The History of Symbolic Execution in Fuzzing

Valentin Huber

November 22, 2023

Abstract

Contents

1 Introduction

- "Today, testing is the primary way to check the correctness of software. Billions of dollars are spent on testing in the software industry, as testing usually accounts for about 50% of the cost of software development. It was recently estimated that software failures currently cost the US economy alone about $60 billion every year, and that improvements in software testing infrastructure might save one-third of this cost."[1]
- "There are many different dynamic analyses that can be described as "fuzzing." A unifying feature of fuzzers is that they operate on, and produce, concrete inputs. Otherwise, fuzzers

might be instantiated with many different design choices and many different parameter settings."[2]

2 Theoretical Principles

- Two approaches: Random mutation as described in *An Empirical Study of the Reliability of UNIX Utilities* by Miller et al.[3] and pure symbolic execution.
- The latter is infeasible for large programs and for any program that interacts with the environment, the former in its purest form is not very effective.
  - "A key disadvantage of classical symbolic execution is that it cannot generate an input if the symbolic path constraint along a feasible execution path contains formulas that cannot be (efficiently) solved by a constraint solver (for example, nonlinear constraints)."[4]
  - "The blackbox and whitebox strategies achieved similar results in all categories. This shows that, when testing applications with highly-structured inputs in a limited amount of time (2 hours), whitebox fuzzing, with the power of symbolic execution, does not improve much over simple blackbox fuzzing. In fact, in the code generator, those grammarless strategies do not improve coverage much above the initial set of seed inputs."[5]
- Generally:
  - "The process begins by choosing a corpus of "seed" inputs with which to test the target program. The fuzzer then repeatedly mutates these inputs and evaluates the program under test. If the result produces "interesting" behavior, the fuzzer keeps the mutated input for future use and records what was observed. Eventually the fuzzer stops, either due to reaching a particular goal (e.g., finding a certain sort of bug) or reaching a timeout."[2]
  - "Different fuzzers record different observations when running the program under test. In a "black box" fuzzer, a single observation is made: whether the program crashed. In "gray box" fuzzing, observations also consist of intermediate information about the execution, for example, the branches taken during execution as determined by pairs of basic block identifiers executed directly in sequence. "White box" fuzzers can make observations and modifications by exploiting the semantics of application source (or binary) code, possibly involving sophisticated reasoning. Gathering additional observations adds overhead. Different fuzzers make different choices, hoping to trade higher overhead for better bug-finding effectiveness."[2]
  - "In any of these cases, the output from the fuzzer is some concrete input(s) and configurations that can be used from outside of the fuzzer to reproduce the observation. This allows software developers to confirm, reproduce, and debug issues."[2]
- With time and in many projects, parts of one approach has been introduced to the other to use where one excels to aid where the other has drawbacks.
- Further approaches have also been introduced into the mix.

2.1 Problems and Mitigation

Heavily based on [4].

2.1.1 Impossible Constraints

- "A key disadvantage of classical symbolic execution is that it cannot generate an input if the symbolic path constraint along a feasible execution path contains formulas that cannot be (efficiently) solved by a constraint solver (for example, nonlinear constraints)."[4]
- Two techniques that alleviate this problem:
  - Concolic Testing (like DART[1]): Run concrete and symbolic execution at the same time, keep mapping between values, solve path constraint with one sub-constraint flipped to get input for an other path.
  - Execution-Generated Testing (EGT) (like EXE[6] and KLEE[7]): Only execute symbolically if any operands are symbolic
- These handle imprecision in symbex (like interaction with outside code (that is not instrumented for symbex), constraint solving timeouts, unhandled instructions (floating point), or system calls) by just using concrete values.
  - If none of the operands are symbolically, just use them
  - If any are, use the concrete values (direct in concolic, solution from path constraint in EGT)
- Downside: missing some feasible paths, and therefore sacrificing completeness.

2.1.2 Path Explosion

Path Explosion: program path count usually exponential in the number of static branches in the code.

- Symbex helps by only looking at possible branches. Example: EXE[6] on `tcpdump`: only 42% of instructions contained symbolic operands, less than 20% of of symbolic branches have both sides feasible[6]
- Prioritization of which path to explore next using heuristics (like statement or branch coverage (and using static analysis to guide), favouring statements that were run the fewest number of times, or random)
- Interleave random and symbolic execution
- Pruning redundant paths ("if a program path reaches the same program point with the same symbolic constraints as a previously explored path, then this path will continue to execute exactly the same from that point on and thus can be discarded."[8])
- Lazy test generation
- Static path merging

2.1.3 Constraint Solving

Dominates runtime

- Irrelevant constraint elimination: Generally, we go from a solvable constraint set (namely the current execution with the solution being the current concrete values) to one where only one constraint changes (the one we flipped). Typically, major major parts of the constraint set are not influenced by the change and can be excluded from what is passed to the solver. We can then just use the values from the previous iteration.
- Incremental solving: Reuse the results of previous similar queries, because subsets of the constraints are still solved by the same results and supersets often do not invalidate existing solutions.

2.1.4 Memory Modelling

Things like modelling `ints` as mathematical integers being imprecise since it ignores over-/underflows, and pointers being hard to deal with. "On the one end of the spectrum is a system like DART[1] that only reasons about concrete pointers, or systems like CUTE[9] and CREST[10] that support only equality and inequality constraints for pointers, which can be efficiently solved.35 At the other end are systems like EXE[6], and more recently KLEE[7] and SAGE[11] that model pointers using the theory of arrays with selections and updates implemented by solvers like STP or Z3."[4]

2.1.5 Handling Concurrency

Testing usually difficult because of the inherent non-determinism. "Concolic testing was successfully combined with a variant of partial order reduction to test concurrent programs effectively."[4]

3 Improvements in Papers

Filter for symbex

- Initial seed selection: up-front analysis[12]–[14], grammar[15], [16]
- Mutation: symbex to choose how many bits to flip[17], taint analysis[18]–[21], dynamic slicing[22], seed properties[23], grammars[24], [25], language constructs knowledge[26]
- Eval: symbex when stuck[18], [27], general symbex[28], speedup through OS optimizations[29] or other low-level primitives[25], [30], [31], removing checks[32], fine-grained runtime analysis[33]
- Observation: longer running time[34], different behavior[35], additional instrumentation[19], [20], static analysis-guided searching[21], [36]
- Seed selection: areas of interest reached[21], [37]–[39], different algorithm[40], [41]

4 Related Works and Methods

- Large scientific body of work
- Review papers
  - Well cited
  - New
  - Specific topics to see if challenges and solutions differ

4.1 Survey Papers

4.1.1 General Survey Papers

- Symbolic execution for software testing in practice: preliminary assessment (2011)[42]
- Fuzzing: The State of the Art (2012)[43]
- Symbolic execution for software testing: three decades later (2013)[4]
- A systematic review of fuzzing techniques (2018)[44]
- Fuzzing: A Survey (2018)[45]
- Fuzzing: State of the Art (2018)[46]
- Evaluating Fuzz Testing (2018)[2]
- Fuzzing: hack, art, and science (2020)[47]
- A Systematic Review of Search Strategies in Dynamic Symbolic Execution (2020)[48]
- Fuzzing: Challenges and Reflections (2021)[49]
- Exploratory Review of Hybrid Fuzzing for Automated Vulnerability Detection (2021)[50]
- The Art, Science, and Engineering of Fuzzing: A Survey (2021)[51]
- Fuzzing: A Survey for Roadmap (2022)[52]
- Fuzzing vulnerability discovery techniques: Survey, challenges and future directions (2022)[53]
- Demystify the Fuzzing Methods: A Comprehensive Survey (2023)[54]
- A systematic review of fuzzing (2023)[55]

4.1.2 Specific Survey Papers

- Network protocol fuzz testing for information systems and applications: a survey and taxonomy (2016)[56]
- A Survey of Dynamic Analysis and Test Generation for JavaScript (2017)[57]
- Fuzzing the Internet of Things: A Review on the Techniques and Challenges for Efficient Vulnerability Discovery in Embedded Systems (2021)[58]
- Firmware Fuzzing: The State of the Art (2021)[59]
- Research on Fuzzing Technology for JavaScript Engines (2021)[60]
- Ethereum Smart Contract Analysis Tools: A Systematic Review (2022)[61]
- A Survey on the Development of Network Protocol Fuzzing Techniques (2023)[62]
- Embedded fuzzing: a review of challenges, tools, and solutions (2022)[63]
- Fuzzing of Embedded Systems: A Survey (2022)[64]

5 Conclusions

5.1 Future Work

References

[1] P. Godefroid, N. Klarlund, and K. Sen, "Dart: Directed automated random testing," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '05, Chicago, IL, USA: Association for Computing Machinery, 2005, pp. 213–223, ISBN: 1595930566. DOI: `10.1145/1065010.1065036`. [Online]. Available: `https://doi.org/10.1145/1065010.1065036`.

[2] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, "Evaluating fuzz testing," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18, Toronto, Canada: Association for Computing Machinery, 2018, pp. 2123–2138, ISBN: 9781450356930. DOI: `10.1145/3243734.3243804`. [Online]. Available: `https://doi.org/10.1145/3243734.3243804`.

[3] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Commun. ACM*, vol. 33, no. 12, pp. 32–44, Dec. 1990, ISSN: 0001-0782. DOI: `10.1145/96267.96279`. [Online]. Available: `https://doi.org/10.1145/96267.96279`.

[4] C. Cadar and K. Sen, "Symbolic execution for software testing: Three decades later," *Commun. ACM*, vol. 56, no. 2, pp. 82–90, Feb. 2013, ISSN: 0001-0782. DOI: `10.1145/2408776.2408795`. [Online]. Available: `https://doi.org/10.1145/2408776.2408795`.

[5] P. Godefroid, A. Kiezun, and M. Y. Levin, "Grammar-based whitebox fuzzing," in *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '08, Tucson, AZ, USA: Association for Computing Machinery, 2008, pp. 206–215, ISBN: 9781595938602. DOI: `10.1145/1375581.1375607`. [Online]. Available: `https://doi.org/10.1145/1375581.1375607`.

[6] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler, "Exe: Automatically generating inputs of death," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 2, Dec. 2008, ISSN: 1094-9224. DOI: `10.1145/1455518.1455522`. [Online]. Available: `https://doi.org/10.1145/1455518.1455522`.

[7] C. Cadar, D. Dunbar, and D. Engler, "Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'08, San Diego, California: USENIX Association, 2008, pp. 209–224.

[8] P. Boonstoppel, C. Cadar, and D. Engler, "Rwset: Attacking path explosion in constraint-based test generation," in *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS'08/ETAPS'08, Budapest, Hungary: Springer-Verlag, 2008, pp. 351–366, ISBN: 3540787992.

[9] K. Sen, D. Marinov, and G. Agha, "Cute: A concolic unit testing engine for c," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 5, pp. 263–272, Sep. 2005, ISSN: 0163-5948. DOI: `10.1145/1095430.1081750`. [Online]. Available: `https://doi.org/10.1145/1095430.1081750`.

[10] J. Burnim and K. Sen, "Heuristics for scalable dynamic test generation," in *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, 2008, pp. 443–446. DOI: `10.1109/ASE.2008.69`.

[11] P. Godefroid, M. Y. Levin, and D. A. Molnar, "Automated whitebox fuzz testing," in *Network and Distributed System Security Symposium*, 2008. [Online]. Available: `https://api.semanticscholar.org/CorpusID:1296783`.

[12] J. Wang, B. Chen, L. Wei, and Y. Liu, "Skyfire: Data-driven seed generation for fuzzing," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 579–594. DOI: `10.1109/SP.2017.23`.

[13] B. Shastry, M. Leutner, T. Fiebig, *et al.*, "Static program analysis as a fuzzing aid," in *Research in Attacks, Intrusions, and Defenses*, M. Dacier, M. Bailey, M. Polychronakis, and M. Antonakakis, Eds., Cham: Springer International Publishing, 2017, pp. 26–47, ISBN: 978-3-319-66332-6.

[14] J. Corina, A. Machiry, C. Salls, *et al.*, "Difuze: Interface aware fuzzing for kernel drivers," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 2123–2138, ISBN: 9781450349468. DOI: `10.1145/3133956.3134069`. [Online]. Available: `https://doi.org/10.1145/3133956.3134069`.

[15] G. Grieco, M. Ceresa, and P. Buiras, "Quickfuzz: An automatic random fuzzer for common file formats," *SIGPLAN Not.*, vol. 51, no. 12, pp. 13–20, Sep. 2016, ISSN: 0362-1340. DOI: `10.1145/3241625.2976017`. [Online]. Available: `https://doi.org/10.1145/3241625.2976017`.

[16] G. Grieco, M. Ceresa, A. Mista, and P. Buiras, "Quickfuzz testing for fun and profit," *Journal of Systems and Software*, vol. 134, pp. 340–354, 2017, ISSN: 0164-1212. DOI: `https://doi.org/10.1016/j.jss.2017.09.018`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0164121217302066`.

[17] S. K. Cha, M. Woo, and D. Brumley, "Program-adaptive mutational fuzzing," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 725–741. DOI: `10.1109/SP.2015.50`.

[18] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, "Unleashing mayhem on binary code," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 380–394. DOI: `10.1109/SP.2012.31`.

[19] P. Chen and H. Chen, "Angora: Efficient fuzzing by principled search," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 711–725. DOI: `10.1109/SP.2018.00046`.

[20] Y. Li, B. Chen, M. Chandramohan, S.-W. Lin, Y. Liu, and A. Tiu, "Steelix: Program-state based binary fuzzing," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, Paderborn, Germany: Association for Computing Machinery, 2017, pp. 627–637, ISBN: 9781450351058. DOI: `10.1145/3106237.3106295`. [Online]. Available: `https://doi.org/10.1145/3106237.3106295`.

[21] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, "Vuzzer: Application-aware evolutionary fuzzing," in *Network and Distributed System Security Symposium*, 2017. [Online]. Available: `https://api.semanticscholar.org/CorpusID:2354736`.

[22] U. Kargén and N. Shahmehri, "Turning programs against each other: High coverage fuzz-testing using binary-code mutation and dynamic slicing," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015, Bergamo, Italy: Association for Computing Machinery, 2015, pp. 782–792, ISBN: 9781450336758. DOI: `10.1145/2786805.2786844`. [Online]. Available: `https://doi.org/10.1145/2786805.2786844`.

[23] Y.-D. Lin, F.-Z. Liao, S.-K. Huang, and Y.-C. Lai, "Browser fuzzing by scheduled mutation and generation of document object models," in *2015 International Carnahan Conference on Security Technology (ICCST)*, 2015, pp. 1–6. DOI: 10.1109/CCST.2015.7389677.

[24] H. Yoo and T. Shon, "Grammar-based adaptive fuzzing: Evaluation on scada modbus protocol," in *2016 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2016, pp. 557–563. DOI: 10.1109/SmartGridComm.2016.7778820.

[25] H. Han and S. K. Cha, "Imf: Inferred model-based fuzzer," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 2345–2358, ISBN: 9781450349468. DOI: 10.1145/3133956.3134103. [Online]. Available: https://doi.org/10.1145/3133956.3134103.

[26] A. K. Iannillo, R. Natella, D. Cotroneo, and C. Nita-Rotaru, "Chizpurfle: A gray-box android fuzzer for vendor service customizations," in *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, 2017, pp. 1–11. DOI: 10.1109/ISSRE.2017.16.

[27] N. Stephens, J. Grosen, C. Salls, *et al.*, "Driller: Augmenting fuzzing through selective symbolic execution," in *Network and Distributed System Security Symposium*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:2388545.

[28] B. Zhang, J. Ye, C. Feng, and C. Tang, "S2f: Discover hard-to-reach vulnerabilities by semi-symbolic fuzz testing," in *2017 13th International Conference on Computational Intelligence and Security (CIS)*, 2017, pp. 548–552. DOI: 10.1109/CIS.2017.00127.

[29] W. Xu, S. Kashyap, C. Min, and T. Kim, "Designing new operating primitives to improve fuzzing performance," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 2313–2328, ISBN: 9781450349468. DOI: 10.1145/3133956.3134046. [Online]. Available: https://doi.org/10.1145/3133956.3134046.

[30] A. Henderson, H. Yin, G. Jin, H. Han, and H. Deng, "Vdf: Targeted evolutionary fuzz testing of virtual devices," in *Research in Attacks, Intrusions, and Defenses*, M. Dacier, M. Bailey, M. Polychronakis, and M. Antonakakis, Eds., Cham: Springer International Publishing, 2017, pp. 3–25, ISBN: 978-3-319-66332-6.

[31] S. Schumilo, C. Aschermann, R. Gawlik, S. Schinzel, and T. Holz, "kAFL: Hardware-Assisted feedback fuzzing for OS kernels," in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC: USENIX Association, Aug. 2017, pp. 167–182, ISBN: 978-1-931971-40-9. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/schumilo.

[32] H. Peng, Y. Shoshitaishvili, and M. Payer, "T-fuzz: Fuzzing by program transformation," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 697–710. DOI: 10.1109/SP.2018.00056.

[33] W. Han, B. Joe, B. Lee, C. Song, and I. Shin, "Enhancing memory error detection for large-scale applications and fuzz testing," in *Network and Distributed System Security Symposium*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:3837287.

[34] T. Petsios, J. Zhao, A. D. Keromytis, and S. Jana, "Slowfuzz: Automated domain-independent detection of algorithmic complexity vulnerabilities," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 2155–2168, ISBN: 9781450349468. DOI: `10.1145/3133956.3134073`. [Online]. Available: `https://doi.org/10.1145/3133956.3134073`.

[35] T. Petsios, A. Tang, S. Stolfo, A. D. Keromytis, and S. Jana, "Nezha: Efficient domain-independent differential testing," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 615–632. DOI: `10.1109/SP.2017.27`.

[36] I. Haller, A. Slowinska, M. Neugschwandtner, and H. Bos, "Dowsing for overflows: A guided fuzzer to find buffer boundary violations," in *Proceedings of the 22nd USENIX Conference on Security*, ser. SEC'13, Washington, D.C.: USENIX Association, 2013, pp. 49–64, ISBN: 9781931971034.

[37] M. Böhme, V.-T. Pham, M.-D. Nguyen, and A. Roychoudhury, "Directed greybox fuzzing," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 2329–2344, ISBN: 9781450349468. DOI: `10.1145/3133956.3134020`. [Online]. Available: `https://doi.org/10.1145/3133956.3134020`.

[38] M. Böhme, V.-T. Pham, and A. Roychoudhury, "Coverage-based greybox fuzzing as markov chain," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, Vienna, Austria: Association for Computing Machinery, 2016, pp. 1032–1043, ISBN: 9781450341394. DOI: `10.1145/2976749.2978428`. [Online]. Available: `https://doi.org/10.1145/2976749.2978428`.

[39] C. Lemieux and K. 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*, ser. ASE '18, Montpellier, France: Association for Computing Machinery, 2018, pp. 475–485, ISBN: 9781450359375. DOI: `10.1145/3238147.3238176`. [Online]. Available: `https://doi.org/10.1145/3238147.3238176`.

[40] A. Rebert, S. K. Cha, T. Avgerinos, *et al.*, "Optimizing seed selection for fuzzing," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC'14, San Diego, CA: USENIX Association, 2014, pp. 861–875, ISBN: 9781931971157.

[41] M. Woo, S. K. Cha, S. Gottlieb, and D. Brumley, "Scheduling black-box mutational fuzzing," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13, Berlin, Germany: Association for Computing Machinery, 2013, pp. 511–522, ISBN: 9781450324779. DOI: `10.1145/2508859.2516736`. [Online]. Available: `https://doi.org/10.1145/2508859.2516736`.

[42] C. Cadar, P. Godefroid, S. Khurshid, *et al.*, "Symbolic execution for software testing in practice: Preliminary assessment," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11, Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 1066–1071, ISBN: 9781450304450. DOI: `10.1145/1985793.1985995`. [Online]. Available: `https://doi.org/10.1145/1985793.1985995`.

[43] R. McNally, K. K.-H. Yiu, D. A. Grove, and D. Gerhardy, "Fuzzing: The state of the art," *DSTO Defence Science and Technology Organisation*, 2012. [Online]. Available: `https://api.semanticscholar.org/CorpusID:15447929`.

[44] C. Chen, B. Cui, J. Ma, R. Wu, J. Guo, and W. Liu, "A systematic review of fuzzing techniques," *Computers & Security*, vol. 75, pp. 118–137, 2018, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2018.02.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818300658.

[45] J. Li, B. Zhao, and C. Zhang, "Fuzzing: A survey," *Cybersecurity*, vol. 1, pp. 1–13, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:46928493.

[46] H. Liang, X. Pei, X. Jia, W. Shen, and J. Zhang, "Fuzzing: State of the art," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 1199–1218, 2018. DOI: 10.1109/TR.2018.2834476.

[47] P. Godefroid, "Fuzzing: Hack, art, and science," *Commun. ACM*, vol. 63, no. 2, pp. 70–76, Jan. 2020, ISSN: 0001-0782. DOI: 10.1145/3363824. [Online]. Available: https://doi.org/10.1145/3363824.

[48] A. Sabbaghi and M. R. Keyvanpour, "A systematic review of search strategies in dynamic symbolic execution," *Computer Standards & Interfaces*, vol. 72, p. 103 444, 2020, ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2020.103444. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920548919300066.

[49] M. Boehme, C. Cadar, and A. ROYCHOUDHURY, "Fuzzing: Challenges and reflections," *IEEE Software*, vol. 38, no. 3, pp. 79–86, 2021. DOI: 10.1109/MS.2020.3016773.

[50] F. Rustamov, J. Kim, J. Yu, and J. Yun, "Exploratory review of hybrid fuzzing for automated vulnerability detection," *IEEE Access*, vol. 9, pp. 131 166–131 190, 2021. DOI: 10.1109/ACCESS.2021.3114202.

[51] V. J. Manès, H. Han, C. Han, *et al.*, "The art, science, and engineering of fuzzing: A survey," *IEEE Transactions on Software Engineering*, vol. 47, no. 11, pp. 2312–2331, 2021. DOI: 10.1109/TSE.2019.2946563.

[52] X. Zhu, S. Wen, S. Camtepe, and Y. Xiang, "Fuzzing: A survey for roadmap," *ACM Comput. Surv.*, vol. 54, no. 11s, Sep. 2022, ISSN: 0360-0300. DOI: 10.1145/3512345. [Online]. Available: https://doi.org/10.1145/3512345.

[53] C. Beaman, M. Redbourne, J. D. Mummery, and S. Hakak, "Fuzzing vulnerability discovery techniques: Survey, challenges and future directions," *Computers & Security*, vol. 120, p. 102 813, 2022, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2022.102813. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404822002073.

[54] S. Mallissery and Y.-S. Wu, "Demystify the fuzzing methods: A comprehensive survey," *ACM Comput. Surv.*, vol. 56, no. 3, Oct. 2023, ISSN: 0360-0300. DOI: 10.1145/3623375. [Online]. Available: https://doi.org/10.1145/3623375.

[55] X. Zhao, H. Qu, J. Xu, X. Li, W. Lv, and G.-G. Wang, "A systematic review of fuzzing," *Soft Computing*, pp. 1–30, 2023.

[56] T. L. Munea, H. Lim, and T. Shon, "Network protocol fuzz testing for information systems and applications: A survey and taxonomy," *Multimedia Tools and Applications*, vol. 75, no. 22, pp. 14 745–14 757, Nov. 2016, ISSN: 1573-7721. DOI: 10.1007/s11042-015-2763-6. [Online]. Available: https://doi.org/10.1007/s11042-015-2763-6.

[57] E. Andreasen, L. Gong, A. Møller, *et al.*, "A survey of dynamic analysis and test generation for javascript," *ACM Comput. Surv.*, vol. 50, no. 5, Sep. 2017, ISSN: 0360-0300. DOI: 10.1145/3106739. [Online]. Available: https://doi.org/10.1145/3106739.

[58] M. Eceiza, J. L. Flores, and M. Iturbe, "Fuzzing the internet of things: A review on the techniques and challenges for efficient vulnerability discovery in embedded systems," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 390–10 411, 2021. DOI: 10.1109/JIOT.2021.3056179.

[59] C. Zhang, Y. Wang, and L. Wang, "Firmware fuzzing: The state of the art," in *Proceedings of the 12th Asia-Pacific Symposium on Internetware*, ser. Internetware '20, Singapore, Singapore: Association for Computing Machinery, 2021, pp. 110–115, ISBN: 9781450388191. DOI: 10.1145/3457913.3457934. [Online]. Available: https://doi.org/10.1145/3457913.3457934.

[60] Y. Tian, X. Qin, and S. Gan, "Research on fuzzing technology for javascript engines," in *Proceedings of the 5th International Conference on Computer Science and Application Engineering*, ser. CSAE '21, Sanya, China: Association for Computing Machinery, 2021, ISBN: 9781450389853. DOI: 10.1145/3487075.3487107. [Online]. Available: https://doi.org/10.1145/3487075.3487107.

[61] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H.-N. Lee, "Ethereum smart contract analysis tools: A systematic review," *IEEE Access*, vol. 10, pp. 57 037–57 062, 2022. DOI: 10.1109/ACCESS.2022.3169902.

[62] Z. Zhang, H. Zhang, J. Zhao, and Y. Yin, "A survey on the development of network protocol fuzzing techniques," *Electronics*, vol. 12, no. 13, 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12132904. [Online]. Available: https://www.mdpi.com/2079-9292/12/13/2904.

[63] M. Eisele, M. Maugeri, R. Shriwas, C. Huth, and G. Bella, "Embedded fuzzing: A review of challenges, tools, and solutions," *Cybersecurity*, vol. 5, no. 1, p. 18, Sep. 2022, ISSN: 2523-3246. DOI: 10.1186/s42400-022-00123-y. [Online]. Available: https://doi.org/10.1186/s42400-022-00123-y.

[64] J. Yun, F. Rustamov, J. Kim, and Y. Shin, "Fuzzing of embedded systems: A survey," *ACM Comput. Surv.*, vol. 55, no. 7, Dec. 2022, ISSN: 0360-0300. DOI: 10.1145/3538644. [Online]. Available: https://doi.org/10.1145/3538644.