Operand-Variation-Oriented Differential Analysis for Fuzzing Binding Calls in PDF Readers (Artifact Evaluation)

1 Basic Information

Title: Operand-Variation-Oriented Differential Analysis for Fuzzing Binding Calls in PDF Readers (ID: 810)
Authors: Suyue Guo, Xinyu Wan, Wei You, Bin Liang, Wenchang Shi, Yiwei Zhang, Jianjun Huang, Jian Zhang
Affiliations: Renmin University of China,
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences

2 Purpose of Research Artifact

We proposed a novel approach to extract observable features from the execution traces for inferring the parameter types of binding calls. Then we use the type information to guide the test case (PDFs embedded with Javascript code) generation for fuzzing PDF readers. The submitted artifact evaluates our tool TypeOracle from four aspects: accuracy of type inference, type inconsistency, improving fuzzing performance, and the ability to discover zero-day vulnerabilities.

RQ1: Accuracy of Type Reasoning. The goal of this experiment is to measure the accuracy of the type information inferred by TypeOracle. The result is shown in TABLE I in our paper. Since there is no complete ground truth for the parameter type of binding calls in different PDF readers, we resort to the API reference manual provided by Adobe and reverse engineering to measure the accuracy of type reasoning. We analyzed 251 binding calls for Adobe Reader and 193 for Foxit Reader. TypeOracle has almost 100% precision and around 96% recall.

RQ2: Type Inconsistency. The goal of this experiment is to further demonstrate the advantage of TypeOracle over the API documentation. This part is done by this two comparison: 1) compare the type information of Adobe Reader inferred by TypeOracle with the type information extracted from Adobe API Manual; 2) compare the type information of Adobe Reader inferred by TypeOracle with the type information of Foxit Reader inferred by TypeOracle. The result is shown in Fig. 6 in our paper.

RQ3: Fuzzing Performance. 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 in our paper 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. From the result, type information inferred by TypeOracle incurs 40.38% and 17.09% coverage increment than random testing in Adobe or Foxit. However, coverage guidance only incurs 4.27% and 3.86% coverage increment than random testing in Adobe or Foxit. Favocado and Cooper are two state-of-the-art fuzzers that consider binding calls during fuzzing, they extract parameter types from Adobe's API documentation and 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).

RQ4: Vulnerability Discovery. To evaluate the vulnerability discovery capabilities of different fuzzers in the wild, we deploy 2-week fuzzing campaign for each fuzzer. Gramatron+, Favocado, Cooper, TypeOracle, Favocado+TypeOracle and Cooper+TypeOracle report 2, 6, 5, 33, 10, 18 unique crashes respectively. Further inspection on the crash reports result in 38 zero-day vulnerabilities, the details are shown in Table II in our paper. Even within the first 48 hours, TypeOracle exposed 18 vulnerabilities, which is 2.57 times that of the best of other fuzzers. We should note that Favocado+TypeOracle and Cooper+TypeOracle split time between the mutations of binding calls and the mutations of Javascript templates or PDF page elements, hence exposing fewer vulnerabilities than vanilla TypeOracle within the same time budget.

3 Claimed Badges

* **Reusable.** Our tool is very carefully documented, and the provided manual will be greatly helpful for reproduction.

* **Available.** We have uploaded the source code and execution environment of our tool to public archival repositories. The source code can be downloaded from `https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/TypeOracle/TypeOracleSrc`. The execution environment can be downloaded from `https://zenodo.org/record/7539572#.Y8V7iXbMIuV`.

4 Needed Skills

In order to better understand the reproducing guidance, we assume reviewers have the following technical skills:

1. have knowledge of the basic operation of the Windows operating system, including but not limited to installing/uninstalling software, using the windows command line, and changing the default PDF Reader.
2. have basic knowledge of reverse engineering, and the ability to use reverse engineering tool IDA Pro, especially executing the provided IDA python scripts.
3. have basic knowledge of vulnerability, and have the ability to use windbg.exe to debug programs on Windows operating system.
4. programming language skills: python and C/C++.

5 How to Use

* **Manual.** The manual can be downloaded from `https://github.com/TypeOracle/Manual` which mainly contains three files including REQUIREMENTS.pdf, INSTALL.pdf and REPRODUCE.pdf.

* **Requirement.** The artifacts need to be running on Windows Operating System, and other software such as python, Adobe Acrobat Reader, and Foxit PDF Reader is also needed, for the details please refer to the REQUIREMENTS.pdf file.

* **Installation.** The authors have prepared a ready-to-use environment, which can be downloaded from `https://zenodo.org/record/7539572#.Y8V7iXbMIuV`. If the reviewers plan to install from scratch, they can download the source code from `https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/TypeOracle/TypeOracleSrc` and refer to the INSTALL.pdf file for detailed steps. We recommend to use the provided VM environment for evaluation.

* **Evaluation.** The reviewers can evaluate TypeOracle from the intermediate result or from scratch.
For evaluation from intermediate result: The intermediate result is uploaded to `https://github.com/TypeOracle/TypeOracleIntermediateResults`, please refer the README.pdf file under each RQ folder for the detailed steps to illustrate the result from the intermediate result.
For evaluation from scratch: In the provided VM environment, the source code and work directory of TypeOracle are located in the base directory C:\Users\wxy\TypeOracle. Please refer to the REPRODUCE.pdf file for detailed steps.