http://www.moderntechno.de/index.php/meit/article/view/meit26-01-001
DOI: 10.30890/2567-5273.2023-26-01-001

UDC 004.21

PROTECTION OF SOFTWARE SYSTEMS FROM VIOLATION OF PROGRAM CODE INTEGRITY

Krykunov D.
Ph. D., student
ORCID: 0000-0003-0475-8081
Cherkasy State Technological University,
460 Shevchenko Blvd., Cherkasy, 18006, Ukraine

Abstract. *Protection of program code from software modification is currently a very relevant area of science, since after the imposition of sanctions on the aggressor countries, the purchase and use of licensed software has decreased and the use of unlicensed software has increased. Despite the large number of methods for protecting software against code modification, they are ineffective, namely, the large number of methods for partially or completely disabling this protection. Due to sanctions or other factors, a person who does not have a current software license should not be able to use or edit the program code in order to provide access to use the software.*

The purpose of this article is to collect and analyze the available information in the field of program code protection against modification in order to show the level of research of the scientific problem, to give a critical assessment and conclusions based on published works and to formulate the purpose of further development.

Key words: *software protection, program code modification, encryption, steganography, hashing, obfuscation.*

Introduction

Software protection is a set of measures aimed at protecting software from unauthorized acquisition and use, modification and study of program code and reproduction of analogues.

The development and support of algorithms for protecting software systems from violating the integrity of the program code is a very progressive area of software engineering. Software protection against program code modification protects software from hacking. With the help of protection methods, it is possible to complicate modification and detect modified software. Programs that require program code protection include operating systems, networking and communication programs, text and graphic editors, game modules, compilers, and other software. Currently, there are no software protection methods that can fully protect them.

Software protection against modification of program code can have several purposes:

- *Prevent unauthorized changes*. Protecting your application code from modification can help prevent unauthorized changes that could lead to security breaches, incompatibilities, and other problems. For example, an attacker could modify the program code to inject malicious code or steal confidential information.
- *Maintaining program integrity*. Protecting program code from modification can also help preserve program integrity. If the program code is changed without taking into account all aspects of its operation, it can lead to crashes, errors, and other problems.

– *Compliance with legal requirements*. In some industries, such as financial and healthcare, there are legal requirements related to protecting confidential information and maintaining software integrity. Protecting application code from modification can help organizations comply with these requirements.
– *Protection of intellectual property*. Protecting program code from modification can help preserve intellectual property by preventing copying of the program or changing its functionality without the permission of the copyright holder, and can also be used to obtain guarantees for the use of licensed software, namely, to prevent the circumvention of license verification in the software.

Especially now, during the imposition of sanctions against aggressor countries, the area of software protection against code modification is very relevant, because many users from these countries use software without an up-to-date license, thus generating profits and paying taxes to the aggressor country, which go to the military sphere and are used for military aggressions against peaceful countries. It is necessary to conduct research in this direction in order to complicate the economic situation in the countries on which sanctions are imposed.

In general, protecting software from modification of the program code is an important measure to ensure security, intellectual property protection, and software integrity.

Protection mechanisms that can effectively protect software running in untrusted environments should have the following properties:

1. *Resilience*. The protection has no single points of failure and is hard to disable.
2. *Self-defense*. Able to detect and take actions against tampering.
3. *Configurability.* Protection is customizable and can be made as strong as one needs.
4. *White-box security*. Because any scheme for protection is likely to become publicly known over time, its strength should not be based on its secrecy but rather on the knowledge of a secret key used at protection-install time (but not stored anywhere within the protected program). [1]

Security systems should perform such functions as:

✓ Identification of protected resources.
✓ Authentication of protected resources, i.e. establishing their truth based on comparison with a reference.
✓ Define access rights to protected resources.
✓ Registration of user login to the system and registration of exit from it.
✓ Registering reactions to access rights violations.
✓ Processing of registration logs.
✓ Control of integrity and operability. Control and delimitation can be performed on the basis of tables. [2]

Collection and analysis of available information in the field of program code protection from modification

In order to analyze the information that will help to complicate and make it impossible to edit software worldwide, and especially in the aggressor country, and to understand the study of this issue on their side, it is also necessary to collect information from scientific papers that have been presented in this country.

Scientific problems in the field of software protection against code modification are solved with the help of such protection methods as encryption, steganography, hashing, compression, obfuscation, remote monitoring, regular updates.

1. Encryption of a program is used to prevent disassembly and modification of the program code. This method complicates the work of a reverse engineer, but the written decryptor returns a simplified program code that can be used to modify the program. [3]
2. Steganography disguises the key information needed to decrypt the code as the executable program code. [3] The advantage of steganography is that the information does not attract attention. It can be implemented by using various methods, such as changing bytes of program code, implementing secret messages in metadata, using unused bytes, etc. [4]
3. Hashing calculates check hashes and compares them with the reference hashes that are embedded in the software. [3]
4. Compressing sections of the code being executed, or the entire code, transforms the output to a different form. The advantages of this method include the fact that it makes analysis more difficult and reduces the size of the program, but any compression functions can be identified, especially if they are popular functions such as UPX or ASPack. [3]
5. Remote monitoring involves the program sending information about its work to a remote server where it can be analyzed.
6. Regular software updates can prevent vulnerabilities from being exploited to modify the code.
7. Code obfuscation transforms the program code in such a way that it becomes difficult to understand or incomprehensible to the reverse engineer. This may include changing the names of variables, functions, and classes, removing comments and spaces, rearranging code, and using other techniques to make it difficult to understand the program structure. Code obfuscation can reduce the ability of attackers to analyze the program and conduct attacks. This method of protection is one of the most popular methods. [5] There are the following types of obfuscation:
   - Lexical obfuscation. It involves changing identifiers, keywords, symbolic constants, and other elements of the programming language to make it harder for a reverse engineer to understand the code.
   - Data obfuscation. Changing data structures, variable declaration sequences, and transforming data stores.
   - Preventive obfuscation. This type of obfuscation is designed to prevent the successful application of deobfuscators to the code of a software product. It is aimed at exploiting the shortcomings of deobfuscation tools often used in software. [6]
   - Obfuscation of string literals. The method is based on the peculiarities of the pseudo-random number generator: with the same initial seed value, the sequence of numbers is always the same. The result of the method is an array of numbers that reflects the array and, due to the symmetry of the algorithm, can be converted back to a string. [7]

- Obfuscation of logical expressions. Use of logical expressions that have been simplified or collapsed. [7]

Some of these methods can be used in conjunction to create a more robust software protection against code modification.

Analyzing the existing analogues, it is possible to give a critical assessment of each method of protection:

1. Cryptographic methods of protection in client software are not reliable because the program must be run from the source code, and for this source code to appear, a program or module that decrypts the code must go with this program. If the reverse engineer has this software or module, he has full access to the program code of the software that is the target of the hack.
2. Stenographic methods of protection are ineffective, as integrated development environments can refer to places in the software code where information is hidden. Also, software steganography can be used by attackers to secretly transmit malicious code.
3. The method of comparing checksums has a good basis, but a reverse engineer can also calculate checksums and replace them in the program code, or remove the checksum comparison module altogether.
4. Using separate programs for comparing checksums is also not a reliable solution, because the individual software must also be protected from modification of the program code.
5. Remote monitoring is a good method, but a reverse engineer can intercept messages from unmodified software to a remote server, modify the program code, and send fake messages back to the remote server to show that the software has not been modified.
6. Regular updates will not give a good result, as the reverse engineer will be able to make the software automatically modify modules in new versions of the software being modified.
7. Obfuscation methods can only make the software code more complex, but it is possible to see the logical structures in it. So, a reverse engineer only needs one successful deobfuscation to get the first version of the program. In all subsequent versions of the software, the reverse engineer can use the code from the previous version of the program, replacing the same logical structures, and perform deobfuscation on new, unprocessed parts of the program code.
8. Lexical obfuscation is not a reliable method of protecting program code because it can be bypassed by reverse engineering the source code. In addition, lexical obfuscation can complicate the process of debugging and testing the program code.
9. The obfuscation of logical expressions consists in simplifying these expressions, which contradicts the term obfuscation and the principles of program code security, and if logical operators are complicated, it can affect the running time of the program code.
10. The use of a pseudo-random number generator for obfuscation using seed makes sense only if the pseudo-random number generator is located in the obfuscation and deobfuscation program, and not in the program library, because the version

of the program library can change, and therefore the logic of the pseudo-random number generator can change. It is known that random number generators in the Java programming language and the C# programming language differ in their operation and return different results under the same conditions.

The method of further development should be a set of algorithms built into protected programs that will protect the software from modification of the program code at different stages and states of the software operation, namely:

- Protecting software with client-side architecture.
- Protection of software with client-server architecture.
- Protecting the data transmission channel in client-server applications.

The quality of future algorithms can be assessed by using probability theory formulas, and the written software with these algorithms can be evaluated by metric analysis of software quality.

Probability theory can be used to compare data that will show the chance with which a reverse engineer will be able to edit software without protection, with existing protection methods, and with new protection methods.

Evaluating software quality using the results of metric analysis means that, based on the quality indicators, the values of the relevant quality metrics and the value of the complex quality indicator in the future software are calculated.

Metric analysis helps to solve such practical problems as:

1. Predicting the number of errors in software from the beginning of design.
2. Predicting the level of software complexity and its maintenance based on the analysis of design results.
3. Predicting the level of complexity of testing processes and the number of undetected errors based on the analysis of the program code.
4. Predicting the final size of the program code based on the analysis of software architecture design complexity estimates.
5. Determining the impact of certain characteristics of the program code on the quality of the finished software.
6. Control of the stages of software implementation.
7. Analysis of obvious and hidden defects in the finished software.
8. Identification of the best software development methods and technologies based on their comparison.
9. It helps to identify and fix software problems, improve efficiency, and reduce the risk of errors. [8]

Summary and conclusions

The analysis of existing methods of solving the problems of protecting programs from program code modification has shown that each method has a sufficient number of shortcomings that will negatively affect the success of software protection against disassembly and modification of program code, and no method can provide complete protection against such malicious actions.

Since each protection method does not fully protect the program, but only slows down the reverse engineer's work, we can conclude that disassembling and modifying the program code makes sense if it is commercially feasible and if there is time to obtain information, i.e., the main purpose of protection methods is operations that

make it inexpedient, or rather economically unprofitable, for an attacker to act.

It can also be concluded that the vast majority of protection methods use various types of obfuscation, from which you can always get unobfuscated code using self-written deobfuscators or existing ones and then edit the software.

The method of further development will be a set of algorithms for protecting program code at different stages and states of software operation.

The quality of further development should be assessed with the help of:

- Probability theory formulas to compare the chances of a reverse engineer being able to edit the software.
- Metric analysis of software quality.

As a result of the work done to search and analyze the available information in the field of protecting program code from modification, we can conclude that this area is not sufficiently developed and has no results that could completely make the work of a reverse engineer impossible, and the number of programs being created is increasing every year. These factors make this area very progressive nowadays, and further research work is needed in this area.

References:

1. Chang, H. and Atallah, M. (2002). Protecting Software Code by Guards. Springer Link. DOI: 10.1007/3-540-47870-1_10

2. Kaplun, V., Dmytryshyn, O. and Baryshev Y. (2014). Protection of the software. Institutional repository of Vinnytsia National Technical University. Retrieved from https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/14257/Kaplun-6678619f16033b998a0c233b1e652488.pdf?sequence=1&isAllowed=y

3. Butyn, A. (2018). Methodological aspects of software protection systems development. Cyberleninka. Retrieved from https://cyberleninka.ru/article/n/metodicheskie-aspekty-razrabotki-sistem-zaschity-programmnogo-obespecheniya/viewer

4. Herasymova, I. (2017). Obfuscation as one of the methods of protecting program code. Bauman Moscow State Technical University Kaluga. Retrieved from https://conference.bmstu-kaluga.ru/uploads/userfiles/april_2017_tom_3.pdf#page=98

5. Behera, C. and Lalitha Bhaskari, D. (2015). Different Obfuscation Techniques for Code Protection. Science Direct. DOI: 10.1016/j.procs.2015.10.114

6. Bondarchuk, A., Kornaga, Y., Bazaliy, M., Serhiyenko, P. and Ilin, O. (2020). A method of protecting program code from analysis by obfuscation. Telecommunications and Information Technologies. DOI: 10.31673/2412-4338.2020.045051

7. Davydov, V. (2021). Models and methods of increasing the bytecode-oriented software security of during cyberattacks. Repository of Cherkasy State Technological University. Retrieved from https://er.chdtu.edu.ua/handle/ChSTU/2577

8. Hrytsiuk, Y. and Andrushchakevych, O. (2018). A tool for determining the quality of software by methods of metric analysis. Scientific Bulletin of UNFU. DOI: 10.15421/40280631