Information systems. Computer sciences. Issues of information security
Информационные системы. Информатика. Проблемы информационной безопасности

UDC 004.042
https://doi.org/10.32362/2500-316X-2024-12-2-7-15

RESEARCH ARTICLE

Methods for analyzing the impact of software changes on objective functions and safety functions

Alexander A. Legkodumov [1, @], Boris N. Kozeyev [2, @], Vladimir V. Belikov [3], Andrey V. Korolkov [3]

[1] *SFB Laboratory, Moscow, 127083 Russia*
[2] *ALFA-BANK, Moscow, 107078 Russia*
[3] *MIREA – Russian Technological University, Moscow, 119454 Russia*
[@] *Corresponding author, e-mail: studkkso0416@mail.ru, kozeev.boris2018@yandex.ru*

Abstract

Objectives. This paper examines the various approaches to analyzing the impact of software changes, and suggests a new method using function control flows. Impact analysis of software change can require the investment of a lot of time and competence on the part of the expert conducting it. There is no detailed description of methodology for analyzing the impact of changes and it is not established at a legislative level. The proposed method has three aims: reducing the level of requirements for an expert when conducting software research; localizing code areas to establish defects in information protection functions; and reducing the time spent on analyzing the impact of changes.

Methods. The study analyzes the common methods for analyzing software changes with a description of their positive and negative sides. The possibility of analyzing changes in the control flow of software functions is considered as an alternative to line-by-line comparison of the full volume of source codes. Represented as tree-shaped graphs, the control flows of different versions of the same software are subject to a merging procedure. The final result is analyzed by an expert from the research organization.

Results. The research results of the software change analysis methods are presented with a description of their disadvantages. A description is given of the method for change analysis using function control. This complements existing methods, while eliminating their disadvantages. The study also analyzes the possibility of using this method beyond the tasks defined in the introduction.

Conclusions. The use of methods to localize the most vulnerable code sections is considered one of the most promising areas for analyzing change impact. In addition to searching for vulnerable code sections, it is important to evaluate the effectiveness of the control flow comparison method in the analysis of source code when transferred to another code base.

Keywords: static analysis, cryptographic protection tool, change impact analysis, graph merging, program code analysis

• **Submitted:** 02.08.2023 • **Revised:** 25.09.2023 • **Accepted:** 05.02.2024

For citation: Legkodumov A.A., Kozeyev B.N., Belikov V.V., Korolkov A.V. Methods for analyzing the impact of software changes on objective functions and safety functions. *Russ. Technol. J*. 2024;12(2):7–15. https://doi.org/10.32362/2500-316X-2024-12-2-7-15

Financial disclosure: The authors have no a financial or property interest in any material or method mentioned.

The authors declare no conflicts of interest.

НАУЧНАЯ СТАТЬЯ

Методы анализа влияния изменений программного обеспечения на целевые функции и функции безопасности

А.А. Легкодумов [1, @],
Б.Н. Козеев [2, @],
В.В. Беликов [3],
А.В. Корольков [3]

[1] *СФБ Лаборатория, Москва, 127083 Россия*
[2] *АЛЬФА-БАНК, Москва, 107078 Россия*
[3] *МИРЭА – Российский технологический университет, Москва, 119454 Россия*
[@] *Автор для переписки, e-mail: studkkso0416@mail.ru, kozeev.boris2018@yandex.ru*

Резюме

Цели. В статье рассматриваются различные подходы к выполнению процедуры анализа влияния изменений программного обеспечения (ПО) на его безопасность, а также предложен новый метод проведения процедуры анализа, использующий потоки управления функций. Анализ влияния изменений ПО – достаточно трудоемкая процедура, требующая значительных временных затрат и наличия необходимой компетенции у проводящего ее эксперта. Методика проведения анализа влияния изменений ПО не имеет детального описания и не закреплена на законодательном уровне. Цель предлагаемого метода – снижение уровня требований к эксперту, проводящему исследование ПО; локализация областей кода для исследования на наличие дефектов в функциях, обеспечивающих защиту информации; сокращение времени, затрачиваемого на проведение анализа влияния изменений.

Методы. Проанализированы наиболее распространенные методы анализа изменений: построчное сравнение, система управления версиями, выполнение автоматизированных текстов. Приведено описание положительных и отрицательных сторон методов анализа. Рассмотрена возможность анализа изменений потока управления функциями ПО как альтернатива стандартному построчному сравнению полного объема исходных текстов. После построения потоки управления различных версий одного ПО, представленные в виде древовидных графов, проходят процедуру объединения. Конечный результат анализируется экспертом.

Результаты. Приведены результаты исследования методов анализа изменений ПО с описанием недостатков. Представлено описание метода проведения анализа изменений, использующего поток управления функций, который дополняет существующие методы, устраняя их представленные недостатки. Проанализирована возможность применения данного метода за рамками задач, определенных во введении.

Выводы. Использование методов, локализующих наиболее уязвимые участки кода, выделено как одно из наиболее перспективных направлений для проведения анализа влияния изменений. Помимо поиска уязвимых участков кода, важной является оценка эффективности метода сравнения потоков управления в анализе исходного кода при его переходе на другую кодовую базу.

Ключевые слова: статический анализ, средство криптографической защиты, анализ влияния изменений, объединение графов, анализ программного кода

• **Поступила:** 02.08.2023 • **Доработана:** 25.09.2023 • **Принята к опубликованию:** 05.02.2024

Для цитирования: Легкодумов А.А., Козеев Б.Н., Беликов В.В., Корольков А.В. Методы анализа влияния изменений программного обеспечения на целевые функции и функции безопасности. *Russ. Technol. J.* 2024;12(2):7–15. https://doi.org/10.32362/2500-316X-2024-12-2-7-15

Прозрачность финансовой деятельности: Авторы не имеют финансовой заинтересованности в представленных материалах или методах.

Авторы заявляют об отсутствии конфликта интересов.

INTRODUCTION

Software is deeply integrated into the life of modern man. It is used in medical equipment, cars, banks, airplanes, phones, and many other areas. Much of today's software interacts with the personal data (PD) of its users or may even be installed at a critical facility (CF) allowing interaction with highly valuable data. Compromised CF or theft of PD can result in infrastructure destruction, facility control failure, theft of user resources, and other negative consequences. The security of user data and infrastructure requires embedding certified information security equipment (CISE). This can be specialized software designed to protect private information [1]. In order to protect against new threats, new information security features are constantly being added to the source codes of the information security software, or existing ones are changed. After such a change, special examination of CISE needs to be carried out. The need for such examination is justified by orders of the Federal Service for Technical and Export Control (FSTEC)[1] and the Federal Security Service of Russia.[2]

Developing and maintaining any software is a continuous process wherein the existing functionality is constantly changing. New functions are added, coding styles are updated, optimizations are made, and errors are corrected. Any change made to the software or hardware product (hereinafter referred to as a product) may have an unpredictable impact on a certain part or even on all functions performed by the product. The more changes made to the product, the more difficult it is to trace their impact. In order to detect the changes made, and to establish their nature and quality, a special study called change impact analysis (CIA) [2] is conducted. This is performed, in order to test the software used and to determine the degree of risks associated with any changes made.

Note: in the following text, the expert referred to is an employee of a testing laboratory involved in the CISE change impact analysis.

CIA is a labor-intensive procedure. A significant amount of work is performed manually by the expert and the developer of the protection system. Based on the results of CIA, the aim of the expert is to obtain answers to the following questions:

1. Which program modules and functionalities would be affected by the specific change and how exactly?
2. Would this implementation affect the functionality of the application or individual application modules?

Orders No. 55 of the Federal Security Service of Russia and No. 66 of FSTEC the CISE state that CIA is required after each change. In this way a significant number of changes may accumulate between two versions of the same product, thus increasing the working time of the expert. Thus, the complexity and the amount of resources spent on CIA is additionally affected. Furthermore, a special study

[1] Paragraph 71 of the FSTEC Order No. 55 from April 03, 2018 "Regulation on the Information Protection Equipment Certification System." The paragraph of the Order specifies that the developer of an information protection system should conduct tests of the information protection system involving a testing laboratory (in Russ.). https://fstec.ru/dokumenty/vse-dokumenty/prikazy/prikaz-fstek-rossii-ot-3-aprelya-2018-g-n-55. Accessed May 02, 2023.

[2] Paragraph 41 of the Order of the Federal Security Service of Russia No. 66 dated February 09, 2005 "Regulations on the Development, Production, Implementation, and Operation of Encryption (Cryptographic) Means of Information Protection (Regulations PKZ-2005)." The paragraph specifies that all changes in the design of cryptographic information protection means and their manufacturing technology should be coordinated by the manufacturer of cryptographic information protection means with a specialized organization and the Federal Security Service of Russia (in Russ.). http://pravo.gov.ru/proxy/ips/?docbody=&prevDoc=102900265&backlink=1&nd=102097894&rdk=0. Accessed May 02, 2023.

is required to be conducted by an expert from a third-party organization, unfamiliar with the software and organization of the internal operation of the CISE functions. All the above constitutes the main problem of change analysis [3].

The expert conducting CIA should be highly qualified. They are required to possess:

- an understanding of the software operation;
- knowledge of the programming language in which the product is developed;
- an understanding of the operation of libraries used in the product implementation;
- the ability to search for changed code sections and analyze these sections to evaluate the impact of changes on the product functionality.

The paper considers different approaches to conducting CIA. It presents a method aimed at optimizing the analysis, in order to reduce the level of requirements of the expert and reducing the time needed to obtain meaningful results. When used in this paper, meaningful results are understood to be finding code changes and determining their impact on the CISE quality characteristics.

1. ANALYSIS OF EXISTING CIA METHODS

1.1. Line-by-line comparison

Line-by-line comparison is a method by which an expert performs a line-by-line comparison of different source code versions of the same product, in order to find and evaluate differences. The procedure can be performed using software source code comparison tools such as *Beyond Compare*[3], *Araxis Merge*[4] or the Linux built-in *diff* utility. Source code modification for this method consists of deleting, adding, or changing a line. In most practical cases, the changes found during line-by-line comparison are erroneous insertions which have no effect on functions implemented by the product. Changes which can be classified as erroneous can be divided as follows:

- changing the names of functions or variables;
- removing or adding line breaks;
- deleting or adding comments;
- deleting or adding code lines which do not relate to the functionality implemented by the product (if only not regulated by requirements).

The latter represents idle code sections [4]. This may be code which only participates in developer tests or operates in a certain environment. In addition to erroneous occurrences, this method is relatively ineffective in situations when the product code base switches to another programming language. In this case, any line could be marked as mismatched by automatic analysis tools.

The advantage of this method is that it covers the entire volume of software source code, thus enhancing the possibility of detecting a vulnerability when compared to automated analysis [5]. However, line-by-line analysis is a very costly procedure requiring a lot of expert man-hours along with an in-depth knowledge of the programming language.

1.2. Version control systems

In order to reduce the amount of work performed "manually" by the expert of the research organization, the development company may provide additional materials together with the research materials. These may be, for example, the history of changes generated by a version control system. In modern practice, a version control system is used to optimize the organization of work of several developers on any given product.

This system consists of software to facilitate the management of changing information. This allows changes in the program code to be tracked. It also enables version control, organizing the simultaneous work of several developers, supporting several development directions, and ensuring their interaction. The use of information about source code modifications focuses the attention of the expert and simplifies its understanding.

The advantage of this method is that the change report can be built in automatic mode. There is no need for a separate search of every change by using the line-by-line comparison of two source code versions [6]. However, this advantage contains its main disadvantage. Due to the widespread use of version control systems, the history of changes is available when developing most software. Despite this, the examination can be difficult since the quality of the description of changes depends directly on the employee engaged in describing the changes made during the development. The generated report or change history may provide incomplete or even incorrect information which may hinder a correct CIA.

1.3. Automated tests

Another CIA method implies comparing the test results of two versions of the same software. This method enables checking and confirming that the logic of functions has not changed. This method allows any changes leading to the appearance of defects in the operation of product functions and safety functions to be tracked [7].

[3] https://www.scootersoftware.com/. Accessed May 02, 2023.

[4] https://www.araxis.com/merge/index.en. Accessed May 02, 2023.

The advantages of this method include the possibility of automating testing of a large amount of data. However, any conclusion based on test results cannot be considered reliable, since the developer may add undocumented features to the software. The tests are based on known product features and cannot signal the presence of a software anomaly. Although automated test systems are quite capable of finding errors when changes are implemented incorrectly, the automated test approach may show incomplete or incorrect results in the context of the tests being performed [8]. Depending on the code size being tested and the quality of changes being made, the test set or regression testing, as another approach, lose out to CIA in terms of the amount of resources spent and accuracy [9]. The advantage of CIA is the possibility of testing separate code sections while ignoring the rest of the protection system. Thus, testing modified software using common practices may not be enough to ensure compliance with security requirements. Some code parts may require double checking, deeper analysis, or another approach to testing [10]. The disadvantages of the method also include additional work on creating or modernizing tests for software, if existing functionality has been added or removed. In rare cases, experts of research organizations have to work with incomplete source code, excluding the possibility of conducting CIA using tests.

2. METHOD FOR CONDUCTING CIA

The main problem with CIA is the amount of resources required to conduct it, whether in terms of time required by experts or software knowledge. In order to reduce the time required to analyze test results or carry out line-by-line comparison, the difference in the product logic and the relationships between the functions of the first and second versions of the product may be analyzed [11]. Analysis using the proposed method is divided into the following steps:

- performing static analysis of software to obtain data;
- building the function call sequence based on the data obtained as a result of static analysis [12];
- representing the function call sequence in the form of a tree-shaped graph [13];
- combining the tree-shaped graphs of two versions.

The static analysis of source codes should result in the following data:

- data types used;
- class structures;
- function call sequence [14].

The resulting root graph representing the software control flow shows the first function in the control flow as a root. The graph nodes are other functions which participate in the call chain [15]. In the process of merging the plotted tree-shaped graphs, deleted or added nodes are selected. Based on data obtained from the new graph, the expert can develop hypotheses.

The following hypotheses are selected for testing:

- the node is deleted, implying the functionality is excluded from the software or transferred to another node;
- the node is added, implying a new functionality has been added to the software or it is a transferred functionality from a remote node;
- the order of node calling is changed, meaning changes in the data processing logic have taken place.

We shall consider the following example. Figures 1 and 2 show the sequences of function calls in the form of root graphs G_1 and G_2. The *main* node is the initial call of all other software functions. The func_N node represents any function implemented in the software, where N is a number from 1 to the maximum possible number of function calls.

The result of merging graphs G_1 and G_2 is shown in Fig. 3.

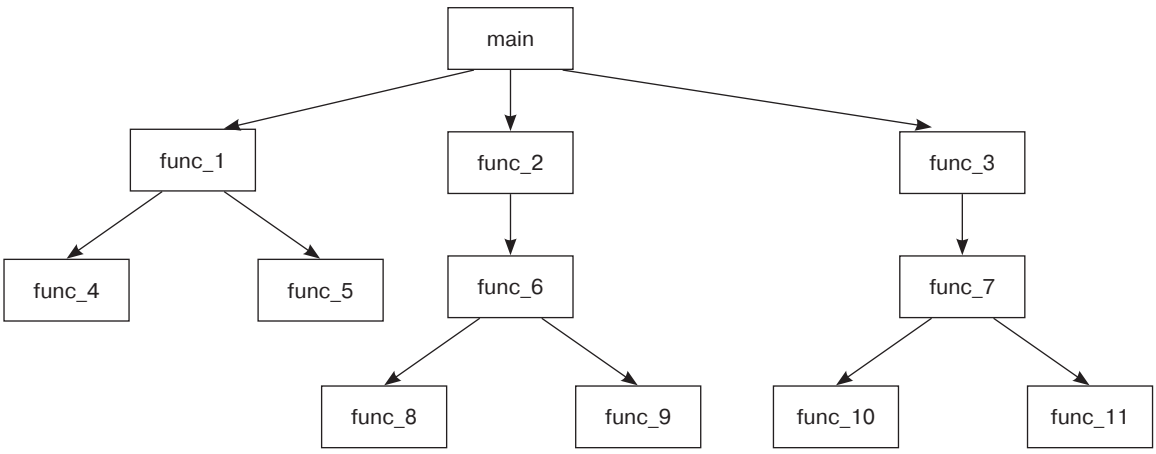

Fig. 1. Flow of function calls for the old software version represented as a tree-shaped graph

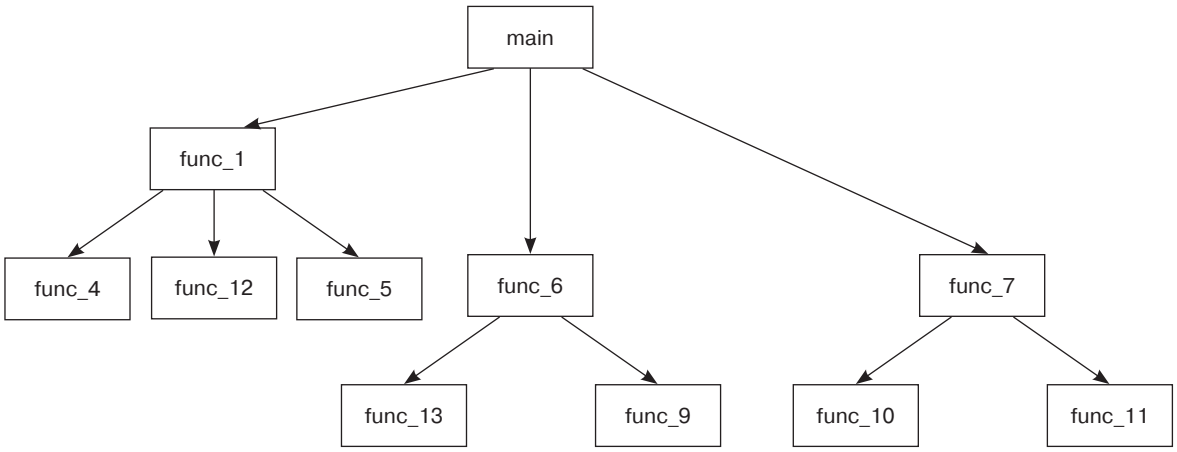

Fig. 2. Flow of function calls for the new software version represented as a tree-shaped graph

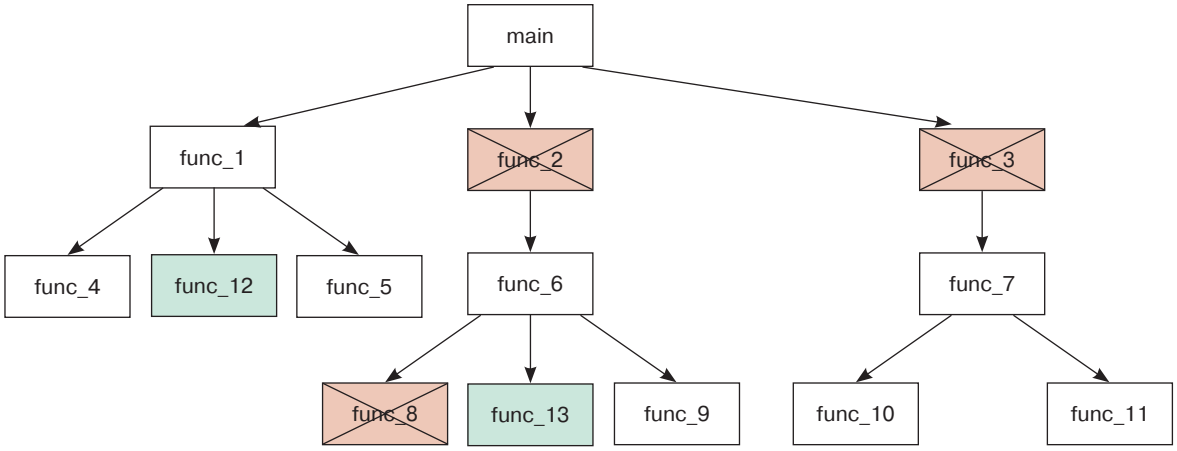

Fig. 3. Result of merging two graphs

Based on the data obtained, the following hypotheses are proposed for testing:

- the func_12 node is added to the left branch. Hypothesis: the node implements new functionality;
- the middle branch has undergone most changes. The func_2 and func_8 nodes are removed. Hypothesis: new func_13 node has either new functionality or implements the capabilities of func_2 and func_8;
- the top func_3 node in the right branch is removed. Hypothesis: the functionality is removed completely or moved to the nodes below it.

Thus, the task is localized and reduced to the task of analyzing versions of specific functions with no direct interaction with the software source code:

- in the left branch, the impact is determined by analyzing operation of func_12;
- in the middle branch, the impact is established by analyzing the result of comparing func_2 and func_8 nodes with func_13 node;
- in the right branch, the impact is established by measuring the significance of the functionality implemented by func_3 and checking the code (using the search program) for the presence of the func_3 implementation in the nodes below it.

CONCLUSIONS

This paper considers the most common methods and tools used to conduct CIA. It also presents the concept of a new method aimed at remedying the disadvantages of the existing methods for studying the impact of software changes on its security. The advantage of the method is in its ability to calculate code sections which have undergone the biggest changes even before the expert interacts with the source code directly.

In addition to its use by experts in research organizations, this method can also be implemented in software development companies, in order to track anomalies in software logic during the development phase.

Further research will focus on using the method in CIA related to transition to a new code base, as well as on improving the accuracy of identifying vulnerable nodes when tainted data is used in the analysis.

Authors' contributions

A.A. Legkodumov—justification of the research concept, development of the research methodology, writing a prototype of the program implementing the method, and writing the text of the article.

B.N. Kozeyev—literature review, analysis and generalization of literature data, conducting research on open source code, and editing the article.

V.V. Belikov—collection and systematization of data, formulation of conclusions, analysis of research results, and editing the article.

A.V. Korolkov—creation of the research model, planning the research, analysis of the research results, and editing the article.

REFERENCES

1. Karpov Yu.G. *Model checking. Verifikatsiya parallel'nykh i raspredelennykh programmnykh system* (*Model checking. Verification of Parallel and Distributed Software Systems*). St. Petersburg: BHV-Petersburg; 2010. 560 p. (in Russ.). ISBN 978-5-9775-0404-1
2. Belikov D.V. The use of static source code analysis in software development and testing. *Studencheskii forum = Student Forum.* 2021;41:90–93 (in Russ.).
3. Belikov D.V. Methods for conducting static analysis of program code. *Studencheskii forum = Student Forum.* 2022;13(192):15–18 (in Russ.).
4. Kazarin O.V., Skiba V.Yu. About one method of verification of settlement programs. *Bezopasnost' informatsionnykh tekhnologii = IT Security* (*Russia*). 1997;3:40–33 (in Russ.).
5. Shchedrin D.A. Application of machine learning methods and analysis of static code of intelligent systems. *Nauchno-issledovatel'skii tsentr "Technical Innovations" = Scientific Journal "Research Center Technical Innovations."* 2023;16:28–32 (in Russ.).
6. Ivannikov V.P., Belevantsev A.A., Borodin A.E., Ignatiev V.N., Zhurikhin D.M., Avetisyan A.I., Leonov M.I. Static analyzer Svace for finding of defects in program source code. *Trudy Instituta sistemnogo programmirovaniya RAN = Proceedings of the Institute for System Programming of the RAS*. 2014;26(1):231–250 (in Russ.). https://doi.org/10.15514/ISPRAS-2014-26(1)-7
7. Viktorov D.S., Samovolina E.V., Mokeeva O.A. The effectiveness of static analysis for finding software defects. *Vestnik Voennoi akademii vozdushno-kosmicheskoi oborony = Bulletin of the Military Academy of Aerospace Defense*. 2021;6:25–39 (in Russ.).
8. Buryakova N.A., Chernov A.V. Classification of partially formalized and formal models and methods of software verification. *Inzhenernyi Vestnik Dona = Eng. J. Don*. 2010;4:129–134 (in Russ.).
9. Efimov A.I. The problem of technological security of software for weapons systems. *Bezopasnost' informatsionnykh tekhnologii = IT Security* (*Russia*). 1994;3–4:22–33 (in Russ.).
10. Efimov A.I., Palchun B.P., Ukhlinov L.M. Methodology for constructing tests for checking technological safety *of programming automation tools based on their functional diagrams. Voprosy zashchity informatsii = Information Security Questions*. 1995;3:30:52–54 (in Russ.).
11. Glukhikh M.I., Itsykson V.M., Tsesko V.A. Using dependencies to improve precision of code analysis. *Aut. Control Comp. Sci*. 2012;46(7):338–344. https://doi.org/10.3103/S0146411612070097
    [Original Russian Text: Glukhikh M.I., Itsykson V.M., Tsesko V.A. Using dependencies to improve precision of code analysis. *Modelirovanie i Analiz Informatsionnykh Sistem*, 2011;18(4):68–79 (in Russ.).]
12. Malikov O.R. Automatic detection of vulnerabilities in the source code of programs. *Izvestiya TRTU*. 2005;4:48–53 (in Russ.).
13. Nesov V.S., Malikov O.R. Using information about linear dependencies to detect vulnerabilities in the source code of programs. *Trudy Instituta sistemnogo programmirovaniya RAN = Proceedings of the Institute for System Programming of the RAS*. 2006;9:51–57 (in Russ.).
14. Vorotnikova T.Yu. Reliable code: static analysis of program code as a means of improving the reliability of software for information systems. *Informatsionnye tekhnologii v UIS = Information Technologies in the UIS*. 2020;2:22–27 (in Russ.).
15. Fritz C., Arzt S., Rashofer S., et al. Highly Precise Taint Analysis for Android Applications. *Technical Report TUD-CS-2013-0113.* EC SPRIDE. May 2013. 14 p. Available from URL: http://www.bodden.de/pubs/TUD-CS-2013-0113.pdf

СПИСОК ЛИТЕРАТУРЫ

1. Карпов Ю.Г. *Model checking. Верификация параллельных и распределенных программных систем*. СПб.: БХВ-Петербург; 2010. 560 с. ISBN 978-5-9775-0404-1
2. Беликов Д.В. Использование статического анализа исходного кода в разработке и тестировании программного обеспечения. *Студенческий форум*. 2021;41:90–93.
3. Беликов Д.В. Методы проведения статического анализа программного кода. *Студенческий форум*. 2022;13(192):15–18.
4. Казарин О.В., Скиба В.Ю. Об одном методе верификации расчетных программ. *Безопасность информационных технологий*. 1997;3:40–33.
5. Щедрин Д.А. Применение методов машинного обучения и анализа статического кода интеллектуальных систем. *Научно-исследовательский центр «Technical Innovations»*. 2023;16:28–32.
6. Иванников В.П., Белеванцев А.А., Бородин А.Е., Игнатьев В.Н., Журихин Д.М., Аветисян А.И., Леонов М.И. Статический анализатор Svace для поиска дефектов в исходном коде программ. *Труды Института системного программирования РАН*. 2014;26(1):231–250. https://doi.org/10.15514/ISPRAS-2014-26(1)-7
7. Викторов Д.С., Самоволина Е.В., Мокеева О.А. Эффективность статического анализа для поиска дефектов программного обеспечения. *Вестник Военной академии воздушно-космической обороны.* 2021;6:25–39.
8. Бурякова Н.А., Чернов А.В. Классификация частично формализованных и формальных моделей и методов верификации программного обеспечения. *Инженерный Вестник Дона*. 2010;4:129–134.
9. Ефимов А.И. Проблема технологической безопасности программного обеспечения систем вооружения. *Безопасность информационных технологий*. 1994;3–4:22–33.
10. Ефимов А.И., Пальчун Б.П., Ухлинов Л.М. Методика построения тестов проверки технологической безопасности инструментальных средств автоматизации программирования на основе их функциональных диаграмм. *Вопросы защиты информации*. 1995;3(30):52–54.
11. Глухих М.И., Ицыксон В.М., Цесько В.А. Использование зависимостей для повышения точности статического анализа программ. *Моделирование и анализ информационных систем*. 2011;18(4):68–79.
12. Маликов О.Р. Автоматическое обнаружение уязвимостей в исходном коде программ. *Известия Таганрогского радиотехнического университета* (*Известия ТРТУ*). 2005;4:48–53.
13. Несов В.С., Маликов О.Р. Использование информации о линейных зависимостях для обнаружения уязвимостей в исходном коде программ. *Труды Института системного программирования РАН*. 2006;9:51–57.
14. Воротникова Т.Ю. Надежный код: статический анализ программного кода как средство повышения надежности программного обеспечения информационных систем. *Информационные технологии в УИС*. 2020;2:22–27.
15. Fritz C., Arzt S., Rashofer S., et al. Highly Precise Taint Analysis for Android Applications. *Technical Report TUD-CS-2013-0113.* EC SPRIDE. May 2013. 14 p. URL: http://www.bodden.de/pubs/TUD-CS-2013-0113.pdf

About the authors

Alexander A. Legkodumov, Cryptographic Analysis Specialist, SFB Laboratory (56/2, Mishina ul., Moscow, 127083 Russia). E-mail: studkkso0416@mail.ru. https://orcid.org/0000-0002-2562-4333

Boris N. Kozeyev, Chief Specialist, ALFA-BANK (27, Kalanchevskaya ul., Moscow, 107078 Russia). E-mail: kozeev.boris2018@yandex.ru. https://orcid.org/0009-0009-0993-8082

Vladimir V. Belikov, Cand. Sci. (Military), Docent, Assistant Professor, Department of Information Security, Institute of Artificial Intelligence, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: belikov_v@mirea.ru. Scopus Author ID 57983605100, https://orcid.org/0000-0003-1423-1072

Andrey V. Korolkov, Cand. Sci. (Eng.), Senior Researcher, Head of the Department of Information Security, Institute of Artificial Intelligence, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: korolkov@mirea.ru. RSCI SPIN-code 3849-6868.

Об авторах

Легкодумов Александр Алексеевич, специалист инженерно-криптографического анализа, ООО «СФБ Лаборатория» (127083, Россия, Москва, ул. Мишина, д. 56, стр. 2). E-mail: studkkso0416@mail.ru. https://orcid.org/0000-0002-2562-4333

Козеев Борис Николаевич, главный специалист, АО «АЛЬФА-БАНК» (107078, Москва, ул. Каланчевская, д. 27). E-mail: kozeev.boris2018@yandex.ru. https://orcid.org/0009-0009-0993-8082

Беликов Владимир Вячеславович, к.воен.н., доцент, доцент кафедры информационной безопасности, Институт искусственного интеллекта ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: belikov_v@mirea.ru. Scopus Author ID 57983605100, https://orcid.org/0000-0003-1423-1072

Корольков Андрей Вячеславович, к.т.н., старший научный сотрудник, заведующий кафедрой информационной безопасности, Институт искусственного интеллекта ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: korolkov@mirea.ru. SPIN-код РИНЦ 3849-6868.

Translated from Russian into English by Kirill V. Nazarov
Edited for English language and spelling by Dr. David Mossop