# Toward Facilitating Root Cause Localization in Fuzzing

Katsunori Aoki
*The University of Tokyo*
*aoki@os.ecc.u-tokyo.ac.jp*

Takahiro Shinagawa
*The University of Tokyo*
*shina@ecc.u-tokyo.ac.jp*

## 1 Abstract

Fuzzing is a popular approach to efficient vulnerability discovery. In particular, mutation-based fuzzing progresses more efficiently than fuzzing with random input by gradually mutating the input seed based on feedback and thereby continuously providing approximately valid input to the target program. By automatically giving a large number of inputs, a fuzzer can generate a large amount of information about the target program execution in crash cases as well as non-crash cases.

Unfortunately, the information reported by the fuzzer only shows the location of the crash, not necessarily the location of the root cause. In fact, the crash location is often far from the root cause in the source code, making root cause analysis difficult and causing the inflation problem where a single root cause can cause multiple crashes in different locations. Root cause localization still relies on daunting manual analysis, and its facilitation is a major research challenge.

Several studies attempted to automate root cause analysis using symbolic execution [3]. However, they require careful consideration to avoid path explosion during the symbolic (re-)execution of the target program. AURORA [1] generates root cause predicates, even for complex bugs such as type confusion, by generating many similar inputs derived from the crash input and comparing the internal state of binary programs. However, AURORA generates numerous false positives and provides only binary-level debugging information.

Our goal is to provide human-friendly information that enables quick approach to and easy understanding of the root cause, using only the information obtained during fuzzing. The key idea is to generate a difference in the data flow graphs between the crashed execution and the normal execution immediately preceding the crashed one. In mutation-based fuzzing, the input that caused a crash is generated by slightly mutating the input to the previous normal execution. Therefore, the difference in the data flow graphs localizes the data source that caused the crash, i.e., the point that is likely to be closer to the root cause. Since crash and normal execution information is already available, no re-execution is required. In addition, to make understanding root cause easier, lexical information at the source code level, such as symbol and type names, is added to the data flow graph to facilitate reading of semantic information such as type confusion.

For the implementation, we used LLVM for source code instrumentation, and created a system that can record data flows in fuzzing with lexical level information and generate a differential data flow graph with the lexical information from the crash and the previous executions. For evaluation, we used Magma [2], a vulnerability dataset for evaluating fuzzing performance that reproduces known vulnerabilities in software libraries, with normal and crash inputs as artifacts.

In our preliminary experiments, we applied the proposed method to four known vulnerabilities in Magma and measured the distance on the data flow graph between the boundary the difference, which corresponds to the point that changed the data flow, and the point where the patch was actually applied (patch point). As a result, we found that the distance was 0 in three cases; in two cases, the patch point was the control flow governing the boundary of the difference, and in the other case, the patch point was tangent to the boundary. In another case, a sanitization patch was applied to a point at distance 1 from the boundary. In all cases, we did not need to follow the data flow far from the boundary, and we could find the patch point by observing near the boundary or by focusing on interesting functions such as malloc. Thus, we demonstrated that the proposed method effectively localizes the root cause.

In the future, we evaluate our approach in more crash cases.

## References

[1] T. Blazytko et al. AURORA: Statistical Crash Analysis for Automated Root Cause Explanation. In *Proc. 29th USENIX Security Symposium*, Aug. 2020.

[2] A. Hazimeh et al. Magma: A Ground-Truth Fuzzing Benchmark. *Proc. ACM Meas. Anal. Comput. Syst.*, 4(3), Dec. 2020.

[3] C. Yagemann et al. ARCUS: Symbolic Root Cause Analysis of Exploits in Production Systems. In *Proc. 30th USENIX Security Symposium*, Aug. 2021.