Research and Development of Binary Code Vulnerability Static Detection Technology

Xianda Zhao, Shuguang Huang, Zulie Pan, Li Yang

College of Electronic Engineering, National University of Defense Technology, Hefei, China

Keywords: binary code; vulnerability; static detection

Abstract: The research on static detection technology of vulnerability based on source code has developed rapidly, and there have been quite a lot of technology accumulation in recent years. For closed source software, program source code cannot be obtained, and the static detection technology based on binary code is not mature at present. This paper focuses on static detection technology of binary code vulnerability and analyzing the advantages and disadvantages of several mainstream binary static detection tools. Finally, according to the key points in the static detection process --- recover binary program information according to disassembler code, the feasible research direction is proposed.

1. Introduction

With the widespread use of computer technology, software plays a key role in Internet information processing. At the same time, the continuous expansion of software volume has led to an increase in software vulnerabilities [1]. In recent years, cybersecurity incidents have gradually become one of the major problems facing the world [2]. In August 2018, the personal information of nearly 130 million users of Huazhu Hotels Group was leaked and received extensive attention from software security researchers.

As for software vulnerability detection technology, most researchers currently use source-oriented vulnerability detection tools. However, in practice, a lot of software exists in the form of binary code. Therefore, the binary-oriented vulnerability detection technology has a strong practical value.

Binary vulnerability detection technology can be divided into static analysis, dynamic analysis and dynamic and static analysis from the perspective of software operation. Because binary programs are easy to implement, researchers generally use dynamic analysis techniques [3], but dynamic analysis can only be detected at runtime, with less automation. In contrast, static analysis technology does not require execution of a program to detect the entire path of the program. Although static analysis technology is more complicated to implement, it is still one of the most effective detection methods[4]. Currently, the reverse analysis of binary programs usually first translates binary code into intermediate representation language. Then develop a design static analysis algorithm to detect program vulnerabilities based on intermediate representation language [5].

For our study, we introduce the development of binary code vulnerability static detection technology. First we discuss two mainstream static detection techniques.Then we introduces the principle and analyzes the advantages and disadvantages of the current mainstream static analysis tools; Finally, we summary the development status of static detection of binary program vulnerability and propose the feasible research direction for binary code vulnerability detection.

2. Binary code static detection technology

Binary code static detection technology can be divided into program structure analysis and program semantic analysis. The detection method based on program structure does not need to restore the execution semantics of the program, but detects the vulnerability by combining the file format features.The static analysis process based on the program semantics combines the program

  DOI: 10.25236/icscbd.2018.024

information to discover the dangerous behavior in the execution semantics of the program. This method restores program information by disassembly. In this section, we introduce two methods that are currently mainstream: pattern-based vulnerability analysis technology and binary code comparison technology.

2.1 Pattern-based vulnerability analysis

The basic idea of binary code vulnerability pattern analysis technology comes from the defect mode checking technology of source code, both of which are located through vulnerability matching or defect model matching and checking [6]. At present, the pattern-based vulnerability mining method is based on the static vulnerability mining method, supplemented by the late dynamic debugging to verify the vulnerability. Generally, in the actual mode analysis, the assembly code is first converted into an intermediate representation [7], and then the related attribute information description is further analyzed accordingly. The pattern-based vulnerability analysis process is shown in Figure 1.

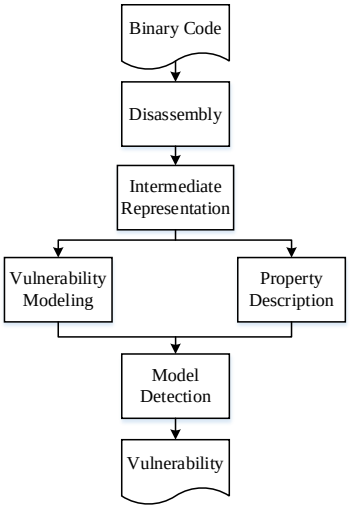

Fig. 1. Pattern-based vulnerability analysis process.

Transforming the assembly code of binary files into efficient intermediate languages is the key to pattern-based vulnerability analysis techniques. As the research focus of the detection process, the reverse intermediate representation should satisfy the following principles:

- Compact instruction set. Using a reduced instruction set can greatly reduce assembly language instruction entries and simplify the analysis process.
- Sufficient register. It can enable the intermediate language to support different processor architectures.
- The addressing method is simple. And the complex addressing modes that are unreadable should be deleted.
- Uniform operand format. Implicit operands can be converted to explicit operands in intermediate language instructions.

The intermediate languages commonly used include the intermediate representation REIL[8] for reverse analysis, the reverse intermediate representation of VEX and Vine. The reverse middle indicates that although the complex assembly language can be transformed into an intermediate language that is easier to analyze, the intermediate representation of the instruction set is still difficult to completely restore the semantic analysis. Therefore, there are still false positives in the actual detection process of the vulnerability.

2.2 Binary code comparison technique

Binary code comparison technology[9] detects the location of the vulnerability by analyzing the differences before and after the software code is patched, which is an effective way to quickly exploit the vulnerability. The binary code comparison technology theory model was created earlier, and some new methods were proposed to improve the comparison effect. In 2004, Halvar Flake first

proposed a structured alignment comparison algorithm [9] to determine whether a function is isomorphic by comparing the control flow graph of a function in a file. In 2011, Cui Baojiang proposed a binary file homology comparison algorithm based on the basic block signature and the jump relationship between basic blocks [10], and implemented the prototype detection tool BinCompare. In 2017, Luo L et al. proposed a binary code similarity comparison method based on the semantic common block of the longest common subsequence [11].

In recent years, artificial intelligence technology has developed rapidly, and some researchers have applied this technology to binary code comparison work. Xu X used a neural network-based method to compare the embedded numerical vector distance to detect similar code [12]; Liu Bingchang et al proposed using the deep neural network (DNN) to extract the internal function features of each binary function, and realized a prototype system αDiff [13].

The general implementation flow of binary code comparison technology is shown in Figure 2.

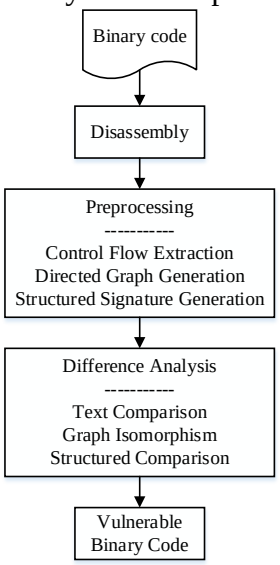

Fig. 2. Binary comparison process.

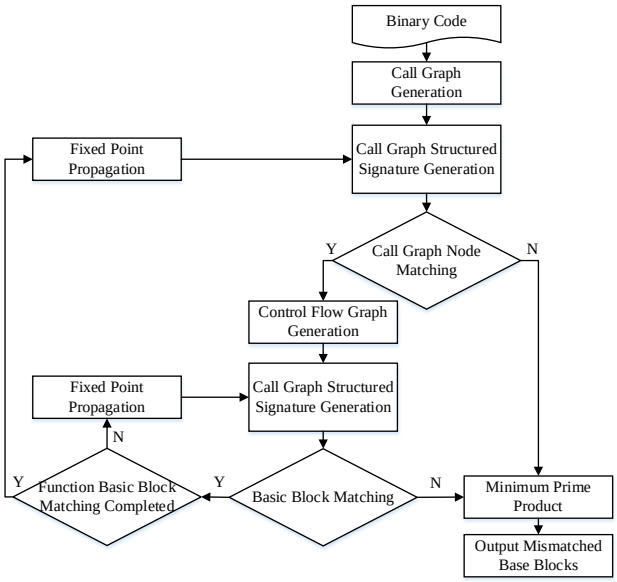

Fig. 3. Structured comparison process.

The focus of binary comparison technology is analysis differences [14], and detection techniques are mainly divided into the following four types:

- Text-based comparison. Text-based comparison is the easiest way to compare two files and disassemble code. However, the vulnerability positioning accuracy is poor and the false positive rate is high.

- Based on the isomorphism of the graph. This technique first abstracts the control flow graph of the binary program, converts the binary program into a directed graph, and then uses the graph theory to solve it.
- Based on structured comparison.This technique focuses on the logical structure changes of the binary file, using the function call graph of the binary program and the control flow graph (CFG) of the specific function. In general, the structured comparison process is shown in Figure 3.

3. Static detection tools

3.1 BinDiff

BinDiff [15] is a well-known binary file comparison tool used to compare the differences in assembly code. BinDiff works on an abstract structure layer of the executable program, enabling multi-platform compatibility. And it is divided into the following comparison strategies:

- Regular comparison. BinDiff developed a function attribute table for regular alignment, calculate the first attribute of each function in the global level.
- Function matching. Function properties are used in two ways. If the attributes of the source and target functions match, then try to match side by side. And BinDiff uses a variety of function matching algorithms, such as hash matching and name hash matching.
- Basic block matching. The basic block matching at the data flow graph level is similar in function matching.

Using BinDiff to view two executable files. The result of the difference between functions is shown in Figure 4.

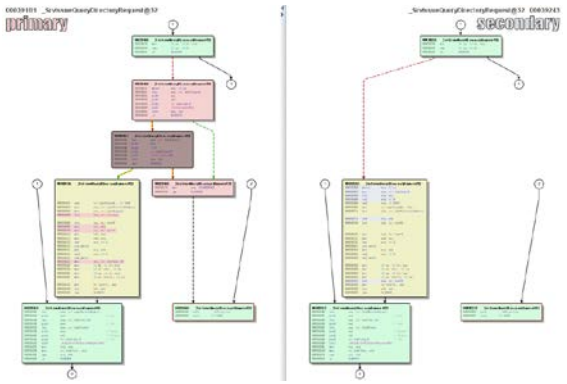

Fig. 4. BinDiff function call comparison.

3.2 BinNavi

BinNavi [16] is a binary code reverse analysis platform developed by Zynamics for assisting vulnerability mining. It allows the user to inspect, manipulate, edit, and annotate control flow graphs of disassembled code and executables, and can locate vulnerabilities by matching vulnerability models.

BinNavi is currently the popular reverse-engineering binary analysis IDE based on graphical visualization. It has the following functions in reverse analysis:

- Open database format. BinNavi's data is stored in the MySQL database, the format is convenient and flexible.
- Integrated Python interpreter. BinNavi allows access to the entire disassembly, Callgraph, and Flowgraph structure.
- Support platform (for debugging): Win32/x86, Linux/x86 (ptrace).It also can be debugged and experimented in WinCE/ARM.

The BinNavi analysis platform can be used to assist in detecting 0day vulnerabilities of non-open source software, as well as analyzing code fragments of existing software vulnerabilities or

malicious attacks. The working process is shown in figure 5.

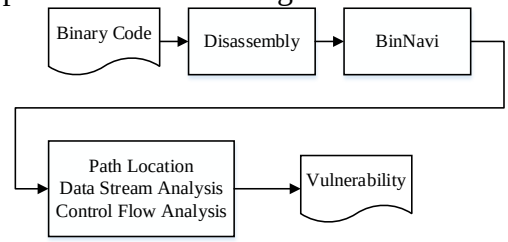

Fig. 5. BinNavi working process.

3.3 BAP

The BAP (Binary Analysis Platform) platform [17] led by Professor Daivd Brumley of Carnegie Mellon University is a powerful open source binary static reverse analysis platform. BAP first disassembles the binary executable, converts the assembly code into an intermediate language BIL, and then performs program analysis. BAP is versatile and has the following features:

- By using an intermediate language, BAP can avoid the difficulty of program analysis directly in assembly language.
- Support for generic code representation, including control flow graph code representation, static single assignment, and program dependency graph.
- Support for Dijkstra's weakest precondition [18] for program verification, as well as multiple constraint solver (SMT) interface calls.

Nowadays, BAP has been widely used in security areas such as automatic generation of code for buffer overflow vulnerability. The overall architecture is shown in Figure 6.

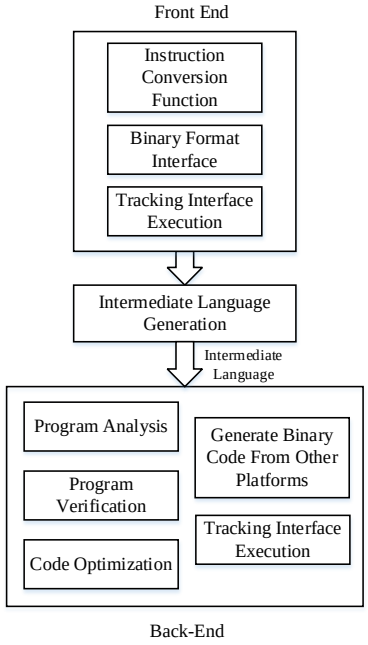

Fig. 6. BAP Overall architecture.

BAP tools are currently used in many binary code analysis and verification efforts. In addition, BAP can be used to perform binary symbol execution or by finding the location where the input may corrupt the security attributes. But BAP also has its limitations: when an indirect jump occurs, it is not feasible to use the weakest precondition to generate the verification condition.

4. Conclusion

In this paper, we mainly analyze the development of binary code vulnerability static detection technology. For the pattern-based vulnerability analysis technology and binary code comparison

technology, we explain the characteristics and limitations of the corresponding tools. At present, the binary static detection tools developed by researchers still cannot meet the security requirements of the program. No tool is perfect, and each tool has certain limitations. Binary code comparison technology can help the utilization of 1day vulnerability but it is difficult to detect 0day vulnerability. In the pattern-based vulnerability detection technology, how to recover the program information according to the disassembled code of the binary program is an important issue that needs to be solved urgently. Based on the current development status of static detection of binary code at present, there are several possible research ideas in the future:

- Transform the disassembly code of the binary into a more general intermediate representation language.
- The combination of static analysis and dynamic analysis of binary programs complement each other.
- Combined with the existing technology accumulation, it is integrated into the same framework to construct a comprehensive binary code vulnerability detection platform.
- Use machine learning to assist with vulnerability detection.

References

[1] Weber S, Karger P A, Paradkar A. A Software Flaw Taxonomy: Aiming Tools at Security[C]//Proc. of ACM Software Engineering for Secure Systems——Building Trustworthy Applications. Louis, Missouri, USA: [s. n.], 2005.

[2] Ghaffarian S M, Shahriari H R. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey[J]. Acm Computing Surveys, 2017, 50(4):1-36.

[3] Choi Y H, Park M W, Eom J H, et al. Dynamic binary analyzer for scanning vulnerabilities with taint analysis[J]. Multimedia Tools & Applications, 2015, 74(7):2301-2320.

[4] Djoudi A, Bardin S. BINSEC: Binary Code Analysis with Low-Level Regions[M]// Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2015:212-217.

[5] Kirchner K, Rosenthaler S. bin2llvm: Analysis of Binary Programs Using LLVM Intermediate Representation[C]// International Conference on Availability, Reliability and Security. ACM, 2017:1-7.

[6] Shizhong Wu. Software vulnerability analysis techniques [M]. Science press, 2014,pp.280-283

[7] Jeong J, Son Y, Oh S. The x86/64 Binary Code to Smart Intermediate Language Translation for Software Weakness[C]// the International Conference. 2017:129-134.

[8] Dullien T, Porst S. REIL: A platform-independent intermediate representation of disassembled code for static code analysis[J]. Cansecwest, 2009.

[9] Flake H. Structural Comparison of Executable Objects[J]. Dimva, 2004:161--173.

CUI Baojiang,MA Ding,HAO Yongle,WANG Jianxin. Comparison of executable objects based on block signatures and jump relations[J]. Journal of Tsinghua University (Science and Technology), 2011(10):1351-1356.

[10] Luo L, Ming J, Wu D, et al. Semantics-Based Obfuscation-Resilient Binary Code Similarity Comparison with Applications to Software and Algorithm Plagiarism Detection[J]. IEEE Transactions on Software Engineering, 2017, PP(99):1-1.

[11] Xu X, Liu C, Feng Q, et al. Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection[J]. 2017.

[12] Liu B, Huo W, Zhang C, et al. αDiff: cross-version binary code similarity detection with DNN[C]//Proceedings of the 33rd ACM/IEEE International Conference on Automated Software

Engineering. ACM, 2018: 667-678.

[13] XIONG Hao,YAN Haihua,GUO Tao,et al. Code Similarity Detection:A Survey[J]. Computer Science, 2010, 37(8):9-14.

[14] Zynamics BinDiff[EB/OL]. https://www.zynamics.com/bindiff.html, 2017.

[15] Zynamics BinNavi[EB/OL]. https://www.zynamics.com/binnavi.html, 2017.

[16] Brumley D, Jager I, Avgerinos T, et al. BAP: A Binary Analysis Platform[C]// International Conference on Computer Aided Verification. Springer-Verlag, 2011:463-469.

[17] Dijkstra E W, Scholten C S. Predicate Calculus and Program Semantics[M]. World Pub. Corp, 1991.