A Preliminary Taxonomy of Techniques Used in Software Fuzzing

Raunak Shakya
Tennessee Technological University
Cookeville, Tennessee
rshakya@students.tntech.edu

Akond Rahman
Tennessee Technological University
Cookeville, Tennessee
arahman@tntech.edu

CCS CONCEPTS

• **Security and privacy → Software security engineering**.

KEYWORDS

fuzzing, scoping review, software security, taxonomy

ACM Reference Format:
Raunak Shakya and Akond Rahman. 2020. A Preliminary Taxonomy of Techniques Used in Software Fuzzing. In *Hot Topics in the Science of Security Symposium (HotSoS '20), April 7–8, 2020, Lawrence, KS, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3384217.3384219

1 INTRODUCTION

Software fuzzing is a testing technique, which generates erroneous and random input to a software so that the software of interest can be monitored for exceptions such as crashes [1]. Both in the open source software (OSS) and proprietary domain, fuzzing has been widely used to explore software vulnerabilities. For example, information technology (IT) organizations such as Google [1] and Microsoft [2] use software fuzzing as part of the software development process. As of Jan 2019, GitHub hosts 2,915 OSS repositories related to fuzzing [3].

Despite the popularity of fuzzing in software industry, practitioners might face challenges in implementing fuzzing due to lack of understanding of what techniques are used to implement software fuzzing. Such limitations can hinder discovery of vulnerabilities in under-explored areas, such as scientific software [6] and microservices [7]. A taxonomy of software fuzzing techniques can be helpful for researchers who are relatively new to the domain of software fuzzing and want to apply software fuzzing.

The goal of the paper is to help researchers in performing software fuzzing by categorizing techniques that are used in software fuzzing literature.

We answer the following research questions:

- ***RQ1**: What techniques are used to perform software fuzzing?*
- ***RQ2**: How frequently do the identified software fuzzing techniques appear in software fuzzing literature?*

[1]https://chromium.googlesource.com/chromium/src/+/master/testing/libfuzzer/README.md
[2]https://www.microsoft.com/en-us/security-risk-detection/success-stories/
[3]https://github.com/search?q=fuzzing&type=Repositories

Our contribution is a taxonomy of techniques reported in prior literature related to software fuzzing.

2 METHODOLOGY

In this section we describe our methodology.

2.1 RQ1: Techniques used in Fuzzing

We conduct a scoping review of publications related to software fuzzing to derive our taxonomy. Using a scoping review researchers can synthesize results using a limited search [2]. According to Munn et al. "*Researchers may conduct scoping reviews instead of systematic reviews where the purpose of the review is to identify knowledge gaps, scope a body of literature, clarify concepts or to investigate research conduct.*". Unlike a systematic literature review, a scoping review is less comprehensive, and can be used as a precursor to conduct a systematic literature review. Scoping review can be useful to collect emerging evidence, which eventually can be used to inform further research decisions [2]. For example, if a researcher is inexperienced in the domain of software fuzzing, and wants to get an understanding of existing topics such as practices and techniques to implement fuzzing, then a scoping review could be useful to that researcher of interest.

We conduct a scoping review by identifying established venues where software fuzzing-related literature is published. We select four conferences: International Conference on Software Engineering (ICSE), Symposium on Foundations of Software Engineering (FSE), Computer and Communications Security (CCS), and USENIX Security Symposium (USENIX). We select these conferences because (i) these conferences are rated as 'flagship conference' by Computing Research & Education (CORE) [4], and (ii) these conferences publish literature related to software fuzzing. Along with ICSE, FSE, CCS, and USENIX, we also include conferences that focus on testing namely, International Symposium on Software Testing and Analysis (ISSTA) and International Conference on Software Testing, Verification and Validation (ICST), as software fuzzing is a sub-category of software testing [1]. We select conferences as they tend to have a shorter review cycle and are more likely to include recent advances in the field of interest [10]. We conduct the review by applying the following steps:

- *Step-1*: We download all papers from 2009 to 2019 for each of the four conferences.
- *Step-2*: We read the title, abstract, and keywords to determine if the downloaded papers are related to fuzzing.
- *Step-3*: Upon completion of Step-2, one rater reads each collected paper, and identifies topics discussed in the paper of interest

[4]http://www.core.edu.au/

using qualitative analysis. For each paper the rater derives a topic by identifying what technique is used by the authors to conduct fuzzing for the software of interest.

- *Step-4*: Similar to Rahman and Williams [8], the rater uses card sorting, a qualitative analysis technique to identify high-level categories from the identified topics in Step-3. Card sorting is a qualitative analysis technique to identify categories from natural language text [11].

Upon completion of the above-mentioned steps, we derive a list of categories that synthesizes the fuzzing-related topics discussed in various papers published in the four conferences of interest.

2.2 RQ2: Frequency of Identified Techniques

We answer RQ2 using the metric 'Publication (x)'. The metric 'Publication (x)' quantifies the proportion of publications that use technique x. We calculate 'Publication (x)' using by calculating the proportion of publications that use technique x.

3 RESULTS

We present the results of our research in this section.

3.1 Answer to RQ1

Following our methodology we collect 2,061 publications. By applying our filtering criteria (Step-2 in Section 2) we obtain 48 publications related to software fuzzing. Finally, using qualitative analysis (Step-3 in Section 2) and card sorting (Step-4 in Section 2) we identify five topics, which we discuss below:

- **Feature mining**: This category related to software fuzzing includes publications that mine features to conduct software fuzzing. We observe researchers mine features from software source code, software specification, operating system kernel calls, memory access logs, protocol communication traffic, CPU usage logs, and program state executions.
- **Symbolic execution**: This category includes publications that use symbolic execution to conduct software fuzzing. Symbolic execution is the practice of generating test cases where instead of real values, artificial values are used for test case generation [5]. In this manner, data is replaced by symbolic values with expression sets.
- **Search-based algorithms**: This category includes publications that use search-based algorithms to conduct software fuzzing. Search-based algorithms are artificial intelligence techniques that use stochastic techniques to explore multi-dimensional search spaces [4].
- **Formal methods**: This category includes publications that use formal methods to perform software fuzzing. Formal methods are system design techniques where testing techniques such as fuzzing are specified as mathematical equations [3].
- **Taint analysis**: This category includes publications that use taint analysis to conduct software fuzzing. Taint analysis is a technique that checks which variables can be modified by the user input [9].

Table 1: Answer to RQ2: Frequency of Identified Topics

Topic	Publication (%)
Feature mining	45.8
Symbolic execution	14.5
Search-based algorithms	14.5
Formal methods	16.7
Taint analysis	14.5

3.2 Answer to RQ2

We observe feature mining to be the most frequently used technique. We provide the values for the 'Publication' metric in Table 1 for each topic. A complete mapping of each of the 48 publications with each identified topic is available online [5].

4 CONCLUSION

A review of existing literature related to software fuzzing can help identify topics and inform researchers on what techniques can be investigated and applied to identify vulnerabilities in unexplored domains such as scientific software. We conduct a scoping review with 48 publications published in well-known academic venues such as ICSE and CCS. We derive five techniques namely, feature mining, symbolic execution, search-based algorithms, formal methods, and taint analysis. We observe the most frequent technique to be feature mining. Our taxonomy might be helpful for researchers in two ways: (i) researchers can use our taxonomy to assess what techniques can be used to identify undiscovered software vulnerabilities in under-explored domains such as scientific software [6], and (ii) derive a taxonomy that is comprehensive. We hope our publication will garner further interest in the domain of software fuzzing.

REFERENCES

[1] Paul Ammann and Jeff Offutt. 2016. *Introduction to software testing*. Cambridge University Press.
[2] Stuart Anderson, Pauline Allen, Stephen Peckham, and Nick Goodwin. 2008. Asking the right questions: scoping studies in the commissioning of research on the organisation and delivery of health services. *Health research policy and systems* 6, 1 (2008), 7.
[3] Edmund M Clarke and Jeannette M Wing. 1996. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)* 28, 4 (1996), 626–643.
[4] Mark Harman. 2007. The current state and future of search based software engineering. In *2007 Future of Software Engineering*. IEEE Computer Society, 342–357.
[5] James C King. 1976. Symbolic execution and program testing. *Commun. ACM* 19, 7 (1976), 385–394.
[6] E. S. Mesh and J. S. Hawker. 2013. Scientific software process improvement decisions: A proposed research strategy. In *2013 5th International Workshop on Software Engineering for Computational Science and Engineering (SE-CSE)*. 32–39. https://doi.org/10.1109/SECSE.2013.6615097
[7] Sam Newman. 2015. *Building microservices: designing fine-grained systems.* " O'Reilly Media, Inc.".
[8] Akond Rahman and Laurie Williams. 2019. A Bird's Eye View of Knowledge Needs Related to Penetration Testing *(HotSoS '19)*. Association for Computing Machinery, New York, NY, USA, Article Article 9, 2 pages. https://doi.org/10.1145/3314058.3317294
[9] E. J. Schwartz, T. Avgerinos, and D. Brumley. 2010. All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask). In *2010 IEEE Symposium on Security and Privacy*. 317–331. https://doi.org/10.1109/SP.2010.26
[10] Moshe Y Vardi. 2009. Conferences vs. journals in computing research. *Commun. ACM* 52, 5 (2009), 5–5.
[11] T. Zimmermann. 2016. Card-sorting: From text to themes. In *Perspectives on Data Science for Software Engineering*, Tim Menzies, Laurie Williams, and Thomas Zimmermann (Eds.). Morgan Kaufmann, Boston, 137 – 141.

[5]http://tiny.cc/hotsos20-fuzz