Fuzz Testing

Jaclina-Iana Bulat, Ovidiu Calota, Leonardo-Iulian Ciurcau

May 12, 2024

1 Planning Phase

1.1 Motivation

The motivation to conduct this systematic literature review (SLR) stems from the observation of significant variances and potential inconsistencies in the methodologies applied in current research within the field of fuzz testing evaluations. Despite the critical role of fuzz testing in software security, there remains a lack of consensus on standard evaluation methodologies that accurately reflect the effectiveness of fuzz testing techniques. This SLR aims to synthesize existing research to identify common evaluation practices, pinpoint prevailing gaps, and suggest improvements to ensure more reliable and standardized methods for assessing fuzz testing tools.

1.2 Research Questions

To guide the systematic literature review, we propose the following research questions:

- RQ1: What methodologies are commonly employed in the evaluation of fuzz testing algorithms in existing literature?
- RQ2: How do these methodologies address the inherent randomness in fuzz testing results?
- RQ3: What are the recommended practices for improving the robustness and reliability of fuzz testing evaluations as suggested by current research?

2 Conducting the SLR

2.1 Rules

The rules for conducting this SLR are developed to ensure a comprehensive, unbiased, and replicable review process. These include:

- Inclusion and Exclusion Criteria: Studies will be selected based on their focus on fuzz testing methodologies, evaluation of fuzz testing tools, and publication in peer-reviewed journals or conferences between 2010 and 2024. Studies that do not explicitly address fuzz testing evaluation methodologies will be excluded.
- Data Sources: Relevant literature will be sourced from databases such as IEEE Xplore, ACM Digital Library, ScienceDirect, and Scopus.
- Data Extraction and Synthesis: Key information such as study objectives, methodologies, outcomes, and recommendations will be systematically extracted using a standardized data extraction form. Data will be synthesized to compare methodologies and aggregate recommendations.
- Quality Assessment: The quality of included studies will be assessed using a predefined checklist that considers aspects such as methodological rigor, clarity of reporting, and relevance to the research questions.

- Handling Discrepancies: Any discrepancies in study selection, data extraction, or quality assessment will be resolved through discussion or consultation with a third-party expert if necessary. By adhering to these rules, the SLR aims to provide a thorough and authoritative overview of the current state of fuzz testing evaluations, thereby contributing to the enhancement of research methodologies in this vital area of software security.

2.2 Summarizing table

After conducting our research based on the mentioned rules, we end up with 10 research papers.

Id	Citation	Title	Year
1	[KRC+18]	Evaluating fuzz testing	2018
2	[LWC+18]	Fuzz testing in practice: Obstacles and solutions	2018
3	[Pak12]	Hybrid fuzz testing: Discovering software bugs via fuzzing and symbolic execution	2012
4	[GLM+08]	Automated whitebox fuzz testing	2008
5	[XMJX+19]	Deephunter: a coverage-guided fuzz testing framework for deep neural networks	2019
6	[LS18]	Fairfuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage	2018
7	[TDB12]	SECFUZZ: Fuzz-testing security protocols	2012
8	[GJZ+18]	Dlfuzz: Differential fuzzing testing of deep learning systems	2018
9	[BGM13]	Billions and billions of constraints: Whitebox fuzz testing in production	2013
10	[LKK+18]	RFUZZ: Coverage-directed fuzz testing of RTL on FPGAs	2018

Table 1: Summarizing table

3 Resumed papers

1. The paper "RFUZZ: Coverage-Directed Fuzz Testing of RTL on FPGAs" [LKK+18] introduces RFUZZ, a novel tool that applies coverage-guided mutational fuzz testing to RTL circuits, enhancing pre-silicon testing efficiency through FPGA-accelerated simulation. This approach minimizes setup requirements and utilizes FPGA speed to provide quick feedback and high test execution rates, making it significantly effective for verifying a range of RTL designs, from communication IPs to complex CPUs. By defining adaptable test inputs, coverage metrics, and including quick reset transformations for deterministic tests, RFUZZ demonstrates notable improvements in coverage across various designs. The tool is offered as open-source software to promote wider adoption and further development in RTL design verification.

2. The paper "Billions and Billions of Constraints: Whitebox Fuzz Testing in Production" [BGM13] discusses the extensive implementation of constraint-based whitebox fuzz testing across numerous large Windows applications at Microsoft. It focuses on the development period from 2007 to 2012, highlighting the deployment of whitebox fuzzing techniques that use symbolic execution and constraint solving to create new program inputs. These inputs aim to expose new execution paths and potential security vulnerabilities. Notably, this approach identified a significant portion of the file fuzzing bugs found during the development of Windows 7, thereby preventing costly security risks. The paper also introduces SAGAN and JobCenter, systems designed to manage and monitor the deployment of whitebox fuzzing across virtual machines, enhancing the scalability and effectiveness of testing processes. Through the analysis of over 3.4 billion constraints, the paper showcases the application and challenges of implementing whitebox fuzzing on a large scale, demonstrating its crucial role in software development and security testing at Microsoft.

3. The paper "DLFuzz: Differential Fuzzing Testing of Deep Learning Systems" [GJZ+18] presents DLFuzz, the first differential fuzzing testing framework aimed at improving the robustness of Deep Learning (DL) systems by optimizing neuron coverage and generating adversarial inputs. Unlike existing methods that require cross-referencing DL systems or extensive manual labeling, DLFuzz iteratively mutates inputs to maximize coverage and reveal incorrect behaviors autonomously. The framework is demonstrated to be more efficient than the state-of-the-art, DeepXplore, in generating adversarial examples with minimal perturbations and achieving higher

neuron coverage without additional system references. It introduces strategies to select neurons that could trigger more logic paths and potentially incorrect outputs, improving the testing process significantly. The paper's experiments on two popular datasets, MNIST and ImageNet, show that DLFuzz can generate a substantially higher number of adversarial inputs with smaller perturbations and better coverage, enhancing the testing of safety-critical DL applications like autonomous driving and malware detection.

4. The paper "SecFuzz: Fuzz-testing Security Protocols" [TDB12] introduces a lightweight, yet effective modular technique for fuzz-testing security protocols that can handle encrypted traffic. The method involves using actual protocol implementations to generate valid inputs, which are then mutated using a set of fuzz operators while a dynamic memory analysis tool monitors execution to detect vulnerabilities. Their technique also enables the fuzzer to access necessary cryptographic elements to effectively mutate encrypted messages. A case study on the Internet Key Exchange (IKE) protocol reveals new vulnerabilities in mature implementations, demonstrating the practical efficacy of their approach compared to traditional model-based fuzzing tools. This method shows promise in enhancing the robustness of security protocol testing without relying on a system's model or source code.

5. The paper "FairFuzz: A Targeted Mutation Strategy for Increasing Greybox Fuzz Testing Coverage" [LS18] discusses an enhancement to the fuzz testing tool American Fuzzy Lop (AFL). They developed a technique called FairFuzz, which improves code coverage in fuzz testing by focusing on rarely executed branches. This targeted mutation strategy identifies these "rare branches" during testing and uses a novel mutation mask algorithm that selectively mutates parts of inputs that influence hitting these branches, leaving other parts of the input intact. The evaluation showed that FairFuzz achieves higher branch coverage faster and maintains coverage over extended periods compared to standard AFL, particularly in programs with complex conditional structures. This technique demonstrates a substantial improvement in testing efficiency without requiring additional instrumentation or complex setup, maintaining AFL's ease of use.

6. The paper "DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks" [XMJX+19] discusses DeepHunter, an innovative fuzz testing framework designed for deep neural networks (DNNs). The authors introduce a metamorphic mutation strategy that generates new, semantically preserved test inputs and utilizes multiple coverage criteria to direct the test generation process. DeepHunter integrates five testing criteria and four seed selection strategies, demonstrating that its metamorphic mutation strategy effectively produces valid new tests with up to a 98% validity ratio. The results show that DeepHunter outperforms existing tools by significantly increasing coverage and the quantity and diversity of detected defects. This is particularly effective during the challenging DNN quantization process for platform migration, proving DeepHunter's utility in enhancing DNN testing and robustness.

7. The paper "Hybrid Fuzz Testing: Discovering Software Bugs via Fuzzing and Symbolic Execution" [Pak12] outlines an innovative approach to software testing that combines fuzz testing and symbolic execution to enhance bug detection efficiency. This method, termed hybrid fuzz testing, initially uses symbolic execution to identify unique program paths and then employs fuzz testing to explore these paths with preconditioned random inputs. The research illustrates how this technique efficiently covers more code and discovers bugs quicker than either method used independently. Through a detailed evaluation, the approach is shown to achieve deeper and broader code coverage, highlighting its potential to significantly improve software testing practices.

8. The paper "Automated Whitebox Fuzz Testing" [GLM+08] focuses on an advanced fuzz testing method that integrates symbolic execution with traditional fuzzing techniques to create a more targeted and effective testing strategy, known as whitebox fuzz testing. This method systematically generates test inputs that dynamically explore and validate more execution paths within a program, significantly enhancing bug detection capabilities. It leverages symbolic execution to identify constraints within code branches, then alters these constraints to force the execution down different paths, increasing coverage and finding subtle bugs that traditional fuzzing methods might miss. The paper presents results showing the successful application of this approach

in detecting bugs in complex software systems, emphasizing its potential to improve software security and reliability through more exhaustive testing.

9. The paper "Fuzz Testing in Practice: Obstacles and Solutions" [LWC+18] discusses the application of fuzz testing in a real-world industrial context, specifically on a proprietary message middleware named libmsg at Huawei. Despite fuzz testing's proven effectiveness in identifying vulnerabilities, the authors encounter several challenges when implementing it, such as system configuration inconsistency, build complexity, and the absence of suitable fuzzing drivers. The paper provides practical solutions to these challenges, such as using existing code snippets to create fuzzing drivers, which lowers costs and technical barriers. By adapting fuzz testing to libmsg, the researchers effectively identify nine previously unknown vulnerabilities, demonstrating the practical challenges and high potential of fuzz testing in industry. This case study not only highlights the effectiveness of fuzz testing but also the need for tailored approaches when applied to complex industrial software systems.

10. The paper titled "Evaluating Fuzz Testing" [KRC+18] and colleagues critically examines how fuzz testing algorithms are evaluated, identifying common pitfalls and proposing guidelines to improve the robustness of these evaluations. It highlights that while fuzz testing has been highly effective in finding critical security bugs, the methodologies for evaluating new fuzzing techniques often suffer from significant shortcomings, such as lack of consistency and rigor in experimental setup and failure to adequately address the randomness inherent in fuzz testing results.

4 Results

- RQ1: What methodologies are commonly employed in the evaluation of fuzz testing algorithms in existing literature?
  RA1: Common methodologies employed in the evaluation of fuzz testing algorithms include coverage-guided mutational fuzz testing, constraint-based whitebox fuzz testing, differential fuzzing testing, and hybrid fuzz testing. These approaches often utilize techniques such as symbolic execution, metamorphic mutation strategies, and dynamic memory analysis to generate test inputs, explore program paths, and detect vulnerabilities across various domains, including hardware design verification, security protocol testing, and deep learning system validation.

- RQ2: How do these methodologies address the inherent randomness in fuzz testing results?
  RA2: To address the inherent randomness in fuzz testing results, methodologies often incorporate coverage metrics to measure the effectiveness of test inputs in exploring program behaviors. Additionally, techniques like symbolic execution and constraint solving help identify constraints and guide the generation of inputs toward specific program paths, increasing the likelihood of revealing vulnerabilities. Some methodologies also use adaptive strategies to iteratively mutate inputs, maximizing coverage and improving the chances of detecting bugs with minimal perturbations.

- RQ3: What are the recommended practices for improving the robustness and reliability of fuzz testing evaluations as suggested by current research?
  RA3: Recommended practices for improving the robustness and reliability of fuzz testing evaluations include ensuring consistency and rigor in experimental setups, incorporating diverse and representative test cases, and addressing the randomness in testing results through statistical analysis and multiple runs. Researchers also advocate for the development of open-source fuzzing tools to promote transparency, reproducibility, and collaboration within the research community. Additionally, the adoption of targeted mutation strategies, adaptive input generation techniques, and integration with symbolic execution or dynamic analysis can enhance the efficiency and effectiveness of fuzz testing evaluations, ultimately leading to more reliable software and system verification processes.

References

[BGM13] Ella Bounimova, Patrice Godefroid, and David Molnar. Billions and billions of constraints: Whitebox fuzz testing in production. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 122–131. IEEE, 2013.

[GJZ+18] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. Dlfuzz: Differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 739–743, 2018.

[GLM+08] Patrice Godefroid, Michael Y Levin, David A Molnar, et al. Automated whitebox fuzz testing. In *NDSS*, volume 8, pages 151–166, 2008.

[KRC+18] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. Evaluating fuzz testing. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 2123–2138, 2018.

[LKK+18] Kevin Laeufer, Jack Koenig, Donggyu Kim, Jonathan Bachrach, and Koushik Sen. Rfuzz: Coverage-directed fuzz testing of rtl on fpgas. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[LS18] 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.

[LWC+18] Jie Liang, Mingzhe Wang, Yuanliang Chen, Yu Jiang, and Renwei Zhang. Fuzz testing in practice: Obstacles and solutions. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 562–566. IEEE, 2018.

[Pak12] Brian S Pak. Hybrid fuzz testing: Discovering software bugs via fuzzing and symbolic execution. *School of Computer Science Carnegie Mellon University*, 2012.

[TDB12] Petar Tsankov, Mohammad Torabi Dashti, and David Basin. Secfuzz: Fuzz-testing security protocols. In *2012 7th International Workshop on Automation of Software Test (AST)*, pages 1–7. IEEE, 2012.

[XMJX+19] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. Deephunter: a coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*, pages 146–157, 2019.