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

АВТОРЕФЕРАТ

диссертации на соискание ученой степени

кандидата технических наук

Омск - 2024

Работа выполнена на кафедре «Прикладная математика и фундаментальная информатика» федерального государственного автономного образовательного учреждения высшего образования «Омский государственный технический университет».

Научный руководитель:	**Белим Сергей Викторович**, доктор физико-математических наук, профессор, профессор кафедры «Комплексная защита информации» федерального государственного автономного образовательного учреждения высшего образования «Омский государственный технический университет».
Официальные оппоненты:	**Аникин Игорь Вячеславович**, доктор технических наук, профессор, заведующий кафедрой систем информационной безопасности федерального государственного бюджетного образовательного учреждения высшего образования «Казанский национальный исследовательский технический университет им. А.Н.Туполева-КАИ».
	Шакурский Максим Викторович, доктор технических наук, доцент, заведующий кафедрой информационной безопасности федерального государственного бюджетного образовательного учреждения высшего образования «Поволжский государственный университет телекоммуникаций и информатики».
Ведущая организация:	Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики», г. Новосибирск.

Защита диссертации состоится «26» сентября 2024 года на заседании диссертационного совета 24.2.350.07, созданного на базе ФГАОУ ВО «Омский государственный технический университет» по адресу: 644050, г. Омск, пр. Мира, д. 11, Главный корпус, ауд. П-202.

С диссертацией можно ознакомиться в научной библиотеке Омского государственного технического университета и на сайте www.omgtu.ru.

Отзыв на автореферат в двух экземплярах, заверенный печатью учреждения, просим направлять по адресу: 644050, г. Омск, пр. Мира, д. 11, ученому секретарю диссертационного совета 24.2.350.07. Тел.: (3812) 62-85-58, e-mail: dissov_omgtu@omgtu.ru.

Автореферат разослан «__ _» _____ 2024 г.

Учёный секретарь
диссертационного совета 24.2.350.07,
кандидат технических наук, доцент Грицай А.С.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность темы исследования

Методы скрытого встраивания данных в легальные сообщения (стеганографические методы) применяются для решения двух задач: скрытая передача информации и внедрение меток безопасности. Стеганографические методы передачи данных используются при необходимости сокрытия самого факта наличия сообщения (скрытые каналы). Метки безопасности являются одной из разновидностей скрытых данных, имеют небольшой объем и применяются для скрытого хранения свойств информационного объекта. Для сокрытия данных используется некоторое легальное открытое сообщение, называемое стеганографическим контейнером (стегоконтейнером, контейнером). Встраивание скрытого сообщения (стеганографическая вставка, стеговставка) осуществляется с помощью модификации данных стегоконтейнера. Модификация стегоконтейнера должна удовлетворять трем условиям. Во-первых, она не должна оказывать заметного влияние на штатное использование стегоконтейнера. Во-вторых, должен существовать алгоритм извлечения скрытых данных без потерь. В-третьих, сложность алгоритма обнаружения скрытых данных должна быть экспоненциальной.

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. 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 скрытых данных 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. 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.

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 115 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.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

Во введении описана актуальность работы, поставлена цель и выдвинуты задачи, аргументирована научная новизна и практическая значимость работы, а также сформулированы положения и результаты, выносимые на защиту, приведены сведения об апробации работы.

В первой главе приведен обзор методов стеганографического встраивания в память исполняемой программы, а также в web-страницы и объекты векторной графики. Выполнен анализ основных методов, используемых для решения этих задач. Указана стойкость существующих методов к стеганографическому анализу.

Во второй главе представлены алгоритмы встраивания метки безопасности в исполняемый код программы с помощью подключаемой библиотеки авторизации.

Встраивание метки безопасности 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. Предложенный 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 с адресом 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 метки безопасности 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. $M_i=M_{i-1}+4\Delta M$, $M_i=M_0+4i\Delta M$.

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

Программный комплекс встраивания и проверки метки безопасности реализован в виде динамической библиотеки авторизации (*auto.dll*) и программы проверки метки безопасности (*verificator.exe*) (Рисунок 1).

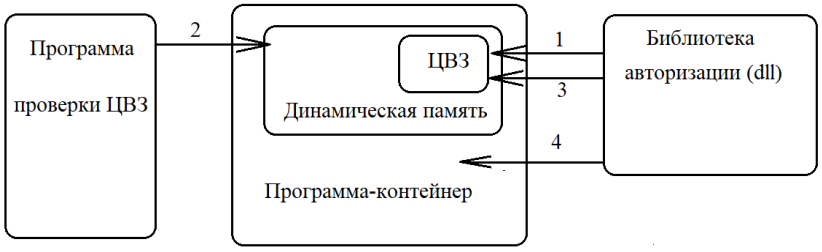

Рисунок 1. Схема взаимодействия модулей формирования и проверки метки безопасности с программой-контейнером. 1 – формирование ЦВЗ, 2 – проверка ЦВЗ, 3 – удаление ЦВЗ, 4 – штатный режим.

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

Метод *bool CheckPassword* (*char* hashPassword*) булевского типа. 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*(). Метод *void WriteCvz*() выполняет встраивание метки безопасности в оперативную память программы контейнера. Метод *void MemoryAllocation*() выделяет большое количество ячеек в динамической памяти процесса-контейнера. Метод *void _writeCvz*() осуществляет запись метки безопасности в память процесса-контейнера. 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 $CvzTime$.

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

1) *Авторизация пользователя*. 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. При вводе имени и пароля псевдопользователя динамическая библиотека переходит в режим формирования метки безопасности.

Программа проверки метки безопасности *verificator.exe* 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.

2) Функция *char** *GetCvz*(*DWORD processId*) получает в качестве параметра идентификатор процесса и возвращает встроенную последовательность.

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.

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.

В третьей главе предложена модель встраивания сообщений в *web*-страницу. В рамках модели реализуется встраивание скрытых данных в *html*-документ и в изображения формата *SVG*. 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.

Предложенный *алгоритм встраивания сообщения* в дерево *html*-тегов состоит из следующих шагов:

1) Рекурсивный обход дерева тегов для поиска пустых листовых вершин.

2) Определение максимального размера сообщения, которое может быть встроено в данный *html*-документ, по количеству найденных пустых листовых вершин определяется.

3) Шифрование встраиваемого сообщения *M* одним из симметричных шифров. Ключ шифрования *k* известен обеим сторонам. $C=E_k(M)$.

4) Разбиение на блоки по 4 бита встраиваемого зашифрованного сообщения *C*.

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

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.

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 замен. Имена классов внутри тегов формируются случайно.

3) К тегам добавляются стили со случайными значениями.

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. $h_i=H(n_i)$. На данном шаге может быть использована любая стойкая хэш-функция. При тестировании алгоритма использовалась функция *SHA*512.

3) Хэш значение представляется как набор символов из английского алфавита с использованием любой подходящей кодировки. $h_i=a_1||a_2||a_3||a_4||...$

4) Формируется метка l_i для символа с номером i на основе первых четырех символов представления хэш-функции. l_i=«*ng*-a_1a_2-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.

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 замен 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 замен. 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*.

Для встраивания скрытых данных в *html*-документ реализовано приложение 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.

Для встраивания скрытых данных в изображения формата *SVG* предложена ключевая стеганографическая схема. 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

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). $M=m_1m_2…m_n$.

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,…,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,…,n).

Предложенный 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.

Встраивание сообщения состоит из следующих шагов:

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

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

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

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.

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 замен. 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.

ЗАКЛЮЧЕНИЕ

1) Разработан метод скрытого встраивания данных в память исполняемой программы с использованием динамической библиотеки аутентификации. Такой подход повышает сложность стегоанализа за счет влияния на момент формирования данных в памяти программы с помощью парольной фразы.

2) Разработан метод встраивания скрытых данных в открытый код web-страницы с помощью модификации дерева тэгов. Данный метод создает новые классы, не влияющие на отображение и функциональные возможности web-страницы. Имена новых классов определяются ключом встраивания и маскируются под имена классов исходной страницы. Предложенный метод обеспечивает средний уровень встраивания 0,003 бит/байт.

3) Разработан метод встраивания скрытых данных в атрибуты SVG-изображения, размещенного на web-странице. Метод использует маскировку встроенных данных под

стилевые классы, определяющие параметры отображения векторной графики. Данный метод обеспечивает встраивание 2 байт в одно изображение.

4) Реализован программный комплекс скрытого встраивания данных на основе разработанных методов.

Публикации автора по теме диссертации

Публикации в журналах из списка ВАК:

1. 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.

2. 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.

3. Белим С.В., Мунько С.Н. Стеганографическое встраивание информации в SVG-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:

4. 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.

5. 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:

6. 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.

7. 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.

8. 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.

9. Белим С.В. Стеганографическое встраивание данных в дерево классов веб-страницы. / С.В. Белим, С.Н. Мунько // Материалы X Международной юбилейной научно-практической конференции «Проблемы информационной безопасности социально-экономических систем». – 2024. – С. 30-31.

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.

Печатается в авторской редакции

Подписано в печать 27.06.2024 г. Формат 60х84/16.

Отпечатано на дупликаторе. Усл.печ.л. 1,3.

Тираж 100 экз. Заказ 42.

Типография: 644050, Омск-50, пр. Мира, 11, т.: 65-32-08.

Омский государственный технический университет,

отдел научной информации