C   Artifact Appendix

C.1   Abstract

This artifact contains a functional version of PolyCruise and the necessary dataset for the evaluation. To facilitate the usage of this artifact, we have prepared a Docker image with the necessary components to execute the artifact and visualize the result. Artifact users can compare the results obtained from executing this artifact with those presented in our paper. It is recommended that the host machine has at least 16GB memory and 32GB hard disk space.

C.2   Artifact check-list (meta-information)

- **Algorithm: No**
- **Program: Yes**
- **Compilation: No**
- **Transformations: No**
- **Binary: No**
- **Model: No**
- **Data set: Contained in the package**
- **Run-time environment: Ubuntu 18.04**
- **Hardware: No**
- **Run-time state: No**
- **Execution: No**
- **Security, privacy, and ethical concerns: None**
- **Metrics: Please refer to our paper**
- **Output: Textual information on the terminal**
- **Experiments: Necessary scripts provided**
- **How much disk space required (approximately)?: 10 GB**
- **How much time is needed to prepare workflow (approximately)?: 10 min**
- **How much time is needed to complete experiments (approximately)?: 20 min**
- **Publicly available (explicitly provide evolving version reference)?: Yes, https://github.com/Daybreak2019/PolyCruise**
- **Code licenses (if publicly available)?: Yes**
- **Data licenses (if publicly available)?: Yes**
- **Archived (explicitly provide DOI or stable reference)?: Yes, https://github.com/Daybreak2019/PolyCruise/releases/tag/v3.0**

C.3   Description

C.3.1   How to access

- Download the Docker image for this artifact

  docker pull daybreak2019/polycruise:1.1

- Download the source code of PolyCruise

  git clone https://github.com/Daybreak2019/PolyCruise.git

C.3.2   Hardware dependencies

The host machine may at least need 16GB memory and 32GB hard disk spaces.

C.3.3   Software dependencies

PolyCruise is mainly developed on LLVM 7.0 and Python 3.7. Other software dependencies such as libxml and cmake are also necessary to build the project. For ease of use of PolyCruise, we have prepared a Docker image with all software dependencies installed.

Moreover, real-world benchmarks have their own particular/additional dependencies. Hence, to fully reproduce the results in the paper, users should install these dependencies successfully and ensure each benchmark's test cases can pass. For demonstration purposes, we use Cvxopt as a concrete example of such benchmarks and have installed all of its dependencies in the Docker image.

More specifically, we note that the Docker image includes all the libraries/framework underlying PolyCruise, thus it can be used for experimenting with other real-world subjects as well (i.e., saving the time/trouble for installing ubuntu, llvm, etc.) On the other hand, since our real-world subjects are sizable, including the complete compilation and run-time environment (e.g., all the third-party library dependencies) for all of them in the single Docker image would make it clumsy to deploy conveniently. Using a traditional virtual machine would aggregate this concern since they are even heavier. This is why we chose to include the setup for one such subject only inside the image at this time.

C.3.4   Data sets

PolyCruise is published with a set of micro-benchmarks included inside its code repository. The real-world benchmarks can be retrieved from GitHub.

C.3.5   Models

N/A.

C.3.6   Security, privacy, and ethical concerns

There are no security, privacy, or ethical concerns with using this artifact.

C.4   Installation

- Step 1: Download the Docker image and run a Docker container based on the image

  docker pull daybreak2019/polycruise:1.1

  docker run -it daybreak2019/polycruise:1.1

Table 1: Effectiveness results of PolyCruise on PyCBench, including #inter-language paths (INT-LP), #Intra-language paths (ITR-LP), #false positives (FP), #false negatives (FN)

Group	#INT-LP	#ITR-LP	#FN	#FP
General flow	10	4	0	0
Global flow	9	0	0	0
Filed sensitivity	8	0	0	2
Object sensitivity	9	2	0	1
Dynamic invocation	4	0	0	0
Total	40	6	0	3

- Step 2: Clone the source of PolyCruise and build the project within the container

  cd /root/
  git clone https://github.com/Daybreak2019/PolyCruise.git
  cd PolyCruise
  ./build.sh

C.5 Experiment workflow

We provided scripts for automating all the experiments during the evaluation. Specifically in this artifact, we setup the environment for the experiments on the micro-benchmarks and a real-world Python-C program Cvxopt.

- Experiment on micro-benchmarks

  In this step, we can run all micro-benchmarks together with the following script:

  cd PolyCruise/PyCBench
  ./RunTest.sh

  If we want to run the micro-benchmarks one by one manually, we can switch to the appropriate directory and execute the script, for example:

  cd PolyCruise/PyCBench/GlobalFlow/3_leak_PyClang
  ./build.sh

- Experiment on Cvxopt

  cd PolyCruise/Experiments/scripts/cvxopt
  ./build.sh build

  The parameter "build" indicates the script to instrument and install Cvxopt before running the tests.

C.6 Evaluation and expected results

C.6.1 Evaluation on micro-benchmarks

We use micro-benchmarks (referred to as PyCBench in our research paper) to evaluate the effectiveness of PolyCruise, and it achieved 93.5% precision on PyCBench. The results are summarized in Table 1. Out of all of the 46 cases in PyCBench, PolyCruise succeeded in 43 and failed in 3.

Reproduction.

After building PolyCruise following the steps in Section C.4, use the following command to run PolyCruise on PyCBench:
cd PolyCruise/PyCBench
./RunTest.sh

Expected results.

Obtain the output in the end:
"Finish test, (Correct/Total) = (43, 46)"

Failed cases:

PyCBench/FieldSensitivity/7_leak_PyClang
PyCBench/FieldSensitivity/8_int-overflow_PyClang
PyCBench/ObjectSensitivity/11_leak_Python

C.6.2 Evaluation on Cvxopt

To verify the capability of cross-language vulnerability detection, we also demonstrate evaluating PolyCruise on real-world Python-C programs. In this artifact, we take Cvxopt as an example to exemplify our claim, PolyCruise succeeded in detecting real-world bugs and got one CVE assigned (Table 6 of our paper).

Reproduction.

Use the following command to build and run tests on Cvxopt:
cd PolyCruise/Experiments/scripts/cvxopt
./build.sh build

Expected results.

Report cross-language information flow paths of the `incomplete comparison` vulnerability as shown in Figure 1.

Figure 1: Example of result on Cvxopt

C.7 Version