A Systematic Review of Binary Program Vulnerabilities Feature Extraction and Discovery Strategy Generation Methods

Bo Zhang\textsuperscript{12*}, Zesheng Xi\textsuperscript{12}

\textsuperscript{1}Institute of Information and Communication, Global Energy Interconnection Research Institute, Nanjing 210009, China
\textsuperscript{2}State Grid Key Laboratory of Information & Network Security, Nanjing 210009, China
\textsuperscript{*}Corresponding author’s e-mail: zhangbo@geiri.sgcc.com.cn

Abstract. At present, the cyber security situation has turned increasingly serious across the globe. The cyberspace has become the fifth domain of various nations following the four domains of land, sea, sky and outer space. It is gradually obvious for the cyber attacks to collectivize and nationalize. Under this circumstance, as the first line of defense in maintaining cyber security defense, the control on the vulnerabilities is extremely significant. In consideration of the problem of low discovering efficiency in current vulnerability fuzzy testing discovery methods, this paper intends to explore the binary program vulnerability feature extraction and vulnerability discovery guiding strategy generation methods. Then, the extraction of the program control flow and data flow information and slicing operation are obtained according to program logic. Depending on a potential assessment model of binary program vulnerabilities based on the main hierarchy analysis method, the paper achieves a reasonable assessment of the potential of vulnerabilities in different slices of the program. Finally, in accordance with the evaluation results, a generation method for the vulnerability discovery path is proposed to improve the efficiency of binary program vulnerability discovery.

1. Introduction
Over the years, with the forceful introduction of “Internet plus” mode technologies represented by the cloud computing, Internet of Things, and big data, and the access of smart terminal devices (electric vehicles, smart sockets, smart appliances, sensors, etc.), the power grid informatization construction is in a stage of rapid development, in particular the swift growth of network assets in some aspects such as generation, transmission, distribution, electricity, and power consumption. A wide variety of network assets have a significant effect on the improvement of the power grid business capabilities and working efficiency of the power grid. However, due to the complicity of business types and business logic in the network assets, it makes it more difficult to protect information security of the power grid. According to the distribution of vulnerabilities from 2016 to 2017 released by China National Vulnerability Database (CNVD), application vulnerabilities account for 62\%, ranking first among various types of vulnerabilities. Meanwhile, it effectively proves from the outbreak of the “Eternal Blue” virus and the consequent harms that the security hazards caused by application vulnerabilities are incalculable. It can be summarized that the invasion and penetration of power grid network assets by means of the security hole is a potential risk factor that threatens the normal operation of the power grid.
Therefore, it is of great necessity to perform the research on the mechanism and technology of automatic discovery for the vulnerabilities of the power grid assets. The efficient vulnerability discovery methods and automated execution technologies are taken into consideration by exploring the feature analysis and extraction of binary program vulnerabilities to establish efficient and accurate discovery capabilities for binary program vulnerabilities. For this reason, the potential security risks which pose serious threats on power grid network assets will be reduced, and the overall security protection capabilities will be enhanced. Finally, the power grid’s ability to respond to cyber security threats will be reinforced.

2. Analysis of Research Status

In recent years, scholars and researchers at home and abroad regarding this field have endeavored a lot to be immersed in the investigations on the related technology of automatic discovery for program vulnerabilities. Related technologies include code coverage-oriented fuzzy testing and automatic discovery of program vulnerabilities by the symbolic execution. The differences in binary vulnerabilities automated discovery tools are mirrored in the methods of vulnerability testing. In addition, relevant researches have been made on the ways how to enhance the fuzzing test or the route sensitivity of the fuzzing test to improve code coverage.

The first step of the vulnerability detection is to extract the typical characteristics of this type of vulnerability, which are analysed by the operating mechanism of a large number of known vulnerabilities. Then, the general laws will be concluded. On this basis, the key features owned by the binary code with security defects will be abstracted. At present, many theoretical studies focusing on the analysis of binary program vulnerability characteristic have been extensive at home and abroad. It can be divided into static code analysis and dynamic stain analysis pursuant to whether it needs actual operation.

(1) Static code analysis refers to the process of review and analysis of code without actually running the binary program code [1]. Through static analysis, in the case that there is no need to construct a condition that triggers a vulnerability, the code is thoroughly examined without increasing the program execution burden. However, static analysis also have some drawbacks such as faulty display and omission.

(2) Dynamic taint analysis [2, 3] means the analysis method, which makes use of the running binary code to collect information during the program running and then discover whether there is a security vulnerability in the code. The academic community has been dedicated to the binary program under the x86 architecture. Specifically, the process is that all program input is marked as pollution data and track the flow of pollution data during the execution of the program. If sensitive functions or sensitive program segments are operating on the stain data, it is assumed that the vulnerability may exist. Compared with static analysis, the method of dynamic detection is more beneficial to analyze whether there are vulnerabilities in binary code, which can effectively reduce false positives. However, it is easy to generate false negatives resulting from the explanation that the test data is difficult to cover all program streams. In addition, dynamic stain analysis will cause the additional cost of the program execution. In recent years, domestic and foreign scholars have proceeded profound theoretical research and tool development with relation to the dynamic pollution analysis and distribution. According to Literature [4], the tool, Untangle, is introduced to make a dynamic analysis of vulnerabilities in binary files based on the taint tracking technology. It will track whether pointers are copied and pointing to released memory, and report all dangling pointers pointing to specific memory within a specified time period. However, this tool requires testers to build the test cases so as to obtain paths, which cover vulnerabilities. Since test data does not completely cover all paths to vulnerabilities, false positives will be produced. Literature [5] presents a dynamic stain propagation analysis scheme, which is based on control flow analysis, symbolic execution, and path predicates to solve the constraint conditions of incomplete stain paths. To be specific, it is divided into two stages. First, obtain the strategies generated by the incomplete stain program branches and determine the additional propagation conditions and paths required for offline analysis. Then, apply the strategies to the process of dynamic stain analysis.
An extended platform TEMU based on QEMU virtual machine is recommended for the sake of solving the difficulty of dynamic analysis program proposed in literature [6]. The TEMU platform performs in-depth analysis of the activities in the kernel and the interaction between multiple processes in a fine-grained dynamic analysis. Both the binary intermediate language and the taint analysis at the bottom instruction set level are confronted with weak semantics, which cannot offer a more intuitive help for software vulnerability analysis. For this reason, literature [7] integrates the dynamic stain analysis technology based on type perception and the symbolic execution technology for type variables. This technology attaches the type information by the stain variables that have been marked as “pollution” to constitute the source of the stain. In the process of tracking the transfer of taints, the semantics of the instructions, public library functions, and taint type information are employed to implement variable-level taint propagation. Meanwhile, the symbolic execution of type variables are performed to obtain program execution path conditions, and form a vulnerability signature database. For the purpose of reducing the time cost of dynamic analysis, literature [8] proposes a dynamic stain analysis method based on the execution trace offline index. This method first adopts dynamic instrumentation to track the program execution process, and then combines the trace file to record the execution status information. Finally, the index files are set up, which only focus on operations related to the stain data during the spread of the stain.

In General, although tremendous researches on dynamic stain analysis technology have been implemented at home and abroad, the existing research still has some problems. For instance, the problems of under pollution and over pollution are caused by the implicit flow problem. The failure to eliminate the stain promptly decreases the efficiency of taint analysis and the accuracy of the analysis results. The additional cost arises from the dynamic instrumentation and dynamic code rewriting technology. As a consequence of the incomplete coverage of the test data path, high false alarm rate and low accuracy are produced.

In order to improve the speed of vulnerability discovery, it is of great significance to give priority to testing code blocks with high possibility of vulnerabilities. This paper aims to attain the evaluation results concerning the possibility of vulnerabilities and related path information through program slicing analysis. In addition, the generation method for vulnerabilities discovery is discussed based on comprehensive analysis of accessibility and complexity.

3. Binary Program Vulnerability Feature Extraction and Discovery Strategy Generation Method

3.1. Overall Research Ideas

1. Research on Preprocessing Technology of Binary Program Based on Slicing
   - Data flow graph extraction based on basic blocks
   - Control flow graph extraction based on basic block

2. Research on the potential assessment model of binary vulnerabilities
   - Vulnerability feature modeling based on control flow feature
   - Vulnerability feature modeling based on data flow feature
   - Binary program suspected vulnerability and reachable path

3. Generation technology of binary vulnerability mining guide strategy
   - Code path complexity calculation
   - Vulnerability dedication and vulnerability probability assessment
   - Guide strategy generation

Figure 1. Overall research framework
In terms of intelligent pre-treatment of the program information of the binary program, firstly, the study analyzes the program slicing, divides the large programs into code blocks, and then perform various analysis and processing according to different code blocks. Finally, the control flow, data flow and program path information in the program are extracted to prepare for the next analysis.

After extracting the relevant information of the program, the approach to establish a model for evaluating the potential of vulnerabilities is described in order to get the result of the potential vulnerabilities related to the information. The model is trained through data samples of known vulnerabilities with the aim to improve the accuracy of model judgment. The information such as the possibility of vulnerabilities corresponding to the code block is obtained.

The code blocks with high potential vulnerabilities are evaluated by analyzing the basic block complexity, code fragment complexity, code path complexity, and vulnerability existence probability. As a result, a test queue and corresponding reachable path information ranked by the priorities of different vulnerability potential code blocks are generated, which provide a path reference basis for the further discovery of efficient vulnerabilities. The overall idea is shown in Fig.1.

3.2. Pre-treatment Technology of Binary Program Based on Slicing

Due to the shortage of auxiliary information such as variable types, symbol information, program structure, function library, etc., the understanding of binary code is far from the source code program. The experienced researchers who are committed to the cyber security can directly analyze the binary assembly code in combination with existing experience to get the complete structure of the program and explore the vulnerability of the binary program. Nevertheless, with regard to the automated analysis program, the lack of understanding of the program structure will make the subsequent analysis impossible. Therefore, a pre-treatment technology based on slicing is proposed for the automatic analysis of binary programs. By carrying out the pre-treatment of the executable files of the binary programs, the logical structure of the codes of the binary program, such as control flow graph and data flow graph are acquired. Moreover, based on the code logic, the binary program slicing is executed to assist subsequent automated analysis.

Research steps of slice-based binary program pre-treatment technology

Step1: Extract the basic blocks of the binary program and the basic block control flow diagram; extract the program instruction set, program entry point, addressing mode, disassembly code and other information through the analysis of the file format and static disassembly, and extract all the basic binary blocks to construct a control flow graph based on the basic blocks of the program.

When the basic blocks of the program are extracted, the entry point of the program is served as the starting point to recursively track its assembly code, and divide the basic blocks of the current codes once the invoking, branch, back, and jump instructions are given. Each basic block is the consecutive code in space and execution sequence. In the process of extracting basic blocks, each basic block information is a node. The invoking relationship between nodes is represented by directed edges. The point set V and the edge set E which are established on the basic block information of the binary program constitute the basic block control flow graph CFG = (V, E) of the binary program.

Step2: Binary program slicing based on code logic: considering that the basic block of a binary program is just a series of instructions, it cannot effectively reflect the code logic. Hence, it is also necessary to slice the program based on code logic.

Models for building the code logic of common programs are listed: such as if-else logic model, while model, do-while model, switch model, for loop model, etc; collect the compilation results of different compilers for the logic of these codes, extract the features of these code instructions, and construct different code logic models.

In the meantime, since the binary program can only obtain the address of the entry point of the function instead of the effective range of the entire function, it is also required to construct a function recognition model to identify all the basic blocks contained in a complete function. In the aspect of the identification of function basic blocks, recursive traversal starts from the entry basic block of the function, and returns to the previous layer when meeting ret basic block and then continues to traverse
(a function may contain multiple ret basic blocks), After the entire traversal process ends, the obtained basic block and control flow graph subgraph are the basic block information and control flow graph of the entire complete function.

Taking the basic block CFG in 1) as input, the code logic slice and function slice of the binary program are obtained.

Step3 Analysis of the data flow between functions of the binary program: on the basis of the basic block CFG in 1) and the function slice in 2), extract the operations of the basic block internal instructions on registers and memory, establish the relationship between the operands, simplify the operation steps of assembly code; establish the relationship between function input and output through forward and reverse speculation methods.

3.3. Construction of Potential Evaluation Model of Binary Vulnerabilities

Current binary program vulnerability automatic discovery technology is carried out on the basis of covering the program code path as much as possible. However, such kind of action will lead to some problems. For instance, the entire automated vulnerability discovery process will waste a lot of time on the non-vulnerable code path search. Additionally, the path explosion also exists. The occurrence points and the number of paths of vulnerabilities in binary programs usually only account for a very small number of all code paths of the entire program.

Therefore, on the foundation of binary program pre-treatment, the control flow and data flow of the code snippets which are obtained after analysis will carry out vulnerability potential assessment modeling for the library function call relationships (such as function call order, stack space, etc.) and data flow between functions (parameter type, length of string parameter, return value type, length of return value, etc.) based on taint analysis to predict potential vulnerability points and reachable paths of binary programs.

Step1 Sample collection: collect different sample programs and utilization processes of various types of vulnerabilities, and put on the label information, including sample instruction set, sample running ring and protection measures, sample vulnerability types, sample vulnerability points and code paths, etc.

Step2 Data pre-treatment: primarily perform data cleaning and filter out some samples that are too special and not universal.

Step3 Establishment of vulnerabilities feature models: a binary program is usually compiled directly from C / C++ and can be run directly on the target machine. However, the lack of boundary inspection for array access in C / C++ may cause the array access of the program to go out of bounds; or the memory access may go out of bounds because of the failure to use the safe memory operation. In order to improve the writing efficiency, binary programs often call a large number of library functions, which often become the direct causes of vulnerabilities due to lack of parameter checking. The laws of causes of these vulnerabilities can be found in the control flow and data flow, so we can build a vulnerability feature model on this basis.

The vulnerability feature model needs to extract the control flow graph and data flow graph of the sample program, and calculate the control flow feature and data flow feature in the process of reproducing the vulnerability, and generate the corresponding vulnerability feature model.

In terms of the control flow feature, because the direct cause of most vulnerabilities is the improper library function call, the control flow is mainly featured by the library function call sequence, while the data flow feature is the data transfer method, parameters type and parameter length before the function call sequence in the control flow feature.

Multiple vulnerabilities feature models are generated for different types of vulnerabilities and saved in the corresponding vulnerabilities feature library. Furthermore, a vulnerability potential assessment model based on pattern matching is designed.

Step4 Test for vulnerabilities feature models: select the test set samples of each vulnerability type as input 1) the obtained vulnerability potential evaluation model, and perform pattern matching with the constructed vulnerability feature model to get the suspected vulnerability point, vulnerability type, risk
and corresponding code path; calculate the accuracy rate, false positive rate, false negative rate, etc.in accordance with the obtained results and actual verification results; analyze and adjust the vulnerability feature model.

3.4. Generation of guidelines for binary vulnerability discovery
On the account that the reachable paths of the basic blocks where the vulnerability point is located are multiple, and the degree of triggering the vulnerability is different for each code path, it may take a much longer time to discover the vulnerabilities if all the code paths are randomly selected and exploited. Due to the fact that the time period required for vulnerability discovery is often very long, the later the vulnerability is discovered, the greater the loss for commercial software. At the same time, the causes of vulnerabilities of multiple reachable paths at the same vulnerability point are similar in the aspect of the binary program. After the vulnerabilities at the vulnerability points are fixed, the vulnerabilities of other paths will be removed simultaneously. Therefore, it is necessary to prioritize the code paths of suspected vulnerabilities. The preference shall be given to the paths that trigger vulnerabilities easily, bring about more harm, and have a higher probability of vulnerabilities.

(1) Complexity calculation of basic blocks
The complexity of each basic block is measured by the number of instructions and types of instructions. The more the number of instructions contains and the more reading and writing instructions of the memory show, and the higher the complexity of the basic block will present. The complexity of each basic block is calculated by the adoption of these two indicators.

(2) Complexity calculation of code snippets
As for the design of complexity calculation methods of each code snippet type, such as the code snippet of a loop type, the number of loops and the number of basic blocks located in the loop body and the weighted sum of the complexity of the basic blocks are the primary measurement indicators for the code snippets of the loop types. As far as the code snippets of conditional branch type are concerned, the number of branches and the weighted sum of the complexity of the basic blocks of each branch are the vital measurement indicators of their complexity.

(3) Complexity calculation of code paths
The complexity of the entire code paths is calculated by means of the complexity of the code snippets and the complexity of the basic blocks that the code path passes through. The if type of code snippets on the code paths should be granted more weight because it may be the data integrity verification point on the path. The more data integrity verification points a path passes, the more difficult it is to construct the corresponding input to the point of vulnerability, which means the higher the complexity of this path.

(4) Impact of vulnerability risks and vulnerability probability factors
In consideration that different types of vulnerabilities may bring about various degrees of damage, the risk factors of vulnerabilities need to be regarded when prioritizing paths. At the same time, because of the prediction of the suspected vulnerability points, there will be a corresponding prediction probability. The greater the probability of the existence of the vulnerability, the higher the priority of the vulnerability point.

(5) Priority ranking of code paths
The code paths of all suspected vulnerability points are prioritized by taking into account of the factors above. If the code paths have the higher priority, it should be discovered earlier. The paths obtained by prioritization are the guiding strategies for subsequent vulnerability discovery.

4. Conclusion
Taking into account of several disadvantages such as low discovery efficiency and poor accuracy of the binary test vulnerability fuzzy testing discovery method, the paper proposes a binary program analysis method based on program slicing, control flow analysis, data flow analysis, etc. Then, the vulnerability potentials of the binary program are analyzed. Finally, a comprehensive result is achieved by an evaluation model. Therefore, the research conducted in this paper has promoted the traditional vulnerability feature detection to take great changes. For instance, the vulnerability detection only
considers the existence of dangerous functions, but it ignores the shortcomings of the relationship between the vulnerability, the control flow graph and data flow graph. After a program snippet with a large potential for vulnerability is gained, the corresponding control flow is output as the guiding discovery strategy for fuzzy testing.

Acknowledgements
This work was supported by the science and technology project of State Grid Corporation of China: "Research on the Theory and Method of Automatic Exploit for Power Grid Application Vulnerabilities" (Contract Number: SGZJ0000KXJS1900214).

References:
[1] Chess B, McGraw G. Static analysis for security[J]. IEEE Security & Privacy, 2004, 2(6): 76-79.
[2] Newsome J, Song D X. Dynamic Taint Analysis for Automatic Detection, Analysis, and SignatureGeneration of Exploits on Commodity Software[C]/NDSS. 2005, 5: 3-4.
[3] Kemerlis V P, Portokalidis G, Jee K, et al. libdft: Practical dynamic data flow tracking for commodity systems[C]/Acm Sigplan Notices. ACM, 2012, 47(7): 121-132.
[4] Caballero J, Grieco G, Marron M, et al. Undangle: early detection of dangling pointers in use-after-free and double-free vulnerabilities[C]/Proceedings of the 2012 International Symposium on Software Testing and Analysis. ACM, 2012: 133-143.
[5] Kang M G, McCamant S, Poosankam P, et al. Dta++: dynamic taint analysis with targeted control-flow propagation[C]/NDSS. 2011.
[6] YIN Heng, SONG D. Temu: Binary Code Analysis via Whole-system Layered Annotative Execution [EB/OL]. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-3.pdf, 2016-6-11.
[7] Zhu Ge Jianwei, Chen Libo, Tian Fan, et al. Type-based Dynamic Stain Analysis Technology [J]. Journal of Tsinghua University (Natural Science Edition), 2012.
[8] Ma Jinxin, Li Zhoujun, Zhang Tao, et al. Research on Stain Analysis Method Based on Execution Tzrace offline Index [J]. Journal of Software, 2017, 28 (9): 2388-2401.