USENIX'23 Artifact Appendix: ARGUS: A Framework for Staged Static Taint Analysis of GitHub Workflows and Actions

Siddharth Muralee[‡][*], Igibek Koishybayev[†][*], Aleksandr Nahapetyan[†], Greg Tystahl[†],
Brad Reaves[†], Antonio Bianchi[‡], William Enck[†], Alexandros Kapravelos[†], Aravind Machiry[‡]
[‡] Purdue University, {smuralee, antoniob, amachiry}@purdue.edu
[†] North Carolina State University, {ikoishy, anahape, gttystah, bgreaves, whenck, akaprav}@ncsu.edu

A Artifact Appendix

A.1 Abstract

ARGUS's artifact contains the source code and corresponding infrastructure to run our taint tracking tool. This is a modified version of the tool presented in the paper, which can generate all the taint summaries mentioned in the paper on the fly (rather than generating offline summaries). Also provides the datasets required to validate the tool and the claims made in the paper.

This document describes how to set-up our prototype, gives an overview of the requirements to replicate some of the experiments conducted in our evaluation, along with instructions to run them.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

There exist no risks associated with executing ARGUS on any system. ARGUS is encapsulated as a Docker image, incorporating all the requisite dependencies necessary for conducting evaluations. It fetches files into Docker's isolated filesystem, thereby obviating any interaction with the host system's filesystem.

ARGUS doesn't directly interact with the repository apart from cloning it, so it is safe to run on any GitHub repository. However, a few of the vulnerabilities described in this document might still not be fixed, it is recommended to test these vulnerabilities using private forks of these repositories, so that an exploitable fork is not public.

A.2.2 How to access

Given that our paper is presently subject to an embargo, we will be provisionally providing all relevant code and datasets in the form of encrypted zip archives. These archives can be accessed at `https://github.com/purs3lab/Argus_artifacts`, under the commit hash `c8a2086`.

[*]Both authors made equal contributions to this work

The decryption password for the `ARGUS.zip` archive is `d7e21ecf50fd0116a76957f285fda57f6426423af446b`. The `VWBench.zip` archive, however, is unencrypted.

A.2.3 Hardware dependencies

None

A.2.4 Software dependencies

The ARGUS is encapsulated as a Linux Docker image. Any system equipped with the capability to execute Linux Docker containers should suffice for the deployment of the tool for evaluation purposes.

The tool also can be executed outside docker, however, requires a Python version 3.8 and CodeQL installed. All the required Python packages can be installed via the Poetry Python package manager.

A.2.5 Benchmarks

The ARGUS was evaluated using two benchmarks:

- **VWbench:** This comprises a collection of vulnerable workflows, curated from security advisories previously reported and published. The VWbench encompasses 24 workflows, stored in the `vwbench.zip` archive, specifically within the `.github/workflows` directory.
- **Realworld Dataset:** This represents a collection of 2.8 million workflows, upon which our tool was assessed. A selection of representative workflows was chosen from this set to serve as sample PoCs, and added in the paper as listings.

A.3 Set-up

A.3.1 Installation

Given that the tool is packaged as a Docker container, the installation procedure merely entails the setup and construction of the container.

1. Install Docker and Docker Compose via the command: `apt-get -y install docker.io docker-compose`
2. Extract the contents of `ARGUS.zip`, which should contain a directory named `Argus`
3. Navigate to the newly created folder and initiate the build process with the command: `docker-compose build`

A successful build, devoid of any complications, signifies that the tool is prepared for utilization.

A.3.2 Basic Test

To validate the proper functioning of the tool, we have retained the SARIF files corresponding to the actions/checkout action within the directory titled `saved_results`, nested inside the `Argus` folder.

These results can be regenerated by executing the following commands: `./run_check.sh`

The resultant SARIF file should be located in the `results` directory. The SARIF files should be consistent with SARIF file starting with `actions#checkout` inside the `saved_results` folder.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): ARGUS possesses the capability to identify all vulnerable workflows within the VWBench benchmark. This claim is substantiated by the results of Experiment E1.

(C2): ARGUS has been deployed to discover new bugs, a claim which is corroborated by Experiment E2.

A.4.2 Experiments

(E1): *[VWBench] [30 human-minutes + 2 compute-hours + 5GB disk]:* This experiment reproduces the VWBench benchmark for the vulnerabilities identified by ARGUS.
Procedure: Ensure that ARGUS generates alerts for each workflow in VWBench. The results will be located in the `results` directory.
Preparation: Extract the contents of `VWBench.zip` and upload it into a private GitHub repository. The workflows should be situated in the `.github/workflows/` directory of the repository. Generate a GitHub token to facilitate the tool's cloning of the GitHub repository. (We neither retain nor collect the GitHub tokens.)
Execution: Deploy the Docker container using the following command: `docker-compose run argus -mode repo -url <username>:<GHToken>@<url_to_git_repo>`
Results: The execution results should be found in the `results` directory. Each workflow should have an accompanying SARIF file containing the results. The SARIF format resembles JSON and can be viewed using online viewers such as as well as the SARIF viewer plugin on Visual Studio Code.

(E2): *[RWDataset] [2 human-hour + 2 compute-hours + 5GB disk]: This experiment reproduces several of the 0day vulnerabilities found by* ARGUS*, specifically the ones listed in the paper.*
How to: The list of vulnerable workflows and actions presented in the paper, is added to the file `rwvulns.md` in the `Argus` folder. The experiment requires running argus on these repositories and verifying that ARGUS can identify these vulnerabilities.
Preparation: None
Execution: To test the workflows run : `./run_test_docker.sh`
Results: The SARIF file present in the `results` directory can be used to identify the security vulnerabilities in these workflows and actions.

A.5 Notes on Reusability

For the large-scale evaluation delineated in our paper, we cached all reports corresponding to each version of each JavaScript and Composite action, as well as reusable workflows, within a MongoDB database. This procedure can be readily replicated by implementing minor modifications to the infrastructure responsible for report generation within our codebase, specifically within `argus_components/report`.

A.6 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/`.