APPENDIX A
ARTIFACT APPENDIX

This appendix contains instructions to obtain the datasets, the trained models, the extracted features, and the source code, and instructions to execute and evaluate SUMo's pipeline, an architecture that can deanonymize Tor onion service sessions. SUMo's pipeline is composed of two phases, the filtering phase and the matching phase. The filtering phase is composed of two machine learning stages, the source separation and the target separation, both of which output flows to the matching phase. The matching phase is composed of a standalone classifier that outputs the final correlation results. The repository includes the source code and instructions on how to run the experiments. These experiments result in the main plots presented in the paper, which can be used to validate the results. Due to particular hardware requirements, the authors have prepared a machine to run the experiments, to which the authors will grant access after being contacted by the evaluators.

A. Description & Requirements

Our datasets (OSTrain, OSValidate, and OSTest) were generated via scripted clients (using Python and Selenium) that browse both the clearnet and our own onion services through the Tor network. We captured all of the traffic exchanged between clients and onion services at the endpoints, thus, each dataset is composed of many raw pcap files. These raw traffic traces include far more information than the features that SUMo requires. However, due to the scarcity of publicly available datasets, we believe that it is important to make the whole traces available, which will allow subsequent studies without compromising real users in the Tor network. This means that our datasets are significantly large (tens of GBs), they take a long time to extract (about 1 hour depending on the hardware), and they take long to extract the relevant features from the pcap files (about 1 hour also). Hence, we suggest using the pre-extracted features (that we also make available) to reproduce the results exactly as presented in the paper. We did not include Figure 7 since it uses ground truth present only in the raw traffic traces, and we thought it to be unnecessarily burdensome since we show similar results in the remaining plots in Experiment 1.

Also, due to the non-deterministic nature of training models and hyperparameter optimization via Bayesian Optimization, we also provide pre-trained models and the hyperparameters used to take the results in the paper. All of these functionalities are still available in the repository's source code and can be explored via the documentation in the repository[1].

Producing the plots may take longer than expected since the pairing of flows depends on the hyperparameters, so that part will also be executed to reproduce the results, and we run the sliding subset sum each time per each threshold value analyzed. This is purely to analyze how different configurations affect SUMo and it would not be required in deployment.

1) How to access: The artifact's repository is available at https://github.com/danielaLopes/sumo, or as a persistent DOI at 10.5281/zenodo.8393374. The three datasets are available at the persistent DOI separately: OSTrain is at 10.5281/zenodo.8362616, OSValidate is at 10.5281/zenodo.8360991, and OSTest is at 10.5281/zenodo.8359342. The trained models for the source separation and for the target separation stages are available at the persistent DOI 10.5281/zenodo.8366378, the extracted features for the three datasets are available at a persistent DOI 10.5281/zenodo.8369700, the extracted features for the three datasets converted in DeepCoFFEA's format are available at a persistent DOI 10.5281/zenodo.8386335, and the DeepCoFFEA models trained with SUMo's features are available at a persistent DOI 10.5281/zenodo.8388196.

2) Hardware dependencies: The evaluation of this artifact depends on the use of a machine equipped with a GPU (we tested this artifact with an NVIDIA RTX A4000 16GB). For simplicity, the authors have prepared a machine to run the experiments. Access to this machine will be granted after the evaluators contact the authors via HotCRP to share an SSH public key that will be used to authorize access to the machine.

3) Software dependencies: The evaluation of this artifact depends on having the following software:

- Ubuntu OS (tested with linux kernel 5.15)
- OpenCL (tested with 3.0), make, g++
- Multiple Python packages (in `requirements.txt`)

The repository contains a `setup.sh` script that installs dependencies in the system where the evaluation will take place.

4) Benchmarks: None.

B. Artifact Installation & Configuration

The reviewer is expected to clone the SUMo repository and to download the artefact containing the features of the dataset.

C. Major Claims

- (C1): SUMO's matching phase achieves high precision and recall when correlating onion service sessions assuming a perfect filtering phase, even when an adversary does not have full traffic coverage. This is proven by experiments E1 and E2 whose results are illustrated in Figures 6, 8, 9, and 10.
- (C2): SUMO's full pipeline remains able to deanomyze onion service sessions with high precision since SUMo's filtering phase can accurately filter sessions between clients and onion services. This is proven by experiment E3 whose results are illustrated in Figures 11 and 12.
- (C3): SUMO performs better than the state-of-the-art correlation technique DeepCoFFEA on onion service traffic. This is proven by experiment E4's results, illustrated in Figures 13(a), 13(b), and 14, especially by comparing the precision-recall curves on Figures 14 and 6.
- (C4): SUMO has 2 orders of magnitude larger throughput than DEEPCOFFEA (while using considerably less memory): This is the result of experiment E5 whose results are illustrated in Figure 15.

D. Evaluation

To reproduce the experiments, we made available a script that runs all the experiments sequentially. Run it by issuing:

```
./experiment_all.sh
```

[1] https://github.com/danielaLopes/sumo/blob/main/README.md

Alternatively, follow this Section, executing one by one, the scripts as suggested. Run the following script that installs the necessary dependencies (only required in the evaluators' machines, since our machine already has all the dependencies) necessary for all the experiments:

```
./setup.sh
```

1) Experiment (E1): [Session Matching with Perfect Filtering Phase] [10 human-minutes + 40 compute-minutes]: This experiment reproduces the effectiveness of SUMo's matching phase assuming a perfect filtering phase. This produces Figures 6, 8, and 9.

[Preparation] The automated script will download the extracted features from OSTrain, OSValidate and OSTest and compile the C code necessary for all the experiments. Run it by issuing:

```
./experiment1_setup.sh
```

[Execution] The automated script will group the pre-extracted features into pairs of flows, which will be passed to our sliding subset sum algorithm that decides which pairs are correlated and not correlated. It will evaluate the results using multiple threshold and minimum session duration values. Run it by issuing:

```
./experiment1.sh
```

[Results]

```
./experiment1_results.sh
```

The plots will be generated in the `./experiment1` in `pdf` and `png` format.

2) Experiment (E2): [Session Matching with Partial Coverage] [10 human-minutes + 2.5 compute-hour]: This experiment reproduces the effectiveness of SUMo's matching phase considering a partial coverage scenario where we simulate an adversary that is missing the coverage of a continent at each time. This produces Figure 10.

[Preparation] Same as E1.

[Execution] The automated script will group the pre-extracted features into pairs of flows by partial coverage scenario (excluding flows captured in a continent in each different scenario), which will be passed to our sliding subset sum algorithm that decides which pairs are correlated and not correlated. It will evaluate the results using multiple threshold values. Run it by issuing:

```
./experiment2.sh
```

[Results] The plots will be generated by the previous command and they will be available at `./experiment2` in `pdf` and `png` format.

3) Experiment (E3): [Session Matching with Imperfect Filtering Phase] [10 human-minutes + 1 compute-hour]: This experiment reproduces the effectiveness of SUMo's pipeline, with each filtering phase's stage evaluated individually and the final matching phase containing the final results using the imperfect filtering phase. This produces Figures 11 and 12.

[Preparation] Same as E1 and the following automated script will download the pre-trained models for the filtering phase. Run it by issuing:

```
./experiment3_setup.sh
```

[Execution] The automated script will group the pre-extracted features into pairs of flows, which will be passed to our sliding subset sum algorithm that decides which pairs are correlated and not correlated. It will evaluate the results using multiple threshold and minimum session duration values. Run it by issuing:

```
./experiment3.sh
```

[Results]

```
./experiment3_results.sh
```

The plots will be generated in the `./experiment3` in `pdf` and `png` format.

4) Experiment (E4): [Comparison with the State of the Art on Flow Correlation] [10 human-minutes + 5 compute-hours]: This experiment reproduces the comparison between SUMo and DeepCoFFEA, the state-of-the-art flow correlation attack for Tor circuits targetting the clearnet. This experiment focus on the dataset characteristics and DeepCoFFEA's effectiveness at correlating the onion service traffic in our datasets. This produces Figures 13(a), 13(b), and 14.

[Preparation] The automated script will download the datasets and source codes, and generate the data required to run experiments with DeepCoFFEA. Run it by issuing:

```
./experiment4_setup.sh
```

[Execution] The automated script will analyze the characteristics of the traffic samples in DeepCoFFEA's dataset and SUMo dataset, the latter separated in two categories, OSTrain combined with OSValidate, and OSTest. They are compared by duration and number of packets exchanged per request as a bar plot. Then, it will train DeepCoFFEA using OSTrain and OSValidate and test it using OSTest, presenting the results in both a ROC and a precision-recall curve. Run it by issuing:

```
./experiment4.sh
```

[Results] The plots will be generated by the previous command and they will be available at `./experiment4` in `pdf` and `png` format.

5) Experiment (E5): [Throughput evaluation] [5 human-minutes + 1 compute-hour]: This experiment reproduces the comparison between SUMo and DeepCoFFEA regarding throughput (pairs/s). This produces Figure 15.

[Preparation] None.

[Execution] The automated script will compile and execute first SUMo, and secondly DeepCoFFEA, producing files with the standard output. Run it by issuing:

```
./experiment5.sh
```

[Results] The script outputs the maximum throughput point, as well as the latency for that point. A plot with the throughput/latency curve for SUMo and DeepCoFFEA will be generated within the folder `./experiment5` in `pdf` format. Run it by issuing:

```
./experiment5_results.sh
```

The curve may change depending on the GPU used, however we expect the trend to be the same as in Figure 15.