VulChecker: Graph-based Vulnerability Localization in Source Code

Yisroel Mirsky[1], George Macon[2], Michael Brown[3], Carter Yagemann[4]
Matthew Pruett[3], Evan Downing[3], Sukarno Mertoguno[3] and Wenke Lee[3]
[1]Ben-Gurion University of the Negev
[2]Georgia Tech Research Institute
[3]Georgia Institute of Technology
[4]Ohio State University

A Artifact Appendix

A.1 Abstract

In this document we describe the artifact for our vulnerability detection tool *VulChecker*. The artifact consists of source code, datasets and pre-trained models for reproducing the results from the USENIX'23 paper. This document we only provide the steps requires to demonstrate that the tool is available and functional.

A.2 Description & Requirements

To reproduce the results from the paper you will need to use the source code and assets available on GitHub. There you will find detailed instructions on how to install the tool or acquire the VM which has the tool already installed. You will also find a detailed guide on how to use the the entire pipeline in your own projects.

You can also find our implementation of the baseline `VulDeeLocator` here:
https://github.com/evandowning/VulDeeLocator

A.2.1 Security, privacy, and ethical concerns

There are no risks in executing our tool since it is a vulnerability detector. However, the provided datasets are deviates of projects collected from GitHub. Therefore, users should consider that the projects may contain security risks if executed and users should note the respective software licenses.

A.2.2 How to access

To gain access to the source code an assets, please check out our GitHub repository at https://github.com/ymirsky/VulChecker

`VulChecker` uses a number of components that must be installed in order for it to operate. Here is a list of components of Vulchecker which we maintain in seperate repositories:

- VulChecker: the core library for processing data and training models. All operations with this library are through a command line tool called hector. https://github.com/ymirsky/VulChecker.git
- LLAP: a plugin to LLVM for extracting ePDGs from cmake C/C++ projects. https://github.com/michaelbrownuc/llap
- Structure2Vec: our pyTorch implimentation of the graph-based neural network by Dai et al. https://github.com/gtri/structure2vec
- vulchecker-misc: a collection of helpful (optional) scripts, such as automatic labeling Juliet samples. https://github.com/michaelbrownuc/vulchecker-misc

Information on where the VM and datasets are hosted can be found the main `VulChecker` repo.

A.2.3 Hardware dependencies

To execute the tool on the VM, you will need a host system with at least 16GB RAM. If you intend to preprocess the raw datasets yourself, you will need significantly more RAM (128GB). You can use the VM to train a model on a CPU, but it is highly recommended to use a system with a cuda GPU (we used an NVIDIA TITAN RTX with 24GB RAM). The VM does not come with cuda installed. Therefore, if you want GPU acceleration, you will either need to (1) install the vGPU and cuda libraries on the VM or (2) make a clean installation on a cuda enabled system with the instruction from the repo.

A.2.4 Software dependencies

To make a clean install of the detection tool (instead of using the VM), you will need a Linux system with Ubuntu Ubuntu 20.04 and python 3.8.10.

A.2.5 Benchmarks

Although we provide source code to the baselines used in the paper, we do not provide end-to-end instructions for running them on our datasets at this time. Please follow our VulDeeLocator repo for updates.

A.3 Set-up

For a quickstart, download the VM (link in the README.md of from the `VulChecker` repo). Configure the VM to have at leag 16GB RAM (prefferably more).

A.3.1 Installation

For a clean installation, follow the clean-install steps provided in the repo's README.md. Note that a clean installation can save several hours since LLVM must be compiled.

A.3.2 Basic Test

Once you have booted the VM, you will find three demo scripts on the desktop. These scripts are also available in `demos/` in the repo if you performed a clean-install. These scripts which demonstrate how the tool can be used on a single project provided (Avian) for CWE-121:

1. Convert a C or C++ project into a set of ePDGs
2. Augment datasets (requires large RAM)
3. Train a model
4. Make predictions with model

You can run them to verify that the tool has been installed successfully.

A.4 Evaluation workflow

To use the tool, a separate model and dataset must be prepared for each CWE (190, 121, 122, 415, 416).

There are three ways to reproduce the results from the paper, depending on how far back into the pipeline you want to go: (1) start from raw source code projects, (2) start from preprocessed datasets, (3) start from preprocessed datasets and pretrained models.

The first approaches can take days to perform. Therefore we recommend following the third approach.

(1) For working with the Raw source code files: Follow the instructions in the repo using the diagram provided there. To get the data, follow the link provided in the repo's README. The training data is the 'clean-wild' projects augmented with the Juliet projects. The test data is the 'wild-labeled'. The parameters used to train our models can be found in the `models/trained_on_aug/` directories next to each model.

(2) For working with the preprocessed files, the training data for CWE<id> can be found here: `/CWE<id>/proc_graphs/wild_augmented-labels`.
Use the file that has the format
CWE<id>_*_clean_<N>_<P>.json.gz
which is the dataset after removing a ratio of <N> negative and <P> positive manifestation points.

The test data (projects with CVEs) can be found here: `/CWE<id>/proc_graphs/*/combined`. Use the file that has the format
CWE<id>_*_clean_<N>_<P>.json.gz

(3) To start with preprocessed datasets and pretrained models, follow the instruction in (2) and model files stored in `models/`

A.4.1 Major Claims

(C1): The provided `VulChecker` tool is functional and can be used to detect vulnerabilities/bugs in source code.

A.4.2 Experiments

(E1): *Tool Execution* [30 human-minutes + 3 compute-hour + 200GB disk + 128GB RAM]:
How to: Download the provided VM and allocate the required RAM to the machine. Run the three demo scripts on the VM's desktop. The demos will operate on a single project (Avian) for CWE-121.
Results: The final script should output a csv listing the likelihoods for each potential manifestation point is an actual bug/vulnerability for CWE-121.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.