Detecting Logical Bugs of DBMS with Coverage-based Guidance

Yu Liang, *Pennsylvania State University;* Song Liu, *Pennsylvania State University and Qi-AnXin Tech. Research Institute;* Hong Hu, *Pennsylvania State University*

https://www.usenix.org/conference/usenixsecurity22/presentation/liang

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

A Artifact Appendix

A.1 Abstract

This artifact is to help users reproduce the results we reported in our *USENIX Security 2022* paper submission. We recommend to run the artifact on a **x86-64** computer with ≥ 20 CPU cores, $\geq 600GB$ of memory and $\geq 1.5TB$ hard drive storage, and with an **Ubuntu 20.04 LTS** operating system. The artifact should reproduce all the Figures and Tables we reported in the paper, and thus can validate the main claims of the paper. Detailed execution steps are elaborated in the artifact *README.md* file.

A.2 Artifact check-list (meta-information)

- **Algorithm:** Coverage-based fuzzing, validity-oriented query mutation and DBMS oracle.
- **Program:** `SQLRight`. The program source code is included in the artifact.
- **Compilation:** afl-clang-fast and gcc-9/g++-9.
- **Binary:** Binaries not included. The programs are built from source.
- **Run-time environment:** OS: Ubuntu 20.04 LTS. Dependencies: python3 runtime and Docker. Requires Root access.
- **Hardware:** A **x86-64** computer with ≥ 20 CPU cores, $\geq 600GB$ of memory and $\geq 1.5TB$ hard drive storage. Hardware specs are publicly available.
- **Metrics:** The reported metrics are: Number of Bugs Detected, Fuzzing Coverage Feedback, Generated Query Validity and Number of Valid Statements per Hour.
- **Output:** All the Figures and Tables in the paper.
- **How much disk space required (approximately)?:** Around $1.0TB$ (10^{12} bytes).
- **How much time is needed to complete experiments (approximately)?:** Around 8834 CPU hours.
- **Publicly available (explicitly provide evolving version reference)?:** Publicly available on Github.
- **Code licenses (if publicly available)?:** MIT License
- **Archived (explicitly provide DOI or stable reference)?:** Yes. Stable reference: `https://github.com/psu-security-universe/sqlright-artifact/tree/57978e5ce697e13414a2bca871d2ef874e77158d`

A.3 Description

A.3.1 How to access

The artifact can be retrieved from Github.

The Github link to the artifact is: `https://github.com/psu-security-universe/sqlright-artifact/tree/57978e5ce697e13414a2bca871d2ef874e77158d`.

A.3.2 Hardware dependencies

The artifact evaluations are run on a **x86-64** computer, recommended with ≥ 20 CPU cores, $\geq 600GB$ of memory and $\geq 1.5TB$ hard drive storage.

A.3.3 Software dependencies

The artifact is evaluated on an **Ubuntu 20.04 LTS** operating system.

A.3.4 Data sets

N/A.

A.3.5 Models

N/A.

A.3.6 Security, privacy, and ethical concerns

N/A.

A.4 Installation

To run the artifact code, user should download the artifact files from the Github website (link provided from above). The *README.md* file contains the detailed instructions to install the Docker environment, and further build the Docker Images required for the fuzzing tests.

A.5 Experiment workflow

The experiments are being hosted inside the Docker virtualized environment. User only needs to call a few scripts guided by the *README.md* file, and the scripts will run the fuzzing evaluations in the background and later generate all the Figures and the Tables we presented in the paper.

A.6 Evaluation and expected results

Here is the main claims of the paper:

- The proposed tool `SQLRight` can find more bugs than State-of-the-arts `SQLancer` and `Squirrel`$_{\texttt{+oracle}}$. `SQLRight` also outperforms existing tools in triggering more program code. This claim can be validated by *Figure 5* and *Figure 8*.
- The Coverage-based guidance helps `SQLRight` generate more diverse queries and accumulate useful mutations, which helps discover more bugs than the no-feedback baselines. This claim can be validated by *Figure 6* and *Table 3*.
- The validity-oriented optimizations in SQLRight can help generate higher validity queries, reduce false positives, and ultimately help discover more bugs. This claim can be validated by *Figure 7*, *Figure 9* and *Table 4*.

Following the instructions provided by the *README.md* files in the artifact, one should be able to independently reproduce all the results (Figures, Tables) shown in our paper. Specifically:

- **Session 3** in the *README.md* contains the instructions to evaluate Comparison with Existing Tools (*Section 5.2* in the paper). It includes the steps to generate the figures from *Figure 5* and *Figure 8* in the paper. It consumes about 6152 CPU hours.
- **Session 4** in the *README.md* contains the instructions to evaluate Contribution of Coverage Feedback (*Section 5.3* in the paper). It includes the steps to generate *Figure 6* and *Table 3* in the paper. It consumes about 726 CPU hours.
- **Session 5** in the *README.md* contains the instructions to evaluate Contribution of Validity (*Section 5.4* in the paper). It includes the steps to generate *Figure 7*, *Figure 9* and *Table 4* in the paper. It consumes about 1956 CPU hours.

The detailed command instructions are elaborated in the *README.md* file. Here we show the expectations for each artifact generated figures/tables:

- **Figure 5a** `SQLite` logical bugs: `SQLRight` should detect the most bugs. On different evaluation around, we expect ≥ 3 bugs being detected by SQLRight in 72 hours.
- **Figure 5b** `MySQL` logical bugs: The current bisecting and bug filtering scripts could slightly over-estimate (or under-estimate) the number of unique bugs for MySQL. Some manual efforts might be needed to scan through the bug reports and deduplicate the bugs to get the most accurate unique bug number. But in general, `SQLRight` should report the most bugs after bisecting (≥ 2 bugs in 72 hours).
- **Figure 5c-e** `SQLite`, `MySQL` and `PostgreSQL` code coverage: `SQLRight` should have the highest code coverage among the other baselines.
- **Figure 5f** `SQLite` query validity: `SQLancer` has the highest query validity, while `SQLRight` performs better than $\texttt{Squirrel}_{\texttt{+oracle}}$.
- **Figure 5g** `MySQL` query validity: sys has higher validity than $\texttt{Squirrel}_{\texttt{+oracle}}$.
- **Figure 5h** `PostgreSQL` query validity: `SQLancer` has the highest query validity, while `SQLRight` performs better than $\texttt{Squirrel}_{\texttt{+oracle}}$.
- **Figure 5i** `SQLRight` valid statements per hour: `SQLancer` has the highest number of valid statements per hour, while `SQLRight` performs better than $\texttt{Squirrel}_{\texttt{+oracle}}$.
- **Figure 5j** `MySQL` valid statements per hour: `SQLRight` has more valid statements per hour than $\texttt{Squirrel}_{\texttt{+oracle}}$.
- **Figure 5k** `MySQL` valid statements per hour: `SQLancer` have the highest valid statements per hour, while `SQLRight` performs better than $\texttt{Squirrel}_{\texttt{+oracle}}$.
- **Figure 6a-b** bugs of `SQLite` (`NoREC` and `TLP`): `SQLRight` should detect the most bugs. On different evaluation around, we expect ≥ 2 bugs being detected by `SQLRight` in 24 hours.
- **Figure 6c-d** coverage of `SQLite` (`NoREC` and `TLP`): `SQLRight` should have the highest code coverage among the other baselines.
- **Figure 7a** `SQLite` logical bugs: `SQLRight` should detect the most bugs. On different evaluation around, we expect ≥ 2 bugs being detected by SQLRight in 24 hours. Additionally, we have muted the $\texttt{SQLRight}_{\texttt{-deter}}$ config in the Artifact logical bugs figure. Because sometimes $\texttt{SQLRight}_{\texttt{-deter}}$ could produce tens of False Positives, which would destroy the plot region and render the script outputs an unreadable plots.
- **Figure 7b** `MySQL` logical bugs: The current bisecting and bug filtering scripts could slightly over or under-estimate the number of unique bugs for MySQL. Some manual efforts might be needed to scan through the bug reports and deduplicate the bugs to get the most accurate unique bug number. In general, SQLRight should report the most bugs after bisecting. On different evaluation around, we expect ≥ 1 bugs from `SQLRight` in 24 hours. Additionally, we have muted the $\texttt{SQLRight}_{\texttt{-deter}}$ config in the Artifact logical bugs figure. Because sometimes $\texttt{SQLRight}_{\texttt{-deter}}$ could produce tens of False Positives, which would destroy the plot region and render the script outputs an unreadable plots.
- **Figure 7c-e** `SQLite` code coverage: `SQLRight` and $\texttt{SQLRight}_{\texttt{-deter}}$ should have the highest code coverage among the other baselines. $\texttt{SQLRight}_{\texttt{-ctx-valid}}$ could have a coverage very close to the SQLRight config, but in general, $\texttt{SQLRight}_{\texttt{-ctx-valid}}$ is slightly worse in coverage compared to `SQLRight`.
- **Figure 7f-h** `SQLRight` and $\texttt{SQLRight}_{\texttt{-deter}}$ should have the highest query validity.
- **Figure 7i-k** `SQLRight` and $\texttt{SQLRight}_{\texttt{-deter}}$ should have the highest number of valid statements per hour.
- **Figure 8a** `SQLite` logical bugs: `SQLRight` should detect the most bugs. On different evaluation around, we expect ≥ 1 bugs being detected by SQLRight in 72 hours.
- **Figure 8b** `MySQL` logical bugs: The current bisecting and bug filtering scripts could slightly over-estimate (or under-estimate) the number of unique bugs for MySQL. Some manual efforts might be needed to scan through the bug reports and deduplicate the bugs to get the most accurate unique bug number. But in general, `SQLRight` should reported the most bugs after bisecting (≥ 1 bugs in 72 hours).
- **Figure 8c-8e** `SQLite`, `MySQL` and `PostgreSQL` code coverage: `SQLRight` should have the highest code coverage among the other baselines.
- **Figure 8f-h** `SQLite`, `MySQL` and `PostgreSQL` query validity: `SQLancer` has the highest query validity, while `SQLRight` performs better than $\texttt{Squirrel}_{\texttt{+oracle}}$.
- **Figure 8i-k** `SQLite`, `MySQL` and `PostgreSQL` valid statements per hour: `SQLancer` has the highest number of valid statements per hour, while `SQLRight` performs better than $\texttt{Squirrel}_{\texttt{+oracle}}$.
- **Figure 9a** `SQLite` logical bugs: `SQLRight` should detect the most bugs. On different evaluation around, we expect ≥ 2 bugs being detected by `SQLRight` in 24 hours. Additionally, we have muted the $\texttt{SQLRight}_{\texttt{-deter}}$ config in the Artifact logical bugs

figure. Because sometimes `SQLRight-deter` could produce tens of False Positives, which would destroy the plot region and render the script outputs an unreadable plots.

- **Figure 9b** `MySQL` logical bugs: The current bisecting and bug filtering scripts could slightly over or under-estimate the number of unique bugs for MySQL. Some manual efforts might be needed to scan through the bug reports and deduplicate the bugs to get the most accurate unique bug number. In general, `SQLRight` should detect the most bugs after bisecting. On different evaluation around, we expect ≥ 1 bugs being reported by `SQLRight` in 24 hours. Additionally, we have muted the `SQLRight-deter` config in the Artifact logical bugs figure. Because sometimes `SQLRight-deter` could produce tens of False Positives, which would destroy the plot region and render the script outputs an unreadable plots.
- **Figure 9c-e** `SQLite`, `MySQL` and `PostgreSQL` code coverage: `SQLRight` and `SQLRight-deter` should have the highest code coverage among the other baselines. `SQLRight-ctx-valid` could have a coverage very close to `SQLRight`, but in general, `SQLRight-ctx-valid` is slightly worse in coverage compared to `SQLRight`.
- **Figure 9f-h** `SQLite`, `MySQL` and `PostgreSQL` query validity: `SQLRight` and `SQLRight-deter` should have the highest query validity.
- **Figure 9i-h** `SQLite`, `MySQL` and `PostgreSQL` valid statements per hour: `SQLRight` and `SQLRight-deter` should have the highest number of valid statements per hour.
- **Table 3** Code coverage triggered by queries with different depths: The mutation depth number could be slightly different between each run. However, the **Max Depth** from `SQLRight NoREC` and `TLP` should be larger than other baselines. And `SQLRight NoREC` and `TLP` should have more queue seeds located in a deeper depth, compared to other baselines.
- **Table 4** False Positives from **Non-Deter**: We have introduced some extra filters that can filter out some obvious False Positives. We includes these filters in the Artifact implementation, in order to reduce the manual efforts for excluding FPs, and to produce a more accurate bug numbers by default. Therefore, the bug number reported by the current Artifact script could be slightly different from the ones we reported in the paper (**Table 4**). For all configurations, the **WITHOUT non-deter** settings should always have less bugs reported compared to the **WITH non-deter** settings, due to the extra False Positives produced by the non-deterministic queries.

A.7 Experiment customization

N/A

A.8 Notes

N/A

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.