USENIX Security '25 Artifact Appendix: Effective Directed Fuzzing with Hierarchical Scheduling for Web Vulnerability Detection

Zihan Lin, Yuan Zhang, Jiarun Dai, Xinyou Huang, Bocheng Xiang, Guangliang Yang, Letian Yuan, Lei Zhang, Fengyu Liu, Tian Chen, and Min Yang

Fudan University

A Artifact Appendix

A.1 Abstract

This is the artifact evaluation appendix for paper "Effective Directed Fuzzing with Hierarchical Scheduling for Web Vulnerability Detection." In this work, we propose a novel directed fuzzing approach, called WDFuzz, that can effectively vet the security of Java web applications. Our WDFuzz approach is two-fold. First, we develop a semantic constraint extraction technique to accurately capture the expected input structures and constraints of web parameters. Second, we implement a hierarchical scheduling strategy that evaluates the potential of each seed to trigger vulnerabilities and prioritizes the most promising seeds. In our evaluation against real-world Java web applications, WDFuzz achieved a 92.6% recall rate in the known vulnerability dataset, finding 3.2 times more vulnerabilities and detecting them 7.1 times faster than the state-of-the-art web fuzzer Witcher.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The evaluation process does not raise any ethical concerns. Our benchmark dataset consists of older versions of web applications with known vulnerabilities. When deploying these applications on the evaluator's local machines, it is recommended not to allow external access to these applications to prevent attacks from external attackers.

A.2.2 How to access

The artifact can be downloaded from the Zenodo record[1]. After downloaded, first concatenate all the splitted compressed files with the following command:

```
cat WDFUZZ_part_* > WDFUZZ.tar.gz
```

Then unzip the file with this command:

```
tar xvf WDFUZZ.tar.gz
```

[1] https://zenodo.org/records/15128608

A.2.3 Hardware dependencies

At least 60 GB RAM and 200 GB free disk space are needed to run the Artifact Evalution. We run our experiments on a Ubuntu 18.04 server with 64-core Intel E7-4820 v2 2.00GHz CPU, 173 GB RAM and 1 TB disk.

A.2.4 Software dependencies

A Unix-like system is required for Artifact Evaluation. We recommend using Ubuntu 18.04. The other software dependencies are listed below.

- **Static Analysis**. Java 17[2] is required to run the Tai-e static analysis.
- **Instrumentation**. (1) Java 8[3] and (2) mvn[4] are needed to compile the instrumentation program. (3) GCC[5] is needed to compile the `LD_PRELOAD` binary.
- **Dynamic Fuzzing**. (1) Python3[6] and libraries (i.e., requests, pyyaml, and selenium) is needed to run the fuzzer's wrapper scripts. (2) Rust[7] is needed to run the core `LibAFL` fuzzer. (3) Docker[8] is needed to set up Web applications. (4) Chrome and chrome-driver[9] are needed to login web applications automatically.

A.2.5 Benchmarks

Our artifact contains 12 open-source web applications as benchmark dataset. Please refer to Table 1 in the paper for the detailed application list. The Docker images of these web applications are located in the `./dataset` directory within the artifact. To implement and setup these

[2] https://www.oracle.com/java/technologies/javase/jdk17-0-13-later-archive-downloads.html

[3] https://www.oracle.com/java/technologies/javase/javase8u211-later-archive-downloads.html

[4] https://maven.apache.org/download.cgi

[5] Use `sudo apt install build-essential` to install.

[6] Use `sudo apt install python3 python3-pip` to install.

[7] https://forge.rust-lang.org/infra/other-installation-methods.html

[8] https://docs.docker.com/engine/install/ubuntu/

[9] https://googlechromelabs.github.io/chrome-for-testing

applications for fuzzing, please refer to the document in `quick-reproduction-guide.pdf`. Due to legal and licensing reasons, we cannot provide the 3 closed-source commercial web applications tested in the paper for Artifact Evaluation. We believe that the experiment results from the open-source applications are sufficient to prove the claims in the paper.

A.3 Set-up

A.3.1 Installation

Python3, Rust, chrome and chrome-driver must be installed to run WDFuzz.

Python3 and library installation. First, we recommend using this command on Ubuntu servers to install Python3 environment:

```
sudo apt install python3 python3-pip
```

Then use this command to install necessary Python3 libraries:

```
pip install requests pyyaml selenium
```

Rust, chrome and chrome-driver installation. We recommend installing Rust, chrome and chrome-driver according to the instructions on their official websites, which are listed in the footnote.

Web applications installation. Before start fuzzing, web applications under test should be installed. We recommend to set up these web applications using the Docker images in the artifact (within `./dataset` directory) and following the guidance in `quick-reproduction-guide.pdf`. The baseline tool Witcher is also installed in the Docker images.

[Optional] Java, GCC, and mvn installation. Java, GCC, and mvn are *optional* to install because we have pre-compiled binaries for instrumentation and ready-to-use static analysis results. To recompile these components, please refer to the installation guide in `WDFuzz-README.pdf`.

A.3.2 Basic Test

Please run the following commands to check if necessary environments have been installed. The output examples are the outputs obtained in our test environment.

- **Check Rust environment.**
  Command:
  ```
  cargo --version
  ```
  Output example:
  ```
  cargo 1.79.0 (ffa9cf99a 2024-06-03)
  ```
  Command:
  ```
  rustc --version
  ```
  Output example:
  ```
  rustc 1.79.0 (129f3b996 2024-06-10)
  ```
- **Check Python3 environment.**
  Command:
  ```
  python3 --version
  ```
  Output example:
  ```
  Python 3.6.9
  ```
- **Check Docker environment.**
  Command:
  ```
  docker --version
  ```
  Output example:
  ```
  Docker version 20.10.21, build
  20.10.21-0ubuntu1~18.04.3
  ```
- **Check chrome-driver environment.**
  Command:
  ```
  chromedriver --version
  ```
  Output example:
  ```
  ChromeDriver 132.0.6834.83 ...
  ```
- **Check web applications under test.**
  You just need to use your browser to check if the websites can be successfully opened.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): WDFuzz achieves 92.6% recall in known vulnerability detection of web applications. Compared to the state-of-the-art web application fuzzer Witcher, WDFuzz finds 3.2 times more vulnerabilities and detects them 7.1 times faster. This is proven by the experiment (E1) described in §5.3 whose results are reported in Table 1.

(C2): Each module of WDFuzz (i.e., entry extraction, constraint extraction, and hierarchical scheduling) contributes to a notable increase in recall rates for known vulnerabilities, with improvements ranging from 20% to 30%. This is proven by the experiment (E2) described in §5.5 whose results are reported in Table 2.

A.4.2 Experiments

(E1): Known vulnerability reproduction [10 human-hours + 40 compute-hours + 100GB disk]: this experiment demonstrates that the proposed WDFuzz outperforms the state-of-the-art fuzzer Witcher in both vulnerability detection effectiveness and efficiency by reproducing the known vulnerabilities in the benchmark web applications.

Preparation: First, please make sure the environments being set up as described in §A.3.1, including Python3, Rust, Docker, chrome, chrome-driver, and web applications under test. Then some directories must be created before testing by running commands `mkdir $HOME/wdfuzz_data`, `mkdir $HOME/wdfuzz_data/fuzz_apps_class` and `mkdir $HOME/wdfuzz_data/jdf_data`.

Execution: Run `fuzz_<app>.sh` in `./DF` directory for the full experiment of WDFuzz, and run

`fuzz.sh` in the docker container of each web application for the Witcher experiment. You can refer to `quick-reproduction-guide.pdf` for more details.

Results: The vulnerability detection results of WDFuzz will be in the directory `$HOME/wdfuzz_data/<date>_<app>`. Each record in the text file `vuln_testcase.log` means that WDFuzz has detected a vulnerability. Please refer to the entry paths of the known vulnerabilities in Table 1 to get the recall rate of the known vulnerabilities. Vulnerabilities that are not in Table 1 are the unknown vulnerabilities detected by WDFuzz.

The results of Witcher experiment will be inside the docker container. Typically, the results are under the directory `/root/user/WICHR/<app>-WICHR`. Please check the `run_summary.txt` file to get the vulnerabilities identified by Witcher.

We provide a set of utility scripts within `log_comparison_scripts.zip`. For comparing WDFuzz's results, you can use the following command to extract the recall of known vulnerabilities from the logs: `python3 wdfuzz_log_compare.py --project <web app name> --log <path to vuln_testcase.log>`. For comparing Witcher's results, use the following command: `python3 witcher_log_compare.py --project <web app name> --log <path to run_summary.txt>`.

(E2): Ablation study [15 human-hours + 60 compute-hours + 30GB disk]: this experiment proves that each module of WDFuzz makes a notable contribution to vulnerability detection by evaluating the performance after ablating each module of WDFuzz.

Preparation: The same as the experiment (E1).

Execution: Run `fuzz_<app>_ablation<1 or 2 or 3>.sh` in `./DF` directory for the ablation study. Please note that for some web applications, CrawlerGo failed to crawl any usable URLs, and as a result there is no `ablation_1` experiment (i.e., the WDFuzz$_b$ baseline).

Results: The same as the experiment (E1).

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/usenixsec2025/.

Table 1: Known vulnerablilities.

Application	Vulnerable entry path

jeecg-boot	`/sys/duplicate/check`
	`/jmreport/queryFieldBySql`
	`/jmreport/show`
	`/sys/api/queryTableDictItemsByCode`
	`/sys/dict/queryTableData`

jeesite	`/a/act/task/start`
	`/userfiles`
	`/a/sys/user/export`

jshERP	`/user/list`
	`/msg/getMsgCountByStatus`
	`/msg/list`
	`/unit/list`
	`/role/list`
	`/account/list`
	`/depotHead/list`
	`/materialProperty/list`

MCMS	`/ms/mdiy/dict/list` (CVE-2021-46383)
	`/ms/cms/category/list`
	`/ms/mdiy/page/verify`
	`/ms/template/writeFileContent`
	`/mdiy/dict/list`
	`/ms/mdiy/dict/list` (CVE-2022-27466)
	`/cms/content/list` (CVE-2022-26585)
	`/ms/template/writeFileContent` (CVE-2021-46062)
	`/ms/template/writeFileContent` (CVE-2021-46063)
	`/cms/content/list` (CVE-2022-23898)
	`/ms/template/unZip`
	`/mdiy/dict/listExcludeApp`

RuoYi	`/system/dept/edit`
	`/demo/form/localrefresh/task`
	`/common/download/resource`
	`/common/download`
	`/system/role/authUser/unallocatedList`
	`/system/role/export`
	`/system/role/authUser/allocatedList`
	`/system/user/export`
	`/system/role/list`
	`/system/user/list`
	`/system/dept/treeData`

SpringBlade	`/blade-user/export-user`
Halo	`/api/admin/themes/fetching`
DreamerCMS	`/admin/search/doSearch` `/search`

PublicCMS	`/admin/ueditor?action=catchimage` (CVE-2020-20914)
	`/admin/sysSite/execSql`
	`/admin/ueditor?action=catchimage` (CVE-2021-27693)
	`/admin/#site_sysSite/script`
	`/admin/ueditor?action=catchimage` (CVE-2024-40543)
	`/admin/cmsTemplate/replace`

Yudao	No known vulnerabilities
lamp-boot	No known vulnerabilities

WebGoat	`/SqlInjection/attack8`
	`/SqlInjection/attack4`
	`/SqlInjection/attack2`
	`/SqlInjection/attack5`
	`/SqlInjectionAdvanced/challenge`
	`/SqlInjection/attack10`
	`/challenge/5`
	`/SqlInjection/attack3`
	`/SqlInjection/attack9`
	`/SqlInjection/assignment5a`
	`/SqlInjectionAdvanced/attack6a`
	`/SqlInjectionMitigations/servers`