Научная статья/Scientific article
УДК 004.056
http://dx.doi.org/10.26583/bit.2026.2.04
https://elibrary.ru/jixmmd

ПРИМЕНЕНИЕ МАШИННОГО ОБУЧЕНИЯ В ЗАДАЧЕ ОБНАРУЖЕНИЯ ПРОГРАММНЫХ КЛОНОВ

Никита А. Грибков✉, Денис В. Иванов
Санкт-Петербургский политехнический университет Петра Великого, ул. Политехническая, 29, литера Б, Санкт-Петербург, 195251, Россия
✉*n.gribkov@ibks.spbstu.ru*

Аннотация. Статья посвящена анализу методов определения схожести фрагментов программного кода, в частности, на уровне бинарных представлений прошивок программно-аппаратных комплексов. Наибольшей эффективности позволяют достичь гибридные методы, сочетающие анализ на нескольких уровнях абстракции. Все подходы классифицируются на основе используемых признаков: синтаксических и семантических. К синтаксическим признакам относятся низкоуровневые элементы, такие как последовательности байт, ассемблерные инструкции, статистические величины и графы потока управления (CFG). Их анализ близок к уровню байт и эффективен для поиска синтаксических клонов. Семантические же признаки, напротив, описывают взаимосвязи в коде и часто представляют собой векторные или графовые модели, построенные с применением алгоритмов машинного обучения, включая NLP-методы. Ключевая проблема – выбор и комбинирование оптимальных источников признаков. Использование только низкоуровневых данных ведет к «недостатку семантики», а опора исключительно на высокоуровневые представления увеличивает число ложных срабатываний. Комбинирование же подходов часто снижает общую эффективность, повышая частоту ошибок обоих типов. В качестве решения предлагается последовательное извлечение и совместное использование низкоуровневых признаков бинарного кода и высокоуровневых семантических признаков, извлекаемых на основе абстрактных синтаксических деревьев (AAST), для оптимизации поиска семантических клонов.

Ключевые слова: *анализ кода, схожесть программного кода, синтаксический и семантический анализ, машинное обучение*

Для цитирования: Грибков, Н., Иванов, Д. (2026). Применение машинного обучения в задаче обнаружения программных клонов. *Безопасность информационных технологий,* 33(2), 28-37. doi: http://dx.doi.org/10.26583/bit.2026.2.04

USING MACHINE LEARNING METHODS IN SOFTWARE CLONE DETECTION

Nikita A. Gribkov✉, Denis V. Ivanov
Peter the Great St.Petersburg Polytechnic University, Politekhnicheskaya Str., 29, Litera B, St. Petersburg, 195251, Russia
✉*n.gribkov@ibks.spbstu.ru*

Abstract. The article is devoted to the analysis of methods for determining the similarity of software code fragments, particularly at the level of binary representations. The highest efficiency is achieved by hybrid methods that combine analysis at multiple levels of abstraction. All approaches are classified based on the features used: syntactic and semantic. Syntactic features include low-level elements such as byte sequences, assembly instructions, statistical measures, and control flow graphs (CFG). Their analysis is close to the byte level and is effective for finding syntactic clones. Semantic features, on the contrary, describe relationships within the code and often represent vector or graph models built using machine learning algorithms, including NLP methods. The key problem is the selection and combination of optimal feature sources. Using only low-level data leads to a "lack of semantics," while relying solely on high-level representations increases the number of false positives. However, combining approaches often reduces overall effectiveness by increasing the frequency of both type I and type II errors. As a solution, the sequential extraction and joint use of low-level binary code features and high-level semantic features

extracted based on Abstract Abstract Syntax Trees (AAST) is proposed to optimize the search for semantic clones.

Keywords: *code analysis, software code similarity, syntactic and semantic analysis, machine learning*

For citation: Gribkov, N., Ivanov, D. (2026). Using machine learning methods in software clone detection. *IT Security (Russia),* 33(2), 28-37. doi: http://dx.doi.org/10.26583/bit.2026.2.04

Введение

Оптимизация ресурсозатрат при проведении испытаний программно-аппаратных комплексов является критически важной задачей, способной существенно повысить эффективность и обоснованность процесса принятия решения о допустимой доверенности комплекса. Традиционные подходы к тестированию, основанные на ручных проверках и регрессионных тестах, зачастую требуют колоссальных временных и человеческих ресурсов, не гарантируя при этом полного охвата всех возможных сценариев работы системы. Это создает значительные риски, так как скрытые дефекты или уязвимости могут остаться невыявленными до момента эксплуатации, что ведет к потенциальным сбоям, нарушениям безопасности и финансовым потерям.

Ключевым решением данной проблемы выступают современные интеллектуальные методы статического и динамического анализа. Эти методы позволяют проводить глубокий автоматизированный аудит кода прошивок компонентной базы на различных уровнях абстракции. Они не ограничиваются поиском уже известных уязвимостей по сигнатурам, а способны выявлять потенциально опасные паттерны и архитектурные недостатки, которые могут привести к возникновению уязвимостей в будущем. Речь идет об анализе потоков данных для обнаружения утечек конфиденциальной информации, проверке корректности обработки ошибок, выявлении небезопасных арифметических операций или неправильного управления памятью.

Комбинирование низкоуровневых и высокоуровневых признаков для определения семантической схожести фрагментов бинарного кода предложено в ряде исследований. При этом выделяется два подхода: применение нескольких методов статического анализа и применение методов статического и динамического анализа.

В [1] применяется статический анализ в методе BinSlayer, развивающем подход BinDiff. Метод включает несколько этапов: сначала BinDiff формирует множество сопоставленных функций, оставляя несопоставленные элементы для экспертного анализа. BinSlayer добавляет этап, где поиск дополнительных соответствий формулируется как задача о назначениях и решается венгерским алгоритмом. Схожесть оценивается через расстояние редактирования графов CFG и CG (Call Graph, граф вызовов), используя низкоуровневые признаки (ассемблерный код, метаданные) и высокоуровневые (CFG, CG).

Исследование [2] также использует низкоуровневые признаки для фильтрации несопоставленных функций перед анализом высокоуровневых. Эвристика основана на сравнении количества базовых блоков и векторных представлений нормализованного ассемблерного кода. После фильтрации семантическая схожесть проверяется на сокращенном множестве кандидатов, что повышает масштабируемость метода. Низкоуровневые признаки включают ассемблерный и нормализованный ассемблерный код, высокоуровневые – CFG.

Метод из [3] решает задачу поиска заимствованных компонентов в прошивках IoT. Он комбинирует низкоуровневые синтаксические признаки (анализ ассемблерного кода базовых блоков) с высокоуровневыми (анализ CFG). На основе [4] строится атрибутный граф потока управления (ACFG) и семантические векторы для оценки схожести фрагментов кода.

Представителем подхода, основанного на комбинировании методов статического и динамического анализа для получения признаков различных уровней, является метод [5]. На первом этапе выполняется статический анализ фрагмента кода и построение его CFG. На втором этапе выполняется динамическая инструментация фрагмента с целью определения участков кода, развертывание которых в памяти происходит на этапе выполнения. На

третьем этапе выполняется модификация CFG с учетом данных, полученных в ходе динамического анализа. Низкоуровневым источником признаков в данном методе является ассемблерный код, высокоуровневыми источниками признаков являются информация о поведении процесса, CFG. Использование динамического анализа и связанных с ним высокоуровневых признаков позволяет снизить влияние методов противодействия анализу программных средств на результат поиска схожих фрагментов кода. В [6] основной этап анализа фрагментов бинарного кода выполняется на основе признаков, формируемых по результатам in-memory-фаззинга. Низкоуровневые признаки, получаемые на основе анализа ассемблерных инструкций, дополняют семантическую сигнатуру каждого анализируемого фрагмента для повышения точности при дальнейшем сравнении.

На основе сравнительного анализа архитектуры, преимуществ и недостатков рассмотренных методов составлена табл. 1.

Интуитивно методы, в которых предлагается последовательное сравнение фрагментов кода на основе признаков различных уровней, являются более эффективными, по сравнению с методами, решение о схожести, в которых принимается на основе комбинации признаков различных уровней. Это подтверждается результатами сравнительного анализа методов, представленными в [7, 8]. Предварительное сопоставление или фильтрация множества фрагментов кода позволяет снизить мощность множества несопоставленных фрагментов, для поиска клонов, в которых требуется применение методов семантического анализа. Как правило, такие методы содержат в своем составе алгоритмы машинного обучения. Поскольку множество несопоставленных фрагментов после этапа синтаксического анализа содержит только фрагменты, различные по структуре и не имеющие синтаксического сходства, модели машинного обучения могут быть сконфигурированы эффективнее для решения более узко специализированной задачи обнаружения семантических клонов. Метод BinSimSearch, представленный в данной работе, основывается, в отличие от иных методов, не только на анализе CG, CFG и ассемблерных инструкций, но и на анализе AAST при выполнении семантического анализа фрагментов. Метод позволяет эффективно определять и исключать из дальнейшего рассмотрения синтаксические клоны, как это делает BinSlayer. Это положительным образом влияет на возможность применения метода к большим наборам бинарных фрагментов кода. Однако предлагаемый метод учитывает особенности работы BinDiff: для программных клонов с низкой метрикой схожести, выработанной BinDiff, выполняется семантический анализ схожести, что позволяет в случае некорректного сопоставления функций на этапе синтаксического анализа скорректировать решение о схожести.

1. Подходы к анализу безопасности программного кода

При оценке безопасности программного обеспечения одной из основных решаемых задач является поиск уязвимостей в программном коде. Все подходы к анализу безопасности можно классифицировать следующим образом:

Анализ состава и конфигурации ПО. Этот подход направлен на проверку соответствия ПО формальным требованиям, предъявляемым внутренней документацией организации, а также нормативными правовыми актами регулирующих государственных органов. Подход позволяет дать формальную оценку уровня защищенности ПО, но не учитывает технические детали, что может привести к упущению существующих недостатков и уязвимостей в программном коде.

Анализ состояния ПО во время эксплуатации. В процессе эксплуатации ПО необходимо обеспечить его безопасность. Для этого реализуется комплекс мер, направленных на контроль поведения программного обеспечения, целостности данных, сканирование сетевого трафика ПО. Такой подход, следовательно, подразумевает использование средств защиты в режиме реального времени в составе информационной системы, в которой функционирует исследуемое ПО.

Таблица 1. Характеристики методов, представленных в связанных исследованиях

Метод	Низкоуровневые источники признаков	Высокоуровневые источники признаков	Методы определения схожести фрагментов (статич. /динамич.)	Способ комбинации источников признаков	Особенности
BinSimSearch	Ассемблерный код, метаданные функций	CG, CFG, AAST	Статические (BinDiff; SimVect [9])	Последовательное применение для снижения мощности множества не сопоставленных фрагментов	Возможность применения к большим наборам фрагментов. Результат BinDiff уточняется методами машинного обучения на основе анализа AAST
BinSlayer [1]	Ассемблерный код, метаданные функций	CG, CFG	Статические (BinDIff; венгерский алгоритм для сопоставления функций по GED)	Последовательное применение для снижения мощности множества не сопоставленных фрагментов	Возможность применения к большим наборам фрагментов. Сопоставленные фрагменты исключаются перед применением венгерского алгоритма
BinSequence [2]	Ассемблерный код, нормализованный ассемблерный код	CFG	Статические (предварительный анализ схожести количества базовых блоков, векторных представлений нормализованного ассемблерного кода; анализ схожести путей в CFG)	Последовательное применение для снижения мощности множества не сопоставленных фрагментов	Возможность применения к большим наборам фрагментов. Анализ схожести фрагментов с использованием теории графов, без методов машинного обучения
[3]	Ассемблерный код	ACFG	Статический (анализ ACFG с помощью графовой нейронной сети [4])	Признаки комбинируются в рамках единого метода для продуцирования решения о схожести фрагментов	Алгоритм анализа трудно масштабировать, т.к. нет предварительного снижения мощности множества фрагментов
[5]	Ассемблерный код	Поведение процесса, CFG	Статический (построение и анализ изоморфизмов CFG), динамический (модификация CFG на основе данных о выполнении кода с инструментацией)	Источники признаков используются последовательно и циклично: построение CFG на основе статического анализа, получение данных динамического анализа, модификация CFG)	Повышенная трудоемкость: требуется безопасная среда исполнения анализируемого ПО. Анализ только CFG не дает достаточно семантической информации для определения схожести
IMF-SIM [6]	Ассемблерный код	Трассы исполнения процесса	Статический (обратный taint-анализ для разрешения типов данных), динамический (построение и сравнение трасс исполнения программы на основе in-memory-фаззинга)	Источники признаков используются последовательно и циклично.	Повышенная трудоемкость. Требуется безопасная среда исполнения анализируемого ПО. Требуется длительный фаззинг для высокого уровня покрытия кода

Анализ программного кода ПО. Подход, основанный на анализе программного кода, включает использование методов статического и динамического анализа кода. Методы статического анализа включают верификацию программного обеспечения, автоматический и ручной статический анализ. Методы верификации безопасности являются формальными и предназначены для определения соответствия кода программного обеспечения требованиям на основе некоторых вычислимых метрик. Анализ может выполняться на уровне исходного кода, двоичного кода или промежуточных представлений. Автоматический статический анализ предполагает использование программного обеспечения: CodeQL, Joern и других. Такие методы позволяют обнаруживать типичные дефекты и уязвимости программного кода, но имеют низкую точность из-за использования синтаксических сигнатур. При ручном статическом анализе акцент делается на экспертной оценке кода. Эксперт может использовать программные инструменты: дисассемблеры, декомпиляторы. Часто ручной и автоматический статический анализ используются вместе для повышения эффективности оценки безопасности. Динамические методы анализа программного кода включают фаззинг, символьное исполнение, автоматический и ручной динамический анализ. Фаззинг позволяет оценить безопасность программного обеспечения с точки зрения покрытия кода. Методы символьного исполнения позволяют проводить доказательную оценку безопасности путем вычисления трассировок, использующих фрагменты кода с недостатками. Автоматический и ручной динамический анализ предполагает экспертную оценку кода во время выполнения с использованием инструментов динамического анализа. По сравнению с фаззингом, экспертный анализ кода во время выполнения обеспечивает меньший охват.

Подход, основанный на анализе состава и конфигурации ПО, позволяет количественно и качественно оценить безопасность программного обеспечения, но в большинстве случаев он является слишком высокоуровневым: анализ используемой кодовой базы не является основой метода, что приводит к пропуску недостатков и ошибок в программном коде. Подход, основанный на анализе состояния программной инфраструктуры во время выполнения, имеет наименьшую полноту и наибольшую продолжительность обнаружения дефектов. Динамический анализ кода для оценки безопасности эффективен с точки зрения обнаружения недостатков и уязвимостей, но требует воспроизведения среды выполнения в тестовом стенде, а также значительных аппаратных ресурсов. Наиболее эффективным подходом для анализа безопасности ПО представляется подход статического анализа. В рамках этого подхода можно реализовать методы с наибольшим покрытием кода и искать уязвимые фрагменты кода на основе как синтаксических, так и семантических сигнатур.

Решение задачи о поиске семантических программных клонов в современных методах основано на применении машинного обучения для построения промежуточных представлений кода, отражающих семантические и структурные признаки, а также на применении машинного обучения напрямую для сравнения фрагментов кода. Далее в работе предлагается использование машинного обучения при построении метода определения программных клонов.

2. Анализ безопасности программного кода на основе решения задачи о поиске программных клонов с использованием машинного обучения

Фрагменты программного кода можно оценивать по степени схожести на различных уровнях представления: бинарном, исходном или промежуточных. В [9, 10] представлен один из вариантов классификации фрагментов кода по данному критерию, выделены синтаксические и структурные характеристики, которые применяются для определения схожести. Авторы приходят к выводу, что гибридные методы, анализирующие код на нескольких уровнях абстракции, демонстрируют наибольшую эффективность по сравнению с другими подходами. В современных исследованиях активно разрабатываются методы на основе алгоритмов машинного обучения для решения задачи определения схожести фрагментов кода.

Методы определения схожести фрагментов бинарных программных кодов могут основываться, как отмечено выше, на анализе синтаксических и семантических признаков [11, 12].

В методе обнаружения синтаксических программных клонов [13] используются последовательности байт для генерации хэш-значений фиксированного размера. На первом этапе анализа бинарного кода метод [14] применяет эти последовательности байт в качестве входных данных для машинного обучения, что позволяет проводить дальнейший семантический анализ. В некоторых исследованиях граф потока управления (CFG) рассматривается как синтаксический признак и используется в качестве основного инструмента для сравнения фрагментов кода [15, 16]. Следовательно, уровень детализации синтаксических признаков сопоставим с уровнем байтов, что обусловлено их определением через синтаксическую схожесть.

Семантические признаки описывают взаимосвязи между фрагментами кода, данные, которые получаются на основе анализа промежуточных представлений программного кода, и векторные представления этих фрагментов [11]. В методах [17, 18] используются векторные представления фрагментов кода, созданные с помощью алгоритмов машинного обучения. Это позволяет свести задачу определения схожести к одной из задач обработки естественного языка (natural language processing, NLP) либо к задаче сравнения графовых представлений фрагментов [19]. Для выделения семантических признаков фрагмента кода необходимо проводить его анализ на различных уровнях гранулярности. Нахождение оптимального набора уровней гранулярности и источников признаков для каждого из них представляет собой сложную проблему, исследованию которой посвящено значительное количество научных работ [см., например 7, 10].

Выбор источников данных для выделения семантических признаков существенно влияет на эффективность методов. Применение исключительно низкоуровневых признаков, таких как анализ двоичного и ассемблерного кода, приводит к недостатку семантической информации. Это проблема характерна для методов анализа бинарного кода, в то время как методы, работающие с исходным кодом ей практически не подвержены. Недостаток семантической информации затрудняет анализ кода, поскольку не используются данные о структуре кода, входных и выходных параметрах, внешних функциях и символах [2].

Применение исключительно высокоуровневых характеристик, таких как графовые и векторные представления, значительно увеличивает вероятность ошибок первого рода в алгоритмах поиска семантических клонов. Это приводит к тому, что фрагменты кода, которые на самом деле не являются клонами, ошибочно распознаются как таковые. Объединение высокоуровневых и низкоуровневых характеристик в рамках одного подхода может не повысить, а, наоборот, снизить эффективность обнаружения синтаксических клонов, что выражается в росте частоты ошибок первого и второго рода. Так, если метод использует высокоуровневые признаки с низким отношением полезной информации к шуму (например, слабо выраженную семантику CFG на гетерогенных платформах), то это приводит к росту FP. Если не выполняется нормализация разных семейств признаков, более «масштабные» признаки могут доминировать в вычисляемой метрике расстояния и вести к росту FN (небольшие синтаксические различия, важные для задачи, теряются среди более значимых семантических) [7]. Для решения проблемы выбора и интеграции различных источников признаков при поиске семантических программных клонов предлагается поэтапное извлечение и использование как низкоуровневых, так и высокоуровневых характеристик. В частности, рассматривается возможность совместного применения семантических признаков, получаемых на основе AAST (Attributed Abstract Syntax Tree), а также синтаксических и структурных характеристик, извлекаемых из бинарного кода программы.

Предлагаемый метод состоит из предварительного и оперативного этапов. На предварительном этапе формируются синтаксические и семантические сигнатуры фрагментов кода, получаемых из сторонних источников (уязвимых/потенциально опасных фрагментов). Синтаксические сигнатуры формируются по алгоритму, аналогичному Bindiff.

Схема алгоритма формирования семантических сигнатур представлена на рис. 1. Общий процесс применения метода представлен на рис. 2.

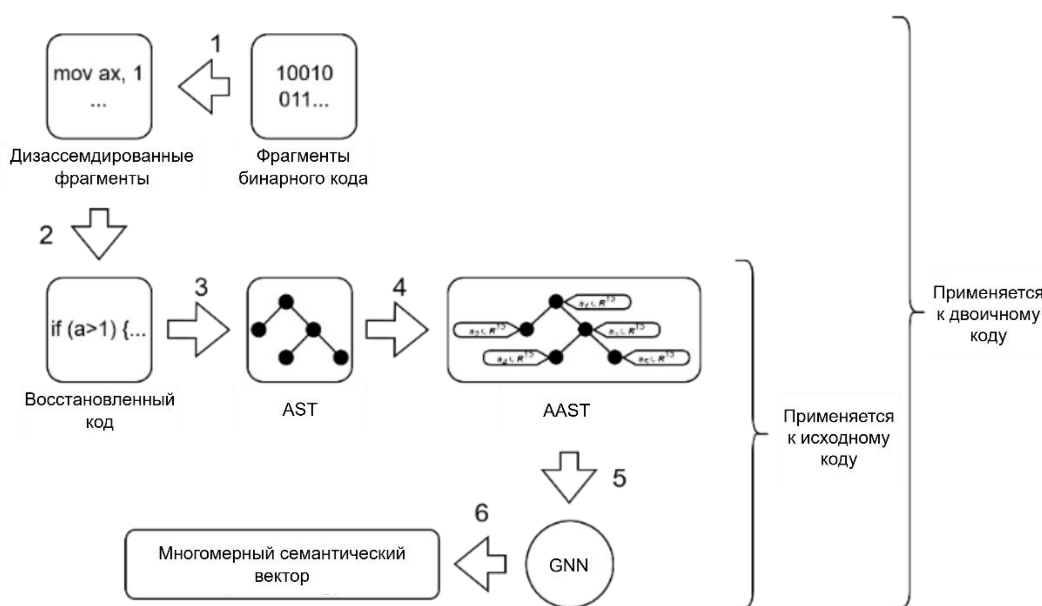

Рис. 1. Схема алгоритма формирования семантических сигнатур

Рис. 2. Общий процесс применения метода

Предлагаемый метод, таким образом, представляет собой каскад алгоритмов. Первоначально в анализируемой кодовой базе выполняется поиск синтаксических клонов и их исключение из дальнейшего анализа (как не представляющих интерес или потенциально уязвимых). На следующем этапе аналогичный процесс выполняется для семантических клонов. Для формирования семантических сигнатур, сравнение которых в действительности выполняется, предлагается использовать глубокую графовую нейросеть. Это позволяет формировать репрезентативные семантические вектора с учетом признаков, включенные в AAST, которое является входным параметром для нейросети.

3. Применение больших языковых моделей в задаче поиска программных клонов

Современной тенденцией при решении задач анализа программного кода является использование больших языковых моделей (LLM). Применение таких методов позволяет решать широкий спектр задач:

1. Обнаружение плагиата. LLM могут помочь выявить фрагменты кода, которые были скопированы из других источников без должного указания авторства.

2. Оптимизация кода. LLM могут анализировать большие объёмы кода и выявлять повторяющиеся шаблоны, которые могут быть оптимизированы или рефакторизованы.

3. Обеспечение соответствия стандартам. LLM могут проверять код на соответствие определённым стандартам и рекомендациям, например, на соответствие стилю кодирования или на отсутствие уязвимостей.

4. Автоматизация тестирования. LLM могут генерировать тестовые случаи для проверки функциональности кода, включая проверку на наличие клонов.

5. Анализ больших кодовых баз. LLM могут обрабатывать большие объёмы кода, что может быть полезно при анализе крупных программных проектов.

Ввиду ориентирования больших языковых моделей на анализ неструктурированных и слабоструктурированных данных возникают сложности с их адаптацией для решения задачи поиска программных клонов. Входными данными для алгоритмов, реализуемых в рамках традиционных подход, является, помимо исходного или двоичного кода фрагмента, ряд их промежуточных представлений, признаки и информация о структуре, извлекаемая из фрагментов. При использовании LLM входными данными для алгоритма исходный или двоичный код фрагмента, с минимальными дополнительными данными. Промежуточные представления кода используются редко. На стадии обучения фрагменты кода часто сопровождаются текстовыми комментариями, которые позволяют модели лучше интерпретировать фрагмент кода.

Использование больших языковых моделей дает, таким образом, ряд плюсов:

1. Повышение точности семантического анализа.
2. Менее трудоемкая предварительная обработка данных.

Недостатки применения LLM в рамках рассматриваемой задачи является снижения доверия к методу, поскольку большая языковая модель оценивается только экспериментально. Кроме того, несмотря на общую тенденцию к более точной интерпретации семантики, по сравнению с традиционными методами, большие языковые модели могут вести себя нестабильно.

Заключение

В работе проанализированы основные подходы к анализу безопасности программного обеспечения, выделены их преимущества и недостатки. На основе анализа предложен метод оценки уровня безопасности программного обеспечения, основанный на оценке количества уязвимых фрагментов кода, включенных в исследуемое программное обеспечение. Сутью предлагаемого метода является использование каскада алгоритмов поиска семантических и синтаксических программных клонов, а также применение графовых нейронных сетей для продуцирования семантических сигнатур фрагментов кода.

Дальнейшие исследования направлены на формализацию результатов метода с целью обеспечения возможности их включения в нормативную и правовую базу, а также на экспериментальную оценку эффективности предложенного метода. Еще одним направлением дальнейших исследований является определение возможностей использования предложенного решения для оценки безопасности больших кодовых баз.

СПИСОК ЛИТЕРАТУРЫ/REFERENCES:

1. Martial Bourquin, Andy King, and Edward Robbins. 2013. BinSlayer: accurate comparison of binary executables. In Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop (PPREW '13). Association for Computing Machinery, New York, NY, USA, Article 4, 1-10. DOI: https://doi.org/10.1145/2430553.2430557.
2. He Huang, Amr M. Youssef, and Mourad Debbabi. 2017. BinSequence: Fast, Accurate and Scalable Binary Code Reuse Detection. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17). Association for Computing Machinery, New York, NY, USA, 155-166. DOI: https://doi.org/10.1145/3052973.3052974.
3. Binbin Zhao, Shouling Ji, Jiacheng Xu, Yuan Tian, Qiuyang Wei, Qinying Wang, Chenyang Lyu, Xuhong Zhang, Changting Lin, Jingzheng Wu, and Raheem Beyah. 2022. A large-scale empirical analysis of the vulnerabilities introduced by third-party components in IoT firmware. In Proceedings of the 31st ACM SIGSOFT International

Symposium on Software Testing and Analysis (ISSTA 2022). Association for Computing Machinery, New York, NY, USA, 442-454. DOI: https://doi.org/10.1145/3533767.3534366.
4. Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. 2017. Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). Association for Computing Machinery, New York, NY, USA, 363-376. DOI: https://doi.org/10.1145/3133956.3134018.
5. Roundy, K.A., Miller, B.P. (2010). Hybrid Analysis and Control of Malware. In: Jha, S., Sommer, R., Kreibich, C. (eds) Recent Advances in Intrusion Detection. RAID 2010. Lecture Notes in Computer Science, v. 6307. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-642-15512-3_17.
6. S. Wang and D. Wu. In-memory fuzzing for binary code similarity analysis. 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), Urbana, IL, USA, 2017, pp. 319-330. DOI: 10.1109/ASE.2017.8115645.
7. Li, B., Zhang, Y., Peng, H., Fan, Q., He, S., Zhang, Y., Shi, S., Zhang, Y., & Ma, A. (2023). Multi-semantic feature fusion attention network for binary code similarity detection. Scientific reports, 13(1), 4096. DOI: https://doi.org/10.1038/s41598-023-31280-w. EDN: CKIEEH.
8. I.U. Haq, J. Caballero A Survey of Binary Code Similarity. arXiv:1909.11424 [cs]. – arXiv, 2019. DOI: https://doi.org/10.48550/arXiv.1909.11424.
9. Грибков, Н.А. Анализ восстановленного программного кода с использованием абстрактных синтаксических деревьев. Н.А. Грибков, Т.Д. Овасапян, Д.А. Москвин. Проблемы информационной безопасности. Компьютерные системы. 2023, № 2(54). DOI: https://doi.org/10.48612/jisp/ruar-u6he-kmd4.
Gribkov N.A., Ovasapyan T.D., Moskvin D A. Analysis of decompiled program code using abstract syntax trees. Information Security Problems. Computer Systems. 2023, no. 2(54). DOI: https://doi.org/10.48612/jisp/ruar-u6he-kmd4. EDN: BGKMKA (in Russian).
10. Marcelli A. et al. How machine learning is solving the binary function similarity problem.31st USENIX Security Symposium (USENIX Security 22). 2022. pp. 2099-2116. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/marcelli (accessed: 1.11.2025).
11. F. Ye, S. Zhou, A. Venkat, et al. MISIM: A Neural Code Semantics Similarity System Using the Context-Aware Semantics Structure. MISIM. arXiv:2006.05265 [cs, stat]. – arXiv, 2021. DOI: https://doi.org/10.48550/arXiv.2006.05265.
12. Zhu, X., Wang, J., Fang, Z., Yin, X., & Liu, S. (2023). BBDetector: A Precise and Scalable Third-Party Library Detection in Binary Executables with Fine-Grained Function-Level Features. Applied Sciences, 13(1), 413. DOI: https://doi.org/10.3390/app13010413. EDN: MMFCZM.
13. xorpd | FCatalog. URL: https://www.xorpd.net/pages/fcatalog.html (accessed: 11.11.2025).
14. Bingchang Liu, Wei Huo, Chao Zhang, Wenchao Li, Feng Li, Aihua Piao, and Wei Zou. 2018. ADiff: cross-version binary code similarity detection with DNN. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18). Association for Computing Machinery, New York, NY, USA, 667-678. DOI: https://doi.org/10.1145/3238147.3238199.
15. Yaniv David, Nimrod Partush, and Eran Yahav. 2016. Statistical similarity of binaries. In Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '16). Association for Computing Machinery, New York, NY, USA, 266-280. DOI: https://doi.org/10.1145/2908080.2908126.
16. J. Pewny, B. Garmany, R. Gawlik, C. Rossow and T. Holz. Cross-Architecture Bug Search in Binary Executables. 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 2015, pp. 709-724. DOI: https://doi.org/10.1109/SP.2015.49.
17. H. Wang, Z. Gao, C. Zhang, et al. CLAP: Learning Transferable Binary Code Representations with Natural Language Supervision. CLAP. arXiv:2402.16928 [cs]. – arXiv, 2024. DOI: https://doi.org/10.48550/arXiv.2402.16928.
18. H. Wang, Z. Gao, C. Zhang, et a.l CEBin: A Cost-Effective Framework for Large-Scale Binary Code Similarity Detection. CEBin. arXiv:2402.18818 [cs]. – arXiv, 2024. DOI: https://doi.org/10.48550/arXiv.2402.18818.
19. Kalinin, M., Zavadskii, E., & Busygin, A. (2023). A Graph-Based Technique for Securing the Distributed Cyber-Physical System Infrastructure. Sensors, 23(21), 8724. DOI: https://doi.org/10.3390/s23218724. EDN: MIBSKV.

Конфликт интересов. *Авторы заявляют об отсутствии конфликта интересов.*
Conflict of interest. *The authors declare no conflict of interest.*

Вклад авторов
Иванов Д.В.: *концептуализация, разработка методологии исследования, написание черновика.*
Грибков Н.А.: *написание и редактирование рукописи, курирование данных, проведение исследования.*

Author Contributions
Ivanov D.V.: *conceptualization, methodology, writing – original draft.*
Gribkov N.A.: *writing – review & editing, data curation, investigation.*

ИНФОРМАЦИЯ ОБ АВТОРАХ:
Никита Андреевич Грибков, *младший научный сотрудник, Институт компьютерных наук и кибербезопасности СПбПУ, Санкт-Петербургский политехнический университет Петра Великого.*
e-mail: n.gribkov@ibks.spbstu.ru,
https://orcid.org/0009-0004-8416-7023,
SPIN-код: 3045-0532,
Researcher ID: JZD-9468-2024,
Scopus Author ID: 58917784600

Денис Вадимович Иванов, *к.т.н.; доцент, Институт компьютерных наук и кибербезопасности СПбПУ, Санкт-Петербургский политехнический университет Петра Великого.*
e-mail: 9361023@gmail.com,
https://orcid.org/0000-0001-8206-2915,
SPIN-код: 4661-8076,
Researcher ID: R-9910-2019,
Scopus Author ID: 57207760864

INFORMATION ABOUT THE AUTHORS:
Nikita Andreevich Gribkov, *Junior Researcher, Institute of Computer Science and Cybersecurity, Peter the Great St. Petersburg Polytechnic University (SPbPU).*
e-mail: n.gribkov@ibks.spbstu.ru,
https://orcid.org/0009-0004-8416-7023,
SPIN-code: 3045-0532,
Researcher ID: JZD-9468-2024,
Scopus Author ID: 58917784600

Denis Vladimirovich Ivanov, *PhD (Tech.); Associate Professor, Institute of Computer Science and Cybersecurity, Peter the Great St. Petersburg Polytechnic University (SPbPU).*
e-mail: 9361023@gmail.com,
https://orcid.org/0000-0001-8206-2915,
SPIN-code: 4661-8076,
Researcher ID: R-9910-2019,
Scopus Author ID: 57207760864

Статья поступила в редакцию 11.11.2025; одобрена после рецензирования 19.02.2026; принята к публикации 12.03.2026
The article was submitted 11.11.2025; approved after reviewing 19.02.2026; accepted for publication 12.03.2026