A test data generation method based on the symbolic execution of the dangerous path

Meng Yongdang

China Satellite Maritime Tracking and Control Department, Jiangyin, 214431, China

Keywords: Dangerous path; symbolic execution; constraint solving; test

Abstract. In this paper, the traditional high fuzzing blindness, code coverage is low, low efficiency and other shortcomings of the test case generation process a study by introducing symbolic execution techniques proposed test case generation method based on symbolic execution of the dangerous path, the method focuses on the dangerous path to the target program, the program generates the corresponding test cases dangerous path, effectively improve the efficiency and relevance of test case generation.

Introduction

Fuzzing software vulnerability detection areas most widely used method by providing malformed input data to the target program and monitor the health of the target program in the form of vulnerability. Fuzzing process is the most important test case generation process, the traditional fuzzy test generate test cases usually random variation source input, although this method is very simple, but its high blindness, generate test cases are mostly not cause program exception, so the efficiency of traditional fuzzing generally low. Traditional fuzzing these shortcomings, many researchers have conducted many studies, the most typical way is to use technology to generate fuzzing symbolic execution of test cases, but these tools often have symbolic execution path explosion problem.

This paper studies a typical problem with traditional fuzzing and the path of symbolic execution explosion problem, test case generation method based on symbolic execution of the dangerous path. First, by generating a mixed node selection algorithm to traverse the path of danger based on the generated object program path, and then dangerous path to the target proceedings symbolic execution, generate test cases. This method effectively improve the test case generation targeted improved fuzzing code coverage, increased fuzzing efficiency.

Node selection algorithm to generate a mixed path traversing the risk-based

Path generation algorithm to traverse the dangerous mix node selects the path based on a node after a visit to the left subtree of the left subtree by constraints be inverted to obtain the right subtree constraints. Each visited node, save the path prefix of the node, so in the future path of the search path can use these prefix matching path prefix, just find the path to the destination node to node, and then combined with the path prefix, generate a new path, without the need to search from the beginning to generate path, which can effectively reduce the generated path overhead. In addition, during the route selection for the left subtree have already searched and searched no right subtree of a node, the minimum depth preference node, its right subtree depth-first search, generate a path. The specific steps of the algorithm performed are as follows:

(A). Using the breadth-first traversal algorithm program control flow graph to the nodes in the graph is sequentially stored in the queue, all nodes are not marked as search condition.

(B). The first path from the root to generate and record the corresponding path constraints, and the non-leaf node marks on the path to the left subtree searched, not the right subtree search status, and save the path for each visited path node prefix. Determine whether the nodes on the path are unsafe function call or write memory block of code in a row where, if put in the path to join the list of dangerous path.

(C) if the queue is empty, traversal has completed, the end of the search; if the queue is not empty, the head of the queue began to search, find the front of the queue with the same depth as the head of the queue node and the status of the same node, starting from these nodes , depth-first search method to search for them right subtree, create a new path segment. Each node on the path prefix to save the newly created route segments, change the node path segment these newly generated on the state of the left subtree and the right subtree search has not the status of the search, then those path segments and their respective paths prefix splicing together to form a new path and record path constraints, and determine whether there is a node on the path unsafe function call or write memory block of code in a row where, if put in the path to join the list of dangerous path. Finally, all of these child nodes front of the queue have been searched node from the team.

(D). Repeating steps (c).

Description algorithm flow shown in Figure 1, where the first generation search path a, the second search path generation b, to generate a third search path c1 and c2; last search generation path d1, d2, d3, d4.

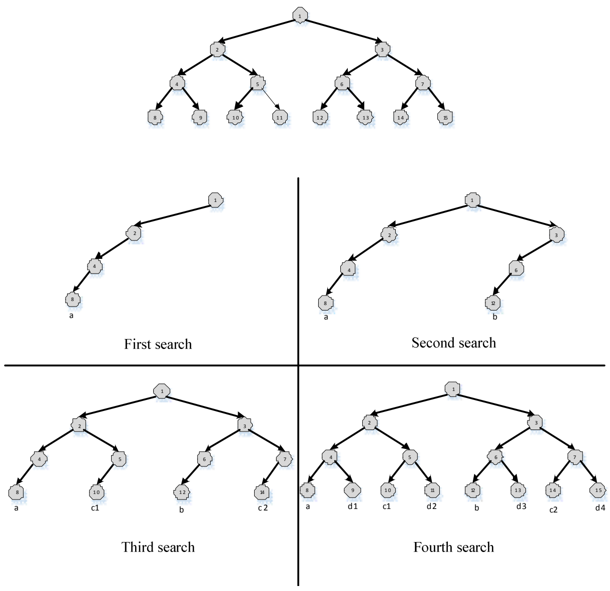

Figure 1. Execution node selection algorithm to generate a mixed sample traverse dangerous path based

Dangerous path test case generation method based on symbolic execution

Test case generation method performed based on first use of the symbol dangerous path generation algorithm to traverse the dangerous mix node selects the path based on the generation of hazardous path, and then the dangerous path symbolic execution, and ultimately generate test cases. In the course of proceedings dangerous path to the target symbol to generate test cases executed, the first input of the target program, symbolic, and then use the target program Valgrind dynamic binary instrumentation analysis, monitoring communication process symbol value in this process collecting constraint dangerous path. Finally, the path constraint to solve dangerous path, and generates the corresponding test cases based on solution results.

Shown in Figure 2, for each dangerous path, first obtain symbolic execution process dangerous path based on its instructions, instruction parsing, and symbol recognition operand instruction judge whether signed variable, symbol recognition mainly through inquiry variable table symbolically achieved symbolic variable table is created at the start of symbolic execution, and in the process of symbolic execution to update and maintain. If the variable is in the symbol table you can not find any symbolic variables, directly on the virtual instruction execution, and parse the next instruction; otherwise requires instruction symbolic execution, the first instruction is converted to an intermediate language code form, then the intermediate language code dynamic stub; in the course of dynamic instrumentation for analysis of symbolic variables related to maintenance, the new symbol symbolic variables to variable table, and according to the existing instruction type variables corresponding symbol in the symbol table variables update operation for unused variables will be deleted; stub dynamic analysis when the instruction is the end of the basic instruction block, according to the jump condition generates the appropriate path constraints, and has a virtual stub instruction execution. Finally, after calling instruction symbol dangerous code block, the output path dangerous path set of constraints. After generating the path constraints, the final means of STP constraint solver to solve them, to generate test cases.

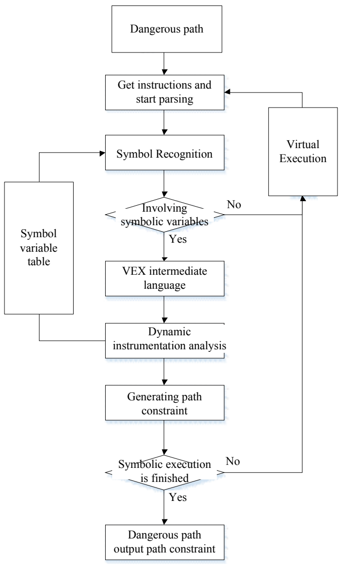
Dangerous path
Get instructions and start parsing
Virtual Execution
Symbol Recognition
Symbol variable table
Involving symbolic variables
No
Yes
VEX intermediate language
Dynamic instrumentation analysis
Generating path constraint
Symbolic execution is finished
No
Yes
Dangerous path output path constraint

Figure 2. Based on the dangerous path of symbolic execution process

Experiment and Analysis

In order to verify the efficiency of the test case generation method proposed in this paper, this paper proposes a method to generate test cases to achieve a vulnerability detection tool SW_Fuzz, then were using SPIKEfile, peach, Fuzzgrind SW_Fuzz four and vulnerability detection tool xpdf software vulnerability detection. SPIKEfile randomly generated test data based on the target software file formats; peach test case generation using two ways, one is randomly generated test data based on the target file format software, or use a small number of data to fill a certain instructive to generate specific data area test cases; Second, random input or modify the template using the data segment to modify the template given input has some enlightening, and then generate test cases. Fuzzgrind using symbolic execution techniques to generate test cases, but it uses a depth-first traversal target program execution state space approach to generate test cases. Xpdf uses four tools to compare test code coverage test results are shown in Table 1 and Figure 3.

Table 1. Coverage comparing the test results

the number of cases / test tools	200	400	600	800	1000
SPIKEfile	32634	38423	39794	40449	40903
peach	44097	53023	58351	61098	62087
Fuzzgrind	49672	67098	75803	79024	81245
SW_Fuzz	53186	72103	82520	89574	93278

The test results can be seen SW_Fuzz code coverage was significantly higher than the other three instruments, and its growth rate is also higher than the other three tools. Using four tools xpdf vulnerability detection test results shown in Table 2.

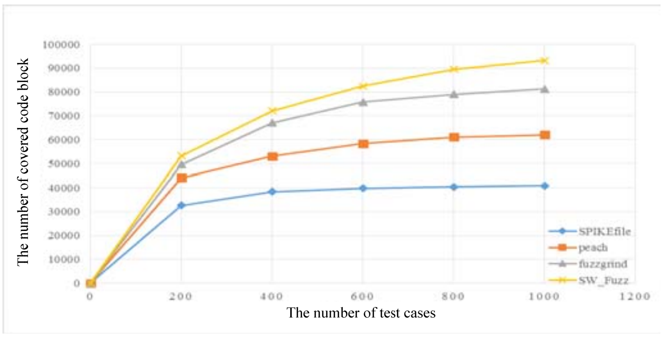

Figure 3. Code coverage growth comparison chart

Table 2. The results contrast detection Vulnerability

project	SPIKEfile	peach	Fuzzgrind	SW_Fuzz
number of generated test cases	1200	1200	1200	1200
Time of test	1.3h	1.4h	6.1h	7.6h
number of basic block coverage	58497	72983	95644	134138
number of code blocks covering dangerous	368	504	866	1249
number of anomalies found	8	16	11	15
number of vulnerabilities discovered	0	0	1	2
proportion of loopholes and exceptions	0%	0%	9.1%	13.3%

As can be seen from the test results, when generating the same number of test cases, the number of basic blocks and code blocks dangerous SW_Fuzz coverage and the number of exceptions and loopholes found significantly more than the other three tools, and the ratio of loopholes and exceptions also higher than the other three tools. From the results above have a higher code coverage and test case generation efficiency test case generation method based on symbolic execution dangerous path.

Summary

In this paper fuzzy test method is based on the problem of vulnerability detection software launched a study, we propose a hybrid generating node selection algorithm based on traversing dangerous path, and use it to generate dangerous path, and then presents a danger sign on the path of execution test case generation methods to improve the efficiency of generation of test cases. The next step will be focused on symbolic execution time and space to bring the issue to continue to increase the cost of research.

Reference

[1] C. Cadar, D. Dunbar, D. Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs[C]. Proceedings of the 8th USENIX conference on Operating systems design and implementation. Berkeley, CA, USA, 2008, 209-224.

[2] C. Cristian, G. Vijay, M.P. Peter, et.al. EXE: Automatically generating inputs of death[C]. CCS 2006: 13th ACM Conference on Computer and ommunications Security, Alexandria, VA, Unitedstates, 2006, 322-335.

[3] K. Sen, D. Marinov, and G. Agha. Cute: a concolic unit testing engine for c. In ESEC/FSE-13:Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering[C], New

York, USA.2005.

[4] P. Godefroid, M. Levin, D. Molnar.Automated Whitebox Fuzz Testing [EB/OL]. [2014-10-22]. http://research.microsoft.com/en-us/um/people/pg/public_psfiles/ndss2008.pdf.