USENIX Security ’25 Artifact Appendix: Sound and Efficient Generation of Data-Oriented Exploits via Programming Language Synthesis

Yuxi Ling* Gokul Rajiv* Kiran Gopinathan† Ilya Sergey*

*National University of Singapore
†University of Illinois Urbana-Champaign

A Artifact Appendix

A.1 Abstract

This appendix presents the artifact of the paper: "Sound and Efficient Generation of Data-Oriented Exploits via Programming Language Synthesis". We first present a detailed description of hardware and software needed to build our framework Doppler and the baseline framework BOPC. We also provide an experiment list and a comprehensive guide to reproduce the experiments with estimated time. Additionally, we discuss how Doppler is reusable for new programs.

A.2 Description & Requirements

This section introduces the necessary requirements to set up the environment for running the artifact and recreating the same experiments as in the paper, including ethical concerns, how to access the artifact, hardware and software dependencies, and benchmarks. We recommend evaluators run the artifact in a Docker container.

A.2.1 Security, privacy, and ethical concerns

There is no security risk for evaluators while executing this artifact in their Docker container. This artifact is fully executed locally without any possible data collection or leakage. All the data used in the experiments are publicly available and do not contain any sensitive information.

A.2.2 How to access

The artifact is available on Github[1]. Two Dockerfiles are provided to build the Docker image for our framework Doppler and the baseline framework BOPC in the Github repository.

A.2.3 Hardware dependencies

A commodity laptop in either ARM or AMD(x86-64) architecture is sufficient to run the artifact. A minimum of 4GB of RAM and 5GB of disk space is recommended. All the experiments in the paper are conducted on a commodity laptop running Ubuntu 22.04 with Intel Core i7 processor, 16GB of RAM, and 512GB of disk space.

A.2.4 Software dependencies

The artifact can be either built from the source code or the Dockerfile. If evaluators choose to build the artifact from the source code, the main dependencies are listed below:

- Ubuntu 22.04 `(Recommended)`
- LLVM 13
- Clang 13
- Z3 4.13.4
- KLEE 3.0

If the evaluators choose to build the artifact from the Dockerfile, the main software required is Docker. A detailed list of environmental dependencies and installation commands is provided in the Dockerfile.

A.2.5 Benchmarks

There are 17 benchmarks used in the paper, including 6 characteristic programs to illuminate Doppler features and 11 real-world programs to show the scalability of it.

A.3 Set-up

This section introduces how to set up the environment and get the executables to run our framework Doppler and the baseline framework BOPC.

A.3.1 Installation

Once the repository is cloned and the Docker is installed, evaluators can build the environment by running the following commands:

- `cd DOPPLER`

[1] https://zenodo.org/records/14718582

- `docker build -t doppler-image .`
- `docker run -it -name my-doppler doppler-image /bin/bash`

In the docker container `my-doppler`, you will get an executable file `DOPPLER/build/doppler`. The same procedure can be applied to the BOPC.

A.3.2 Basic Test

To test if Doppler is correctly installed and executable, direct to the path `DOPPLER/build` and test the command: `./doppler -h`. If the help message is printed, Doppler is correctly installed. You can set up other options and input files to test Doppler further.

To rest if BOPC is correctly installed and executable, direct to the path `BOPC_evaluation` and test the command: `python ./BOPC/source/BOPC.py`. If the warning message complains the input file is missing, BOPC is correctly installed.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Doppler and BOPC work in all benchmark programs. The results are consistent with those shown in Table 4 of our paper.

(C2): The statistics for the grammar synthesised by Doppler in all benchmark programs are consistent with those shown in Table 5 of our paper.

A.4.2 Experiments

(E1): *[Running BOPC demo] [5 human-minutes + 0.5 compute-hour]: run demo programs in BOPC and collect results.*
How to: *Execute the shell script provided in the file* `BOPC_evaluation/tb4_bopc.sh`
Execution: *Detailed instructions are provided in the last section of the file* `BOPC_evaluation/README.md`
Results: *Execution results are stored in a CVS file with the execution time of each program in each attack goal. Succeed tasks will produce a payload file with* `.gdb` *suffix.*

(E2): *[Running BOPC real] [5 human-minutes + 9 compute-hour]: run real-world programs in BOPC and collect results.*
How to: *Execute the shell script provided in the file* `BOPC_evaluation/tb4_bopc.sh` *with a different argument.*
Execution: *Detailed instructions are provided in the last section of the file* `BOPC_evaluation/README.md`
Results: *Execution results are stored in a CVS file with the execution time of each program in each attack goal. Three tasks would cause a timeout of 2 hours.*

(E3): *[Running Doppler demo] [5 human-minutes + 1 compute-hour]: run demo programs in Doppler and collect results.*
How to: *Execute the shell script provided in the file* `DOPPLER_evaluation/demo_run.sh`.
Execution: *It would automatically compile all demo programs and execute Doppler. Detailed instructions are provided in the last section of the file* `DOPPLER_evaluation/README.md`
Results: *Output files include a generated DFA , a generated grammar, a compiler (an executable binary), and a trace file. To reproduce the attack payloads described in Table 4 of the paper, evaluators should manually check the grammar and determine wether curtain attacks are possible. To reproduce the grammar statistics, evaluators should check the execution log files and get the statistics.*

(E4): *[Running Doppler real] [5 human-hour + 12 compute-hour]: run real-world programs in Doppler and collect results.*
How to: *Execute the shell script provided in the file* `DOPPLER_evaluation/real_run.sh` *with different arguments.*
Execution: *It would automatically compile one real program and execute Doppler each time. Detailed instructions are provided in the last section of the file* `DOPPLER_evaluation/README.md`
Results: *Output files include a generated DFA , a generated grammar, a compiler (an executable binary), and a trace file. The procedure to reproduce results described in the paper is the same as (E3).*

A.5 Notes on Reusability

To run a new program with Doppler, users need to do the following preparations:

- Know the position of the vulnerability.
- Add KLEE annotations into the source code and compile it to LLVM IR file.
- Claim corruptable variables in a JSON file.

More detailed instructions for each step can be found in the README file in the Github repository.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.