METHODS AND ALGORITHMS FOR PROTECTING EXECUTABLE PROGRAM CODE FROM DYNAMIC AND STATIC ANALYSIS

Sh.S.Musayev
Cybersecurity Center" state unitary enterprise, Uzbekistan
R.B.Quryozov
Master's degree, Faculty of Cyber-Security,Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Uzbekistan

Abstract. *The basic idea of the article is that techniques of dynamic and static analysis must be used in combination with each other to increase the effectiveness of binary code analysis. In the article authors make contributions in binary code decompilation and dynamic execution analysis techniques. The results are applied to the problem of software protection against unauthorized reverse engineering. Authors used analysis of the basic program control flow algorithm to obfuscate the program and protect it against research*

Keywords. *Dynamic, static analysis, control flow, software protection, obfuscation.*

Introduction. Software companies faces serious risks associated with software security because the process of software development always contains some errors and mistakes. In turn, these errors can create favorable conditions for emergence serious vulnerabilities in software. Vulnerabilities exploitation affects not only the system security that uses vulnerable product but also the company competitive advantages engaged in the software development. It should be noted that the process of vulnerabilities research is greatly complicated, because in this process there is number of fundamental technological problems. Also, in practice any attack technique on software security is based on preliminary disassembly and analysis of binary code protection mechanism. The purpose of this research is to restore protection algorithm, to identify its weaknesses and undocumented features for their future modification and (or) to automate the process of attack . To resolve these problems techniques of static and dynamic binary analysis are widely used. In this articles authors demonstrated techniques of dynamic and static binary analysis for vulnerabilities detection in binary code. Dynamic analysis is based on the execution of the program on the CPU. In turn, the static analysis is based on the analysis PE-module technique in memory. Dynamic analysis technique is used to restore software execution flow. Authors used static binary analysis technique to detect vulnerabilities patterns in PE (portable executable) modules. Authors demonstrated that binary execution flow obfuscation was effective for software security against unauthorized research. Authors use dynamic binary execution techniques for software obfuscation. In turn, author uses a static analyzer for analysis of execution and data flow algorithms. The paper introduces original techniques of dynamic and static analysis and also optimization of existing ones. We used a combination of static and dynamic analysis techniques to maximize the effectiveness of protection and vulnerabilities detection. To implement the static analyzer, we propose to use our previous works decompilation and intermediate language (IL) implementation . Thus, in this article we make the following contributions:

1. IL analysis technique. We introduce a new scheme for static binary analysis — that allows us to search vulnerabilities in the intermediate language (IL).

2. Dynamic execution analysis technique. The paper provides optimization algorithms for dynamic binary code execution.

3. Hybrid analysis technique. Technique combines static and dynamic algorithms for software protection against unauthorized research.

Materials. To implement the technique of static analysis, binary code must be interpreted into the intermediate language. In our previous work, we have formulated and described this IL. To translate x86 mnemonics in the IL we used a basis which consisted of 16 basic operations. This notation allows us to describe the code in resource - oriented. Registry resource is denoted by bracers which entered number of resources. Memory resource is indicated by brackets which entered resource address. Number of such resources is taken out of brackets. The result is separated by «=». The operations and constants are written «as is».

Methods. It should be noted that static analysis is a resource-intensive process because research is performed in whole PE module of analyzed software. Dynamic analysis is typically more precise than static analysis because it works with real values in the runtime mode. For the same reason, dynamic analyses are often much simpler than static analyses.

Results. These two approaches (static and dynamic) are complementary. The combination of techniques is used because the static analysis does not allow analyzing the run-time values of the program, so-called problem: «What You See Is Not What You eXecute». In case of dynamic analysis it is not possible to analyze all possible execution paths and states of the program. Thus, in this paper we use a combination of the methods described above, the binary code sequence is tested by static and dynamic analyzer. It should be noted that the main problem in the implementation of the technologies described above is performance. During the static analysis the main performance problem that it is necessary to analyze the whole program code but in the case of large programs it is problematic. The process of dynamic analysis has also serious problems with performance because the technique used for sequential analysis of each instruction in the software. Let's consider this problem in details. To perform dataflow analysis an effective analyzer must collect as much data as possible, it should perform a single step of a certain execution path and save registers values in each step of software execution. There are several methods to perform single step in executable module. It should be noted that the described technologies have been implemented as the software tools.

1. Win 32 API debugging through «official» interfaces. To do this, we must run automatic debugger, forcing singlestepping by settings TF bit in the executive context and collecting information about registers, memory and flags. The Intel x86 processor uses complex instruction set computer (CISC) architecture, which means there is a modest number of special-purpose registers instead of large quantities of generalpurpose registers.

If TF flag set in TRUE, the processor will raise a STATUS_SINGLE_STEP exception after the execution of one instruction. However, it is slow technique, because a context switch after each instruction and the debugger needs to retrieve context and resume execution.

2. Dynamic binary instrumentation (DBI) technique. Binary instrumentation is technique that modifies a binary program, either pre-execution or during execution to observe, monitors and modify a binary program. As described, at the first stage launcher loads test application, injects DBI dll with instrumentation code, calculates entry point of test application, injects jump instruction to DBI dll and starts the program. After jump, DBI code performs some analysis and inject second jump in second instruction. This allows doing efficient analysis in the context of the process with the ability to dynamically modify the binary code. To compare techniques of debugging and DBI a simple program was written which runs a loop given number of times to collect some benchmarks. The techniques of static and dynamic analysis through DBI were tested in the task of binary code protection. In this technique, protector injects additional operations and instruction in the protected software.

It should be noted that static analysis is a resource-intensive process because research is performed in whole PE module of analyzed software. Dynamic analysis is typically more precise than static analysis because it works with real values in the runtime mode. For the same reason, dynamic analyses are often much simpler than static analyses. Describing the dynamic binary analysis technique let's consider the concept of a linear plot of binary code. The section of binary code is linear when the section doesn't contain any instructions of control transfer to another section of the binary code. Instructions, which are nonlinear.

The basic approach described in this paper is to provide information about program execution such as: functions calls, resources which were used on linear plots. This approach allows us to execute some blocks of the program and perform control under CPU registers, flags and memory after each executed linear block. The results of this execution allow us to track the control and data flow of the program and describe the sequence of the program steps with saving result of each of them.

REFERENCES:

1. Shudrak M. Lubkin I. «The method and code protection technique against unathorized analyze». «Software and systems» magazine, Tver, vol. 4. 2022.

2. Sang Kil Cha, Thanassis Averginos, Alexandre Rebert and David Brumley «Unleashing Mayhem on Binary Code» in Proc. of the 2020 IEEE Symposium on Security and Privacy.

3. Zhi Liu; Xiaosong Zhang; Xiongda Li; «Proactive Vulnerability Finding via Information Flow Tracking» Multimedia Information Networking and Security (MINES), 2020 International Conference on , vol., no., pp.481-485.

4. Marco Cova; Viktoria Felmetsger; Greg Banks; Giovanni Vigna; "Static Detection of Vulnerabilities in x86 Executables, "Computer Security Applications Conference, 2019. ACSAC '06. 22nd Annual, vol., no., pp.269-278.

5. Darwish, S.M.; Guirguis, S.K.; Zalat, M.S.; «Stealthy code obfuscation technique for software security» Computer Engineering and Systems (ICCES), 2020 International Conference on., pp.93-99.
6. Haibo Chen; Liwei Yuan; Xi Wu; Binyu Zang; Bo Huang; Pen-chung Yew; «Control flow obfuscation with information flow tracking» Microarchitecture, 2019. MICRO-42. 42nd Annual IEEE/ACM International Symposium on , vol., no., pp.391-400.