USENIX Security '24 Artifact Appendix: Web Platform Threats: Automated Detection of Web Security Issues With WPT

Pedro Bernardo[*†], Lorenzo Veronese[*†], Valentino Dalla Valle[‡],
Stefano Calzavara[‡], Marco Squarcina[†], Pedro Adão[§], Matteo Maffei[†]
[†] TU Wien
[‡] Università Ca' Foscari Venezia
[§] Instituto Superior Técnico, Universidade de Lisboa, and Instituto de Telecomunicações

A Artifact Appendix

A.1 Abstract

We present a practical framework to formally and automatically detect security flaws in client-side security mechanisms. In particular, we leverage Web Platform Tests (WPT), a popular cross-browser test suite, to automatically collect browser execution traces and match them against Web invariants, i.e., intended security properties of Web mechanisms expressed in first-order logic. We demonstrate the effectiveness of our approach by validating 9 invariants against the WPT test suite, discovering violations with clear security implications in 104 tests for Firefox, Chromium and Safari. This artifact includes the source code of all the components of the trace verification pipeline and the execution traces and outputs for all experiments presented in the paper. In particular, all violations discovered in the WPT tests (Sec 5.1) and the new testing suite used to assess the impact of the comprehensiveness of tests on the pipeline results (Sec 5.3). To support the independent validation of the results, the artifact includes scripts to re-execute the pipeline on the violating tests.

A.2 Description & Requirements

We provide in this section all the information necessary to download the artifact and recreate the same experimental setup used to run the trace verification pipeline.

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

The artifact is available at the following stable URL: https://github.com/SecPriv/web-platform-threats/tree/201bda9cca58064e61c3d13cae592e4688585f94

[*]Shared first authorship

A.2.3 Hardware dependencies

The artifact does not require any specific hardware features. As the execution of the pipeline includes running a browser and the Z3 theorem prover, we recommend using a computer with at least 8 CPU cores/threads and 8 GB of RAM. Furthermore, we suggest having at least 10GB of disk space as the required docker images have a size of approximately 5GB and the browser execution may create additional temporary files.

A.2.4 Software dependencies

All software dependencies of the artifact are packaged as Docker containers. Hence, a working Docker Engine installation is required to test the artifact. Additionally, running the scripts to reproduce the experiments requires a recent version of bash, python and GNU make. All the components of the artifact have been tested on Linux.

For licensing reasons, the artifact does not include a copy of the MacOS virtual machine containing Safari; thus, the experiments in A.4.2 use the traces collected for the paper. However, the repository includes a guide (`runner/safari/README.md`) for installing the required components on an existing MacOS installation to obtain new execution traces.

A.2.5 Benchmarks

The repository of the artifact provides, in the `results` folder, the execution traces and the verification results for all experiments where a violation was discovered.

A.3 Set-up

We describe in the following the steps required for the installation and the basic functionality test of the artifact.

A.3.1 Installation

The environment to run the trace verification pipeline is packaged into three Docker containers that can be downloaded using the `wpt-check` script or the `make` command as follows.

```
./wpt-check pull     # or make pull
```

Note that the `wpt-check` command requires the current user to be in the `docker` group. Refer to the `README.md` file in the root folder of the repository for additional instructions, e.g., how to build from sources, and a description of each container.

A.3.2 Basic Test

The functionality of the pipeline can be checked by executing it on one WPT test:

```
./wpt-check run firefox '/cookies/secure/set-from-dom.https.sub.
     html'
```

The output of the script should include the WPT run output, the parsed trace and the Z3 output. In particular, for each invariant and trace, it should print `unsat` as in the following.

```
(error "line 1 column 28: sort already defined List")
===== trace1 =====
host-invariant
unsat
(error "line 1 column 28: sort already defined List")
===== trace1 =====
http-only-invariant
unsat
(error "line 1 column 28: sort already defined List")
===== trace1 =====
samesite-cookies-confidentiality
unsat
...
```

Note that the error message printed by z3 refers to the fact that the solver already provides a built-in `List` datatype, which we define to retain compatibility with multiple solvers. The error does not have any impact on the solutions as the built-in datatype is identical to our definition.

The tests may generate multiple traces if they are composed of multiple sub-tests: the parsed trace printed by the SMT-LIB generator includes the name of each test, as exemplified in the following output:

```
% 'secure' cookie visible in HTTP request
1710759552459='net-request'("3","http://web-platform.test:8000/
     testharness_runner.html",'M-GET','type-main_frame',"http://
     web-platform.test:8000/testharness_runner.html",some("http://
     web-platform.test:8000/testharness_runner.html"),nil,nil,'
     request-headers'(none,none),none)
...
```

A.4 Evaluation workflow

We describe below the steps to reproduce and validate the results of the paper.

A.4.1 Major Claims

(C1): We show the effectiveness of our trace verification pipeline by verifying our 9 invariants against the WPT test suite, discovering violations with clear security implications in 104 tests (Sec. 5.1). These violation can be validated by experiment (E1), which verifies that all SAT results in Table 2 can be reproduced.

(C2): We show that employing a more comprehensive test suite has the potential to identify additional violations; while our focus for this paper is WPT, our pipeline can be applied to alternative testing suites, potentially improving its efficacy (Sec. 5.3). The new violations can be validated by experiment (E2), which runs the pipeline on the testing suite presented in Table 5, verifying that the discovered SAT results are reproducible.

A.4.2 Experiments

(E1): *WPT violations*. This set of experiments validates the SAT results of Table 2 for each browser by executing the runner and the Z3 solver for each violation. We split the experiment into 3 sub-experiments, one for each browser, that require the same preparation and execution steps.
How to: From the `results/wpt` folder, execute the `check-result.py` script. Refer to (E1.A, E1.B, E1.C) for the required parameters. The script executes `wpt-check` for all SAT results, displaying for each browser the number of tests and the successfully validated results (i.e., the SAT that are still SAT after re-executing the pipeline). The optional argument `--no-runner` will not run a new instance of Firefox/Chromium and use the trace collected for the paper experiments. Refer to the `README.md` file in the `results/wpt` folder for additional information.
Preparation: The experiment requires a working installation of the pipeline Docker containers obtained after executing the steps described in A.3.
Results: After executing all tests, the script will output for each invariant its name and the number of SAT results that could be reproduced as exemplified in the following.

```
blockable-mixed-content-filtered: 21/21 SAT
```

These results match the corresponding (browser, invariant) cell of Table 2.

(E1.A): *WPT violations (Firefox) [5 human-minutes + 70 compute-minutes]*.
Execution: Execute the following command in the `results/wpt` directory.

```
./check-results.py -b firefox --no-runner
```

(E1.B): *WPT violations (Chromium) [5 human-minutes + 50 compute-minutes]*.
Execution: Execute the following command in the `results/wpt` directory.

```
./check-results.py -b chromium --no-runner
```

Note that the execution of the Chromium browser (without `--no-runner`) is not always deterministic and may be affected by the available resources, potentially resulting in unexpected output. The `check-results.py` script implements a retry mechanism that re-executes the runner up to 20 times. In case the experiment fails we suggest re-running the `check-results.py` script.

(E1.C): *WPT violations (Safari) [5 human-minutes + 5 compute-minutes].*
Execution: Execute the following command in the `results/wpt` directory.

```
./check-results.py -b Safari
```

The script will run Z3 on the traces collected for the paper experiments. Refer to the `README.md` file in the `runner/safari` folder for instructions on how to run an instrumented Safari and obtain new JSON execution traces. These traces can be validated using the `./wpt-check verify` command.

(E2): *Comprehensiveness of Tests: additional tests [5 human-minutes + 10 compute-minutes].* This experiment validates the SAT results of Table 5 by executing the runner and the Z3 solver for each violation.
How to: From the `results/beyond_wpt` folder, execute the `check-results.py` script. Similarly to test (E1), the script executes `wpt-check verify` on the traces obtained by running each test, displaying the solver results. Refer to the `README.md` file in the `results/beyond_wpt` folder of the repository for additional information.
Preparation: The experiment requires a working installation of the pipeline Docker containers obtained after executing the steps described in A.3.
Execution: Execute the following command in the `results/beyond_wpt` directory.

```
./check-results.py
```

Results: After executing all tests, the script will output for each browser and invariant its name, the expected result and the obtained result as in the following.

```
[webspec_host_frames - firefox] host-invariant expected: "sat
     " got: "sat"
```

Each line corresponds to a non-empty cell (test, browser, invariant) of Table 5.

A.5 Notes on Reusability

Updating Browser or WPT versions. The trace verification pipeline presented in this paper can be applied to newer versions of the browsers or WPT by updating the respective lines of the `runner/Dockerfile` file. In particular,

- Commenting line 24 of the Dockerfile and rebuilding the container will make `wpt` download the latest stable version of Firefox;
- removing `-revision 1185653` from line 26 will make `wpt` download the latest stable version of Chromium.
- The WPT repository can be updated by changing the URL passed to `start.sh` in line 18 of the Dockerfile. Note that some modifications are required for the instrumentation to detect the start and end of each test. These changes can be obtained by comparing the `wpt-security` branch with the upstream `main`.

Adding new invariants. New invariants can be added to the `trace_matching/trace_matching.smt` file as functions (`define-fun`) from `Trace` to `Bool` and are automatically added to the checks for each parsed execution trace by the SMT-LIB generator.

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/.