ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

A Artifact Appendix

A.1 Abstract

The artifact provided is the implementation of ARBITER framework along with the VD implementations for 4 CWE types and for the Juliet data set. The framework, as well as, the VD templates are written in Python. The artifact contains a helper script written in Python that invokes the Arbiter API when provided with a VD template on a target binary both of which can be specified via command-line arguments. The artifact requires machines that contain 1 logical core and at least 4 GB of RAM. A containerized copy of the artifact is available and can be used on any systems that support docker. The software requirements for installing are Python (version at least 3.8) and *angr*. The artifact has been tested on a machine running Ubuntu 18.04. The artifact also contains the list of packages that were used for evaluation, a JSON file containing the MD5 hashes of each of the binaries as well as the actual binaries from the Juliet data set that were evaluated.

Our paper describes ARBITER as a combination of static analysis and dynamic symbolic execution that can be used to detect classes of vulnerabilities in binary programs with high scalability and low false positive rate. The large scale evaluation on 76,516 x86-64 binaries in Ubuntu repositories as well as the evaluation on Juliet Test Suite (v1.3) highlight these properties of ARBITER .

In order to validate the experiments, one can repeat them using the provided templates and compare the results with those presented in the paper. Since the underlying techniques used by ARBITER contain a degree of non-determinism, the results may vary slightly when evaluating on larger binaries. However, the overall results will be comparable to those presented in the paper.

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

- **Binary:** Binary executables from the Juliet Test Suite (v1.3) that were used during the evaluation are included in the artifact.
- **Data set:** The artifact contains a list of packages from the Ubuntu repository and a JSON file that contains the MD5 sums of each binary used for the evaluation.
- **Run-time environment:** The artifact was verified to work on Ubuntu 18.04 and requires Python (version at least 3.8) and *angr* binary analysis framework.
- **Hardware:** Our experiments were performed on a kubernetes cluster where each pod was provided 1 logical core and 4 GB of RAM. However, each pod could request up to 8 GB of RAM.
- **Execution:** The artifact contains a helper script that can be executed using the Python interpreter. The arguments to this script include the VD template to use as well as the target binary to analyze.
- **Metrics:** The metric used in the experiment is the false positive rate of the bugs reported.
- **Output:** Each template outputs the bugs that it detects in the target binary. The results are also stored into log files and json files that are saved on the disk.
- **Experiments:** To prepare and run the experiments, steps required are as follows.
  1. Download the relevant binary if required.
  2. Clone the artifact repository from https://github.com/jkrshnmenon/arbiter and install it or pull the docker image 4rbit3r/arbiter:latest.
  3. Execute the helper script named *run_arbiter.py* provided in vuln_templates/ with a VD template and a path to the target binary as arguments.
- **How much disk space required (approximately)?:** The total disk space used, including downloaded binaries and generated output files, for our experiment is approximately 350 GB.
- **How much time is needed to prepare workflow (approximately)?:** Installing the framework or using the docker container should take nearly 5 minutes. However, downloading the binaries could take up to 1 minute per package. Even if this process is performed in parallel, it could take up to 1 hour to download all the packages depending upon the network speed.
- **How much time is needed to complete experiments (approximately)?:** Our evaluation was performed on a kubernetes cluster that allowed running 800 tasks at a time. With that constraint, our evaluation of 76,516 binaries took nearly 2 days to complete per template.
- **Publicly available (explicitly provide evolving version reference)?:** The artifact is available at https://github.com/jkrshnmenon/arbiter/releases/tag/v1.1 and as a docker image 4rbit3r/arbiter:latest
- **Workflow frameworks used?:** We used kubernetes in our experiments that allowed us to run 800 tasks at a time with each task being allotted 1 logical core and 4 GB of RAM.

A.3 Description

A.3.1 How to access

The artifact is publicly available at https://github.com/jkrshnmenon/arbiter/releases/tag/v1.1 and as a docker image 4rbit3r/arbiter:latest.

A.3.2 Hardware dependencies

The artifact requires 1 logical core and at least 4 GB of RAM.

A.3.3 Software dependencies

The artifact has been verified to work on Ubuntu 18.04 and requires Python (at least version 3.8) and the *angr* python package.

A.3.4 Data sets

A list of the packages used and JSON file containing the binaries and their MD5 sums are provided. The corresponding binaries can be downloaded from the Ubuntu repositories. The binaries from the Juliet Test Suite (v1.3) are provided with the artifact.

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

N/A

A.4 Installation

The artifact contains a setup script that can be executed to install the framework. After the repository has been cloned, the following command can be used to install the framework : *python setup.py install*.

A.5 Experiment workflow

The experiment was performed on a kubernetes cluster. In order to deploy tasks on the cluster, a docker image has to be specified. We create docker images for each CWE type that would execute the corresponding template script against a specified target binary and wrote the results to disk. This process was repeated for each CWE type using the corresponding template script.

Since the Juliet data-set only provides documentation about the locations of vulnerabilities in terms of source code files and line numbers, a mapping between this location and the address of corresponding function in the compiled binary is required.

ARBITER provides the function address where the vulnerability has been detected. This information, combined with the ground-truth from the Juliet data-set can be used to evaluate the false-positive rate of ARBITER .

A.6 Evaluation and expected results

The paper highlights the high scalability and low false positive rates of ARBITER . The key results that highlight these properties are :

- The large scale evaluation on 76,516 x86-64 binaries on 4 different CWE types.
- The resulting reports that were manually evaluated and the false positive rate was determined to be nearly 40%.
- The evaluation on the Juliet Test Suite (v1.3) where the false positive rate was found to be nearly 23%.

In order to reproduce these results, the templates can be evaluated against the target binaries and the results generated can be manually verified. The expected false positive rate across all the 4 CWE types on the entire data set of 76,516 x86-64 binaries in the Ubuntu repositories is nearly 40% while the expected false positive rate for the binaries from the Juliet Test Suite is nearly 25%.

A.7 Experiment customization

The existing templates can be evaluated on any x86-64 user-space binary. The easiest way to evaluate the existing templates on a new binary is to use the *run_arbiter.py* helper script and specifying the VD template to use as well as a path to the new binary as argument.

The process of implementing a new VD template for a different CWE type has been described in the paper and demonstrated in the comments inside the *examples* directory. This process can be followed in order to evaluate ARBITER using a new template.

A.8 Notes

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.