A Survey of Hybrid Fuzzing based on Symbolic Execution

Tao Zhang
Cyberspace Institute of Advanced Technology
Guangzhou University
Guangzhou, China
wayne-tao@Outlook.com

Yu Jiang
Cyberspace Institute of Advanced Technology
Guangzhou University
Guangzhou, China
jiangyu@gzhu.edu.cn

Runsheng Guo
Cyberspace Institute of Advanced Technology
Guangzhou University
Guangzhou, China
guorenshen@Outlook.com

Xiaoran Zheng
Cyberspace Institute of Advanced Technology
Guangzhou University
Guangzhou, China
2111906108@gzhu.edu.cn

Hui Lu†
Cyberspace Institute of Advanced Technology
Guangzhou University
Guangzhou, China
luhui@gzhu.edu.cn

ABSTRACT

Fuzzing has now developed into an efficient method of vulnerability mining. Symbolic execution is also a popular software vulnerability mining technology. Both are research hotspots in the field of network and information security. Hybrid fuzzing is the addition of symbolic execution technology on the basis of traditional fuzzing, and has now developed into a new branch of fuzzing. This article studies the existing hybrid fuzzing methods, reviews the development and evolution process and technical core of hybrid fuzzing, and compares the performance of currently well-known hybrid fuzzing through an experimental method based on symbolic execution. Finally, it discusses the existing problems in the field of hybrid fuzzing testing, and tries to look forward to its future development trend.

CCS CONCEPTS

•Security and privacy~Software and application security~Software security engineering

KEYWORDS

Hybrid fuzzing, symbolic execution, Fuzzing

1 Introduction

In recent years, vulnerabilities have become a security field, and the number of vulnerabilities reports has increased year by year, which has had a serious impact on people's daily lives. With the blowout growth of the number of software and the successive outbreaks of malicious software vulnerability exploits, automated software vulnerability mining technologies have become particularly important. In the field of automated software vulnerability mining, there are two rapidly developing technologies: fuzzing technology and symbolic execution technology.

Fuzzing is the most popular automatic software vulnerability mining tool. The fuzz testing has become a software vulnerability mining technology with good practice effects since the new century, and widely used in academia and industry. By inputting a large number of random test cases into the program to be tested (the test program processed by piling), the test information such as coverage and possible crash information can be obtained. fuzzing has the advantages of high degree of automation, low consumption rate and high utilization rate of computing resources. It belongs to program dynamic analysis technology and has been proved to be effective by many practices [1], [2], [3], and has been widely used in academic and industrial circles. In the aspect of vulnerability mining, fuzzing has played a very amazing effect. Taking AFL (American Fuzzy lop, hereinafter referred to as AFL), a typical mutation based fuzzing tool, for example, hundreds of CVEs have been found since its development in 2014, which does not include the CVE vulnerabilities that were found in many AFL based papers [4], [5], [6]. However, due to the high degree of automation, fuzzing can not test all the possible paths of the program to be tested, and can not cover all the states of the program to be tested. Therefore, in the face of complex branching applications, only the depth of the path can be achieved as soon as possible, and the breadth of the full coverage state can not be achieved.

Symbolic execution is a kind of program analysis technology. The key idea is to represent the program input with symbols instead of specific values, and then get the program input by analyzing the program, and ensure that the input can make the

CIAT 2020, December 4–6, 2020, Guangzhou, China

ACM ISBN 978-1-4503-8782-8/20/10…$15.00
https://doi.org/10.1145/3444370.3444570

code execute in a specific area [7]. If the program to be tested is visually displayed in the form of symbolic execution tree, it can be easily understood that symbolic execution has great advantages in the face of complex branches. Through program path exploration, we can break through the shallow complex branches and achieve full coverage of program state. However, due to the limitation of computation, we can not reach a deeper path. Obviously, fuzzing and symbol execution have their own advantages and disadvantages. Hybrid fuzzing technology combines the advantages of both sides, so as to reach a deeper path and obtain higher coverage. Hybrid fuzzing is better than using symbols execution or fuzzing alone [8], [9], [10].

This paper reviews the development of hybrid fuzzing, an automated software vulnerability mining technology, discusses the key technologies, summarizes the typical tools in the process of technology development. Finally, the future development direction of hybrid fuzzing is discussed, and the relevant summary of this paper is made.

2 Technical overview

2.1 Vulnerability mining technology

At present, there are many techniques for vulnerability mining, the most important of which are three kinds: static analysis, dynamic analysis and symbolic execution.

Static analysis is a software testing technology to analyze the program without executing it. The commonly used analysis techniques include lexical analysis, syntax analysis, abstract syntax tree analysis, semantic analysis, control flow analysis, data flow analysis, stain analysis, invalid code analysis, etc., which are widely used in vulnerability mining. The advantage of static analysis is that it can quickly analyze tens of thousands of lines of code, and provide provable and complete analysis results with formal method [11]. The disadvantages of static analysis are also obvious. The analysis results fail to provide accurate POC (vulnerability trigger proof example). The purpose of analysis is to mine software vulnerabilities, but the analysis results can not generate the input that can trigger the vulnerability. For example, for PNG image reading program, static analysis is used to get the results, but PNG images that can trigger a vulnerability cannot be obtained The PNG file needs to be built according to the analysis results.

Dynamic analysis can execute the program to be tested. Usually, the method of inserting piles is used to obtain all kinds of paths, and the coverage information can be obtained through the path. Different from static analysis, dynamic analysis can monitor the operation of the program to be tested. When the program runs in error, the input that triggers the error is saved. If it is also applied to the PNG image reading program, the corresponding PNG image file that can trigger the error can be saved. Dynamic analysis can also detect dependencies that static analysis cannot detect, such as polymorphic dynamic dependencies. The disadvantages of dynamic testing mainly come from the instrumentation technology, which is to insert specific operations into the program to be tested, so as to detect the program and achieve the purpose of testing. Therefore, this part of code inserted (high-level language or programming language) must have a certain negative impact on the original program to be tested. Another drawback is that it is impossible to get the full coverage rate, because the running of the program to be tested is based on user interaction or automatic testing, which can not guarantee the full coverage of all possible positions of the program.

The emergence of symbol execution technology can be traced back to 1976. King J C. et al. put forward this idea for the first time .[12] Symbolic execution uses program interpretation and constraint solving techniques to generate program input to explore the state space of the program to be tested to trigger vulnerabilities. However, symbol execution should cover all program paths as much as possible. For example, for a conditional judgment statement, two paths must be generated to ensure full coverage of the current link. Therefore, a large number of paths will be triggered in the program to be tested, that is, "path explosion". [12]

Although the mainstream of vulnerability mining is the above three categories, the current research is no longer simply using a single technology, but tends to combine a variety of technologies. The original design idea of fuzzing belongs to dynamic analysis [13]. Dynamic monitoring of test program is carried out through pile insertion technology. However, static analysis technology is also added to mainstream fuzzing tools, such as T-fuzzy[14] based on fuzzing and stain analysis technology.

2.2 Fuzzing

fuzzing technology was first proposed by Miller B P, Koski D and others in 1995. It tested the security of UNIX system and found many vulnerabilities in components [13]. The core of fuzzing is unexpected input. Whether it is based on mutation or generation, it is to get abnormal input different from manual test. Once the program exception is triggered, the input will be saved and finally analyzed by the exploiter. As shown in Figure 1, The core process of fuzzing is as follows: give the initial seed of fuzzy test tool, and then get the input data stream through mutation. The fuzzy test tool will pass the input data to the program to be tested, and track the program through the pile insertion technology to obtain the coverage information. Then the coverage rate guides the previous process, and goes round and round until it encounters the collapse situation. Finally, the researcher sends the input data to the program to be tested The crash was evaluated and POC verified. In the flow chart of fuzzing, there is room for improvement in each stage. Among them, there are three most important ones: seed set, fuzzy test tool and program to be tested.

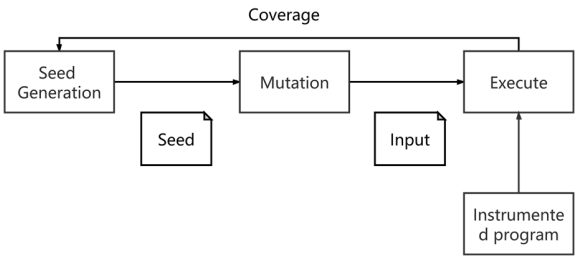

Figure 1. The core process of fuzz testing

Seed set is the cornerstone of fuzzing. A large number of random inputs come from seed mutation technology. High quality seeds are very important for the efficiency of fuzzing, so tools like AFL will provide seed banks. For a fuzzing tool like SLF [15], which can generate seeds by itself at the beginning, there is no need to provide seed sets. However, at the beginning of the SLF test program, predetermined seeds will be generated to replace seeds in the form of data flow. Therefore, seeds are essential.

fuzzing tool is the core of the whole test. The current mainstream fuzzing tools should have two abilities: one is the ability to generate a large number of random inputs. When the seed file (data stream) is obtained, a large number of inputs need to be generated quickly. Each input will be transmitted to the program to be tested in the form of data stream, and execute a certain path, if you want as many as possible Through the various paths of the program, it is necessary to generate as many different inputs as possible; the second is the coordination ability of multiple programs at the same time. Although fuzzing is an endless loop, it does not mean that the test of the second input will be carried out after the end of a test, because a large number of input is generated from the seed file and the operation is carried out in sequence In the execution phase, a large number of input data streams will be temporarily stored in the memory, which makes the program execution a bottleneck of the process, and also seriously slows down the overall efficiency of fuzzing. Therefore, fast and effective execution of the program to be tested is the second important ability of fuzzing Technology.

The program to be tested is the ultimate goal of fuzzing, and all tests are carried out around the program to be tested. For different test objectives, the program to be tested will have some formal differences, and researchers need to manually intervene to make corresponding adjustments for different types of objectives. For example, for library testing, testers usually need to have a certain understanding of the source code, and then write different programs according to different objective functions.

2.3 Hybrid Fuzzing Technology

As mentioned above, the current fuzzing test has certain room for improvement in various stages. Many branches have been developed by combining other technologies. Among them, the hybrid fuzzing technology is a combination of fuzzing technology and symbolic execution technology.

The purpose of this is to combine the advantages of the two technologies to complement each other and alleviate the shortcomings of using one technology alone: low-overhead fuzzing can greatly alleviate the path explosion problem of mixed execution, and mixed execution technology can make up for the execution space of fuzzing. The problem of incomplete traversal. Specifically: the use of a low-overhead fuzzing framework allows faster space exploration when there is a wide traversal space, and AFL, as a Fuzzing framework that has been continuously studied by a large number of researchers, is also very powerful; When the traversal space is limited, high-overhead hybrid execution can be carried out, so as to dig deeper execution space.

There is already a proof of work. Hybrid fuzzing technology has shown good results in automated vulnerability mining, and the combination of the two will achieve better results in code coverage and vulnerability discovery efficiency than using either alone.

3 Research Status of Hybrid Fuzzing

In 2007, Miller B.P. et al. proposed the concept of hybrid testing [13], which combines symbolic execution with automated software testing. One of the advantages of symbol execution is that it can find all the states in the adjacent area of the designated location. One of the advantages of fuzzing is that it can go deep into the state of the program quickly. The author combines the two methods to propose a mixed test method, and the experiment proves that the effect of the mixed test is much better than that of the single case. In the experimental stage, by adding a mixed execution interface to the top level of cute program design, a red black tree structure program and VIM software are tested for vulnerability mining, and the experimental data are given. The author did not clearly point out whether the two technologies could make up for each other's shortcomings, and from the experimental results, the hybrid test combined the advantages of the two technologies, and the author did not improve the two technologies. Although the author did not explicitly propose the hybrid fuzzing technology in this paper, the random testing idea used in it is consistent with the fuzzing, so this research has become the conceptual prototype of many hybrid fuzzing later. In 2012, Pak B S. and others first proposed the concept of hybrid fuzzing [16].

After hybrid fuzzing was proposed, it has not made great progress in the field of vulnerability mining and software testing. With the continuous improvement of computer computing power and algorithm research, symbolic execution technology has made a breakthrough in constraint solving and other aspects. Compared with symbolic execution, fuzzing has not made significant progress, and hybrid fuzzing has not made a breakthrough. In 2015, the emergence of AFL [5] brought new ideas to the industrial and academic circles. It is a milestone development of fuzzing and a turning point of hybrid fuzzing technology. The idea of coverage oriented brings design ideas to hybrid fuzzing, and the fork server technology in AFL also provides implementation ideas for the later hybrid fuzzing technology.

In 2016, N.J. et al. Proposed a new hybrid fuzzing technology, Driller [10]. Different from the hybrid testing in 2007, Driller combines the best efficient fuzzing technology and symbol execution technology by using fork server idea, and realizes the efficient vulnerability mining of binary files. If AFL has executed x rounds and no new state transition (that is, a new code block transition) is found on the bitmap, it means that AFL is stuck. At this time, angr is called for symbolic execution. Each specific input corresponds to a single path in the PathGroup. In each step of the PathGroup, each branch is checked to ensure that the latest jump instruction introduces a path unknown to the previous AFL. When such a jump is found, the SMT solver is queried to create an input to drive execution to the new jump. This input is fed back to AFL, and AFL is mutated in future fuzzy steps. This feedback loop allows us to balance the expensive symbolic execution time with the cheap fuzzing time, and it reduces the low semantic insight of fuzziness into program operations.

Driller establishes the development direction of hybrid fuzzing technology, fuzzing technology plus symbolic execution technology.

In 2018, Hybrid fuzzing has exploded, and researchers at home and abroad have been attracted. QSYM [17] proposed by Yun I and SAVIOR [18] proposed by Chen y deeply analysed the advantages and disadvantages of hybrid fuzzing. They all show excellent results. Afeer [9] proposed by Xie Xiaofei, Li Xiaohong et al. combines AFL and KLEE [19] through a branch coverage method, generates a large number of test cases using AFL, and then searches the coverage information obtained by AFL based on Klee. The search results are used to guide the generation of test cases, and the test cases covering only the uncovered branches are obtained. Through the LAVA-M data set, the author further proves that the hybrid fuzzing technology can get better test results than using fuzzing alone or symbol execution.

In 2020, hybrid fuzzing technology has become one of the most important coverage oriented fuzzing branches. Researchers begin to be dissatisfied with the improvement of the combination mode, and improve the symbolic execution to better serve the fuzzing technology. Pangolin proposed by Rongxin Wu et al. improves the efficiency of vulnerability mining by modifying the constraint solving part in symbol execution. [20] Firstly, pangolin proposed a polyhedron path solving method to improve the efficiency of constraint solving, and the results of this constraint solving were used to guide the mutation process of fuzzing stage, and the compliance execution and fuzzing process were combined more closely to improve the efficiency of vulnerability mining. Since 2007, as shown in Table 1, many researchers have optimized the hybrid fuzzing step by step, and selected representative papers for comparison.

Table 1. Research timeline

Year	Method	Features
2007	Hybrid testing	The concept of hybrid testing was first proposed.
2012	Hybrid fuzzing	Hybrid fuzzing is officially proposed for the first time.
2016	Driller	Driller is based on the fuzz tool AFL and the symbolic execution tool angr. When the fuzzy program is stuck, the symbolic execution is called to solve the input that can reach the new path, so that the fuzz can quickly break through the conditional judgment statement.
2018	QSYM	A hybrid fuzzing technique that focuses more on symbolic execution. QSYM is a practical concolic execution engine tailored for hybrid fuzzing.
2019	Afleer	The trend of domestic research on hybrid fuzzing begins.
2020	PANGOLIN	For the first time, it is optimized for symbolic execution in hybrid fuzzing, not fuzzing.

The two developments of hybrid fuzzing in 2012 and 2016 were the advancement with symbolic execution as the core and the advancement with fuzzing as the core. The former is because the emergence of many symbolic execution engines such as KLEE has greatly improved the efficiency of symbolic execution, making it possible to combine symbolic execution and fuzzing, and the solution speed in symbolic execution can keep up with fuzzing. The latter is due to the development of fuzz testing, such as the emergence of AFL, Libfuzzer, etc., making coverage-oriented fuzzing the mainstream, and this also enables hybrid fuzzing to usher in new technical ideas. The combination of constraint solving and coverage has also become the mainstream in the following years. So far, there have been many good tools for hybrid fuzzing, both academically and industrially.

4 Further Directions

In terms of fuzzing, the mode of fusion of multiple technologies has become the mainstream, as one of the most important branches: hybrid fuzzing technology, which is based on the combination of compliance execution technology and fuzzing technology. Coverage fuzzing testing is also the most important one. It is often used to incorporate hybrid fuzzing testing technology. Therefore, it is possible to improve the combination method that is conducive to improving the coverage rate. For example, the use of instrumentation information for constraint solving can speed up the improvement of coverage in the fuzzing process. Therefore, it is the future trend to use compliance execution to increase the coverage rate, that is, maintaining the advantages of compliance execution in hybrid fuzzing and increasing the coverage rate of the original fuzzing test.

In addition, how to better combine compliance execution and fuzz testing is also a development direction. A more efficient combination will increase the efficiency of the fuzzing program itself, thereby improving the efficiency of vulnerability mining.

There has been a lot of research on hybrid fuzzing, there are also many potentials for development. However, it must be admitted that hybrid fuzzing technology faces many deficiencies and challenges.

First, in many cases, hybrid fuzzing is still inefficient, such as vulnerabilities mining for some large-scale software, whether it is Driller or Pangolin, it has no performance. It produces very good results, and even because of the addition of symbolic execution, hybrid fuzzing is less efficient than pure fuzzing.

The second case is expensive. In the allocation of computing resources, constraint solving is expensive. When the coverage is less than 20%, Driller's execution of the program may cause more than 100 times the operating cost of the program itself.

As PANGOLIN has done, improving symbolic execution itself is also a way to improve efficiency. By using more advanced symbolic execution engines or using new constraint solving algorithms, the efficiency of symbolic execution can be improved, thereby achieving the goal of improving the efficiency of hybrid fuzzing. . Hybrid fuzzing includes not only symbolic execution, but also fuzzing. Using a fuzzing tool that is faster and more efficient than AFL can also improve the overall vulnerability mining efficiency of hybrid fuzzing.

In summary, the development direction is divided into three categories: improving the combination of fuzzing and symbolic execution; improving the efficiency of hybrid fuzzing or reducing resource overhead; starting from fuzzing or symbolic execution to improve efficiency.

5 Conclusions

Since the hybrid fuzzing test was proposed, it has experienced two upsurges. The first moment was that this technology has been officially proposed and named hybrid fuzzing. Since then, researchers have begun to pay attention to such a branch. The second moment was that Driller[10] which combined AFL defined a mode of hybrid fuzzing. A large number of researchers began to pay attention to this type of research. Hybrid fuzzing technology has become one of the most important branches of fuzzing technology. With the development of compliance execution and the development of fuzzing, hybrid fuzzing technology will also have lasting development.

ACKNOWLEDGMENTS

This work is supported by the Guangdong Province Key Area R&D Program of China under Grant No. 2019B010137004 and the National Natural Science Foundation of China under Grant No. U1636215, No.61871140, No. 61972108 and the National Key research and Development Plan under Grant No. 2018YFB0803504, and Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme (2019).

REFERENCES

[1] M. Zalewski, American fuzzy lop, https://lcamtuf.coredump.cx/afl/
[2] Google: honggfuzz (2018). https://github.com/google/honggfuzz
[3] Serebryany K. OSS-Fuzz-Google's continuous fuzzing service for open source software[J]. 2017.
[4] Gan S, Zhang C, Qin X, et al. Collafl: Path sensitive fuzzing[C]//2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018: 679-696.
[5] C. Lyu, S. Ji, C. Zhang, Y. Li, W. H. Lee, Y. Song, R. Beyah(2019). MOPT: Optimized mutation scheduling for fuzzers. In 28th USENIX Security Symposium. (USENIX Security 19) , 1949-1966.
[6] Böhme M, Pham V T, Roychoudhury A(2017). Coverage-based greybox fuzzing as markov chain. IEEE Transactions on Software Engineering, 45(5): 489-506.
[7] Baldoni R, Coppa E, D'elia D C, et al. A survey of symbolic execution techniques[J]. ACM Computing Surveys (CSUR), 2018, 51(3): 1-39.
[8] Majumdar, Rupak , and K. Sen . "Hybrid Concolic Testing." International Conference on Software Engineering IEEE, 2007.
[9] Xie Xiaofei, Xie Xiaofei, Li Xiaohong, et al. Hybrid testing method based on symbolic execution and fuzzing [J]. Journal of Software, 2019, Vol.30Issue(10):3071-3089.
[10] Stephens N, Grosen J, Salls C, Dutcher A, Vigna G(2016, February). Driller: Augmenting Fuzzing Through Selective Symbolic Execution. NDSS, 16(2016): 1-16.
[11] Chess B, McGraw G. Static analysis for security[J]. IEEE security & privacy, 2004, 2(6): 76-79.
[12] King J C. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7): 385-394.
[13] Miller B P, Koski D, Lee C P, et al. Fuzz revisited: A re-examination of the reliability of UNIX utilities and services[R]. University of Wisconsin-Madison Department of Computer Sciences, 1995.
[14] Peng H, Shoshitaishvili Y, Payer M. T-Fuzz: fuzzing by program transformation[C]//2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018: 697-710.
[15] You W, Liu X, Ma S, Perry D, Zhang X, Liang B(2019, May). SLF: fuzzing without valid seed inputs. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 712-723.
[16] Pak B S. Hybrid fuzz testing: Discovering software bugs via fuzzing and symbolic execution[J]. School of Computer Science Carnegie Mellon University, 2012.
[17] QSYM Yun I, Lee S, Xu M, et al. {QSYM}: A practical concolic execution engine tailored for hybrid fuzzing[C]//27th {USENIX} Security Symposium ({USENIX} Security 18). 2018: 745-761.
[18] Chen Y, Li P, Xu J, et al. SAVIOR: towards bug-driven hybrid testing[C]//2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020: 1580-1596.
[19] Cadar C, Dunbar D, Engler D R. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs[C]//OSDI. 2008, 8: 209-224.
[20] PANGOLIN: Incremental Hybrid Fuzzing withPolyhedral Path AbstractionHeqing Huang, Peisen Yao, Rongxin Wu, Qingkai Shi and Charles Zhang.