MISOFUZZ: A MODULAR INFRASTRUCTURE FOR SCALABLE FUZZING ORCHESTRATION

Nicole Gervasoni, Mikhail I. Lubinets, Ridhi Jain, and Lucas C. Cordeiro
Technology Innovation Institute, Abu Dhabi, UAE
{nicole.gervasoni, mikhail.lubinets, ridhi.jain, lucas.cordeiro}@tii.ae

ABSTRACT

Traditional fuzzing approaches can be time-consuming and resource-intensive, particularly for large and complex software systems. Thus, the performance optimization of a fuzzing tool is often a key concern in its usage. In this workshop, we present MISOFUZZ, a modular cooperative cloud-native fuzzing platform that aims to simplify the integration of fuzzing tools by providing a unified interface for managing them. MISOFUZZ is a user-friendly solution for researchers who want to implement cooperative fuzzing in their workflow. It simplifies setting up and managing a distributed deployment of analysis tasks and provides a unified interface for tracking and evaluating the testing results. Our tool distinguishes itself from the state-of-the-art cooperative framework COLLABFUZZ [9] by leveraging the scheduling core of FUSEBMC [5] and extending its capabilities through module clustering. We believe MISOFUZZ will contribute to advancing vulnerability discovery and increasing the accessibility of cooperative fuzzing for researchers. A demonstration is available at: https://www.youtube.com/watch?v=q_LfxiTHeIo.

1 INTRODUCTION

Multiple factors can impact the performance of a fuzzing tool, including the complexity of the program under test, the size of the input space, the quality of the seed inputs, and the chosen fuzzing strategy [3, 6]. Balancing these factors can be challenging, and different tools and techniques may be better suited to different use cases. *Cooperative fuzzing*, an approach to software testing that involves running multiple instances of a fuzzing tool in parallel, each with a different configuration or input set, addresses this issue. This approach can improve the performance of the fuzzing process and increase the likelihood of discovering bugs [7]. By distributing the workload across multiple instances and sharing the discovered seeds, the time required to complete the fuzzing process can be reduced. This can be especially beneficial for more extensive or more complex programs, where different fuzzers can adopt various strategies to increase coverage and decrease computation time.

The main disadvantages of the existing cooperative fuzzing tools are either lack of scalability (e.g., *FuSeBMC* [5] runs on a single processor) or subpar input generation techniques and subsequent inability to reach the code paths protected by the complex predicate guards (*Collabfuzz* [9]) within a reasonable time. Our approach is derived from the strengths of the two solutions mentioned above, aiming to eradicate the mentioned weaknesses.

Our main contributions are summarized as follows:
(1) Provide a scalable execution environment for *FuSeBMC* that spans across multiple hardware nodes and processors.
(2) Introduce software (bounded) model checking [1] assisted fuzzing into the realm of large-scale cooperative fuzzing, currently populated by "simpler" fuzzing solutions, effectively combining testing and verification approaches.
(3) Implement a distributed fuzzing framework, with ease of integration and modularity as priority goals.
(4) Evaluate the resulting tool, benchmarking it against the conventional parallel fuzzing frameworks (*CollabFuzz* [9], *EnFuzz* [7]).

2 MISOFUZZ: A MODULAR INFRASTRUCTURE FOR SCALABLE FUZZING ORCHESTRATION

Our tool goes beyond the typical multi-process approach [8, 4, 2], allowing various fuzzers to work simultaneously on the same target in a scalable and modular fashion.

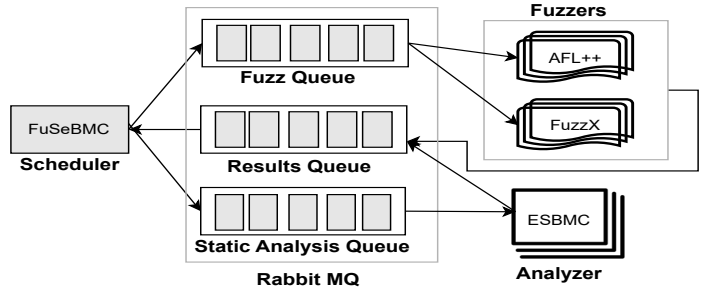

Figure 1: An overview of the MISOFUZZ architecture.

MISOFUZZ deploys a Kubernetes cluster across multiple powerful machines to efficiently distribute fuzzing tasks. Additionally, we extend the cooperative algorithm implemented by *FuSeBMC* to scale and schedule analysis tasks.

Although cooperative fuzzing is a practical testing approach, it is also critical to managing the coordination and communication between the fuzzers and analyzers. To address this, we developed a custom data exchange format. Implemented over the message broker, it provides a robust communication tool, greatly simplifying the integration of existing and new analysis tools.

The analysis engines are the critical part of the fuzzing process. In order to run the engines in a distributed environment, the respective tools have to be encapsulated in an adapter that implements the data-exchange format mentioned above and provides the means to spawn the engine onto the cluster. We implement the adapters for *AFL++* [2] and *ESBMC* [1], which were previously supported by *FuSeBMC* [5]. The analysis engines are scaled on-demand, depending on the available hardware resources and the job backlog pressure.

We demonstrate the effectiveness of our approach with the analysis results of a sample target, where *FuSeBMC* [5] cooperates with multiple fuzzers in a distributed fashion, including a novel fuzzer, currently in the final stages of development within our research group, as shown in Figure 1. The diagram describes the high-level architecture of MISOFUZZ, with *FuSeBMC* as a scheduler for a distributed cluster of the Fuzzing and Analysis workers. The message exchange over the network is enabled through the *RabbitMQ* message broker and consists of *Task Queues* (Fuzz Queue, Static Analysis Queue) and the *Results Queue*. The worker pools are created for each adapted engine integrated with *FuSeBMC* and are scaled independently. Workers in each pool retrieve the tasks from their respective queues and push the results back to *FuSeBMC* upon completion. The proposed architecture is modular and is not limited to the engines featured in the scope of this iteration of MISOFUZZ.

REFERENCES

[1] M. R. Gadelhaet al. Esbmc 5.0: An industrial-strength c model checker. ASE, 2018.
[2] A. Fioraldi et al. AFL++: Combining incremental steps of fuzzing research. In *USENIX (WOOT 20)*, 2020.
[3] G. Klees et al. Evaluating fuzz testing. *CCS*, 2018.
[4] J. Liang et al. Pafl: Extend fuzzing optimizations of single mode to industrial parallel mode. ESEC/FSE, 2018.
[5] K. Alshmrany et al. Fusebmc: A white-box fuzzer for finding security vulnerabilities in c programs. FASE, 2021.
[6] M. Böhme et al. Fuzzing: On the exponential cost of vulnerability discovery. ESEC/FSE, 2020.
[7] Y. Chen et al. Enfuzz: Ensemble fuzzing with seed synchronization among diverse fuzzers. In *USENIX*, 2019.
[8] Y. Li et al. A large-scale parallel fuzzing system. ICAIP, 2018.
[9] S. Österlund et al. Collabfuzz: A framework for collaborative fuzzing. EuroSec, 2021.