TuneFuzz: Adaptively Exploring Target Programs

Han Zheng
HexHive, EPFL
Lausanne, Switzerland
han.zheng@epfl.ch

Flavio Toffalini
HexHive, EPFL
Lausanne, Switzerland
flavio.toffalini@epfl.ch

Mathias Payer
HexHive, EPFL
Lausanne, Switzerland
mathias.payer@epfl.ch

ABSTRACT

In this report, we present TuneFuzz, an extension of FishFuzz that introduces two key improvements: an optimization that targets different sets of code locations (allowing user-selection of targets) and removes the need for Link Time Optimization. Subsequently, TuneFuzz achieves the 2nd place in SBFT24.

KEYWORDS

fuzzing, sanitizer, input prioritization

ACM Reference Format:
Han Zheng, Flavio Toffalini, and Mathias Payer. 2024. TuneFuzz: Adaptively Exploring Target Programs. In *2024 ACM/IEEE International Workshop on Search-Based and Fuzz Testing (SBFT '24), April 14, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3643659.3648564

1 INTRODUCTION

Fuzzing [2, 9, 12] is an automated software testing technique, which has proven its efficiency in bug hunting [8] and draws interest from both academia and industry. Traditional Greybox Fuzzers [5–7] treat all code locations equally, trying to maximize the code coverage by exploring the whole program space. Directed Greybox Fuzzers [1, 3], however, narrow the search to the predefined target sites and focus on exploiting bugs in the given locations.

FishFuzz [13] is a Sanitizer-Guided Greybox Fuzzer, which tries to find a balance between *exploration* and *exploitation* by introducing a new input prioritization mechanism. In the evaluation, FishFuzz notably boosts the coverage and finds up to 2.8x bugs compared to the baseline. However, the current FishFuzz prototype requires Linking Time Optimization (LTO) [4], which swaps the order of the coverage pass and sanitizer pass, thereby introducing superfluous instrumentations and reducing the execution speed. On the other hand, LTO mode introduces compatibility issues on some FuzzBench targets [10]. Concurrently, FishFuzz relies on sanitizer instrumentation for guidance [11], which limits its application to non-sanitized targets.

In this report, we propose TuneFuzz as an FishFuzz extension to address the above challenges. Specifically, TuneFuzz introduces static analysis passes as an LLVM patch and targets all code regions. Moreover, TuneFuzz introduces additional engineering optimizations in queue culling to enhance performance.

To better demonstrate TuneFuzz's capability in complex scenarios, we evaluate TuneFuzz across four real-world targets. Overall, TuneFuzz boosts coverage up to 79.62% over the baseline AFL++. In the final SBFT24 fuzzing competition, TuneFuzz ranks 2nd.

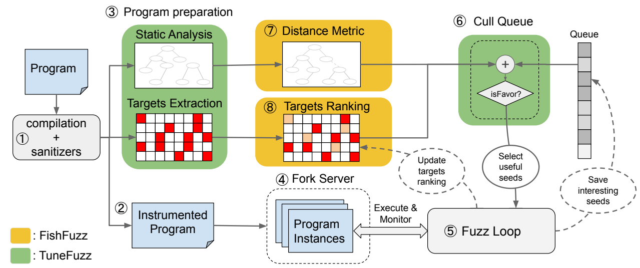


Figure 1: Overview of TuneFuzz workflow, The green boxs indicate the TuneFuzz modifications compared to FishFuzz.

2 DESIGN AND IMPLEMENTATION

Figure 1 depicts the overall workflow. TuneFuzz inherits the overall design of FishFuzz and primarily optimizes the Preprocessing (③) and Queue Culling modules (⑥).

Static Analysis FishFuzz requires Linking Time Optimization to run the analysis pass after the sanitizer pass. To mitigate the LTO requirement, TuneFuzz patches the LLVM source code to enforce the correct order of the analysis passes. Consequently, TuneFuzz allows the FishFuzz analysis without modifying the compile process.

Target Extraction One of the FishFuzz core contributions involves scaling the target set to hundred thousands of targets. This leaves the potential for TuneFuzz to target all code locations. Compared to FishFuzz, which targets upon sanitizer labels, TuneFuzz allows the user to decouple the sanitizer, boosting the execution speed in 'classic' undirected greybox fuzzing mode while still benefiting from FishFuzz's fast *exploration*.

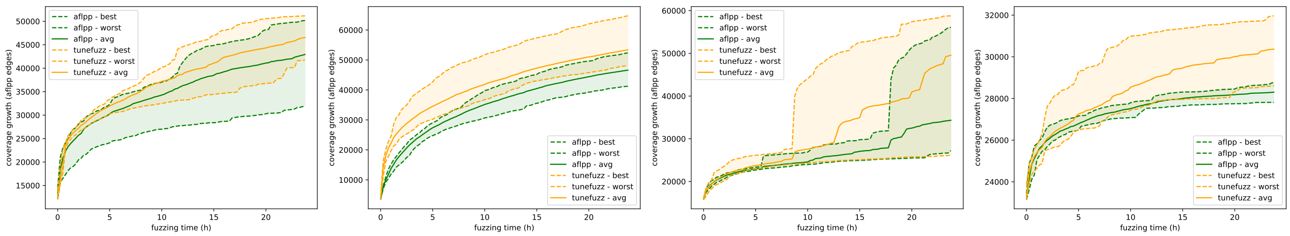
Cull Queue While vanilla FishFuzz emphasizes the *exploitation* capability, the FuzzBench competition prioritizes

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SBFT '24, April 14, 2024, Lisbon, Portugal

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0562-5/24/04
https://doi.org/10.1145/3643659.3648564



(a) FFmpeg **(b) WireShark** **(c) V8** **(d) PDFium**

Figure 2: Coverage of TuneFuzz and AFL++ in 10 round's 24h campaign.

exploration more. TuneFuzz, inspired by the FuzzBench results, fine-tuned each stage's hyper-parameter to optimize the *exploration* strategy.

3 EVALUATION

Before the SBFT24 competition, we evaluate TuneFuzz against vanilla AFL++ [5] on four complex real-world applications. We choose Wireshark, FFmpeg, Chromium V8, and Chromium Pdfium as benchmarks. To ensure a fair comparison, we replay the results on AFL++ instrumented binary to avoid edge collision. Our setup follows the same configuration as FuzzBench (*e.g.,* havoc-only, no sanitizer) and mount the disk as tmpfs to allow in-memory fuzzing. We release the full setup and 10 rounds' corpus for replication https://github.com/kdsjZh/FishFuzz-Seed-eval.

Figure 2 demonstrate that TuneFuzz significantly outperforms AFL++. For the Chromium component V8 and PDFium, we observe a notable boost starting from 10h, TuneFuzz covers 79.62% and 37.94% more edges compare with AFL++ (we exclude the edges covered by the initial corpus). In WireShark and FFmpeg, TuneFuzz enhances AFL++ as well, TuneFuzz's average edge finding in Wireshark is even better than AFL++'s best coverage

CONCLUSION

TuneFuzz, an extension of FishFuzz, has been successfully integrated into FuzzBench. Our evaluation demonstrates TuneFuzz' *exploration* capability by notably boosting over AFL++. In SBFT'24 competition, TuneFuzz ranks the 2nd place. It is openly available at https://github.com/HexHive/FishFuzz.

REFERENCES

[1] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. 2017. Directed greybox fuzzing. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 2329–2344.

[2] Marcel Böhme, Van-Thuan Pham, and Abhik Roychoudhury. 2016. Coverage-based greybox fuzzing as markov chain. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1032–1043.

[3] Hongxu Chen, Yinxing Xue, Yuekang Li, Bihuan Chen, Xiaofei Xie, Xiuheng Wu, and Yang Liu. 2018. Hawkeye: Towards a desired directed grey-box fuzzer. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 2095–2108.

[4] Mary F Fernandez. 1995. Simple and effective link-time optimization of Modula-3 programs. In *Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation*. 103–115.

[5] Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. 2020. {AFL++}: Combining incremental steps of fuzzing research. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*.

[6] Shuitao Gan, Chao Zhang, Peng Chen, Bodong Zhao, Xiaojun Qin, Dong Wu, and Zuoning Chen. 2020. {GREYONE}: Data flow sensitive fuzzing. In *29th USENIX security symposium (USENIX Security 20)*. 2577–2594.

[7] Shuitao Gan, Chao Zhang, Xiaojun Qin, Xuwen Tu, Kang Li, Zhongyu Pei, and Zuoning Chen. 2018. Collafl: Path sensitive fuzzing. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 679–696.

[8] google. 2023. ossfuzz bugs. https://security.googleblog.com/2023/02/taking-next-step-oss-fuzz-in-2023.html.

[9] Caroline Lemieux and Koushik Sen. 2018. Fairfuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*. 475–485.

[10] Jonathan Metzman, László Szekeres, Laurent Simon, Read Sprabery, and Abhishek Arya. 2021. Fuzzbench: an open fuzzer benchmarking platform and service. In *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 1393–1403.

[11] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitriy Vyukov. 2012. {AddressSanitizer}: A fast address sanity checker. In *2012 USENIX annual technical conference (USENIX ATC 12)*. 309–318.

[12] Michal Zalewski. 2013. american fuzzy lop. https://lcamtuf.coredump.cx/afl/.

[13] Han Zheng, Jiayuan Zhang, Yuhang Huang, Zezhong Ren, He Wang, Chunjie Cao, Yuqing Zhang, Flavio Toffalini, and Mathias Payer. 2023. {FISHFUZZ}: Catch Deeper Bugs by Throwing Larger Nets. In *32nd USENIX Security Symposium (USENIX Security 23)*. 1343–1360.