PURL: Safe and Effective Sanitization of Link Decoration

Shaoor Munir and Patrick Lee, *University of California, Davis;* Umar Iqbal, *Washington University in St. Louis;* Zubair Shafiq, *University of California, Davis;* Sandra Siby, *Imperial College London*

https://www.usenix.org/conference/usenixsecurity24/presentation/munir

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

USENIX Security '24 Artifact Appendix
PURL: Safe and Effective Sanitization of Link Decoration

Shaoor Munir
UC Davis
smunir@ucdavis.edu

Patrick Lee
UC Davis
pelee@ucdavis.edu

Umar Iqbal
Washington University in St. Louis
umar.iqbal@wustl.edu

Zubair Shafiq
UC Davis
zubair@ucdavis.edu

Sandra Siby
Imperial College London
s.siby@imperial.ac.uk

A Artifact Appendix

A.1 Abstract

This artifact contains the necessary code required to first, crawl websites using a customized version of OpenWPM, second, create a graph representation of crawl using a processing pipeline, and third, to run the classification module for PURL.

A.2 Description & Requirements

Following are the hardware and software requirements to run this artifact:

- Linux operating system (Ubuntu 18.04+)
- Memory: 64GB RAM or higher
- Python 3.9+
- Network connectivity (required for initial website crawl and to download filter lists)

A.2.1 Security, privacy, and ethical concerns

There are no security and ethican concerns. The initial crawl spawns multiple Firefox instances and visits websites provided by the user. Although the generated profiles are temporary and not link-able to the user, the IP address used to visit those websites might be tracked and stored by different actors present on those websites.

A.2.2 How to access

The artifact is available the GitHub repository stable tree link: https://github.com/shaoormunir/purl/tree/0bf3b05de6b0f2805eaf4e34d5402e473438f731

A.2.3 Hardware dependencies

There are no specific hardware requirements, the crawl can work with the memory and computation power required to spawn a Firefox instance. Similarly, the processing step and the classification step can also be run on any decently specced computers, however, due to the nature of graph creation and to speed up the classification pipeline (and avoid occasional crashes due to memory spikes) a processor with more than 4 cores and more than 64 GB of memory is recommended.

A.2.4 Software dependencies

The provided install.sh script creates a conda environment and installs required dependencies. Python 3.9+ is recommended to simplify the process.

A.2.5 Benchmarks

None.

A.3 Set-up

To run PURL, you first need to run OpenWPM on a select subset of websites. PURL relies on a customized version of OpenWPM which is included in the `OpenWPM` folder of this repository.

Step 1: Install OpenWPM Dependencies

First, navigate to the `OpenWPM` folder in the repository:

```
cd OpenWPM
```

Next, install the required dependencies by running the provided installation script:

```
./install.sh
```

(In case there's an error which says that `gcc` is missing from your system, run `sudo apt install build-essential`)

This script will create a conda environment named `openwpm` and install all necessary dependencies. Once the installation is complete, activate the conda environment:

```
conda activate openwpm
```

We will use this environment for the rest of the pipeline.

Step 2: Crawl Websites Using OpenWPM

To begin crawling websites, create a `.csv` file containing the list of websites you want to crawl. Ensure the URLs are listed under the `url` column and include the protocol (http/https). Here is an example of the CSV format:

```
url
https://www.example.com
https://www.example2.com
```

With your `.csv` file ready, run the crawl using the following command from within the `OpenWPM` folder:

```
python3 crawl_sites.py
--websites <path_to_csv_file>
--output_dir <output_directory>
--num_browsers <number_of_browsers>
--starting_index <starting_index>
--log_file <log_file>
--third_party_cookies <always|never>
```

Explanation of Arguments:

- `-websites`: Path to the `.csv` file with the list of websites to crawl.
- `-output_dir`: Path to the directory where the crawl data will be stored.
- `-num_browsers`: Number of browsers to use for the crawl (recommended: 5-10).
- `-starting_index`: Index of the first website in the `.csv` file to crawl.
- `-log_file`: Path to the log file where crawl logs will be stored.
- `-third_party_cookies`: Whether to allow third-party cookies (`always` or `never`).

Note: OpenWPM processes websites in batches of 1,000. If your CSV file contains fewer than 1,000 websites, a single output file will be generated. For more than 1,000 websites, multiple output folders will be created, each containing up to 1,000 websites' data. The output files will be stored in the specified `output_directory`. Each output folder will contain the following two important files:

- `crawl-data.sqlite` (SQLite database containing the crawl data).
- `content.ldb` (LevelDB database containing the content data such as JavaScript loaded on the website).

Step 3: Run Preprocessing for PURL

The next step involves generating a graph and its corresponding features for each website. To do this, navigate to the `Pipeline` folder and run the following command:

```
cd ../Pipeline
python3 run.py
--features <path_to_feature_file>
--folder <crawl_data_folder>
--tag <file_tag>
```

If running the above command results in the following error: `OSError: "enchant-2: cannot read file data: Is a directory`, then run this command first:

```
sudo apt install libenchant-2-dev &&
export PYENCHANT_LIBRARY_PATH=
/usr/lib/x86_64-linux-gnu/libenchant-2.so
```

Explanation of Arguments:

- `-features`: Path to the file containing the features to extract (default: `features_new.yaml`).
- `-input_data`: Path to the folder where the crawl data was stored. For example, if the data was stored in a `datadir` directory within the `OpenWPM` folder, the path should be `../OpenWPM/datadir`.
- `-tag`: Tag to add to the output files. This helps differentiate between multiple experiments.

Running this command will generate four files for each run of crawled data:

- `features_n.csv` (features to be used by classifier to predict tracking/non-tracking use of link decoration)
- `graph_n.csv` (graph representation of websites)
- `exfils_n.csv` (storage value exfiltration observed through link decorations and payloads)
- `labels_n.csv` (labels of link decorations observed during the crawl)

Where `n` is the index of the run.

These files will be stored in the same directory as the `run.py` script.

Step 4: Run the Classification Pipeline for PURL

Navigate to the `classification` folder within the `Pipeline` directory:

```
cd classification
```

Run the classification pipeline with the following command:

```
python3 classify.py
--result_dir <result_directory>
--iterations <number_of_iterations>
```

Explanation of Arguments:

- `-result_dir`: Path to the directory where the classification results will be stored.
- `-iterations`: Number of iterations needed to cover the complete dataset. For instance, if 9 feature files were generated in the previous step (from `features_0.csv` to `features_8.csv`), the number of iterations should be 9.

This command will execute the classification pipeline and store the results, feature importances, and other metrics, along with the model save files, in the specified `result_directory`.

A few important files that are produced as a result of this step are:

- `accuracy`: This file contains the accuracy metrics of each fold of the training step. This can be used to determine the best model to pick for the next step (another file `scores` contains the confusion matrices for each fold for more information). The accuracy number here is calculated against the ground truth which is described in detail in Section 4.1 of the accepted paper.
- `tp_n`, where `n` ranges from 0 to 9: Shows the predictions of each trained model in a fold against the selected test data. The resulting file has the following structure:

  ```
  Ground Truth Label |$|
  Model Prediction |$|
  Link Decoration Name |$|
  Visit ID
  ```

- `model_n.sav`, where `n` ranges from 0 to 9: Checkpoints of the trained models. These checkpoints can be used to run the best performing model in the next step.

Step 5: Run the Best Model on the Complete Dataset

Finally, run the best model on the complete dataset. Navigate to the `classification` folder within the `Pipeline` directory:

```
cd classification
```

Execute the following command:

```
python3 classify_with_model.py
--result_dir <result_directory>
--model_path <model_save_file_path>
--iterations <number_of_iterations>
--generate-filterlist/--no-generate-filterlist
--label_base_path <base_path_for_labels>
```

Explanation of Arguments:

- `-result_dir`: Path to the directory where the results will be stored.
- `-model_path`: Path to the saved model file, this model is the best performing model from the previous step (based on accuracy score or any other chosen metric).
- `-iterations`: Number of iterations needed to cover the complete dataset.
- `-generate-filterlist/-no-generate-filterlist`: Option to generate a filterlist based on the model results (`-generate-filterlist` or `-no-generate-filterlist`).
- `-label_base_path`: The base directory and file name path for the labels. For example, if the labels are stored in `../Data/`, and the labels are named `labels_0.csv`, `labels_1.csv`, etc., the base path would be `../Data/labels_`.

Running this step will produce three different files:

- `filterlist.txt`: This file will contain all the tracking link decorations along with the website on which they were observed, making it easy to block such link decorations.
- `results.csv`: This file contains information about each link decoration and the label assigned to the link decoration by the model. The CSV file contains three major pieces of information: features of the link decoration as observed during the crawl, the label of the link decoration in the Ground Truth (Negative for non-tracking, Positive for tracking, and Unknown if missing in ground truth), and the label assigned by the model.

- `feature importance`: This file contains the most important features used by the model to identify tracking link decorations, related to the results discussed in the Appendix of the cited paper.

A.3.1 Installation

Dependencies will be installed in the previous step.

A.3.2 Basic Test

Run the following to test if the basic functionality is working:

```
conda activate openwpm
python3 demo.py
```

This should crawl 10 test sites, if the crawl finishes without any issues, then the crawling pipeline is functioning without any issues.

A.4 Evaluation workflow

The evaluation can be divided into two different types:

- Evaluating on sites mentioned in the PURL paper.
- Evaluating on any set of sites.

To evaluate on the sites mentioned in the paper, run the pipeline on the `sites.csv` file present in `OpenWPM` directory. To evaluate on any other random set of sites, the reviewer can provide their own set of sites based on the CSV format described in previous section.

A.4.1 Major Claims

The major claims of the paper "PURL: Safe and Effective Sanitization of Link Decoration" regarding this system are:

- **PURL achieves 98% accuracy in classifying tracking link decorations. This can be verified by checking the accuracy of the model on the test data.** The accuracy of the model can be found in the `accuracy` file generated in step 4.
- **PURL is able to generate a filter list containing tracking link decorations for each website provided.** This can be verified by checking the filter list generated in step 5. The filter list contains all the tracking link decorations along with the website on which they were observed. The filter list can be used to block such link decorations.

A.4.2 Experiments

(E1): *[100 sites Experiment] [30 human-minutes + 1 compute-hour + 10GB disk]: This is a smaller experiment to test* PURL*'s results on 100 sites.*

How to: *Use `sites-100.csv` in `OpenWPM` folder to perform the crawl, and then each subsequent step is performed on the resultant crawl.*

Preparation: *Installation and successful run of `openwpm` enviornment as described in step 1.*

Execution: *Follow steps 2 to 5.*

Results: PURL *will achieve an accuracy of around 98% on the link decorations observed during the crawl. The exact number will be variable depending on the conditions of the crawl.*

(E2): *[20K sites Experiment] [30 human-minutes + 200 compute-hour + 500GB disk]: This is the larger experiment to test* PURL*'s results on 20000 sites.*

How to: *Use `sites.csv` in `OpenWPM` folder to perform the crawl, and then each subsequent step is performed on the resultant crawl.*

Preparation: *Installation and successful run of `openwpm` enviornment as described in step 1.*

Execution: *Follow steps 2 to 5.*

Results: PURL *will achieve an accuracy of around 98% on the link decorations observed during the crawl. The exact number will be variable depending on the conditions of the crawl.*

A.5 Evaluation Workflow with Large Dataset

The repository provides a dataset for 20,000 websites that can be used to run the pipeline's steps 4 and 5. The data can be downloaded from the following link: https://zenodo.org/records/12667973.

After downloading the dataset, use the following commands to run steps 4 and 5:

```
ls *tar.xz |xargs -n1 tar -xvzf
cd classification
python3 classify.py
--label_base_path "../labels_new_"
--iterations 20
python3 classify_with_model.py
--model_path <model_path>
--generate-filterlist
--label_base_path "../labels_new_"
--iterations 20
```

A.6 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/usenixsec2024/.