ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

# USENIX Security ’25 Artifact Appendix: Web Execution Bundles: Reproducible, Accurate, and Archivable Web Measurements

Florian Hantke
CISPA Helmholtz Center for Information Security

Peter Snyder
Brave Software

Hamed Haddadi
Imperial College London & Brave Software

Ben Stock
CISPA Helmholtz Center for Information Security

## A Artifact Appendix

Recently, reproducibility has become a cornerstone in the security and privacy research community, including artifact evaluations and even a new symposium topic. However, Web measurements lack tools that can be reused across many measurement tasks without modification, be robust to circumvention, and remain accurate across the wide range of behaviors in the Web. As a result, most measurement studies use custom tools and varied archival formats, each of unknown correctness and significant limitations, systematically affecting the research’s accuracy and reproducibility.

To address these limitations, we present WebREC, a Web measurement tool that is, compared against the current state-of-the-art, *accurate* (i.e., correctly measures and attributes events not possible with existing tools), *general* (i.e., reusable without modification for a broad range of measurement tasks), and *comprehensive* (i.e., handling events from all relevant browser APIs). We also present *.web*, an archival format for the accurate and reproducible measurement of a wide range of past website behaviors. We empirically evaluate WebREC’s accuracy, by replicating well-known Web measurement studies and showing that WebREC’s results more accurately match our baseline. We then assess if WebREC and *.web* succeed as general-purpose tools, which could be used to accomplish many Web measurement tasks without modification. We find that this is so: 70% of papers discussed in a 2024 web crawling SoK paper could be conducted using WebREC as is, and a larger number (48%) could be leverage against *.web* archives without requiring any new crawling.

## A.1 Abstract

The goal of the artifacts evaluation is to see if you are able to work with our tool of if there are any issues we can improve. Therefore, we would like you to focus on testing and playing around with our tool.

To make it clear, all the functionality of the tool we are talking about in the paper (WebREC), is implemented in *pagegraph-crawl* [1], a tool we continuously maintain. We decided to do so to keep the anonymity during the peer-reviewing process. A tool to analyze the archives is *pagegraph-query* [2]. As mentioned, we would like you to focus on these two and tell us if there is anything to improve that would help you to get easier started with it.

Additionally, we provide the code used in this paper in on Zenodo [3]. This includes the pipeline to run pagegraph-crawl together with the two proxies (mitmdump and warcprox) mentioned in the paper, scripts to replay responses from the archives, and scripts for analysis.

## A.2 Description & Requirements

This section list all the information necessary to recreate the same experimental setup we used to test the artifact.

### A.2.1 Security, privacy, and ethical concerns

Please make sure not to spam any website with the crawler. Other than that, we see no security, privacy, and ethical concerns.

### A.2.2 How to access

You can access the code under the following repositories:

- pagegraph-crawl: https://github.com/brave/pagegraph-crawl
- pagegraph-query: https://github.com/brave-experiments/pagegraph-query
- Analysis pipeline: https://doi.org/10.5281/zenodo.14718651

[1] https://github.com/brave/pagegraph-crawl
[2] https://github.com/brave-experiments/pagegraph-query
[3] https://doi.org/10.5281/zenodo.14718651

### A.2.3 Hardware dependencies

We developed our pipeline under MacOS and run it on a Ubuntu 22.04.4 LTS server. We do not expect any specific hardware needed for our tools.

### A.2.4 Software dependencies

The code requires Python (tested with 3.10.12) and Node.js (tested with 20.11.1) to be installed. If you run the code on a server, a virtual X-server like `xvfb` might be needed. The repositories include a `requirements.txt` for Python dependencies, which can be installed using `pip install -r requirements.txt`, and a `package.json` for Node.js dependencies, installed via `npm install`.

Additionally, Brave Nightly, the testing and development version of Brave needs to be install. Follow the instructions at: https://brave.com/download-nightly/

### A.2.5 Benchmarks

None

## A.3 Set-up

This section contains all the installation and configuration steps required to prepare the environment to be used for artifact evaluation.

### A.3.1 Installation

Below are the installation steps for all repositories. Each step is independent and should start from your initial working directory.

**Pagegraph Crawl.** To install the crawler, clone the repository and build it with `npm`.

```
git clone git@github.com:brave/pagegraph-crawl.git
cd pagegraph-crawl
npm install
npm run build
```

**Pagegraph Query.** To use the query tool, clone the repository and install the Python requirements.

```
git clone git@github.com:brave-experiments/pagegraph-query.git
cd pagegraph-query
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

If you use a Python version before 3.11, you will get an error that StrEnum is missing. You can fix this by installing *StrEnum* (`pip install StrEnum`) and adding `from strenum import StrEnum` at the top of the types file (`pagegraph-query/pagegraph/types.py`). Also, you need to remove `StrEnum` as import from `enum` in line 4.

**Experiment Pipeline.** To setup the experiment pipeline described in Section 4 to collect page graph, HAR, and WARC files, and replays responses from HAR and WARC, follow the steps below. Clone the repository, set up a virtual environment, and install dependencies. Due to version conflicts, in addition to the explained `.venv` you will need a second `.venv_warc` installing `requirements_warc.txt` to run `replay_warc.py` later. All requirements for the analysis scripts can be installed with `requirements_analysis.txt`, we recommend a `.venv_analysis` environment.

```
git clone https://github.com/cispa/WebREC.git
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
cd src
```

Now, you still need to create a config file `config.py` with the following template:

```
TELEGRAM_API_KEY = 'secret'
TELEGRAM_CHAT_ID = 1
JS_HOOKING = True
SCRIPT_PATH = "./js_injections/js_hooks.js"
INITIALIZATION_BREAK = 60
BRAVE_EXEC_PATH = "/opt/brave.com/brave-nightly/brave-browser-nightly"
```

After creating the config, you are able to run the init command. It loads and installs the additionally needed tools such as pagegraph-crawl.

```
python main.py --init --output ./output
```

If you want to test against sites that deploy a CSP, you need to patch pagegraph crawl with `await page.setBypassCSP(true)` as described in the readme of our repository.

If you want to test running in multiple threads, we also recommend to path pagegraph crawl with xvfb retires as described in the readme of our repository.

**Analysis via Jupyter Notebooks** For the analysis files, we use Jupyter Notebooks. There are various ways to execute these files. One way would be to open it in VSCode which automatically shows options to execute it. As an alternative, you can use the Web base version as follows:

1. Install Jupyter Notebook (if not already installed):

   ```
   pip install notebook
   ```

2. Navigate to the directory containing your notebook:

   ```
   cd /path/to/your/notebook
   ```

3. Launch Jupyter Notebook:

   ```
   jupyter notebook
   ```

4. Open the `.ipynb` file in your browser and run the cells.

### A.3.2 Basic Test

This section provides instructions to perform simple functionality tests for the tools. Each test should run in the individual repositories you previously cloned.

**Pagegraph Crawl.** To test if the crawler is built, you can run `npm run crawl -- -h` to get a list of commands. As a next test, you can run the first crawl as described below.

```
# Working directory ./pagegraph-crawl
npm run crawl -- -u "https://example.org" -t 5
 ↪ -o output/ --har --har-body --screenshot

ls output
page_graph_https___example_org__
 ↪ 1736505833.graphml
 ↪ page_graph_https___example_org__
 ↪ 1736505833.png
page_graph_https___example_org__1736505833.har
```

Upon successful execution, the `output` directory will contain:

- A pagegraph file (`.graphml`) representing the page behavior.
- A screenshot of the website (`.png`).
- A HAR file (`.har`) capturing all requests made within the page's context.

If the tool cannot detect your installed Brave Nightly browser, specify its path manually using: `-b /Applications/Brave Browser Nightly.app/Contents/MacOS/Brave Browser Nightly`

**Pagegraph Query.** To verify the installation, run `python run.py -h` to list available commands. Then validate the output of a previous crawl with the following command:

```
# Working directory ./pagegraph-query
python run.py validate
 ↪ ../pagegraph-crawl/output/page_[...].graphml

{"meta": {"versions": {"tool": "0.9.5",
 ↪ "graph": "0.7.1"}, "url":
 ↪ "https://example.org/"}, "report":
 ↪ {"success": true}}

```

**Experiment Pipeline.** To test and familiarize yourself with the experiment pipeline, run `python main.py --help` to view available options. To perform a test crawl, execute:

```
# Working directory ./webrec
cd src
python main.py --output ./output --workers 1
 ↪ --origins https://example.org
```

Upon successful execution, the `output/` directory will contain a timestamped folder with a subdirectory named `https_example.org` holding all the generated files and archives. If the execution fails, retries will be stored in directories such as `https_example.org_failed_attempt_X`. Check `output/<time>/<domain>/logs/` for detailed logs to debug issues.

If you try to run the pipeline inside a docker, it might be that you need to turn of the sandbox mode by passing the `--no-sandbox` option via the `-extra-args` in `src/misy.py:182`.

## A.4 Evaluation workflow

This section includes all the operational steps and experiments which can be performed to evaluate the functionality of our artifacts.

### A.4.1 Major Claims

The major claims we make are the following:

**(C1):** Pagegraph-crawl can generate a graph representing page behavior, a HAR file and a screenshot for any webpage in its page context. Pagegraph-query can analyze the behavior graph. To try this out, follow experiment E1.

**(C2):** The pipeline described in Sec. 3 for the project generates page behavior graphs, HAR, and WARC files. It also allows to replay from HAR and WARC generating the datasets for the replay experiments. Experiment E2 verifies this.

**(C3):** The pipeline can be used to identify and measure pages that use JavaScript-added inline event handlers that would violate *unsafe-inline*. Experiment E3 demonstrates this.

**(C4):** The pipeline can be used to compare the number of HTTP requests between different archiving file formats. Experiment E4 demonstrates this.

**(C5):** The pipeline can be used to compare the appearances of JavaScript executions between different archiving file formats. Experiment E5 demonstrates this.

### A.4.2 Experiments

This section describes the experiments to prove the claims from the previous section.

**(E1):** [Pagegraph Crawl and Query] [X human-minutes]: As first experiment, we would like you to play around with `pagegraphl-crawl` and `pagegraphl-query` and give us feedback if you had any problems setting it up or running it. Take as much time as you want.

**(E2):** [Experiment Pipeline] [30 human-minutes]: In this experiment, please test if you get the experiment pipeline running to collect pagegraph, HAR, and WARC files and replay from HAR and WARC.
**How to crawl:** With the `.venv` environment, run `python main.py --output ./output --workers 1 --origins https://example.org https://google.com` or use your list of origins.
**How to replay HAR:** With the `.venv` environment, run `python replay_har.py --initial-crawl output/timestamp/` with *timestamp* being the one of your previous crawl.
**How to replay WARC:** With the `.venv_warc` environment, run `python replay_warc.py --initial-crawl output/timestamp/` with *timestamp* being the one of your previous crawl.
**Results:** Verify the output directory contains one timestamp directory with subdirectories for each visited domain. Each domain directory should include `.graphml`, `.har`, and `.warc` files, along with replay directories for HAR (`mitmd_replay`) and WARC (`warc_replay`).

**(E3):** [Event Handler Experiment] [15 human-minutes]: Run a simple version of the Event Handler (Document Object Model) experiment from the paper to validate the pipeline's detection of JavaScript-added inline event handlers.
**Preparation:** Change the `config.py` as follows:

```
JS_HOOKING = True
SCRIPT_PATH = "./js_injections/js_hook_smurf_new.js"
```

Then, run the test page in `examples/` on your local machine, for example with `python3 -m http.server`.
**How to:** Run the experiment pipeline for this page: `python main.py --output ./output --workers 1 --origins http://localtest.me:8000` and generate the WARC and HAR replay directories as in E2. Important: Do not use localhost, but a domain like *localtest.me* that points to localhost. Otherwise, the traffic bypasses the proxies.
For the analysis, change in the `analysis/` directory and the `.venv_analysis` environment. Then execute `event_handler.py`. Change `CRAWL_PATH` in line 18 of this file to point to your crawl. To see the results, open and run `csp_analysis.ipynb` with the produced CSV.
**Results:** The result should show that PG, Smurf, HAR Smurf, WARC Smurf and All Smurf_1 all have found one origin with an JavaScript-added inline event handlers that would violate *unsafe-inline*.

**(E4):** [Resources Experiment] [15 human-minutes]: Run a simple version of the Requested Resources experiment from paper to compare the number of resource requests across archiving formats.
**Preparation:** Change the `config.py` as follows:

```
JS_HOOKING = True
SCRIPT_PATH = "./js_injections/js_hooks.js"
```

Then, run the test page in `examples/` on your local machine, for example with `python3 -m http.server`.
**How to:** Run the experiment pipeline for this page: `python main.py --output ./output --workers 1 --origins http://localtest.me:8000`. This time, you do not need replays. Important: Do not use localhost, but a domain like *localtest.me* that points to localhost. Otherwise, the traffic bypasses the proxies.

For the analysis, change in the `analysis/` directory and the `.venv_analysis` environment. Then execute `requests_analysis.py`. Change `CRAWL_PATH` in line 16 of this file to point to your crawl. To see the results, open and run `requests_analysis.ipynb` with the produced CSV `requests_results_timestamp_blocklist.csv`.
**Results:** The result should show that PG and HAR have the same number of requests. Additionally, you can see that the page loaded the 3rd party tracker *googletagmanager.com* blocked by the easyprivacy list.

**(E5):** [JS Executions Experiment] [15 human-minutes]: Run a simple version of the JavaScript experiment from paper and compare JavaScript execution counts across different archiving formats. In order for you to not compile your own browser, we added a hook for `window.setTimeout` that the test page uses.
**Preparation:** Change the `config.py` as follows:

```
JS_HOOKING = True
SCRIPT_PATH = "./js_injections/
    ↪ js_hooks_js_experiment.js"
```

Then, run the test page in `examples/` on your local machine, for example with `python3 -m http.server`.
**How to:** Run the experiment pipeline for this page: `python main.py --output ./output --workers 1 --origins http://localtest.me:8000` and generate the WARC and HAR replay directories as in E2. Important: Do not use localhost, but a domain like *localtest.me* that points to localhost. Otherwise, the traffic bypasses the proxies.
For the analysis, change in the `analysis/` directory and the `.venv_analysis` environment. Then execute `js_compare.py`. Change `CRAWL_PATH` in line 17 of this file to point to your crawl. To see the results, open and run `js_analysis.ipynb` with the produced CSV `js_results_timestamp.csv`.
**Results:** The result should show that PG, HAR, and WARC have the same number (1) of appearances: *Equal number of appearances: 1 / 1 (1.0).*

## A.5 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/.