USENIX Security '25 Artifact Appendix: CloudFlow: Identifying Security-sensitive Data Flows in Serverless Applications

Giuseppe Raffa
Royal Holloway University

Jorge Blasco
Universidad Politécnica

Dan O'Keeffe
Royal Holloway University

Santanu Kumar Dash
University of Surrey

A Artifact Appendix

This artifact appendix is dedicated to `CloudFlow` and `CloudBench`. The first is a serverless-specific static analysis framework, whilst the second is a suite of microbenchmarks used for the evaluation of `CloudFlow`.

A.1 Abstract

In our paper, we present `CloudFlow`, a novel framework to statically detect security-sensitive data flows in serverless applications. To achieve this, `CloudFlow` leverages the infrastructure definition provided by the developer to identify the events, permissions and entry points of an application. We evaluate our framework against a new suite of 40 microbenchmarks, `CloudBench`. Furthermore, we analyse 104 real-world applications selected from AWSomePy, a recent dataset. To the best of our knowledge, this is the largest security-focused analysis of serverless applications to date. Our results show that `CloudFlow` passes all microbenchmarks, apart from three, and detects 11 code injection and information leakage vulnerabilities in the analysed real-world applications. To support the verification of the functionality of our framework and the reproducibility of our results, we open-source both `CloudFlow` and `CloudBench`. In addition, we provide a ready-to-use environment that facilitates their execution. Finally, we make available for future reference a report detailing the security-sensitive data flows identified in AWSomePy.

A.2 Description & Requirements

The artifact presented in this appendix consists of two components. The first is a Linux Ubuntu 20.04 Virtual Machine (VM) that contains the source code of `CloudFlow` and `CloudBench` as well as a pre-processed version of AWSomePy. The VM also includes the installation of the `CloudFlow` command-line utility and Pysa, the underlying static analysis tool used in our framework. The second component is the data flow report of the AWSomePy dataset analysis. This includes the information obtained from `CloudFlow` and our observations on the manually assessed security-sensitive data flows.

A.2.1 Security, privacy, and ethical concerns

The usage of `CloudFlow` and `CloudBench` does not involve taking any destructive steps. With regard to the vulnerabilities discussed in §5.2 and §5.3 of our paper, they were responsibly disclosed in November 2024. More information on this topic can be found in the Ethics Considerations section of the paper.

A.2.2 How to access

Our artifact is available on Zenodo at the following address:

- `https://doi.org/10.5281/zenodo.15609299`

The provided VM is password-protected, and it can be accessed as follows:

- User: Giuseppe Raffa
- Password: USENIX2025

We recommend importing the VM in a recent version of VirtualBox™. If the Ubuntu desktop does not appear automatically after logging in, we suggest adjusting the window size from the main VirtualBox interface by clicking on *View → Adjust Window Size*.

A.2.3 Hardware dependencies

No special hardware is required to run `CloudFlow` and `CloudBench`. However, we recommend allocating 8GB of RAM to the VM. This value is already configured in the provided `.ova` file.

A.2.4 Software dependencies

When we exported our VM, we chose *Open Virtualization Format 2.0*, as this was the most recent version of the standard supported by our hypervisor. While we are not aware of any limitations relating to the Open Virtualization Format, as previously mentioned, we recommend importing the VM in a recent version of VirtualBox.

A.2.5 Benchmarks

`CloudBench` is the only relevant collection of benchmarks. We note that `CloudBench` is included in our VM.

A.3 Set-up

`CloudFlow` was implemented as a command-line (CLI) utility that runs within a dedicated Python Virtual Environment (VE). The latter is available within the folder `pysa_experiments` located on the `Desktop` of the VM. As detailed in §A.3.2, the VE **must be activated** before launching the utility. Also, we recommend deactivating the VE by executing the command *deactivate* in the console prior to switching off the VM.

A.3.1 Installation

The shared execution environment is a fully-fledged, ready-to-use VM. Therefore, no installation is required. The above-mentioned `pysa_experiments` folder contains the required source code within the following folders[1]:

- `awsomepy-serverless`: Pre-processed dataset.
- `cloudbench`: Microbenchmarks suite.
- `cloudflow`: Framework source code[2].

The folder `pysa_experiments` also contains two VEs[3]:

- `experiments`: Main `CloudFlow` VE. This includes Pyre version **0.9.10** (package `pyre-check`).
- `venv-sapp`: Auxiliary VE for report generation. This includes SAPP version **0.5.2** (package `fb-sapp`).

To allow future modifications of our artifact, we note that our VM includes on its `Desktop` a folder called `pyre-check`. It contains a collection of resources obtained from the Pyre GitHub repository[4]. This folder is part of our environment because we were unable to install and configure Pyre in such a way that those resources were automatically detected. We emphasize that `CloudFlow` does **not** work without that folder.

[1]To improve their accessibility, our source code and dataset are also included in a ZIP archive stored in Version v1 of the Zenodo record.

[2]Note that all the `CloudFlow` configuration files are stored within the folder `pysa_experiments/cloudflow/cloudflow/config`.

[3]`CloudFlow` activates the auxiliary VE to use the APIs of the SAPP reporting tool. We made this choice due to installation problems experienced while trying to make Pyre and SAPP available within the same VE.

[4]`https://github.com/facebook/pyre-check`

A.3.2 Basic Test

The objective of the basic functionality test described in this section is to check that `CloudFlow` can be successfully executed. The test consists of 4 steps:

(S1): VE Activation. Open a CLI window and select the `pysa_experiments` folder as the current folder. Activate the `CloudFlow` VE by running the following command: *source experiments/bin/activate*
Expected Output: The CLI prompt will change, and it will include the VE name, i.e., `experiments`, in parenthesis.

(S2): VE Content. The content of the VE can be inspected by executing the following command: *pip list*
Expected Output: `CloudFlow` is present in the list of the packages available in the VE. Specifically, the information associated to `CloudFlow` is the version 0.0.1 and its local installation path.

(S3): Unit Tests Execution. Our framework includes unit tests that can be run to exercise most of its modules. After selecting `pysa_experiments/cloudflow` as current folder, the unit tests can be launched with the following command: *pytest -k "not microbenchmarks"*
Expected Output: The command executes 79 unit tests. All of them are expected to pass.

(S4): Single Microbenchmark Execution. With the same current folder as the previous step, the `CloudBench` microbenchmark `owasp-serverless-injection` can be run by launching `CloudFlow` in single repository mode, i.e., with the *-s* CLI flag[5].
Expected Output: The framework identifies a security-sensitive data flow between line 12 and line 26 of the file `httphandler.py`. The result is displayed in an on-screen table. Importantly, `CloudFlow` supports multiple CLI flags. To access their documentation, execute the command: *cloudflow -h*

A.4 Evaluation workflow

In this section, we first summarize the major claims made in our paper (§A.4.1). Next, we describe a set of experiments (§A.4.2). Each of these is focused on a specific claim.

A.4.1 Major Claims

In what follows, we list the major claims of our paper:

(C1): `CloudFlow` passes all the microbenchmarks in the `CloudBench` suite, apart from three. The results of our microbenchmark-based evaluation are reported in §4.2 of our paper.

[5]*cloudflow -s /home/giuseppe/Desktop/pysa_experiments/cloudbench/intra-procedural/s3-service/owasp-serverless-injection*

(C2): `CloudFlow` detects 205 security-sensitive data flows in 104 AWSomePy applications. We analyse such data flows in §5.2 of our paper. We note that the AWSomePy dataset contains 145 applications in total. However, we focus our attention on 104 of them after an initial triage, summarized in Table 3 of the paper.

(C3): `CloudFlow` identifies 11 vulnerabilities in the analysed AWSomePy applications. Such vulnerabilities are summarized in Table 5 of our paper. The latter also includes two vulnerability case studies in §5.3.

A.4.2 Experiments

In this section, we describe the proposed experiments.

(E1): Microbenchmark-based Evaluation. We estimate: 2 compute-hours + 5 human-minutes. This experiment is designed to validate our claim C1.

Description: `CloudFlow` is launched in microbenchmark mode, i.e., with a CLI flag that allows executing all the `CloudBench` microbenchmarks. The analysis of the results is performed by running a PyTest module.

Preparation: Ensure that the `CloudFlow` VE is active (§A.3.2) and that `pysa_experiments/cloudflow` is the current folder.

Execution: Two steps are required. First, launch the execution of all `CloudBench` microbenchmarks with the command: *cloudflow -mb all*. Second, when the execution of the microbenchmarks is complete, run the relevant PyTest module as follows: *pytest -k microbenchmarks*

Results: Check that the PyTest on-screen summary reports that three microbenchmarks out of 40 have failed. The names of the failed microbenchmarks are specified in §4.2 of our paper[6].

(E2): Security-sensitive Data Flows in AWSomePy. We estimate: 5 compute-hours + 30 human-minutes. This experiment is designed to validate our claim C2.

Description: `CloudFlow` is launched in multiple repository mode, i.e., with the *-m* CLI flag, and generates a report. The latter needs to be compared with the report available on Zenodo.

Preparation: Same as the previous experiment.

Execution: Launch the analysis of the AWSomePy dataset[7]. Unlike the previous experiment, `CloudFlow` is launched with the additional *-cf* flag, used to specify a required configuration file. At the end of the execution, the framework generates a data flow report within the folder `pysa_experiments/cloudflow-report-files`.

Results: To correctly compare the report generated by this experiment with that available on Zenodo, we observe that the latter contains:

- A total of 222 data flows. These include those detected in applications filtered out during our dataset triage as well as others categorized as invalid. The 205 data flows mentioned in claim C2 are those classified as **Valid** in the column Assessment. We expect that the execution of this experiment will identify the same 222 data flows, as the folder `awsomepy-serverless` (§A.3.1) includes all the AWSomePy applications for completeness.
- Six columns **not** generated by `CloudFlow` (from Assessment to Comments). We make these columns publicly available to better document our analysis.

(E3): Vulnerabilities in AWSomePy. We estimate: 3.5 human-hours[8]. This experiment is designed to validate our claim C3.

Description: This experiment requires manually validating at least a sample of the 11 security-sensitive data flows classified as vulnerabilities in the report available on Zenodo. These are marked with the value **Yes** in the column Security Vulnerability.

Preparation: Download the report in CSV format available on Zenodo. This includes all the required information, e.g., the location of sources and sinks.

Execution: Analysis of at least a sample of the security-sensitive data flows classified as vulnerabilities. With regard to the case studies discussed in §5.3 of the paper, we observe that[9]:

- The Code Injection via HTTP case study refers to the vulnerabilities found in the application with identifier 040AA.
- The Exception Traceback Leakage case study refers to the vulnerabilities found in the application with identifier 022AA.

Results: We expect the manual validation to confirm the identified vulnerabilities. We note that the vulnerabilities found in the applications 040AA and 140AA are associated with data flows featuring the same source and sink. To the best of our knowledge, they are reported twice by Pysa because two data flows connecting the source with the sink are detected. Therefore, we consider these as separate data flows.

A.5 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/.

[6]For a shorter on-screen report: *pytest -k microbenchmarks --tb=no*

[7]*cloudflow -m /home/giuseppe/Desktop/pysa_experiments/awsomepy-serverless/repositories -cf awsomepy_dataset_config_file.yml*

[8]Estimate based on the assumption that the validation of a data flow takes, on average, 20 minutes, and that all the relevant data flows are validated.

[9]Note that the code listings provided in the paper to illustrate our case studies are simplified versions of the actual application code.