RCABench: Open Benchmarking Platform for Root Cause Analysis

Keisuke Nishimura, Yuichi Sugiyama, Yuki Koike,
Masaya Motoda, Tomoya Kitagawa, Toshiki Takatera, and Yuma Kurogome

Ricerca Security, Inc.

***Abstract*—Fuzzing has contributed to automatically identifying bugs and vulnerabilities in the software testing field. Although it can efficiently generate crashing inputs, these inputs are usually analyzed manually. Several root cause analysis (RCA) techniques have been proposed to automatically analyze the root causes of crashes to mitigate this cost. However, outstanding challenges for realizing more elaborate RCA techniques remain unknown owing to the lack of extensive evaluation methods over existing techniques. With this problem in mind, we developed an end-to-end benchmarking platform, RCABench, that can evaluate RCA techniques for various targeted programs in a detailed and comprehensive manner. Our experiments with RCABench indicated that the evaluations in previous studies were not enough to fully support their claims. Moreover, this platform can be leveraged to evaluate emerging RCA techniques by comparing them with existing techniques.**

I. Introduction

Fuzzing has contributed to automatically identifying bugs and vulnerabilities in the software testing field; it is the process of randomly generating inputs and providing them to a program to find crashing inputs [1], [2]. Fuzzing is simple and easy to deploy compared to other testing methods, and it can also automatically and efficiently find crashing inputs, helping an enormous number of software programs improve their quality. In fact, OSS-Fuzz [3], a popular fuzzing infrastructure, is used in more than 650 open-source software (OSS) projects and has found more than 40,500 bugs and vulnerabilities [4].

However, crash analysis, which is required after fuzzing, is difficult and can be a bottleneck in making software testing scalable. We must manually analyze these inputs later because fuzzers only generate crashing inputs. The cost of analyzing the crashing inputs generated by fuzzers is very high for two reasons. First, fuzzers sometimes generate numerous crashing inputs, the root causes of which are the same. For example, a fuzzer generated more than 254,000 crashing inputs for 39 unique bugs in an experiment conducted in a previous study [5]. Second, fuzzers randomly generate crashing inputs, which means that the inputs can contain a significant amount of noise. In other words, these inputs can include byte sequences that are not essentially related to the crash causes and hence can be removed. During crash analysis, analysts need to determine which parts of the inputs are related to the causes, which costs significantly as the noise increases.

To reduce such costs in crash analysis, various automated techniques have been proposed in the field of *triage* [2], [5], [6], [7], [8]. Triage is the process of analyzing and reporting inputs that cause crashes [2]. In triage research, *root cause analysis (RCA)* has attracted particular attention in recent years. RCA, also known as localization or fault localization, is the process of identifying lines, basic blocks, or conditions related to the root cause of a crash caused by a crashing input; it provides developers with hints about the root cause. While there are several different RCA approaches [9], [10], [11], this study focuses on statistical fault localization, which infers the program states correlated with the crash causes by contrasting the execution of crashing and non-crashing inputs.

However, RCA is relatively underdeveloped among triage topics. As discussed in detail later, state-of-the-art RCA techniques are not infallible. For example, in DeFault [8], the average false positive rate was 9.2 %. Furthermore, RCA is not widely applied in industry, whereas deduplication, another triage technique, is integrated into major fuzzers, such as AFL [12] and honggfuzz [13]. Thus, the existing RCA techniques are neither accurate nor practical enough to be widely used in the real world; this is because RCA techniques have not been thoroughly improved owing to undiscovered outstanding challenges, that is, the problems to be solved to realize more elaborate RCA techniques. One of the causes that make such challenges still unknown would be the lack of extensive evaluation methods over existing techniques. We found that the following three points were unnoticed and should be considered to realize extensive evaluation methods:

Non-uniqueness of root cause definition There can be several possible patches to fix a complex bug. If we define (the location in the source code of) the root cause as the location that should be fixed, there can be multiple candidates for them. It is not obvious for evaluators to define where root causes lie, while it is certainly necessary for evaluating RCA techniques. Despite this vagueness, existing techniques [6], [7], [8] did not fully disclose the ground truth of their evaluations. This makes it difficult to reproduce the experiments in existing studies.

Decoupling RCA steps We found that the existing techniques consist of two separable steps: *data augmentation* and *feature extraction*. However, these techniques have not been evaluated separately to determine the performance of each step.

Workshop on Binary Analysis Research (BAR) 2023
3 March 2023, San Diego, CA, USA
ISBN 1-891562-84-3
https://dx.doi.org/10.14722/bar.2023.23004
www.ndss-symposium.org

Variance-aware evaluation for data augmentation The existing techniques augment data using various fuzzing methods, particularly those that are altered for data augmentation. The evaluations need to consider the random nature of fuzzing.

Considering these three points, we developed RCABench, an end-to-end benchmarking platform, to reveal the challenges of RCA. We provide a detailed and comprehensive evaluation of existing techniques for various targets and find some cases where exsting techniques cannot correctly analyze. Moreover, this platform can be used to evaluate the new RCA techniques proposed in the future by comparing them with existing techniques.

Overall, the main contributions of this work are as follows:

- We present three problems in the evaluation methods of existing RCA studies.
- We developed RCABench, an open-source benchmarking platform[1]; it provides a more standardized evaluation and helps to summarize the outstanding challenges in RCA.
- Through experiments with RCABench, we identified several insights into the pitfalls of existing techniques and provided examples to motivate further research.

II. Root Cause Analysis

Root cause analysis (RCA), also known as localization or fault localization, is a process of automatically identifying lines, basic blocks, or conditions related to the root cause of a crash; it aids in debugging and reduces the cost of the crash analysis. We analyzed state-of-the-art RCA techniques [6], [7], [8] and identified two separable processes that are common to all. In this study, we refer to these as *data augmentation* and *feature extraction*[2]. In this section, we describe these in detail.

A. Data Augmentation

Data augmentation is the process of generating new crashing and non-crashing inputs from a given crashing input; this is the first process in RCA, and the generated inputs are used as datasets for the feature extraction. Therefore, the quality of the dataset affects the RCA results.

The existing techniques augment inputs using various fuzzing methods that are specially altered for data augmentation. For example, Aurora [6] uses the *crash exploration mode* provided by AFL [12], a typical coverage-guided fuzzer. In this study, we refer to this as *AFLcem*. VulnLoc [7] proposed *ConcFuzz*, a directed fuzzer for efficiently generating inputs that exercise execution paths in the neighborhood of the path taken by a given crashing input, aiming at augmentation of higher quality. These methods use a single crashing input as the *initial seed* and automatically generate crashing/non-crashing inputs by randomly mutating it. In fuzzing, an initial seed is an input provided at the beginning of a fuzzing campaign.

B. Feature Extraction

Feature extraction analyzes the root cause using a dataset generated by data augmentation. This process consists of two steps. First, an analyzer records the state of the targeted program at runtime while executing it with each input in the dataset. For example, in Aurora, executed instructions and variable values are recorded. Next, the analyzer compares traces between the crashing and non-crashing inputs and statistically infers their differences. This difference is indicated as the root cause. The existing techniques estimate the lines or basic blocks related to the root cause. Aurora also estimates predicates, that is, simple Boolean expressions that represent the conditions to be met before a crash occurs. When actually used, analyzers do not report only the most likely root cause candidate but instead multiple candidates in descending order of the level of confidence that analyzers assign to them. In this study, we denote VulnLoc and Aurora analyzers *VulnLocFE* and *AuroraFE*, respectively.

To better illustrate this step, we take as an example CVE-2016-10094 in LibTIFF, an open-source library. As shown in Listing 1, this vulnerability causes a heap buffer overflow owing to an off-by-one error. Specifically, the program crashes when the variable *count* is four and the statements inside the patched *if* statement are executed. Generally, we refer to this *if* statement as the root cause location and "*count == 4*" as the root cause predicate. If this *if* statement appears frequently in the program traces for crashing inputs and infrequently for non-crashing inputs, the statement can be identified as the root cause location. Similarly, if there is a distinguishable difference in the value of the variable *count* between two sets of traces, the predicate can be identified.

Listing 1: Developer patch for CVE-2016-10094 in LibTIFF.

```
   if(TIFFGetField(input, TIFFTAG_JPEGTABLES,
                   &count, &jpt) != 0) {
-      if (count >= 4) {
+      if (count > 4) {
         int retTIFFReadRawTile;
         _TIFFmemcpy(buffer, jpt, count - 2);
```

III. Challenges in RCA Evaluation

In this section, we describe three previously unconsidered points which are imperative to extensive RCA evaluations.

A. Non-uniqueness of Root Cause Definition

Sometimes, there are multiple ways to fix a bug; suppose that function B triggers a bug when it processes the data produced by function A because the produced data conform to rule X, whereas B expects rule Y. In this case, we can make A comply with Y or B comply with X.

In such cases, if we define the root cause locations as the locations in the source code that should be fixed, there can be multiple candidates for root cause locations. Therefore, it is difficult to correctly include all of them as the ground truth in RCA evaluations. The evaluators currently define the ground truth manually by coming up with all the possible patches, and hence, the evaluators sometimes miss some of the root cause locations and use different ground truths.

[1] https://github.com/RICSecLab/RCABench

[2] Although these processes were not explicitly defined in the RCA study, inspired by similar efforts in the machine learning field, we refer to them as data augmentation and feature extraction.

To illustrate more simply that there are multiple root cause locations for a bug, we take CVE-2017-15232 in Libjpeg as an example; this vulnerability causes a null pointer dereference owing to the lack of a code to check for a null pointer. There are several possible fixes for this vulnerability, as shown in Listings 2 and 3. The first method, as shown in Listing 2, is to insert a code to check for a null pointer before the *for* statement. Another way, as shown in Listing 3, is to do the same at the beginning within the *for* statement. Thus, the root cause location is not uniquely determined, and identifying all the candidates is difficult; this can occur frequently with bugs whose root cause is the absence of code.

The existence of multiple root cause locations makes it difficult to determine the ground truth and prevents the evaluation results from being identical. In addition, the existing studies did not fully disclose the ground truth, making evaluators have difficulty reproducing the existing experiments accurately.

Listing 2: Developer patch for CVE-2017-15232 in Libjpeg.

```
+  if (output_buf == NULL && num_rows)
+    ERREXIT(cinfo, JERR_BAD_PARAM);
   for (row = 0; row < num_rows; row++) {
     jzero_far((void *) output_buf[row],
       (size_t) (width * sizeof(JSAMPLE)));
```

Listing 3: Another developer patch for CVE-2017-15232 in Libjpeg.

```
   for (row = 0; row < num_rows; row++) {
+    if (output_buf == NULL)
+      ERREXIT(cinfo, JERR_BAD_PARAM);
     jzero_far((void *) output_buf[row],
       (size_t) (width * sizeof(JSAMPLE)));
```

B. Decoupling Data Augmentation and Feature Extraction

As described in Section II, we found that state-of-the-art RCA methods [6], [7], [8] consist of two separable steps: data augmentation and feature extraction. However, the evaluations in these previous studies did not decouple the data augmentation and feature extraction. Evaluators should investigate the performance of each process independently because these are two separable steps. For example, VulnLoc [7] proposed ConcFuzz as a data augmentation method but did not evaluate its relative performance by replacing it with AFLcem, an existing alternative algorithm. In other words, it has not been fully confirmed that ConcFuzz generates datasets of higher quality than AFLcem. Thus, the pure performance achieved by each step of proposed methods was not measured.

C. Variance-aware Evaluation of Data Augmentation

In existing studies [6], [7], [8], evaluations did not consider the variable characteristics of data augmentation. As described in Section II, data augmentation generates a dataset for feature extraction using fuzzing. Therefore, the quality of the dataset may depend on the configuration of fuzzers, such as the initial seeds and duration of a fuzzing campaign. The existing studies have not dealt with this concern and have not been able to evaluate the impact of data augmentation on RCA results in a variance-aware manner. Specifically, the following three variables should be considered:

Data augmentation time The existing studies did not evaluate RCA techniques with various values of the time spent in data augmentation. For example, in Aurora, the data augmentation time was fixed to only one value, either 2 or 12 h, depending on targeted programs. However, the data augmentation time can affect RCA results in multiple ways. We can generate a dataset with a larger amount of crashing/non-crashing inputs by spending more time in fuzzing; this may increase the dataset diversity and improve the accuracy of feature extraction. However, it is also plausible that overfitting occurs, similarly to data augmentation in machine learning, making the accuracy worse.

Initial seed The existing studies prepared only one specific crashing input as an initial seed for fuzzing in data augmentation. For example, in VulnLoc, the initial seed is the input used as a proof-of-concept when reporting vulnerabilities. In fuzzing, the difference in the initial seeds is known to affect performance, such as coverage and bug finding [14], [15]. Data augmentation using fuzzing may also affect the accuracy of feature extraction; therefore, the evaluator should prepare several initial seeds. In addition, the existing studies have not focused on the characteristics of initial seeds. For example, the initial seed generated by a fuzzer tends to be noisier and more complex than that generated by an analyst manually.

Fuzzing randomness The existing studies have not considered the randomness of fuzzing in data augmentation. They evaluated each method using only a dataset from a single fuzzing run. However, fuzzing is a highly stochastic process. Hence, the generated dataset changes with each run, which can affect the results of RCA. RCA techniques must be evaluated multiple times to address this problem. In fuzzing studies, it has already been standard practice to run fuzzers multiple times and evaluate the results statistically if possible [16]. The same approach is required in RCA studies.

IV. PROPOSAL: RCABENCH

We propose RCABench, an end-to-end benchmarking platform that can run RCA techniques on selected bugs and check whether their results match the predefined locations of root causes[3]. The design of the RCABench was motivated by the insights described in Section III. For each RCA technique, the data augmentation and feature extraction steps were decoupled, which enabled the comparison and evaluation of the augmentation and extraction methods separately.

Currently, RCABench supports two augmentation methods and two extraction methods. The available augmentation methods are *AFLcem*, used in Aurora [6], and *ConcFuzz*[4], proposed in VulnLoc [7]. The available extraction methods are the AuroraFE and VulnLocFE. We decoupled the augmentation and extraction steps and abstracted their interfaces so that each augmentation method could be connected to each extraction method interchangeably since the original implementations of AuroraFE and VulnLocFE are incompatible with ConcFuzz and AFLcem, respectively. Consequently, RCABench

[3] Some RCA techniques (e.g., VulnLoc and Aurora) indicate the candidates for root cause locations as pairs of addresses of an assembly instruction and their corresponding source line numbers. RCABench uses line numbers for the check because the addresses of instructions are too fine-grained to decide whether the address is a root cause location.

[4] For ConcFuzz, the time spent in saving its internal data at the end is not included in the augmentation time.

can evaluate previously untested combinations of AFLcem × VulnLocFE and ConcFuzz × AuroraFE. Note that we used only the root cause locations inferred by AuroraFE to compare the performance of the techniques. Supporting and evaluating the root cause predicates included in the outputs of AuroraFE are left for future work.

RCABench provides multiple popular real-world programs containing actual bugs and vulnerabilities as targets of RCA. Currently, seven targets have been prepared, all of which were used in the evaluations of Aurora and VulnLoc [6], [7]. We show the lists and summary of their root causes in Table II. Our criteria for selecting targets are the availability of the source code and the diversity of root and crash causes. As discussed in the previous section, the selection of root cause locations can vary and be biased. Therefore, we first registered to RCABench several reasonable candidates for root cause location as ground truth for each target. For stable re-evaluation, RCABench publicly exposes these root cause locations, along with their brief explanations. RCABench also includes one or more initial seeds for each target to support augmentation methods that require a crashing input as an initial seed. For targets with multiple seeds available, we selected a crashing input used in bug disclosure or explanation to the developers as the baseline.

V. Benchmark Results

This section describes the results of the proposed benchmark RCABench. Through benchmarking, we answered the following questions:

- **RQ1:** Which RCA techniques can perform accurate analysis on each bug?
- **RQ2:** Does the increase in data augmentation time improve accuracy?
- **RQ3:** Do initial seeds affect accuracy?
- **RQ4:** Does the randomness of data augmentation affect accuracy?

All results shown here were obtained on a 256-CPU (AMD EPYC 7742) machine with 2TB memory and Ubuntu 20.04 operating system. To investigate the relationship between the data augmentation time and the accuracy of RCA techniques, we ran the data augmentation process up to an imposed time limit and, in each of the first 5, 15, 30, and 45 minutes and every hour thereafter during the execution, RCABench saved the dataset produced by the data augmentation at that time and analyzed root cause with the dataset. For Targets #1-4,6, we set the time limit to 4h, and for Targets #5,7, we extended it to 12h in accordance with the evaluation of Aurora [6].

A. Which RCA techniques can perform accurate analysis on each bug? (RQ1)

Table I presents the overall comparisons of RCA techniques for each target. The numbers in the table indicate the rank of the correct answer (actual root cause location) among the location candidates reported by each RCA technique, ordered by the level of confidence assigned by the technique. A lower rank indicates that the RCA technique can infer the root cause location more accurately ("1" is the best score). In this experiment, we set VulnLocFE to report up to Top-200 candidates, in accordance with the original paper. "–" indicates that the correct answer was not included in the candidates produced by the RCA technique. "N/A" indicates that no data could be obtained because the technique tried to produce quite huge files or took a long time for file I/O, which were impossible to handle with our limited machine resources.

TABLE I: Results of four RCA techniques in different data augmentation times.

	Program	D.A. Time	A × A	C × A	A × V	C × V
#1	LibTIFF	15 m	15	9	2	13
		2 h	9	33	2	12
		4 h	9	47	2	12
#2	Libjpeg	15 m	–	–	23	32
		2 h	–	15	12	23
		4 h	–	14	12	17
#3	Libjpeg	15 m	16	–	2	1
		2 h	7	–	2	1
		4 h	6	–	2	1
#4	Libxml2	15 m	28	–	19	16
		2 h	28	–	23	16
		4 h	27	–	19	17
#5	mruby	15 m	41	105	59	11
		4 h	30	60	–	45
		12 h	31	60	–	45
#6	readelf	15 m	1	4	4	4
		2 h	1	1	4	4
		4 h	1	1	4	4
#7	Lua	15 m	–	–	1	1
		4 h	–	N/A	–	N/A
		12 h	32	N/A	–	N/A

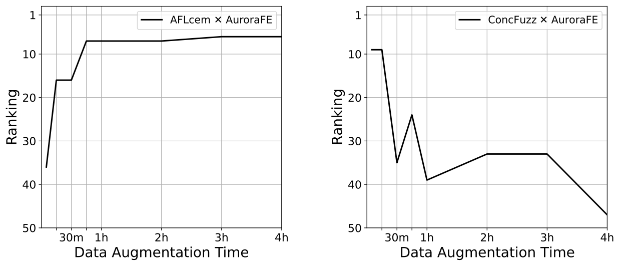
"C × A" means ConcFuzz × AuroraFE and "A × V" means AFLcem × VulnLocFE.
"N/A": No data were obtained. "–": The root cause location did not appear in the candidates reported by an RCA technique.

Table I indicates that no technique can predict the root cause locations with high accuracy for *all* targets: while ConcFuzz × VulnLocFE successfully inferred the correct location of the root cause in Target #3 with the highest accuracy, its predictions for Targets #6 were less accurate than AFLcem × AuroraFE. Our newly tested combination, AFLcem × VulnLocFE, outperformed the existing methods for Targets #1 and #2. However, they failed to find the root causes of some other targets. This result implies that the characteristics of the targets, which can be analyzed with high accuracy, would be different for each RCA technique. This up-and-down situation depending on the targeted programs is similar to fuzzer benchmarking, as seen in some results [17], [18].

Answer: The technique that gave the highest rank to the correct root cause was different for each bug, and there was no universal technique that was most accurate.

B. Does the increase in data augmentation time improve accuracy? (RQ2)

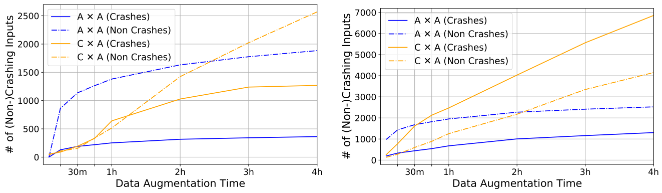
Next, we compared the results of each technique for each target with different data augmentation times. Table I lists the results in three data augmentation times (15m, 2h, 4h for Targets #1-4,6 and 15m, 4h, 12h for Targets #5,7) for each technique and target pair. We also show two detailed examples of how accuracy changes with time in Figure 1. In 19 cases, out of the 26 results excluding "N/A", the accuracy remained the same (e.g., Target #6 except ConcFuzz × AuroraFE)



(a) Target #3

(b) Target #1

Fig. 1: Accuracy vs. data augmentation time.



(a) Target #1

(b) Target #6

Fig. 2: The number of generated inputs over time. A × A, and C × A denote AFLcem × AuroraFE, and ConcFuzz × AuroraFE, respectively.

or improved as the data augmentation time increased (e.g., AFLcem × AuroraFE on Target #3 in Figure 1a).

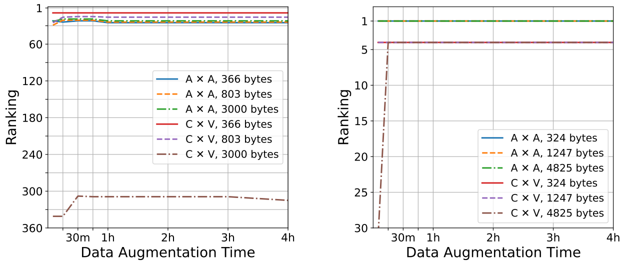
However, the increase in data augmentation time worsened the accuracy in some pairs of RCA techniques and targets, such as ConcFuzz × VulnLocFE on Target #5. Figure 1b shows an example of the deteriorating trend of ConcFuzz × AuroraFE on Target #1. In this example, the highest accuracy was achieved up to 15m, and its accuracy declined thereafter.

Answer: While the accuracy improved or did not change over time in many cases, there were a few cases in which the accuracy was degraded.

Thus, sometimes data augmentation time eventually affects the accuracy. This fact indicates that somehow data augmentation time affects the quality of datasets produced by data augmentation. To analyze how it affects the quality, we inspected how the number and ratio of samples (i.e., crashing/non-crashing inputs) in a dataset changes as data augmentation time increases. Figure 2 plots the number of samples versus data augmentation time for Target #1 and #6.

When looking at the ratio of samples produced by ConcFuzz × AuroraFE in Figure 2a, we see that the number of non-crashing inputs starts exceeding that of crashing inputs considerably in one hour. This would force feature extraction methods to find out root cause locations with imbalanced datasets of crashing/non-crashing inputs, which is very similar to a situation called *imbalanced data classification* in the machine learning field [19]. Generally, imbalanced data can cause poor accuracy in these classifying tasks. Actually, in Table I, the accuracy of ConcFuzz × AuroraFE perceptibly decreases as the ratio gets imbalanced.

Another implicative fact is that the numbers of samples for Target #6 shown in Figure 2b are large from the beginning, compared to those of Target #1. Perhaps, this would have made all the RCA techniques achieve the high accuracy in Target #6 even in 15m, considering that a large number of samples usually leads to high accuracy in classifying tasks.



(a) Target #4

(b) Target #6

Fig. 3: The transitions of the accuracy over time for different initial seeds. A × A, and C × V denote AFLcem × AuroraFE and ConcFuzz × VulnLocFE, respectively. In Target #4 and #6, the 803-byte and 324-byte seeds are their original seed, respectively.

C. *Do initial seeds affect accuracy? (RQ3)*

To answer RQ3, we investigated whether the accuracy changed depending on the initial seed. For this purpose, we first ran AFLcem against Target #4 and #6 to produce various crashing inputs. Then, for each target, we randomly selected two different-length inputs from the produced inputs as initial seeds and evaluated the RCA techniques with them. We selected these two targets because we could find the crashing inputs that were much smaller or larger than the original crashing input. Thus, we believe that the newly produced initial seeds were very different from the original ones, in terms of seed size and the method of producing them. Note that most of the original seeds were created manually, which may make a significant difference between the original and new seeds with regard to whether noise exists in them, as described in Section I.

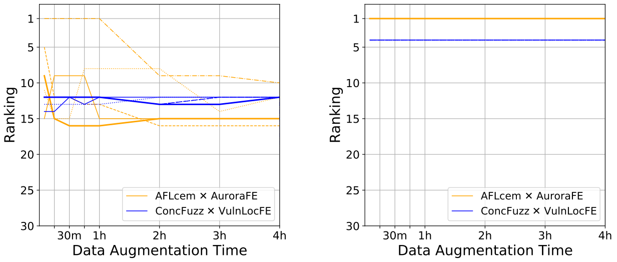
Figure 3 shows accuracy versus data augmentation time for different initial seeds. While, in Target #6, the accuracies are little affected by the difference of initial seeds in both AFLcem × AuroraFE and ConcFuzz × VulnLocFE, the two added initial seeds of Target #4 affected the accuracy of ConcFuzz × VulnLocFE, and one of them had a significant impact in particular. This result is consistent with the fact that the performance of a fuzzer can be affected by the initial seeds [14], [15].

Answer: The difference in initial seeds sometimes affects accuracy. This implies that evaluators should make the initial seeds public to avoid cherry-picking and for reproducibility.

D. *Does the randomness of data augmentation affect accuracy? (RQ4)*

We observed the randomness effect on the accuracy by evaluating the techniques five times. For ConcFuzz, different seeds of its random number generator were set in each trial.

Consequently, we observed some non-negligible variances as predicted in Section III, while the accuracy was very stable in some targets. Figure 4 shows the results of AFLcem × AuroraFE and ConcFuzz × VulnLocFE in Target #1 and #6. While both RCA techniques in Target #6 and ConcFuzz × VulnLocFE in Target #1 had little divergence in their accuracy,



(a) Target #1

(b) Target #6

Fig. 4: The effect of the randomness in data augmentation on the accuracy. Five attempts were performed.

AFLcem × AuroraFE showed significant divergence in the accuracy in Target #1. Specifically, one of the five trials outperformed the others. If one cherry-picked only this trial, it could be concluded that AFLcem × AuroraFE was more accurate than ConcFuzz × VulnLocFE, although the two techniques achieved similar accuracies on average; this suggests that it is important to evaluate RCA techniques multiple times and, if possible, perform statistical analysis of the results.

Answer: For some combinations of techniques and targets, randomness in data augmentation leads to non-negligible variances in accuracy. This result suggests that experiments to evaluate RCA techniques should be conducted multiple times to reduce the effect of randomness as much as possible.

VI. Discussion and Future Work

A. *Threats to Validity*

Although the evaluations on RCABench and our research questions provided thought-provoking claims in Section V, we admit that two major threats may spoil some of the claims.

The first one is the non-uniqueness of root cause definition. As previously noted, we have been aware that multiple root cause locations can be the ground truth, and mitigated risk by making our definition public and upgradable. However, this is just a temporary countermeasure in the sense that particular techniques can be underestimated still; it is possible that some techniques report different valid root cause locations than our definition while the others report ours. A more robust and better definition and evaluation method of accuracy is desired.

The second one is fuzzing randomness. While RQ4 revealed that it undoubtedly threatens the evaluation validity in previous studies, it also threatens the validity of our evaluation, especially for RQ1 and RQ3 (note that RQ2 should be cared about even within one trial, and hence its claim is valid in that sense regardless of the threats). We acknowledge that our results do not have a statistical significance due to our limited computational resources and should be carefully reviewed by others. Nevertheless, it is sure that the effectiveness and superiority of the existing techniques are at least not so obvious as the evaluations in the existing studies claimed. Moreover, the lack of a statistical significance can be eventually resolved in the future since we release RCABench as an OSS platform and other researchers are also able to take benchmarks.

While the above two points threaten the internal validity, the external validity is another concern because seven programs are not enough to fully understand the behavior and performance of RCA techniques against a wide variety of targeted programs. For preparing more target programs with diverse root causes, the programs constituting benchmarks known in the fuzzing field [20], [21], [17], [22] would be reasonable candidates. In particular, Magma [21] and FuzzBench [22] provide a suite of programs that contain bugs found in the real world and are widely used. Evaluating techniques with these programs would further clarify their practical effectiveness.

B. *Possible Improvement of Existing Techniques*

In our experiments, we found some cases where RCA techniques failed to analyze the root causes with high accuracy, owing to their nature. A striking example is that ConcFuzz × AuroraFE ranked the root cause locations of Target #1 (CVE-2016-10094) very low in RQ1 at data augmentation times of 2h and 4h; this is probably because the ideal difference that should exist between its generated crashing/non-crashing inputs is whether or not a certain variable is at a certain value, while ConcFuzz focuses on control flow and the generated inputs did not have enough diversity of values. This suggests the possibility of further improvement of data augmentation methods by considering features other than the control flow.

Another concern about the existing techniques is that implementation designs differ among them. For example, to trace a program, Aurora [6] uses Intel PIN [23] and VulnLoc uses DynamoRIO [24]. In addition, the data augmentation method of VulnLoc is written in the programming language Python, whereas that of Aurora in C/C++. Thus, there is a possibility that they have different performances owing to their implementation methods. If VulnLoc were written in C/C++, VulnLoc would be able to run faster. In the fuzzing field, some frameworks have already been proposed so that different algorithms can be implemented in a uniform way to solve such a problem [25], [26]. For example, LibAFL [25] is a framework for building fuzzers in a modular manner. LibAFL can reduce the cost of combining multiple fuzzing algorithms into a single fuzzer and can fairly and objectively evaluate the algorithms within the common implementation. If a similar modular framework is presented for RCA, researchers would be able to take fairer evaluations. Also, it would allow them to implement and evaluate a new algorithm more easily.

VII. Conclusion

Although fuzzing is a mature method for automatically finding bugs, *root cause analysis (RCA)* techniques for discovered bugs are not full-grown. One of its causes is that the environment for a comprehensive evaluation of existing RCA techniques was inadequate, making it difficult to discover the outstanding problems. Therefore, we developed a benchmark platform, RCABench, for automatic and extensive evaluation. Our experiments indicated that the evaluations in previous studies were not enough to fully support their claims and found some cases where the representative techniques failed to analyze with high accuracy.

We believe that this initiative fosters future RCA research by assisting researchers to propose and evaluate emerging RCA techniques, as this study gives a glimpse of it. To shed light on and help resolve the hidden challenges of RCA, we would like to continue adding various targets and techniques and making RCABench a more insightful platform.

Acknowledgment

We would like to thank the anonymous reviewers for their helpful feedback. We also gratefully acknowledge the authors of Aurora and VulnLoc, who made their implementations and experiment configurations publically available. This work was supported by the Acquisition, Technology & Logistics Agency (ATLA) under the Innovative Science and Technology Initiative for Security 2020 (JPJ004596).

References

[1] H. Liang, X. Pei, X. Jia, W. Shen, and J. Zhang, "Fuzzing: State of the art," *IEEE Trans. Reliab.*, vol. 67, no. 3, pp. 1199–1218, 2018. [Online]. Available: https://doi.org/10.1109/TR.2018.2834476

[2] V. J. M. Manès, H. Han, C. Han, S. K. Cha, M. Egele, E. J. Schwartz, and M. Woo, "The art, science, and engineering of fuzzing: A survey," *IEEE Trans. Software Eng.*, vol. 47, no. 11, pp. 2312–2331, 2021. [Online]. Available: https://doi.org/10.1109/TSE.2019.2946563

[3] K. Serebryany, "OSS-Fuzz - Google's continuous fuzzing service for open source software," Talk at the 26th USENIX Security Symposium, 2017.

[4] Google, "Oss-fuzz," 2022. [Online]. Available: https://github.com/google/oss-fuzz#trophies

[5] Z. Jiang, X. Jiang, A. Hazimeh, C. Tang, C. Zhang, and M. Payer, "Igor: Crash deduplication through root-cause clustering," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021.* ACM, 2021, pp. 3318–3336. [Online]. Available: https://doi.org/10.1145/3460120.3485364

[6] T. Blazytko, M. Schlögel, C. Aschermann, A. Abbasi, J. Frank, S. Wörner, and T. Holz, "AURORA: statistical crash analysis for automated root cause explanation," in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020.* USENIX Association, 2020, pp. 235–252. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/blazytko

[7] S. Shen, A. Kolluri, Z. Dong, P. Saxena, and A. Roychoudhury, "Localizing vulnerabilities statistically from one exploit," in *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security, Virtual Event, Hong Kong, June 7-11, 2021.* ACM, 2021, pp. 537–549. [Online]. Available: https://doi.org/10.1145/3433210.3437528

[8] X. Zhang, J. Chen, C. Feng, R. Li, W. Diao, K. Zhang, J. Lei, and C. Tang, "Default: Mutual information-based crash triage for massive crashes," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022.* ACM, 2022, pp. 635–646. [Online]. Available: https://doi.org/10.1145/3510003.3512760

[9] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Trans. Software Eng.*, vol. 42, no. 8, pp. 707–740, 2016. [Online]. Available: https://doi.org/10.1109/TSE.2016.2521368

[10] H. A. de Souza, M. L. Chaim, and F. Kon, "Spectrum-based software fault localization: A survey of techniques, advances, and challenges," *CoRR*, vol. abs/1607.04347, 2016. [Online]. Available: http://arxiv.org/abs/1607.04347

[11] P. Agarwal and A. P. Agrawal, "Fault-localization techniques for software systems: a literature review," *ACM SIGSOFT Softw. Eng. Notes*, vol. 39, no. 5, pp. 5:1–5:8, 2014. [Online]. Available: https://doi.org/10.1145/2659118.2659125

[12] "American fuzzy lop (afl)," https://lcamtuf.coredump.cx/afl/.

[13] "honggfuzz," https://honggfuzz.dev/.

[14] A. Herrera, H. Gunadi, S. Magrath, M. Norrish, M. Payer, and A. L. Hosking, "Seed selection for successful fuzzing," in *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021.* ACM, 2021, pp. 230–243. [Online]. Available: https://doi.org/10.1145/3460319.3464795

[15] D. Wolff, M. Böhme, and A. Roychoudhury, "Explainable fuzzer evaluation," *CoRR*, vol. abs/2212.09519, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2212.09519

[16] M. Böhme, L. Szekeres, and J. Metzman, "On the reliability of coverage-based fuzzer benchmarking," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022.* ACM, 2022, pp. 1621–1633. [Online]. Available: https://doi.org/10.1145/3510003.3510230

[17] Y. Li, S. Ji, Y. Chen, S. Liang, W. Lee, Y. Chen, C. Lyu, C. Wu, R. Beyah, P. Cheng, K. Lu, and T. Wang, "UNIFUZZ: A holistic and pragmatic metrics-driven platform for evaluating fuzzers," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021.* USENIX Association, 2021, pp. 2777–2794. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/li-yuwei

[18] Google, "Fuzzbench reports," 2020. [Online]. Available: https://www.fuzzbench.com/reports/

[19] H. Kaur, H. S. Pannu, and A. K. Malhi, "A systematic review on imbalanced data challenges in machine learning: Applications and solutions," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 79:1–79:36, 2019. [Online]. Available: https://doi.org/10.1145/3343440

[20] B. Dolan-Gavitt, P. Hulin, E. Kirda, T. Leek, A. Mambretti, W. K. Robertson, F. Ulrich, and R. Whelan, "LAVA: large-scale automated vulnerability addition," in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016.* IEEE Computer Society, 2016, pp. 110–121. [Online]. Available: https://doi.org/10.1109/SP.2016.15

[21] A. Hazimeh, A. Herrera, and M. Payer, "Magma: A ground-truth fuzzing benchmark," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 3, pp. 49:1–49:29, 2020. [Online]. Available: https://doi.org/10.1145/3428334

[22] J. Metzman, L. Szekeres, L. Simon, R. Sprabery, and A. Arya, "FuzzBench: an open fuzzer benchmarking platform and service," in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021.* ACM, 2021, pp. 1393–1403. [Online]. Available: https://doi.org/10.1145/3468264.3473932

[23] C. Luk, R. S. Cohn, R. Muth, H. Patil, A. Klauser, P. G. Lowney, S. Wallace, V. J. Reddi, and K. M. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005.* ACM, 2005, pp. 190–200. [Online]. Available: https://doi.org/10.1145/1065010.1065034

[24] D. Bruening, Q. Zhao, and S. P. Amarasinghe, "Transparent dynamic instrumentation," in *Proceedings of the 8th International Conference on Virtual Execution Environments, VEE 2012, London, UK, March 3-4, 2012 (co-located with ASPLOS 2012).* ACM, 2012, pp. 133–144. [Online]. Available: https://doi.org/10.1145/2151024.2151043

[25] A. Fioraldi, D. C. Maier, D. Zhang, and D. Balzarotti, "LibAFL: A framework to build modular and reusable fuzzers," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022.* ACM, 2022, pp. 1051–1065. [Online]. Available: https://doi.org/10.1145/3548606.3560602

[26] "Fuzzing unification framework," https://github.com/fuzzuf/fuzzuf.

Appendix

TABLE II: Details of targeted vulnerabilities.

	Program	CVE ID	Root Cause	Crash Cause
#1	LibTIFF	CVE-2016-10094	off-by-one error	heap buffer overflow
#2	Libjpeg	CVE-2018-19664	incomplete check	heap buffer overflow
#3	Libjpeg	CVE-2017-15232	missing check	null pointer dereference
#4	Libxml2	CVE-2017-5969	incomplete check	null pointer dereference
#5	mruby	None	missing check	type confusion
#6	readelf	CVE-2019-9077	missing check	heap buffer overflow
#7	Lua	CVE-2019-6706	missing check	use-after-free

Target #5 was not assigned a CVE ID but was assigned ID 185041 in the HackerOne platform.