(State of) The Art of War:Offensive Techniques in Binary Analysis

Razieh Eskandari

May 10, 2019

1 Problem

One of the challenging tasks of the computer security is to automatically analyze executables, which is applied to excavate the vulnerabilities, to perform program verification, or to generate the signature for the vulnerabilities. The problem has become even more critical with the increasing possibility of injecting the malicious code into the executable at the compile time, while the security checks confirmed the safety of source code.
The problem of Binary Analysis (BA) to find crashing inputs is analogous to the halting problem.However, looking for some vulnerable criteria (such as memory corruption) in the executable makes the problem feasible to solve.
Although there are numerous techniques proposed for BA, they are either just a research prototype or their result could not be reproduced regarding massive effort which is needed to replicate the approach again. Additionally, providing a meaningful comparison among different BA methods, which is an unresolved challenge, could give a good insight into the problem. There are different metrics defined in the literature for evaluating the efficiency of BA methods. Some BA approaches do not execute the program from the beginning, and hence they suffer from **replayability** of bug. Others might lack **semantic insights** into the program. Some might be deficient in full code coverage, while others have many false positives. Thus,DARPA organized the Cyber Grand Challenge (CGC) competition in 2016 to evaluate current automatic BA techniques with some unified efficiency metrics and by anidentical CGC dataset of binaries.
Taking CGC as a motivator, ANGR provides a Python-based unified framework which integrates most of the current BA approaches as well as facilitating future novel BA developments on the top of its general framework. Additionally, ANGR provides a basis for comparing BA techniques.
It is notable that all of the source codes are available in public repositories at GitHub.

2 Challenges

The following challenges should be addressed in order to design a unified BA framework:

- The current state of art BA techniques are not built upon the same hardware architecture or even the same operating system. In addition, they even use different analysis paradigm for memory modeling, data modeling, etc. This challenge implicates the need for a good high-level abstraction for instruments, data, etc.
- The proposed framework should cover all of the BA approaches which fall into the categories presented in Figure 1.

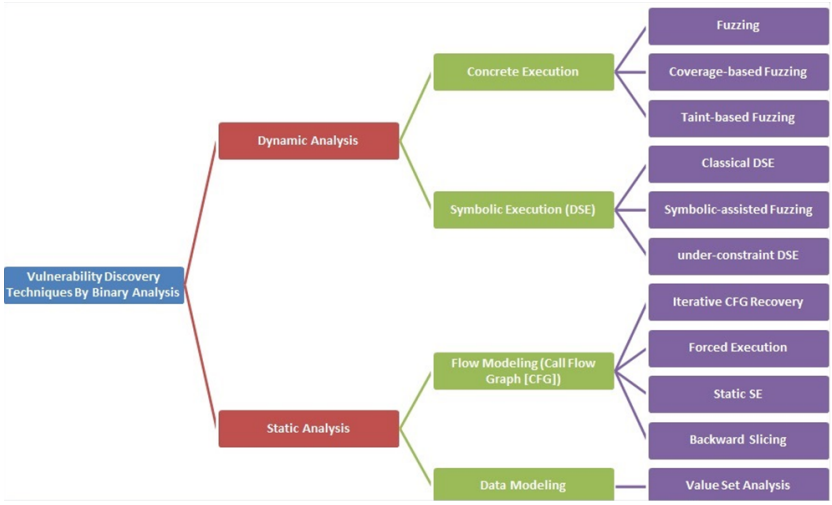

Figure 1: BA approaches

- The third challenge is related to the three exploitation issues: crash reproduction, exploit generation, and exploit hardening. The ability to reproduce crashing inputs helps to identify actual security issues and becomes more important in the case of neglecting false positive detections. Since the detection of non-exploitable vulnerabilities (such as Null-pointer dereference in modern operating systems) can be used for triaging of bugs, the exploit generation problem should be addressed as well.
- Finally, since current modern operating systems utilizes defenses that make exploitable vulnerabilities unable to execute (for example, by address space layout randomization (ASLR) techniques), the generated exploit should also be hardened against such defenses.

3 Solution

ANGR provides the following modules to abstract and solve the problem:

a. *CLE module*, which stands for CLE loads everything, abstracts all binary formats and is responsible for loading ELF/PE binaries and libraries.

b. *PyVEX module*, which is a Python-binding module for Valgrinds VEX could abstract different hardware platform by translating all binary codes of different platforms into an intermediate representation (IR).

c. *SimVEX module*, which is provided to encapsulate the memory states (registers, memory, open files, etc.).

d. *Claripy module* that is composed of several front/back ends. Back-ends handle data modeling abstraction forSatisfiability Modulo Theories(SMT) solver. It is worth mentioning that the SMT solver is used for solving formulas generated during some of the BA approaches. Front-ends augment additional functionality for back-ends.

e. *Analysis module*. Before starting to analysis the binary via a given technique, *Path* and *PathGroup* is created. The Path is used for the abstraction of execution and PathGroup is to classify paths to facilitate the management of path during symbolic execution.

Since the Call Flow Graph (CFG) is a fundamental technique used in many BA approaches, ANGR provide some CFG recovery algorithm as its next novel achievement.A node in CFG is a basic block of code (i.e., the part of code without any jump instruction) and an edge is a jump instruction. To infer the jump target (direct or indirect jumps), ANGR provides four techniques: forced execution, backward slicing, symbolic execution, and value set analysis.
After producing the CFG, analyzing the binary could be started. The following current-state-of-art BA techniques are currently reproduced on the top of the ANGR framework within about 2-30 days which shows ANGR makes the reproduction of a BA method much easier: dynamic symbolic execution based on Various [1, 2, 3] or Veritesting [4], symbolic-assisted fuzzing based on Driller [5], under-constrained symbolic execution based on UCSE [6], static analysis by value set analysis based on VSA [7] and Fuzzing based on AFL [8].
In addition, three modules for crash reproduction based on Replayer [9], exploit generation based on AEG [10], and exploit hardening based on Q [11] was implemented.

4 Evaluation

A comparative evaluation of the aforementioned technique is provided, in terms of the number of true positive crash identified, the rate of false positive, scalability, semantic insights, and replayability.
The results reflect that path merging techniques used to avoid the path-explosion during symbolic execution lead to less binary crashing, so less vulnerability identification.One possible justification could be path merging causes more complex expressions which cannot be solved by SAT solver in a reasonable time.
According to the results, surprisingly fuzzing is more powerful than various Dynamic Symbolic Execution (DSE) approaches in term of crash identified. However, it lacks semantic insight. DSE prefers short paths and it fails to find crashes in deep paths, since the analysis-complexity of DSE is exponentially growingwith the length of the path. Hence, DSE is not scalable.
Another unexpected result is high false positive rates for both under-constrained SE and VSA approaches. By avoiding considering the context of functions, in both techniques, the results are non-replayable with high false positive rates.

5 Future Work

The future work can be mentioned as per following:

- The symbolic-assisted fuzzing could be introduced as the best current approach among the aforementioned implemented ones, in terms of accuracy, replayability, semantic insight, and scalability. However, there is still a lot of work to be done in the binary analysis research field.
- Most of the BA methods execute the program without modeling the randomness sources such as current time. Hence, it is probable that the crashing input of the analysis environment is not replayable in the real-world circumstance. Modeling randomness could be explored as future work.

- Although ANGR uses the Z3 SMT solver, it is recommended to use different solvers for different types of expressions. Since each distinct solver could be suitable for some specific kind of expression.
- The heuristics used to avoid path-explosion in symbolic execution, such as limiting the path length or neglect function with many jump targets are still needed to be improved.
- The heuristics used for CFG recovery can combine paths which are obtained from fuzzing or symbolic-assisted fuzzing in order to provide more code coverage.

References

[1] C. Cadar, D. Dunbar, and D. R. Engler, "KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs," in *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings*, pp. 209–224, 2008.

[2] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, "Unleashing mayhem on binary code," in *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pp. 380–394, 2012.

[3] V. Chipounov, V. Kuznetsov, and G. Candea, "S2E: a platform for in-vivo multi-path analysis of software systems," in *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2011, Newport Beach, CA, USA, March 5-11, 2011*, pp. 265–278, 2011.

[4] T. Avgerinos, A. Rebert, S. K. Cha, and D. Brumley, "Enhancing symbolic execution with veritesting," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, (New York, NY, USA), pp. 1083–1094, ACM, 2014.

[5] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Driller: Augmenting fuzzing through selective symbolic execution," in *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016.

[6] D. A. Ramos and D. R. Engler, "Under-constrained symbolic execution: Correctness checking for real code," in *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, pp. 49–64, 2015.

[7] G. Balakrishnan, T. Reps, D. Melski, and T. Teitelbaum, *WYSINWYX: What You See Is Not What You eXecute*, pp. 202–213. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[8] M. Zalewski, "American fuzzy lop fuzzer," *URL http://lcamtuf. coredump. cx/afl/. VITA VITA*, 2015.

[9] J. Newsome, D. Brumley, J. Franklin, and D. X. Song, "Replayer: automatic protocol replay by binary analysis," in *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pp. 311–321, 2006.

[10] T. Avgerinos, S. K. Cha, B. L. T. Hao, and D. Brumley, "AEG: automatic exploit generation," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*, 2011.

[11] E. J. Schwartz, T. Avgerinos, and D. Brumley, "Q: exploit hardening made easy," in *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*, 2011.