USENIX Security '26 Artifact Appendix: BUIzz: Finding Policy Enforcement Bugs via Interaction Simulation on the Browser User Interface

Mingi Jung *UNIST*
Donggyu Kim *UNIST*
Mijung Kim *UNIST*
Seongil Wi *UNIST*

A Artifact Appendix

A.1 Abstract

This artifact contains the implementation of BUIzz, including its COLLECTOR, SIMULATOR, and DETECTOR components, as well as the policy-specific test servers used in our evaluation. The artifact also includes a minimal working example (`example/`) that allows evaluators to run the complete detection workflow and reproduce representative policy-enforcement bugs reported in the paper, including a CVE-assigned vulnerability.

Because BUIzz operates by simulating Browser User Interface (BUI) interactions at the operating-system level, it requires an interactive Windows desktop session with a real display and cannot run headless or in Continuous Integration (CI) environments. Beyond reproducing the paper's findings, the artifact can serve as a foundation for future research on browser security testing.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

While running, BUIzz controls the physical mouse and keyboard, and serves local test pages via `hosts`-file redirection with an `mkcert` root CA. We recommend a spare machine and undoing these host changes afterwards. To reproduce already-patched bugs without downgrading an installed browser, use the version-pinned portable Brave in `example/`. All experiments are conducted locally and do not interact with external services or user data.

A.2.2 How to access

Source code with a permanent DOI: https://doi.org/10.5281/zenodo.20422485.

A.2.3 Hardware dependencies

A single x86-64 machine running an interactive Windows desktop session with an attached display is required. No special hardware (GPU/FPGA) is required.

- **Display:** a real, single primary monitor at a standard resolution. The SIMULATOR computes on-screen link coordinates for OS-level input, so it requires a foreground window and cannot run headless.
- **Memory/disk:** the paper used commodity desktops with 8–32 GB RAM (Table 6); 16 GB RAM and ∼50 GB of free disk are comfortable for the functional tests in this artifact.
- **Recommended display:** a single 2560×1440 primary monitor.

A.2.4 Software dependencies

Tested on Windows 11 with Python 3.11.

- **OS:** Windows 10/11 (64-bit).
- **Python:** Python 3.11.x with `playwright` 1.53, `pyautogui` 0.9.54, `pywinauto` 0.6.9, `psutil`, `pywin32`, `webdriver-manager`, and `mysql-connector-python` 9.0.0.
- **Database:** MySQL 8.0.x running on `localhost:3306`. By default, the artifact uses the username `root` and password `1234`. These defaults can be overridden using the `DB_HOST`, `DB_USER`, and `DB_PASSWORD` environment variables. `makeDB.py` automatically creates the required database and tables.
- **Containers:** Docker Desktop with `docker compose` support.
- **TLS:** `mkcert` (installed automatically by `certs.ps1`).
- **Browsers:** Chrome 138, Firefox 139, Edge 136, Opera 118, Brave 1.79, and Whale 4.32 were used in the paper. Since several reported bugs have already been patched in newer releases, the artifact includes a version-pinned portable Brave browser for reproducibility. The minimal example downloads and uses this browser automatically. Note that Chrome must also be installed, because it serves as the baseline browser for all Chromium-based targets (see Section A.4.2).

A.2.5 Benchmarks

- **Datasets:** the test pages reused from prior work (DiffCSP, XSR-Framework) and WPT (Table 2 of the paper),

all included in the repository. No external download is required.

- **Evaluation metric:** pre/post-interaction enforcement inconsistencies, grouped into distinct bugs. See `example/` for the end-to-end reproduction and the expected output.

A.3 Set-up

A.3.1 Installation

1. Install Python 3.11, MySQL 8.0, and Docker Desktop. Ensure that both MySQL and Docker are running before continuing.
2. From the repository root, run `setup.ps1` in a PowerShell session running as Administrator. This script installs the required Python dependencies, downloads the Playwright browser binaries, and configures the test domains in the Windows `hosts` file.
   ```
   PowerShell -ExecutionPolicy Bypass
   -File setup.ps1
   ```
3. Generate the local TLS certificates used by the test servers:
   ```
   PowerShell -ExecutionPolicy Bypass
   -File certs.ps1
   ```
4. Create the database schema required by the artifact:
   ```
   python makeDB.py
   ```

A.3.2 Basic Test

As a sanity check, run the following commands to verify that the scenario generator, report server, fuzzer, and database are configured correctly. The example uses the SameSite policy on Chrome.

```
# start the report server
cd server/samesite/corpus && docker compose up

# generate depth-1 scenarios
python fuzzer/user_scenario_gen.py ^
    -s samesite -b chrome -d 1

# drive the browser
python fuzzer/fuzzer.py ^
    -s samesite -b chrome
```

Expected outcome. The scenario generator should print one `[+] Saved:` message for each generated scenario. Chrome should launch automatically and execute the generated interactions, eventually terminating with `[FIN] Fuzzing is over`. Successful execution also creates records with `event_type='interaction'` in the `event_entry` table.

Evaluators may stop the execution after confirming that the artifact successfully controls the BUI-level interaction. Completing the entire run is not necessary for this basic functionality test.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Detection of BUI-level policy-enforcement bugs. The artifact reproduces representative policy-enforcement bugs discovered by BUIzz and demonstrates its ability to identify enforcement inconsistencies introduced by BUI-level interactions. Experiments (E1) and (E2) validate this claim.

A.4.2 Experiments

Both experiments are performed using the minimal working example provided in `example/README.md`. The example uses a version-pinned portable Brave browser and a set of pre-configured scenarios that reproduce representative bugs from the paper. Unless stated otherwise, all commands should be executed from the `example/` directory.

One-time preparation. After completing the installation steps in Section A.3.1, run `setup.py` from the `example/` directory. The script downloads the portable Brave browser and prepares a browser profile with the required settings. The same settings can also be applied manually by following the instructions in `example/README.md`.

```
# Download the version-pinned Brave browser
python setup.py
```

Note on the baseline. `scenario_base_line.py` collects the pre-interaction baseline. For Chromium-based targets, the current implementation uses Chrome as the baseline browser, so the script locates the installed Chrome executable even when the target browser is Brave (`-b brave`). A regular Chrome installation must therefore be present in addition to the portable Brave browser prepared by `setup.py`.

(E1): SameSite split-view bug (CVE-2025-48980) [1 human-hour + 1 compute-minute]. This experiment reproduces the SameSite bypass triggered by "Open link in split view' on Brave (Claim C1; Figure 1, Idx #20 in Table 4).

Preparation: Complete the one-time preparation steps in Section A.4.2.

Execution:

```
# deploy the SameSite scenario

python mini_scenario.py 1

# start the SameSite report server

cd ..\server\samesite\corpus
docker compose up -d
cd ..\..\..\example

# collect the baseline
```

```
python scenario_base_line.py ^
-b brave -s samesite

# run OS-level interaction simulation

python ..\fuzzer\fuzzer.py ^
-s samesite -b brave

# compare pre/post-interaction behavior

python ..\analyzer.py ^
-s samesite -b brave

# group inconsistencies into bugs

python ..\deduping.py ^
-s samesite -b brave
```

Expected outcome: The analyzer prints a message such as `[strict] flagged N/M interaction rows` and writes `bugs/samesite/interaction_diff_brave.txt`. The deduplication step should report a distinct bug such as `(Open link in split view, , https:)`. The flagged record contains `violation: strict`, indicating that a `SameSite=Strict` cookie was sent on a cross-site navigation where it should have been blocked.

(E2): CSP split-view bugs [1 human-hour + 1 compute-minute]. This experiment reproduces CSP bypasses triggered by opening `blob:` or `data:` documents in split view on Brave (Claim C1; Idx #7 and Idx #6 in Table 4).

Preparation: Complete the one-time preparation steps in Section A.4.2. In addition, shut down the SameSite report server started in (E1) and remove the Docker containers and images it left behind before starting (E2). Resources from the SameSite environment interfere with the CSP test servers, so (E2) must be started from a clean Docker state.

Execution: For the `blob:` case, run:

```
python mini_scenario.py 2

cd ..\server\csp
docker compose up -d
cd ..\..\example

python scenario_base_line.py ^
-b brave -s csp1

python ..\fuzzer\fuzzer.py ^
-s csp1 -b brave

python ..\analyzer.py ^
-s csp -b brave

python ..\deduping.py ^
-s csp -b brave
```

For the `data:` case, repeat the same commands after replacing `mini_scenario.py 2` with `mini_scenario.py 3`.

Expected outcome: After "Open link in split view", the nonce-less script in the `blob:` or `data:` document executes, indicating that CSP was not inherited or that the navigation was improperly allowed.

Because BUIzz drives a real browser via OS-level input, do not move the mouse or switch windows while the fuzzer is running; the exact flagged counts may vary slightly across runs, but the distinct bugs should still be reproduced.

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/usenixsec2026/.