Russian Technological Journal.
2024;12(2):7–15

ISSN 2782-3210 (Print)
ISSN 2500-316X (Online)

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

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.

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.

ВВЕДЕНИЕ

Программное обеспечение (ПО) глубоко интегрировано в жизнь современного человека. Оно используется в медицинском оборудовании, автомобилях, банках, самолетах, телефонах и т.д. Большая часть современного ПО взаимодействует с персональными данными (ПД) своих пользователей или даже может быть установлена на критически важном объекте (КВО), обеспечивая взаимодействие с особо ценными данными. Компрометация КВО или кража ПД могут привести к разрушению инфраструктуры, потере управления объектом, хищению ресурсов пользователей и другим негативным последствиям. Для обеспечения безопасности данных пользователей и инфраструктуры требуется встраивание сертифицированных средств защиты информации (СЗИ). СЗИ может представлять собой специализированное ПО, предназначенное для защиты информации конфиденциального характера [1]. Для того чтобы иметь возможность защищаться от новых угроз, в исходные коды СЗИ постоянно добавляются новые функции безопасности информации или меняются уже имеющиеся. После каждого подобного изменения сертифицированное СЗИ должно проходить специальные исследования. Необходимость проведения исследований обосновывается приказами Федеральной службы по техническому и экспортному контролю (ФСТЭК)[1] и ФСБ России[2].

Разработка и сопровождение любого ПО представляет собой непрерывный процесс, в котором постоянно изменяется существующая функциональность: добавляются новые функции, обновляются стили кодирования, производится оптимизация, исправляются ошибки. Любое изменение, внесенное в программный либо программно-аппаратный продукт (далее – *изделие*), может оказать непредсказуемое влияние на определенную часть или даже на все функции, выполняемые изделием, и чем больше изменений вносится в изделие, тем сложнее становится отследить их влияние. Для того чтобы обнаружить внесенные изменения, определить их характер и качество, проводится специальное исследование, называемое анализом влияния изменений (анализом изменений, АИ) [2] – это подход к тестированию ПО, который используется для определения степени рисков, связанных с любыми изменениями, внесенными в ПО.

Примечание: далее по тексту, *экспертом* называется сотрудник испытательной лаборатории, который занимается АИ сертифицированных СЗИ.

Значительный объем работ выполняется вручную экспертом и разработчиком СЗИ, поэтому АИ является трудоемкой процедурой. По результатам АИ эксперт стремится получить ответы на следующие вопросы:

1. Какие программные модули и функциональные возможности будут затронуты конкретным изменением и как именно?

[1] Пункт 71 приказа ФСТЭК от 03.04.2018 г. № 55 «Положение о системе сертификации средств защиты информации». В пункте приказа определено, что разработчик СЗИ проводит испытания СЗИ с привлечением испытательной лаборатории. https://fstec.ru/dokumenty/vse-dokumenty/prikazy/prikaz-fstek-rossii-ot-3-aprelya-2018-g-n-55. Дата обращения 02.05.2023. [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] Пункт 41 приказа ФСБ России от 09.02.2005 г. № 66 «Положение о разработке, производстве, реализации и эксплуатации шифровальных (криптографических) средств защиты информации (Положение ПКЗ-2005)». В пункте приказа определено, что все изменения в конструкции средства криптографической защиты информации и технологии их изготовления изготовитель средства криптографической защиты информации должен согласовывать со специализированной организацией и ФСБ России. http://pravo.gov.ru/proxy/ips/?docbody=&prevDoc=102900265&backlink=1&nd=102097894&rdk=0. Дата обращения 02.05.2023. [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.]

2. Повлияет ли эта реализация на функциональные возможности приложения или отдельных модулей приложения?

Так как, в соответствии с приказами № 55 ФСБ России и № 66 ФСТЭК, АИ сертифицированного СЗИ требуется проводить после каждого изменения, то между двумя версиями одного продукта может накопиться достаточное количество изменений, что приведет к увеличению времени работы эксперта. На сложность и количество ресурсов, затрачиваемых на проведение АИ, в таком случае дополнительно влияет и то, что специальное исследование проводится экспертом сторонней организацией, который заранее не знаком с ПО и организацией внутренней работы функций СЗИ. Все это, в совокупности, формирует основную проблему АИ [3].

Эксперт, проводящий АИ, должен обладать высокой квалификацией. От него требуется:

- понимание работы ПО;
- знание языка программирования, на котором разрабатывался продукт;
- понимание работы библиотек, использованных в реализации изделия;
- поиск измененных участков кода, проведение анализа этих участков для оценки влияния изменений на функциональность изделия.

В статье рассмотрены различные подходы к проведению АИ, представлен метод, оптимизирующий проведение анализа, направленный на снижение уровня требований к эксперту и сокращение времени до получения значимых результатов. В контексте данной работы, под значимыми результатами понимается нахождение изменений кода, а также определение их влияния на качественные характеристики СЗИ.

1. АНАЛИЗ СУЩЕСТВУЮЩИХ МЕТОДОВ ПРОВЕДЕНИЯ АИ

1.1. Построчное сравнение

Построчное сравнение – метод, используя который эксперт проводит построчное сравнение различных версий исходных кодов одного изделия для поиска и оценки различий. Процедура может быть выполнена с применением таких средств сравнения исходных текстов ПО как приложения *Beyond Compare*[3], *Araxis Merge*[4] или встроенная в Linux утилита *diff*. Изменением исходных кодов для данного метода считается удаление, добавление или изменение строки. На практике, в большинстве случаев, находимые при построчном сравнении изменения являются ошибочным вхождением, т.к. они не оказывают никакого влияния на реализуемые изделием функции. Такие изменения, которые можно отнести к ошибочным, делятся на:

- изменения названий функций или переменных;
- удаление или добавление переносов строк;
- удаление или добавление комментариев;
- удаление или добавление строк кода, который не относится к реализуемому изделием функционалу – только если это не регламентируется требованиями.

Последнее представляет собой незадействованные участки кода [4]. Это может быть код, который участвует только в тестах разработчика или работает в определенной среде функционирования. Помимо ошибочных вхождений, данный метод плохо показывает себя в ситуациях, когда кодовая база продукта совершает переход на другой язык программирования. В таком случае любая строка будет помечаться инструментами автоматического анализа как несовпадающая.

Плюс данного метода заключается в том, что при его использовании покрывается весь объем исходных текстов ПО, что, в свою очередь, увеличивает возможность обнаружения уязвимого места, в сравнении с проведением автоматизированного анализа [5]. Однако построчный анализ является очень затратной процедурой, занимающей большое количество человеко-часов эксперта, а также требующей от него глубокого знания языка программирования.

1.2. Системы управления версиями

Чтобы уменьшить объем работ, проводимых «вручную» экспертом исследовательской организации, компания-разработчик может передать вместе с материалами исследования дополнительные материалы, например, историю изменений, сформированную системой управления версиями. В современной практике для оптимизации организации работы нескольких разработчиков над одним изделием используется система контроля версий.

Такая система представляет собой ПО, используемое для облегчения работы с изменяющейся информацией, которое помимо управления версиями, организации одновременной работы нескольких разработчиков, поддержки нескольких направлений разработки, а также обеспечения их взаимодействия, позволяет отслеживать изменения программного кода. Использование информации о модификациях исходного кода упрощает его понимание за счет концентрации внимания эксперта.

Преимущество данного метода заключается в том, что отчет об изменениях может быть построен

[3] https://www.scootersoftware.com/. Дата обращения 02.05.2023. / Accessed May 02, 2023.

[4] https://www.araxis.com/merge/index.en. Дата обращения 02.05.2023. / Accessed May 02, 2023.

в автоматическом режиме, каждое изменение не нужно искать отдельно, проводя построчное сравнение двух версий исходных текстов [6]. Но в этом достоинстве кроется его главный недостаток. Благодаря повсеместному использованию систем контроля версий, при разработке большинства программ доступна история изменений. Однако экспертиза может оказаться затруднительной, т.к. качество описания внесенных изменений напрямую зависит от сотрудника, который занимался описанием проведенных изменений в ходе разработки. Поэтому сформированный отчет или история изменений могут предоставлять неполную или даже неверную информацию, которая может помешать провести корректный АИ.

1.3. Автоматизированные тесты

Еще одним методом проведения АИ можно считать сравнение результатов тестирования двух версий одного ПО. Такой метод позволяет проверить и подтвердить факт того, что логика работы функций не изменилась. С помощью данного метода можно отслеживать изменения, приводящие к появлению дефектов в работе функций изделия и функций безопасности [7].

К плюсам данного метода можно отнести возможность автоматизации проверки большого объема данных. Однако вывод по результатам тестирования нельзя считать достоверным, т.к. разработчиком могут быть добавлены незадокументированные возможности в ПО, тесты строятся на основе известных функций изделия и не могут сигнализировать о наличии аномалии ПО. Несмотря на то, что автоматизированная система тестов вполне способна находить ошибки при неправильной реализации изменений, подход с использованием автоматизированных тестов может показать неполный или некорректный результат в контексте проводимых испытаний [8]. Набор тестов или, по-другому, регрессионное тестирование, в зависимости от объема тестируемого кода и качества внесенных изменений, проигрывает АИ в количестве затрачиваемых ресурсов и точности [9]. Преимущество АИ заключается в возможности проверки отдельных участков кода с игнорированием остальной части СЗИ. Таким образом, проведения тестирования измененного ПО, в соответствии с общеизвестными практиками, может быть недостаточно для обеспечения соответствия требованиям безопасности. Некоторые части кода могут потребовать двойной проверки, более глубокого анализа или другого подхода к тестированию [10]. К минусам метода также можно отнести проведение дополнительных работ по созданию или модернизации тестов для ПО, в случае, если был добавлен или удален уже существующий функционал. В редких случаях экспертам исследовательских организаций приходится работать с неполным объемом исходных текстов, что исключает возможность проведения АИ с использованием тестов.

2. МЕТОД ПРОВЕДЕНИЯ АИ

Основная проблема проведения АИ заключается в количестве требуемых для его проведения ресурсов, будь то время эксперта или знания о ПО. Чтобы сократить время анализа результатов тестирования или построчного сравнения, можно проанализировать разницу в логике работы изделия, а также связях между функциями первой и второй версий изделия [11]. Задача анализа с использованием предлагаемого метода делится на следующие шаги:

- проведение статического анализа ПО для получения данных;
- построение последовательности вызова функций на основе полученных в результате статического анализа данных [12];
- представление последовательности вызова функций в виде древовидного графа [13];
- объединение древовидных графов двух версий.

По результатам проведения статического анализа исходных текстов должны быть получены данные:

- об используемых типах данных;
- о структурах классов;
- о последовательности вызовов функций [14].

Построенный корневой граф, представляющий поток управления ПО, содержит в качестве корня первую функцию в потоке управления. Узлами графа являются остальные функции, участвующие в цепочке вызовов [15]. Построенные древовидные графы проходят процедуру объединения, в процессе которого выделяются удаленные или добавленные узлы. На основе данных, полученных из нового графа, эксперт вырабатывает гипотезы.

Выделяются для проверки следующие гипотезы:

- узел удален – функционал исключен из ПО либо перенесен в другой узел;
- узел добавлен – в ПО добавлен новый функционал либо это перенесенный функционал из удаленного узла;
- изменен порядок вызова узлов – произошли изменения в логике обработки данных.

Рассмотрим следующий пример. На рис. 1 и 2 представлены последовательности вызовов функций в виде корневых графов G_1 и G_2. Узел main является начальным вызовом всех остальных функций ПО. Узел func_N, где N – число от 1 до максимально возможного числа вызовов функций, представляет собой любую реализованную в ПО функцию.

На рис. 3 представлен результат объединения графов G_1 и G_2.

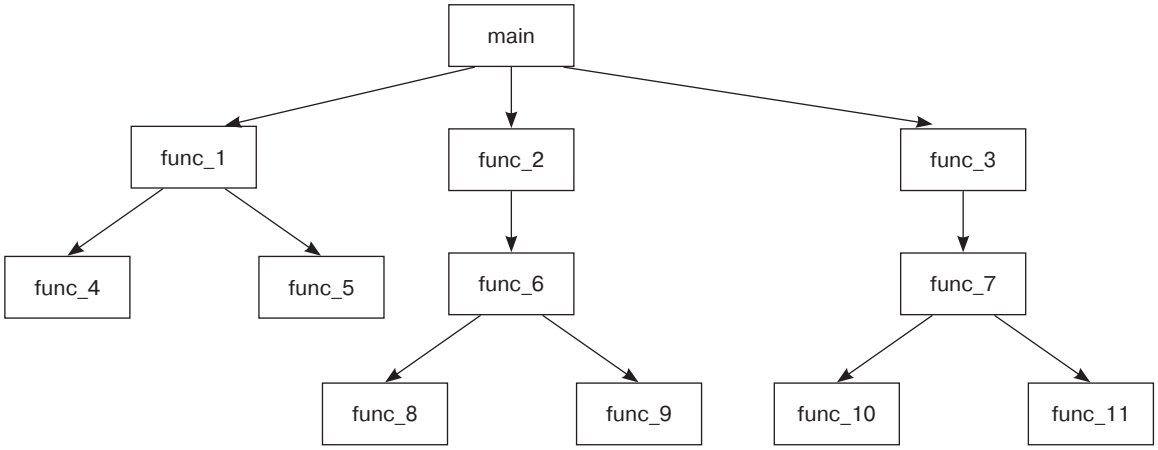

Рис. 1. Представленный в виде древовидного графа поток вызовов функций для старой версии ПО

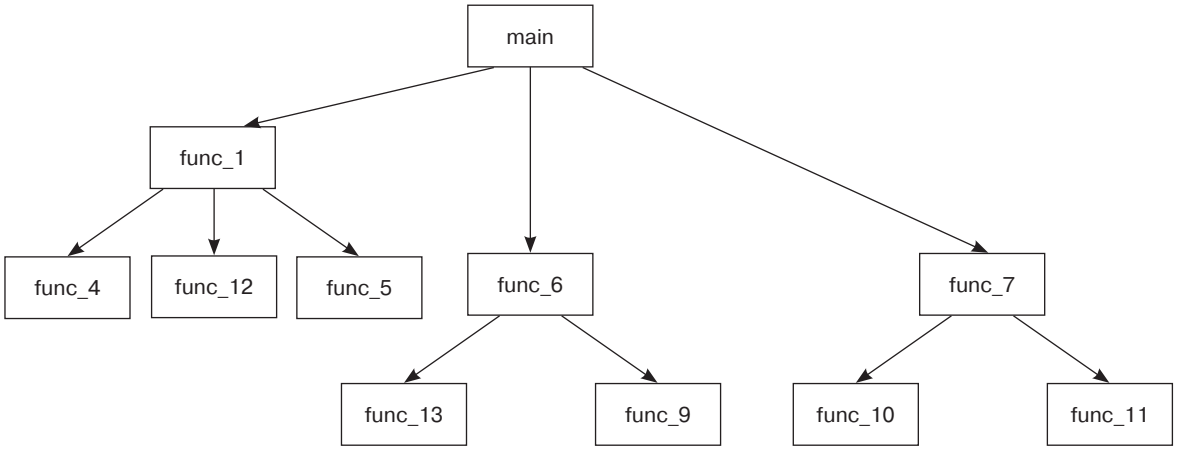

Рис. 2. Представленный в виде древовидного графа поток вызовов функций для новой версии ПО

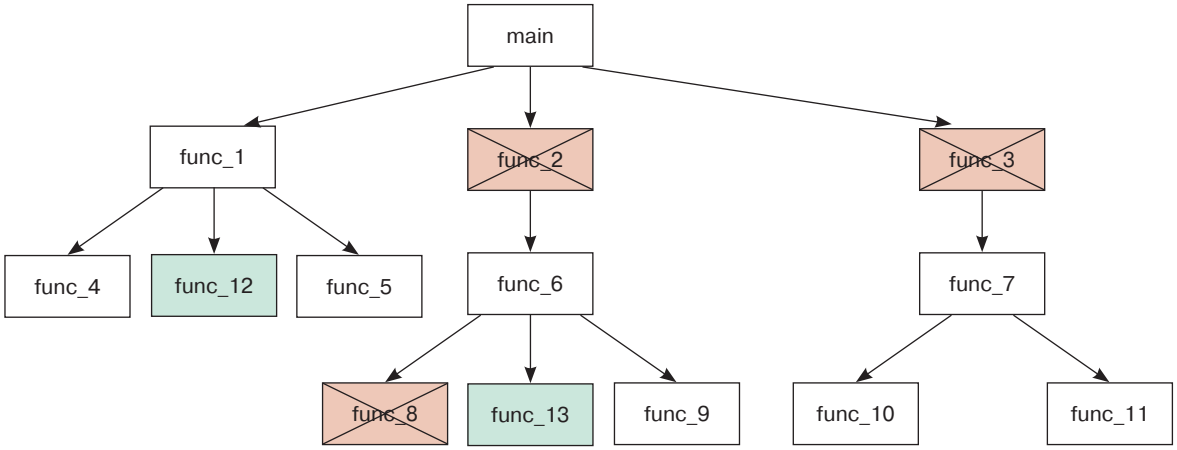

Рис. 3. Результат совмещения двух графов

На основе полученных данных выдвигаются следующие гипотезы для проверки:

- в левой ветке был добавлен узел func_12. Гипотеза: узел реализует новый функционал;
- средняя ветка подверглась наибольшим изменениям. Узлы func_2 и func_8 были убраны. Гипотеза: новый узел func_13 имеет либо новый функционал, либо реализует возможности func_2 и func_8;
- в правой ветке был удален верхний узел func_3. Гипотеза: функционал удален полностью либо перенесен в нижестоящие узлы.

Таким образом, без прямого взаимодействия с исходным текстом ПО задача была локализована и сокращена до задачи анализа версий конкретных функций:

- в левой ветке влияние определяется анализом работы func_12;
- в средней – анализом результата сравнения узлов fucn_2 и func_8 с узлом func_13;
- в правой – определением значимости функционала, реализуемого func_3 и проверкой (с помощью программы поиска) текста наличия реализации func_3 в нижестоящих узлах.

Последующий анализ проводится стандартными методами АИ.

ЗАКЛЮЧЕНИЕ

В работе рассмотрены наиболее распространенные методы и инструменты проведения АИ, а также представлена концепция нового метода, целью которого является исправление недостатков существующих методов проведения исследования влияния изменений ПО на его безопасность. Преимущество метода заключается в его возможности вычислять участки кода, которые подверглись наибольшим изменениям, еще до прямого взаимодействия эксперта с исходными текстами

Помимо применения экспертами в исследовательских организациях метод может быть внедрен в компании, занимающиеся разработкой ПО, для отслеживания появления аномалий в логике работы ПО на этапе разработки.

Дальнейшие исследования будут посвящены использованию метода для АИ, связанных с переходом на новую кодовую базу, а также повышению точности определения уязвимых узлов при использовании в анализе помеченных данных.

Вклад авторов

А.А. Легкодумов – обоснование концепции исследования, разработка методологии исследования, написание прототипа программы, реализующей метод, написание статьи.

Б.Н. Козеев – сбор данных литературы, анализ и обобщение данных литературы, проведение исследований на открытом коде, редактирование статьи.

В.В. Беликов – сбор и систематизация данных, формулировка выводов, анализ результатов исследования, редактирование статьи.

А.В. Корольков – создание модели исследования, планирование исследований, анализ результатов исследования, редактирование статьи.

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.

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

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

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.

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.