ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

A Artifact Appendix

A.1 Abstract

This artifact provides the code for the tool and the experiments, as described in the paper. The artifact describes two workflows. The first workflow, WebGraph, runs the classification pipeline: we crawl a set websites, build their graph representations and extract features from these representations, and train a classifier to detect advertising and tracking services (ATS) on the sites. The second workflow, Robustness, consists of experiments to perform content and structural mutations on graph representations to evade a classifier. The artifact consists of three components. First, the source code in a GitHub repository that allows a user to set up and run the described workflows from scratch. Second, two docker images with all dependencies installed, one with just the code and the other with a sample database of 100 crawled sites to test the pipelines. Three, a Google Drive folder with datasets from a larger crawl of 10,000 sites, to validate the performance of the classifier and use for other experiments. All the steps required to run and evaluate the pipeline are described in this document and the repository READMEs.

A.2 Artifact check-list (meta-information)

- **Program:** WebGraph (sources included), Forked OpenWPM (link to repo included)
- **Data set:** Sample crawl of 100 sites to test the pipeline. SQLite and LDB databases (included in the docker image). Google Drive link to datasets form a 10k website crawl for further evaluation.
- **Run-time environment:** The project has been run and tested on Ubuntu 18.04. For the setup, you need to have python3, miniconda, binutils, pip, gcc and g++ installed. All requirements are outlined in the repository. We also provide a Docker image with all the dependencies installed.
- **Hardware:** Having a pod on a Kubernetes cluster is preferable but not necessary due to necessity to stay running for long periods of time.
- **Execution:** During the crawl, the user should use the openwpm virtual environment created during the installation of OpenWPM.
- **Metrics:** Classification metrics (accuracy / precision / recall /F1-score / feature importances).
- **Output:** The crawl outputs SQLite and LDB databases. The WebGraph pipeline takes in the crawl output, and outputs three csv files: graph.csv, features.csv and labelled.csv (the details of the files are outlined in the code-base). The classification task outputs a log for the metric statistics in a report file. The robustness evaluation task generates CSV graph files that can be fed into a trained model, and information about classification switches caused by the mutation.
- **Experiments:** Scripts and instructions to fully reproduce the paper's results are provided in the artifact README files.
- **How much disk space required (approximately)?:** The code-base size is around 18.5 MB. The output size varies depending on the number of websites crawled (around 3-4 MB per website on average). The input database size varies also depending on the number of crawls (around 4.5 MB per website on average).
- **How much time is needed to prepare workflow (approximately)?:** The setup of the repository code and the environment should take around 20 to 45 minutes. The crawling time depends on the number of websites (from hours to days). In case you want to accelerate the setup time, you can use the provided docker image with a pre-built project.
- **How much time is needed to complete experiments (approximately)?:** The total pipeline time depends on the number of websites being analyzed (hours to days approximately). Our tests show that on average, WebGraph takes 0.72 seconds to build the graph, 15 seconds to extract features, and 0.25 seconds to train and test each website. The structure robustness experiments use the WebGraph workflow, but perform graph building and training at every iteration, so the time increases accordingly.
- **Publicly available?:** Yes, on https://github.com/spring-epfl/WebGraph
- **Code licenses (if publicly available)?:** MIT
- **Archived (explicitly provide DOI or stable reference)?:** https://github.com/spring-epfl/WebGraph/releases/tag/usenix-artifacts-final

A.3 Description

A.3.1 How to access

The source code is available as a stable tag on GitHub at the following URL: https://github.com/spring-epfl/WebGraph/releases/tag/usenix-artifacts-final. To access it, you can either download the zipped source code or clone the repository.

We also provide two docker images with all the dependencies installed to avoid the setup phase. The image details are as follows:

- **WebGraph image:** Available at https://hub.docker.com/r/springepfl/webgraph. This image contains all the dependencies and the code, and can be used to run the entire pipeline.

- **WebGraph-demo image:** https://hub.docker.com/r/springepfl/webgraph-demo. In addition to all the dependencies and code, this image contains a database of 100 sites (crawled using OpenWPM). The image can be used to work with some test data.

Finally, we provide a Google Drive link with a dataset from a large crawl of 10k websites to evaluate the classifier. The dataset consist of features and labels for different feature configurations of AdGraph and WebGraph. The dataset can be used to replicate all the results of Table 2 in the paper. Link to the dataset: https://drive.google.com/drive/folders/1nDH74p9tLVLvm62DfrsxcO7mraWAWiZa?usp=sharing.

A.3.2 Hardware dependencies

The code is meant to be run on Ubuntu 18.04 with the dependencies mentioned in section A.2. We recommend running the code on a server instance with a fast access to the Internet to accelerate computations. The code-base size is around 18.5 MB. The dataset size can span from 400 MB to several GB depending on the number of crawled websites. We provide a sample dataset of around 100 sites (400 MB).

A.3.3 Software dependencies

- **Custom OpenWPM:** If you intend to run crawls on your own, you need to download a custom OpenWPM tool from this URL: https://github.com/sandrasiby/OpenWPM/tree/webgraph. The installation instructions can be found in the README section of the repository. A sample crawl is included in the WebGraph code-base; you can copy it and run it in the OpenWPM code base. Further instructions are included in the README section of the repository. For convenience, OpenWPM is also installed in the WebGraph Docker image.

- **Docker:** If you opt to use the Docker image, you need to install Docker. After that, you can launch a Docker container using the image and run the experiments in the container.

A.3.4 Data sets

We include a sample dataset in the artifact. Additionally, you can crawl your own dataset using the custom OpenWPM tool following the instructions in the README.

A.3.5 Models

While we do not include a model in the artifact, the dataset on Google Drive can be used to build a model.

A.3.6 Security, privacy, and ethical concerns

N/A

A.4 Installation

- **Installing locally on Ubuntu:** Initially you need to setup the environment as described above. Next, follow the instructions in the OpenWPM README to download and install the OpenWPM tool-set. Finally, download the WebGraph code-base and follow the instructions in the README to setup the project and run the various evaluation tasks.

- **Using the Docker container:** We provide a pre-built Docker image that you can load in a docker environment and start running the experiments immediately.

A.5 Experiment workflow

There are two workflows in the artifact. The main workflow is the WebGraph process. This process consists of crawling sites, building their graph representation, and training a classifier based on features extracted from the graph representations. The second workflow handles the robustness experiments (Section 3 and Section 5 of the paper). This workflow uses the graphs generated by the WebGraph process, and performs different types of mutations in order to evade the classifier. We perform two types of mutations – content mutation (Section 3 in paper) and structure mutation (Section 5 in paper). We describe the workflows below:

- **WebGraph:**

  1. Gather crawl data, either using OpenWPM, or the sample database we provide: To run a crawl with OpenWPM, follow the instructions on the WebGraph repository README to install and activate the environment for OpenWPM. Then, update the script `demo.py` in the OpenWPM codebase to feed in the list of sites you want to crawl, and run the script. The crawl process results in a `datadir` directory containing the output files of the crawl.
  2. Build graph representations, and extract features and labels: Edit the file in the WebGraph repository, `features.yaml` to select the feature set that you want to extract. Run the script `code/run.py`. The README contains information on the arguments accepted by the script. This script first reads in the OpenWPM output files and creates graph

representations of each site that we crawled. It then extracts the feature representations based on the parameters provided in `features.yaml`. Then , it extracts labels for each node in the graph representation using filter-lists. The output of the script is three CSV files: a graph representation file, a features file, a labels file.

3. Train the classifier based on the features and labels files, and get the classifier evaluation reports. Run the classification process (`code/classification/classify.py`) to perform 10-fold cross-validation. This consists of classifier metrics (accuracy, precision, recall), the ground truth and predicted labels of the classifier, and feature importances.

- **Robustness (content mutation):**

  1. Run the WEBGRAPH workflow to obtain graph files and classifier predictions. Use the argument `−−save_model` when running classification so as to have a trained model against which the mutation attacks can be run.
  2. For the sites that you want to perform content mutation on, run `code/run.py` to generate the graphs/features/labels.
  3. Get original classifier predictions by running `code/classification/classify_with_model.py`
  4. Run `robustness/content_mutation/content_mutation.py`. The README in the folder describes the inputs required for the script. Running the script yields a new graph file with content mutation applied on the graph nodes. This can be fed as input to feature extraction and labelling (`code/run_extraction.py`) and then to the trained classifier model to evaluate performance on mutated data.
  5. Structure mutation will yield an output file indicating how many classifier predictions (on adversarial and non-adversarial nodes) switched as a result of the mutation. This file helps analyze the impact of the mutation on the adversary's performance.

- **Robustness (structure mutation):**

  1. Run the WEBGRAPH workflow to obtain graph files and classifier predictions. Use the argument `−−save_model` when running classification so as to have a trained model against which the mutation attacks can be run.
  2. Run `robustness/structure_mutation/greedy_mutation.py` (the README in the folder provides the instructions on what to adjust in the config file for the script). This will perform the structure mutation.
  3. Structure mutation will yield an output file, `diff_stats`, indicating how many classifier predictions (on adversarial and non-adversarial nodes) switched as a result of the mutation. This file helps analyze the impact of the mutation on the adversary's performance. The output file called `overall_stats` gives you an overview of how many nodes originally had to be flipped. The success rate and the collateral damage can be calculated from these output files, as described in Sec 5.3 (page 10/11) of the paper.

We provide a detailed explanation of how to run these workflows in the repository READMEs.

A.6 Evaluation and expected results

The main tool in the paper is WEBGRAPH, which classifies URLs on sites as advertising and tracking (ATS) or benign (non-ATS). WEBGRAPH performs comparably to other classifiers despite not using brittle content features, due to the addition of a new set of features, *flow*, based on ATS behavior. Our artifact enables a user to run the WEBGRAPH pipeline: from the crawl to the graph creation and feature extraction to the classifier training. We provide a test dataset of 100 crawled sites to analyze how the pipeline works. At the same time, this test dataset is too small to validate the performance of the classifier. In order to facilitate verification of WEBGRAPH's performance, we also provide feature and labels from a crawl of 10,000 sites. These can be fed into the classifier to obtain results, and used to train a model that can be used in further experiments (such as the robustness workflow). We opt to provide the processed features and labels for two reasons. First, the raw crawl database of 10,000 sites would be large (in the order of several GB) without necessarily providing much value for evaluation. Second, the processed features and labels can be used for other experiments (an evaluator can use as many or as few sites as they desire to train the model). We note that these files themselves are ≈ 600MB. The datasets provided can also be used to replicate the results shown in Table 2 of the paper, but running the classification process (Step 3 of WEBGRAPH workflow) with the feature and labels file.

The paper performs many experiments related to content and structure mutations. The artifact, therefore, also provides code to generate these mutations. A user can run the mutations on graphs generated from either their own crawled data, or on the data that we provide. The READMEs in the robustness sections of the artifact include information on what the expected output of the mutations are. The code also allows a user to run the entire pipeline in two modes: WEBGRAPH and AdGraph. The content mutation robustness workflow, run with AdGraph mode, can be used to replicate the results of Section 3. The content and structure mutation workflows, run with WEBGRAPH mode, can be used to generate the results of Section 5. Note that in order to generate the results close to the values in the paper, you would have to run crawls with 10,000 sites (for the content mutation experiments) and 100 sites (for the structure mutations experiments).

A.7 Experiment customization

The workflows can be customized as follows:

- **WEBGRAPH:**
  1. The script to run the experiment, `code/run.py`, takes in an argument, `−−mode`, which allows you to specify the system you want to run: AdGraph or WEBGRAPH.
  2. The feature extraction process can be modified to use different categories and types of features, as required. In the default version, we do not use content features (which are used in other tools such as AdGraph), but content features can be extracted by modifying `code/features.yaml`. New features can also be added to the classifier.
  3. The labelling process can be modified to include additional filter lists.
- **Robustness:**
  1. The content mutation process can be modified to mutate the URLs in different ways. The process also offers the option to perform the two scenarios described in Section 3 (third party random mutation, and third party as a subdomain of the first party).
  2. The structure mutation process currently offers four types of mutations that the user can choose from. This can be updated as required.
  3. The structure mutation process currently calculates desired and undesired switches for an adversary as described in the paper. This definition can be modified in the code to account for other adversarial goals.

We provide descriptions of the various customization parameters in the repository READMEs.

A.8 Version

Based on the LaTeX template for Artifact Evaluation V20220119.