Федеральное государственное автономное образовательное учреждение
высшего образования
«Омский государственный технический университет»

На правах рукописи

Мунько Сергей Николаевич

СТЕГАНОГРАФИЧЕСКОЕ ВСТРАИВАНИЕ ИНФОРМАЦИИ В ПАМЯТЬ ИСПОЛНЯЕМОГО КОДА И КОД ВЕБ-СТРАНИЦЫ

Специальность 2.3.6. – Методы и системы защиты информации, информационная безопасность

Диссертация
на соискание учёной степени
кандидата технических наук

Научный руководитель:
доктор физико-математических наук, профессор
Белим Сергей Викторович

Омск 2024

Оглавление

Введение

Актуальность темы исследования

Методы скрытого встраивания данных в легальные сообщения (стеганографические методы) применяются для решения двух задач: скрытая передача информации и внедрение цифровых водяных знаков. Стеганографические методы передачи данных применяются при необходимости сокрытия самого факта наличия сообщения. Цифровые водяные знаки применяются для решения задач подтверждения авторства цифрового контента. В обоих случаях используется некоторое легальное открытое сообщение, называемое стеганографическим контейнером (стегоконтейнером). Встраивание скрытого сообщения (стеганографическая вставка, стеговставка) осуществляется с помощью модификации данных стегоконтейнера [15,16,19]. Модификация стегоконтейнера должна удовлетворять двум условиям. Во-первых, она не должна быть легко обнаружима и не должна оказывать заметного влияние на штатное использование стегоконтейнера. Во-вторых, должен существовать алгоритм извлечения скрытых данных без потерь.

Стеганографические методы получили широкое распространение с развитием открытых сетей. Большинство современных методов сокрытия информации разработано для мультимедийных данных. Мультимедийный контент активно передается в сети Интернет и имеет большой объем файлов, что позволяет использовать его в качестве стегоконтейнера. В последнее десятилетие также получили развитие методы сетевой стеганографии, использующие в качестве стеганографических контейнеров сетевые протоколы [18,26,27,32,34]. Однако вместе с методами стеганографического встраивания активно развиваются и методы стеганографического анализа, позволяющие обнаруживать скрытые данные. В обоих этих случаях

используется возможность нарушать структуру стегонаграфического контейнера в небольшом объеме, так как это не приводит к заметным эффектам.

Большой вклад в развитие методов встраивания ЦВЗ внесли: E. Koch, J. Zhao, D. Bcnham, N. Memon, B.-L. Yeo, M. Yeung, C. Podilchuk, W. Zeng, C.-T. Hsu, J.-L. Wu, B. Tao, B. Dickinson, I. Cox, J. Kilian, T. Leighton, T. Shamoon, M. Barni, F. Barlolini, V. Cappcllini, A. Piva, J. Fridrich, В.Г. Грибунин, В.А. Митекин, Н. Мемон, И.Н. Оков, Б.Я. Рябко, И.В. Туринцев, А.Н. Фионов, и др.

Встраивание скрытых сообщений в жестко структурированные объекты имеет большие сложности, так как может нарушить использование стеганографического контейнера по его прямому назначению. К контейнерам с жесткой структурой относятся исполняемые программы [6,13,22,25]. Даже небольшие изменения программного кода или значения переменных и констант может приводить к невозможности работы программы. При этом стеганографическое встраивание данных в программы может решить задачи соблюдения авторских прав. Для этого необходимо встроить цифровой водяной знак. Исполняемый код в браузере является неотъемлемой частью web-страниц и имеет не менее широкое распространение, чем мультимедийный контент. Это делает исполняемый код перспективным объектом для использования в качестве стегоконтейнера. На сегодняшний день встраивание в память исполняемого кода осуществляется с помощью самой программы на этапе ее запуска, что делает такую схему не устойчивой к динамическому анализу кода. Для стеганографии программ с открытым кодом используются методы, разработанные для текстовых файлов и имеющие низкий объем встроенной информации [29,42,43].

В связи с чем актуальной является задача разработки методов скрытого встраивания данных в исполняемые программы, учитывающие их специфику и обладающие большой емкостью.

Цель диссертации является разработка методов скрытого встраивания данных в исполняемые программы и программы с открытым кодом.

Для достижения поставленной цели были решены следующие **задачи**:

1) Разработка метода встраивания и извлечения скрытой информации в память исполняемой программы с временными ограничениями существования в динамической памяти.

2) Разработка метода встраивания и извлечения скрытой информации в исходный код web-страницы на основе модификации дерева тэгов.

3) Разработка метода встраивания и извлечения скрытой информации в классы, определяющие свойства SVG-изображения.

4) Реализация и тестирование программных комплексов на основе разработанных методов.

Объектом исследования являются программы с исполняемым двоичным кодом и веб-страницы с открытым кодом, рассматриваемые как стеганографические контейнеры для скрытой передачи информации.

Предметом исследования являются методы встраивания и извлечения скрытой информации в программы с исполняемым двоичным кодом и веб-страницы с открытым кодом.

Методы исследования: В диссертационном исследовании использованы методы теории графов, алгоритмы обхода деревьев, криптографические методы защиты информации, методы статического и динамического анализа кода программ.

Основные положения, выносимые на защиту:

1) Метод встраивания и извлечения меток безопасности в память исполняемой программы. Особенностью метода является временные ограничения на присутствие в памяти исполняемого кода, а также

дополнительная защита момента времени формирования метки безопасности в памяти программы.

2) Метод встраивания и извлечения скрытой информации в исходный код веб-страницы. Особенностью метода является модификация иерархии тэгов и маскировка изменений под легитимные исполняемые классы.

3) Метод встраивания и извлечения скрытой информации в классы, определяющие свойства SVG-изображения. Особенностью метода является встраивание скрытой информации не в само изображение, а в его атрибуты.

4) Программные комплексы на основе разработанных методов.

Научная новизна:

1) Предложен метод встраивания метки безопасности в память исполняемой программы с ограниченным временем существования. Новизна состоит в ограничении времени присутствия метки безопасности в динамической памяти программы. Момент появления метки безопасности определяется ключевой информацией. Время присутствия метки в памяти также ограничено. Временные ограничения существенно повышают сложность стегоанализа кода со стороны злоумышленника.

2) Предложен метод встраивания данных в открытый код web-страницы на основе расширения дерева тэгов. Новизна состоит в том, что встраивание не использует атрибуты уже существующих тэгов, а модифицирует дерево тэгов web-страницы так, чтобы не оказывать влияния на отображение страницы в браузере. Предложен метод сокрытия факта такой модификации, усложняющий стегоанализ web-страницы.

3) Предложен метод встраивания скрытых данных в SVG-изображение, размещенное на web-странице. Новизна состоит в том, что осуществляется модификация не данных векторной графики, атрибутов классов, определяющих свойства изображения.

Практическая и научная значимость результатов

Научная и практическая значимость результатов состоит в разработке новых алгоритмов, использующих исполняемый и открытый код программ для встраивания скрытой информации. Разработаны программные комплексы, реализующие предложенные алгоритмы. Данные программные комплексы могут быть использованы как для скрытой передачи данных, так и для встраивания цифровых водяных знаков. Результаты диссертационной работы использованы в деятельности ООО «СМАРТФОРС» и ООО «Лет ИТ БИ».

Апробация работы

Основные результаты диссертации докладывались и обсуждались на следующих научных конференциях: «Проблемы машиноведения» (Омск, 2021), «Прикладная математика и фундаментальная информатика» (Омск, 2021, 2022, 2023), «Нанотехнологии. Информация. Радиотехника» (Омск, 2021). «Актуальные проблемы информатики, радиотехник и связи» (Самара, 2024), «Проблемы информационной безопасности социально-экономических систем» (Гурзуф, 2024). Зарегистрировано три программы для ЭВМ.

Степень достоверности результатов работы

Все полученные результаты обоснованы адекватностью применяемых методов и подтверждаются реализацией и тестированием программных комплексов.

Публикации

Материалы диссертации опубликованы в 9 изданиях, из них 3 статьи в журналах из списка, рекомендованного ВАК, 2 статьи, индексируемых в международной базе Scopus и 3 свидетельства о регистрации программ.

Структура и объем диссертации

Диссертация содержит: введение, 3 главы, заключение и библиографический список. Общий объем диссертации 113 страниц, библиографический список содержит 121 источник.

Личный вклад автора

Все публикации выполнены в соавторстве с научным руководителем. Автор диссертации принимал участие во всех этапах подготовки публикаций: постановке задачи, обработке результатов компьютерного эксперимента и обсуждении результатов. Программное обеспечение реализовывалось автором лично.

Соответствие паспорту специальности

Результаты, полученные в диссертации, соответствуют следующим пунктам паспорта специальности 2.3.6. Методы и системы защиты информации, информационная безопасность:

5. Методы, модели и средства (комплексы средств) противодействия угрозам нарушения информационной безопасности в открытых компьютерных сетях, включая Интернет.

15. Принципы и решения (технические, математические, организационные и др.) по созданию новых и совершенствованию существующих средств защиты информации и обеспечения информационной безопасности.

Глава 1. Стеганографические методы встраивания информации в исполняемые программы

1.1 Общая постановка задачи стеганографического встраивания информации

Стеганография – это способ передачи или хранения информации с учётом сокрытия самого факта наличия такой информации. В этом состоит основное отличие стеганографии от криптографии, которая скрывает содержимое сообщения с помощью математических преобразований. В большинстве случаев стеганография используется как дополнительный инструмент защиты информации совместно с криптографией. Скрываемые данные принято называть стегосообщением или стеговставкой, а данные, внутри которых находится стегосообщение, принято называть стеганографическим контейнером (стегоконтейнером, контейнером).

Существующие методы стеганографии можно разделить на три класса: классические, компьютерные и цифровые.

К методам классической стеганографии относят использование симпатических (невидимых) чернил, применение микрофотоснимков и др. Компьютерная стеганография основана на использовании в качестве контейнеров служебной компьютерной информации, например, пакеты сетевых протоколов. В цифровой стеганографии роль стегоконтейнеров играют файлы различного формата. Современные методы встраивания информации позволяют внедрять скрытую информацию в файлы различного формата: аудио, видео, текстовые, двоичные коды программ и т.д. Применение цифровой стеганографии позволяет решать различные задачи обеспечения информационной безопасности: защита авторского права, подтверждение подлинности документов, формирование уникальных

идентификаторов в системах электронного документооборота и др. Для характеристики методов стеганографии применяют два основных параметра - объем встроенного сообщения (емкость стегоконтейнера) и устойчивость к стеганографическому анализу (обнаружение факта встраивания данных).

Методы стеганографии ориентированы на конкретные типы стегоконтейнеров, так как используют структуры данных, применяемые для информации конкретного вида. Методы, разработанные для одного вида контейнера, как правило, не применимы для другого типа контейнера. В связи с этим методы стеганографии классифицируют по типу используемого контейнера: аудиоконтейнеры, графические контейнеры, текстовые контейнеры, программные контейнеры (исполняемые файлы), видеоконтейнеры, сетевые контейнеры.

Общая постановка задачи построения стеганографической системы традиционно описывается в терминах криптографических протоколов. Два абонента A и B могут обмениваться информацией по некоторому каналу связи. При этом третья сторона V может полностью читать все сообщения между A и B. Причем абоненты A и B знают, что их переписка полностью читается. Требуется передать скрытое сообщение от A к B так, чтобы V о нем не знал. То есть необходимо организовать скрытый канал между A и B, используя открытый канал передачи сообщений. Причем скрывается сам факт наличия такого канала передачи сообщений. Для решения этой задачи абонент A формирует легальное сообщение X и выполняет его преобразование $X'=F(X,m,k_1)$. X' – это новое сообщение, которое должно выглядеть как легальное и не вызывать подозрений у V. k_1 – это ключ встраивания, известный A и B и не известный V. F() – это алгоритм встраивания. m – секретное сообщение. Абонент B, получив сообщение X' применяет к нему некоторое преобразование $m=G(X',k_2)$ и получает скрытое сообщение m. k_2 – это ключ извлечения сообщения. В большинстве случаев ключ встраивания и ключ извлечения совпадают $k_1=k_2$, то есть схемы

встраивания являются симметричными. Однако теоретически допускается существование ассиметричных схем с различными ключами $k_1 \neq k_2$. G() – это алгоритм извлечения сообщения. Алгоритмы F() и G() считаются открытыми. Стойкость схемы должна обеспечиваться только знанием ключей встраивания и извлечения. Причем алгоритмы F() и G() должны быть такими, что абонент V, зная их не мог по информации, хранящейся в X’ обнаружить скрытый канал. Однако такое требование часто является избыточным и в реальных системах достаточно, чтобы задача обнаружения скрытого канала обладала высокой трудоемкостью. В этой схеме сообщение X является стегоконтейнером, m – стеговставкой.

При использовании стеганографического подхода используется понятие стеганографической схемы. Стеганографическая схема включает в себя не только алгоритм модификации контейнера с помощью некоторого алгоритма, но алгоритм распределения стеганографических вставок по нескольким контейнерам. Стеганографические алгоритмы встраивания данных в контейнеры всегда тесно взаимосвязаны и зависимы от структуры и вида контейнера. Для алгоритмов распределения частей стеговставки по нескольким контейнерам существуют несколько стандартных подходов [20,21,23,39,44]. Эти подходы ориентированы на решение трех задач:

1) Отслеживание копий файлов.

2) Проверка подлинности и целостности данных.

3) Скрытая передача данных.

Первый из таких подходов основан на цифровых отпечатках (*Digital Fingerprint*). Эта схема применяется тогда, когда существует несколько копий одного контейнера и их создатель хочет отследить распространение каждой копии. В этом случае стеганографическое встраивание используется для добавления в каждую копию контейнера уникального идентификатора или цифрового отпечатка. Злоумышленник для незаконного распространения копии контейнера может решать две цели:

1) удаление цифрового отпечатка из контейнера («слабая цель»);

2) искажение цифрового отпечатка («сильная цель»).

В первом случае злоумышленник может распространять незарегистрированную копию, что не позволяет отследить источник незаконного распространения копии. Во втором случае злоумышленник замаскировывает свои действия, создавая ложный след происхождения копии контейнера.

Цифровые отпечатки могут применяться для отслеживания копий видеоконтента и аудиоконтента, а также платного программного обеспечения. Кроме этого цифровые отпечатки применяются для отслеживания распространения профилей пользователей. По результатам исследования Принстонского университета [84] не менее 60% из тысячи самых популярных *web*-сайтов передают информацию третьим сторонам, которые занимаются созданием сетевых профилей или отпечатков посетителей сайтов и продают их рекламодателям и агрегаторам данных.

Второй подход основан на формировании цифровых водяных знаков (*Digital Watermarking*). В отличие от цифровых отпечатков, цифровые водяные знаки представляют собой одинаковые встроенные данные во все копии контейнера [29,30,31]. Цифровой водяной знак встраивается в контейнер для подтверждения его подлинности, подтверждения авторства документа или подтверждения целостности документа. В первых двух случаях применяются устойчивые цифровые водяные знаки. Основным требованием к устойчивым цифровым знакам является возможность их извлечения после модификаций контейнера. Например, устойчивые цифровые водяные знаки в видеофайле должны сохраняться при перекодировке формата видеопотока. Для контроля целостности файла используются «хрупкие» цифровые водяные знаки. Данный цифровой водяной знак должен разрушаться практически полностью при минимальных модификациях контейнера.

Третий вид задач, решаемых стеганографическими методами, связан со скрытой передачей данных. Эта задача существенно отличается от цифровых отпечатков и цифровых водяных знаков. Первые две задачи ориентированы на защиту контейнера и его содержимого. Сам факт существования цифровых водяных знаков и цифровых отпечатков может быть открытым. Открытость факта встраивания в некоторый момент времени является частью стеганографической схемы. Для доказательства действий злоумышленника необходимо обнародовать информацию о наличии стеганографической вставки. При скрытой передачи сообщений появление информации о присутствии стеганографической вставки дискредитирует всю схему. Для выявления факта скрытого встраивания применяется стеганографический анализ или стегоанализ [33,36,116]. Задачами стегоанализа является обнаружение самого факта встраивания, а также определение максимально возможной информации о параметрах встраивания. Большинство стеганографических алгоритмов используют ключевую информацию для встраивания сообщения. Для легального обнаружения и извлечения сообщения необходимо знание ключа. Если ключ встраивания неизвестен, то задача обнаружения стеговставки должна быть вычислительно трудной. Методы стегоанализа выявляют уязвимости алгоритмов встраивания, позволяющие обнаруживать и извлекать встроенные сообщения без знания ключа встраивания.

Алгоритм встраивания существенно зависит от используемого контейнера. Наибольшее распространение получили алгоритмы встраивания в мультимедийные данные [1,2,4,5,37,38]. Это связано с тем, что растровые изображения, аудиофайлы и видеофайлы содержат большие объемы малозначимой информации. Наибольшее развитие получил метод замены наименее значимого бита [70,72,75,80]. Этот метод основан на нечувствительности человеческого глаза к малым изменениям оттенков синего цвета. Поэтому подмена младших битов для пикселей в синем канале

изображения на скрытые данные не может быть обнаружена визуально. Подобные изменения воспринимаются как шум очень низкой интенсивности. Существует пороговое соотношение сигнал/шум, ниже которого стеганографические вставки не обнаружимы [47,50]. Для выявления таких изменений требуется разработка отдельных методов, основанных на алгоритмах анализа данных [12,119]. Причем эти методы не дают гарантированного обнаружения стеганографической вставки, а позволяют обнаружить, с некоторой вероятностью, ее существование. Ключевой информацией для такого метода служат координаты пикселей, в которых выполняется подмена младших битов. Если модифицируемые пиксели образуют локальную область, то ее положение и размеры могут быть определены с некоторой точностью на основе сравнительного анализа нескольких слоев [91,110]. Если модифицируемые пиксели распределены по всему изображению, то задача их определения без дополнительных данных может быть решена полным перебором. Наличие стеганографической вставки может быть обнаружено также частотными методами анализа [120]. Встраивание чужеродной информации изменяет частотное распределение байтов в файле изображения.

Распределение встраиваемых данных по всему изображению также может быть выполнено с помощью стеганографических методов, работающих в частотной области изображения. Для этого выполняется дискретное преобразование Фурье или дискретное косинусное преобразование изображения. Скрытое встраивание информации выполняется с помощью изменения коэффициентов разложения [69,73,112]. Стегоконтейнер со встроенным сообщением получается при обратном преобразовании из частотной области в координатную. В итоге изменения происходят во всех пикселях изображения одновременно. Обнаружение такой вставки может быть выполнено, если известен алгоритм встраивания и

известен сам факт встраивания [58]. В противном случае стегоанализ может быть выполнен только прямым перебором.

1.2 Встраивание информации в исполняемый код

Встраивание скрытых данных в код программы является одной из наиболее сложных задач стеганографии. Это связано со свойствами файлов программ. Во-первых, файлы программ содержат мало незначительной информации, которая может подменяться скрытыми данными. Во-вторых, файлы программ имеют жесткую структуру, нарушения которой может приводить к невозможности выполнения программного кода. Под исполняемыми программами можно понимать исполняемый двоичный код или программу с открытым кодом на языке высокого уровня. Эти два случая требуют различного подхода к встраиванию скрытых данных. Однако обе задачи относятся к одной области стеганографии, так как двоичный код может быть дисассемблирован и представлен в виде программы на языке ассемблера.

Алгоритмы скрытого встраивания, использующие в качестве контейнера программы с открытым кодом, как правило, основаны на методах текстовой стеганографии. Код программы рассматривается как частный случай текстового контейнера. Встраивание в этом случае происходит, в основном, в текстовые константы, используемые программой.

Стеганографические алгоритмы для исполняемого программного кода не являются универсальными. Встраивание данных изменяет код программы, при этом оно не должно вносить критические изменения, мешающие штатной работе программы. Каждый тип исполняемого файла имеет свою структуру. Поэтому для каждого типа исполняемого файла разрабатывается свой стеганографический алгоритм.

Один из методов скрытого встраивания данных в программный код основан на возвратно-ориентированном программировании [84]. Изначально возвратно-ориентированное программирование (*ROP*) разрабатывалось как метод взлома компьютерной безопасности, который позволяет злоумышленнику выполнять код при наличии средств защиты, таких как защита исполняемого пространства и подпись кода. В этом методе злоумышленник получает контроль над стеком вызовов, чтобы перехватить поток управления программой. После этого злоумышленник выполняет тщательно выбранные последовательности машинных инструкций, которые уже присутствуют в памяти машины. Такие последовательности инструкций получили название «гаджеты». Каждый гаджет обычно заканчивается инструкцией возврата и находится в подпрограмме или коде общей библиотеки. Связанные вместе, эти гаджеты позволяют злоумышленнику выполнять произвольные операции в обход средств защиты, препятствующих более простым атакам. Для использования гаджетов при встраивании сообщения используется обфускация программного кода. К исполняемой программе добавляется набор команд, не видимых при статическом анализе кода. Данные команды формируются при выполнении программы и могут быть выявлены только при динамическом анализе кода. Исходный исполняемый код дизассемблируется полностью либо фрагментарно, после чего в него добавляются дополнительные команды [59, 98, 99,117]. Метод получил название RopSteg. *RopSteg* принимает в качестве входных данных набор двоичных команд программы *P* и некоторую последовательность команд *I*. RopSteg модифицирует исходный код так, чтобы встроенные команды *I* были скрыты от статического анализа программы, но видны при динамическом анализе программы. Набор команд I является стеговставкой. Компьютерный эксперимент показал, что данный метод увеличивает объем программы от 3% до 5%. Время выполнения программы увеличивается от 20 мс до 40 мс.

Переносимый исполняемый файл для операционных систем Windows (*Portable Executable*, *PE*) имеет множество свойств, позволяющих скрыто встраивать данные. К таким свойствам можно отнести неопределённость размера файла, сложность файловой структуры и единый формат файла. Разработано несколько алгоритмов стеганографического встраивания данных в PE-файлы. Эти алгоритмы используют различные свойства данного формата исполняемых файлов. Существует три основных метода скрытого встраивания данных в PE-файлы:

1) метод сокрытия информации, основанный на перенаправлении вызовов в *PE*-файле [17, 114];

2) метод сокрытия информации на основе модификации данных PE-файла [105, 114];

3) метод сокрытия информации, основанный на импорте PE-файла [45,104].

Существующие алгоритмы скрытия файлов *PE* имеют следующие недостатки:

1) Избыточное пространство файлов PE-файла является открытым и поддается анализу на основе известной структуры файла. На рынке программного обеспечения присутствуют мощные инструменты анализа PE-файлов *PE*, такие как *Stud_PE* и *PE Explorer Lord PE*. Анализ кода PE-файлов позволяет выявлять избыточное пространство, в результате чего наличие вставки может быть обнаружено.

2) Скрытые данные сконцентрированы в одной области, что повышает риск их обнаружения.

3) Структура *PE*-файла стандартизирована, что снижает объем информации, которую необходимо проанализировать для обнаружения стеговставки.

4) Отсутствует возможность противостоять удалению и модификации встроенных данных.

В качестве примера можно привести стеганографический алгоритм, использующий перенос функций внутри файла *PE* [121]. Предложенный алгоритм дизассемблирует раздел кода PE-файла, выделяет участки кода системных и пользовательских функций и переносит их в последний раздел файла. Встраиваемые данные размещаются в освободившемся кодовом пространстве. Стеговставка формируется таким образом, чтобы быть связанной с основными функциями *PE*-файла. Такой подход маскирует встроенную информацию и не позволяет выявлять ее с помощью статического анализа кода.

Стеганографические алгоритмы могут быть ориентированы не только на вычислительные системы общего назначения, но и исполняемые коды для программируемых логических интегральных схем (ПЛИС) [62]. В этом случае встраивается цифровой водяной знак для защиты программного обеспечения от изменений. Цифровой водяной знак содержит данные мониторинга и обеспечивают целостность данных программы. Встраивание выполняется с помощью эквивалентного преобразования программы, которое не изменяет его работы. Вследствие этого обнаружение цифрового водяного знака становится невозможным. Проверка целостности выполняется с помощью повторного мониторинга и сравнения результатов с извлеченным цифровым водяным знаком. Для дополнительной защиты встроенных данных выполняется обфускация программы. Причем обфускация и встраивание данных выполняются совместно с использованием одного ключа встраивания. Этот подход скрывает данные мониторинга и не раскрывает факт проведения контроля целостности в отношении программный кода. Обфускация программного кода, который является частью метода, усложняет стегоанализ [3] и делает трудоемким процесс принятия решения о наличии или отсутствии дополнительных встроенных данных. Кроме того, обфускация существенно усложняет извлечение цифрового водяного знака из программного кода при неизвестном ключе

встраивания. Эффективность предлагаемого метода выражается в уменьшении вероятности обнаружения встроенных данных с помощью стегоанализа. Эксперимент показал снижение этой вероятности на 10,32% по сравнению с известными подходами.

Другой подход к встраиванию цифрового водяного знака в исполняемый код ПЛИС основан на модификации информационных объектов [78]. Цифровой водяной знак, встроенный в программный код, содержит контрольный хэш. Контроль целостности обеспечивается при условии, что извлечение цифровых водяных знаков и восстановление исходного состояния кода происходит одновременно. Основная цель метода состоит в увеличении объема встроенного цифрового водяного знака. Такое увеличение объема дает возможность использовать более широкий набор хэш-функций для мониторинга целостности программы. В частности, есть возможность использовать хэш-функции, обладающие большей криптографической стойкостью. Увеличение эффективного объема достигается за счет предварительной подготовки программного кода ПЛИС, выполняемой перед встраиванием цифрового водяного знака. В ходе этой подготовки биты информационного объекта переводятся в заранее определенное состояние. Состояние битов определяется ключом встраивания. Приведение целевых битов в предопределенное состояние выполняется с помощью эквивалентных преобразований, аналогичных тем, которые используется для встраивания цифровых водяных знаков. Цифровой водяной знак в рамках предложенного метода содержит только контрольную хэш-сумму. В нем нет информации для восстановления исходного состояния программы. Это отсутствие обусловлено тем, что исходное состояние восстанавливается по правилам, описанным в стенографическом ключе. Стенографический ключ содержит правила для приведение целевых битов в заданное состояние. Метод отличается от аналогичных тем, что предоставляет больший объем места для встраивания хэш суммы.

В работе [24] предлагаются подходы к внедрению ЦВЗ в аппаратные контейнеры, построенные на основе LUT-ориентированной архитектуры (LUT-контейнеры). К таким контейнерам относятся, например, микросхемы FPGA (Field Programmable Gate Array), являющиеся широко используемой элементной базой для построения компьютерных и управляющих систем. По многим параметрам FPGA конкурируют с микропроцессорами и микроконтроллерами, а по параметрам производительности и возможности организации параллельных вычислений превосходят их. Пусть необходимо встроить цифровой водяной знак в LUT-контейнер, реализующий некоторую вычислительную или управляющую функцию. Для этого осуществляется обход множества блоков LUT-контейнера в заданном порядке и встраивание отдельных частей скрытых данных. В результате получается измененный LUT-контейнер, который выполняет исходные целевые функции, но при этом хранит цифровой водяной знак. Скрытое встраивание последовательности битов возможно, только если схема, содержащаяся в контейнере, имеет больше одного уровня. Так же встроить информацию можно не во все блоки контейнера, а только в те, которые не подключены к выходам схемы, так как отсутствует следующий за ним блок.

В отличие от традиционных мультимедийных стегоконтейнеров, LUT-контейнеры являются активными и влияют друг на друга. Кроме этого, данные, хранящиеся в контейнере, имеют точные значения, не допускающие малых изменений. Взаимное влияние элементарных единиц контейнера друг на друга дает возможность производить локальные изменения их содержимого, не меняя при этом глобальной функциональности контейнера. Точное представление данных порождает подходы к противодействию активным стегоатакам типа «стирание секретной информации». Такая защита недоступна для традиционных мультимедийных контейнеров.

Структура данных в LUT-контейнерах не имеет существенной статистической связи между битами как одного блока, так и соседних блоков. Это не позволяет произвести пассивный статистический стегоанализ такого контейнера.

Мониторинг целостности программного кода широко используется для защиты изменений в исполняемых программах. Стеганографическое встраивание кода целостности совместно с организационными мерами позволяет реализовать политику ограничения запуска программ [64,108]. Встраивание цифровых водяных знаков позволяет реализовать оперативный контроль целостности исполняемых модулей [17,35,45]. Мониторинг целостности, как правило используется совместно с другими методами защиты данных: криптографическими, организационными и техническими.

1.3 Скрытое встраивание данных в HTML-код

При создании скрытого канала связи на основе стеганографических методов отправитель прячет секретное сообщение в стегоконтейнер. В качестве такого контейнера может использоваться web-страница. В этом случае для создания скрытых каналов связи могут быть задействованы различные приложения и платформы в сети Интернет. Методы стеганографии, использующие web-страницу в качестве стегоконтейнера, должны опираться на один из языков разметки текста: HTML, XHTML, XML, SMIL и т.д. При этом, изменения в коде web-страницы должны быть такими, чтобы они не влияли на ее отображение в браузере [48, 55,60,66,77,82,85,87,88,90,96,97,100,106,113]. Следует отметить, что язык разметки гипертекста HTML менее чувствителен к изменению синтаксиса при отображении в web-браузере, чем другие языки web-программирования, такие как Java или Python. Таким образом, отправитель может скрыть

секретное сообщение в файле web-страницы, изменяя его исходные коды так, чтобы эти изменения не влияли на отображение web-страницы в браузере.

Обычный *HTML*-файл *web*-страницы состоит из нескольких элементов, каждый из которых элемент начинается с некоторого тэга <*tagname*> и заканчивается соответствующим ему тэгом </*tagname*>. В качестве примера таких тэгов можно привести <*body*>, <*table*>, <*form*> и другие. Каждый тэг может иметь несколько атрибутов, принимающих определенные значения. Атрибуты предоставляют дополнительную информацию для правильного отображения элемента в браузере. Чаще всего используют два метода стеганографии *web*-страниц для встраивания скрытого сообщения.

Первый метод заключается в добавлении нескольких специальных символов в код web-страницы, таких как, пробелы и символы табуляции [66,77,85,97]. Такие специальные символы не визуализируются, но при этом они анализируются web-браузером. В исследованиях [60,88] предложен метод использования различных символов, таких как &#x20 или &#32, которые отображаются в web-браузере как пустые места. В дополнение к этому методу в работах [55,77,113] применены криптографические методы для улучшения необнаруживаемости встроенного сообщения. В работе [82] предложен метод, который вставляет элементы, содержащие только теги без содержимого (атрибуты и характеристики). Например, тексты между и выделены полужирным шрифтом, но если нет текста между тегами, в web-браузере не будет никаких изменений, даже если эти элементы добавлены.

В работе [68] при скрытии информации используются определенные символы из определенных слов. Например, первый символ первого слова каждого абзаца. Эта схема позволяет скрывать один символ секретного сообщения. Объединение этих символов дает полное сообщение. Кроме этого, применяется подход текстовой стеганографии, использующий пунктуацию текста. Идея этого подхода заключается в использовании знаков

препинания, таких как запятая, точка с запятой, кавычки для кодирования секретного сообщения. В исследованиях [46,51,76,86] предложен стеганографический алгоритм, использующий метода сдвига строки. В этом методе строки текста смещаются на некоторое расстояние, например, на 1/300 дюйма вверх или вниз. Затем информация скрывается путем создания скрытой уникальной формы текста. Авторы работ [76,102] предложили использовать метод переноса слов. В этом методе для скрытия информации нужно сдвинуть слова по горизонтали или изменить расстояние между словами. В исследовании [103] рассматриваются синонимы определенных слов, чтобы скрыть сообщение в тексте. В этом методе выбираются определенные слова из текста и определяются их синонимы. Для сокрытия сообщения используются слова вместе с их синонимами. В статье [92] предложен метод скрытия информации путем добавления дополнительных пробелов в текст. Эти пробелы могут быть размещены в конце каждой строки, в конце каждого абзаца или между словами. В исследованиях [52, 71,101] предлагаются методы для текста на арабском языке, персидском языке и языке урду. Одной из особенностей этих языков является обилие точек в его буквах. Буквы с одной точкой могут использоваться, чтобы скрыть информацию. Модифицируемыми параметрами для встраивания являются смещение положения точки немного выше по вертикали относительно стандартной точки. Такие техники основаны на том, что каждый язык имеет свои особенности. Слова каждого языка, за исключением иероглифической формы записи, состоят из сочетания одной или нескольких гласных и согласных букв. Эти гласные и согласные буквы и их комбинация служит основой применения этой же техники для текстов на языке хинди [71]. В работе [71] предложен метод текстовой стеганографии, который скрывает секретное сообщение в английский текст с использованием различных вариантов написания слов. В английском языке некоторые слова имеют разные значения. Это различие в написании лежит в основе

стеганографии. В исследовании [74] предложен метод текстовой стеганографии на основе смайликов.

Второй метод состоит в изменении порядка атрибутов в *HTML*-файле. Этот метод использует тот факт, что порядок атрибутов в одном и том же теге не влияет на его отображение.

В работе [93] предложен подход к текстовой стеганографии, который использует *HTML*-тэги и их атрибуты для скрытого встраивания сообщения. Основная идея предлагаемой методики заключается в сокрытии сообщений с помощью изменение порядка атрибутов. Порядок атрибутов внутри тэга не влияет на отображение *HTML*-документа в браузере. Предлагаемый метод состоит из трех этапов: генерация ключевого файла, процесс встраивания и процесса извлечения. Основой данного метода является генерация ключевого файла. Ключевой файл представляет собой набор комбинаций символов, хранящихся в виде строк и столбцов. Эти комбинации генерируются путем сканирования *HTML*-документа. При сканировании анализируются комбинации атрибутов тэгов *HTML*-файла. На основе порядка следования атрибутов в исходном файле формируется ключевой файл.

Автор работы [61] предлагает простую межъязыковую синтаксическую модель *matchertext*, которая позволяет осуществлять встраивание строки на любом совместимом языке в строку на любом другом языке с помощью простого «копирования и вставки». Этот межъязыковой подход применим для стекганографического встраивания в файлы, использующие синтаксис *URI*, *HTML* и *JavaScript*. Необходимость встраивать допустимые строки на одном языке в допустимые строки на другом языке является обычным явлением в практике программирования. В качестве примеров можно привести встраивание регулярных выражений *URI*, запросов *SQL* или *HTML* [95,118]. Языки программирования используют разметку внутри строковых констант для передачи данных и значений переменных, введенных пользователем в *web*-форму, исполняемому коду. Этот же подход

применяется для передачи значений из кода *JavaScrip*t в запросы *SQL*. Столь же распространенной проблемой, возникающей при встраивании строки одного языка в строку другого языка, является необходимость преобразования встроенной строки в определенные символы, чувствительных к «основному» языку и синтаксическому контексту. При этом процессоры основного языка должны правильно интерпретировать встроенный текст. Например, регулярное выражение "[^"]*" соответствует строкам в двойных кавычках, но для встраивания в C-подобный язык, оно должно быть записано как *re.match*("\"[^\"]*\""). Все встроенные экземпляры символов двойных кавычек должны преобразовываться так, чтобы встроенные кавычки не заканчивали строковую константу преждевременно. В работе [107] применяется синтаксис *MinML* для правильной передачи встроенной строки. *MinML* – это краткий, но универсальный альтернативный синтаксис для языков разметки, производных от *SGML*, такие как *HTML* и *XML*. *MinML* использует символы сопоставления для выделения структуры вместо сопоставления пары тэгов, означающих начало и конец структуры. Например, вместо <*em*>выделение</*em*> записывается *em*[*emphasis*], и вместо «Цитата» записывается «[Цитата]. *MinML* поддерживает безэкранное встраивание *matchertext* в другие языки с помощью последовательности, подобной *[*matchertext*], а также с помощью escape-последовательностей. Экспериментальное расширение генератора статических *web*-сайтов *Hugo* позволяет создавать *web*-сайты с помощью языка *MinML*.

В статье [83] предлагается схема встраивания цифровых водяных знаков на основе создания таблицы, кодирующей содержание *web*-страницы. Схема работы алгоритма состоит в том, что в исходной *HTML*-странице находятся все английские прописные и строчные буквы, а также арабские цифры. Далее выполняется преобразование символов в эквивалентные им цифры '*A*' и '*a*' => 0 , '*C*' и '*c*' => 2 и так далее, цифры '0-9' преобразуются в «26-35». Все символы сортируются по их номерам в соответствии с расположением в

кодовой таблице *ASCII*. Далее формируются матрицы, элементы которых зависят от номера строки и столбца исходного кода *web*-страницы. Цифровые водяные знаки встраиваются отдельно в каждый столбец и каждую строку исходного кода *HTML*-страницы. Извлечение происходит путем сравнения матриц *HTML*-страницы со встроенным сообщением и матриц исходной *HTML*-страницы.

Существует несколько инструментов для стеганографического встраивания скрытых сообщений в web-страницы, использующих эти два метода: *Wbstego* [115], *Invisible Secret* [81], *Snow* [109].

Важным вопросом также является протокол передачи скрытых данных через web-страницу. Этот протокол формирует скрытый канал передачи информации. В статье [94] рассматривается протокол формирования скрытого информационного канала с участием двух абонентов. Отправитель загружает скрытое сообщение на *web*-сервер. *Web*-сервер изменяет существующую web-страницу с помощью стеганографического метода встраивания сообщений. Получатель имеет доступ к web-странице и может прочитать встроенные секретные данные при известном ключе встраивания. Для поддержания постоянно действующего скрытого канала связи предложена динамическая операционная модель (*TDOM*) обновления встроенных данных. На основе этой модели разработаны два динамических алгоритма *TDOA-C* и *TDOA-U*. Эти модели работают на основе меток времени. Такой подход позволяет контролировать замену скрытой информации на *web*-странице. При смене скрытой информации сервер формирует новую *web*-страницу с новыми данными и заменяет ей старый вариант.

Современные web-страницы используют встроенные исполняемые модули с открытым кодом, написанные, например, на *JavaScript*. Проблема встраивания цифровых водяных знаков в исполняемые модули с открытым кодом может быть представлена как алгоритм добавления структуры W в

программу P. При этом структура W должна обнаруживаться и извлекаться даже после некоторых преобразований программы P. К таким преобразованиям относится оптимизация и обфускация. Встроенная структура не должна влиять на производительность и корректность работы программы P. Более того, встроенная структура W должна иметь статистические свойства такие, чтобы ее нельзя было обнаружить статистическим анализом программы. Дополнительное требование состоит в максимизации размера W для того, чтобы скрытый канал имел большую пропускную способность.

В статье [63] в качестве контейнера используется открытый программный код. Для этого контейнера предлагается метод встраивания цифрового водяного знака на основе динамического графа. Основная идея состоит во встраивании цифрового водяного знака в топологию динамически построенного графа. Из-за эффектов смещения указателя кода, который манипулирует динамическими структурами графа, эту структуру трудно анализировать. К программному коду применяются преобразования, сохраняющие семантику. Эти преобразования вносят значительные изменения в динамический граф, но являются трудно отслеживаемыми.

В статье [47] предлагается метод встраивания цифрового водяного знака в программу на языке Java для предотвращения кражи программ. Цифровые водяные знаки содержат данные об авторских правах разработчика и встраиваются в классы jles. Такой метод встраивания прозрачен для пользователя, но позволяет идентифицировать законность использования программой данных классов. Процедура кодирования цифровых водяных знаков состоит во внедрения в класс фиктивного метода. На первом этапе внедрения цифровых водяных знаков фиктивный метод класса, который никогда не будет выполнен, добавляется к целевой исходной программе на языке Java. Этот фиктивный метод предоставляет место для кодового слова цифрового водяного знака. Этот фиктивный метод должен

иметь размер, достаточный для размещения цифрового водяного знака. Самое главное, что нужно сделать, это добавить вызов этого метода в исходной программе. Поскольку большие программы изначально содержат много методов, которые редко выполняются, это не позволит пользователям программы легко найти фиктивный метод. Разработчику файла класса следует вручную ввести фиктивный метод и добавить его вызов. Поскольку злоумышленники могут декомпилировать и прочитать файл с описанием классов, разработчик должен аккуратно написать метод-пустышку, так чтобы он не казался пустышкой. Однако, если разработчик не заботится о возможности декомпиляции, или просто предпочитает простые цифровые водяные знаки, можно использовать автоматическое добавление фиктивного метода вместо его ручного описания. На следующем этапе исходная программа компилируется вместе с фиктивными методами обычным компилятором Java. Следующий шаг учитывает проверку байтового кода. Верификатор байтового кода проверяет синтаксическую правильность и согласованность типов классов файла. Чтобы сохранить синтаксическую правильность и единообразие типов данных, используются два подхода. Первый способ сохранить синтаксическую корректность – это ограничить место перезаписи. Числовой код операции, помещающей значение в стек, и код операции, увеличивающей значение в стеке, могут быть перезаписаны с соблюдением синтаксической корректности и соответствия типов. При извлечении цифрового водяного знака предполагается, что известна связь между байтовым кодом и сопоставляемыми им битами, а также отображение битов последовательности на алфавит. Алгоритм извлечения цифрового водяного знака состоит в процедуре, обратной встраиванию. Операнды и коды операций в каждом методе заменяются на битовую последовательность. После этого, на основе правил назначения битов, происходит замена последовательности битов на последовательность символов. После этого, цифровой водяной знак извлекается из фиктивного метода. Предложенный

метод устойчив к аддитивной атаке, атаке с помощью обфускации и атаке декомпиляцией-перекомпиляцией [47].

Однако ни один из методов встраивания цифровых водяных знаков неустойчив относительно всех типов атак.

1.4 Встраивание информации в объекты векторной графики

Стеганографические методы встраивания активно используют изображения в качестве стегоконтейнеров [28,65,107]. Для встраивания активно применяются графические контейнеры различного типа BMP, PNG, JPEG и т.д. Важной характеристикой является максимально возможный размер данных, которые могут быть скрытно встроены в изображение. Наибольшее распространение получил метод встраивания наименее значащего бита (LSB) и его модификации [49,87]. Этот метод эффективен для объектов растровой графики. Также он может быть расширен и применен для встраивания скрытых данных в видеофайлы. Например, в работе [67] предложен метод стеганографического встраивания в видеофайлы высокого разрешения формата AVI. Метод основан на направленных графических узорах. Такой подход преобразует видеофайл в растровый формат и использует направленные графические элементы для размещения битов данных в видеоконтейнере с использованием метода LSB.

Применение методов, разработанных для растровой графики, к векторным изображениям сталкивается с рядом трудностей. Во-первых, форматы файлов векторной графики ориентированы на оптимизацию по информационному объему. Поэтому количество малозначительных данных сведено к минимуму. Во-вторых, файлы векторной графики имеют достаточно четкую структуру. Даже небольшие изменения в файле могут

приводить к значительным последствиям, которые легко обнаружимы визуально и разрушают скрытность передачи данных.

Одним из форматов векторной графики, широко используемым в сети Интернет является SVG (Scalable Vector Graphics). Данный формат основан на языке разметки xml и легко встраивается в HTML-документы. Как правило, формат SVG используется для отображения небольших логотипов или иконок на web-страницах. При использовании SVG-файлов в качестве стегоконтейнера необходимо учитывать три особенности этого формата. Во-первых, файлы распространяются с открытым исходным кодом и легко поддаются анализу штатными средствами, не требуют дополнительного программного обеспечения. Эта особенность накладывает ограничения на формат встраивания данных. Во-вторых, файлы этого формата, как правило, имеют маленький объем и любая громоздкая встроенная информация легко обнаружима. Эта особенность существенно ограничивает объем встраиваемой информации. В-третьих, формат SVG имеет жесткую структуру, основанную на небольшом множестве допустимых команд формирования изображения. Эта особенность накладывает требования на формат встраивания и алгоритм встраивания данных.

Для формата SVG было предложено несколько методов встраивания скрытых данных. Изображения в этом формате представляют собой набор точек и линий. Файл изображения содержит координаты точек и параметры линий. В статье [111] предложен метод скрытия данных, основанных на прорисовке отдельных точек поверх линий. Если линии и точки имеют одинаковый цвет и толщину, то точки визуально не различимы на фоне линии. Кодирование встроенной информации основано на количестве и взаимном размещении точек поверх линии. Обнаружение такой вставки требует анализа взаимного расположения графических элементов изображения и, для многих изображений, является трудоемкой задачей. В статье [14] предложен метод встраивания данных в SVG-файлы на основе

разбиения кривых Безье. Разбиение и представление кривых базируется на кодировании скрытых данных. Представлены алгоритмы прямого и обратного представления данных. В статье [40] предложен алгоритм встраивания данных в PDF-документы. Предложенный алгоритм состоит из извлечения изображения из PDF-файла, преобразования его в формат SVG, встраивания скрытых данных и обратного размещения в PDF-документ. Для встраивания в SVG-изображение используется дискретное вейвлет преобразование и битовые плоскости полутонового контейнера. Модификация метода наименее значащего бита для SVG-графики представлена в статье [53]. В предложенном алгоритме встраивание выполняется с помощью небольшой модификации параметров линий и точек. В статье [54] предложено работать с файлом формата *SVG,* как с текстовым файлом и использовать стеганографические алгоритмы, разработанные для текстовых контейнеров. Автор показывает, что при определенных условиях объем передаваемого сообщения может превышать значения, характерные для растровой графики.

1.5 Выводы

По результатам данной главы можно сделать следующие выводы:

1. Существующие методы стеганографии основаны на использовании памяти исполняемого кода, как стегоконтейнера. Актуальной является задача разработки методов, использующих динамические библиотеки для встраивания и извлечения скрытый данных в области памяти основной программы. Также при проектировании стеганографических алгоритмов необходимо учитывать возможности статического и динамического анализа исполняемого кода.

2. Существующие методы скрытого встраивания информации в HTML-документы основаны на методах текстовой стеганографии. Актуальным является разработка методов встраивания, основанная на рассмотрении HTML-кода, как исполняемого программного кода. Перспективным является направление, рассматривающее HTML-код с точки зрения подхода CSS, который остается неразработанным.

3. Существующие методы скрытого встраивания данных в SVG-изображения являются либо адаптацией аналогичных методов для растровой графики, либо модификацией методов текстовой стеганографии. Актуальной является задача разработки методов скрытого встраивания данных в SVG-изображения, использующих их, как часть HTML-кода с использованием технологий CSS.

Глава 2. Стеганографическое встраивание данных в память исполняемого кода с ограничением по времени

2.1 Введение

Сформулируем основные требования к встраиванию меток безопасности в исполняемый код программы.

1) Метка безопасности отсутствует в исходном коде программы как значение некоторой константы. Статический анализ кода открытого или дизассемблированного кода легко обнаруживает такие метки.

2) Метка безопасности формируется динамически в процессе выполнения программы.

3) Метка безопасности формируется не сразу, а на некотором этапе выполнения программы. Этап появления фрагмента программы по формированию метки безопасности является секретной информацией.

4) Метка безопасности уничтожается в некоторый момент выполнения программы. Метка безопасности существует в оперативной памяти некоторый интервал времени. В этот интервал времени метка безопасности может быть извлечена. Такой подход существенно усложняет динамический анализ кода с целью обнаружения меток безопасности.

5) Метка безопасности зависит от ключевой информации. Ключевая информация вводится пользователем в интерактивном режиме.

6) Команды по формированию метки безопасности распределены по всему исходному коду. Это требование позволяет противодействовать удалению метки безопасности из исходного кода.

7) Метка безопасности формируется в оперативной памяти. Метка безопасности не должна занимать непрерывной блок адресного пространства.

8) Метка безопасности разбивается на блоки. Эти блоки размещаются в памяти с использованием некоторой динамической структуры данных. Адреса размещения блоков зависят от ключевой информации, вводимой пользователем.

Для выполнения этих требований необходимо разработать три алгоритма: алгоритм формирования метки безопасности в процессе выполнения программы, алгоритм размещения метки безопасности в оперативной памяти, алгоритм извлечения метки безопасности.

Размещение метки безопасности в оперативной памяти традиционно выполняется с помощью выделения дополнительной памяти и записи в неё информации. Выделение независимой памяти под метку безопасности может быть обнаружено с помощью отслеживания *API* вызовов программы. Для выделения памяти программа должна обратиться к операционной системе. Если метка безопасности хранится в отдельных ячейках памяти, то она может быть обнаружена последовательным анализом всех выделенных областей памяти. Адреса выделенных областей памяти определяются при перехвате системных вызовов. Последовательная модификация значений в ячейках памяти позволяет определить, какие из ячеек содержат важную для работы программы информацию, а какие ячейки памяти хранят метку безопасности, не участвующую в работе программы. Для предотвращения этой атаки память под метку безопасности должна выделяться одновременно с выделением памяти под структуры, используемые программой для решения основных задач.

В этой главе встраивание осуществляется на этапе разработки программы на языке высокого уровня. В этом случае выделение памяти под метку безопасности может быть осуществлено путем увеличения размера областей памяти, выделяемой под основные переменные. При таком подходе расположение метки безопасности не может быть обнаружено при анализе системных вызовов по выделению памяти. Расширение области памяти для

хранения метки безопасности может быть осуществлено одним из трех способов:

1) Увеличение размеров динамических массивов.

2) Введение дополнительных полей в существующие классы.

3) Увеличение размеров ячеек памяти, выделяемых под переменные.

В первом случае к существующему массиву добавляется некоторое количество ячеек, предназначенных для хранения метки безопасности. Если такой массив один, то метка безопасности полностью записывается в эти ячейки памяти. Если массивов несколько, то метка безопасности распределяется между ними. Распределение частей метки безопасности между несколькими массивами может быть организовано различными способами. Метка безопасности w может быть разбита на части $w_1, \dots, w_n$.

$$w = w_1||w_2|| \dots ||w_n.$$

Части метки безопасности w_i $(i = 1,..,n)$ распределяются между дополнительными ячейками массивов в лексикографическом порядке (Рисунок 2.1). Для того, чтобы не возникло неопределенности выбирается один тип переменной. Встраивание производится во все массивы данного типа. Количество добавляемых элементов массива определяется размером метки безопасности. Разбиение метки безопасности может быть выполнено на равные доли. В этом случае упрощается реализация встраивания, но также упрощается поиск метки безопасности. Метка безопасности может разбиваться на неравные доли. Этот подход существенно усложняет стеганографический анализ. Программа встраивания метки безопасности должна анализировать исходный код программы, отыскивать массивы заданного типа и увеличивать их размер. Увеличение размера выполняется изменением параметров размера массива. Программа встраивания должна

запоминать имена массивов, исходное количество элементов в них и количество элементов, добавленных к ним. После этого, программа формирует список встраивания и встраиваемую информацию.

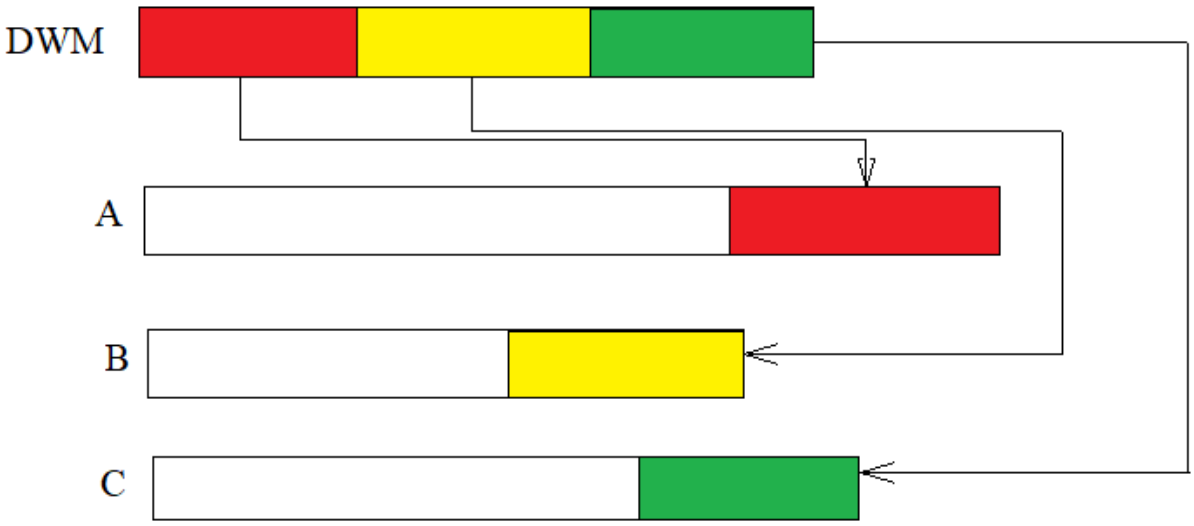

Рисунок 2.1. Части метки безопасности распределяются между массивами в лексикографическом порядке.

В качестве альтернативы лексикографического упорядочивания можно хранить структуру с порядком следования массивов. Но это потребует создания новой динамической структуры.

Второй способ состоит в случайном выборе порядка встраивания информации в добавленные элементы массива. В этом случае необходимо использовать какую-то динамическую структуру данных. Для этой цели может быть использован однонаправленный динамический список (Рисунок 2.2). В дополнительные ячейки массивов в произвольном порядке встраиваются элементы динамического списка. В таком подходе необходимо решить три задачи. Создание указателя на первый элемент динамического списка. Настройка ссылок для перехода между элементами списка. Преобразование элементов списка к формату массива и обратное

преобразование. Такой подход значительно сложнее поддается стеганографическому анализу. Ссылка на первый элемент может быть организована как отдельная переменная. Но это подход, не удовлетворяющий сформулированным выше требованиям. Новая переменная требует отдельного выделения памяти. Для первого элемента списка может быть выбран массив первый или последний в лексикографическом порядке. Следует учесть, что подход на основе динамического списка уменьшает объем встраиваемой метки безопасности и требует большой объем памяти для служебной информации.

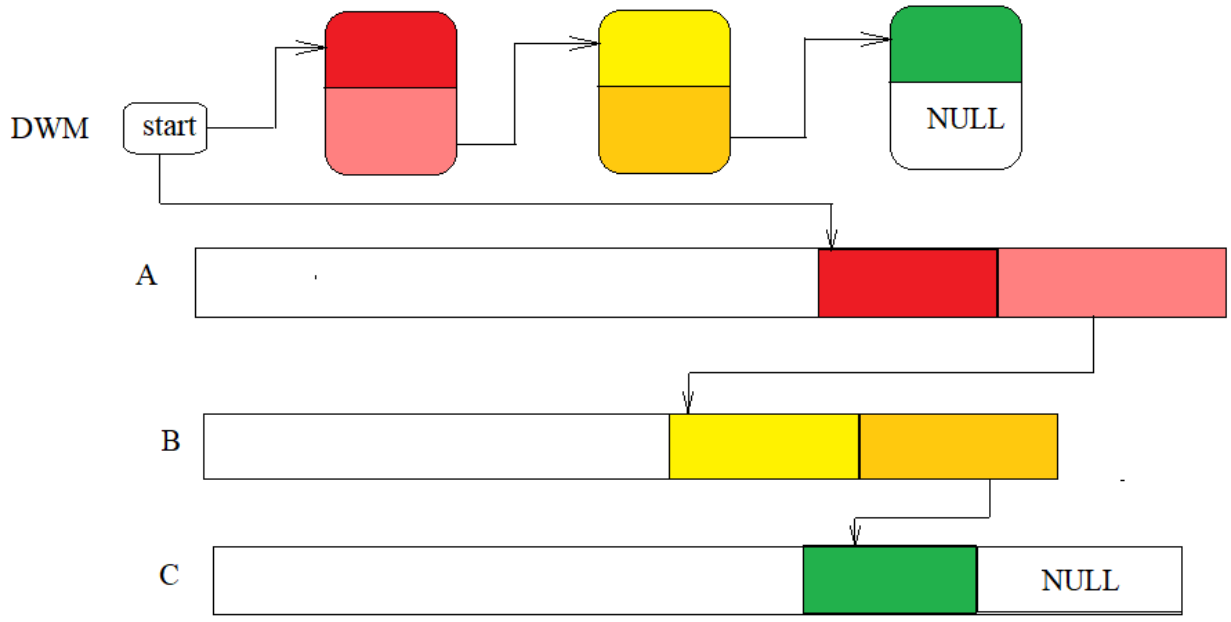

Рисунок 2.2. Части метки безопасности распределяются между массивами на основе списка.

При создании дополнительных полей в существующих классах могут быть использованы такие же подходы, как и для массивов. Части метки безопасности могут быть распределены между объектами одного класса в лексикографическом порядке (Рисунок 2.3). Для размещения частей метки

безопасности между объектами одного класса может быть использован динамический список (Рисунок 2.4).

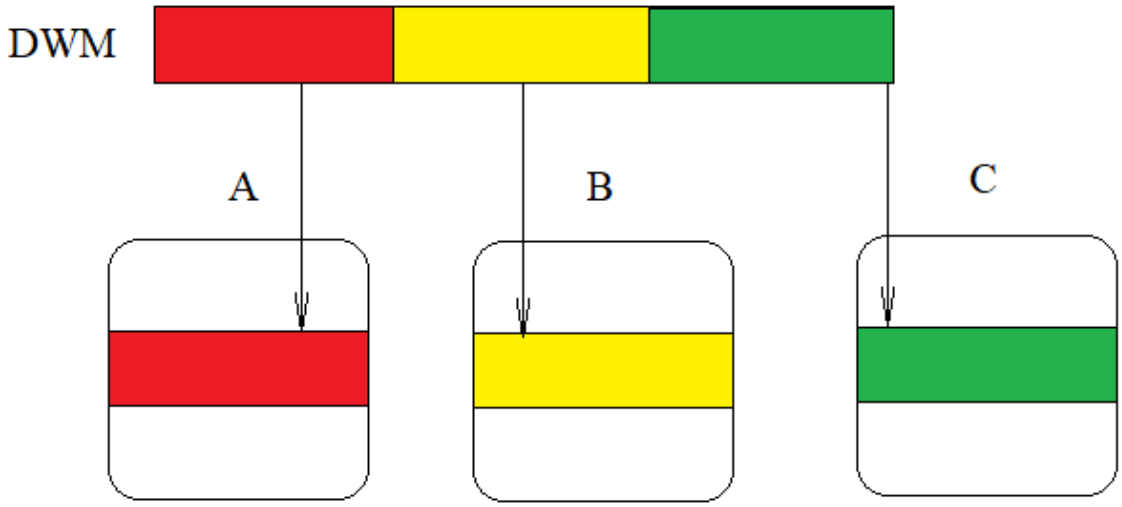

Рисунок 2.3. Части метки безопасности распределяются между объектами одного класса в лексикографическом порядке.

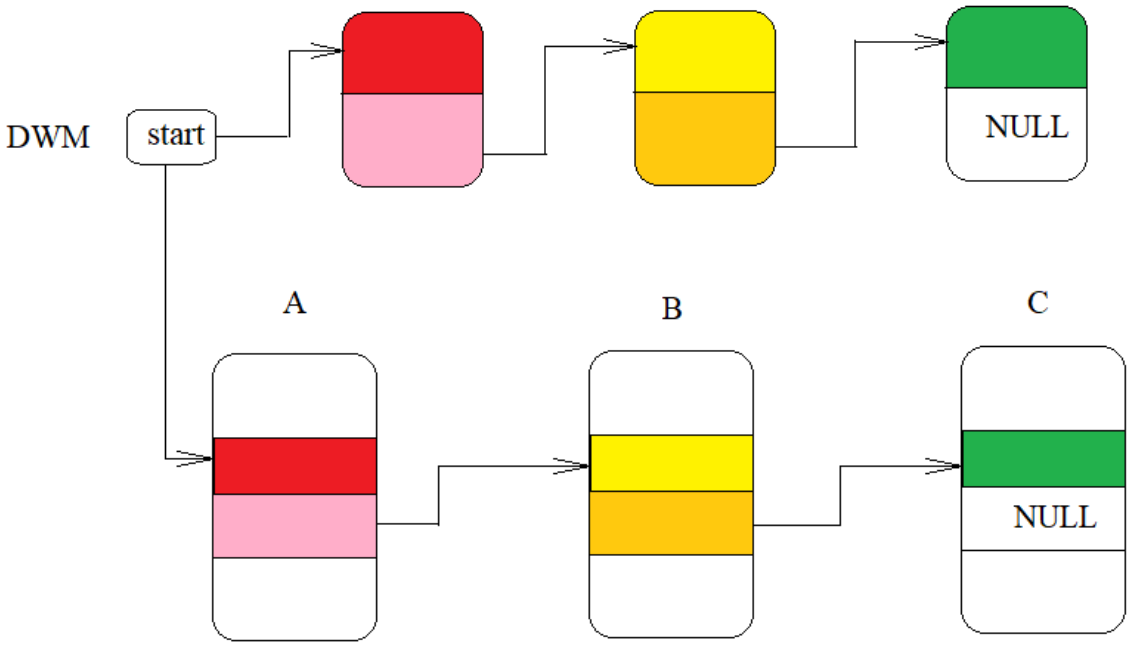

Рисунок 2.4. Части метки безопасности распределяются между объектами одного класса на основе списка.

Более сложной задачей является встраивание метки безопасности с помощью увеличения размера ячеек отдельных переменных. Технология состоит в увеличении размера ячейки памяти, выделяемой под переменную. Этот подход сталкивается с большим количеством трудностей, так как требует переопределения арифметических операций. Также нужно следить за тем, чтобы не происходило переполнение ячеек памяти. В обычной ситуации переполнение приводит к потере старших битов. При встраивании меток безопасности переполнение может привести к модификации встроенных данных.

Извлечение метки безопасности должно производится самой программой. Извлечение скрытого сообщения сторонней программой требует вмешательства в чужое адресное пространство. Такое вмешательство запрещено операционной системой. Некоторое событие должно вызывать запуск участка кода программы по извлечению цифрового водяного знака. Таким событием может быть нажатие некоторого набора клавиш. Второй вариант состоит в вводе в некоторое окно парольной фразы. Событие по выводу метки безопасности должно быть секретным. В обоих случаях должна активироваться некоторая подпрограмма по выводу метки безопасности. Эта подпрограмма должна быть встроена в исходный код программы вместе с самой меткой безопасности. Подпрограмму извлечения метки безопасности можно рассматривать как часть метки безопасности. Эта подпрограмма должна обладать следующими свойствами:

1) Обладать информацией о размещении метки безопасности в памяти.

2) Формироваться автоматически на этапе встраивания метки безопасности.

3) Не выделяться на фоне остальной программы по своему стилю и структуре.

Реализация системы автоматического формирования и встраивания такой подпрограммы в исходный код программы имеет ряд трудностей технического характера.

2.2 Алгоритм встраивания и извлечения цифрового водяного знака

Метка безопасности, как правило, представляет собой строку символов, сформированную с помощью криптографического алгоритма. Поэтому эта строка удовлетворяет требованиям к псевдослучайным последовательностям. Она не может быть обнаружена с помощью сканирования памяти программы, так как неотличима от остаточной информации, хранящейся в памяти. Обнаружение метки безопасности возможно только при известных адресах размещения ее фрагментов в памяти. Сформулируем требования к алгоритму выработки адресов размещения скрытой информации в памяти программы.

1) Адрес размещения метки безопасности должен быть привязан к параметрам запуска программы. Это требование необходимо для обнаружения и проверки метки безопасности.

2) Адрес первого блока метки безопасности должен быть уникален при каждом запуске программы.

3) Размещение частей в памяти программы должно быть уникально при каждом запуске программы.

4) Задача выявления модуля алгоритма, формирующего метку безопасности, должна быть трудной.

Для осложнения поиска модуля формирования метки безопасности в памяти программы во время атаки необходимо сделать трудноопределимой точку входа в этот модуль. В предлагаемом решении авторизация пользователя в программе осуществляется внешней динамической библиотекой. Для формирования метки безопасности в памяти программы используется дополнительная учетная запись псевдопользователя. При вводе

данных этого псевдопользователя программа сообщает об ошибке авторизации, но при этом запускает модуль формирования метки безопасности в памяти программы. Данные о пользователях защищаются стандартным способом с помощью алгоритмов хеширования с добавлением «соли». Список пользователей не хранится в открытом виде. Поэтому при анализе исполняемого кода взломщику невозможно отследить момент формирования метки безопасности в памяти программы.

Алгоритм формирования метки безопасности включает пять шагов.

1) Метка безопасности C разбивается на блоки по 2 байта.

$$C=C_0||C_1||\ldots||C_n.$$

2) Выделяется динамическая область памяти размером B. Адрес данной области памяти обозначим M.

3) Адрес размещения ΔM первого блока метки безопасности C_0 вычисляется на основе адреса M и смещения h. Величина h вычисляется как функция от времени запуска программы T_0. При этом должно выполняться условие $h<B$.

$$h=h(T_0).$$
$$\Delta M=M+h.$$

При реализации программы была использована функция:

$$h=(T_0 \bmod 100)/10.$$

Размер блока выбирался B=10 байт.

4) Вычисляется смещение адреса размещения первого блока M_0 относительно базового адреса программы M_{start}.

$$M_0=\Delta M - M_{start}.$$

Переменная, хранящая адрес M_0, имеет глобальную область видимости в динамической библиотеке аутентификации. Адрес размещения этой переменой служит ключевой информацией размещения цифрового водяного знака в памяти программы. Поэтому при реализации программы необходимо предусмотреть его копирование на внешний носитель.

5) Первый блок метки безопасности размещается по адресу M_0. Размещение остальных блоков метки безопасности выполняется по адресам M_i.

$$M_i=M_{i-1}+4\Delta M,$$

$$M_i=M0+4i\Delta M.$$

Извлечение метки безопасности осуществляется отдельным приложением. Программе верификации передается два параметра: имя процесса, в котором необходимо проверить метку безопасности, и ключевая информация M_0.

Алгоритм проверки метки безопасности состоит из семи шагов.

1) По имени процесса определяется его базовый адрес M_{start} с помощью системных функций.

2) На основе базового адреса M_{start} и ключевой информации M_0 определяется адрес размещения первого блока метки безопасности.

$$\Delta M= M_0 + M_{start}.$$

3) По адресу ΔM извлекается два байта, которые являются первым блоком C_0 метки безопасности.

4) По имени процесса с помощью системных функций извлекается время запуска процесса T_0.

5) По времени запуска T_0 может быть вычислено смещение $h=h(T_0)$.

6) Размещение остальных блоков метки безопасности вычисляется так же, как при встраивании.

$$M_i=M_0+4i\Delta M.$$

Каждый блок C_i имеет размер два байта.

7) Метка безопасности вычисляется как объединение блоков с помощью операции конкатенации.

$$C=C_0||C_1||\ldots||C_n.$$

Предложенная схема встраивания метки безопасности жестко связана как с программой, в которую производится встраивание, так и с пользователем, осуществляющим встраивание. Связь с программой реализуется через время запуска и адрес первого блока. Адрес первого блока определяется при компиляции программы за счет глобальной видимости переменной. Переменная, содержащая адрес первого блока метки безопасности, хранится в сегменте данных программы, а не в динамической памяти. Привязка к конкретному сеансу запуска программы осуществляется при вычислении шага смещения адресов. Таким образом при каждом новом запуске программы блоки метки безопасности будут размещаться по разным адресам. Защита от извлечения метки безопасности с помощью анализа памяти средствами дизассмеблирования осуществляется сокрытием момента времени формирования его в памяти программы. Пользователь, осуществляющий проверку, должен запустить процесс аутентификации и ввести имя псевдопользователя. Только после этого в памяти формируется метка безопасности. Таким образом, для взлома системы формирования

метки безопасности злоумышленник должен предварительно реализовать успешную атаку на подсистему аутентификации программы.

2.3 Программный комплекс формирования ЦВЗ в памяти программы

2.3.1 Общая структура программного комплекса

Программный комплекс встраивания и проверки метки безопасности реализован в виде динамической библиотеки авторизации (*auto.dll*) и программы проверки метки безопасности (*verificator.exe*) (Рисунок 2.5).

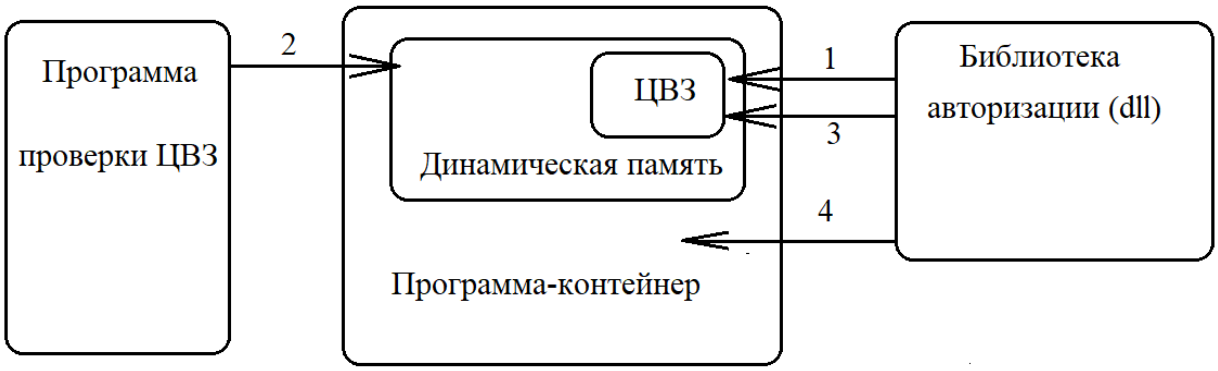

Рисунок 2.5. Схема взаимодействия модулей формирования и проверки метки безопасности с программой-контейнером. 1 – формирование МБ, 2 – проверка МБ, 3 – удаление МБ, 4 – штатный режим.

В качестве программы-контейнера может использоваться любой программный комплекс на этапе его разработки. Единственным требованием является вызов динамической библиотеки *auto.dll* на каком-либо этапе работы программы.

Динамическая библиотека авторизации вызывается основной программой для идентификации и аутентификации пользователя. После этого программа, вызвавшая библиотеку, переходит в режим ожидания сообщения о входе зарегистрированного пользователя.

Динамическая библиотека авторизации *auto.dll* после вызова может работать в двух режимах:

1) ***Авторизация пользователя***. В данном режиме динамическая библиотека формирует запрос на имя пользователя и пароль. Далее библиотека выполняет штатные операции проверки имени и пароля пользователя. Использован стандартный подход с хешированием аутентифицирующей информации. Проверка подлинности пользователя выполняется по базе данных хэш-значений. Если пользователь зарегистрирован в системе, то библиотека формирует сообщение о разрешении допуска и передает его основной программе (программе-контейнеру).

2) ***Формирование МБ***. Кроме зарегистрированного пользователя в базе данных присутствует псевдопользователь, соответствующий метке безопасности. Этот псевдопользователь не дает право входу в программу. При вводе имени и пароля псевдопользователя динамическая библиотека переходит в режим формирования метки безопасности. Имя и пароль псевдопользователя формируются на этапе создания базы данных зарегистрированных пользователей. Возможен также вариант работы только с парольной защитой. В этом случае имя пользователя отсутствует. Библиотека реагирует только на два пароля: пароль для продолжения работы в штатном режиме и пароль для формирования метки безопасности.

После перехода в режим формирования метки безопасности динамическая библиотека обращается напрямую к памяти программы-контейнера. Библиотека записывает в память программы информацию, необходимую, для формирования метки безопасности. После этого библиотека включает счетчик времени и переходит в штатный режим работы. Для метки безопасности устанавливается период ее существования. Если истекает время существования метки безопасности, то библиотека авторизации обращается к памяти программы и удаляет информацию,

записанную при формировании метки безопасности. Обращение к памяти для удаления метки безопасности осуществляется прозрачно для пользователя. Программа-контейнер и динамическая библиотека продолжают функционировать в штатном режиме. Если в логике работы программы предусмотрена повторная аутентификация, воспользовавшись которой пользователь попытается сформировать метку безопасности до истечения времени действия уже сформированной метки безопасности, то библиотека не будет повторно обращаться к памяти программы контейнера, а просто обнулит счетчик, продлив время существования уже существующих встроенных данных.

Для определения адресов встраивания данных в динамическую память программы-контейнера динамическая библиотека использует параметры запуска процесса этой программы и ключевую информацию. Ключевая информация также известна программе, проверяющей метку безопасности.

Программа проверки метки безопасности *verificator.exe* реализована в виде консольного приложения. Данное приложение решает следующие задачи:

1) ***Поиск программы-контейнера***. При решении этой задачи приложение ищет окно авторизации, формируемое библиотекой авторизации. Данное окно является специфическим. Окно может быть не активным в данный момент, но оно присутствует в системе. По окну авторизации определяется идентификатор процесса, из которого запущено данное окно. Используя API операционной системы по идентификатору процесса, определяются параметры запуска процесса. Если окно не найдено, то приложение выдает соответствующее сообщение.

2) ***Проверка МБ***. По параметрам процесса и ключу встраивания определяются адреса в динамической памяти программы-контейнера для хранения метки безопасности. После этого происходит обращение к памяти программы и чтение данных. Прочитанные данные объединяются и

сравниваются с ожидаемым содержанием метки безопасности. Приложение выдает пользователю сообщение о результатах проверки.

Программная реализация динамической библиотеки *auto.dll* и приложения проверки метки безопасности *verificator.exe* выполнено с помощью системных вызовов *WinAPI.*

2.3.2 Программная реализация библиотеки auto.dll

Динамическая библиотека авторизации *auto.dll* реализована на языке *C++* для ОС *Windows*. В библиотеке реализованы следующие функции, связанные с встраиванием меток безопасности:

1) Метод *bool CheckPassword* (*char* hashPassword*) булевского типа. Аргументом метода является массив символьного типа, в который передается хэш пароля. Метод возвращает флаг «*TRUE*», если хэш-значение переданного пароля совпадает с хэш-значением пароля псевдопользователя, активирующего режим встраивания метки безопасности. В противном случае возвращается значение «*FALSE*». Если данный метод возвращает ложное значение, то библиотека остается в режиме ожидания ввода авторизующей информации. Если метод возвращает истинное значение, то библиотека вызывает функцию *WriteCvz*().

2) Метод *void WriteCvz*() выполняет встраивание метки безопасности в оперативную память программы контейнера. Этот метод решает три задачи: выделение памяти для метки безопасности, запись данных в выделенную память и очистка памяти после истечения времени существования метки безопасности. Для решения каждой из этих задач реализованы отдельные методы, вызываемые методом *WriteCvz*(). Для выделения памяти вызывается метод *MemoryAllocation*(), для записи метки безопасности вызывается метод *_writeCvz*(), для удаления метки безопасности используется метод *ClearCvz*().

3) Метод *void MemoryAllocation*() выделяет большое количество ячеек в динамической памяти процесса-контейнера. Выделение большого объема динамической памяти необходимо для затруднения поиска данных, соответствующих метке безопасности. Если встраиваемые данные предварительно зашифрованы, то их статистические характеристики не отличимы от «мусора», оставшегося в оперативной памяти. Не существует эффективных алгоритмов, отделяющих полезные данные, обладающие статистическими характеристиками случайной последовательности от истинно случайной последовательности. По сути, встраиваемые данные маскируются под случайную последовательность.

Этот метод вызывается из контекста процессора-контейнера, поэтому выделение динамической памяти выполняется штатным образом. После выделения памяти, адрес первой ячейки записывается в переменную $startMemoryBlockContainer$. Эта переменная доступна всем методам динамической библиотеки *auto.dll*.

4) Метод *void _writeCvz*() осуществляет запись метки безопасности в память процесса-контейнера. Запись скрытых данных выполняется в последовательность ячеек. Адрес первой ячейки вычисляется на основе времени старта процесса контейнера $creation\_time.dwLowDateTime$. Этот параметр определяется с помощью функции *GetProcessTimes*(), входящей в *WinAPI*. Стартовый адрес в выделенной динамической памяти вычисляется по формуле:

$$startAdressIndex = (creation\_time.dwLowDateTime \,\%\, 100) / 10.$$

Адрес первой ячейки для встраивания символов метки безопасности находится из соотношения:

$$baseAdress = startMemoryBlockContainer + startAdressIndex.$$

Адрес ячейки с номером i вычисляется по формуле:

$$Adress[i] = baseAdress + startAdressIndex \cdot i.$$

В каждую ячейку с адресом $Adress[i]$ встраивается один символ, имеющий номер i в последовательности символов метки безопасности. После встраивания всей последовательности инициализируется переменная $CvzTime$, в которой хранится время создания метки безопасности.

5) Метод *void ClearCvz*() удаляет из памяти метку безопасности по истечении промежутка времени $CvzInterval$. Интервал времени существования метки безопасности отсчитывается от момента завершения записи его в память процесса-контейнера $CvzTime$. Метод осуществляет перезапись ячеек памяти по адресам $Adress[i]$ случайной последовательностью. Простое освобождение памяти не гарантирует возможности злоумышленнику найти информацию в «мусоре» оперативной памяти.

Алгоритм встраивания метки безопасности представлен на блок-схеме (рисунок 2.6).

Рисунок 2.6 Блок-схема встраивания метки безопасности.

2.3.3 Программная реализация verificator.exe

Программа проверки метки безопасности реализована в виде консольного приложения на языке *C++*. В программе реализованы следующие функции проверки метки безопасности в динамической памяти программы-контейнера:

1) Функция *main*() использует функцию *FindWindow*() из *WinAPI* для поиска окна авторизации, созданного библиотекой *auto.dll*. Поиск данного окна позволяет определить идентификатор процесса, в динамическую память которого выполнено встраивание метки безопасности. Если процесс найден, то вызывается функция *char* GetCvz*(*DWORD processId*). В противном случае выдается сообщение об отсутствии процесса. Эта функция возвращает встроенную последовательность символов метки безопасности. Данная последовательность выводится на экран и проверяется на соответствие эталонной последовательности, вписанной в код программы верификации.

2) Функция *char* GetCvz*(*DWORD processId*) получает в качестве параметра идентификатор процесса и возвращает встроенную последовательность. Первая задача, которую необходимо решить – это определить адрес ячейки памяти, начиная с которой производилось встраивание. Эту проблему решает функция *DWORD_PTR GetBaseAdress*(*DWORD processId*), которая возвращает переменную $baseAdress$, хранящую указатель на первую ячейку встроенной метки безопасности. Адрес ячейки памяти с номером $index$ вычисляется по формуле:

$$Adress[index] = baseAdress + startAdressIndex * index.$$

Вычисление адресов выполняется в цикле. Количество ячеек памяти для считывания известно заранее, так как программа выполняет верификацию известной метки безопасности.

3) Функция *DWORD_PTR* GetBaseAdress(*DWORD processId*) получает в качестве параметра идентификатор процесса и возвращает адрес ячейки динамической памяти $baseAdress$, начиная с которой выполняется встраивание метки безопасности. Стартовый адрес состоит из начального адреса процесса и смещения относительно начального адреса. Начальный адрес процесса является индивидуальным для каждого процесса и определяется операционной системой. Смещение относительно начального адреса является одинаковым для всех меток безопасности, формируемых библиотекой *auto.dll* в данной программе. Одинаковость относительного смещения следует из алгоритма, заложенного в динамическую библиотеку, и его реализации. Относительное смещение определяется на этапе создания программы с помощью отладчика и хранится в программе проверки метки безопасности в виде константы. Таким образом, программа верификации метки безопасности является жестко настроенной на конкретное приложение. Это ограничение является обычным для любой стеганографической схемы. Алгоритм проверки привязан к конкретному стеганографическому контейнеру. Изменения в программе контейнере требует повторного вычисления относительного смещения.

После получения базового адреса с помощью функций *WinAPI* определяется время запуска процесса контейнера, которое хранится в переменной $creation\_time.dwLowDateTime$. На основе этих двух переменных вычисляется стартовый адрес встраивания:

$$startAdressIndex = (creation\_time.dwLowDateTime \,\%\, 100) / 10.$$

Базовый адрес встраивания находится по формуле:

$$baseAdress = startAdress + startAdressIndex.$$

Полученное значение возвращается вызывающей функции.

Блок-схема верификации метки безопасности представлена на рисунке 2.7.

Рисунок 2.7 Блок-схема проверки метки безопасности.

2.4 Практическое использование программного комплекса

Предложенный алгоритм встраивания меток безопасности в память исполняемого кода был внедрен в деятельность ООО «СМАРТФОРС», занимающейся разработкой программных модулей для различных проектов на условиях аутсорсинга.

Метки безопасности в памяти исполняемого кода формировались в исполняемых кодах, предоставляемых компаниям по их техническому заданию. Одной из составляющих модуля являлась библиотека аутентификации. Встраивание метки безопасности согласовывалось с компанией заказчиком. Метка безопасности содержала три составляющих:

1) наименование компании-заказчика,

2) наименование компании-поставщика («SMARTFORCE»),

3) номер версии программного кода,

4) дату поставки модуля.

Вся информация конкатенировалась в одну строку и шифровалась алгоритмом AES. Данный алгоритм шифрования был выбран, так как он является международным стандартом, а компания ООО «СМАРТФОРС» работает с интернациональными проектами. Ключ шифрования генерировался для каждого модуля независимо с помощью надежного генератора ключей. Политика хранения и использования ключей определяется политикой информационной безопасности ООО «СМАРТФОРС».

С помощью встроенных меток безопасности решались следующие задачи:

1) Отслеживание использования модуля компанией-заказчиком только в программном обеспечении, оговоренном в техническом задании. При разработке модуля в договоре оговаривается для какого программного комплекса он используется. При использовании модуля в программном

обеспечении, не предусмотренном первоначальным договором, заключается дополнительный договор. Такой подход защищает исполнителя от неправомерного использования программных модулей. Заказчик и исполнитель могут совместно проверять какие модули в какое программное обеспечение встраиваются при возникновении нештатных ситуаций в работе программы.

2) Контроль версий, используемых заказчиком. При возникновении сбоя в работе программного обеспечения заказчика по вине модуля исполнителя, выполняется проверка версии модуля и даты его предоставления. Если заказчик использовал версию не соответствующую техническому заданию, то это может быть обнаружено с помощью встроенной метки безопасности.

3) Контроль версий при разработке и тестировании программного обеспечения. ООО «СМАРТФОРС» использует бизнес-модель с распределенной разработкой между исполнителями. Исполнители не привязаны в своем расположении к офису и находятся в различных городах. Допустимо нахождение работника в другом государстве. При передаче такому работнику программного модуля для тестирования он проверяет соответствие версии и даты в документации с информацией метки безопасности.

2.5 Результаты

Сформулируем основные результаты данной главы:

1) Предложен алгоритм встраивания и извлечения метки безопасности в память исполняемой программы с использованием динамической библиотеки аутентификации.

2) Подключение внешней динамической библиотекой позволяет работать с памятью процесса для встраивания и извлечения скрытой информации.

3) Использование внешней библиотеки аутентификации для формирования метки безопасноти позволяет управлять его временными характеристиками, такими как момент времени формирования и длительность существования. Такой подход усложняет стеганографический анализ с помощью динамического анализа кода и повышает устойчивость стеганографической вставки к внешним атакам.

4) Для встраивания и извлечения метки безопасности применяются парольные фразы, что делает стеганографическую систему ключевой.

5) Метка безопасности записывается в память в распределенном виде, что существенно усложняет его поиск и выделение с помощью анализа памяти.

6) Выполнена программная реализация библиотеки аутентификации с модулем встраивания метки безопасности, которая может быть использована в любом приложении.

Результаты данной главы опубликованы в работах [7,8,9,56,57].

Глава 3. Скрытое встраивание данных в web-страницу

3.1 Введение

В современной сети Интернет широкое распространение получила информация, распространяемая с открытым кодом. К такой информации можно отнести *HTML*-страницы, *SVG*-изображения и *JS*-приложения. Такой подход существенно повышает скорость передачи информации, однако снижает возможности скрытой передачи данных, которая является основной задачей стеганографии. Большинство стеганографических алгоритмов использует легальные передаваемые файлы в качестве контейнеров для встраивания скрытой информации. При этом в исходном файле-контейнере должно быть достаточно много данных, изменение которых слабо сказывается на представлении документа и не может быть обнаружено визуально. В связи с чем, наблюдается активное развитие алгоритмов стеганографического встраивания в мультимедийные данные. Наибольшее количество алгоритмов разработано для растровых изображений. Информация, передаваемая в открытом виде, по сути, представляет собой текстовый документ. Скрытое встраивание в текстовые документы является одной из самых сложных задач стеганографии. Основная трудность состоит в том, что количество данных, которые мало влияют на отображение документа, является небольшими. Изменение одного значимого символа приводит к существенным визуальным эффектам. Изменение буквы или знака препинания может быть легко обнаружено и исправлено, так как противоречит правилам языка, на которых написан документ. Простая проверка орфографии и пунктуации может полностью разрушить встроенное сообщение. Существующие алгоритмы для встраивания используют пробелы и отступы между словами. Но такое встраивание также не является

надежным, так как может быть достаточно легко обнаружено с помощью алгоритмов с линейной трудоемкостью.

Документы с открытым кодом в сети Интернет основаны на языке разметки *xml*. Каждый документ, кроме отображаемого текста, содержит управляющие тэги, определяющие формат, стили и другие свойства отображаемого текста или изображения. Эти тэги предоставляют дополнительную возможность передать скрытые данные в структурированном документе с открытым кодом. При этом необходимо решить три задачи. Во-первых, определение тегов, оказывающих минимальное влияние на отображение текста, которое не может быть обнаружено визуально, либо введение новый тегов в документ, никак не сказывающихся на отображении текста. Во-вторых, изменения в документе не должны обнаруживаться штатными или статистическими методами. В-третьих, необходима схема определения тегов, в которые произошло встраивание. Причем задача определения тэгов должна низкой трудоемкостью для отправляющей и принимающей стороны и высокой трудоемкостью для стегоаналитика.

Первая задача, связанная с формированием измененных тегов, требует анализа структуры документа. В любой достаточно большой документ могут быть введены новые тэги. Эти тэги будут составлены в соответствии со всеми требованиями. Возможность введения этих тегов основана на том, что программист имеет возможность создания собственных стилей, которые в дальнейшем применяются к элементам документа. Поэтому существует возможность создать класс стилей, который не будет применяться ни к какому элементу документа. Выявление таких стилевых классов при анализе документа требует установления логических связей. Отсюда вытекает, что созданный класс не должен быть изолированным, но не должен быть применен при отображении частей документа. В этом случае отследить логику работы браузера при отображении документа автоматически нельзя и

требуется интеллектуальный анализ с участием человека. Если документ достаточно большой, то «ручная» работа по его анализу и интерпретации становиться трудоемкой, что обеспечивает надежное сокрытие данных. Этот подход решает вторую задачу.

Третья задача обнаружения тегов отправляющей и принимающей стороной может быть решена на основе ключевой схемы. Алгоритм формирования новых тегов должен включать ввод ключевой информации. Отправитель сообщения при встраивании данных использует ключ встраивания. Получатель также распознает тэги со встроенной информацией используя ключ встраивания. Оба эти алгоритма должны характеризоваться низкой трудоемкостью. Алгоритм определения тэгов встраивания без знания ключа должен иметь высокую трудоемкость. В идеале этот алгоритм должен быть основан на полном переборе, то есть иметь экспоненциальную трудоемкость. В этом случае предварительное шифрование встроенных данных делает систему высокозащищенной.

Документы, использующие язык разметки, имеют древовидную структуру. Обозначим граф, соответствующий структуре документа через $T=(V,E)$. V – множество вершин графа. E – множество дуг графа. Дерево задает иерархическую структуру документа. Иерархия реализуется за счет вложенности тэгов. Например, в HTML-документах иерархия может отслеживаться по тэгам <*div*>. В других объектах на основе языка разметки xml тэги могут быть иными.

3.2 Встраивание скрытых данных в HTML-документ

3.2.1 Общие требования к системе

Предлагаемая система предназначена для скрытой передачи данных через *web*-страницы. Отправляющая сторона должна иметь доступ к

модификации *HTML*-кода. Принимающая сторона должна владеть информацией о том, на какой странице сайта размещена скрытая информация, а также некоторой ключевой информацией, без которой извлечение данных невозможно. Такая схема получила название ключевой стеганографической системы. Передающая сторона встраивает дополнительную информацию в web-страницу, используя некоторый ключ, известный также принимающей стороне. Получатель информации открывает страницу в режиме просмотра *HTML*-кода и, используя общий ключ, извлекает скрытое сообщение. Следует учитывать, что просмотр кода страницы доступен любому пользователю с помощью штатных функций браузера. Поэтому схема встраивания должна максимально затруднять, а в идеале делать невозможным, выявление встроенной информации без знания ключа. Исходя из этой схемы, можно сформулировать требования к разрабатываемой стеганографической системе:

1) Встраиваемое сообщение не должно влиять на отображение web-страницы в любом браузере.

2) Встроенное сообщение не должно определяться из анализа HTML-кода без знания ключа встраивания.

3) Встроенное сообщение должно быть максимально интегрировано в web-страницу, чтобы затруднить задачу поиска вставки стегоаналитику.

Любой *HTML*-документ имеет древовидную структуру. Для встраивания будем использовать листовые вершины. Листовые вершины определяются из анализа кода HTML-страницы, как тэги, не имеющие потомков. При этом существуют ограничения на листовые вершины, которые могут быть задействованы в алгоритме. Для встраивания не могут быть использованы теги: '*TITLE*', '*STYLE*', '*LINK*', '*SCRIPT*', '*META*', '*P*', '*INPUT*', '*BASE*', '*IMG*'. Листовые теги, в которые может быть добавлена скрытая информация, будем называть свободными листовыми вершинами.

3.2.2 Алгоритм встраивания сообщения

Алгоритм встраивания сообщения в дерево HTML-тегов состоит из следующих шагов:

1) Выполняется рекурсивный обход дерева тегов для поиска пустых листовых вершин. В дальнейшем эти листовые вершины используются как родительские для создания стеганографических вершин.

2) По количеству найденных пустых листовых вершин определяется максимальный размер сообщения, которое может быть встроено в данный HTML-документ. В одну листовую вершину может быть встроено 2 блока по 4 бита информации.

3) Встраиваемое сообщение M шифруется одним из симметричных шифров. Ключ шифрования k известен обеим сторонам.

$$C=E_k(M).$$

4) Встраиваемое зашифрованное сообщение C разбивается на блоки по 4 бита.

$$C=C_1||C_2||...||C_n.$$

5) Для каждых двух блоков генерируется класс с именем l_i. Один новый класс соответствует одному символу.

6) Сформированные классы встраиваются в пустые листовые вершины. Причем встраивание производится в случайном порядке, чтобы дополнительно запутать аналитика.

Классы, соответствующие блокам встраиваемой информации, должны быть внешне неотличимы от легальных классов. Необходимо автоматически генерировать имя и стили классов.

Новые классы генерируются следующим образом:

1) Для каждых двух блоков встраиваемой информации генерируется класс с именем l_i. В дальнейшем этот класс будем называть классом символов.

2) В класс добавляется два тега. Теги t_i формируются для каждого блока встраиваемой информации по Таблице 1. Имена классов внутри тегов формируются случайно.

3) К тегам могут добавляться стили со случайными значениями.

Таблица 1. Кодирование четырехбитных последовательностей с помощью тегов (последовательности приведены в шестнадцатеричной системе счисления).

Последовательность	**Тег**	**Последовательность**	**Тег**
0	s	8	b
1	div	9	a
2	h1	A	span
3	h2	B	dl
4	h3	C	dt
5	h4	D	em
6	h5	E	i
7	h6	F	city

Алгоритм формирования случайных имен классов состоит из четырех шагов.

1) Выполняется обход всего дерева классов в HTML-документе. Существующие имена записываются в динамический массив.

2) Формируются имена новых классов. Для этого из массива имен классов случайным образом извлекается один элемент.

3) К извлеченному имени добавляется случайная строка символов.

4) Полученное имя проверяется на уникальность путем сравнения с именами уже существующих классов и добавляется в массив имен.

5) Имена классов символов должны быть неотличимы от обычных имен классов *HTML*-документа. Для этого имена должны эмитировать последовательности символов какого-либо широко используемого приложения. Например, могут быть использованы имена фреймворка *Angular* для своих классов, которые имеют структуру «*ng*-<2 символа>-<2 символа>». Четыре символа, присутствующие в конце имени, формируются фреймворком и не имеют явной закономерности, которая может быть обнаружена при анализе *HTML*-кода.

Алгоритм формирования имени класса символа с номером i при ключе встраивания d состоит из следующих шагов:

1) Вычисляется индекс символа с номером i, как конкатенация ключа встраивания d номера блока i.

$$n_i = d||i.$$

2) Вычисляется хэш-значение от индекса n_i.

$$h_i = H(n_i).$$

3) Может быть использована любая стойкая хэш-функция. При тестировании алгоритма использовалась функция *SHA*512.

4) Хэш значение представляется как набор символов из английского алфавита с использованием любой подходящей кодировки.

$$h_i = a_1||a_2||a_3||a_4||...$$

5) Формируется метка l_i для символа с номером i на основе первых четырех символов представления хэш-функции.

$$l_i = «ng\text{-}a_1a_2\text{-}a_3a_4».$$

Ключ встраивания d используется только при формировании имен классов. В качестве этого ключа может использоваться любая строка символов, так как он используется как аргумент хэш-функции. Требования к длине ключа встраивания d определяются политикой безопасности организации. Ключ встраивания должен быть достаточно длинным, чтобы исключить полный перебор.

3.2.3 Извлечение сообщения

Извлечение сообщения выполняется на основе поиска соответствующих имен классов. Имена классов зависят только от ключа встраивания и порядкового номера блока. Ключ встраивания d известен принимающей стороне, поэтому при извлечении сообщения используется тот же самый алгоритм формирования имен классов, как и на принимающей стороне. Чтобы извлечь блок C_i необходимо выполнить следующие шаги:

1) Вычислить имя класса l_i символа с номером i с помощью алгоритма формирования имен для ключа встраивания d.

2) Найти в *HTML*-коде класс с именем l_i.

3) Для найденного класса выполнить обратное преобразование тегов по Таблице 1 и получить значение встроенных блоков C_i.

Порядок встраивания классов в пустые листовые вершины может быть произвольным, так как порядковый номер участвует в формировании имени класса символа. Для извлечения сообщения последовательно генерируются имена для классов символов. После чего, происходит поиск класса с нужным

именем и анализ его содержимого. Из содержимого класса извлекаются закодированные блоки сообщения и декодируются по таблице кодировки. Сообщение формируется как конкатенация полученных блоков. Порядок следования блоков соответствует порядку их извлечения из HTML-кода. Извлечение прекращается тогда, когда в документе не удается обнаружить класс с очередным именем. Полученное сообщение расшифровывается на ключе шифрования *k*.

$$M=D_k(C).$$

3.2.4 Пример встраивания сообщения

Рассмотрим пример формирования тегов для встраивания. Будем встраивать последовательность символов *M*=«*munko*» без шифрования, поэтому встраиваемая последовательность совпадает с исходной *C*=*M*=«*munko*». Цель этого примера продемонстрировать алгоритм встраивания. Используем таблицу кодировки ASCII. Запишем ASCII кодировку встраиваемой последовательности в шестнадцатеричной системе счисления посимвольно.

C= «6D 75 6E 6B 6F».

Разбиение на четырехбитные блоки будет иметь следующий вид:

C=«6 D 7 5 6 E 6 B 6 F».

Используя таблицу 1, получаем последовательность тегов.

t_1=«h5», t_2=«em», t_3=«h6», t_4=«h4», t_5=«h5», t_6=«i», t_7=«h5», t_8=«dl», t_9=«h5», t_{10}=«city».

В качестве ключа шифрования используем строку *d*=«shifr», а в качестве хэш-функции алгоритм SHA512. Вычислим метки тегов.

l_1=«*ng*-09-99», l_2=«*ng*-*d*6-*a*2», l_3=«*ng*-3*e*-85», l_4=«*ng*-*e*8-34», l_5=«*ng*-*ea*-54».

В данном случае должно быть сформировано пять классов, каждый из которых будет содержать два блока встроенной информации. Фрагмент кода представлен ниже.

```
<div class=” ng-09-99”>
    <h5 class=”mat-chevron-34”></h5>
    <em class=”mat-header-64”></em>
</div>
<div class=”ng-d6-a2” >
    <h6 class=”mat-typography_emGU”></h6>
    <h4 class=”mat-container_R4tF”></h4>
</div>
<div class=”ng-3e-85”>
    <h5 class=” portal-menu-placeholde_av4”></h5>
    <i class=” search-panel-button-sa45”></i>
</div>
<div class=” ng-e8-34” >
    <h5 class=” search-icon-placeholder_HgR5”></h5>
    <dl class=” search-icon--RfqH”></dl>
</div>
<div class=” ng-ea-54” >
    <h5 class=” search-panel-fghD”></h5>
    <city class=” portal-menu-F3g”></city>
</div>
```

При извлечении сообщения каждый из этих классов легко отыскивается в коде по имени. Присутствие этих классов никак не влияет на отображение страницы в браузере. Выделение классов, добавленных для встраивания сообщения, из общего множества классов, формируемых фреймворком *Angular*, на основе анализа кода страницы сводится к прямому перебору на множестве имен классов. Причем для каждого варианта набора имен классов необходимо выполнять взлом шифра. Прямой перебор необходим, так как при формировании оригинальных имен классов используются наборы чисел, формируемых фреймворком. Использование хэш-функции для синтеза имен встраиваемых классов не вносит существенного изменения в случайное распределение оригинальных имен классов, формируемых фреймворком *Angular*.

3.2.5 Программный комплекс встраивания скрытых данных в HTML-документ

Программный комплекс представляет собой приложение, реализованное на языке программирования JavaScript. Приложение включает два модуля. Первый модуль осуществляет встраивание скрытых данных, второй модуль извлекает скрытые данные. В программе реализован единый интерфейс, позволяющий вызывать оба модуля. Оба модуля в качестве параметров запрашивают имя файла-контейнера и ключ встраивания. Модуль встраивания запрашивает дополнительно строку символов, которая встраивается в файл-контейнер. Оба модуля используют метод *getAllClasses*().

Метод *getAllClasses*() в качестве параметра получает указатель на файл-контейнер формата *HTML*. Метод осуществляет проход по HTML-файлу и формирует массив имен всех классов, присутствующих в файле. Данный массив на этапе встраивания используется для проверки уникальности имен

классов. Встраивание скрытого сообщения основано на создании новых классов, которые не должны изменять отображение web-страницы в браузере. При создании новых имен программа автоматически генерирует имена классов. Возможна ситуация, при которой сгенерированное имя класса будет совпадать с именем класса, уже существующего в исходном документе. Вновь созданный класс с таким же именем приведет к искажению отображения документа. Поэтому каждое сгенерированное имя класса проверяется на присутствие в данном массиве имен.

Кроме этого, метод *getAllClasses*() осуществляет подсчет количества классов, не имеющих потомков. Эти классы являются листовыми в дереве классов и могут быть использованы для встраивания скрытого сообщения. Количество листовых вершин определяет максимальный объем сообщения, который может быть встроен в данный HTML-документ.

Обе задачи, решаемые данным методом, могут быть выполнены за один проход по документу, поэтому алгоритм обладает линейной трудоемкостью.

3.2.5.1 Модуль встраивания сообщения

При активизации режима встраивания сообщения вызывается метод *excriptionFile*(), который считывает значение полей из формы ввода. Результатом работы этого метода является массив имен классов исходного HTML-документа и массив символов встраиваемого сообщения. Блок-схема алгоритма встраивания сообщения представлена на рисунке 3.1

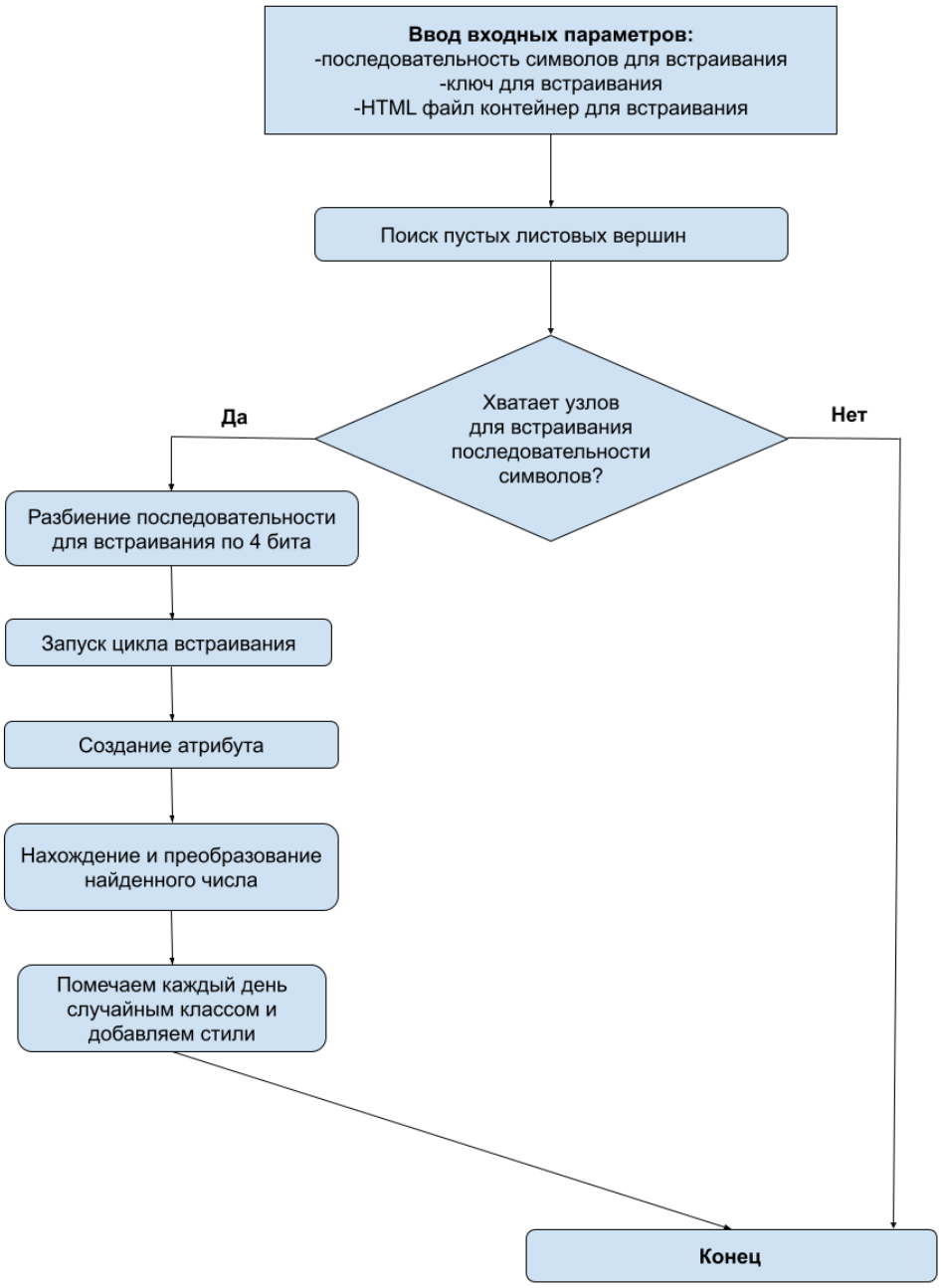

Рисунок 3.1 Блок-схема алгоритма встраивания сообщения

Встраивание сообщения осуществляется циклически. За один раунд цикла встраивается два символа сообщения. Для встраивания очередных двух символов осуществляется обход дерева тэгов HTML-файла (дом-дерева). При обнаружении очередного листового тэга формируются параметры добавляемого тэга и вызывается метод *encryptNodeTag*(), который осуществляет встраивание.

После того как, все символы встроены необходимо замаскировать вновь созданные классы. Для этого всем добавленным классам добавляются случайные стили. Основным требованием к новым стилям является отсутствие изменений в отображении исходного HTML-документа в браузере. Например, стиль *opacity*:1 отвечает за прозрачность, а стиль *background:none* отвечает за заливку. К документу добавляется тэг *style*, в который случайным образом добавляются эти стили для всех добавленных классов. Такое действие наполняет классы и не позволяет обнаруживать их при простом просмотре кода. Добавленные стили должны нести фиктивные стили документа, которые не влияют на его отображение в браузере. Однако добавление таких стилей позволяет защитить встроенный электронный цифровой знак от автоматического стегоанализа. Обнаружение классов с пустыми стилями требует интеллектуального анализа кода.

1) Метод *excriptionFile*() считывает пользовательские параметры и вычисляет параметры встраивания. Пользовательскими параметрами являются имя HTML-файла, ключ встраивания, последовательность символов встраиваемого сообщения. После этого этот метод вызывает метод *getAllClasses*() для получения массива имен классов файла-контейнера и количества листовых вершин. В одну листовую вершину дерева классов может быть встроено не более 8 бит информации. Произведение количества листовых вершин на 8 бит определяет максимальный доступный размер встраиваемого сообщения. Длина пользовательского сообщения сравнивается с максимально доступной длинной для определения возможности встраивания. Если сообщение пользователя превышает максимально доступный объем, то на экран выводится соответствующее сообщение: «Превышена максимально возможная длина встраиваемого сообщения» и программа ожидает ввода нового скрытого сообщения.

Если длина сообщения пользователя позволяет осуществить встраивание, то создается массив встраиваемых символов. Для этого

встраиваемая строка разбивается на блоки по 4 бита. Один такой блок считается одним встраиваемым символом. После этого управление передается модулю встраивания.

2) Метод *encryptNodeTag*() осуществляет встраивание символов в домашнее дерево. Методу передается четыре параметра: случайный пустой тэг с дочерними элементами, два встраиваемых символа, индекс встраиваемых символов и ключ встраивания. Формирование параметров метода *encryptNodeTag*() осуществляется в основной программе.

Индекс символа ($index$) представляет собой его номер во встраиваемой последовательности. На основе индекса символа $index$ и ключа встраивания key формируется имя класса. Первым этапом формирования имени класса является вычисление хэш-функции от конкатенации индекса символа и ключа встраивания.

$$shaKey[index] = sha512(key \;||\; index).$$

Новое имя класса состоит из префикса ng и строки символов, извлеченных из хэш-значения.

$$name[i] = ng - shaKey.substring(0,2) - shaKey.substring(2,4).$$

Такой формат имен классов использует фреймворк *Angula*r. Формальная проверка не сможет отличить классы, созданные для встраивания от классов, сформированных фреймворком. Этот класс встраивается в родительский пустой тэг.

Встраиваемые символы представляются в шестнадцатеричной системе счисления. В таком представлении символ занимает 4 бита. По таблице кодировки символу сопоставляются определенные тэги. В тэг добавляется случайно сгенерированный класс. Присутствие случайного класса усложняет

стегоанализ документа. Имя класса генерируется на основании таблицы имен классов, которая создается заранее и используется программой встраивания. Выбирается первое случайное наименование класса. К этому имени добавляется случайная последовательность символов. Полученное имя класса проверяется на уникальность путем сравнения с именами уже существующих в файле-контейнере классов. Таблица существующих классов была ранее создана методом *getAllClasses*(). Если новое имя уникально, то оно добавляется к таблице классов документа. Если произошло совпадение, то генерируется новое имя класса. После этого полученный тэг с классом добавляется в качестве дочернего в пустой родительский тэг.

3.2.5.2 Модуль извлечения сообщения

Модуль извлечения сообщения в качестве параметров получает имя HTML-файла со встроенным сообщением и ключ встраивания. Блок-схема алгоритма извлечения сообщения представлена на рисунке 3.2.

После запуска процесса извлечения цифрового водяного знака вызывается метод *Decryption*(), который возвращает список встроенных тэгов в порядке следования символов встроенного сообщения.

Формирование встроенного сообщение осуществляется проходом по списку встроенных тэгов. У каждого тэга есть два дочерних узла. В каждом из этих узлов зашифровано по 4 бита из встроенного сообщения. По таблице кодировки восстанавливаются шестнадцатеричные коды символов. Конкатенация этих символов дает встроенное сообщение.

Метод *Decryptio*n() считывает файл и создает пустой список для найденных узлов. Количество встроенных символов заранее неизвестно. Поэтому поиск символов осуществляется в бесконечном цикле. Алгоритм осуществляет выход из цикла, если не удается найти очередной встроенный символ. Счетчик встроенных символов инициализируется нулевым

значением ($index = 0$). После каждого найденного символа счетчик увеличивается на единицу ($index + +$). Счетчик используется как индекс встроенных символов. После этого формируется имя класса по известному ключу встраивания и индексу символа.

$$shaKey[index] = sha512(key \ || \ index).$$

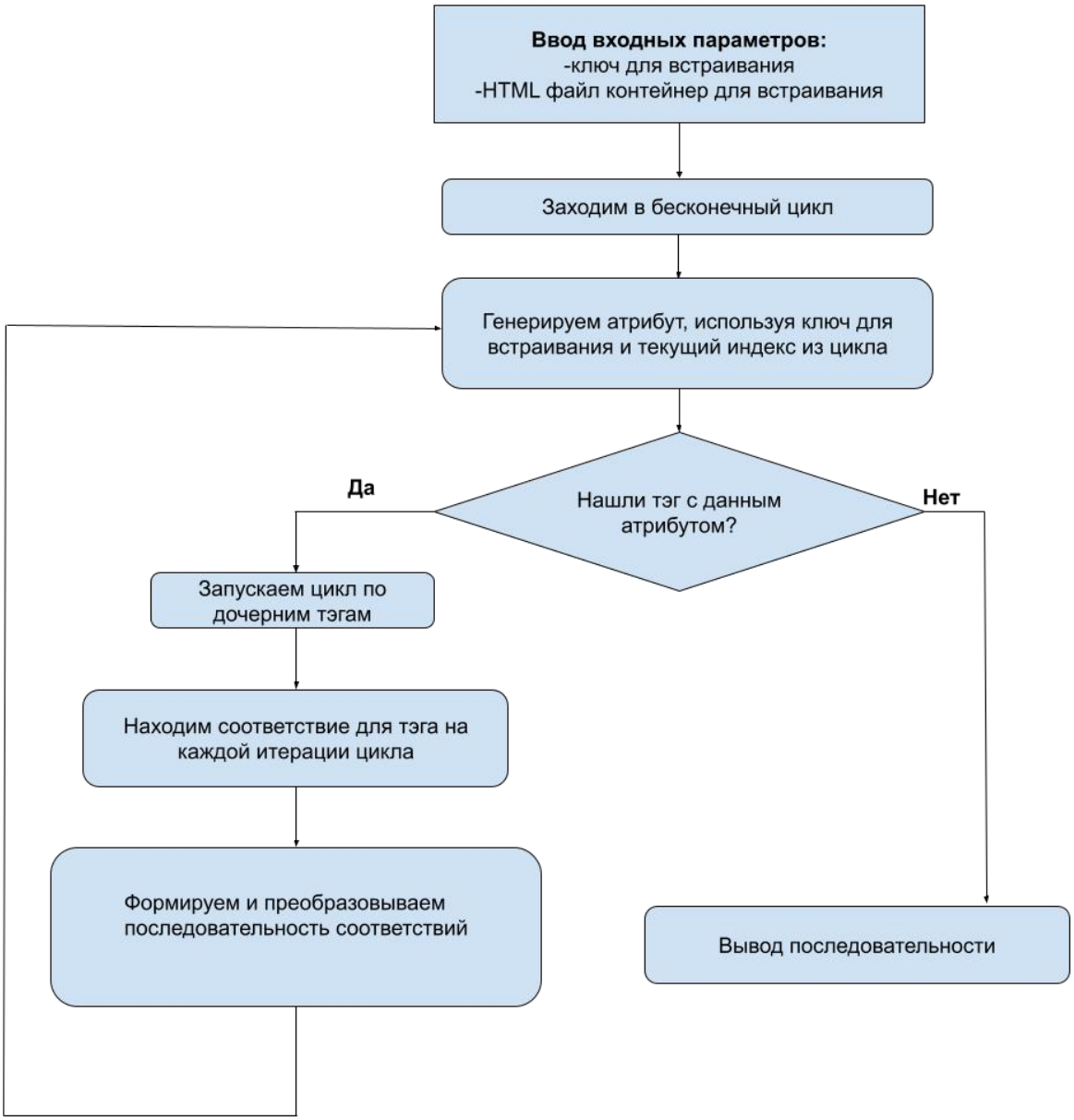

Рисунок 3.2 Блок-схема алгоритма извлечения сообщения

На основе этого хэш-значения формируется и префикса ng формируется имя класса.

$$name[i] = ng - \ shaKey.substring(0,2) - \ shaKey.substring(2,4).$$

Далее программа осуществляет поиск в HTML-документе класса с таким именем. Если класс с именем $name[i]$ не найден, то происходит выход из цикла и процесс извлечения цифрового водяного знака завершен. Если класс с именем $name[i]$ обнаружен в документе, то соответствующий тэг добавляется в список найденных узлов. Такой алгоритм обеспечивает правильный порядок поиска и добавления в список тэгов с встроенными символами.

3.3 Встраивание скрытых данных в изображения формата SVG

3.3.1 Методы встраивания и извлечения данных

Будем использовать ключевую стеганографическую схему. Отправитель должен обладать правами администрирования web-страницы, содержащей *SVG*-изображение. Отправитель модифицирует представление изображения, внося в него скрытую информацию. Модификация осуществляется таким образом, чтобы отображение изображения на *web*-странице оставалось неизменным. Это является необходимым требованием. Получатель, для извлечения скрытых данных, должен обладать информацией о том, в каком изображении выполнено встраивание, а также ключом встраивания. Схема должна быть реализована так, чтобы без знания ключа невозможно было извлечь скрытые данные даже, если известно в каком изображении оно встроено. Передача данных может осуществляться как однократно, так и многократно. Если скрытый канал используется

многократно, то для обеих сторон требуется дополнительная ключевая информация, связанная с моментами времени, в которые происходит обновление данных и время, в течение которого скрытые данные остаются актуальными. То есть, принимающая и передающая стороны должны предварительно договориться о временных окнах передачи сообщений. Многократный канал требует полной неизменности видимого представления изображения. В противном случае стегоаналитик может провести сравнение внешнего вида картинки в разные моменты времени и обнаружить факт встраивания.

Векторная графика формата *SVG* имеет преимущество перед другими форматами, связанное с использованием языка разметки, основанного на XML. Поэтому *SVG*-изображения могут непосредственно вставляться в код HTML-документа и быть его неотъемлемой частью. Такое встраивание осуществляется с помощью тэга <*svg*>. Прямое встраивание *SVG*-рисунка дает дополнительные возможности управления его отображением на *web*-странице. Появляется возможность изменять цвет, размер и даже анимировать изображение через *CSS*-стили. Для этого в тегах <*svg*> предусмотрена возможность использования атрибута *class* с указанием имени класса. Также классы со стилями могут быть указаны в тэге раздела <*div*>, в который помещается *SVG*-код. В обоих этих случаях стили класса будут применяться ко всему изображению в целом. Существует также возможность связывания классов стилей с отдельными деталями изображения. Эта возможность реализуется добавлением имени класса в атрибуты тэга <*path*>.

Будем маскировать встраиваемую информацию, используя *CSS*-стили управления *SVG*-изображением. Для этого будем изменять атрибуты тэгов, связанных с картинкой. Незаметность встраивания можно обеспечить добавлением классов, стили которых никак не влияют на вид изображения. Для корректного встраивания необходимо сформировать метку, указывающую на то, что в данный тэг проведено встраивание. Эта метка

должна быть неотличимой от штатной информации, чтобы ее не смог выделить аналитик. С другой стороны она должна нести информацию о блоке встроенной информации для принимающей стороны. Стандарт описания тэгов формата *SVG* допускает встраивание произвольных данных только в два атрибута. Первый атрибут *id* задает идентификатор фрагмента изображения. Второй атрибут *class* позволяет программисту создать класс стилей и применить их к фрагменту изображения, описанному внутри тэга <*path*>. Изменение остальных атрибутов явно влияет на отображаемую картинку. Таким образом, встраивание возможно только в тэги <*path*>, не содержащие идентификаторы.

Блок встраиваемой информации состоит из двух частей: метки встраивания и скрытых данных. Все встраиваемое сообщение M представим в виде последовательности блоков m_i (i=1,…,n).

$$M=m_1m_2\ldots m_n.$$

Для встраивания формируются пары, состоящие из метки блока l_i (i=1,…,n) и закодированного блока $c_i=C(m_i)$ (i=1,…,n). Где $C()$ некоторая функция кодирования сообщения. Функции кодирования должна сопоставляться функция декодирования $B()$, выполняющая обратное преобразование.

$$m_i=B(c_i)\ \ (i=1,\ldots,n).$$

Встраивание происходит в атрибуты тэга <path>.

<*path id*=”l_i” *class*=”c_i”…>.

Метка блока l_i (i=1,…,n) формируется на основе ключа встраивания k_1 и номера блока i с помощью некоторой функции $L()$.

$$l_i=L(k_1,i)\ (i=1,\ldots,n).$$

Такой подход позволяет встраивать блоки скрытого сообщения в произвольном порядке. Для поддержки легитимного вида встроенного сообщения необходимо в разделе стилей HTML-документа создать класс с именем c_i. Стили этого класса должны быть определены таким образом, чтобы не изменять конечный вид изображения в браузере. Необходимо определить четыре алгоритма:

1) Ключевой алгоритм формирования меток $L()$.

2) Алгоритм кодирования сообщения $C()$ и формирования имен классов c_i.

3) Алгоритм декодирования сообщений $B()$, позволяющий получать по именам классов c_i исходные блоки сообщений m_i.

4) Алгоритм формирования описания класса с именем c_i, не изменяющих вид изображения.

Если описанные выше алгоритмы определены, то встраивание сообщения состоит из следующих шагов:

1) Исходное сообщение представляется в виде последовательности блоков m_i $(i=1,\ldots,n)$.

2) Блоки сообщения кодируются и формируются имена встраиваемых классов $c_i=C(m_i)$ $(i=1,\ldots,n)$.

3) На основе ключа встраивания k_1 и номера блока i формируется метка блока $l_i=L(k_1,i)$ $(i=1,\ldots,n)$.

4) В *HTML*-документе осуществляется поиск SVG-изображений, у которых тэги <*path*> не содержат определенного атрибута *id*. Количество таких тэгов определяет объем информации, который может быть встроен.

5) В найденные тэги в случайном порядке добавляется определение атрибутов: *id*=”*li*” *class*=”*ci*”.

6) В разделе описания стилей HTML-документа добавляется описание классов с именами c_i (i=1,…,n).

Алгоритм извлечения сообщения состоит из шагов аналогичных встраиванию.

1) Получатель осуществляет перебор номеров блоков i начиная с нуля.

2) Для каждого номера блока на основе ключа встраивания k_1 вычисляется метка $l_i=L(k_1,i)$. Ключ встраивания k_1 известен только отправителю и получателю. Алгоритм формирования метки является открытым.

3) Для каждой метки l_i осуществляется поиск тэга <*path*> содержащего атрибут *id*=”l_i”.

4) Если такой тэг найден, то считывается следующий за ним атрибут class, из которого извлекается имя класса c_i. Если такой блок не найден, то сообщение извлечено полностью и алгоритм заканчивает работу.

5) Полученное имя декодируется, что позволяет получить блок сообщения ($m_i=B(c_i)$).

6) Полное сообщение формируется, как конкатенация извлеченных блоков m_i.

Для каждого блока сообщения осуществляется поиск метки по всему HTML-документу. Это позволяет размещать встраиваемые блоки в документ в произвольном порядке, что дополнительно запутывает взломщика. Тэги <*path*> могут содержать свои классы, определяющие свойства изображения. Для однозначности извлечения встраиваемый класс размещается так, чтобы он был первым после атрибута *id*. Также встраиваемый класс не должен противоречить имеющимся классам. Это требование накладывает ограничения на алгоритм формирования описания классов.

3.3.2 Описание алгоритмов

Для формирования меток используем ключевую схему на основе хэш-функций. Пусть ключ встраивания равен k_1. Используем некоторую хэш-функцию $h(x)$. Для формирования метки m_i встроенного символа с номером i, вычисляем хэш-функцию от конкатенации ключа встраивания и номера метки $h(k_1||i)$. Хэш-значение имеет длину, определяемую видом хэш-функции. Для современных хэш-функций эта длина может составлять от 256 до 512 бит. Представим хэш-значение в виде последовательности символов, разбивая последовательность нулей и единиц на блоки по 8 бит и используя кодировку ASCII. В этом случае хэш-значение представляется в виде последовательности букв и цифр. С помощью последовательного перебора символов хэш-значения отыскиваем первые две буквы и первые две цифры. После этого переносим в метку m_i эти буквы и цифры. Поиск букв необходим для того, чтобы метка удовлетворяла требованиям к правильным идентификаторам.

Рассмотрим пример формирования метки. Пусть ключ встраивания имеет значение k_1= «test». Сформируем метку для символа с номером i=0. В качестве хэш-функции $h(x)$ используем алгоритм SHA-512.

h(“test0”)=«**a45**89**a**e194a3eeeb99409b2287cc8ce5ae2fec1929935f2d0efc58606eb2575d3d036b7b116d360ce9325387df9014c2c5ea5273c8086de4d3454b634a4232a4». Метка будет иметь вид m_0=«aa45». Аналогичные вычисления для следующей метки на том же ключе дают значение m_1=«be16».

Кодирование встраиваемого сообщения должно выполняться таким образом, чтобы его внешний вид был неотличим от обычных имен классов. Самый простой подход состоит в использовании шифрования каким-либо симметричным шифром с последующим применением таблицы замен. Ключ шифрования k_2 должен быть известен как отправителю, так и получателю.

Пусть используется симметричный шифр $E_{k2}()$. Алгоритм расшифрования обозначим $D_{k2}()$. Исходный текст A шифруется на ключе k_2.

$$M=E_{k2}(A).$$

Полученное сообщение M разбивается на блоки по 9 бит, что соответствует трем восьмеричным числам.

$$m_i=m_{i1}m_{i2}m_{i3}.$$

Для имени класса используем слова, наиболее часто встречающиеся в подобных классах и обозначающие действия над изображением. Каждой восьмеричной цифре сопоставляем одно слово по табл. 1. Отображение, задаваемое таблицей, обозначим через $T()$. Объединяем три слова в одно имя.

$$c_i=C(m_i)= T(m_{i1})||"-"||T(m_{i2})||"-"||T(m_{i3}).$$

Таблица 1. Кодирование цифр сообщения

Код символа	$T()$
0	blue
1	center
2	right
3	gradient
4	floating
5	flex
6	block
7	app

Рассмотрим пример. Пусть блок сообщения m_i=155. Кодирование по таблице 1 дает имя класса c_i=«center-flex-flex».

При извлечении встроенного сообщения получатель выполняет обратные действия. Сначала происходит декодирование отдельных блоков $m_i=T^{-1}(c_i)$ по таблице 1, после чего все блоки конкатенируются в общее сообщение M. Расшифровывая сообщение на ключе k_2, получатель получает исходное сообщение $A=D_{k2}(M)$. Приведем пример SVG-изображения со встроенным сообщением.

<SVG>

<path fill-rule="evenodd" clip-rule="evenodd" **id="aa45" class="center-flex-flex"** d="M3.25 5C3.25 4.58579 3.58579 4.25 4 4.25H16C16.4142 4.25 16.75 4.58579 16.75 5V6.74161C17.9549 7.28388 18.75 8.57138 18.75 10C18.75 11.4286 17.9549 12.7161 16.75 13.2584V15C16.75 15.4142 16.4142 15.75 16 15.75H4C3.58579 15.75 3.25 15.4142 3.25 15V13.2584C2.04515 12.7161 1.25 11.4286 1.25 10C1.25 8.57138 2.04515 7.28388 3.25 6.74161V5ZM4.75 5.75V7.87747L4.16501 8.00941C3.40858 8.18001 2.75 8.96111 2.75 10C2.75 11.0389 3.40858 11.82 4.16501 11.9906L4.75 12.1225V14.25H15.25V12.1225L15.835 11.9906C16.5914 11.82 17.25 11.0389 17.25 10C17.25 8.96111 16.5914 8.18001 15.835 8.00941L15.25 7.87747V5.75H4.75Z" ></path>

</SVG>

Каждый используемый класс должен быть описан. В противном случае аналитик сможет в автоматическом режиме обнаружить классы без описания и выявить встроенное сообщение. Поэтому в разделе стилей необходимо добавлять описание каждого создаваемого класса. Основное требование состоит в том, чтобы стили класса никак не сказывались на отображении рисунка. Необходимо рассмотреть два случая. Первый случай состоит в том, что у тэга <*path*> в исходном коде могут отсутствовать подключаемые классы. В этом случае применяются стили по умолчанию. Чтобы создаваемый класс не изменял изображение необходимо в его описание

добавить описание стилей со значениями по умолчанию. Второй случай состоит в том, что в исходном коде к тэгу <*path*> уже добавлено описание некоторого класса. В этом случае простое добавление классов по умолчанию может изменить отображение рисунка. Поэтому необходимо найти соответствующий класс и скопировать из него стили. Некоторые из этих стилей случайным образом удаляются. Также в класс добавляются описания стилей, отсутствующих в исходном классе, со значениями по умолчанию. В итоге описание классов отличается и не может быть обнаружено автоматически.

3.3.3 Возможные атаки на систему

Рассмотрим возможные атаки на предложенную схему. Ключ шифрования k_2 не может быть раскрыт при использовании стойкого алгоритма шифрования. Для раскрытия ключа k_1 требуется взлом хэш-функции, что также является задачей трудоемкой. Таблица кодирования $T()$ может быть восстановлена прямым перебором, так как ее размер достаточно мал, но для этого необходимо получить хотя бы одно исходное сообщение и его закодированную форму. Исходное сообщение шифруется, поэтому прямой перебор будет давать некоторые двоичные последовательности. Без знания ключа шифрования k_2 невозможно выделить истинную исходную последовательность из этого множества. Поэтому извлечь исходное сообщение невозможно.

Рассмотрим возможность обнаружения самого факта встраивания в автоматическом режиме. Важна стойкость схемы именно к автоматическому анализу контейнера. Любые стеганографические алгоритмы могут быть раскрыты с привлечением интеллектуального анализа человеком. Даже хорошо зарекомендовавший себя метод наименее значащего бита может быть раскрыт, если изображение будет послойно анализироваться человеком

[70]. Тогда как создание алгоритма, выполняющего те же требования автоматически, сталкивается с большими трудностями. Будем исходить из предположения, что аналитику известна схема встраивания, но не известен ключ встраивания. Для современных сайтов необходим автоматический анализ страниц *web*-сайта. Статистический анализ показал, что отдельные *web* -сайты, связанные с продажей товаров, могут содержать более 200 *SVG*-изображений, с общим количеством тэгов <*path*> более 400. Для автоматического анализа предложенной схемы встраивания в *SVG*-изображения необходимо проанализировать атрибуты *id* и *class* во всех тэгах <*path*>. Все тэги, содержащие эти два атрибута одновременно, являются потенциально подозрительными. Однако полный перебор не может быть организован, так как порядок встраивания блоков сообщения не совпадает с порядком следования тэгов в HTML-коде. Единственный способ состоит в анализе описания классов в разделе стилей. Однако он не может быть выполнен в автоматическом режиме, так как требует проверки применения стилей при выполнении HTML-кода. Следует отметить, что задача аналитика существенно осложняется программным обеспечением, применяемым при разработке HTML-страниц, которое генерирует имена классов и идентификаторы тэгов по своим алгоритмам. Задача аналитика может быть еще более усложнена, если после встраивания всем не использованным при встраивании тэгам <*path*> случайным образом добавить идентификаторы и имена классов, созданные генератором случайных чисел. Такое добавление не значительно увеличит размер документа и никак не скажется на его отображении в браузере.

3.3.4 Программный комплекс встраивания данных в SVG-изображения

Программный комплекс реализован в виде web-приложения, состоящего из двух методов. Первый метод (*encriptionFile*()) осуществляет

встраивание сообщения в SVG-изображение, размещенное на HTML-странице. Второй метод (*fileDecription*()) выполняет извлечение встроенного сообщения. В программе реализован единый интерфейс для запуска обоих методов. Каждому методу соответствует своя вкладка.

3.3.4.1 Встраивание сообщения

Для встраивания сообщения у пользователя запрашивается имя *HTML*-файла, текст сообщения и ключ встраивания. После запуска процесса встраивания вызывается метод *encriptionFile*(). На первом этапе данный метод осуществляет рекурсивный обход HTML-документа и поиск дочерних *svg*-тэгов *path*. В результате обхода формируется список тэгов, не содержащих атрибута *id*. В эти тэги будет производится встраивание информации.

Основной цикл метода осуществляет проход по тексту сообщения и посимвольное встраивание ее в тэги *path*. Для каждого символа случайным образом выбирается пустой тэг *path*. Следует учитывать, что список пустых тэгов *path* уменьшается после встраивания каждого символа текстового сообщения. По номеру символа i в текстовом сообщении и ключу встраивания k формируется метка тэга li. Для формирования метки используется алгоритм хэширования *sha*512. Данный алгоритм хэширования формирует строку длиной 512 бит. Хэш-значение может быть представлено в виде последовательности символов в ASCII-кодировке. Для формирования идентификатора li выбирается первые четыре символа из этой строки.

$$l_i = sha512(k||i).substring(0,4).$$

После этого к пустому тэгу *path* добавляется атрибут *id=li*.

Для каждого символа используется восьмеричное представление его *ASCII*-кода. Каждому символу сопоставляется класс, имя которого формируется по таблице кодирования. Каждой восьмеричной цифре сопоставляется одна символьная строка. Код символа состоит из нескольких цифр. Символьные строки, соответствующие различным цифрам разделены дефисом. Итогом является строка имени класса *ci*. В тэг *path* добавляется атрибут *class=ci*.

Класс с именем *ci* также добавляется в раздел описания стилей. При этом классу случайным образом добавляются стили, не влияющие на отображение документа в браузере.

В результате работы метода в тэгах *path*, которые в исходном документе присутствовали без атрибутов, появляется два новых атрибута.

Блок-схема алгоритма представлена на рисунке 3.3.

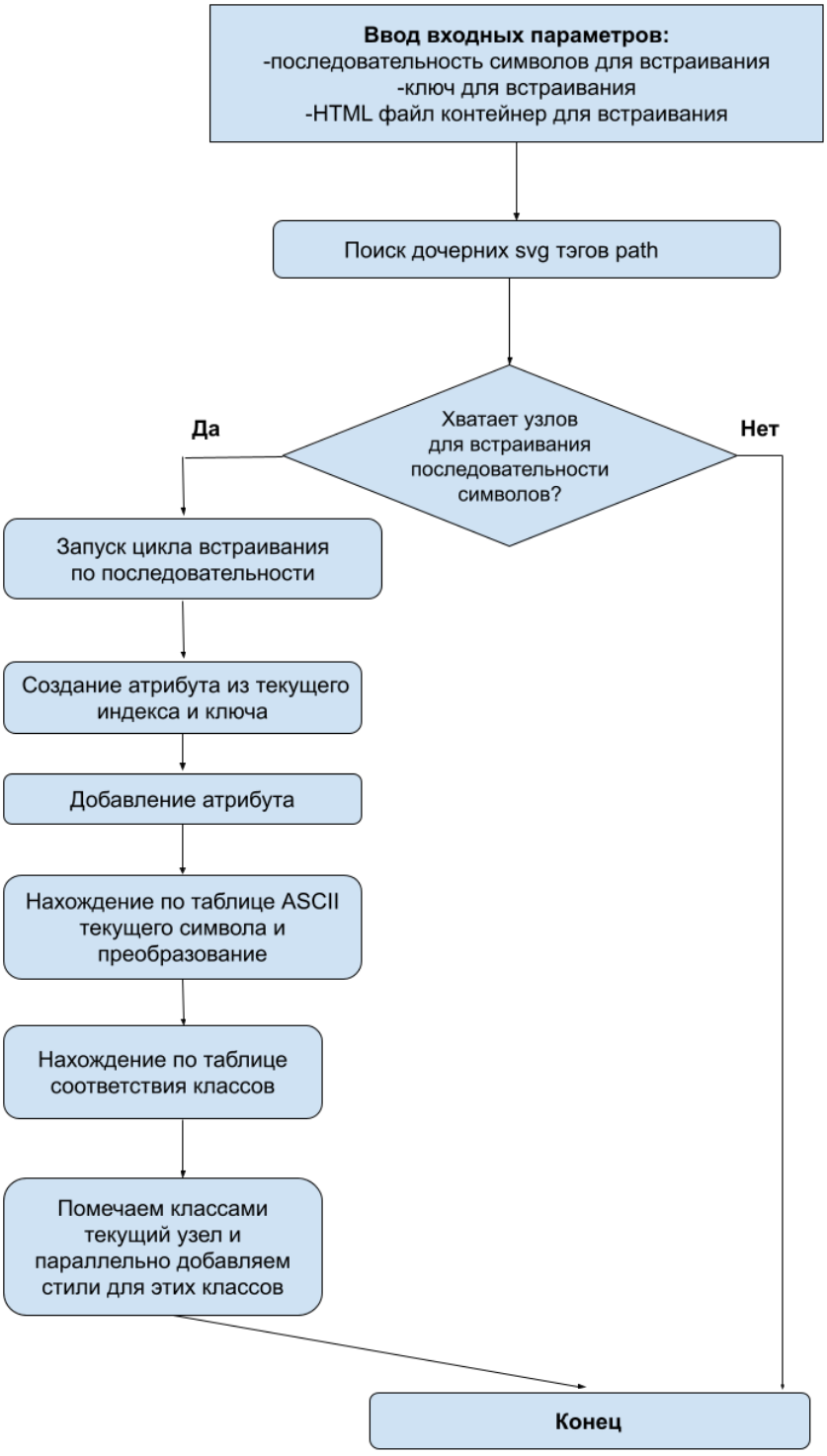

Рисунок 3.3 Блок-схема алгоритма встраивания сообщения в SVG-изображение.

3.3.4.2 Извлечение сообщения

Для запуска извлечения сообщения программа запрашивает имя HTML-файла и ключ встраивания. После этого запускается метод *fileDecription*(). Поиск символов встроенного сообщения выполняется в бесконечном цикле, так как количество встроенных символов неизвестно. Перед запуском цикла счетчик символов i инициализируется нулевым значением. В каждой итерации цикла счетчик увеличивается на единицу. На основе счетчика i и ключа встраивания k вычисляется идентификатор символа li.

$$l_i = sha512(k||i).substring(0,4).$$

После этого программа осуществляет обход документа и поиск тэга path, имеющего атрибут *id=li*. После того как тэг найден, из него извлекается имя класса. По имени класса восстанавливается восьмеричный код символа с применением обратной таблицы кодирования. Полученная последовательность является *ASCII*-кодом очередного символа.

Цикл выполняется до тех пор, пока программа находит в *HTML*-файле очередной тэг с заданным идентификатором. Если очередной идентификатор найти не удается, происходит выход из цикла и встроенная строка считается прочитанной.

Блок-схема алгоритма извлечения встроенной строки представлена на рисунке 3.4

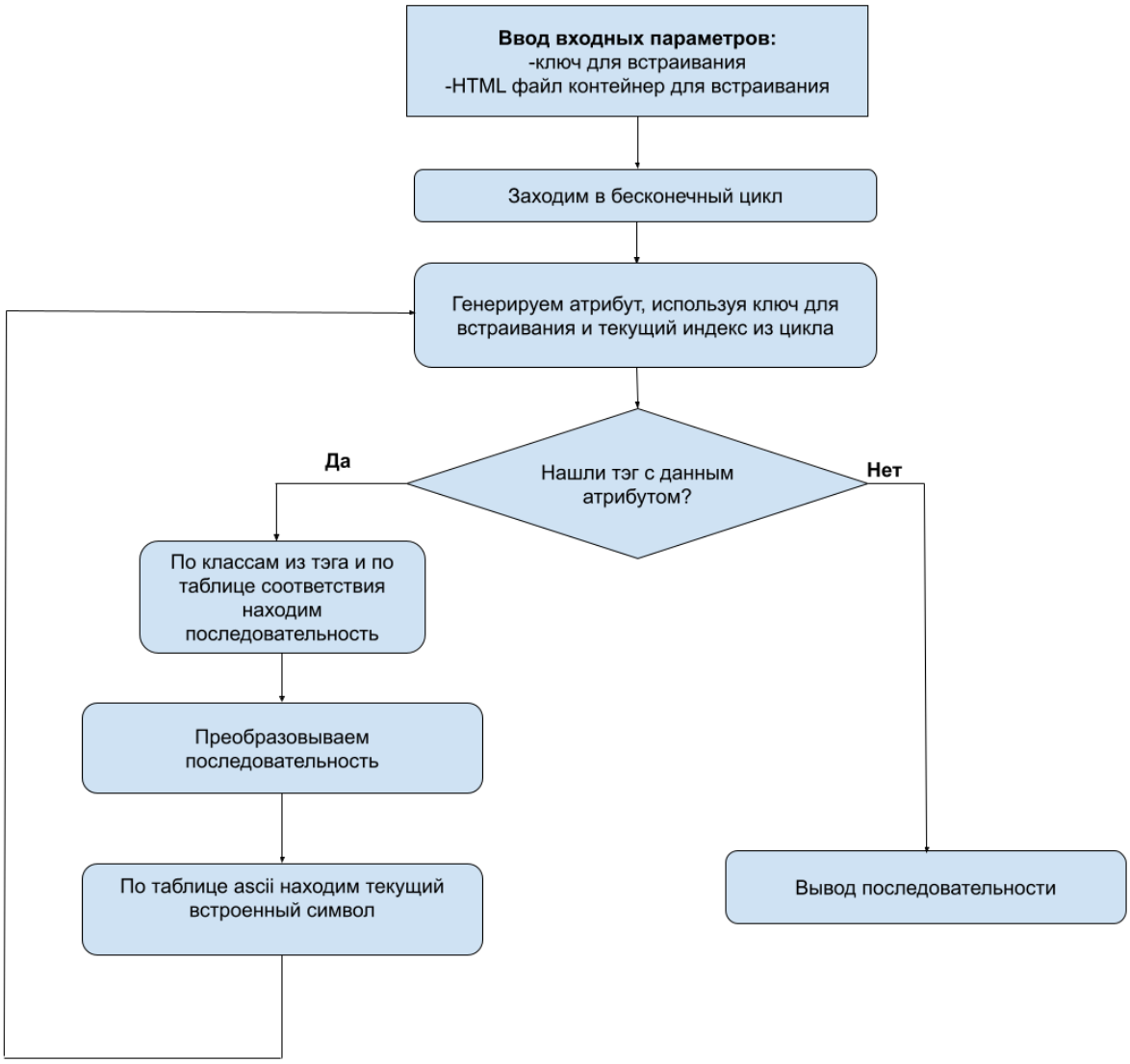

Рисунок 3.4 Блок-схема алгоритма извлечения встроенной строки.

3.4 Выводы

Предложенная схема [10] позволяет использовать код *HTML*-код документа для стеганографического встраивания информации. Объем данных, которые могут быть встроены, зависит от объема *HTML*-страницы. Статистический анализ 100 случайно выбранных сайтов показал, что количество свободных листовых вершин составляет не менее 16% от общего

количества тегов. В среднем современная *HTML*-страница содержит не менее 700 тегов, что соответствует не менее 112 свободных листовых вершин. В такую страницу может быть встроено сообщение из 112 байт. Учитывая размеры сайтов, средняя пропускная способность составляет 0,003 бит на байт, что является приемлемым для современных стеганографических систем. Если необходимо передать сообщение большого объема, то может быть задействовано несколько web-сайтов. На каждом из этих сайтов размещается часть сообщения, в конце которого ставиться гиперссылка на следующий блок.

В предложенном алгоритме формирование имен классов в тегах маскируется под некоторый фреймворк. В общем случае можно проводить анализ имен классов на конкретной странице. В качестве первой составляющей имени классов символов может использоваться имя, наиболее часто встречающееся на странице, в которую производится встраивание.

Предложенный алгоритм является открытым, что не влияет на его стойкость. Стойкость обеспечивается секретностью ключей шифрования и встраивания. Без знания ключа встраивания выделение тегов, добавленных при встраивании сообщения, от тегов, присутствующих в коде *HTML*-страницы изначально, сводится к прямому перебору. Имена классов, добавляемых при встраивании сообщения, формируются на основе ключа встраивания и хэш-функции. Алгоритм формирования имен классов выбран таким образом, чтобы не нарушать статистических закономерностей множества легальных имен классов. Таким образом, сам алгоритм не требует засекречивания. Стойкость обеспечивается политикой хранения ключей, что является общим требованием ко всем системам защиты информации.

Предложенный алгоритм может использоваться и без шифрования, что формально снизит его стойкость. В этом случае возможна атака на систему, основанная на знании таблицы кодирования и переборе различных вариантов из декодированных последовательностей, полученных для листовых вершин.

Однако этот алгоритм полного перебора характеризуется экспоненциальной трудоемкостью и для достаточно больших web-страниц не может быть выполнен за приемлемое время.

Рассмотрение *SVG*-изображения в контексте *HTML*-документа, в который он встроен, обобщает разработанный подход к стеганографии с использованием *web*-страниц [11]. Этот подход позволяет использовать атрибуты изображения для встраивания данных. Кроме этого, появляется возможность размещения дополнительной информации в самом *HTM*L-документе, дополнительно маскирующей факт скрытого встраивания информации. Выбранный подход никак не ограничивает возможности использования *SVG*-изображений для скрытой передачи информации, так как этот формат разработан для представления графической информации на *HTML*-страницах и использует язык разметки на основе стандарта *XML*.

3.5 Внедрение результатов

Алгоритм встраивания стеганографических вставок в *web*-страницы был внедрен в деятельность ООО «Лет ИТ БИ». Данная компания занимается разработкой *web*-страниц и связанного с ними программного обеспечения. Разработка осуществляется в рамках большого проекта совместно с компаниями, расположенными в других странах (Узбекистан, Турция, Кипр). Каждая компания отвечает за свой сегмент совместного проекта.

Стеганографические вставки в *web*-страницы использовались для подтверждения метаданных, связанных с изменениями в проекте. При внесении изменений в проект одним из участников, он направляет уведомление остальным участникам проекта. Одновременно в *web*-страницу проекта внедряется скрытая информация, содержащая мета-данные, позволяющие идентифицировать изменения. Таким образом стеганографическая вставка является подтверждением сделанных изменений.

Встраиваемые данные содержат информацию:

1) дата внесения изменений,

2) код участника, внесшего изменения,

3) код модуля, в который внесены изменения,

4) комментарии.

Такой подход к совместной работе над проектом позволяет решать следующие задачи:

1) Контролировать внесение изменений в совместный проект. Уведомление о внесении изменений может высылаться заранее. В уведомлении указывается, когда и какие изменения вступят в силу. Встроенная информация позволяет подтверждать, что изменения произошли.

2) Аутентифицировать сторону, внесшую изменению. Подтверждением подлинности автора изменения является код участника. Так как схема использует шифрование данных, то чтение и подмена кода участника злоумышленником является трудоемкой.

3) Контроль версий при архивировании. В целях безопасности происходит периодическое сохранение всего проекта на сервере. Встроенная информация позволяет контролировать и восстанавливать информацию о сохраненной версии проекта.

Предложенная схема встраивания скрытой информации в *web*-страницу расширяет дерево классов. Это свойство предложенного алгоритма позволяет осуществлять встраивание данных нескольким участникам без возникновения коллизий. Скрытые данные одного участника не влияют на данные другого участника.

3.6. Результаты

Сформулируем основные результаты данной главы:

1) Предложен алгоритм встраивания и извлечения скрытых данных в *HTML*-страницу на основе расширения дерева классов. Предложенный алгоритм создает дополнительные листовые вершины. Новые классы не влияют на отображение web-страницы в браузере. Имена классов выбираются таким образом, чтобы соответствовать общему стилю имен классов, генерируемых автоматически. Такой выбор имен классов делает невозможным автоматический стегоанализ кода *HTML*-страницы.

2) Предложенная схема использует два ключа. Первый ключ – это ключ шифрования, необходимый для выравнивания статистических данных встраиваемого сообщения. Второй ключ – это ключ встраивания, необходимый для упорядочивания блоков сообщения, распределенных по *HTML*-странице случайным образом. Извлечение сообщения возможно только при знании обоих ключей.

3) Реализован программный комплекс встраивания данных в *HTML*-страницу и последующего извлечения данных.

4) Предложен алгоритм встраивания скрытых данных в *SVG*-изображение, размещенное на *web*-странице. Алгоритм использует стилевые классы, определяющие параметры отображения векторной графики. Встроенное сообщение кодируется с помощью дополнительных атрибутов. Для затруднения стегоанализа расширяется список стилевых классов.

5) Предложенная стеганографическая схема является ключевой. Ключ встраивания определяет имена добавленных классов и их порядок при извлечении сообщения.

6) Реализован программный комплекс встраивания данных *SVG*-изображение на *HTML*-странице.

Заключение

В заключении приведем основные результаты.

1. Предложен метод встраивания цифрового меток безопасности в память исполняемой программы с помощью подключения динамической библиотеки аутентификации. При этом:

1.1. Предложен алгоритм встраивания и извлечения метки безопасности в память исполняемой программы с использованием динамической библиотеки аутентификации.

1.2. Подключение внешней динамической библиотекой позволяет работать с памятью процесса для встраивания и извлечения скрытой информации.

1.3. Использование внешней библиотеки аутентификации для формирования метки безопасности позволяет управлять его временными характеристиками, такими как момент времени формирования и длительность существования. Такой подход усложняет стеганографический анализ с помощью динамического анализа кода и повышает устойчивость стеганографической вставки к внешним атакам.

1.4. Для встраивания и извлечения цифрового водяного знака применяются парольные фразы, что делает стеганографическую систему ключевой.

1.5. Метка безопасности записывается в память в распределенном виде, что существенно усложняет его поиск и выделение с помощью анализа памяти.

1.6. Выполнена программная реализация библиотеки аутентификации с модулем встраивания метки безопасности, которая может быть использована в любом приложении.

2. Предложены методы встраивания скрытых данных в открытый код *web*-странице и объекты векторной графики на *web*-странице на основе модификации иерархии классов. При этом:

2.1. Предложен алгоритм встраивания и извлечения скрытых данных в *HTML*-страницу на основе расширения дерева классов. Предложенный алгоритм создает дополнительные листовые вершины. Новые классы не влияют на отображение *web*-страницы в браузере. Имена классов выбираются таким образом, чтобы соответствовать общему стилю имен классов, генерируемых автоматически. Такой выбор имен классов делает невозможным автоматический стегоанализ кода *HTML*-страницы.

2.2. Предложенная схема использует два ключа. Первый ключ – это ключ шифрования, необходимый для выравнивания статистических данных встраиваемого сообщения. Второй ключ – это ключ встраивания, необходимый для упорядочивания блоков сообщения, распределенных по HTML-странице случайным образом. Извлечение сообщения возможно только при знании обоих ключей.

2.3. Реализован программный комплекс встраивания данных в *HTML*-страницу и последующего извлечения данных.

2.4. Предложен алгоритм встраивания скрытых данных в *SVG*-изображение, размещенное на *web*-странице. Алгоритм использует стилевые классы, определяющие параметры отображения векторной графики. Встроенное сообщение кодируется с помощью дополнительных атрибутов. Для затруднения стегоанализа расширяется список стилевых классов.

2.5. Предложенная стеганографическая схема является ключевой. Ключ встраивания определяет имена добавленных классов и их порядок при извлечении сообщения.

2.6. Реализован программный комплекс встраивания данных *SVG*-изображение на *HTML*-странице.

Публикации автора по теме диссертации

Публикации в журналах из списка ВАК:

А1. Белим С.В., Мунько С.Н. Стеганографическое встраивание данных в код HTML-документа / С.В. Белим, С.Н. Мунько // Вестник компьютерных и информационных технологий. – №11. – 2022. – С. 37-44.

А2. Белим С.В., Мунько С.Н. Алгоритм встраивания цифрового водяного знака в динамическую память исполняемого кода / С.В. Белим, С.Н. Мунько // Проблемы информационной безопасности. Компьютерные системы. – 2022. – №2. – С. 30-34.

А3. Белим С.В., Мунько С.Н. Стеганографическое встраивание информации в SWG-изображения на WEB-странице / С.В. Белим, С.Н. Мунько // Безопасность информационных технологий. – Т.30, № 2. – 2023. – С. 116-126.

Публикации в изданиях, индексируемых в базе Scopus:

А4. Belim S. V., Belim S. Yu., Munko S.N. Embed digital watermarks in executable program memory / S. V. Belim, S. Yu. Belim, S.N. Munko // Journal of Physics: Conference Series. – 2021. – Vol.1901, No.1. – P.7.

А5. Belim S. V., Munko S.N. Algorithm for Digital Watermark Generation in Executable Program Memory / S. V. Belim, S.N. Munko // CEUR. – 2021. – P.5

Публикации в прочих изданиях:

А6. Белим С.В., Мунько С.Н. Алгоритм динамического формирования цифрового водяного знака в памяти программы / С.В. Белим, С.Н. Мунько // Прикладная математика и фундаментальная информатика. – 2021. – Т.7, №4. – С. 12-17.

А7. Белим С.В., Мунько С.Н. Разработка алгоритма стеганографического скрытия информации в исходном коде программы / С.В. Белим, С.Н. Мунько // Материалы региональной молодежной научно-практической конференции «Нанотехнологии. Информация. Радиотехника». – 2021. – С. 54-57.

А8. Мунько С.Н. Стеганография для SVG-изображений в HTML-коде / С.Н. Мунько, С.В. Белим, // Материалы XXXI Российской научно-технической конференции «Актуальные проблемы информатики, радиотехники, связи». – 2024. – С. 123.

А9. Белим С.В. Стеганографическое встраивание данных в дерево классов веб-страницы. / С.Н. Мунько, С.В. Белим, // Материалы X Международной юбилейной научно-практической конференции «Проблемы информационной безопасности социально-экономических систем». – 2024. – С. 85.

Объекты интеллектуальные собственности:

1) Свидетельство о государственной регистрации программы для ЭВМ № 2021617713 Российская Федерация. Встраивание цифровых водяных знаков в память исполняемой программы : № 2021617129: заявл. 19.05. 2021 г. опубл. (зарег.) 19.05.2021 / С.Н. Мунько, С.В. Белим ; заявитель Ом. гос. техн. ун-т. – 1 с.

2) Свидетельство о государственной регистрации программы для ЭВМ № 2022619931 Российская Федерация. Встраивание информации в HTML-документ : № 2022618999: заявл. 20.05.2022 г. : опубл. (зарег.) 27.05.2022 / С.Н. Мунько, С.В. Белим ; заявитель Ом. гос. техн. ун-т. – 1 с.

3) Свидетельство о государственной регистрации программы для ЭВМ № 2022685371 Российская Федерация. Встраивание информации в svg изображения : № 2021617129: заявл. 16.12. 2022 г. : опубл. (зарег.) 22.12.2022 / С.Н. Мунько, С.В. Белим ; заявитель Ом. гос. техн. ун-т. – 1 с.

Список литературы

1 Абасова, А. М. Алгоритм повышения устойчивости к деструктивным воздействиям цифровых водяных знаков, встраиваемых в цветное изображение / А. М. Абасова // Известия ЮФУ. Технические науки. – 2014. – №8 (157). URL: https://cyberleninka.ru/article/n/algoritm-povysheniya-ustoychivosti-k-destruktivnym-vozdeystviyam-tsifrovyh-vodyanyh-znakov-vstraivaemyh-v-tsvetnoe-izobrazhenie (дата обращения: 10.10.2023).

2 Абасова, А. М., Бабенко, Л. К. Защита информационного содержания изображений в условиях наличия деструктивного воздействия / А. М. Абасова, Л. К. Бабенко // Вопросы кибербезопасности. – 2019. – №2 (30). – URL: https://cyberleninka.ru/article/n/zaschita-informatsionnogo-soderzhaniya-izobrazheniy-v-usloviyah-nalichiya-destruktivnogo-vozdeystviya (дата обращения: 10.10.2023).

3 Алгоритмы и методы защиты программного кода на базе обфускации [Текст] / А. В. Красов, И. П. Зуев, П. В. Карельский [и др.] // I-methods. Информатика, вычислительная техника и управление. – 2020. Т. 12, № 1. – С.12.

4 Балтаев, Р. Х., Лунегов, И. В. Стеганографический метод встраивания информации с использованием шумоподобной последовательности и сохранением статистической модели изображений / Р. Х. Балтаев, И. В. Лунегов // Кибернетика и программирование. – 2018. – №5. – URL: https://cyberleninka.ru/article/n/steganograficheskiy-metod-vstraivaniya-informatsii-s-ispolzovaniem-shumopodobnoy-posledovatelnosti-i-sohraneniem-statisticheskoy (дата обращения: 10.10.2023).

5 Баранник, В.В., Юдин, А.К., Фролов, О.В. Технология повышения безопасности информационных ресурсов на основе использования функционального преобразования при косвенном

стеганографическом встраивании / В. В. Баранник, А. К.Юдин, О. В. Фролов // Радиоэлектроника и информатика. – 2016. – №1. – URL: https://cyberleninka.ru/article/n/tehnologiya-povysheniya-bezopasnosti-informatsionnyh-resursov-na-osnove-ispolzovaniya-funktsionalnogo-preobrazovaniya-pri-kosvennom (дата обращения: 10.10.2023).

6 Баранник, Д. В., Бекиров, А. Э. Концепция структурного стеганографического кодирования с маскированием / Д. В. Баранник, А. Э. Бекиров // АСУ и приборы автоматики. – 2014. – №168. – URL: https://cyberleninka.ru/article/n/kontseptsiya-strukturnogo-steganograficheskogo-kodirovaniya-s-maskirovaniem (дата обращения: 10.10.2023).

7 Белим, С.В., Мунько, С.Н. Алгоритм встраивания цифрового водяного знака в динамическую память исполняемого кода / С.В. Белим, С.Н. Мунько // Проблемы информационной безопасности. Компьютерные системы. – 2022. – №2. – С. 30-34.

8 Белим, С.В., Мунько, С.Н. Алгоритм динамического формирования цифрового водяного знака в памяти программы / С.В. Белим, С.Н. Мунько // Прикладная математика и фундаментальная информатика. – 2021. – Т.7, №4. – С. 12-17.

9 Белим, С.В., Мунько, С.Н. Разработка алгоритма стеганографического скрытия информации в исходном коде программы / С.В. Белим, С.Н. Мунько // Материалы региональной молодежной научно-практической конференции Нанотехнологии. Информация. Радиотехника. – 2021. – С. 54-57.

10 Белим, С.В., Мунько, С.Н. Стеганографическое встраивание данных в код HTML-документа / С.В. Белим, С.Н. Мунько // Вестник компьютерных и информационных технологий. – №11. – 2022. – С. 37-44.

11 Белим, С.В., Мунько, С.Н. Стеганографическое встраивание информации в SWG-изображения на WEB-странице / С.В. Белим, С.Н.

Мунько // Безопасность информационных технологий. – Т.30, № 2. – 2023. – С. 116-126.

12 Белим, С.В., Вильховский, Д.Э. Алгоритм выявления стеганографических вставок типа LSB-замещения на основе метода анализа иерархий [Текст] / С.В Белим, Д.Э. Вильховский // Вестник компьютерных и информационных технологий. – 2018. – №4. – С. 25-33.

13 Блинова, Е. А. Математическая модель стеганографической системы на основе ключевой информации в виде стегонаборов / Е.А. Блинова // Системный анализ и прикладная информатика. – 2022. – №3. – URL: https://cyberleninka.ru/article/n/matematicheskaya-model-steganograficheskoy-sistemy-na-osnove-klyuchevoy-informatsii-v-vide-stegonaborov (дата обращения: 12.10.2023).

14 Блинова, Е.А., Урбанович, П.П. Стеганографический метод на основе встраивания скрытых сообщений в кривые Безье изображений формата SVG. [Текст] / Е.А. Блинова, П.П. Урбанович // Журнал Белорусского государственного университета. Математика. Информатика. – 2021. – Т. 3. С. 68-83.

15 Бречко, А. А. Параметрическая стеганография / А.А. Бречко // Известия ТулГУ. Технические науки. – 2021. – №12. – URL: https://cyberleninka.ru/article/n/parametricheskaya-steganografiya (дата обращения: 12.10.2023).

16 Бречко, А. А., Булгакова М. И. Способ скрытия информационного взаимодействия / А.А. Бречко, М.И. Булгакова // Известия ТулГУ. Технические науки. – 2022. – №5. – URL: https://cyberleninka.ru/article/n/sposob-skrytiya-informatsionnogo-vzaimodeystviya (дата обращения: 12.10.2023).

17 В. Столлингс. Криптография и сетевая безопасность: принципы и работа. – 7 изд. Великобритания, Харлоу: Pearson Education Limited, 2017. – 671 с.

18 Гатчин, Ю.А. Теория информационной безопасности и методология защиты информации [Текст] / Ю.А. Гатчин, В.В. // Сухостат. – СПб.: СПбГУ ИТМО, 2010. – 98 с.

19 Герлинг, Е.Ю., Ахрамеева, К.А. Обзор современного программного обеспечения, использующего методы стеганографии / Е.Ю. Герлинг, К.А. Ахрамеева // Экономика и качество систем связи. – 2019. – №3 (13). – URL: https://cyberleninka.ru/article/n/obzor-sovremennogo-programmnogo-obespecheniya-ispolzuyuschego-metody-steganografii (дата обращения: 12.10.2023).

20 Головешкин, А.В., Михалкович, С.С. Устойчивая алгоритмическая привязка к произвольному участку кода программы / А.В. Головешкин, С.С. Михалкович // Программные системы: теория и приложения. – 2022. – №1 (52). – URL: https://cyberleninka.ru/article/n/ustoychivaya-algoritmicheskaya-privyazka-k-proizvolnomu-uchastku-koda-programmy (дата обращения: 12.10.2023).

21 Гордиенко, В.В., Довгаль, В.М., Лукина, А.В. Способы криптографии и стеганографии для защиты электронных документов от подделки на основе отображения Лоренца / В.В. Гордиенко, В.М.Довгаль, А.В. Лукина // Auditorium. – 2018. – №2 (18). URL: https://cyberleninka.ru/article/n/sposoby-kriptografii-i-steganografii-dlya-zaschity-elektronnyh-dokumentov-ot-poddelki-na-osnove-otobrazheniya-lorentsa (дата обращения: 12.10.2023).

22 Жиляков, Е. Г., Черноморец, А. А., Болгова, Е. В. Исследование устойчивости стеганографии в изображениях / Е.Г.Жиляков, А.А.Черноморец, Е.В. Болгова // Экономика. Информатика. – 2014. – №1-1 (172). – URL: https://cyberleninka.ru/article/n/issledovanie-ustoychivosti-steganografii-v-izobrazheniyah (дата обращения: 11.10.2023).

23 Жиляков, Е.Г., Лихолоб, П.Г., Медведева, А.А. Исследование некоторых стеганографических алгоритмов / Е.Г.Жиляков, П.Г. Лихолоб

А.А. Медведева // Научный результат. Информационные технологии. – 2016. – №2. – URL: https://cyberleninka.ru/article/n/issledovanie-nekotoryh-steganograficheskih-algoritmov (дата обращения: 11.10.2023).

24 Защелкин, К. Метод внедрения цифровых водяных знаков в аппаратные контейнеры с LUT-ориентированной архитектурой [Текст] / К. Защелкин // Информатика и математические методы в моделировании. –2013. Т. 3, № 4. – С. 369-38.

25 Клименко, А.В., Горбачев, В.Н. Слепое детектирование цифрового водяного знака в изображениях с палитрой без изменения цвета / А.В.Клименко, В.Н. Горбачев// Труды Кольского научного центра РАН. – 2016. – №6-7 (40). – URL: https://cyberleninka.ru/article/n/slepoe-detektirovanie-tsifrovogo-vodyanogo-znaka-v-izobrazheniyah-s-palitroy-bez-izmeneniya-tsveta (дата обращения: 11.10.2023).

26 Козачок А.В. Алгоритм маркирования текстовых документов на основе изменения интервалов между словами, обеспечивающий устойчивость к преобразованию формата / А.В. Козачок // Труды ИСП РАН. – 2021. – №4. – URL: https://cyberleninka.ru/article/n/algoritm-markirovaniya-tekstovyh-dokumentov-na-osnove-izmeneniya-intervalov-mezhdu-slovami-obespechivayuschiy-ustoychivost-k (дата обращения: 11.10.2023).

27 Кудрина, М.А., Дулимова, И.Е. Скрытие информации в аудиофайлах методами стеганографии /М.А. Кудрина, И.Е. Дулимова// НиКа. – 2017. – URL: https://cyberleninka.ru/article/n/skrytie-informatsii-v-audiofaylah-metodami-steganografii (дата обращения: 11.10.2023).

28 Менщиков, А.А. Шниперов, А.Н. Метод скрытого встраивания информации в векторные изображения [Текст] / А.А. Менщиков, А.Н. Шниперов // Доклады ТУСУРа. – 2015. – № 1 (35). – С.100-106

29 Нечта, И. В. Новый метод внедрения скрытых сообщений в алфавитное меню /И.В. Нечта // Вестник СибГУТИ. – 2018. – №2 (42). – URL:

https://cyberleninka.ru/article/n/novyy-metod-vnedreniya-skrytyh-soobscheniy-v-alfavitnoe-menyu (дата обращения: 11.10.2023).

30 Нечта, И.В. Построение хрупкого цифрового водяного знака, применяемого коллективом соавторов в исполняемых файлах [Текст] / И.В. Нечта // Вестник НГУ. Серия: Информационные технологии. – 2020. – Т. 18, № 1 (70). – С. 65–73.

31 Никольская, К. Ю. Обфускация и методы защиты программных продуктов. [Текст] / К. Ю. Никольская, А. Д. Хлестов // Вестник УрФО. Безопасность в информационной сфере. – 2015. – № 2. – С. 7–10.

32 Нистюк, О. А., Урбанович, П. П. Метод и математическая модель стеганографического преобразования информации на основе модификации контура символов текста-контейнера / О.А. Нистюк, П.П. Урбанович// Труды БГТУ. Серия 3: Физико-математические науки и информатика. – 2022. – №2 (260). – URL: https://cyberleninka.ru/article/n/metod-i-matematicheskaya-model-steganograficheskogo-preobrazovaniya-informatsii-na-osnove-modifikatsii-kontura-simvolov-teksta (дата обращения: 11.10.2023).

33 Першина, И.В., Нечай, А.А. Программные методы сокрытия информации / И.В. Першина, А.А. Нечай // Экономика и социум. – 2015. – №1-4 (14). – URL: https://cyberleninka.ru/article/n/programmnye-metody-sokrytiya-informatsii (дата обращения: 10.10.2023).

34 Пономарев И. В., Строкин Д. И. Стеганографические методы встраивания и обнаружения сокрытых сообщений, использующие gif-изображения в качестве файлов-контейнеров / И.В.Пономарев, Д.И. Строкин // Известия АлтГУ. – 2022. – №1 (123). – URL: https://cyberleninka.ru/article/n/steganograficheskie-metody-vstraivaniya-i-obnaruzheniya-sokrytyh-soobscheniy-ispolzuyuschie-gif-izobrazheniya-v-kachestve-faylov (дата обращения: 10.10.2023).

35 Развитие модели в отказоустойчивых вычислениях [Текст] / О. Дрозд, В. Харченко, А. Ручинский [и др.] // 10-я Международная конференция по надежным системам, услугам. – DESSERT, 2019. – С.2-7.

36 Савельева, М. Г., Урбанович, П. П. Растрирование web-документов и использование его характеристик для стеганографической защиты авторских прав на электронный контент /М.Г. Савельева, П.П.Урбанович// Труды БГТУ. Серия 3: Физико-математические науки и информатика. – 2023. – №1 (266). – URL: https://cyberleninka.ru/article/n/rastrirovanie-web-dokumentov-i-ispolzovanie-ego-harakteristik-dlya-steganograficheskoy-zaschity-avtorskih-prav-na-elektronnyy (дата обращения: 10.10.2023).

37 Савостьянчик, В. В., Алефиренко, В. М. Выбор программных средств компьютерной стеганографии для исследования эффективности скрытия графической информации / В.В. Савостьянчик, В.М. Алефиренко // Science Time. – 2017. – №11 (47). – URL: https://cyberleninka.ru/article/n/vybor-programmnyh-sredstv-kompyuternoy-steganografii-dlya-issledovaniya-effektivnosti-skrytiya-graficheskoy-informatsii (дата обращения: 10.10.2023).

38 Слипенчук, П. В. Стеганография в кодах, исправляющих ошибки / П.В. Слипенчук // Инженерный журнал: наука и инновации. – 2015. – №3 (3). – URL: https://cyberleninka.ru/article/n/steganografiya-v-kodah-ispravlyayuschih-oshibki (дата обращения: 10.10.2023).

39 Сокол, Д.Т., Прокопов, К.В., Довгаль, В.М. Методика стеганографии с использованием текстового контейнера на основе детерминированно-хаотических рядов / Д.Т. Сокол, К.В. Прокопов, В.М. Довгаль // Auditorium. – 2016. – №4 (12). – URL: https://cyberleninka.ru/article/n/metodika-steganografii-s-ispolzovaniem-tekstovogo-konteynera-na-osnove-determinirovanno-haoticheskih-ryadov (дата обращения: 10.10.2023).

40 Стеганографическая защита изображений из PDF документов на основе конвертора PDF-SVG [Текст] / В.Н. Горбачев, И.К. Метелёв, Е.М. Кайнарова [и др.] // GraphiCon. – 2017. – С.108-111.

41 Фримучков, А.Н. Применение скремблирования для усложнения обнаружения скрытой информации, записанной методом LSB / А.Н. Фримучков // Достижения науки и образования. – 2017. – №1 (14). – URL: https://cyberleninka.ru/article/n/primenenie-skremblirovaniya-dlya-uslozhneniya-obnaruzheniya-skrytoy-informatsii-zapisannoy-metodom-lsb (дата обращения: 11.10.2023).

42 Частикова, В. А., Аббасов, Т. О., Аббасова, С. С. Методика распознавания скрытой информации в изображениях на основе алгоритмов стеганографии / В.А. Частикова, Т.О. Аббасов, С.С. Аббасова // Вестник Адыгейского государственного университета. Серия 4: Естественно-математические и технические науки. – 2020. – №3 (266). – URL: https://cyberleninka.ru/article/n/metodika-raspoznavaniya-skrytoy-informatsii-v-izobrazheniyah-na-osnove-algoritmov-steganografii (дата обращения: 14.10.2023).

43 Шутько, Н. П. Защита авторских прав на электронные текстовые документы методами стеганографии / Н.П. Шутько // Труды БГТУ. Серия 3: Физико-математические науки и информатика. – 2013. – №6. – URL: https://cyberleninka.ru/article/n/zaschita-avtorskih-prav-na-elektronnye-tekstovye-dokumenty-metodami-steganografii (дата обращения: 12.10.2023).

44 Шутько, Н. П. Особенности и формальное описание процесса осаждения секретной информации в текстовые документы на основе стеганографии / Н.П. Шутько // Труды БГТУ. Серия 3: Физико-математические науки и информатика. – 2014. – №6 (170). – URL: https://cyberleninka.ru/article/n/osobennosti-i-formalnoe-opisanie-protsessa-osazhdeniya-sekretnoy-informatsii-v-tekstovye-dokumenty-na-osnove-steganografii (дата обращения: 12.10.2023).

45 A fair blind ignature scheme to revoke malicious vehicles in VANETs / X. Wang, J. Jianming, Z. Shujing [et al.] // Computers. Mater. Continua. – 2019. –Vol.58, No1. – P. 249-262.

46 A Framework of Text-based Steganography Using SD Form Semantics Model [Text] / M. Niimi, S. Minewaki, H. Noda [et al.] // Pacific Rim Workshop on Digital Steganography. – 2003.

47 A new efficient approach for the removal of impulse noise from highly corrupted images [Text] / E.Abreu, M. Lightstone, S.K. Mitra [et al.] // IEEE Transactions on Image Processing. – 1996. – V.5. – P. 1012-1025.

48 A practical method for watermarking Java programs [Text] / A. Monden, H. Iida, K. Matsumoto [et al.] // Proceedings 24th Annual International Computer Software and Applications Conference. – COMPSAC. – 2000.

49 A secure covert communication model based on video steganography [Text] / S. Hanafy [et al.] // IEEE Military Commu- nications Conference. – IEEE. – 2008. – P.16.

50 A universal noise removal algorithm with an impulse detector / R. Garnett, T. Huegerich, C. Chui [et al.] // IEEE Transactions on Image Processing. – 2005. – V. 14, No. 11. – P. 1747-1754.

51 Alattar, A.M., Alattar, O.M. Watermarking electronic text documents containing justified paragraphs and irregular line spacing [Text] / A.M. Alattar, O.M. Alattar // Proceedings of SPIE. – 2004. – Vol.5306. – P. 685-695.

52 Alla, K., Prasad, R.S.R. Prasad. An Evolution of Hindi Text Steganography [Text] / K.Alla, R.S.R. Prasad // Sixth International Conference on Information Technology New Generations. – 2009. – P.1577 - 1578.

53 Almutairi, A. A Comparative study on steganography digital images: a case study of scalable vector graphics (SVG) and portable network graphics (PNG) images formats [Text] / A. Almutairi // International Journal of Advanced Computer Science and Applications. – 2018. – V. 9, No.1. – P. 170-175.

54 Almutairi, B. A new steganography method for scalable vector graphics (SVG) images based on an improved LSB algorithm [Text] / B. Almutairi // International Journal of Computer Science and Network Security. – 2019. – V.19, N.10. – P.99-104.

55 Bajaj, I., Aggarwal, R.K. RSA Secured Web Based Steganography Employing HTML Space Codes and Compression Technique [Text] / I. Bajaj, R.K. Aggarwal // In Proceedings of the 2019 International Conference on Intelligent Computing and Control Systems, Madurai, India. – IEEE: New York, NY, USA. – 2019. – P.865–868.

56 Belim S. V., Belim S. Yu., Munko S.N. Embed digital watermarks in executable program memory / S. V. Belim, S. Yu. Belim, S.N. Munko // Journal of Physics: Conference Series. – 2021. – Vol.1901, No.1. – P.7.

57 Belim S. V., Munko S.N. Algorithm for Digital Watermark Generation in Executable Program Memory / S. V. Belim, S.N. Munko // CEUR. – 2021. – P.5

58 Belim, S.V., Vilkhovskiy, D.E. Method of detecting hidden data transmission via the Koch-Zhao steganographic algorithm [Text] / S.V. Belim, D.E. Vilkhovskiy // Journal of Physics: Conf. Series. – 2019. – V.1210. P.5.

59 Can dres provide long-lasting security the case of return-oriented programming and the avc advantage [Text] / S. Checkoway, A. J. Feldman, B. Kantor [et al.] In Proceedings of the 2009 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections. – EVT, 2009.

60 Chou, Y.C., Huang, C.Y., Liao, H.C. A reversible data hiding scheme using cartesian product for HTML file [Text] / Y.C. Chou, C.Y. Huang, H.C. Liao // In Proceedings of the 2012 Sixth International Conference on Genetic and Evolutionary Computing, Kitakyushu, Japan. – IEEE: New York, NY, USA, 2012. – P.153-156.

61 Clarke, J. SQL Injection Attacks and Defense [Text] / J. Clarke // Syngress, 2nd edition. – 2012.

62 Co-Embedding Additional Security Data and Obfuscating Low-Level FPGA Program [Text] / K. Zashcholkin, O. Drozd, R. Shaporin [et al.] // IEEE East-West Design & Test Symposium. – EWDTS, 2020. – Vol.1. – P.1-5.

63 Collberg, C. Thomborson, C. Software watermarking: models and dynamic embeddings [Text] / C. Collberg, C. Thomborson // POPL '99: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languagesJanuary. – 1999. – P. 311-324.

64 Cooperation and Sustainable Development [Text] / A. Bogoviz, A.Suglobov, A.Maloletko [et al.] // Lecture Notes in Networks and Systems. LNNS, 2022. – Vol. 245.

65 Dalal, J. Video Steganography Techniques in Spatial DomainA Survey [Text] / J. Dalal // Proceedings of the International Conference on Computing and Communication Systems. – Springer, 2018.

66 Digital Watermarking and Steganography [Text] / I. Cox, M. Miller, J. Bloom [et al.] // Morgan kaufmann: San Francisco. – CA, USA, 2007.

67 Directed graph pattern synthesis inLSB technique on video steganography [Text] / D. Bhattacharyya [et al.] // Advances in Computer Science and Information Technology. – Springer Berlin Heidelberg, 2010.

68 Document marking and identification using both line and word shifting [Text] / S.H. Low, N.F. Maxemchuk, J.T. Brassil [et al.] // Proceedings of the Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies. –INFOCOM '95. – 1995. – Vol.2. P.853-860.

69 Efficiency assessment of the steganographic coding method with indirect integration of critical information [Text] / O. Yudin, R. Ziubina, S. Buchyk [et al.] // 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT). – 2019. – P. 36-40.

70 Elharrouss, O., Almaadeed, N., Al-Maadeed, S. An image steganography approach based on k-least significant bits (k-LSB) [Text] / O.

Elharrouss, N. Almaadeed, S. Al-Maadeed, // 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies. – ICIoT. 2020. – P. 131-135.

71 Emoticon-based Text Steganography in Chat [Text] / Z.H. Wang, C.C. Chang, D. Kieu [et al.] //Second AsiaPacific Conference on Computational Intelligence and Industrial applications. – 2009.

72 Fateh, M., Rezvani, M., Irani, Y. A new method of coding for steganography based on LSB matching revisited [Text] / M. Fateh, M. Rezvani, Y. Irani // Security and Communication Networks. – 2021. – Vol.5. – P.1-15.

73 Fedorov, O., Omelchenko, A., Yaurov, A. Tuning parameters of the Koch and Zhao stego algorithm [Text] / O. Fedorov, A. Omelchenko, A. Yaurov // 2019 29th International Conference Radioelektronika. – 2019. – P. 1-5.

74 Ford, B. Matchertext: Towards Verbatim Interlanguage Embedding [Text] / B. Ford // Programming Languages. – 2022. – P.22.

75 Gutub, A., Al-Shaarani, F. Efficient implementation of multi-image secret hiding based on LSB and DWT steganography comparisons [Text] / A. Gutub, F. Al-Shaarani // Arabian Journal for Science and Engineering. – 2020. – V. 45. P. 2631-2644.

76 Huang, D., Yan, H. Inter word Distance Changes Represented by Sine Waves for Watermarking Text Images [Text] / D.Huang, H.Yan // IEEE Transactions on Circuits and Systems for Video Technology. – 2001. – Vol.11, No.12. – P.1237-1245.

77 Imran, S., Khan, A., Ahmad, B. Text Steganography Utilizing XML, HTML and XHTML Markup Languages [Text] / S. Imran, A.Khan, B. Ahmad //Int. J. Inf. Technol. Secur. – 2017 – Vol.9. – P.99–116.

78 Increasing the Effective Volume of Digital Watermark Used in Monitoring the Program Code Integrity of FPGA-Based Systems [Text] / K. Zashcholkin, O. Drozd, R. Shaporin [et al.] // Yulian SulimaEast-West Design & Test Symposium. EWDTS, 2019.

79 Information hiding scheme based on PE file resource data [Text] / D. Qingfeng, Y. Wang, Z. Kaize [et al.] // Comput. Eng. – 2009. – Vol.35, No.13, P.128-130.

80 Inverted LSB image steganography using adaptive pattern to improve imperceptibility [Text] / S. Rustad, D.R.I.M. Setiadi, A. Syukur [et al.] // Journal of King Saud University - Computer and Information Sciences. – 2022. – V. 34, No.6.

81 Invisible Secret. Available online [Электронный ресурс]. – URL: http://www.invisiblesecrets.com/ (accessed on 2 December 2020).

82 Jaiswal, R.J., Patil, N.N. Implementation of a new technique for web document protection using Unicode [Text] / R.J. Jaiswal, N.N. Patil // In Proceedings of the 2013 International Conference on Information Communication and Embedded Systems, Chennai, India. – IEEE: New York, NY, USA, 2013. –P. 69-72.

83 Jin, C., Zhang, D., Pan, M. A Novel Web Page Watermark Scheme for HTML Security [Text] / C. Jin, D. Zhang, M. Pan. // International Conference of Information Science and Management Engineering. – 2010.

84 Kangjie, L., Siyang, X., Debin, G. RopSteg: program steganography with return oriented programming. [Text] / L. Kangjie, X. Siyang, G. Debin // CODASPY '14: Proceedings of the 4th ACM conference on Data and application security and privacyMarch. – 2014. – P.265-272.

85 Katzenbeisser, S. Petitcolas, F.A.P. Digital Watermarking [Text] / S. Katzenbeisser, F.A.P. Petitcolas // Artech House. – London, UK, 2000. – Vol. 2. No.13.

86 Kim, Y., Moon K., Oh I. A Text Watermarking Algorithm based on Word Classification and Inter word Space Statistics / Y. Kim, K. Moon, I. Oh // Proceedings of the Seventh International Conference on Document Analysis and Recognition. ICDAR, 2003. – P.775–779.

87 Kis, D. Pataki, N. Source Code-based Steganography [Text] / D. Kis, N. Pataki // In Proceedings of the 10th International Conference on Applied Informatics, Eger, Hungary. – 2017. – P.157-162.

88 Lee, I.S., Tsai, W.H. Secret communication through webpages using special space codes in HTML files [Text] / I.S. Lee, W.H. Tsai // Int. J. Appl. Sci. Eng. – 2008. – Vol. 6. –P.141-149.

89 Li, Y., Shi, X. Research on PE file information hiding technology [Text] / Y. Li, X. Shi // Netw. Secur. Technol. Appl. – 2017. – P.51-52.

90 Mahato, S., Yadav, D.K., Khan, D.A. A modified approach to text steganography using HyperText markup language [Text] / S. Mahato, D.K. Yadav, D.A. Khan // In Proceedings of the 2013 Third International Conference on Advanced Computing and Communication Technologies, Rohtak, India. – New York, NY, USA, 2013. – P.40–44.

91 Mazumdar, D., Das, A., Pal, S.K. MRF based LSB steganalysis: a new measure of steganography capacity [Text] / D. Mazumdar, A. Das, S.K. Pal // PReMI: International Conference on Pattern Recognition and Machine Intelligence. – 2009. – V. 5909. – P. 420-425.

92 Memon, J.A., Khowaja, K., Kazi, H. Evaluation of steganography for Urdu [Text] / J.A. Memon, K. Khowaja, H. Kazi // Journal of Theoretical and Applied Information Technology. – P. 232-237.

93 Moerland T. Steganography and Steganalysis [Электронный ресурс]. – URL: www.liacs.nl/home/tmoerlan/privtech.pdf.

94 Mohit G. A Novel Text Steganography Technique Based on HTML Documents [Text] / G. Mohit // International Journal of Advanced Science and Technology. – 2011. – Vol. 35. – P.129-138.

95 Multiple layer Text security using Variable block size Cryptography and Image Steganography [Text] / Ch.Kumar [et al.] // IEEE International Conference on Computational Intelligence and Communication Technology. – IEEE-CICT, 2017.

96 Novel Steganography over HTML Code [Text] / A. Odeh, K. Elleithy, M. Faezipour // In Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering, Springer: Cham, Switzerland. – 2015. –P. 607-611.

97 Rafat, K.F. Cutting Edge Steganography Using HTML Document-An Appraisal [Text] / K.F Rafat // Int. J. Comput. Sci. Inf. Secur. – 2016. – Vol.14, No.960.

98 Return-oriented programming without returns [Text] / M. Winandy, L. Davi, A. Dmitrienko [et al.] // In Proceedings of the 17th ACM conference on Computer and Communications Security. – CCS, 2010.

99 Return-oriented programming: Systems,languages, and applications [Text] / R. Roemer, E. Buchanan [et al.] // ACM Transactions on Information and System Security. – ACM, 2009. – Vol. 15. – P.1-34.

100 Shen, D., Zhao, H. A novel scheme of webpage information hiding based on attributes [Text] / D.Shen, H. Zhao // In Proceedings of the 2010 IEEE International Conference on Information Theory and Information Security. – TX, USA, IEEE: New York, NY, USA, 2010. – P.1147–1150.

101 Shirali-Shahreza, M. Text Steganography by Changing Words Spelling [Text] / M. Shirali-Shahreza // International Journal of Advanced Communication Technology. ICACT, 2008. – Vol.3. – P.1912 - 1913.

102 Shirali-Shahreza, M. H., Shirali-Shahreza, S. A New Approach to Persian/Arabic Text Steganography [Text] / M. H. Shirali-Shahreza, S. Shirali-Shahreza // Proceedings of 5th IEEE/ACIS international Conference on Computer and Information Science and 1st IEEE/ACIS. – 2006.

103 Shirali-Shahreza, M. H., Shirali-Shahreza, S. A Robust Page Segmentation Method for Persian/Arabic Document / Steganography [Text] / M. H. Shirali-Shahreza, S. Shirali-Shahreza // WSEAS Transactions on Computers. – 2005. – Vol. 4, No.11. – P.1692-1698.

104 Shiva Darshan, S.L., Jaidhar, C.D. Performance evaluation of filter-based feature selection techniques in classifying portable executable files [Text] / S.L. Shiva Darshan, C.D. Jaidhar // Proc. Comput. Sci. – 2018. – P.125.

105 Similarity hash based scoring of portable executable files for efficient malware detection in IoT [Text] / A.P. Namanya, I.U. Awan, J.P. Disso, M. Younas, Future Generation Computer Systems. – 2019.

106 Singh, R.K., Alankar, B. A Novel Approach For Data Hiding In Web Page Steganography Using Encryption With Compression Based Technique [Text] / R.K. Singh, B. Alankar // IOSR J. Comput. Eng. – 2016. – Vol.18. – P.73-77.

107 Singh, U. A Steganography Technique for Hiding Information in Image [Text] / U. Singh // International Journal of Emerging Technologies in Computa- tional and Applied Sciences. – IJETCAS, 2014. – P.134-137.

108 Sklavos N., Chaves R., Natale G. Di. Hardware Security and Trust [Text] / N. Sklavos, R. Chaves, G. Di Natale. – Springer, 2017. – p. 253.

109 Snow. Available online [Электронный ресурс]. – URL: http://www.darkside.com.au/snow/ (accessed on 2 December 2020).

110 Steganalysis of LSB using energy function [Text] / P.P. Amritha, M. S. Muraleedharan, K. Rajeev [et al.] // Advances in Intelligent Systems and Computing. – 2016. – V. 384. – P. 549-558.

111 Steganographic algorithm for information hiding using scalable vector graphics images [Text] / B. Mados, J. Hurtuk, M. Čopjak [et al.] // Acta Electrotechnica et Informatica. – 2014. – V.14, No.4. – P. 42-45.

112 Sun, Z., Li, C., Zhao, Q. Hide chopin in the music: efficient information steganography via random shuffling [Text] / Z. Sun, C. Li, Q. Zhao // IEEE International Conference on Acoustics, Speech and Signal Processing. – ICASSP, 2021. – P. 2370-2374.

113 Tariq, M.A., Khan, A.T.A.A., Ahmad, B. Boosting the Capacity of Web based Steganography by Utilizing HTML Space Codes: A blind

Steganography Approach [Text] / M.A. Tariq, A.T.A.A. Khan, B.Ahmad // IT Ind. – 2017 –Vol.5. – P.29–36.

114 Tian, Z., Li, Y., Yang, L. Research on PE file information hiding technology based on import table migration [Text] / Z. Tian, Y. Li, L.Yang // Comput. Sci. –2016. – Vol.43, No.1. – P. 207-210.

115 Wbstego. Available online [Электронный ресурс]. – URL: http://wbstego.wbailer.com/ (accessed on 2 December 2020).

116 Westfeld, A., Pfitzmann, A. Attacks on steganographic systems: breaking the steganographic utilities EzStego, Jsteg, Steganos and Stools and some lessons learned [Text] / A. Westfeld, A. Pfitzmann // 3rd International Workshop on Information Hiding. – 2000. – P. 61-76.

117 When good instructions go bad: generalizing return-oriented programming to risc [Text] / E. Buchanan, R. Roemer, H. Shacham [et al.] // In Proceedings of the 15th ACM conference on Computer and communications security. – CCS, 2008. – P.27-38.

118 XSS Attacks: Cross Site Scripting Exploits and Defense [Text] / S. Fogie, J.Grossman, R. Hansen [et al.] // Syngress, 1st edition. – 2007.

119 Youngho, Ch., Simun Yu. A Time-Based Dynamic Operation Model for Webpage Steganography Methods [Text] / Ch.Youngho, Yu.Simun // Electronics. –2020. –Vol 9, No.12:2113. – P.22.

120 Yun, Q. Shi, Sutthiwan, P., Chen, L. Textural features for steganalysis [Text] / Q. Yun Shi, P. Sutthiwan, L. Chen // International Workshop on Information Hiding. – 2013. – V. 769. – P. 63 -77.

121 Zuwei, T. Hengfu, Y. Code fusion information-hiding algorithm based on PE file function migration [Text] / T. Zuwei, Y. Hengfu // EURASIP Journal on Image and Video Processing. – 2021. Vol. 2.

Приложения

Приложение А

Свидетельства о регистрации программ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2021617713

«Встраивание цифровых водяных знаков в память исполняемой программы»

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Омский государственный технический университет» (RU)***

Авторы: ***Мунько Сергей Николаевич (RU), Белим Сергей Викторович (RU)***

Заявка № **2021617129**
Дата поступления **19 мая 2021 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***19 мая 2021 г.***

Руководитель Федеральной службы по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат 0x02A5CFBC00B1ACF59A40A2F08092E9A118
Владелец Ивлиев Григорий Петрович
Действителен с 15.01.2021 по 15.01.2035

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2022685371

«Встраивание информации в svg изображения»

Правообладатель: ***Федеральное государственное автономное образовательное учреждение высшего образования «Омский государственный технический университет» (RU)***

Авторы: ***Белим Сергей Викторович (RU), Мунько Сергей Николаевич (RU)***

Заявка № **2022684947**
Дата поступления **16 декабря 2022 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***22 декабря 2022 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат 68b80077e14e40f0a94edbd24145d5c7
Владелец **Зубов Юрий Сергеевич**
Действителен с 2.03.2022 по 26.05.2023

Ю.С. Зубов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2022619931

Встраивание информации в html-документ

Правообладатель: ***Федеральное государственное автономное образовательное учреждение высшего образования «Омский государственный технический университет» (RU)***

Авторы: ***Мунько Сергей Николаевич (RU), Белим Сергей Викторович (RU)***

Заявка № **2022618999**

Дата поступления **20 мая 2022 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***27 мая 2022 г.***

Руководитель Федеральной службы по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат 68b80077e14e40f0a94edbd24145d5c7
Владелец Зубов Юрий Сергеевич
Действителен с 2.03.2022 по 26.05.2023

Ю.С. Зубов

УТВЕРЖДАЮ
Генеральный директор
ООО «Лет ИТ БИ»
_____ П.А. Опучин

АКТ

О внедрении результатов диссертационной работы Мунько С.Н. «Стеганографическое встраивание информации в память исполняемого кода и код веб-страницы», представленной на соискание ученой степени кандидата технических наук по специальности 2.3.6. Методы и системы защиты информации, информационная безопасность.

Настоящий акт составлен в том, что результаты диссертационной работы Мунько Сергея Николаевича, а именно:

- алгоритм скрытого встраивания данных в HTML-код;
- алгоритм скрытого встраивания данных в SVG-изображение, размещенное на веб-странице;

использовались ООО «Лет ИТ БИ» для внедрения цифровых временных меток в версии веб-страниц и изображений на них. Данные цифровые метки позволили отслеживать обновления информации на веб-странице нескольким независимым пользователям и разработчикам в процессе создания сайта.

Состав комиссии:

22.12.23 Е.В. Олянский
22.12.23 А.В. Трофимов
22.12.23 Е.Е. Позднякова

УТВЕРЖДАЮ
Генеральный директор
ООО «СМАРТФОРС»

_____ А.М. Федосеев

АКТ

о внедрении результатов диссертационной работы Мунько С.Н. «Стеганографическое встраивание информации в память исполняемого кода и код веб-страницы», представленной на соискание ученой степени кандидата технических наук по специальности 2.3.6. Методы и системы защиты информации, информационная безопасность.

Настоящий акт составлен в том, что результаты диссертационной работы Мунько Сергея Николаевича, а именно:

- алгоритм встраивания данных в память исполняемой программы;
- модуль аутентификации для скрытого встраивания данных;

использовались ООО «СМАРТФОРС» для внедрения цифровых индикаторов в копии разрабатываемого программного обеспечения и отслеживания копий разрабатываемого программного обеспечения.

Состав комиссии:

18.12.23 Д.Е. Левен

18.12.23 Н.Н. Пивкин