Санкт-Петербургский государственный университет

Кафедра системного программирования
Математическое обеспечение и администрирование
информационных систем

Архипов Иван Сергеевич

Нанесение водяных знаков на программное обеспечение

Отчёт по преддипломной практике

Научный руководитель:
д.ф-м.н., профессор А.Н.Терехов
Консультант:
старший преподаватель Уральского федерального университета,
А.Е.Сибиряков

Санкт-Петербург
2023

SAINT-PETERSBURG STATE UNIVERSITY

Software Engineering
Software and Administration of Information Systems

Arkhipov Ivan

Watermarking software

Report on pre-graduate practice

Scientific supervisor:
prof. A.N.Terekhov
Consultant:
Senior Lecturer at Ural Federal University, A.E.Sibiryakov

Saint-Petersburg
2023

Содержание

Введение

Для многих компаний, занимающихся разработкой программного обеспечения, пиратство программных продуктов является огромной проблемой. По оценкам, убытки от пиратства в индустрии в 2009 году составили более 50 миллиардов долларов [1]. Проведённое в 2007 году исследование [2] показало, что 35% установленного в 2006 году программного обеспечения является пиратским. С распространением Интернет-технологий данная проблема становится всё более острой [3, 4].

Для решения этой проблемы была разработана технология под названием «цифровой водяной знак». Речь идёт о некоторой информации, встраиваемой в программный продукт без ущерба для его эксплуатации [5]. Водяные знаки стали популярны для защиты мультимедийных объектов, таких как изображения [6, 7, 8], видео [9, 10, 11, 12] и аудио [13, 14, 15]. Активно ведутся разработки цифровых знаков применительно к программному обеспечению [16, 17, 18, 19, 20].

Как правило, водяные знаки программного обеспечения предназначены прежде всего для защиты авторских прав [21]. Для этого водяной знак может содержать информацию о правообладателе продукта [22], чтобы доказать авторство, или о покупателе программного обеспечения [23], чтобы имелась возможность выявить нарушителя. Также водяные знаки могут использоваться для проверки целостности данных [24].

Работа с водяным знаком включает внесение водяного знака в продукт, эксплуатацию продукта без нарушения функциональности и распознание водяного знака [25].

На кафедре системного программирования СПбГУ в исследовательской группе под руководством М.В.Баклановского была разработана концепция МАК[1], открывающая новые возможности для преобразования программного кода. В рамках этой концепции был предложен новый подход внесения водяных знаков.

[1] https://youtu.be/n-YBy9iQnw4. Accessed: 11-05-2023.

В рамках данной магистерской диссертации был разработан прототип фреймворка для работы с водяными знаками в программном обеспечении на основе концепции MAK и проведена его апробация.

1 Цели и задачи

Целью работы является разработка, на основе концепции MAK, прототипа фреймворка для работы с водяными знаками в программном обеспечении. Для достижения обозначенной цели были поставлены следующие задачи:

- обзор подходов к внесению водяных знаков в программное обеспечение, изучение концепции MAK (включая ограничения концепции), а также предшествующей работы по этому проекту;
- создание архитектуры фреймворка;
- прототипная реализация фреймворка;
- проведение экспериментального исследования с реализованным прототипом (выбор тестового приложения, анализ полученных результатов).

2 Обзор

2.1 Обзор проблемы

Водяные знаки могут быть классифицированы как статические и динамические. Статические водяные знаки внедрены в код и/или данные компьютерной программы. Динамические водяные знаки хранят информацию в состояниях выполнения программы [26].

В данном подразделе рассмотрены особенности каждой группы водяных знаков, а также приведены многочисленные примеры.

2.1.1 Статические водяные знаки

Замена кода или данных

Первые запатентованные алгоритмы внедрения водяных знаков в программное обеспечение [27, 28] основывались на замене определенной части кода или данных на значение водяного знака. Однако такие методы были уязвимы для атак, основанных на сравнении копий программного обеспечения, поскольку злоумышленник мог определить местоположение водяного знака путем сравнения результатов.

Позднее был разработан новый метод внедрения водяных знаков в программное обеспечение, который основывается на замене инструкций байткода в фиктивных методах Java [29]. Идея заключается в присвоении битовых значений определенным инструкциям байткода и замене существующих инструкций на биты кодирования, соответствующие значению водяного знака. Так как фиктивные методы не запускаются на исполнение, то водяной знак не имеет семантических ограничений. Главное требование к такому водяному знаку – его проходимость через верификатор байткода Java [30].

Данный метод внедрения водяных знаков был объединен с обфусцированием Фукусимой и др [31, 32]. Это позволило защитить обнаружение водяного знака путем создания множества различий между копиями программного обеспечения, а не только различий

в положении водяного знака. Идея заключается в том, чтобы каждую копию программного обеспечения обфусцировать по-разному.

Методы, основанные замене кода, были разработаны также для исходных кодов Java [33] и ассемблера x86 [34, 35].

Штерн и др. [36] представили надежный метод внедрения водяных знаков на объекты (ROW - robust object watermarking), который отличается от других подходов тем, что он рассматривает код как статистический объект, а не как последовательность инструкций. Этот метод создает более устойчивый водяной знак путем распределения его по всему коду, а не только в определенном месте. Он основан на изменении частоты групп инструкций в коде, чтобы внести водяной знак, и может использовать другие статистические характеристики программы.

В своей работе Штерн и др. [36] применили предложенный подход для архитектуры x86. В дальнейшем, этот подход был адаптирован для Java байткода [37, 38]. Каррэн и др. [39] использовали эту технику, опираясь на статистические данные, полученные из графа вызова программы. Позднее была предложена улучшенная версия метода водяного знака [40], которая более устойчива к атакам, основанным на сравнении копий программы.

Перестановка кода

Дэвидсон и Мирволд [41] предложили один из первых алгоритмов внесения водяного знака на основе перемещений линейных участков кода. В данном подходе линейный участок кода это множество последовательных инструкций с одной точкой входа и одной точкой выхода. Для внесения водяного знака такие участки перемешиваются без изменения функциональности программы. Для верификации водяного знака проверяется порядок линейных участков [42].

Хаттанда и Итикава [43] сделали анализ водяного знака на основе перемещений линейных участков кода, внеся его в несколько программ на языке C и проанализировав такие метрики как размер и скорость работы программы. Их исследование показало, что что размер программы с водяным знаком увеличивается на 9-24%,

а скорость работы составляет от 86% до 102% от скорости работы исходной программы.

В работе [44] был реализован и проанализирован алгоритм внесения водяного знака на основе перестановки групп цепочек линейных участков кода. Авторы пришли к выводу, что чем меньше в коде локальных изменений, тем выше скрытность водяного знака.

М.Ширали-Шахреза и С.Ширали-Шахреза [45] предложили водяной знак, основанный на перестановках операций в математических выражениях. Идея заключается в перестановке коммутативных математических операций, таких как сложение, чтобы не изменить семантику программы. Ша и др. [46] предложили очень похожий метод, основанный на перестановке коэффициентов операндов. Позже был опубликован водяной знак [47] для структурного программирования, основанный на перестановки в выражениях с последовательностями операций, ориентированными на функциональную зависимость.

Гонг и др. [48] предложили алгоритм внесения водяного знака для Java, основанный перестановке в пуле констант в файле Java класса. Пул констант в Java – это массив элементов переменной длины, содержащий все константы, используемые в классе Java [30].

Распределение регистров

В 1998 году Ку и Потконьяк [49] предложили подход, основанный на распределении регистров. В 2000 году авторы опубликовали метод для сохранения информации о покупателе программного продукта [50], основанный на ранее разработанной технологии [49]. В алгоритме QP (по именам авторов Qu and Potkonjak) в граф, моделирующий отношения между переменными в программном методе, добавляются рёбра и вершины в зависимости от значения водяного знака. Каждая вершина графа обозначает переменную. Вершины соединены ребром, если времена жизни переменных в программе имеют пересечение. Затем граф раскрашивается, чтобы минимизировать количество используемых регистров и гарантировать, что две используемые ("живые") переменные не будут зани-

мать один регистр. Чжу и Томборсон [51] представили улучшенную версию оригинального алгоритма.

Серьёзным недостатком метода QP оказалось то, что имееется возможность вставить два разных значения водяного знака в исходный граф и получить один и тот же граф с водяным знаком [51, 52, 53]. Также было показано, что граф в алгоритме QP можно модифицировать таким образом, что при его использовании во внесении водяного знака можно извлечь любое сообщение [54]. Ку и Потконьяк не видят в этом существенную проблему, утверждая, что будет трудно создать значимое сообщение, особенно если исходное сообщение зашифровано односторонней функцией [55].

Майлз и Коллберг [56] предложили новый алгоритм, QPS. В этом подходе тройки вершин для модификации с целью внесения водяного знака выбираются таким образом, чтобы они были изолированными единицами, которые не будут влиять на другие вершины в графе. Экспериментальные результаты [56] показали, что алгоритм имеет довольно низкую скорость и подвержен множеству простых атак, таких как обфускация. Однако было продемонстрировано, что он довольно скрытен и надежен. Другими словами, такой водяной знак трудно обнаружить злоумышленнику, но легко извлечь автору.

Чжу и Томборсон [52] представили ещё одно улучшение оригинального алгоритма, которое они назвали QPI. Для извлечения водяного знака QPI требует оригинальный граф и граф с водяным знаком. На основе сравнения этих двух графов извлекается содержание водяного знака.

Алгоритмы на базе изменения цветов (Colour Change – CC) и перестановки цветов (Colour Permutation – CC) [57, 58] в графе представляют из себя улучшение QPS, где модифицируется не граф, а его раскраска. Значение водяного знака конвертируется в натуральное число, выбирается номер перестановки цветов раскраски графа с таким числом и меняются цвета. Ли и Лю [59] опубликовали более эффективную версию алгоритма CC.

Цзян и др. [60] представили метод на основе алгоритма шифро-

вания RSA [61] и QPI [52]. Авторы предложили перед нанесением шифровать водяной знак. Даже если злоумышленник сможет извлечь информацию, он не сможет её расшифровать.

Изменение графа потока управления программы

Алгоритмы на базе изменения графов потока управления программы основаны на трудности анализа таких графов [62], а в общем случае и невозможности [63]. Коллберг и др. [64] описали несколько способов внедрения целых чисел в графовые структуры. Эти способы можно разделить на три разновидности: нумерование, изменение указателей вершин и перестановка.

Алгоритмы на основе нумерования представляют водяной знак в виде числа и выбирают граф с таким номером из специфического семейства графов. В качестве семейства могут выступать Parent-Pointer Trees (PPT), в которых имеется единственное ребро между вершиной и её родителем, Planted Planar Cubic Trees (PPCT), у которых все вершины кроме корня имеют двух потомков [65].

PPCT является довольно эффективной структурой, так как каждый узел имеет ссылку только на своего родителя. Но нанесённый на такую структуру водяной знак очень хрупкий и не вызывает доверия, так как добавление даже одного узла меняет значение водяного знака. Чтобы сделать PPCT сделать более устойчивой к атакам, было предложено а) маркировать каждый лист указателем на себя, и б) сделать внешний цикл через корень и все листья [66].

Алгоритмы на основе изменения указателя вершины добавляют дополнительное поле указателя в каждую вершину связанного списка длиной k для кодирования цифры по основанию k. Например, указатель на себя может означать 0, указатель на следующую вершину 1 и так далее. Несколько исследований [67, 68, 69, 70, 71, 72] предлагают такой метод в комбинации с PPCT, где граф с изменёнными указателями конвертируется в PPCT.

Алгоритмы на основе перестановок используют ту же структуру связанного списка, что и методы с изменением указателя вершины, но обладают свойствами исправления ошибок. При внесении перестановка определяется числом, полученным из водяного знака. Пе-

рестановка кодируется в графе добавлением дополнительных рёбер между вершинами.

Приводимые графы перестановок (Reducible permutation graph – RPG) [73, 74] очень похожи на графы с закодированными перестановками, но они ближе к графам потока управления программы (control-flow graph – CFG), так как являются приводимыми графами потока [75]. RPG, как и CFG, содержит единственный узел входа, единственный узел выхода, преамбулу, содержащую ноль или более узлов, из которых достижимы все остальные узлы, и тело, которое содержит в себе значение водяного знака, внесённого методом перестановки [76].

Венкатесан и др. [73] предложили первый статический метод нанесения водяного знака на основе графа потока управления, Graph Theoretic Watermarking (GTW), который прячет значение в топологию графа. Позже алгоритм был запатентован [74]. Основная идея состоит в нанесении водяного знака в RPG, а затем конвертации RPG в граф потока управления добавлением новых рёбер потока управления.

Вместе с добавлением водяного знака алгоритм добавляет фиктивные рёбра в граф потока управления между случайными вершинами, чтобы защитить водяной знак от атак, основанных на статическом анализе. Такой анализ может обнаружить возможные точки соединения рёбер оригинального CFG и добавленных после нанесения водяного знака [77].

Коллберг и др. [78] представили собственную модификацию алгоритма GTW. Они измерили дополнительные время и размер, необходимые для внесения водяного знака, а также оценили устойчивость метода к различным атакам. Исследователи предложили два подхода (расщепление числа на основе частичных сумм и на основе китайской теореме об остатках) для расщепления значения водяного знака, чтобы спрятать его части в нескольких небольших CFG. Авторы обнаружили, что достичь высокой степени скрытности является большой проблемой. Например, блоки с нанесённым водяным знаком на 20% состояли из арифметических инструкций,

а в среднем методы Java содержат 1% таких операций [79]. Водяные знаки рамером до 150 бит увеличивали размер программ от 40% до 75%, а время работы программ увеличивалось от 0% до 36% [78].

Непрозрачные предикаты

Непрозрачные предикаты это такие булевы выражения, значения которых для программы известны априори. Для автоматизированного анализа программного кода сложно найти значения таких предикатов, поэтому неизвестно, может ли определённая часть кода (которая может и содержать, и не содержать водяной знак) быть удалена [80].

Впервые данная техника была предложена Коллбергом и др [80]. Арбуа [81] опубликовал способ внесения водяного знака в Java байткод. Автор предложил прятать значение как закодированные константы внутри непрозрачных предикатов. Чтение водяного знака осуществлялось нахождением таких предикатов и декодирования спрятанного значения.

Майлз и Коллберг [82] сравнили два метода на основе непрозрачных предикатов [80, 81]. Исследователи пришли к выводу, что алгоритм Арбуа [81] лучше алгоритма Коллберга [80]. Сравнение проводилось на основе нескольких характеристик, например, таких как устойчивость и невидимость.

Алитаволи и др. [83] предложили более устойчивый к атакам на основе декомпиляции и обфускации метод.

Абстрактная интерпретация

Абстрактная интерпретация – это технология на основе статического анализа, предназначенная прежде всего для верификации программного обеспечения [84]. Данная технология используется для доказательства того, что абстрактная семантика программы удовлетворяет абстрактной спецификации, игнорируя самые различные маловажные детали семантики и спецификации. Абстрактная интерпретация может ответить на вопросы, которые не требуют знания о всех возможных вызовах программы или которые требуют неточный ответ о корректности программы.

Основная идея сокрытия водяного знака таким методом заклю-

чается в том, что значение может быть извлечено только с помощью абстрактной интерпретации определённой семантики кода [85]. Для извлечения водяного знака, внесённого таким образом, используется статический анализ свёртки констант. При обычном выполнении программы переменная принимает несколько значений, однако при абстрактной интерпретации программы переменная показывает значение водяного знака.

2.1.2 Динамические водяные знаки

"Пасхальные яйца"

Одной из самых известных техник является "пасхальное яйцо"[86]. Данный метод основан на внедрении кода, срабатывающего на определённых и очень редких входных данных. Недостатком такого подхода является то, что, во-первых, код относительно легко найти, так как он не влияет на работу остальной программы, во-вторых, по этой же причине его легко извлечь, и в-третьих, не все заказчики готовы в свой продукт добавлять незадекларированное поведение.

Харуаки Тамада и др. [87] предложили целый ряд водяных знаков на основе "пасхальных яйц" для программ на Java. Первый водяной знак предполагает изменение константных значений в полях класса, что относится к статическим водяным знакам. Второй водяной знак предполагал такое внедрение "пасхального яйца" , которое изменяет последовательность вызовов методов в программе. Третий подход основан на изменении иерархической структуры классов. Чётвёртый водяной знак основан на изменении классов, используемых для реализации определённых функций в программе. Позже исследователи провели дополнительный анализ предложенных методов и поставили новые эксперименты [88].

Водяные знаки в потоках

Технологию внесения водяного знака в многопоточных программах предложили Награ и Томборсон [89]. Их алгоритм заключается в добавлении новых семафоров так, чтобы не менялась семантика программы. После внесения необходимо убедиться, что срабатывает только небольшое подмножество добавленных семафоров.

Динамический путь исполнения программы

В подразделе обзора про статические водяные знаки были описаны основные подходы к нанесению водяных знаков на основе изменения графа потока управления программы, а также приведены примеры статических водяных знаков на базе данного подхода. В этом подразделе представлены динамические водяные знаки на основе этой технологии.

Коллберг и др. [90] предложили метод, наносящий водяной знак в структуру программы во время выполнения. Данный алгоритм основан на том, что такая структура является её неотъемлемой частью, её трудно проанализировать, так как придётся следить за поведением программы, и трудно извлечь, так как эта структура тесно связана с семантикой программы. Недостатком является изменение пути выполнения в результате различных преобразований, например, в результате оптимизаций или разработки новой версии продукта.

Майлз и Джин [23] предложили способ, ориентированный на создание новых путей исполнения программы. Исследователи предлагают конвертировать безусловные переходы функций в инструкции ветвления, которые содержат адрес перехода. Такие инструкции авторы предлагают использовать для подтверждения авторства.

Гупта и Пьепшик [91] разработали автоматизированную атаку, основанную на манипуляциях с указателем стека и не требующую много ресурсов. Такая атака способна убрать водяной знак Майлза и Джина [23]. Позже Гупта и Пьепшик [92] улучшили метод Майлза и Джина так, чтобы он был устойчив к атакам на основе манипуляцией с указателем стека, предложив более сложные преобразования с вызовами функций.

Чен и др. [93] представили представили метод внесения водяного знака с помощью обфускации. Значение водяного знака делится на несколько частей. Каждая часть прячется в своей ветви обфусцированного кода.

Тянь и др. [94] опубликовали алгоритм внесения водяного знака

посредством внедрения исполняемых команд, не меняющих семантику программы.

Динамические структуры данных

На основе работы Коллберга и Томборсона [86] Палсберг и др. [95] предложили способ внедрения водяного знака в динамическую структуру данных программы.

Хи [96] разработал метод, использующий кодирование констант в графе данных во время исполнения.

Камел и Алблуви [21] предложили технику нанесения водяных знаков структуры данных и виде R-дерева и его вариаций.

Бриеш и Удай [97] опубликовали способ внедрения водяного знака в базу данных. Значение делится на несколько частей и кодируется в различных атрибутах.

Return-Oriented Programming

На основе ориентированного на возвраты программирования [98] Ма и др. [99] предложили новый способ внесения водяного знака. Данный подход предполагает внесение кода, содержащего значение, в специально организованную структуру данных. Этот код в данных выглядит как обычные данные, но при определённых условиях код может быть вызван. Собственно вызов такого спрятанного кода позволяет восстановить значение водяного знака.

Гипотеза Коллатца

Ма и др. [20] предложили способ внесения водяного знака с использованием гипотезы Коллатца (3n+1 дилемма, сиракузская проблема) [100, 101]. Авторы предлагают изменять условные переходы с помощью обфускации. Эти процедуры запутывания построены таким образом, что они способны выражать значение водяного знака в виде итеративно выполняемых действий ветвления, произошедших во время вычисления значения по вышеупомянутой гипотезе. Используя взаимно однозначное соответствие между натуральными числами и их представление исходя из гипотезы, можно извлечь водяной знак.

2.1.3 Сравнение статических и динамических водяных знаков

Статические и динамические водяные знаки обладают своими характерными особенностями. В зависимости от требований к водяному знаку может быть предпочтителен тот или иной водяной знак.

Статические водяные знаки не требуют запуска программы для распознания, что значительно экономнее в плане ресурсов [102]. Например, нет необходимости прикладывать усилия для поддержки различных эффектов, вызванных запуском программного обеспечения на различных платформах.

Методы внесения статических водяных знаков не могут обеспечить высокую устойчивость к различным атакам [102, 103], но их легко использовать для защиты различных частей кода программного обеспечения путем дублирования [102, 105].

Динамические водяные знаки могут обеспечить высокую скрытность, особенно для объектно-ориентированного программного обеспечения, которое требует очень сложных структур кучи [102, 103, 105].

2.2 Характеристики водяных знаков

При работе с водяными знаками стоит чётко понимать, какими свойствами они обладают и для каких целей они подходят. Например, если водяной знак легко обнаружить, то он может служить предупреждением для злоумышленников, что ПО защищено от нелицензионного использования. Разные водяные знаки могут иметь разные цели, даже противоречащие друг другу. В связи с этим в современной литературе [102, 103, 105, 107] сложились следующие характеристики водяных знаков:

- Надёжность. В случае, когда необходимо доказать авторство продукта, желательно, чтобы водяной знак легко распознавался. Также он должен быть достаточно надёжным, чтобы его распознавание не имело ложных срабатываний.

- Требуемый объём ресурсов. Водяной знак всегда требует дополнительных ресурсов. Если эти ресурсы сопоставимы с ресурсами, необходимыми для работы программы, то, во-первых, это будет препятствовать работе продукта, и во-вторых, так водяной знак будет легко обнаружить.

- Невидимость. Внесённый водяной знак должен быть достаточно видимым, чтобы его можно было распознать для верификации продукта. С другой стороны, надо спрятать его так, чтобы он не был обнаружен злоумышленником.

- Защита частей кода, а не всего проекта целиком. Злоумышленник может захотеть украсть не целиком код, а только его часть. Для этого водяной знак должен быть распределён по всему программному коду.

- Устойчивость. Программное обеспечение может быть изменено по многим причинам: дальнейшая разработка, оптимизация, запутывание, намеренное изменение злоумышленником. Хочется, чтобы при изменениях кода сохранилась возможность распознать водяной знак. В некоторых случаях, наоборот, нужно, чтобы водяной знак ломался при малейшем изменении и сигнализировал о несанкционированном изменении кода.

2.3 Обзорная литература

Первые обзорные статьи о водяных знаках на программное обеспечение охватывали отдельные принципы, на которых строились алгоритмы внесения: методы на основе непрозрачных предикатов [82] и распределения регистров [56].

Кумар и Каур [103] представили обзор статических и динамических водяных знаков в Java байткоде.

Гамильтон и Данисик [104] опубликовали подробный обзор на статические водяные знаки.

Очень хороший обзор в целом по водяным знакам и конкретно по водяным знакам в программном обеспечении можно найти в

[105]. Относительно внесения водяных знаков в программное обеспечение в этом документе представлено не только многообразие методов, но и виды атак на программное обеспечение, небольшая теоретическая формализация внесения и распознавания водяных знаков, краткий обзор продуктов внесения водяных знаков и краткий анализ различных методов.

Из-за новизны цифровых водяных знаков как направления исследования (особенно для программного обеспечения) многие важные аспекты не исследованы и не освещены. Например, в зачаточном состоянии находится теоретическая часть водяных знаков. Из-за этого, а также специфики области отсутствуют какие-либо количественные характеристики водяных знаков и их анализ. Подробнее о направлениях развития в данной области можно прочесть в [106].

Анализ водяных знаков с точки зрения характеристик, описанных выше, был сделан в [102]. Однако в этой статье отражены не все характеристики и водяные знаки, а анализ лишён точности и основан на общих рассуждениях, поэтому исследование сложно назвать точным и всеобъемлющим. Однако из-за специфики области и отсутствия теоретического фундамента бОльшего и нельзя достичь.

Заслуживают упоминания статьи [107, 108]. Особенно подробный обзор приведён в статье [25].

2.4 Концепция MAK

На сегодняшний день широко распространены два принципа архитектуры ЭВМ. Одна из них, архитектура фон Неймана, основана на принципе хранения программного кода и данных в одном устройстве. Исходя из этого, в данной архитектуре возможна самомодификация кода. Во второй архитектуре, Гарвардской, данные и код хранятся на разных устройствах. В этом случае физически запрещена самомодификация кода.

На кафедре системного программирования СПбГУ в исследовательской группе под руководством М.В.Баклановского была разра-

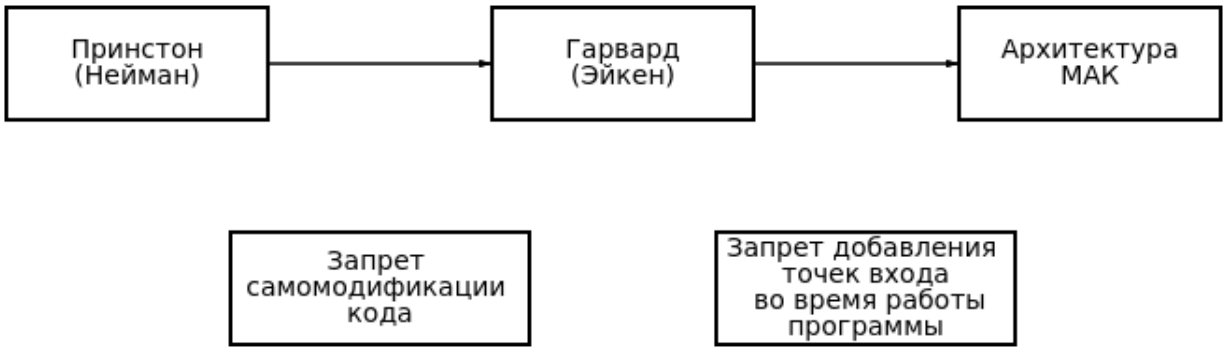


Рис. 1: Архитектурные концепции

ботана концепция МАК[2] (рис. 1). В ней на программы накладывается ещё одно важное ограничение: все возможные адреса переходов в программе должны быть известны до её исполнения. Авторы МАК характеризуют это ограничение как "запрет добавления точек входа во время работы программы".

В Гарвардской архитектуре нельзя определить точки входа программы. Из-за этого нельзя "раздвинуть" линейные участки кода, то есть оставить между линейными участками пустую память в секции кода, перемешать их, записать что-нибудь в промежутки между линейными участками, так как мы не можем заранее сказать, придёт ли управление в точку с конкретным адресом.

В концепции МАК этого недостатка нет, что позволяет записывать водяной знак в промежутки между линейными участками кода.

Большой проблемой в концепции МАК является отсутствие гарантии отсутствия динамического вычисления адресов переходов. Средствами статического анализа возможно найти такие места в программе и не рассматривать её в рамках МАК. Но дело в том, что такие конструкции, как, например, switch или виртуальные методы, используют динамические вычисления адресов. Для работы с такими конструкциям предлагается в рамках МАК не рассматривать области памяти, содержащие таблицы для switch и виртуальных методов.

[2] https://youtu.be/n-YBy9iQnw4. Accessed: 11-05-2023.

2.5 Водяные знаки в МАК

Как было написано в обзоре архитектуры МАК, благодаря её концепции возможно "раздвинуть" линейные участки кода, то есть оставить между линейными участками пустую память в секции кода. В эту пустую память становится возможным записать водяной знак.

Способ внесения водяного знака зависит от его цели. Например, если нужен видимый водяной знак, то можно просто его туда записать. Если нужен невидимый водяной знак, то нужен способ его спрятать. Например, можно замаскировать его под код. Если есть необходимость сделать водяной знак трудно извлекаемым, то можно внедрить в него исполняемые команды программы, тем самым применив обфускацию. Для устойчивости к атакам на основе статического анализа можно на крайне редких входных данных передавать управление на область памяти, содержащей водяной знак.

Хочется иметь способ наносить целый комплекс водяных знаков с разными характеристиками. Для этого необходим способ понимать, что в данной области записан водяной знак. М.В.Баклановским и его группой такой способ был назван "каркасным водяным знаком".

На данный момент для маркировки водяного знака в конкретной секции кода предлагается использовать определённые битовые последовательности. С одной стороны, в секции кода маркирующая последовательность не должна попадаться очень часто, так как это накладывает трудности на извлечение и верификацию водяного знака. С другой стороны, такие последовательности не должны выглядеть неестественно в секции кода, так как иначе злоумышленнику будет легко обнаружить местонахождение водяного знака.

Группа М.В.Баклановского обнаружила, что программный код имеет вполне определённые вероятностные характеристики. В ходе их экспериментов было выяснено, что нулей в программном коде примерно 60 %, а единиц 40 %.

В качестве маркирующих можно использовать "редкие" битовые последовательности. Под "редкими" понимаются последова-

тельности, которые встречаются не во всех бинарных файлах в секции кода. Такой выбор является компромиссом между простотой извлечения водяного знака с последующей верификацией и сложностью обнаружения злоумышленником местоположения водяного знака. С одной стороны, таких последовательностей будет немного в битовом представлении программы, а может и вообще не быть, что обеспечивает простоту в извлечении водяного знака. С другой стороны, такие последовательности не будут выглядеть неестественно в секции кода, так как имеются случаи, когда такая последовательность может возникнуть в обычном бинарном исполняемом файла, что усложняет работу злоумышленнику.

Для сокрытия водяного знака можно использовать статистические характеристики бинарного кода. В области расположения значения водяного знака можно добавить ненесущие информацию биты для обеспечения в данной области кода таких же статистических характеристик, как у обычного бинарного кода. Такая техника не даст злоумышленнику обнаружить водяной знак по отличающимся статистикам.

2.6 Предшествующая работа по проекту

Тема данной работы рассматривалась в курсовой работе Смирнова Дениса Павловича в 2018 году [109].

Его работа содержит очень хороший обзор водяных знаков. Некоторые важные характеристики и замечания к конкретным методам не были найдены даже в современных обзорных статьях.

Также Денисом Павловичем был сделан обзор программных продуктов, предоставляющих функциональность внесения и распознавания водяных знаков. В ходе работы он перенёс часть функциональности MAK на Windows, которая позволяет реализовать водяные знаки.

В работе Дениса Павловича исследовательская часть требует значительной доработки. Были собраны некоторые необходимые данные, однако не все описаны. У описанных данных отсутствует анализ. Отсутствуют прототип внесения и распознавания водяного

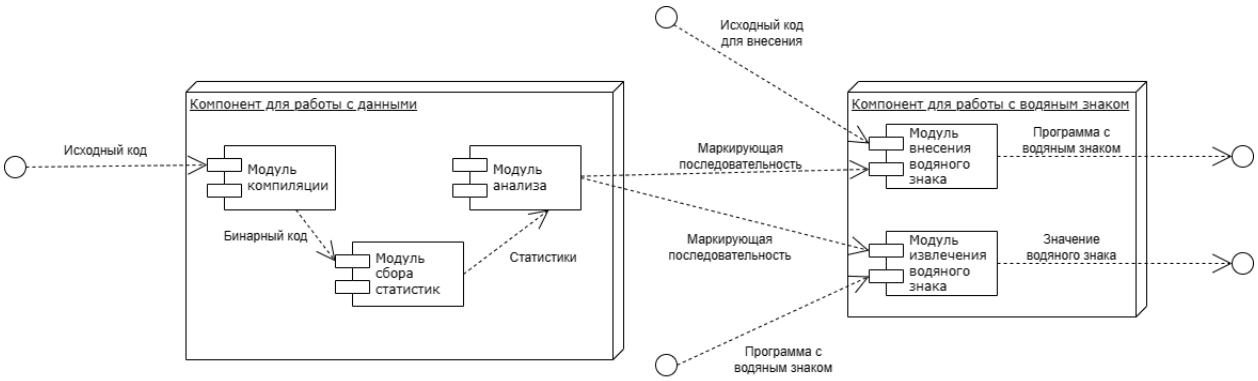


Рис. 2: Архитектура фреймворка

знака в рамках МАК.

3 Архитектура фреймворка

Фреймворк состоит из двух компонентов: компонент для работы с данными[3] и компонент для работы с водяным знаком[4]. Первый компонент содержит модуль компиляции, модуль сбора статистик и модуль анализа. Второй компонент содержит модуль внесения водяного знака и модуль извлечения водяного знака. Схематично архитектура фреймворка представлена на рис. 2.

Модуль компиляции преобразует набор файлов с исходным кодом в набор бинарных файлов. Модуль сбора статистик по набору бинарных файлов собирает доли последовательностей длиной до 10 включительно для каждого файла датасета. Модуль анализа по собранным долям находит статистики и подходящие для маркировки последовательности. Модуль внесения водяного знака внедряет в программу значения водяного знака, а модуль извлечения водяного знака позволяет достать из файла внедрённое значение.

Для поиска статистик и маркирующих последовательностей пользователь использует функциональность компонента для работы с данными. Для этого ему необходимо предоставить файлы с исходными кодами, с которых необходимо собрать данные, указать ком-

[3] https://github.com/IvanArkhipov1999/Binary-code-statistics-research. Accessed: 11-05-2023.

[4] https://github.com/IvanArkhipov1999/watermark-prototype. Accessed: 11-05-2023.

пилятор и флаги компиляции. На выходе пользователь получает “редкие” последовательности, подходящие для маркировки, среднее, стандартное отклонение, дисперсию, минимальное и максимальное значения для долей вхождения всех последовательностей длины до 10 включительно и последовательности, доли которых подчиняются нормальному распределению.

Для внесения и извлечения водяного знака пользователь использует компонент для работы с водяным знаком. Для внесения водяного знака пользователь предоставляет файл с исходным кодом программы, маркирующую последовательность, полученную от компонента для работы с данными, и значение водяного знака. На выходе пользователь получает программу, содержащую водяной знак. Для извлечения водяного знака пользователь предоставляет маркирующую последовательность и программу с водяным знаком. На выходе пользователь получает значение водяного знака.

Для реализации модулей компиляции, сбора статистик, внесения и извлечения водяного знака был использован язык Python ввиду своей простоты и удобства. Для реализации модуля анализа был выбран язык R ввиду своей направленности на статистические расчёты.

4 Особенности реализации

4.1 Модуль компиляции

Модуль компиляции реализован на языке Python[5] и состоит из одного файла *pass_and_compile.py*[6]. Язык Python был выбран ввиду своей простоты.

Файл *pass_and_compile.py* содержит единственную функцию *numbered_pass_and_compile*. Данная функция принимает путь до директории с исходным кодом датасета, путь до директории для

[5] https://www.python.org/. Accessed: 11-05-2023.

[6] https://github.com/IvanArkhipov1999/Binary-code-statistics-research/blob/main/pass_and_compile.py. Accessed: 11-05-2023.

скомпилированных файлов, регулярное выражение имени исходных файлов в виде строки (например, если нужно скомпилировать все файлы с расширение cpp, то необходимо передать *.cpp), компилятор в формате строки (это может быть путь до компилятора в файловой системе или название компилятора, если путь до него имеется в системных путях) и список флагов компиляции в виде списка строк, где каждая строка содержит отдельный флаг.

Функция *numbered_pass_and_compile* проходит директорию с исходными программами и его внутренние директории на всю глубину, компилирует заданным компилятором с заданными флагами все исходные файлы, соотвествующие регулярному выражению, и сохраняет бинарный файл по заданному пути. Если компиляция для какого-либо файла провалилась, то функция переходит к следующему файлу. Так сделано, чтобы не останавливать долговременную подготовку датасета бинарных файлов из-за ошибки компиляции на одном файле. Все бинарные файлы имеют нумерованные имена.

Пример использования модуля компиляции приведён в функции main[7].

4.2 Модуль сбора статистик

Модуль сбора статистик реализован на языке Python и состоит из одного основного файла *statistics.py*[8] и двух вспомогательных файлов *text_segment.py*[9] и *csv_local.py*[10]. Как и в случае с модулем компиляции, Python был выбран ввиду своей простоты.

В качестве статистик кода были выбраны доли битовых последовательностей длины от 1 до 10 включительно. Это позволяет

[7] https://github.com/IvanArkhipov1999/Binary-code-statistics-research/blob/main/main.py. Accessed: 11-05-2023.

[8] https://github.com/IvanArkhipov1999/Binary-code-statistics-research/blob/main/statistics.py. Accessed: 11-05-2023.

[9] https://github.com/IvanArkhipov1999/Binary-code-statistics-research/blob/main/text_segment.py. Accessed: 11-05-2023.

[10] https://github.com/IvanArkhipov1999/Binary-code-statistics-research/blob/main/csv_local.py. Accessed: 11-05-2023.

найти и статистические характеристики кода, и "редкие" битовые последовательности.

Файл *statistics.py* содержит единственную функцию *proportions_length*. Она принимает путь до директории с бинарными файлами, длину последовательностей, доли которых должны быть вычислены, и постфикс для именования столбцов с долями каждой последовательности.

Функция *proportions_length* обходит директорию с бинарными файлами, извлекает секцию кода с помощью вспомогательной функции *text_segment*, вычисляет доли всех последовательностей заданной длины, добавляет их в словарь (ключом в словаре является сама последовательность) и в конце формирует массив посчитанных долей. Каждый столбец в возвращаемом массиве имеет имя вида "последовательность"+"заданный постфикс" и содержит доли определённой последовательности для каждого файла в заданном датасете. Все столбцы упорядочены в лексикографическом порядке.

Вспомогательный файл *text_segment.py* содержит единственную функцию *text_segment*. Она принимает на вход путь до файла в формате elf и возвращает список байтов секции кода.

В elf файле имеется множество секций с самой различной информацией: данные для работы программы, данные об исполняемом файле, программный код.

Для нахождения статистик необходимо уметь извлекать секцию кода .text из elf файла.

Исходя из спецификации [116] был разработан и реализован в функции *text_segment* следующий алгоритм извлечения секции кода из исполняемого файла для 64-битных архитектур:

1. Получить ссылку на section header table (смещение 0x28 от начала файла). В section header table хранится различная информация о секциях. Запись с информацией о каждой секции занимает 10 байт.

2. Получить индекс секции .shstrtab в section header table (смеще-

ние 0x3E от начала файла). Секция .shstrtab хранит название секций, ссылки на которые хранятся в section header table.

3. Идти по section header table по секциям. По смещению (смещение 0x00 от начала записи) на .shstrtab определить, информация о какой секции хранится в данной записи таблицы.

4. Если дошли до записи о секции .text, то получить смещение секции от начала файла (смещение 0x18 от начала записи) и её размер (смещение 0x20 от начала записи). Можно извлекать данные.

Вспомогательный файл *csv_local.py* содержит две функции *write_data_to_csv* и *add_data_to_csv*. Каждая из них принимает путь до файла в формате csv[11] и массив долей последовательностей. Функция *write_data_to_csv* записывает в заданный файл массив, а *add_data_to_csv* добавляет в заданный файл новые столбцы с долями подпоследовательностей. Формат csv выбран ввиду удобства представления табличных данных.

Пример использования модуля сбора статистик приведён в функции main[12].

4.3 Модуль анализа

Модуль анализа реализован на языке R[13] и состоит из одного файла *analysis.R*[14]. Язык R был выбран ввиду своей направленности на статистический анализ данных. В начале файла идёт чтение данных из csv-файла, сформированного модулем сбора статистик.

В качестве статистик для каждой подпоследовательности были выбраны следующие метрики: среднее, стандартное отклонение, медиана, минимальное и максимальное значения.

[11] https://www.rfc-editor.org/rfc/rfc4180. Accessed: 11-05-2023.

[12] https://github.com/IvanArkhipov1999/Binary-code-statistics-research/blob/main/main.py. Accessed: 11-05-2023.

[13] https://www.r-project.org/. Accessed: 11-05-2023.

[14] https://github.com/IvanArkhipov1999/Binary-code-statistics-research/blob/main/analysis.R. Accessed: 11-05-2023.

Функция *to_hist_0_1* по данным о доле подпоследовательности для каждого файла строит гистограмму в пределах от 0 до 1 и отображает значения описанных выше метрик.

Функция *to_hist_min_max* по данным о доле подпоследовательности для каждого файла строит гистограмму в пределах от минимального до максимального значения и линиями отображает среднее значение и медиану. Такая гистограмма удобная для визуализации последовательностей, имеющих маленькие доли в файле. Похожим образом работает функция *to_hist_density*, только дополнительно на гистограмме она отображает эмпирическую плотность.

Также модуль анализа проверяет принадлежность выборки долей каждой последовательности нормальному распределению. Знание о принадлежности распределения долей последовательности позволяет более качественно моделировать бинарные последовательности с теми же статистическими характеристиками, что даёт бОльшую скрытность водяному знаку. Коэффициенты нормального распределения вычисляются по методу максимального правдоподобия[15]. Принадлежность выборки нормальному распределению с такими коэффициентами определяется по результатам теста Колмогорова-Смирнова[16]. Значение p-value для потверждения гипотезы опринадлежности выборки распределению взято равным 0.05.

Функция *ks_test_mle_norm* проводит тест Колмогорова-Смирнова для долей заданной последовательности и нормального распределения с коэффициентами, вычисленными по методу максимального правдоподобия. В ней используется функция ks.test[17], реализованная в пакете dgof языка R.

Функция *ks_test_mle_norm_analysis* проходит по всем последовательностям и выводит в печать те, выборки долей которых

[15] https://en.wikipedia.org/wiki/Maximum_likelihood_estimation. Accessed: 11-05-2023.

[16] https://en.wikipedia.org/wiki/Kolmogorov-Smirnov_test. Accessed: 11-05-2023.

[17] https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/ks.test. Accessed: 11-05-2023.

принадлежат нормальному распределению. Помимо самой подпоследовательности функция выводит p-value, а также параметры μ и σ.

Для поиска "редких" последовательностей реализована функция *zero_min_analysis*. В ней происходит обход по всем столбцам с долями последовательностей. Последовательности, которые не содержатся хотя бы в одном файле, выводятся в печать. Дополнительно выводится среднее значение долей таких последовательностей и процент файлов, не содержащих эту последовательность.

Вызов описанных выше функций на данных реализован в конце файла *analysis.R*.

4.4 Модуль внесения водяного знака

Модуль внесения водяного знака реализован на языке Python и состоит из одного основного файла *apply_watermark.py*[18] и двух вспомогательных файлов *compile.py*[19] и *insert.py*[20]. Python был выбран ввиду своей простоты.

Файл *apply_watermark.py* содержит единственную функцию *apply_watermark*. Она принимает путь до исходного файла, значение водяного знака, путь до компилятора и маркирующую последовательность.

В начале работы функция *apply_watermark* считает количество байт, которое нужно выделить для водяного знака. Оно складывается из байт для маркирующей последовательности, байта длины водяного знака и значения водяного знака. Таким образом, сообщение для внедрения не должно превышать 256 байт.

Далее с помощью функций из вспомогательных файлов программа транслируется в коды ассемблера, в текст на ассемблере вставляются мусорные команды для выделения памяти под водя-

[18] https://github.com/IvanArkhipov1999/watermark-prototype/blob/main/apply_watermark.py. Accessed: 11-05-2023.

[19] https://github.com/IvanArkhipov1999/watermark-prototype/blob/main/compile.py. Accessed: 11-05-2023.

[20] https://github.com/IvanArkhipov1999/watermark-prototype/blob/main/insert.py. Accessed: 11-05-2023.

ной знак, программа с мусорными командами компилируется в бинарный файл и в этот файл в область памяти с мусорными командами вносится водяной знак.

Файл *compile.py* содержит единственную функцию *compile*. Она принимает путь до исходного файла, путь до скомпилированного файла, путь до компилятора и флаги компиляции.

Функция *compile* компилирует переданный текст программы переданным компилятором с указанными флагами.

Файл *insert.py* содержит две функции: *insert_nop_commands* и *insert_watermark*.

Функция *insert_nop_commands* принимает путь до ассемблерного файла для внедрения мусорных команд, количество мусорных команд и путь до файла с мусорными командами. Она внедряет мусорные команды в текст ассемблера исходной программы. В качестве мусорных команд функция использует команды nop. Вставляет эти команды функция перед возвратом из функции main исходной программы. Чтобы не нарушить семантику исходной программы, дополнительно перед мусорными командами генерируется безусловный переход на возврат из функции main. На данный момент функция работает только для архитектуры x86_64, но возможно расширение.

Функция *insert_watermark* принимает путь до бинарного файла с мусорными командами, значение водяного знака, количество байт для водяного знака, маркирующую последовательность и путь до файла с внесёнными водяным знаком. Функция ищет в бинарном файле область памяти, содержащей определённой количество nop команд. После этого в данную облать записываются маркирующая последовательность, длина водяного знака и сообщение.

На данный момент внесение водяного знака работает только для C++ и архитектуры x86_64, но возможно расширение на другие языки и платформы.

4.5 Модуль извлечения водяного знака

Модуль извлечения водяного знака реализован на языке Python

и состоит из одного файла *extract_watermark.py*[21]. Python был выбран ввиду своей простоты.

Файл *extract_watermark.py* содержит единственную функцию *extract_watermark*. Она принимает путь до бинарного файла с внедрённым водяным знаком и маркирующую последовательность. Функция сканирует бинарный файл, находит маркирующую последовательность, считывает длину водяного знака и возвращает строку с сообщением. Если маркирующей последовательности не найдено, то возвращается пустая строка.

5 Эксперименты

5.1 Выбор данных для экспериментов

Для сбора информации о программном коде необходимо выбрать данные для анализа. В качестве данных для нахождения статистических характеристик и "редких" битовых последовательностей была выбрана тестовая база GCC. В ней имеется большое количество разнообразных файлов с программным кодом, пригодных для сборки и получения бинарного файла.

Далее необходимо определиться, какими средствами получить бинарный файл с кодом для анализа. Для компиляции был выбран один из наиболее популярных на данный момент компиляторов gcc[22]. Всего было подготовлено 3884 бинарных файла.

После выбора исходных файлов и средства компиляции нужно выбрать формат бинарных файлов. Это может быть, например, один из форматов исполняемого файла[23] или динамической библиотеки[24]. Для компиляции был выбран формат elf[25] (Executable

[21] https://github.com/IvanArkhipov1999/watermark-prototype/blob/main/extract_watermark.py. Accessed: 11-05-2023.

[22] https://gcc.gnu.org/. Accessed: 11-05-2023.

[23] https://en.wikipedia.org/wiki/Comparison_of_executable_file_formats. Accessed: 11-05-2023.

[24] https://en.wikipedia.org/wiki/Library_(computing). Accessed: 11-05-2023.

[25] https://en.wikipedia.org/wiki/Executable_and_Linkable_Format. Accessed: 11-05-2023.

and Linkable Format).

В конце необходимо определиться, под какие архитектуры производить компиляцию исходных кодов. Были выбраны следующие архитектуры: x86_64[26], arm64[27] и mips64el[28]. Выбор архитектур обусловлен целью продемонстрировать работоспособность водяного знака на основе концепции MAK в разнообразных архитектурах.

Весь анализируемый датасет лежит в отдельном репозитории[29]. Компиляция была произведена с флагом -std=gnu++11.

При таком выборе и подготовке датасета возникает естественный вопрос о репрезентативности выборки. Будут ли статистики другими, если выбрать другие исходные файлы, другой язык исходных файлов, другой компилятор, другой формат бинарного файла, другие флаги компиляции, другие архитектуры?

На данный момент область, занимающаяся получением подобных характеристик программы из бинарного представления, появилась относительно недавно. Первая публикация была сделана в 2010 году [110]. В ней авторы предложили метод определения компилятора по бинарному коду. Подобные исследования были сделаны для извлечения информации об авторстве бинарного кода [111, 112], флагах компиляции [113, 114] и используемых тулчейнах [115].

Вполне возможно, что упомянутые выше факторы так или иначе влияют на распределение статистик бинарного кода. Как будет показано ниже, статистические характеристики бинарных программ для разных архитектур отличаются друг от друга.

Одной из задач работы является продемонстрировать работоспособность предложенного группой М.В.Баклановского метода, взяв в качестве данных исходники на одном из распространённых языков программирования, скомпилированных одним из популярных компиляторов в непохожие друг на друга архитектуры. Результаты работы не гарантируют, что полученные в ходе анализа статисти-

[26] https://en.wikipedia.org/wiki/X86-64. Accessed: 11-05-2023.

[27] https://en.wikipedia.org/wiki/AArch64. Accessed: 11-05-2023.

[28] https://en.wikipedia.org/wiki/MIPS_architecture. Accessed: 11-05-2023.

[29] https://github.com/IvanArkhipov1999/Binaries-dataset. Accessed: 11-05-2023.

ки можно будет использовать для других случаев. Однако работа показывает, что скорее всего при выборе другого языка, компилятора, платформы и других характеристик метод будет работать. А одним из результатов работы являются необходимые инструменты для подготовки, сбора и анализа статистических данных бинарного кода с другими характеристиками.

В качестве характеристик кода были выбраны доли битовых последовательностей различной длины. Это позволяет найти и статистические характеристики кода, и редкие битовые последовательности.

В качестве статистик для каждой последовательности были выбраны следующие метрики: среднее, стандартное отклонение, медиана, минимальное и максимальное значения. Также каждая выборка долей каждой последовательности проверяется на принадлежность нормальному распределению, параметры которого вычислены по методу максимального правдоподобия. Подробнее об анализе можно прочитать в разделе "Модуль анализа".

В предыдущей работе по данной теме [109] в качестве редких битовых последовательностей рассматривались исключительно последовательности из единиц, так как доля единиц в бинарном файле, как было выяснено группой Баклановского М.В., равна 40%. Как будет показано далее, это не самый лучший подход.

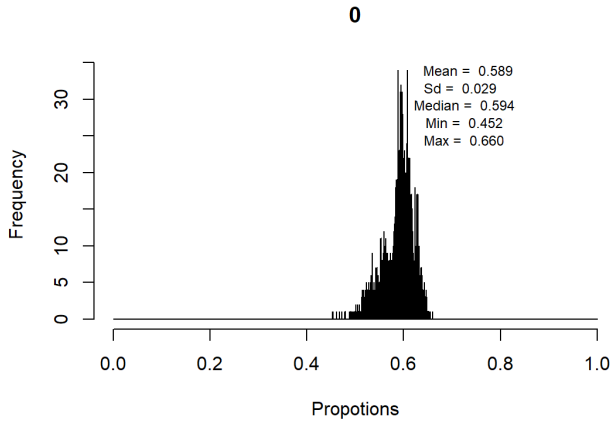
В следующих подразделах для каждой архитектуры будут приведены гистограммы долей последовательностей длиной до 2-х включительно, гистограммы нескольких последовательностей, выборки долей которых соответствуют нормальному распределению, и гистограммы некоторых "редких" последовательностей. Полный перечень статистик и "редких" последовательностей не приведён ввиду их большого объёма, но выложен в отдельном репозитории[30].

5.2 Архитектура x86_64

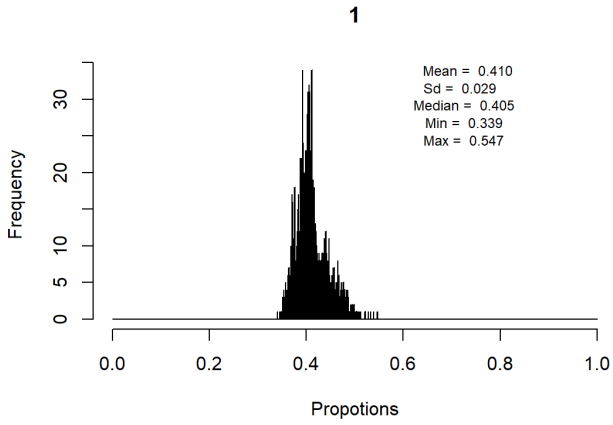
В данном подразделе представлены результаты для архитек-

[30] https://github.com/IvanArkhipov1999/Binaries-dataset. Accessed: 11-05-2023.

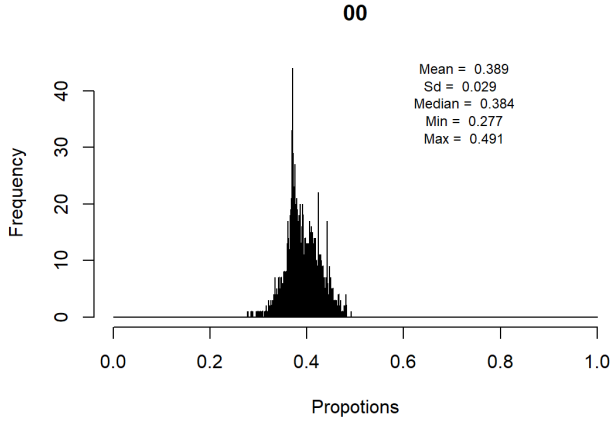
туры x86_64. Вычисленные статистики для последовательностей длины меньше 2-х представлены на рисунке 3.



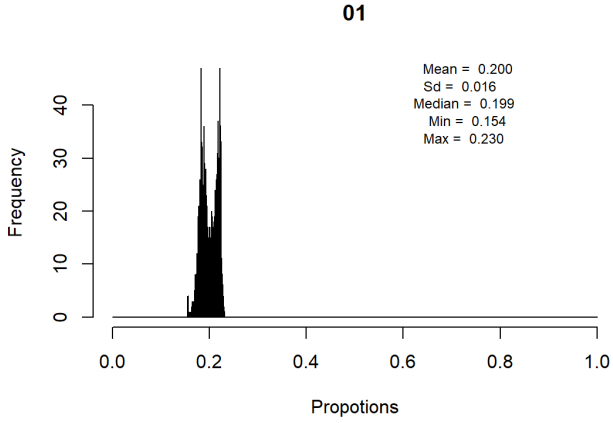
(a) Последовательность "0"



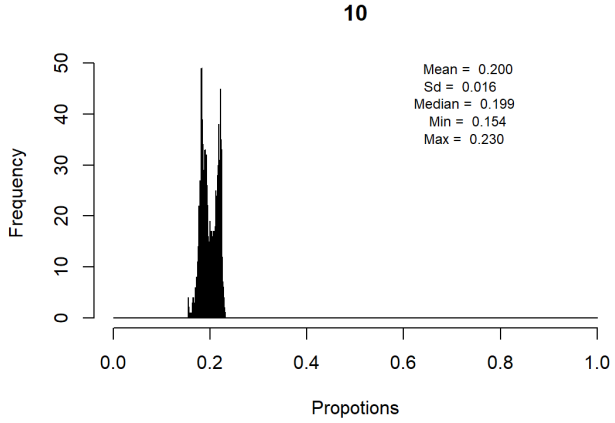
(b) Последовательность "1"



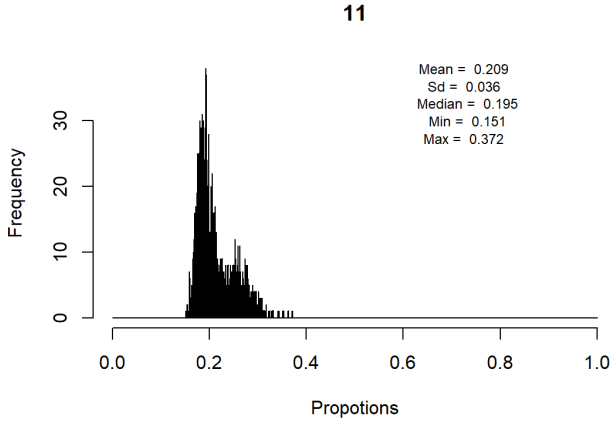
(c) Последовательность "00"



(d) Последовательность "01"



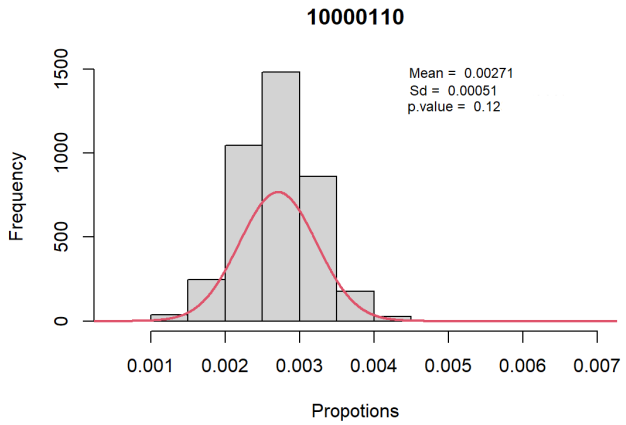
(e) Последовательность "10"



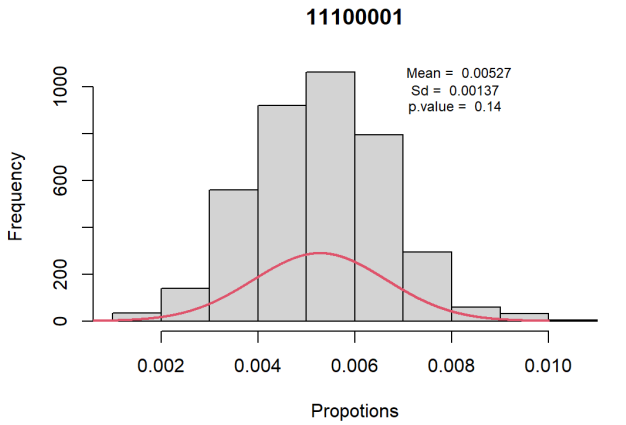
(f) Последовательность "11"

Рис. 3: Распределение последовательностей для x86_64

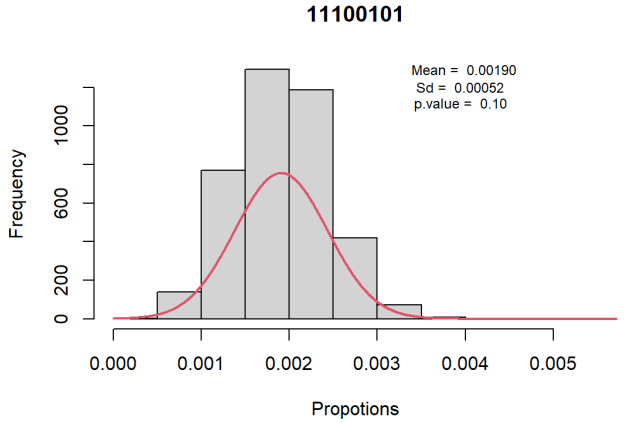
Подтвердилось утверждение о том, что в среднем в бинарном коде нулей 60%, а единиц 40%.



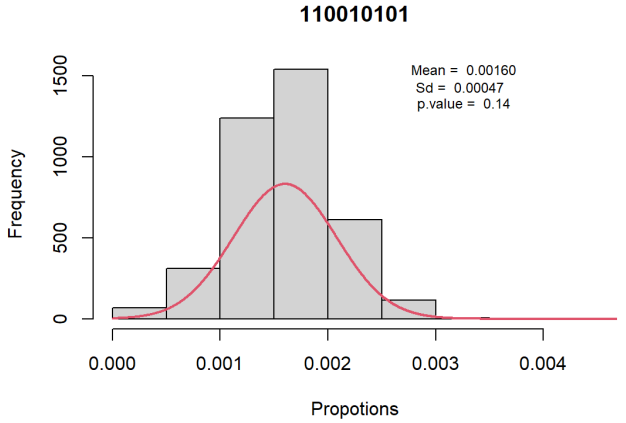
(a) Последовательность "10000110"



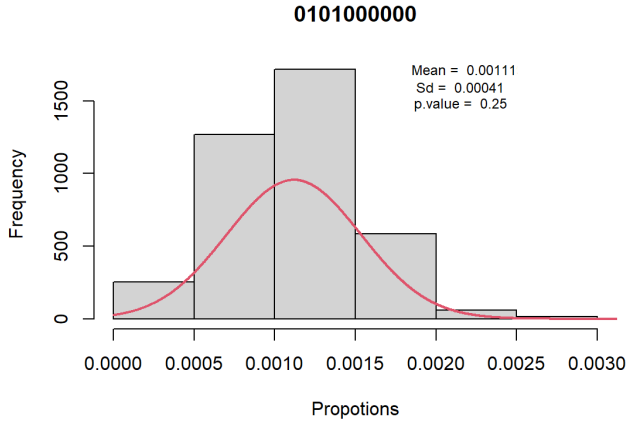
(b) Последовательность "11100001"



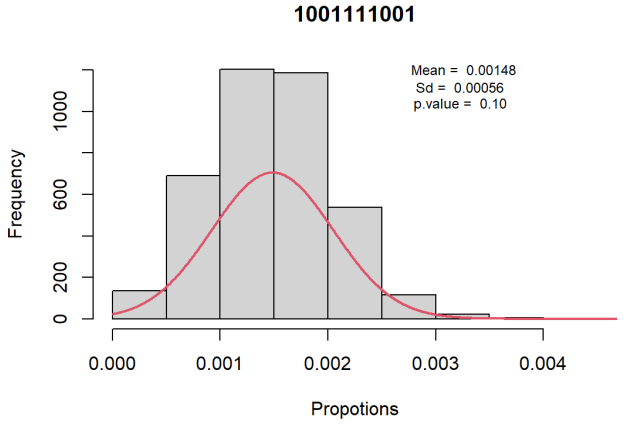
(c) Последовательность "11100101"



(d) Последовательность "110010101"



(e) Последовательность "0101000000"

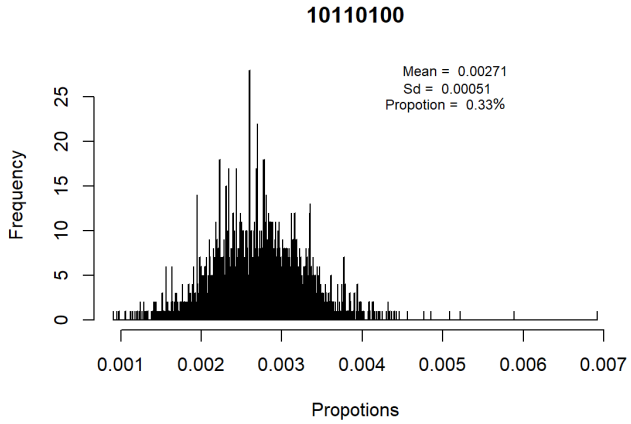


(f) Последовательность "1001111001"

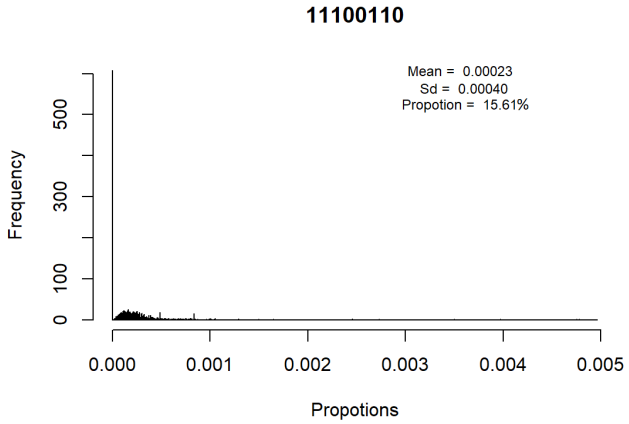
Рис. 4: Распределение последовательностей для x86_64

Уже при рассмотрении последовательностей небольшой длины становится видно, что брать в качестве редких последовательности, состоящие только из единиц, это не лучший подход. При рассмотрении последовательностей длины 2 оказалось, что доля последо-

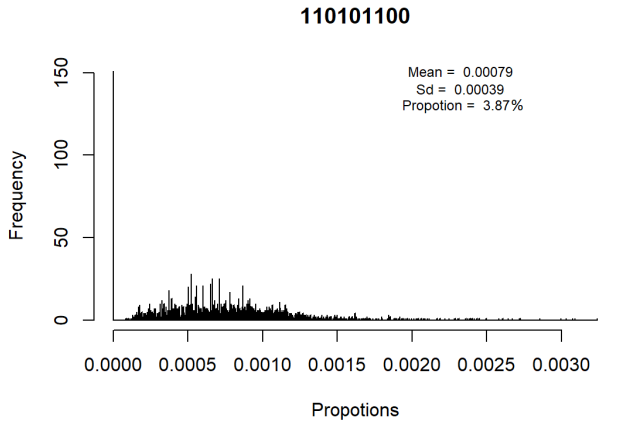
вательностей 01, 10 и 11 примерно одинакова.



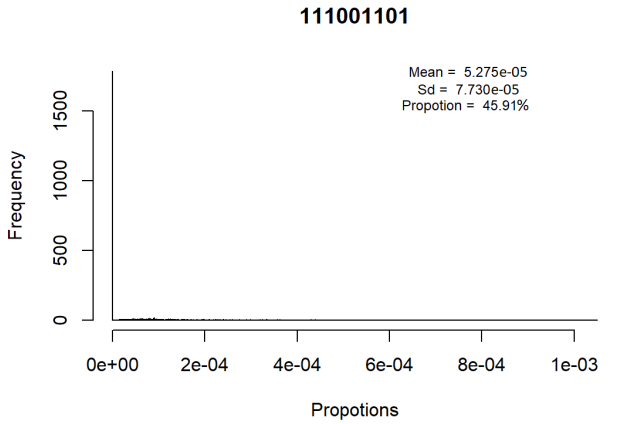
(a) Последовательность "10110100"



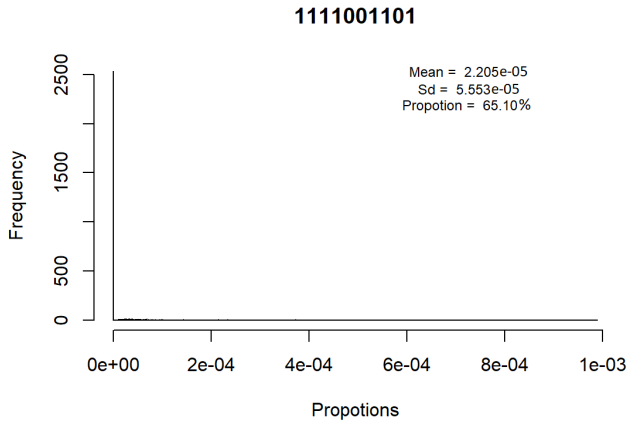
(b) Последовательность "11100110"



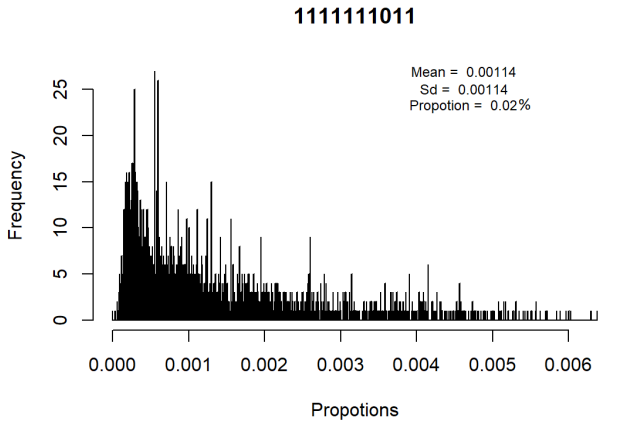
(c) Последовательность "110101100"



(d) Последовательность "111001101"



(e) Последовательность "1111001101"



(f) Последовательность "1111111011"

Рис. 5: Распределение последовательностей для x86_64

На рисунке 4 приведены некоторые последовательности, распределение долей которых соотвествует нормальному. Графики содержат информацию о нормальном распределении и значение p-value,

с которым прошёл тест Колмогорова-Смирнова для данной последовательности.

Всего было найдено 13 последовательностей длиной 8, 9 и 10 бит, распределение долей которых соответствует нормальному.

На рисунке 5 представлены некоторые "редкие" последовательности. Графики содержат информацию о среднем, стандартном отклонении и доле файлов, не содержащих данную последовательность.

Всего было найдено 508 редких подпоследовательностей длиной 8, 9 и 10 бит. Процент частоты нахождения в коде таких подпоследовательностей самый разный, что позволяет регулировать невидимость водяного знака.

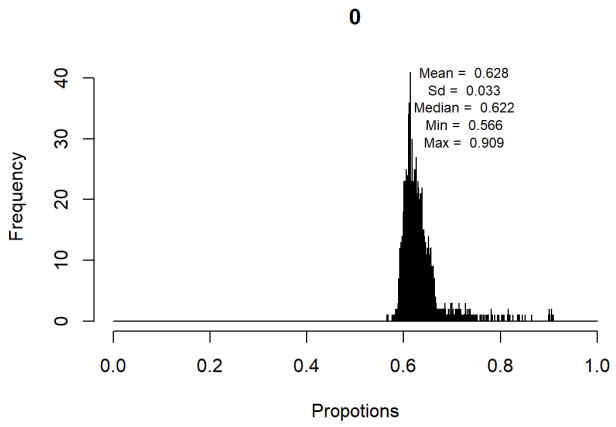
5.3 Архитектура arm64

В этом подразделе представлены результаты для архитектуры arm64. Вычисленные статистики для последовательностей длины меньше 2-х представлены на рисунке 6.

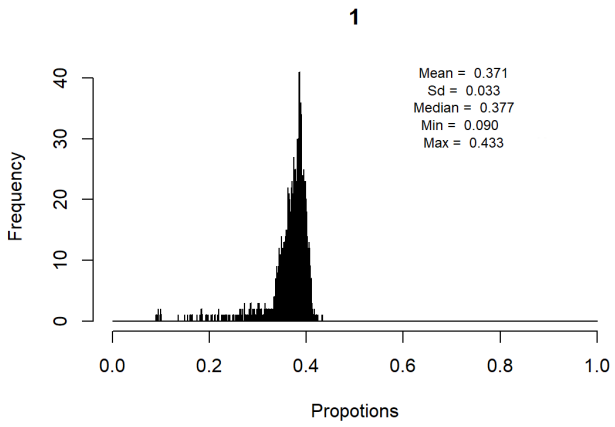
По сравнению с архитектурой x86_64 в коде arm64 содержится немного больше нулей. Подтверждается соотношение количества нулей к единицам примерно как к 3:2.

На рисунке 7 приведены некоторые последовательности, распределение долей которых соотвествует нормальному. Графики содержат информацию о нормальном распределении и значение p-value, с которым прошёл тест Колмогорова-Смирнова для данной последовательности.

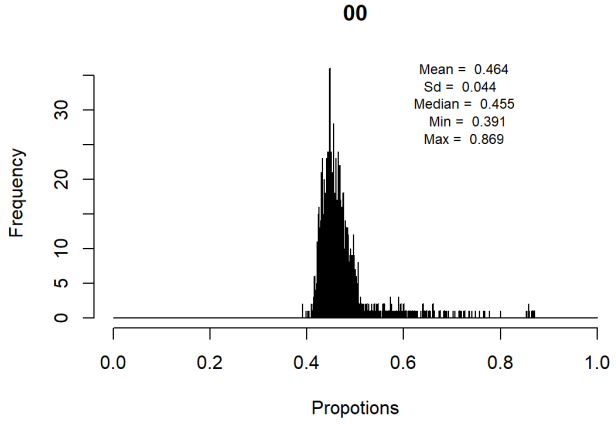
Всего было найдено 45 последовательностей длиной 5, 6, 7, 8, 9 и 10 бит, распределение долей которых соответствует нормальному, что существенно больше по сравнению с архитектурой x86_64 и mips64el. Следует отметить также и диапазон длин последовательностей, который шире для архитектуры arm64.



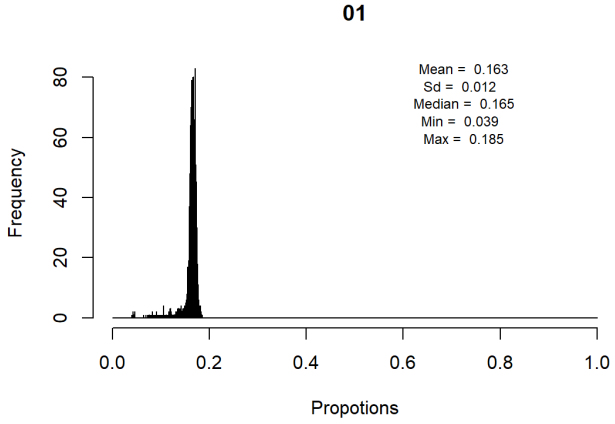
(a) Последовательность "0"



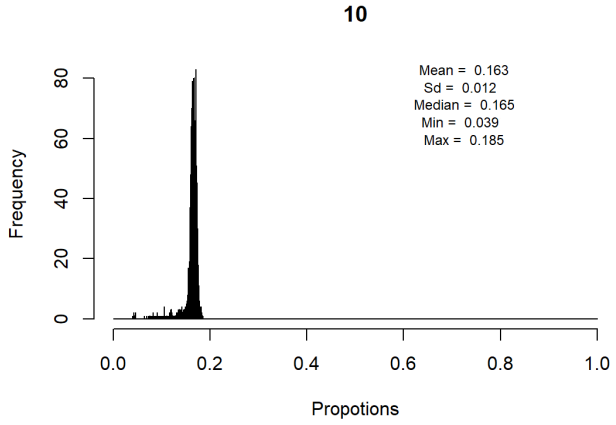
(b) Последовательность "1"



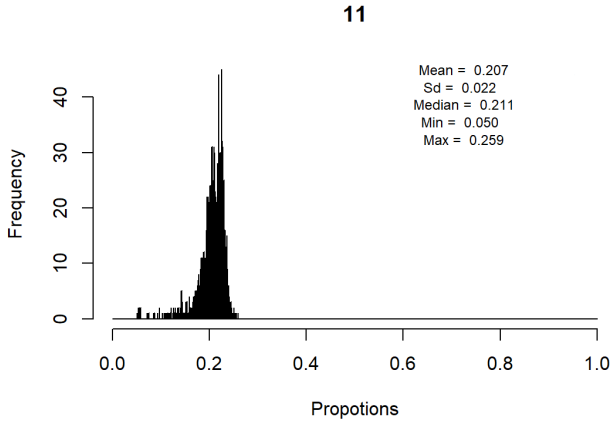
(c) Последовательность "00"



(d) Последовательность "01"



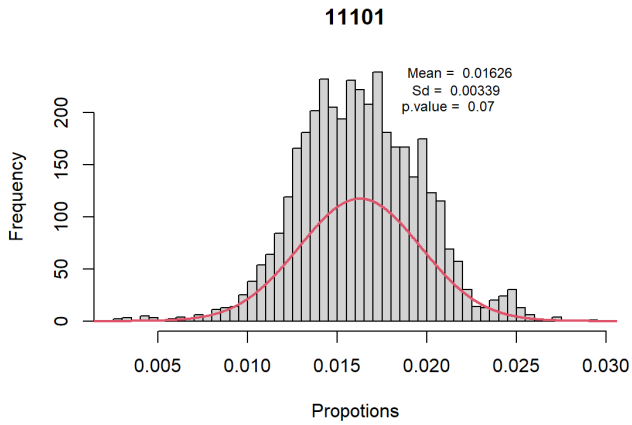
(e) Последовательность "10"



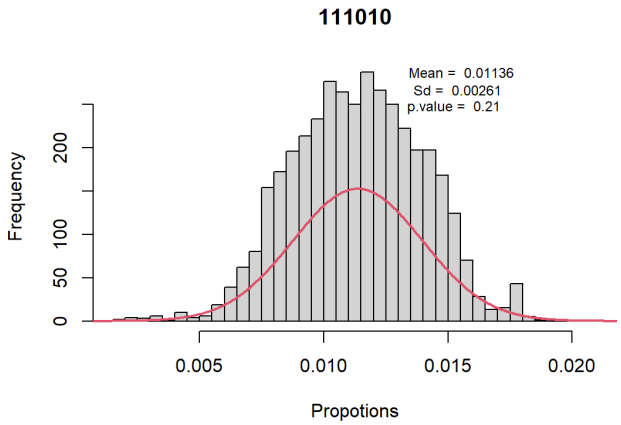
(f) Последовательность "11"

Рис. 6: Распределение последовательностей для arm64

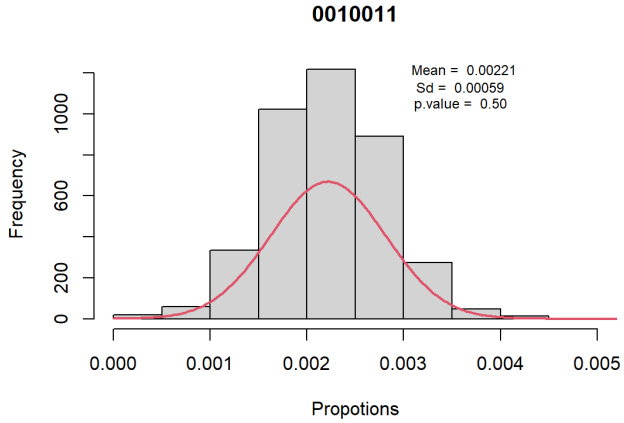
На рисунке 8 представлены некоторые "редкие" последовательности. Графики содержат информацию о среднем, стандартном отклонении и доле файлов, не содержащих данную последовательность.



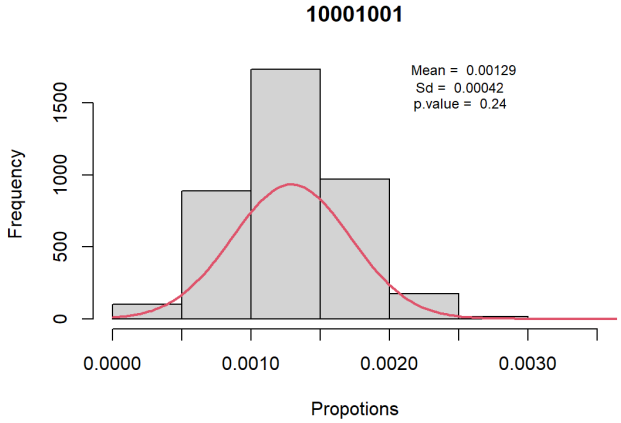
(a) Последовательность "11101"



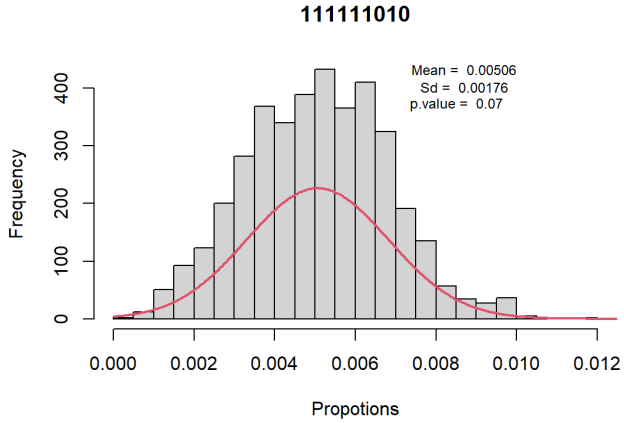
(b) Последовательность "111010"



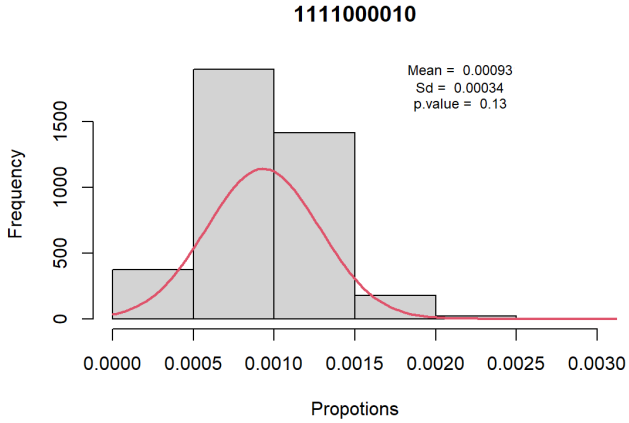
(c) Последовательность "0010011"



(d) Последовательность "10001001"



(e) Последовательность "111111010"

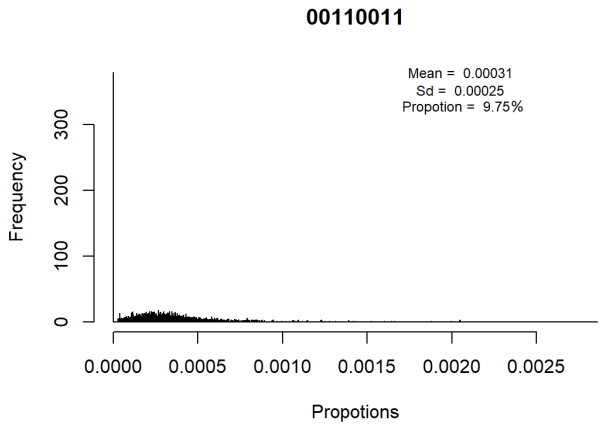


(f) Последовательность "1111000010"

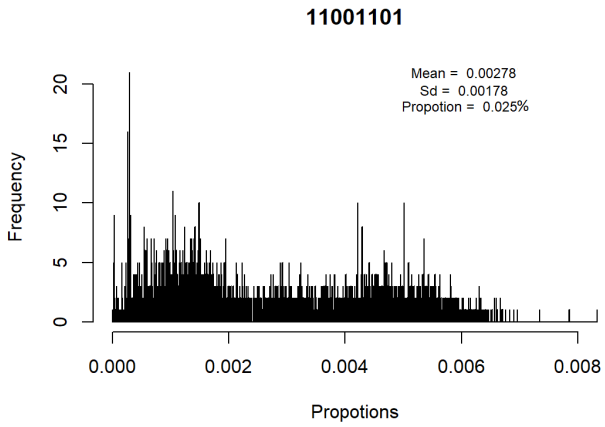
Рис. 7: Распределение последовательностей для arm64

Как и для архитектуры x86_64, здесь тоже имеется вариативность в выборе "редких" ппоследовательностей в плане частоты нахождения, что позволяет регулировать невидимость водяного знака. Всего найдено 517 редких последовательностей длиной 8, 9 и 10

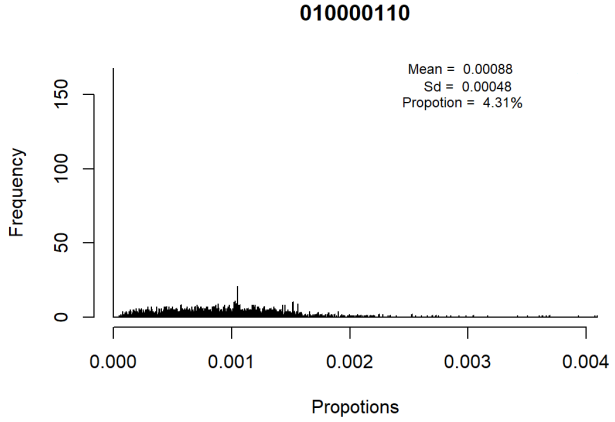
бит.



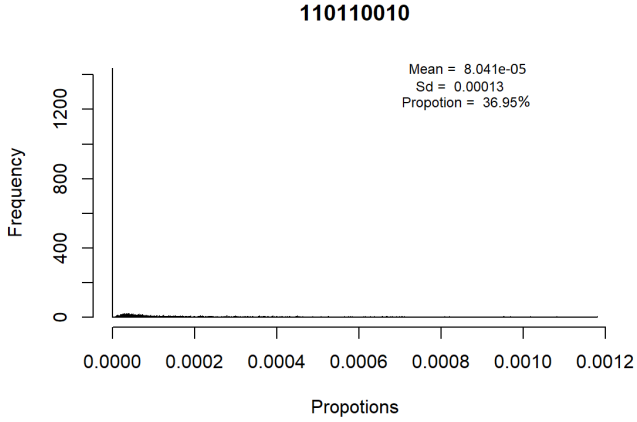
(a) Последовательность "00110011"



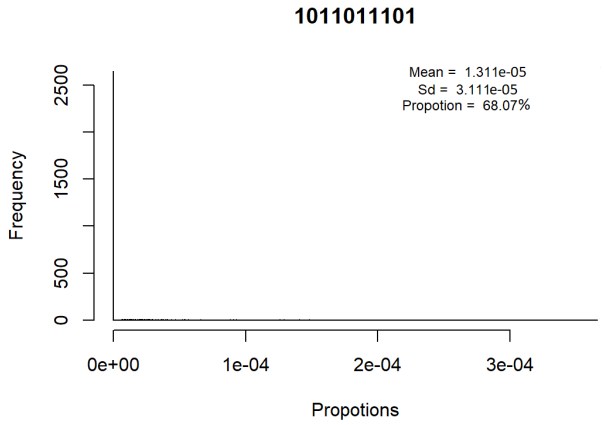
(b) Последовательность "11001101"



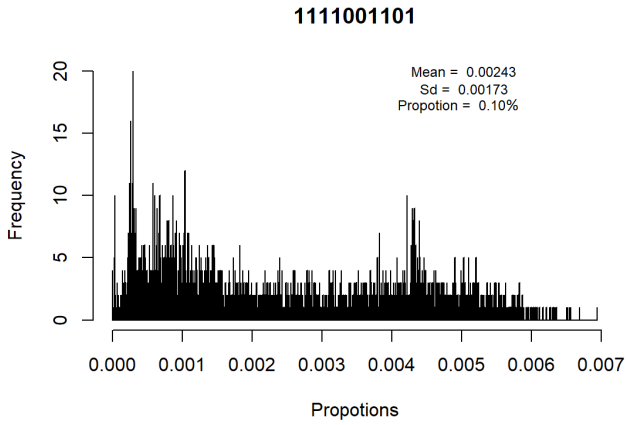
(c) Последовательность "010000110"



(d) Последовательность "110110010"



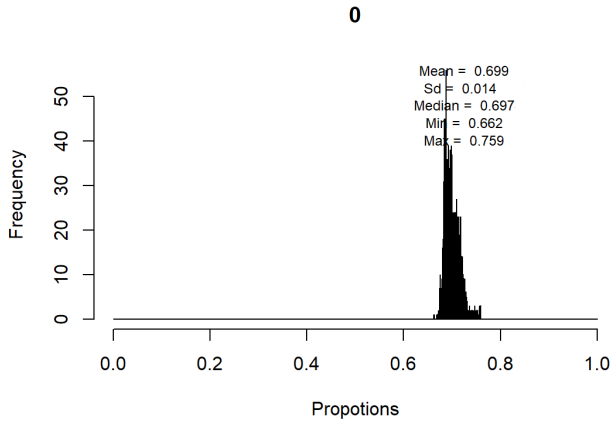
(e) Последовательность "1011011101"



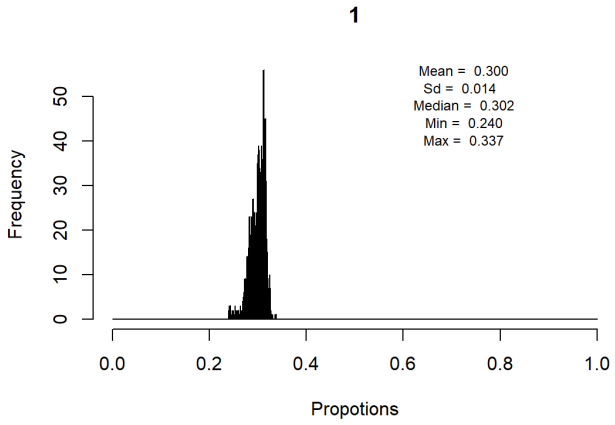
(f) Последовательность "1111001101"

Рис. 8: Распределение последовательностей для arm64

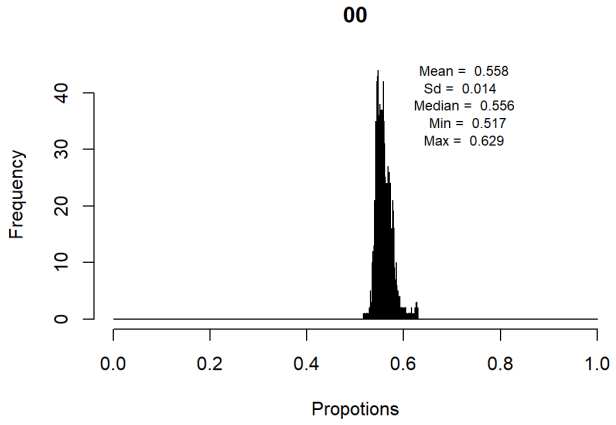
5.4 Архитектура mips64el



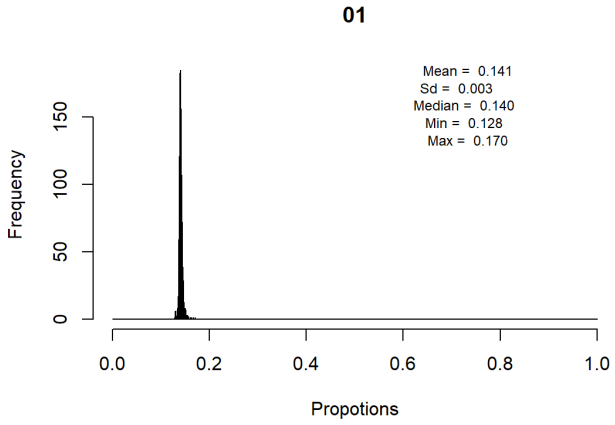
(a) Последовательность "0"



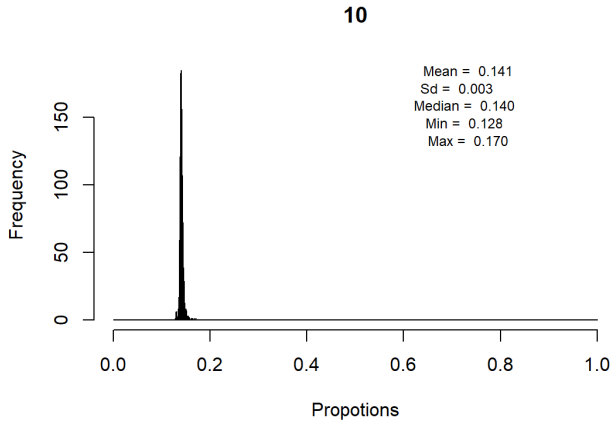
(b) Последовательность "1"



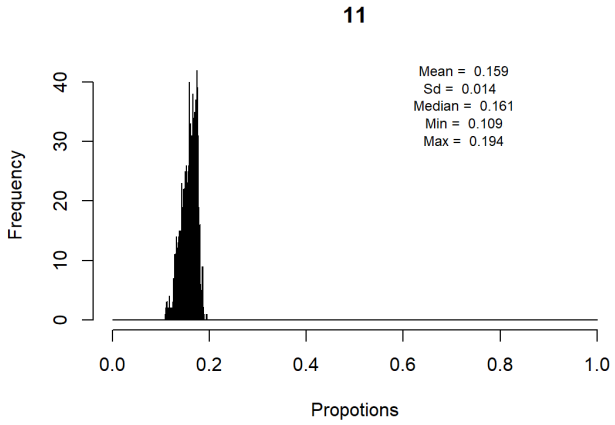
(c) Последовательность "00"



(d) Последовательность "01"



(e) Последовательность "10"

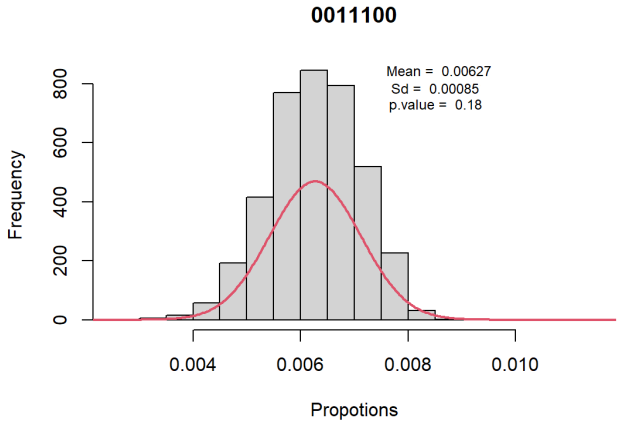


(f) Последовательность "11"

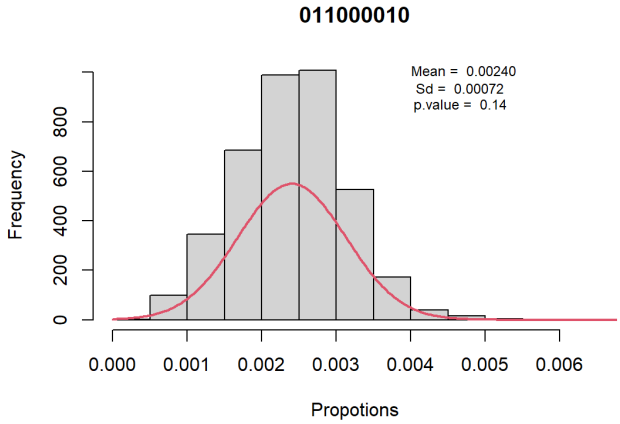
Рис. 9: Распределение последовательностей для mips64el

В этом подразделе представлены результаты для архитектуры mips64el. Вычисленные статистики для последовательностей дли-

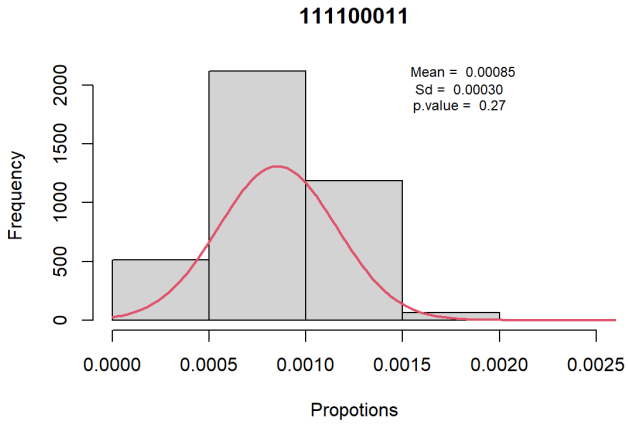
ны меньше 2-х представлены на рисунке 9.



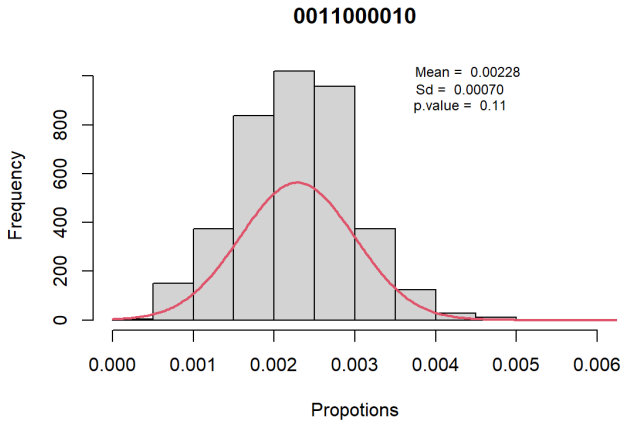
(a) Последовательность "0011100"



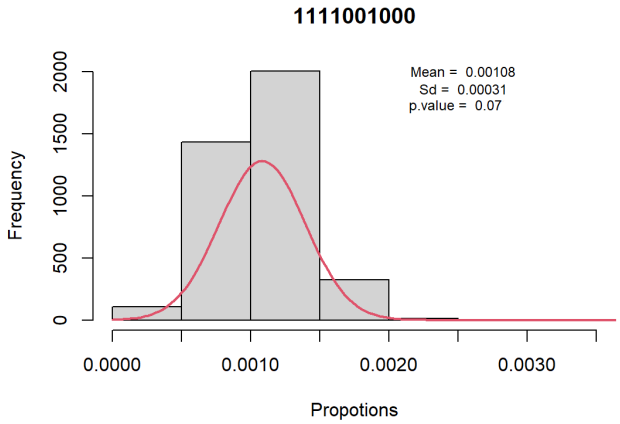
(b) Последовательность "011000010"



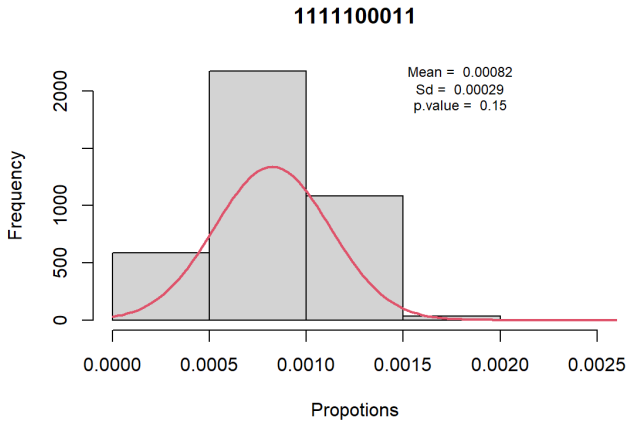
(c) Последовательность "111100011"



(d) Последовательность "0011000010"



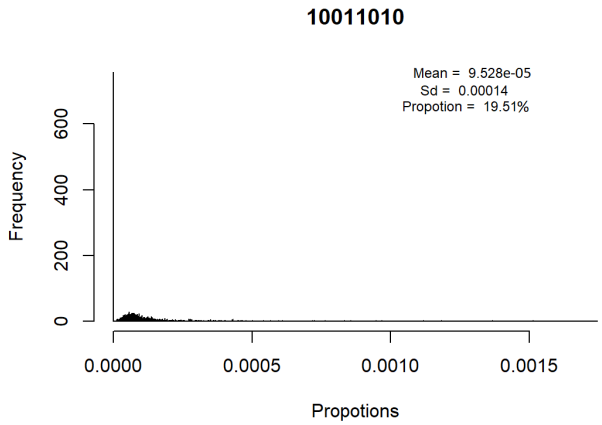
(e) Последовательность "1111001000"



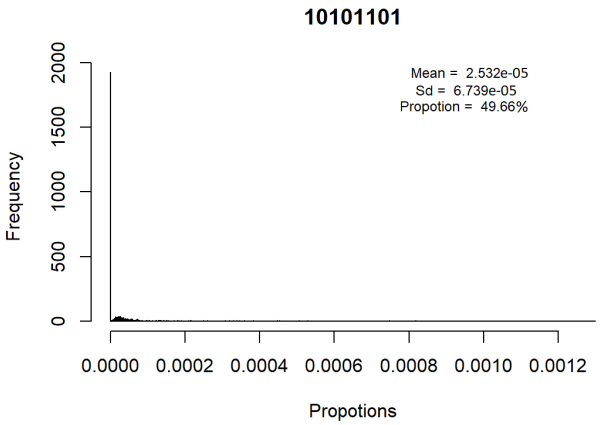
(f) Последовательность "1111100011"

Рис. 10: Распределение последовательностей для mips64el

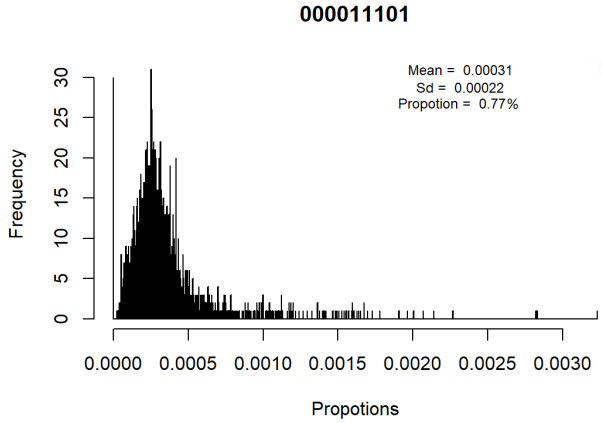
По сравнению с рассмотренными выше архитектурами в mips64el содержится значительно больше нулей. Нулей примерно 70%, а единиц 30%.



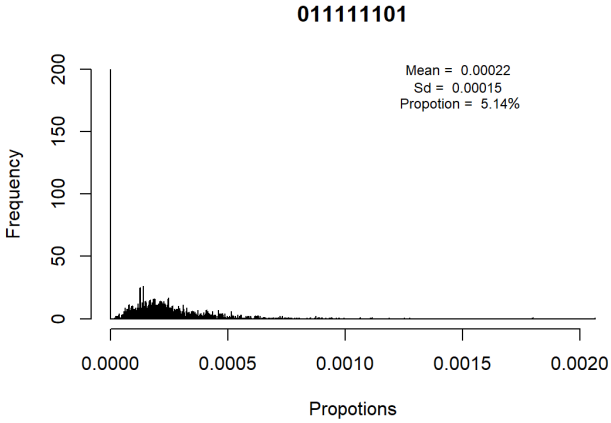
(a) Последовательность "10011010"



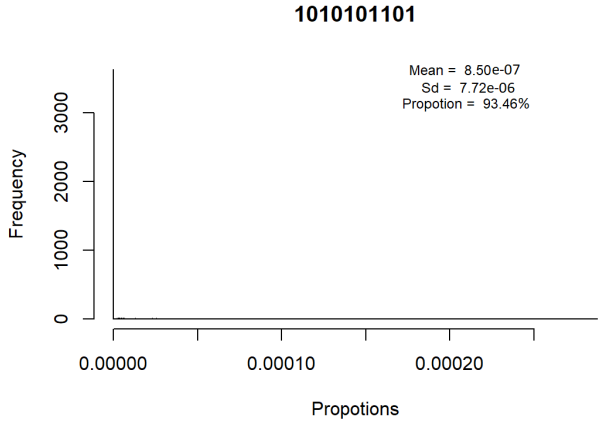
(b) Последовательность "10101101"



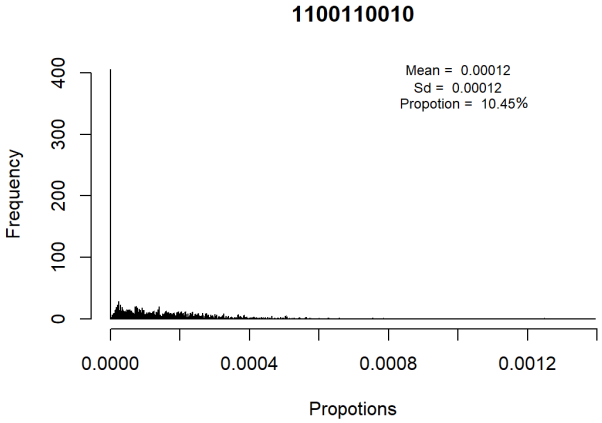
(c) Последовательность "000011101"



(d) Последовательность "011111101"



(e) Последовательность "1010101101"



(f) Последовательность "1100110010"

Рис. 11: Распределение последовательностей для mips64el

На рисунке 10 приведены некоторые последовательности, распределение долей которых соотвествует нормальному. Графики содержат информацию о нормальном распределении и значение p-value, с которым прошёл тест Колмогорова-Смирнова для данной

последовательности.

Всего было найдено 7 последовательностей длиной 8, 9 и 10 бит, распределение долей которых соответствует нормальному.

На рисунке 11 представлены некоторые "редкие" последовательности. Графики содержат информацию о среднем, стандартном отклонении и доле файлов, не содержащих данную последовательность.

Всего найдено 406 "редких" последовательностей, что значительно меньше, чем в случае x86_64 и arm64. Как и для рассмотренных выше случаев, здесь присутствует разнообразие частот нахождения последовательностей в бинарных файлах, что позволяет управлять невидимостью водяного знака.

5.5 Апробация компонента работы с водяным знаком

Для апробации компонента работы с водяным знаком в соответствующем репозитории[31] был настроен CI.

В рамках тестирования для каждого теста из набора производится внесение водяного знака в код и его извлечение. Затем сравнивается значение внесённого и извлечённого водяного знака.

В качестве тестового набора выступают несколько небольших примеров на C++.

Тестирование производится на операционной системе ubuntu 18.04.

Заключение

В рамках данной работы были достигнуты следующие результаты:

- Выполнено сравнение статических и динамических водяных знаков. Изучена концепция MAK - её возможностей, достоинств и

[31] https://github.com/IvanArkhipov1999/watermark-prototype. Accessed: 11-05-2023.

недостатков. Выполнено описание водяных знаков в MAK, приведены примеры и ограничения их применимости. Изучены результаты предыдущей работы по проекту.

- В рамках создания архитектуры фреймворка выделена компонента для работы с данными (модуль компиляции, модуль сбора статистик и модуль анализа) и компонента для работы с водяным знаком (модуль внесения водяного знака и модуль извлечения водяного знака). Фреймворк поддерживает C++ и архитектуру x86_64, имеются возможности расширения на другие языки и архитектуры процессоров.

- Создан прототипный вариант разработанной архитектуры. Компонент работы с данными был создан на языках Python (модули компиляции и сбора статистик) и R (модуль анализа) и поддерживает любые языки и архитектуры процессора. Компонента работы с водяным знаком реализован на Python.

- Для апробации разработанного фреймворка был выбран набор тестов для g++. Было подготовлено по 3884 бинарных файлов для архитектур x86_64, arm64 и mips64el. Для тестирования компонента работы с водяным знаком был реализован CI с несколькими тестовыми примерами.

Список литературы

[1] Business Software Alliance. Piracy impact study 2010: The economic benefits of reducing software piracy, 2010.

[2] Business Software Alliances (BSA). Piracy study, Fourth Annual BSA And IDC global software, http://www.bsa.org/globalstudy/; 2007.

[3] Peitz, Martin, and Patrick Waelbroeck. "The effect of internet piracy on CD sales: Cross-section evidence."Available at SSRN 511763 (2004).

[4] Oganyan, V. A., M. V. Vinogradova, and D. V. Volkov. "Internet piracy and vulnerability of digital content."European Research Studies 21.4 (2018): 735-743.

[5] Johnson, Neil F., and Sushil Jajodia. "Exploring steganography: Seeing the unseen."Computer 31.2 (1998): 26-34.

[6] Potdar, Vidyasagar M., Song Han, and Elizabeth Chang. "A survey of digital image watermarking techniques."INDIN'05. 2005 3rd IEEE International Conference on Industrial Informatics, 2005.. IEEE, 2005.

[7] Begum, Mahbuba, and Mohammad Shorif Uddin. "Digital image watermarking techniques: a review."Information 11.2 (2020): 110.

[8] Coatrieux, Gouenou, et al. "A review of image watermarking applications in healthcare."2006 International conference of the IEEE Engineering in Medicine and Biology Society. IEEE, 2006.

[9] Asikuzzaman, Md, and Mark R. Pickering. "An overview of digital video watermarking."IEEE Transactions on Circuits and Systems for Video Technology 28.9 (2017): 2131-2153.

[10] Doerr, Gwenael, and Jean-Luc Dugelay. "A guide tour of video watermarking."Signal processing: Image communication 18.4 (2003): 263-282.

[11] Yu, Xiaoyan, Chengyou Wang, and Xiao Zhou. "A survey on robust video watermarking algorithms for copyright protection."Applied Sciences 8.10 (2018): 1891.

[12] Paul, Rini T. "Review of robust video watermarking techniques."IJCA Special Issue on Computational Science 3 (2011): 90-95.

[13] Hua, Guang, et al. "Twenty years of digital audio watermarking—a comprehensive review."Signal processing 128 (2016): 222-242.

[14] Arnold, Michael. "Audio watermarking: Features, applications and algorithms."2000 IEEE International conference on multimedia and expo. ICME2000. Proceedings. Latest advances in the fast changing world of multimedia (cat. no. 00TH8532). Vol. 2. IEEE, 2000.

[15] Kim, Hyoung Joong, et al. "Audio watermarking techniques."Intelligent Watermarking Techniques 7 (2004): 185.

[16] Aiswarya, K. K., et al. "Application of Secret Sharing Scheme in Software Watermarking."International Conference on Information and Communication Technology for Intelligent Systems. Springer, Singapore, 2020.

[17] Mullins, J. Alex, et al. "Evaluating Security of Executable Steganography for Digital Software Watermarking."SoutheastCon 2022. IEEE, 2022.

[18] Bento, Lucila, et al. "Software watermarking: contributions that transcend the theme."Cadernos do IME-Série Informática 47 (2022): 14-18.

[19] Kang, Honggoo, et al. "SoftMark: Software Watermarking via a Binary Function Relocation."Annual Computer Security Applications Conference. 2021.

[20] Ma, Haoyu, et al. "Xmark: dynamic software watermarking using Collatz conjecture."IEEE Transactions on Information Forensics and Security 14.11 (2019): 2859-2874.

[21] Kamel, Ibrahim, and Qutaiba Albluwi. "A robust software watermarking for copyright protection."computers & security 28.6 (2009): 395-409.

[22] Venkatachalam, Balaji. "Software watermarking as a proof of identity: A study of zero knowledge proof based software watermarking."International Workshop on Digital Watermarking. Springer, Berlin, Heidelberg, 2005.

[23] Myles, Ginger, and Hongxia Jin. "Self-validating branch-based software watermarking."International Workshop on Information Hiding. Springer, Berlin, Heidelberg, 2005.

[24] Sun, Xingming, et al. "Digital watermarking method for data integrity protection in wireless sensor networks."International Journal of Security and Its Applications 7.4 (2013): 407-416.

[25] Dey, Ayan, Sukriti Bhattacharya, and Nabendu Chaki. "Software watermarking: Progress and challenges."INAE Letters 4.1 (2019): 65-75.

[26] Collberg, Christian, and Clark Thomborson. On the limits of software watermarking. Department of Computer Science, The University of Auckland, New Zealand, 1998.

[27] Holmes, Keith. "Computer software protection."U.S. Patent No. 5,287,407. 15 Feb. 1994.

[28] Samson, Peter R. "Apparatus and method for serializing and validating copies of computer software."U.S. Patent No. 5,287,408. 15 Feb. 1994.

[29] Monden, Akito, et al. "A practical method for watermarking java programs."Proceedings 24th Annual International Computer

Software and Applications Conference. COMPSAC2000. IEEE, 2000.

[30] Venners, Bill. "The java virtual machine."Java and the Java virtual machine: definition, verification, validation (1998).

[31] Fukushima, Kazuhide, and Kouichi Sakurai. "A software fingerprinting scheme for java using classfiles obfuscation."International Workshop on Information Security Applications. Springer, Berlin, Heidelberg, 2003.

[32] Fukushima, Kazuhide, et al. "A software fingerprinting scheme for java using class structure transformation."Transactions of Information Processing Society of Japan 46.8 (2005): 2042-2052.

[33] Pervez, Zeeshan, Yasir Mahmood, and Hafiz Farooq Ahmad. "Semblance based disseminated software watermarking algorithm."2008 23rd International Symposium on Computer and Information Sciences. IEEE, 2008.

[34] Thaker, Smita. "Software watermarking via assembly code transformations."San Jose State University (2004).

[35] El-Khalil, Rakan, and Angelos D. Keromytis. "Hydan: Information hiding in program binaries."International Conference on Informaton and Communications Security. 2004.

[36] Stern, Julien P., et al. "Robust object watermarking: Application to code."International Workshop on Information Hiding. Springer, Berlin, Heidelberg, 2000.

[37] Hachez, Gael. "A comparative study of software protection tools suited for e-commerce with contributions to software watermarking and smart cards."Universite Catholique de Louvain (2003).

[38] Collberg, Christian, and Tapas Ranjan Sahoo. "Software watermarking in the frequency domain: implementation, analysis, and attacks."Journal of Computer Security 13.5 (2005): 721-755.

[39] Curran, D., Neil J. Hurley, and Mel Ó. Cinnéide. "Securing Java through software watermarking."PPPJ. Vol. 3. 2003.

[40] Ai, Jieqing, et al. "A stern-based Collusion-Secure software watermarking algorithm and its implementation."2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07). IEEE, 2007.

[41] Robert Davidson and Nathan Myhrvold. Method and system for generating and auditing a signature for a computer program, June 1996. Microsoft Corporation, US Patent 5559884.

[42] Myles, Ginger, et al. "The evaluation of two software watermarking algorithms."Software: Practice and Experience 35.10 (2005): 923-938.

[43] Hattanda, Kazuhiro, and Shuichi Ichikawa. "The evaluation of Davidson's digital signature scheme."IEICE transactions on fundamentals of electronics, communications and computer sciences 87.1 (2004): 224-225.

[44] Anckaert, Bertrand, Bjorn De Sutter, and Koen De Bosschere. "Covert communication through executables."Program Acceleration through Application and Architecture Driven Code Transformations: Symposium Proceedings. 2004.

[45] Shirali-Shahreza, Mohammad, and Sajad Shirali-Shahreza. "Software watermarking by equation reordering."2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications. IEEE, 2008.

[46] Sha, Zonglu, Hua Jiang, and Aicheng Xuan. "Software watermarking algorithm by coefficients of equation."2009 Third International Conference on Genetic and Evolutionary Computing. IEEE, 2009.

[47] Sharma, B. K., R. P. Agarwal, and Raghuraj Singh. "An efficient software watermark by equation reordering and fdos."Proceedings

of the International Conference on Soft Computing for Problem Solving (SocProS 2011) December 20-22, 2011. Springer, New Delhi, 2012.

[48] Gong, Daofu, et al. "Hiding informationin in java class file."2008 International Symposium on Computer Science and Computational Technology. Vol. 2. IEEE, 2008.

[49] Qu, Gang, and Miodrag Potkonjak. "Analysis of watermarking techniques for graph coloring problem."Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design. 1998.

[50] Qu, Gang, and Miodrag Potkonjak. "Fingerprinting intellectual property using constraint-addition."Proceedings of the 37th annual design automation conference. 2000.

[51] Zhu, William, and Clark Thomborson. "Extraction in software watermarking."Proceedings of the 8th workshop on Multimedia and security. 2006.

[52] Zhu, William, and Clark Thomborson. "Algorithms to watermark software through register allocation."International Conference on Digital Rights Management. Springer, Berlin, Heidelberg, 2005.

[53] Zhu, William, and Clark Thomborson. "Recognition in software watermarking."Proceedings of the 4th ACM international workshop on Contents protection and security. 2006.

[54] Le, Tri Van, and Yvo Desmedt. "Cryptanalysis of UCLA watermarking schemes for intellectual property protection."International Workshop on Information Hiding. Springer, Berlin, Heidelberg, 2002.

[55] Qu, Gang, and Miodrag Potkonjak. "Hiding signatures in graph coloring solutions."International Workshop on Information Hiding. Springer, Berlin, Heidelberg, 1999.

[56] Myles, Ginger, and Christian Collberg. "Software watermarking through register allocation: Implementation, analysis, and attacks."International Conference on Information Security and Cryptology. Springer, Berlin, Heidelberg, 2004.

[57] Lee, Hakun, and Keiichi Kaneko. "New approaches for software watermarking by register allocation."2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing. IEEE, 2008.

[58] Lee, Hakun, and Keiichi Kaneko. "Two new algorithms for software watermarking by register allocation and their empirical evaluation."2009 Sixth International Conference on Information Technology: New Generations. IEEE, 2009.

[59] Li, Jun, and Quan Liu. "Design of a software watermarking algorithm based on register allocation."2010 2nd International Conference on E-business and Information System Security. IEEE, 2010.

[60] Jiang, Zetao, Rubing Zhong, and Bina Zheng. "A software watermarking method based on Public-Key cryptography and graph coloring."2009 Third International Conference on Genetic and Evolutionary Computing. IEEE, 2009.

[61] Rivest, Ronald L., Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems."Communications of the ACM 21.2 (1978): 120-126.

[62] Ghiya, Rakesh, and Laurie J. Hendren. "Is it a tree, a DAG, or a cyclic graph? A shape analysis for heap-directed pointers in C."Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. 1996.

[63] Ramalingam, Ganesan. "The undecidability of aliasing."ACM Transactions on Programming Languages and Systems (TOPLAS) 16.5 (1994): 1467-1471.

[64] Collberg, Christian, et al. "Error-correcting graphs for software watermarking."Proceedings of the 29th workshop on graph theoretic concepts in computer science. 2003.

[65] Read, Ronald C., ed. Graph theory and computing. Academic Press, 2014.

[66] Nagra, Jasvir, and Christian Collberg. Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection: Obfuscation, Watermarking, and Tamperproofing for Software Protection. Pearson Education, 2009.

[67] Chen, XiaoJiang, et al. "A dynamic graph watermark scheme of tamper resistance."2009 Fifth International Conference on Information Assurance and Security. Vol. 1. IEEE, 2009.

[68] Ping, Zhou, Chen Xi, and Yang Xu-Guang. "The software watermarking for tamper resistant radix dynamic graph coding."Inform. Technol. J 9 (2010): 1236-1240.

[69] Jian-Qi, Zhu, Liu Yan-Heng, and Yin Ke-Xin. "A novel planar IPPCT tree structure and characteristics analysis."Journal of Software 5.3 (2010): 344-351.

[70] Yong, Wang, and Yang Yixian. "A software watermark database scheme based on PPCT."CIHW2004 (2004).

[71] Zhu, Jianqi, Yanheng Liu, and Kexin Yin. "A novel dynamic graph software watermark scheme."2009 First International Workshop on Education Technology and Computer Science. Vol. 3. IEEE, 2009.

[72] Zhu, Jianqi, Kexin Yin, and Yanheng Liu. "A novel DGW scheme based on 2D_PPCT and permutation."2009 International Conference on Multimedia Information Networking and Security. Vol. 2. IEEE, 2009.

[73] Venkatesan, Ramarathnam, Vijay Vazirani, and Saurabh Sinha. "A graph theoretic approach to software watermarking."International

Workshop on Information Hiding. Springer, Berlin, Heidelberg, 2001.

[74] Venkatesan, Ramarathnam, and Vijay Vazirani. "Technique for producing through watermarking highly tamper-resistant executable code and resulting “watermarked” code so formed."U.S. Patent No. 7,051,208. 23 May 2006.

[75] Hecht, Matthew S., and Jeffrey D. Ullman. "Flow graph reducibility."Proceedings of the fourth annual ACM symposium on Theory of computing. 1972.

[76] Chroni, Maria, and Stavros D. Nikolopoulos. "Encoding watermark integers as self-inverting permutations."Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies. 2010.

[77] Arora, Sanjeev, Satish Rao, and Umesh Vazirani. "Expander flows, geometric embeddings and graph partitioning."Journal of the ACM (JACM) 56.2 (2009): 1-37.

[78] Collberg, Christian, et al. "Graph theoretic software watermarks: Implementation, analysis, and attacks."Information Hiding: 6th International Workshop, IH 2004, Toronto, Canada, May 23-25, 2004, Revised Selected Papers 6. Springer Berlin Heidelberg, 2005.

[79] Collberg, Christian, et al. "More on graph theoretic software watermarks: Implementation, analysis, and attacks."Information and Software Technology 51.1 (2009): 56-67.

[80] Collberg, Christian, Clark Thomborson, and Douglas Low. "Manufacturing cheap, resilient, and stealthy opaque constructs."Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. 1998.

[81] Arboit, Genevieve. "A method for watermarking java programs via opaque predicates."The Fifth International Conference on Electronic Commerce Research (ICECR-5). 2002.

[82] Myles, Ginger, and Christian Collberg. "Software watermarking via opaque predicates: Implementation, analysis, and attacks."Electronic Commerce Research 6.2 (2006): 155-171.

[83] Alitavoli, Mohammad, Mahdi Joafshani, and Aida Erfanian. "A novel watermarking method for java programs."Proceedings of the 28th Annual ACM Symposium on Applied Computing. 2013.

[84] Cousot, Patrick, and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints."Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. 1977.

[85] Cousot, Patrick, and Radhia Cousot. "An abstract interpretation-based framework for software watermarking."ACM Sigplan Notices 39.1 (2004): 173-185.

[86] Collberg C, Thomborson C. Software watermarking: Models and dynamic embeddings. Principles of Programming Languages (POPL'99); 1999. p. 311–24.

[87] Tamada, Haruaki, et al. Detecting the traft of programs using birthmarks. Nara Institute of Science and Technology, 2003.

[88] Tamada, Haruaki, et al. "Design and evaluation of birthmarks for detecting theft of java programs."IASTED Conf. on Software Engineering. 2004.

[89] Nagra, Jasvir, and Clark Thomborson. "Threading software watermarks."Information Hiding: 6th International Workshop, IH 2004, Toronto, Canada, May 23-25, 2004, Revised Selected Papers 6. Springer Berlin Heidelberg, 2005.

[90] Collberg, Christian, et al. "Dynamic path-based software watermarking."Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation. 2004.

[91] Gupta, Gaurav, and Josef Pieprzyk. "A low-cost attack on branch-based software watermarking schemes."Digital Watermarking: 5th International Workshop, IWDW 2006, Jeju Island, Korea, November 8-10, 2006. Proceedings 5. Springer Berlin Heidelberg, 2006.

[92] Gupta, Gaurav, and Josef Pieprzyk. "Software watermarking resilient to debugging attacks."Journal of multimedia 2.2 (2007): 10-16.

[93] Chen, Zhe, Chunfu Jia, and Donghui Xu. "Hidden path: dynamic software watermarking based on control flow obfuscation."2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC). Vol. 2. IEEE, 2017.

[94] Tian, Zhenzhou, et al. "Software plagiarism detection with birthmarks based on dynamic key instruction sequences."IEEE Transactions on Software Engineering 41.12 (2015): 1217-1235.

[95] Palsberg, Jens, et al. "Experience with software watermarking."Proceedings 16th Annual Computer Security Applications Conference (ACSAC'00). IEEE, 2000.

[96] He, Yong, and M. Sc. Tamperproofing a software watermark by encoding constants. Diss. University of Auckland, 2002.

[97] Mehta, Brijesh B., and Udai Pratap Rao. "A Novel approach as Multi-place Watermarking for Security in Database."arXiv preprint arXiv:1402.7341 (2014).

[98] Shacham, Hovav. "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)."Proceedings of the 14th ACM conference on Computer and communications security. 2007.

[99] Ma, Haoyu, et al. "Software watermarking using return-oriented programming."Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security. 2015.

[100] LaTourette, Kelly S. Explorations of the collatz conjecture. Bethlehem, PA, 2007.

[101] Andrei, Ştefan, and Cristian Masalagiu. "About the Collatz conjecture."Acta Informatica 35.2 (1998): 167-179.

[102] Lim, Hyun-Il. "A performance comparison on characteristics of static and dynamic software watermarking methods."Indian Journal of Science and Technology 8.21 (2015): 1-6.

[103] Kumar, Krishan, and Prabhpreet Kaur. "A Comparative Analysis of Static and Dynamic Java Bytecode Watermarking Algorithms."Software Engineering: Proceedings of CSI 2015. Springer Singapore, 2019.

[104] Hamilton, James, and Sebastian Danicic. "A survey of static software watermarking."2011 World Congress on Internet Security (WorldCIS-2011). IEEE, 2011.

[105] Nematollahi, Mohammad Ali, Chalee Vorakulpipat, and Hamurabi Gamboa Rosales. Digital watermarking. Springer Singapore:, 2017.

[106] Zhang, Yanqun. "Digital watermarking technology: A review."2009 ETP international conference on future computer and communication. IEEE, 2009.

[107] Mnkash, Sarah H., and Matheel E. Abdulmunem. "A review of software watermarking."Iraqi Journal of Science (2020): 2740-2750.

[108] Zhu, William, Clark Thomborson, and Fei-Yue Wang. "A survey of software watermarking."Intelligence and Security Informatics: IEEE International Conference on Intelligence and Security

Informatics, ISI 2005, Atlanta, GA, USA, May 19-20, 2005. Proceedings 3. Springer Berlin Heidelberg, 2005.

[109] Смирнов Д.П. Нанесение водяных знаков на программное обеспечение. Курсовая работа кафедры системного программирования СПбГУ, 2018

[110] Rosenblum, Nathan E., Barton P. Miller, and Xiaojin Zhu. "Extracting compiler provenance from program binaries."Proceedings of the 9th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering. 2010.

[111] Rosenblum, Nathan, Xiaojin Zhu, and Barton P. Miller. "Who wrote this code? identifying the authors of program binaries."Computer Security–ESORICS 2011: 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings 16. Springer Berlin Heidelberg, 2011.

[112] Caliskan, Aylin, et al. "When coding style survives compilation: De-anonymizing programmers from executable binaries."arXiv preprint arXiv:1512.08546 (2015).

[113] Pizzolotto, Davide, and Katsuro Inoue. "Identifying compiler and optimization level in binary code from multiple architectures."IEEE Access 9 (2021): 163461-163475.

[114] Chen, Yu, et al. "Himalia: Recovering compiler optimization levels from binaries by deep learning."Intelligent Systems and Applications: Proceedings of the 2018 Intelligent Systems Conference (IntelliSys) Volume 1. Springer International Publishing, 2019.

[115] Rosenblum, Nathan, Barton P. Miller, and Xiaojin Zhu. "Recovering the toolchain provenance of binary code."Proceedings

of the 2011 International Symposium on Software Testing and Analysis. 2011.

[116] TIS Committee. Tool Interface Standard (TIS); Executable and Linking Format (ELF); Specification. Version 1.2. May 1995