Visualization Task Taxonomy to Understand the Fuzzing Internals (Registered Report)

Sriteja Kummita
Fraunhofer IEM
Paderborn, Germany
sriteja.kummita@iem.fraunhofer.de

Miao Miao
University of Texas at Dallas
Richardson, TX, USA
mmiao@utdallas.edu

Eric Bodden
Heinz Nixdorf Institute, Paderborn University
Fraunhofer IEM
Paderborn, Germany
eric.bodden@uni-paderborn.de

Shiyi Wei
University of Texas at Dallas
Richardson, TX, USA
swei@utdallas.edu

Abstract

Greybox fuzzing is used extensively in research and practice. There are umpteen improvements proposed in the literature to improve greybox fuzzing. However, to what extent do these improvements affect the internal components (or internals) of a given fuzzer is not yet understood as the improvements are mostly evaluated in terms of code coverage and bug finding capability. Such an evaluation is insufficient to understand the effect of improvements on the internals of fuzzer. Some of the literature developed tools to visualize the outcomes of the fuzzing to enhance the understanding. However, they only focus on high-level information and no previous research on visualization has been dedicated to understanding fuzzing internals.

To close this gap, we propose the first step towards the development of a fuzzing-specific visualization framework: a taxonomy of visualization analysis tasks that fuzzing experts desire to help them understand the internals of fuzzing. Our approach involves conducting semi-structured interviews with fuzzing experts and using qualitative data analysis to systematically extract the task taxonomy from the interview data. We also evaluate the support of existing visualization tools for fuzzing through the lens of our taxonomy. In our pilot study, we conducted interviews with six fuzzing experts and extracted a preliminary taxonomy. We aim to conduct another 20 interviews to gain more insights and make the taxonomy more robust at Phase 2.

CCS Concepts

• **Human-centered computing** → **User studies**; **Usability testing**; • **Security and privacy** → Software security engineering.

Keywords

Task taxonomy, Fuzzing, Fuzz testing, Visualization analysis

FUZZING '24, September 16, 2024, Vienna, Austria
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1112-1/24/09
https://doi.org/10.1145/3678722.3685530

ACM Reference Format:
Sriteja Kummita, Miao Miao, Eric Bodden, and Shiyi Wei. 2024. Visualization Task Taxonomy to Understand the Fuzzing Internals (Registered Report). In *Proceedings of the 3rd ACM International Fuzzing Workshop (FUZZING '24), September 16, 2024, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3678722.3685530

1 Introduction

Fuzzing or fuzz testing is a dynamic testing technique in which the system under test (SUT) is iteratively executed with semi-randomly created inputs with the aim to find interesting behaviors such as security vulnerabilities. Greybox fuzzing (GF) is a technique that uses feedback from the runtime environment such as code coverage, to guide its iterative input generation and prioritization process [12]. GF is extensively used in research and practice due to its high performance in automated vulnerability discovery [4, 24]. There are numerous publications in GF that propose improvements to the internals of the fuzzer[1] [2, 3, 20, 25, 42, 44, 47]. However, the evaluation of such improvements is complex due to the inherent randomness in fuzzing [31].

Currently, the most used evaluation criteria in the fuzzing community are runtime code coverage and bug finding capability. A recent study of 250 fuzzing publications from the top conferences by Schloegel et al [34] reveals that: 77% of the papers evaluate the improvements using some kind of code coverage and 71% evaluate using bug finding capability. Though it is tempting to infer that a fuzzer that reaches more code or finds more bugs is better than others, it is hard to measure the effect of improvements made to the fuzzer on its internals when evaluated using such holistic metrics.

To understand fuzzing in depth, some literature focused on visualizing the outcomes of the fuzzing campaigns [11, 16, 43, 48]. However, the existing visualizations for fuzzing only focus on high-level information such as the covered code, the call graph and interesting inputs generated over time. None of these visualizations provide information about how the internals of fuzzing work.

To design and develop a suitable visualization framework, it is important to understand the questions (or tasks) that the user[2] asks to derive insights by visually analyzing the data [7, 35]. These tasks

[1] In the context of this paper, fuzzer refers to a tool that performs greybox fuzzing.
[2] In the context of this paper, a *user* refers to a fuzzing expert.

might be exploratory in nature and help in revealing new important insights to the users.

To fill the gap in this area, this study presents the taxonomy of visualization analysis tasks to understand the internals of fuzzing. The tasks are derived by conducting semi-structured interviews with fuzzing experts and performing qualitative data analysis using open coding protocol [27]. To extract reliable information from interviews, we only recruited the participants having expertise in GF and followed the coding protocol with two coders [39].

Advantages of such a taxonomy are three-fold: (1) it guides the design and development of domain specific visualization framework [28, 37]; (2) it helps in evaluating the usability of existing visualization frameworks; (3) it suggests new criteria for fuzzing evaluations as opposed to the holistic metrics [26]. While similar taxonomies of visualization analysis tasks do exist in other domains such as biological pathway data [30], graph visualizations [19], and temporal network visualizations [1], we are the first to conduct such a study in the area of fuzzing.

The main contributions of our work are as follows:

- Semi-structured interviews with fuzzing experts to gather insights on understanding the internals of fuzzing.
- Qualitative data analysis protocol to analyze the interview data.
- Taxonomy of visualization analysis tasks that fuzzing experts ask about the internals of fuzzing.
- Evaluation of existing visualization tools in fuzzing using our task taxonomy.

For the Phase 1 submission, we detail our approach and present the results from our pilot study of six interviews. In total, we have extracted 54 visualization analysis tasks across different combinations of fuzzing internals. Our study design and qualitative data analysis protocol helps us gather more insights for Phase 2: specifically, we plan to increase the scope of the study and conduct more interviews by approaching fuzzing experts in academia and industry and work towards the contributions.

The rest of the paper is organized as follows: Section 2 provides the necessary background information and motivates the study using an example visualization analysis task. Section 3 details the approach of the study and Section 4 discusses the results from our pilot study. Section 5 details the plan for the actual study and Section 6 discusses the threats to validity. Section 7 discusses the related work and Section 8 concludes the paper.

2 Background & Motivation

2.1 Greybox Fuzzing Internals

A greybox fuzzer executes the SUT (target) using iteratively generated inputs to increase runtime coverage and/or find abnormal behaviors. Figure 1 shows the common internal components (internals or internal stages) of the fuzzer. *Instrumentation* injects code into the target to get runtime feedback. *Initial seeds* help in starting the fuzzing campaign and they greatly influence the overall performance [13, 22, 32]. The fuzzer uses the initial seeds and the corresponding runtime information (e.g., branch coverage) in the next stages. *Search strategy* aims at selecting a seed from the seed queue that increases coverage based on some heuristics [17]. Intuitively search strategy decides the selection orders of the seeds in the seed queue for further processing. *Power schedule* assigns energy (number of mutations) to the selected seed using some heuristics (e.g., coverage gain) [46]. *Mutations* apply different mutation strategies to the selected seed (e.g., havoc or splice [10]) and generate mutants until the assigned energy is exhausted. *Execution* runs the instrumented target with the mutants and monitors the runtime environment for interesting behaviors such as crashes, increased coverage, etc., and decides if the corresponding seed or the mutant should be added to the seed queue or not.

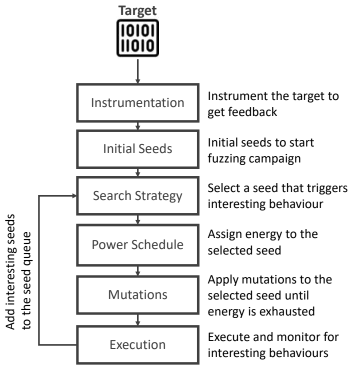
Target
10101
11010
Instrumentation
Instrument the target to get feedback
Initial Seeds
Initial seeds to start fuzzing campaign
Search Strategy
Select a seed that triggers interesting behaviour
Power Schedule
Assign energy to the selected seed
Mutations
Apply mutations to the selected seed until energy is exhausted
Execution
Execute and monitor for interesting behaviours
Add interesting seeds to the seed queue

Figure 1: Internal stages of greybox fuzzing.

We categorized these internal stages based on different claims made in the literature and after carefully inspecting different survey publications in GF [9, 21, 24, 26, 31, 46].

2.2 Visualization Analysis

In the context of this paper, we define a visualization analysis task as:

> *A question that a fuzzing expert asks to understand the internal stages of fuzzing using visualization analysis.*

Consider an example visualization analysis task, *how do the mutation strategies perform between two fuzzers?* The traditional way of answering the task is to evaluate the fuzzers using the holistic metrics (e.g., runtime code coverage or bug finding capability). Such an evaluation provides little to no information about the mutation strategies. However, evaluation using visualization analysis can provide some detailed insights into the task.

An example interactive visualization analysis workflow for the above task is shown in Figure 2. The workflow is developed using a domain-agnostic visualization framework, vega-lite [33], for demonstration purposes only. It shows the collected data from a fuzzing campaign with two fuzzers, AFLFast [6] and AFLVanilla (a variant of AFLFast after replacing AFLFast's search strategy with the one from AFL 1.94b [45]), for two mutation strategies (havoc and splice), on the target, *cxxfilt*.[3] The campaign is run for 24 hours and is repeated five times. The horizontal grouped line graphs (Figure 2a and Figure 2b) show the branch coverage over time for five fuzzing runs (trials), one line for each trial as shown in the legend to

[3] https://sourceware.org/binutils/docs/binutils/c_002b_002bfilt.html

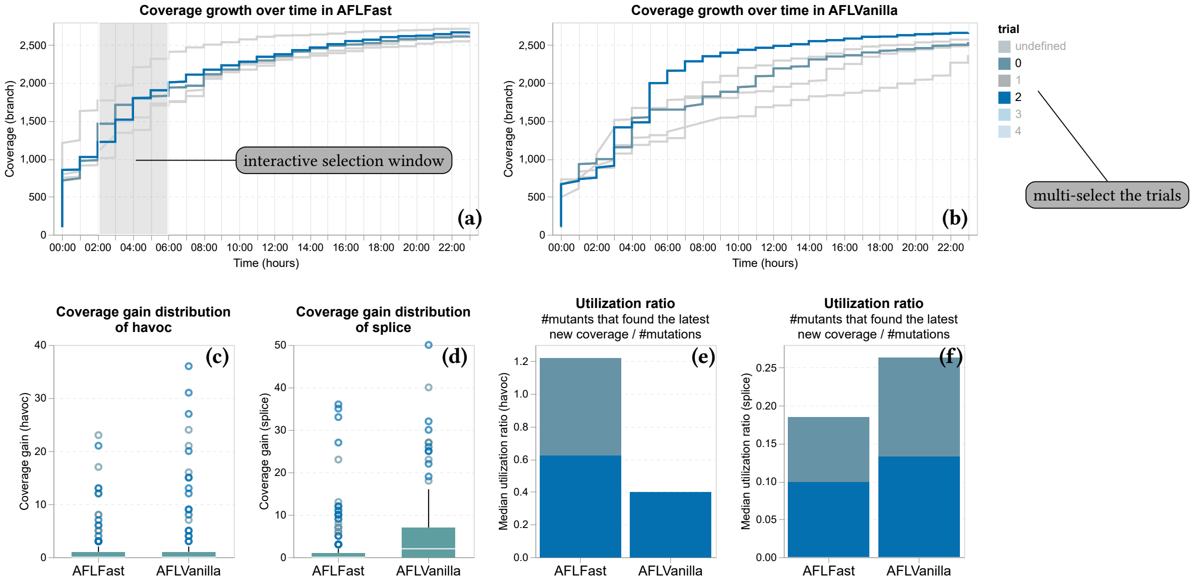

Figure 2: Example visualization workflow to understand the mutation strategies in AFLFast and AFLVanilla.

the right of Figure 2b. A user can select one or more trials simultaneously by clicking at the corresponding legend items and visually analyze the graphs (first and third trails are selected in Figure 2a and Figure 2b). There is a steep increase in coverage between 2nd and 6th hours according to the line graphs. The user can draw a window from 2nd to 6th hour (shown in Figure 2a) to analyze the underlying data. The user can also move the window on the x-axis and the data in other graphs are automatically synchronized.

Figure 2c and Figure 2d shows the coverage gain of different mutants generated by the mutation strategies (havoc and splice) for the selected window using box plots. When looking at just Figure 2c, it is not clear which fuzzer has better havoc mutation strategy. However, if we look at the corresponding utilization ratio (ratio of the number of mutations that found the latest new coverage to the total number of mutations) in Figure 2e, the AFLFast's havoc mutation strategy shows favorable performance in both of the selected trails (first and third trails are represented using two different colors). Figure 2d and Figure 2f show that AFLVanilla performs better in the splice mutation strategy respectively. The user can then verify for the similar pattern in other trails and time windows before arriving at a conclusion.

There are multiple ways of designing a visualization workflow for a single analysis task and choosing the right workflow is not the focus of this paper. This paper focuses on extracting the desired list of analysis tasks from the fuzzing experts. Having such a taxonomy of visualization analysis tasks provides a strong background to develop a fuzzing-specific visualization framework that can be used to specify what to visualize (data requirements), and how to visualize (visualization idioms [29], visualization workflows, and user interactions). The taxonomy also suggests new evaluation criteria focusing on fuzzing internals (and their combinations) to the fuzzing community.

3 Approach

Our approach involves in conducting semi-structured interviews with fuzzing practitioners and experts and performing qualitative data analysis to extract the necessary information and derive the taxonomy of visualization analysis tasks to understand fuzzing internals. The reproducibility of our approach lies in the design of the interview and formally describing the information extracted from the free-form interview text. The soundness of our approach is proportional to the expertise of the study participants. Figure 3 shows the overview of the approach, and the following sub-sections detail the steps.

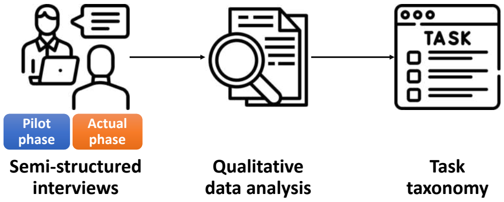

Figure 3: Overview of the approach.

3.1 Semi-structured Interviews

The goal of the study is to extract the visualization analysis tasks that the fuzzing practitioners are interested in and help to understand the internals of fuzzing. We follow semi-structured interview format in our study as they are flexible and provide a platform for open-ended discussions [14]. They allow the interviewer for follow-up questions to stimulate further discussions, receive spontaneous feedback, and gather deeper insights. This interview format requires using an initial question catalogue to steer the interview and focus on the topic of interest [36] (our artifact [18] contains the initial question catalogue used in the study). The interviews are carried out in two phases, pilot phase and actual phase where the main difference is in the participants.

3.1.1 Participants. To obtain meaningful insights, we restricted the study to the participants who have expertise in GF. In the *pilot phase*, we contacted seven fuzzing experts in our closed contacts and six of them agreed to participate in the study (details of the pilot study are described in Section 4). The participation was voluntary and without any incentives.

In the *actual phase*, we plan to use this paper to reach out to the wider audience in the fuzzing community and industry for participation. We are aiming for about 20 interviews with fuzzing experts with highest level of expertise to produce robust results (the plan for the actual study is described in Section 5).

3.1.2 Design. The interview is divided into four parts: (1) introduction and background, (2) identifying internal stage compositions, (3) identifying visualization analysis tasks, and (4) miscellaneous.

In the first part, the participant is asked to provide the demographical information on their expertise in GF. The moderator then provides background information on the internal stages of fuzzing (Section 2.1) and visualization analysis task with an example interactive visualization workflow (Section 2.2). The participant is also asked to comment on the categorization of the internal stages. In the second part, the participant is asked to identify the related internal stages and provide a reason about why the internal stages are related. In the third part, the participant is prompted to ask questions that help in understanding each internal stage and their relationships (identified in the previous part) using visualization analysis. The moderator also tries to ask follow-up questions and clarifications regarding the relationships to get deeper insights and stimulate further discussions. The final part of the interview is used to discuss additional ideas and clarifications if any.

Two test runs were conducted with researchers to adapt the interview design. The second and third parts are the most important parts of the interview as they provide the necessary information for qualitative data analysis and extracting the task taxonomy.

3.1.3 Data Collection. Each participant is contacted via an e-mail by providing general information about the study and the consent form. After agreeing to participate, the interviews are scheduled virtually using Microsoft Teams[4]. The average duration of the interview is 60 minutes, and the interviews are recorded (screen and audio) for post-processing. We used ConceptBoard[5] to keep the interview sessions interactive and record the intermediate data.

3.1.4 Ethics. We ensured that the signed consent form is received from the participants before proceeding with the interview. The consent form provides information about the study, voluntary participation, data collection and privacy. The study design is also approved by the corresponding ombudsperson of the Ethics committee.

3.2 Qualitative Data Analysis

We follow open coding protocol [27] to conduct qualitative data analysis on the second and third parts of the anonymized interview data. Open coding protocol requires agreeing up on the initial set of codes before performing the data analysis. The coding protocol is iterative; i.e., when new codes are identified, all the interview data should be re-analyzed to fit in the new set of codes. We separate all the visualization analysis tasks mentioned by the participants along with the corresponding explanations before performing the data analysis.

> Example visualization analysis task
>
> I can think about a question between initial seeds and execution. How does the energy vary from each initial seed to their particular mutants?

3.2.1 Task specification. We will use the example visualization analysis task from one of the interviews shown in the text box above, to understand the coding process.

Initially two coders agreed to code the interview data using the *Categorization* mentioned in Table 1: the example is coded as *inter stage task* because it involves two internal stages, initial seeds and execution.

After coding one interview transcript, the coders met to discuss the disagreements. However, using just the categorization was insufficient for the coders to uniquely identify the coded data and it also did not capture the necessary information from the interview transcript (the second part in our example is completely ignored).

Hence, the initial code is extended by adding three more attributes as shown in the Table 1: stage(s), related data, and type of data analysis. Such an updated code, which we denominate as *task specification*, provides a formal definition of the visualization analysis task, helps to uniquely identify the coded data, and facilitate disagreements. The task specification is agreed by the whole research team after discussing on the interview data. The third column in Table 1 shows the task specification for our example.

Table 1: Description of task specification.

Code	Description	Example
Categorization	*intra stage task* (only one internal stage is involved), *inter stage task* (multiple internal stages are involved), *overall task* (all internal stages are involved), *non-visualization task* (when the question can be answered without visualization analysis)	inter stage task
Stage(s)	The involved stages in the task according to the categorization in Figure 1.	initial seeds, execution
Related data	The data that needs to be collected from the internal stages to inform on the task.	initial seeds and their mutants, energy assignments to the seeds
Type of data analysis	The operations that needed to be performed on the related data for analysis.	variation of energy between initial seeds and their mutants

[4] https://www.microsoft.com/en-us/microsoft-teams/group-chat-software
[5] https://conceptboard.com/

3.2.2 *Coding protocol.*

The interview data is split evenly between two researchers (coders). The first coder has been working in different projects involving GF and whitebox fuzzing [24] for three years and participated in multiple user studies gaining knowledge in conducting interviews. The second coder has participated in various qualitative data analysis projects in the past two years enriching the expertise in the field. Both the coders heavily work with the well-known fuzzers, AFL [45], AFL++ [8], LibFuzzer [6]. Considering the combined expertise of the coders, we can argue that we are well qualified to conduct the study described in this paper.

The coders analyzed and coded the data individually using the task specification discussed above. After the analysis the coders discussed each other's work to resolve the disagreements. A disagreement is defined as the difference between the coded data in at least one part of the task specification.

This is an adaptation of the gold coding standard [39] where the reliability coder verifies the master coder's work. In our case, each coder acted as the reliability coder for other coder's work. When there are further disagreements, a third coder is also involved to arrive at a consensus. The final list of task specifications are then reviewed by the rest of the research team for approval. To measure the reliability of the coding process, we report the percentage of disagreements between the coders.

3.3 Task Taxonomy

Once we have the agreed list of task specifications, we can further group them based on the duplicate and subset relationships before extracting the taxonomy. We ignored all the agreed non-visualization task specifications and only extracted the task taxonomy from the remaining specifications.

We formally define a task specification, T, as the tuple, *T(C, S, D, A)*, containing four elements: categorization (C), involved stages (S), related data (D), and type of data analysis (A).

$$T = T' \iff C = C' \land S = S' \land D = D' \land A = A' \tag{1}$$

$$T \subset T' \iff C = C' \land S \subset S' \land D \subset D' \land A = A' \tag{2}$$

Two tasks, denoted as $T(C, S, D, A)$ and $T'(C', S', D', A')$ respectively, are considered duplicates when each element in T is same as the corresponding element in T' (Equation 1). T is a subset of T' when the elements S, D are subset of the corresponding elements S', D' and the elements C, A are same as the corresponding elements C', A' (Equation 2).

4 Pilot Study

4.1 Participants

Six out of seven experts we invited agreed to participate in the interviews. All the participants work as research associates with up to three years of experience in fuzzing. We asked each participant to choose an expertise level among beginner, mediocre, and expert in conceptual knowledge and practical knowledge in GF. Participants with conceptual knowledge can provide insights from the perspective of using fuzzers whereas participants with practical knowledge can provide insights from the software-developer perspective. Table 2 shows the expertise profile of the participants.

Table 2: Expertise profile of the participants of the study.

Participant	Conceptual knowledge	Practical knowledge
P1	beginner	beginner
P2	mediocre	beginner
P3	mediocre	mediocre
P4	mediocre	mediocre
P5	mediocre	mediocre
P6	mediocre	beginner

In our pilot study, we only were able to interview participants with up to mediocre level of expertise. The ideal participant should have the highest level of conceptual expertise and practical expertise. However, it is hard to find such ideal participants. In our actual study (Section 5), we aim to interview participants from the fuzzing community with the highest level of expertise either in conceptual knowledge or practical knowledge to get more meaningful insights.

4.2 Greybox Fuzzing Internals

We asked each participant in the pilot semi-structured interview to comment on the proposed internal stages of GF (Figure 1). We grouped the responses into three categories: accepted (completely agree with the categorization), partially accepted (agree with the categorization while providing some suggestions), and rejected (disagree with the categorization).

Among the six participants, two of them provided partial acceptance with suggestions and the rest accepted the proposed internal stages. None of the participants rejected our categorization. Table 3 shows the suggestions from the two partially accepted participants.

Table 3: Participant suggestions on the internal stages of GF.

Participant	Response	Suggestion
P2	partially accepted	*Search strategy* and *Power schedule* should be part of *Mutations*.
P4	partially accepted	A filtering stage between *Execution* and *Search strategy* decides if the seed should be added back to the queue or not.

Discussion. P2's suggestion was based on their experience with the fuzzer, Jazzer[7] (mentioned in the interview). Though the suggestion is to merge the internal stages, we manually verified the implementation of Jazzer, and found that it internally uses LibFuzzer's Entropic power schedule[8] [5] and random search strategy[9] by default. The suggestion from participant P4 is already included in the *Execution* stage according to our categorization. Considering these reasons, we report that all the participants accept the categorization.

[6] https://llvm.org/docs/LibFuzzer.html

[7] https://github.com/CodeIntelligenceTesting/jazzer

[8] https://github.com/llvm/llvm-project/commit/f3c2e0bcee64b0905addaefe9cd0c9ad4d20ac6f

[9] https://github.com/llvm/llvm-project/blob/f68548135b8f9a02beac842646ab89bcaad9d400/compiler-rt/lib/fuzzer/FuzzerLoop.cpp#L721

4.3 Open Coding Protocol

We systematically followed the qualitative data analysis protocol to extract task specifications from the anonymized interview data. In the total of 110 tasks specifications, there were only 16 initial conflicts. All were resolved by discussion between the coders. Hence, the percentage of initial disagreement was 14.5%. We then discarded 12 non-visualization task specifications and 6 task specifications due to insufficient information, and merged two task specifications into one as they are derivatives. Finally, we are left with 91 task specifications to extract the taxonomy.

4.4 Taxonomy

We followed the equations described in Section 3.3 to extract the final taxonomy shown in Tables 4 and 5. After removing 37 duplicates (Equation 1), we are left with 54 unique task specifications. Due to the space constraints, we only show the involved stages and the type of data analysis (analysis task) for each of the 54 task specifications. The anonymized interview data and the complete information on each task specification can be found in our artifact [18].

Table 4 shows the inter-stage tasks and Table 5 shows the intra-stage, and overall tasks. The first column in the tables show the involved GF internal stages (Figure 1) and the second column describes the analysis task. The duplicates identified in the analysis tasks (Equation 1) is shown using the icon, (x ⧉), where 'x' represents the number of duplicates. For example, we have identified five duplicates of Task 22 (see Table 4) and six duplicates of Task 46 (see Table 5). The icon (⊂ T) tells that the current analysis task is a subset (Equation 2) to the analysis task 'T'. For example, Tasks 5 and 6 are subsets of Task 4 (see Table 4).

Though the above mentioned taxonomy is only from the pilot study, it already provides a strong base for fuzzing visualizations and suggests new criteria for fuzzing evaluations. To develop visualizations that focus on understanding fuzzing internals, the framework should at least provide workflows that cover the proposed taxonomy. When developing the workflows, one can identify different internal stage combinations to consider from our taxonomy: according to the Tasks 25 - 31, a visualization workflow for *power schedule* should consider the relationships with *mutations, search strategy* and *execution* internal stages; the internal stages, *instrumentation* and *initial seeds*, may not interact with each other but they interact with all the remaining four internal stages (observed from the Tasks 1 - 20); the internal stage, *mutations*, only interacts with *execution* (Tasks 32 - 39). The example visualization workflow discussed in Section 2.2 relates to the Task 34 and the example visualization analysis task discussed in Section 3.2 relates to the Task 14. Our taxonomy also provides insights in prioritizing the development of workflows. For example, the most frequent analysis task (in terms of duplicates) is to visualize the timeline across different internal stages (Tasks 11, 46, and 53). It accounts for 24.3% (9) of the total duplicates (37).

To evaluate an improvement to one of the fuzzing internal stages, the proposed taxonomy provides the combination of internal stages that should also be considered in the evaluation. For example, the evaluation of an improved power schedule should consider its relationships with *mutations, search strategy* and *execution* internal stages (according to the Tasks 25 - 31). Our taxonomy also hints towards the data that can be observed to evaluate the internal stages and its relationships (the related data corresponding to each analysis task is provided in our artifact [18]).

5 Plan for the Actual Study

To complete the actual study in Phase 2 of the publication process, we plan to expand the research in scope and depth, with the following four goals. First, we will conduct more interviews among fuzzing experts. We aim to approach experts in the academia and industry with the highest level of expertise to gather more meaningful insights. For this reason, we plan to use workshop as one of the platforms to reach out to the experts in the field of GF.

Second, we will extend the scope of the semi-structured interview. The question catalogue used in the pilot study only focuses on extracting the visualization analysis tasks that the experts ask about the fuzzing internals. We plan to extend the catalogue and hence the scope of the interview to gather insights on the experience of using existing visualization tools for fuzzing. Such an extended scope allows us to understand the problems faced by the users when using the tools and contributes to the design and development of the future tools.

Third, after extracting the taxonomy from the actual study, we plan to evaluate the existing visualization tools for fuzzing and discuss their support for the tasks in our taxonomy.

Finally, we plan to evaluate our approach by answering the following research questions:

(1) *RQ1.* What are the different visualization analysis tasks to understand the internal stages of fuzzing?
(2) *RQ2.* To what extent do the state-of-the-art fuzzing visualization tools support the extracted visualization analysis tasks?

6 Threats to Validity

We have identified several threats to validity. First, the internal stages of GF do not consider the contributions of hybrid approaches such as the combinations of GF with symbolic execution [38, 40, 41], static analysis [15, 23], etc. Our proposed internal stages can be considered as the basic stages in most of the fuzzers and the hybrid approaches add additional stages to it. We intentionally scoped the research to internal stages in GF alone for two reasons: visualizing the internal stages of a fuzzer itself is not researched in the community and we hope to make the initial contribution in this area; finding experts working with hybrid approaches is rather difficult when compared to the experts working in GF. However, we plan to consider such extensions to our work in the future.

Second, the interviews are intentionally semi-structured to facilitate open-ended discussions with the fuzzing experts and gather meaningful insights. The taxonomy extracted from such a free-form interview data is not exhaustive and final: it can be extended or refined by conducting more interviews and we are open for additional contributions. To mitigate this threat, we scoped the interview discussions to focus on understanding fuzzing internals using our question catalogue and aim to conduct up to 20 interviews to make the taxonomy robust.

Table 4: Taxonomy of inter-stage visualization analysis tasks from pilot study.

Legend, GF internal stages from Figure 1: I instrumentation, IS initial seeds, S search strategy, P power schedule, M mutations, E execution;
x represents task ID; (x ⧉) represents 'x' duplicates of the analysis task; (⊂ T) represents the current analysis task is subset to the analysis task 'T';
interesting behaviours refer to runtime information such as code coverage, crashes, resource consumption, execution speed.

Stage(s)			Analysis task
I	M	E	1 coverage growth over time for each mutant
			2 overlapping of code coverage of each mutant and instrumented code statements
	S		3 overlapping of code coverage of selected seeds and failed instrumented code; selection frequency of the seeds that execute the failed instrumented code
	E		4 (1 ⧉) comparison of runtime metrics between instrumented and non-instrumented targets
			5 (⊂ 4) comparison of code coverage between instrumented and non-instrumented targets
			6 (⊂ 4) (1 ⧉) comparison of compilation and execution time between instrumented and uninstrumented target
			7 percentage of instrumented code statements that are failed to execute at runtime; errors and location of the runtime failed instrumented code
	S		8 relationship (eg. correlation) between instrumented code blocks and selection orders
	P		9 (2 ⧉) relationship between different instrumentation types and energy assignments
IS	M	E	10 comparison of interesting behaviours between initial seeds and mutants; analyze the parts of initial seeds and mutants that contributed to the interesting behaviours
			11 mutation timeline, number of mutants from each initial seed; code coverage for each initial seed and mutant
	S		12 (2 ⧉) percentage of runtime similarities; selection frequency of runtime similar seeds
			13 (⊂ 22) relationship between selection orders and interesting behaviours of different initial seed sets
	P	M	14 (⊂ 29) distribution of energy for each initial seed and its mutants
	E		15 (⊂ 12) comparison of code coverage between different initial seed sets
			16 (⊂ 12, 15) comparison of code coverage between initial seed set with an empty seed
			17 (⊂ 10) (2 ⧉) impact of initial seeds on interesting behaviours
	M		18 comparison between mutated parts of the initial seeds
	S		19 (⊂ 13) (1 ⧉) relationship (eg. correlation) between selection orders and initial seeds
	P		20 percentage of assigned power used by initial seeds
S	E		21 relationship (eg. correlation) between selection orders and filtered seeds
			22 (5 ⧉) relationship (eg. correlation) between selection orders and interesting behaviours
	M		23 (2 ⧉) relationship (eg. correlation) between selection orders and mutation strategies
	P		24 (3 ⧉) distribution of energy for each selected seed; correlation between selection orders and energy assignments
P	M	E	25 percentage of assigned power used by each selected seed; percentage of mutants provided the coverage gain
	S		26 relationship (eg. correlation) between energy assignments, selection orders and interesting behaviours
	E		27 (3 ⧉) relationship (eg. correlation) between energy assignments and interesting behaviours
			28 relationship (eg. correlation) between energy assignment and filtered seeds
	M		29 variation in the energy across different mutants
			30 (1 ⧉) distribution of energy for each mutant and each mutation strategy
			31 relationship between frequency of mutant and the energy assignment to the mutant; relationship between number of mutants and the energy assignment to the mutants
M	E		32 (2 ⧉) comparison of interesting behaviours across different mutation strategies
			33 relationship (eg. correlation) between mutation strategies and filtered seeds
			34 (1 ⧉) relationship (eg. correlation) between mutation strategies and interesting behaviours
			35 comparison of coverage growth rate across different mutation strategies
			36 comparison of runtime, number of mutants generated, time to generate mutants across different mutation strategies
			37 (⊂ 32) comparison of runtime and resource consumption across different mutation strategies
			38 relationship between mutation strategy changes and interesting behaviours
			39 time to generate valid mutants; number of valid mutants (successful execution)

Table 5: Taxonomy of intra-stage and overall visualization analysis tasks from pilot study.

Legend, GF internal stages from Figure 1: I instrumentation, IS initial seeds, S search strategy, P power schedule, M mutations, E execution, ALL all internal stages; x represents task ID; (x ⧉) represents 'x' duplicates of the analysis task; (⊂ T) represents the current analysis task is subset to the analysis task 'T'; interesting behaviours refer to runtime information such as code coverage, crashes, resource consumption, execution speed.

Stage	Analysis task
I	40 percentage of instrumented code statements that are syntactically correct; the instrumented code which failed in compilation
	41 (1 ⧉) percentage of instrumented code per class/function
	42 number of lines of code are added to the target
IS	43 (1 ⧉) percentage of structural similarities
S	44 (⊂ 31) analyze selection order frequency between seeds
P	45 distribution of energy assignment per seed over time
M	46 (6 ⧉) mutation timeline of each mutant
	47 compare the number of mutants generated under different mutation strategy
	48 record number of mutants generated overall
	49 record what parts of the seed are mutated
E	50 number and duration of executions
	51 executions that improve code coverage over time
ALL	52 (2 ⧉) impact of initial seeds on the other stages
	53 (3 ⧉) timeline of the seeds with respect to all stages
	54 record code coverage and bugs found over time

Third, the robustness of our taxonomy also depends on the number of participants and the expertise they have in GF. We were only able to interview participants who have beginner and mediocre expertise in the pilot study. To mitigate this threat, we hope to interview more participants who have the highest level of expertise in the actual study.

Fourth, the derived tasks are data driven, and subjective to the coders. There is a possibility that different coders might produce different tasks. To mitigate this threat and ensure reliability of our results, we followed a systematic qualitative data analysis methodology (described in Section 3.2). In our pilot study, we only have 16 (14.5%) cases of disagreement which can be considered as a small portion.

7 Related Work

To the best of our knowledge, we are the first to conduct such an interview-based study with fuzzing experts to understand the interested tasks.

7.1 Fuzzing Visualization Tools

There are a few visualization tools for fuzzing that focus on high-level data but not on the internals of fuzzing. VisFuzz provides a user interface to view the call graph and control-flow graph reachability in the browser and provides realtime intervention of the running fuzzing process. It aids the user to understand the bottlenecks in the source code, modify the fuzzing input and construct targeted seeds or modify the test driver to bypass the bottlenecks [48]. A recently published registered report, InFuzz [43], also provides HTML user interface to guide the users in identifying bottlenecks and intervening the fuzzing process to improve runtime code coverage. FMViz provides visualizations to understand the mutation in AFL [45] by exporting byte level changes to the input to an image [16]. FuzzSplore also provides visualizations for the mutations along with code coverage and interesting test cases generated over time [11].

None of the above discussed tools provide an infrastructure to understand the internals of fuzzing. We plan to evaluate their support to the tasks in our taxonomy as described in Section 5.

7.2 Visualization Task Taxonomy

Visualization analysis tasks play an important role in designing a suitable domain specific visualization framework and evaluation of the existing frameworks. However, there is no literature so far that provides information on such tasks to understand fuzzing internals. Here we list the related work from other domains that have similar goals of extracting taxonomy of visualization analysis tasks. Lee et al. developed a taxonomy of tasks to improve the existing graph visualization systems and to evaluate them [19]. The authors reviewed user studies of graph visualization techniques and categorized the graph analysis tasks into four groups: topology-based tasks, attribute-based tasks, browsing tasks, and overview tasks. Ahn et al. provide task taxonomy for temporal network visualization [1] to guide future visualization tools and encourage novel research questions. The authors reviewed 53 existing visualization systems to extract the taxonomy. The taxonomy is refined and evaluated using interviews with 12 network analysis experts.

Murray et al. proposed task taxonomy for the analysis of biological pathway data [30] and is the closest to our methodology. They conducted free-form interviews with seven domain experts and grouped the taxonomy into three categories: attribute tasks, relationship tasks, and modification tasks. The authors also examined the existing visualization techniques that support the tasks and detail the gaps revealed by the task taxonomy.

8 Conclusions

In this paper, we stressed the importance of visualization analysis tasks as a pre-requisite for the development of visualization framework and the need of such tasks to understand the internals of greybox fuzzing (GF). We have identified that there is no systematic research conducted so far in this area and we hope to make the first contribution. We presented our approach of conducting semi-structured interviews with fuzzing experts and performing qualitative data analysis to extract the visualization analysis tasks. Our preliminary results from the pilot study summarizes the different types of these tasks.

We have also presented our plan for the actual study to conduct more interviews and extend the scope and depth of the interview. Through the actual study, we aim to make our results robust and provide richer task taxonomy. We hope that our contributions will shed light towards further research opportunities in fuzzing visualizations and fuzzing evaluations.

In the future, we plan to use the task taxonomy as the base to design and develop a fuzzing-specific visualization framework that focuses on understanding fuzzing internals.

9 Revision Requirements

This camera-ready version of the paper does not incorporate the following revision requirement due to the unavailability of time:

(1) Make sure to have a sufficient set of interviewed people and expertise within your initial user study. We ask you to interview 10 people, 5 of which are experts, to address this concern.

We plan to address this requirement at TOSEM stage 1 submission.

Acknowledgments

This work was partly supported by the Fraunhofer Internal Programs under Grant No. PREPARE 840 231, and NSF grants CCF-2008905 and CCF-2047682.

References

[1] Jae-wook Ahn, Catherine Plaisant, and Ben Shneiderman. 2014. A Task Taxonomy for Network Evolution Analysis. *IEEE Transactions on Visualization and Computer Graphics* 20, 3 (2014), 365–376. https://doi.org/10.1109/TVCG.2013.238

[2] Cornelius Aschermann, Sergej Schumilo, Tim Blazytko, Robert Gawlik, and Thorsten Holz. 2019. REDQUEEN: Fuzzing with Input-to-State Correspondence. In *NDSS*, Vol. 19. 1–15. https://doi.org/10.14722/ndss.2019.23371

[3] Jinsheng Ba, Gregory J. Duck, and Abhik Roychoudhury. 2023. Efficient Greybox Fuzzing to Detect Memory Errors. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 37, 12 pages. https://doi.org/10.1145/3551349.3561161

[4] Marcel Böhme, Cristian Cadar, and Abhik Roychoudhury. 2021. Fuzzing: Challenges and reflections. *IEEE Software* 38, 3 (2021), 79–86. https://doi.org/10.1109/MS.2020.3016773

[5] Marcel Böhme, Valentin JM Manès, and Sang Kil Cha. 2020. Boosting fuzzer efficiency: An information theoretic perspective. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 678–689. https://doi.org/10.1145/3368089.3409748

[6] 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* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1032–1043. https://doi.org/10.1145/2976749.2978428

[7] Matthew Brehmer and Tamara Munzner. 2013. A Multi-Level Typology of Abstract Visualization Tasks. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2376–2385. https://doi.org/10.1109/TVCG.2013.124

[8] 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) (WOOT'20)*. USENIX Association. https://doi.org/10.5555/3488877.3488887

[9] Andrea Fioraldi, Dominik Christian Maier, Dongjia Zhang, and Davide Balzarotti. 2022. LibAFL: A Framework to Build Modular and Reusable Fuzzers. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 1051–1065. https://doi.org/10.1145/3548606.3560602

[10] Andrea Fioraldi, Alessandro Mantovani, Dominik Maier, and Davide Balzarotti. 2023. Dissecting American Fuzzy Lop: A FuzzBench Evaluation. *ACM Trans. Softw. Eng. Methodol.* 32, 2, Article 52 (mar 2023), 26 pages. https://doi.org/10.1145/3580596

[11] Andrea Fioraldi and Luigi Paolo Pileggi. 2021. FuzzSplore: Visualizing Feedback-Driven Fuzzing Techniques. https://doi.org/10.48550/arXiv.2102.02527 arXiv:2102.02527 [cs.CR]

[12] Patrice Godefroid. 2020. Fuzzing: hack, art, and science. *Commun. ACM* 63, 2 (jan 2020), 70–76. https://doi.org/10.1145/3363824

[13] Adrian Herrera, Hendra Gunadi, Shane Magrath, Michael Norrish, Mathias Payer, and Antony L. Hosking. 2021. Seed selection for successful fuzzing. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual, Denmark) *(ISSTA 2021)*. Association for Computing Machinery, New York, NY, USA, 230–243. https://doi.org/10.1145/3460319.3464795

[14] S.E. Hove and B. Anda. 2005. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05)*. 10 pp.–23. https://doi.org/10.1109/METRICS.2005.24

[15] Heqing Huang, Yiyuan Guo, Qingkai Shi, Peisen Yao, Rongxin Wu, and Charles Zhang. 2022. BEACON: Directed Grey-Box Fuzzing with Provable Path Pruning. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 36–50. https://doi.org/10.1109/SP46214.2022.9833751

[16] Aftab Hussain and Mohammad Amin Alipour. 2021. FMViz: Visualizing Tests Generated by AFL at the Byte-level. https://doi.org/10.48550/arXiv.2112.13207 arXiv:2112.13207 [cs.SE]

[17] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. 2018. Evaluating Fuzz Testing. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 2123–2138. https://doi.org/10.1145/3243734.3243804

[18] Sriteja Kummita, Miao Miao, Eric Bodden, and Shiyi Wei. 2024. Pilot Study Artifact. https://github.com/sritejakv/visualization_taxonomy_pilot_study

[19] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. 2006. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization* (Venice, Italy) *(BELIV '06)*. Association for Computing Machinery, New York, NY, USA, 1–5. https://doi.org/10.1145/1168149.1168168

[20] 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* (Montpellier, France) *(ASE '18)*. Association for Computing Machinery, New York, NY, USA, 475–485. https://doi.org/10.1145/3238147.3238176

[21] Jun Li, Bodong Zhao, and Chao Zhang. 2018. Fuzzing: a survey. *Cybersecurity* 1, 1 (Dec. 2018), 6. https://doi.org/10.1186/s42400-018-0002-y

[22] Hongliang Liang, Xiaoxiao Pei, Xiaodong Jia, Wuwei Shen, and Jian Zhang. 2018. Fuzzing: State of the Art. *IEEE Transactions on Reliability* 67, 3 (Sept. 2018), 1199–1218. https://doi.org/10.1109/TR.2018.2834476

[23] Stephan Lipp, Severin Kacianka, Alexander Pretschner, and Marcel Böhme. 2024. SAST-Guided Grey-Box Fuzzing. (2024).

[24] Valentin JM Manès, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J Schwartz, and Maverick Woo. 2021. The Art, Science, and Engineering of Fuzzing: A Survey. *IEEE Transactions on Software Engineering* 47, 11 (2021), 2312–2331. https://doi.org/10.1109/TSE.2019.2946563

[25] Valentin J. M. Manès, Soomin Kim, and Sang Kil Cha. 2020. Ankou: guiding grey-box fuzzing towards combinatorial difference. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1024–1036. https://doi.org/10.1145/3377811.3380421

[26] Lucas McDonald, Muhammad Ijaz Ul Haq, and Ashley Barkworth. 2021. Survey of Software Fuzzing Techniques. (Dec. 2021).

[27] Matthew B Miles and A Michael Huberman. 1994. *Qualitative data analysis: An expanded sourcebook.* sage.

[28] Tamara Munzner. 2009. A Nested Model for Visualization Design and Validation. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (Nov. 2009), 921–928. https://doi.org/10.1109/TVCG.2009.111

[29] Tamara Munzner. 2014. *Visualization analysis and design.* CRC press.

[30] Paul Murray, Fintan McGee, and Angus G Forbes. 2017. A taxonomy of visualization tasks for the analysis of biological pathway data. *BMC bioinformatics* 18 (2017), 1–13. https://doi.org/10.1186/s12859-016-1443-5

[31] Mathias Payer. 2019. The Fuzzing Hype-Train: How Random Testing Triggers Thousands of Crashes. *IEEE Security & Privacy* 17, 1 (Jan. 2019), 78–82. https://doi.org/10.1109/MSEC.2018.2889892

[32] Alexandre Rebert, Sang Kil Cha, Thanassis Avgerinos, Jonathan Foote, David Warren, Gustavo Grieco, and David Brumley. 2014. Optimizing seed selection for fuzzing. In *Proceedings of the 23rd USENIX Conference on Security Symposium* (San Diego, CA) *(SEC'14)*. USENIX Association, USA, 861–875.

[33] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization & Computer Graphics (Proc. InfoVis)* (2017). https://doi.org/10.1109/tvcg.2016.2599030

[34] M. Schloegel, N. Bars, N. Schiller, L. Bernhard, T. Scharnowski, A. Crump, A. Ale-Ebrahim, N. Bissantz, M. Muench, and T. Holz. 2024. SoK: Prudent Evaluation Practices for Fuzzing. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 140–140. https://doi.org/10.1109/SP54263.2024.00137

[35] Hans-Jörg Schulz, Thomas Nocke, Magnus Heitzler, and Heidrun Schumann. 2013. A Design Space of Visualization Tasks. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2366–2375. https://doi.org/10.1109/TVCG.2013.120

[36] C.B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering* 25, 4 (1999), 557–572. https://doi.org/10.1109/32.799955

[37] Michael Sedlmair, Miriah Meyer, and Tamara Munzner. 2012. Design Study Methodology: Reflections from the Trenches and the Stacks. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (2012), 2431–2440. https://doi.org/10.1109/TVCG.2012.213

[38] Nick Stephens, John Grosen, Christopher Salls, Andrew Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. 2016. Driller: Augmenting fuzzing through selective symbolic execution.. In *NDSS*, Vol. 16. 1–16.

[39] Moin Syed and Sarah C. Nelson. 2015. Guidelines for Establishing Reliability When Coding Narrative Data. *Emerging Adulthood* 3, 6 (2015), 375–387. https://doi.org/10.1177/2167696815587648

[40] Yaëlle Vinçont, Sébastien Bardin, and Michaël Marcozzi. 2022. A Tight Integration of Symbolic Execution and Fuzzing (Short Paper). In *International Symposium on Foundations and Practice of Security*. Springer, 303–310. https://doi.org/10.1007/978-3-031-08147-7_20

[41] Mingzhe Wang, Jie Liang, Yuanliang Chen, Yu Jiang, Xun Jiao, Han Liu, Xibin Zhao, and Jiaguang Sun. 2018. SAFL: increasing and accelerating testing coverage with symbolic execution and guided fuzzing. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 61–64. https://doi.org/10.1145/3183440.3183494

[42] Wei-Cheng Wu, Bernard Nongpoh, Marwan Nour, Michaël Marcozzi, Sébastien Bardin, and Christophe Hauser. 2024. Fine-grained Coverage-based Fuzzing. *ACM Trans. Softw. Eng. Methodol.* 33, 5, Article 138 (jun 2024), 41 pages. https://doi.org/10.1145/3587158

[43] Qian Yan, Huayang Cao, Shuaibing Lu, and Minhuan Huang. 2023. InFuzz: An Interactive Tool for Enhancing Efficiency in Fuzzing through Visual Bottleneck Analysis (Registered Report). In *Proceedings of the 2nd International Fuzzing Workshop (FUZZING 2023)*. Association for Computing Machinery, New York, NY, USA, 56–61. https://doi.org/10.1145/3605157.3605847

[44] Tai Yue, Pengfei Wang, Yong Tang, Enze Wang, Bo Yu, Kai Lu, and Xu Zhou. 2020. EcoFuzz: Adaptive Energy-Saving Greybox Fuzzing as a Variant of the Adversarial Multi-Armed Bandit. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2307–2324.

[45] Michal Zalewski. 2013. *American fuzzy lop.* https://github.com/mirrorer/afl

[46] Xiaoqi Zhao, Haipeng Qu, Jianliang Xu, Xiaohui Li, Wenjie Lv, and Gai-Ge Wang. 2024. A systematic review of fuzzing. *Soft Computing* 28, 6 (2024), 5493–5522. https://doi.org/10.1007/s00500-023-09306-2

[47] 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)*. USENIX Association, Anaheim, CA, 1343–1360.

[48] Chijin Zhou, Mingzhe Wang, Jie Liang, Zhe Liu, Chengnian Sun, and Yu Jiang. 2019. VisFuzz: Understanding and Intervening Fuzzing with Interactive Visualization. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1078–1081. https://doi.org/10.1109/ASE.2019.00106

Received 2024-06-21; accepted 2024-07-22