USENIX'23 Artifact Appendix
PoliGraph: Automated Privacy Policy Analysis using Knowledge Graphs

Hao Cui, Rahmadi Trimananda, Athina Markopoulou, Scott Jordan

University of California, Irvine

A Artifact Appendix

A.1 Abstract

In our main paper, we proposed PoliGraph, a type of knowledge graph, for the analysis of data collection statements in a privacy policy. We developed PoliGraph-ER, an NLP system to generate PoliGraph from the text of a given privacy policy. This appendix provides instructions on how to access our artifacts, how to use PoliGraph-ER, and how to reproduce main results in our paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Web scraping is restricted by certain websites. Inappropriate use of crawlers (*e.g.,* without rate limit) can lead to the banning of your IP address. While building the benchmark dataset, we limited our crawling to publicly accessible privacy policy webpages and enforced an appropriate rate limit.

To our knowledge, no other artifacts associated with the paper pose security, privacy and ethical risks to evaluators.

A.2.2 How to access

Source Code. The Git repository is at https://github.com/UCI-Networking-Group/PoliGraph.git. The version for this artifact evaluation is tagged as `USENIX-AE-v1`.

Dataset. Please see Section A.2.5.

Project Page. For up-to-date information, please check our project page: https://athinagroup.eng.uci.edu/projects/auditing-and-policy-analysis/.

A.2.3 Hardware dependencies

We recommend using an x86-64 Linux machine with a minimum of 32 GiB of memory, 20 GiB free disk space (after setting up conda) and an NVIDIA GPU with 24 GiB of video memory for this artifact evaluation. A GPU is needed to run transformer-based NLP models at a tolerable performance.

For *artifact reviewers*, we will offer SSH access to our GPU server, which serves as the test machine. We will post the login details on HotCRP.

A.2.4 Software dependencies

PoliGraph-ER is written in Python and requires several Python libraries. We recommend using conda on Linux to manage dependencies. All the code required for the artifact evaluation has been tested on a Debian 11 GNU/Linux system with Miniconda3 version 23.3.1.

For the full list of software dependencies, please refer to `environment.yml` in our Git repository.

A.2.5 Benchmarks

PoliGraph-ER Extra Data. We provide an archive file `poligrapher-extra-data.tar.gz`[1], which contains the NER model, purpose classification model and global entity ontology required by PoliGraph-ER. These resources can be generated using the scripts released in our Git repository. We have released them for the ease of reproducibility.

Benchmark dataset. We used privacy policies from the PoliCheck project as the benchmark dataset. The privacy policies, although publicly available, may be copyright protected. Users must sign a consent form before accessing it. Please follow the instructions at our project page to obtain the dataset.

For *artifact reviewers*, we provide the dataset directly on our server (`~/dataset`). Please refer to `README.md` in the directory for details about the content.

A.3 Set-up

We provide step-by-step instructions at `docs/usenix-artifact-evaluation.md` in the Git repository[2].

[1] `poligrapher-extra-data.tar.gz`: https://drive.google.com/file/d/1qHifRx93EfTkg2x1e2W_lgQAgk7HcXhP/view?usp=sharing

[2] Document - Artifact Evaluation: https://github.com/UCI-Networking-Group/PoliGraph/blob/USENIX-AE-v1/docs/usenix-artifact-evaluation.md

A.3.1 Installation

For *artifact reviewers*, we have set up the software environment. Please skip this step and continue to Section A.3.2. The instructions below are provided for completeness.

1. Git clone the PoliGraph repository (see Section A.2.2 for the URL) and change to the cloned directory:
```
$ git clone <GITHUB_URL> -b USENIX-AE-v1
$ cd poligraph/
```
2. Download the extra-data tarball (see Section A.2.5) and extract its contents to `poligrapher/extra-data`:
```
$ tar xf /path/to/poligrapher-extra-data.tar.gz \
    -C poligrapher/extra-data
```
3. Create a conda environment named `poligraph` with all the dependencies installed, and activate it:
```
$ conda env create -n poligraph -f environment.yml
$ conda activate poligraph
```
4. Initialize the Playwright library required by the crawler:
```
$ playwright install
```
5. Install PoliGraph-er as a Python module:
```
$ pip install --editable .
```

A.3.2 Basic Test

Here we illustrate how to use PoliGraph-er to generate a PoliGraph from the text of a real privacy policy[3].

1. Run the HTML crawler to download the privacy policy webpage (see Footnote 3 for the full URL):
```
$ python -m poligrapher.scripts.html_crawler \
    <PRIVACY_POLICY_URL> example/
```
2. Run the NLP pipeline on the privacy policy:
```
$ python -m poligrapher.scripts.init_document example/
```
3. Run the annotators to discover relations:
```
$ python -m poligrapher.scripts.run_annotators example/
```
4. Build the PoliGraph:
```
$ python -m poligrapher.scripts.build_graph \
    --pretty example/
```
   The argument `--pretty` is only needed if you would like to generate the graph in GraphML format.

To view the generated PoliGraph, you may use a text editor to open `example/graph-original.yml`. The YAML file describes the graph in a human-readable format. For example, the object below is an edge $\textit{we} \xrightarrow{\text{COLLECT}} \textit{statistical user datum}$ with a purpose "services" as the attribute.

```
- source: we
  target: statistical user datum
  key: COLLECT
  text: ...
  purposes:
    services: ...
```

Please see `docs/view-poligraph.md` in the Git repository for more instructions on how to view a PoliGraph.

[3]The privacy policy of "Puzzle 100 Doors": https://web.archive.org/web/20230330161225id_/https://proteygames.github.io/

A.4 Evaluation workflow

A.4.1 Major Claims

(C0): PoliGraph *Generation.* We show that PoliGraph-er is used to generate PoliGraphs for 6,084 privacy policies in our benchmark dataset.

(C1): *Comparison to Prior Policy Analyzers.* In Section 4.2 of the main paper, we compared the collection relations inferred from PoliGraphs to data collection tuples extracted by PolicyLint. We claimed that our approach identifies 40% more collection relations (tuples) than the prior work, with 97% precision.

(C2): *Policies Summarization:* In Section 5.1 of the main paper, we use PoliGraph to summarize a large corpus of privacy policies and reveal common patterns among them. We will reproduce main results reported in Figures 8a, 8b and 8c, and Findings 1 and 2.

(C3): *Correct Definitions of Terms:* In Section 5.2 of the main paper, we use PoliGraph to access the correctness of definitions of terms in privacy policies. We show examples of different definitions in Table 5, and non-standard terms in Table 6. We will reproduce the results.

A.4.2 Experiments

For easy copy and paste of commands, we recommend following the link provided in Footnote 2. It also provides additional details about the steps and results.

Estimated Time: We expect that the human time required for each experiment is less than 10 minutes. The compute time for E0 is about 4 hours. The compute time for other experiments is negligible (a few minutes).

(E0): *PoliGraph Generation.* This generates PoliGraphs for all privacy policies in our benchmark dataset. E1-E3 will be based on these generated PoliGraphs.

Preparation: Download and extract the benchmark dataset (see Section A.2.5). For *artifact reviewers*, we have already extracted the dataset to `~/dataset`.

Execution: Do Steps 2-4 in Section A.3.2 on all privacy policies in the benchmark dataset as follows:
```
$ cd ~/dataset
$ python -m poligrapher.scripts.init_document dedup/*
$ python -m poligrapher.scripts.run_annotators dedup/*
$ python -m poligrapher.scripts.build_graph dedup/*
```
On the test machine, `init_document` takes about 2.5 hours, `run_annotators` takes 40 minutes, and `build_graph` takes 15 minutes to complete.

Results: After this experiment, each subdirectory under `~/dataset/dedup`, which corresponds to a privacy policy, should have a generated PoliGraph in it:
```
$ ls dedup/*/graph-original.yml | wc -l
6084
```
You may optionally check the generated PoliGraphs (`graph-original.yml`) as we explain in Section A.3.2.

(E1): *Comparison to Prior Policy Analyzers.*

Preparation: This experiment requires manually labeled ground truth data and output from PolicyLint. For *artifact reviewers*, we provided:

- Ground truth data at `~/dataset/external/manual-collection-relations.csv`
- PolicyLint's `ext/` directory, which contains all the input and output data of PolicyLint, at `~/dataset/external/policylint-ext`.

Please finish E0 before this experiment.

Execution:

1. The scripts needed for this experiment are in the `evals/tuples/` directory in the Git repository. Copy it to the dataset directory for convenience:

```
$ cd ~/dataset
$ cp -Tr ~/poligraph/evals/tuples ./eval-tuples
```

2. Convert collection relations inferred through POLIGRAPHs into tuples in a CSV file:

```
$ python eval-tuples/export_poligraph_tuples.py \
    -o eval-tuples/result-poligraph.csv s_test/*
```

The contents in `s_test/` link to a subset of 200 privacy policies, which we use as the test set in the main paper.

3. Convert PolicyLint tuples that belong to the same set of privacy policies into a CSV file:

```
$ python eval-tuples/export_policylint_tuples.py \
    -e external/policylint-ext \
    -o eval-tuples/result-policylint.csv s_test/*
```

4. Compare results from both sides to ground truth:

```
$ python eval-tuples/evaluate.py \
    external/manual-collection-relations.csv \
    eval-tuples/result-poligraph.csv
$ python eval-tuples/evaluate.py \
    external/manual-collection-relations.csv \
    eval-tuples/result-policylint.csv
```

Results: The output in Step 4 corresponds to the precisions and recalls reported in Table 4 of our main paper. You may open `eval-tuples/result-poligraph.csv` and `result-policylint.csv` to view tuples inferred through PoliGraph and those found by PolicyLint.

(E2): *Policies Summarization.*

Preparation: Please finish E0 before this experiment.

Execution:

1. The scripts needed for this experiment are in the `analyses/summarization` directory in the Git repository. Copy it to the dataset directory for convenience:

```
$ cd ~/dataset
$ cp -Tr ~/poligraph/analyses/summarization ./summarization
```

2. Generate statistics over the entire dataset:

```
$ python summarization/collect-and-purpose-statistics.py \
    -o summarization/ dedup/*
```

3. Generate figures of policies summarization results:

```
$ python summarization/plot.py \
    summarization/ summarization/figure.pdf
```

Results: In Step 2, the script prints statistics that are reported in Section 5.1 of the main paper. For example:

```
# of policies that disclose the collection of known
  categories: 4093
```

It also produces CSV files in `summarization/` containing statistics of data collection, sharing and usage purposes. In Step 3, the output PDF file `summarization/figure.pdf` reproduces Figure 8 in the main paper.

(E3): *Correct Definitions of Terms.*

Preparation: Please finish E0 before this experiment.

Execution:

1. The scripts needed here are in the `analyses/term-definitions/` directory in the Git repository. Copy it to the dataset directory for convenience:

```
$ cd ~/dataset
$ cp -Tr ~/poligraph/analyses/term-definitions \
    term-definitions
```

2. Run the script to find term definitions that do not align with our global ontologies:

```
$ python term-definitions/check-misleading-definition.py \
    dedup/*
```

3. Run the script to aggregate non-standard terms found in privacy policies into a CSV file:

```
$ python term-definitions/check-self-defined-terms.py \
    -o term-definitions/non-standard-terms.csv dedup/*
```

Results: Step 2 creates a `misleading_definitions.csv` file in each privacy policy's subdirectory that contains all the different definitions. We use `xsv`, a command-line CSV parser, to obtain counts of different definitions in Table 5 in the main paper. For example:

```
$ xsv cat rows dedup/*/misleading_definitions.csv \
  | xsv search -s parent '^non-personal information$' \
  | xsv frequency -s child
```

The output matches results in Table 5 in the main paper:

```
field,value,count
child,ip address,126
child,geolocation,123
......
```

Step 3 generates a CSV file `non-standard-terms.csv`, which contains results in Table 6 in the main paper about non-standard terms. The rows are like:

```
type,term,def_count,use_count,possible_meanings
DATA,technical information,126,311,advertising id|age|...
```

This row indicates that the data type "technical information" is defined in 126 and used in 311 privacy policies, and possible specific data types are listed in the last column.

A.5 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/.