Research on the collaborative analysis technology for source code and binary executable based upon the unified defect mode set

Xiaobing Liang
Institute of Metrology
China Electric Power Research Institute
Beijing, China
liangxbg@163.com

Baojiang Cui
School of Computer
Beijing University of Posts & Telecommunications
Beijing, China
cuibj@bupt.edu.cn

Yingjie Lv
Institute of Metrology
China Electric Power Research Institute
Beijing, China
lyingjie@epri.sgcc.com.cn

Yilun Fu
Institute of Metrology
China Electric Power Research Institute
Beijing, China
fuyilun@epri.sgcc.com.cn

Abstract—The security defect detection technology based on source code usually makes use of static analysis methods to detect security vulnerabilities in the test software, which does not consider the runtime information of program execution and the interaction information with the program runtime surrounding environment, so the security defect detection technology based on source code usually results in higher false positive rate. The binary program vulnerability detection methods based upon dynamic analysis usually have lower false positive rate, but their effectiveness depends entirely on test case generation, the detection efficiency of dynamic analysis based on binary program is lower. Moreover, the research of source code vulnerability detection technique and binary executable vulnerability detection technique are essentially fragmented, without considering the relationship between the two classes of methods. In order to improve the efficiency for source code and binary executables vulnerability detection, we presents a collaborative analysis method for vulnerability detection of source code and binary executables based upon the unified defect mode set, meanwhile, we verify the effectiveness of our method using a concrete example in this paper .

Keywords- Source code defect mode; binary executables defect mode; unified defect mode set; collaborative analysis.

I. INTRODUCTION

As the scale of software becomes larger and the structure of software becomes more complex, the number of security vulnerabilities in software has been showing a rising trend inevitably. In order to control the number of software vulnerabilities, and improve software quality and security, security researchers in domestic and abroad have developed many detection tools for software vulnerability. For software with source code, there are many automatic testing tools based on the source code oriented security vulnerability model, such as Coverity Prevent[1], which is a static analysis tool based on vulnerability mode matching and grammar scanning, which is developed by Stanford University. This tool uses parallel and incremental analysis to find security vulnerabilities which are difficult to be located, and could lead to the collapse of the program. Klocwork K8[3] is a static analysis tool for source code. It could first model the source code of the test software, then simulate all possible execution paths of the software and detect possible security vulnerabilities, logic errors in the test software. CodeSonar[4] is a static analysis tool for detecting security defects in the test software, which is developed by using C/C++ and we could obtain the source code. It makes use of context-sensitive data flow analysis methods to find buffer overflow vulnerabilities, null pointer references, memory leaks and unreachable paths in the test software. DTS [5, 6] is a automatic detection system based on source defect model developed by GongYunzhan, which explores the source defect patterns, defect pattern description and unified testing framework and other issues.

For automatic detection of security defects in binary code, there are some typical tools can detect security defects in binary code [7-9], such tools are common for certain types of software vulnerabilities, and analysis the test binary program using Conclic Execution. The key step is to judge to do the symbolic execution or actual execution during the running process of the binary program, and determine whether the value of the register or memory unit is related to

978-1-4799-8873-0/11 $31.00 © 20115 IEEE
DOI 10.1109/IMIS.2015.40

the input variables of binary program, if relevant, symbolic execution is carried out, otherwise the binary program is actually executing. It combined the symbolic execution with actually execution, made full use of characters that symbolic execution can accurately simulate the related running information about the binary program, which can also improve the accuracy of the analysis, and reduce false alarm rate of defect detection.

With the in-depth study of security defects automatic detection technologies about the test software based on the source code and binary code, we find that software defect automatic detection technology based on defect mode has high efficiency, and convenient to implement. But this method just statically analyze the structural information about the test program, it does not consider the relevant information when the program is running, which lead to a high false negative rate and false alarm rate. The automatic detection technology for binary code security defects based on Conclic Execution which takes full use of the character that symbolic execution can accurately analyze the code property and the related information when program runs, with low false alarm rate, but this method will have to face with expensive computational cost and path explosion problem due to the complexity and depth-nest loop when simulate all execution paths with all input variables.

II. PROBLEM DESCRIPTIONS

Source code-oriented security flaw detection technology can easily scale, automate, but has high false positive rate, false negative rate, binary program-oriented security flaws dynamic detection technique has a low false positive rate and accuracy, but the computation cost is expensive. Now, source code-oriented security defect detection technology and binary program-oriented security defect detection technology are almost fragmented, but the relevance between them is not considered.

III. DEFECT MODE

A. Description of Source Code Security Defect Based on Mode State Machine

For buffer overflow vulnerabilities in the test program with source code, Wang Lei[13,14] firstly establishes a buffer overflow attribute description language to constraint describe the instrumentation code, then use model checking technology to detect buffer overflow vulnerabilities in the test program. For the special requirements of test case generation for secure operating system, Cheng Liang[16] designs the model of buffer overflow vulnerability, and then uses the model checking technology to precisely detect the vulnerabilities which are caused by the potential buffer overflow in the program. Xu Chao[15] instruments the test program and generates test cases using safety state transition based on model checking. Jin Dahai[17] puts forward a method that uses progress-constraint information to describe process defect mode, it can improve the accuracy of software static analysis. He Kaiduo[18] presents a model checking method for modeling and verifying the source code based on predicate abstraction and counterexample refinement technology.

B. Formal Description for Binary Code Security Defect Patterns

For the study of security defect pattern of binary code, Wen Weiping[20] studied the research ideas and trends of binary code security vulnerability mining, then pointed out that at present the research of binary code vulnerability mining mainly focus on three aspects:(1) Software vulnerability mining model and process research; (2) Vulnerability mining technology based on structured-patch comparison; (3) Research on assemble instruction sequence searching and locating based on the pattern of vulnerability feature. Wen Weiping[21, 22] divided the defect mode of binary code into danger function defect mode and logical error mode, then used the finite state machine to model the defect mode based on danger function. Through the analysis of research status of automatic detection technology[7, 8, 9, 19, 23] for binary code defects at present, we found that the defect detection technology for binary code is lack of abstraction and understanding about binary code defect patterns, and use the unified formal description language to describe the security defect pattern of binary code.

C. BCFA Model of Binary Code

Firstly, we build the BCFA (Binary Control Flow Automata) model of Binary code, which is consisted of a finite number of nodes and sides, each BCFA has only one entry node and one exit node. The instruction sequences which are continuously executing consists a node of BCFA, jump instructions constitute state transition function of BCFA.

BCFA model can be seen as a state transition system, the semantic information of binary code in the process of actual execution can be described by combining program control flow diagram with dynamic running information. Using $P(C_i)$ denotes context information of the corresponding node, the context information includes path constraint conditions, instruction address and the register values nearing node instruction address. State $\xi = < i, JumpConds, Boolean >$, where i denotes the index of ith node in BCFA, JumpConds denotes node jump constraint conditions, $Boolean = \{0,1\}$. $f(\xi)$ denotes mapping program state to the corresponding node location，

$$f(< i, JumpConds, Boolean >) =< i, P(C_i) >$$

BCFA model can be expressed in form (1).

$$\xi_0(P(C_0)) \xrightarrow{f_1} \xi_1(P(C_1)) \cdots \xrightarrow{f_n} \xi_n(P(C_n)) \quad (1)$$

Where $\xi_0(P(C_0))$ denotes the initial state of binary code, the form (2) which is projected by $\xi_0, \xi_1, \cdots, \xi_n$ is called the path.

$$\xi_0(P(C_0)), \xi_1(P(C_1)), \cdots, \xi_n(P(C_n)) \quad (2)$$

The state transition process of state transition system τ is denoted by state conversion sequence $s_0 s_1 \cdots s_n$, the state

space of binary code is denoted by Ω. The state transition process of state transition system is the finite sequence of state, the context information of program running is considered during the abstract process.

The input data processing of binary code can be described by the tag migration system[19] $ST =< B, b_1, T, \Delta >$, where B denotes the collection of basic block of binary program, b_1 denotes the first basic block that can be executed after reading input data, T denotes the collection of state transition of binary program. $\Delta = B \times T \times B$denotes the state transition relations of binary program, where the transition relationship τ is defined as follows：

$$\tau = \begin{cases} \text{DecideTrue}(c), b_{next} = b_A \\ \text{DecideTrue}(\neg c), b_{next} = b_B \end{cases} \quad (3)$$

For tag transition system of binary code , DecideTrue is used to determine whether the corresponding state transition condition is true, if it is true, jump to basic block b_A and execute the instructions in b_A, if not, jump to basic block b_B and execute the instructions in b_B. The basic idea based on symbolic execution and constraint solving is that binary program can be expressed as {PostCond}{PreCond}Q using Hoare logic, where PreCond is a precondition that binary program need to be met before it execute, PostCond is a post-condition that binary program should be met after it execute. Assume that the constraint conditions Cond1∧Cond2∧$\cdots$∧Condn. can be deduced by PreCond in the symbolic execution process of binary program, then it should be:

$$PreCond \wedge Cond_1 \wedge Cond_2 \cdots \wedge Cond_n \rightarrow PostCond \quad (4)$$

For solving the equation (5), if there is a set of solutions to meet the constraints, it means there exists a set of inputs that have made the result of program running do not tally with the program specification. If the program specification is correct, the program must contain errors.

$$PreCond \wedge Cond_1 \wedge \cdots \wedge Cond_n \wedge \neg PostCond \quad (5)$$

IV. FORMAL DESCRIPTION OF SECURITY DEFECT FOR BINARY CODE BASED ON THE PATTERN OF BCFA MODEL

So far, the unified, accepted conclusions have not yet formed for the definition, classification, characteristics and vulnerability mining method of binary code security defect. The research of automatic detection technology for binary code security defect at home and abroad usually focuses on some specific vulnerabilities, the effect and efficiency of study also have certain limitation [7, 8, 9, 22, 23]. Wen Weiping [23] divides the types for security defects of binary code into function model and logic error model, where the function model mainly focuses on modeling the specific danger functions and the danger functions involved mainly are (_strcpy, _strcat, _sprintf, _lstrcpyA, _lstrcatA, wsprintfA, sprintf, MultiByteToWideChar, strcpy, strcat, recv). This classification method takes a important step for detecting binary code security defect in large scale, but this classification method has a limitation that it cannot describe security defect of binary program, such as using hard coding to realize the functions of the danger functions [24], the functions such as *strcpy, memcpy* will be disappointed in assemble layer, so only modeling the danger functions may be faced the problem of low efficiency detecting security defect, what's more, some security defects may exist in the functions written by the programmers. Therefore, the key problem that this paper focuses is to explore how to use the unified formalized description language to describe the defect mode of binary code and realize the automation, scale detection for binary code.

Definition 1 FSA(Finite State Automata)， $FSA = (Q, \Sigma, \delta, q_0, F)$, where Q is for the set of finite state,Σdenotes input collection, $q_0 \in Q$ denotes the initial state of the program， $F \subset Q$ denotes accepting state and ending state of the finite state machine, δ： $Q \times \Sigma \rightarrow Q$ denotes the state transition function, $\delta(q_i, x) = q_j$ means that the state q_i transfers into $q_j$after accepting input x.

For classification of binary code security defect, the classification method is similar to the classification method of source code security defect pattern. The security defect model which was caused by complex loop structure in binary code is established as follows. Many security problems [24] caused by the complex loop structure because there exists the loop-writing-memory operation or loop-copy ing-memory operation in inner loop during the process of writing memory while the programmer does not consider comprehensive boundary which will be written into. The security defect mode caused by the complex loop structure is given as follows:

M1={<WriteBuffer,DataSource, wBufferLen, dBufferLen, LoopNum, Opt>| Opt = Write, wBufferLen <LoopNum dBufferLen|DataSource ∈InputSource}

where *WriteBuffer* denotes the buffer that is written into data, *wBufferLen* denotes the length of *WriteBuffer*, *DataSource* denotes the data sources of the buffer of *WriteBuffer*, LoopNum denotes the number of executing loop, *Opt* denotes the style that operates the memory, *Opt={write,copy}*, *write* denotes the operation of writing data into memory, *copy* denotes the operation of copying data in memory, *InputSource* denotes the input source of corresponding binary code.

V. COLLABORATIVE ANALYSIS MECHANISM FOR SECURITY DEFECT MODE BETWEEN THE SOURCE CODE AND THE BINARY CODE

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper.

A. *Unified Defect Model for Source Code and Binary Code*

Although the defect mode of source code has different manifestations with the defect mode of binary code, they are same in the essence of considering from the angle that the defect is triggered, and have the same trigger condition. So, the security defect for source code and binary code has the unified defect mode set.

Definition 2 Defect mode set M, where $M = \{M_1, M_2, \cdots, M_n\}$, M_i, $1 \le i \le n$, denotes defect modes that are corresponding to different security defects.

For different security defect type, there is different defect mode M_i, for the security defect that is associated with input validation. The corresponding defect mode is defined as:

$$M_i = \{< funName, \cdots, DestVar, \cdots, DataFroVar \cdots > | len(DataFroVar) > len(DestVar) \ or \ DataFroVar \in UserInputSource\}$$

where funName denotes the function name of this type of defect form, $DestVar$ denotes variable in function that data is written into ,$DataFroVar$ denotes the data source that is written into variable $DestVar$, $len(DataFroVar)$ denotes the length of variable DataFroVar, $len(DestVar)$ denotes the length of variable $DestVar$, $UserInputSource$ is the input source corresponding to the binary code.

For buffer overflow vulnerability caused by dangerous function, such as buffer overflow vulnerability caused by strcpy(*Destbuf,SourBuf*). Its vulnerability trigger condition is len(*SourBuf*)>len(*DestBuf*), or the data in variable *SourBuf* comes from the input file controlled by the user. The buffer overflow vulnerability mode caused by dangerous function is:

{<strcpy,DestBuf,SourBuf>|len(SourBuf)>len(DestBuf) or SourBuf∈ UserInputSource}

The automatic detection algorithm for security defects based on the unified defect mode set of source code and binary code can be described as the followings:

(1) Firstly, we analyze the test software (source code or binary code) statically, then obtain the suspicious code defect position or suspicious defect function set $X = \{x_i | i \in Z\}$ in the test software.

(2) For $\forall x_i \in X$,if $x_i \in M_k$,according to the defect mode M_k of set M, making further data stream analysis for the test software (the different characters for source code or binary code), then obtain the related information of the elements involved in M_k;

(3) According to the trigger condition of the definition of the defect mode M_k and the related information obtained from the second step of data flow analysis, we can determine whether it meets the trigger condition in the definition of defect mode M_k . If it meets the trigger condition, there may exist detect mode M_k which is correspond to this security defect;

（4）If there exist security defects according to the type of the security defects and the result of data stream analysis in the second step, we use symbolic execution and constraint solving (for source code or binary code) to generate specific test cases to trigger the potential security defects in the test software.

B. *Collaborative Analysis Method Based on the Unified Defect Mode Set of Source Code Defect Mode and Binary Code Defect Mode*

Through analyzing the correlations between source code defect mode and binary code defect mode, we can find that: (1) Static analysis for source code does not consider the relevant information about program compiling and runtime situation, which could cause that the defect modes through source code static analysis are lack of relevance with the actual vulnerability modes during the running process of the test program. (2) Lack of better understanding of the essence of vulnerability in the binary executables, even lack of using abstract formal description of the vulnerability essence. (3) In most cases, there is no record about the context information in the program execution environment during the process of mining and validating vulnerability, which results that the test cases that generate can drive program execution to the potential defect position, but because of lacking the context information of program execution, which need a lot of human intervention in the process of vulnerability analysis, the above three reasons could cause low vulnerability analysis efficiency.

For the source code, we firstly instrument the complex program structure in the source code, then carry out the analysis of the complex structure information in the process of the program execution, after the preliminary analysis for the complex structures in the program, finally extract the defect mode feature for complex program structures. We should summarize abstract and extract all kinds of characteristics for different defect modes. Using pattern characteristics to reflect the manifestation form of defects in source code, and use formal description languages to express the defect mode, forming a defect mode for complex program structure.

For binary code, we firstly obtain the context information based on instrumentation on the binary code and record the underlying vulnerabilities, then using a unified formal description to describe the vulnerability pattern based on binary code modeling technology. On this basis, we can make a deep analysis on the mechanism, position and forms of security vulnerability about binary code, then using formal description language to describe the essence of the binary vulnerability.

VI. CONCLUSIONS

An analysis method for software security defect detection is proposed in this paper based on the collaborative analysis for the source code defect mode and the binary code defect mode. On this basis, the correlations between the source code defect mode and the binary code defect mode are given in this paper, furthermore, the collaborative analysis method based on the unified defect mode set is verified through the experiment. From the two different latitudes for the source code defect mode and the binary code mode, we want to obtain more essential and formal description method for software security defect by analyzing the relationship between the source code defect detecting and the binary code defect detecting.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their hard work, helpful comments and suggestions. This research was supported by the Science and Technology Project of China State Grid Corp. This research was also supported by the National Natural Science Foundation of China (No.61170268).

REFERENCES

[1] http://www.coverity.com/.
[2] https://www.fortify.com/products/hpfssc/source-code-analyzer.html/.
[3] http://www.klocwork.com/.
[4] http://www.grammatech.com/products/codesonar/overview.html.
[5] Yang Zhao Hong, Gong Yun Zhan, Qing Xiao, et al. DTS-A software defects testing system[C]. In proceedings of the 8th IEEE International Working Conference on Source Code Analysis and Manipulation. Beijing: SCAM, 2008: 269-270.
[6] Wang Yawen, Research on software testing technology based on defect pattern [Doctoral Dissertation], Beijing, Beijing University of Posts and Telecommunications, 2009.
[7] Patrice Godefroid, Michael Y.Levin and David Molnar. Automated whitebox fuzz testing [C]. In proceedings of the 15th Annual Network and Distributed System Security Symposium. San Diego CA , 2008.
[8] David Molnar, Xue Cong Li and David A. Wagner. Dynamic test generation to find integer bugs in x86 binary linux programs [C]. In proceedings of the 18th conference on USENIX security symposium, 2009.
[9] Tielei Wang, Wei Tao, Guofei Gu, et al. TaintScope: A checksum aware directed fuzzing tool for automatic software vulnerability detection [C]. In proceedings of the 31st IEEE Symposium on Security and Privacy. Oakland, 2010.
[10] http://www.kclocwork.com/resources/whitepapers.asp.
[11] http://findbugs.sourceforge.net/bugdescriptions.html.
[12] Paul S, Prakash A. A Framework for Source Code Search Using Program Patterns[J]. IEEE Trans. on Software Engineering, 1994, 20(6):463-475.
[13] Wang Lei, Li Ji and Li Boyang. Precisely detecting buffer overflow vulnerabilities[J]. Journal of Computer Research and Development, 2008, vol.36(11), pp:2200-2204.
[14] Wang Lei, Chen Gui, Jin Maozhong.Detection of code vulnerability via constraint-based analysis and model checking[J]. Journal of Computer Research and Development, 2011, vol 48(9), pp:1659-1666.
[15] Xu Chao, He YanXiang and son on. Method to detect buffer overflow in C programs[J]. Application Research of Computers, 2012, vol 29(2), pp:617-620.
[16] Cheng Liang, Zhang Yang and Feng Dengguo. Approach of degenerate test set generation based on secure state transition[J]. Journal of Software, 2010, vol 21(3), pp:539-547.
[17] Jin Dahai, Gong Yunzhan, et al. The application of constraint-procedure information in software static testing[J]. Journal of Computer-Aided Design & Computer Graphics, 2011, vol.23(3), pp:534-542.
[18] He Kaiduo, Gu Ming, et al.Source-oriented software model checking and its implementation [J]. Computer Science, 2009, vol.36(1), pp:267-272.
[19] Lu Xicheng, Li Gen, et al. High-Trusted –Software-Oriented automatic testing for integer overflow bugs[J]. Journal of Software, 2010, vol.21(2), pp:179-193.
[20] Wen Weiping, Wu Xingli, et al. Research ideas and trends for software security vulnerability mining[J].Netinfo Security, 2009, pp:78-80.
[21] Xu YouFu, Wen Weiping and Wan Zhengsu. Research on security vulnerability mining method based on vulnerability model checking[J]. Netinfo Security, 2011, pp:72-75.
[22] Liu Bo, Wen Weiping, et al. ClearBug-an improved automated vulnerability analysis tool[J]. Netinfo Security, 2009, pp:28-31.
[23] Cui Baojiang, Guo Pengfei and Wang Jianxin, et al. Binary code execution path based on symbolic and actual program execution[J]. Journal of Tsinghua University, 2009, Vol.49, No.S2, pp32-35.
[24] Wei Qiang, Jin Ran and Wang Qingxian. Buffer overflow detection model based on intermedia assembly[J]. Computer Engineering, 2009, vol.35, No.3, pp169-172.