USENIX'23 Artifact Appendix: Spying through your voice assistants: Realistic voice command fingerprinting (#462)

Dilawer Ahmed
North Carolina State University

Aafaq Sabir
North Carolina State University

Anupam Das
North Carolina State University

A Artifact Appendix

A.1 Abstract

The artifact, written in Python, contains the code and dataset for the major part of our work on fingerprinting voice commands across the three most popular voice assistants. The artifact contains 7 datasets that we used along with code for our data collection process which we used to collect the datasets and the code we used to process the data and get the results that we presented in our paper.

A.2 Description & Requirements

A.2.1 Environment setup

A.2.2 Security, privacy, and ethical concerns

There are no major security or privacy considerations except that you will be collecting network traffic from the routers which can include traffic from other devices connected to this network. You would need to be careful about who has access to this dataset

A.2.3 How to access

The code can be accessed from `https://github.com/dilawer11/va-fingerprinting/tree/0dd1ec3a65e843e366e81ffd29721593bc8043b1`. The datasets can be downloaded from `https://privacy-datahub.csc.ncsu.edu/publication/ahmed-usenix-2023/`. An archived version through Zenodo is available at `https://doi.org/10.5281/zenodo.8037394`

A.2.4 Hardware dependencies

For analysis following specifications are minimum:

- CPU: (x86-64) 8 cores
- RAM: 16 GB
- OS: Debian Linux (Ubuntu recommended)

For data collection following specifications are recommended:

- CPU: (x86-64) 4 cores
- RAM: 8 GB
- OS: Debian Linux (Ubuntu recommended)
- Speaker: Any Stereo PC Speakers
- Routers: 2x OpenWRT (ssh-capable) Linux routers
- Voice Assistant: Alexa, Google Assistant or Siri supported smart speaker

A.2.5 Software dependencies

You can either use the Docker image (provided at `https://privacy-datahub.csc.ncsu.edu/publication/ahmed-usenix-2023/` or setup your own environment (using either the Dockerfile or manually installing all dependencies). The github repository contains the more detailed instructions on how to setup the environment manually and on Docker. The software dependencies which can be installed using `apt-get` are:

- python3.9 (*analysis*, *data collection*)
- python3-pip (*analysis*, *data collection*)
- tshark (*analysis*)
- tcpdump (*router*, *data collection*)
- dumpcap (*data collection*)

The following python packages also need to be installed for analysis: `autogluon.tabular[fastai, xgboost, ray, lightgbm]`, `pandas`, `scikit-learn`, `matplotlib`, `tqdm`, `seaborn`, `datetime`, `plotly`, `jupyterlab`, `python-dotenv`. For data collection the following packages need to be installed: `python-dotenv`, `google-cloud-texttospeech`, `gtts`, `tqdm`, `selenium`.

A.2.6 Benchmarks

The following datasets are available at the URLs provided above:

- simple_50_alexa
- simple_50_siri
- simple_50_google
- simple_100_alexa
- skills_100_alexa
- stream_15_alexa
- mix_100_alexa

The results presented in Tables 2 and 3 of our paper are mostly computed on these datasets and can be used as benchmarks for the setup.

A.3 Set-up

The following subsections describe how to complete the setup

A.3.1 Docker Setup

To set up via the provided Docker container you need to download it from the given link. After you have downloaded it use the following command to load the docker container image into your Docker system. Replace the `path/to/dockerimage` with the absolute path of the downloaded docker image

```
$ docker load < path/to/dockerimage
```

After you have loaded the Docker image you can then create and start a container by using the following command from inside the root repository directory

```
$ docker run -it
  -v $('pwd'):/va-fingerprinting
   vafingerprint
```

The command above starts the container and mounts the repository directory (including source and data) to the container to any changes are reflected and persistent in local storage.

A.3.2 Manual Installation

To manually install the software dependencies use the following commands:

```
$ apt-get update && apt-get install
  packagenames
```

Replace the *packagenames* with the packages that need to be installed based on the type of setup i.e. *analysis*, *data collection*. The *analysis* packages need to be installed on the machine where the raw data is input (e.g our datasets) and results need to be computed. The *data collection* packages need to be installed on the machine where you want to dump the data and want to run the speaker control script for data collection. The *router* packages need to be installed on the router which you want to capture traffic from. In this appendix, we are primarily focused on the *analysis* part.

To install the Python packages use the following commands:

```
$ pip3 install -r path/to/requirements.txt
```

Use the requirements file from the *setup* directory in the GitHub repo. The `requirements_analysis.txt` file contains the pip dependencies for the analysis and the `requirements_collection.txt` contains the pip dependencies for data collection.

To download and setup the datasets you can download them from the URL mentioned above. Then unzip them in the data directory of the cloned GitHub repostory. The detailed instructions can be found in the `data/README.md` file.

A.3.3 Basic Test

After the setup is complete and you have set up the dependencies and the *test* dataset. Use the following command from the root of the cloned GitHub repository to run a basic test that dependencies are properly installed (for the analysis).

```
$ sh src/scripts/test.sh
```

If the script runs without any errors and outputs the results then everything should work fine.

A.4 Evaluation workflow

We provide the details and steps necessary to recompute the results of activity detection and invocation detection which are a major portion of our work in realistic voice activity fingerprinting.

A.4.1 Major Claims

(C1): We achieved more than 98% accuracy while fingerprinting invocations on voice assistants. This is described in Section 4.2 of our paper and in Table 2.

(C2): We improved the state-of-the-art accuracy in voice command fingerprinting on a variety of datasets we collected. The results are presented in Section 5.5-5.7 and in Table 3 of our paper

A.4.2 Experiments

To re-compute the results for the datasets make sure you have already set up the datasets as described previously and conducted a simple test.

A common step to both experiments is to process the PCAPs to remove the unnecessary information and convert to CSV format for easier processing with Python and Pandas. To achieve this you can use the script provided as such

```
$ python3 src/scripts/PCAP2CSV.py -i
  path/to/data/dataset
```

(E1): Invocation Detection [10 human-minutes + 3 compute-hour]: This experiment seeks to evaluate the invocation detection performance on the datasets. The results, once computed, should look similar to Table 2 of our paper

How to: Running this experiment is fairly simple, however, it might take long to actually compute the results. We will initially need to convert the PCAP files to CSV and then create sliding windows based on *invoke records*. Then we will extract features from the windows and finally train the model. As a final step, we can aggregate results across datasets and create a table

Preparation: After you have completed the setup mentioned above. You can then use the `src/scripts/PCAP2CSV.py` script to convert PCAP files to CSV. If you have already converted (for another experiment) you should skip this step.

Execution: After the CSV files are created you would then, for each dataset, need to create windows, extract features and train models. We have provided a default short hand argument to the scripts which can automatically do this. For each dataset you can compute the results using the following command

```
$ python3 src/scripts/InvocationDetection.py
auto-train -i path/to/data/dataset
```

For further options and information you can either use the `-h` option or see the GitHub README file.

Results: To create the table with all datasets you can run the following script with the options as follows

```
$ python3 src/scripts/PostProcessing.py
id-table -d path/to/data
```

This will display a table with result metrics across different models and datasets similar to Table 2 of the paper

(E2): Activity Detection [10 human-minutes + 3 compute-hour]: This experiment is to evaluate the performance of our voice activity fingerprinting method on the datasets we collected. This experiment will reproduce the results in Table 3 of our paper

How to: Similar to last experiment we will need the CSV files (converted from PCAPs) from which windows are created based on *invoke records*. We then extract features and train the AutoGluon model.

Preparation: You can skip this step if the PCAPs for the dataset are already converted to CSV. Otherwise, run the following script to convert PCAP to CSVs.

```
$ python3 src/scripts/PCAP2CSV.py -i
path/to/data/dataset
```

Execution: To create windows, extract features and train the AutoGluon model you can use the following command.

```
$ python3 src/scripts/ActivityDetection.py
auto-train -i path/to/data/dataset
```

Alternatively, instead of the `auto-train` option you can pass the `windows`, `features`, `train` arguments to the script (in this order) to complete this experient.
More information on this script is provided in the GitHub README file

Results: To create the table with all datasets you can run the following script with the options as follows

```
$ python3 src/scripts/PostProcessing.py
ad-table -d path/to/data
```

This will display a table with result metrics across different datasets similar to Table 3 of the paper

A.5 Notes on Reusability

Our artifact was designed to be a prototype and hence is not nearly optimized enough for production. To help with reusing and adding additional functionality we have tried to make the code easier to read and functionality separated into modules. For the extension of a particular module, only that module can be extended while keeping the rest of the code base largely untouched.

The `src/iotpackage/Utils.py` file contains some utility functions that can help with extending the functionality and also help future researchers by optimizing some workflows. For example, the `DataFrame2LatexTable` can convert a Pandas dataframe into a latex table format to save time.

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/.