A Qualitative Analysis of Fuzzer Usability and Challenges

Yunze Zhao
University of Maryland

Wentao Guo
University of Maryland

Harrison Goldstein
University of Maryland

Daniel Votipka
Tufts University

Kelsey Fulton
Colorado School of Mines

Michelle Mazurek
University of Maryland

Abstract

Fuzzing is a widely adopted technique for uncovering software vulnerabilities by generating random or mutated test inputs to trigger unexpected behavior. However, little is known about how developers actually use fuzzing tools in practice, the challenges they face, and where current tools fall short. This study investigates the human side of fuzzing via 18 semi-structured interviews with fuzzing users across diverse domains. These interviews explore participants' workflows, frustrations, and expectations around fuzzing, revealing critical usability gaps and design opportunities. The results can inform the next generation of fuzzing tools to improve user experience, reduce manual effort, and enable more effective integration of fuzzing into real-world workflows.

1 Introduction

Fuzzing is an automated software testing technique that generates unexpected or malformed inputs to uncover crashes, hangs, and other anomalous behaviors in target programs [11,19]. It plays a central role in secure software development, with tools like AFL++, libFuzzer, and OSS-Fuzz responsible for identifying thousands of critical bugs across widely used systems [1,2,4,7,8].

Despite its technical success and widespread use, fuzzing tools are often difficult to configure, integrate, and interpret—especially for developers and engineers outside of the security research community [3, 8]. Existing research has largely prioritized performance improvements (e.g., maximizing code coverage [10,17], better mutations [6,9], improving efficiency [16,18]) over usability concerns such as learnability, workflow integration, and actionable feedback.

This mismatch between technical advancement and practical usability creates barriers to adoption and reduces the effectiveness of fuzzing in real-world contexts. Understanding how developers actually use fuzzers—where they succeed, where they struggle, and how they adapt tools to fit their workflows—is essential for improving tool design. While a few recent studies have begun to document fuzzer usability issues [5, 12–15], they often offer limited visibility into how experienced practitioners incorporate fuzzers into their daily workflows.

To address this gap, we conducted a qualitative study of 18 experienced fuzzing practitioners across academia and industry. Through in-depth semi-structured interviews, we explored their workflows, challenges, and expectations, aiming to surface actionable insights for designing more usable, human-centered fuzzing tools. Our study focuses on the following research questions:

RQ1: What specific challenges do users face across the lifecycle of a fuzzing campaign in real-world deployments?
RQ2: What strategies do practitioners use to address or work around these challenges?
RQ3: What improvements to fuzzing tools would better support practical adoption and real-world workflows?

2 Methods

We conducted 18 semi-structured interviews with security engineers, software developers, and researchers who use fuzzing tools in practice. Participants were recruited via community outreach, targeted invitations based on fuzzing activity, and snowball sampling. To ensure broad coverage, we included participants with varying experience levels and roles (e.g., development, DevOps, research). Interviews were conducted remotely and transcribed using local automated tools. Transcripts were analyzed using collaborative thematic analysis, resulting in a codebook organized around three core stages of fuzzing workflows: (1) configuring fuzzers and preparing

USENIX Symposium on Usable Privacy and Security (SOUPS) 2025.
August 10–12, 2025, Seattle, WA, United States.

targets, (2) running and monitoring fuzzing campaigns, and (3) integrating fuzzers into broader development processes.

3 Findings

Our analysis revealed usability challenges, participants' strategies to cope with these challenges, and suggestions for fuzzing tool improvements across three key areas: setup and configuration, monitoring and output interpretation, and workflow integration.

Users' Understanding of Fuzzers: While all participants had experience with fuzzing, their conceptual understanding varied widely. Many described fuzzing as essential—uniquely suited to uncovering bugs that traditional testing misses—but simultaneously viewed fuzzers as black boxes. This tension arose from how they learned to use fuzzers: mostly through informal resources like blog posts, trial-and-error experimentation, and skimming source code.

As a result, mental models were often fragmented. Participants exhibited inconsistent understanding on concepts or how to assess fuzzing effectiveness. These gaps led to potential inconsistent practices and uncertainty when interpreting tool behavior. Despite this, all participants expressed strong belief in fuzzing's potential, describing it as a critical component of their software testing mindset, even when it didn't integrate smoothly with their workflows.

Setup and Configuration: Participants reported that setting up fuzzing campaigns was one of the most difficult parts of the workflow. Challenges included compiler instrumentation, configuring input delivery, identifying appropriate entry points, and understanding tool-specific flags. Many defaulted to using test suite inputs or guessed which functions "looked interesting" to fuzz.

To cope, participants employed fast, iterative setup cycles, getting the fuzzer running minimally and quickly, refining as they go. Some used Docker to isolate builds, while others relied on simplified harnesses to validate execution before expanding coverage. Participants wished for more automation to reduce the manual burden of setup. Suggested improvements included automatic harness generation based on program structure, static analysis tools to identify promising fuzzing targets, and IDE integrations to guide configuration. Several envisioned plugins that could scaffold a fuzzing campaign with minimal user input—e.g., selecting a function, auto-generating a test harness, and running the fuzzer with sane defaults.

Monitoring and Output Interpretation: Once campaigns were running, participants reported struggling to interpret output or measure progress. Coverage metrics, crash logs, and timeout reports often lacked context or clarity. Users distrusted de-duplication heuristics and typically wrote custom scripts or manually triaged results to reduce noise.

Participants emphasized the need for more actionable runtime feedback to better understand and monitor fuzzing campaigns. Many requested dashboards or lightweight visualizations to make the process more transparent and interactive. A recurring theme was the desire for tools that could signal when a campaign had likely stalled or reached diminishing returns. More specifically, participants suggested enhancements such as richer logging that includes mutation history and parent seed tracking, customizable crash classification to better support triage, and notifications when the fuzzer is no longer making progress. Others envisioned IDE-based integrations or web dashboards that tie fuzzing results directly to source code locations, helping users contextualize and act on findings more effectively.

Workflow Integration and Tool Flexibility: Participants noted that many fuzzing tools are poorly integrated into real-world workflows, and its tooling was often designed for narrow use cases. Adapting fuzzers to large or modern codebases required custom scripts or significant manual effort. Advanced tools like hybrid fuzzers or symbolic-assisted fuzzers offered benefits theoretically, but were described as difficult to configure or evaluate in practice.

To address these issues, participants advocated for modular fuzzer designs where components such as mutators, feedback engines, schedulers could be replaced independently. Rather than relying on rigid, bundled toolchains, they wanted standalone components with well-documented APIs that could be composed to fit different environments. They also called for more adaptable fuzzers that could handle more complex targets (e.g., business logic, distributed systems) and better interoperate with IDEs, debuggers, or static analysis tools. Others emphasized the need for broader applicability, supporting not just C/C++ binaries, but also modern languages and layered systems. Participants also proposed lightweight integration hooks for CI pipelines and the ability to export results in developer-friendly formats.

4 Discussion

We conclude with a few directions for improving fuzzers based on participants' responses. Fuzzers should provide better defaults, guided configuration, and automation for setup tasks such as harness generation and target selection to reduce early-stage friction. They should also adopt modular, flexible architectures, allowing users to customize and swap components for advanced usage through well-documented APIs. In terms of output, fuzzers should deliver richer runtime feedback—such as mutation history, progress indicators, or customizable crash classifications—along with improved crash de-duplication techniques. Finally, fuzzers should integrate more seamlessly into development workflows by supporting CI pipelines and IDE integrations, and by offering interactive dashboards that improve visibility into fuzzing campaigns.

References

[1] Fuzzing introspection of oss-fuzz projects.

[2] Google fuzztest.

[3] libFuzzer – a library for coverage-guided fuzz testing. — LLVM 21.0.0git documentation.

[4] Marcel Boehme, Cristian Cadar, and Abhik ROYCHOUDHURY. Fuzzing: Challenges and Reflections. 38(3):79–86.

[5] Marcel Böhme, Cristian Cadar, and Abhik Roychoudhury. Fuzzing: Challenges and reflections. *IEEE Software*, 38(3):79–86, 2020.

[6] Sang Kil Cha, Maverick Woo, and David Brumley. Program-adaptive mutational fuzzing. In *2015 IEEE Symposium on Security and Privacy*, pages 725–741. IEEE, 2015.

[7] Max Eisele, Marcello Maugeri, Rachna Shriwas, Christopher Huth, and Giampaolo Bella. Embedded fuzzing: A review of challenges, tools, and solutions. 5(1):18.

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

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

[10] Chenyang Lyu, Shouling Ji, Yuwei Li, Junfeng Zhou, Jianhai Chen, and Jing Chen. Smartseed: Smart seed generation for efficient fuzzing. *arXiv preprint arXiv:1807.02606*, 2018.

[11] Richard McNally, Ken Yiu, Duncan Grove, and Damien Gerhardy. Fuzzing: The state of the art. 2012.

[12] Timothy Nosco, Jared Ziegler, Zechariah Clark, Davy Marrero, Todd Finkler, Andrew Barbarello, and W. Michael Petullo. The industrial age of hacking. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1129–1146. USENIX Association, August 2020.

[13] Olivier Nourry, Yutaro Kashiwa, Bin Lin, Gabriele Bavota, Michele Lanza, and Yasutaka Kamei. The human side of fuzzing: Challenges faced by developers during fuzzing activities. *ACM Trans. Softw. Eng. Methodol.*, 33(1), November 2023.

[14] Stephan Plöger, Mischa Meier, and Matthew Smith. A qualitative usability evaluation of the clang static analyzer and libFuzzer with CS students and CTF players. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, pages 553–572. USENIX Association, August 2021.

[15] Stephan Plöger, Mischa Meier, and Matthew Smith. A usability evaluation of afl and libfuzzer with cs students. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[16] Dongdong She, Abhishek Shah, and Suman Jana. Effective seed scheduling for fuzzing with graph centrality analysis. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2194–2211. IEEE, 2022.

[17] Junjie Wang, Bihuan Chen, Lei Wei, and Yang Liu. Skyfire: Data-driven seed generation for fuzzing. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 579–594. IEEE, 2017.

[18] Xiaoqi Zhao, Haipeng Qu, Wenjie Lv, Shuo Li, and Jianliang Xu. Moofuzz: Many-objective optimization seed schedule for fuzzer. *Mathematics*, 9(3):205, 2021.

[19] Xiaogang Zhu, Sheng Wen, Seyit Camtepe, and Yang Xiang. Fuzzing: a survey for roadmap. *ACM Computing Surveys (CSUR)*, 54(11s):1–36, 2022.

5 Appendix: Interview Protocol

Note: The following questions and messages represent the general structure and content of the interview, but may not be asked verbatim. The phrasing or order may be adjusted to fit the flow of the conversation or specific context.

5.1 Introduction

Hi, my name is _____. Thank you for taking the time to participate in our interview study. Our research focuses on understanding how developers and computer security professionals like you use fuzzing tools in your work. We aim to identify both the strengths and challenges of these tools to help guide improvements and make them more effective for the broader computer science community.

Your responses are valuable to us, and we want to assure you that your data will be stored securely. We will not release any sensitive information about you. Importantly, we do not expect you to share any corporate secrets or proprietary information. If at any point you're concerned about disclosing such information, please feel free to skip any questions.

This interview will be transcribed to support later analysis, and all transcripts will be destroyed once the study is complete. We will also share the results of our study with you.

Before we begin, do you have any questions about the study or the consent form you filled out?

5.2 Broad Questions / Background

Before we dive into the main questions, I'll start with a few general questions about your experience using fuzzing tools. These initial questions help us understand the context of your work.

- Can you describe a typical scenario where you've used a fuzzing tool?
  - What has your experience been like in those situations?
  - How frequently do you use fuzzing tools, and in what contexts?
- How did you learn to use fuzzing tools?
  - Were there any resources or people that helped along the way?

5.3 Specific Uses and Challenges

Now that we've covered some background, I'd like to focus on the specific ways you use fuzzing tools and the challenges you may have encountered. This will help us understand the practical aspects of fuzz testing in your work.

- Can you walk me through the steps you follow when setting up and using a fuzzing tool?
  - what's involved in setting up the driver, providing seeds, and analyzing the outputs?
  - Of these tasks, which ones are most time-consuming or challenging, and why?
- What do you think are some good use cases for the fuzzer you use?
  - Have you had to modify or adapt a fuzzer? Can you give an example?
  - Can you share an example of a critical bug found using fuzzing?
  - **If no example**: What types of findings do you consider most valuable?
- What expectations do you have when using a fuzzer? Has it ever failed to meet them?
  - Did you work around the limitation? Use other tools?
- Is there anything you changed about your fuzzing process over time to make it more successful?
- Do you fuzz complex functions? If so, how do you choose them? If not, why?
- What challenges have you experienced integrating fuzzing into other testing processes?
  - Are those challenges related to the tool, workflow, or something else?
  - Do you foresee any learning curve or adoption challenges in getting your team or teammate (intern) to use fuzzing tools effectively? How might you address these challenges?
  - **If no integration**: Do you think integrating fuzzing tools with your existing testing processes (like unit testing, static analysis, or CI/CD) would require significant adjustments? If so, what kind of adjustments would be necessary? What are the challenges?
  - Have these challenges impacted testing efficiency?
  - What part of the tool presents challenges for integration?

5.4 Managing and Interpreting Fuzzer Output

Next, let's talk about how you manage and interpret the results from fuzzing tools. This section is about understanding how you handle the data generated by fuzzers and how you make decisions based on that information.

- At your work, how do you interpret and manage the results generated by a fuzzer?
  - What tools or methods do you use to organize and prioritize fuzzer findings?
  - if they mention specific tools: How effective do you find these tools in helping you manage large volumes of data?
- Do the fuzzing tool you use in your workflow produce a large amount of output?
  - **if yes**:
    - What are some of the biggest challenges you've encountered when managing the large volume of findings generated by fuzzers? How do you usually handle those?
    - Can you give us an example of the challenges you mentioned?
    - Which issues do you address first?
    - What factors influence your prioritization process?
  - **if no**:

        - How do you review the output generated by your fuzzing tool? What are you looking for in these outputs?
        - Do you feel the fuzzing tool provides enough actionable insights even with a limited number of results?
- Have you ever encountered cases where a critical issue was hidden in less important findings? How did you identify it?
    - What strategies do you use to manage duplicate reports?
    - **if they struggle with duplication**: Have you found any tools or methods that help reduce the time spent on managing duplicates?
- How do you decide if you've done enough fuzzing?
    - Do you feel unsure about whether enough fuzz testing has been done? What steps do you take when you're uncertain? Can you give me an example?
    - What are some techniques you use to verify that you have done enough testing
    - **if they mention uncertainty in completion**: How do you handle situations where you're unsure if enough testing has been done?

5.5 Opportunities for Improvement

Finally, I'd like to explore your thoughts on how fuzzing tools could be improved. This section is focused on identifying potential enhancements that could make fuzz testing more effective and efficient in your work.

- In your experience, are there any recurring issues or frustrations with fuzzing tools that you wish could be addressed? What specific improvements would make the most difference in your day-to-day work?
    - Sometimes new features come with trade-offs. Do you think adding these improvements might introduce new challenges, such as increasing complexity or generating too much data to process?
    - **if they suggest a specific feature**: Have you seen any tools that already offer something similar, or would this be a completely new capability?
- What feedback do you think fuzzer can provide to boost your confidence that your program is thoroughly tested?
- How well do you think fuzzing fits with the other tools and testing methods you currently use? Are there any testing tools you'd like to see integrated with your fuzzer to the testing process?
    - What specific capabilities would you want from such an integration?
    - Do you think integrating fuzzers with tools like static analysis or UI-based reverse engineering would make it easier to find and fix issues? How would that look in your workflow?
    - Are there any concerns or challenges you think might arise from integrating these tools?
- If you could create the ultimate fuzzing tool for your work, what are the top three features it would need to have? How would those features solve the biggest issues you face today?
    - **if they struggle to describe features**: Is there a particular problem or inefficiency in your current process that you would want this ideal tool to solve?

5.6 Wrapping Up

Thank you so much for your time and insights today. Your information is incredibly valuable to our study, and we truly appreciate your contribution. Before we wrap up,

- Is there anything else you'd like to share about your experience with fuzzing or anything we haven't covered that you think is important?
- Do you have any questions about this research or anything we've discussed?
- Lastly, if you know anyone else who might be interested in participating in this study, we'd greatly appreciate it if you could refer them to us.

We'll be sure to share the results with you once the study is complete. Thanks again, and have a great day!