УДК 004.89

А.В. Куртукова, А.С. Романов, А.А. Шелупанов

Разработка методики идентификации авторства бинарных и дизассемблированных кодов программы на основе ансамбля современных методов обработки естественного языка

Данная статья является частью цикла исследований, направленных на решение проблем идентификации авторства программного кода. Анализ бинарного или дизассемблированного кода является важнейшей задачей информационной безопасности, разработки программного обеспечения и компьютерной криминалистики ввиду необходимости защиты результатов интеллектуальной деятельности и авторского права, а также определения авторов вредоносных программ. Любая программа представляет собой машинный код, который может быть дизассемблирован (преобразован в текст на языке ассемблера) при помощи специализированных инструментов и проанализирован на предмет авторства по аналогии с текстом на естественном языке. Для решения обозначенной проблемы в статье предлагается методика на основе ансамбля fastText, метода опорных векторов (SVM) и авторской гибридной нейронной сети. Предложенная методика оценивалась на исходных кодах на языках C и C++, собранных с платформ GitHub и Google Code Jam, скомпилированных в исполняемые файлы и дизассемблированных инструментами реверс-инжиниринга. Средняя точность идентификации автора дизассемблированного кода предложенной методикой составила более 0,9. Методика также была апробирована на исходных кодах, в результате чего средняя точность составила 0,96 для простых случаев и более 0,85 для сложных (обфускация, стандарты кодирования и др.).

Ключевые слова: исходный код, машинное обучение, автор, нейронные сети, ансамбль, дизассемблер.
DOI: 10.21293/1818-0442-2023-26-4-53-60

Идентификация автора программного кода является важнейшей задачей в цифровой криминалистике [1] и обнаружении плагиата [2]. Решения этой задачи особенно полезны при судебных разбирательствах, связанных с вопросами интеллектуальной собственности и авторских прав, а также при расследованиях случаев разработки и распространения вредоносного программного обеспечения. Существующие методы идентификации автора компьютерной программы можно разделить на три группы: анализирующие исходный код [3–7], ассемблерный код дизассемблированной программы [8–12] и универсальные методы, применимые в обоих случаях [13, 14]. Хотя эти методы основаны на разных алгоритмах и подходах, все они имеют общий принцип работы: определение уникального стиля кодирования автора-программиста.

Уникальный стиль программиста характеризуется шаблонами проектирования, языковыми конструкциями, форматированием блоков кода, стилем комментариев к коду, идентификаторами, переменными, наименованиями функций, «запахами кода» [15] – фрагментами кода, не соответствующими правилам написания кода на языке, используемом автором программы. Перечисленные характеристики представляют исходный код программы, а не дизассемблированный. Однако некоторые из них остаются распознаваемыми даже после компиляции и дизассемблирования программы. Таким образом, определение авторства на основе дизассемблированного кода является более методологически сложным и требует более современных решений, адаптированных к этому виду анализа.

Целью данного исследования является разработка комплексного решения на основе алгоритмов обработки естественного языка, позволяющего с высокой точностью идентифицировать автора программы, используя как исходные, так и дизассемблированные коды.

Обзор литературы

При разработке и совершенствовании методики идентификации автора программного кода важно учитывать опыт, полученный исследователями ранее, и принимать во внимание преимущества, недостатки и ограничения тех или иных подходов.

Задача анализа программного кода с целью установления авторства имеет несколько фундаментальных решений, обеспечивающих приемлемый для их использования в прикладных задачах результат.

Большая часть методов [12–14], помимо основанных на глубоких нейронных сетях (НС) [16], оперирует множеством признаков, предложенным в исследовании [9]. Идея состоит в применении графов семантического потока (SFG) и потока управления (CFG), полученных путем реверс-инжиниринга, с целью получения из них последовательной трассировки операций. Итоговое множество признаков, полученное таким образом, включает вызовы библиотек, идиомы, графлеты, *n*-граммы и шаблоны функций.

Согласно результатам анализа аналогов разрабатываемой методики, такое множество является достаточным для получения высокой точности (более 0,7 для наборов данных Google Code Jam (GCJ) [17] и Codeforces [18]) простыми статистическими моделями, такими как метод опорных векторов (SVM) и абстрактные синтаксические деревья (АСД).

Исходя из анализа трудов [3–15], можно сделать вывод о том, что при решении задачи идентификации автора программного кода следует учитывать ряд факторов:

1. Большинство признаков, которые используются для определения авторства программы, описывают ее функциональность, а не авторский стиль, что может негативно сказаться на обобщающей способности классификатора.

2. При формировании признакового пространства важно предусмотреть фильтрацию для повышения информативности отдельно взятых признаков и их отделения от шума.

3. Большое влияние на оценку эффективности подхода оказывает домен данных. Зачастую исследователи используют программные коды с соревнований либо устаревшие наборы данных, находящиеся в открытом доступе, что негативно сказывается на объективности оценки и не позволяет оценить обобщающую способность подходов.

Следует отметить, что несмотря на безусловную эффективность подходов на основе CFG и SFG совместно с классическими алгоритмами машинного обучения, их использование в качестве основы для универсальной методики идентификации автора программного кода недопустимо. Это обусловлено двумя аспектами:

1. Подходы адаптированы под анализ дизассемблированных кодов и не применимы к исходным кодам.

2. Трудоемкость процесса и необходимость экспертных знаний низкоуровневых языков для его осуществления.

Нейросетевые подходы и ансамбли на их основе могут стать эффективной и универсальной альтернативой, не требующей дополнительных временных затрат на формирование релевантного множества признаков.

Набор данных

Для объективной оценки эффективности реализуемых подходов был необходим объемный и репрезентативный набор данных. При формировании набора обязательным было условие, что он не должен содержать данные, принадлежащие одному домену (как соревновательные данные GCJ). Это обусловлено, во-первых, необходимостью получить точную оценку и избежать ее искусственного завышения за счет решения одинаковых задач одними и теми же авторами-программистами, во-вторых, важностью реализации подхода с высокой обобщающей способностью без привязки к специфике данных. Так было решено сформировать собственный набор бинарных кодов программ.

Набор данных был собран при помощи интерфейса программирования приложения (API) открытого хостинга IT-проектов GitHub. В его состав вошли исходные коды, соответствующие требованиям:

– язык программирования – C и C++;

– длина кода не менее 5 строк;

– не менее 20 файлов с исходным кодом в репозиториях автора;

– возможность компиляции с помощью GCC;

– наличие сопроводительного файла с информацией о настройках компиляции.

Отобранные исходные коды были скомпилированы при помощи коллекции компиляторов GNU (GCC) [19], а затем дизассемблированы интерактивным дизассемблером (IDA) Pro [20]. Подробная информация об исходных и дизассемблированных кодах представлена в табл. 1.

Таблица 1

Набор данных GitHub

Характеристика	Исходные коды	Дизассемблированные коды
Общее число авторов	167	
Общее число файлов	12 779	5 661
Ср. число файлов автора	63	25
Макс. число файлов автора	132	51
Мин. число файлов автора	20	20
Ср. число строк кода	146	1 677

Так как целевым направлением применения методики является информационная безопасность, было решено сформировать релевантный этой тематике набор данных. В него вошли вредоносные исполняемые файлы известных хакерских группировок (рис. 1). Данные были получены на основании отчетов организаций, занимающихся обеспечением кибербезопасности, и дизассемблированы аналогично данным GitHub.

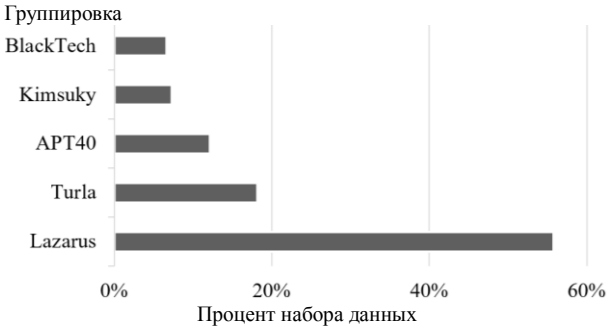

Рис. 1. Статистика набора данных вредоносных кодов

Предварительные эксперименты

Ранние исследования, посвященные решению задачи идентификации автора дизассемблированного кода программы, позволили оценить возможность применения наиболее эффективных моделей, исходя из смежных работ [21–23]: гибридной НС (HNN) на основе архитектур Inception-v1 и двунаправленных рекуррентных блоков, метода опорных векторов (SVM) и fastText.

В качестве метрики применялась точность, рассчитанная в результате процедуры 10-кратной перекрестной проверки. Обобщенные результаты эксперимента с дизассемблированными данными GitHub представлены в табл. 2.

Таблица 2

Результаты экспериментов GitHub

Число авторов	HNN	SVM	fastText
2	0,84	0,83	0,91
5	0,78	0,78	0,86
10	0,66	0,62	0,79

На основании полученных значений точности была рассчитана статистическая значимость результатов при помощи непараметрических апостериорных тестов Фридмана и Неменьи. Для оценки разницы между моделями использовался апостериорный тест Неменьи. Графическая интерпретация Дешмара представлена на рис. 2, где график *а* демонстрирует результаты для 2 авторов, *б* – для 5 и *в* – для 10 соответственно.

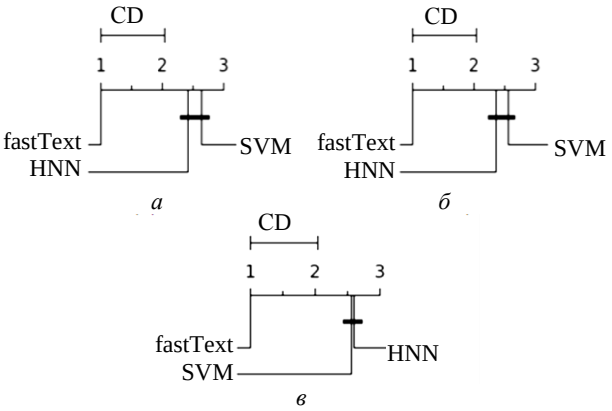

Рис. 2. Результаты для
2 авторов – *а*, 5 авторов – *б*, 10 авторов – *в*

Тесты показали, что эффективности HNN и SVM приблизительно равны, так как разница между их средними рангами меньше расчетного значения критической разницы (CD). Результат, полученный fastText, напротив, оказался значимым и несопоставимым с другими моделями.

Еще один эксперимент был выполнен на основе набора вредоносных исполняемых файлов, разработанных хакерскими группировками. Полученные результаты представлены в табл. 3.

Таблица 3

Результаты экспериментов на вредоносных кодах

Число авторов	HNN	SVM	fastText
2	0,89	0,9	0,94
3	0,85	0,87	0,9
4	0,82	0,8	0,85
5	0,76	0,74	0,79

Исходя из результатов предварительных экспериментов, был сделан вывод об особой эффективности fastText для идентификации автора дизассемблированного кода программы. Однако применение fastText в качестве универсального метода, исходя из ранних работ [21–23], не представлялось возможным из-за его низкой в сравнении с HNN эффективности для анализа исходных кодов программ.

Предварительные эксперименты позволили выдвинуть гипотезу об эффективности ансамбля классификаторов, состоящего из HNN, SVM и fastText, как универсального метода решения задачи идентификации автора программы на основе исходного или дизассемблированного кода.

Универсальная методика идентификации автора программы по его исходным или дизассемблированным кодам

Исходя из выдвинутой на основе предварительных экспериментов гипотезы об эффективности ансамбля классификаторов, была предложена усовершенствованная методика идентификации автора программы (рис. 3).

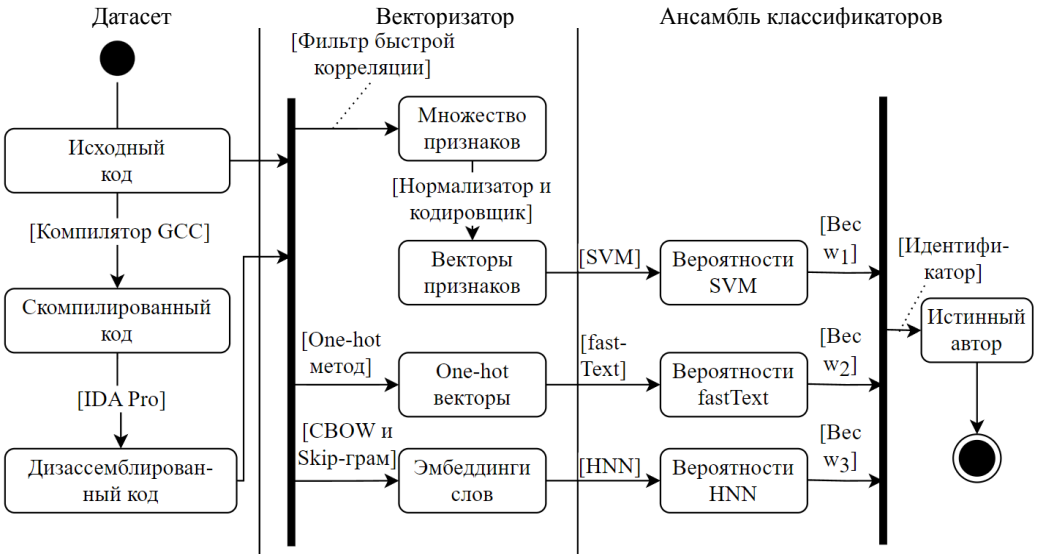

Рис. 3. Методика идентификации автора программы на основе ансамбля SVM, fastText, HNN

Первый этап посвящен подготовке обучающих данных к векторизации. Для случая идентификации автора по исходному коду файл сразу передается в модуль векторизации и не подлежит предобработке. Для случая идентификации автора по дизассемблированному коду исходный код компилируется с помощью GCC, а затем дизассемблируется IDA Pro.

Полученный в результате реверс-инжиниринга файл передается в модуль векторизации.

Второй этап заключается в преобразовании обучающих данных в векторный вид. Каждый из классификаторов требует соответствующий ему формат входных данных:

– SVM принимает на вход множество нормализованных и векторизованных признаков. В качестве признакового пространства для SVM использовались паттерны, описанные в исследовании [9].

– HNN работает с текстом как исходного, так и дизассемблированного кода без какой-либо предварительной обработки. Текст в его первоначальном виде передается прямому посимвольному кодировщику. Прямое посимвольное кодирование создает вектор из 255 нулей и 1 единицы для каждого отдельно взятого символа текста. При этом единица устанавливается на позиции, соответствующей коду символа, согласно американскому стандартному коду обмена информацией (ASCII).

– fastText также принимает на вход исходные коды как текст. Представления слов (эмбеддинги) создаются fastText автоматически при помощи кодирования непрерывным мешком слов (CBOW).

Последний этап состоит в обучении ансамбля классификаторов для последующей идентификации автора программы. Оптимальные гиперпараметры для классификаторов SVM и HNN были определены экспериментально при помощи жадного поиска, а для fastText использовалась встроенная в инструмент функция автовалидации.

Для SVM использовались следующие параметры:

– тип – многоклассовая классификация;

– алгоритм обучения – метод последовательной оптимизации;

– ядро – сигмоидальное;

– параметр регуляризации C = 1;

– допустимый уровень ошибки = 0,00001.

В качестве параметров для HNN были выбраны:

– функция оптимизации – Adadelta;

– функция активации – Softmax;

– функция прореживания – Dropout(0,2);

– промежуточная функция активации – ReLU;

– leaning rate = 10^{-4};

– rho = 0,95;

– eps = 10^{-7}.

Большинство параметров fastText по умолчанию оказались оптимальными:

– learning rate = 0,9;

– threads = 15;

– lrUpdateRate = 5.

Главный принцип работы ансамбля состоит в применении весовых коэффициентов, представленных на схеме как w_1, w_2 и w_3, к результатам работы отдельных классификаторов. Для анализа дизассемблированных кодов наибольший вес (w_3 = 0,4) присваивается решениям fastText как наиболее точного классификатора для данного случая. Для решений, принятых двумя другими классификаторами, устанавливаются равные веса: w_1 = 0,3 для SVM и w_2 = 0,3 для HNN соответственно. Таким образом, наибольшую ценность будут представлять вероятности, полученные от fastText. Единственный случай, когда решением fastText можно будет пренебречь, это совпадение в вероятностях HNN и SVM. Тогда их суммарный коэффициент составит 0,6 против коэффициента fastText 0,4.

Для анализа исходных кодов программ наибольший вес присваивается решениям HNN (w_2 = 0,4). А решениям классификаторов fastText и SVM как менее эффективным в сложных случаях присваиваются равные веса. Аналогично случаю с бинарными кодами: w_3 = 0,3 для fastText и w_1 = 0,3 для SVM соответственно. Принцип идентификации автора исходного кода таким ансамблем аналогичен анализу дизассемблированного кода.

Апробация методики

Разработанная методика была апробирована на исходных и дизассемблированных кодах программ. Эксперименты были проведены как для простых, так и для сложных случаев идентификации автора. Число спорных авторов для части экспериментов было повышено до 20. Результаты работы ансамбля классификаторов для анализа дизассемблированных кодов представлены в табл. 4. Для анализа исходных кодов в простых случаях – в табл. 5.

Таблица 4

Результаты для дизассемблированных кодов

Число файлов	10	20	30
5 авторов	0,86	0,9	0,96
10 авторов	0,85	0,87	0,93
20 авторов	0,78	0,83	0,86

Таблица 5

Результаты для простых исходных кодов

Число файлов	10	20	30
5 авторов	0,95	0,97	0,98
10 авторов	0,93	0,95	0,96
20 авторов	0,92	0,95	0,96

В результате применения ансамбля к дизассемблированным кодам был получен прирост в точности более 0,1 в сравнении с применением классификаторов по отдельности. При достаточном количестве обучающих файлов точность превышает 0,9.

Результаты, полученные ансамблем для исходных кодов, сопоставимы с полученными ранее [23], т.е. ансамбль негативного влияния на данный случай не оказывает.

Помимо простых случаев идентификации автора исходного кода программы, важно проверить и сложные, чтобы убедиться в устойчивости усовершенствованной методики. К таким случаям относятся обфускация, следование стандартам кодирования, командная разработка и добавление искусственно сгенерированных кодов.

В табл. 6 представлены результаты идентификации автора обфусцированного исходного кода. В качестве обфускатора использовался AnalyzeC [24]. Процесс обфускации AnalyzeC предполагает:

– полное удаление комментариев и пробелов;

– добавление псевдосложного кода, не изменяющего функциональность программы;

– директив препроцессора;

– замену строк шестнадцатеричным эквивалентом.

Результаты, полученные ансамблем, сопоставимы с полученными ранее [23], т.е. ансамбль негативного влияния на этот случай не оказывает.

Таблица 6

Результаты для обфусцированных исходных кодов

Число файлов	10	20	30
5 авторов	0,85	0,9	0,91
10 авторов	0,72	0,78	0,85
20 авторов	0,64	0,72	0,75

Следующий случай – идентификация автора исходного кода, написанного командой разработчиков. В этом случае программисты используют систему управления версиями (GitLab, GitHub) и фиксируют изменения в репозитории проекта с помощью коммитов. Исходный код одной программы может содержать признаки сразу нескольких авторов. Поэтому возможность установления авторства на основании коммитов особенно важна.

Информация о коммитах, их содержимом и авторах была получена в процессе сбора данных при помощи API GitHub. В табл. 7 представлены результаты идентификации автора исходного кода, сформированного из коммитов. Прирост точности, полученный ансамблем классификаторов вместо отдельной HNN [23], составил в среднем 0,03.

Таблица 7

Результаты для обфусцированных исходных кодов

Число файлов	10	20	30
5 авторов	0,89	0,94	0,97
10 авторов	0,86	0,91	0,94
20 авторов	0,83	0,87	0,91

Случай, вызвавший наибольшие сложности в ранних исследованиях, заключается в использовании исходного кода, написанного в соответствии со стандартами кодирования (табл. 8). Такой код пишется разработчиками с целью упрощения поддержки, а также улучшения читабельности кода, но сводит к минимуму уникальные признаки автора-программиста. Для оценки использовались исходные коды Linux Kernel [25], написанные на C/C++, в соответствии с общепринятыми стандартами. Для данного случая также был получен прирост, составивший в среднем 0,03, в сравнении с использованием HNN в отдельности [23].

Таблица 8

Результаты для исходных кодов, написанных по стандартам кодирования

Наличие стандарта	Число файлов	5 000	7 000	10 000
Без стандартов	10	0,76	0,83	0,89
Со стандартами		0,42	0,48	0,62
Без стандартов	20	0,92	0,95	0,96
Со стандартами		0,69	0,76	0,83
Без стандартов	30	0,95	0,97	0,98
Со стандартами		0,75	0,84	0,89

Последний сложный случай вызван ростом популярности моделей семейства Generative Pre-trained Transformer (GPT) и их эффективностью для генерации исходных кодов программ. Эксперименты были проведены для наиболее сложного случая анализа искусственно сгенерированного кода – разграничение авторства между разными генеративными моделями – GPT-3, GPT-2 и RuGPT-3 (табл. 9). В данном случае использование ансамбля оказало положительное влияние и дало прирост в точности более 0,07 в сравнении с HNN по отдельности [23].

Таблица 9

Результаты для искусственно-сгенерированных кодов

10 файлов	20 файлов	30 файлов
0,86	0,9	0,94

Для того чтобы убедиться в том, что усовершенствованная методика не вносит негативного эффекта в сложных случаях в сравнении с простыми, было решено провести T-тест для парных выборок. Его смысл заключается в попарном сравнении результатов перекрестной проверки в простых и сложных случаях и оценке *p*-значения для каждого из них в отдельности. Нулевая гипотеза, состоящая в том, что разница не является статистически значимой, принимается при *p*-значениях свыше 0,05. Альтернативная гипотеза предполагает критическую потерю в точности. Итак, для пары результатов «простой исходный код» – «обфусцированный исходный код» *p*-значение составило 2,35. Для пары «простой исходный код» – «коммит» – 0,06. Для пары «простой исходный код» – «искусственно сгенерированный исходный код» – 0,88. И для пары «простой исходный код» – «код, написанный по стандартам кодирования» – 1,83. Ни одна из пар не дала результата ниже границы в 0,05, т.е. разница в точности между простыми и сложными случаями критической не является.

Также был проведен ряд экспериментов, чтобы удостовериться, что усовершенствованная методика не уступает в точности аналогам, разработанным другими исследователями. Так как в большинстве работ для оценки своих подходов они использовали набор данных GCJ, было решено провести дополнительные эксперименты по анализу дизассемблированных кодов с помощью усовершенствованной методики. В набор были включены данные GCJ 2009 и 2010 для более объективного сравнения. Метрики и число авторов были взяты в соответствии с указанными в статьях. Следует отметить, что в одной из работ была использована отличная от точности метрика эффективности [15]. Метрика F0,5, применяемая авторами, рассчитывается как

$$F0{,}5=\frac{1{,}25\times(\text{precision}\times\text{recall})}{0{,}25\times(\text{precision}+\text{recall})}. \qquad (1)$$

Результаты данных экспериментов приведены в табл. 10. Из таблицы видно, что усовершенствованная методика на основе ансамбля классификаторов не уступает методам, предложенным другими исследователями ранее. В части случаев ансамбль проде-

монстрировал значительный прирост в точности. Кроме того, полученные ансамблем на GCJ результаты оказались лучше полученных на GitHub. Это объясняется тем, что оценка точности классификаторов на данных GCJ не является достаточно объективной из-за их специфики. Такие данные являются однородными и позволяют классификатору сосредоточиться только на авторских признаках, игнорируя функциональные отличия программ и их специфику. В отличие от GCJ, набор данных GitHub состоит из неоднородных данных (разный опыт программистов и многообразие решаемых задач), а эксперименты моделируют решение реальных задач. Это делает оценку более объективной.

Таблица 10

Результаты для исходных кодов, написанных по стандартам кодирования

Работа	Число авторов	Метрика	Результат авторов	Наш результат	Разница в эффективности
Alrabaee S. [17]	5	F0,5	0,8	0,96	+ 0,16
	10	F0,5	0,76	0,93	+ 0,17
	20	F0,5	0,71	0,88	+ 0,17
Rosenblum N. [9]	5	Точность	0,93	0,97	+ 0,04
	20	Точность	0,77	0,87	+ 0,1
Alrabaee S. [19]	3	Точность	0,93	0,99	+ 0,06
	5	Точность	0,9	0,97	+ 0,07
	7	Точность	0,82	0,96	+ 0,14

Заключение

Согласно полученным результатам, исходную методику на основе HNN удалось усовершенствовать, адаптировав под анализ дизассемблированных кодов программ, и сделать универсальное решение на основе ансамбля классификаторов.

Исходя из проведенных исследований, можно выделить следующие преимущества методики:

1. Универсальность. Возможность определять автора как бинарного, так и исходного кода на основе текста программы и выделенных признаков.

2. Эффективность. Точность методики независимо от сложности задачи и специфики данных во всех экспериментах превышает 0,85. Этого достаточно для использования методики при решении практических задач.

3. Независимость от осложняющих факторов. Методика устойчива как к преднамеренным (обфускация, стандарты кодирования, командная разработка, искусственная генерация), так и к непреднамеренным (изменения в стиле, связанные с увеличением опыта и повышением квалификации программиста) искажениям кода программы.

К ограничениям методики, а также возможностям ее дальнейшего совершенствования можно отнести следующие аспекты. Во-первых, для достижения высокой точности идентификации автора дизассемблированного кода программа должна быть предварительно деобфусцирована, так как даже минимальная обфускация существенно снижает эффективность методики. Во-вторых, результаты напрямую зависят от количества авторов и обучающих данных на каждого автора. При увеличении числа авторов или нехватке числа обучающих экземпляров эффективность системы снижается.

Данная работа выполнена при финансовой поддержке Министерства науки и высшего образования РФ в рамках базовой части государственного задания ТУСУРа на 2023–2025 гг. (проект № FEWM-2023-0015).

Литература

1. A Forensic Traceability Index in Digital Forensic Investigation / S. Rahayu, S. Shahrin, N. Hafeizah, R. Yusof, M.F. Abdollah // Journal of Information Security. – 2013. – Vol. 4, No. 1. – P. 19–32.

2. Schleimer S. Winnowing: local algorithms for document fingerprinting / S. Schleimer, D.S. Wilkerson, A. Aiken // Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD '03). – Association for Computing Machinery, NewYork, NY, USA, 2003. – P. 76–85.

3. Large-Scale and Language-Oblivious Code Authorship Identification / M. Abuhamad, T. AbuHmed, A. Mohaisen, D. Nyang // Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. – Toronto, ON, Canada, 2018. – P. 101–114.

4. RoPGen: Towards Robust Code Authorship Attribution via Automatic Coding Style Transformation / L. Zhen, G. Chen, C. Chen, Y. Zou, S. Xu // Proceedings of the 2022 IEEE 44th International Conference on Software Engineering (ICSE). – Pittsburgh, PA, USA, 2022. – P. 1906–1918.

5. Holland C. Code authorship identification via deep graph CNNs / C. Holland, N. Khoshavi, G. Jaimes // Proceedings of the 2022 ACM Southeast Conference (ACM SE ‘22). – 2022. – P. 144–150.

6. Bogomolov E. Authorship attribution of source code: A language-agnostic approach and applicability in software engineering / E. Bogomolov, V. Kovalenko, Y. Rebryk, A. Bacchelli, T. Bryksin // Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. – Athens, Greece, 2021. – P. 932–944.

7. Ullah F. Source code authorship attribution using hybrid approach of program dependence graph and deep learning model / F. Ullah, J. Wang, S. Jabbar, F. Al-Turjman, M. Alazab // IEEEAccess. – 2019. – Vol. 7. – P. 141987–141999.

8. Binary Authorship Verification with Flow-aware Mixture-of-Shared Language Model. – URL: https://arxiv.org/pdf/2203.04472, свободный (дата обращения: 18.11.2023).

9. Rosenblum N. Who Wrote This Code? Identifying the Authors of Program Binaries / N. Rosenblum, X. Zhu, B.P. Miller. – URL: https://pages.cs.wisc.edu/~jerryzhu/pub/Rosenblum11Authorship.pdf, свободный (дата обращения: 18.11.2023).

10. Alrabaee S. BinGold: Towards robust binary analysis by extracting the semantics of binary code as semantic flow graphs (SFGs) / S. Alrabaee, L. Wang, M. Debbabi // Dig. Investig. – 2016. – Vol. 18. – P. 11–22.

11. Caliskan-Islam A. When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries. – arXiv preprint. – arXiv:1512.08546. – 2017. – URL: https://arxiv.org/abs/1512.08546, свободный (дата обращения: 21.10.2023).

12. Alrabaee S. OBA2: An Onion Approach to Binary code Authorship Attribution / S. Alrabaee, N. Saleem, S. Preda, L. Wang, M. Debbabi // Dig. Investig. – 2014. – Vol. 11. – P. 94–103.

13. Caliskan-Islam A. Deanonymizing programmers via code stylometry // Proceedings of the 24th USENIX Security Symposium, Washington. – 2015. – P. 255–270. – URL: https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-caliskanislam.pdf, свободный (дата обращения: 18.11.2023).

14. Alrabaee S. On the Feasibility of Malware Authorship Attribution / S. Alrabaee, P. Shirani, M. Debbabi, L. Wang // Dig. Investig. – 2016. – Vol. 28. – P. 3–11.

15. Zia T. Source Code Author Attribution Using Author's Programming Style and Code Smells / T. Zia, M.I.J. Ilyas // Intell. Syst. Appl. – 2017. – Vol. 5. – P. 27–33.

16. Google Code Jam. – URL: https://codingcompetitions.withgoogle.com/codejam, свободный (дата обращения: 18.11.2023).

17. Codeforces. – URL: https://codeforces.com, свободный (дата обращения: 18.11.2023).

18. GCC, the GNU Compiler Collection. – URL: https://gcc.gnu.org, свободный (дата обращения: 21.10.2023).

19. IDA Pro. – URL: https://hex-rays.com/ida-pro, свободный (дата обращения: 18.11.2023).

20. Куртукова А.В. Идентификация автора исходного кода методами машинного обучения / А.В. Куртукова, А.С. Романов // Труды СПИИРАН. – 2019. – № 18. – С. 741–765.

21. Kurtukova A. Source Code Authorship Identification Using Deep Neural Networks / A. Kurtukova, A. Romanov, A. Shelupanov // Symmetry. – 2020. – Vol. 12. – 2044. DOI:10.3390/sym12122044

22. Kurtukova A. Complex Cases of Source Code Authorship Identification Using a Hybrid Deep Neural Network / A. Kurtukova, A. Romanov, A. Shelupanov, A. Fedotova // Future Internet. – 2022. – Vol. 14. – P. 287. DOI: /10.3390/fi14100287

23. AnalyzeC. – URL: https://github.com/ryarnyah/AnalyseC, свободный (дата обращения: 18.11.2023).

24. Linux Kernel Coding Style. – URL: https://www.kernel.org/doc/html/v4.10/process/coding-style.html (дата обращения: 18.11.2023).

Куртукова Анна Владимировна
Аспирант каф. комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)
Томского государственного университета
систем управления и радиоэлектроники (ТУСУР)
Ленина пр-т, 40, г. Томск, Россия, 634050
Тел.: +7-905-991-67-13
Эл. почта: av.kurtukova@gmail.com

Романов Александр Сергеевич
Канд. техн. наук, доцент каф. КИБЭВС ТУСУРа
Ленина пр-т, 40, г. Томск, Россия, 634050
Тел.: +7 (382-2) 41-34-26
Эл. почта: alexx.romanov@gmail.com

Шелупанов Александр Александрович
Д-р техн. наук, проф., президент ТУСУРа
Ленина пр-т, 40, г. Томск, Россия, 634050
Тел.: +7 (382-2) 90-71-55
Эл. почта: saa@tusur.ru

Kurtukova A.V., Romanov A.S., Shelupanov A.A.

Development of a methodology for identifying the authorship of binary and disassembled program codes based on an ensemble of modern natural language processing methods

This article is part of a series of studies aimed at solving problems of identifying the authorship of source code. The analysis of binary or disassembled code is a critical task in information security, software development, and computer forensics due to the need to protect intellectual property and copyright, as well as to identify the authors of malware. Any program is a machine code that can be disassembled (converted into text in assembly language) using specialized tools and analyzed for authorship by analogy with text in natural language. To solve this problem, the article proposes a technique based on the fastText ensemble, support vector machine (SVM) and the author-developed hybrid neural network. The proposed methodology was evaluated on source codes in C and C++ languages, collected from the GitHub and Google Code Jam platforms, compiled into executable files and disassembled using reverse engineering tools. The average accuracy of identifying the author of disassembled code using the proposed method was more than 0.9. The technique was also tested on source codes, resulting in an average accuracy of 0.96 in simple cases and more than 0.85 in complex cases (obfuscation, coding standards, etc.).
DOI: 10.21293/1818-0442-2023-26-4-53-60

References

1. Rahayu S., Shahrin S., Hafeizah N., Yusof R., Abdollah M.F. A Forensic Traceability Index in Digital Forensic Investigation. *Journal of Information Security*, 2013, vol. 4., no. 1, pp. 19–32.

2. Schleimer S., Wilkerson D.S., Aiken A. Winnowing: local algorithms for document fingerprinting. *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03),* Association for Computing Machinery, New York, NY, USA, 2003, pp. 76–85.

3. Abuhamad M., AbuHmed T., Mohaisen A., Nyang D. Large-Scale and Language-Oblivious Code Authorship Identification. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Toronto, ON, Canada, 2018, pp. 101–114.

4. Zhen L., Chen G., Chen C., Zou Y., Xu S. RoPGen: Towards Robust Code Authorship Attribution via Automatic Coding Style Transformation. *Proceedings of the 2022 IEEE 44th International Conference on Software Engineering* (*ICSE*), Pittsburgh, PA, USA, 2022, pp. 1906–1918.

5. Holland C., Khoshavi N., Jaimes G. Code authorship identification via deep graph CNNs. *Proceedings of the 2022 ACM Southeast Conference (ACM SE '22),* 2022, pp. 144–150.

6. Bogomolov E., Kovalenko V., Rebryk Y., BacchelliA., Bryksin T. Authorship attribution of source code: A language-agnostic approach and applicability in software engineering.

Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, 2021, pp. 932–944.

7. Ullah F., Wang J., Jabbar S., Al-Turjman F., Alazab M. Source code authorship attribution using hybrid approach of program dependence graph and deep learning model. *IEEEAccess*, 2019, vol. 7, pp. 141987–141999.

8. Song Q., Zhang Y., Ouyang L., Chen Y. BinMLM: Binary Authorship Verification with Flow-aware Mixture-of-Shared Language Model. Available at: https://arxiv.org/pdf/2203.04472, free (Accessed: November 18, 2023).

9. Rosenblum N., Zhu X., Miller B.P. Who Wrote This Code? Identifying the Authors of Program Binaries. Available at: https://pages.cs.wisc.edu/~jerryzhu/pub/Rosenblum11Authorship.pdf, free (Accessed: November 18, 2023).

10. Alrabaee S., Wang L., Debbabi M. BinGold: Towards robust binary analysis by extracting the semantics of binary code as semantic flow graphs (SFGs). *Digital Investigation*, 2016, vol.18, pp. 11–22.

11. Caliskan-Islam A. When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries. Available at: https://arxiv.org/abs/1512.08546, free (Accessed: November 18, 2023).

12. Alrabaee S., Saleem N., Preda S., Wang L., DebbabiM. OBA2: An Onion Approach to Binary code Authorship Attribution. *Digital Investigation*, 2014, vol. 11, pp. 94–103.

13. Caliskan-Islam A., Harang R., Liu A. Deanonymizing programmers via code stylometry. *Proceedings of the 24th USENIX Security Symposium*, 2015, pp. 255–270. Available at: https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-caliskan-islam.pdf, free (Accessed: November 18, 2023).

14. Alrabaee S., Shirani P., Debbabi M., Wang L. On the Feasibility of Malware Authorship Attribution. *Digital Investigation,* 2016, vol. 28, pp. 3–11.

15. Zia T., Ilyas M.I.J. Source Code Author Attribution Using Author's Programming Style and Code Smells. *Intelligent SystemswithApplications*, 2017, vol. 5, pp. 27–33.

16. Google Code Jam. Available at: https://codingcompetitions.withgoogle.com/codejam, free (Accessed: November 18, 2023).

17. Codeforces. Available at: https://codeforces.com/, free. (Accessed: October 25, 2023).

18. GCC, the GNU Compiler Collection. Available at: https://gcc.gnu.org, free (Accessed: November 18, 2023).

19. IDA Pro. Available at: https://hex-rays.com/ida-pro/, free (Accessed: October 25, 2023).

20. Kurtukova A.V., Romanov A.S. [Identification author of source code by machine learning methods]. *SPIIRAS Proceedings*, 2019, vol. 18, no. 3, pp. 741–765 (in Russ.).

21. Kurtukova A., Romanov A., Shelupanov A. Source Code Authorship Identification Using Deep Neural Networks. *Symmetry,* 2020, Vol. 12, 2044. DOI: 10.3390/sym12122044.

22. Kurtukova A., Romanov A., Shelupanov A., Fedotova A. Complex Cases of Source Code Authorship Identification Using a Hybrid Deep Neural Network. *Future Internet*, 2022, vol. 14, 287. DOI: /10.3390/fi14100287.

23. AnalyzeC. Available at: https://github.com/ryarnyah/AnalyseC, free (Accessed: November 18, 2023).

24. Linux Kernel Coding Style. Available at: https://www.kernel.org/doc/html/v4.10/process/coding-style.html, free (Accessed: November 18, 2023).

Anna V. Kurtukova
Postgraduate student, Department of Complex Information Security of Electronic Computer Systems (KIBEVS),
Tomsk State University of Control Systems and Radioelectronics (TUSUR)
40, Lenin pr., Tomsk, Russia, 634050
Phone: +7-905-991-67-13
Email: av.kurtukova@gmail.com

Aleksandr S. Romanov
Candidate of Sciences in Engineering, Associate professor, KIBEVS TUSUR
40, Lenin pr., Tomsk, Russia, 634050
Phone: +7 (382-2) 41-34-26
Email: alexx.romanov@gmail.com

Alexandr A. Shelupanov
Doctor of Science in Engineering, Professor, President TUSUR
40, Lenin pr., Tomsk, Russia, 634050
Phone: +7 (382-2) 90-71-55
Email: saa@tusur.ru