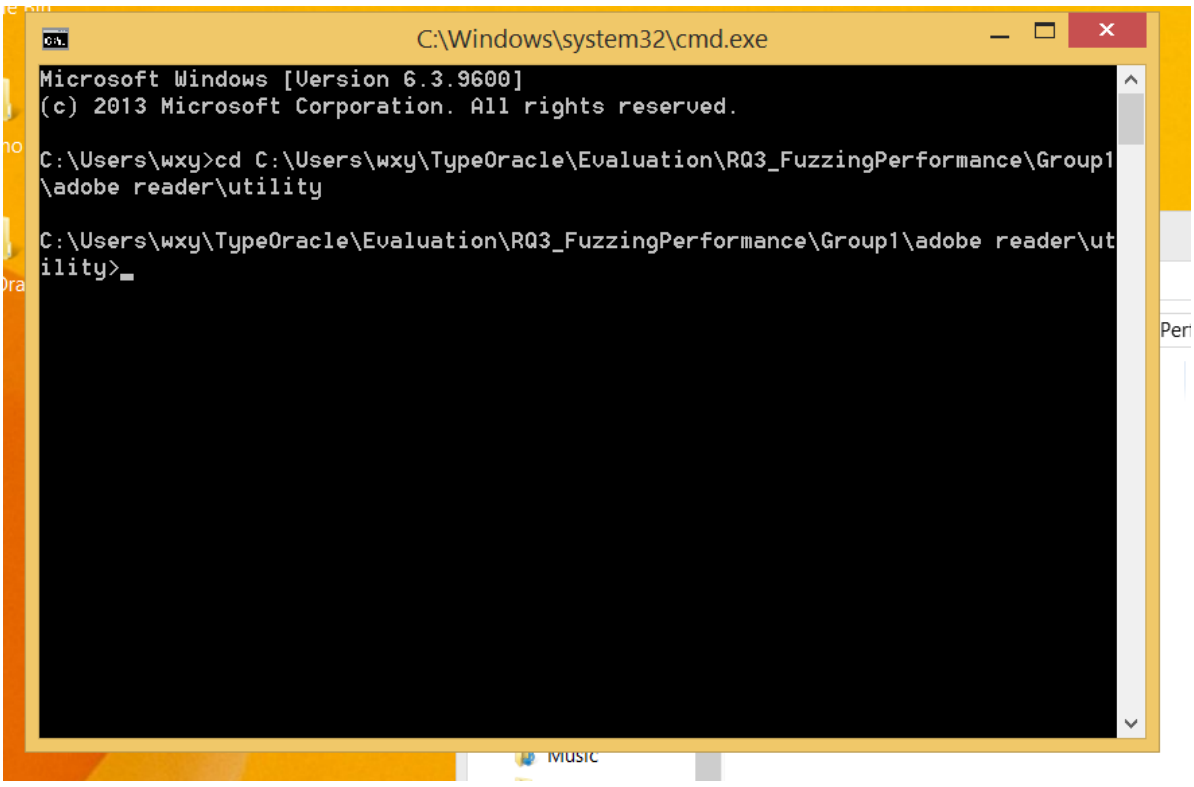
how to reproduce

1. open cmd.exe



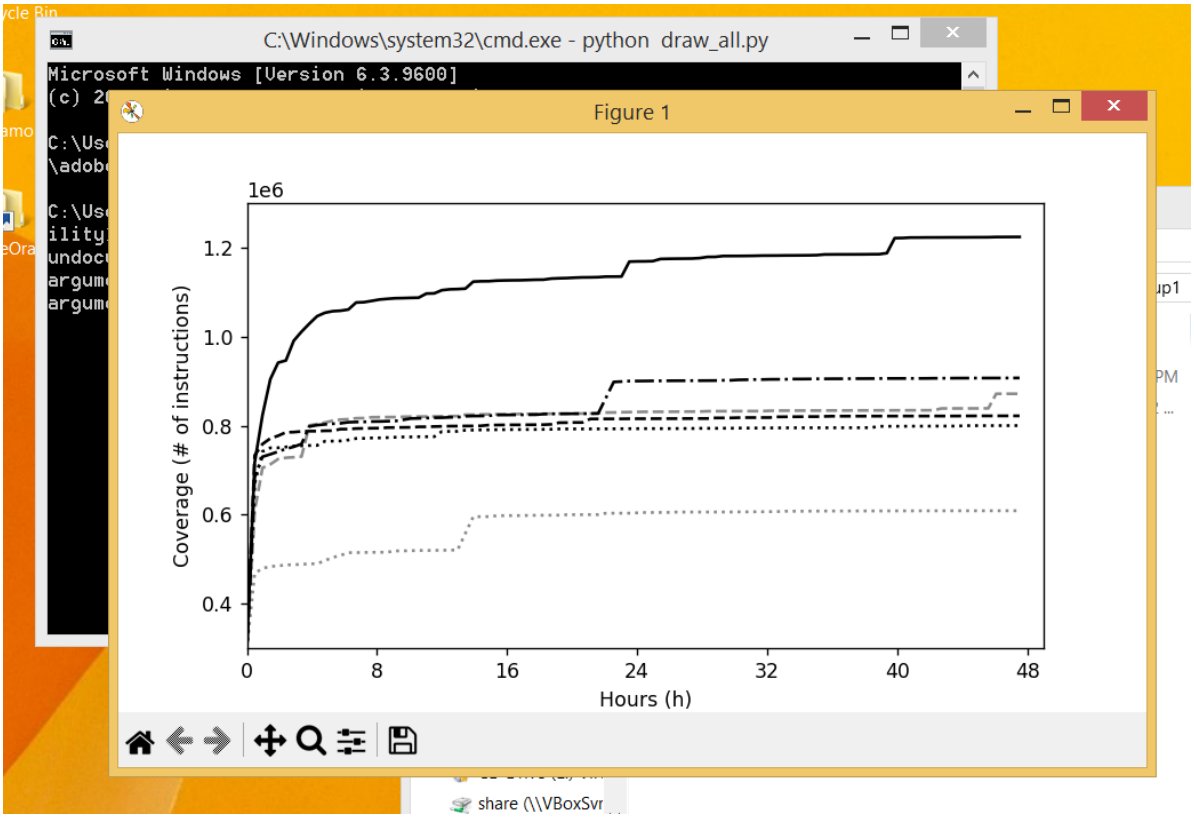
2. cd to the utility folder (we use Group1's adobe reader as an example)

```
cd C:\Users\wxy\TypeOracle\Evaluation\RQ3_FuzzingPerformance\Group1\adobe
reader\utility
```



3. execute the following command and you will see the result.

```
python draw_all.py
```


C:\Windows\system32\cmd.exe - python draw_all.py
Microsoft Windows [Version 6.3.9600]
Figure 1
1e6
1.2
1.0
0.8
0.6
0.4
Coverage (# of instructions)
0
8
16
24
32
40
48
Hours (h)