

# Hakuin Artifact Evaluation

Link to the Artifact:

<https://zenodo.org/record/7752413>

## Abstract

Hakuin is a framework for optimizing the inference of textual data with Blind SQL Injection (BSQLI). In our experiments, we used two databases (DB) called the schema DB and the generic DB to measure how many requests does Hakuin (and other tools) need to exfiltrate them (RQ1 and RQ2). In addition, we measured how many requests per character (RPC) it takes to extract the data and how this improves throughout the inference process (RQ3). This artifact evaluation has three objectives. First, it opens the source codes for the Open Research Objects badge. Second, it shows how to reproduce the results presented in our paper (RQ1 and RQ2). Lastly, it provides the RPC measurements data for verification purposes (RQ3).

## Instructions

The repository has the following structure:

```
/data
  /corpora          # all the corpora mentioned in the paper (Section III-B)
  /models           # the schema models (Section III-B)
/experiments
  /dbanswers        # the schema DB (Section IV-A)
  /generic_db       # the generic DB (Section IV-A)
  /www              # vulnerable web application (Section IV-B)
/hakuin             # framework source code
```

### Objective #1 - Source Code Review

Please, see the /hakuin directory.

### Objective #2 - Reproducing Hakuin's Results (RQ1 and RQ2)

The original evaluation looked like this. We locally set-up a web application vulnerable to SQLI (see /experiments/www), ran Hakuin, and measured the number of requests in the application itself.

To simplify the artifact evaluation, we moved the web application functionality directly to our framework, so the SQL queries run offline in the same process as the tool. This is much faster and easier to set up and it does the same thing.

Installation and execution:

```
pip3 install . -e # this will install Hakuin in the editable mode
cd ./experiments
python3 experiment_schema_db_offline.py # this will output the number of schema requests
# and RPC (RQ1: Section IV-C, Table II)
python3 experiment_generic_db_offline.py # this will output the number of generic DB
# requests and RPC (RQ2: Section IV-C, Table III)
python3 experiment_generic_db_offline.py <table> <column> # same as above but for a
# specific column
```

## Objective #2 - Reproducing the Results of Other tools (RQ1 and RQ2)

To measure the other tools, you need to set up a vulnerable web application:

```
pip install fastapi # installs fastapi
pip install "uvicorn[standard]" # installs uvicorn
cd ./experiments/www
uvicorn main:app --port 8000 --reload # this will run the web application
```

### SQLMap:

- 1) Download and install SQLMap version 1.6.11.7-dev according to the instructions in its [GitHub repository](#).
- 2) Choose a generic DB column and run:  

```
python sqlmap.py -u 'http://127.0.0.1:8000/?name=George' --level 5 --risk 3
--dbms SQLite --technique B --threads 10 --dump -C <col> -T <table>
```
- 3) The tool will stop and ask the following question:

```
[17:22:27] [INFO] testing connection to the target URL
[17:22:28] [INFO] checking if the target is protected by some kind of WAF/IPS
[17:22:28] [INFO] testing if the target URL content is stable
[17:22:28] [INFO] target URL content is stable
[17:22:28] [INFO] testing if GET parameter 'name' is dynamic
[17:22:28] [WARNING] GET parameter 'name' does not appear to be dynamic
[17:22:28] [WARNING] heuristic (basic) test shows that GET parameter 'name' might not be injectable
[17:22:28] [INFO] testing for SQL injection on GET parameter 'name'
[17:22:28] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[17:22:28] [INFO] GET parameter 'name' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --code=200)
[17:22:28] [INFO] checking if the injection point on GET parameter 'name' is a false positive
GET parameter 'name' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
```

- 4) Do not answer the question yet. At this point, SQLMap already sent some requests to fingerprint the DBMS, so to make the measurement fair, you need to reset the request counter. To do this, access "<http://127.0.0.1:8000/reset>" from your browser.
- 5) Answer the question with "n"
- 6) Once the inference stops, visit "<http://127.0.0.1:8000/counter>" and write down the measurement
- 7) Repeat the process for all columns

- 8) Repeat the process for schema:

```
python sqlmap.py -u 'http://127.0.0.1:8000/schemas?name=1=1' --level 5 --risk 3  
--dbms SQLite --technique B --threads 10 --schema
```

### BBQSQL:

- 1) Download and install BBQSQL version 1.2 according to the instructions in its [GitHub repository](#).
- 2) Copy “bbqsql\_benchmark.py” from the “experiments” directory in Hakuin’s repository to the “scripts” directory in the BBQSQL’s repository.
- 3) Infer the schema:  

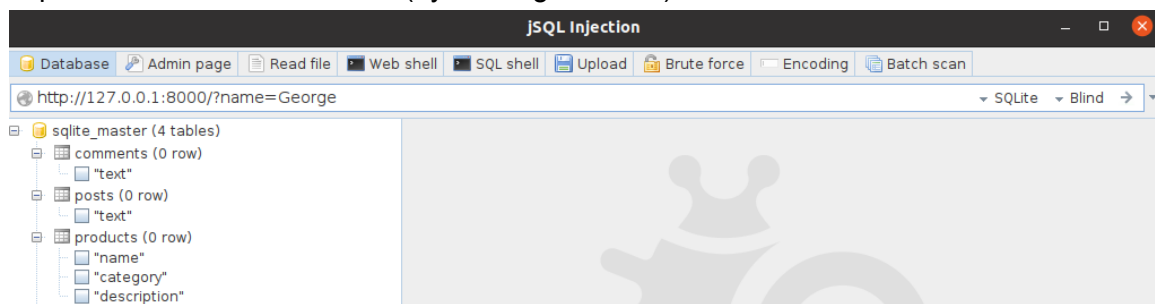
```
python 2 ./scripts/bbqsql_benchmark.py schema # note that its python 2 not 3
```
- 4) Read and reset the counter in the web application
- 5) Infer a generic DB column:  

```
python 2 ./scripts/bbqsql_benchmark.py <table> <column>
```
- 6) Repeat step 5) for all columns and read the counter in the web application

### jSQL Injection:

- 1) Download jSQL Injection version 0.84 from its [GitHub repository](#) and make sure you have a Java runtime installed.
- 2) Launch the tool:  

```
java -jar jsq-injection-v0.84.jar
```
- 3) Set the URL line of the tool to “http://127.0.0.1:8000/?name=George”, the DBMS to “SQLite”, and hit enter
- 4) Expand all columns on the left (by clicking on them):



- 5) Reset the counter in the web application
- 6) Select a single generic DB column, then right-click on its table name and click “load”
- 7) Wait until the inference finishes and then read the counter in the web application
- 8) Repeat step 5) for each column
- 9) To infer the schema, relaunch the tool, change the URL to “http://127.0.0.1:8000/schemas?name=1=1”, set the DBMS to SQLite, and hit enter
- 10) Before you expand the columns on the left, reset the counter in the web application (to exclude the fingerprinting requests)
- 11) Expand the columns on the left
- 12) Read the counter from the web application

