**Санкт-Петербургский государственный университет**

Кафедра системного программирования
Математическое обеспечение и администрирование
информационных систем

Архипов Иван Сергеевич

# Нанесение водяных знаков на программное обеспечение

Отчёт по производственной 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 industrial 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. Убытки от пиратства в индустрии составляют 12 миллиардов в год [1]. С распространением интернет-технологий данная проблема становится всё более острой. Поэтому защита программного обеспечения от нелицензионного использования является как никогда актуальной задачей.

Для решения данной проблемы была разработана технология под названием "цифровой водяной знак". Это некая информация, встраиваемая в цифровой продукт без повреждения его эксплуатации. Как правило, водяной знак содержит сведения о владельце программного обеспечения. Работа с водяным знаком состоит из нанесения водяного знака, то есть встраивания дополнительной информации в цифровой продукт, эксплуатации продукта без нарушения функциональности и распознания водяного знака с целью подтверждения авторства в случае незаконного использования программного обсепечения.

На кафедре системного программирования СПБГУ в исследовательской группе под руководством М.В.Баклановского была разработана архитектура МАК, открывающая новые возможности для преобразования программного кода. В рамках этой архитектуры ими был предложен новый подход нанесения водяных знаков. Данная работа посвящена исследованию, необходимому для реализации данного подхода.

# 1 Цели и задачи

Целью работы является создание спецификации нанесения водяных знаков на программное обеспечение. Для достижения обозначенной цели были посталены следующие задачи:

- обзор водяных знаков;
- освоение стека технологий;
- подготовка данных для нахождения статистик и редких битовых последовательностей;
- нахождение определённых статистик и редких битовых последовательностей;
- анализ метода нанесения водяного знака;
- написание спецификации.

# 2 Обзор

## 2.1 Обзор водяных знаков

По реализации водяные знаки делятся на статические и динамические. Рассмотрим особенности каждой из этой группы и приведём примеры. В конце раздела даны ссылки на литературу, посвящённую более подробному обзору существующих технологий.

### 2.1.1 Статические водяные знаки

Эти водяные знаки не требуют запуска программы для распознания. Их достоинством является удобство, так как нет необходимости прикладывать усилия для поддержки различных эффектов, вызванных запуском программного обеспечения на различных платформах. Однако подобные водяные знаки сравнительно несложно извлечь: достаточно проанализировать код и данные, нет необходимости анализа работы программы.

Статические водяные знаки делятся на водяные знаки данных и водяные знаки кода. В первом случае в определённую секцию данных заносится необходимая информация. Такие водяные знаки очень просто извлечь, так как он, по сути, и не спрятан.

Водяной знак кода содержит спрятанную информацию в самом программном коде. Например, можно использовать перестановку линейных участков кода [2] или выделение регистров [3, 4, 5, 6].

### 2.1.2 Динамические водяные знаки

Данные водяные знаки основаны на внедрении информации в поведение программы. Их извлечение требует анализа поведения и состояний программы, что гораздо более трудоёмко, чем анализ текста программы или данных.

Одной из самых известных техник является пасхальное яйцо [1, 7]. Данный метод основан на внедрении кода, срабатывающего в 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.

Другим примером динамического водяного знака является метод, основанный на динамическом пути исполнения программы. Например, информация может быть спрятана в последовательности вызова функций. Из-за того, что структура программы достаточно сложно, непросто провести анализ пути исполнения программы. Также сложной задачей является и извлечение такого водяного знака, так как он сильно связан с семантикой программы. Следовательно, этот метод может обеспечить достаточную устойчивость к атакам [8]. Недостатком является изменение пути выполнения в результате различных преобразований, например, в результате оптимизаций или разработки новой версии продукта.

Примеров можно привести гораздо больше, однако в данной работе стоит сосредоточиться и на других аспектах. Ниже приведён обзор на литературу, куда можно обратиться за более исчерпывающим обзором и анализом конкретных водяных знаков.

## 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 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. Нанесённый 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.

- Ортогональность. Хочется иметь возможность наносить несколько водяных знаков для различных целей. Для этого нужно обеспечить независимость водяных знаков и исключить их влияние друг на друга.

## 2.3 Обзор литературы

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 коли-

чественные характеристики водяных знаков и их анализ. Подробнее о направлениях развития в данной области можно прочесть в [9].

Очень хороший обзор в целом по водяным знакам и конкретно по водяным знакам в программном обеспечении можно найти в [10]. Относительно нанесения водяных знаков на программное обеспечение в этом документе представлено не только многообразие методов, но и виды атак на программное обеспечение, небольшая теоретическая формализация нанесения и распознавания водяных знаков, краткий обзор продуктов нанесения водяных знаков и краткий анализ различных методов.

Попытка проанализировать водяные знаки с точки зрения характеристик, описанных выше, была препринята в [11]. 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 [12, 13]. Из них можно подчерпнуть сведения о различных водяных знаках, не описанных в обзорных статьях выше.

## 2.4 Архитектура МАК

На сегодняшний день известно два приниципа архитектуры ЭВМ. Одна из них, архитектура фон Неймана, основна на приниципе хранения программного кода и данных в одном устройстве. Исходя из этого, в данной архитектуре возможна самомодификация кода. Во второй архитектуре, Гарвардской архитектуре, 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 разработана архитектура МАК (рис. 1). В ней на программы накладывается ещё одно важное ограничение: запрет добавления точек

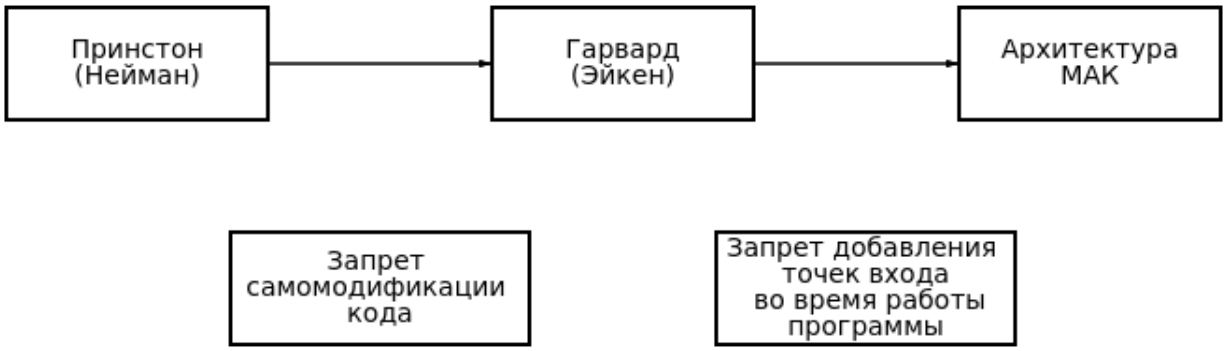


Рис. 1: Архитектурные концепции

входа во время работы программы.

В Гарвардской архитектуре нельзя определить точки входа программы (по теореме Райса). Из-за этого нельзя "раздвинуть" линейные участки кода, перемешать их, записать что-нибудь в промежутки между линейными участками, так как мы не можем заранее сказать, придёт ли управление в данную точку.

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

## 2.5 Предшествующая работа

Тема данной работы рассматривалась в курсовой работе Смирнова Дениса Павловича в 2018 году [14].

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 функционал нанесения и распознавания водяных знаков. В ходе работы он перенёс часть функциональности MAK на Windows, которая позволяет реализовать водяные знаки. В отличие от работы Дениса Павловича, данная работа носит исследовательский характер, её целью является создание спецификации, а не программного продукта. Поэтому в работе не рассматривается данный вопрос.

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

# 3 Водяные знаки в MAK

Как было написано в обзоре архитектуры MAK, можно "раздвинуть" линейные участки кода, а в них записать водяной знак. Способ нанесения водяного знака зависит от его цели. Например, если нужен видимый водяной знак, то можно просто его туда записать, если нужен невидимый, то нужен способ его спрятать.

Хочется иметь способ наносить целый комплекс водяных знаков с разными характеристиками. Для этого необходим способ понимать, что в данной области записан водяной знак. Этот способ называется каркасный водяной знак.

На данный момент предлагается использовать редкие битовые последовательности. Но такие последовательности легко найти, потому нужен способ их спрятать.

Оказалось, что программный код имеет вполне определённые вероятностные характеристики. Например, эмпирически было выяснено, что нулей в программном коде примерно 60 %, а единиц 40 %.

Чтобы спрятать водяной знак, нужно "подогнать" эмпирические статистики в области, где спрятан водяной знак, такие, как у обычного кода.

Также для сокрытия водяного знака можно использовать такие "редкие" последовательности, которые не всегда встречаются в исполняемом файле. Для этого нужно провести анализ частоты нахождения различных битовых последовательностей в коде.

# 4 Извлечение секции кода из исполняемого файла

В elf (Executable and Linkable Format) [15] 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 [15] был разработан и запрограммирован [1] 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 (смещение 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

# 5 Выбор данных для анализа

Для сбора информации о программном коде необходимо выбрать данные для анализа. Были рассмотрены следующие вари-

[1] Весь код для подготовки и анализа данных расположен в репозитории: https://github.com/IvanArkhipov1999/Binary-code-statistics-research. Accessed: 03-01-2023.

анты:

- Все исполняемые файлы на компьютере. Этот вариант предоставляет больше всего данных, однако имеет ряд проблем. Во-первых, сторонний читатель не может повторить эксперименты, описанные в работе. Во-вторых, имеются трудности с определением принадлежности файла к elf формату. В-третьих, все исполняемые файлы собраны только под одну архитектуру, поэтому возникают проблемы с анализом статистик для других архитектур

- Несколько крупных проектов. Из-за малого количества файлов данный выбор даёт информации о распределении статистик

- Тестовая база Clang [2]. Имеется большое количество разнообразных файлов с программным кодом. Однако данные тесты проверяют только кодогенерацию, поэтому они не пригодны для сборки и получения elf-файла

- Тестовая база GCC [3]. В данном варианте также имеется множество разнообразных тестов, которые пригодны для сборки и получения elf-файла

Итак, в качестве данных для анализа была выбрана тестовая база GCC. Весь анализируемый датасет лежит в отдельном репозитории [4].

# 6 Вычисление статистических характеристик и поиск редких последовательностей

В качестве характеристик кода были выбраны доли битовых по-

[2] https://github.com/llvm/llvm-project/tree/main/clang/test/CodeGen. Accessed: 03-01-2023.

[3] https://github.com/gcc-mirror/gcc/tree/master/libstdc++-v3/testsuite. Accessed: 03-01-2023.

[4] https://github.com/IvanArkhipov1999/Binaries-dataset. Accessed: 03-01-2023.

следовательностей различной длины. Это позволяет найти и статистические характеристики кода, и редкие битовые последовательности.

В предыдущей работе по данной теме [14] в качестве редких битовых последовательностей рассматривались исключительно последовательности из единиц, так как доля единиц в бинарном файле равна 40%. Как будет показано далее, это не самый лучший подход.

Рассмотрены все возможные битовые последовательности длины от 1 до 10 включительно. Найдены все битовые последовательности длины до 10 включительно, которые попадаются не во всех бинарных файлов из датасета.

В качестве примера для анализа были изучены исполняемые файлы следующих архитектур: x86_64, arm64 и mips64el.

В следующих подразделах для каждой архитектуры будут приведены небольшая таблица долей последовательностей длиной до 3-х включительно с соответствующими комментариями и таблица с некоторыми редкими последовательностями с комментариями к ней. Полный перечень статистик и редких последовательностей не приведён ввиду их большого объёма.

## 6.1 Архитектура x86_64

В данном подразделе представлены результаты для архитектуры x86_64. Вычисленные статистики для последовательностей длины меньше 3-х представлены в таблице 1.

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

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 последовательностей 01, 10 и 11 примерно одинакова. А при рассмотрении последовательностей длины 3 выяснилось, что самые редкие последовательности содержат две единицы и один нуль. При этом такие

| Последовательность | Среднее | Стандартное отклонение |
|---|---|---|
| 0 | 0.5934 | 0.0302 |
| 1 | 0.4066 | 0.0302 |
| 00 | 0.3945 | 0.0323 |
| 01 | 0.1989 | 0.0164 |
| 10 | 0.1988 | 0.0164 |
| 11 | 0.2077 | 0.0363 |
| 000 | 0.2667 | 0.0379 |
| 001 | 0.1279 | 0.012 |
| 010 | 0.1316 | 0.0182 |
| 011 | 0.0674 | 0.0037 |
| 100 | 0.1278 | 0.0121 |
| 101 | 0.071 | 0.0052 |
| 110 | 0.0673 | 0.0037 |
| 111 | 0.1404 | 0.0339 |

Таблица 1: Доли последовательностей для архитектуры x86_64

последовательности имеют малое стандартное отклонение по сравнению с другими последовательностями.

В таблице 2 приведены некоторые редкие последовательности с их средней долей в файле и процентом файлов, не содержащих данные последовательности.

| Последовательность | Среднее | Доля файлов |
|---|---|---|
| 10110100 | 0.003 | 0.333% |
| 11100110 | 0.00023 | 15.614% |
| 110101100 | 0.0008 | 3.878% |
| 111001101 | 5.275e-05 | 45.917% |
| 1111001101 | 2.205e-05 | 65.1% |
| 1111111011 | 0.0011 | 0.026% |

Таблица 2: Редкие последовательности для архитектуры x86_64

Всего было найдено 508 редких последовательностей длиной 8, 9 и 10 бит. Процент частоты нахождения в коде таких последо-

вательностей самый разный, что позволяет регулировать невидимость водяного знака. Среди найденных последовательностей нет ни одной, состоящей исключительно из единиц, что ещё раз подтверждает несостоятельность подхода, используемого в предыдущей работе [14].

## 6.2 Архитектура arm64

В этом подразделе представлены результаты для архитектуры arm64. Вычисленные статистики для последовательностей длины меньше 3-х представлены в таблице 3.

| Последовательность | Среднее | Стандартное отклонение |
|---|---|---|
| 0 | 0.6284 | 0.033 |
| 1 | 0.3716 | 0.033 |
| 00 | 0.4646 | 0.0446 |
| 01 | 0.1637 | 0.0126 |
| 10 | 0.1637 | 0.0126 |
| 11 | 0.2078 | 0.0226 |
| 000 | 0.3684 | 0.0505 |
| 001 | 0.0962 | 0.0074 |
| 010 | 0.092 | 0.0074 |
| 011 | 0.0718 | 0.0068 |
| 100 | 0.0962 | 0.0074 |
| 101 | 0.0676 | 0.0068 |
| 110 | 0.0718 | 0.0068 |
| 111 | 0.1361 | 0.017 |

Таблица 3: Доли последовательностей для архитектуры arm64

По сравнению с архитектурой x86_64 в коде arm64 содержится немного больше нулей. Подтверждается соотношение количества нулей к единицам примерно как к 3:2. Для последовательностей длины 3 в архитектуре arm64 имеет место меньшее стандартное отклонение чем для тех же последовательностей в архитектуре x86_64.

В таблице 4 приведены некоторые редкие последовательности с их средней долей в файле и процентом файлов, не содержащих данные последовательности.

| Последовательность | Среднее | Доля файлов |
|---|---|---|
| 00110011 | 0.0003 | 9.759% |
| 11001101 | 0.0028 | 0.026% |
| 010000110 | 0.0009 | 4.314% |
| 110110010 | 8.041e-05 | 36.954% |
| 1011011101 | 1.311e-05 | 68.079% |
| 1111001101 | 0.0024 | 0.103% |

Таблица 4: Редкие последовательности для архитектуры arm64

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

## 6.3 Архитектура mips64el

В этом подразделе представлены результаты для архитектуры arm64. Вычисленные статистики для последовательностей длины меньше 3-х представлены в таблице 5.

По сравнению с рассмотренными выше архитектурами в mips64el содержится значительно больше нулей. Нулей примерно 70%, а единиц 30%. Также для mips64el характерно небольшое стандартное отклонение для многих битовых последовательностей.

В таблице 6 приведены некоторые редкие последовательности с их средней долей в файле и процентом файлов, не содержащих данные последовательности.

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

| Последовательность | Среднее | Стандартное отклонение |
|---|---|---|
| 0 | 0.7 | 0.014 |
| 1 | 0.3 | 0.014 |
| 00 | 0.5587 | 0.0143 |
| 01 | 0.141 | 0.0031 |
| 10 | 0.141 | 0.0031 |
| 11 | 0.1952 | 0.0148 |
| 000 | 0.4605 | 0.0128 |
| 001 | 0.0981 | 0.0032 |
| 010 | 0.0778 | 0.0048 |
| 011 | 0.0632 | 0.0036 |
| 100 | 0.0981 | 0.0032 |
| 101 | 0.0429 | 0.0027 |
| 110 | 0.0632 | 0.0036 |
| 111 | 0.0959 | 0.0115 |

Таблица 5: Доли последовательностей для архитектуры mips64el

| Последовательность | Среднее | Доля файлов |
|---|---|---|
| 10011010 | 9.528e-05 | 19.516% |
| 10101101 | 2.532e-05 | 49.665% |
| 000011101 | 0.0003 | 0.772% |
| 011111101 | 0.0002 | 5.149% |
| 1010101101 | 8.503e-07 | 93.46% |
| 1100110010 | 0.0001 | 10.453% |

Таблица 6: Редкие последовательности для архитектуры mips64el

последовательностей в бинарных файлах, что позволяет управлять невидимостью водяного знака.

# Заключение

На данный момент изучена предметная область и стек технологий, сделан обзор, сформулирована цель, поставлены задачи, под-

готовлены и проанализированы все необходимые данные. За оставшийся семестр на основе сделанных вычислений необходимо сформулировать спецификацию алгоритма нанесения и извлечения водяного знака, а также сделать анализ метода.

# Список литературы


[1] Collberg, C. and Thomborson, C. 2002. " Watermarking, tamper-proofing, and obduscation-tools for software protection". IEEE Trans. Software Eng., pages 735–746.

[2] Davidson, R.I. and N. Myhrvold. 1996. Method and system for generating and auditing a signature for a computer program. Google Patents

[3] Wong, J.L., G. Qu, and M. Potkonjak. 2004. Optimization-intensive watermarking techniques for decision problems. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 23(1): 119–127.

[4] Qu, G. and M. Potkonjak. 2000. Hiding signatures in graph coloring solutions. In Information hiding. Springer.

[5] Myles, G. and C. Collberg. 2004. Software watermarking through register allocation: Implementation, analysis, and attacks. In Information security and cryptology-ICISC 2003, 274–293. Springer.

[6] Zhu, W. and C. Thomborson. 2006. Algorithms to watermark software through register allocation. In Digital rights management. Technologies, issues, challenges and systems, 180–191. Springer.

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

[8] Kim, H.J., and Y.H. Choi. 2003. A novel echo-hiding scheme with backward and forward kernels. IEEE Transactions on Circuits and Systems for Video Technology 13(8): 885–889.

[9] Yanqun Zhang. 2009. Digital Watermarking Technology: A Review. 2009 ETP International Conference on Future Computer and Communication.

[10] Mohammad Ali Nematollahi, Chalee Vorakulpipat, Hamurabi Gamboa Rosales. 2017. Digital Watermarking. Techniques and Trends. Springer.

[11] Lim, H.-I. 2015. A performance comparison on characteristics of static and dynamic software watermarking methods. Indian Journal of Science and Technology 8(21).

[12] Sarah H. Mnkash, Matheel E. Abdulmunem. 2020. A Review of Software Watermarking. Iraqi Journal of Science, 2020, Vol. 61, No. 10, pp: 2740-2750

[13] William Zhu, Clark Thomborson , Fei-Yue Wang. 2005. A Survey of Software Watermarking. ISI 2005: Intelligence and Security Informatics pp 454-458

[14] Смирнов Д.П. 2018. Нанесение водяных знаков на программное обеспечение. Курсовая работа кафедры системного программирования СПБГУ

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