USENIX'23 Artifact Appendix: Cookie Crumbles: Breaking and Fixing Web Session Integrity

Marco Squarcina
TU Wien

Pedro Adão
Instituto Superior Técnico, ULisboa
Instituto de Telecomunicações

Lorenzo Veronese
TU Wien

Matteo Maffei
TU Wien

A Artifact Appendix

A.1 Abstract

This artifact is provided to support the evaluation of all the results presented in the paper. In particular, (i) the cross-browser testing suite used to validate the results presented in Table 2, (ii) the toolchain developed to automatically test server-side cookie parsers (Section 4.2.2), (iii) the dataset and processing code of our cookie measurement study (Section 4.4), (iv) reproducible proof-of-concept attacks against vulnerable Web frameworks (Section 6), as well as (v) the ProVerif models and scripts (Section 7).

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 analysis and experiments.

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

The artifact is available at https://doi.org/10.5281/zenodo.8220368.

A.2.3 Hardware dependencies

The artifact does not require any specific hardware features. Notice that the repository includes a copy of the dataset used in the measurement study (Section 4.4), which is about 3GB in size.

A.2.4 Software dependencies

Most software dependencies of the artifact are packaged as Docker containers, hence we require a working Docker Engine installation before testing the artifact. When some components are not packaged as Docker containers, we provide instructions to execute them on the host machine. The readme files of each subfolder describe the specific requirements for each component. All the components of the artifact have been tested on Linux.

A.2.5 Benchmarks

The source dataset for the measurement study (Section 4.4) is the Archive dataset[1] using the optimized tables from Web Almanac.[2] Since this dataset is in the public domain, we include a copy of the processed dataset via Google BigQuery in the artifact. The resulting dataset is about 3GB in size and is provided in the `measurement` folder of the artifact. All processing queries are also included in the artifact.

A.3 Set-up

We describe in the following the steps required for the installation and the basic functionality test of the artifact. We split each of the following subsections in 5 paragraphs, each detailing the specific steps required for each subfolder. We advise reviewers to install and evaluate one component at a time.

A.3.1 Installation

Browser Tests. The browser test suite can be installed via Docker.

```
cd browser-tests
docker compose up --build
```

Ensure that ports 80 and 443 are available on the local machine and that cookies.localtest.me and sub.cookies.localtest.me resolve to 127.0.0.1. If this is not the case, follow the instructions in the readme file. Detailed information on how to install a specific version of Firefox is also included in the readme file.

Reflectors. This component includes the toolchain developed to automatically test server-side cookie parsers. Reflectors are minimal programs implemented in one of the

[1] https://httparchive.org/
[2] https://almanac.httparchive.org/

tested backends that parses HTTP requests containing a Cookie header and returns a JSON dump of the cookie names and values. All supported reflectors for PHP, ReactPHP, and Werkzeug can be installed using Docker.

```
cd reflectors
docker compose up --build
```

Please ensure that ports 1700, 1701, and 1702 are free on the local machine. The fuzzer runs on the host machine and has been tested on Python 3.10.6. The only dependency is the `requests` library, which can be installed via pip.

```
pip install --user requests
```

Measurement. The dataset for the cookie measurement study is provided in the `measurement` folder. The script used to analyze the dataset (`analyze.py`) is written in Python 3 and requires no third-party modules. The other Python script (`draw.py`) is used to generate the plot in the paper and requires the `matplotlib` and `numpy` library. The scripts have been tested on Python 3.10.6.

```
pip install --user matplotlib numpy
```

Web Frameworks. Each vulnerable framework can be installed via Docker and requires the usage of some environment variables (detailed in the readme file), for instance:

```
cd web-frameworks/express-pre-login
export VERSION="v0.5.3"; docker-compose --env-file
      ../testing.env up -d --build
```

Ensure that ports 80 and 443 are available on the local machine and that localtest.me and attack.localtest.me resolve to 127.0.0.1. The automatic testing script runs on the host machine and has been tested in Python 3.11.3. The dependencies are the `requests` and `bs4` libraries, which can be installed via pip.

```
pip install --user requests bs4
```

ProVerif. We provide an exact copy of our testing environment in the docker image `wert310/proverif:a2e281f`.

```
docker pull wert310/proverif:a2e281f
```

A.3.2 Basic Test

Browser Tests. Point your browser to `http://cookies.localtest.me` and `https://cookies.localtest.me` to ensure that the test suite is running. Accept the self-signed certificates for the HTTPS test. Both URLs should display the "Cookie Integrity Evaluator" page.

Reflectors. Ensure that the reflectors are running by executing a simple request to each of them.

```
for port in 1700 1701 1702; do curl -H "Cookie:␣foo=bar"
      "http://localhost:${port}"; echo; done
```

The output should be `{"foo":"bar"}` repeated 3 times. Notice that the presence of an additional whitespace character in the last row is not an issue.

Measurement. The measurement study can be executed by running the `analyze.py` script. Ensure that the script is working by running it without arguments.

```
./analyzer.py
Usage: python3 ./analyzer.py <csv_directory>
```

Similarly, the plot can be generated by running the `draw.py` script.

```
./draw.py
```

Web Frameworks. Point your browser to `http://localtest.me` and login with credentials `alice:alice`. Transfer 1 credit to `bob` and ensure your final balance is 999 credits. Access `http://attack.localtest.me`. The attacker's site is running if you obtain information in the debug session after pressing `Set-Pression`.

ProVerif. The functionality of the test can be checked by running ProVerif on one of the models without applying the fix. Run a shell of the testing environment:

```
docker run --rm -ti -v$PWD:/mnt --workdir /mnt
      wert310/proverif:a2e281f bash
```

Then execute the Flask model without fix:

```
stdbuf -o0 make -B run-flask
```

The output should include:

```
Query event(app_action_successful(cp_9,token_6)) ==>
      event(app_action_begin(b_9,token_6)) cannot be proved.
```

showing that our invariant does not hold for Flask without applying our proposed mitigation. We provide technical details on the formalization and instructions on how to verify all frameworks in the `README.md` file in the `proverif` folder.

A.4 Evaluation workflow

Below we describe the steps to reproduce the evaluation of the paper.

A.4.1 Major Claims

(C1): We performed a thorough cross-browser evaluation of known cookie integrity attacks and introduced new attack vectors classified along 4 different categories: serialization collisions due to nameless cookies (Section 4.2.1), server-side parsing vulnerabilities (Section 4.2.2), cookie jar desynchronization issues (Section 4.2.3), and broken composition of (compliant) parsers (Section 4.2.4). The artifact follows the methodology presented in (Section 4.3). Browser-side experiments can be reproduced using the provided test suite (**E1.A**), while server-side experiments can be reproduced using the fuzzer and reflectors (**E1.B**). The vulnerability affecting the AWS Lambda Proxy integrations has been fixed by Amazon and cannot be reproduced.

(C2): We also conducted a cookie measurement study aimed at assessing the prevalence of cookie name prefixes, secure cookies and nameless cookies in the top 100K websites (Section 4.4). The results of the study can be reproduced using the provided dataset and scripts (**E2**).

(C3): We performed a systematic security analysis of the top 13 Web frameworks, exposing CORF token fixation and session fixation vulnerabilities in 9 of them. Experiment **E3** reproduces these experiments as well as the results of our disculosure process.

(C4): We formally verified of the correctness of our proposed mitigation to the synchronizer token pattern using the ProVerif protocol verifier (**E4**).

A.4.2 Experiments

(E1.A): *Browser Test Suite [15 human-minutes + 2 compute-minute + 100MB disk].* Execute the test suite on Firefox-104 to match relevant findings presented in Table 2. This experiment is functional to reproduce browser-side cookie issues (C1). Due to space constraints, details on how to install Firefox an understand the output of the test suite are provided in the `browser-tests/README.md`.

(E1.B): *Server-Side Reflectors [15 human-minutes + 5 compute-minutes].* This experiment is meant to reproduce server-side cookie issues (C1). A simple fuzzer generates variations of the Cookie header, sends the same request to all reflectors, and records any differences in the JSON dumps. The provided `reflectors/README.md` file explains in detail how to interpret the obtained CSV file and match it to the 3 CVEs assigned to the discovered vulnerabilities in PHP, ReactPHP, and Werkzeug.

(E2): *Cookie Measurement [15 human-minutes + 3 compute-minutes + 3GB disk].* Reproduce the results of the cookie measurement study on the top-100K websites (C2), including the output of Table 3, Figure 4, and Table 4. From the `measurement` folder, execute the analyzer script on the two provided datasets:

```
python3 ./analyzer.py data-2021-07-01
python3 ./analyzer.py data-2022-06-01
```

A detailed explanation of the output of the script is provided in the `measurement/README.md`. Notice that the queries to obtain the datasets from Web Archive are also available in the same folder.

(E3): *Web Frameworks [10 human-minutes + 15 compute-minutes].* Each framework is provided with an automatic testing script that can be used to test the application.

```
cd express-pre-login
export VERSION="v0.5.3"; docker-compose --env-file
    ../testing.env up -d --build
echo "Should␣be␣vulnerable␣to␣pre-login"
python3 test-express-pre-login.py
```

For convenience, we also provide a script `test_all.sh` that builds and tests all the applications in sequence. Applications can also be manually tested. For the example above, the following tests can be performed: (i) Access `http://attack.localtest.me/` in a browser and press `Set Pre-session`. (ii) Open a new tab in the browser and access `http://localtest.me/`. (iii) Login as one of the users, `alice`, `bob`, or `john_doe`. The password is equal to the name of the user. You shoud start with 1000 credits. (iv) Return to the tab `http://attack.localtest.me/` and execute a transfer of 1 credit to `attacker`. (v). Return to the tab `http://localtest.me/` and refresh. Your balance should now be 999 credits. Further details in `web-frameworks/README.md`.

(E4): *ProVerif Verification [30 human-minutes + 7 compute-minutes].* Verification of the correctness of our proposed fix, i.e., refreshing the token upon login, to the synchronizer token pattern for all 7 frameworks vulnerable to the CORF token fixation (pre-login) attack. To run the experiment, follow the instructions in the `proverif/README.md` file by executing the for loop listed under the "Verifying all frameworks" section. The output of the above loop will contain, for each of the 7 frameworks, the checked properties and the ProVerif results. The expected output of each framework should contain 6 reachability queries with result `cannot be proved`, showing that all events in the model are reachable. As the last line it should contain

```
RESULT event(app_action_successful(cp_18,token_6)) ==>
    event(app_action_begin(b_9,token_6)) is true.
```

proving that our expected invariant is true after applying the fix to the framework.

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/`.