DOI: 10.15514/ISPRAS-2019-31(2)-12

Выявление характерных особенностей программ для борьбы с компьютерным пиратством на основе интеллектуального анализа графов

[1] *С. Сарвар, ORCID: 0000-0001-9714-6580 <sohail.sarwar@seecs.edu.pk>*
[1] *З.Уль Кайум, ORCID: 0000-0003-4230-6895 <zia@aiou.edu.pk>*
[2]*М. Сафьян, ORCID: 0000-0003-4501-9699 <msafyan@gcul.edu.pk>*
[3,4]*М. Икбал, ORCID: 0000-0002-8438-6726 <miqbal@lsbu.uk>*
[2]*Я. Махмуд, ORCID: 0000-0002-9676-2100 <yasir@iqra.edu.pk>*
[1] *Университет Гуджарата, Пакистан*
[2] *Правительственный университет колледжа, Лахор, Пакистан*
[3] *Лондонский университет Саут Бэнк, Великобритания*
[4] *Университет Эссекса, Великобритания*

Аннотация. Расширение числа и ассортимента компонентов программного обеспечения в значительной степени подчеркивает необходимость защиты прав интеллектуальной собственности (IPR), затрудняемую компьютерным пиратством, для борьбы с которым требуются эффективные меры. Выявление характерных особенностей программного обеспечения предназначено для противодействия незаконному заимствованию права собственности на программное обеспечение путем установления его происхождения. В этой статье предлагается новый подход к выявлению характерных особенностей программ, основанный на сочетании методов интеллектуального анализа текстов и графов. Элементы кода программы и их связи с другими элементами идентифицируются на основе их особенностей (конструкций кода) и преобразуются в конструкции языка манипулирования графами. Характерные особенности программного обеспечения, выводимые путем исследования теоретических свойств графа (на основе коэффициента кластеризации), используются для установления сходства или различия двух программ. Предложенная методика была оценена по показателям достоверности, устойчивости, заимствования методов, обнаружения модифицированного кода и самокопирования. Результаты подтверждают эффективность предложенного подхода для противодействия незаконному заимствованию права собственности. Сравнительный анализ предложенного подхода с современными решениями показывает лучшие результаты при выявлении свойств и связей узлов программы и при использовании динамических методов анализа графов без дополнительных накладных расходов (таких как увеличение размера программы и затрат на обработку).

Ключевые слова: право интеллектуальной собственности; незаконное использование программного обеспечения; характерные особенности программного обеспечения; интеллектуальный анализ графов; достоверность и устойчивость.

Для цитирования: Сарвар С., Ул Кайум З., Сафьян М., Икбал М. , Махмуд Я. Выявление характерных особенностей программ для борьбы с компьютерным пиратством на основе интеллектуального анализа графов. Труды ИСП РАН, том 31, вып. 2, 2019 г., стр. 171-186. DOI: 10.15514/ISPRAS-2019-31(2)-12

Graphs Resemblance based Software Birthmarks through Data Mining for Piracy Control

[1] *S. Sarwar, ORCID: 0000-0001-9714-6580 <sohail.sarwar@seecs.edu.pk>*
[1] *Z.Ul Qayyum, ORCID: 0000-0003-4230-6895 <zia@aiou.edu.pk>*
[2]*M. Safyan, ORCID: 0000-0003-4501-9699 <msafyan@gcul.edu.pk>*
[3,4]*M. Iqbal, ORCID: 0000-0002-8438-6726 <miqbal@lsbu.uk>*
[2]*Y. Mahmood, ORCID: 0000-0002-9676-2100 <yasir@iqra.edu.pk>*
[1] *Department of Computer Science, University of Gujrat, Pakistan*
[2] *Department of Computing, GC University Lahore, Pakistan*
[3] *School of Engineering, London South Bank University, England*
[4] *School of Computer Science and Electronic Engineering University of Essex, England*

Abstract. The emergence of software artifacts greatly emphasizes the need for protecting intellectual property rights (IPR) hampered by software piracy requiring effective measures for software piracy control. Software birthmarking targets to counter ownership theft of software by identifying similarity of their origins. A novice birthmarking approach has been proposed in this paper that is based on hybrid of text-mining and graph-mining techniques. The code elements of a program and their relations with other elements have been identified through their properties (i.e code constructs) and transformed into Graph Manipulation Language (GML). The software birthmarks generated by exploiting the graph theoretic properties (through clustering coefficient) are used for the classifications of similarity or dissimilarity of two programs. The proposed technique has been evaluated over metrics of credibility, resilience, method theft, modified code detection and self-copy detection for programs asserting the effectiveness of proposed approach against software ownership theft. The comparative analysis of proposed approach with contemporary ones shows better results for having properties and relations of program nodes and for employing dynamic techniques of graph mining without adding any overhead (such as increased program size and processing cost).

Keywords: Intellectual Property Rights (IPR); Software Ownership Theft; Software Birthmarking; Graph Mining; Credibility and Resilience.

For citation: Sarwar S., Ul Qayyum Z., Safyan M., Iqbal M., Mahmood Y. Graphs Resemblance based Software Birthmarks through Data Mining for Piracy Control. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 2, 2019. pp. 171-186 (in Russian). DOI: 10.15514/ISPRAS-2019-31(2)-12

1. Введение

Достижения программной инженерии изменили ход технического прогресса на основе ряда инновационных идей. Реализация этих идей повлекла за собой смену парадигмы с переходом к мобильным приложениям, интеллектуальным технологиям («умные» дома, офисы, города и т.д.), новым вычислительным парадигмам (грид и облачные вычисления), обучению на основе продвинутых технологий (Technology Enhanced Learning, TeL), Интернет вещей (IoT) и т.д. Эти инновационные разработки программного обеспечения, основанные на новизне идей (то есть на интеллектуальной собственности), сталкиваются с потенциальными угрозами. Эти угрозы правам интеллектуальной собственности (intellectual property right, IPR) включают компьютерное пиратство, незаконное заимствование прав собственности, реверс-инжиниринг, копирование программного обеспечения (или его частей) и т.д.

В одном из опубликованных исследований утверждается, что более 50% потребителей технологий работают с применением пиратского программного обеспечения [1]. Основной причиной этого широко распространенного нарушения законов о пиратстве является присущая программным продуктам природа, позволяющая легко их воспроизводить и распространять, что отличает программные продукты от продуктов других отраслей. Программные продукты приобретаются клиентами с получением прав только на использование только на их использования без права на какие-либо изменение или. Однако эти этические обязательства нарушаются в результате незаконного распространения программного обеспечения, что влечет огромные убытки для поставщиков.

Поставщики программного обеспечения защищают авторские права на свои продукты, отслеживая передовые подходы [6], которые гарантируют пользователям, имеющим разрешение, полноценное использование программного обеспечения и его настройку при необходимости. Почти все аспекты программного обеспечения скрываются, чтобы предотвратить его несанкционированную продажу. Так что конечной целью продавца продукта программного обеспечения является защита компьютерного пиратства [7].

Для борьбы с компьютерным пиратством разработан ряд методов [2] [3] [4] [5]. Эти методы направлены на противодействие растущему объему пиратского программного обеспечения, реверс-инжинирингу, взлому программного обеспечения и его незаконному использованию. В частности, используются следующие методы.

- Цифровые водяные знаки (Watermarking): используются для подтверждения владения программным обеспечением.
- Проверка подлинности (Tamper-proofing): программа уничтожается при незаконном использовании.
- Обфускация (Obfuscation): скрывает структуру программного обеспечения для противодействия обратной инженерии.
- Шифрование (Encryption): защищает программное обеспечение путем кодирования с использованием открытых или закрытых ключей.
- Выявление характерных особенностей (Birthmarking): незаконное использования программного обеспечения обнаруживается на основе его уникальных свойств.

Все перечисленные методы, кроме последнего, могут добавлять в коде различные операторы принятия решения (decision statement); следовательно, размер кода увеличивается, что влечет за собой снижение производительности и, следовательно, возможности многократного применения. Кроме того, при применении методы выявления характерных особенностей программ идентифицируются уникальные характеристики программы (называемые внутренними свойствами), такие как элементы переменных, циклов, ветвлений, присваиваний и т.д. [8]. Нетривиальной задачей является изменение этих программных конструкций, и еще труднее обосновать эти изменения [7], [9] при выявлении факта незаконного использования программного обеспечения (или прав собственности).

Учитывая вышесказанное, мы предлагаем новый метод выявления характерных особенностей программного обеспечения с использованием интеллектуального анализа графов. Для выявления характерных черт каждого метода и класса используются теория графов [8] и сетевые методы [9]. Предлагаемый метод по своей природе является статическим, основанным на синтаксической структуре программы. Эта структура используется для вычисления значения свойств каждого элемента программы и связей между этими элементами. Впоследствии значения свойств преобразуются в графы. Теоретические свойства этих графов (на основе коэффициента кластеризации) позволяют сравнивать характерные особенности (в виде графов) двух программ. В результате две программы классифицируются как схожие или несхожие (выявляется, была ли скопирована программа, или был заимствован один класс или метод(ы) класса украден).

Кроме того, предлагаемый метод статического определения характерных особенностей программ основе графов может определить, была ли изменена программа. Метод выявляет характерные особенности программного обеспечения на основе связей внутренних *характеристик* каждого метода в программе. Эти *связанные характеристики* играют ключевую роль в выполнении функциональных требований к каждому методу. Все характерные особенности уровня методов классов программ используются для построения характерных особенностей программы. Эти статистические построения обеспечивают меру сходства или различий между различными программами. Аспекты предлагаемой методики были оценены по показателям достоверности и устойчивости на разных уровнях детализации при отслеживании требуемых накладных расходов.

Оставшаяся часть статьи организована следующим образом. В разд. 2 рассматриваются основные понятия выявления характерных особенностей программ и приводится краткий обзор распространенных методов. В разд. 3 подробно описывается техника определения характерных особенностей программ, представленная в этой работе. В разд. 4 обсуждается оценка предлагаемого подхода. Разд. 5 завершает представленную работу с указанием направлений будущих исследований.

2. Обзор литературы

В этом разделе представлен обзор существующих методов борьбы с компьютерным пиратством в целом; особое внимание уделяется методам выявления характерных особенностей программ.

Пиратство программного обеспечения стало глобальной проблемой, о чем свидетельствует коммерческая оценка в 62,7 млрд. долл. США объема используемого нелицензионного программного обеспечения в 2013 году [1]. Аналогичные исследования указывают на рост пиратства на 42-43% только в 2014 году [2]. Пиратство можно разделить на две основные области: нелегальное распространение программного обеспечения и реверс-инжиниринг. Хотя для минимизации компьютерного пиратства было выполнено множество разработок: обфускация (obfuscation), проверка подлинности (tamper-proofing), цифровые водяных знаки (watermarking), хеширование и контроль потока управления, но их эффективность можно значительно улучшить. Обфускация трансформирует программу, делая ее менее понятной при сохранении семантики [11] [12]. Проверка подлинности [7] основана на расширении кода специальными средствами [13], что чревато дополнительными накладными расходами. Таким образом, эти методы снижают производительность программного обеспечения, требуя специальной среды выполнения, такой как виртуальная Java-машина [14].

Цифровые водяные знаки программного обеспечения – это метод защиты интеллектуальной собственности приложения. Смысл этого метода тот же самый, что и у обычных цифровых водяных знаков, когда в текстовых, графических, аудио- или видеоданных размещается уникальный идентификатор таким образом, что он не может быть обнаружен людьми [19-23].

Некоторые исследователи использовали методы инженерии знаний для классификации плагиата [24]. Внутренний плагиат, внешний плагиат и авторство могут быть обнаружены [25] с помощью интеллектуального анализа текста. В [26] обсуждается практическая и эффективная методика обнаружения пиратского программного обеспечения с использованием метафорического анализа. Для обнаружения незаконного использования программного обеспечения использовались атрибуты минимизации.

В некоторых работах для противодействия компьютерному пиратству предлагалось использовать гибрид методов выявления характерных особенностей и водяных знаков [22] [24] [25], поскольку характерные особенности идентифицирует только внутренние свойства программы. Сочетание обнаружения характерных особенностей и водяных знаков может быть эффективной мерой по борьбе с пиратством без ущерба для производительности программного обеспечения [15].

Помимо обнаружения незаконного использования копий программ, выявление характерных особенностей программного обеспечения использовалось для обнаружения вредоносных программ [27, 28].

В [29] предлагался метод выявления характерных особенностей электронных схем для предсказания возможности их повторного использования.

Вопросы распространенных методов классификации вредоносных программ в приложениях на основе Android обсуждались в [35]. Система FalDroid, основанная на анализе, показала достаточно высокую точность.

В следующих подразделах приводятся классификация методов обнаружения характерных особенностей программного обеспечения и обзор результатов соответствующих исследований.

2.1 Характерные особенности программного обеспечения

Характерными особенностями программного обеспечения называется уникальный набор характеристик компьютерной программы или ее компонента. Каждое характерная особенность программы обладает двумя свойствами: надежностью и устойчивостью к трансформации [7], [15], [16], [9]. Характерные особенности программы могут быть статическими или динамическими.

2.2 Статические характерные особенности

В отношении к понятию статических характерных особенностей программ мы полагаемся на основополагающую работу [8].

Пусть *a* и *b* – это фрагменты кода двух разных программ и *z* – программный компонент, извлекающий уникальный набор особенностей из некоторого метода (методов). У *a* существует набор характерных особенностей *в том и только в том случае, когда*:

1) *z(a)* извлекается из фрагмента *a* (при отсутствии данных о других компонентах программы и
2) если b является копией a, то *z(a)=z(b)*.

2.3 Динамические характерные особенности

Другим вариантом являются динамические характерные особенности. Это понятие было введено в фундаментальной работе [15].

Пусть *a* и *b* – это фрагменты кода двух разных программ, вводящие с консоли входные данные *c*. *z* – программный компонент, извлекающий уникальный набор свойств программ *a* и *b*. Тогда можно утверждать, что *z(a,c)* является набором характерных особенностей *a в том и только в том случае, когда*:

1) *z(a,c)* получается из a только после выполнения фрагмента *a* после ввода с консоли данных *c* и
2) если *b* является копией *a*, то *z(a,c)=z(b,c)*.

Некоторые другие вариации характерных особенностей программ можно обнаружить в [8, 9, 15, 16], где для Java-программ устанавливаются четыре значимых типа особенностей:

1) наличие констант в полях классов (CVFV),
2) последовательность вызовов методов (SMC),
3) структура наследования (IS)
4) используемые классы (UC).

Эти характерные особенности не обладают необходимой устойчивостью и тривиальны по своей природе, поэтому легко подвержены трансформации [15]. Подход к выявлению характерных особенностей, основанный на анализе полных путей в программе (whole program path, WPP) и предложенный в [16], может страдать от таких уязвимостей, как преобразования цикла, или атак с подстановкой кода [17].

Другой динамический подход к выявлению характерных особенностей, более устойчивый, чем WPP, основан на последовательности вызовов интерфейса прикладного программирования (API) [9]. Однако окно коротких вызовов методов может справиться только с ограниченным набором вызовов API.

Методы динамического выявления характерных особенностей для многопоточных приложений с использованием системных вызовов, связанных с потоками, были предложены

в [17] [18]. Выявление характерных особенностей для обнаружения плагиата в многопоточных программах является сложной задачей. В [36] был предложен методик динамического выявления характерных особенностей TOB-PD, который демонстрирует надежную работу.

Принимая во внимание недостатки существующих работ и исследований, была мы предлагаем новый и эффективный метод выявления характерных особенностей программного обеспечения. Предлагаемый подход, описываемый в разд. 3, представляет собой гибрид методов анализа текста и графов. Методы интеллектуального анализа графов, такие как кластеризация и поиск клик [30, 31], используются еще и для идентификации модифицированного кода. Применение графов [32], позволяет учитывать связи между элементами метода, а также классов программ (или программ) для определения происхождения соответствующих компонентов.

3. Предлагаемый метод

Предлагаемый метод работает с внутренними особенностями программного кода, идентифицируя элементы в методе(ах), свойства элементов и их связи с другими элементами в других программах. Элементы программы (внутри класса или метода) представляют свойства и связи между элементами. Этими элементами являются переменные методов, повторяемость/циклы, присваивания и операторы принятия решений. Связь элементов с их свойствами трансформируется в граф. Узлы в графе представляют элементы метода, а ребра представляют – связи между элементами (как показано на рис. 4 и 5). Это подразумевает, что в фрагментах кода элементы метода всех должны иметь как минимум одно соединение (или связь) с другими элементами [8]. Другими словами, для графа *G*, с узлами *N* и ребрами *E* должно выполняться следующее условие.

$\boldsymbol{G} = \{\boldsymbol{N}, \boldsymbol{E}\}, \boldsymbol{N} = \{\boldsymbol{n_1}, \boldsymbol{n_2}, \dots, \boldsymbol{n_k}\}, \boldsymbol{E} = \{\boldsymbol{e_1}, \boldsymbol{e_2}, \dots, \boldsymbol{e_m}\}$

Для каждой пары узлов $\boldsymbol{n_i}$ и $\boldsymbol{n_j}$ $(\boldsymbol{i} \neq \boldsymbol{j})$ существует ребро $\boldsymbol{e_l}$. соединяющее $\boldsymbol{n_i}$ и $\boldsymbol{n_j}$.

Чтобы сделать набор характерных особенностей более надежным, мы выбираем фрагмент кода (или метода) включающий многочисленные элементы и свойства (вместо взаимно-однозначного отображения между узлами и ребрами). Это приводит к сложному преобразованию внутренних характеристик методов, называемых характерными особенностями программ. Такие характеристики (или характерные особенности) позволяют установить, является ли фрагмент кода копией или оригиналом.

Примеры связей между элементами показаны в табл. 1.

Табл. 1. Внутренние характеристики: свойства и связи 1.
Table 1. Intrinsic Characteristics: Properties and Relations in Elements

Название элемента	Свойства				
	Глобальная переменная	Локальная переменная	Цикл	Условие	Данные
Глобальная переменная		присваивание	использование	использование	присваивание
Локальная переменная	присваивание	присваивание	использование	использование	присваивание
Цикл	использование	использование	одинаковые уровни	использование	использование
Условие	выход	использование	использование	одинаковые уровни	использование
Данные	присваивание	присваивание	использование	использование	присваивание

Имеются разные типы связей (включение или обобщение) между циклами и связи по одноуровневости (sibling relation) с другими циклами. Подобные типы связей могут наблюдаться между глобальными и локальными переменными. Эти комбинации связей и свойств элементов методов важны для классификации характерных особенностей.

Процесс сборки и формирования набора характерных особенностей проиллюстрирован на рис. 1. Первым этапом является *очистка и маркировка* кода. Программный код сканируется и очищается от примесей, таких как неиспользуемые экземпляры или комментарии. Затем выявляются свойства методов программы, элементы методов и связи. Эти элементы и их свойства собираются для получения графового представления характерных особенностей уровня методов. Извлеченный набор характерных особенностей помогает различить две программы (реализованные в изолированных средах) на основе свойств и элементов методов.

Как правило, характерные особенности программ используются для обнаружения исходного кода, когда у двух программ имеется одинаковое происхождение. Однако на основе использования графовых связей можно обнаружить и то, какая программа из сравниваемых программ является исходной: ее граф содержит более строгие и реалистичные связи.

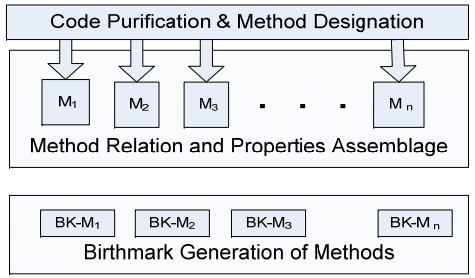

Рис. 1. Процесс формирования набора характерных особенностей
Fig 1. Process for Birthmark Generation

Характерные черты двух программ считаются схожими, если их свойства и связи совпадают при соблюдения условий достоверности. Условие достоверности гласит, что все коды, разработанные в изолированных средах, не должны иметь сходства. Кроме того, если содержание копируемой программы изменяется, такое преобразование также должно раскрываться по набору характерных особенностей (это называется устойчивостью набора характерных особенностей) [7, 8, 9, 10, 11]. Предлагаемый метод обеспечивает обе требуемые черты характерных особенностей, которые обсуждались в подразделе 2.1.

```
Int a;
for(int i=0;i<5; i++)
{
   if(i<3)
        a=1;
   else
        a=3;
}
```

```
char b;
for(int i=0;i<10; i++)
{
   if(i='Z')
        b='A';
   else
        b='Z';
}
```

Рис. 2. Характерные особенности двух разных программ
Fig 2. Code Sample with Birthmark Comparison of two Different Programs

Как показано в [7-9], при традиционном построении наборов характерных особенностей приведенные фрагменты программ будут неразличимы. Следовательно, традиционные методы устойчивость не обеспечивают. Логика этих двух примеров программ различна, и правильный метод построении наборов характерных особенностей может это распознать. Предлагаемый нами метод способен правильно идентифицировать эти программы как отличающиеся одна от другой. Эта обеспечивается за счет выявления отдельных элементов и их свойств, как показано в табл. 1.

3.1 Извлечение набора характерных особенностей

Процесс извлечения характерных особенностей из кода метода состоит с следующем.

- Читается код метода(ов) и определяются элементы в методе(ах).
- Определяются свойства в элементах метода(ов).
- Определяются связи элементов и вычисляются веса ребер.
- На основе весов ребер связи преобразуются в графы.
- На основе связей и свойств формируется набор характерных особенностей.

Генерируемый набор характерных особенностей состоит из различных программных конструкций: элементов метода, свойств элементов и связей между идентифицированными элементами. Сгенерированный набор характерных особенностей используется для выявления степени сходства или несходства рассматриваемых методов.

```
void PrintStack () {
        int i;
        if (count == -1) return;
        else
        {
                for (i=0 i<count; i++)
                {
                        Print (Stack[i]);
                }
        }
}
```

Рис. 3. Фрагмент кода гипотетической программы
Fig 3. Code Sample from Hypothetical Software Program

Поясним суть предлагаемого метода на примере фрагмента кода гипотетической программы *A*, приведенного на рис. 3. Метод *PrintStack ()* в этом фрагменте выводит содержимое стека на консоль (если стек не пуст). В этом фрагменте кода используются две глобальные переменные (*count* и *stack*), локальная переменная метода (*i*), два оператора принятия решения (*if*/*else*) и один цикл. В этом фрагменте элементы кода *i*, *if* и *else* имеют связи по одноуровневости. В то же время между элементом *else* и элементом *for* имеется связь по подчиненности (*parent-child*). Графовое представление элементов и связей показано на рис. 4.

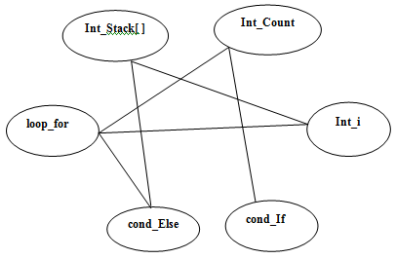

Рис. 5. Граф элементов и связей для фрагмента с рис. 3
Fig 5. Sample pre-birthmark computation graph

После того как для обеих программ (исходной программы и предполагаемой копии) сформированы наборы характерных особенностей (в форме графов), должна быть рассчитана степень сходства этих наборов характерных особенностей (каждый из которых представляет соответствующую программу).

3.2 Выявление сходства

Процесс выявления сходства наборов характерных особенностей (графов) двух программ состоит в следующем.

- Одна из программ полагается оригинальной, а другая – предполагаемой копией.
- В обоих графах выявляются *схожие* элементы.
- Сравниваются коэффициенты кластеризации и значения свойства схожих элементов.
- Вычисляется степень сходства между элементами фрагментов кода.

Процесс выявления сходства может привести к одному из следующих четырех результатов.

a. *Полное сходство*: когда все элементы сформированных характерных черт метода(ов) схожи по свойствам и связям; это означает, что рассматриваемые методы в коде являются «полными копиями» исходных методов, как показано на рис. 4.

b. *Модифицированное сходство*: если копируются все свойства исходной программы, получается модифицированная копия. Методы дублируются для незаконной настройки и маскировки программы.

c. *Предполагаемое сходство*: если некоторые связи и свойства оказываются схожими, копия считается измененной. Этот случай сложно обнаружить, так как можно, а можно и не доказать, что вторая программа является копией.

d. Отсутствие сходства: если нет сходства в свойствах и связях наборов характерных свойств, можно утверждать, что программы имеют различное происхождение.

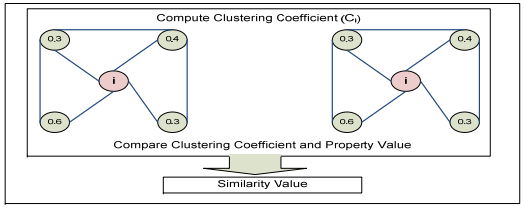

Рис. 4. Полное сходство двух графов
Fig 4. Similarity Comparison of two graph nodes

Первый и последний случаи довольно просты и могут быть достаточно легко проверены, в отличие от второго и третьего случаев, когда имеются частичная модификация или трансформация. С учетом этих сценариев определяются четыре уровня сходства элементов. Уровень 0 подразумевает отсутствие сходства между элементами, уровень 1 представляет предполагаемое сходство, уровень 2 означает, что, возможно, код был модифицирован, а уровень 3 означает полное сходство.

Предполагаемое сходство и сходство с модификацией выявляются с использованием установленного порога. Если ε представляет собой заданный порог, то степень сходства может быть рассчитана по формуле *eq(i)*.

$$P(p_i) \,||\, P(q_i) \;> \varepsilon > P(q_i) \;||\, P(p_i) \qquad eq(i)$$

Уровни сходства 2 и 3 можно различить, измеряя расстояние между свойствами элемента. Расстояние между свойствами элементов *pi* и элементов *qi* вычисляется по формуле *eq(ii)*.

$$\sum_{j=1}^{m} P(p_i) \,||P(q_i) \qquad eq(ii)$$

Если между расстоянием между элементами (p_i, q_i) и (q_i, p_i) появляется определенный порог, то устанавливается сходство на уровне 2, то есть с сходство с модификацией. Когда для обеих программ вычисляется сходство для каждого элемента всего метода, вычисляется расстояние

между свойствами p и q. Рассчитанное расстояние для всех элементов метода позволяет установить сходство на уровне 3, то есть предполагаемое сходство.

4. Анализ и перспективы

Степень сходства различных программ вычислялась на основе взвешенного коэффициента кластеризации разных узлов графа характерных особенностей. Предложенный метод выявления происхождения программ на основе графов оценивается по показателям достоверности, устойчивости, обнаружения измененного кода и самокопирования с целью оценки общей эффективности. Кроме того, был проведен сравнительный анализ с применением нескольких атак, при этом основное внимание уделялось атакам с целью преобразования и модификации кода. Результаты атак трансформации и модификации оценивались с помощью матрицы несоответствий для методов объектов. Каждый из аспектов оценки обсуждается в следующих подразделах.

4.1 Достоверность и устойчивость

Свойство достоверности набора характерных свойств не позволяет идентифицировать сходство независимых программ. Свойство устойчивости позволяет идентифицировать сходство измененных и преобразованных программ. Свойство устойчивости важно, когда программы модифицируются путем атаки на связи, существующие в наборе характерных особенностей. Например, про два независимо разработанных фрагмента кода можно утверждать, что они несходные с нулевым процентом сходства; с другой стороны, фрагменты кода, имеющие 100% сходства, по-видимому являются идеальными копиями. Процент сходства был разделен на 10 равных уровней, уровень 1 показывает сходство от 0% до 10%, уровень 2 – 11-20%, и т.д.; уровень 10 демонстрирует сходство от 90% до 100%.

В табл. 2 приведены результаты выявления сходства выбранных программ с упором на достоверность и устойчивость. Степени сходства рассчитывались для пар независимо разработанных программ и между одной программой и ее полной копией. Результаты показывают, при сравнении программы с ней же самой выявляется наличие совершенной копии, тогда как в других случаях устанавливается полное отсутствие сходства.

Табл. 2. Достоверность и устойчивость при выявлении сходства
Table 2. Similarity Classification for Credibility and Reliability

Пакет Java-программ	Банкомат	Библиотечная система	Торговая точка	Больничная система	Метод k-средних
Банкомат	100	0	0	0	0
Библиотечная система	0	100	0	0	0
Торговая точка	0	0	100	0	0
Больничная система	0	0	0	100	0
Метод k-средних	0	0	0	0	100

Результаты, представленные в табл. 3, в графической форме показаны на рис. 6. На горизонтальной оси показаны программные пакеты банкомата, торговой точки, библиотечной системы и больничной системы. Степень сходства банкоматов с банкоматами составляет 100%, а банкоматов с остальными программами – 0%. Каждая программа разрабатывалась независимо; поэтому между ними не было обнаружено сходство.

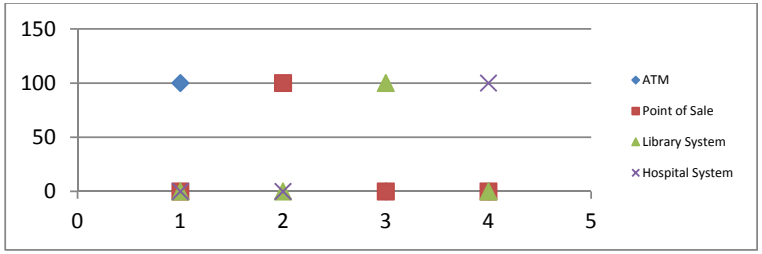

Рис. 6. Результаты достоверности и устойчивости для выбранных систем
Fig 6. Credibility and Reliability results for selected systems.

4.2 Выявление модификаций и трансформаций

Программы модифицируются и трансформируются для того, чтобы скрыть существующие наборы характерных особенностей. Предложенный метод выявляет сходство как преобразованных, так и измененных программы на основе прогнозного анализа. У каждого элемента в наборе характерных особенностей имеется значение свойства, основанное на связях между элементами (узлами), и эти значения используются для выявления сходства в модифицированных и преобразованных кодах. В табл. 3 показана матрица несоответствий для похожих и непохожих объектов. Точность, рассчитанная для похожих и непохожих объектов, составляет 0,90. Коды методов объектов класса были изменены, но не преобразованы.

Табл. 3. Вычисление степени сходства для измененных программ на основе матрицы несоответствия
Table 3. Confusion Matrix Similarity Calculation for Modified Programs

Классы	Похожие объекты	Непохожие объекты	Всего
Похожие объекты	974	36	1010
Непохожие объекты	164	826	990
Всего	1138	862	2000

4.2 Анализ атак

Характерные особенности извлекаются из методов как связи между элементами программы вместе со свойствами элемента. Чтобы предложенный метод обнаружения заимствований не сработал, код (то есть методы) нужно полностью переписать. В результате время, требуемое для модификации кода, может превысить время, необходимое для разработки новой программы. Преобразование небольших программ кажется тривиальным, но преобразовывать крупные программы затруднительно. Кроме того, изменение одного или нескольких методов может оказаться бесполезным для злоумышленника, поскольку у каждого из этих методов имеются зависимости.

В другом сценарии успешной атаки на предложенный метод код можно изменить, добавив в программу дополнительные блоки (внешние циклы, внешние условия) или преобразовав цикл *for* в цикл *while* и т.д. Такие изменения кода правильно обнаруживаются, но устанавливается предполагаемое, а не полное копирование.

Другим известным недостатком предлагаемого метода является невозможность обнаружения преобразования одной переменной в две переменные. Например, переменную a можно заменить на две новые переменные b и c для сохранения значения переменных и дальнейшей обработки (рис. 7). Такие преобразования трудно идентифицировать. Однако такие преобразования увеличивают размер кода и, следовательно, время обработки. Кроме того, такие преобразования отнимают много времени, утомительны, и время, требуемое для модификации или преобразования, может превышать фактическое время разработки.

Поскольку предлагаемый подход извлекает характерные особенности уровня метода, можно определить сходство методов даже при очень низкой степени сходства программ целиком. Поэтому поддержка набора характерных особенностей на уровне методов позволяет противодействовать копированию методов и, следовательно, пиратству в более широком спектре.

Пример	Преобразование
`int a;`	`int a = 10;`
`a = 10;`	`int b = a*a;`
	`int c = b/a;`

Рис. 7. Пример преобразования кода
Fig 7. Example of Code Transformation

4.4 Сравнение с распространенными подходами

Ниже представлены результаты сравнения предложенного подхода с тремя распространенными методами: набор характерных особенностей по полным путям программы (Whole Program Path, WPP) [15], метод на основе K-грамм [16] и динамические наборы характерных особенностей [9]. Для сравнения были выбраны четыре разные и независимые программы из sorceforge.net [34]. Каждая программа сравнивалась сама с собой и с другими тремя программами. Сравнение подходов проиллюстрировано на рис. 8. Как видно из результатов, предложенный метод на основе графа работает лучше, чем другие методы, когда программный код сравнивается с самим собой. Причиной лучшей эффективности нашего подхода является учет связей и свойств каждого узла в графе (наборе характерных особенностей).

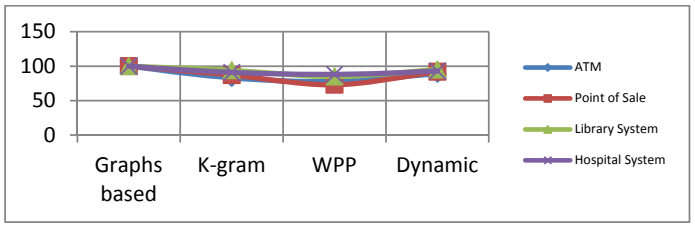

Рис. 8. Сравнение с распространенными методами
Fig 8. Comparison with Prevalent Techniques

Далее в коде программы были скопированы различные методы для проверки возможности выявления сходства методов в рассматриваемой программе. Экспериментальные результаты показаны в табл, 4. При использовании других методов статического формирования наборов характерных особенностей классификация методов была невозможна.

Табл. 4. Обнаружение копирования методов
Table 4. Method Copy Detection Comparison

Пакеты программ	Метод на основе графов	K-граммы	WPP	Динамический метод
Банкомат	30	0	0	0
Торговая точка	20	0	0	0
Библиотечная система	10	0	0	0
Больничная система	10	0	0	0

Например, в программном пакете банкомата 30% методов были классифицированы как «скопированные». Другие методы копирование вообще не определяют. Таким образом, можно утверждать, что предложенный метод является эффективной мерой против компьютерного пиратства для обнаружения копирования методов и программ целиком.

Как отмечалось в разд. 3, в методе, основанном на графах, используется структура метода с существующими связями между элементами программы. Сравнительно просто можно выполнить преобразование специального вида, скрывающее копирование кода. Поэтому предлагаемый метод может быть неэффективной мерой защиты от копирования для программ, в которых содержатся методы небольшого размера. При наличии крупных методов специальное преобразование кода является громоздким, и время, необходимое для преобразования каждого элемента метода, может превышать время разработки нового метода.

5. Заключение и направления будущих исследований

Для обнаружения незаконного использования прав собственности на программное обеспечение был предложен новый подход к формированию наборов характерных особенностей программ, основанный на графах. Уникальные характеристики кода (элементов методов) с соответствующими связями преобразуются в графы для выявления сходства или различия программ. Характерные особенности уровня методов дополняют другие методы выявления сходства. Сравниваются два набора характерных особенностей, и элементу набора характерных особенностей присваивается показатель сходства с учетом его глубины. На основе таких вычисления выявляется один из четырех уровней сходства двух программ. Предложенный метод соответствует принципам надежности и устойчивости наборов характерных особенностей программ. С помощью предложенного подхода в коде также могут быть обнаружены измененные элементы. Предложенный подход сравнивался с распространенными методиками; сравнение показывает, что предложенный метод классификации превосходит другие методы, даже если потоки исполнения программы не меняются, но логика кода является преобразованной.

В будущем мы планируем выполнить эксперименты с динамическим решением для обнаружения незаконного использования программного обеспечения с использованием гибридных наборов характерных особенностей программ и водяных знаков.

Список литературы / References

[1]. Ninth Annual BSA Global Software 2013 Piracy Study, Business Software Alliance (BSA), 2013.

[2]. Anckaert B., De Sutter, D. Chanet, and Bosschere K. Steganography for executables and code Transformation Signatures. Lecture Notes in Computer Science, volume 3506, 2005, pp. 425–439.

[3]. Fu B., Richard G., and Chen Y. Some New Approaches for Preventing Software Tampering. In Proc. of the 44th Annual Southeast regional conference, pp. 655–660, 2006.

[4]. Collberg S., Thomborson C. Watermarking, tamper-proofing, and obfuscation-tools for software protection. IEEE Transactions on Software Engineering, vol. 28, issue 8, 2002, pp. 735–746.

[5]. Udupa S. K., Debray K., and Madou M. Deobfuscation: Reverse engineering obfuscated code. In Proc. of the 12th Conference on Reverse Engineering, 2009, pp. 10–19.

[6]. Palsberg J., Krishnaswamy S., Kwon M. Experience with software watermarking. In Proc. of the 16th Computer Security Applications Conference, 2003, pp. 308–316.

[7]. Bai Y., Sun X., Sun G., Deng X., and Zhou X. Dynamic k-gram based software birthmark. In Proc. of the 19th Australian Conference on Software Engineering, 2008, pp. 644–649.

[8]. Mahmood Y., Pervez Z., Sarwar S., and Ahmed H. F. Similarity Level Method Based Static Software Birthmarks. In Proc. of the International Symposium on High Capacity Optical Networks and Enabling Technologies, 2008, pp. 205–210.

[9]. Schuler D., Dallmeier V., Lindig C. A dynamic birthmark for java. In Proc. of the 22nd IEEE/ACM International conference on Automated software engineering, 2007, pp. 274–283.

[10]. Nazir S., Shahzad S., Binti N., Alias, and Anwar S. A Novel Rules Based Approach for Estimating Software Birthmark. The Scientific World Journal, 2015.

[11]. Jorge E. N., Pirmez L., Costa O., Boccardo R., Bento M. Tiny Watermark: a code obfuscation-based software watermarking framework for wireless sensor networks. In Proc. of the ICWN'14-The 2014 International Conference on Wireless Networks, 2014.

[12]. Nayakoji N., Sonavane S. JavaScript Theft Detection using Birthmark and Subgraph Isomorphism. Journal of Engineering Computers & Applied Sciences, vol. 3, no. 8, 2014, pp. 1–5.

[13]. Che S. and Wang Y. A Software Watermarking Based on PE File with Tamper-proof Function. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 12, no. 2, 2014, pp. 1012–1021.

[14]. Zhu F. Concepts and techniques in software watermarking and obfuscation. PhD Thesis, The University of Auckland, New Zealand, 2007, 160 p.

[15]. Myles G., and Collberg C. Detecting software theft via whole program path birthmarks. Lecture Notes in Computer Science, vol. 3225, 2004, pp. 404–415.

[16]. Myles G. and Collberg C. K-gram based software birthmarks. In Proc. of the 2005 ACM symposium on Applied computing, 2005, pp. 314–318.

[17]. Tian Z., Liu T., Zheng Q. A new thread-aware birthmark for plagiarism detection of multithreaded programs. In Proc. of the 38th International Conference on Software Engineering Companion, 2016, pp. 734–736.

[18]. Tian Z., Liu T., Zheng Q. Exploiting Thread-Related System Calls for Plagiarism Detection of Multithreaded Programs. Journal of Systems and Software, vol. 119, 2016, pp. 136-148.

[19]. Bhattacharya S. Survey on Digital Watermarking–A Digital Forensics & Security Application. International Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, issue 11, 2014.

[20]. Khan A., Siddiqa A., Munib S. A recent survey of reversible watermarking techniques. Information Sciences, vol. 279, 2014, pp. 251–272.

[21]. Zhou W., Zhang X., and Jiang X. AppInk: watermarking android apps for repackaging deterrence. In Proc. of the 8th ACM SIGSAC symposium on Information, computer and communications security, 2013, pp. 1–12.

[22]. Ren C., Chen K., and Liu P. Droidmarking: resilient software watermarking for impeding android application repackaging. In Proc. of the 29th ACM/IEEE international conference on Automated software engineering, 2014, pp. 635–646.

[23]. Guang S., Xiaoping F., Sha F., Yingjie S. Software Watermarking in the Cloud: Analysis and Rigorous Theoretic Treatment. Journal of Software Engineering, vol. 9, issue 2, 2015, pp. 410–418.

[24]. Rubini P., Leela S. A Survey On Plagiarism Detection In Text Mining. International Journal of Research in Computer Applications and Robotics, vol. 1, issue 9, 2013, pp. 117-119.

[25]. Oberreuter G., and Velásquez J. D. Text mining applied to plagiarism detection: The use of words for detecting deviations in the writing style. Expert Systems with Applications, vol. 40, issue 9, 2013, pp. 3756–3763.

[26]. Rana H., Stamp M. Hunting for Pirated Software Using Metamorphic Analysis. Information Security Journal: A Global Perspective, vol. 23, issue 3, 2014, pp. 68–85.

[27]. Costa M., Gong Z. Web structure mining: an introduction. In Proc. of the 2005 IEEE International Conference on Information Acquisition, 2005, pp. 590-595.

[28]. Vemparala S., Di Troia F., Corrado V., Austin H., and Stamo M. Malware Detection Using Dynamic Birthmarks. In Proc. of the 2016 ACM- International Workshop on Security And Privacy Analytics, 2016, pp. 41–46.

[29]. Zeng K., and Athanas P. A q-gram birthmarking approach to predicting reusable hardware. Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016, pp. 1112–1115.

[30]. Bogdanov P., Baumer B., Basu P. As Strong as the Weakest Link: Mining Diverse Cliques in Weighted Graphs. Lecture Notes in Computer Science, vol. 8188, 2013, pp. 525–540.

[31]. Getoor L., Diehl P. Link mining: a survey. ACM SIGKDD Explorations Newsletter, vol. 7, issue 2, 2005, pp. 3–12.

[32]. Kavitha D., Rao M., Babu K. A Survey on Assorted Approaches to Graph Data Mining, International Journal of Computer Applications, vol. 14, no. 1, 2011, pp. 43–46.

[33]. Tamada H., Nakamura M., Monden A. Java Birthmarks–Detecting the Software Theft. IEICE transactions on information and systems, Vol. E88-D, issue 9, 2005, pp. 2148–2158.

[34]. Sorceforge.net, accessed on Nov 2, 2017.

[35]. Fan M., Liu J., Luo X. et al. Android Malware Familial Classification and Representative Sample Selection via Frequent Sub-graph Analysis, IEEE Transactions on Information Forensics and Security, vol. 13, issue 8, 2018, pp. 1890-1905.

[36]. Tian Z., T. Liu, Zheng Q. et al. Reviving Sequential Program Birthmarking for Multithreaded Software Plagiarism Detection. IEEE Transactions on Software Engineering, vol. 44, issue 5, 2018, pp. 491-511.

Информация об авторах / Information about authors

Сохаил САРВАР получил степень магистра в области информационных технологий в Национальном университете науки и технологии, Исламабад, Пакистан. Получил степень PhD на компьютерном факультете университета Гуджарата, Пакистан. Исследовательские интересы включают электронное обучение, семантические технологии и методы инженерии знаний.

Sohail SARWAR received the M.S. degree in information technology from National University of Science and Technology, Islamabad, Pakistan, and the Ph.D. degree in the Department of Computing, University of Gujrat, Pakistan. His research interests include e-learning, semantic technologies and knowledge engineering techniques.

Зия УЛЬ КАЙУМ в настоящее время является профессором в университете Гуджарата в Пакистане. Он получил степень PhD в области компьютерных наук в Университете Лидса, Великобритания, в 2005 году. В число научных интересов входят искусственный интеллект, инженерия знаний, интеллектуальный анализ знаний, семантический Web и электронное обучение.

Zia UL QAYYUM is currently a professor at University of Gujrat, Pakistan. He received his Ph.D. degree in computer science from Leeds University UK in 2005. His research interests include artificial intelligence, knowledge engineering, data mining, semantic web and e-learning.

Мухаммад САФЬЯН работает в Правительственном университете колледжа, Лахор, Пакистан. Он получил степень магистра в 2009 г. Национальном университете науки и технологии. Область его научных интересов включает отображение онтологий, электронное обучение, семантическое распознавание активностей.

Muhammad SAFYAN is in Government College University (GCU), Lahore. He received his MS degree from National University of Sciences and Technology in 2009. His area of interest is ontology alignment, e-learning and semantic activity recognition.

Муддессар ИКБАЛ работает старшим преподавателем в Лондонском университете Саут Бэнк и в университете Эссекса, Великобритания. Он получил степень PhD в Кингстонском университете, Лондон, Великобритания. Его научные интересы включают сетевые технологии 5G, мультимедийные облачные вычисления, мобильные граничные вычисления, туманные вычисления, интернет вещей, программно-конфигурируемые сети, виртуализацию сетевых функций, качество восприятия, облачные инфраструктуры и службы.

Muddessar IQBAL is working as senior lecturer in London South Bank University and University of Essex, England. Dr. Iqbal completed his PhD from Kingston University in 2010. His research interests include 5G networking technologies, multimedia cloud computing, mobile edge computing, fog computing, Internet of Things, software-defined networking, network function virtualisation, quality of experience, and cloud infrastructures and services.

Ясир МАХМУД работает преподавателем в Правительственном университете колледжа, Лахор, Пакистан. Он получил степень магистра в Национальном университете науки и технологии. Область его научных интересов включает отображение онтологий, электронное обучение, защиту программного обеспечения.

Yasir MAHMOOD is a lecturer in Government College University (GCU), Lahore. He received his MS degree from National University of Sciences and Technology. His area of interest is ontology alignment, e-learning and software protection.