Optimization Research on Initial Seeds of the Database Management Systems Fuzzing Framework Squirrel

Optimization Research on Initial Seeds of the Database Management Systems Fuzzing Framework Squirrel

Manning Song[1*]

[1] School of Information Science & Engineering, Yunnan University, Kunming, Yunnan, 650500, China

[*]Corresponding author's e-mail: songmanning@mail.ynu.edu.cn

Abstract. Squirrel is one of the most effective fuzzing frameworks for finding memory errors in database management systems. But its efficiency is affected by initial seeds significantly. In order to improve the fuzzing efficiency of Squirrel, we optimized its initial seeds. The main factors affecting its efficiency are the average grammatical distance and the number of texts of initial seeds. Increasing these two factors at the same time can greatly improve the fuzzing efficiency of Squirrel.

1. Introduction

Database management systems(DBMSs) are used in all walks of life all over the world[1,2]. DBMSs have brought great convenience to our work and life. At the same time, errors of DBMSs have also brought threats to our information security. Fuzzing can find bugs in DBMSs[3,4]. But it is hard to generate semantics-correct Structured Query Language (SQL) statements to test DBMSs[5]. Squirrel provides a relatively effective method to solve this problem[6]. But like many fuzzing framework, the choice of initial seeds can affect its fuzzing efficiency[7]. There are questions over what attributes of initial seeds can affect the efficiency and how can we optimize the initial seeds.

2. Materials and methods

2.1. The method to calculate the average grammatical distance

We define the grammatical distance between two SQL texts as the Levenshtein distance[8] between the SQL grammatical structures of them.

This paper presents an efficient method for getting the SQL grammatical structures from SQL texts. (1) We classify tokens in SQL texts into two groups. SQL keywords and mathematical operators are called structure. Other tokens are called data[6]. (2) Extra spaces and the data in SQL texts are deleted, such as table names, column names, and constants. In SQL statements, keywords separated by one space or multiple spaces have the same effect. Therefore, the extra spaces are removed. In this way, keywords are separated by only one space. (3) The SQL keyword groups are joined by '_', such as 'ORDER_BY' and 'BETWEEN_AND'. (4) The structure is split with the space ' '. (5) The grammatical distance is calculated with the Levenshtein distance algorithm. An element in the structure list is just like a character in a string.

The average grammatical distance(AGD) of the initial seed ={SQL1, SQL2, ... , SQLn} can be expressed as:

$$AGD = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}}{n(n-1)} \quad (1)$$

The d_{ij} in equation (1) is the grammatical distance between SQLi and SQLj.

2.2. The methods to optimize initial seeds

In Squirrel, mutating the structure one time means that inserting, replacing or deleting the optional operands one time. If the the grammatical distances between SQL grammatical structures in initial seeds are small, Squirrel will be more likely to generate the same grammatical structure soon. It can be inferred that increasing the AGD of initial seeds can improve the fuzzing efficiency of Squirrel.

In Squirrel, it accepts SQL texts in initial seeds to initialize the intermediate representation(IR) library. More SQL texts with different grammatical structures in initial seeds usually means more IR types in library. If there are SQL texts with the same grammatical structure in initial seeds, only one of these SQL texts will be kept. It can be inferred that increasing the number of SQL texts in initial seeds can improve the fuzzing efficiency of the Squirrel.

In conclusion, the fuzzing efficiency of Squirrel can be improved by increasing the AGD of initial seeds and the number of SQL texts in initial seeds.

2.3. The seeds in experiments

200 SQL texts are selected from the SQL data set as the candidate seed. This operation are repeated multiple times. In this way, there is a list of candidate seeds. The candidate seed with higher AGD is chosen as the optimized seed. Initial seeds for SQLite and PostgreSQL are optimized in experiments. The initial seeds provided by Squirrel are the seeds before optimization. The attributes of seeds are shown in Table 1.

Table 1. Attributes of seeds.

Attributes	SQLite		PostgreSQL	
	Before optimization	After optimization	Before optimization	After optimization
AGD	9.64	15.75	4.45	106.99
The number of SQL texts	30	200	16	200

2.4. Experimental configuration

The SQLite version is 3.35.0. The PostgreSQL version is 13.2. Both SQLite and PostgreSQL were evaluated in the Ubuntu 19.10 operating system on a computer with Intel® Xeon(R) CPU E7-4850 v2 @ 2.30GHz, 8 cores and 24G RAM. Each fuzzing instance of SQLite ran with 1 CPU and 4G RAM for 12 hours. Each fuzzing instance of PostgreSQL ran in docker with the Ubuntu 16.04 operating system, 1 CPU and 4G RAM for 12 hours. These processes were repeated 5 times. The average results are reported.

The fuzzing efficiency is mainly measured with the path coverage. The more the path coverage, the higher the fuzzing efficiency.

3. Results and discussion

3.1. The results of optimizing initial seeds

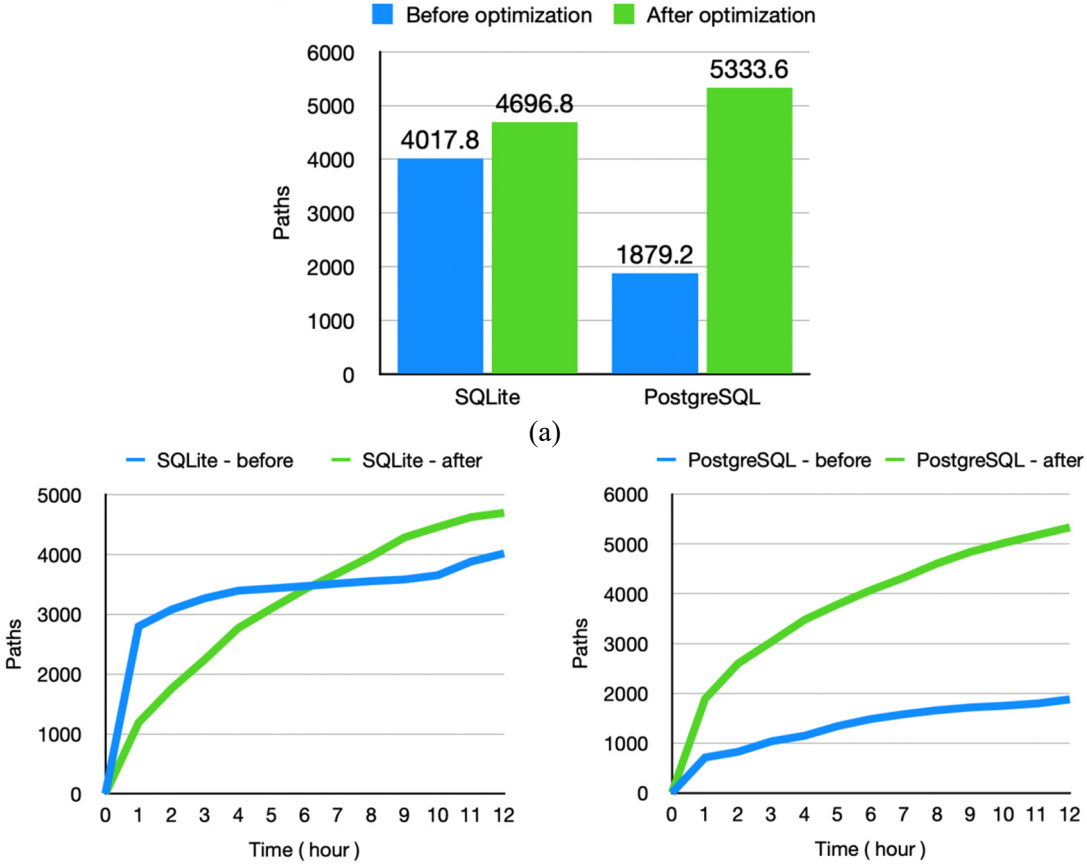

Figure 1. Results of optimization. (a) The total paths of fuzzing SQLite and PostgreSQL within 12 hours. (b) Compare different seeds in SQLite. (c) Compare different seeds in PostgreSQL.

The total paths of fuzzing SQLite and PostgreSQL in 12 hours are shown in Figure 1 (a). SQLite and PostgreSQL were tested with initial seeds before optimization and after optimization separately. When SQLite was tested, the number of paths increased from 4,017.8 to 4,696.8 within 12 hours. The fuzzing efficiency increased by 16.90%. When PostgreSQL was tested, the number of paths increased from 1,879.2 to 5,333.6 within 12 hours. The fuzzing efficiency increased by 183.82%.

The results of fuzzing SQLite and PostgreSQL with different initial seeds within 12 hours are shown in Figure 1 (b) and (c). In the first 6 hours, when SQLite was tested, the test efficiency of the seed before optimization is higher. But after that, the optimized seed is more efficient. In addition, when SQLite was tested with the optimized seed, 3 crashes were triggered within 12 hours. When PostgreSQL was tested, the optimized seed is always more efficient than the seed before optimization.

The results of experiments are summarized as follows. In general, the fuzzing efficiency of the optimized seeds are higher than the fuzzing efficiency of the seeds before optimization. Whether testing SQLite or PostgreSQL, initial seeds can be optimized by increasing the AGD of initial seeds and increasing the number of texts in initial seeds, which can effectively improve the fuzzing efficiency of Squirrel. This is generally sufficient to produce good results.

3.2. *Contributions of attributes of initial seeds*

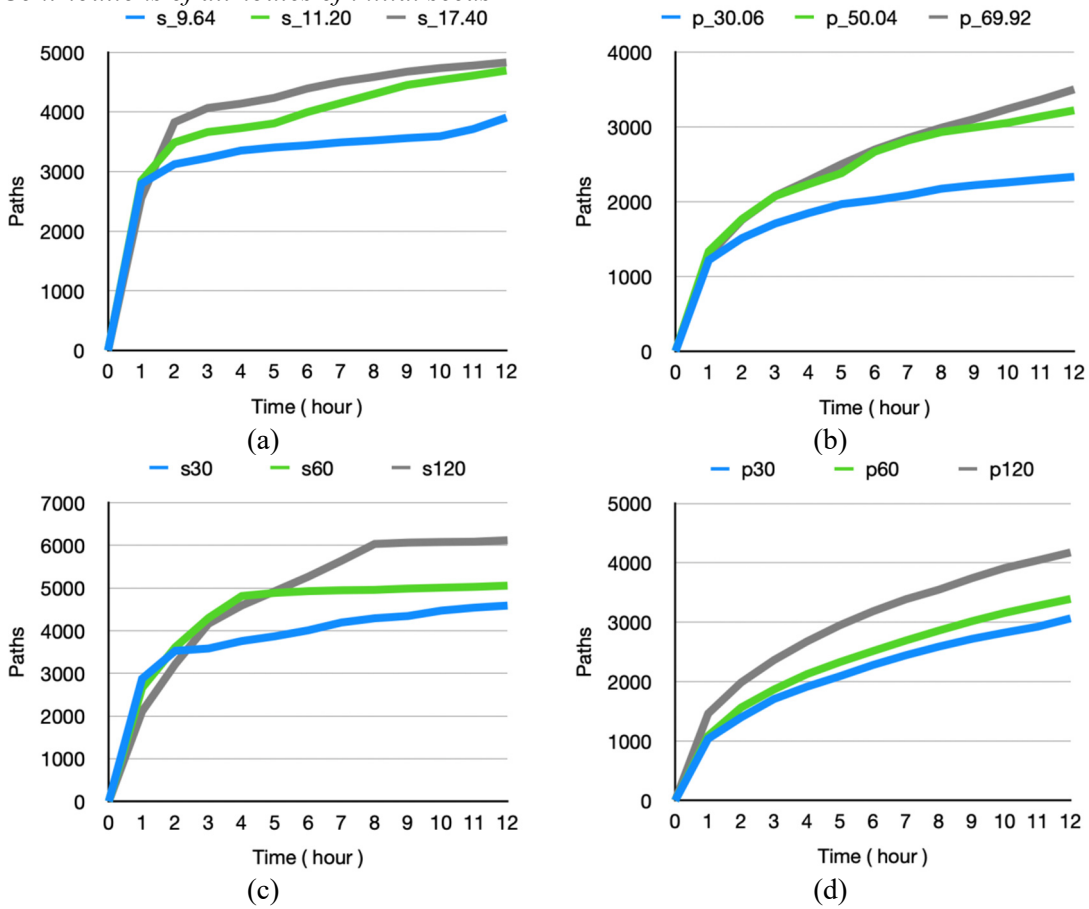

Figure 2. Contributions of attributes of initial seeds. (a) Fuzzing SQLite with initial seeds with different AGD, but the number of SQL texts is 30. (b) Fuzzing PostgreSQL with initial seeds with different AGD, but the number of SQL texts is 15. (c) Fuzzing SQLite with initial seeds with different number of SQL texts, but the AGD is 14.37. (d) Fuzzing PostgreSQL with initial seeds with different number of SQL texts, but the AGD is 84.04.

As shown in Figure 2 (a) and (b), when SQLite and PostgreSQL were tested with initial seeds with different AGD but the same number of SQL texts, the higher the AGD, the higher the efficiency. The overall shapes of the curves of the same DBMS are similar. The larger the AGD, the higher the curve.

As shown in Figure 2 (c) and (d), when SQLite and PostgreSQL were tested with seeds with different number of SQL texts but the same AGD, the higher the number of SQL texts, the higher the efficiency. In SQLite, the initial seeds with the number of SQL texts in 30, 60 and 120, respectively, after 2, 4, and 8 hours, the fuzzing efficiency begins to decrease significantly. In SQLite, the greater the number of SQL texts, the later the fuzzing efficiency of Squirrel begins to decrease. This phenomenon is not so obvious in PostgreSQL.

The results demonstrate two things. Not only increasing the AGD of initial seeds but also increasing the number of SQL texts in initial seeds can improve the fuzzing efficiency of Squirrel.

3.3. *Analysis of crashes*

When SQLite was tested with the optimized seed, 3 crashes were triggered within 12 hours. One test case that triggered crash is shown in Table 2. SQLite was compiled with the C/C++ memory error detector AddressSanitizer in the Ubuntu 19.10 operating system. From the information given by AddressSanitizer, the three crashes correspond to the same error. A pointer error occurred in the

sqlite3VdbeAppendP4 function of the file sqlite3.c. The pointer points to the address 0x000000000000 (zero page). This error is caused by the use of SQL variables in the OVER clause in the trigger program.

This error has been fixed in the trial version of SQLite 3.35.0 updated on GitHub on January 21, 2021. This error still exists in the official version of SQLite released before that.

Table 2. A test case that triggered crash.

ID	Test case
1	CREATE TABLE v0 (v1 CHECK(v1 = zeroblob (NOT zeroblob (NOT zeroblob (DISTINCT 1.100000 + 10.100000) <= v1) = v1) NOT LIKE 'x')) ; CREATE TRIGGER x AFTER INSERT ON v0 BEGIN INSERT OR REPLACE INTO v0 (v1 , v1 , v1) VALUES (0 , NULL , 'v1') ON CONFLICT DO NOTHING ; END ; CREATE TRIGGER r1 AFTER INSERT ON v0 BEGIN SELECT v1 , count (count ()) OVER(ORDER BY@v1) AS myname FROM v0 ; END ; INSERT INTO v0 (v1 , v1) VALUES (127 , 10) ,(0 , 10) ,(10 , 3) ,(9223372036854775807 , 10) ,(12 , 8) ,(2 , 2) ,('v1' , 8) ,('x' , 127) ,('MED BOX' , 10) ,('v1' , 1) ,('v0' , 10) ,('v1' , 10) ,('LG PKG' , 1) ,('x' , 1) ,('Brand#23' , 2) ,('v1' , 10) ; ALTER TABLE t2 RENAME TO t3 ;

4. Conclusions

Through comparative experiments, the influence of the two factors, the AGD of initial seeds and the number of SQL texts of initial seeds, on the fuzzing efficiency of Squirrel was studied. The higher the AGD, the higher the efficiency. The higher the number of SQL texts, the higher the efficiency. It means that the fuzzing efficiency of Squirrel can be improved by increasing the AGD of seeds and the number of SQL texts of seeds.

Are there other factors that have a great impact on fuzzing efficiency of Squirrel? This is a question worth continuing to study. Due to the limitation of experimental equipment, MySQL and MariaDB could not be tested. In the future, when there are servers with larger RAM and unlimited network speeds that can meet the experimental conditions, initial seeds of MySQL and MariaDB will be optimized.

Acknowledgements

We would like to thank Rui Zhong who is the first author of Squirrel for his insightful suggestions about experiments.

References

[1] SQLite. Most Widely Deployed and Used Database Engine[OL]. https://www.sqlite.org/mostdeployed.html.
[2] MySQL. MySQL Customers[OL]. https://www.mysql.com/customers.
[3] Blazytko T, Bishop M, Aschermann C, et al. {GRIMOIRE}: Synthesizing structure while fuzzing[C].28th {USENIX} Security Symposium ({USENIX} Security 19). 2019: 1985-2002.
[4] Chen P, Chen H. Angora: Efficient fuzzing by principled search[C]. 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018: 711-725.
[5] Lo E, Binnig C, Kossmann D, et al. A framework for testing DBMS features[J]. The VLDB Journal, 2010, 19(2): 203-230.
[6] Zhong R, Chen Y, Hu H, et al. SQUIRREL: Testing Database Management Systems with Language Validity and Coverage Feedback[C]. Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. 2020: 955-970.
[7] Klees G, Ruef A, Cooper B, et al. Evaluating fuzz testing[C]. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018: 2123-2138.
[8] Po D K. Similarity Based Information Retrieval Using Levenshtein Distance Algorithm[J]. Int. J. Adv. Sci. Res. Eng, 2020, 6(04): 06-10.