SQIRL: Grey-Box Detection of SQL Injection Vulnerabilities Using Reinforcement Learning

Salim Al Wahaibi, Myles Foley, and Sergio Maffeis, *Imperial College London*

https://www.usenix.org/conference/usenixsecurity23/presentation/al-wahaibi

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

This artifact contains the codebase to run SQIRL, a novel approach to detecting SQL injection vulnerabilities using deep reinforcement learning with multiple worker agents. Each worker intelligently fuzzes the input fields discovered by an automated crawling component. It also includes all the code required to run the different versions of SQIRL including its random (RAND-SQIRL), and federated (FED-SQIRL) variants. The requirements to create the SQLiMicroBenchmark (SMB) are also included. We further detail how to run SQIRL on the SMB in order to reproduce the results found in the main body of the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

To reduce the effect of possible security concerns the SMB runs inside a docker container. The SMB should *not* be exposed on externally available ports as the SMB contains deliberately vulnerable samples.

SQIRL has the potential to find zero-day SQLi vulnerabilities in web applications. However, we have designed SQIRL to be a greybox tool that requires privileged access to the webapplication-under-test. This prevents malicious user from using this on targets that are unware they are being tested.

A.2.2 How to access

We release the artifact as a repository, a stable version of which can be found at: https://github.com/ICL-ml4csec/SQIRL/tree/5a444ee7782a33a097f345fca837125ac2505ee0

A.2.3 Hardware dependencies

None

A.2.4 Software dependencies

- Python >= 3.8.16
- Python pip >= 23.0.1
- Docker Engine >= Docker version 23.0.5, build bc4487a
- Docker Compose version >= v2.17.3

A.2.5 Benchmarks

The SQLiMicroBenchmark or SMB is provided in the artifact in the directory `SQLiMicroBenchmark/`. This requires Docker in order to be run, and tested on.

A number of baselines are used as a point of comparison to SQIRL. We do not document here how to install and setup these baselines, we refer the reader to the relevant baselines if they wish to compare against these. Please see the Appendix in the main paper for the configuration used for the baselines.

Baselines:

- OWASP ZAP v2.11.1
- Sqlmap v1.6
- BurpSuite Pro v2022.6.1
- Arachni v1.6.1.3
- Wapiti 3.1.2

If user wish to experiment with the production grade web applications we provide here the list of web applications and plugins where relevant:

1. WordPress core v6.0 and plugins (Download Monitor WordPress V 4.4.4, WP User Frontend 3.5.25, Sliced Invoices 3.8.2, Plugin Photo Gallery 1.5.34, Supsystic Ultimate Maps 1.1.12, WP Statistics 13.0.7, JoomSport)
2. B2evolution v7.2.3-stable
3. BBpress v2.6.9
4. Big tree CMS v4.4.16
5. Drupal v9.3.18
6. Joomla v4.2.0
7. Admidio v4.0
8. Gila CMS
9. Media wiki v1.38.2
10. Pbboard v3.0.3
11. Impresscms v1.4.4
12. WackoWiki v6.0.31
13. Sourcecodester E-learning System v1.0,
14. Sparks Hotel Management System v1.0

A.3 Set-up

A.3.1 Installation

Clone Repository:

```
git clone https://github.com/ICL-ml4csec/SQIRL
```

Install docker: If the docker engine and docker compose requirements are not already met, then install them. Instructions for this can be found here.

Setup SMB: From the `SQLiMicroBenchmark/` directory create the required docker containers: `docker-compose up -d`. This can take a minute or two to install and configure. Note that one of the containers 'db-seeder' will sleep for 40 seconds before configuring the 'db' docker container, and will then exit. This is the intended behaviour, after the configuration containers 'db' and 'php-apache' should be running.

The log file required by SQIRL must be edited to provide read and write access `chmod +rw mysql/general.log`.

SQIRL dependencies: We recommend running the SQIRL framework from a virtual Python environment. We recommend using conda, which can be installed following the guide here. Note conda is not required and other frameworks such as poetry or venv can be used. After install create the conda environment:

- `conda create -name sqirl python=3.9`

Figure 1: Expected result for basic test of SQIRL, showing help options.

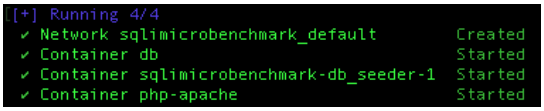

Figure 2: Expected result for basic test of SMB, showing all containers have started.

SQIRL packages: Installing the SQIRL packages with pip can be done using the following command: `conda activate sqirl && pip install -r requirements.txt`.

A.3.2 Basic Test

SQIRL: `python sqirl.py -help`

This should startup SQIRL and then display the flags that can be used when running, as shown in Figure 1.

SMB: `cd SQLiMicroBenchmark && docker-compose up -d && cd ..`

This will start the containers required by the SMB, resulting in the expected result shown in Figure 2

A.4 Evaluation workflow

We provide here the framework and instructions for using SQIRL and the SMB.

A.4.1 Major Claims

(C1): SQIRL is able to find more vulnerabilities than existing state-of-the-art-scanners and achieve 0 false positives.

```
[!] reached win criteria
[!] payload: 131149265861 AND  SLEEP (0)
[!] URL: http://localhost:8000/functions_external/sqli1.php
[!] Parameter: name
```

Figure 3: Example of a vulnerability identified by SQIRL and shown in the log files `results_stats_X.stats`, where `X` is the worker agent that found the vulnerability.

(C2): SQIRL is able to find vulnerabilities in a lower number of requests than other scanners.

A.4.2 Experiments

(E1): [10 human minutes + 20 compute hours + 16GB disk + 6CPU]: train SQIRL on the SMB to ensure training functionality.

How to: First the SMB and python environment must be set up. Then SQIRL can be run to start training. After training has finished the log files and trained model can be seen in a new sub-directory in `stats_logs`.

Preparation: Activate the docker container `docker-compose up -d`. Then activate the environment for SQIRL: `conda activate sqirl`.

Execution: Run from a terminal window Example **A** in Table 1. Note this should run as a single command and copying directly from the Table may cause an error due to a newline.

Results: Each agent should finish running, after which the worker server can be closed. There will be a new directory in `stats_logs` that will contain a new model in addition to log data.

(E2): [10 human minutes + 1 compute hour + 16GB disk + 6CPU]: Test SQIRL on the SMB to ensure get test results functionality.

How to: First the SMB and python environment must be set up. Then the SQIRL can be run to test for the SQLi in the SMB. SQIRL will create a new sub-directory in `stats_logs` containing log files and the model checkpoint that resulted from the new run.

Preparation: Activate the docker container `docker-compose up -d`. Then activate the environment for sqirl: `conda activate sqirl`.

Execution: Run from a terminal window Example **B** in Table 1. Note that the `dir_from_training` should be changed to that from training in Experiment **E1**. Where the `save_dir` is the resulting directory from the first experiment.

Results: In the new sub-directory in `stats_logs` the log file '`results_stats_1.stats`' will contain the vulnerabilities found by SQIRL. These are identified by the pattern shown in Figure 3. The number of requests used to find these vulnerabilities can then be found by accessing `http://localhost:8000/server-status` and is identified by the total number of accesses.

Table 1: Example commands used to run SQIRL. **A**: Training SQIRL with 4 worker agents, **B**: Testing SQIRL with 1 worker agent.

```
A: python3 sqirl.py -u http://localhost:8000/training.php --log_file ./SQLiMicroBenchmark/mysql/general.log \
        --loss_criteria 200 --win_criteria 14 --agent 4 -i 4
B: python3 sqirl.py -u http://localhost:8000/no_feedback.php --log_file  ./SQLiMicroBenchmark/mysql/general.log \
        --agent 4 --model_dir ./stats_logs/dir_from_training/Checkpoint_Worker_Server/
```

A.5 Notes on Reusability

SQIRL is designed to be independent of the SQL database that is being tested. We have developed SQIRL to work with `mysql v5.X`, this can be extended by adding in the ability to parse the required logs to the SQL Proxy (`SQL_Proxy.py`). Any tokens specific to the database syntax would also be required by the environment in order to generate syntactically correct payloads.

Note that for newer versions of `mysql` the error logging functionality changed to include malformed queries in the general log. This can lead to SQIRL producing false positives so it is advised to use `mysql v5.X` when testing SQIRL.

A.6 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/.