Метод вложения информации в код программ на основании графа зависимости объектов

Ахрамеева Ксения Андреевна
кандидат технических наук, доцент кафедры защищенных систем связи Санкт-Петербургского государственного университета телекоммуникаций им. проф. М.А. Бонч-Бруевича, г. Санкт-Петербург, Россия, cbor.mail@gmail.com

Зайдуллин Рустем Рамилевич
студент Санкт-Петербургского государственного университета телекоммуникаций им. проф. М.А. Бонч-Бруевича, г. Санкт-Петербург, Россия, rstm2808@gmail.com

Таров Евгений Викторович
студент Санкт-Петербургского государственного университета телекоммуникаций им. проф. М.А. Бонч-Бруевича, г. Санкт-Петербург, Россия, tarov25@mail.ru

АННОТАЦИЯ

Введение: в настоящее время проблема безопасности информации становится все более значимой и актуальной в свете постоянно растущих угроз информационной безопасности. Для защиты данных существует множество методов, включая использование стеганографии – науки о скрытой передаче информации. Внедрение данных в исходный код программ является одним из возможных способов ее использования. Тем не менее, существующие методы внедрения информации в текст приложений зачастую ограничены и требуют введения дополнительных структур данных, изменения внешнего вида программы или использования специальных меток и идентификаторов для обнаружения встроенной информации. Вместо этого, наиболее эффективным методом является использование графа зависимости элементов в коде программы. Данная структура показывает взаимосвязь между элементами кода приложения и их зависимости друг от друга. Использование графа зависимости дает возможность компактно и эффективно встраивать данные в исходный код, предоставляя удобную основу для вложения информации. Этот метод обеспечивает более высокий уровень безопасности, так как информация может быть скрыта в коде без использования дополнительных структур данных, что уменьшает вероятность обнаружения встроенной информации. **Цель работы:** в данной работе предлагается способ вложения информации в исходный код программ на основании графа зависимости элементов. **Результаты:** в работе представлены результаты исследований в области цифровой стеганографии, с фокусом на вложении информации в код программ. Рассмотрены различные методы и техники, позволяющие внедрять скрытую информацию в текст программ, а также приведено подробное описание алгоритма вложения данных при помощи графа зависимости. Представлены результаты экспериментов по вложении информации в код программы на основании предложенного алгоритма, которые показали преимущества использования графа зависимости в сравнении со стандартными методами.

КЛЮЧЕВЫЕ СЛОВА: информационная безопасность; стеганография; вложение информации; вложение в код программ; граф зависимости.

Введение

В настоящее время проблема безопасности информации является одной из наиболее актуальных и значимых в области информационных технологий. Для защиты информации применяют различные методы, включая использование техник стеганографии, которая является наукой о скрытой передаче данных [1-6]. Один из возможных способов ее применения – это внедрение информации в исходный код программ [7]. Однако, существующие методы вложения информации в текст приложений обычно имеют ряд ограничений, связанных с необходимостью введения дополнительных структур данных, изменения внешнего вида программы или использования специальных меток и идентификаторов для обнаружения встроенной информации.

Наиболее эффективным методом решения данных проблем является использование графа зависимости элементов в коде программ. Данная структура предоставляет удобную основу для вложения информации, позволяя эффективно и компактно встраивать данные в исходный код.

Цель данной работы заключается в разработке и реализации алгоритма вложения информации на основе графа зависимости программы. Актуальность работы связана с возрастающей потребностью в безопасном хранении и передаче информации, а также с необходимостью разработки более эффективных методов стеганографии.

Обзор существующих подходов и методов в области вложения информации в код программ

В области информационной безопасности и стеганографии существует множество подходов и методов, позволяющих встраивать скрытую информацию в код программ. Ниже рассмотрим некоторые из них [7-12].

Метод встраивания информации в комментарии: данный подход предполагает использование комментариев в коде программы для встраивания скрытой информации. Комментарии, как правило, не влияют на выполнение программы и могут использоваться для хранения дополнительных данных, включая скрытую информацию.

Метод встраивания информации в идентификаторы переменных и функций: этот подход предполагает использование идентификаторов переменных и функций в коде программы для встраивания скрытой информации. Идентификаторы могут быть изменены таким образом, чтобы содержать дополнительную информацию, невидимую для обычного анализа кода.

Метод встраивания информации в строковые литералы: данный подход предполагает использование строковых литералов в коде программы для встраивания скрытой информации. Строки могут быть изменены таким образом, чтобы содержать дополнительные данные, которые могут быть извлечены при выполнении программы.

Метод встраивания информации в структуру кода программы: данный подход предполагает изменение структуры кода программы, таких как порядок выполнения операций, использование различных алгоритмов и т.д., для встраивания скрытой информации. Этот подход может быть сложным и требовать более тонкой обработки кода программы.

Метод встраивания информации в метаданные программы: этот подход предполагает использование метаданных программы, таких как атрибуты, теги, комментарии и т.д., для встраивания скрытой информации. Метаданные могут быть изменены таким образом, чтобы содержать дополнительные данные, использованные в рамках выполнения программы.

Подробное описание алгоритма вложения информации в код программы на основе графа зависимостей элементов.

Алгоритм вложения информации в код программы на основе графа зависимостей элементов представляет собой процесс скрытого встраивания данных в структуру кода программы. Он основывается на анализе зависимостей между элементами, такими как функции, классы, переменные и модули, и использовании этих зависимостей для встраивания дополнительной информации без видимых изменений в функциональности программы [13-17]. Ниже приведено подробное описание этого алгоритма.

1. Исходный код программы разбивается на отдельные компоненты (функции, методы, классы, модули и т.д.), после чего для каждого из них анализируется зависимость от других составляющих. Например, если в функции A проинициализированы некоторые переменные, то между ними устанавливается связь. При этом для каждой связи между компонентами в графе создается ребро или дуга для соответствующих узлов. Если компонент A зависит от компонента B, то дуга идет от B к A.

2. Выбираются элементы программы, в которые будет встроена информация. Эти элементы выбираются таким образом, чтобы встраивание информации было максимально незаметным и не влияло на работу программы. Можно выбирать элементы, которые имеют низкую чувствительность к изменениям, такие как комментарии, пробелы или неиспользуемый код.

3. Производится встраивание самих данных. Для этого можно использовать различные методы, такие как замена определенных символов на биты информации, изменение идентификаторов переменных или атрибутов классов, или добавление дополнительных атрибутов, методов или других элементов. Метод встраивания информации определяется таким образом, чтобы исходный код программы оставался пригодным для выполнения. Важно, чтобы внедренная информация не приводила к ошибкам в работе программы и не вызывала подозрений.

4. Производится проверка незаметности изменений в коде программы. Это может быть выполнено с помощью автоматического анализа кода, тестирования программы или визуального сравнения кода до и после встраивания информации.

Для извлечения вложенной информации производят обратный процесс анализа графа зависимостей и извлечение данных из выбранных элементов программы, в которые была встроена информация.

Обоснование выбора узлов графа для вложения информации

Правильный выбор узлов графа может обеспечить эффективное и надежное внедрение информации, а также усложнить ее обнаружение [18-20]. Для выбора узлов графа следует руководствоваться следующими факторами:

1. Важность узлов: выбор узлов, которые имеют высокую важность в контексте функционирования программы, является критическим. Это могут быть узлы, отвечающие за основную логику программы, обработку важных данных или взаимодействие с внешними системами. Внедрение информации в такие узлы может повысить сложность ее обнаружения и обеспечить более надежную защиту внедренной информации. Для вложения в данном случае необходимы очень надежные алгоритмы, которые с большой вероятностью не нарушат целостность узла.

2. Частота использования узлов: узлы, которые часто используются в программе, могут быть также подходящими для внедрения информации. Это могут быть узлы, вызываемые из различных частей программы или используемые в циклах, условных операторах или других часто выполняемых операциях. Внедрение информации в такие узлы может обеспечить более равномерное распределение внедренной информации по программе и усложнить ее обнаружение.

3. Структура графа: структура графа программы также может влиять на выбор узлов для внедрения информации. Узлы, находящиеся в глубине графа или имеющие сложные связи с другими узлами, могут быть менее очевидными местами для поиска стеганографической активности. Выбор таких узлов может усложнить обнаружение внедренной информации.

Выбор узлов графа для вложения информации должен быть основан на анализе конкретной программы, ее структуры и особенностей, а также на требованиях к безопасности и ожидаемой степени обнаружения внедренной информации. Это позволит обеспечить эффективное вложение информации.

Описание экспериментов, проведенных для проверки эффективности и корректности предложенного алгоритма

Для эксперимента рассмотрим простой код (т.е. без использований объектно-ориентированного программирования (ООП), рекурсий и специфичных особенностей различных языков). Для построения графа зависимостей производится анализ функций, объявленных в исходном коде программы. Для этого определяются следующие обозначения узлов:

• P – объявление переменной (т.е. процесс выделения памяти или ее перезаписывания);

• E – выражение (под выражением понимается любая часть кода, имеющая какой-либо оператор);

• R – конец функции и возврат значения.

Также примем следующие обозначения:

• под весом функции понимается количество узлов, которые задействованы в блоке анализируемой функции;

• под наибольшими функциями – функции, имеющие наибольший вес;

• под глубиной функции – максимальное количество узлов, по которым необходимо пройти, чтобы дойти до узла R;

• под четностью функции – четность ее веса.

Для вложения информации рассматривается ориентированный граф, который отражает зависимости переменных, выражений и конца в функции. В примере простого кода каждый блок функции начинается с объявления переменных или аргументов функции, таким образом первым узлом всегда является выражение (т.е. узел E), используемое для объявления, а второй узел – переменная (т.е. узел P) для сохранения результата выражения.

Таким образом путь, имеющий наибольшую глубину, начинается с узла E. При этом можно также определить процесс добавления одного «пустого» узла E, который ничего не делает с точки зрения алгоритма программы, но увеличивает общий вес функции на единицу (данная, казалось бы, бесполезная опция будет обговорена далее). Примеры представлены на рисунке 1.

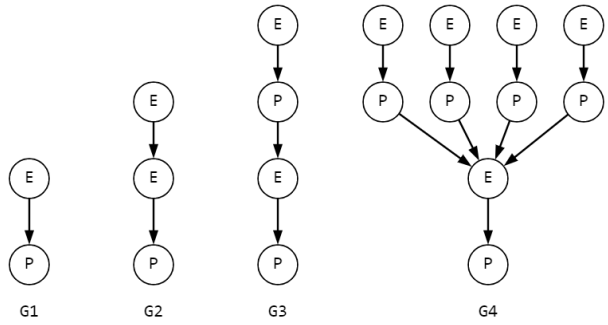

Рис. 1. Примеры добавления данных

Из рисунка 1 граф G1 обозначает начало исходного графа. Обозначим глубину исходного графа G1 как p. Граф G2 показывает, как добавляется «пустой» узел, который никак не влияет на суть работы функции, но и не какой информации не несет в себе (т.е. в графе G2 никакой информации не было вложено), однако он увеличивает вес на единицу, что будет использовано в дальнейшем (здесь глубина будет p+1). Граф G3 уже содержит в себе вложенную информацию (здесь вложение информации происходит в первых двух узлах и в результате чего данное вложение сохраняется в переменной P второго узла, при этом глубина равна p+2). Граф G4 показывает вложения четырех объектов (т.е. здесь объявляются 4 переменные, в которых хранятся вкладываемые объекты, глубина также равна p+2).

Одним из преимуществ такого подхода является то, что вложение информации может быть незаметным для внешнего наблюдателя, так как оно осуществляется путем добавления дополнительных узлов E и P, которые могут выглядеть как естественная часть кода программы. Это может усложнить обнаружение наличия скрытой информации, так как визуально код может выглядеть нормальным и не вызывать подозрений.

Но возникают следующие вопросы: каким образом выбирать функции и узлы в них для вложения информации и как извлекать данные?

Выбор функций для вложения информации основывается на четности их веса. Для этого из них выбираются те, которые имеют больший вес и глубину. Такой подход является эффективным для скрытого вложения информации в программу, так как функции с более высоким весом и глубиной менее подозрительны для вложения, так как содержат более сложный код и больше узлов, что затрудняет обнаружение стеганографической активности.

Для вложения данных должен выбираться узел, который имеет самую большую глубину. Таким образом в графе зависимости будет несколько путей, имеющих самую большую глубину (точнее их глубина будет p+2, тут стоить обратить внимание на граф G3 и G4 из рисунка 1). В результате чего процесс извлечения информации следующий: необходимо пройтись по путям с самой большой глубиной и выписать значения первых двух узлов каждого пути.

В таком случае алгоритм вложения и извлечения данных в код программ на основании графа зависимости выглядит следующим образом:

Положим, что необходимо вложить C объектов в код. Для этого:

1. для каждой из функций строится граф зависимости;

2. анализируется граф каждой функции и определяется ее вес и глубина;

3. выбираются n функций с наибольшими весами (если их веса одинаковые, то выбираются по глубине);

4. выбирается функция, которая не попала в 3 шаг, но которая имеет больший вес среди оставшихся и наименьшую функцию из шага 3. Если четности данных двух функций одинаковы, то в качестве результирующей четности выбрать противоположную им, если же они разные, то выбрать четность наименьшей функции из шага 3 (здесь четность веса функции является тем параметром, в соответствии с которым производится дальнейшее извлечение информации);

5. равномерно распределяется количество объектов на все функции (здесь следует учесть, что вкладывать необходимо небольшое число объектов для большей скрытности, потому $\frac{C}{n} \approx 1$);

6. производится вложение объектов и, если результирующий вес функции получился не подходящей четности, то добавляется пустой узел (здесь нужно обратиться к рисунку 1, в котором вложение самих данных происходило в графах G3 и G4, а добавление пустых узлов в графе G2). Результат выполнения данного шага – граф с вложением;

7. для извлечения информации строятся графы зависимости для всех функций и определяется вес и глубина каждой из них. После чего формируется последовательность элементов в порядке возрастания (здесь под элементами подразумеваются функции, при этом для сравнения используется их вес и глубина). Рассматриваются наибольшие элементы и определяется, образуют ли они подпоследовательность одинаковой четности в соответствии с их весами. Если она образуется, то переходим к следующему шагу, если нет, то делается вывод об отсутствии вложения (это объясняется тем, что в качестве результирующей четности в соответствии с шагом 4 берется значение, противоположное значению четности функции, которая не попала в 3 шаг, но которая имеет больший вес среди оставшихся, таким образом данная функция выступает в роли метки конца);

8. для каждого элемента подпоследовательности из шага 7, анализируются графы и определяются пути с самой большой глубиной, после чего выбираются первые два узла, из которых извлекаются данные. Полученное множество данных является вложенной информацией.

Пример вложения

Пусть дана простая программа на языке Python. В данной программе определены десять функций. Обозначим данные функции как f_1, f_2, ... f_{10}. Необходимо вложить в нее 14 объектов.

В соответствии с предложенным алгоритмом для вложения информации производится анализ всех функций и строятся графы зависимостей, в соответствии с которыми определяются их веса и глубина. Полученные результаты представлены в виде табл. 1 в порядке возрастания значения веса (при этом, если у функций одинаковый вес, то сортировка происходит по глубине также в порядке возрастания).

Табл. 1. Параметры функций

Функция	Вес	Глубина
f_2	5	3
f_6	9	6
f_1	10	5
f_4	14	5
f_{10}	15	5
f_8	15	6
f_5	22	9
f_9	25	11
f_3	34	12
f_7	43	17

Для вложения информации взято значение $n = 5$. Это означает, что объекты вкладываются в 5 функций, имеющих самый большой размер: f_8, f_5, f_9, f_3 и f_7. При этом можно заметить, что вес самого маленького элемента (f_8) совпадает с весом самой большой функции, не попавшей в n самых больших (т.е. f_{10}). Таким образом, в соответствии с 4 шагом предложенного алгоритма веса первых 5 самых больших функций после вложения должны быть четными (т.к. f_8 является нечетным).

Вложение в 5 функций, имеющих самый большой размер: f_8, f_5, f_9, f_3 и f_7, произведено в узлы, которые имеют наибольшую глубину. Полученные результаты представлены в виде табл. 2.

Таблица 2. Параметры функций после вложения

Функция	Вес	Глубина	Вложено объектов	Добавлено узлов
f_2	5	3	0	0
f_6	9	6	0	0
f_1	10	5	0	0
f_4	14	5	0	0
f_{10}	15	5	0	0
f_8	20	8	2	5
f_5	28	11	3	6
f_9	32	13	3	7
f_3	40	14	3	6
f_7	50	19	3	7

Рассмотрим более подробно процесс вложения информации на примере функции f_8. Изначально ее вес равняется 15, а глубина – 6. Это можно наблюдать на рисунке 2.

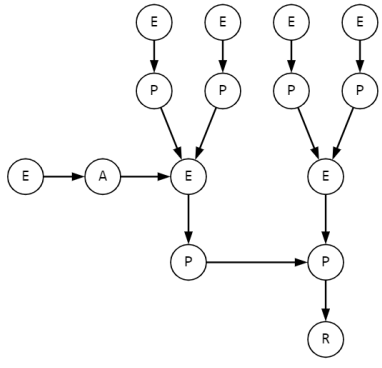

Рис. 2. Граф зависимости исходного кода программы

После построения и анализа графа зависимости исходной функции f_8 происходит вложения данных по предложенному алгоритму, в результате чего, исходя из таблицы 2, происходят следующие изменения: добавляется 5 узлов, где 4 из них служат для вложения информации, а 1 узел является пустым и служит только для регулирования четности функции, при этом в данный граф вкладывают 2 объекта (один объект – это два узла *E* и *P*, соединенных одной дугой). Результирующий граф с вложением представлен на рисунке 3.

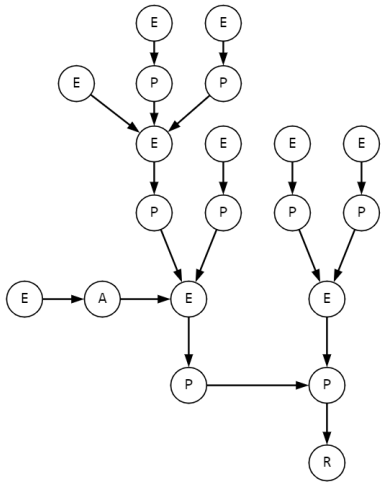

Рис. 3. Граф зависимости после вложения информации

Для остальных функций граф с вложением формируется аналогичным образом.

В качестве примера приведем также результат вложения информации при помощи стандартных методов – рисунок 4.

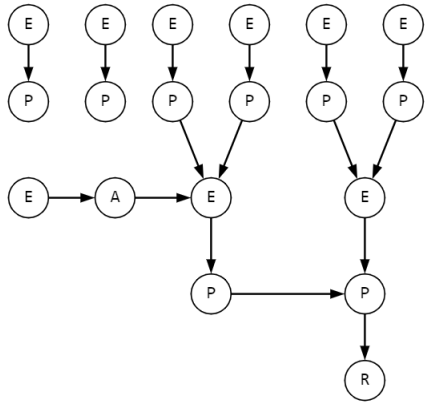

Рис. 4. Граф зависимости после вложения при помощи стандартных методов

Как видно из рисунка 4, при вложении информации при помощи стандартных методов будут появляться «висячие» узлы, которые сразу будут заметны на графе, а также при визуальном анализе программы, что сразу выдаст присутствие вложения.

Заключение

В работе рассмотрены методы вложения информации, основанные на встраивании информации в комментарии, идентификаторы переменных и функций, строковые литералы, структуру кода программы и метаданные. А также предложен и описан алгоритм нового метода вложения информации в исходный код приложений с использованием графа зависимостей элементов в нем.

Проведенное исследование показало, что использование графа зависимостей для вложения информации в код программ имеет следующие преимущества:

Скрытность: вложенная информация незаметна для внешнего наблюдателя, так как она интегрируется в структуру программы без изменения ее внешнего вида или поведения.

Эффективность: граф зависимости программы предоставляет естественную структуру для вложения информации, так как в нем отражаются связи между различными компонентами программы. Данное свойство позволяет эффективное и компактное вложение информации без необходимости введения дополнительных структур.

Универсальность: граф зависимости является абстрактным представлением программы и используется для вложения информации в различные языки программирования и типы программ. Это делает алгоритм вложения информации на основе графа зависимости более универсальным и применимым в различных контекстах.

Более сложные манипуляции с данными: граф зависимости предоставляет более мощные возможности для манипуляции с данными внутри кода программы. Например, можно использовать зависимости между различными элементами графа для скрытой передачи данных или сокрытия информации в определенных путях выполнения программы. Это позволяет произвести процесс встраивания более гибким и мощным с точки зрения манипуляции данными.

Более сложные методы обнаружения: вложение информации при помощи графа зависимости затрудняет обнаружение встроенной информации. Такие методы, как статический ана-

лиз или поиск неиспользуемого кода, являются менее эффективными для обнаружения информации, встроенной с использованием графа зависимости, что повышает уровень скрытности вложенной информации и делает ее более защищенной от обнаружения.

В дальнейшем необходимо разработать новые методы, учитывающие множественные уровни зависимостей и использующие модели машинного обучения для повышения точности и эффективности процесса вложения информации на основании графа зависимостей.

Литература

1. *Коржик В.И., Небаева К.А., Герлинг Е.Ю., Догиль П.С., Федянин И.А.* Цифровая стеганография и цифровые водяные знаки. Часть 1. Цифровая стеганография. СПбГУТ. СПБ., 2016. 226 с.

2. *Грибунин В.Г., Оков И. Н., Туринцев И.В.* Цифровая стеганография М.: Солон-Пресс, 2018. 272 с.

3. *Красов А. В.* Метод защиты авторских прав и целостности программного обеспечения на основе внедрения ЦВЗ в исполняемый код // Перспективы науки. 2022. № 4(151). С. 16-25.

4. *Герлинг Е. Ю., Ахрамеева К. А.* Обзор современного программного обеспечения, использующего методы стеганографии // Экономика и качество систем связи. 2019. № 3(13). С. 51-58.

5. *Герлинг Е. Ю., Ахрамеева К. А.* Использование стеганографии в социальных сетях и мессенджерах. Актуальные проблемы инфотелекоммуникаций в науке и образовании: сборник научных статей: в 4х томах, Санкт-Петербург, 24–25 февраля 2021 года / Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича. Том 1. Санкт-Петербург: Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича, 2021. С. 63- 67.

6. *Нечта И.В.* Эффективные методы включения Скрытой информации в тексты программ. // Российская научно-техническая конференция «Информатика и проблемы телекоммуникаций». Новосибирск, ФГОБУ ВПО «СибГУТИ», 26-28 апреля, 2009. С. 22.

7. *Нечта И. В*. Разработка методов обеспечения безопасности использования информационных технологий, базирующихся на идеях стеганографии : специальность 05.13.17 "Теоретические основы информатики" : автореферат диссертации на соискание ученой степени кандидата технических наук / Нечта Иван Васильевич. Новосибирск, 2012. 20 с.

8. *Красов А. В., Верещагин А. С., Цветков А. Ю.* Аутентификация программного обеспечения при помощи вложения цифровых водяных знаков в исполняемый код // Телекоммуникации. 2013. № S7. С. 27-29.

9. *Верещагин М. В., Красов А. В.* Скрытое вложение в байт-код Java на основе использования метода переопределяемых значений переменных // Актуальные проблемы инфотелекоммуникаций в науке и образовании (АПИНО 2018) : VII Международная научно-техническая и научно-методическая конференция. Сборник научных статей. В 4-х томах, Санкт-Петербург, 28 февраля 2018 года / Под редакцией С.В. Бачевского. Том 1. Санкт-Петербург: Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича, 2018. С. 152-154.

10. *Красов А. В., Борисов В. И.* Исследование применимости известных методов внедрения цифровых водяных знаков к исполняемым файлам Unix-подобных систем // Известия высших учебных заведений. Технология легкой промышленности. 2022. Т. 56, № 2. С. 38-42. DOI 10.46418/0021-3489_2022_56_02_07.

11. *Красов А. В., Шариков П. И.* Обеспечение безопасности java программ посредством вложения программного водяного знака // Актуальные проблемы инфотелекоммуникаций в науке и образовании (АПИНО 2018) : VII Международная научно-техническая и научно-методическая конференция. Сборник научных статей. В 4-х томах, Санкт-Петербург, 28 февраля 2018 года / Под редакцией С.В. Бачевского. Том 1. Санкт-Петербург: Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича, 2018. С. 517-520.

12. *Сидорова Е. В., Дмитриева Н. Г., Калинина Н. А.* Решение задачи построения графа зависимостей программных модулей в системе Node.js // Труды НГТУ им. Р.Е. Алексеева. 2019. № 4(127). С. 44- 52. – DOI 10.46960/1816-210X_2019_4_44.

13. *Щербаков А. С.* Быстрый алгоритм учета зависимостей данных при анализе и тестировании программного обеспечения СБИС // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2016. № 2. С. 76-83.

14. *Воевода А. А., Романников Д. О.* Способы представления программ и их анализ // Сборник научных трудов Новосибирского государственного технического университета. 2014. № 3(77). С. 81- 98.

15. *Новиков А. С.* Построение графа информационных зависимостей по машинному коду // Известия Тульского государственного университета. Технические науки. 2016. № 2. С. 180-187.

16. Платформенно-независимый и масштабируемый инструмент поиска клонов кода в бинарных файлах / А. К. Асланян, Ш. Ф. Курмангалеев, В. Г. Варданян [и др.] // Труды Института системного программирования РАН. 2016. Т. 28, № 5. С. 215-226. DOI 10.15514/ISPRAS-2016-28(5)-13.

17. *Барыкин М. П., Борзунова Т. Л.* Элементы теории графов и некоторые их приложения. М-во образования Рос. Федерации, Волгогр. гос. техн. ун-т. Волгоград : Политехник, 2003. 52 с. ISBN 5-230-04083-1.

18. *Клековкин Г. А., Коннова Л. П., Коннов В. В.* Геометрическая теория графов. 2-е изд., испр. и доп. Москва : Издательство Юрайт, 2017. 240 с. ISBN 978-5-534-04812-4.

19. *Непретимова Е. В., Донец В. С.* Алгоритмы теории графов: анализ и программная реализация // Естественные науки - основа настоящего и фундамент для будущего : материалы VII ежегодной научно-практической конференции Северо-Кавказского федерального университета «Университетская наука - региону», Ставрополь, 03–29 апреля 2019 года / отв. ред. В.И. Шипулин. Ставрополь: Северо-Кавказский федеральный университет, 2019. С. 52-54.

THE METHOD OF EMBEDDING INFORMATION IN THE PROGRAM CODE BASED ON THE GRAPH OF OBJECT DEPENDENCIES

KSENIA A. AHRAMEEVA
PhD, St-Petersburg, Russia, cbor.mail@gmail.com

RUSTEM R. ZAYDULLIN
Student, St-Petersburg, Russia, rstm2808@gmail.com

EVGENIY V. TAROV
Student, St-Petersburg, Russia, tarov25@mail.ru

ABSTRACT

Introduction: The paper presents the results of research in the field of digital steganography, with a focus on embedding information in program code. Various methods and techniques are considered that make it possible to introduce hidden information into the program code, and a detailed description of the developments and experimental results is proposed. The results of experiments on embedding information in the program code are presented.

Keywords: information security; steganography; information embedding; embedding in program code; dependency graph.

REFERENCES

1. Korzhik V.I., Nebaeva K.A., Gerling E.Yu., Dogil P.S., Fedyanin I.A. Digital steganography and digital watermarks. Part 1. Digital steganography. SPbSUT. SPB., 2016. 226 p.
2. Gribunin V.G., Okov I.N., Turintsev I.V. Digital Steganography. M.: Solon-Press, 2018. 272 p.
3. Krasov A. V. Method of protecting copyright and software integrity based on the introduction of digital watermarks into the executable code. Prospects of Science. 2022. No. 4 (151). Pp. 16-25.
4. Gerling E. Yu., Akhrameeva K. A. A review of modern software using steganography methods. Economics and quality of communication systems. 2019. No. 3 (13). Pp. 51-58.
5. Gerling E. Yu., Akhrameeva K. A. The use of steganography in social networks and messengers. Actual problems of infotelecommunications in science and education: collection of scientific articles: in 4 volumes, St. Petersburg, 24– February 25, 2021 / St. Petersburg State University of Telecommunications. prof. M.A. Bonch-Bruevich. Volume 1. St. Petersburg: St. Petersburg State University of Telecommunications. prof. M.A. Bonch-Bruevich, 2021. Pp. 63-67.
6. Something I.V. Effective methods for including Hidden Information in program texts. Russian Scientific and Technical Conference "Computer Science and Problems of Telecommunications". Novosibirsk, FGOBU VPO "SibGUTI", April, 2009. P. 22.
7. Nechta I. V. Development of methods for ensuring the security of using information technologies based on the ideas of steganography: specialty 05.13.17 "Theoretical foundations of informatics": abstract of the dissertation for the degree of candidate of technical sciences. Nechta Ivan Vasilyevich. Novosibirsk, 2012. 20 p.
8. Krasov A. V., Vereshchagin A. S., Tsvetkov A. Yu. Software authentication by embedding digital watermarks in the executable code. Telecommunications. 2013. No. S7. Pp. 27-29.
9. Vereshchagin M. V., Krasov A. V. Hidden embedding in Java bytecode based on the use of the method of redefined variable values. Actual problems of infotelecommunications in science and education (APINO 2018): VII International Scientific -technical and scientific-methodical conference. Collection of scientific articles. In 4 volumes, St. Petersburg, February 28 - 01, 2018. Edited by S.V. Bachevsky. Volume 1. St. Petersburg: St. Petersburg State University of Telecommunications. prof. M.A. Bonch-Bruevich, 2018. Pp. 152-154.
10. Krasov A. V., Borisov V. I. Study of the applicability of known methods for introducing digital watermarks to executable files of Unix-like systems. Light industry technology. 2022. T. 56, No. 2. Pp. 38-42. DOI 10.46418/0021-3489_2022_56_02_07.
11. Krasov A. V., Sharikov P. I. Ensuring the security of java programs by embedding a software watermark. Actual problems of infotelecommunications in science and education (APINO 2018): VII International Scientific, Technical and Scientific -methodological conference. Collection of scientific articles. In 4 volumes, St. Petersburg, February 28 - 01, 2018. Edited by S.V. Bachevsky. Volume 1. St. Petersburg: St. Petersburg State University of Telecommunications. prof. M.A. Bonch-Bruevich, 2018. Pp. 517-520.
12. Sidorova E. V., Dmitrieva N. G., Kalinina N. A. Solving the problem of constructing a dependency graph of program modules in the Node.js system. Proceedings of NNSTU im. R.E. Alekseev. 2019. No. 4 (127). Pp. 44-52. DOI 10.46960/1816-210X_2019_4_44.
13. Shcherbakov A. S. A fast algorithm for taking into account data dependencies in the analysis and testing of VLSI software. Problems of development of promising micro- and nanoelectronic systems (MES). 2016. No. 2. Pp 76-83.
14. Voevoda A. A., Romannikov D. O. Ways of representing programs and their analysis. Collection of scientific works of the Novosibirsk State Technical University. 2014. No. 3 (77). Pp. 81-98.
15. Novikov A. S. Building an information dependency graph using machine code. Bulletin of the Tula State University. Technical science. 2016. No. 2. Pp. 180-187.

16. Aslanyan A. K., Kurmangaleev Sh. F., Vardanyan V. G. [et al.] A platform-independent and scalable tool for searching code clones in binary files. Proceedings of the Institute for System Programming of the Russian Academy of Sciences. 2016. T. 28, No. 5. Pp. 215-226. DOI 10.15514/ISPRAS-2016-28(5)-13.

17. Barykin M. P., Borzunova T. L. Elements of graph theory and some of their applications. Ministry of Education Ros. Federation, Volgograd. state tech. un-t. Volgograd: Polytechnic, 2003. 52 p. ISBN 5-230-04083-1.

18. Klekovkin G. A., Konnova L. P., Konnov V. V. Geometric graph theory. 2nd ed., Rev. and additional.Moscow: Yurayt Publishing House, 2017. 240 p. ISBN 978-5-534-04812-4.

19. Nepretimova E. V., Donets V. S. Graph theory algorithms: analysis and software implementation. Natural sciences - the basis of the present and the foundation for the future: materials of the VII annual scientific and practical conference of the North Caucasus Federal University "University science - the region", Stavropol, April 03-29, 2019. ed. IN AND. Shipulin. Stavropol: North Caucasian Federal University, 2019. Pp. 52-54.